405

DAMIR VANDIĆ

Intelligent Information Systems for Web Product Search

DAMIR VANDIĆ - Intelligent Information Systems for Web Product Search

ERIM

Intelligent Information Systems for Web Product Search

Intelligent Information Systems for Web Product Search

Intelligente informatiesystemen voor het online zoeken naar producten

Thesis

to obtain the degree of Doctor from the
Erasmus University Rotterdam
by command of the
rector magnificus

Prof.dr. H.A.P. Pols

and in accordance with the decision of the Doctorate Board

The public defense will be held on

Friday 10 February 2017 at 13:30 hours

by

Damir Vandić
born in Sarajevo, Bosnia and Herzegovina

Erasmus University Rotterdam

Doctoral Committee

Doctoral dissertation supervisor:	Prof.dr.ir. U. Kaymak
Other members:	Prof.dr. F.M.G. de Jong
	Prof.dr.ing. M. Gaedke
	Prof.dr. R. Baeza-Yates
	Prof.dr.ir. R. Dekker
	Prof.dr. T. Martin
Co-supervisor:	Dr. F. Frasincar

Erasmus Research Institute of Management - ERIM
The joint research institute of the Rotterdam School of Management (RSM)
and the Erasmus School of Economics (ESE) at the Erasmus University Rotterdam
Internet: http://www.erim.eur.nl

ERIM Electronic Series Portal: http://hdl.handle.net/1765/1

ERIM PhD Series in Research in Management, 405
ERIM reference number: EPS-2017-405-LIS
ISBN: 9789058924605

Design: PanArt, www.panart.nl
This publication (cover and interior) is printed by Tuijtel on recycled paper, BalanceSilk®.
The ink used is produced from renewable resources and alcohol free fountain solution.
Certifications for the paper and the printing production process: Recycle, EU Ecolabel, FSC®, ISO14001.
More info: http://www.tuijtel.com

SIKS Dissertation Series No. 2017-06
The research reported in this thesis has been carried out in cooperation with SIKS, the Dutch Research School for Information and Knowledge Systems (`http://www.siks.nl`)

Preface

"*[A] quotation is a handy thing to have about, saving one the trouble of thinking for oneself, always a laborious business.*"
Alan Alexander Milne (1882 – 1956)

During my Ph.D. trajectory I have had the support of many people, and for that, I am forever grateful. First and foremost, I would like to thank my copromotor and daily supervisor, dr. Flavius Frasincar. Flavius, I really enjoyed the numerous lunches, meetings, and conferences we spent together. You went above and beyond in your supervision of my Ph.D. trajectory and I am aware that I am very lucky in this regard! Next, I would like to thank my promotor Prof.dr.ir. Uzay Kaymak. Your help with my application for the NWO Mosaic grant has been indispensable. The meetings and discussions with you were not only inspirational, but they have always resulted in new ways for me to tackle the problem at hand. Furthermore, I would like to express my gratitude towards the members of my doctoral committee. Thank you so much for critically reviewing my dissertation.

Various organizations have played important roles in making my Ph.D. trajectory possible. I am thankful for the funding and academic support provided by the Econometric Institute, the Erasmus Research Institute for Management (ERIM), the Dutch Research School for Information and Knowledge Systems (SIKS), and the NWO Mosaic program. I would like to thank Anneke, Carien, Marjon, Marianne, and Ursula from the Econometric Institute for their efforts at the secretariat and in the office management. Thank you for being so patient and kind.

My colleagues at the university were not only a source of inspiration, but they also made sure that working on my dissertation remained fun throughout the years. Frederik and Alexander, my paranymphs, thank you for all the conversations, support, and entertaining breaks from work. Thank you for all the fruitful collaborations, it was a real pleasure working together. Many more other colleagues have made my

time at Erasmus University Rotterdam a pleasant experience. In particular, I would like to thank Charlie, Kim, Nalan, Rui, Tommi, Viorel, Wim, and Yingqian for all the good times we had at university.

Special thanks goes out to the (former) students of the Econometric Institute Bachelor's and Master's programs. In particular I would like to call out Didier, Gerhard, Hanno, Iris, Jim, Lennart, Marnix, Nikki, Raymond, Rick, Ronald, Sabri, Sjoerd, Steven, and Wim. I really enjoyed supervising you and I am proud of the papers that we have managed to publish together.

Last but not least, I am grateful for my family, in particular my mother, father, and sister, for their everlasting support and words of wisdom. Without you, I would not be the person I am today. Thank you, my friends, Jan-Willem, Lieke, Marien, Hannah, Laurens, and Renske, for all the fun time we spent together. Britt, Zjenja, Olga, and Tonio, it was nice to be able to share my Ph.D. experiences with you. Lilia and Leonid, thank you for believing in me and for raising such a sweet and kind daughter!

Dear Anna, thank you for supporting me throughout the years. Whether I was working late or in the weekends, or being away because of traveling for conferences, you always understood and you were always there for me. You remind me that there are more important things to life than obtaining a Ph.D. degree.

Rotterdam, December 2016
Damir Vandić

Table of Contents

List of Figures

List of Tables

Chapter 6

Chapter 7

List of Algorithms

Chapter 1

Introduction

Over the last few years, online shopping has become very popular among consumers. According to a recent projection from Forrester Research, e-commerce spending in the United States will hit approximately $480 billion in 2019, with an expected compound annual growth rate of 10% over the next five years (Mulpuru et al., 2015). Predictions for the future also show us that e-commerce will continue to grow and play an important role in society. One of the reasons behind this growth is the increase in product specificity and consumer preference variation. Another reason is that the product search space on the Web has grown, as even traditional stores without a Web shop are increasingly starting to offer their services online. Because of this increase in online shopping, there is a growing need for efficient and effective product search engines. However, the rapid growth of e-commerce introduces some challenges. For example, due to the wide variety of products and many different online stores where these products can be bought, a large fraction of online shoppers get confused or are overwhelmed by the information they get presented while searching for products (Horrigan, 2008).

In an attempt to lighten this information overload burden on consumers, there are several product search engines that aggregate product descriptions and price information from the Web and allow the user to easily query this information. Google Shopping, Tweakers PriceWatch, and Kieskeurig.nl, where the latter two are Dutch initiatives, are three examples of product search engines of this type. However, all of these search engines expect to receive the data from the participating Web shops in a customized format. The downside of this is that the Web shops have to transform

their data more than once into a custom format, as each product search engine requires a different format.

Because currently most product information aggregation services rely on Web shops to send them their data, there is a big opportunity for solutions that aim to tackle this problem using a more automated approach. There are a couple of advantages to this. First, it is highly unlikely that a product search engine will cover most of the Web shops on the Web if they require the Web shops to send their data. Second, Web shop owners more easily enjoy a greater visibility on the Web because there is no need anymore to provide custom data feeds for every product search engine they want to be listed in. They can spend more time focusing on their business and improving the usability of their Web shop, which is inherently beneficial for consumers. Third, consumers are able to search for a wider range of products and, with proper automation, are also more likely to search across country borders, whereas in the current situation most product search engines operate within a single country.

The four major search engines, i.e., Bing, Google, Yahoo!, and Yandex, recognize this potential for e-commerce and also other domains. In recent years, they have started parsing structured data from Web pages in order to more accurately extract the conveyed information represented on those pages. For this purpose, the four search engines have developed a semantic vocabulary called schema.org (Google, Microsoft, Yahoo and Yandex, 2017). Schema.org is a very broad vocabulary with which the search engines aim to have a high-level shared vocabulary that focuses on popular Web concepts.

Because there are many benefits of having product search engines that are able to automatically crawl and process the available product information and offerings on the Web, this dissertation addresses key aspects of designing and implementing such a system. In the next few sections, we will discuss in detail the components that encompass such a system, the contributions of this dissertation, and the outline of the thesis chapters.

1.1 Research Objectives

The underlying problem statement of this dissertation can be summarized as:

How to design Web product search engines that automatically aggregate product information and allow users to perform effective and efficient queries?

The product search engine platform that we propose, enables automated product information aggregation and querying. It is designed for sources of product information that are both annotated, using vocabularies such as schema.org (Google, Microsoft, Yahoo and Yandex, 2017), as well as Web pages that describe products without any annotations.

Figure 1.1 shows an overview of the proposed platform in terms of the various underlying processing steps. We can see that the system accepts *semi-structured* product information or *structured* product information from the Web. Semi-structured information in this context is a set of product features represented by some recurring markup structure (e.g., HTML tables). Structured information in this context refers to product descriptions represented in RDF. Furthermore, we assume that product

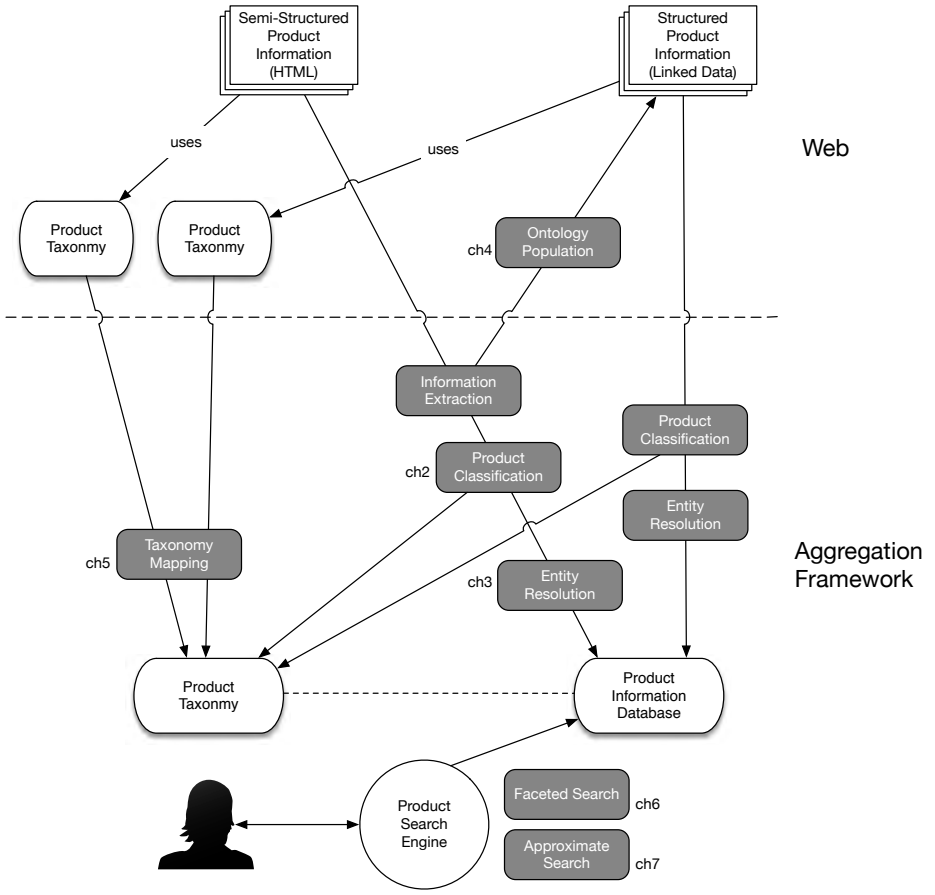

Figure 1.1: An overview of the Web product information aggregation framework.

descriptions from both type of sources are optionally classified into a product taxonomy, a feature that is common among many Web shops nowadays. The key aspect of the platform is to transform these input product descriptions into product descriptions that can be uniformly queried, regardless of the source, and that each product description is classified into a given product taxonomy.

The type of product description (semi-structured or structured) determines the necessary processing pipeline that provides the required end result. Product information that is semi-structured requires to parse the data and exctract the information, which occurs in the *Information Extraction* phase. Then, in the *Product Classification* phase, the product description needs to be classified into the main product taxonomy. Before finally being stored in the product database, the *Entity Resolution* phase identifies the product descriptions that are duplicates, i.e, the ones that describe the very same product. For structured product information, the processing pipeline is similar except for the fact that the first process, i.e., *Information Extraction* is not necessary as we are dealing in this case with a representation from which information is more or less readily available. However, even though the next two processes are the same as with the semi-structured product descriptions, i.e., *Product Classification* and *Entity Resolution*, the employed algorithms differ because in the case of semi-structured data, even after the *Information Extraction* phase, the information quality is lower than the one when dealing with structured data.

When the data from the Web is stored in the product database using the internally managed product taxonomy, it can be queried by users. In this case, two processes play a key role: (1) the interaction of the user with the search system and (2) the ability of the search engine to answer the queries of users. For this purpose, faceted and fuzzy search are approaches that work well with e-commerce systems due to the nature of the data (i.e., many product features and consumer needs that are not always perfectly known upfront). This dissertation focuses on two related topics, i.e., *Facet Ordering* for addressing the user interaction with the search system, and *Approximate Search* for addressing the ability of the search engine to answer queries.

Additionally, in order to be able to more easily map product descriptions in the internal product taxonomy, we investigate approaches for *Taxonomy Mapping*. This vastly increases the performance of the product classification processes as the algorithms can take more evidence into account when classifying the product descriptions. Last, in order to promote the usage of annotated data on the Web, we discuss approaches for *Ontology Population*, where the goal is to transform parsed semi-structured data into instances of a given ontology.

From the main problem statement and the presented framework, there are several research questions that follow. We discuss below which ones are addressed in this dissertation.

Question 1: How can we classify products into an existing taxonomy using only their textual descriptions?
The first issue that needs to be solved in a system that automatically crawls product information on the Web is to classify incoming product descriptions using an existing product taxonomy. We make the assumption here that one is able to parse HTML pages and extract semi-structured data from tables (i.e., page titles and key/value string pairs). This assumption is well supported by the amount of research done in this area (Crescenzi et al., 2001; Gatterbauer and Bohunsky, 2006; Gupta et al., 2003; Krüpl et al., 2005; Pinto et al., 2003; Tengli et al., 2004). For this reason, we do not focus on the *Information Extraction* phase depicted in Figure 1.1. In order to answer this research question, we propose a multi-level hierarchical classification algorithm that makes use of the structure of the taxonomy to derive the most optimal product description classifications.

Question 2: What is a suitable approach for performing large-scale entity resolution in product descriptions?
Many of the crawled Web pages will have different descriptions of the products they are referring to, resulting in duplicate data. Entity resolution, i.e., the task of finding descriptions that refer to the same entity, or sometimes referred to as 'duplicate detection', is therefore an important task in such a system. This task can be performed on the fly, i.e., as the product descriptions are crawled, or in batches, i.e., where entity resolution is applied on a set of new product descriptions using a set of previously processed product descriptions. In this dissertation we devise an algorithm that is suited to perform this task on a large scale and in a distributed environment.

Question 3: How can we effectively populate ontologies from semi-structured product data using lexico-syntactic mappings?
Most of the information found on Web pages are understandable by humans, but unfortunately not by machines. A possible solution to this problem can be the realization of the Semantic Web and its associated Linked Data initiatives (Hepp, 2015; Hitzler and Janowicz, 2013; Shadbolt et al., 2006; Vijayalakshmi et al., 2011). If online product information was not only shared using unannotated HTML, but also using a representation that machines are capable of parsing and reasoning about,

e.g., the RDF data model (Bizer et al., 2009), then this would open up a wide range of novel applications due to the fact that data sharing between different sources would become much easier. Allowing machines to understand concepts like persons, companies, products, etc., facilitates automatic aggregation of information over resources. For instance, consider a Web page that only describes the battery life of a mobile phone and that another Web page describes just the color. If a computer understands that 'battery life' and 'color' are properties of a mobile phone, and it can identify that the two Web pages are about the same resource (a specific mobile phone), then it can aggregate this information in order to describe this one product using both of its properties. Because of the opportunities that Linked Data for e-commerce brings us, and the fact that we would like a system like the one we envision in this dissertation to be as open as possible, we consider this research question to be relevant. To answer this research question, we devise an algorithm that is able to semi-automatically convert product descriptions into instances of an ontology.

Question 4: How to design an approach that automatically maps one product taxonomy into another, using only the category names?
When crawled product descriptions contain a product category, one would want to understand what this category entails and how it maps to the existing categories in the system. In this way, the system maximizes the use of information provided by the product description. In other words, there is a need for a way to map the product taxonomy of the external Web shop to the internal product taxonomy of the product search engine. Such a mapping can be useful for various data processing steps. For example, when a product description is found, the mapping can be used to determine the most appropriate internal product category, utilizing possibly both the information from the mapped taxonomies, as well as the information from the product description itself. For this purpose, we propose an algorithm that can generate mappings between two taxonomies, based solely on the names and hierarchy of categories.

Question 5: How can we reduce the consumer search effort by ranking the displayed facets on a per-query basis?
Nowadays, many Web shops make use of a so-called *faceted navigation* or *faceted search* user interface (Hearst, 2006; Tunkelang, 2009). One of the reasons why faceted search is popular among Web shops is that users find it intuitive (Hearst et al., 2002; Kules et al., 2009). The term 'facet' has a rather ambiguous interpretation, as there are different types of facets. In this work, we refer to facets as the combination of

a property and its value, such as `WiFi:true` or `Lowest price (€):64.00`. Furthermore, facets are usually grouped by their property in user interfaces, in order to prevent them from being scattered around, and, thereby, confusing the user. In other words, the facet properties, such as `Color`, are shown first, and each property presents the actual values (e.g., `Red`, `Green`, and `Blue`).

One of the difficulties with faceted search, especially in e-commerce, is that a large number of facets are available. In general-domain faceted browsing systems, it is not uncommon to simply display all facets. Displaying all facets may be a solution when a small number of facets is involved, but it can overwhelm the user for larger sets of facets (Sinha and Karger, 2005b). Some Web sites that use faceted search have a manual, 'expert-based' selection procedure for facets (Kieskeurig.nl, 2017; Tweakers.net, 2016). However, the task of manually selecting and ordering facets, and keeping this information up-to-date, requires a significant amount of effort. Furthermore, faceted search allows for interactive query refinement, in which the importance of specific facets and properties may change during the search session. Therefore, it is likely that a predefined list of facets might not be optimal in terms of the number of clicks needed to find the desired product. In order to deal with this problem, we propose an approach for dynamic facet ordering in the e-commerce domain.

Question 6: How can we improve fuzzy product search by allowing users to specify facet preference rankings?

There are many Web shops with very large product catalogs. This makes it hard for users to find their desired product in an efficient manner. It has been shown that many users encounter this difficulty while shopping online (Horrigan, 2008), which can negatively impact the turnover for Web shop owners. This emphasizes the need for better search interfaces for browsing through product catalogs on the Web.

Throughout the years various interaction paradigms have been proposed to deal with the above problem. As mentioned previously, faceted search is one of the most popular paradigms for browsing structured data in information systems (Hearst, 2006), especially product catalogs in e-commerce stores. Despite the advantages of faceted search, there are also some serious drawbacks. Traditionally, faceted search imposes a rather strict Boolean model on whether a document matches the query terms: either it matches the query or it does not. While this model allows for a quick drill-down into the catalog, it also means that a user might miss some potentially desired products if they do not completely match the query. This results in a user having to change the query afterwards to include more documents, which increases

search time and makes exploratory search more error-prone. To answer this research question, we design a novel algorithm that is specifically geared towards approximate faceted search within a product catalog of a Web shop. The main focus is on improving the retrieval of relevant products, taking into account the importance of the specified user requirements.

1.2 Contributions

The contributions of this dissertation are as following. First, we evaluate a hierarchical classification algorithm specifically in the e-commerce domain, for which very little related work exists. Although Ding et al. (2002) focus on the same issue, their work leaves room for improvement because it compares only three methods (VSM, k-Nearest Neighbor, and Naïve Bayes), and more importantly, the results for the hierarchical classification are not promising, with an obtained highest accuracy of 38%. Furthermore, this dissertation contributes to this topic by providing a detailed discussion on the best feature selection methods for product descriptions in the context of classification. Apart from this dissertation, to the best of our knowledge, there have not been any studies that revealed which parts of a product description can be best used for hierarchical classification of products.

Second, different from existing approaches (Baxter et al., 2003; De Vries et al., 2009; Gravano et al., 2001; Hernández and Stolfo, 1998), we propose a scalable entity resolution algorithm for semi-structured product descriptions, where the proposed approach has linear time complexity in terms of the number of input descriptions. Another important contribution is that our evaluation is quite extensive, e.g., we have used 3 large data sets. Furthermore, different from previously proposed approaches, where the focus was on the two-source entity resolution problem, we also address the multi-source entity resolution problem, i.e., handling descriptions of multiple Web shops at the same time.

Third, we propose an algorithm that is able to populate ontologies describing product domains by using tabular product information as input, something that has received very little attention in the literature. Furthermore, most of the proposed approaches, which focus on other domains than e-commerce, rely on natural language processing, using the syntactical context of a text to derive facts, while our approach uses only the given tabular data. Additionally, unlike other approaches, such as Holzinger et al. (2006), our approach employs a GoodRelations-based ontology (Hepp, 2008) for annotating instances. This makes our approach compatible

with major search engines because schema.org (Google, Microsoft, Yahoo and Yandex, 2017) supports annotations represented using this ontology and search engines are able to parse information represented using this vocabulary.

Fourth, in order to accommodate and improve the product classification step, a novel taxonomy mapping algorithm is proposed that is able to map highly heterogeneous product taxonomies coming from multiple sources. By utilizing the structure of the taxonomies, combined with similarity methods that capture both syntactical and semantic similarity, the proposed algorithm is able to outperform previously proposed methods (Noy and Musen, 2003; Park and Kim, 2007) both in terms of precision and recall.

Fifth, we propose a novel facet ordering algorithm for the e-commerce domain, whereas previous approaches have mostly been focused on other domains (Kim et al., 2014b; Koren et al., 2008b; Liberman and Lempel, 2014). Previously proposed solutions have also often assumed that there is a ranking of the results, based on a preceding keyword-based query or external data, which is often not the case for e-commerce. Furthermore, our approach ranks properties and facets, unlike existing algorithms (Kim et al., 2014b; Koren et al., 2008b; Liberman and Lempel, 2014; Vandic et al., 2013a), which filter (or select) properties and facets. Last, none of the proposed approaches from the literature focuses on the performance (time-wise) of the proposed algorithms, which is a very important aspect that the proposed solutions must consider in order to be useful in practice.

Last, complementing the facet ordering algorithm, an approximate (fuzzy) search engine is proposed that can take into account the ranked facet preferences of users. Even though there are previously proposed approaches that are specifically geared towards e-commerce, they have various shortcomings. Many of these approaches, such as Li et al. (2011), rely on well-estimated user statistics, which are not readily available in practice as most Web shops do not have the infrastructure or skills to collect this information. Other methods, such as Burke (2002) and Pu and Chen (2005), deviate too much from what users expect from a classical Web shop, which is often the commonly encountered faceted navigation. Furthermore, they fail to show that the newly proposed approaches always yield an improvement in user satisfaction and/or search engine performance. The approach proposed in this dissertation lies closer to what users use on a daily basis and is likely to be more successful when deployed in practice.

1.3 Declaration of contribution

In this section, the contributions of authors to the publications related to this dissertation are discussed.

Chapter 2

The author of this dissertation has independently performed the majority of the work in this chapter, accompanied with the feedback from the promoter and co-promoter.

Chapter 3

The author of this dissertation has independently performed the majority of the work in this chapter, accompanied with the feedback from the promoter and co-promoter.

Chapter 4

To this chapter, the author of this dissertation contributed significantly by (1) actively being involved in the development and evaluation of the algorithm, (2) collecting and preparing the data set, and (3) writing major parts of the published journal article.

Chapter 5

The author of this dissertation contributed significantly to the publication of this chapter by (1) investigating related approaches for automated taxonomy (i.e., doing a literature study), (2) actively being involved in the setup of the evaluation framework and processing of the results from the evaluation, (3) by assisting the other co-authors in the design and implementation of the proposed algorithms, and (4) summarizing the results over multiple experiments into a conference paper and a journal article.

Chapter 6

To this chapter, the author of this dissertation contributed significantly by (1) actively being involved in the development and evaluation of the proposed algorithms, (2) helping collect the data set, (3) implementing the proposed algorithms in a Web application and performed a user study using this implementation, and (4) writing the majority of the journal article.

Chapter 7

The author of this dissertation contributed significantly to this chapter by (1) actively being involved in the development and evaluation of the proposed algorithms, (2) helping collect the data set, (3) implementing the proposed algorithms in a Web application and performed a user study using this implementation, and (4) writing the majority of the journal article.

1.4 Outline of the dissertation

Figure 1.1, discussed in Section 1.1, shows an overview of the research topics discussed so far and the chapters in which they are addressed. In Chapter 2, we answer the first research question by proposing the Hierarchical Product Classification (HPC) framework, which is able to classify products using a hierarchical product category taxonomy. The second research question is addressed in Chapter 3, where a scalable framework for multi-source entity resolution is proposed. Chapter 4 addresses the third research question by proposing FLOPPIES, a framework capable of semi-automatic ontology population of tabular product information from Web stores. In Chapter 5, we address the fourth research question by proposing SCHEMA, an algorithm for automated mapping between heterogeneous product taxonomies in the e-commerce domain. The fifth research question is addressed in Chapter 6, where a framework for dynamic facet ordering in e-commerce is presented. Chapter 7 addresses the sixth and last research question, by proposing a novel framework specifically geared towards approximate faceted search within the product catalog of a Web shop. Last, in Chapter 8, we summarize the findings of this dissertation and provide directions for future work.

Chapter 2

Towards Automatic Product Description Classification*

In *this chapter we propose the Hierarchical Product Classification (HPC) framework for the purpose of classifying products using a hierarchical product taxonomy. The framework uses a classification system with multiple classification nodes, each residing on a different level of the taxonomy. The innovative part of the framework stems from the definition of classification recipes that can be used to construct high-quality classifier nodes, using the product descriptions in the most optimal way. These classifier recipes are specifically tailored for the e-commerce domain. The use of these classifier recipes enables flexible classifiers that adjust to the taxonomy depth-specific characteristics of product taxonomies. Furthermore, in order to gain insight into which components are required to perform high quality product classification, we evaluate several feature selection methods and classification techniques in the context of our framework. Based on 3000 product descriptions obtained from Amazon.com, HPC achieves an overall accuracy of 76.80% for product classification. Using 110 categories from CircuitCity.com and Amazon.com, we obtain a precision of 93.61% for mapping the categories to the taxonomy of shopping.com.*

---

*This chapter is based on the article "D. Vandic, F. Frasincar, and U. Kaymak. Multi-Level Hierarchical Product Classification in E-commerce. *IEEE Transactions on Cybernetics*, 2017, under review."

2.1 Introduction

The World Wide Web (WWW) has drastically changed the availability and exchange of information. Nowadays, consumers and businesses more often make use of e-commerce (Mulpuru et al., 2015). Most studies from literature focus on approaches that personalize the experience (Lee et al., 2012) and enhance the purchase decisions (Wang et al., 2013; Yang, 2010) of users visiting Web shops. Product taxonomies are related to these research fields and have been widely used for the organization of many kinds of information on the Web. Product taxonomies help customers find relevant products and allow businesses to organize the offered product assortments. Figure 2.1 shows an example of the product taxonomy of Amazon.com, where the consumer has a good overview of the products that are offered.

Usually, there is no uniform description of the same products among different vendors on the Web, which forces business-to-business e-commerce to address the issues that arise with heterogeneous information (Ng et al., 2000), such as synonyms and homonyms. Similarly, the automatic classification of products is becoming more important as this enables companies to lower costs by spending less time on this task. Without automatic classification, one has to manually classify products and the cost of this process will keep increasing as the heterogeneous information on the Web keeps growing.

In this chapter, we investigate text classification techniques for the purpose of providing effective hierarchical product classification. We loosely define product classification as the task of 'assigning a product to an existing or new category, given a product description'. Hierarchical classification can be considered as a classification that takes the hierarchical structure of the taxonomy into account. In this work, the classification is determined to be static, i.e., we assume that the classification of products will not change over time.

Product descriptions on the Web usually contain information like the title, brand, features description, and (optionally) reviews of the product. If the product description is extracted from an existing system, it can also contain the category it was assigned in that system. On the Web, product descriptions are often not structured and not classified, i.e., the product category is missing or does not belong to a standard taxonomy (Fensel et al., 2001). There are various taxonomies that can be used for the purpose of product classification, including the United Nations Standard Products and Services Code (UNSPSC) standard (UNSPSC.org, 2014).

The main focus of this chapter is on the classification of products into an existing product taxonomy. The product taxonomy refers to a predefined hierarchy of product

categories. The goal of this research is to evaluate text classification techniques for the purpose of effective product classification, and to provide a framework that deals with various issues encountered in practice that impede this process. For this system, we can identify three main requirements, (1) the classification of products to both internal and leaf nodes in the category hierarchy, thereby supporting classification to multiple nodes (multiple classification), (2) dealing with product descriptions that may contain a category, and (3) providing a decision algorithm to identify the cases where no matching category exists.

The proposed Hierarchical Product Classification (HPC) framework requires a given category hierarchy, without existing product description associations. Given the above requirements, we can identify three scenarios that can occur when a new product has to be classified, as shown in Figure 2.2. A product description can contain a category that does not necessarily have to be present in the existing category taxonomy. If the product description contains a category, the system must be able to determine whether or not the product should be classified to an existing category in the hierarchy (scenario 2), or that it should be classified using the given category by adding it to the hierarchy (scenario 3), i.e., when there is no match. The classification algorithm should use, if available, the new given category in this decision, as it can provide valuable information concerning the classification of the product. If the

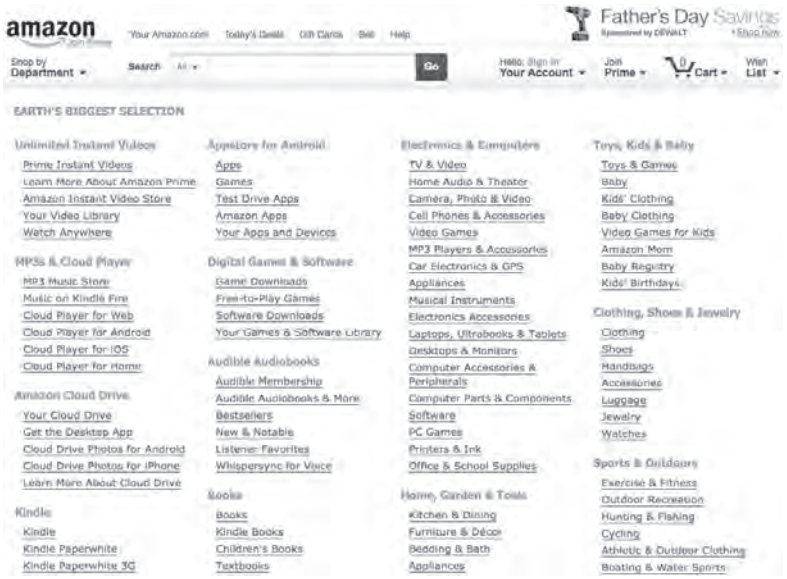

Figure 2.1: Product categories from Amazon.com.

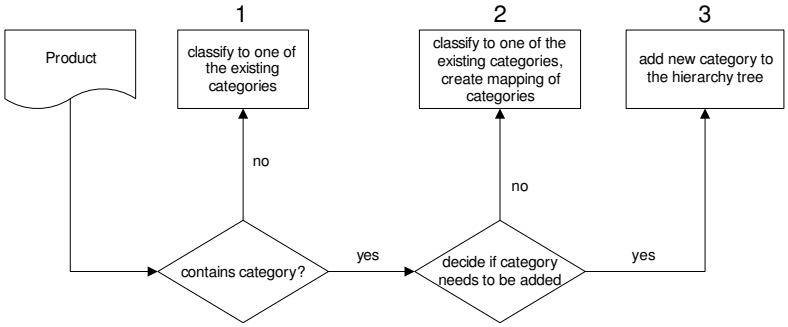

Figure 2.2: Three possible scenarios when a new product needs to be classified.

product description does not contain a category, the task is to classify the product to one of the existing categories (scenario 1). The main focus of this chapter is on scenarios 1 and 2. However, the proposed category mapping algorithm for scenario 2 can be used with a decision function to determine whether the product description 'fits' one of the categories present in the system or whether the computed mapping is false, i.e., whether it is necessary to perform the steps for scenario 3.

The contributions of this chapter are threefold. First, we evaluate several classification techniques on large, real- world product description data sets, which has not been done before. Second, we propose a high precision algorithm that makes use of syntactic and semantic similarities in order to map a given product category to an existing taxonomy of product categories. Third, and last, this work gives a clear overview of which feature selection methods in product descriptions provide the most accurate classifications.

This chapter starts with a survey of the current literature on related classification techniques, feature selection methods, and evaluation techniques. We discuss these topics in Section 2.2. In Section 2.3 we present the details of the proposed framework. We provide an overview of the evaluation results in Section 2.4, where we assess the framework with real-world data from Amazon.com. Finally, in Section 2.5, we summarize our findings and give directions for future research.

2.2 Related Work

According to Yang and Liu (1999), automated text classification (TC) is a learning task, defined as assigning predefined category labels to new documents based on the likelihood suggested by a training set of labeled documents. Classification techniques

can be categorized by two aspects. First, the difference between *flat* and *hierarchical* classifiers is that flat classifiers assign documents to categories at one level, i.e., there is no category hierarchy, as opposed to hierarchical classifiers, where the hierarchy of categories must be taken into account by the classifier. Second, a classifier can be an *independent binary* classifier or *m-ary* ($m > 2$) classifier. Given a document, an independent binary classifier makes a yes/no decision for each category, while an *m-ary* classifier typically consists of multiple classifiers (e.g., one for each category) and computes a ranked list of candidate categories for each document.

In the literature, we can find two main approaches for hierarchical text classification, i.e., the *big-bang* approach and the *top-down level-based* approach (Sun and Lim, 2001). These two approaches are not tied to a specific classification technique because they only prescribe how one or more text classifiers should be used. In the *big-bang* approach, only a single classifier is used in the classification process. Given a document, the classifier assigns it to one or more categories in the category taxonomy. In the *top-down level-based* approach, one or more classifiers are constructed at each level of the category taxonomy and each classifier works as a flat classifier at its level. A document will first be classified by the classifier at the root level. It will then be further classified into one or more lower categories by their corresponding classifiers. This process continues until it reaches a final category that could be a leaf category or an internal category. Different types of classification techniques have been developed that can be used with both approaches. These include rule-based techniques (Sasaki and Kita, 1998; Shih and Chen, 1996; Wang et al., 1999), probabilistic approaches (Chakrabarti et al., 1997; Koller and Sahami, 1997; McCallum et al., 1998; Toutanova et al., 2001), fuzzy (Lin and Wang, 2002; Wang and Chiang, 2007), support vector machine approaches (Dumais and Chen, 2000; Oza et al., 2009; Sun and Lim, 2001; Yu et al., 2003), neural networks (Lin and Chen, 1996; Ruiz and Srinivasan, 2002; Weigend et al., 1999), and cluster-based techniques (Li et al., 2007; Steinbach et al., 2000).

A major issue for all text classification techniques is the high dimensionality of the feature space. Several feature selection methods exist that can be used in combination with a threshold to achieve a desired degree of term elimination. *Term frequency thresholding* (TF) is the simplest technique for vocabulary reduction. The frequency of each term is computed and a minimum threshold for this frequency is used to remove terms. *Information gain* (IG) is another feature selection method that is used frequently in the field of machine learning (Mitchell, 1996). IG measures the number of bits of information obtained for category prediction by knowing the

presence or absence of a term in a document. *Mutual information* (MI) is a criterion commonly used in statistical language modeling of words associations (Church and Hanks, 1990). This criterion has by convention value zero when there is no document that contains the considered word pair. To use this criterion for feature selection, one can compute the *average* or *maximum* mutual information value for a term. Another popular statistical criterion is the χ^2 *statistic* (CHI). The χ^2 *statistic* measures the lack of independence between two words and can be compared to the χ^2 distribution with one degree of freedom. Like the MI, the CHI statistic has a value of zero when the two words are independent. This statistic can be used for feature selection in the same manner as MI (computing an average or maximum). The difference between CHI and MI is that CHI is a normalized value, and hence CHI values are comparable.

The authors of (Yang and Pedersen, 1997) have performed an empirical study, comparing different feature selection methods, including the TF, IG, and CHI methods. The authors used two m-ary classifiers, a k-Nearest Neighbor classifier (kNN) (Yang, 1994), and a regression based method named Linear Least Squares Fit mapping (LLSF) (Yang and Chute, 1994). They consider recall and precision as performance measures. The authors conclude that IG and CHI provide the most effective aggressive term removal (up to 90% of the original feature space) without losing classification accuracy.

As previously discussed, there are roughly two approaches to classification, i.e., the top-down and big-bang approaches. Because top-down approaches use multiple classifiers, they address the issue of separating the noisy terms from the useful ones. This task is usually highly dependent on the location in the category hierarchy. For example, 'mobile phone' may be a good feature for a top level classification (e.g., *Electronics*), but becomes useless when drilled down to *Electronics/Mobile Phones*.

In (Chakrabarti et al., 1997), a typical top-down approach is proposed, where a Bernoulli model is assumed for the document generation. A method based on Fisher's discriminant indices is used for feature selection, which takes place at each category node. The authors compared their approach with a weighted one-level cosine classifier. Their approach showed better results with respect to the micro-averaged recall (Yang, 1999), i.e., 0.66 versus the 0.48 result of the cosine classifier.

Ding et al. (2002) propose a system that classifies products using existing classification standards, such as UNSPSC (UNSPSC.org, 2014). The authors consider three methods and the main focus of the system is the business-to- business environment. For non-hierarchical classification the best result comes from the Naïve Bayes Classifier, i.e., 78%, outperforming the Vector Space Model (VSM) (Salton

et al., 1975) and the kNN algorithm. We hypothesize that this performance can be increased by employing a top-down classification system where feature selection is performed on each node level, separately. For hierarchical classification the highest accuracy obtained is 38%.

D'Alessio et al. (2000) propose an approach where documents are classified only to leaf nodes of the category hierarchy. Classification is done by taking the weighted sum of feature occurrences that should be larger than the category threshold. The innovative contribution of this approach is the possibility of restructuring an initial hierarchy or building a new one from scratch, topics that are outside the scope of our approach.

Wang et al. (1999) identify several issues with the *top-down level-based* approaches. Among other aspects, the *closeness of classification* is not addressed by these approaches, e.g., classifying a mobile phone, which belongs to the category 'Mobile Communications' as 'Electronics' is a smaller error compared to classifying it as 'Clothes'. For this reason, there are a number of approaches proposed in the literature that are designed using the Big-Bang approach. Weigend et al. (1999) propose a two-level classification, where their approach is characterized by a probabilistic framework. Ruiz and Srinivasan (2002) present the design and evaluation of an approach based on the Hierarchical Mixture of Experts model. As our solution, this approach also uses a divide- and-conquer strategy to define smaller categorization problems based on a predefined hierarchical structure. With respect to accuracy, the approach of Ruiz and Srinivasan (2002) shows better results compared to Yang (1996) and Lewis et al. (1996), where a nearest neighbor classifier and a linear classifier are used, respectively.

Sun et al. (2014) propose *Chimera*, an approach for classifying product descriptions that combines learning, rules (created by employees), and crowdsourcing. The authors argue that using rules (in conjunction with learning) is valuable and that research should focus more on helping analysts create and manage these more effectively. Although this approach provides interesting results, it is difficult to compare it with our approach. First, the system relies on significant human effort. For example, the system uses a manually curated list of 20,000 brands in the classification step. Another example is the use of rules and crowdsourcing in the system. This makes it very difficult to compare this approach with ours, which is fully automatic. Second, the focus of the classification task seems to differ from ours. Whereas we propose a system for hierarchical product classification, i.e., using a deep multi-level taxonomy, the Chimera system focuses more on a large scale, flat, taxonomy, consisting

of only two levels. The different scope makes a direct comparison with our solution unsuitable.

We can draw several conclusions from the literature overview. First, besides Ding et al. (2002), none of the related work that aims to solve the same task as our approach focuses on specifically classifying product descriptions. The work in Ding et al. (2002) has some significant limitations, as it compares only three methods (VSM, k-Nearest Neighbor, and Naïve Bayes), and more importantly, the results for hierarchical classification are not promising as the highest accuracy that is obtained is 38%. Second, there is no literature on feature selection for product descriptions. It is not clear which parts of a product description can be used for hierarchical classification of products. The paper aims to fill these gaps by thoroughly evaluating the effects of using the different parts of a product description in combination with well-known feature selection methods.

2.3 The HPC Framework

In this section we present the Hierarchical Product Classification (HPC) framework for classifying product descriptions using a hierarchical product category taxonomy. In the next sections the different components of the HPC framework are discussed in detail. The preparation of the data set is discussed in Section 2.3.1. In Section 2.3.2, we discuss the HPC classification system.

2.3.1 Data set processing

The preparation of the data set is part of the HPC framework, as product descriptions are usually very heterogeneous, especially with respect to the level of detail. For this reason, the HPC framework assumes that a product description has at least the following required parts: (1) product title (text), (2) brand of the product (nominal), (3) price of the product (number), and (4) description of the features of the product (text). To avoid the ambiguity of the term *product description*, we will introduce the term *features description* for description of the features of a product and *product description* will refer to the collection of all four parts (title, brand, price, and features description).

More formally, we define the vocabulary of unique words of all alphabetic parts (i.e., title and features description) of a product description as the vector

$\mathbf{w} = (w_1, w_2, \ldots, w_n)$. A product description d_i is then represented as

$$\left(\mathbf{x}^i_{title}, \mathbf{x}^i_{desc}, p^i, b^i\right) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{B} \tag{2.1}$$

where $\mathbf{x}^i_{title}$ and $\mathbf{x}^i_{desc}$ represent the title and features description of product description i, respectively. These vectors contain the counts for each word from the vocabulary. The term p^i represents the price and $b^i \in \mathbb{B}$ represents the brand, where $\mathbb{R}$ is the set of real numbers and $\mathbb{B}$ is the set of all known product brands. This definition is necessary as the HPC framework addresses each part of a product description differently. Furthermore, the set $C = \{c_1, c_2, \ldots, c_n\}$ represents all known product categories, with a total of n categories. The hierarchy is then represented as

$$H = \{(c_a, c_b) \,|c_a, c_b \in C \wedge c_a \leq c_b\} \tag{2.2}$$

where $\leq$ denotes the subsumption relationship. A set of product descriptions is denoted by $D = \{d_1, d_2, \ldots, d_m\}$, where $d_i \in D$ represents a product description i, i.e., $d_i = \left(\mathbf{x}^i_{title}, \mathbf{x}^i_{desc}, p^i, b^i\right)$. Also, we let the vector $\mathbf{y}$ denote the category mappings of the product descriptions. Consequently, $\mathbf{y}$ contains m values. We assume here that a product belongs only to one category (the most specific one).

In the data preparation process, there are the two main steps that are performed on the content of the product descriptions. First, all stop words are removed from the title and features description. The HPC framework does not define a stop word list, this has to be specified by the user. This enables the user of the system to perform fine adjustments to decide which words are considered stop words and which are not. The removal of stop words eliminates the noise stop words introduce. The accuracy of a classification algorithm often increases after the removal of stop words. Even though in our evaluations we have used a standardized stop word list, a more automated approach could be employed, such as the one proposed in (Wilbur and Sirotkin, 1992).

After the stop words are removed, the remaining words of the product title and features description are stemmed. Many word stemming algorithms exist and the HPC framework does not restrict the usage of any particular stemming algorithm. The default stemming algorithm is the Porter stemming algorithm (Porter, 1997). After the stemming process has completed, we have a set of product descriptions that are prepared for the classification system processes.

2.3.2 Classification system

The *classification system*, the core of the HPC framework, is used to classify product descriptions and it consists of a hierarchy of *classifiers nodes* (a hierarchy similar to the product taxonomy nodes, but without the product taxonomy leaves). A classifier node is a collection of classifiers that are trained on different parts of the product description. The classification system is based on the top-down approach. The reason for choosing the top-down approach is that it can select different features depending on the classifier location in the taxonomy. As mentioned earlier, features 'mobile' and 'phone' may be appropriate for a decision between *Electronics*, *Home & Garden*, and *Sports*, but become less useful when the classifier has to decide between the children of the category *Electronics/Mobile Phones*.

We propose the so-called K-level top-down approach, where $K > 1$. The parameter K is the highest level of the product taxonomy where classifier nodes will be placed. If $K = 2$, then the classification takes place on the first and second level of the taxonomy (i.e., levels 0 and 1). Figure 2.3 shows an example of a classification system with $K = 2$. We can see that the first classifier node decides between the categories 'Electronics' and 'Sports' (level 0). If 'Electronics' is chosen by the first classifier node, then the second classifier node has to classify to either 'Home', 'Communication', 'Knives', 'Mobile Phones', or 'Monitors'. In this case, this is the last

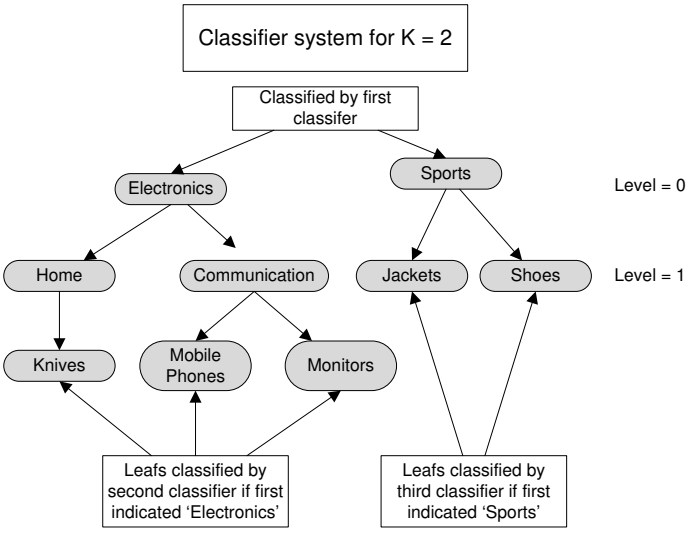

Figure 2.3: K level top-down approach for $K = 2$.

classifier and therefore it classifies to the leaves of the sub-taxonomy. If 'Sports' was chosen, then another classifier (also on level 1) had to decide between 'Jackets' and 'Shoes'. In this case the leaves are also the children of the node 'Sports'.

In the HPC framework, classifier nodes are constructed by using *classifier recipes*. A classifier recipe contains the necessary information to construct a classifier node. It defines which classification techniques and feature selection methods are used for what parts of the product description. It is important to note that each level in the category hierarchy can have its own classifier recipe. Consequently, classifier nodes can differ from level to level in the category hierarchy. Figure 2.4 shows the structure of a classifier recipe. A classifier recipe consists of four components. The first two components each define a feature selector and a text classifier, which are used for the title and the features description. The third component is a classification algorithm that operates on the brand and price. The fourth component is a specialized algorithm that is used in the case that a category is present in the product description. In this chapter, we propose and evaluate such an algorithm. For the brand and price, one can choose any classifier that takes as input one numerical and one categorical variable.

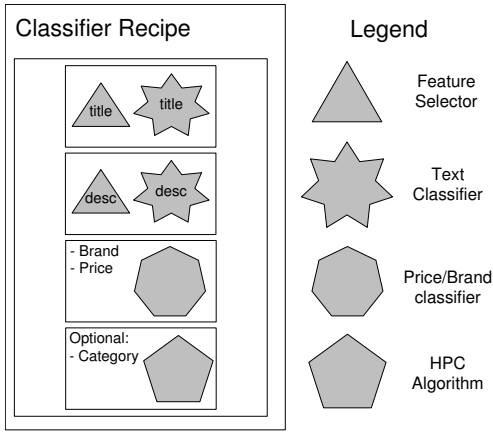

Figure 2.4: The structure of a classifier recipe.

Constructing classifier nodes

For each node, a classifier recipe is used. The classifier node encompasses four classifiers that use different parts of the product description (i.e., (1) title, (2) description, (3) brand and price, and (4), [optional] category). In order to construct a classifier

node, one needs to have a classifier recipe, a training set

$$D \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{B}, \tag{2.3}$$

and a target vector $\mathbf{y}$ where the values are taken from the set of categories C. A product description d_i, as discussed in Section 2.3.1, is represented as

$$\left(\mathbf{x}^i_{title}, \mathbf{x}^i_{desc}, p^i, b^i\right) \in D, \tag{2.4}$$

For both the title and the features description, a classifier recipe defines the feature selector and text classifier (first classifier and second classifier, respectively). A feature selector selects relevant features, given a feature matrix $\mathbf{X}$ (where each column represents one feature) and a target vector $\mathbf{y}$. The text classifier must be a function that takes as input a product description vector $\mathbf{x} \in \mathbb{R}^n$ and outputs one of the categories, predefined by the set C (see Section 2.3.1). In order to construct the classifiers for the title and description in the classifier recipe, the feature selector is first applied to the training sets $\mathbf{x}_{title}$ and $\mathbf{x}_{desc}$. Next, the classifier is trained on the training set through cross validation, to obtain reliable results and to prevent the classifier to overfit the data. The 'best' classifier is chosen, i.e., the one with the highest precision.

The third part of the classifier recipe defines the usage of the brand and the price of the product description for the purpose of classification. This classifier is trained using cross validation on the price and brand training data, and the 'best' classifier is chosen, i.e., the one with the highest precision. A classifier recipe defines a threshold β. If the precision of the best classifier, when considering only instances of the brand provided in a product description, is below β, then this classifier is not used because it is unreliable. This condition is determined at runtime. The recipe also defines a threshold parameter δ, this threshold represents the minimum number of instances in the training data set in order to use this classifier. The δ threshold ensures that this classifier is used only when there is enough data to make a reliable decision for the brand/price combination.

Product categories can have different names across systems but also different hierarchies can be used. For instance, 'Nintendo DS Games' can be a child of 'Games', where on another system it is a child of a more specific category 'Console Games'. The fourth and last part of a classifier recipe defines the classifier of the given category in a product description. It requires an algorithm that takes a string input (the given category) and outputs a list of possible matches, along with the corresponding

scores (similar to an *m-ary* classifier). The score should be between 0 and 1 and the category with the highest score is the one which matches the best. The HPC framework defines a custom algorithm for this purpose. When there is no category given in the product description, this classifier is not used.

In order to meet the above requirements, we propose the Category Mapping algorithm, which is also employed in (Vandic et al., 2012b). The goal of the Category Mapping algorithm is to identify to which existing product category the given product category should be mapped. There are two difficulties with this process. First, one has to deal with syntactic variations and with semantic variations. The syntactic variations are for example singular/plural forms, abbreviations, and typographical mistakes. The semantic variations are synonyms and homonyms. In order to deal with these issues we developed an algorithm which is able to determine the correct category for a product with high precision. Before we give the details of the algorithm, we need to explain existing text similarity measures and other similarity functions that are used in the algorithm.

The Levenshtein distance (Levenshtein, 1966) is a metric for measuring the amount of difference between two strings (i.e., the so-called edit distance). The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. We denote it by alv_{ij}, which is the absolute Levenshtein distance between strings i and j. The HPC framework uses the *normalized* Levenshtein distance, which is a function of the absolute Levenshtein distance. We use the notation lv_{ij}, which is the normalized Levenshtein distance between strings i and j. The normalized Levenshtein distance is defined as

$$\mathrm{lv}\,(i,j) = \frac{\mathrm{alv}(i,j)}{\max(\mathrm{length}(i),\mathrm{length}(j))} \tag{2.5}$$

The normalized Levenshtein distance addresses the issue of short string lengths. If you have two strings, of both length 24, then an absolute Levenshtein distance of 3 is not large. However, with two strings of length 6 this distance is quite large as it is 50% of the tag length. According to the absolute Levenshtein distance these two distances are the same. But the normalized Levenshtein distances are in this case 0.125 and 0.5. This indicates that, according to the normalized Levenshtein distance, the two pairs of strings do not have the same distance, i.e., the first pair is more similar.

The function calcCosineSim (A, B) is used to compute the cosine similarity between two sets of words A and B, and it is defined as follows:

$$\text{calcCosineSim}\,(A, B) = \frac{|A \cap B|}{\sqrt{|A|}\sqrt{|B|}} \tag{2.6}$$

With avgLvSim (A, B), the average Levenshtein similarity between two sets of words can be computed. Using the normalized Levenshtein distance function $\text{lv}\,(i, j)$ for words i and j, we can give the definition of the function avgLvSim (A, B), where A and B are sets of words, as following:

$$\text{avgLvSim}\,(A, B) = \sum_{a \in A} \sum_{b \in B} (1 - \text{lv}\,(a, b)) \frac{\text{length}\,(a) + \text{length}\,(b)}{\sum\limits_{a \in A} \sum\limits_{b \in B} \text{length}\,(a) + \text{length}\,(b)} \tag{2.7}$$

Algorithm 2.1 shows the steps taken to find a matching product category, given a new category name. It requires to have an existing set of categories C. The algorithm also requires to have the set Y of synonyms/syntactic variations of the provided category name. For this purpose, we use WordNet (Miller, 1995) to gather the category synonyms. The process starts by combining the category name, which needs to be mapped to an existing category, with all syntactic variations and synonyms of that category name, obtained from WordNet, in one set Z (line 1). After that, the empty

Algorithm 2.1: The category matching algorithm.

Input : The new category c to be matched to an existing category (text).
Output : The best matching category with the corresponding computed similarity.
Data : The set C (set of categories).
The set Y (synonyms of the new category c).

```
1  Z = Y ∪ {c};
2  S = {};
3  // for each category pair from Z and C, compute their similarity
4  foreach z in Z do
5  |   foreach c' in C do
6  |   |   A = cleanAndSplit(z);
7  |   |   B = cleanAndSplit(c');
8  |   |   S = S ∪ {(c', getCatSim(A, B))};
9  |   end
10 end
11 return {(r, m) ∈ S | ∀(y, n) ∈ S : n ≤ m}
```

set S is created (line 2). In lines 3 through 8, the set S is filled. For each combination between a category from the set Z and a category from the set C, the category names are cleaned. The cleaning of category names is necessary in order to remove any 'noise'. For example, some users write in words 'Camera and Photography' and others might write the abbreviated form 'Camera & Photography'. We solve this issue by replacing occurrences of both 'and' and '&' by a space character. After the category names are cleaned, the similarity between them is computed and added to the set S. The similarity is stored as a pair together with the category from the set C (the set of existing categories). The function that is used to calculate the similarity between two cleaned category names is given by:

$$\text{getCatSim}(A, B) = \lambda \cdot \text{calcCosineSim}(A, B) + (1 - \lambda) \cdot \text{avgLvSim}(A, B) \quad (2.8)$$

where A and B are sets of words. The function $\text{calcCosineSim}(A, B)$ is defined by Equation 2.6 and $\text{avgLvSim}(A, B)$ is defined by Equation 2.7. The sets A and B are obtained by splitting a category name on the space character. This is achieved by using the function $\text{cleanAndSplit}(\cdot)$, which also 'cleans' the category names, i.e., it replaces the word 'and', the word 'or', the character '&', as well as parentheses, comma's, and other special characters, with a space character. When all combinations are processed and the set S is filled, a category needs to be chosen. The category with the highest score in the set S is selected as the matching product category. If multiple categories exist with the highest score, then the average cosine similarity between the feature vectors of each category and the product description is computed, and the category with the highest cosine similarity is chosen. If the highest score is below γ, then this classifier is not used in the process of classification.

Constructing the classification system

In the previous section, we discussed the different parts of a classifier recipe and how we construct one classifier node, given a set of labeled product descriptions. In order to construct a complete HPC classification system, at least one recipe is needed. As we will see in Section 2.4, it is advisable to use different classifier recipes on each level. In this section we discuss the design and implementation of the complete HPC classification system.

Algorithm 2.2 shows the basic steps to construct an HPC classification system. The algorithm starts by creating a classifier for level -1, this level is one level above the level where the top-level categories reside (level 0). The first root level classifier

Algorithm 2.2: The HPC system construction process.

```
1  CF = {cf_-1} // set of classifier nodes with top-level classifier
2  Q = empty queue ;
3  foreach c in C_top do
4  |  enqueue (Q, (c, 0));
5  end
6  i = 0;
7  while notEmpty (Q) do
8  |  (c, L) = dequeue (Q);
9  |  if L = K − 2 then
10 |  |  cf_i = classifier at node c trained on leaf categories under c, using
   |  |   recipe for level L;
11 |  else
12 |  |  cf_i = classifier at node c trained on children categories of c, using
   |  |   recipe for level L;
13 |  |  foreach ch in children(c) do
14 |  |  |  enqueue (Q, (ch, L + 1));
15 |  |  end
16 |  end
17 |  CF = CF ∪ {cf_i};
18 |  i = i + 1;
19 end
```

always exists, independent of the value of K. The root level classifier is added to the set of classifier nodes. This task is performed in lines 1 through 3. From line 4, the algorithm starts a breadth-first traversal through the category hierarchy, creating classifier nodes where necessary and stopping when it hits a leaf category node or the current level has exceeded $K - 2$. The breadth-first traversal function performs a check for each category, starting with the top-level categories. If the level of the category node is equal to $K - 2$, then a classifier is created which is trained on the leaf category nodes of that category. If this is not the case, then an intermediary classifier node is created and trained on the children of the current category node, its children are also added to the queue to be visited. One should note that whenever a classifier node is created, the corresponding classifier recipe for that level (which the category node resides on) is used.

Classification Algorithm

The classification process starts at the root level classifier node, which classifies the product description to one of the top-level categories. Next, the algorithm continues

the classification until the classification results in a category leaf node or the maximum classification depth has been reach (represented by the K parameter). The algorithm chooses the next classifier node based on the previous classification.

In each classifier node, the classification is performed by following a simple voting system. The algorithm classifies a single product description into one of the existing categories from the set C, as defined in Section 2.3.1. The algorithm starts by asking each component to cast a vote on the target category, i.e., each component performs classification, outputting one product category. The next step is to check if there is a category in the set S which has the highest amount of votes. If there exist such a category, then that is the category which is returned as the best match. In the case that no such category exists, the classifier node flags the product description as unclassifiable. In this case, Scenario 3 would be useful to consider, i.e., there is a need for modifying the existing category hierarchy by adding one or more new categories to the hierarchy. However, this scenario is out of the scope of this paper.

The example shown in Figure 2.3 highlights these steps for $K = 2$. Because $K = 2$, there can be only 2 classification steps. The first classifier node decides between the categories 'Electronics' and 'Sports' (level 0) and the second classifier node, regardless of the outcome of the first classifier, will classify the product description to one of the leaves. These leaves are 'Knives', 'Mobile Phones', or 'Monitors' in case the first classifier chose 'Home' or 'Communication', and 'Jackets' or 'Shoes' in case the first classifier chose for 'Sports'.

2.4 Evaluation

In this section, we evaluate the proposed framework and its components. The goal is to find what the best approach is for classifying product descriptions. First, in Section 2.4.1, we give an overview of the data collection process for the evaluation of the HPC framework. We also briefly discuss how we implemented the HPC framework for the purpose of this evaluation. Then, in Section 2.4.2, we give an extensive evaluation of the HPC framework, which includes a discussion of the results for the considered feature selection methods and the classifications algorithms.

2.4.1 Data Collection

For the evaluation of the feature selection methods and classification algorithms, we collected a large data set of product descriptions. The product descriptions are obtained from Amazon.com, using the Amazon Web Services (AWS) API (Amazon.com,

2017a). This process was implemented in Java. The product category taxonomy that is used in the evaluation is constructed from existing Amazon.com categories. Because Amazon.com contains many product categories (around 120,000), we have chosen to use only a subset from all these categories. For the evaluation of the category mapping algorithm, we used data sets from CircuitCity.com and Amazon.com.

There are in total 319 product categories in the constructed product taxonomy, which is a simplified but representative view of the original taxonomy. The categories are located in a hierarchical taxonomy that consists of 4 levels. On the first level, there are 4 categories: 'Electronics', 'Office Products', 'Musical Instruments', and 'Clothing'. In order to have enough data for the training and testing of the classification models, the data set of product descriptions is collected in such a way that the minimum number of products per category is 200. The total number of collected product descriptions is 419,832, with 18,206 unique brands. Unfortunately, only 235,105 products are annotated with a brand. The same holds for the price; only 201,519 product descriptions contain a price. In order to speed-up the retrieval of the product descriptions, a multi-threaded crawler was developed in order to fetch and process the product descriptions.

2.4.2 Results

In this section, we discuss the evaluation of the different aspects of the HPC framework. We first evaluate the HPC framework for scenario 1, i.e., when no category is present in the product description and the product description needs to be classified to one of the existing categories. Then, we focus on the performance of the feature selection and classification algorithm components, as well as the overall performance of the HPC framework. Next, we evaluate the proposed algorithm for scenario 2, i.e., when a category is present in the product description. This consists of an evaluation of the proposed category mapping algorithm.

Although we do not show graphs for every pair of a feature selection algorithm and a classification algorithm (due to the high number of combinations), we stress that we evaluated all possible combinations for both the title property and the features description property. In the text we sometimes refer to these results by numbers instead of graphs.

Feature Selection Approaches

The four feature selection methods that are evaluated are Term Frequency (tf), Mutual Information (mi), Information Gain (ig) and Chi Square (chi). The reason for choosing these feature selection methods is that Information Gain and Chi Square have shown good results in the literature (Yang and Pedersen, 1997). Furthermore, in a general text categorization context, the Term Frequency method performs surprisingly well as well, while the Mutual Information was found to perform badly (Yang and Pedersen, 1997). We want to investigate if these findings also hold when the employed data set consists of product descriptions.

Figures 2.5 and 2.6 show us a comparison of all pairs of the four feature selection methods for the title property and features description property, respectively. Given a feature selection size, each comparison is based on the number of same features that have been selected by the corresponding two feature selection approaches. On the x-axis of the figures, the feature selection sizes are shown. On the y-axis the ratio between the number of same features and the total selected features is shown. For performance reasons, the comparison is performed on a subset of the data set with 3000 product descriptions. The reason why the x-axis range in Figure 2.5 is lower than the x-axis range in Figure 2.6 is because the title property contains less features to choose from than the features description property.

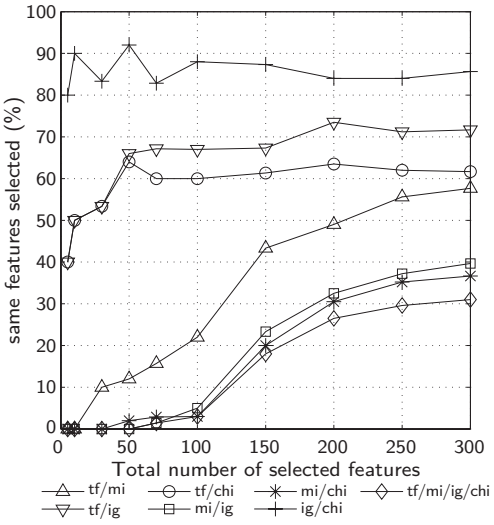

Figure 2.5: Comparing feature selection methods similarity for the title property.

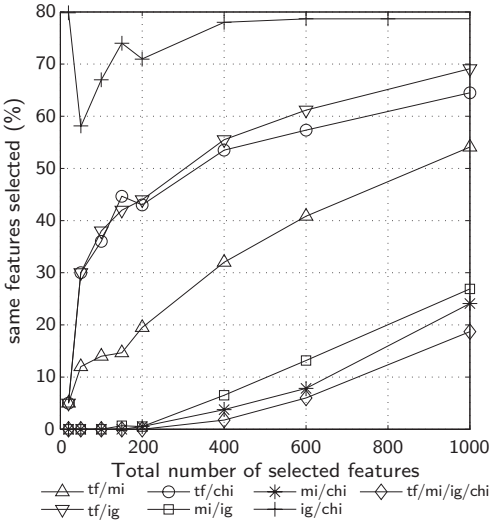

Figure 2.6: Comparing feature selection methods similarity for the features description property.

There are several interesting findings that follow from these two figures. First, we observe that the Information Gain and the Chi Square method have a high overlap in selected features for both the title and features description property. This is in line with findings of other studies, where Information Gain and Chi Square have been found to be highly correlated in terms of accuracy (Singh et al., 2010; Yang and Pedersen, 1997). For the title property, at a feature selection size of 50, more than 45 features are the same ($ratio > 0.90$). Second, the results suggest that the Term Frequency method selects features similar to those from Information Gain and Chi Square only for larger total number of selected features. This indicates that the findings of (Yang and Pedersen, 1997) (i.e., a strong correlation between Term Frequency, Information Gain, and Chi Square) partially applies also to product descriptions. Third, we observe that the Mutual Information shows low ratios for all methods. Only when the feature selection size is 100 or larger, the Mutual Information and Term Frequency method start showing a resemblance in their feature selection process. Last, the results of the comparisons for the features description property, shown in Figure 2.6, indicate that the ratio pair ordering is the same as the ratio pair ordering of the product title. We do notice that most of the ratios are lower than the ones for the product title, which suggests that the product descriptions are more heterogeneous than product titles and that this causes more variation between the methods.

Besides analyzing the relatedness of the different feature selection methods, we also analyzed the performance of each feature selection method. For this we use the accuracy metric from information retrieval, which is in our context equal to the precision because we always classify a product description and we consider this to be a *positive*. Figures 2.7 and 2.8 give us an overview of the accuracy of the feature selection methods for several feature selection sizes. The goal is here to compare feature selection methods, which means that we need to fix the classifier for now. Later on, we will discuss the different combinations of feature selection methods and classification algorithms and their performance. We chose to use the Naïve Bayes classifier in this case because it is fast, and more importantly, it requires no parameters to be set. This is useful because the performance of the Naïve Bayes is then affected only by the used feature selection method. The results for these experiments are obtained by performing a five-fold cross-validation procedure on a data set of 5,000 products and the four main root categories.

From the results we obtain three interesting findings. First, the Information Gain and Chi Square methods are relatively similar and have the highest accuracy. For the title property, as shown in Figure 2.7, we observe slightly higher accuracy values for the Chi Square method, but we have found these differences not to be significant at a 95% confidence level, using a paired t-test. On the other hand, for the features description property, shown in Figure 2.8, the difference between the Information Gain and Chi Square is significant at a 95% confidence level. We can conclude that the Information Gain shows significantly better results for the features description property. The reason for this is most likely that the heterogeneity in the features

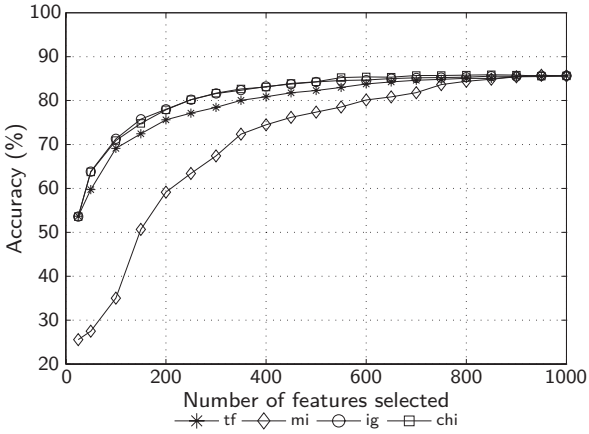

Figure 2.7: Comparing feature selection methods on accuracy for the title property.

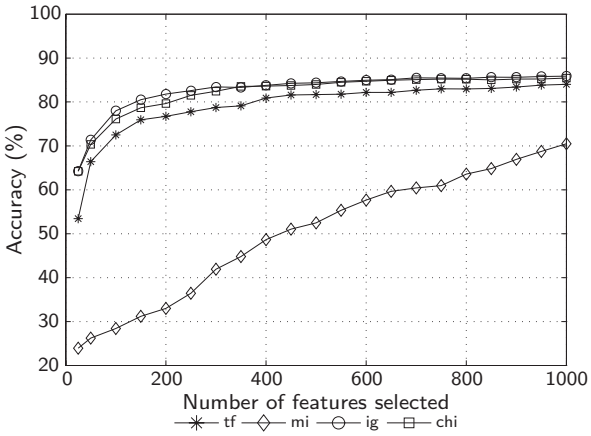

Figure 2.8: Comparing feature selection methods on accuracy for the features description property.

description values makes it difficult for the Chi Square feature selection method to measure the degree of independence between the selected features and the categories. At the same time, with its higher performance, the Information Gain method seems to be able to more easily measure the reduction in entropy when knowing the feature.

Second, we observe that the Term Frequency method performs better than the Mutual Information method. The weakness of the mutual information criterion, i.e., that the values are strongly influenced by the marginal probabilities of terms, is validated by these results. Yang and Pedersen (1997) reported similar findings on a Reuters news data set. The authors found that the Information Gain and Chi Square methods give the best accuracy, but that the Term Frequency method, although performing worse, is highly correlated with the two. The authors suggest to use the Term Frequency method because the trade-off between effectiveness and computational cost, compared to the Information Gain and Chi Square method, is in favor of the Term Frequency method. Our results support this claim, both for the title property as for the features description property.

Third, when considering the general influence of the number of features selected (x-axis), we observe that for the title property, the performance increases more gradually than for the features description property. For example, for the title property, the accuracy does not change anymore at approximately 600 features, while for the features description property, the accuracy barely changes after 400 selected features. This indicates that the title property is more sensitive to the number of selected features than the features description property.

Classification Algorithms

In the previous section, we focused on the feature selection algorithms. In this section we focus on the evaluation of the different classifier components. We only present the results for the k-Nearest Neighbor classifier and the Support Vector Machines classifier. We already presented the results for the Naïve Bayes algorithm in the previous section.

K-Nearest Neighbor classifier. The initial impression of the k-Nearest Neighbor (kNN) classifier is that it does not perform very well. We find that the kNN classification technique is not able to deal appropriately with the product description data. This is different from what other authors have found, where kNN was able to deliver acceptable performance for the purpose of general text categorization (Han et al., 2001).

In order to obtain valid accuracy values, we again performed a five-fold cross validation procedure on the top level categories, with 3000 training product descriptions. Figures 2.9 and 2.10 show the accuracy results for two different configurations with the kNN classifier. The figures show the accuracy for different values of k, i.e., the number of selected neighbors, and for two features selection methods. Overall, the variance of the results is low, i.e., there is not much difference between the feature selection methods, except for the Mutual Information method, which performs far worse than the others. This is illustrated by Figure 2.9, where we can see that the Mutual Information reaches a level of 0.55 accuracy when 200 features are selected. In contrast, we found that other feature selection methods only need 50 features to

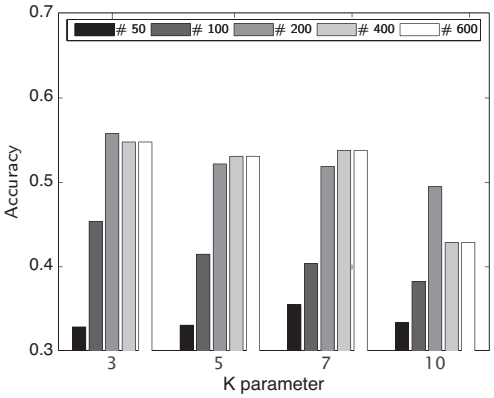

Figure 2.9: Summary of the accuracy on the title property for the kNN classifier, using the Mutual Information features selection method.

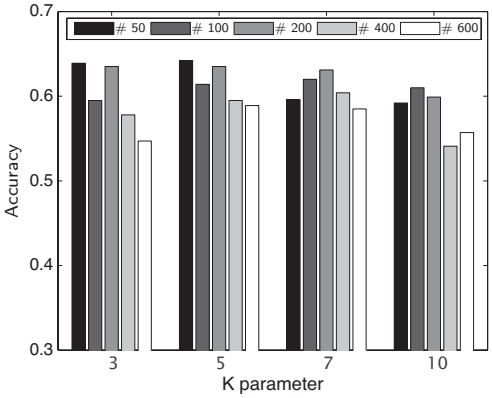

Figure 2.10: Summary of the accuracy on the features description property for the kNN classifier, using the Chi Square features selection method.

obtain such an accuracy (or higher). The same holds for the features description property, i.e., only the performance is even worse, with an accuracy of 0.45 at 400 selected features.

The results also show us that sometimes less is more, e.g., Figure 2.10 shows us that the accuracy is higher when 200 features are selected than when 400 features are selected. The only exception is the Mutual Information, which shows an approximately linear relationship (which eventually diminishes) between the number of selected features and the accuracy.

Furthermore, we observe that when k increases, the kNN classifier gets more sensitive for modeling noise. Figure 2.10 shows us that at $k = 7$, the accuracy for selected feature counts 400 and 600 is higher than at $k = 10$. It seems as if the kNN picks up noise from the extra selected features because of the higher k value.

Support Vector Machines. For the evaluation of the Support Vector machines we performed the same cross validation procedure (five-fold, with 3000 training samples). We have chosen for the SVM one-against-one approach. With the one-against-one approach, one needs to train more SVMs than with the one-against-all approach. Our experiments showed that the one-against-one approach performs better with respect to accuracy. That is why we provide a thorough evaluation of the one-against-one approach (with many parameter combinations). Further, we fixed the choice of the kernel. We choose the Radial Basis function (RBF) kernel (Bishop, 2007) because the RBF kernel was found to give the best performance on text categorization (Joachims, 1998). Following from our experimental setup, we have to

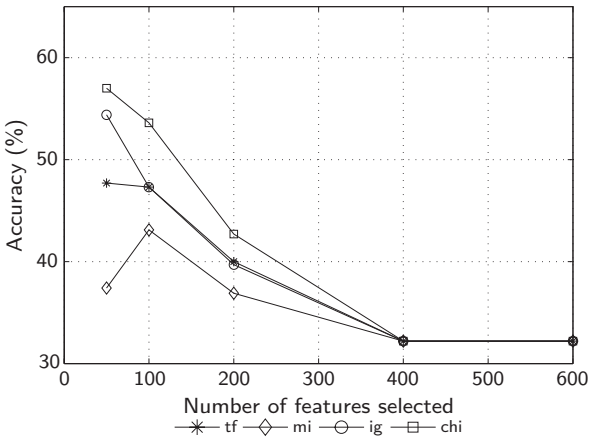

Figure 2.11: Accuracy on the title property for the SVM classifier (with $\gamma = 1$, box constraint = 100).

optimize only two parameters: the box constraint in the dual form notation of the SVM definition, and the γ parameter, which determines the width of the RBF kernel.

The general results of the SVM classifier are better than the kNN method, as we found the highest accuracy to be 78.07%. We observe that the SVM classifier is very sensitive to the two parameters. Typical behavior is shown in Figure 2.11. We notice that a value $\gamma = 1$ is not suitable as the accuracy does not exceed 60% and the accuracy drops as the number of selected features increases. Joachims (1998) reports an optimal γ of 0.6. Our results show that the optimal value of γ for our data set is much larger, somewhere around 50. The higher value of γ indicates that the classification model required for our task is relatively complex, i.e., the influence of single features can be quite large. The reason for this is that the employed data set is to a relatively high degree semi-structured when compared to traditional text classification data sets (such as news articles or blog posts).

The most optimal configuration of the SVM classifier, for the features description property, is for $\gamma = 50$ and box constraint = 100. The accuracy results for this configuration are shown in Figure 2.12. We observe that the overall accuracy is higher for the features description property, most likely because of the extra terms it contains compared to the title property, which helps the classifier to obtain an accurate classification.

A surprising result is that the best accuracy of the Naïve Bayes classifier is higher than the best accuracy of the SVM classifier. As shown in Figures 2.7 and 2.8, for both the title and features description property, the Naïve Bayes classifiers obtain

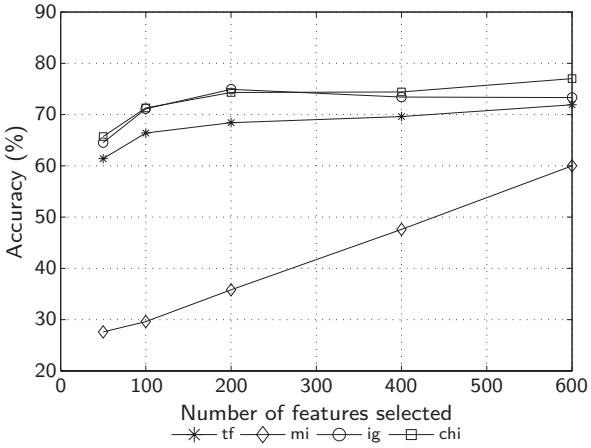

Figure 2.12: Accuracy on the features description property for the SVM classifier (with $\gamma = 50$, box constraint $= 100$).

an accuracy well above 80%, while the accuracy of the best SVM does not exceed 80%. Studies from the past have shown that the Naïve Bayes classifier can achieve comparable performance as the SVM classifier (Huang et al., 2003) and our findings provide further evidence for this claim.

Evaluating the HPC Framework

In this section, we evaluate the HPC framework as a whole. Table 2.1 shows us the accuracy of a $K = 3$ HPC classification system. The results are obtained through cross validation and 3000 training product descriptions. In this case, both for the title and features description, a Naïve Bayes classifier is used with an Information Gain feature selector set to select 400 features. The Price/Brand classifier is implemented using Quadratic Discriminant Analysis (QDA) (McLachlan, 2004). This method aims to classify the category given the brand and the price. The price is first transformed by applying the natural logarithm. This classifier is trained only on categories that contain the brand.

Table 2.1 show the accuracy for each considered level of the category hierarchy. The first level is the level where the root product categories reside. For example, we can see that the Naïve Bayes classifier on the title has achieved an accuracy of 74.55% on the first level. The 'Total' column indicates the total accuracy for a level. This is the accuracy of the system, the other column are referred as the accuracy of the individual classifiers.

Level	Total	Title	Features desc.	Price/Brand
0	82.87%	74.55%	85.48%	13.21%
1	64.76%	69.37%	60.31%	13.24%
2	79.94%	82.65%	83.86%	34.58%

Table 2.1: Accuracy for $K = 3$ classification system, with Naïve Bayes for title and features description, and Information Gain with select count equal to 400.

One might expect that the prices across product categories for a certain brand follow a particular distribution, and are thereby useful as an input for a classifier, however, this assumption fails for the Amazon.com data set. As we can see in Table 2.1, the precision on level 0 is 13.21%. The Price/Brand property has also been evaluated with other classifiers, such as logistic regression and neural networks, although the results remained the same. From these results, we can conclude that the price and brand are not usable in this context.

We can further observe that at level 0, the feature description is the best property to choose, as the accuracy of the classifier on this property is 85.48%. For the second level, the title is the best property to be chosen as it has the highest accuracy on that level. One possible explanation for this is that at the second level, the model words from the title boost the classifier more than they do on the first level, where the classification is more coarse-grained. Finally, on the last level, the classifier on the feature description performs the best.

Table 2.2 shows us an example where the accuracy, on level 0, of the system is higher than the individual classifiers. In this case, the best classifiers is trained on the title property, giving a 74.55% accuracy. Levels 1 and 2 show different behavior than in Table 2.1, as on level 1 the features description property scores better and on the third level the title property.

Level	Total	Title	Features desc.	Price/Brand
0	75.04%	74.55%	72.92%	13.21%
1	64.35%	62.14%	66.00%	15.36%
2	81.42%	83.45%	77.63%	37.23%

Table 2.2: Accuracy for $K = 3$ classification system, with Naïve Bayes for title, SVM ($\gamma = 50$, box constraint=100) for features description, and Information Gain with select count equal to 400.

Last, Table 2.3 shows us a system where only Naïve Bayes classifiers are used on the title and features description and 1000 features are selected by the Information Gain method. This combination gives the best results, with a 83.52% accuracy on level 0. For levels 1 and 2, similar results are obtained as for the classifier in Table 2.1, i.e., the title scores better on level 1 and the features description scores better again on level 2, with an average accuracy of 76.80%.

Level	Total	Title	Features desc.	Price/Brand
0	83.52%	74.55%	86.13%	13.21%
1	64.45%	69.37%	60.80%	13.21%
2	82.42%	82.97%	84.11%	34.58%

Table 2.3: Accuracy for $K = 3$ classification system, with Naïve Bayes for title and features description, and Information Gain with select count equal to 1000.

The Category Mapping Algorithm

For the evaluation of the Category Mapping algorithm, we collected 110 unique categories from CircuitCity.com and Amazon.com. After collecting these categories we manually mapped the collected categories to the Shopping.com categories. We do not provide only one mapping per category, but a list of possible correct mappings for all 110 categories. The first category on the list is the best choice, the second was the second best, etc. Table 2.4 shows some examples of these manual annotations. This manual mapping is used to benchmark our algorithm.

The goal of the algorithm is to map categories as much as possible to categories specified in the first chosen category, but there is always some subjectivity involved. For instance, for the mapping of 'Blu-Ray & DVD Players' one could assign 'DVD players' as first choice while one could also assign 'Blu-Ray Players' as a first choice.

The algorithm for category mapping has only the threshold parameter γ (not to be confused with the SVM γ parameter). In order to obtain the optimal value, we experimented with values between 0 and 1 with a step size of 0.05. Using this procedure, we determined that $\gamma = 0.80$ gives the best results. Table 2.5 shows the results for the algorithm with $\gamma = 0.80$ on the 110 categories. We observe that only 6.37% of the 110 categories are not correctly mapped to one of the corresponding manually chosen Shopping.com categories. This yields that 93.63% of the categories are correctly mapped to one of the corresponding five manually assigned categories.

77.27% of the 110 categories, which is 82.53% of total percentage correctly classified categories (93.63%), are mapped to the first manually chosen category.

Original category	**#1 choice**	**#2 choice**	**#3 choice**	**#4 choice**
Blu-Ray & DVD Players	Blu-ray Players	DVD Players	DVD Drives	Car DVD Players
Networking & Internet	Networking	Networking Hub and Switches	Other Network Devices	
Power Supplies	System Power Supplies			
Webcams	Web Cameras	Digital Cameras		
Memory/Ram	Random Access Memory (RAM)	Computer Memory	Memory Cards	

Table 2.4: An excerpt of the golden standard category mappings.

Manual mapping	**Percentage assigned to**
1^{st} choice	77.27%
2^{nd} choice	6.36%
3^{rd} choice	8.18%
4^{th} choice	0.91%
5^{th} choice	0.91%
1^{th}, 2^{nd}, 3^{rd}, 4^{th}, or 5^{th} choice	93.63%
Misclassification	6.37%

Table 2.5: Results of the category mapping algorithm using $\gamma = 0.80$.

2.5 Conclusions and Future Work

This chapter proposes the Hierarchical Product Classification framework for the purpose of product classification using a product category taxonomy. The framework defines a classification system with K levels that is used to classify a product description to one of the leaves. The innovative part of the framework stems from several aspects. First, the framework uses *classification recipes* to construct classifier nodes. The classification recipes allow for flexible classifiers, i.e., different classifiers and different feature selection can be used on each of the levels of the product category taxonomy. Furthermore, in order to provide a more complete picture of the

components needed to perform high quality product classification, we have evaluated several feature selection methods and classification techniques.

From the obtained results we can draw several conclusions. First, we have found the k-Nearest Neighbor algorithm to be unsuitable as an independent classifier. Besides the computational cost, the accuracy is too low to be useful in practice. Furthermore, we have shown that with our product data set, the Naïve Bayes classifier can perform better than Support Vector Machines, obtaining an average accuracy of 76.80% for product classification. When considering the properties of a product description, we have found that the features description provides better predictors for the top levels but that the title provides better predictors for the lower levels, except for the last level, where the features description gives again better results.

In the case that a product description contains a category, we make use of our proposed Category Mapping algorithm, which is a novel algorithm that makes use of semantic and syntactic matching. The algorithm achieves a precision of 93.63% on a manually mapped test set. It makes use of the average Levenshtein similarity in order to deal with syntactic variations of product categories and the cosine similarity for semantic similarity. WordNet is used to obtain a set of synonyms for each word in the product category, increasing the search space, and thus, the recall.

In future work we want to further investigate the interaction effects between the classification algorithms and the feature selection methods. One approach would be to research different combination strategies at different levels of the product category taxonomy. Another approach would be to use ensemble techniques to combine classifiers, i.e., a classifier on the title, a classifier on the features description, and a classifier on both the title and description. A third option is to use ensemble techniques to combine and evaluate the category mapping algorithm with the previously presented text classifiers.

Chapter 3

Large-Scale Web Product Entity Resolution*

CONSUMERS *are increasingly using the Web to find product information and make online purchases. This is reflected by the ongoing growth of worldwide e-commerce sales. Entity resolution is an important task that supports many services that have arisen because of this growth, such as Web shop aggregators. In this chapter, we propose a scalable framework for multi-source entity resolution. Our blocking approach employs model words to produce blocks that make our solution are highly effective and efficient. An in-depth evaluation, performed using millions of experiments and three large datasets, shows that our model words-based approach outperforms other approaches in most cases. Furthermore, we also evaluate our approach with an imperfect similarity function, from which we conclude that model words-based blocking schemes provide the best blocks with respect to the* F_1*-measure.*

---

*This chapter is based on the article "D. Vandic, F. Frasincar, and U. Kaymak. Scalable Entity Resolution for Web Product Descriptions. *IEEE Transactions on Knowledge and Data Engineering*, 2017, under review."

3.1 Introduction

Over the last few years we have experienced a tremendous growth of online shopping. According to a recent report from Forrester Research, e-commerce spending in the United States will hit approximately $414 billion in 2018 (Mulpuru et al., 2015), which is 11% of the estimated total retail sales in the same year. To keep up with this growth, various online aggregation services have arisen that allow the user to search for products across multiple Web shops. One of the main tasks of such a service is *entity resolution*. Entity resolution can be described as the process of matching entity descriptions, between two or more data sources that describe the very same real-world entities. Performing this task on the Web is harder than in traditional, relational databases. For example, on the Web, we have to deal with highly heterogeneous data and loosely defined schema's. Furthermore, with Web data there are usually many more data sources than with relational databases, e.g., an online aggregation application needs to aggregate information from many different shops.

Previously, we have studied various approaches for product entity resolution on the Web (van Bezu et al., 2015). The focus of the current work is to investigate how the scalability of such entity resolution approaches can be improved using *blocking* schemes. Blocking schemes are used to reduce the number of pair-wise similarities that need to be computed, by assigning each product to one or more 'blocks' (i.e., clusters) and only computing the pair-wise similarity for pairs of products that have at least one block in common (Papadakis et al., 2013). In this chapter, we investigate which blocking schemes work best with product description data on the Web and which part of a product description is the most valuable for the blocking process (i.e., the title or the description). Furthermore, we perform an in-depth analysis of the trade-off between blocking schemes that result in computing a very few pair-wise similarities (more efficient approaches) versus ones with very high recall (more effective approaches).

The novelty of this study stems from several aspects. First, we propose a scalable framework for entity resolution operating on multiple sources at the same time (i.e., multiple Web shops). Second, we propose to use *model words*, previously employed for similarity matching (van Bezu et al., 2015), to create effective and efficient blocking schemes. Third, our findings give insight in which blocking schemes work well with product description data on the Web. Last, using millions of experiments, we perform an in-depth evaluation of our blocking approach, under the assumption of a perfect matching function, but, different from related blocking scheme studies (Baxter et al.,

2003), also using a non-perfect, but well-performing, matching function (van Bezu et al., 2015).

The structure of this chapter is as follows. First, we discuss the related work in Section 3.2. In Section 3.3, we describe a scalable framework for multiple-source entity resolution. Next, in Section 3.4, we evaluate the performance of our blocking approaches and benchmark against a state-of-the-art solution that does not apply blocking. Last, in Section 3.5, we conclude and identify possible further research.

3.2 Related Work

Entity resolution is a well-studied research topic (Benjelloun et al., 2009; Dong et al., 2005; Koudas et al., 2006). Elmagarmid et al. (2007) and Christen (2012) give an overview of this field and highlight the related scalability issues. In the literature, we can find several proposed solutions to scale entity resolution approaches, blocking schemes are the most commonly employed approaches (Christen, 2012). Based on their focus, blocking schemes can be categorized into two broad categories (Papadakis et al., 2013), i.e., methods focusing on *block building* and methods focusing on *block transformations*.

Block building methods focus on producing blocks such that the number of detected duplicates is high and the number of required comparison is low. A straightforward way of blocking in situations where a schema is available, is to simply group the entities based on the values of selected key(s) (Christen, 2012). For example, persons stored in a database might be grouped based on postal code, greatly reducing the number of comparisons. With Web data, there is usually no predefined schema, so it is not possible to preselect particular 'keys' of entity descriptions. Instead, what is often done, is to extract *all values*, usually accompanied with a particular filter (e.g., a stop words filter) to reduce the number of comparisons (Christen, 2012; Papadakis et al., 2013, 2015).

In the literature, we can find various approaches that build on the basic approach of representing every entity by one or more keys and creating groups based on their equality, where pair-wise comparisons are only made within these groups (Papadakis et al., 2015). Suffix array approaches (De Vries et al., 2009) employ suffixes of the extracted values (of fixed lengths), where entities that share a suffix of a value are placed in the corresponding block. Bigrams and q-gram blocking (Baxter et al., 2003; Gravano et al., 2001) work in a similar way, only instead of using suffixes, these methods create clusters of entities that share at least one bi- or q-gram of a

value. The Sorted Neighborhood approach (Hernández and Stolfo, 1998) sorts all extracted values and uses a sliding window of configurable size to create clusters iteratively (at each time moving the window one position down the sorted list). Canopy clustering (McCallum et al., 2000) involves the use of a cheap approximate distance measure to divide the data in overlapping subsets.

Usually the above presented methods perform better when combined with a block transformation step. Block transformation methods aim to improve the effectiveness and/or efficiency by analyzing the produced blocks and transforming them into new ones. One approach that falls into this category is the iterative blocking approach (Kim and Lee, 2010; Whang et al., 2009). Other approaches focus on scaling existing block building techniques, such as Kolb et al. (2012), where a map reduce algorithm is presented for executing the sorted neighborhood approach using multiple passes.

Some approaches focus on scaling existing block building techniques, such as Kolb et al. (2012). The authors of this work propose a map-reduce algorithm for executing the sorted neighborhood approach using multiple passes. In this work, we do not aim to parallelize existing approaches in order to be able to handle larger datasets. Instead, we are interested in blocking schemes that are highly scalable by design. This means that the blocking operation must have linear time complexity in terms of the number of input descriptions. In particular, we focus on methods that output blocks based on solely a single input description, i.e., we do not focus on methods that require the knowledge of the whole dataset. The reason for this is that this makes the approach easily parallelizable, as the block generation can be distributed across nodes in a cluster and the pairs (and corresponding blocks) can be efficiently collected (either offline or directly using a hash-based reduce phase). In order to keep the evaluation fair, we therefore disregard methods such as the Sorted Neighborhood approach (Hernández and Stolfo, 1998) and Canopy clustering (McCallum et al., 2000). Furthermore, a common drawback of all the previously discussed methods is that they do not focus on product descriptions. For our proposed approach, we experiment with a token extraction technique (i.e., *model words*) that proved to work well for similarity computations between product descriptions (van Bezu et al., 2015). Last, different from the discussed solutions, which focus on the two-source entity resolution problem, we address in this chapter is the multi-source entity resolution problem (i.e., handle descriptions of multiple Web shops at the same time).

3.3 Product Entity Resolution

The proposed framework entails three main steps, highlighted in Figure 3.1. First, a blocking scheme is applied in order to reduce the number of pair-wise similarities that have to be computed, explained in Section 3.3.1. Second, a product similarity function is applied to the relevant product pairs, i.e, the pairs that have at least one block in common, covered in Section 3.3.4. Third and last, and discussed in Section 3.3.5, a clustering procedure is applied to identify duplicates.

There are three assumptions that we make in this work. First, the input is a collection of product descriptions from at least two Web shops. These product descriptions are represented by a title and a set of key/value pairs representing features of the product. Unlike in traditional database blocking literature, the key/value pairs do not follow any predefined schema, i.e., Web shops can use completely different vocabularies for describing the same set of features. Second, the entity resolution task concerns itself with multiple sources. Here we assume that we know the source of a product description and that within each source (e.g., a Web shop) there are no duplicates. In a two-source ER setting this would be referred to as 'Clean-Clean Entity Resolution' (Clean-Clean ER) task (Papadakis et al., 2013). Third, because most product datasets are of such size that after computing the similarities, an efficient, non-distributed clustering procedure can be employed to uncover the duplicate entries, we focus on the scalability of the pair-wise computations. The reason for

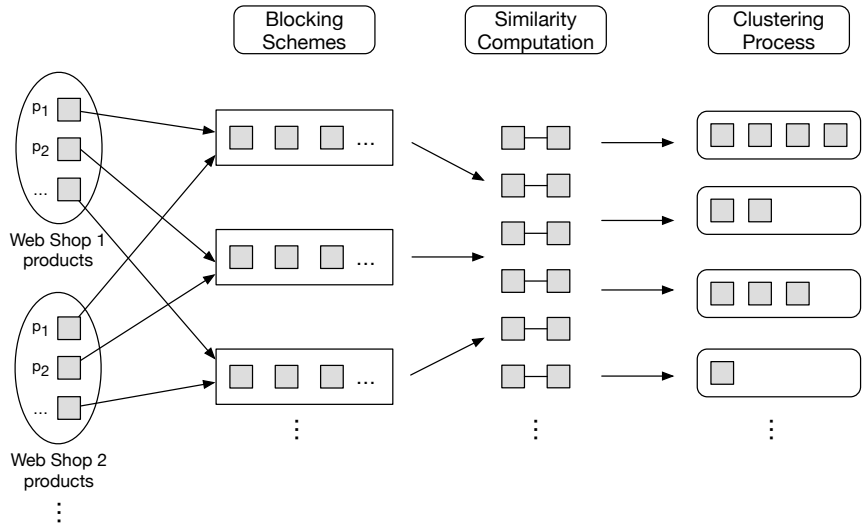

Figure 3.1: A high-level overview of the steps in our entity resolution framework.

this is that, even with moderately sized datasets, the number of pair-wise similarities that need to be computed increases quadratically.

3.3.1 Blocking Schemes

We define a blocking scheme as a function that provides a many-to-many mapping between a product description and a *block*. A block is an entity with a unique ID and is usually represented by (a part of) a value from the entity description.

The set of possible blocks is determined by the employed blocking scheme. In our approach, a blocking scheme consists of the following three components:

- the *source selector*;
- the *text tokenizer*;
- and an optional *block transformer*.

Figure 3.2 shows the details of our blocking framework. First, the source selector component determines from which part of the product description blocks are extracted. Second, the tokenizer component takes as input, from the source selector component, one or more strings, and outputs the union of the extracted blocks from each of these strings. Third, and last, the optional block transformer component transforms the produced blocks B into a new set of block B', with the goal of optimizing the efficiency of the previously produced blocks.

As mentioned previously, we assume that the product descriptions consist of a title and a set of key/value pairs. In this study, we therefore consider a *title source* and a *key/value pairs source*, where the latter will be referred to as the *description source* for brevity. The description source selector refers solely to the values of the key/value pairs; in our study we ignore the keys of the product descriptions. The reason for this is that these are usually not very discriminative, i.e., descriptions across Web shops have many keys in common. On the other extreme, the ones that are not that common, occur rather infrequent and are therefore not very useful.

For the text tokenizers we first consider the *word* tokenizer. This tokenizer first cleans the input string(s) by removing all punctuation characters (e.g., apostrophe, brackets, colons, comma's, etc.) and the dollar sign character, which might prevent some words from being tokenized properly. In the literature, this kind of tokenizer is very common in unstructured datasets, where the schema is unknown and the used values are highly heterogeneous (Papadakis et al., 2013). For example, the title "Philips 4000 Series, 29" - Best Buy" would result in the following extracted

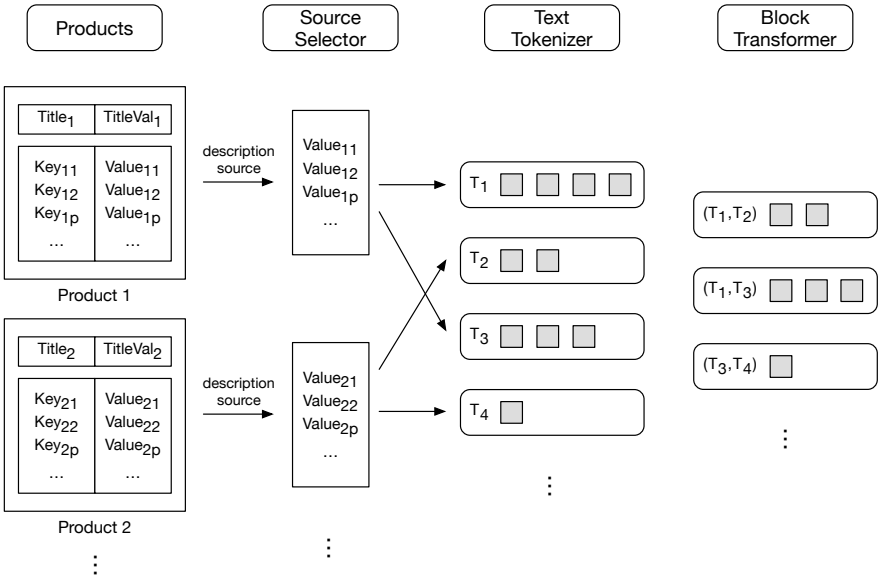

Figure 3.2: A more in-depth look into the details of our blocking approach. Shaded squares represent product descriptions. $T_1, T_2, \ldots, T_n$ represent tokens extracted from a source.

blocks: `philips`, `4000`, `series`, `29"`, `buy`. Besides the regular *word* tokenizer we also consider the q-gram (in particular, trigram) tokenizer (Baxter et al., 2003; Gravano et al., 2001) and suffix array tokenizers (De Vries et al., 2009). These tokenizers use a scheme that performs the tokenizing technique on words extracted from the input string.

The titles of product descriptions can contain a lot of information for products with many technical specifications. For example, for TV's, the titles often contain terms like `1080p` and `100hz`. We refer to such terms as *model words* and they are defined as words that have at least one digit character and at least one character that is not a digit. The following are examples of extracted model words:

- `Philips 4000 Series 29" Class LED 720p 60Hz HDTV 29PFL4508F7`
  yields model words `720p`, `60hz`, `29pfl4508f7`, `29"`
- `Panasonic - 42" Class/Plasma/720p/600Hz/HDTV`
  yields model words `720p`, `42"`, `600hz`

It has been shown that these model words can be effectively used to obtain high accuracy entity resolution (van Bezu et al., 2015). We therefore propose the *model words tokenizer* that extracts these model words from the input string(s). For the previous example, i.e., "Philips 4000 Series, 29" - Best Buy", this would mean that we extract the block `29"`.

Last, for our approach, we consider the use of a *combinations* block transformer. This transformer takes as input the blocks from a tokenizer and outputs all the k combinations (sets of k elements) of these blocks. Again, with the previous example and a particular tokenizer, a $k = 2$ *combinations* block transformer would yield all pairs of the blocks that the tokenizer outputted. With $k = 3$, it would yield all triplets of the blocks that the tokenizer outputted. This transformation will result in a reduction of the total number of similarity-pairs that need to be computed, i.e, the longer the combination, the fewer products will match the combination.

The above discussed components, i.e., the source selector, the text tokenizer, and the optional block transformer components, form together a blocking scheme. Once these three component have been chosen, the blocks are generated per given product description. The corresponding product is then mapped to each of the generated blocks.

3.3.2 Blocking Schemes Aggregators

Similar to the transformers for the blocks, on the level of blocking schemes, we consider *aggregations of blocking schemes* that might improve the effectiveness and/or efficiency. For this purpose, we consider two type of aggregations: (1) the *union* of the output blocks, focusing on improving effectiveness, and (2) the *conjunction* of the output blocks, moving the focus on improving the efficiency of the outputted blocks.

The *union* aggregator blocking scheme takes two blocking schemes and outputs mappings that are outputted by *either* of the two blocking schemes. For example, if blocking scheme x maps entity $p1$ to block $b1$ and blocking scheme y maps entity $p1$ to block $b2$, then this blocking scheme would map $p1$ to both $b1$ and $b2$.

The *conjunction* aggregator blocking scheme takes two blocking schemes, say $s1$ and $s2$, and combines their outputs in the following way. First, the mappings are computed using $s1$. Then, for each of the blocks in the mapping, the entities mapped to the block are fetched. The blocking scheme $s2$ is called on these entities, yielding a new set of mappings. The final outputted mappings are a combination of the block from $s1$ and the blocks from $s2$.

Finally, we also consider the *all pairs blocking scheme*, which is actually a scheme that does not perform blocking at all, it considers all pairs. The other approaches are evaluated w.r.t. the percentage of pairs computed compared to the all pairs method.

3.3.3 Blocking Schemes Identifiers

Throughout this paper we adhere to a blocking scheme naming convention that represents a combination of the previously discussed components. A blocking scheme is represented in the form of `$source.$tokenizer`, where the `$transformer` is optional (words that start with a dollar sign represent a variable).

For `$source`, we use `t` for the title source and `d` for the description source. We use `wo` for the word tokenizer, `3q` for the qgram (trigram) tokenizer, `sx` for the suffix array tokenizer, and `mw` for the model words tokenizer. For example, `t.mw` would represent a blocking scheme that extracts model words from the title and `d.wo` a blocking scheme that extracts words form the description source. Additionally, when a combinations block transformer is applied, we append k to the name of the tokenizer. For example, `d.mw3` represents the blocking scheme that extracts model words from the description source and forms triples of these to output blocks.

The union and conjunction block transformers are represented as `a_$operator_b`, where `a` and `b` are blocking schemes and `$operator` can be either `+` (for union transformations) or `x` (for conjunction transformations). For example, `t.mw3_x_d.mw2` represents the conjunctive combination of two blocking schemes that extract model words triples from the title and model word pairs from the description. This means that this model will output only blocks that are both outputted by the blocking schemes.

3.3.4 Similarity Computation

In previous work, we have focused on computing similarities between product descriptions (van Bezu et al., 2015). The approach we employ in this chapter for computing the similarities is the state-of-the-art MSM approach from our previous study. MSM is able to compute similarities for products that are represented in a semi-structured data model, e.g., tabular Web product information.

Algorithm 3.1 shows a high-level overview of how the product similarity is computed. In Algorithm 3.2 we see the function KeySim(a, b), which is used to compute the similarity between product descriptions a and b based solely on the product attributes. This function is used by Algorithm 3.1 to compute the final product

Algorithm 3.1: MSM Similarity.

Required functions:

- `KeySim`(a, b) computes the similarity between two product descriptions a and b based solely on the product attributes (described in Algorithm 3.2);
- `titleSim`(t_a, t_b) gives the similarity between title t_a and t_b;
- `mw`(W) extracts the *model words* for a given set of words W;
- `mwSim`(mw_a, mw_b) the similarity between model words mw_a and mw_b.

```
// Computes similarity between product descriptions a and b
def ProdSim(a, b):
    (m, I, J, keySim*) := KeySim(a, b)
    // m is the number of key matching keys between a and b
    // I is the set of keys from a that do not match
    // J is the set of keys from b that do not match
    // keySim* is similarity between the keys of a and b
    // extract model words from set of keys that do not match
    I_mw := mw(I)
    J_mw := mw(J)
    // compute the model words and title similarity
    mwSim := mwSim(I_mw, J_mw)
    titleSim := titleSim(a.title, b.title)
    if titleSim = 0 then
        // μ is the weight for the title similarity
        θ1 := m/min(|a.keys|, |b.keys|) θ2 := 1 − θ1
    else
        θ1 := (1 − μ) · m / min(|a.keys|, |b.keys|) θ2 := 1 − μ − θ1
    end
    // final similarity
    sim* := θ1 · keySim* + θ2 · mwSim + μ ∗ titleSim
    return sim*
```

description similarity. The similarity function of MSM takes as input two product descriptions and outputs a similarity between 0 and 1. The main idea of the approach is to first compute a similarity based on the keys of two product descriptions (i.e., attributes). Then, with the remaining, unmatched keys, we compute a similarity based on the proposed *model words*. Model words are defined as words that contain at least one alphanumeric character. Last, we compute a title similarity and combine the three similarities into one final weighted average similarity. For the titleSim() function we employ a cosine-based similarity. The keySim() and keysMatch() func-

Algorithm 3.2: MSM Key Similarity.

Required functions:

- `clean`(w) removes punctuation characters at end of word w;
- `keysMatch`(k_a, k_b) determines whether two keys represent the same attribute;
- `keySim`(k_a, k_b) gives the similarity between keys k_a and k_b;
- `valSim`(k_a, k_b) gives the similarity between the values of keys k_a and k_b;
- `titleSim`(t_a, t_b) gives the similarity between title t_a and t_b;
- `mw`(W) extracts the *model words* for a given set of words W;
- `mwSim`(mw_a, mw_b) the similarity between model words mw_a and mw_b.

```
// Computes the similarity between two product descriptions
def KeySim(a, b):
    m := 0 // number of matches
    w := 0 // weight of matches
    I := a.keys and J := b.keys // keys without a match
    sim := 0
    foreach k_{i,a} in a.keys do
        k_{i,a} := clean(k_{i,a})
        foreach k_{j,b} in b.keys do
            k_{j,b} := clean(k_{j,b})
            if keysMatch(k_{i,a}, k_{j,b}) then
                I := I \ k_{i,a}
                J := J \ k_{j,b}
                keySim := keySim(k_{i,a}, k_{j,b})
                valueSim := valSim(k_{i,a}, k_{j,b})
                sim := sim + keySim * valueSim
                m := m + 1
                w := w + keySim
            end
        end
    end
    keySim* := 0
    if w > 0 then
        keySim* := sim / w
    end
    return (m, I, J, keySim*)
```

tions are based on the q-gram similarity measure. For more details we refer the reader to van Bezu et al. (2015).

3.3.5 Clustering

After the similarities between the relevant entity pairs have been computed, we apply a clustering step to discover the duplicate entities. The reason for this is that we have multiple sources and the similarity function is not transitive. In other words, if one has A sim B, B sim C, then this does not imply A sim C. Using a clustering approach, these transitive similarities can be taken into account.

As part of our framework, the clustering method takes as input the product description similarities and outputs clusters of products in which its members are considered to be duplicates. Because our approach supports multiple sources and we assume that within each source there are no duplicates, the algorithm should not cluster products that are originating from the same Web shop.

The MSM similarity (van Bezu et al., 2015) is used in an approach where an adaptation of hierarchical clustering is employed. The main idea of this clustering approach is to use single linkage in all cases except for those where a infinite distance is encountered, in which case complete linkage is applied instead. These infinite distances are assigned to product description pairs originating from the same Web shop. Therefore, it is unlikely that the modified linkage criterion in this approach will assign such descriptions into the same cluster.

3.4 Evaluation

For the evaluation of our framework, we have used the following three datasets*:

- *4shops* - consists of 1,624 descriptions of televisions, obtained from Amazon.com, Newegg.com, BestBuy.com, and TheNerds.net;
- *abt-buy* - consists of 2,175 descriptions of various electronics products;
- *amz-ggl* - consists of 4,591 descriptions of various software products.

First, in Section 3.4.1, we evaluate the effectiveness and efficiency of the proposed blocking approach. Then, in Section 3.4.2, we analyze how the performance of the proposed entity resolution algorithm is affected by the considered blocking schemes.

3.4.1 Blocking Evaluation

We evaluate our blocking approach using three measures that are commonly used in literature (Christen, 2012; Papadakis et al., 2013, 2015). These measures are

*available from `https://goo.gl/cexJfJ` and `https://goo.gl/hm0bhD`

computed based on the pairs that need to be considered as a result of a blocking scheme. First, we consider a measure of efficiency, which is referred in literature as *pairs quality* (PQ). It is defined as

$$PQ = \frac{\text{found duplicates}}{\text{executed comparisons}}$$

This can be considered as the precision measure from information retrieval. Second, we consider a measure of effectiveness, also known as the *pairs completeness* (PC). Essentially, it is the recall measure from information retrieval and it is defined as

$$PC = \frac{\text{found duplicates}}{\text{total number of duplicates}}$$

Last, we consider the reduction rate (RR), defined as

$$RR = 1 - \frac{\text{executed comparisons with blocking}}{\text{executed comparisons without blocking}}$$

This is the reduction in the percentage of pairs that need to be considered after applying a blocking scheme, compared to the total number of pairs (without applying a blocking scheme). The total number of pairs excludes pairs of product descriptions that are from the same Web shop, as we assume that we are dealing with 'Clean-Clean ER' and that in most contexts, the source of the product description is known.

Considered configurations

These are the configurations that we consider:

1. `t.{wo,mw}{2..4}` and `t.{wo,mw}` (6 combinations)
2. `t.{3q,sx}` (2 combinations)
3. `d.{wo,mw}{2..4}` and `d.{wo,mw}{2..4}` (6 combinations)
4. `d.{3q,sx}` (2 combinations)
5. `a_+_b` where `a` and `b` are unique pairs from (1), (2), (3), and (4) (120 combinations)
6. `a_x_b` where `a` and `b` are unique pairs from (1) and (3), respectively (66 combinations)

The total number of considered configurations is therefore 202 (16+120+66). For each of these methods, and the three considered datasets, we computed the previously discussed measures on 40 bootstraps, where each bootstrap is a stratified random sample of the original data. Next, we only kept methods that had an average PC higher than 0.50 and an average RR measure greater than 0.50 (measured across datasets). For these methods we then compute the average PC and average RR for each dataset. Then, for each dataset, we split the methods into three equally-sized groups ('low', 'medium', and 'high'), which is based on the average PC where 'high' represents the group having the highest PC measure. Finally, for each group in each dataset, we select only the top 3 methods w.r.t. the RR.

Figures 3.3, 3.4, and 3.5 show the trade-off between the PC and RR measures for each group and each dataset. Each dot in a graph represents the obtained performance for a particular bootstrap. Furthermore, Tables 3.1, 3.2, and 3.3 show for each top method in each dataset the mean and standard deviation for PC, RR, PQ, block count, and blocking duration. For the PC measure, we have used paired t-tests with Bonferroni corrected p-values to determine if differences are statistically significant. For sake of brevity, in our discussions, we will only highlight the non-significant differences (because most of the differences are significant).

What we first notice is that the suffix array tokenizers and trigram tokenizers do not appear in the results as top-performing methods. This might suggest that traditional tokenizing techniques are not efficient enough for Web data, such as the product descriptions in the three considered datasets. Furthermore, from Fig-

Scheme	dataset	PC	RR	PQ	Block count	Blocking duration (ms)
`t.mw3_x_d.mw`	4-shops	0.74/0.04	0.98/0.00	0.07/0.01	49,331.45/2,122.70	55.00/18.50
`t.mw3_x_d.wo`	4-shops	0.77/0.04	0.98/0.00	0.05/0.00	134,687.58/5,640.74	144.90/12.94
`t.mw3`	4-shops	0.78/0.04	0.98/0.00	0.04/0.00	2,539.20/98.60	2.28/3.77
`t.mw_x_d.wo`	abt-buy	0.52/0.02	1.00/0.00	0.39/0.04	16,998.68/504.64	20.75/4.58
`t.wo4_x_d.wo`	abt-buy	0.65/0.02	1.00/0.00	0.47/0.03	1,039,981.45/62,697.68	1,441.33/217.08
`t.wo4`	abt-buy	0.71/0.02	1.00/0.00	0.44/0.03	60,376.30/7,362.27	54.43/22.35
`t.wo3_x_d.mw3`	amz-ggl	0.54/0.02	1.00/0.00	0.41/0.05	90,529.18/11,833.81	209.98/68.36
`t.wo3_x_d.mw4`	amz-ggl	0.59/0.02	1.00/0.00	0.41/0.05	133,298.35/38,686.92	262.43/103.16
`t.wo3_x_d.wo`	amz-ggl	0.63/0.02	1.00/0.00	0.42/0.04	1,151,049.95/103,369.37	1,453.68/173.81

Table 3.1: Blocking evaluation results for the considered blocking methods in '**low** category' (w.r.t to the PC measure). For each measure, we aggregated the scores achieved over the 40 bootstraps. Each cell shows (1) the mean and (2) the standard deviation.

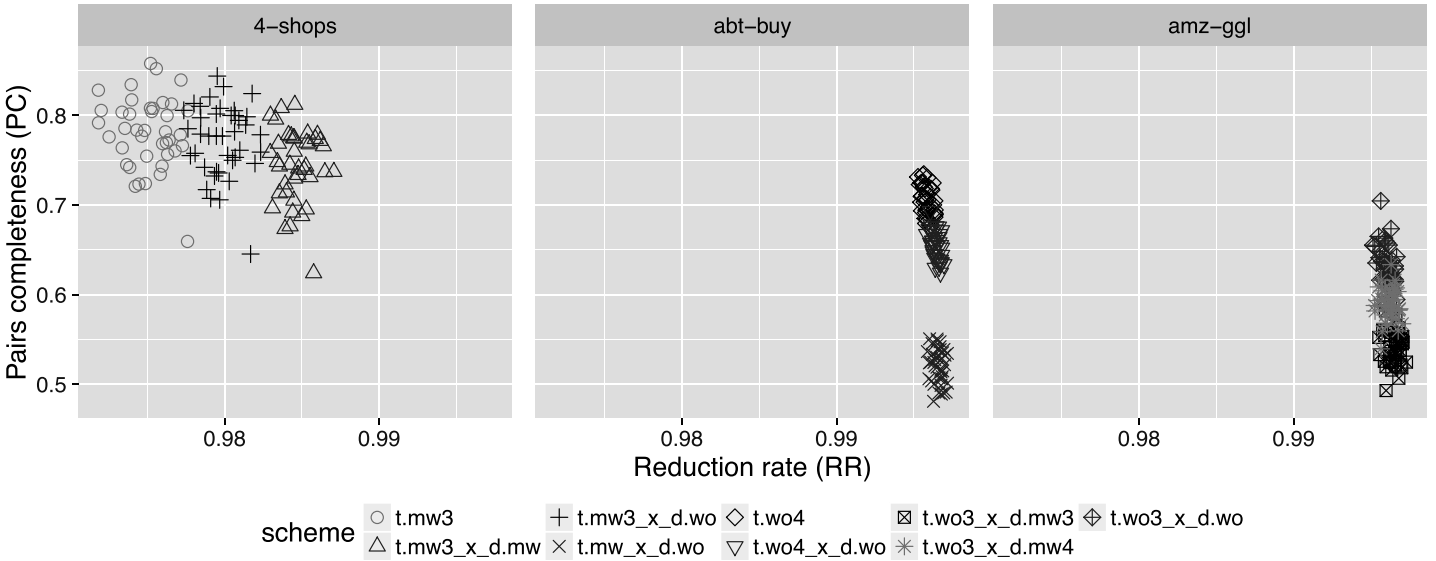

Figure 3.3: Scatter plots for the 'low' PC category blocking methods showing the trade-off between the pairs quality (PQ) and pairs completeness (PC), where each point represents one of the 40 bootstraps.

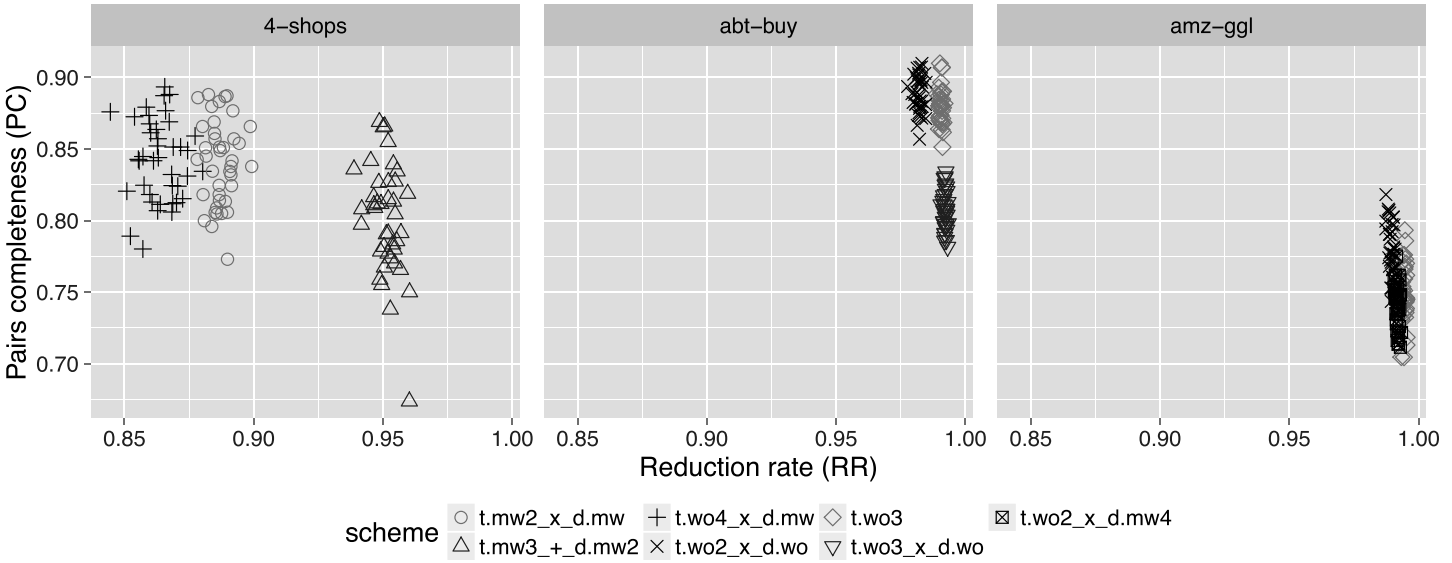

Figure 3.4: Scatter plots for the 'medium' PC category blocking methods showing the trade-off between the pairs quality (PQ) and pairs completeness (PC), where each point represents one of the 40 bootstraps.

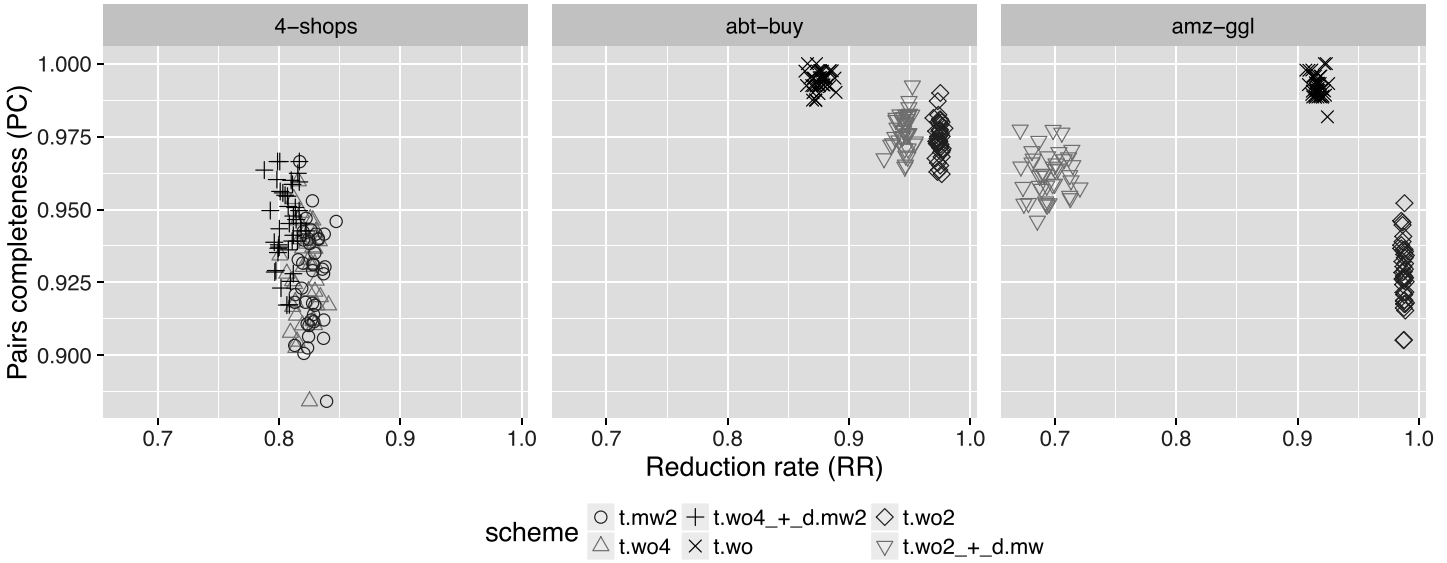

Figure 3.5: Scatter plots for the 'high' PC category blocking methods showing the trade-off between the pairs quality (PQ) and pairs completeness (PC), where each point represents one of the 40 bootstraps.

Scheme	dataset	PC	RR	PQ	Block count	Blocking duration (ms)
t.mw2_x_d.mw	4-shops	0.84/0.03	0.89/0.00	0.01/0.00	45,208.00/1,527.39	55.78/18.08
t.mw3_+_d.mw2	4-shops	0.80/0.04	0.95/0.00	0.02/0.00	5,268.55/171.25	11.60/19.34
t.wo4_x_d.mw	4-shops	0.84/0.03	0.86/0.01	0.01/0.00	2,674,844.73/154,913.98	4,064.73/607.06
t.wo3_x_d.wo	abt-buy	0.81/0.01	0.99/0.00	0.27/0.02	611,670.70/25,527.48	816.03/83.14
t.wo2_x_d.wo	abt-buy	0.89/0.01	0.98/0.00	0.13/0.01	248,044.33/7,655.92	325.93/27.40
t.wo3	abt-buy	0.88/0.01	0.99/0.00	0.24/0.02	31,452.78/1,760.35	27.43/10.56
t.wo2_x_d.mw4	amz-ggl	0.74/0.02	0.99/0.00	0.24/0.02	95,143.25/36,259.98	188.58/88.68
t.wo3	amz-ggl	0.75/0.02	0.99/0.00	0.36/0.04	29,719.25/2,441.90	27.58/10.95
t.wo2_x_d.wo	amz-ggl	0.78/0.02	0.99/0.00	0.20/0.02	537,230.35/32,067.84	688.13/165.69

Table 3.2: Blocking evaluation results for the considered blocking methods in '**medium** category' (w.r.t to the PC measure). For each measure, we averaged the scores achieved over the 40 bootstraps. Each cell shows (1) the mean and (2) the standard deviation.

Scheme	dataset	PC	RR	PQ	Block count	Blocking duration (ms)
t.mw2	4-shops	0.93/0.02	0.83/0.01	0.01/0.00	1,902.48/60.47	2.20/3.50
t.wo4	4-shops	0.93/0.02	0.82/0.01	0.01/0.00	157,152.50/10,788.26	156.70/97.48
t.wo4_+_d.mw2	4-shops	0.95/0.01	0.81/0.01	0.01/0.00	159,881.88/10,788.41	255.70/111.22
t.wo	abt-buy	0.99/0.00	0.88/0.01	0.02/0.00	1,573.63 /40.29	5.93/11.33
t.wo2	abt-buy	0.97/0.01	0.97/0.00	0.10/0.01	11,140.50/385.18	10.70/6.74
t.wo2_+_d.mw	abt-buy	0.98/0.01	0.95/0.01	0.05/0.00	12,006.83/400.41	23.83/12.56
t.wo	amz-ggl	0.99/0.00	0.92/0.00	0.03/0.00	1,305.78/40.24	4.75/11.16
t.wo2	amz-ggl	0.93/0.01	0.99/0.00	0.20/0.02	9,706.18/450.01	10.28/8.47
t.wo2_+_d.mw	amz-ggl	0.96/0.01	0.70/0.01	0.01/0.00	10,121.03/455.96	71.35/55.31

Table 3.3: Blocking evaluation results for the considered blocking methods in '**high** category' (w.r.t to the PC measure). For each measure, we averaged the scores achieved over the 40 bootstraps. Each cell shows (1) the mean and (2) the standard deviation.

ures 3.3, 3.4, and 3.5, we can see that for the 'low' group, the methods are all in the upper range w.r.t. the reduction rate (RR), with the lowest average being 0.98 for the '4-shops' dataset. The average PC in this group varies between 0.52 and 0.78 across datasets. All differences w.r.t. the PC measure are found to be statistically significant (across datasets), except the difference between `t.mw3_x_d.wo` and `t.mw3`, and `t.wo3_x_d.mw3` and `t.mw_x_d.wo`. Taking into account, besides the PC and RR measures, the number of blocks and the duration, we would argue that, for this group, `t.mw3` provides the optimal balance between speed and recall.

For the 'medium' group we can see that there is more variation w.r.t. the reduction rate (RR), ranging from an average of 0.86 to 0.99. For the PC measure, the variation is similar compared to the 'low' group but the range is smaller, with averages going from 0.74 to 0.89. For this group, the difference between the PC measure is for the following methods not statistically significant (across datasets):

- `t.wo4_x_d.mw` and `t.mw2_x_d.mw`
- `t.wo2_x_d.wo` and `t.mw2_x_d.mw`
- `t.wo2_x_d.wo` and `t.wo4_x_d.mw`
- `t.wo3` and `t.mw2_x_d.mw`
- `t.wo3` and `t.mw3_._d.mw2`
- `t.wo3` and `t.wo2_x_d.wo`
- `t.wo3_x_d.wo` and `t.mw3_._d.mw2`
- `t.wo3_x_d.wo` and `t.mw3_._d.mw2`
- `t.wo3_x_d.wo` and `t.mw3_._d.mw2`

We can also notice that `t.wo4_x_d.mw` generates a large number of blocks. This is caused by the fact that the word tokenizer produces many tokens and that the high number of combinations lead to an exponential growth in the number of produced blocks. Looking at the average durations for the '4-shops' dataset, we can see that two schemes stand out because of their low duration: `t.mw2_+_d.mw` and `t.mw3_+_d.mw2`. For this group, we notice that the model words tokenizer is part of the best performing schemes for the '4-shops' dataset. For the *amz-ggl* dataset, the cleaning word tokenizers seems to achieve better efficiency. This is as expected, because the model words work best with technical descriptions, which are most common among consumer electronics.

In the 'high' group, most methods are able to achieve a relatively high reduction rate, except for the '4-shops' dataset. The reason for this is that there are multiple Web shops and thus, there is more variation in the descriptions. As the results for the other two groups demonstrated, the '4-shops' dataset seems to be the most difficult for the methods to achieve a high reduction rate. With respect to the PQ measure, all methods are found to statistically differ, except for `t.wo4` and `t.mw2`, and `t.wo2` and `t.wo4_._d.mw2`. Given these results, for the '4-shops' dataset there is not a

clear winner. For the other two datasets, the scheme `t.wo` seems to give the best performance w.r.t. PC and RR.

The results from these experiments demonstrate the scalability of our approach. For example, in Table 3.3, which shows the results for the 'high' group, we can see that the method `t.wo2_+_d.mw` takes on average 71.35 ms for *amz-ggl* and 23.83 ms for *abt-buy* for a bootstrap sample (an increase of factor 2.99). Given the fact that the *amz-ggl* data set has 4,591 descriptions and the *abt-buy* data set has 2,175 descriptions (a factor 2.11 difference in size), we can see that the increase in blocking duration is very close to linear.

3.4.2 Entity Resolution Evaluation

In order to validate the found results in the previous section, we perform another evaluation, i.e., one that assesses the performance of the entity resolution algorithm when a blocking scheme is applied. For this part, we used only the '4-shops' dataset as this dataset has the required characteristics for the MSM algorithm (unstructured key/value data). The other two datasets have only a single description field, making a comparison with these datasets not possible. Different than the blocking evaluation, the entity resolution evaluation is implemented on top of a cluster computing framework, i.e., the Apache Spark framework (Meng et al., 2016; Zaharia et al., 2010). Because we had access to a large cluster, we were able to run 100 bootstraps of the MSM algorithm with each of the blocking schemes. For MSM we tried 375 different parameter configurations. Previously, in Section 3.4.1, we have explained which are the scheme configurations that we consider. Using the considered configurations, we arrive at a total number of experiments of 7,575,000 (100 bootstraps $\times$ 375 MSM instances $\times$ 202 blocking scheme configurations). We used a cluster of 640 cores and an additional caching layer for computing the MSM similarities.

Table 3.4 shows the results of the top-3 methods and the all-pairs method w.r.t. the F_1-measure using the MSM matching function. We found that only the methods `t.mw3_x_d.wo` and `t.mw3` are statistically different w.r.t the F_1. With respect to the average duration per bootstrap sample, we can report that all measured mean differences are statistically significant, except for the difference between `t.mw3_x_d.wo` and `ap`, i.e., the top-2 methods exhibit the same performance. However, considering the relatively small difference between the top-3 methods (i.e., `t.mw3_x_d.wo`, `t.mw3_x_d.mw`, and `t.mw3`), we expect all top-3 methods to provide reasonably good performance.

Scheme	F_1-measure	Precision	Recall	Mean sample duration (ms)
`t.mw3_x_d.wo`	0.5382	0.5411	0.5380	447.15
`t.mw3_x_d.mw`	0.5376	0.5511	0.5272	221.68
`t.mw3`	0.5368	0.5389	0.5376	49.52
`ap`	0.5248	0.4723	0.5922	406.8

Table 3.4: The top-3 best performing methods (w.r.t. the F_1-measure) and the all-pairs method for the entity resolution evaluation, showing the average of each of the measures, taken over the 100 considered bootstraps. The scheme `ap` stands for the scheme that does not apply any blocking but instead just considers all pairs.

It is interesting to see that the all pairs scheme does not give the best performance and that the best performing methods do not come from the group that scores the highest on PC. At first, this seems counterintuitive, however, there is a reasonable explanation for this. It seems that the blocking schemes are able to factor a significant amount of pairs that would lead to false positives. This is also reflected by the increased precision. Apparently, the decrease in recall (due to the computation of less pairs) is less than the increase in precision, and thus resulting in a higher F_1-measure. This is a very interesting observation since most studies on blocking schemes are performed with the assumption of the availability of a perfect matching function. The found results in these experiments suggest that with imperfect matching functions, such as MSM, the best performing methods are not necessary the ones with the highest PC measure.

Further experiments are needed to generalize the findings regarding the contribution of the considered tokenizers. However, the results from our experiments do suggest interesting research directions. For example, the top-3 methods all utilize the model words tokenizer. This might suggest that for product descriptions containing technical specifications, such as the '4-shops' dataset, model word tokenizers are more suitable than word tokenizers.

3.5 Conclusion

In this chapter, we have proposed a scalable approach for multi-source entity resolution using various blocking schemes. Our approach consists of three main components: (1) a blocking scheme, (2) a product similarity function, and (3) a clustering procedure. Various blocking schemes, which operate on the title, the description, or both, are evaluated with a perfect similarity function, as well as an imperfect, but

well-performing one (van Bezu et al., 2015). The framework has been evaluated using an extensive set of experiments and three large datasets used commonly in entity resolution studies.

The most important finding with regards to the blocking evaluation is that experimental setups that evaluate blocking schemes with only a perfect matching function are most likely not sufficient, i.e., one should also consider imperfect matching functions. Furthermore, the results suggest that for our similarity function and clustering procedure, which together achieve an F_1-measure of 0.525 when using all pairs, blocking methods that compute very few pairs achieve a higher performance that ones that have a high PC. In particular, we find that the blocking scheme that extract model word triples from the title (i.e., `t.mw3`) gives the best trade-off between effectiveness and efficiency on our test dataset, achieving an F_1-measure of 0.537. The higher F_1-measure, compared to the F_1-measure of 0.525 for all pairs, suggests that the employed similarity function is sensitive to the number of pairs it needs to consider.

Chapter 4

Ontology Population of Product Information from Tabular Data*

With the vast amount of information available on the Web, there is an urgent need to structure Web data in order to make it available to both users and machines. E-commerce is one of the areas in which growing data congestion on the Web impedes data accessibility. This chapter proposes FLOPPIES, a framework capable of semi-automatic ontology population of tabular product information from Web stores. By formalizing product information in an ontology, better product comparison or parametric search applications can be built, using the semantics of product attributes and their corresponding values. The framework employs both lexical and pattern matching for classifying products, mapping properties, and instantiating values. It is shown that the performance on instantiating TVs and MP3 players from Best Buy and Newegg.com looks promising, achieving an F_1 of approximately 77%.

*This chapter is based on the article "L. Nederstigt, S. Aanen, D. Vandic, and F. Frasincar. FLOPPIES: A Framework for Large-Scale Ontology Population of Product Information from Tabular Data in E-commerce Stores. *Decision Support Systems*, 2014, 59: 296-311"

4.1 Introduction

A few decades ago, it was hard to imagine the enormous impact the Web would have on our daily lives these days. However, with the vast amount of information available, still doubling in size roughly every five years (Zhang et al., 2008), there is a serious need to structure all the Web data in order to keep it findable. With this aim in mind, the Semantic Web (Berners-Lee et al., 2001) was conceived in 2001. In the past years, some developments based on the ideas of the Semantic Web have been adopted for large-scale use. One of these is the introduction of a semantic vocabulary called schema.org (Google, Microsoft, Yahoo and Yandex, 2017), proposed by the four major search engines Bing, Google, Yahoo!, and Yandex. Schema.org is a very broad vocabulary with which the search engines aim to have a high-level shared vocabulary that focuses on popular Web concepts. This means that it is by no means an effort to have an ontology of 'everything' or an ontology that is very specialized in one domain. Furthermore, Google introduced recently the Knowledge Graph (Google Knowledge Graph, 2017), which is a project that augments search results with appropriate semantic metadata. Despite these recent movements, which are often attributed to the concept of 'Linked Data' (Bizer et al., 2009), the initially envisioned Semantic Web is still at its infancy.

One of the areas in which growing data congestion on the Web has serious consequences, is the field of e-commerce (Vijayalakshmi et al., 2011). Today's search engines are still primarily keyword-based, fail to work with syntactical differences, and are language-dependent. Web-wide parametric product search is unavailable, making it difficult for users to find the optimal purchase for their needs. According to existing research (Horrigan, 2008), a large fraction of online shoppers get confused or are overwhelmed by the information they get presented while searching for products. The result can be that prices become the determining factor for purchases on the Web. This situation is not optimal for both buyers and sellers: the buyers could be better off with a more expensive product if that would fit better to their needs, whereas the sellers might want to be competitive on other characteristics than pricing alone (Li et al., 2011).

This research focuses on the extraction of product information from tabular data sources on the Web, such as product information pages. Many major e-commerce shops use, in one way or another, a tabular format for the product specifications. This especially holds for complex (technical) products. For example, Amazon, BestBuy.com, Walmart, and Shopping.com, which are 4 well-known e-commerce sites, all represent product information in a tabular format.

In order to extract product information from tabular data, we propose *FLOPPIES: a Framework for Large-scale Ontology Population of Product Information in E-commerce Stores*. FLOPPIES is a semi-automatic approach, aided by user-defined ontology annotations in the form of lexical representations and regular expressions. Using the tabular data often available on Web store product pages, which conveys the factual information about a product, the ontology-driven framework creates a structured knowledge base of product information. In order to achieve this goal, FLOPPIES employs user-defined annotations for lexical and pattern matching, which facilitates product classification, property association, and value extraction. Our knowledge base, the proposed OWL (W3C OWL Working Group, 2012) ontology *OntoProduct*, defines specific properties and characteristics for 24 consumer electronic product classes. Figure 4.1 provides an overview of the input and output of the framework, based on our product ontology.

The proposed *OntoProduct* ontology is mapped to the GoodRelations ontology for e-commerce (Hepp, 2008), which is a more formal ontology than the schema.org vocabulary and is developed and maintained by Martin Hepp since 2002. It is a highly standardized vocabulary that not only can describe product data, but also company data and product offerings. This ontology aims to specify all aspects that come into play in the domain of e-commerce. For example, it supports statements to depict time frames for which an offering is valid. Fortunately, in 2012, the schema.org team announced that GoodRelations has been integrated in their vocabulary, which

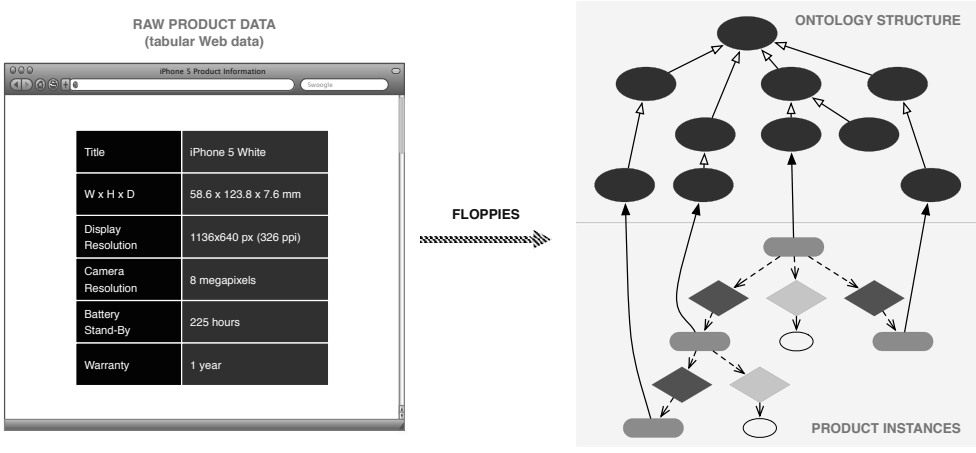

Figure 4.1: Overview of the input and output of the proposed framework. The tabular Web product data on the left (input) is transformed into the product instances part on the right (output), using the given ontology structure.

means that schema.org can now be used to describe more granular product information (Google, Microsoft, Yahoo and Yandex, 2017). Although GoodRelations defines concepts that can be used to describe product classes, i.e., their hierarchy and the associated product properties, the actual product classes, such as 'Phone' or 'Television', are not defined. This is one of the reasons why we propose the *OntoProduct* ontology and a system that can semi-automatically extract instances from unstructured product information.

When product information is formalized in an ontology, better product comparison or recommendation applications can be built, employing more intelligent parametric search by exploiting the semantics of product attributes and their corresponding values. Furthermore, there will be no need for existing Web stores to provide their data in a specific format (which is currently the case), as search engines will be able to effectively 'pull the information' from the Web stores themselves by consuming the annotated product information on the Web pages. Information could be more easily aggregated in order to have a very extensive source of product information. A prototype that utilizes Semantic Web technology to aggregate product information from multiple sources, as a means to improve product comparison, has been implemented in (Vandic et al., 2012b).

The formalization of product information has several advantages in practice for both business and consumers. For example, solving the information heterogeneity problem in e-commerce can lead to serious improvements in the business information exchange (Ng et al., 2000). Furthermore, the consumers' product retrieval capabilities will increase because of the more intelligent product search engines. For example, search engines will be able to better rank products because they can reason about how values of a product attribute relate to one another. This is best illustrated with an example. Consider the facts that 'HSPDA' is faster than '3G' and that '3G' is faster than 'GPRS'. From these facts, a semantic search engine can deduce that 'HSPDA' is faster than 'GPRS' if the property 'faster than' is declared to be transitive. This reasoning can help in cases where fuzzy search is needed, i.e., when a user is searching for a phone with 'HSPDA' but none actually exist with the current selection of properties and the next best phone has to be displayed. FLOPPIES supports these developments by providing a semi-automatic method to store actual facts about a product (i.e., the values of its attributes) in a product ontology. As a result, one has access to a knowledge base that is understandable for both humans and machines.

This chapter is organized as following. First, related research approaches are discussed in Section 4.2. Then, Section 4.3 explains the proposed framework in detail. Section 4.4 evaluates the performance of FLOPPIES in a component-wise analysis and compares it with a baseline approach. Last, conclusions and future work directions are given in Section 4.5.

4.2 Related Work

In this section, we discuss some similar research approaches for ontology population that are applicable in the e-commerce field. Furthermore, some existing product ontologies are reviewed, as such an ontology is required for instantiation in our problem context. The scope of this research is the ontology population itself, and not HTML table extraction. Therefore, approaches focusing on this topic are not discussed in this chapter.

4.2.1 Ontology Population Approaches

Due to the wealth of information that is now available, both on the Web and within organizations, it would be impractical to manually instantiate all that information in an ontology. Therefore, several semi-automatic ontology population approaches have been conceived in recent years, which are also applicable to the e-commerce domain.

Holzinger et al. (Holzinger et al., 2006) propose a fully autonomous process, which only needs some initial seed knowledge about a specific product domain. Using this knowledge, it performs a knowledge extraction process on Web pages, which retrieves tabular data that is relevant to the product domain from the Web page. However, instead of populating an ontology with the extracted information using an external framework that contains custom logic, they propose a more integrated approach in which parts of the required logic are replaced by OWL reasoning in the ontology. Once the tabular data has been extracted from the Web page, content spotters are employed, which detect specific values through regular expressions and are able to annotate this information with OWL statements. Afterwards, additional facts can be derived from the annotated tabular data using the domain-specific ontology that was given as the seed knowledge for the process. The authors argue that this provides a more modular and transparent system in which logical tables and domain models can be easily substituted.

A different approach, using the semantic lexicon WordNet (Fellbaum, 1998), is proposed by Patel et al. (Patel et al., 2003). OntoGenie is a semi-automatic tool

that takes domain ontologies and unstructured data, often in the form of natural language, as input. It first maps the concepts in a domain ontology to a WordNet equivalent. Then it captures the terms occurring in Web pages and tries to map each word to a WordNet concept. Finally, the relationships between the domain ontology concepts and the words on the Web pages can be determined by examining their mappings to WordNet. It employs the principle of locality to compute the distance between concepts, using information discovered from other pages, for increasing the recall.

Ontosophie (Celjuska and Vargas-Vera, 2004) is a strictly semi-automatic system for ontology population, requiring user input for each information extraction cycle. It consists of a natural language processor, which uses shallow syntactical parsing for annotating terms in sentences. The next process is deriving a set of extraction rules from the annotated documents. A conceptual dictionary induction system is employed for this phase, which uses a training corpus to derive a dictionary of concept nodes. The extraction rules are generated using the different combinations of concept nodes occurring in the sentences of the training corpus. However, as not every extraction rule might be correct or specific enough, Ontosophie also computes a rule confidence factor for each extraction rule, using K-fold cross-validation. During this process, it merges rules giving identical results and assigns each rule a confidence factor. After reviewing all the generated extraction rules, the extraction rules with a sufficient rule confidence factor are used to populate an ontology. The authors argue that it is important for the user to maintain control of the process, while only presenting suggestions that the process considers to be correct. Therefore, Ontosophie comes up with instantiation suggestions and ultimately lets the user decide on whether it should instantiate the information or not. Furthermore, it employs configurable thresholds for setting the desired minimum confidence factor for making the suggestions.

OntoSyphon (McDowell and Cafarella, 2008) is an unsupervised ontology population system, which takes any ontology as input and uses it to specify Web searches. Using both the ontology and the additional information obtained from the Web, it is able to automatically identify entities and their relations. The advantage of this approach is that the entire Web can be used as a corpus for instantiating entities in the ontology.

Our approach differs from these approaches on several aspects. First, with the exception of (Holzinger et al., 2006), the aforementioned methods are not specifically targeted at populating an ontology with (tabular) product information gathered from

the Web. Second, the other methods generally rely on natural language processing, using the syntactical context of a text to derive facts, while our approach focuses on tabular data. Last, even though the framework we propose shows some resemblance with the approach in (Holzinger et al., 2006), as both use regular expressions and lexical representations for entity annotation, there is on important difference. Unlike other approaches, including the approach in (Holzinger et al., 2006), our approach employs a GoodRelations-based ontology for annotating instances, making it compatible with major search engines (GoodRelations is already supported by some of the major search engines). The approaches that are discussed in this section do not share this advantage.

Even though most other methods are not directly applicable to the discussed problem, we can, nevertheless, re-use some of their elements. For instance, the classification of products can be achieved by mapping the category hierarchy of a Web store, if it is available, to product classes in the ontology. It could use a similar approach as (Patel et al., 2003), to create the mappings by employing WordNet.

In addition, the proposed value instantiation process, as used by the framework, employs a set of different value extraction rules capable of converting key-value pairs to the proper format for instantiating the ontology. Unfortunately, as there is no freely available implementation of a relevant ontology population framework, and not enough information to precisely recreate an existing framework, we cannot compare the performance of our proposed framework to that of the aforementioned frameworks.

4.2.2 Ontologies for E-commerce

Ontologies have been successfully applied in various domains, ranging from mechanical engineering (Guo et al., 2011, 2012, 2013) to biology (Ciaramita et al., 2005; Gene Ontology Consortium and others, 2000; Holmans et al., 2009). Also, various ontologies and categorization standards have been proposed for usage in the e-commerce domain. These can help businesses in a variety of ways, for example by improving communication possibilities between companies, and by automating various processes such as stock management.

A commonly used classification system for products is the *United Nations Standard Products and Services Code* (UNSPSC) (UNSPSC, 2012). Though UNSPSC is not freely available, it is applied broadly as it covers a very wide range of products. The UNSPSC dataset has also been converted into an OWL (W3C OWL Working Group, 2012) ontology for research use, though it is questionable whether the

purely hierarchical data structure of UNSPSC benefits from such a conversion (Hepp, 2010b). Similar to UNSPSC, *eCl@ss* (eCl@ss e.V., 2017) provides a wide base of product classes and descriptions. It is also a commercial standard, competing with UNSPSC, though more successful in Germany and containing properties per product class as well. For eCl@ss, an OWL conversion project is also maintained for research purposes (Hepp, 2006, 2010a). A third categorization standard worth mentioning is the *Open Directory Project (ODP)* (DMOZ, 2017), which is a project aiming to categorize the Web. Its shopping division consists of roughly 44,000 categories, but the classes have no further information attached to them.

In the e-commerce domain, another project, *RosettaNet* (Damodaran, 2004), is a non-profit standard for sharing business to business information. It is based on XML (Bray et al., 1997), and is mostly used in the supply chain area. These and other general e-commerce categorization standards are evaluated and discussed in a survey by Hepp et al. (Hepp et al., 2006).

Moving on to projects more related to Semantic Web, *GoodRelations* (Hepp, 2008) is a high-potential ontology describing products and service offerings for e-commerce. It has been adopted by various large Web stores in the form of RDFa (Adida et al., 2012) annotations. Furthermore, by mapping it to the schema.org vocabulary, the project is increasingly gaining attention from major search engines, which offer support for a growing set of annotations from GoodRelations. However, GoodRelations only specifies the way in which products and services need to be described, and does not contain product-specific properties or product classes.

In an attempt to augment GoodRelations with product classes and properties for consumer electronics, the *Consumer Electronics Ontology* (CEO) (CEO, 2017) has been developed. Although this ontology includes a subclass-of relationship between the product entities, product attribute information is not available.

There are some other approaches that are related to the product ontology that we propose. One of them is the productontology.org project (Martin Hepp, 2017b). This project publishes an ontology that extends the GoodRelations ontology with approximately 300,000 product and service definitions, based on the automatic extraction of these from Wikipedia. It contains some basic properties that are mapped to GoodRelations. However, these properties are very general and apply to many products. There are not many properties that are product specific, such as 'maximum load' for washing machines and 'cpu speed' for laptops. Furthermore, existing ontologies miss the metadata needed for appropriate extraction of properties and their values from text.

There are also other efforts that do not directly rely on Wikipedia for the schema creation (Martin Hepp, 2013, 2017a). Although the ontologies that are proposed in these projects contain more detailed schemas, they fail to address the issue of formal semantics with respect to the unit of measurements. Our proposed OntoProduct ontology does address this aspect by integrating an existing ontology-driven on units of measurement.

4.3 FLOPPIES Framework

In this section, we provide a detailed explanation of the FLOPPIES framework. First, the general processes for the framework are discussed in an overview. Then we elaborate on the ontology that is used for instantiation. Last, each step of the framework is explained in more detail.

4.3.1 Framework Overview

The goal of the FLOPPIES framework is to structure consumer product information from Web sites in order to improve product search or recommendation systems. To achieve this goal, several steps are required: extraction of key-value pairs from (tabular) Web data; instantiation of the product data into an ontology; product entity resolution to detect and aggregate duplicate products from different Web sources and; an application that uses the instantiated ontology data for product search or recommendation. A lot of research effort has already been invested in extraction of (tabular) Web site data (Chang et al., 2006), and in (product) duplicate detection (Elmagarmid et al., 2007; Kopcke and Rahm, 2010). Therefore, these steps are left outside the scope of this research.

The FLOPPIES framework starts with the assumption that product information in the form of key-value pairs is present. Collecting this data is often trivial, as many Web stores already offer product information in tabular form, ordered as key-value pairs. FLOPPIES uses this *raw product data*, as we refer to it, for instantiating the individual products and their features into a predefined product ontology. This domain ontology has to be able to describe individual products with specific properties for each type of product. For instance, a TV shares some properties with digital audio players (i.e., 'screen size'), but it also has properties that digital audio players do not possess (i.e., 'remote control'). Although significant effort has been put into establishing ontologies on the Web, a domain-specific ontology for products with

the required amount of specificity does not yet exist. Therefore, we introduce the *OntoProduct* ontology, which will be explained in more detail in the next subsection.

Figure 4.2 provides an overview of the FLOPPIES framework. It starts with the raw product data, the key-value pairs describing a product, as input. The final output of the framework is an OntoProduct ontology file, instantiated with the product and its features.

Between input and output, we identify three main processes, as shown in the diagram. First, it is necessary to obtain the type of product that is being instantiated: the *Classification* process. The classes are predefined in the ontology and determine the possible properties of the product. Most Web stores nowadays have some kind of product class or category data of each product available. Therefore, the Classification process in the FLOPPIES framework is seen as optional, in case the class data is not available.

The second framework process is called *Property Matching*. This step is used to create a link between the key-value pair keys from the raw product data, and the properties from the ontology. This is dependent on the product class, as the class determines the possible ontology properties that can be linked. Figure 4.3 indicates more clearly what Property Matching is about.

Note that, as the figure indicates with the mapping of 'Maximum Resolution', one raw product key can be mapped to multiple ontology properties. This is required as some raw product key-value pairs combine multiple characteristics of a product, which are separately stored in the ontology.

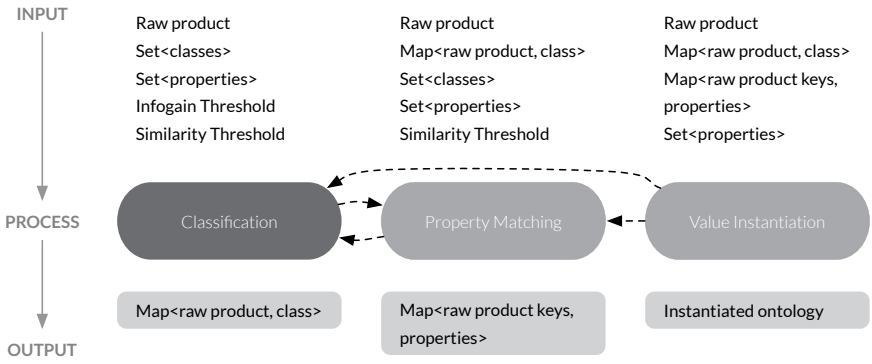

Figure 4.2: Overview of the processes in the proposed framework. Dashed lines indicate a usage relationship. Classification is only used when no class data is available.

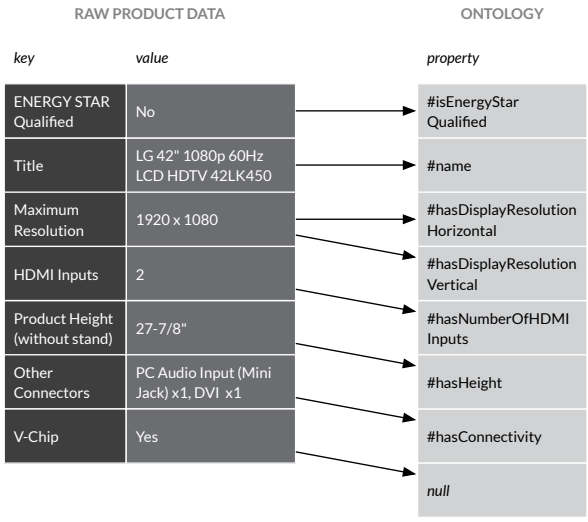

Figure 4.3: Example of property matching between (real-world) raw product data and ontology properties from OntoProduct.

The third and last process in the FLOPPIES framework is that of *Value Instantiation*. This part uses the class obtained from Classification, or directly from input data if the class is available, together with the result of Property Matching to instantiate the values in the ontology. Value Instantiation is very much about content spotting, parsing, and creating property assertions in the ontology. After the Value Instantiation, the raw product information from the Web has been structured and semantically annotated using an ontology. From that point on, applications can use the data to improve for example product search or facilitate product recommendation.

4.3.2 The OntoProduct Ontology

As Section 4.2.2 discussed, there have been various attempts to create product ontologies. However, unfortunately none of them are freely available and are both broad and specific enough to describe products in the domain this research uses: consumer electronics. Therefore, we introduce a new OWL (W3C OWL Working Group, 2012) product ontology for consumer electronics, which builds on the foundations of existing work: *OntoProduct*. It was conceived by four domain experts, who used training data originating from existing Web stores to create the ontology structure.

Dependencies of OntoProduct

OntoProduct is fully compatible with GoodRelations, known as 'The Web Vocabulary for E-commerce' (Hepp, 2008). However, GoodRelations is a high level ontology, which misses the specificity that is required to describe product features in detail. Another project led by GoodRelations' creator Martin Hepp, the Consumer Electronics Ontology (CEO) (CEO, 2017), attempts to extend GoodRelations with specific properties and product classes for better description possibilities of products. Although CEO provides a fruitful extension to GoodRelations, it only defines product properties and some product classes, but not the links between these. OntoProduct, nevertheless, uses CEO as a base, and extends it with new properties, product classes, and relations between these. In total, OntoProduct contains 24 product classes and 270 distinct product properties from the consumer electronics domain, which allows for the instantiation of product information with sufficient detail.

In e-commerce, many product features are quantitative and use a unit of measurement. For example, the weight of a product can be given in pound or in kilogram, resulting in a very different meaning. To cope with this problem, OntoProduct requires a unit of measurement to be linked to quantitative values. Although GoodRelations does not include a standard list of units of measurement, nor a way to define for example the used notations, we were able to extend it with another ontology that does enable to do this: the Units of Measurement Ontology (MUO) (Berrueta and Polo, 2009). MUO provides the ontology structure for working with units of measurement, but does not yet contain the instances. For the instances, OntoProduct uses the Unified Code for Units of Measure code system (UCUM) (Schadow and McDonald, 2010). The authors of MUO have made the dataset available to use UCUM in conjunction with the MUO ontology.

OntoProduct Structure

Figure 4.4 gives an example of an instantiation in the OntoProduct ontology. Any instantiated product individual, such as `op:LG-47LV` in this example, is member of a product class, in this case `ceo:TV`. This product class determines which properties are valid for the type of product that is being instantiated. In general, we identify three important property types: *quantitative object properties* (i.e. `ceo:hasWidth`), *qualitative object properties* (i.e. `ceo:hasDataFormat`), and *data properties* (i.e. `ceo:hasTouchscreen`). OntoProduct contains 57 qualitative object properties (with 783 qualitative individuals), 151 quantitative object properties, and 62 data properties. The domain of these properties entails one or more product classes, to define which

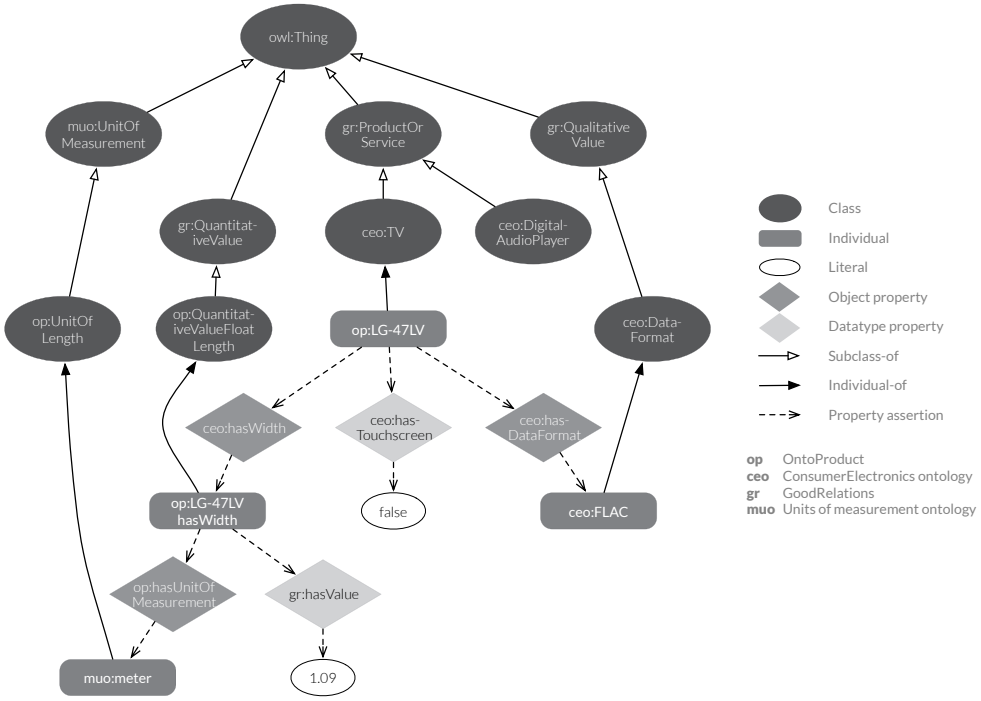

Figure 4.4: Example of an instantiated TV in the OntoProduct ontology.

characteristics a product can have. The range of the properties depends on the type: object properties have a range of respectively quantitative and qualitative values, whilst data properties point to data types. In the case of qualitative values, the range also determines the possible units of measurement that can be attached to some property value.

OntoProduct Metadata

As Section 4.3.1 mentioned before, FLOPPIES is a semi-automatic framework for product ontology instantiation. The reason we do not present it as being automatic, is because the algorithms largely depend on ontology annotations for linking product properties to raw product keys, and for parsing values from the raw product data. In practice, this means that for new data sources (i.e., a new Web store), the ontology needs to be annotated with appropriate metadata. For example, one Web store might specify the property of diagonal display size as 'Display size' while another uses 'LCD screen size'. Moreover, the lexical representations in the ontology can be used to enable processing for data with differing denominations or even from different

languages. In OntoProduct Metadata, which is an extension to OntoProduct used purely for the purpose of assisting the ontology instantiation with human input, the lexical representations can be applied to all properties and qualitative value individuals.

Next to lexical representations, OntoProduct Metadata can be used to annotate quantitative object properties and data properties with regular expressions (Friedl, 2006). Regular expressions provide a pattern to which the raw product values should match for a certain property. This is used for Property Matching to filter out possible faulty mappings. In addition, regular expressions are used in the Value Instantiation process to parse numeric data from the raw product values, by means of grouping. Grouping is commonly used in regular expressions to select certain parts of a matching region in the input value. For instance, consider the key-value pair `['Refresh Rate','60Hz']`, which can be mapped to the ontology property `op:hasScreenRefreshRate`. A screen refresh rate needs to have a unit of measurement for the frequency, commonly measured in Hertz (Hz), therefore we annotate the property with the following regular expression: `(\d+)\s?(?:Hz|Hertz)`. A regular expression searches the raw product value for a region which corresponds to the specified pattern, in this case a numerical value followed by either 'Hz' or 'Hertz'. If the search succeeds, it stores the numerical value in a separate group, which can be retrieved by the Value Instantiation process to instantiate the numerical value with the property `gr:hasValue`.

As another example of the flexibility offered by regular expressions, take key-value pair `['Dimensions','55.3" x 33.2" x 1.3"']`. Since there is no property to specify 'dimensions' in the ontology, it is required to break up the raw product value into multiple instantiations. Using lexical representations, the user could annotate ontology property `ceo:hasWidth` with 'Dimensions' for improved property matching. Adding a regular expression would enable the Value Instantiator to detect a match with value '`55.3" x 33.2" x 1.3"`', and select the first number, 55.3, from it through grouping. Similarly, the height and depth can be annotated for improved matching and parsing.

Annotation of properties is one of the key reasons why FLOPPIES is successful in instantiation, as we shall see. The user can help the computer by specifying recognition patterns in the form of regular expressions, and lexical representations, after which the computer can automatically instantiate most of the products with their various characteristics. For practical use, one could consider building a (Web) application to make the annotation easier for the end-user, for example by pre-

collecting lexical representations from raw product data which the user can select for addition to the OntoProduct Metadata database. For this research however, the ontology editor Protégé (Gennari et al., 2003) was used to create the annotations.

4.3.3 Classification

As mentioned in the previously given overview, the first core framework process of Classification is optional. Class data is often already available in Web data sources, for example through means of a category hierarchy. When a category hierarchy is available, a category mapping algorithm, such as SCHEMA(Aanen et al., 2012), can be used to obtain mappings between the category hierarchy and the product classes in the ontology. However, this subsection explains the process we propose to use when class data is not available. It uses the Property Matching process (explained in the next subsection), to measure the best fit between a raw product and the ontology product classes.

Figure 4.2 shows that the input of the Classification process consists of the raw product to classify, the sets of total classes and properties in the ontology, and two threshold parameters. The output of the algorithm is an association (type-of) between the raw product and an ontology class, such as 'TV' or 'Camcorder'. Algorithm 4.1 explains how the proper class is determined.

Classification computes the *highest information gain* per key-value pair to create a fit score per product class (by taking into account all key-value pairs): the average information gain. The information gain measures the specificity of a property for a certain product class. The information gain used here differs from the "classical" information gain measure used for instance-based classifications with decision trees.

Algorithm 4.2 explains how the highest information gain between one key-value pair and a product class is computed. As visible from the pseudo-code, the algorithm searches for the best-fitting property to a key-value pair. For this property, it returns the information gain, which is thus the *highest* information gain. It is the added value of the fact that the raw product has a certain property, in relation to finding the correct product class. A matching property that is used for many product classes, such as 'width', adds little value, whereas a specific one, such as 'TV tuner', yields a higher information gain. For every product class, the highest information gains per key-value pair of the raw product are aggregated, and their average is computed in order to obtain the average information gain. Based on this measure, the best class is chosen, as Algorithm 4.1 illustrates.

Algorithm 4.1: Classification of a Raw Product.

Input : A set of product classes C from the ontology and a set of key-value pairs K from the raw product description.
Output : The best matching clas for the given product description.
Data : Average Information Gain Threshold t.
Required functions:
- `maxIG`(k, c), returns the highest normalized information gain between a key-value pair $k \in K$ and a product class $c \in C$

```
1  i_c ← 0 // Keep track of average information gain i_c for each c ∈ C
2  foreach k in K do
3  |  /* Add the highest information gain for this key-value pair
   |     and each product class, divided by the total number of
   |     key-value pairs for normalization, to the total information
   |     gain of each product class.                                  */
4  |  foreach c in C do
5  |  |  i_c = i_c + maxIG(k, c)/|K|;
6  |  end
7  end
8  // Determine the best matching product class and return it if its
      score exceeds the threshold
9  c* ← null;
10 score* ← 0;
11 foreach c in C do
12 |  if i_c ≥ t and i_c > score* then
13 |  |  c* ← c;
14 |  |  score* ← i_c;
15 |  end
16 end
17 return c*
```

The information gain is dependent on the *Property Match Score*, as Algorithm 4.2 depicted. This is actually the score that is computed by the Property Matching process, and explains the dependency of Classification on the Property Matching process. Algorithm 4.3 explains how the score is computed. The details will however be explained in the subsection on Property Matching.

The Classification process is dependent on two parameters, as stated in the requirements of the algorithms and Figure 4.2. The first, the *Average Information Gain Threshold*, is used to strike a desirable balance between the recall and precision of the algorithm. When no threshold is used, products with a very low average infor-

Algorithm 4.2: Computing the Highest Information Gain.

Input : A key-value pair k from the raw product description and product class $c \in C$ from the ontology, which is to be matched with k.
Output : The maximum information gain.
Data : Similarity Threshold s;
Set of product classes C from the ontology;
Set of properties P_c from the ontology, for which c is in the domain of each $p \in P_c$.
Required functions:

- `propMatchScore`(k, p), which computes the similarity score between k and $p \in P_c$
- `propDomainSize`(p), which returns the number of classes in C that are entailed by the domain of property $p \in P_c$

```
1  ig* ← 0;
2  foreach p in P_c do
3  |   score ← propMatchScore(k, p);
4  |   if score ≥ s then
5  |   |   ig ← 1 − propDomainSize(p)/|C|;
6  |   |   if ig > ig* then
7  |   |   |   ig* ← ig;
8  |   |   end
9  |   end
10 end
11 return ig*
```

mation gain will still be classified, but with a high probability of failure. When the Average Information Gain Threshold is set, high-risk classifications will be skipped, that is, the classifier will return `null`. This moment could be used in an application to ask for user input, to prevent the product ontology from getting polluted. The higher the Average Information Gain Threshold, the higher the precision and the lower the recall of the Classification process. The second parameter is the *Similarity Threshold*, which is actually a parameter from the Property Match process. It will therefore be explained in the next subsection.

4.3.4 Property Matching

As depicted in Figure 4.2, Property Matching is dependent on the result of Classification (a product class linked to the raw product), the raw product, the sets of ontology properties and classes, and the Similarity Threshold. The goal of Property

Algorithm 4.3: Property Match Score.

Input : A key-value pair k from the raw product description and product class $c \in C$ from the ontology for which to compute the Property Match Score using k.
Output : The maximum information gain.
Data : Similarity Threshold s;
Set of product classes C from the ontology;
Set of properties P_c from the ontology for class c;
Set of lexical representations L_p for each $p \in P_c$;
Set of regular expressions R_p for each $p \in P_c$.
Required functions:

- `levenshtein`(k, L_p), which computes the maximum normalized Levenshtein similarity score between k and L_p
- `regexMatch`(k, R_p), which matches value from k with regular expressions in R_p

```
1  S_lxc ← levenshtein(k, L_p);
2  if R_p ≠ ∅ then
3  |  if regexMatch(k, R_p) = true then
4  |  |  S_rgx ← 1;
5  |  else
6  |  |  S_rgx ← 0;
7  |  end
8  |  S* ← (S_lxc + S_rgx)/2;
9  else
10 |  S* ← S_lxc;
11 end
12 return S*
```

Matching is to map each raw product key to an ontology property, as preparation for the Value Instantiation. To achieve this goal, the *Property Match Score* between each key-value pair from the raw product and each ontology property is computed using Algorithm 4.3.

The Property Match Score consists of two components: a lexical comparison between the raw product key and the ontology property, and a regular expression match. The regular expression match is optional, and depends on whether the ontology property is annotated with a regular expression in the OntoProduct Metadata or not. As explained in Section 4.3.2, the regular expressions work as a filter for finding the right ontology properties to match, based on the raw product values. For instance, key-value pair `['Product Height (without stand)','27-7/8"']` from

Figure 4.3 would not be mapped to property 'hasHeight' if the regular expression of this property would not match to values with fractions such as 27-7/8.

The second component of the Property Match Score, the lexical comparison, uses the normalized Levenshtein similarity score to compare the raw product key to each lexical representation of the ontology property, which are part of the OntoProduct Metadata file. The Levenshtein distance (Levenshtein, 1966) is a widely used edit distance measure for measuring the amount of difference between sequences of characters. Property Match Score uses the normalized Levenshtein similarity, which inverts the distance to transform it to a similarity, and then normalizes it by dividing with the maximum sequence length to become an index with range $[0, 1]$, where 1 would indicate that the sequences are equal. Of all lexical representations attached to the ontology property, the maximum similarity between a lexical representation and the raw product key is used.

For each key-value pair from the raw product, the ontology property with the highest Property Match Score is chosen under one condition: it must have a score that exceeds the Similarity Threshold (see Algorithm 4.2). This is a parameter of the framework that indicates how strict the Property Matching process should work regarding its mappings. When the threshold is very low, many raw product keys will be mapped, but with the chance of having a higher error rate. When the threshold is very high, less raw product keys will be associated with a property, but with higher accuracy. In the Evaluation section, we optimize the Similarity Threshold so that the algorithm works well under most conditions.

One special situation that can occur is when multiple properties match to a key-value pair with the same Property Match Score. In this case, the raw product key is mapped to all properties that have the same score, that is, if the Similarity Threshold has been exceeded. This characteristic enables for example the display resolution properties from Figure 4.3 to be linked correctly with the key-value pair for resolution. In this case, both properties share the same lexical representation of 'Maximum Resolution', with which it has been annotated manually in OntoProduct Metadata. For this reason, the lexical score is equal. Moreover, the regular expressions of the display resolution properties both match to the value of the key-value pair, which results in both properties ending up with the same Property Match Score. Grouping in the regular expression enables the Value Instantiation process to extract the proper numeric data (for horizontal and vertical) from the complete raw product value.

4.3.5 Value Instantiation

Once the class of the raw product has been determined, and its key-value pairs have been mapped to ontology properties, the framework is ready for Value Instantiation. This step uses the output of the first two core process, in order to respectively create a product individual within the proper class, and to associate each value using the correct property. Value Instantiation consists of a collection of parsers, content spotters, and instantiation tools. Figure 4.5 shows a flowchart that highlights this process. For Value Instantiation, a clear distinction is made between qualitative and quantitative object properties, and data properties. These are therefore separately explained in the following subsections. The procedure from the flowchart is followed for every key-value pair from the raw product.

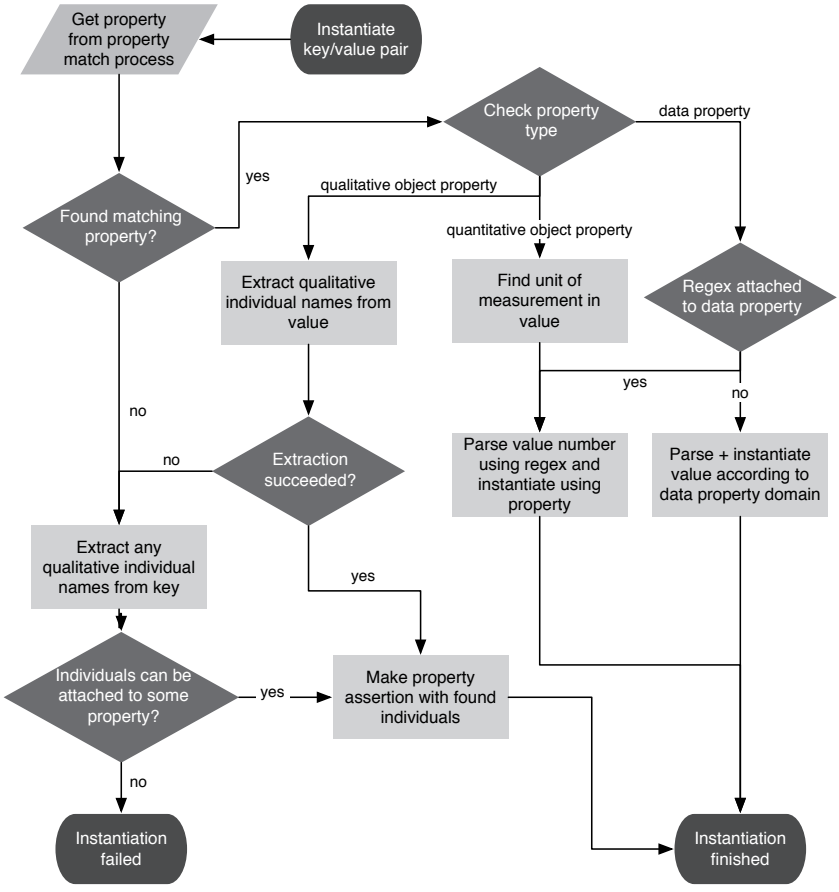

Figure 4.5: Overview of the instantiation process as a flowchart.

Instantiation of Qualitative Object Properties When the Property Matching process has linked a key-value pair to a qualitative object property, all qualitative values from the ontology that are in the range of the property are gathered. The goal is to find one or multiple of these qualitative values in the raw product value. Often, Web stores combine multiple qualitative values in one key-value pair, as is the case with 'Other Connectors' in Figure 4.3, for example. First, the lexical representations of all qualitative individuals are sorted on length, longest first. Then, the algorithm tries to find a matching lexical representation in the raw product value. If the search succeeds, the corresponding qualitative individual is attached to the product individual by means of the property found in the Property Matching process, and the matching part is removed from the raw product value string. This continues until no matches can be found anymore. The reason to order the procedure on lexical representation length, is that shorter labels might be contained in longer ones, leading to errors in parsing. This would for example be the case while parsing the raw product value `SDHC, MemoryStick, CompactFlash`; if the ontology contains qualitative value individuals for both SDHC and SD, the SD could match first without sorting, causing a faulty instantiation.

Extracting Qualitative Individuals Product Keys The 'normal' way in which qualitative values are instantiated, is through the control path just described. Property Matching links the key-value pair to a qualitative object property, after which qualitative individuals are extracted from the raw product value. Two special situations arise however, in which qualitative values are parsed differently, as Fig 4.5 denotes: When a qualitative property is found, but the Value Instantiation process is incapable of extracting qualitative values, or, when the result of the Property Matching process for the key-value pair is `null`. In these cases, the Value Instantiation process does not examine the raw product value for qualitative individuals, but the raw product key. Although this might seem counterintuitive, it is actually an important aspect of the Value Instantiation process. For example, a common situation in which it is needed to examine the raw product key instead of the value, is for qualitative properties such as 'Features'. Many features, such as 'Sleep Timer', are often not structured as `['Feature','Sleep Timer']` in the key-value pairs, but more likely as `['Sleep Timer','Yes']`. In the last case, Property Matching will be unsuccessful, as Sleep Timer is a qualitative individual (from the features class), and not a property in the ontology. In this situation, the raw product key will be examined for matches with any qualitative individuals from the ontology, in a sim-

ilar fashion as with 'normal' qualitative value instantiations, in which the Property Matching result is used. When a qualitative individual is found in the raw product key, the ontology is checked for properties that both have a range that includes the found individual, and a domain that entails the product class of the current product individual is entailed. Such a property is needed to be able to link the qualitative individual to the product individual in case that the property was not previously discovered with the Property Matching process.

Finding a qualitative individual in the raw product key does not provide sufficient information on itself to be able to assert ontology knowledge axioms. Whether the assertion can be made, also depends on the raw product value. Using what we call the *Boolean Value Converter*, the raw product value is checked on terms such as 'false', 'no', 'none', '0', '-', 'optional', 'null', 'N/A', 'not available', and 'not applicable', and aborts the instantiation when such a term is encountered. If the raw product value passes this test, the ontology is instantiated with property assertions, each containing one found qualitative individual.

The extraction of qualitative individuals from the raw product key enables the Value Instantiation process to handle key-value pairs like `['Sleep Timer','Yes']`. As mentioned before, and as Figure 4.5 makes clear, this procedure is also followed when 'normal' qualitative value instantiation is unsuccessful, that is, when there is a result from Property Matching, but no qualitative individuals can be found in the raw product value. This problem arises for example with `'AM/FM Tuner','Yes'`, which does have a match with ontology property 'hasRadioTuner' based on one of its lexical representations, but does not contain qualitative individuals in the raw product value. In this case, looking at the raw product key solves the problem and successfully instantiates `hasRadioTuner` to `AM` and `hasRadioTuner` to `FM`.

Instantiation of Quantitative Object Properties Parsing and instantiating quantitative values is very different from working with qualitative values. All quantitative values are parsed using regular expressions. By means of grouping, these enable to select the numeric data from the raw product value, disregarding additional content such as the unit of measurement. Note that some key-value pairs need multiple instantiations. Hence, multiple groups may exist in the regular expression, or the complete expression can match multiple times in one raw product value. The regular expressions come from the Ontoproduct Metadata, which is manually defined. When Property Matching has linked the key-value pair to a quantitative property, and no regular expression is attached to the property through the Onto-

Product Metadata, then a default regular expression for parsing values is used. The default regular expression is a generic value extractor and is capable of extracting numerical values.

Extracting the Unit of Measurement Usually, a quantitative value contains a unit of measurement. This unit of measurement is parsed in a similar fashion as parsing qualitative raw product values. As discussed in Section 4.3.2, the quantitative properties refer to a fixed set of possible units of measurement. For every parsed numeric value from the raw product value, an associated unit of measurement is searched, and if possible, the new quantitative value individual is linked to this unit individual by means of the 'hasUnitOfMeasurement' property. Figure 4.4 gives an indication of how a value individual is linked with the product individual and unit of measurement. When no unit of measurement is found, it is simply not instantiated.

Instantiation of Data Properties The third and last type of instantiation is when the Property Matching process returned a data property. Data properties are less commonly used than object properties in OntoProduct. Mostly, they are used for Boolean assertions (i.e., 'hasTouchscreen'), numeric data without unit of measurement (i.e., 'region code'), and strings (i.e., 'product name'). The values can be parsed in two ways: using a regular expression that is attached to the property, or, using a specific parsing method based on the datatype range of the data property. When a key-value pair linked to a data property needs to be instantiated, and the property, say 'hasTouchscreen', appears to have a data range of `xsd:boolean`, a boolean parser is used. This parser aims to find terms in the raw product value, using exact lexical matching, that could indicate whether the data value should be true or false. Similar parsers are used for integers, floats, and strings (or literals).

Finalizing the Value Instantiation Using all extraction rules described above, Value Instantiation is capable of converting a raw product key-value pair into ontology assertions. For each key-value pair of the raw product, the process, as made visible in Figure 4.5, is repeated. Though there are various points at which parsers could fail, preventing actual instantiation, it is easy to keep track of all failures and handle these separately. An application could for example hand the problematic key-value pairs over to the user, which could then instantiate them manually.

4.4 Evaluation

This section presents an overview and discussion of the performance of the FLOPPIES framework on our data, by means of a component-wise analysis of the various steps in the framework. First, we elaborate on how the experiment has been conducted, and which performance measures have been used throughout the evaluation. Afterwards we present the results, and discuss the performance of the framework by comparing it with the performance of a baseline approach.

4.4.1 Evaluation Design

This section discusses how the evaluation experiment has been set up. It provides a detailed overview of the used data and the methods employed to train the FLOPPIES framework.

The raw product data was obtained from two different Web sources, in order to increase the heterogeneity in the data set. Both sources are Web stores: Best Buy (BestBuy.com, 2017) and Newegg.com (Newegg.com Inc., 2017), which are large and well-known retailers in consumer electronics. As the research is focused on populating an ontology with product data, the Web crawler was intentionally kept simple. It crawls through eight predefined categories and obtains product data from them, using fixed extraction rules that are specific to each Web store. Seven of these categories are represented by a product class in the ontology, which means the products can be instantiated, whereas one category is not. By including a category that does not exist as a product class in the ontology, we can check whether the framework correctly refuses to instantiate the products from this category. For each product, the title and a tabular list, containing property information about the product as key-value pairs, were extracted from the Web store and stored along with product data from other products belonging to the same category. The end result consists of sets of products, each set describing a category from a specific Web store.

As mentioned earlier in Section 4.3.2, a part of the obtained product data is used to augment the ontology by enriching it with metadata. The metadata consists of lexical representations and regular expressions, which are manually annotated to ontology entities. The raw product keys are used to add lexical representations to properties, whereas the raw product values are used to construct regular expressions, which are also annotated to properties. The resulting metadata can be used by the FLOPPIES framework to match tabular data, originating from the Web, with properties in the ontology, and for instantiation of the values. For a proper evaluation

of the FLOPPIES framework it is important to assess its performance on data that was not used to enhance the ontology. Therefore, each data set obtained by the crawler is split into a training and a test set, using a 60% – 40% split which randomly distributes the products in the file across both sets. This ensures that we have data available, for each category and from each Web store, that can be used for either training or testing. After splitting the raw product data, we obtain a training set consisting of 1046 products in total, whereas the test set contains 672 products.

Each step in the framework depicted in Figure 4.2 is evaluated separately. In order to compute the performance measures we have to be able to compare the output of each step in the framework with a reference, known as the golden standard. The golden standard for the Classification process can be generated automatically in our case, as the products from each product class are stored in separate training or test data sets, and the name of each set corresponds to the correct product class in the ontology.

Unfortunately, creating the golden standard for the Property Matching process is far more complicated and therefore it cannot be generated automatically. Due to the sheer amount of different properties, either originating from the tabular data or the ontology, it is not feasible to provide a complete golden standard manually. Therefore, for evaluation of the Property Matching process, the software prompts the user for input whenever it comes across a mapping from the Property Matching process that it has not encountered before. The user can then select whether the mapping is correct or not and the user input is stored in a knowledge base, which can be consulted the next time the evaluation is performed.

For evaluating the Value Instantiation process we manually instantiated products in the ontology beforehand, thus creating a golden standard. As manually instantiating products is a very time-consuming process, we decided to instantiate a subset of the data, namely TVs and MP3 players, consisting of 48 complete products from both Web stores. Because the golden standard is only available for the manually instantiated products and not for all the products, we only evaluate the performance of this step for these products. We have chosen for TVs and MP3 players because TVs are generally described with much detail in both Web stores, whereas the tabular data obtained from MP3 players is often pretty scarce and lacking in detail on the Web store page. In order to analyze how the two considered Web shops compare in terms of the product descriptions they use, we computed the overlap in product attributes and values. For the TVs category, there are on average 7.2% matching keys. We computed this average over all the pairs of product descriptions that describe the

same product. For one product pair we compute the match value by dividing the number of matching keys by the maximum number of matches (i.e., $\min(|K_a|, |K_b|)$, where K_a and K_b represent the product attributes of description a and b, respectively). For MP3-players, the percentage of matching keys is much lower, i.e., 0.6%. Furthermore, we also computed, for the keys that matched, the overlap in the corresponding values. We found that for TVs 57.4% of these values match, while for MP3-players this is 12.8%.

For component evaluation of Property Matching, perfect class data was used as input, enabling a more accurate analysis of this component. This is done as the Property Matching process uses the product class as a filter, i.e., it only tries to match tabular data with properties from the ontology that are valid for the specific product class. By ensuring that the supplied input for the Property Matching process is completely accurate, we can evaluate the performance of this particular component in a more objective manner. Evaluation of the Value Instantiation is dependent on both Classification and Property Matching. As no golden standard for Property Matching is available, the Value Instantiation is evaluated with performance dependency of this step. Since the Classification process is seen as optional, the Value Instantiation will be evaluated both with perfect class input and with the result from the Classification process.

The FLOPPIES framework uses two different parameters, the *Average Information Gain Threshold* and the *Similarity Threshold*, for which the optimal values need to be computed. However, due to the interoperability between the Classification and the Property Matching processes, optimizing both parameters might seem like a convoluted process. Fortunately, because there is a golden standard for the Classification process, perfect class input for the Property Matching process can be used. This allows for the computation of the optimal value for the Similarity Threshold, as other variables are eliminated and thus the differences in performance are now solely caused by varying the Similarity Threshold value. Afterwards the optimal value for the Average Information Gain Threshold can be computed, given the optimal Similarity Threshold.

It is preferable to compare the results obtained by the FLOPPIES framework with another approach. However, as there is no freely available implementation of other relevant ontology population frameworks, and not enough information to precisely recreate a framework, we decided to create baseline approaches as well.

The baseline Classification process computes the lexical similarity, using the longest common substring as measure, between the raw product title and each prod-

uct class label name in the ontology for the classification. The baseline Property Matching process tries to find the highest normalized Levenshtein similarity score between a key-value pair from the raw product data and the lexical representations of a property from the ontology. The used baselines are straightforward and based on purely lexical approaches. We have chosen these as we want to investigate if the addition of semantics in the ontology population processes can provide benefits compared to lexical-based approaches. This type of baselines have been used also in the past for comparing lexical and semantic approaches (e.g., TF-IDF versus CF-IDF (Baziz et al., 2005)).

We have opted not to evaluate the performance of the FLOPPIES framework against a different process for the Value Instantiation process, because it is more like a collection of different value extraction rules rather than a single unified algorithm. Together they form the logic to parse and instantiate a wide array of values, but removing some rules for creating a simpler process would obviously only yield lower results and therefore would not really contribute to a useful evaluation of the framework.

We have implemented the software and the experiments in Java. For the storage and retrieval of RDF data, we have used the Jena library (McBride, 2002). Furthermore, we have used the Google Guava (Google, 2017) library for caching and improved type primitives support.

4.4.2 Performance Measures

This section describes the performance measures that were used to evaluate the FLOPPIES framework and explains the used definitions for each step in the framework. For the evaluation of the framework we use a binary classification scheme, which is commonly used for evaluating the performance of classification and mapping algorithms. We employ the standard measures that can be computed with such a scheme, e.g., precision, recall, and the F_1 we have (Baeza-Yates and Ribeiro-Neto, 2011). However, in this case we need to use a slightly adapted form, as it is not a pure binary problem.

For the Classification process, a *true positive* (TP) indicates that the framework has mapped a raw product to the correct product class. Unlike regular binary classification, where a *false positive* (FP) would mean that the framework mapped something which it should not have mapped at all, here it could also mean that it should have mapped the raw product, but it mapped to a wrong product class instead. A *true negative* is a raw product that has been correctly mapped to `null`, whereas

a *false negative* (FN) indicates a raw product that should have been mapped to a product class, but the framework mapped it to `null`.

The evaluation of the Property Matching process basically follows the same definitions as the Classification process, but it maps key-value pairs to properties, rather than mapping raw products to a product class. Note that a single key-value pair can be mapped to multiple properties, which could result in a slightly different amount of mappings per algorithm run, depending on the used parameter values.

Rather than individually evaluating all RDF triples created by the Value Instantiation process, we adopt a graph-based evaluation approach. The reason for this is trivial: consider a key-value pair like `['Product Width', '1.09m']` from the raw product data. This key-value pair should be instantiated with multiple RDF triples, as depicted by Figure 4.4, because we need to instantiate the value, the unit of measurement and the property assertion separately. Leaving out one of the triples would mean that the other triples loose most of their meaning, as a value is rather meaningless without a unit of measurement and vice versa. Therefore, we combine the triples of a quantitative value and evaluate them as a whole. In other words, for each triple where the instantiated product individual is the subject, we evaluate its subgraph as a whole.

As we manually instantiated 48 products for the golden standard, the instantiated products by the FLOPPIES framework can be compared to the products in the golden standard. Within this context a true positive means that a property was correctly instantiated, as it also occurs in the golden standard. A false positive indicates that the property should not have been instantiated at all, or that the associated value, unit of measurement, or individual, is wrong or missing. Whenever the golden standard contains a property that the instantiated product by the framework does not have, it is counted as a false negative. Note that there are no true negatives in the evaluation of the Value instantiation process, as the instantiated ontology is only being compared to the golden standard ontology, and non-existing assertions cannot be counted. One could propose to count the number of key-value pairs from the raw product data, for which no instantiation has been made while manually creating the golden standard ontology. However, since there is no direct relation between the number of key-value pairs and the number of instantiated facts, it is impossible to count the number of true negatives using this way. This is because one key-value pair can contain any number of facts that require to be separately stored in the ontology.

Using the aforementioned definitions, the following performance measures can be computed:

$$\text{recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$
$$\text{specificity} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$
$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{F}_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$
$$\text{F}_{0.5} = 1.25 \cdot \frac{\text{precision} \cdot \text{recall}}{0.5 \cdot \text{precision} + \text{recall}}$$

The F_1 is the harmonic mean of precision and recall, which means that both precision and recall are equally important. However, for the evaluation of the optional Classification process, the $\text{F}_{0.5}$ is also computed, for which the precision is twice as important as the recall. This score might be more preferable to use as performance measure, as instantiating raw products with the wrong product class would pollute the ontology and does not contribute to solving the search problems on the Web. It is envisioned that the Classification process uses a conservative approach and prompts the user for input when it cannot determine the correct product class with enough certainty. The $\text{F}_{0.5}$ is more useful for this usage scenario, but we also include the F_1 for the Classification process in the results for completeness.

4.4.3 Results

This section presents the obtained results for each step of FLOPPIES, along with an in-depth discussion of these results.

Training set results

First of all, the two parameters that are used by the FLOPPIES framework need to be optimized. Therefore, we run the algorithm with different parameter values on the training set. Due to the interoperability between the Classification process and the Property Matching process, we first optimize the Similarity Threshold parameter in the Property Matching process, using the golden standard from the classification step as input. In order to find the optimal value, we raised the threshold from 0 to

1 in steps of 0.05. Table 4.1 shows the results of the Property Matching process on the training set, both for the FLOPPIES framework and the baseline algorithm.

At first, the framework obtains a better F_1 by increasing the Similarity Threshold, until the score stabilizes, between 92% and 95%, from a Similarity Threshold of 0.70 onwards. As expected, the precision increases and the recall decreases when the Similarity Threshold is increased, due to the stricter lexical matching. At the optimal Similarity Threshold level (0.80), the number of false positives has declined to 395 out a total of 28038 mappings, whereas the number of false positives at a Similarity Threshold of 0.60 was quite a bit higher: 1462 out of 28146 mappings. Note that the small discrepancy between the total number of mappings is caused by the fact that a single key-value pair can be mapped to multiple properties if their similarity scores are both the same. Although the number of false positives continues to drop when increasing the Similarity Threshold beyond 0.80, the sharp increase in false negatives prevents it from obtaining a higher F_1. A total of 987 false negatives has been measured at the optimal value of 0.80, which gradually increases to 2109 when a Similarity Threshold of 1.00 is used.

Also worthy to note is the enhanced precision of the FLOPPIES framework compared to that of the baseline algorithm, scoring 97.28% at the optimal Similarity Threshold against 49.07% respectively. This is due to the fact that the baseline algorithm uses an optimistic approach, which enables it to actually score better on true positives than the FLOPPIES framework: 16971 against 14136. However, it comes at the expense of a large number of false positives, which considerably lowers the precision and therefore also the F_1 measure.

Process	Similarity Threshold	Precision	Recall	Accuracy	Specificity	F_1
Baseline	-	49.07%	100.00%	49.07%	0.00%	65.84%
FLOPPIES	0.60	71.21%	97.91%	78.01%	55.78%	82.45%
FLOPPIES	0.65	82.54%	95.67%	86.71%	76.14%	88.62%
FLOPPIES	0.70	90.90%	94.93%	92.03%	88.54%	92.87%
FLOPPIES	0.75	92.90%	94.40%	93.14%	91.69%	93.64%
FLOPPIES	0.80	97.28%	93.47%	95.07%	96.94%	95.34%
FLOPPIES	0.85	99.05%	90.78%	94.60%	99.00%	94.73%
FLOPPIES	0.90	99.87%	90.66%	94.96%	99.86%	95.04%
FLOPPIES	0.95	99.89%	88.10%	93.62%	99.89%	93.62%
FLOPPIES	1.00	99.90%	85.86%	92.43%	99.90%	92.35%

Table 4.1: Training set results for the Property Matching process using golden standard classification.

Using the optimal Similarity Threshold of 0.80, obtained from the first step, the Average Information Gain Threshold of the Classification process can now be optimized. By keeping the Similarity Threshold constant and varying the Average Information Gain Threshold, raising it from 0 to 1 in steps of 0.05, the results in Table 4.2 are obtained. As is evident from the results, the Average Information Gain Threshold functions as a parameter for finding the optimal trade-off between precision and recall. Generally speaking, the precision will increase when the threshold is increased as well, at the expense of a decline in recall. In other words, increasing the threshold means that the algorithm cannot classify as many products as before, but the ones it did classify are more likely to be correct. This is due to the fact that a higher threshold means that the properties of a product need to convey more specific information about the product, in order for the algorithm to map them to a product class from the ontology. Therefore, a product with a high Average Information Gain can be more reliably classified than a product with a lower Average Information Gain.

In contrast to the Similarity Threshold in the Property Matching process, the optimal value for the Average Information Gain Threshold is relatively low. The Similarity Threshold is a threshold operating on a lexical matching score, whereas the Average Information Gain Threshold operates on an average, namely the Average Information Gain for all key-value pairs from a raw product. This explains the difference in the optimal value, especially considering that nearly every product also has very generic key-value pairs, like the weight of a product, that help bring down the Average Information Gain. Also interesting to note is the difference between the F_1 and $F_{0.5}$s. Because the $F_{0.5}$ emphasizes the precision, the highest $F_{0.5}$ of 70.18% is obtained with an Average Information Gain Threshold of 0.20, whereas the highest F_1 is achieved using an Average Information Gain Threshold of 0.15. As argued in

Process	Average IG Threshold	Precision	Recall	Accuracy	Specificity	F_1	$F_{0.5}$
Baseline	-	29.83%	100.00%	29.83%	0.00%	45.95%	34.70%
FLOPPIES	0.00	49.33%	100.00%	49.33%	0.00%	66.07%	54.89%
FLOPPIES	0.05	49.33%	99.81%	49.33%	0.00%	66.07%	54.93%
FLOPPIES	0.10	54.93%	83.53%	50.67%	5.24%	66.27%	58.97%
FLOPPIES	0.15	68.91%	69.20%	57.07%	29.81%	69.06%	68.97%
FLOPPIES	0.20	72.17%	63.19%	56.31%	39.13%	67.38%	70.18%
FLOPPIES	0.25	70.00%	48.37%	46.94%	43.01%	57.21%	64.25%
FLOPPIES	0.30	58.05%	28.68%	32.50%	43.01%	38.39%	48.18%

Table 4.2: Training set results for Classification using optimal Similarity Threshold of 0.80.

Section 4.4.1, achieving a high precision is paramount for the Classification process, as it is better to ask the user for input rather than instantiating products with the wrong product class. Therefore, we consider an Average Information Gain Threshold of 0.20 as optimal for the training set, because it achieves the most precision and the highest $F_{0.5}$.

In addition, the results show that it can be quite difficult to classify products based on their properties alone. While this may seem a trivial task to humans, the differences in product properties between multiple product classes is often smaller than you would imagine. For instance, consider a camcorder and a digital photo camera: both are small, have a lens, connect to a computer through USB, use memory cards to store information, and so on. They share many characteristics, but there is essentially only one defining characteristic that separates them: a camcorder is meant for shooting video, whereas a digital photo camera is meant for shooting pictures. And even in this example the line between the two is blurry, as many digital photo cameras nowadays are perfectly capable of shooting videos as well. This high degree of function integration between products can be found in numerous products within the domain of consumer electronics, which makes the classification of products, based purely on product properties, a non-trivial task. Fortunately, practically every Web store contains a product category hierarchy, which can be used for the classification of products. That is why the Classification process in the FLOPPIES framework is optional and is only meant as a backup whenever insufficient information is available.

To complete the evaluation on the training data, the Value Instantiation process is executed using the output from the previous steps in the framework. Table 4.3 shows the results of this process when using either the golden standard classification or the output from the Classification process. As the training set contains 27 out of the 48 products that were manually instantiated in the golden standard ontology, the performance on those 27 products is evaluated. At first glance the results seem counterintuitive, as the Classification process of the FLOPPIES framework actually has a slightly better F_1 than the Golden standard, scoring 83.79% and 83.64% re-

Classification	Precision	Recall	Accuracy	F_1	Instantiation rate
Golden standard	82.11%	85.23%	71.89%	83.64%	100.00%
FLOPPIES Classification	81.67%	86.05%	72.11%	83.79%	96.30%

Table 4.3: Training set results for Value instantiation using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80).

spectively. However, this is caused by the method used to evaluate this part of the framework, which is explained in more detail in Section 4.4.2. Because the evaluation is performed on the instantiated products in the ontology, the products that were not instantiated are not evaluated. As the Classification process could not determine the product class of one MP3 player, due to the lack of specific product information, the Value Instantiation process only instantiated 26 of the 27 products, resulting in a product instantiation rate of 96.30%. Using the golden standard means that the product does get instantiated, but the Property Matching and Value Instantiation process have relatively more difficulty with this particular MP3 player, which results in the slightly lower F_1.

From these results we can conclude that the FLOPPIES framework as a whole performs rather well when instantiating TVs and MP3 players. However, it still fails to instantiate some properties or it is unable to instantiate them correctly. Error analysis on the instantiated products reveals that occasionally the framework is not capable of extracting and instantiating all individuals from a list of qualitative values. For example, consider the key-value pair `['System Requirements','Windows: 2000 or later; Mac: OS X 10.4 or later']`, which can be instantiated with the property `ceo:hasCompatibleOperatingSystem`. Any person, who manually instantiates this key-value pair, would also instantiate property assertions for the versions of Windows and Mac OS X that were released after Windows 2000 and Mac OS X 10.4 respectively. However, for our Value Instantiation process it is difficult to determine for which individuals it should instantiate property assertions, as it is trying to match the value with the lexical representations of individuals from the ontology. Therefore, it is able to instantiate property assertions for the individuals `'ceo:Windows2000'` and `'ceo:MacOSXTiger'`, as their lexical representations are also present in the value of the key-value pair, but later versions are not recognized. Fortunately, because the Value Instantiation process is using a set of value extraction rules, we could easily add a new rule to replace 'or later' in the value with the lexical representations of the referred individuals. By adding a new property assertion between the individuals in the ontology, which states that a certain individual is the successor of the other individual, the Value Instantiation process could learn to instantiate property assertions for all compatible operating systems. We consider creating new value extraction rules and augmenting the ontology with more details about the relationship between individuals as useful future work for improving the framework.

Test set results After optimizing both parameters of the FLOPPIES framework on the training set, the performance of the framework on the test data can be evaluated. Table 4.4 shows that the performance of the FLOPPIES framework on the classification of products from the test data is equal to the performance on the training data. The F_1 dropped slightly, from 67.38% to 66.24%, while the $F_{0.5}$ measure dropped from 70.18% to 69.18%. Relatively more products are marked as a false positive though: 124 out of 672 (18.45%) against 182 out of 1046 products (17.40%).

Although the Classification process is optional within the framework, more work on lowering the amount of false positives would be beneficial, as these errors could cause more problems later on in the Property Matching and Value Instantiation processes. One way to achieve this could be to also take the value of the key-value pairs into consideration for the information gain score. For example, many consumer electronics have an LCD display, which means that the property currently does not yield much information gain for our framework. However, the value could help differentiate between product classes and increase the information gain for this property. For instance, both a TV and an MP3 player have an LCD display, but if the display size of a raw product is 40", it is most likely that the product is a TV. By comparing this numerical value with TVs and MP3 players that are already instantiated in the ontology, a higher information gain for this property can be achieved, thus making it easier to determine the correct product class. Therefore, we consider differentiating between values for the purpose of product classification as a useful future addition to the framework.

The Property Matching process also scores roughly the same on the test and training data, as can be seen in Table 4.5. The precision and recall have decreased a little bit, which is caused by the fact that the key-value pairs from the test data were not used to 'train' the ontology by adding lexical representations and regular expressions. Although the test data contains some new raw product keys, the Property Matching was still able to match many key-value pairs with properties, because the Similarity Threshold allows it to also map raw product keys with slight lexical

Process	Precision	Recall	Accuracy	Specificity	F_1	$F_{0.5}$
Baseline	29.64%	100.00%	29.46%	0.00%	45.52%	34.30%
FLOPPIES	71.30%	61.84%	53.27%	28.74%	66.24%	69.18%

Table 4.4: Test set results for Classification using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80).

Process	Precision	Recall	Accuracy	Specificity	F_1
Baseline	48.30%	100.00%	48.30%	0.00%	65.14%
FLOPPIES	96.95%	93.27%	94.80%	96.58%	95.07%

Table 4.5: Test set results for Property Matching using golden standard classification and optimal Similarity Threshold of 0.80.

variations. In practice, this means that a semi-automatic approach would only require training the algorithm with a few products from each product class in order to achieve satisfactory performance on Property Matching for all the products in a Web store.

By analyzing the results, we conclude that the regular expressions in conjunction with the lexical representations are often capable of correctly mapping key-value pairs to properties in the ontology. For example, the key 'Product Dimensions' is correctly mapped to `ceo:hasWidth`, `ceo:hasHeight`, and `ceo:hasDepth`, which demonstrates the usefulness of regular expressions for the Property Matching process.

While the Property Matching process performs quite satisfactory on most key-value pairs, it sometimes gets confused between properties representing a quantitative measure without a unit of measurement. Consider raw product keys 'DVI Inputs' and 'HDMI Inputs', of which only 'HDMI Inputs' can be mapped with `ontoproduct:hasNumberOfHDMIInputs` in the ontology. Unfortunately, the Property Match process also creates a mapping from 'DVI Inputs' to `ontoproduct:hasNumberOfHDMIInputs`, as their lexical similarity is fairly high and they both describe a count of inputs. This could be avoided by raising the Similarity Threshold, which in turn would mean that the framework is less capable of automatically mapping slightly varying raw product keys. However, as shown in Section 4.4.3, stricter lexical matching degrades the overall performance of the framework.

When running the FLOPPIES framework in its entirety, the results on the test data in Table 4.6 are obtained. Unlike the previous steps in the framework, the performance of the Value Instantiation process on the test data is considerably lower than the performance on the training data: the F_1 dropped from around 83% to approximately 77%. This is because the test data contains a few keys and values from key-value pairs that have a considerably different lexical representation than those used for annotating the ontology. While the Similarity Threshold allows for some lexical variation to occur, a key-value pair with a considerably different lexical representation still would not exceed the threshold, and thus it cannot be mapped to a property in the ontology. This means does not find as many mappings for the test

Classification	Precision	Recall	Accuracy	F_1	Instantiation rate
Golden standard	77.12%	76.09%	62.07%	76.60%	100.00%
FLOPPIES Classification	76.99%	77.41%	62.87%	77.20%	90.48%

Table 4.6: Test set results for Value instantiation using optimal Average Information Gain Threshold (0.20) and Similarity Threshold (0.80).

set as for the training set. The effect can also be observed in the product instantiation rate when using the Classification process of the FLOPPIES framework to perform the classification, which drops from 96.30% to 90.48%. Two MP3 players from the total set of 21 products in the test set could not be classified by the Classification process. Regardless of the decline in performance though, the FLOPPIES framework still performs quite well, based on the obtained results, on instantiating TVs and MP3 players in the ontology.

4.5 Conclusions

This chapter proposes FLOPPIES, a framework capable of semi-automatic ontology population of product information from Web stores. It employs a predefined ontology, compatible with the GoodRelations ontology for e-commerce, in order to formalize the raw product information contained in tabular format on product pages in Web stores. With product information formalized in an ontology, better product comparison or recommendation applications could be built, using full parametric search. Furthermore, it could facilitate the aggregation and exchange of product information between multiple Web sites without relying on Web stores to provide their data in a specific format, as is the case with current comparison platforms.

The framework consists of an optional Classification process, which can identify the product class of a raw product by analyzing its key-value pairs and computing an Average Information Gain between each product class in the ontology and the key-value pairs from the raw product. It uses the second step in the framework, the Property Matching process, to compute this score. The Property Matching process computes a Similarity Score between a key-value pair and properties in the ontology, using both lexical matching and pattern matching with regular expressions. After the key-value pairs have been mapped to properties in the ontology, the Value Instantiation process instantiates the product information. A set of different value extraction rules is employed in order to instantiate the correct values and units of measurement.

The performance of the framework is compared to the performance of a baseline approach, which merely uses lexical matching for the Classification and Property Matching processes. Product information from 1718 products, spread across eight different consumer electronic product categories from Best Buy and Newegg.com, was gathered and split into a training and test set. The training set was used to annotate the ontology with lexical representations and regular expressions, which are used to improve the performance of the matching and parsing processes. Afterwards, it is used in the component-wise analysis to compute the optimal parameter values for the Similarity Threshold and Average Information Gain Threshold, which are used in the framework as a cut-off for the Property Matching and Classification process respectively. Last, using the optimal parameter values, the performance of the all the steps in the framework on the test data is evaluated.

It is shown that the FLOPPIES framework performs considerably better than the baseline approach for the Classification process, achieving an $F_{0.5}$ of 69.18% against 34.30%, due to the better precision. The Property Matching process also scores better than the baseline approach with an F_1 of 95.07% against 65.14%, due to the use of both lexical matching and pattern matching. The evaluation of the Value Instantiation process was performed using a graph-based approach, comparing it to a manually instantiated ontology with 48 products. Although running the framework with the optional Classification process resulted in a classification of only 45 out of 48 products, it did manage to achieve a similar F_1 as when using perfect classification input, scoring roughly 83% and 77% for the training and test set, respectively.

For future research, there are several ideas that can further improve product ontology population. First, FLOPPIES currently only uses the tabular data from product pages. However, often also textual descriptions are available, next to the semi-structured key-value pairs. Through text mining, one could try to use the descriptions to extract additional knowledge. Another unexplored possibility for the framework, is to use already instantiated ontology data for the instantiation of new data. For example, using data mining techniques such as clustering, the algorithm could learn when to match certain properties.

The Classification process uses most of the time the raw product keys, via the Property Match Score. The raw product values however can sometimes also provide a good indication of the proper class. Take for example the key-value pair `['Capacity','10-cup']`; the key is not very informative, however the value is a better indicator that we are might be dealing here with a coffee machine.

The Value Instantiation process could be enhanced by adding new value extraction rules and by creating new property assertions between individuals in the ontology that further specify the relationship between them. By formally defining in the ontology that 'Windows XP' is the successor to 'Windows 2000', the framework could also instantiate a property assertion for 'Windows XP' when it encounters a raw product value such as 'Windows 2000 or later'.

In the current framework, the regular expressions provide a reliable way for parsing values and filtering properties. However, regular expressions are labor-intensive to build, and the user needs quite some technical background in order to be able to make these. In the past years, there has been some successful research on automated generation of patterns like these. One could consider using such a technique for this framework, although it might affect the accuracy of the overall framework.

Chapter 5

An Automated Approach for Taxonomy Mapping in E-commerce*

THIS *chapter proposes SCHEMA, an algorithm for automated mapping between heterogeneous product taxonomies in the e-commerce domain. SCHEMA utilises word sense disambiguation techniques, based on the ideas from the algorithm proposed by Lesk, in combination with the semantic lexicon WordNet. For finding candidate map categories and determining the path-similarity we propose a node matching function that is based on the Levenshtein distance. The final mapping quality score is calculated using the Damerau-Levenshtein distance and a node-dissimilarity penalty. The performance of SCHEMA was tested on three real-life datasets and compared with PROMPT and the algorithm proposed by Park & Kim. It is shown that SCHEMA improves considerably on both recall and F_1-score, while maintaining similar precision.*

*This chapter is based on the article "L. Nederstigt, D. Vandic, and F. Frasincar. A Lexical Approach for Taxonomy Mapping. *Journal of Web Engineering*, 2016, 1&2: 84-109" and the conference publication "S. Aanen, L. Nederstigt, D. Vandic, and F. Frasincar. SCHEMA – An Algorithm for Automated Product Taxonomy Mapping in E-commerce. *9th Extended Semantic Web Conference (ESWC 2012)*, 2012, pp. 300-314."

5.1 Introduction

In recent years the Web has increased dramatically in both size and range, playing an increasingly important role in our society and world economy. For instance, the estimated revenue for e-commerce in the USA grew from $7.4 billion in 2000 to $34.7 billion in 2007 (Horrigan, 2008). Furthermore, a study by Zhang et al. (Zhang et al., 2008) indicates that the amount of information on the Web currently doubles in size roughly every five years. This exponential growth also means that it is becoming increasingly difficult for a user to find the desired information.

To address this problem, the Semantic Web was conceived to make the Web more useful and understandable for both humans and computers, in conjunction with usage of ontologies, such as the GoodRelations (Hepp, 2008) ontology for products. Unfortunately, as it stands today, the vast majority of the data on the Web has not been semantically annotated, resulting in search failures, as search engines do not understand the information contained in Web pages. Traditional keyword-based search cannot properly filter out irrelevant Web content, leaving it up to the user to pick out relevant information from the search results.

Search failures manifest themselves in e-commerce as well (Vijayalakshmi et al., 2011). In addition, more than half of the surveyed users in the aforementioned study on online shopping in the USA (Horrigan, 2008), have encountered various frustrations when shopping online. Due to the absence of Web-wide faceted product search, it is difficult to find the product which satisfies the user's needs best. Users switch between Web-wide keyword-based search results and price comparison tools to find the 'best' product. As this is a time-consuming process, prices are often the determining factor for a purchase. This is an unwanted situation for both buyer and seller: the buyer might like a more expensive product, because it suits his needs better, whereas the seller would like to be able to differentiate his offering on other characteristics than pricing alone. The solution would be to realize a uniform presentation of Web product information, which requires the data to be annotated and structured. A method for the aggregation of data from Web stores is to use the existing hierarchical product category structure: the product taxonomy. By matching the product taxonomies from different Web stores, it becomes easier to compare their products. This should contribute towards solving the search problems encountered by users when shopping online.

In this chapter, we introduce the *Semantic Category Hierarchy for E-commerce Mapping Algorithm* (SCHEMA), to be used for mapping between heterogeneous product taxonomies from multiple sources. It employs *word sense disambiguation*

techniques, using WordNet (Miller, 1995), to find synonyms of the correct sense for the category name. Furthermore, it uses lexical similarity measures, such as the *Levenshtein distance* (Levenshtein, 1966), together with structural information, to determine the best candidate category to map to. In order to evaluate SCHEMA, its performance is compared on recall and precision with PROMPT (Noy and Musen, 2003) and the algorithm proposed by Park & Kim (Park and Kim, 2007).

The structure of this chapter is as follows. In Section 5.2 related work is reviewed. Section 5.3 presents SCHEMA, our framework for taxonomy mapping. Section 5.4 discusses the evaluation results of SCHEMA, compared to existing approaches. Last, in Section 5.5, we give our conclusions and suggest future work.

5.2 Related Work

The field of taxonomy or schema mapping has generated quite some interest in recent years. It is closely related to the field of ontology mapping, with one important difference: whereas for matching of taxonomies (hierarchical structures), and schemas (graph structures), techniques are used that try to guess the meaning implicitly encoded in the data representation, ontology mapping algorithms try to exploit knowledge that is explicitly encoded in the ontologies (Shvaiko and Euzenat, 2005). In other words, due to the explicit formal specification of concepts and relations in an ontology, the computer does not need to guess the meaning. In order to interpret the meaning of concepts in an ontology or schema, algorithms often exploit the knowledge contained in generalized *upper ontologies*, such as SUMO (Niles and Pease, 2001) or WordNet (Miller, 1995). In this way the semantic interoperability between different ontologies is enhanced, facilitating correct matching between them. The semantic lexicon WordNet plays a specifically important role in many mapping algorithms, helping to overcome the ambiguity occurring in natural language, often in combination with word sense disambiguation approaches, such as the approach of Lesk (Banerjee and Pedersen, 2002; Lesk, 1986). In addition to the usage of upper ontologies for producing the mappings between ontologies and schemas, lexical similarity measures are also often used. Using lexical similarity measures helps algorithms to deal with slight lexical variations in words. The Levenshtein distance (Levenshtein, 1966) is known as the edit distance, and has been augmented to allow for transposition of characters in the Damerau-Levenshtein distance (Damerau, 1964), both utilized in our algorithm.

In their algorithm for product taxonomy mapping, Park & Kim (Park and Kim, 2007) propose to use a disambiguation technique in combination with WordNet to obtain synonyms for a category name, in order to find candidate paths for matching. The candidate paths are assessed using co-occurrence and order consistency, which evaluate the overlap and the order of the categories between the source and candidate path, respectively. While specifically targeted at e-commerce, some phenomenons that occur frequently in product taxonomies are neglected, such as composite categories, in which multiple concepts are combined. Various other (database) schema matching approaches exist. SimilarityFlooding (Melnik et al., 2002) uses the similarity between adjacent elements of schema entities to score possible mappings, but does not take the frequently occurring terminological variations, applicable to e-commerce, into account. COMA++ (Aumueller et al., 2005) provides a collection of simple matching algorithms and combinations of these. Some approaches use class attribute data for matching, such as S-Match (Giunchiglia et al., 2005) and CUPID (Madhavan et al., 2001). A good overview of existing approaches has been made in recent surveys for schema matching (Do et al., 2002; Rahm and Bernstein, 2001; Shvaiko and Euzenat, 2005).

PROMPT (Noy and Musen, 2003) is a general-purpose ontology mapping tool, which uses predefined (automatic or manual) mappings, called *anchors*, as guidelines for the mapping of similar nodes. However, due to its emphasis on mapping ontologies in general, it fails in matching many categories when employed for product taxonomy mapping. H-Match (Castano et al., 2003) uses WordNet for determining the correct contextual and linguistic interpretation of concepts, combined with semantic ontology data. Yu et al. (Yu et al., 2009) propose to use an upper ontology, in order to create a semantic bridge between various e-commerce standards. QOM (Ehrig and Staab, 2004) uses only simple similarity measures, aiming to reduce time complexity, without significant loss of accuracy. Ehrig & Sure (Ehrig and Sure, 2004) propose a rule-based approach, combined with neural networks. Other approaches are discussed in recent surveys for ontology mapping (Kalfoglou and Schorlemmer, 2003).

5.3 SCHEMA

This section discusses the SCHEMA framework, together with all the assumptions for our product taxonomy matching algorithm. Figure 5.1 illustrates the high-level overview of the framework. This sequence of steps is executed for every category in the source taxonomy. First, the name of the source category is disambiguated, to

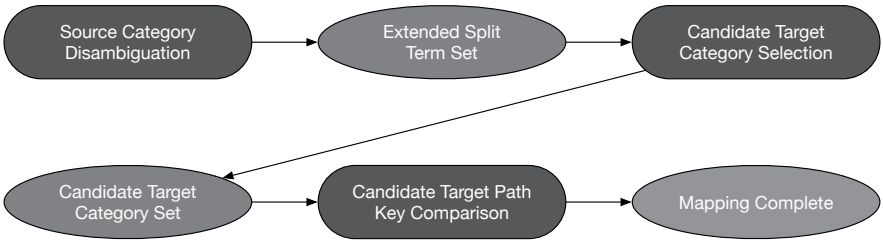

Figure 5.1: Framework overview for SCHEMA.

acquire a set of synonyms of the correct sense. This set is used to find candidate categories from the target taxonomy, and is needed to account for the varying denominations throughout taxonomies. After the *Candidate Target Category Selection*, every candidate category path is compared with the path of the source category, by means of the *Candidate Target Path Key Comparison*. The best-fitting candidate target category is selected as the winner. The objective of SCHEMA is to map source categories to a selected target category, if and only if, all products in the source category fit in the selected target category. This reflects our definition of a successful and meaningful category mapping. First, the general assumptions — the basis for the development of SCHEMA — are explained. Next, each step of the framework, as shown in Figure 5.1, will be discussed in more detail.

5.3.1 General Assumptions

In product taxonomies, a frequently seen phenomenon is that of composite categories. These are nodes, that combine multiple — usually related — classes into one, like category 'Movies, Music & Games' from Amazon. Each of the three parts could have been a separate class as well, as different product concepts are represented. An assumption in the development of SCHEMA was that composite categories need to be treated adequately, as the target taxonomy might not use the same conjunction of classes. To handle the phenomenon, SCHEMA splits categories on ampersands, commas, and the string 'and'. The resulting set of classes, making up the composite category, is called the *Split Term Set*.

Product taxonomies are tree-structured data schemes, and thus have a root node. However, in product taxonomies, root categories (e.g. 'Products' or 'Shopping') are meaningless, as they do not provide information about the products falling under. The assumption used for SCHEMA is that, as root nodes are meaningless, they should get automatically mapped in taxonomy matching. Furthermore, roots should

be disregarded in all computations, such as in path comparisons. Figure 5.2 shows that the root categories (the left-hand side categories) are matched by SCHEMA, despite being lexically dissimilar.

Between different product taxonomies, it is apparent that varying degrees of specialization exist with respect to the product classification. This could mean that there possibly is no direct match for a very specific source category in the target taxonomy. In such a case, it makes sense to match the source category to a more general target category, as from a hierarchical definition, products from a specific category should also fit into a more general class. Figure 5.2 shows that category 'Books' (Overstock) is mapped to 'Books' (Amazon), as one would expect. Unfortunately, there is no direct match for 'Humor Books' (Overstock) in Amazon. However, humor books are also a kind of books, so SCHEMA will map this category to the more general 'Books' category from Amazon. The more general category is found by following the defined mapping for the parent of the current source category. Note that root mappings are precluded. SCHEMA's last assumption is, that as usage of capitals in category names does not affect the meaning, all lexical matching is performed case-insensitive.

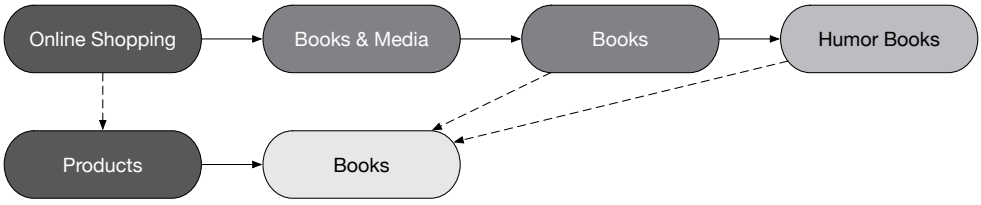

Figure 5.2: Mapping example going from Overstock (top) to Amazon (bottom) categories. Normal lines indicate a parent-child relationship; dashed lines indicate SCHEMA's mapping.

5.3.2 Source Category Disambiguation

The first step in creating a mapping for a category from the source taxonomy, is to disambiguate the meaning of its name. As different taxonomies use varying denominations to identify the same classes, it is required that synonyms of the source category label are taken into account for finding candidate target categories. However, using all synonyms could result in inclusion of synonyms of a faulty sense, which could for example cause a 'laptop' to be matched with a book to write notes (i.e., a notebook). To account for this threat, SCHEMA uses a disambiguation procedure in combination with WordNet (Miller, 1995), to find only synonyms of the correct sense

for the current source category. This procedure is based on context information in the taxonomy, of which can be expected that it gives some insight into the meaning of the source category name. Concerning the general assumption on composite categories in Section 5.3.1, SCHEMA disambiguates every part of the source category (Split Term Set) separately. The result after disambiguation is called the *Extended Split Term Set*. Note that the target taxonomy does not play a role in the source category disambiguation.

Algorithm 5.1 explains the procedure that is used to create the Extended Split Term Set for the current source category. First, Algorithm 5.1 splits the (composited) source category into separate classes: the Split Term Set, using the function splitComposite $(\cdot)$. The same split is performed for all children, and for the parent of the source category, which will act as 'context' for the disambiguation process.

Algorithm 5.1: Finding Source Category's Extended Split Term Set.

Input : The source category to disambiguate (w_{src}), the parent of w_{src} (w_{parent}), and the children of w_{src} (W_{children}).
Output : The extended split term set of the source category w_{src}.
Required functions:

- `splitComposite`(w), which splits composite category name w into a set of individual classes: a split term set W.
- `disambiguate`$(w_{\text{target}}, W_{\text{context}})$, disambiguates a word using a set of context words, resulting in a set of correct synonyms (described by Algorithm 5.2).

```
// First, all used categories get split on composite classes
W* ← splitComposite(w_src);
W_parent ← splitComposite(w_parent);
W_child ← ∅;
foreach w_currentChild in W_children do
    W_child ← W_child ∪ splitComposite(w_currentChild);
end
W_context ← W_child ∪ W_parent;
Ω ← ∅ // the extended split term set
// For every split part of w_src, find the extended term set
foreach w_srcSplit in W* do
    Δ ← disambiguate(w_srcSplit, W_context) // extended term set
    Δ ← Δ ∪ {w_srcSplit} // always include the original split term
    Ω ← Ω ∪ {Δ};
end
return Ω
```

Algorithm 5.2: Context-Based Target Word Disambiguation.

Input : The target word to disambiguate (w_{target}) and the set of context words (W_{context}).
Output : The synset with the highest similarity to the target word.
Required functions:

- `getSynsets`(w), gives all synonym sets (representing one sense in WordNet), of which word w is a member;
- `getRelated`(S), gives synonym sets directly related to synset S in WordNet, based on hypernymy, hyponymy, meronymy and holonymy. Result includes synset S as well;
- `longestCommonSubstring`(w_a, w_b), which computes the length of the longest common sequence of consecutive characters between two strings, corrected for length of the longest string, resulting in an index in the range $[0, 1]$;
- `getGloss`(S), returns the gloss associated to a synset S in WordNet.

```
1  Z ← getSynsets(w_target) // Z holds all possible senses
2  score* ← 0;
3  synset* ← ∅;
4  // Evaluate every possible sense (synset) S ∈ Z of target word
       w_target
5  foreach S in Z do
6      senseScore ← 0;
7      R ← getRelated(S);
8      // For every pair of context words & (related) glosses
9      foreach (S_related, w_context) in R × W_context do
10         gloss ← getGloss(S_related);
11         senseScore ← senseScore + longestCommonSubstring(gloss, w_context);
12     end
13     if senseScore > score* then
14         score* ← senseScore;
15         synset* ← S // Update best known synset so far
16     end
17 end
18 return synset*
```

Next, the disambiguation procedure itself, which will be discussed shortly, is called for every split part of the source category. The result, the Extended Split Term Set, contains a set of synonyms of the correct sense for each individual split term. The Extended Split Term Set is used in SCHEMA to find candidate target categories, and to evaluate co-occurrence of nodes for path-comparison.

As explained before, disambiguation of the source category name is based on a set of words from its context. The idea to use this context is based on a well-known algorithm for word sense disambiguation from Lesk (Lesk, 1986). However, traditional dictionary glosses, used by Lesk, may not provide sufficient vocabulary for successful matching. Therefore Banerjee & Pedersen (Banerjee and Pedersen, 2002) propose to use the rich semantic relations of WordNet, considering also related glosses of both target and context words to reduce this effect. Unfortunately, this introduces another problem: the computation time increases exponentially with the number of context words. To prevent computation time from exploding, Kilgarriff & Rosenzweig (Kilgarriff and Rosenzweig, 2000) propose to use Lesk's traditional algorithm with heuristics to simplify the search. Instead of using a dictionary gloss for every context word, they propose to use only the context words. This method reduces time complexity, but has similar vocabulary-related restrictions as the original Lesk algorithm. SCHEMA uses the best of these procedures, utilizing the rich semantic relations of WordNet for the target word, while comparing only to the plain terms from the context, as described in Algorithm 5.2. For every possible sense of the target word, the overlap between its related glosses and the plain context words is assessed. The length of the longest common substring is used as similarity measure, and the sense with the highest accumulated score is picked as winner.

5.3.3 Candidate Target Category Selection

The result of the Source Category Disambiguation, the Extended Split Term Set, is used to find matching categories in the target taxonomy. This set of candidate categories is basically a pre-selection for the decision to which target category the current category can be mapped to. The selection relies on SCHEMA's definition of a category node match, *Semantic Match*, described by Algorithm 5.3, which is used consistently throughout SCHEMA. It is used to classify a source category and a target category as equivalent or dissimilar, utilising the enriched information provided by the Extended Split Term Set for the source category, in combination with lexical matching to evaluate similarity between the category names. For the composite categories, SCHEMA assumes that with respect to the split terms, the source category is a subset of the target category. This ensures that all products in a mapped source category fit in the target category.

For every split part of the source category, Semantic Match checks whether there is a matching part in the target category. A match can mean either that the source split part is contained as separate component in a target part, or that they share

Algorithm 5.3: Semantic Match.

Input : The target taxonomy category name: w_{target} and the extended split term set E, with sets of synonyms S of the correct sense for every split term of the source category

Data : The Node Match Threshold t_{node}, defines the minimum degree of lexical similarity in order to classify two class names as equal.

Required functions:

- `splitComposite`(w), splits composite category name w into a set of individual classes: a split term set W;
- `levenshtein`(w_a, w_b), computes the edit distance between two strings;
- `casc`(w_a, w_b), indicates whether string w_a contains string w_b as separate part (middle of another word is not sufficient)

```
1  W_target ← splitComposite(w_target);
2  matches ← true // initial assumption: source split term set is
       subset of target
3  foreach S_srcSplit in E do
4  |   matchFound ← false;
5  |   foreach (w_srcSplitSyn, w_targetSplit) in S_srcSplit × W_target do
6  |   |   edit_dist ← levenshtein(w_srcSplitSyn, w_targetSplit);
7  |   |   // Normalise distance based on length and convert to
   |   |      similarity measure
8  |   |   similarity ← 1 − edit_dist/ max(w_srcSplitSyn, w_targetSplit);
9  |   |   if casc(w_targetSplit, w_srcSplitSyn) then // contains as separate
   |   |      component
10 |   |
11 |   |   |   matchFound ← true;
12 |   |   end
13 |   |   else if similarity ≥ t_node then
14 |   |   |   matchFound ← true;
15 |   |   end
16 |   end
17 |   if matchFound = false then
18 |   |   matches ← false
19 |   end
20 end
21 return matches
```

a lexical similarity based on the normalised Levenshtein index (Levenshtein, 1966), exceeding a chosen threshold. When all split parts of the source category have a match in the target category, the match is considered semantically correct.

Figure 5.3 shows some candidates that have been found for category 'Tubs' from Overstock. The Source Category Disambiguation procedure discussed in Section 5.3.2 results in the following Extended Split Term Set: `{{Tubs, bathtub, bathing tub, bath, tub}}`. Synonym 'bath' is sufficient for candidate category 'Kitchen & Bath Fixtures' (at the top of Figure 5.3), to be selected. As 'bath' is included in split target part 'Bath Fixtures' (as separate word), it matches, according to Algorithm 5.3, making target category 'Kitchen & Bath Fixtures' a superset of source category 'Tubs'. Hence it is classified as a semantic match, and thus selected as proper candidate target category.

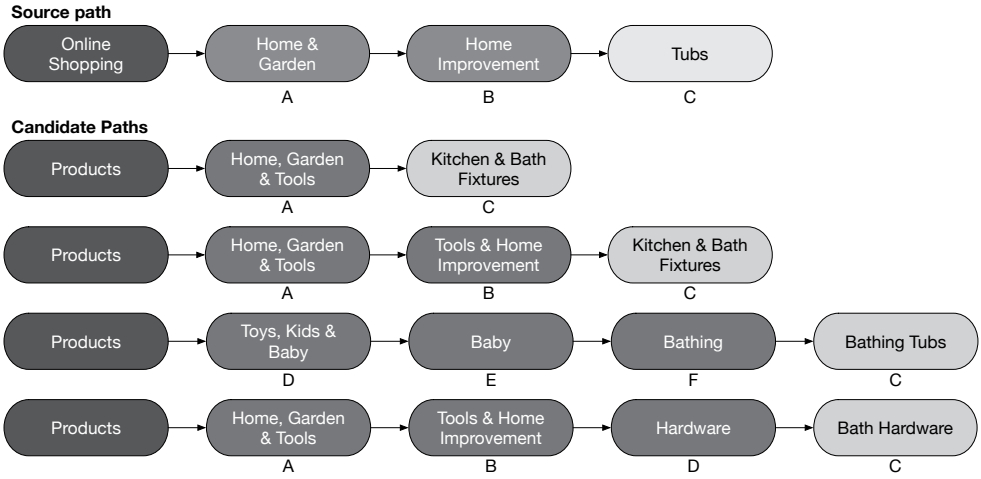

Figure 5.3: Source category path for 'Tubs' in Overstock, with associated candidate target categories from Amazon

5.3.4 Candidate Target Path Key Comparison

SCHEMA's last step is to select the best alternative from the set of found candidate target categories, using a method that scores the similarity of the source category path against a candidate target path. This *Candidate Target Path Key Comparison* is used for every element from the set of candidate target paths. The candidate with the highest score is selected as winner. The idea of the Candidate Path Key Comparison is simple in nature, though powerful in the sense that it assesses similarity based on both structural and lexical relatedness.

For both source and candidate target path, a key is generated for every node (category) in the path. This is done in such a way, that every unique node gets

a unique key. Similarly, when two nodes — independent from the path they come from — are seen as identical, they are labeled with the same key. An important question is: when are two nodes seen as identical? A straightforward way would be to base this purely on lexical similarity of the category names. However, SCHEMA uses a richer source of information for nodes from the source path: the Extended Split Term Set. Two nodes from the source path are seen as identical if and only if their Extended Split Term Sets are the same. A node from the source path and a node from the candidate target path are seen as identical when Algorithm 5.3, the Semantic Match procedure, decides so. The result is a key list for both the source path and the current candidate target path.

Figure 5.3 shows the key list for the source and candidate targets paths for category 'Tubs'. The candidate path at the bottom, is a good example of how Semantic Match classifies nodes as being similar. Candidate node 'Tools & Home Improvement' is assigned the same key ('B') as source node 'Home Improvement', as the first one is a superset of the last one, thus all products under the second should fit into the first. Considering candidate 'Bath Hardware' itself, one of the synonyms of source category 'Tubs' ('bath'), is included in the name of the candidate category. Hence, 'Bath Hardware' gets the same key ('C') as 'Tubs'.

For the key lists found for source and candidate path, the similarity is assessed using the Damerau-Levenshtein distance (Damerau, 1964). This measure captures the (dis)similarity and transposition of the nodes, hence both the number of co-occurring nodes and the consistency of the node order are taken into account. As the Damerau-Levenshtein distance is used in normalized form, a dissimilar node in a long candidate path is weighted as less bad than the same dissimilar node in a shorter path, which can unfortunately lead to biased results. Therefore, a penalty is added for every unique key assigned solely to the candidate path, or more precise: for every node for which no match exists in the source path. The formula used as similarity measure for the key lists is as follows:

$$candidateScore = 1 - \frac{\text{damLev}(K_{\text{src}}, K_{\text{candidate}}) + p}{\max(K_{\text{src}}, K_{\text{candidate}}) + p} \tag{5.1}$$

where K is a key list, p the penalty (# dissimilar nodes in candidate path), damLev() computes the Damerau-Levenshtein distance between two key lists, and max() computes the maximum length of two key lists.

In Figure 5.3, the uppermost and lowermost candidate paths give an example of the penalty's usefulness. One is too short, the other too long. The shortest

('Kitchen & Bath Fixtures') does not contain new nodes in comparison to the source path. With just one edit operation (insertion of key 'B'), it gets a candidate score of $1 - \frac{1+0}{3+0} = \frac{2}{3}$. The longest contains a new node: 'Hardware'. This gives the long path a penalty of 1, while the edit distance is also 1 (deletion of key 'D'), resulting in a score of $1 - \frac{1+1}{4+1} = \frac{3}{5}$. Without penalty it would score $\frac{3}{4}$, causing it to win from the short path, which does not contain unrelated nodes. Clearly, we prefer the first candidate path, because the second candidate path possibly changes the meaning of node 'C' as it has as parent a new node 'D'.

Once the candidate target category with the highest score has been found, it is mapped if the score exceeds the *Final Similarity Threshold* (t_{final}). This threshold prevents the algorithm of performing incorrect mappings, and should not be confused with the Node Match Threshold used in Algorithm 5.3. When a path does not pass the Final Similarity Threshold, or when no candidate paths have been found, the source category is mapped to the mapping of its parent (but excluding the root). The complete framework procedure then repeats for the next source taxonomy category.

5.4 Evaluation

In order to assess SCHEMA's performance, it is compared to similar algorithms. We have chosen to compare it with PROMPT (Noy and Musen, 2003), being a general-purpose algorithm that is well-known in the field of ontology mapping. Additionally, the algorithm of Park & Kim (Park and Kim, 2007) is included in the comparison, due to their focus on product taxonomy mapping in particular. First, we briefly discuss how the evaluation has been set up. Then, we present the results for each algorithm and discuss their relative performance.

5.4.1 Evaluation Design

Three product taxonomies from real-life datasets were used for the evaluation. The first dataset contains more than 2,500 categories and is from Amazon (`www.amazon.com`). The second dataset is from Overstock (`https://www.overstock.com`) and contains more than 1,000 categories. Overstock is an online retailer with RDFa-tagged product pages for the GoodRelations (Hepp, 2008) ontology. The last dataset contains over 44,000 categories and is from the shopping division in the Open Directory Project (ODP, `www.dmoz.org`). Using these three datasets, six different combinations of source and target taxonomies can be made. In order to evaluate the algorithms' performance on the mappings, it is required that each of the mappings is done man-

ually as well. However, as the datasets are too large to manually map every category, we have taken a random sample of five hundred category nodes from each dataset. For every node it is assured that its ancestors are included in the sample as well. The mappings are made from a sampled source taxonomy to a full target taxonomy. Occasionally there are multiple nodes in the reference taxonomy to which a source category node could be correctly mapped. To account for this fact, the manual mapping may define multiple correct mappings for each source category node. The manual mappings were collectively made by three independent individuals, in order to prevent bias.

Each algorithm performed a mapping for every combination of datasets. SCHEMA and the algorithm of Park & Kim carried out multiple mappings, with different parameter values for each combination. Both algorithms use a final score threshold, referred to as t_{final}, ranging from 0 to 1, with increments of 0.05. Furthermore, SCHEMA uses a threshold for node matching, denoted by t_{node}, with range 0.50 to 1 and increments of 0.025. The completed mappings, generated by the algorithms, are compared with the manual mappings, in order to obtain their performance measures. Though ordinary classification and confusion matrix measures apply, the situation is slightly different as there are n 'positive' classes (all target categories), and only one negative (null mapping). We therefore define the 'false positives' as number of mappings to an incorrect path (either wrong or null), and the 'false negative' as incorrect mappings to null. The 'true' classes are similar to those in binary classification.

5.4.2 Results

Table 5.1 presents a comparison of average precision, recall and F_1-score for every algorithm. Tables 5.2, 5.3, and 5.4 give a more detailed overview of the results achieved by SCHEMA, the algorithm of Park & Kim, and PROMPT, respectively.

As shown in Table 5.1, SCHEMA performs better than PROMPT and the algorithm of Park & Kim, on both average recall and F_1-score. The recall has improved considerably with 221% in comparison to the algorithm from Park & Kim,

Algorithm	Precision	Recall	F_1-score	Senses found	WSD accuracy
PROMPT	28.93%	16.69%	20.75%	n/a	n/a
Park & Kim	47.77%	25.19%	32.52%	5.70%	83.72%
SCHEMA	42.21%	80.73%	55.10%	82.03%	84.01%

Table 5.1: Comparison of the best average results for each algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score	t_{node}	t_{final}
A → ODP	27.27%	40.00%	34.12%	52.50%	35.90%	0.800	0.25
A → O.co	36.34%	49.40%	34.30%	82.69%	50.49%	0.850	0.15
ODP → A	57.49%	68.94%	51.70%	93.66%	71.24%	0.875	0.30
ODP → O.co	39.13%	50.70%	29.59%	95.03%	55.43%	0.850	0.25
O.co → A	53.72%	56.60%	29.13%	84.96%	65.83%	0.850	0.15
O.co → ODP	39.30%	45.80%	27.27%	75.52%	51.69%	0.925	0.30
Average	42.21%	51.91%	38.26%	80.73%	55.10%		

Table 5.2: Best results for SCHEMA

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score	t_{final}
A → ODP	35.77%	34.00%	57.89%	16.84%	22.90%	0.05
A → O.co	60.16%	47.20%	76.78%	25.61%	35.92%	0.00
ODP → A	37.06%	41.48%	51.94%	30.29%	33.33%	0.00
ODP → O.co	36.76%	35.87%	48.68%	25.09%	29.82%	0.10
O.co → A	61.14%	36.20%	52.11%	29.89%	40.15%	0.00
O.co → ODP	55.71%	36.60%	62.87%	23.42%	32.98%	0.50
Average	47.77%	38.56%	58.38%	25.19%	32.52%	

Table 5.3: Best results for Park & Kim algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score
A → ODP	13.55%	25.40%	44.17%	8.08%	10.12%
A → O.co	51.69%	45.40%	74.44%	22.02%	30.89%
ODP → A	20.20%	35.47%	46.44%	19.61%	19.90%
ODP → O.co	20.86%	29.86%	42.64%	16.18%	18.22%
O.co → A	50.00%	32.20%	45.96%	25.66%	33.92%
O.co → ODP	17.27%	25.80%	47.73%	8.57%	11.46%
Average	28.93%	32.36%	50.23%	16.69%	20.75%

Table 5.4: Best results for PROMPT

and 384% against PROMPT. This can be partly attributed to the ability of SCHEMA to cope with lexical variations in category names, using the Levenshtein distance metric, as well as the ability to properly deal with composite categories. Furthermore, SCHEMA maps a category node to its parent's mapping when no suitable candidate path was found, improving the recall when the reference taxonomy only includes a more general product concept. Achieving a high recall is important in e-commerce applications, as the main objective is to automatically combine the products of het-

erogeneous product taxonomies in one overview, in order to reduce search failures. A low recall means that many categories would not be aligned, which would mean that many products will be missing from search results. For this reason, it is generally better to map to a more general category rather than not mapping at all. Worthy to mention is the slight decrease in average precision for SCHEMA compared with the algorithm of Park & Kim: 42.21% against 47.77%. This is due to the fact that there is a trade-off between precision and recall: achieving a higher recall means that an algorithm has to map more categories, resulting in possible imprecision when the similarity between categories is low. Both SCHEMA and the algorithm of Park & Kim use configurable final thresholds to filter out weaker matches, but it cannot fully prevent mistakes from occurring. Despite the slightly worse performance on precision, SCHEMA manages to find a more suitable trade-off between precision and recall for product taxonomy mapping than PROMPT and the algorithm of Park & Kim. This is illustrated by the good performance on recall and the higher F_1-score of 55.10%. PROMPT uses a conservative mapping approach, well-suited for general ontology mapping, but unsuitable for e-commerce due to the small portion of mappings. The algorithm of Park & Kim performs better in this regard, especially on precision, but the recall is hampered by the fact that it neglects the existence of composite categories. Furthermore, it uses a rather strict lexical matching procedure between category names, in which a category name has to be a full substring of the other, creating issues when slight lexical variations occur. In addition, the disambiguation procedure from Park & Kim only manages to find a sense in WordNet in 5.70% of the total categories on average. Unfortunately, the rather good accuracy of disambiguation (83.72%) is therefore based on a very small amount of cases, making the number rather untrustworthy. The Lesk-based disambiguation algorithm employed by SCHEMA performs well on both the percentage of senses found and the accuracy, scoring 82.03% and 84.01%, respectively.

5.5 Conclusions & Future Work

This chapter proposes SCHEMA, an algorithm capable of performing automated mapping between heterogeneous product taxonomies in e-commerce. The main objective for developing SCHEMA is facilitating the aggregation of product information from different sources, thus reducing search failures when shopping online. To achieve this objective, SCHEMA utilizes word sense disambiguation techniques on category labels, based on the ideas from the algorithm proposed by Lesk (Lesk,

1986), in combination with the WordNet semantic lexicon. Furthermore, it deals with domain-specific characteristics, such as composite categories, and lexical variations in category labels. It employs a node matching function, based on inclusiveness of the categories in conjunction with the Levenshtein distance for the class labels, for finding candidate map categories and for assessing the path-similarity. The final mapping quality score is calculated using the Damerau-Levenshtein distance, with an added penalty for dissimilar nodes in the target path.

The performance of our algorithm was evaluated using three real-life datasets and compared with PROMPT and the algorithm of Park & Kim. This evaluation demonstrates that SCHEMA achieves a considerably higher average recall than the other algorithms, with a relatively small loss of precision. The average F_1-score resulted in 55.10% for SCHEMA, against 20.75% for PROMPT, and 32.52% for the approach of Park & Kim.

As future work, we would like to improve SCHEMA by making use of part-of-speech tagging. As a noun is often more important for concept similarity than an adjective, it makes sense to distinguish between them and treat them accordingly. Another possibility is to combine the hierarchical category structure with product information, as the data fields in product instances could yield extra information for the taxonomy mapping. Additionally, this work could support the implementation of a system that is capable of autonomously matching products and product taxonomies from different sources.

Chapter 6

Dynamic Facet Ordering for Product Search Engines*

FACETED *browsing is widely used in Web shops and product comparison sites. In these cases, a fixed ordered list of facets is often employed. This approach suffers from two main issues. First, one needs to invest a significant amount of time to devise an effective list. Second, with a fixed list of facets it can happen that a facet becomes useless if all products that match the query are associated to that particular facet. In this work, we present a framework for dynamic facet ordering in e-commerce. Based on measures for specificity and dispersion of facet values, the fully automated algorithm ranks those properties and facets on top that lead to a quick drill-down for any possible target product. In contrast to existing solutions, the framework addresses e-commerce specific aspects, such as the possibility of multiple clicks, the grouping of facets by their corresponding properties, and the abundance of numeric facets. In a large-scale simulation and user study, our approach was, in general, favorably compared to a facet list created by domain experts, a greedy approach as baseline, and a state-of-the-art entropy-based solution.*

*This chapter is based on the article "D. Vandic, S. Aanen, F. Frasincar, and U. Kaymak. Dynamic Facet Ordering for Faceted Product Search Engines. *IEEE Transactions on Knowledge and Data Engineering*, 2017, to appear"

6.1 Introduction

Nowadays, many Web shops make use of the so-called *faceted navigation* user interface (Hearst, 2006), which is in literature often referred to as 'faceted search' (Tunkelang, 2009). One of the reasons why faceted search is popular among Web shops is that users find it intuitive (Hearst et al., 2002; Kules et al., 2009). The term 'facet' has a rather ambiguous interpretation, as there are different types of facets. In this work, we refer to facets as the combination of a property and its value, such as `WiFi:true` or `Lowest price (€):64.00`. Furthermore, facets are usually grouped by their property in user interfaces, in order to prevent them from being scattered around, and, thereby, confusing the user. In other words, the facet properties, such as `Color`, are shown first, and each property presents the actual values (e.g., `Red`, `Green`, and `Blue`). Figure 6.1 shows an example of a faceted search user interface, where the same concepts apply (e.g., the 'Featured Brands' property with its values 'Samsung', 'Motorola', 'Nokia', etc.).

Faceted search is primarily helpful in situations where the exact required result is not known in advance. As opposed to product search using keyword-based queries, facets enable the user to progressively narrow down the search results in a number of steps by choosing from a list of query refinements, as opposed of having to provide the complete query at once. However, one of the difficulties with faceted search, especially in e-commerce, is that a large number of facets are available. In general-domain faceted browsing systems, it is not uncommon to simply display all facets. Displaying all facets may be a solution when a small number of facets is involved, but it can overwhelm the user for larger sets of facets (Sinha and Karger, 2005b).

Currently, most commercial applications that use faceted search have a manual, 'expert-based' selection procedure for facets (Amazon.com, 2017a; Kieskeurig.nl, 2017; Tweakers.net, 2016). However, selecting and ordering facets manually requires a significant amount of manual effort. Furthermore, faceted search allows for interactive query refinement, in which the importance of specific facets and properties may change during the search session. Therefore, it is likely that a predefined list of facets might not be optimal in terms of the number of clicks needed to find the desired product.

In order to deal with this problem, we propose an approach for dynamic facet ordering in the e-commerce domain. The focus of our approach is to handle domains with sufficient amount of complexity in terms of product attributes and values. Consumer electronics (in this work 'mobile phones') is one good example of such a domain. As part of our solution, we devise an algorithm that ranks properties by their

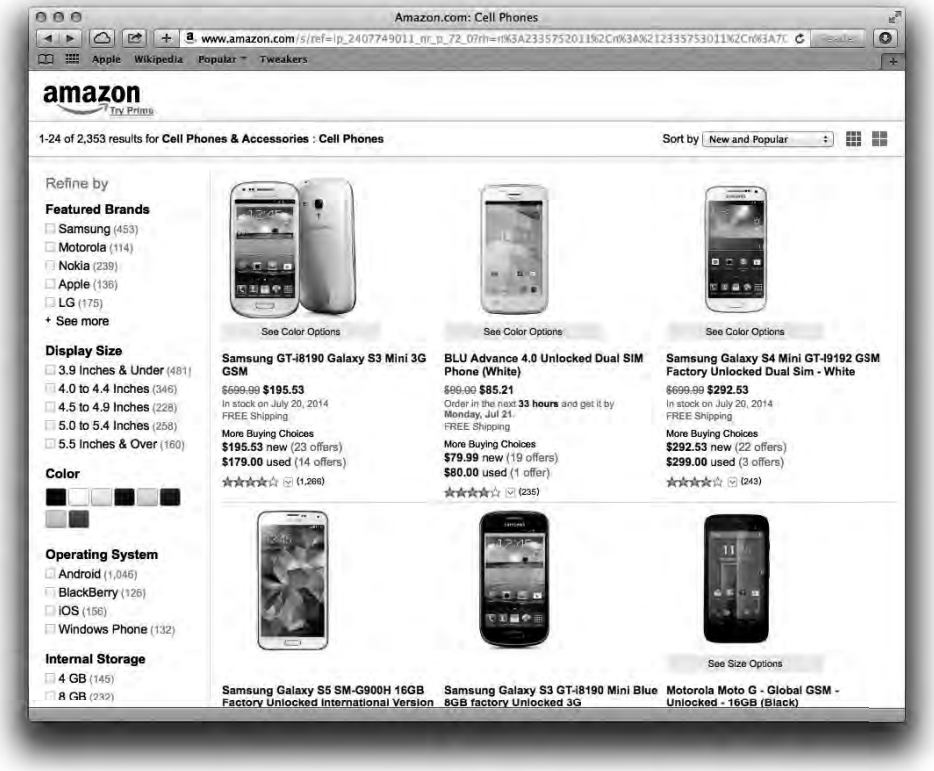

Figure 6.1: Screenshot of Amazon.com (Amazon.com, 2017a), showing a typical faceted search user interface in e-commerce.

importance and also sorts the values within each property. For property ordering, we identify specific properties whose facets match many products (i.e., with a high impurity). The proposed approach is based on a facet impurity measure, regarding qualitative facets in a similar way as classes, and on a measure of dispersion for numeric facets. The property values are ordered descending on the number of corresponding products. Furthermore, a weighting scheme is introduced in order to favor facets that match many products over the ones that match only a few products, taking into account the *importance* of facets.

6.2 Related Work

We can find approaches in the literature that focus on personalized faceted search (Herlocker et al., 1999; Koren et al., 2008b; Sacco and Tzitzikas, 2009a). However, we do not discuss these, as, unlike our approach, they require some sort of explicit user ratings. Therefore, we only consider related work that does not require any explicit user input other than the query.

The faceted search system proposed in (Dash et al., 2008) focuses on both textual and structured content. Given a keyword query, the proposed system aims to find the *interesting* attributes, which is based on how surprising the aggregated value is, given the expectation. The main contribution of this work is the *navigational* expectation, which is, according to the authors, a novel interestingness measure achieved through judicious application of p-values. This method is likely not to be suitable for the domain of e-commerce, where also small data sets occur and statistically deriving interesting attributes is not possible.

In (Liberman and Lempel, 2014), a framework for general-domain facet selection is proposed, with the aim to maximize the rank promotion of desired documents. There are many aspects in the proposed approach that make it not applicable in an e-commerce environment. First, two main assumptions are made: (1) the search process is initiated using a keyword-based query, and (2) the result is a ranked list of documents. These are serious limitations, as many Web shop users start with a facet selection instead of a keyword-based search, and product ranking is often not supported. Therefore, the framework we propose does not use these two assumptions. Second, the proposed solution does not consider multiple iterations of the search process (i.e., multiple drill-downs). Third, the authors do not differentiate between facet types. Consequently, numeric facets are treated in the same way as qualitative facets (discussed in Section 6.3), thereby losing their ordinal nature. Fourth, the authors assume that a user can only perform a drill-down using only conjunctive semantics. In our study, we use the common disjunctive semantics for values and conjunctive semantics for properties and take into account the possibility of drill-ups. This means that result set sizes are expected to both increase and decrease during the search session, either by deselecting a facet or choosing an additional facet in a property (e.g., selecting 'Samsung' when 'Apple' is already selected). Fifth and last, the authors do not distinguish in their approach between values (e.g., `Samsung`) and properties (e.g., `Brand`), they only consider the combination of values and properties.

In (Vandic et al., 2013a) the approach of (Liberman and Lempel, 2014) was extended and improved with a focus on product search. Using additional user as-

sumptions and the same theoretic approach as (Liberman and Lempel, 2014), two new methods for facet sorting were developed. Even though this approach improves upon the original algorithm, it still suffers from the same issues discussed above.

A more recent approach provides another method for facet selection (Kim et al., 2014b), or 'dynamic categorization' as the authors refer to it. The selection process is based on ontological data from a Semantic Web environment. However, due to a limited usage of rich ontological relationships, the algorithms can also be applied to semi-structured data, as suggested earlier in this chapter. The study is an extension of earlier work of the authors, which was based on the idea of selecting more descriptive facets using an entropy-based measure (Zhu et al., 2013). Similar to (Liberman and Lempel, 2014; Vandic et al., 2013a), this approach does not consider numeric facets and the use of disjunctive semantics for values.

Summarizing, most of the related approaches that have been proposed, with the exception of (Vandic et al., 2013a), do not explicitly focus on the e-commerce domain (Kim et al., 2014b; Koren et al., 2008b; Liberman and Lempel, 2014). Furthermore, these solutions often assume that there is a ranking of the results, based on a preceding keyword-based query or external data, which is often not the case for e-commerce. Also, our approach ranks properties and facets, unlike existing algorithms (Kim et al., 2014b; Koren et al., 2008b; Liberman and Lempel, 2014; Vandic et al., 2013a), which filter (or select) properties and facets. Lastly, none of the approaches from the literature that we discussed emphasize the performance aspect of the proposed algorithms. However, in order to be useful in practice, for most Web shops, it is important that the proposed solutions are responsive.

6.3 Facet Optimization Algorithm

Before discussing the details of our approach, we need to elaborate on the assumptions and the used terminology. From the perspective of user interface design, we distinguish between two main facet types: *qualitative facets* (e.g., `WiFi:true`) and *numeric facets* (e.g., `Lowest price (€):64.00`). We further distinguish between two types of qualitative facets: *nominal facets* and *Boolean facets*. Nominal facets are, for example, those for the property `Display Type`, and can have any nominal value. Boolean facets are for instance `Multitouch`, and have only three options from an interface perspective: `true`, `false`, or `No preference`.

Unlike previous studies, as discussed in Section 6.2, our approach treats numeric facets differently than qualitative facets. When creating facets from source data

(e.g., tabular data), every unique property-value combination is converted into a facet. For numeric facets, the same process is applied. However, numeric values can be widely dispersed, especially in large data sets. For facets, however, that would lead to a list of possibly hundreds of different values. One way to deal with that is to create predefined, fixed ranges of values and use these as facets. However, it is never certain whether the predefined ranges will match the user's preferences. Furthermore, fixed ranges can become useless when a result set has only products that fall into one predefined range. For our approach, we have chosen to let the user define custom ranges of values to select. In a product search engine, such custom ranges can be represented using a slider widget. From a technical point of view, however, these custom ranges are considered as selecting a set of facets in one click, i.e., each numeric value is still represented as a separate facet.

The approach we propose aims to order properties and facets in such a way that any individual product could be found quickly and effectively. We put the leading emphasis on property ordering, as we expect that it has the largest impact on the user effort. A straightforward way to order properties would be by presenting those properties on top that feature equal-sized facet counts for the facets of that property, which is an effect that is for instance visible in the entropy-based approach of (Vandic et al., 2013a). However, this would still require many clicks in total, possibly leading to long search times. Our approach aims to rank more specific properties higher. The reason behind is that we believe that users are to a limited extent, and possibly unconsciously, aware that selecting more unique features of the target product will result in a faster drill-down. Even in situations where this is not true, ranking more specific properties higher will increase the chance that the user will use specific facets for drill-down, resulting in a shorter search session duration. As an example consider a user who is searching for a Nokia smartphone capable of playing his collection of MP3 music, and both features are equally important. We expect the user to start by selecting `Brand:Nokia` instead of `Audio Formats:MP3`. The user may be aware of the fact that most smartphones are capable of playing MP3 audio, thus selecting that facet will not lead to a quick drill-down. Filtering only Nokia phones will presumably have a much larger impact on the result set than filtering phones that support MP3. The effect of ranking the individual facets (i.e., `Nokia` vs. `Samsung`) is assumed to be limited. We therefore expect that popularity is a more suited metric for this purpose.

When the user selects facets from a more specific property, the result set will decrease in size quickly. Since the most specific facets only apply to few products, it would be ineffective to present those on top, as the target product is unknown

to the system. Given that we assume that ordering properties has more effect than ordering facets, we therefore compute the impurity of properties as a whole, based on the specificity of its facets. Combined with weighting for the number of products on which it applies, this method will give us those properties and facets on top, that will most likely lead to the quickest drill-down for most of the possible target products. At the same time, the weighting that we introduce lowers the rank of properties with many missing values in the data, as those cannot be employed for drill-down.

A query in a search session is defined as a collection of previously selected facets. We have decided to apply disjunctive semantics to a selection of facets within a property. For facets across different properties, we use a conjunctive semantics. For example, selecting the facets `Brand:Samsung`, `Brand:Apple`, and `Color:Black` results in `(Brand:Samsung OR Brand:Apple) AND Color:Black`. Several e-commerce stores on the Web (e.g., Amazon.com and BestBuy.com) use the same principle, which, from a user experience point-of-view, is very intuitive.

Our approach assumes that users can undertake two types of actions: drill-down and roll-up. A drill-down is defined as an action of selecting one or more facets, leading to a reduction of the result set size. A roll-up action increases the result set size, which is likely to happen when the user notices that the selected facets are too strict. A roll-up action can be achieved in three ways: (1) selecting a qualitative facet from a property for which a selection already exists (e.g., adding `Brand:Samsung` to a query containing `Brand:Apple`), (2) deselecting the only selected facet of a property, and (3) broadening a numeric range. We will be using the notations described in Table 6.1, which will be described in further details in the following sections.

Figure 6.2 summarize the complete search session flow assumed in our approach. Throughout the search session, we assume that there exists a single target product d_u that the user wants to find, and that the user will eventually be able to find it. Although the user may not know the name of the product, (s)he will be able to identify it by means of the characteristics of the product (F_{d_u}). The process starts with a complete result set containing all products from the catalog D and an empty user query q. Our approach then initiates two processes, i.e., (1) computing the property scores and (2) computing the facet scores, discussed in Section 6.3.1 and 6.3.2, respectively. When the system completes, the user view is updated showing the properties and facets in the computed order.

In the next step, the user evaluates the result set size. If the result set size is too large to scan manually ($|D_q| > n$), the user will continue to drill-down. Otherwise, the user will scan the result set and check if the target product is found. If the target

D	Set of products (product catalog)
P	Set of properties
F	Set of facets
$F_p \subseteq F$, $(p \in P)$	Set of facets for property p
$F_d \subseteq F$, $(d \in D)$	Set of facets for product d
$q \subseteq F$	Query
$D_q \subseteq D$	Result set returned for query q
D_f, $(\forall d \in D_f : f \in F_d)$	Set of products associated to facet f
$r_q^O(f)$, $(f \in F_p)$	Rank of facet f for facet ordering scheme O in the result set (dependent on query q)
$r_q^O(p)$, $(p \in P)$	Rank of property p for facet ordering scheme O in the result set (dependent on query q)
$d_u \in D$	Target product for user u
X	Variable indicating user effort
M	Selected drill-down model in user simulation
n	Maximum number of products in the result set the user is willing to scan in the user simulation
t	Iteration indicator (state) of search session

Table 6.1: Summary of notations.

product is found, the search session is completed and considered successful. The user will perform a roll-up in the case that the desired product was not found, which will increase the result set size and the same process repeats again. In Section 6.4 we will further elaborate on the details of the employed simulation models for all these steps.

6.3.1 Computing Property Scores

We now discuss the details of computing property scores, shown as one of the first two processes in Figure 6.2. The outcome of the property scores is used to first sort the properties, after which the facet scores, discussed in the next section, are used to sort the values within each property. In Figure 6.3, we zoom into the main steps of computing the property score. As shown by the diagram, the score for each property is computed separately and can thus be done in parallel. Furthermore, the individual steps depend on the facet type (qualitative or numeric).

Disjoint Facet Counts

We designed the proposed algorithm in such a way that more specific facets and properties are ranked higher. To support the algorithm in identifying more specific facets, we introduce the *disjoint facet count*. This metric is used to compute the

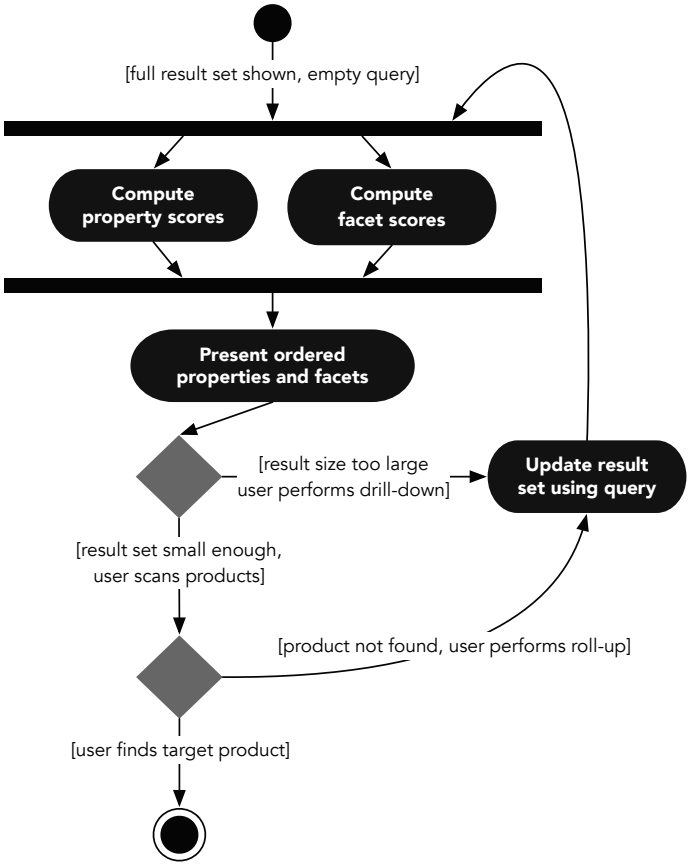

Figure 6.2: Activity diagram describing the main flow of a search session.

score for qualitative properties. The disjoint facet count is the number of products from the result set matching each facet f of property p. The classical facet count for a facet f, for a given query q, is defined as:

$$\text{count}(f, q) = |D_q \cap D_f| = \sum_{d \in D_q} \begin{cases} 1 & \text{if } f \in F_d \\ 0 & \text{if } f \notin F_d \end{cases} \tag{6.1}$$

The disjoint facet count is then defined as:

$$\text{disjointCount}(f, q) = \sum_{d \in D_q} \begin{cases} 1 & \text{if } F_p \cap F_d \equiv \{f\} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

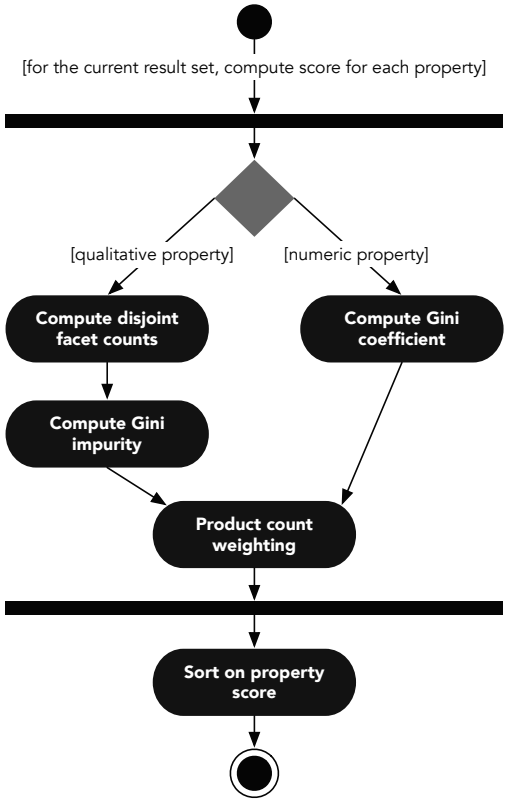

Figure 6.3: Activity diagram showing the individual steps in the property score computation process.

where p is the property of facet f, $f \in F_p$, and $\{f\}$ is the singleton set containing f. More general facets such as `Audio Formats:MP3` will thus have a low disjoint count, as most products that have this facet also support other audio formats besides MP3. Conversely, facets from the property `Brand` are likely to have relatively high counts, as most products are associated to only one brand.

In Table 6.2 we show the tabular product data of a data sample that was taken from our evaluation dataset from (Tweakers.net, 2016). Table 6.3 shows how the tabular data has been transformed into facets and the corresponding final scores.

Scoring Qualitative Properties

Figure 6.3 shows that qualitative properties are partly treated differently compared to numeric properties. For qualitative properties, we employ the *Gini im-*

Product Name	Property			
	Audio Formats	Brand	Diagonal Screen Size (inch)	Lowest Price (€)
Nokia 6230i	mp3	N/A	1.5	80.33
LG KU990 Viewty	aac, midi, mp3, mpeg 4, wav, wma	LG	3	79.00
Sony Ericsson C902	aac, mp3	Sony Ericsson	2	129.95
LG KF510	aac, mp3	LG	2.2	N/A
Apple iPhone 4	aac, aac+, aax, aax+, aiff, mp3, wav	Apple	3.5	459.95
LG Nexus 4 8GB	flac, mp3	LG	4.7	382.90
Samsung Galaxy S4	aac, ac3, amr-nb, eaac+, flac, mp3, ogg, wav, wma	Samsung	N/A	494.99

Table 6.2: Example data from Tweakers.net (2016).

purity (Breiman et al., 1984) to assess their 'uniqueness' or specificity in terms of describing certain products. We could have used Shannon's entropy (Shannon, 2001) for the same goal. Various studies have investigated this choice. In (Raileanu and Stoffel, 2004), the authors find that these two methods produce tree splits that are not significantly different from each other. One of the few differences that tend to be present, is that the Gini impurity tends to produce the most pure nodes (Breiman, 1996), which is why we chose to use it.

In the context of facet properties, we are looking for those properties with the highest impurity. At that point, it becomes desirable to initiate a new 'split', i.e., a facet selection, in order to reduce the impurity. We define the Gini impurity for facet selection as follows:

$$\text{giniImpurity}(p,q) = 1 - \sum_{f \in F_p} \left(\frac{\text{disjointCount}(f,q)}{\sum_{g \in F_p} \text{disjointCount}(g,q)} \right)^2 \tag{6.3}$$

where $p \in P_{\text{qualitative}}$ and $q \subset F$, with the fraction denominator being the total number of products from the result set associated to a a single facet from property p. It should be noted that since the relative frequency of products is represented by the fraction in Equation (6.3), the measure is independent of the number of products associated to values by means of property p.

Scoring Numeric Properties

In the previous section, we explained how the Gini impurity can be employed to score qualitative properties. It would be possible to use the same methods for numeric

Property & Facets					Scores		
Property	Facet	Facet Count	Disjoint Facet Count	Prod. Count Weighting	Gini Coeff.	Gini Impurity	Property Score
Audio Formats	aac	5	0	$\frac{1}{7}$	N/A	0.00000	0.00000
	aac+	1	0				
	aax	1	0				
	aax+	1	0				
	ac3	1	0				
	aiff	1	0				
	amr-nb	1	0				
	eaac+	1	0				
	flac	2	0				
	midi	1	0				
	mp3	7	1				
	mpeg4	1	0				
	ogg	1	0				
	wav	3	0				
	wma	2	0				
Brand	Apple	1	1	$\frac{6}{7}$	N/A	0.66667	0.57143
	LG	3	3				
	Samsung	1	1				
	Sony Erricson	1	1				
Diagonal Screen Size (inch)	1.5	1	1	$\frac{6}{7}$	0.21006	N/A	0.18005
	2.0	1	1				
	2.2	1	1				
	3.0	1	1				
	3.5	1	1				
	4.7	1	1				
Lowest Price (€)	79.00	1	1	$\frac{6}{7}$	0.35561	N/A	0.30481
	80.33	1	1				
	129.95	1	1				
	382.90	1	1				
	459.95	1	1				
	494.99	1	1				

Table 6.3: The computed scores for the considered properties in Table 6.2. This example uses parameter values $|D| = 7$, $|P| = 4$, and $q = \emptyset$. The value 'N/A' stands for 'not applicable' (e.g., Gini coefficient is only computed for numeric properties). Looking at the final property scores (last column), we can conclude that `Brand` is more important than `Audio Formats` and that the `Lowest Price (€)` is more important than `Diagonal Screen Size (inch)`.

facets as well, similar to related work in which numeric facets are treated as being qualitative (Kim et al., 2014b; Liberman and Lempel, 2014; Vandic et al., 2013a). However, this would lead to a loss of information, as each value would be treated as being a nominal. We could for instance imagine a result set of products in a similar price range. Regardless of the fact that the prices are similar, there is a good probability that most products will still have a unique value for price. In the data we used for evaluation, over 90% of the products has a unique price. However, when we disregard the fact that 'unique' prices may actually be quite similar, this would lead to a very high Gini impurity score. With property `Lowest Price (€)` being used in our example for drill-down, however, selecting a certain range of prices would still include most of the products, as their prices are similar. The property is thus not effective for drill-down.

Therefore, for numeric properties, we have chosen to use the knowledge about the distribution of the numeric values for computing property scores. It is fairly straightforward to imagine that it may be useful to drill-down using a numeric property when the values for the result set are widely dispersed. When the facets are nearly uniformly distributed over the complete range of values, a drill-down using a user-defined range would lead to a large reduction of the result set. On the other hand, when most of the values are similar, such as in the example of having a result set with products of the same price range, drilling down using a numeric property will hardly reduce the result set size and thus be ineffective to use. For assessing the dispersion of numeric facets, we employ the Gini coefficient (Ceriani and Verme, 2012). The measure has proved to work effectively for examining the distribution of values. We adapt the original Gini index for use in our context:

$$\begin{aligned}
\text{giniCoefficient}(p,q) &= \frac{1}{m}\left(m+1-2\left(\frac{\sum_{i=1}^{m}(m+1-i)f_i}{\sum_{i=1}^{m} f_i}\right)\right) \\
&= \frac{2\sum_{i=1}^{m} if_i}{m\sum_{i=1}^{m} f_i} - \frac{m+1}{m} \\
\text{given } f_i &\in F_p^* \text{ for } i = 1 \text{ to } m \\
F_p^* &= \{f_i \mid f_i \in F_p \cap F_d,\ d \in D_q,\ f_i \leq f_{i+1}\} \\
m &= |F_p^*| \\
p &\in P_{\text{quantitative}}
\end{aligned} \tag{6.4}$$

where F_p^* represents the values for numeric property p for the products in the result set, indexed in non-decreasing order ($f_i \leq f_{i+1}$), with f_i being the facet ranked at index i. This measure gives us an indication of the dispersion of values. The more dispersed, the more likely it is that a drill-down using this property will largely affect the result set.

In Table 6.3 we give the Gini coefficients for the considered properties from the example data (shown in Table 6.2). As an example, we will now compute the Gini coefficient for `Diagonal Screen Size (inch)`. We assume that the query is empty and thus all 6 facets can be included in the computation. By ordering these facets in an ascending way, we obtain $F_p^* = \{1.5, 2.0, 2.2, 3.0, 3.5, 4.7\}$ and $m = 6$. The index is then given by:

$$\begin{aligned}
\mathrm{G} &= \frac{2\sum_{i=1}^{m} i f_i}{m\sum_{i=1}^{m} f_i} - \frac{m+1}{m} \\
&= \frac{2 \cdot (1 \cdot 1.5 + 2 \cdot 2.0 + 3 \cdot 2.2 + 4 \cdot 3.0 + 5 \cdot 3.5 + 6 \cdot 4.7)}{6 \cdot (1.5 + 2.0 + 2.2 + 3.0 + 3.5 + 4.7)} - \frac{6+1}{6} \\
&= \frac{2 \cdot (69.8)}{6 \cdot (16.9)} - \frac{7}{6} \\
&= 0.21006
\end{aligned}$$

which is the index that is also mentioned in Table 6.3. From the table we can also conclude that the Gini for `Lowest Price (€)` is higher, suggesting that the values for that property are more dispersed than those of `Diagonal Screen Size (inch)`. Similar to the Gini impurity for qualitative facets, the Gini coefficient for properties is independent of the number of products that have this property. Therefore the measure needs to be weighted, which is what we will discuss in the next section.

Product Count Weighting

With the Gini impurity and the Gini coefficient, we now have metrics to score both qualitative and numeric properties. As mentioned in the previous sections, this score is independent from the number of products on which it is based. This could possibly lead to problems, as properties that occur within few products will obtain a relatively high score. To compensate for this, we introduce the *product count weighting*. The product count weighting is used to transform the Gini indices, resulting in the final *property score*. Additionally, it provides a way to cope with missing values, as properties with many missing associations will be ranked lower. We define the final

property score as:

$$\text{propertyScore}(p, q) = gini(p, q) \cdot \sum_{f \in F_p} \frac{\text{disjointCount}(f, q)}{|D_q|} \tag{6.5}$$

where *gini* is either the Gini impurity or the Gini coefficient (depending on the property type). The term with which *gini* is multiplied is the product count weighting term. Table 6.3 shows the product count weighting for each property. If we take for instance property `Lowest Price (€)`, we can compute the property score using Eq. 6.5 and the Gini from the table as follows:

$$\begin{aligned} \text{score} &= 0.35561 \cdot \frac{1+1+1+1+1+1+1}{7} \\ &= 0.35561 \cdot \frac{6}{7} \\ &= 0.30481 \end{aligned}$$

As we can see, the second term, the product count weighting, is $\frac{6}{7}$, corresponding to the value in Table 6.3 for `Lowest Price (€)`. Multiplying it by the Gini score obtained earlier this gives us the property score, by which we can rank properties using $r_q^O(p)$, with O referring to our approach in this case.

One should note that, strictly speaking, the Gini impurity and the Gini coefficient are not directly comparable to one another. For our use case, however, this does not lead to problems, as both measure the specificity of a property, one for qualitative and one for quantitative. Another approach to handling qualitative and quantitative properties would be to try to find unified similarity measure. However, we believe that it is difficult to compare qualitative and quantitative properties in the first place and having two separate lists of facets (one for qualitative properties and one for quantitative properties) would make the browsing of products more difficult for the end user. The empirically obtained results suggest that this approach is working adequately in practice.

6.3.2 Computing Facet Scores

We now discuss the details of computing facet scores, shown as one of the first two processes in Figure 6.2. For numeric properties, value ordering is neglected, as these are often represented with a slider widget in user interfaces. For qualitative properties our approach employs the facet count from Equation (6.1), ranking facets descending

on count, per property. As the target product is unknown to the system, this will increase the chance that a facet matching the target product is placed on top.

In the evaluation, we compare our approach to the one proposed in (Kim et al., 2014b). To have a fair comparison, we have implemented a version of their method that includes the same facet sorting as our algorithm, as the authors themselves have neglected this aspect. The difference in results can thus be completely accounted to property sorting.

6.4 Evaluation

In this section, we discuss the evaluation of our proposed approach. The evaluation is based on (1) simulated user sessions, where the simulation framework is derived from previous literature, and (2) a study involving real users.

6.4.1 Experimental Framework

Figure 6.4 gives an overview of the concepts that underlie the evaluation framework. In our experimental setup, one simulation process represents an individual search session, which we will refer to as an experiment. Each experiment contains the selection of one drill-down model, one ordering scheme, and one target product. Furthermore, some of the drill-down models and ordering schemes contain stochastic aspects. Therefore each experiment is repeated 50 times, in order to reduce the variability of the results. For each experiment we record six different metrics. For the target products, we have decided to use every product in our data set as a target product d_u, in order to get the most reliable results from the data that we have available.

Drill-Down Models

There are three drill-down models that we consider, based on the ones proposed in (Koren et al., 2008b; Liberman and Lempel, 2014). These drill-down models rely on five key assumptions, i.e., (1) rationality: the user will end the session once target product is found, (2) practicality: the user will use no more than a fixed number of clicks when looking for the target product, (3) feasibility: the user will perform a roll-up when the target product disappears from the result set, (4) omnisciency: once presented with the facets, the user knows which ones belong to the target product, and (5) linearity: the user scans the properties from top to bottom. Because some

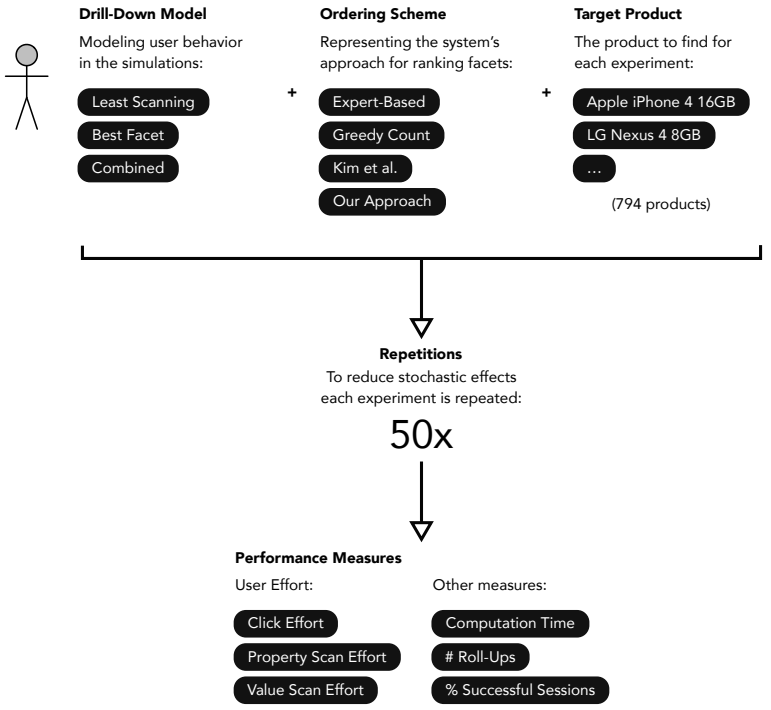

Figure 6.4: Overview of the various concepts and phases underlying the evaluation framework. The 50 repetitions are applied to all combinations that include the Combined Drill-Down Model, as this is the only stochastic drill-down model. All considered performance measures are averaged over these 50 repetitions and the t-tests were performed using the metrics for each target product as samples.

of these assumptions are very restrictive, all drill-down models relax one or more of these assumptions. It is, however, useful to identify the theoretical boundaries that may apply to user behavior in order to make a simulation that is more realistic.

In the *Least Scanning Drill-Down Model*, M_S, the user u scans the list of facets F starting from the top. When u encounters a facet $f \in F_{d_u}$ (a facet associated with the target product), (s)he will select that facet without further scanning.

The *Best Facet Drill-Down Model*, M_B, assumes that when u is searching for d_u and is scanning F, u identifies the single facet that will reduce the result set size most, while d_u is still included in the result set. In other words, the user will choose the 'best' drill-down option, regardless of the property or facet rank. The Best Facet Drill-Down Model minimizes the number of clicks at the expense of possibly scanning more facets.

Last, the *Combined Drill-Down Model* M_C provides a more realistic simulation of user behavior by allowing faulty selections (i.e., clicks that will exclude the target product from the result set). This model assumes that the user u scans the list of facets F starting from the top. When u encounters a facet f (s)he will consider selecting f with probability α_f when the target product d_u is associated with this facet, and β_f when it is not. For α_f and β_f we use:

$$\alpha_f = \frac{\alpha}{|F_p \cap F_{d_u}|}, \quad \beta_f = \frac{\beta}{|F_p \setminus F_{d_u}|} \tag{6.6}$$

where $f \in F_p$ and $\alpha + \beta = 1$. Once u has a certain facet in consideration, the decision whether to select it will be made stochastically using the *Facet Importance Factor* γ_f, defined as follows:

$$\gamma_f = \begin{cases} 1 - \frac{r_q^O(f)-1}{|F_{d_u} \setminus q|-1} & \text{if } f \in F_{d_u} \ (\alpha \text{ case}) \\ 1 & \text{if } f \notin F_{d_u} \ (\beta \text{ case}) \end{cases} \tag{6.7}$$

where $r_q^O(f)$ is a function that returns the rank of f in a list of candidate facets $F_{d_u} \setminus q$ (unselected facets associated with d_u), and the fraction denominator $|F_{d_u} \setminus q| - 1$ is a normalization factor to bring the measure between 0 and 1. When a facet is not selected during a scan, either due to the stochastic effect from α_f or β_f, or due to its Facet Importance Factor γ_f, the user will resume scanning the following facet until a selection has been made.

Ordering Schemes

For effectively evaluating the performance of our approach, we perform a comparison with other ordering schemes. The *Expert-Based* scheme is the fixed-order scheme from (Tweakers.net, 2016), which is created manually by a team of dedicated editors. Since manually defined schemes are used in nearly all current applications on the Web, it provides a useful comparison with dynamic ordering methods such as the one proposed in this study. The *Kim et al.* approach, proposed in (Kim et al., 2014b), is a state-of-the-art method for sorting properties. Their proposed scheme fits the e-commerce domain well and because it is an entropy-based approach, it is an interesting candidate in the comparison. Although the original paper suggested source data in the form of an ontology, the algorithms can be applied to semi-structured data as well, as the authors also suggest. The last baseline we employ is the *Greedy Count* scheme. Greedy Count appears regularly in related work as a simple baseline

for evaluation (Koren et al., 2008b; Liberman and Lempel, 2014). It orders properties and facets descending on the number of matching products. In order to fit into our environment, the Greedy Count uses the following definition for the property score:

$$\text{greedyCountPropScore}(p, q) = \frac{\max_{f \in F_p} \text{count}(f, q)}{|D_q|} \tag{6.8}$$

The properties are thus ordered based on the maximum of the facet counts of their values. The facets themselves are naturally sorted on facet counts as well, as defined in our approach and the one we implemented for the *Kim et al.* approach. This means that all automatic approaches that we evaluate use the same facet ordering technique, which makes the comparison more fair.

Performance Measures

The performance of the ordering schemes given the different drill-down models is measured using various metrics. We consider three user effort metrics. First, the *click effort* X_c measures how often a facet was (de)selected or a range was adapted. Second, the *property scan effort* X_p measures how much effort is put in scanning properties and is defined by:

$$X_p = \sum_{t,\, |D_q^t| > n \,\wedge\, t \leq 100} \frac{r_q^O(p_t^M)}{|P|} \tag{6.9}$$

where n is the maximum number of products in the result set the user is willing to scan, and p_t^M refers to the property that is selected by the user given drill-down model M at iteration t. Last, the *value scan effort* X_f measures how much effort is put in scanning values and is defined by:

$$X_f = \sum_{t,\, |D_q^t| > n \,\wedge\, t \leq 100} \frac{r_q^O({f_t}^M)}{|F_p|} \tag{6.10}$$

where f_t^M refers to the facet $f \in F_p$ that is selected by the user given drill-down model M at iteration t. As there is no list of facets for quantitative properties, the scanning effort for selecting a range of numerical values is defined as $X_f = 0$.

Besides the user effort metrics, we record three other measures during the experiments. First, the total computation time needed for one experiment to compute or retrieve the order of facets. Second, the number of roll-up user actions that were needed on average in each search session (applicable only to the Combined Drill-

Down Model). Last, the percentage of successful sessions represents the number of experiments in which the target product was found within the maximum number of clicks.

For our experiments, we have gathered data from Tweakers Pricewatch (Tweakers.net, 2016). Tweakers PriceWatch is the largest Dutch price comparison Web site, with a comprehensive collection of product characteristics available in tabular format. The complete catalog contains 794 mobile phones, 53 properties, and 1,816 facets. Of these facets 348 are qualitative, against 1,468 numeric facets. The imported data was cleaned and converted to a more structured format (i.e., we used a custom, predefined schema). With 3 drill-down models, 4 ordering schemes, 794 mobile target products, and 50 repetitions for the Combined Drill-Down Model, we have run over 150,000 experiments, storing over half a terabyte of experimental data. We also implemented our approach in a Web application, shown in Figure 6.5. Obviously, running all these experiments on one computer is unfeasible. Instead, we used a cluster of 100 instances, hosted on Amazon Web Services (Amazon.com, 2017b), to run the experiments. In the end, we stored half a gigabyte of performance metrics for the different experiments.

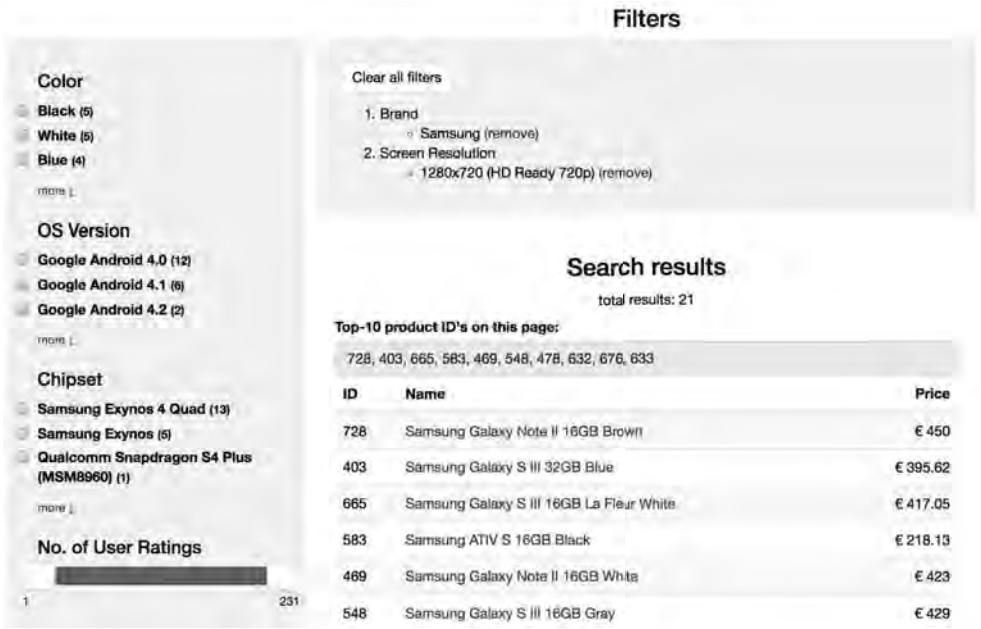

Figure 6.5: Screenshot of the Web application that implements our approach, available at `http://facet-sorting.eur.dvic.io`.

6.4.2 Results from the simulated experiments

In this section we present and discuss the results obtained from our experiments. We have performed t-tests to assess whether the observed differences for the click, property scan effort, and value scan effort are significant. Based on these tests we can conclude that all the found differences are significant, with the largest p-value being 0.00026.

Tables 6.4, 6.5, and 6.6 show the results for Least Scanning, Best Facet, and Combined Drill-Down models, respectively. We can make several important observations. First, in terms of the number of clicks, our approach seems to outperform the other methods, except in the case of the Best Facet Drill-Down Model, where each approach performs equally well. Furthermore, for the Combined Drill-Down Model, our approach results in the lowest number of roll-ups and the highest percentage of successful sessions.

Second, we observe that our approach, in most cases, performs best in terms of property and facet scan effort, except for the Combined and Least Scanning Drill-Down Model, respectively. However, although the found differences are statistically significant, it can be argued that they are not relevant, as there were no large effect sizes found. Furthermore, we assume that in practice the property and facet scanning efforts are not the key factors that contribute to the true perceived user effort. We assume that the number of clicks and the responsiveness of the approaches play a much more important role here.

Third and last, in terms of computational time, our approach outperforms the other automatic approaches, often needing orders of magnitude less time to return the sorted facets for a query. For example, the total computation time for the *Kim et al.* method, on average, is more than 1 second per click. Our approach needs approximately 100 milliseconds per click, which fits the requirements of Web shops and other e-commerce applications, where latencies in terms of seconds are found to be highly undesired (Nah, 2004). The reason for why the method of *Kim et al.* is slower stems from the fact that it relies on computing the the conditional entropy for every property pair p_i, p_j $(p_i \neq p_j)$, which in turn relies on computing the entropy between the property p_i and all property values $b \in V_j$, where V_j are all the values for property j.

We have also found that ranking specific facets higher does sometimes have a downside. This occurs when a facet is so specific that the user has difficulties to identify it. For instance, the qualitative `Screen Resolution` property is ranked relatively high initially. There are so many different screen resolutions available that

	Ordering Scheme			
	Expert-Based	Greedy Count	Kim et al.	Our approach
user effort:				
# clicks (X_c)	4.0	28.2	19.7	2.3
# clicks std dev	1.24	18.65	14.04	0.68
prop scan effort (X_p)	0.0538	0.1914	0.0630	0.0267
prop scan effort std dev	0.0273	0.0891	0.0351	0.0124
facet scan effort (X_f)	0.1462	0.2438	0.4550	0.2111
facet scan effort std dev	0.0908	0.0952	0.1516	0.1718
other measures:				
computation time (ms)	4	23,386	49,818	187
computation time std dev	3.7	26,832.4	45,129.9	74.9
successful sessions (%)	100.00%	100.00%	100.00%	100.00%

Table 6.4: Results for the Least Scanning Drill-Down Model, with $n = 10$, $\alpha = 0.9$, and maximum 100 iterations.

	Ordering Scheme			
	Expert-Based	Greedy Count	Kim et al.	Our approach
user effort:				
# clicks (X_c)	1.5	1.5	1.5	1.5
# clicks std dev	0.52	0.52	0.52	0.52
prop scan effort (X_p)	0.3474	0.7232	0.5804	0.2399
prop scan effort std dev	0.2607	0.2091	0.1939	0.2257
facet scan effort (X_f)	0.4659	0.4796	0.4946	0.4547
facet scan effort std dev	0.2730	0.2736	0.2695	0.2764
other measures:				
computation time (ms)	2	25	1,507	160
computation time std dev	0.9	213.2	638.1	61.9
successful sessions (%)	100.00%	100.00%	100.00%	100.00%

Table 6.5: Results for the Best Facet Drill-Down Model, with $n = 10$, $\alpha = 0.9$, and maximum 100 iterations.

the user might be overwhelmed by the decision to choose one. The users might also be indifferent with respect to the different resolutions, which makes the property less attractive. At the same time, the property `Lowest Price (€)`, which is generally considered a more useful property for filtering products, is ranked lower. This shows that achieving faster drill-down does not only involve mathematical optimization but also taking into account user experience and behavior.

	Ordering Scheme			
	Expert-Based	Greedy Count	Kim et al.	Our approach
user effort:				
# clicks (X_c)	30.7	62.9	59.8	18.8
# clicks std dev	20.05	27.98	20.01	9.77
prop scan effort (X_p)	0.1220	0.1681	0.1524	0.2268
prop scan effort std dev	0.0232	0.0255	0.0297	0.0261
facet scan effort (X_f)	0.3904	0.4842	0.5443	0.3075
facet scan effort std dev	0.0599	0.1100	0.0325	0.0308
other measures:				
computation time (ms)	16	118,155	113,336	2,843
computation time std dev	12.6	72,772.1	53,871.0	2,094.0
# rollups mean	10.7	10.0	16.6	6.2
successful sessions (%)	90.96%	64.00%	79.53%	99.07%

Table 6.6: Results for the Combined Drill-Down Model, with $n = 10$, $\alpha = 0.9$, and maximum 100 iterations.

6.4.3 Results using the experiment with real users

Besides the extensive experiments performed using simulation, we also performed an experiment with real users. The experiment consisted of 10 small tasks*, where each task would take the user approximately one minute to complete. The tasks were generated by a script that randomly selects products and includes all properties of the product in the task description. However, for the sake of brevity, properties with multiple values (e.g., 'Audio Formats') were reduced to one (randomly selected) value. For each task, the user was given a set of product features. The users were instructed to find the product(s) that matched all the given properties. We evaluated two systems, where each user performed the first half of the tasks with one system and the second half of the tasks with the other system. The order of the systems was alternated among users in order to compensate for the the learning effect that may occur. One system was a faceted search interface using the algorithm proposed in this dissertation† and the other system was a 'standard' faceted search interface‡. The 'standard' faceted search interface has no special features other than those commonly encountered on the Web and employs a fixed facet list, which is obtained from the Web shop from which the data set is originating (Tweakers.net, 2016).

*all tasks are available at `https://db.tt/5DRnsIhS`
†available at `http://facet-sorting.eur.dvic.io`
‡available at `http://std-prod-search.eur.dvic.io`

We had a total of 27 users who participated in the experiment, consisting of 17 males and 10 females. There were 19 users that were between 20 and 30 years old, 6 users that were between 31 and 40 years old, and 2 users that was between 40 and 50 years old. These users were mostly students and colleagues from our university and other universities and there was no financial reimbursement for the participation in the experiment.

Table 6.7 shows the behavior of the users who participated in the experiment, for each of the systems. We can see that most users chose to filter based on the qualitative facets (such as the brand), as indicated by the event 'List facet select'. We notice that users needed less numeric facet changes with our approach than with the standard approach (event 'Numeric facet change'). The results from our user study also suggest that users do not reformulate the query often. Table 6.7 shows that the filters were cleared only twice in the whole study (event 'Clear all filters'). We can also see that the users spend more time drilling down or rolling up (events 'List facet select' and 'List facet deselect'). Using a paired t-test (measured per task), we can conclude that the users significantly had less interaction (i.e., less events) with our approach than with the standard approach ($p = 0.001867$). We also considered the user effort in terms of how long it took the users to complete the tasks. On average, the users spent 72.4 seconds per task with our approach and 79.9 seconds with the standard approach. The standard deviation is 33.2 seconds for our approach and 33.0 seconds for the standard approach. A paired t-test shows that the difference is significant although the evidence is not very strong ($p = 0.047170$). This might be due to the fact that there is a large difference among users and 27 users is too little to factor out that effect.

Event type	Standard approach	Our approach
List facet select	364	376
Toggle collapsed	182	143
Numeric facet change	198	84
List facet deselect	18	2
Boolean facet change	5	2
Numeric facet remove	2	4
Boolean facet remove	4	0
Clear all filters	2	2
Change page number	2	3

Table 6.7: Event counts in the user experiments.

6.5 Conclusion

Nowadays most Web shops employ a form of faceted navigation in their user interface, often referred as 'faceted search'. This type of search has many advantages over keyword-based search, as it enables the user to progressively narrow the search results in a number of steps by choosing from a list of query refinements, instead of having to provide the query at once. However, the current state of faceted search is suffering from two major issues. First, most Web shops use a 'expert-based' selection procedure for facets (Amazon.com, 2017a; Kieskeurig.nl, 2017; Tweakers.net, 2016), requiring significant manual work as often many facets are involved. Second, a fixed facet list does not optimally make use of the interactive nature of faceted search, i.e., the importance of facets can change drastically depending on the query.

In this work, we proposed an approach that automatically orders facets such that the user finds its desired product with the least amount of effort. The main idea of our solution is to sort properties based on their facets and then, additionally, also sort the facets themselves. We use different types of metrics to score qualitative and numerical properties. For property ordering we want to rank properties descending on their impurity, promoting more selective facets that will lead to a quick drill-down of the results. Furthermore, we employ a weighting scheme based on the number of matching products to adequately handle missing values and take into account the property product coverage.

We evaluate our solution using an extensive set of simulation experiments, comparing it to three other approaches. While analyzing the user effort, especially in terms of the number of clicks, we can conclude that our approach gives a better performance than the benchmark methods and in some cases even beats the manually curated 'Expert-Based' approach. In addition, the relatively low computational time makes it suitable for use in real-world Web shops, making our findings also relevant to industry. These results are also confirmed by a user-based evaluation study that we additionally performed.

In future we would like to replicate our study on a different domain than cell phones, thereby addressing one of the limitations of the current evaluation. Also we would like to investigate the use of other metrics, such as facet and product popularity, for determining the order and optimal set of facets.

Chapter 7

Approximate Search and User Preference Ranking in Faceted E-commerce Search*

One of the problems that e-commerce users face is that the desired products are sometimes not available and Web shops fail to provide similar products due to their exclusive reliance on Boolean faceted search. User preferences are also often not taken into account. In order to address these problems, we present a novel framework specifically geared towards approximate faceted search within the product catalog of a Web shop. It is based on adaptations to the p-norm extended Boolean model, to account for the domain-specific characteristics of faceted search in an e-commerce environment. These e-commerce specific characteristics are, for example, the use of quantitative properties and the presence of user preferences. Our approach explores the concept of facet similarity functions in order to better match products to queries. In addition, the user preferences are used to assign importance weights to the query terms. Using a large-scale experimental setup, we conclude that the proposed algorithm outperforms the considered benchmark algorithms.

*This chapter is based on the article "D. Vandic, L. Nederstigt, F. Frasincar, and U. Kaymak. Approximate Search and User Preference Ranking for Faceted Navigation. *ACM Transactions on the Web*, 2017, under review."

7.1 Introduction

E-commerce has nowadays become very popular among consumers. However, there are many Web shops with very large product catalogs, which makes it difficult for users to find their desired product. It has been shown that many users encounter this difficulty while shopping online, which can negatively impact the turnover for Web shop owners (Horrigan, 2008). This emphasizes the need for better search interfaces for browsing through product catalogs on the Web.

Throughout the years various interaction paradigms have been proposed to deal with the above problem. Faceted search is one of the most popular paradigms for browsing structured data in information systems (Hearst, 2006), especially product catalogs in e-commerce stores. This paradigm is useful for exploratory search, as users can iteratively select facets that they find important to constantly refine their query and enhance the result set (Kules et al., 2009). Furthermore, it has been shown that faceted search is perceived as easy to understand and effective at guiding people to relevant documents within catalogs (Fagan, 2013).

Despite the advantages of faceted search, there are also some serious drawbacks. Traditionally, faceted search imposes a rather strict Boolean model on whether a document matches the query terms: either it matches the query or it does not. While this model allows for a quick drill-down into the catalog, it also means that a user might miss some potentially desired products if they do not completely match the query. This results in a user having to change the query afterwards to include more documents, which increases search time and makes exploratory search more error-prone.

Such strict faceted search is not desired in e-commerce, as users usually do not have completely strict requirements. For example, a user searching for a smartphone under €200 with '*Android*' as Operating System and '*Samsung*' as brand, might also consider Android smartphones from other brands if they are much cheaper than similar phones from Samsung. At the same time, having Android as the Operating System could be a hard requirement for the user. Another problem with strict faceted search is that there is no adequate ranking method. This is usually resolved by letting the user sort the list on one metric, such as price or popularity, but it would be more useful if the products could be sorted on how well they match the query. For this purpose, the user should also be able to rank the specified requirements on importance (e.g., a battery life requirement might be more important than a color requirement).

Most of the currently proposed approaches that aim to tackle this problem are not specifically geared towards the domain of e-commerce. In particular, the existence of quantitative facets, such as price, might pose a problem to these algorithms, as they do not consider nearly identical numerical values as being similar to a certain degree. Furthermore, past approaches for e-commerce also do not allow the user to explicitly specify which requirements are important and which ones are not. To our knowledge, a faceted search method that unifies the ranking of products and the ranking of user requirements has not yet been proposed.

In this chapter, we propose a novel algorithm specifically geared towards approximate faceted search within a product catalog of a Web shop. The main focus is on improving the retrieval of relevant products, taking into account the importance of the specified user requirements. We adapt the p-norm extended Boolean model (Salton et al., 1983) to cope with the domain-specific characteristics of faceted product search. The adaptations consist of a facet similarity function, both for quantitative and qualitative product properties. We also propose a term weighting method that takes in account user preferences. The algorithm is implemented as part of a Web application, with real-life data obtained from Tweakers.net Pricewatch (Tweakers.net, 2016), which is a Dutch Web shop price aggregation service. An extensive evaluation is performed by means of simulations and a battery of quality and performance metrics.

7.2 Related Work

In the literature, we can find some proposed approaches that aim to improve exploratory search within a product catalog. Burke (2002) proposes a recommender system based on case-based reasoning. This system matches a user's need with a product based on previously solved user needs that are similar to the current user's needs. The authors augment this approach by combining it with example critiquing, which shows a product to the user and then allows the user express his/her dissatisfaction with a particular facet of that product. This information can then be used to find another product that has a better value for this facet, possibly at the cost of having a worse value for other facets, thereby facilitating trade-off-based navigation.

Similarly, in (Pu and Chen, 2005) an approach is proposed that uses example critiquing techniques. They propose an interaction paradigm in which users first declare their initial preferences, after which some matching example documents are displayed. The user subsequently examines these examples and either accepts one

document or performs critiquing on one of the examples. This process consists of improving some of the attribute values by allowing a compromise on one or more other attribute values. Afterwards, the system applies a decision strategy called the weighted additive sum rule, which multiplies each attribute value of a document with the corresponding importance weight in the query, followed by adding up these values to obtain an overall score. The documents with the highest scores are displayed to the user, after which the user can either continue to perform exploratory search or accept one of the presented documents. The authors show that users were able to improve their decision accuracy by using the trade-off support provided by this system.

In (Li et al., 2011), the authors examine an approach that employs concepts from the field of utility theory. They argue that buying a product is different from finding relevant products, therefore it is possible to calculate the expected utility of products and rank them based on the largest utility surplus. The utility surplus is defined as the difference between two components, namely the utility of the product and the utility of money. The utility of the product is defined as the sum of expected utilities for each of its characteristics, whereas the utility of money is defined as an increasing concave function. In order to obtain the expected utilities, they collect aggregated data about market shares and demand, from which they derive an estimated utility. They show that users tend to prefer their surplus-based ranking when searching for hotels in comparison to other considered rankings.

Although faceted search is regarded as a useful means to explore a catalog, care needs to be taken that the interface does not become too cluttered and obscured, which could result in a diminished user performance due to information overload (Sinha and Karger, 2005a). General concepts from the literature for implementing faceted navigation are explained and reviewed in Sacco and Tzitzikas (2009b). One of the ideas discussed for reducing the overload of information is to either filter or order the facets according to their importance.

Displaying all facets may be a solution when a small number of facets is involved, but it can overwhelm the user for larger sets of facets (Sinha and Karger, 2005b; Zheng et al., 2013). Studies such as Dash et al. (2008); Herlocker et al. (1999); Kim et al. (2014a); Koren et al. (2008a); Sacco and Tzitzikas (2009a); Vandic et al. (2013b) focus primarily on optimizing the decision of which facets to show in the user interface. These algorithms aim to reduce user effort by ordering the facets in such a way that the most important ones are listed first.

The study in Kim et al. (2014a) is based on earlier work of the authors which coined a basic idea for selecting more descriptive facets using an entropy-based mea-

sure (Zhu et al., 2013). The approach focuses on Semantic Web and Linked Data data sets, similar to the approaches in Arenas et al. (2014a,b); Cuenca Grau et al. (2016); Stolz and Hepp (2015). The reduction in user effort is achieved by calculating gain ratios for each facet, which is based on conditional entropy. The conditional entropy indicates how much the value for a facet can be predicted by the value for another facet. By summing up all the gain ratios of the facets belonging to each property, the total gain ratio per property is computed. Last, the properties are ranked in decreasing order of total gain ratio and presented to the user. This process is repeated after each interaction between the user and the search engine.

Even though the previously discussed approaches are specifically geared towards e-commerce, they have various shortcomings. Approaches such as (Li et al., 2011), which rely on well-estimated user statistics, are not usable in practice. The reason for this is that some Web shops do not have the infrastructure to collect the user data or that the product catalogs are relatively large and changing rapidly. Other methods, such as (Burke, 2002; Pu and Chen, 2005), do not rely on previously collected user statistics, but they suffer from other issues. One of the main arguments against these example critiquing approaches is that they deviate too much from the currently used interaction paradigms (i.e., regular faceted navigation). We argue that an approach that lies closer to what users use on a daily basis will be more successful when deployed in practice. Therefore, our focus is to have an approach that introduces minimal changes to the classical faceted navigation interface present in current Web shops, while supporting approximate search and explicit user preferences. The only user interface adaption we make is that users can specify importance weights for the selected facets, which can be achieved in many different, non-intrusive ways.

7.3 Approximate Product Search

Before diving into the details of the framework, we first discuss facets in more detail and give an overview of all the framework components. Product information in Web shops is usually stored in the form of tabular data, consisting of key-value pairs, where the key denotes the *property* to which the value belongs. Each distinct combination of a key and value occurring in the product catalog is considered to be a facet. For example, a key-value pair, or facet, of a product might be `['Color', 'Black']`. Later on, if there is no ambiguity, we use the term 'value' to refer to a facet, in order to shorten the explanations. There are various property types, which need to be

treated differently, both in the front-end as well as in the back-end. There are two main types of properties, namely qualitative and quantitative properties.

Qualitative properties are further discerned based on whether they allow one or multiple values per product. Typically these two types are presented differently to the user, either in the form of radio buttons, which only allow one selection, or a list of check boxes that allows multiple selections. An example of the first type of qualitative property is a qualitative Boolean property, e.g., the key-value pair `['WiFi', 'True']`. In contrast, a key-value pair like `['Supported Audio Formats', 'MP3, FLAC, AAC']` allows multiple values for a product. In this case, each element in the list of values should be treated as a separate facet. There is an important exception to how single-valued qualitative properties other than Boolean qualitative properties are treated. These are namely still treated as a multi-valued property in user interfaces. One example of this would be the 'Brand' property. The reason for this is that users may wish to include products from multiple brands in the query (e.g., in order to compare them). Therefore, we treat multiple selections of facets from a property like 'Brand' as a generalized disjunctive query. This is consistent with the interpretation that many major Web shops employ, such as Amazon.com and BestBuy.com. Quantitative properties have a single numerical value, where every distinct numerical value is considered to be a facet. Different from boolean properties, a user can select multiple values for a quantitative property by selecting a range of values. For example, the user could search for a product with a price between €100 and €200, which encompasses multiple facets.

Queries submitted to a faceted search engine consist of a selection of facets, from which a Boolean expression is constructed. Traditionally the 'Standard Boolean Model' is used to evaluate the expression, which means that a product is only included in the result set if the Boolean expression evaluates to true for the product. Our proposed algorithm replaces this strict model with an 'Extended Boolean Model' that computes a similarity score for each product, thus enabling approximate search. The constructed query expression is usually made disjunctive for facets belonging to the same property, and conjunctive for facets belonging to different properties. Large Web shops, such as Amazon.com, employ this principle, as it is considered to be intuitive to the users to form the query in this way. For example, choosing `['WiFi', 'True']` and `['Camera Flash Type', 'Single LED, Dual LED, Xenon']` would search for all products that have WiFi and a Camera Flash Type of either 'Single LED', 'Dual LED', or 'Xenon'.

7.3.1 Framework Overview

The proposed algorithm employs a similarity score between the query and each product in the catalog, which indicates how closely a product matches the query. In order to be able to compute this score, it is necessary to convert both products and query to vectors first. Note that each distinct numerical value of a quantitative property is treated as a separate facet, and is therefore assigned to an element in the vector representation.

Figure 7.1 illustrates all the framework processes involved to compute the product-query similarities. In this figure, rectangles represent processes and ovals represent input/output of a processes. Furthermore, darker-gray ovals are dependent on the query, while light-gray ovals are not. This means that the light-gray processes can be computed in advance to improve the computational time.

First, let us discuss the process of converting *product facets* to *product vectors* and *inverse document frequency vectors* (i.e., the first row of the figure). The facets associated with a product are first converted to a binary weight vector, which is called the *raw term frequency vector*. A weight of zero at index i indicates that the product does not contain the corresponding facet, whereas a weight of one means that it does. Afterwards, a *facet similarity function* is applied to each element in the raw term frequency vector that has a value of zero. This function converts the binary weights to weights that range from zero to one. This can be interpreted as to how similar the product is to the considered missing facet (based on all available product data). A different weighting approach is used for the various facets in the vector, depending on whether the facet corresponds to a qualitative or quantitative property. We will discuss this process in more detail in Section 7.3.2. The last step in this part of the flow is to convert the product vector to the classical inverse document frequency vectors (Jones, 1972).

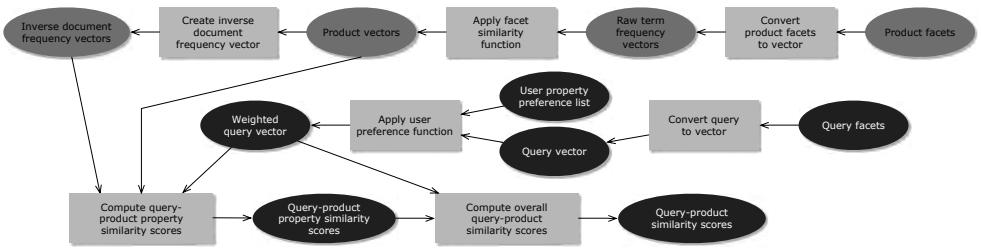

Figure 7.1: Overview of the framework.

On the second row of Figure 7.1 we see the processes that encompass the query flow, converting *query facets* to a *weighted query vector*. All of these processes are dependent on the query and have to be computed each time a query is submitted to the search engine. Every time a query is submitted to the search engine, the algorithm first converts it to a vector, which has the same dimensions as the previously created vectors for products. Similar to the raw term frequency vectors, a weight of zero at index i indicates that the query does not contain the corresponding facet, whereas a weight of one means that it does. A quantitative facet is contained in the query if its associated numerical value is within the query range for the associated property.

In addition to the query facets, the user also submits a *user property preference list* to the search engine. This preference list consists of all the involved properties in the facet selections and is ranked descending on the importance (determined by the user). The *user preference function* takes the query vector and the user preference list as input to determine the *weighted query vector*. The weight query vector is computed by multiplying each value in the query vector with a property importance weight. The details of this procedure are discussed in Section 7.3.3.

After both the products and the query have been converted to a vector, it is possible to compute the similarity between them. This is depicted on the third row of Figure 7.1. The similarity score is calculated using the *p-norm extended Boolean model* (Salton et al., 1983), and is a two-staged process. First, the similarity of each property in the query is computed for every product using the p-norm extended Boolean formula for disjunctive queries. Second, the overall similarity between the query and each product is computed. This step uses the similarity scores for each property from the previous step and weights them again according to their associated weights in the weighted query vector. Afterwards, these weights are plugged into the p-norm extended Boolean formula for conjunctive queries, which results in an overall similarity score. These steps are described in detail in Section 7.3.4. The end result of the process is a similarity score between zero and one for each product, which indicates how closely a product matches the query. Last, the products are sorted on their similarity scores in descending order and are presented to the user.

We move on now with the discussion of each of the processes depicted in Figure 7.1 in more detail. Table 7.1 provides an overview of the notations that are used throughout the rest of this chapter.

D	Product catalog
P	Properties
$P_{\text{qualitative}} \subseteq P$	Qualitative properties
$P_{\text{quantitative}} \subseteq P$	Quantitative properties
$F_p \subseteq F,\ p \in P$	Facets for property p
$f_p \in F_p$	Facet for property p
$F_d \subseteq F,\ d \in D$	Facets for product d
$f_d \in F_d$	Facet for product d
$F_{p,d} = F_p \cap F_d$	Property p facets for product d
$f_{p,d} \in F_{p,d}$	Property p facets for product d
$D_f,\ \forall d \in D_f : f \in F_d$	Products d associated with facet f
$q \subseteq F$	Query
$D_q \subseteq D$	Result set returned for query q

Table 7.1: Summary of used notations.

7.3.2 Product IDF Vectors

We start with the detailed explanation of the process of converting *Product Facets* to *Product Vectors* and *Inverse document frequency vectors*, i.e., the first row of Figure 7.1. The first step in converting product data to vectors consists of extracting the facets from the available tabular product data. Figure 7.2 depicts an example of the facet extraction process with three products. Each distinct key-value pair is converted to a facet and is assigned a vector index. The vector index of every facet corresponds to an element within the vector. Using the extracted facets for every product, we construct *raw term frequency vectors*. These vectors are straightforward binary vectors indicating if a facet is present or not. For example, for product 1 in Figure 7.2 the raw term frequency vector would be $(1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0)$.

After the raw term frequency vectors have been created, the *facet similarity function* is applied to each weight in the vector. This conversion is needed as binary weights are too strict for the purpose of approximate product search. Ideally we want the search engine to rank and not filter the products using the submitted query. For example, consider a user who is searching for a product with a price between €100 and €200. Now suppose that a product exists with a price of €99.95. As the price of the product does not fall within the specified range, the product does not match the query and thus gets a low similarity score or is even not included in the results at all. Intuitively this could be considered as an incorrect decision, as the price nearly matches the lower bound of the range, thus the product might be interesting to the user as well. Besides being too strict, using binary weights also gives another problem: any quantitative value that lie within the specified range is considered equally

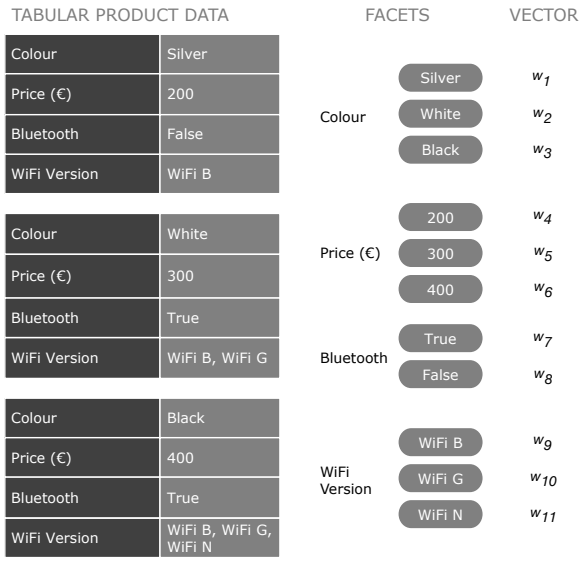

Figure 7.2: Three example products and their facets.

dissimilar from the query. This means that, for a query where the target price is €100, a product with a price of €500 would achieve the same similarity score as the product with a price of €99.95 (all other aspects being equal), which is clearly undesirable, as the latter product should be considered as being more similar to the query.

The same problems also arise when searching for products with a qualitative property. If a product does not contain the same qualitative facets as the query, it gets excluded. Furthermore, qualitative facets that are not included in the query are treated as equally dissimilar from those in the query. This does not always make sense, as some facets are conceptually quite similar, whereas others are not. For example, consider some of the audio file formats that exist, such as 'MP3', 'Ogg Vorbis', and 'WAV'. Whereas 'WAV' is an uncompressed and lossless format, both 'MP3' and 'Ogg Vorbis' are examples of a compressed and lossy format. Thus it can be argued that the latter two audio formats are more similar to each other than 'WAV' and either one of these formats.

The facet similarity function aims to address the issues outlined above by converting the weights in the raw term frequency vector to a weight in the range of zero to one, which can be interpreted as the degree of similarity between a product and each facet. The result of applying the function on the raw term frequency vector is

called the *product vector*. Depending on whether a facet belongs to a quantitative or quantitative property, a different approach is used to compute its weight in the product vector. This similarity function is defined as:

$$\text{facetSim}(f_p, d) = \begin{cases} 0 & \text{if } F_{p,d} = \emptyset \\ 1 - \texttt{min}_{f_{p,d} \in F_{p,d}} \dfrac{|f_p - f_{p,d}|}{\max(F_p) - \texttt{min}(F_p)} & \text{if } p \in P_{\text{quantitative}} \\ \max\limits_{f_{p,d} \in F_{p,d}} \dfrac{|D_{f_p} \cap D_{f_{p,d}}|}{|D_{f_{p,d}}|} & \text{if } p \in P_{\text{qualitative}} \end{cases} \tag{7.1}$$

Evidently, if a product does not have any of facets associated with the property p, the similarity score remains zero, as it is not possible to compute the degree of similarity in that case. For the first example product from Figure 7.2 we obtain the following product vector: $(1, 0, 0, 1, \frac{1}{2}, 0, 0, 1, 1, \frac{2}{3}, \frac{1}{3})$.

The score for quantitative properties is based on the distance between the numerical value of the facet f_p and the numerical value of the product. Note that, although $F_{p,d}$ is defined as a set, there is often only one element in the set for quantitative properties, as a product usually does not have multiple numerical values for the same property. The distance between the numerical values is normalized by dividing it by the range of the values for the property, thus obtaining a relative dissimilarity.

The facet similarity score for qualitative properties computes a similarity score between the facet f_p and each facet of property p that is also associated with product d, i.e., each facet $f_{p,d}$ in $F_{p,d}$. The similarity score is based on the fraction of products with facet $f_{p,d}$ that are also associated with facet f_p. After computing all the similarity scores, the highest score is selected and is used as the weight for facet f_p in the product vector. The rationale behind this score is that facets that occur frequently together within product data are likely to be conceptually similar.

For instance, consider the different versions of the WiFi standard, such as 'WiFi B', 'WiFi G', and 'WiFi N'. Each consecutive version of the standard enhances the previous version and is backwards compatible, thus products adhering to a particular version also adhere to the previous versions. Suppose that the user is searching for a product with 'WiFi N'. Then products which do not have 'WiFi N', but do have 'WiFi G', are a better alternative than products that only have 'WiFi B', as the versions are conceptually more similar. For example, suppose that there are 20, 10, and 5 products associated with 'WiFi B', 'WiFi G', and 'WiFi N', respectively, and each product with 'WiFi N' also has 'WiFi B' and 'WiFi G'. Then the similarity

score for the facet 'WiFi N' would be $\frac{5}{20}$ $(= \frac{1}{4})$ for products with only 'WiFi B', and $\frac{5}{10} (= \frac{1}{2})$ for products with 'WiFi G'.

After the product vectors have been created, the *inverse document frequency vectors* are computed (Salton and Buckley, 1988). Inverse document frequency vectors are useful for faceted search, as it provides the means to weight facets in the query, based on how often they occur within the product catalog. Products that have the more uncommon facets in the query will therefore receive a higher score than products which do not have those facets. For example, if a user searches for a product that has Bluetooth and is dust resistant, it is more likely that products with dust resistance are more important to the user than products that only have Bluetooth, as this facet occurs less frequently in the catalog.

Even though the idea of using the inverse document frequency can be useful in our context, we propose some adaptations to make it fit better. We define the inverse document frequency of item i (a facet) as following:

$$\mathrm{idf}(i) = \frac{\log\left(N / \sum_{n=1}^{N} w_{n,i}\right)}{\log N} \tag{7.2}$$

where $w_{n,i}$ corresponds to the value for item i for product n and N is the number of products in the catalog.

First, the p-norm extended Boolean model that is used by the framework requires weights to be in the range of zero to one. However, this is currently not the case, as weights can be any positive number. Therefore, the weights are normalized through dividing by $\log N$.

In addition, the term n in the original idf formula is exchanged for a slightly different term. Rather than being equal to the number of products that are associated with a facet, the new term is defined as the sum of facet similarities of all products. In other words, it is a summation of the facet weights in all product vectors. This is done in order to prevent a bias towards facets associated with few products, i.e., the size of the set D_f is quite small, while many products do have a high facet similarity score for such a facet. In this way, we obtain inverse document frequency weights for these facets that better reflect how often they occur within the catalog, as facet similarities are now taken into account. The inverse document frequency vector for

the first example product vector from Figure 7.2 is:

$$\left(\frac{\log 3/1}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/1\frac{1}{2}}{\log 3}, \frac{\log 3/2}{\log 3}, \frac{\log 3/1\frac{1}{2}}{\log 3}, \frac{\log 3/2}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/3}{\log 3}, \frac{\log 3/2\frac{2}{3}}{\log 3}, \frac{\log 3/1\frac{5}{6}}{\log 3}\right) \quad (7.3)$$

As it becomes clear from the previous example, a facet weight is equal to zero when all the products in the catalog have a score of one for the facet in their product vector, whereas a facet weight is equal to one when only a single product has a non-zero score for the facet in the product vector. This is logical, because including a facet in the query that every product has does not provide any extra information to the search engine. Conversely, when only one product is associated with a facet, it provides the maximum amount of information gain to the search engine.

7.3.3 Weighted Query Vectors

The process of converting a query to a vector is very similar to the process used to convert a product to a vector. Similar to product data, we extract facets from a submitted query. There is a small difference in how quantitative key-value pairs are handled though. Rather than having a single numerical value, queries have a numerical range as value for a quantitative property. Figure 7.3 illustrates how this process works with an example. The dark-gray facets are not included in the query, whereas the light-gray facets are included. For example, the price range $[200, 325]$ is mapped to the facets '200' and '300', but not to '400', as that value is out of range. The obtained query vector consists of binary weights, where a weight of zero means that the corresponding facet is not included in the query, and a weight of one means that it is included. For example, the query vector for the query in Figure 7.3 is defined as $(0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1)$.

With the (binary) query vectors, every facet is regarded as being equally important to the user. However, it can be argued that this is often not the case, as the user has both hard and soft requirements for a desired product. For example, a user searching for a smartphone under €200 with '*Android*' as Operating System and '*Samsung*' as brand, might also consider Android smartphones from other brands if they are much cheaper than similar phones from Samsung. Therefore, the weights in the query vector should be adjusted according to the user preferences, in such a way that it becomes possible to make a distinction between hard and soft requirements.

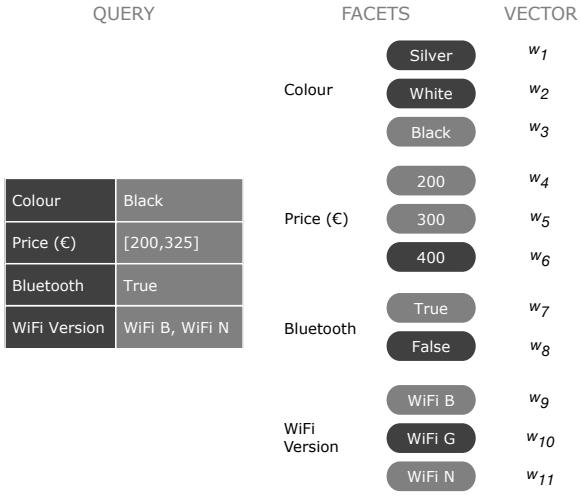

Figure 7.3: A query and the corresponding facets.

Assigning correct weights to the facets can partially be accomplished by using the inverse document frequency vector, as previously discussed. However, as these weights are solely based on the product data available in the catalog, the result is a vector that is the same for every user. It is therefore unsuitable for taking the user preferences into account, as not every user who submits a query has the same requirements. That is why instead we propose to extend the query by including a ranked list of properties, the *user property preference list*, which ranks the properties in descending order of importance to the user.

Every property p that occurs within the query, i.e., there is at least one facet f_p in query q, is included in the list. The reason for aggregating the preferences on a property level rather than on an individual facet level is twofold. First, queries submitted to a faceted search engine are typically quite large in size and therefore rather complex. Having to manage a list of preferences should not be too obtrusive for the user during the querying process. This is why the list should ideally contain only a few elements, even when the query contains many facets. Aggregating on a property level ensures that the preference list stays concise and manageable while the user gradually refines his query. Second, the query itself is also structured in a similar way, as it is disjunctive for facets belonging to the same property, and conjunctive between properties. This suggests that facets belonging to the same property are of equal relative importance to the user, whereas facets belonging to different properties

are not. This usually reflects the user intentions better than a fully conjunctive or disjunctive query. We therefore structure the user preferences in the same way.

The equation for converting a facet weight from the query vector to a weight that takes user preferences into account is given by:

$$\text{weight}(w_i, p) = \frac{1}{\text{rank}(p)} \cdot w_i \tag{7.4}$$

where w_i is the facet weight in the query vector, p is the property to which the facet belongs, and `rank(`p`)` is a function that returns the rank of property p in the user property preference list. The old weight w_i in the query vector is multiplied with the simplest form of Zipf's law, which was first proposed in (Zipf, 1935), in order to obtain a new weight that takes the user preferences into account. Suppose that the user preference list of the user, who submitted the example query in Figure 7.3, is equal to `['Color', 'Bluetooth', 'Price', 'WiFi Version']`, then applying the user preference function to the query vector yields the weighted query vector $\left(0, 0, 1, \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{2}, 0, \frac{1}{4}, 0, \frac{1}{4}\right)$.

7.3.4 Query-Product Similarity Score

Now that both the products and the query are converted to vectors, we compute a similarity score between each product and the query. The similarity score is calculated using the p-norm extended Boolean model. The framework first computes the similarity scores for each property, followed by the overall similarity score.

The standard Boolean model is the usual method for querying faceted product data. Each submitted query is converted to a Boolean expression and only products that completely match the expression are returned. As is evident from the example product data in Figure 7.2, none of the products match the query. However, there are some products that nearly match the query, but have the wrong color or their price is too high. Ideally, these trade-off possibilities should be presented to the user without the user having to make his query less specific first.

In addition, the strictness of the model also does not allow for user preferences to be taken into account. The user might have both soft and hard requirements for desired facets, but the standard Boolean model only returns product that completely match the query. It is therefore not possible to indicate that a particular term in the query is more important than another, as they are inherently all equally important.

Another deficiency of the standard Boolean model is the implicit lack of ranking within the result set. As each product in the result set matches the query, it is not

possible to distinguish between them. This means that a product, which has both 'WiFi B' and 'WiFi N', is not ranked higher than a product with only 'WiFi B' or 'WiFi N' for the example query in Figure 7.3, while, using generalized disjunction, it could be argued that it would preferable to do so.

It is clear that the standard Boolean model is not suitable for approximate product search, which means that an alternative model has to be used. One of the alternatives is the vector space model (Salton et al., 1975). Although this model does allow term weighting, the query is interpreted as a fully disjunctive query. This is obviously not ideal for faceted product search, as usually queries consist of conjunctive and disjunctive terms.

Therefore, the p-norm extended Boolean model (Salton et al., 1983) is used by the framework, which offers both the advantages of the vector space model and allows for conjunctive queries. It essentially softens the strict *AND* and *OR* Boolean logic operators of the standard Boolean model by calculating a distance between the desired values, i.e., the values in the query, and the values within a product vector. This distance is based on the p-norm distance, a generalization of the Euclidean and Manhattan distance measures, which includes a parameter p to vary the degree of strictness of the operator. In our approach, we use a value of two for the p parameter, which is a well-known and frequently used norm called the Euclidean norm. The results in (Salton et al., 1983) indicate that a value between two and five is optimal, whereas larger values decrease the performance improvements over the standard Boolean model. For disjunctive queries, the similarity for an n-dimensional vector using the p-norm extended Boolean model is given by:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \sqrt[p]{\frac{\mathbf{q}_1^p \cdot \mathbf{d}_1^p + \mathbf{q}_2^p \cdot \mathbf{d}_2^p + \cdots + \mathbf{q}_n^p \cdot \mathbf{d}_n^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{7.5}$$

where p is the p-norm parameter and q is the query vector and d is the document vector. The similarity for a conjunctive query and an n-dimensional vector is given by:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = 1 - \sqrt[p]{\frac{\mathbf{q}_1^p \cdot (1 - \mathbf{d}_1)^p + \mathbf{q}_2^p \cdot (1 - \mathbf{d}_2)^p + \cdots + \mathbf{q}_n^p \cdot (1 - \mathbf{d}_n)^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{7.6}$$

where p is the p-norm parameter and q is the query vector and d is the document vector.

The first step in calculating the query-product similarity scores is calculating the similarity score per property. As it is assumed that queries between facets from the same property are disjunctive, Equation (7.5) is used for this step. Because the score

is calculated for each property p, a sub-vector $\mathbf{d}$ is created from the product vector including only facets associated with property p. The same process is applied to the weighted query vector and the inverse document frequency vector, resulting in sub-vectors $\mathbf{q}$ and $\mathbf{idf}$, respectively. Last, the similarity score between $\mathbf{d}$ and $\mathbf{q}$ is computed, resulting in a score between zero and one.

Note that the weights for each facet in the query vector $\mathbf{q}$ are all equal, as they belong to the same property and the preference weighting is performed on a property-level. However, it can be argued that the facets should be weighted, as some facets are more unique than others, thus a product having that facet should be regarded as being more similar to the query. The reason for this distinction is that the user included an uncommon facet in the query, which could indicate that it is an important facet to him. For example, suppose that a query contains the facets 'WiFi G' and 'WiFi N' associated with the property 'WiFi Version'. As the versions are backwards compatible, there are relatively many products with 'WiFi G' compared to products with 'WiFi N'. In this situation it is more likely that the user would prefer products which have 'WiFi N' rather than products with only 'WiFi G'. Therefore, the query vector weights are multiplied by the inverse document frequency weights, in order to assign different weights to the facets in the query. The equation is therefore given by:

$$\text{propSim}(\mathbf{q^i}, \mathbf{d^i}) = \sqrt[p]{\frac{(idf_1^i \cdot q_1^i)^p \cdot {d_1^i}^p + \cdots + (idf_n^i \cdot {q^i}_n)^p \cdot {d_n^i}^p}{(idf_1^i \cdot q_1^i)^p + \cdots + (idf_n^i \cdot q_n^i)^p}} \tag{7.7}$$

where p is the p-norm parameter and q_j^i, idf_j^i, and d_j^i are the query, the inverse document frequency, and the product vector values for property i and vector index j (the vector corresponding only to property i), respectively.

Note that the query weights in the denominator are also multiplied by the inverse document frequency. This is done in order to keep the score normalized to a maximum value of one. Otherwise, even when the document would have all the facets in the query, i.e., $\mathbf{d}$ consists only of values of one, the resulting score might not be equal to one, which is counter-intuitive. In this case, the numerator can only be equal to the denominator, thus resulting in a score of one, if the inverse document frequency vector also consists only of values of one, which would mean that all the facets may only occur once in the product catalog. We therefore normalize the score by also multiplying the query vector weights in the denominator with the inverse document frequency weights. Consider the example weighted query vector

$$\left(0, 0, 1, \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{2}, 0, \frac{1}{4}, 0, \frac{1}{4}\right),$$

and the first example product vector

$$\left(1, 0, 0, 1, \frac{1}{2}, 0, 0, 1, 1, \frac{2}{3}, \frac{1}{3}\right),$$

and the inverse document frequency vector in Equation (7.3). After decomposing each vector into the sub-vectors for each property, as previously discussed, the property similarity scores are computed as follows:

$$\text{propSim}(\mathbf{q_p}, \mathbf{d_p}) \approx 0.90 \tag{7.8}$$

$$\text{propSim}(\mathbf{q_w}, \mathbf{d_w}) \approx 0.33 \tag{7.9}$$

$$\text{propSim}(\mathbf{q_c}, \mathbf{d_c}) = \text{propSim}(\mathbf{q_b}, \mathbf{d_b}) = 0 \tag{7.10}$$

where c, p, b, and w are the properties 'Color', 'Price (€)', 'Bluetooth', and 'WiFi Version', respectively.

Looking at the property similarity scores, we can observe that the product has nothing in common with the query regarding the properties 'Color' and 'Bluetooth'. This is due to the fact that the product only has non-zero scores in its product vector for facets that are not included in the query. In other words, it has a wrong value for 'Color', namely 'Silver' instead of 'Black', and it does not have Bluetooth. Furthermore, because there are no other products associated with 'Silver' or 'False' that are also associated with 'Black' or 'True', the weights of the product for 'Black' and 'True' remain zero in the product vector, as there is no facet similarity. In contrast, the product achieves a high score for the property 'Price (€)', as its value (200) is within the range of the query ([200,325]).

While the product does have one out of two of the facets associated with the property 'WiFi Version' in the query, it only gets a score of 0.33. This is due to the fact that it only has a score of one in its product vector for the most common 'WiFi Version', namely 'WiFi B'. As all products have a score of one for that facet, its weight in the inverse document frequency vector is zero, as it conveys no information regarding the product that the user wants to buy. However, because some of the products that have 'WiFi B' also have 'WiFi N' associated with them, the product has a weight of 1/3 associated with 'WiFi N' in its product vector. Therefore, the property similarity score of this product is not equal to zero, but equal to 1/3.

After the property similarity scores have been computed, the *overall similarity score* can be computed. As the query is assumed to be conjunctive between prop-

erties, the framework uses Equation (7.6) for this step. Furthermore, it uses the property similarity scores for each property as the weights for the product.

Because the facets are now aggregated on a property level, it is also necessary to aggregate the weights in the vectors. First, the property similarity scores are combined to form the aggregated product vector. Using the decomposed property similarities for the first product vector in Figure 7.2, we determine this vector to be $(0, 0.8993, 0, 0.33)$. Then, the similarities for each property have to be weighted, such that products with a high similarity score for important properties are promoted in the search results. As previously discussed, a list of properties in the query, ranked in decreasing order of importance, is obtained from the user. Using this list, Equation (7.4) is applied to each weight in the query vector in order to transform it to the weighted query vector. Subsequently, the highest weight per property in the weighted query vector is collected and together they form the weighted aggregated query vector. For the example weighted query vector, this vector is $\left(1, \frac{1}{3}, \frac{1}{2}, \frac{1}{4}\right)$. After the aggregated query and product vectors have been obtained, the overall similarity between them can be computed as follows:

$$\text{overallSim}(\mathbf{q}, \mathbf{d}) = \sqrt[p]{\frac{\mathbf{q}_1^p \cdot \mathbf{d}_1^p + \mathbf{q}_2^p \cdot \mathbf{d}_2^p + \cdots + \mathbf{q}_n^p \cdot \mathbf{d}_n^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{7.11}$$

where p is the p-norm parameter and q is the query vector and d is the document vector. For the first example product vector $(0, 0.8993, 0, 0.33)$ and the weighted example query vector $\left(1, \frac{1}{3}, \frac{1}{2}, \frac{1}{4}\right)$ the overall similarity score is computed as following:

$$\text{overallSim}(\mathbf{q}, \mathbf{d}) = \sqrt{\frac{1^2 \cdot 0^2 + \frac{1}{3}^2 \cdot 0.8993^2 + 0^2 \cdot \frac{1}{2}^2 + \frac{1}{4}^2 \cdot 0.33^2}{1^2 + \frac{1}{3}^2 + \frac{1}{2}^2 + \frac{1}{4}^2}} \approx 0.0522 \tag{7.12}$$

As we can see, a high similarity score of one property is offset by low scores for the more important properties, thus the product receives a low overall similarity score of 0.05.

The final scores for the the three products from Figure 7.2 is 0.05, 0.15, and 0.79, respectively. Note that, as none of the products has achieved a perfect overall similarity score, no products would have been returned if the standard Boolean model was used. This would result in the user having to adjust his query in order to find trade-off possibilities, whereas this is not needed in the proposed framework. Clearly the third product matches the query better than the other products, due to having perfect similarity scores for three of the four properties in the query. It therefore matches most of the user's requirements, apart from the price, which is too high. However, the user has indicated previously that the price is a relatively

soft requirement for him. The user preference function has taken this into account by assigning a weight of 1/3 to the price property, which helps boost the overall similarity score of the third product. This assists the user in finding the best trade-off possibilities, given his preferences, if there is no product that completely matches his query.

7.4 Evaluation

In this section we first discuss the experimental setup and then the obtained results. Before we start with the discussion of our experimental setup, we need to extend our notations to accommodate our explanations, shown in Table 7.2.

p_t	Selected property for user action t
F_t	Selected facet for user action t
$U_a \subseteq P$,	User property preference list (actual)
$U_g \subseteq P$,	User property preference list (generated)
$r_p(f)$	Rank of displayed facet f from property p
$r_p(p)$	Rank of displayed property p
$r_u(p, U)$	Rank of property p in user property pref. list U
$r_q(f)$	Rank of facet f in candidate facet list
$r_q(d)$	Rank of product d in result set D_q for query q
$d_u \in D$	Target product for user u
X_u	User effort for preference reordering
X_p	User effort for property scanning
X_f	User effort for facet scanning
T	Number of user actions per simulation
t, $t \leq T$	Current number of clicks
N	Number of top results scanned
α	Prob. picking correct property p
β	Prob. picking incorrect property p
α_f	Prob. picking correct facet f
β_f	Prob. picking incorrect facet f
γ_f	Facet importance factor for facet f
ρ	Quantitative facet range uncertainty

Table 7.2: Additional notations used in the evaluation procedure.

7.4.1 Experimental setup

For our experiments we use the interaction model proposed in (Pu and Faltings, 2004), where the user is presented a list of facets and iteratively updates the query by

selecting one or more facets. In our setup, a search session is defined as an interactive process of updating the query, until the user has no desire to further refine the query. More specifically, first, the user decides whether the query should be updated any further, which is based on the number of user actions executed so far, and the total number of user actions that he is willing to perform. If the user wants to update the query, the user considers the generated property preference list first, a ranked list of properties in the query in descending order of importance. The user compares the generated user preference list with the actual property preference list, which reflects his or her preferences regarding the properties associated with the target product. If the user spots an inconsistency within the ranking of the properties in actual and generated user preference list, the generated list is updated by dragging the first incorrectly ranked property to the desired position. Afterwards, the preference list is submitted to the search engine, which subsequently updates the result set by taking the new user preferences into account.

If the user cannot perform preference reordering, either because the ranking between the actual and generated user preference list is already consistent, or the employed algorithm does not take user preferences into account, the query is updated by selecting an additional facet instead. It is assumed that the user scans the list of displayed facets in a linear order, from the top to the bottom. The ordering of the list is based on the expert-based ordering employed by (Tweakers.net, 2016), and does not change during the search session. Every time a facet is examined, the user first tries to identify it as a facet belonging to the target product. Rather than being able to identify each facet of the target product with full accuracy, the user has only a chance of α to correctly identify it. Conversely, the user has a chance of β to falsely identify a facet as belonging to the target product. For our evaluation, we use $\alpha = 0.9$ and $\beta = 0.1$.

In order to prevent a bias towards examining facets from properties which have many facets, we distribute the α and β chances over all the facets per property. Otherwise, a facet from a property with many facets, such as 'Color', would be included more often in the query than a facet from a Boolean qualitative property, such as 'Bluetooth'. Therefore, the chance to consider selecting a facet is α_f or β_f, depending on whether the facet is associated with the product.

After identifying a facet as belonging to the target product, either correctly or incorrectly, the user will consider whether he wants to actually select the facet for inclusion in the query. After the user has made a selection, the updated query and generated preference list are submitted to the search engine and the number of user

actions is incremented. The search engine subsequently computes new similarity scores between all the products and the query, and returns a new ranked result set. This process continues until the user is no longer willing to perform more user actions, at which point the simulated search session is concluded.

We vary the total number of clicks per search session using 5, 10, 20, and 30 clicks. Furthermore, the target product, which the user is looking for, is also varied. The combination of algorithm, target product, and number of clicks are used as inputs to the evaluation framework. Each combination of inputs for simulating a search session is defined as one experiment. Note that each experiment is executed fifty times, in order to reduce the stochasticity, which is inherent in the drill-down model that simulates a user.

For evaluating the performance of the algorithms, a real-life dataset was obtained from (Tweakers.net, 2016), which is a commercial Web site with a large community consisting of more than half a million Dutch and Belgian technology enthusiasts. We have chosen to focus the research on faceted search within a product catalog consisting entirely of mobile phones. The reason for this is that products in this category differ greatly in functionality and appearance, introducing heterogeneity in the data. The data of 794 different mobile phones was obtained from Tweakers.net using a combination of tools and scripts.

In addition to our proposed algorithm, the performance of several other algorithms is measured and compared to each other. The first algorithm, which is used as a baseline, employs a simple similarity function for each product, based on the number of facets in the query that the product matches. The p-norm extended Boolean model (Salton et al., 1983) forms the basis of our proposed algorithm. It therefore makes sense to compare the results of the proposed algorithm with those of the regular p-norm extended Boolean model. A variation of the proposed algorithm that does not take user preferences into account is the third algorithm against which we compare the performance of our approach.

Last, for our experiments we needed to model the user behavior when interacting with the faceted search engine. For this purpose, based our assumptions on the ones previously outlined in (Koren et al., 2008b; Liberman and Lempel, 2014): rationality, omnisciency, linearity, and practicality. For more details on these assumptions we refer the reader to (Koren et al., 2008b; Liberman and Lempel, 2014). In our approach we relax the omnisciency and linearity assumptions by introducing some stochasticity to the model when selecting facets to include in the query, which is implemented differently for qualitative and quantitative facets.

User Property Preference Reordering

Under the practicality assumption, a user continues refining his query until he has invested a certain amount of effort into formulating the query. The effort is measured in the number of user actions t, which can also be defined as clicks, that a user has performed. The total amount of effort that a user is willing to put into formulating the query is a constant denoted by T, which is a fixed number for each experiment. The user will continue refining the query q while $t < T$, or until it is not possible to drill-down any further.

When user preferences are involved, the user can perform a drill-down in two ways: either he reorders the user property preference list U_g, or he selects additional facets to include in the query. Note that the user is not allowed to deselect facets, as it is interesting to measure how well the fuzzy search algorithms can cope with search errors. It is therefore required to model the user's preferences regarding the properties of the target product d_u, so that the evaluation procedure knows when to reorder the properties in U_g. As the Rationality and omnisciency assumptions dictate that the user is able to identify properties and facets associated with d_u, the actual user property preference list U_a consists of all the properties associated with d_u, thus $\forall p \in U_a : F_p \cap F_{d_u} \neq \emptyset$. The ordering is based on the degree of uniqueness of the facets associated with each property and the target product. In our experiments, we multiply the inverse document frequency weights with the product vector weights and take the average for each property in order to obtain a ranking of importance. By ranking the property importance scores in descending order, we obtain the actual user property preference list U_a. This is a fixed list of preferences that is defined in advance for each distinct target product d_u.

During each search session, another list of property preferences is generated, denoted by U_g. This list is updated when the user selects additional facets to include in the query q, and only consists of properties which are associated with at least one facet in the specified query q.

Qualitative Facet Drill-Down

Whenever the user cannot perform a reordering of the property preferences, either because the ordering is correct or the algorithm does not take user preferences into account, the user examines the displayed facets. The order in which the facets are displayed is first defined by the order of the properties in P. All the facets associated with a property p, denoted by the list F_p, are grouped and displayed beneath each other. Within the list of facets associated with a property, denoted by F_p, the indi-

vidual facets f are ranked in descending order, according to the number of products that are associated with each facet. In other words, the list is ranked according to the size of D_f for each facet f.

The omniscience assumption states that the user is able to correctly identify all the facets that belong to the target product d_u, i.e., the user is able to identify the members of the set F_{d_u}. In our setup, rather than being able to identify each member of F_{d_u} with full accuracy, the user has only a chance of α to correctly identify it now. In addition, he has a chance of β to falsely identify a facet as being a member of F_{d_u}, with $\alpha + \beta = 1$. We fix the values for these parameters to 0.9 and 0.1 for α and β, respectively. In order to prevent a bias towards examining facets from properties which have many facets, we distribute the α and β chances over all the facets per property. Otherwise, a facet from a property with many facets, such as 'Color', would be included more often in the query than a facet from a Boolean qualitative property, such as 'Bluetooth'. We therefore define the following for each property p in the list of properties P:

$$\alpha_f = \frac{\alpha}{|F_p \cap F_{d_u}|}, \text{ where } f \in F_p \tag{7.13}$$

$$\beta_f = \frac{\beta}{|F_p \setminus F_{d_u}|}, \text{ where } f \in F_p \tag{7.14}$$

$$\alpha + \beta = 1$$

where α is the probability of selecting a correct facet ($f \in F_{d_u}$) and β is the probability of selecting an incorrect facet ($f \notin F_{d_u}$).

After a user has identified a facet as being a member of F_{d_u}, either correctly or incorrectly, he will consider whether he wants to actually select the facet for inclusion in the query. Depending on whether the facet f is a member of F_{d_u} or not, this chance is different and is given by:

$$\gamma_f = \begin{cases} 1 - \frac{r_q^I(f)-1}{|F_{d_u} \setminus q|-1} & \text{if } f \in F_{d_u} (\alpha \text{ case}) \\ 1 & \text{if } f \notin F_{d_u} (\beta \text{ case}) \end{cases} \tag{7.15}$$

where $r_q(f)$ is a function that returns the rank of the facet f in the list of candidate facets. If the user deems the facet to be important enough, he selects it. The query then gets updated and submitted to the search engine, after which the updated result set is displayed. If he decides not to select the facet, the user will continue examining the lists of facets until he finds a facet to select.

Quantitative Facet Drill-Down

Drilling down into quantitative facets is mostly similar to the qualitative facets procedure, with a few exceptions. The difference lies in how the facets are treated, which stems from the difference in presentation to the user. Rather than being presented with a list of facets to choose from, quantitative facets are displayed as a range slider. When the user has decided to select the quantitative facet, he still needs to determine the range to select. The range selection is performed by first selecting a mean, denoted by μ_f^i, where i indicates how often the facet has been previously selected. The function for determining the mean μ_f^i is given by:

$$\mu_f^i = \begin{cases} f_{d_u} & \alpha \\ \mu_f^{i-1} & \text{if } i > 0 \\ x \sim U([\,\mathtt{min}_{d \in D}(f_d),\, \max\limits_{d \in D}(f_d)\,]) & \beta \\ x \sim U([\,\mathtt{min}_{d \in D}(f_d),\, \max\limits_{d \in D}(f_d)\,]) & \text{if } f_{d_u} = \text{null} \end{cases} \tag{7.16}$$

In the first case, the facet has not been selected before, and the target product has a value f_{d_u} associated with this facet. Therefore, that value is selected as the mean for the range. If the facet has been previously selected, i.e., $i > 0$, the existing mean is reused, which is denoted by μ_f^{i-1}. Last, it is also possible to make a mistake and select an incorrect mean. This is either the case for β, where the target product has a value for this facet, but the user did not correctly identify it, or the target product does not have a value for this facet. In both these cases the mean is given by $x \sim U$, which denotes a randomly selected value from the range of values in the catalog, following a random distribution. Note that even when a wrong mean is selected, there still exists the possibility that the facet value of the product, is still within the selected range.

After determining the mean, the standard deviation (σ_f^q) of the products in the result set having a value for this property is calculated, which is subsequently used to define the lower and upper boundaries of the range. Therefore, the range selection function is given by:

$$[\,f_{\min},\, f_{\max}\,] = \left[\,\mu_f^i - \sigma_f^q,\, \mu_f^i + \sigma_f^q\,\right] \tag{7.17}$$

where i denotes how often the facet has been selected previously. This makes it possible to select a quantitative facet multiple times, each time narrowing the range.

Selecting a quantitative facet multiple times could potentially make the range converge to a single value, which is undesired, as it is assumed that the user does not know the exact value. Therefore, the possibility to select the facet again and further narrow down the range is only given when the following condition holds:

$$f_{max} - f_{min} \geq \rho \cdot \sigma_f \tag{7.18}$$

where σ_f is the standard deviation of the facet values of all the products in the product catalog D, and ρ is a factor that models the uncertainty regarding the exact value of the target product. When the span of the range, denoted by $f_{max} - f_{min}$, is less than the standard deviation multiplied by ρ, the user does not narrow the range any further. Empirical analysis revealed that realistic ranges are obtained within our simulation model when $\rho = 2$, although the effect of setting a different value has not been studied extensively.

7.4.2 Results from the simulated experiments

Various performance measures are collected during the experiments. In order to assess the performance of the algorithms, it is useful to gather some measures related to the position of the target product. First, we define the 'last position' of the product as the outcome of the $r_q(d_u)$ function after T user actions. This function returns the rank of the target product d_u in the result set D_q. Second, we define the 'average position' as the average outcome of the $r_q(d_u)$ function computed after each user action. Third, we use the 'first iteration in top-N' metric to measure how quickly an algorithm manages to promote the target product to the first top N results. Similarly, the 'any iteration in top-N percentage' metric measures the percentage of sessions in which the target product was promoted at least once to the top N results. Last, the 'success percentage' indicates in how many search sessions the target product was in the top-N results after T user actions. In addition to collecting performance measures related to the position of the target product in the result set, we also collect measures regarding the user effort that is required to formulate the query. The user effort involved can be split up into various components, related to the various tasks a user performs while formulating the query. The first task we define is the reordering of the user property preference list. The other two tasks are related to scanning the list of displayed properties P (property scan effort) and the list of displayed facets F (value scan effort). Last, the computation time of the algorithm required for computing the ranking of the products in the result set is also measured. The time is

measured in milliseconds and consists only of the time to compute the ranking. The time needed to simulate the user's behavior is therefore not included in this measure.

As previously outlined, all combinations of an algorithm, a target product, and a total number of clicks were executed, with each experiment being conducted fifty times. This means that a total of 635,200 search sessions were simulated (4 algorithms $\times$ 794 products $\times$ 4 total clicks $\times$ 50 repetitions). Because each search session consists of 16.25 ($= \frac{5+10+20+30}{4}$) user actions on average, it results in a total of 10,322,000 simulated user actions. Due to these vast numbers, the experiments were run in parallel on a cluster of server nodes (Amazon.com, 2017b), in order to speed up the evaluation.

Table 7.3 shows an overview of the obtained performance measures for the four algorithms and for $T = \{5, 10, 20\}$. We do not discuss the case of $T = 30$ for the sake brevity, however, we can report that the relative results are the same as for $T = 20$. Only the absolute differences between the algorithms became larger.

At first sight, from Table 7.3 we can see that for $T = 5$, the algorithms perform fairly similar. This is because the queries are relatively homogeneous when only five clicks are used, making the effect of the ranking algorithms less clearly visible. However, we do make some interesting observations. First, we notice that the results for the 'simple rank' algorithm are relatively good. It outperforms all other algorithms for the last position metric ($p < 0.00001$). With respect to the success percentage metric, it outperforms our approach with user preference ranking but performs worse than our approach without user preference ranking (all differences are significant with $p < 0.00001$). After analyzing the reason for this behavior we discovered that the main disadvantage of the 'simple rank' algorithm, i.e., not distinguishing between conjunctive and disjunctive aspect in a query, is not visible when limiting the number of clicks to 5. The reason for this is that the facets are scanned linearly and the first five facets belong to properties for which almost all products only have one value. These facets are 'Price', 'Brand', 'Operating System (OS)', 'OS version', and 'Color'. As a result, in almost all simulated sessions, a conjunctive query was performed, therefore not allowing algorithms that support disjunctive queries to distinguish themselves.

Second, we notice that the p-norm approach has a low success percentage, with a significant difference w.r.t. our approach without user preference ranking ($p < 0.00001$). The reason for this is that the p-norm algorithm is not able to cope with the quantitative properties properly. However, we do observe that the p-norm achieves a higher average for the success percentage than our approach that includes

	Simple Rank	P-norm	Our Algorithm without Preferences	Our Algorithm with Preferences
For $T = 5$:				
Last Position Mean	84.25	95.05	103.66	102.14
Avg. Position Mean	152.94	159.58	141.77	145.95
Any Iteration Top-N Percentage	44.74%	44.53%	55.47%	44.03%
First Iteration Top-N Mean	3.34	3.29	3.01	2.89
Success Percentage	40.23%	37.01%	44.86%	36.56%
Preference Reorder Count Mean	–	–	–	1.1
Property Scan Effort Mean	0.0986	0.0988	0.0987	0.0706
Value Scan Effort Mean	0.2652	0.2635	0.2625	0.1844
Computation Time Mean (ms)	1380	2232	1207	1199
For $T = 10$:				
Last Position Mean	75.11	93.01	122.84	79.25
Avg. Position Mean	113.10	124.44	126.75	114.05
Any Iteration Top-N Percentage	68.78%	68.99%	71.96%	66.87%
First Iteration Top-N Mean	4.76	4.71	3.97	4.47
Success Percentage	57.73%	53.30%	51.85%	53.78%
Preference Reorder Count Mean	–	–	–	2.8
Property Scan Effort Mean	0.1287	0.1288	0.1286	0.0822
Value Scan Effort Mean	0.3521	0.3526	0.3522	0.2219
Computation Time Mean (ms)	2645	4692	2496	2442
For $T = 20$:				
Last Position Mean	97.68	124.35	169.00	79.15
Avg. Position Mean	100.24	118.06	137.75	96.32
Any Iteration Top-N Percentage	76.58%	76.19%	76.69%	78.51%
First Iteration Top-N Mean	5.74	5.62	4.54	5.97
Success Percentage	55.77%	48.52%	44.36%	59.95%
Preference Reorder Count Mean	–	–	–	5.8
Property Scan Effort Mean	0.1647	0.1648	0.1648	0.1045
Value Scan Effort Mean	0.4407	0.4413	0.4404	0.2836
Computation Time Mean (ms)	5259	10100	5315	5088

Table 7.3: Experimental results for $T = \{5, 10, 20\}$, $N = 20$, and $\alpha = 0.9$.

user preference ranking. The reason for this difference (even though not significant with $p = 0.39608$) might be explained due to the fact that the user in our approach spends 1.1 out of 5 clicks on average on facet re-ordering. This results in less clicks used for the actual query refinement.

However, our approach with user preference ranking does have the best performance on property and value scan effort (with $p < 0.00001$). The reason for this is similar to why the success percentage is relatively low: some user effort is moved from scanning facets to re-ordering facets. We argue that this effort can be more useful for search engines than selecting new facets because it adds more information

as to how to rank the products, without making the actual query more complex (i.e., introducing new facets). We also see that our approach (both with and without user preference ranking) outperforms the other algorithms on the 'first iteration top-N' metric (with $p < 0.00001$). We can conclude from this that our approach significantly improves the ability to promote the target product while the user is searching.

When we look at the results for $T = 10$, we notice that the performance of all algorithm improves (e.g., the success percentage increases for all approaches). This is as expected because the queries are more refined when 10 clicks are used instead of 5. We further observe that the 'simple rank' algorithm again outperforms the other approaches on the last position metric ($p < 0.001$) and that our algorithms are again the best w.r.t. the first iteration top-N metric. Another observation that we make is that property and value scan efforts have increased for all approaches. This is due to the fact that the more clicks there are, the more scanning occurs because already selected facets are skipped in subsequent iterations, as facets cannot be selected twice. However, our approach with user preference ranking still outperforms the others (with $p < 0.00001$). The reason for this is that also the number of clicks spent on property scanning is increased (i.e., 2.8 out of 10).

For $T = 20$ we make an interesting observation: compared to $T = 10$, all approaches except our algorithm with user preference ranking perform worse for the last position metric. The reason for this is the increased probability of including incorrect facets in the query, which obviously negatively impact the last position metric. We conclude from this that property re-ordering by the user can combat this issue by placing properties lower on the preference list for which the user is unsure if the target product has them. Furthermore, we see that for the first iteration top-N metric again our approach without the user preference ranking is the best and that similar to previous findings, the scanning effort for properties and values has increased. However, our approach, without the user preference ranking, does not perform so well for the last position metric. After performing error analysis, we found out that this is caused by the fact that our similarity for qualitative values, which relies on co-occurrence values, does not always help. What we do observe is that our approach that includes user preference ranking seems to resolve this issue, as it scores the best for the last position metric with an average last position of 79.15.

7.4.3 Results using an experiment with users

Besides the extensive experiments performed using simulation, we also performed an experiment with real users. We had a total of 27 users who participated in the

experiment, consisting of 17 males and 10 females. There were 19 users that were between 20 and 30 years old, 6 users that were between 31 and 40 years old, and 2 users that was between 40 and 50 years old. These users were mostly students and colleagues from our university and other universities and there was no financial reimbursement for the participation in the experiment.

The implementation of the Webshop that implements the algorithm proposed in this paper can be found online*. Figure 7.4 shows a screenshot of this system. We have also implemented the 'standard' Webshop†, i.e., one that has no special features other than those commonly encountered on the Web.

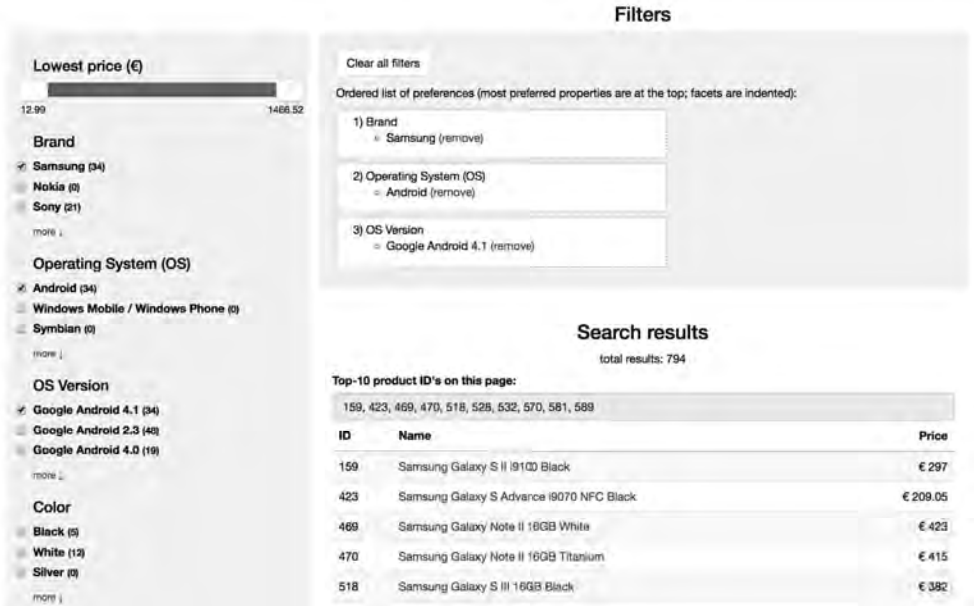

Figure 7.4: Screenshot of our approach that implements our approach.

The experiment consisted of 10 small tasks in which each user was given 7 product features and their corresponding preference scores, indicating how important the product features are. The goal was to find the 10 products that best match the given product features, i.e., to find the 10 products that have the highest score according to the preference list. The assigned preference scores were computed using the Zipfian distribution and multiplied with 100 and rounded to the nearest integer (e.g, $1/1 \times 100$, $1/2 \times 100$, $1/3 \times 100$, etc.). The users were performing the tasks

*`http://approx-prod-search.eur.dvic.io`
†`http://std-prod-search.eur.dvic.io`

using only the 'standard' system. The scores for our system were simply computing by selecting the facets in the order of how they were presented in the task, and answering with the 10 products that were returned by the system. Using this scoring system, our approach achieved an average score of 13.945 while the standard system achieved an average score of 9.170, which we found to be statistically significant using a paired two-sample t-test (with $p = 0.04149$). Table 7.4 shows the behavior of an average user who participated in the experiment. We can see that most users chose to filter based on numeric facets (such as the price). This also shows us that the users selected on average at least 18 facets to come up with their answers. Using our system this is does not need to be more than 7, i.e., the number of presented product features.

Event type	Total occurrence	Average per user occurrence
Numeric facet change	2142	107.10
Toggle collapsed	389	19.45
List facet select	372	18.60
List facet deselect	135	6.75
Boolean facet change	129	6.45
Numeric facet remove	56	2.8
Boolean facet remove	23	1.15

Table 7.4: Event counts for the user experiment.

7.5 Conclusion

In this chapter, we propose a novel framework specifically geared towards approximate faceted search within a product catalog of a Web shop. It explores the concept of facet similarity functions for both quantitative and qualitative facets, in order to express the degree of similarity between various facets. We propose adaptations to the classical p-norm extended Boolean model to account for the domain-specific characteristics of faceted search in an e-commerce environment. Our approach allows the user to specify a property preference list and takes this into account when weighting the query terms. In order to assess the performance of the proposed algorithm, it is compared with a simple baseline algorithm, based on the fraction of facets in the product data that match those in the query, as well as the p-norm extended Boolean

model. It is shown that in many cases the proposed algorithm compares favorably to the other algorithms w.r.t. many different metrics (especially for large number of clicks). Our approach also scores best with regards to the percentage of search sessions in which the target product has appeared in the top-N results at least once. Also, it is better able to exploit the quantitative facets in order to promote the target product in the result set. Besides the experiments we performed using evaluation we have also performed an experiment with real users. From this experiment we can conclude that our approach finds more relevant products using less effort. In future work we would like to explore the usage of ontologies that contain information about similarities between qualitative entities for the purpose of computing a similarity. Another enhancement to the framework could be the inclusion of the negation operator for defining queries.

Chapter 8

Conclusions

In this dissertation, we have addressed techniques for intelligent Web Product Information Systems and devised algorithms to make product search easier for consumers. In order to answer the problem statement of this dissertation – i.e., how to design Web product search engines that automatically aggregate product information and users are able to perform effective and efficient queries – we have discussed topics such as product classification, entity resolution, ontology population, and faceted product search. In the next section, we summarize the most important findings and provide additional directions for future research.

8.1 Concluding Remarks

First, we have proposed a hierarchical product classification framework, which is a multi-level classification system used to classify a product description to one of the category leaves of a product taxonomy. From the obtained results we can draw several conclusions. First, we have found the k-Nearest Neighbor algorithm to be unsuitable as an independent classifier. Besides the computational cost, the accuracy is too low to be useful in practical applications. Furthermore, the results from the evaluation suggest that with our product data set, the Naïve Bayes classifier performs better than Support Vector Machines, obtaining an average accuracy of 76.80% for product classification. When considering the properties of a product description, we found that the features description provides better predictors for the top levels, but that the title provides better predictors for the lower levels, except for the last level, where the features description gives again better results.

We have also proposed a scalable approach for multi-source entity resolution using various blocking schemes. Our approach consists of three main components: (1) a blocking scheme, (2) a product similarity function, and (3) a clustering procedure. Various blocking schemes, which operate on the title, the description, or both, are evaluated with both a perfect similarity function, as well as an imperfect, but well-performing one (van Bezu et al., 2015). The most important finding with regards to the blocking evaluation is that experimental setups that evaluate blocking schemes with only a perfect matching function are most likely not sufficient, i.e., one should also consider imperfect matching functions. Furthermore, the results suggest that for our similarity function and clustering procedure, which together achieve an F_1-measure of 0.525 when using all pairs, blocking methods that compute very few pairs achieve a higher performance that ones that have a high pair completeness. In particular, we find that the blocking scheme that extract model word triples from the title gives the best trade-off between effectiveness and efficiency on our test data set, achieving an F_1-measure of 0.537.

Once product descriptions have been classified and duplicates have been removed, we can instantiate product descriptions into an ontology. For this purpose, we have proposed a framework capable of semi-automatic ontology population of product information from Web shops. The performance of the framework is compared to the performance of a baseline approach, which merely uses lexical matching for the Classification and Property Matching processes. Evaluation results show us that our framework performs considerably better than the baseline approach for the Property Matching process, with an F_1-measure of 95.07%, due to the use of both lexical matching and pattern matching. The evaluation of the Value Instantiation process was performed using a graph-based approach, comparing it to a manually instantiated ontology. The results from this evaluation indicate that our proposed algorithm is also promising in this regard.

As part of the ontology management process, we have proposed SCHEMA, an algorithm capable of performing automated mapping between heterogeneous product taxonomies in e-commerce. The performance of our algorithm was tested on three real-life datasets and compared with the performance of two other proposed approaches from literature. The evaluation demonstrates that SCHEMA achieves a considerably higher average recall than the other algorithms, with a relatively small loss of precision.

In order to improve the usability of searching for products, we proposed an approach that automatically orders facets such that the user finds its desired product

with the least amount of effort. We evaluate our solution using an extensive set of simulation experiments, comparing it to three other approaches. While analyzing the user effort, especially in terms of the number of clicks, we can conclude that our approach gives a better performance than the benchmark methods and in some cases even beats the manually curated 'Expert-Based' approach. These results have been also validated using a study involving real users. In addition, the relatively low computational time makes it suitable for use in real-world scenarios, making our findings also relevant to industry.

In addition, we proposed a novel framework specifically geared towards approximate faceted search within a product catalog of a Web shop. In order to assess the performance of the proposed algorithm, it was compared with a baseline approach that used the fraction of facets in the product data that match those in the query, as well as the p-norm extended Boolean model. The results from the experiments show that, in many cases, the proposed algorithm compares favorably to the other algorithms with respect to various important metrics. Our approach also scores best with regards to the percentage of search sessions in which the target product has appeared at least once in the top-N results. Furthermore, our algorithm is better able to exploit the information from the quantitative facets in order to promote the target product in the result set. A user-based study confirms the results obtained from the experimental simulation studies.

8.2 Future work

For the hierarchical product classification framework, we want to further investigate the interaction effects between the classification algorithms and the feature selection methods. One approach would be to research different combination strategies at different levels of the product category taxonomy. Another approach would be to use ensemble techniques to combine classifiers, i.e., a classifier on the title, a classifier on the features description, and a classifier on both the title and description. A third option would be to use ensemble techniques to combine and evaluate the category mapping algorithm with the previously presented text classifiers.

In the ontology population approach, we are interested in improving the Value Instantiation process by adding new value extraction rules and by creating new property assertions between individuals in the ontology that further specify the relationship between them. For example, by formally defining in the ontology that 'Windows XP' is the successor to 'Windows 2000', the framework could also instantiate a property

assertion for ‘Windows XP’ when it encounters a raw product value such as ‘Windows 2000 or later’.

With respect to our proposed taxonomy mapping approach, we would like to investigate the use of part-of-speech tagging. As a noun is often more important for concept similarity than an adjective, it makes sense to distinguish between them and treat them accordingly. Another possibility is to combine the hierarchical category structure with product information because the data in product descriptions could hold valuable information for the taxonomy mapping. Additionally, this work could support the implementation of a system that is capable of autonomously matching products and product taxonomies from different sources.

In regards to the proposed facet ordering algorithm, we would like to investigate in more depth how much effort it takes for users to get used to a system that changes the order of the facets each time the query changes. The results of such a study might indicate that a hybrid system, where some part of the facets remain static for the sake simplicity, works better for some users.

For the proposed product search approach, we would like to explore the possibility of incorporating ontologies that contain information about relationships between qualitative entities in the computation of the final similarity between a query and a set of products. Another enhancement to the framework could be the inclusion of the negation operator for defining faceted-based queries, which is currently not implemented.

Bibliography

S. Aanen, L. Nederstigt, **D. Vandic**, and F. Frasincar. SCHEMA - An Algorithm for Automated Product Taxonomy Mapping in E-commerce. In *9th Extended Semantic Web Conference (ESWC 2012)*, volume 7295, pages 300–314. Springer, 2012.

B. Adida, M. Birbec, S. McCarron, and I. Herman. RDFa Core 1.1 W3C Recommendation 07 June 2012. Technical report, W3C, 2012. `http://bit.ly/18BvYJL`.

Amazon.com. Largest Online Retailer in the United States. `http://www.amazon.com`, 2017a.

Amazon.com. Amazon Web Services. `http://aws.amazon.com`, 2017b.

M. Arenas, B. Cuenca Grau, E. Evgeny, S. Marciuska, and D. Zheleznyakov. Towards Semantic Faceted Search. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)*, pages 219–220. ACM, 2014a.

M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, and E. Jimenez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)*, pages 123–126. ACM, 2014b.

D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and Ontology Matching with COMA++. In *ACM SIGMOD International Conference on Management of Data 2005 (SIGMOD 2005)*, pages 906–908. ACM, 2005.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search.* Addison-Wesley Professional, 2011.

S. Banerjee and T. Pedersen. An Adapted Lesk Algorithm for Word Sense Disambiguation using WordNet. In *3rd International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2002)*, pages 136–145, 2002.

R. Baxter, P. Christen, and T. Churches. A Comparison of Fast Blocking Methods for Record Linkage. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 3, pages 25–27. ACM, 2003.

M. Baziz, M. Boughanem, and N. Aussenac-Gilles. Conceptual indexing based on document content representation. In *Context: nature, impact, and role*, pages 171–186. Springer, 2005.

O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. *The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *The Scientific American*, 284(5):34–43, 2001.

D. Berrueta and L. Polo. MUO — An Ontology to Represent Units of Measurement in RDF, 2009. `http://goo.gl/fDsuk`.

BestBuy.com. Retailer and Online Retailer of Consumer Electronics. `http://www.bestbuy.com`, 2017.

C. M. Bishop. *Pattern Recognition And Machine Learning*. Springer-Verlag, 2007.

C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML). *World Wide Web Journal*, 2(4):27–66, 1997.

L. Breiman. Technical Note: Some Properties of Splitting Criteria. *Machine Learning*, 24(1):41–47, 1996.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. CRC press, 1984.

R. Burke. Interactive Critiquing for Catalog Navigation in E-commerce. *Artificial Intelligence Review*, 18(3-4):245–267, 2002.

S. Castano, A. Ferrara, and S. Montanelli. H-MATCH: An Algorithm for Dynamically Matching Ontologies in Peer-Based Systems. In *1st VLDB Int. Workshop on Semantic Web and Databases (SWDB 2003)*, pages 231–250, 2003.

D. Celjuska and M. Vargas-Vera. Ontosophie: A Semi-automatic System for Ontology Population from Text. Technical report, KMi Institute, 2004. `http://bit.ly/13EegA4`.

CEO. Consumer Electronics Ontology — An Ontology for Consumer Electronics Products and Services. `http://bit.ly/12Ir4bG`, 2017.

L. Ceriani and P. Verme. The Origins of the Gini Index: Extracts from Variabilità e Mutabilità (1912) by Corrado Gini. *The Journal of Economic Inequality*, 10(3): 421–443, 2012.

S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 446–455. Morgan Kaufmann Publishers Inc., 1997.

C. Chang, M. Kayed, R. Girgis, and K. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18 (10):1411–1428, 2006.

P. Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9): 1537–1555, 2012.

K. W. Church and P. Hanks. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

M. Ciaramita, A. Gangemi, E. Ratsch, J. Saric, and I. Rojas. Unsupervised Learning of Semantic Relations Between Concepts of a Molecular Biology Ontology. In *19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 659–664. Morgan Kaufmann Publishers Inc., 2005.

V. Crescenzi, G. Mecca, P. Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

B. Cuenca Grau, E. Kharlamov, and D. Zheleznyakov. Faceted Search over RDF-based Knowledge Graphs. *Journal of Web Semantics*, 37, 2016.

S. D'Alessio, K. Murray, R. Schiaffino, and A. Kershenbaum. The Effect of Using Hierarchical Classifiers in Text Categorization. In *Proceedings of 6th International Conference Recherche d'Information Assistee par Ordinateur*, pages 302–313, 2000.

F. J. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, 1964.

S. Damodaran. B2B Integration over the Internet with XML: RosettaNet Successes and Challenges. In *13th International World Wide Web Conference (WWW 2004)*, pages 188–195. ACM, 2004.

D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic Faceted Search for Discovery-Driven Analysis. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 3–12. ACM, ACM, 2008.

T. De Vries, H. Ke, S. Chawla, and P. Christen. Robust Record Linkage Blocking Using Suffix Arrays. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 305–314. ACM, 2009.

Y. Ding, M. Korotkiy, B. Omelayenko, V. Kartseva, V. Zykov, M. Klein, E. Schulten, and D. Fensel. GoldenBullet: Automated Classification of Product Data in E-commerce. In *Proceedings of the 5th International Conference on Business Information Systems*, 2002.

DMOZ. Open Directory Project. `http://www.dmoz.org/`, 2017.

H.-H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. In *Web, Web-Services, and Database Systems (NODe 2002)*, volume 2593 of *LNCS*, pages 221–237. Springer, 2002.

X. Dong, A. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 85–96. ACM, 2005.

S. Dumais and H. Chen. Hierarchical classification of Web content. In *Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval*, pages 256–263. ACM, 2000.

eCl@ss e.V. eCl@ss — Classification and Product Description, 2017. `http://bit.ly/11bB2zw`.

M. Ehrig and S. Staab. QOM - Quick Ontology Mapping. In *International Semantic Web Conference 2004 (ISWC 2004)*, pages 683–697, 2004.

M. Ehrig and Y. Sure. Ontology Mapping — An Integrated Approach. In *1st European Semantic Web Symposium. The Semantic Web: Research and Applications (ESWS 2004)*, volume 3053 of *LNCS*, pages 76–91. Springer, 2004.

A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

J. C. Fagan. Usability Studies of Faceted Browsing: A Literature Review. *Information Technology and Libraries*, 29(2):58–66, 2013.

C. Fellbaum. *WordNet: An Electronic Lexical Database*. The MIT press, 1998.

D. Fensel, Y. Ding, B. Omelayenko, E. Schulten, G. Botquin, M. Brown, and A. Flett. Product Data Integration in B2B E-Commerce. *IEEE Intelligent Systems*, 16(4): 54–59, 2001.

J. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006.

W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the CSS2 visual box model. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1313. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Gene Ontology Consortium and others. Gene Ontology: Tool for the Unification of Biology. *Nature genetics*, 25(1):25–29, 2000.

J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: An Algorithm And An Implementation of Semantic Matching. In *Dagstuhl Seminar Proceedings of Semantic Interoperability and Integration 2005*, 2005.

Google. Guava - Google Core Libraries, 2017. `http://bit.ly/11Y4ww0`.

Google Knowledge Graph. Inside Search. `http://www.google.com/insidesearch/features/search/knowledge.html`, 2017.

Google, Microsoft, Yahoo and Yandex. schema.org, 2017. `http://schema.org`.

L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava, et al. Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th International Conference on Very Large Data Bases*, volume 1, pages 491–500, 2001.

Y. Guo, J. Hu, and Y. Peng. Research on CBR System Based on Data Mining. *Applied Soft Computing*, 11(8):5006–5014, 2011.

Y. Guo, J. Hu, and Y. Peng. A CBR System for Injection Mould Design Based on Ontology: A Case Study. *Computer-Aided Design*, 44(6):496–508, 2012.

Y. Guo, Y. Peng, and J. Hu. Research on High Creative Application of Case-based Reasoning System on Engineering Design. *Computers in Industry*, 64(1):90–113, 2013.

S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. Dom-based content extraction of HTML documents. In *Proceedings of the 12th International Conference on World Wide Web*, pages 207–214. ACM, 2003.

E.-H. Han, G. Karypis, and V. Kumar. Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 53–65. Springer-Verlag, 2001.

M. Hearst. Design Recommendations for Hierarchical Faceted Search Interfaces. In *29th Annual International Conference on Research & Development on Information Retrieval (ACM SIGIR 2006)*, pages 1–5. ACM, 2006.

M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K.-P. Yee. Finding the Flow in Web Site Search. *Communications of the ACM*, 45(9):42–49, 2002.

M. Hepp. Products and Services Ontologies: a Methodology for Deriving OWL Ontologies from Industrial Categorization Standards. *International Journal on Semantic Web and Information Systems*, 2(1):72–99, 2006.

M. Hepp. GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In *16th International Conference on Knowledge Engineering (EKAW 2008)*, volume 5268 of *LNCS*, pages 329–346. Springer, 2008.

M. Hepp. eClassOWL, 2010a. `http://www.heppnetz.de/projects/eclassowl`.

M. Hepp. unspscOWL, 2010b. `http://www.heppnetz.de/projects/unspscowl`.

M. Hepp. The Web of Data for E-Commerce: Schema.org and GoodRelations for Researchers and Practitioners. In *International Conference on Web Engineering (ICWE 2015)*, pages 723–727. Springer, 2015.

M. Hepp, J. Leukel, and V. Schmitz. A Quantitative Analysis of Product Categorization Standards: Content, Coverage, and Maintenance of eCl@ss, UNSPSC, eOTD, and the RosettaNet Technical Dictionary. *Knowledge and Information Systems*, 13 (1):77–114, Dec. 2006.

J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *22nd Annual International Conference on Research and Development in Information Retrieval (ACM SIGIR 1999)*, pages 230–237. ACM, 1999.

M. A. Hernández and S. J. Stolfo. Real-World Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

P. Hitzler and K. Janowicz. Linked Data, Big Data, and the 4th Paradigm. *Semantic Web*, 4(3):233–235, 2013.

P. Holmans, E. K. Green, J. S. Pahwa, M. A. Ferreira, S. M. Purcell, P. Sklar, et al. Gene Ontology Analysis of GWA Study Data Sets Provides Insights Into the Biology of Bipolar Disorder. *American journal of human genetics*, 85(1):13–24, 2009.

W. Holzinger, B. Krüpl, and M. Herzog. Using Ontologies for Extracting Product Features from Web Pages. In *5th International Semantic Web Conference (ISWC 2006)*, pages 286–299. Springer, 2006.

J. B. Horrigan. Online Shopping. *Pew Internet & American Life Project Report*, 36, 2008.

J. Huang, J. Lu, and C. X. Ling. Comparing Naive Bayes, Decision Trees, and SVM with AUC and Accuracy. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 553–556. IEEE, 2003.

T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the European Conference on Machine Learning*, pages 137–142. Springer-Verlag, 1998.

K. S. Jones. A Statistical Interpretation of Term Specificity and its Application in Retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

Kieskeurig.nl. Major Dutch price comparison engine with detailed product descriptions. `http://www.kieskeurig.nl`, 2017.

A. Kilgarriff and J. Rosenzweig. Framework and Results for English SENSEVAL. *Computers and the Humanities*, 34(1–2):15–48, 2000. doi: 10.1023/A:1002693207386.

H.-J. Kim, Y. Zhu, W. Kim, and T. Sun. Dynamic Faceted Navigation in Decision Making using Semantic Web Technology. *Decision Support Systems*, 61(1):59–68, 2014a.

H.-J. Kim, Y. Zhu, W. Kim, and T. Sun. Dynamic Faceted Navigation in Decision Making using Semantic Web Technology. *Decision Support Systems*, 61:59–68, 2014b.

H.-s. Kim and D. Lee. HARRA: Fast Iterative Hashed Record Linkage for Large-scale Data Collections. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 525–536. ACM, 2010.

L. Kolb, A. Thor, and E. Rahm. Multi-pass Sorted Neighborhood Blocking With Mapreduce. *Computer Science-Research and Development*, 27(1):45–63, 2012.

D. Koller and M. Sahami. Hierarchically Classifying Documents Using Very Few Words. In *Proceedings of the 14th International Conference on Machine Learning*, pages 170–178. Morgan Kaufmann Publishers Inc., 1997. ISBN 1-55860-486-3.

H. Kopcke and E. Rahm. Frameworks for Entity Matching: A Comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

J. Koren, Y. Zhang, and X. Liu. Personalized Interactive Faceted Search. In *Proceedings of the 17th International Conference on World Wide Web*, pages 477–486. ACM, ACM New York, NY, USA, 2008a.

J. Koren, Y. Zhang, and X. Liu. Personalized Interactive Faceted Search. In *17th International Conference on World Wide Web (WWW 2008)*, pages 477–486. ACM, 2008b.

N. Koudas, S. Sarawagi, and D. Srivastava. Record Linkage: Similarity Measures and Algorithms. In *Proceedings Of The 2006 ACM SIGMOD International Conference on Management of Data*, pages 802–803. ACM, 2006.

B. Krüpl, M. Herzog, and W. Gatterbauer. Using visual cues for extraction of tabular data from arbitrary HTML documents. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, pages 1000–1001. ACM, 2005.

B. Kules, R. Capra, M. Banta, and T. Sierra. What Do Exploratory Searchers Look at in a Faceted Search Interface? In *9th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2009)*, pages 313–322. ACM, 2009.

Y.-H. Lee, P. J.-H. Hu, T.-H. Cheng, and Y.-F. Hsieh. A Cost-sensitive Technique for Positive-Example Learning Supporting Content-Based Product Recommendations in B-to-C E-commerce. *Decision Support Systems*, 53(1):245–256, 2012.

M. Lesk. Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In *5th Annual International Conference on Systems Documentation (SIGDOC 1986)*, pages 24–26. ACM, 1986.

V. I. Levenshtein. Binary Codes Capable of Correction Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training Algorithms for Linear Text Classifiers. In *Proceedings of the 19th Annual International Conference on Research and Development in Information Retrieval*, pages 298–306. ACM, 1996.

B. Li, A. Ghose, and P. G. Ipeirotis. Towards a Theory Model for Product Search. In *20th International Conference on World Wide Web (WWW 2011)*, pages 327–336. ACM Press, 2011.

T. Li, S. Zhu, and M. Ogihara. Hierarchical Document Classification Using Automatically Generated Hierarchy. *Journal of Intelligent Information Systems*, 29(2): 211–230, 2007.

S. Liberman and R. Lempel. Approximately Optimal Facet Value Selection. *Science of Computer Programming*, 94:18–31, 2014.

C.-F. Lin and S.-D. Wang. Fuzzy Support Vector Machines. *IEEE Transactions on Neural Networks*, 13(2):464–471, 2002.

C.-H. Lin and H. Chen. An Automatic Indexing and Neural Network Approach to Concept Retrieval and Classification of Multilingual (Chinese-English) Documents.

IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 26 (1):75–88, Feb 1996.

J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.

Martin Hepp. Extensions for GoodRelations for Specific Industries. `http://bit.ly/1gl6ZM0`, 2013.

Martin Hepp. The OPDM project. `http://bit.ly/1b4YUHB`, 2017a.

Martin Hepp. The Product Types Ontology: High-precision identifiers for product types based on Wikipedia. `http://bit.ly/GEbALr`, 2017b.

B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

A. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng. Improving Text Classification by Shrinkage in a Hierarchy of Classes. In *Proceedings of the 15th International Conference on Machine Learning*, pages 359–367. Morgan Kaufmann, 1998.

A. McCallum, K. Nigam, and L. H. Ungar. Efficient Clustering of High-dimensional Data Sets With Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178. ACM, 2000.

L. McDowell and M. Cafarella. Ontology-Driven, Unsupervised Instance Population. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):218–236, 2008.

G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, 2004.

S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *18th International Conference on Data Engineering (ICDE 2002)*, pages 117–128. IEEE, 2002.

X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

T. Mitchell. *Machine Learning*. McGraw Hill, 1996.

S. Mulpuru, V. Boutan, C. Johnson, S. Wu, and L. Naparstek. Forrester Research eCommerce Forecast, 2014 to 2019. `https://goo.gl/6b1fh3`, 2015.

F. F.-H. Nah. A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

Newegg.com Inc. Online Retailer in Consumer Electronics. `http://www.newegg.com`, 2017.

W. K. Ng, G. Yan, and E.-P. Lim. Heterogeneous Product Description in Electronic Commerce. *SIGecom Exchanges*, 1(1):7–13, 2000.

I. Niles and A. Pease. Towards a Standard Upper Ontology. In *International Conference on Formal Ontology in Information Systems 2001 (FOIS 2001)*. ACM, 2001.

N. F. Noy and M. A. Musen. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, 59(6): 983–1024, 2003.

N. Oza, J. Castle, and J. Stutz. Classification of Aeronautics System Health and Safety Documents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(6):670–680, Nov 2009.

G. Papadakis, E. Ioannou, T. Palpanas, C. Niederee, and W. Nejdl. A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2665–2682, 2013.

G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-Agnostic vs Schema-Based Configurations for Blocking Methods on Homogeneous Data. *Proceedings of the VLDB Endowment*, 9(4):312–323, 2015.

S. Park and W. Kim. Ontology Mapping between Heterogeneous Product Taxonomies in an Electronic Commerce Environment. *International Journal of Electronic Commerce*, 12(2):69–87, 2007.

C. Patel, K. Supekar, and Y. Lee. OntoGenie: Extracting Ontology Instances from WWW. In *Workshop on Human Language Technology for the Semantic Web and Web Services*, 2003. `http://bit.ly/10eUcWH`.

D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242. ACM, 2003.

M. F. Porter. An Algorithm for Suffix Stripping. *Readings in information retrieval*, pages 313–316, 1997.

P. Pu and L. Chen. Integrating Tradeoff Support In Product Search Tools For E-commerce Sites. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 269–278. ACM, ACM New York, NY, USA, 2005.

P. Pu and B. Faltings. Decision Tradeoff using Example-Critiquing and Constraint Programming. *Constraints*, 9(4):289–310, 2004.

E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.

L. E. Raileanu and K. Stoffel. Theoretical Comparison between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence*, 41 (1):77–93, 2004.

M. E. Ruiz and P. Srinivasan. Hierarchical Text Categorization Using Neural Networks. *Information Retrieval*, 5(1):87–118, 2002.

G. M. Sacco and Y. Tzitzikas. *Dynamic Taxonomies and Faceted Search*, volume 25. Springer, 2009a.

G. M. Sacco and Y. Tzitzikas. *Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience*, volume 25. Springer, 2009b.

G. Salton and C. Buckley. Term-weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(7):613–620, 1975.

G. Salton, E. A. Fox, and H. Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

M. Sasaki and K. Kita. Rule-Based Text Categorization Using Hierarchical Categories. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2827–2830, 1998.

G. Schadow and C. J. McDonald. UCUM — The Unified Code for Units of Measure. `http://unitsofmeasure.org`, 2010.

N. Shadbolt, W. Hall, and T. Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

C. E. Shannon. A Mathematical Theory of Communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

F. Shih and S.-S. Chen. Adaptive Document Block Segmentation and Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26 (5):797–802, Oct 1996.

P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics IV*, (3730):146–171, 2005.

S. R. Singh, H. A. Murthy, and T. A. Gonsalves. Feature Selection for Text Classification Based on Gini Coefficient of Inequality. In *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining (FSDM 2010)*, volume 10, pages 76–85, 2010.

V. Sinha and D. R. Karger. Magnet: Supporting Navigation in Semistructured Data Environments. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 97–106. ACM, ACM New York, NY, USA, 2005a.

V. Sinha and D. R. Karger. Magnet: Supporting Navigation in Semi-structured Data Environments. In *24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pages 97–106. ACM, 2005b.

M. Steinbach, G. Karypis, and V. Kumar. A Comparison of Document Clustering Techniques. 00 034, University of Minnesota, 2000.

A. Stolz and M. Hepp. Adaptive Faceted Search for Product Comparison on the Web of Data. In *International Conference on Web Engineering*, pages 420–429. Springer, 2015.

A. Sun and E. P. Lim. Hierarchical Text Classification and Evaluation. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 521–528. IEEE Computer Society, 2001.

C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *Proceedings of the VLDB Endowment*, 7(13):1529–1540, 2014.

A. Tengli, Y. Yang, and N. L. Ma. Learning table extraction from examples. In *Proceedings of the 20th International Conference on Computational Linguistics*, page 987. Association for Computational Linguistics, 2004.

K. Toutanova, F. Chen, K. Popat, and T. Hofmann. Text Classification in a Hierarchical Mixture Model for Small Training Sets. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 105–113. ACM, 2001.

D. Tunkelang. Faceted Search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–80, 2009.

Tweakers.net. Tweakers.net Pricewatch. `https://tweakers.net/pricewatch/`, 2016.

UNSPSC. United Nations Standard Products and Services Code, 2012. `http://bit.ly/13Ef5Ja`.

UNSPSC.org. United Nations Standard Products and Services Code, 2014. `http://www.unspsc.org`.

R. van Bezu, S. Borst, R. Rijkse, J. Verhagen, F. Frasincar, and **D. Vandic**. Multi-component Similarity Method for Web Product Duplicate Detection. In *30th Symposium On Applied Computing (SAC 2015)*. ACM, 2015. available at `http://goo.gl/82jHd5`.

D. Vandic, F. Frasincar, and F. Hogenboom. Scaling Pair-Wise Similarity-Based Algorithms in Tagging Spaces. In *12th International Conference on Web Engineering (ICWE 2012)*, pages 46–60, 2012a.

D. Vandic, J. W. J. van Dam, and F. Frasincar. Faceted Product Search Powered by the Semantic Web. *Decision Support Systems*, 53(3):425–437, 2012b.

D. Vandic, J. W. J. van Dam, and F. Frasincar. A Semantic-Based Approach for Searching and Browsing Tag Spaces. *Decision Support Systems*, 54(1):644–654, 2012c.

D. Vandic, F. Frasincar, and U. Kaymak. Facet Selection Algorithms for Web Product Search. In *22nd ACM International Conference on Information and Knowledge Management (CIKM 2013)*, pages 2327–2332. ACM, 2013a.

D. Vandic, F. Frasincar, and U. Kaymak. Facet Selection Algorithms for Web Product Search. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 2327–2332. ACM, ACM New York, NY, USA, 2013b.

D. Vandic, D. Nibbering, and F. Frasincar. A Case-Based Analysis of the Effect of Offline Media on Online Conversion Actions. In *22nd International World Wide Web Conference (WWW 2013), Companion Volume*, pages 125–126, 2013c.

D. Vandic, S. Aanen, F. Frasincar, and U. Kaymak. Dynamic Facet Ordering for Faceted Product Search Engines. *IEEE Transactions on Knowledge and Data Engineering*, 2017. to appear.

B. Vijayalakshmi, A. GauthamiLatha, D. Y. Srinivas, and K. Rajesh. Perspectives of Semantic Web in E-commerce. *International Journal of Computer Applications*, 25(10):52–56, 2011.

W3C OWL Working Group. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). Technical report, W3C, 2012. `http://bit.ly/c4CWDL`.

H. Wang, Q. Wei, and G. Chen. From Clicking to Consideration: A Business Intelligence Approach to Estimating Consumers' Consideration Probabilities. *Decision Support Systems*, 56(0):397–405, 2013.

K. Wang, S. Zhou, and S. C. Liew. Building Hierarchical Classifiers Using Class Proximity. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 363–374. Morgan Kaufmann, 1999.

T.-Y. Wang and H.-M. Chiang. Fuzzy Support Vector Machine for Multi-class Text Categorization. *Information Processing & Management*, 43(4):914–929, 2007.

A. S. Weigend, E. D. Wiener, and J. O. Pedersen. Exploiting Hierarchy in Text Categorization. *Information Retrieval*, 1(3):193–216, 1999.

S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity Resolution With Iterative Blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 219–232. ACM, 2009.

W. J. Wilbur and K. Sirotkin. The Automatic Identification of Stop Words. *Journal of information science*, 18(1):45–55, 1992.

Y. Yang. Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval. In *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval*, pages 13–22. Springer-Verlag New York, Inc., 1994.

Y. Yang. An Evaluation of Statistical Approaches to MEDLINE Indexing. In *Proceedings of the American Medical Informatics Association Annual Fall Symposium*, pages 358–362, 1996.

Y. Yang. An Evaluation of Statistical Approaches to Text Categorization. *Information retrieval*, 1(1-2):69–90, 1999.

Y. Yang and C. G. Chute. An Example-Based Mapping Method for Text Categorization and Retrieval. *ACM Transactions on Information Systems*, 12(3):252–277, 1994.

Y. Yang and X. Liu. A Re-examination of Text Categorization Methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49. ACM, 1999.

Y. Yang and J. O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420. Morgan Kaufmann Publishers Inc., 1997.

Y. C. Yang. Web User Behavioral Profiling for User Identification. *Decision Support Systems*, 49(3):261–271, 2010.

H. Yu, J. Yang, and J. Han. Classifying Large Data Sets Using SVM's with Hierarchical Clusters. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, pages 306–315. ACM, 2003.

Y. Yu, D. Hillman, B. Setio, and J. Heflin. A Case Study in Integrating Multiple E-commerce Standards via Semantic Web Technology. In *International Semantic Web Conference 2009 (ISWC 2009)*, pages 909–924, 2009.

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot topics in Cloud Computing*, volume 10, page 17, 2010.

G.-Q. Zhang, G.-Q. Zhang, Q.-F. Yang, S.-Q. Cheng, and T. Zhou. Evolution of the Internet and its Cores. *New Journal of Physics*, 10(12):123027, 2008.

B. Zheng, W. Zhang, and X. F. B. Feng. A Survey of Faceted Search. *Journal of Web engineering*, 12(1&2):041–064, 2013.

Y. Zhu, D. Jeon, W. Kim, J. Hong, M. Lee, Z. Wen, and Y. Cai. The Dynamic Generation of Refining Categories in Ontology-Based Search. In *Semantic Technology*, volume 7774 of *Lecture Notes in Computer Science*, pages 146–158. 2013.

G. K. Zipf. *The Psycho-biology of Language.* Houghton, Mifflin, 1935.

Summary in English

Over the last few years, online shopping has become very popular among consumers. However, this rapid growth of e-commerce has also introduced some issues. Users can get confused or are overwhelmed by the information they get presented while searching online for products. In an attempt to lighten this information overload burden on consumers, there are several product search engines that aggregate product descriptions and price information from the Web and allow the user to easily query this information. However, because it is difficult to understand all the different ways online shops represent their production information, most product search engines expect to receive the data from the participating Web shops in a custom format. In this thesis, we investigate how to design Web product search engines that automatically aggregate product information and allow users to perform effective and efficient queries on this data. We first focus on how to classify products into an existing taxonomy using only their textual descriptions. For this purpose, we propose a multi-level hierarchical classification algorithm that makes use of the structure of the taxonomy to derive the most optimal product description classifications. Next, we focus on the problem of finding duplicates among product descriptions in order to keep the managed data set clean. We devise an algorithm that can perform this tasks on large-scale, in a distributed environment. Because of the opportunities that Linked Data for e-commerce brings us, and the fact that we would like a system like the one we envision in this dissertation to be as open as possible, we also investigate how one can effectively populate ontologies from semi-structured product data using lexico-syntactic mappings and how to design an approach that automatically maps one product taxonomy into another using only the category names. Last, we perform two studies where we (1) investigate how we can reduce the consumer search effort by ranking the displayed facets on a per-query basis and (2) how we can improve fuzzy product search by allowing users to specify facet preference rankings.

Nederlandse Samenvatting (Summary in Dutch)

Het online kopen van producten is over de afgelopen paar jaar enorm in populariteit gestegen. Deze sterke toename in e-commerce heeft echter ook een aantal problemen voor de gebruikers met zich meegebracht. Vanwege het grote aanbod op het internet zien veel online bezoekers vaak door de bomen het bos niet meer. Naar aanleiding van deze ontwikkeling zijn er zoekmachines ontstaan die productinformatie van het web verzamelen en op één plek beschikbaar stellen voor de gebruikers. Deze zoekmachines vereisen echter dat de webshops hun gegevens aanleveren op een specifieke wijze. De reden hiervoor is dat niet iedere webshop de productinformatie online op dezelfde wijze representeert. In dit proefschrift onderzoeken wij hoe zoekmachines ontwikkeld kunnen worden die automatisch productinformatie verzamelen van het Web. Allereerst onderzoeken we hoe we producten kunnen classificeren op basis van hun tekstuele beschrijvingen. Dit heeft geleid tot het ontwikkelen van een meerdere niveaus hiërarchisch classificatie algoritme, beschreven in hoofdstuk 2. In hoofdstuk 3 ontwikkelen wij een algoritme voor het ontdubbelen van grote hoeveelheden online productbeschrijvingen. In hoofdstuk 4 onderzoeken wij hoe zoekmachines uit gestructureerde en geannoteerde data op het internet gegevens kunnen verwerken. Aansluitend hierop bekijken we in hoofdstuk 5 hoe twee verschillende producthiërarchieën samengevoegd kunnen worden, gebruikmakend van alleen de categorienamen in beide hiërarchieën. Hoofdstuk 6 beschrijft een onderzoek waarbij gebruikers een zoekmachine voor producten gebruiken dat de filters (producteigenschappen) sorteert op basis van de zoekopdracht, met het doel om gebruikers zo snel mogelijk het product te laten vinden waarnaar ze op zoek zijn. Afsluitend, in hoofdstuk 7, onderzoeken we de effectiviteit van een zoekmachine die, gegeven een zoekopdracht, productfilters niet strict toepast, maar op zoek is naar de *meest geschikte* product(en).

About the Author

Damir Vandić was born in Sarajevo, Bosnia and Herzegovina, on 30th of April 1987. He holds a cum laude B.Sc. degree and a cum laude M.Sc. degree in Economics and Informatics, obtained at Erasmus University Rotterdam, The Netherlands. His research interests cover areas such as machine learning, decision support systems, and Web information systems.

In October 2010, Damir obtained a NWO Mosaic scholarship and started his Ph.D. research under the auspices of the Erasmus Center for Business Intelligence (ECBI) at the Erasmus Research Institute of Management (ERIM), the Econometric Institute at the Erasmus School of Economics (ESE), and the Dutch Research School for Information and Knowledge Systems (SIKS). During his Ph.D. research, he went abroad to the United States, where he spent four months at Google as part of a research internship with the YouTube team in Mountain View, California.

Damir has published 27 peer-reviewed papers in the proceedings of prestigious international conferences, such as CAiSE, CIKM, DEXA, ESWC, and WWW. Additionally, Damir has published 8 articles in renowned journals such as Transactions on Knowledge and Data Engineering, Decision Support Systems, Expert Systems with Applications, and Journal of Web Engineering. He has also been active in the research community as a reviewer for journals such as Decision Support Systems, Expert Systems with Applications, and Information Systems.

ERIM Ph.D. Series Overview

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT (ERIM)

ERIM PH.D. SERIES RESEARCH IN MANAGEMENT

The ERIM PhD Series contains PhD dissertations in the field of Research in Management defended at Erasmus University Rotterdam and supervised by senior researchers affiliated to the `Erasmus Research Institute of Management (ERIM)`. All dissertations in the ERIM PhD Series are available in full text through the ERIM Electronic Series Portal: `http://hdl.handle.net/1765/1`. ERIM is the joint research institute of the Rotterdam School of Management (RSM) and the Erasmus School of Economics at the Erasmus University Rotterdam (EUR).

DISSERTATIONS LAST FIVE YEARS

Abbink, E., *Crew Management in Passenger Rail Transport*, Promoter(s): Prof.dr.L.G. Kroon & Prof.dr. A.P.M. Wagelmans, EPS-2014-325-LIS, `http://repub.eur.nl/pub/76927`

Acar, O., *Crowdsourcing for Innovation: Unpacking Motivational, Knowledge and Relational Mechanisms of Innovative Behavior in Crowdsourcing Platforms*, Promoter(s): Prof.dr.ir. J.C.M. van den Ende, EPS-2014-321-LIS, `http://repub.eur.nl/pub/76076`

Akin Ates, M., *Purchasing and Supply Management at the Purchase Category Level: strategy, structure and performance*, Promoter(s): Prof.dr. J.Y.F. Wynstra & Dr. E.M.van Raaij, EPS-2014-300-LIS, `http://repub.eur.nl/pub/50283`

Akpinar, E., *Consumer Information Sharing*, Promoter(s): Prof.dr.ir. A. Smidts, EPS-2013-297-MKT, `http://repub.eur.nl/pub/50140`

Alexander, L., *People, Politics, and Innovation: A Process Perspective*, Promoter(s): Prof.dr. H.G. Barkema & Prof.dr. D.L. van Knippenberg, EPS-2014-331-S&E, http://repub.eur.nl/pub/77209

Alexiou, A., *Management of Emerging Technologies and the Learning Organization: Lessons from the Cloud and Serious Games Technology*, Promoter(s): Prof. S.J. Magala, Prof. M.C. Schippers, & Dr. I. Oshri, EPS-2016-404-ORG, http://repub.eur.nl/pub/93818

Bannouh, K., *Measuring and Forecasting Financial Market Volatility using High-frequency Data*, Promoter(s): Prof.dr. D.J.C. van Dijk, EPS-2013-273-F&A, http://repub.eur.nl/pub/38240

Ben-Menahem, S., *Strategic Timing and Proactiveness of Organizations*, Promoter(s): Prof.dr. H.W. Volberda & Prof.dr.ing. F.A.J. van den Bosch, EPS-2013-278-S&E, http://repub.eur.nl/pub/39128

Benning, T., *A Consumer Perspective on Flexibility in Health Care: Priority Access Pricing and Customized Care*, Promoter(s): Prof.dr.ir. B.G.C. Dellaert, EPS-2011-241-MKT, http://repub.eur.nl/pub/23670

Benschop, N., *Biases in Project Escalation: Names, frames & construal levels*, Promoter(s): Prof.dr. K.I.M. Rhode, Prof.dr. H.R. Commandeur, Prof.dr. M.Keil, & Dr. A.L.P. Nuijten, EPS-2015-375-S&E, hdl.handle.net/1765/79408

Berg, W.v.d., *Understanding Salesforce Behavior using Genetic Association Studies*, Promoter(s): Prof.dr. W.J.M.I. Verbeke, EPS-2014-311-MKT, http://repub.eur.nl/pub/51440

Betancourt, N., *Typical Atypicality: Formal and Informal Institutional Conformity, Deviance, and Dynamics*, Promoter(s): Prof.dr. B. Krug, EPS-2012-262-ORG, http://repub.eur.nl/pub/32345

Beusichem, H.v., *Firms and Financial Markets: Empirical Studies on the Informational Value of Dividends, Governance and Financial Reporting*, Promoter(s): Prof. A. de Jong & Dr. G. Westerhuis, EPS-2016-378-F&A, http://repub.eur.nl/pub/93079

Bliek, R.d., *Empirical Studies on the Economic Impact of Trust*, Promoter(s): Prof.dr.J. Veenman & Prof.dr. Ph.H.B.F. Franses, EPS-2015-324-ORG, http://repub.eur.nl/pub/78159

Blitz, D., *Benchmarking Benchmarks*, Promoter(s): Prof.dr. A.G.Z. Kemna & Prof.dr.W.F.C. Verschoor, EPS-2011-225-F&A, http://repub.eur.nl/pub/22624

Boons, M., *Working Together Alone in the Online Crowd: The Effects of Social Motivations and Individual Knowledge Backgrounds on the Participation and Performance of Members of Online Crowdsourcing Platforms*, Promoter(s): Prof.dr. H.G. Barkema & Dr. D.A. Stam, EPS-2014-306-S&E, http://repub.eur.nl/pub/50711

Brazys, J., *Aggregated Marcoeconomic News and Price Discovery*, Promoter(s): Prof.dr.W.F.C. Verschoor, EPS-2015-351-F&A, http://repub.eur.nl/pub/78243

Burger, M., *Structure and Cooptition in Urban Networks*, Promoter(s): Prof.dr. G.A.van der Knaap & Prof.dr. H.R. Commandeur, EPS-2011-243-ORG, http://repub.eur.nl/pub/26178

Byington, E., *Exploring Coworker Relationships: Antecedents and Dimensions of Interpersonal Fit, Coworker Satisfaction, and Relational Models*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2013-292-ORG, http://repub.eur.nl/pub/41508

Camacho, N., *Health and Marketing: Essays on Physician and Patient Decision-Making*, Promoter(s): Prof.dr. S. Stremersch, EPS-2011-237-MKT, http://repub.eur.nl/pub/23604

Cancurtaran, P., *Essays on Accelerated Product Development*, Promoter(s): Prof.dr. F.Langerak & Prof.dr.ir. G.H. van Bruggen, EPS-2014-317-MKT, http://repub.eur.nl/pub/76074

Caron, E., *Explanation of Exceptional Values in Multi-dimensional Business Databases*, Promoter(s): Prof.dr.ir. H.A.M. Daniels & Prof.dr. G.W.J. Hendrikse, EPS-2013-296-LIS, http://repub.eur.nl/pub/50005

Carvalho, L.d., *Knowledge Locations in Cities: Emergence and Development Dynamics*, Promoter(s): Prof.dr. L. Berg, EPS-2013-274-S&E, http://repub.eur.nl/pub/38449

Consiglio, I., *Others: Essays on Interpersonal and Consumer Behavior*, Promoter(s): Prof.dr. S.M.J. van Osselaer, EPS-2016-366-MKT, http://repub.eur.nl/pub/79820

Cox, R., *To Own, To Finance, and To Insure - Residential Real Estate Revealed*, Promoter(s): Prof.dr. D. Brounen, EPS-2013-290-F&A, http://repub.eur.nl/pub/40964

Cranenburgh, K.v., *Money or Ethics: Multinational corporations and religious organisations operating in an era of corporate responsibility*, Promoter(s): Prof. L.C.P.M. Meijs, Prof. R.J.M. van Tulder, & Dr. D. Arenas, EPS-2016-385-ORG, http://repub.eur.nl/pub/93104

Darnihamedani, P., *Individual Characteristics, Contextual Factors and Entrepreneurial Behavior*, Promoter(s): Prof. A.R. Thurik & S.J.A. Hessels, EPS-2016-360-S&E, http://repub.eur.nl/pub/93280

Deichmann, D., *Idea Management: Perspectives from Leadership, Learning, and Network Theory*, Promoter(s): Prof.dr.ir. J.C.M. van den Ende, EPS-2012-255-ORG, http://repub.eur.nl/pub/31174

Deng, W., *Social Capital and Diversification of Cooperatives*, Promoter(s): Prof.dr. G.W.J. Hendrikse, EPS-2015-341-ORG, http://repub.eur.nl/pub/77449

Depecik, B., *Revitalizing brands and brand: Essays on Brand and Brand Portfolio Management Strategies*, Promoter(s): Prof. G.H. van Bruggen, Dr. Y.M. van Everdingen, & Dr. M.B. Ataman, EPS-2016406-MKT, http://repub.eur.nl/pub/93507

Desmet, P., *In Money we Trust? Trust Repair and the Psychology of Financial Compensations*, Promoter(s): Prof.dr. D. de Cremer, EPS-2011-232-ORG, http://repub.eur.nl/pub/23268

Dollevoet, T., *Delay Management and Dispatching in Railways*, Promoter(s): Prof.dr. A.P.M. Wagelmans, EPS-2013-272-LIS, http://repub.eur.nl/pub/38241

Doorn, S.v., *Managing Entrepreneurial Orientation*, Promoter(s): Prof.dr. J.J.P.Jansen, Prof.dr.ing. F.A.J. van den Bosch, & Prof.dr. H.W. Volberda, EPS-2012-258-STR, http://repub.eur.nl/pub/32166

Douwens-Zonneveld, M., *Animal Spirits and Extreme Confidence: No Guts, No Glory*, Promoter(s): Prof.dr. W.F.C. Verschoor, EPS-2012-257-F&A, http://repub.eur.nl/pub/31914

Duca, E., *The Impact of Investor Demand on Security Offerings*, Promoter(s): Prof.dr.A. de Jong, EPS-2011-240-F&A, http://repub.eur.nl/pub/26041

Duursema, H., *Strategic Leadership: Moving Beyond the Leader-Follower Dyad*, Promoter(s): Prof.dr. R.J.M. van Tulder, EPS-2013-279-ORG, http://repub.eur.nl/pub/39129

Duyvesteyn, J.E.S.o.S.F.I.M., *Empirical Studies on Sovereign Fixed Income Markets*, Promoter(s): Prof.dr. P.Verwijmeren & Prof.dr. M.P.E. Martens, EPS-2015-361-F&A, hdl.handle.net/1765/79033

Eck, N.v., *Methodological Advances in Bibliometric Mapping of Science*, Promoter(s): Prof.dr.ir. R. Dekker, EPS-2011-247-LIS, http://repub.eur.nl/pub/26509

Elemes, A., *Studies on Determinants and Consequencesof Financial Reporting Quality*, Promoter(s): Prof.dr. E.Peek, EPS-2015-354-F&A, http://hdl.handle.net/1765/79037

Ellen, S.t., *Measurement, Dynamics, and Implications of Heterogeneous Beliefs in Financial Markets*, Promoter(s): Prof.dr. W.F.C. Verschoor, EPS-2015-343-F&A, http://repub.eur.nl/pub/78191

Erlemann, C., *Gender and Leadership Aspiration: The Impact of the Organizational Environment*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2016-376-ORG, http://repub.eur.nl/pub/79409

Eskenazi, P., *The Accountable Animal*, Promoter(s): Prof.dr. F.G.H. Hartmann, EPS-2015-355-F&A, http://repub.eur.nl/pub/78300

Essen, M.v., *An Institution-Based View of Ownership*, Promoter(s): Prof.dr. J. van Oosterhout & Prof.dr. G.M.H. Mertens, EPS-2011-226-ORG, http://repub.eur.nl/pub/22643

Evangelidis, I., *Preference Construction under Prominence*, Promoter(s): Prof.dr. S.M.J.van Osselaer, EPS-2015-340-MKT, http://repub.eur.nl/pub/78202

Faber, N., *Structuring Warehouse Management*, Promoter(s): Prof.dr. MB.M. de Koster & Prof.dr. Ale Smidts, EPS-2015-336-LIS, http://repub.eur.nl/pub/78603

Fernald, K., *The Waves of Biotechnological Innovation in Medicine: Interfirm Cooperation Effects and a Venture Capital Perspective*, Promoter(s): Prof.dr. E.Claassen, Prof.dr. H.P.G.Pennings, & Prof.dr. H.R. Commandeur, EPS-2015-371-S&E, http://hdl.handle.net/1765/79120

Fliers, P., *Essays on Financing and Performance: The role of firms, banks and board*, Promoter(s): Prof. A. de Jong & Prof P.G.J. Roosenboom, EPS-2016-388-F&A, http://repub.eur.nl/pub/93019

Fourne, S., *Managing Organizational Tensions: A Multi-Level Perspective on Exploration, Exploitation and Ambidexterity*, Promoter(s): Prof.dr. J.J.P. Jansen & Prof.dr.S.J. Magala, EPS-2014-318-S&E, http://repub.eur.nl/pub/76075

Gaast, J.v.d., *Stochastic Models for Order Picking Systems*, Promoter(s): Prof. M.B.M de Koster & Prof. I.J.B.F. Adan, EPS-2016-398-LIS, http://repub.eur.nl/pub/93222

Gharehgozli, A., *Developing New Methods for Efficient Container Stacking Operations*, Promoter(s): Prof.dr.ir. M.B.M. de Koster, EPS-2012-269-LIS, http://repub.eur.nl/pub/37779

Gils, S.v., *Morality in Interactions: On the Display of Moral Behavior by Leaders and Employees*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2012-270-ORG, http://repub.eur.nl/pub/38027

Ginkel-Bieshaar, M.v., *The Impact of Abstract versus Concrete Product Communications on Consumer Decision-making Processes*, Promoter(s): Prof.dr.ir. B.G.C.Dellaert, EPS-2012-256-MKT, http://repub.eur.nl/pub/31913

Gkougkousi, X., *Empirical Studies in Financial Accounting*, Promoter(s): Prof.dr.G.M.H. Mertens & Prof.dr. E. Peek, EPS-2012-264-F&A, http://repub.eur.nl/pub/37170

Glorie, K., *Clearing Barter Exchange Markets: Kidney Exchange and Beyond*, Promoter(s): Prof.dr. A.P.M. Wagelmans & Prof.dr. J.J. van de Klundert, EPS-2014-329-LIS, http://repub.eur.nl/pub/77183

Heij, C., *Innovating beyond Technology. Studies on how management innovation, co-creation and business model innovation contribute to firm's (innovation) performance*, Promoter(s): Prof.dr.ing. F.A.J. van den Bosch & Prof.dr. H.W. Volberda, EPS-2012-370-STR, http://repub.eur.nl/pub/78651

Hekimoglu, M., *Spare Parts Management of Aging Capital Products*, Promoter(s): Prof.dr.ir. R. Dekker, EPS-2015-368-LIS, http://hdl.handle.net/1765/79092

Heyde Fernandes, D.v.d., *The Functions and Dysfunctions of Reminders*, Promoter(s): Prof.dr. S.M.J. van Osselaer, EPS-2013-295-MKT, http://repub.eur.nl/pub/41514

Heyden, M., *Essays on Upper Echelons & Strategic Renewal: A Multilevel Contingency Approach*, Promoter(s): Prof.dr.ing. F.A.J. van den Bosch & Prof.dr. H.W.Volberda, EPS-2012-259-STR, http://repub.eur.nl/pub/32167

Hoever, I., *Diversity and Creativity*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2012-267-ORG, http://repub.eur.nl/pub/37392

Hogenboom, A., *Sentiment Analysis of Text Guided by Semantics and Structure*, Promoter(s): Prof.dr.ir. U.Kaymak & Prof.dr. F.M.G. de Jong, EPS-2015-369-LIS, http://hdl.handle.net/1765/79034

Hogenboom, F., *Automated Detection of Financial Events in News Text*, Promoter(s): Prof.dr.ir. U. Kaymak & Prof.dr. F.M.G. de Jong, EPS-2014-326-LIS, http://repub.eur.nl/pub/77237

Hollen, R., *Exploratory Studies into Strategies to Enhance Innovation-Driven International Competitiveness in a Port Context: Toward Ambidextrous Ports*, Promoter(s): Prof.dr.ing. F.A.J. Van Den Bosch & Prof.dr. H.W.Volberda, EPS-2015-372-S&E, hdl.handle.net/1765/78881

Hoogendoorn, B., *Social Entrepreneurship in the Modern Economy: Warm Glow, ColdFeet*, Promoter(s): Prof.dr. H.P.G. Pennings & Prof.dr. A.R. Thurik, EPS-2011-246-STR, http://repub.eur.nl/pub/26447

Hoogervorst, N., *On The Psychology of Displaying Ethical Leadership: A Behavioral Ethics Approach*, Promoter(s): Prof.dr. D. de Cremer & Dr. M. van Dijke, EPS-2011-244-ORG, http://repub.eur.nl/pub/26228

Hout, D.v., *Measuring Meaningful Differences: Sensory Testing Based Decision Making in an Industrial Context; Applications of Signal Detection Theory and Thurstonian Modelling*, Promoter(s): Prof.dr. P.J.F. Groenen & Prof.dr. G.B. Dijksterhuis, EPS-2014-304-MKT, http://repub.eur.nl/pub/50387

Houwelingen, G.v., *Something To Rely On*, Promoter(s): Prof.dr. D. de Cremer & Prof.dr. M.H. van Dijke, EPS-2014-335-ORG, http://repub.eur.nl/pub/77320

Hurk, E.v.d., *Passengers, Information, and Disruptions*, Promoter(s): Prof.dr. L.G.Kroon & Prof.mr.dr. P.H.M. Vervest, EPS-2015-345-LIS, http://repub.eur.nl/pub/78275

Hytonen, K., *Context Effects in Valuation, Judgment and Choice: A Neuroscientific Approach*, Promoter(s): Prof.dr.ir. A. Smidts, EPS-2011-252-MKT, http://repub.eur.nl/pub/30668

Iseger, P.d., *Fourier and Laplace Transform Inversion with Applications in Finance*, Promoter(s): Prof.dr.ir. R. Dekker, EPS-2014-322-LIS, http://repub.eur.nl/pub/76954

Jaarsveld, W.v., *Maintenance Centered Service Parts Inventory Control*, Promoter(s): Prof.dr.ir. R. Dekker, EPS-2013-288-LIS, http://repub.eur.nl/pub/39933

Jalil, M., *Customer Information Driven After Sales Service Management: Lessons from Spare Parts Logistics*, Promoter(s): Prof.dr. L.G. Kroon, EPS-2011-222-LIS, http://repub.eur.nl/pub/22156

Kappe, E., *The Effectiveness of Pharmaceutical Marketing*, Promoter(s): Prof.dr. S.Stremersch, EPS-2011-239-MKT, http://repub.eur.nl/pub/23610

Karreman, B., *Financial Services and Emerging Markets*, Promoter(s): Prof.dr. G.A.van der Knaap & Prof.dr. H.P.G. Pennings, EPS-2011-223-ORG, http://repub.eur.nl/pub/22280

Khanagha, S., *Dynamic Capabilities for Managing Emerging Technologies*, Promoter(s): Prof.dr. H.W. Volberda, EPS-2014-339-S&E, http://repub.eur.nl/pub/77319

Kil, J., *Acquisitions Through a Behavioral and Real Options Lens*, Promoter(s): Prof.dr.H.T.J. Smit, EPS-2013-298-F&A, http://repub.eur.nl/pub/50142

Klooster, E.v.t., *Travel to Learn: the Influence of Cultural Distance on Competence Development in Educational Travel*, Promoter(s): Prof.dr. F.M. Go & Prof.dr. P.J. van Baalen, EPS-2014-312-MKT, http://repub.eur.nl/pub/51462

Koendjbiharie, S., *The Information-Based View on Business Network Performance: Revealing the Performance of Interorganizational Networks*, Promoter(s): Prof.dr.ir. H.W.G.M. van Heck & Prof.mr.dr. P.H.M. Vervest, EPS-2014-315-LIS, http://repub.eur.nl/pub/51751

Koning, M., *The Financial Reporting Environment: The Role of the Media, Regulatorsand Auditors*, Promoter(s): Prof.dr. G.M.H. Mertens & Prof.dr. P.G.J. Roosenboom, EPS-2014-330-F&A, http://repub.eur.nl/pub/77154

Konter, D., *Crossing Borders with HRM: An Inquiry of the Influence of Contextual Differences in the Adoption and Effectiveness of HRM*, Promoter(s): Prof.dr. J. Paauweand Dr. L.H. Hoeksema, EPS-2014-305-ORG, http://repub.eur.nl/pub/50388

Korkmaz, E., *Bridging Models and Business: Understanding Heterogeneity in HiddenDrivers of Customer Purchase Behavior*, Promoter(s): Prof.dr. S.L. van de Velde & Prof.dr. D. Fok, EPS-2014-316-LIS, http://repub.eur.nl/pub/76008

Kroezen, J., *The Renewal of Mature Industries: An Examination of the Revival of theDutch Beer Brewing Industry*, Promoter(s): Prof.dr. P.P.M.A.R. Heugens, EPS-2014-333-S&E, http://repub.eur.nl/pub/77042

Kysucky, V., *Access to Finance in a Cros-Country Context*, Promoter(s): Prof.dr. L. Norden, EPS-2015-350-F&A, http://repub.eur.nl/pub/78225

Lam, K., *Reliability and Rankings*, Promoter(s): Prof.dr. Ph.H.B.F. Franses, EPS-2011-230-MKT, http://repub.eur.nl/pub/22977

Lander, M., *Profits or Professionalism? On Designing Professional Service Firms*, Promoter(s): Prof.dr. J. van Oosterhout & Prof.dr. P.P.M.A.R. Heugens, EPS-2012-253-ORG, http://repub.eur.nl/pub/30682

Langhe, B.d., *Contingencies: Learning Numerical and Emotional Associations in an Uncertain World*, Promoter(s): Prof.dr.ir. B. Wierenga & Prof.dr. S.M.J. van Osselaer, EPS-2011-236-MKT, http://repub.eur.nl/pub/23504

Lee, C., *Big Data in Management Research: Exploring New Avenues*, Promoter(s): Prof.dr. S.J. Magala & Dr. W.A. Felps, EPS-2016-365-ORG, http://repub.eur.nl/pub/79818

Legault-Tremblay, P., *Corporate Governance During Market Transition: Heterogeneous responses to Institution Tensions in China*, Promoter(s): Prof.dr. B. Krug, EPS-2015-362-ORG, http://repub.eur.nl/pub/78649

Lenoir, A.A.Y.T.t.M.A.C.i.a.G.W., *Are Your Talking to Me? Addressing Consumers in a Globalised World*, Promoter(s): Prof.dr. S. Puntoni & Prof.dr. S.M.J. van Osselaer, EPS-2015-363-MKT, http://hdl.handle.net/1765/79036

Leunissen, J., *All Apologies: On the Willingness of Perpetrators to Apologize*, Promoter(s): Prof.dr. D. de Cremer & Dr. M. van Dijke, EPS-2014-301-ORG, http://repub.eur.nl/pub/50318

Li, D., *Supply Chain Contracting for After-sales Service and Product Support*, Promoter(s): Prof.dr.ir. M.B.M. de Koster, EPS-2015-347-LIS, http://repub.eur.nl/pub/78526

Li, Z., *Irrationality: What, Why and How*, Promoter(s): Prof.dr. H. Bleichrodt, Prof.dr. P.P. Wakker, & Prof.dr. K.I.M. Rohde, EPS-2014-338-MKT, http://repub.eur.nl/pub/77205

Liang, Q., *Governance, CEO Identity, and Quality Provision of Farmer Cooperatives*, Promoter(s): Prof.dr. G.W.J. Hendrikse, EPS-2013-281-ORG, http://repub.eur.nl/pub/39253

Liket, K., *Why 'Doing Good' is not Good Enough: Essays on Social Impact Measurement*, Promoter(s): Prof.dr. H.R. Commandeur & Dr. K.E.H. Maas, EPS-2014-307-STR, http://repub.eur.nl/pub/51130

Loos, M.v.d., *Molecular Genetics and Hormones: New Frontiers in Entrepreneurship Research*, Promoter(s): Prof.dr. A.R. Thurik, Prof.dr. P.J.F. Groenen, & Prof.dr. A. Hofman, EPS-2013-287-S&E, http://repub.eur.nl/pub/40081

Lovric, M., *Behavioral Finance and Agent-Based Artificial Markets*, Promoter(s): Prof.dr.J. Spronk & Prof.dr.ir. U. Kaymak, EPS-2011-229-F&A, http://repub.eur.nl/pub/22814

Lu, Y., *Data-Driven Decision Making in Auction Markets*, Promoter(s): Prof.dr.ir.H.W.G.M. van Heck & Prof.dr. W. Ketter, EPS-2014-314-LIS, http://repub.eur.nl/pub/51543

Ma, Y., *The Use of Advanced Transportation Monitoring Data for Official Statistics*, Promoter(s): Prof. L.G. Kroon & Dr. Jan van Dalen, EPS-2016-391-LIS, http://repub.eur.nl/pub/80174

Manders, B., *Implementation and Impact of ISO 9001*, Promoter(s): Prof.dr. K. Blind, EPS-2014-337-LIS, http://repub.eur.nl/pub/77412

Markwat, T., *Extreme Dependence in Asset Markets Around the Globe*, Promoter(s): Prof.dr. D.J.C. van Dijk, EPS-2011-227-F&A, http://repub.eur.nl/pub/22744

Mees, H., *Changing Fortunes: How China's Boom Caused the Financial Crisis*, Promoter(s): Prof.dr. Ph.H.B.F. Franses, EPS-2012-266-MKT, http://repub.eur.nl/pub/34930

Mell, J., *Connecting Minds: On The Role of Metaknowledge in Knowledge Coordination*, Promoter(s): Prof.dr.D.L. van Knippenberg, EPS-2015-359-ORG, http://hdl.handle.net/1765/78951

Meuer, J., *Configurations of Inter-firm Relations in Management Innovation: A Study in China's Biopharmaceutical Industry*, Promoter(s): Prof.dr. B. Krug, EPS-2011-228-ORG, http://repub.eur.nl/pub/22745

Micheli, M., *Business Model Innovation: A Journey across Managers' Attention and Inter-Organizational Networks*, Promoter(s): Prof.dr. J.J.P. Jansen, EPS-2015-344-S&E, http://repub.eur.nl/pub/78241

Mihalache, O., *Stimulating Firm Innovativeness: Probing the Interrelations between Managerial and Organizational Determinants*, Promoter(s): Prof.dr. J.J.P. Jansen, Prof.dr.ing. F.A.J. van den Bosch, & Prof.dr. H.W. Volberda, EPS-2012-260-S&E, http://repub.eur.nl/pub/32343

Milea, V., *News Analytics for Financial Decision Support*, Promoter(s): Prof.dr.ir. U.Kaymak, EPS-2013-275-LIS, http://repub.eur.nl/pub/38673

Moniz, A., *Textual Analysis of Intangible Information*, Promoter(s): Prof. C.B.M. van Riel, Prof. F.M.G de Jong, & Dr. G.A.J.M. Berens, EPS-2016-393-ORG, http://repub.eur.nl/pub/93001

Mulder, J., *Network design and robust scheduling in liner shipping*, Promoter(s): Prof. R. Dekker & Dr. W.L. van Jaarsveld, EPS-2016-384-LIS, http://repub.eur.nl/pub/80258

Naumovska, I., *Socially Situated Financial Markets: A Neo-Behavioral Perspective on Firms, Investors and Practices*, Promoter(s): Prof.dr. P.P.M.A.R. Heugens & Prof.dr. A.de Jong, EPS-2014-319-S&E, http://repub.eur.nl/pub/76084

Neerijnen, P., *The Adaptive Organization: the socio-cognitive antecedents of ambidexterity and individual exploration*, Promoter(s): Prof. J.J.P. Jansen, P.P.M.A.R. Heugens, & Dr T.J.M. Mom, EPS-2016-358-S&E, http://repub.eur.nl/pub/93274

Nielsen, L., *Rolling Stock Rescheduling in Passenger Railways: Applications in short term planning and in disruption management*, Promoter(s): Prof.dr. L.G. Kroon, EPS-2011-224-LIS, http://repub.eur.nl/pub/22444

Nuijten, A., *Deaf Effect for Risk Warnings: A Causal Examination applied to Information Systems Projects*, Promoter(s): Prof.dr. G.J. van der Pijl, Prof.dr. H.R. Commandeur, & Prof.dr. M. Keil, EPS-2012-263-S&E, http://repub.eur.nl/pub/34928

Oord, J.v., *Essays on Momentum Strategies in Finance*, Promoter(s): Prof. H.K. van Dijk, EPS-2016-380-F&A, http://repub.eur.nl/pub/80036

Osadchiy, S., *The Dynamics of Formal Organization: Essays on bureaucracy and formal rules*, Promoter(s): Prof.dr. P.P.M.A.R. Heugens, EPS-2011-231-ORG, http://repub.eur.nl/pub/23250

Ozdemir, M., *Project-level Governance, Monetary Incentives, and Performance in Strategic R&D Alliances*, Promoter(s): Prof.dr.ir. J.C.M. van den Ende, EPS-2011-235-LIS, http://repub.eur.nl/pub/23550

Peers, Y., *Econometric Advances in Diffusion Models*, Promoter(s): Prof.dr. Ph.H.B.F.Franses, EPS-2011-251-MKT, http://repub.eur.nl/pub/30586

Peters, M., *Machine Learning Algorithms for Smart Electricity Markets*, Promoter(s): Prof.dr. W. Ketter, EPS-2014-332-LIS, http://repub.eur.nl/pub/77413

Porck, J., *No Team is an Island: An Integrative View of Strategic Consensus between Groups*, Promoter(s): Prof.dr. P.J.F. Groenen & Prof.dr. D.L. van Knippenberg, EPS-2013-299-ORG, http://repub.eur.nl/pub/50141

Porras Prado, M., *The Long and Short Side of Real Estate, Real Estate Stocks, and Equity*, Promoter(s): Prof.dr. M.J.C.M. Verbeek, EPS-2012-254-F&A, http://repub.eur.nl/pub/30848

Poruthiyil, P., *Steering Through: How organizations negotiate permanent uncertainty and unresolvable choices*, Promoter(s): Prof.dr. P.P.M.A.R. Heugens & Prof.dr. S.J.Magala, EPS-2011-245-ORG, http://repub.eur.nl/pub/26392

Pourakbar, M., *End-of-Life Inventory Decisions of Service Parts*, Promoter(s): Prof.dr.ir.R. Dekker, EPS-2011-249-LIS, http://repub.eur.nl/pub/30584

Pronker, E., *Innovation Paradox in Vaccine Target Selection*, Promoter(s): Prof.dr.H.J.H.M. Claassen & Prof.dr. H.R. Commandeur, EPS-2013-282-S&E, http://repub.eur.nl/pub/39654

Protzner, S.M.t.g.b.d. & forecasting supply: A behavioral perspective on demand, *Mind the gap between demand and supply*, Promoter(s): Prof.dr. S.L. van de Velde & Dr. L. Rook, EPS-2015-364-LIS, http://repub.eur.nl/pub/79355

Pruijssers, J., *An Organizational Perspective on Auditor Conduct*, Promoter(s): Prof.dr. J. van Oosterhout & Prof.dr. P.P.M.A.R. Heugens, EPS-2015-342-S&E, http://repub.eur.nl/pub/78192

Retel Helmrich, M., *Green Lot-Sizing*, Promoter(s): Prof.dr. A.P.M. Wagelmans, EPS-2013-291-LIS, http://repub.eur.nl/pub/41330

Rietdijk, W., *The Use of Cognitive Factors for Explaining Entrepreneurship*, Promoter(s): Prof.dr. A.R. Thurik & Prof.dr. I.H.A. Franken, EPS-2015-356-S&E, http://repub.eur.nl/pub/79817

Rietveld, N., *Essays on the Intersection of Economics and Biology*, Promoter(s): Prof.dr. A.R. Thurik, Prof.dr. Ph.D. Koellinger, Prof.dr. P.J.F. Groenen, & Prof.dr. A. Hofman, EPS-2014-320-S&E, http://repub.eur.nl/pub/76907

Rijsenbilt, J., *CEO Narcissism: Measurement and Impact*, Promoter(s): Prof.dr.A.G.Z. Kemna & Prof.dr. H.R. Commandeur, EPS-2011-238-STR, http://repub.eur.nl/pub/23554

Roza, L., *Employee Engagement In Corporate Social Responsibility: A collection of essays*, Promoter(s): L.C.P.M. Meijs, EPS-2016-396-ORG, http://repub.eur.nl/pub/93254

Rubbaniy, G., *Investment Behaviour of Institutional Investors*, Promoter(s): Prof.dr.W.F.C. Verschoor, EPS-2013-284-F&A, http://repub.eur.nl/pub/40068

Rösch, D.M.E. & Liquidity, *Market Efficiency and Liquidity*, Promoter(s): Prof.dr. M.A. van Dijk, EPS-2015-353-F&A, http://hdl.handle.net/1765/79121

Santos Nogueira, R.d. Almeida e, *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty*, Promoter(s): Prof.dr.ir. U. Kaymak & Prof.dr. J.M.C. Sousa, EPS-2014-310-LIS, http://repub.eur.nl/pub/51560

Schoonees, P.M.f.M.R.S., *Methods for Modelling Response Styles*, Promoter(s): Prof.dr. P.J.F. Groenen, EPS-2015-348-MKT, http://repub.eur.nl/pub/79327

Schouten, M., *The Ups and Downs of Hierarchy: the causes and consequences of hierarchy struggles and positional loss*, Promoter(s): Prof. D.L. van Knippenberg & Dr. L.L. Greer, EPS-2016-386-ORG, http://repub.eur.nl/pub/80059

Shahzad, K., *Credit Rating Agencies, Financial Regulations and the Capital Markets*, Promoter(s): Prof.dr. G.M.H. Mertens, EPS-2013-283-F&A, http://repub.eur.nl/pub/39655

Smit, J., *Unlocking Business Model Innovation: A look through the keyhole at the inner workings of Business Model Innovation*, Promoter(s): Prof. H.G. Barkema, EPS-2016-399-S&E, http://repub.eur.nl/pub/93211

Sousa, M.d., *Servant Leadership to the Test: New Perspectives and Insights*, Promoter(s): Prof.dr. D.L. van Knippenberg & Dr. D. van Dierendonck, EPS-2014-313-ORG, http://repub.eur.nl/pub/51537

Spliet, R., *Vehicle Routing with Uncertain Demand*, Promoter(s): Prof.dr.ir. R. Dekker, EPS-2013-293-LIS, http://repub.eur.nl/pub/41513

Staadt, J., *Leading Public Housing Organisation in a Problematic Situation: A Critical Soft Systems Methodology Approach*, Promoter(s): Prof.dr. S.J. Magala, EPS-2014-308-ORG, http://repub.eur.nl/pub/50712

Stallen, M., *Social Context Effects on Decision-Making: A Neurobiological Approach*, Promoter(s): Prof.dr.ir. A. Smidts, EPS-2013-285-MKT, http://repub.eur.nl/pub/39931

Tarakci, M., *Behavioral Strategy: Strategic Consensus, Power and Networks*, Promoter(s): Prof.dr. D.L. van Knippenberg & Prof.dr. P.J.F. Groenen, EPS-2013-280-ORG, http://repub.eur.nl/pub/39130

Troster, C., *Nationality Heterogeneity and Interpersonal Relationships at Work*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2011-233-ORG, http://repub.eur.nl/pub/23298

Tsekouras, D., *No Pain No Gain: The Beneficial Role of Consumer Effort in Decision-Making*, Promoter(s): Prof.dr.ir. B.G.C. Dellaert, EPS-2012-268-MKT, http://repub.eur.nl/pub/37542

Tuijl, E.v., *Upgrading across Organisational and Geographical Configurations*, Promoter(s): Prof.dr. L. van den Berg, EPS-2015-349-S&E, http://repub.eur.nl/pub/78224

Tuncdogan, A., *Decision Making and Behavioral Strategy: The Role of Regulatory Focusin Corporate Innovation Processes*, Promoter(s): Prof.dr.ing. F.A.J. van den

Bosch, Prof.dr. H.W. Volberda, & Prof.dr. T.J.M. Mom, EPS-2014-334-S&E, http://repub.eur.nl/pub/76978

Uijl, S.d., *The Emergence of De-facto Standards*, Promoter(s): Prof.dr. K. Blind, EPS-2014-328-LIS, http://repub.eur.nl/pub/77382

Vagias, D., *Liquidity, Investors and International Capital Markets*, Promoter(s): Prof.dr.M.A. van Dijk, EPS-2013-294-F&A, http://repub.eur.nl/pub/41511

Valogianni, K., *Sustainable Electric Vehicle Management using Coordinated Machine Learning*, Promoter(s): Prof. H.W.G.M. van Heck & Prof. W. Ketter, EPS-2016-387-LIS, http://repub.eur.nl/pub/93018

Vasconcelos, M. Teixeira de, *Agency Costs, Firm Value, and Corporate Investment*, Promoter(s): Prof.dr. P.G.J. Roosenboom, EPS-2012-265-F&A, http://repub.eur.nl/pub/37265

Veelenturf, L., *Disruption Management in Passenger Railways: Models for Timetable,Rolling Stock and Crew Rescheduling*, Promoter(s): Prof.dr. L.G. Kroon, EPS-2014-327-LIS, http://repub.eur.nl/pub/77155

Venus, M., *Demystifying Visionary Leadership: In search of the essence of effective-vision communication*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2013-289-ORG, http://repub.eur.nl/pub/40079

Vermeer, W., *Propagation in Networks:The impact of information processing at the actor level on system-wide propagation dynamics*, Promoter(s): Prof.mr.dr. P.H.M. Vervest, EPS-2015-373-LIS, http://repub.eur.nl/pub/79325

Versluis, I., *Prevention of the Portion Size Effect*, Promoter(s): Prof. Ph.H.B.F. Franses & Dr. E.K. Papies, EPS-2016-382-MKT, http://repub.eur.nl/pub/79880

Vishwanathan, P., *Governing for Stakeholders: How Organizations May Create or Destroy Value for their Stakeholders*, Promoter(s): Prof. J. van Oosterhout & Prof. L.C.P. M. Meijs, EPS-2016-377-ORG, http://repub.eur.nl/pub/93016

Visser, V., *Leader Affect and Leadership Effectiveness: How leader affective displays influence follower outcomes*, Promoter(s): Prof.dr. D.L. van Knippenberg, EPS-2013-286-ORG, http://repub.eur.nl/pub/40076

Vlam, A., *Customer First? The Relationship between Advisors and Consumers of Financial Products*, Promoter(s): Prof.dr. Ph.H.B.F. Franses, EPS-2011-250-MKT, http://repub.eur.nl/pub/30585

Vries, J.d., *Behavioral Operations in Logistics*, Promoter(s): Prof.dr. M.B.M de Koster & Prof.dr. D.A. Stam, EPS-2015-374-LIS, http://repub.eur.nl/pub/79705

Vuren, M. Roza-van, *The Relationship between Offshoring Strategies and FirmPerformance: Impact of innovation, absorptive capacity and firm size*, Promoter(s): Prof.dr. H.W. Volberda & Prof.dr.ing. F.A.J. van den Bosch, EPS-2011-214-STR, http://repub.eur.nl/pub/22155

Wagenaar, J., *Practice Oriented Algorithmic Disruption Management in Passenger Railways*, Promoter(s): Prof. L.G. Kroon & Prof. A.P.M. Wagelmans, EPS-2016-390-LIS, http://repub.eur.nl/pub/93177

Waltman, L., *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*, Promoter(s): Prof.dr.ir. R. Dekker & Prof.dr.ir. U. Kaymak, EPS-2011-248-LIS, http://repub.eur.nl/pub/26564

Wang, P., *Innovations, status, and networks*, Promoter(s): Prof. J.J.P. Jansen & Dr. V.J.A. van de Vrande, EPS-2016-381-S&E, http://repub.eur.nl/pub/93176

Wang, T., *Essays in Banking and Corporate Finance*, Promoter(s): Prof.dr. L. Nordenand Prof.dr. P.G.J. Roosenboom, EPS-2015-352-F&A, http://repub.eur.nl/pub/78301

Wang, Y., *Information Content of Mutual Fund Portfolio Disclosure*, Promoter(s): Prof.dr. M.J.C.M. Verbeek, EPS-2011-242-F&A, http://repub.eur.nl/pub/26066

Wang, Y., *Corporate Reputation Management: Reaching Out to Financial Stakeholders*, Promoter(s): Prof.dr. C.B.M. van Riel, EPS-2013-271-ORG, http://repub.eur.nl/pub/38675

Weenen, T., *On the Origin and Development of the Medical Nutrition Industry*, Promoter(s): Prof.dr. H.R. Commandeur & Prof.dr. H.J.H.M. Claassen, EPS-2014-309-S&E, http://repub.eur.nl/pub/51134

Wolfswinkel, M., *Corporate Governance, Firm Risk and Shareholder Value*, Promoter(s): Prof.dr. A. de Jong, EPS-2013-277-F&A, http://repub.eur.nl/pub/39127

Yang, S., *Information Aggregation Efficiency of Prediction Markets*, Promoter(s): Prof.dr.ir. H.W.G.M. van Heck, EPS-2014-323-LIS, http://repub.eur.nl/pub/77184

Yuferova, D., *Price Discovery, Liquidity Provision, and Low-Latency Trading*, Promoter(s): Prof. M.A. van Dijk & Dr. D.G.J. Bongaerts, EPS-2016-379-F&A, http://repub.eur.nl/pub/93017

Zaerpour, N., *Efficient Management of Compact Storage Systems*, Promoter(s): Prof.dr.ir. M.B.M. de Koster, EPS-2013-276-LIS, http://repub.eur.nl/pub/38766

Zhang, D., *Essays in Executive Compensation*, Promoter(s): Prof.dr. I. Dittmann, EPS-2012-261-F&A, http://repub.eur.nl/pub/32344

Zwan, P.v.d., *The Entrepreneurial Process: An International Analysis of Entry and Exit*, Promoter(s): Prof.dr. A.R. Thurik & Prof.dr. P.J.F. Groenen, EPS-2011-234-ORG, http://repub.eur.nl/pub/23422

Propositions

1. For a more accurate evaluation, studies on blocking methods for entity resolution should also evaluate the proposed approaches using non-perfect matching functions. (Chapter 3)

2. Using lexical matching *and* pattern matching simultaneously improves the instantiation of ontologies from semi-structured data. (Chapter 4)

3. The performance of taxonomy mapping approaches is improved when the similarity of the nodes in a candidate taxonomy path is taken into account as part of the final similarity. (Chapter 5)

4. Ordering facets in a product search user interface lowers the user effort for drilling-down to the desired product. (Chapter 6)

5. Fuzzy product search improves the ability of users to find products for a query for which there is not an exact match. (Chapter 7)

6. In computational studies, careful replication of existing research is just as important as proposing new algorithms.

7. The value and contribution of a proposed approach is not only reflected by statistical significance.

8. Scientific competitions and standardized evaluations should be encouraged more in all fields of Computer Science.

9. For a Ph.D. candidate, being proficient in software development is both a curse and a blessing for the efficiency of the Ph.D. trajectory.

10. Internships at a large IT company such as Google teach important skills that cannot be taught at a university. Therefore, all Ph.D. candidates in Computer Science should be encouraged to do an internship at such a company.

11. The ingredients for a successful and happy Ph.D. candidate is the right balance between doing research, visiting conferences, and publishing in scientific journals.

Over the last few years, we have experienced an increase in online shopping. Consequently, there is a need for efficient and effective product search engines. The rapid growth of e-commerce, however, has also introduced some challenges. Studies show that users can get overwhelmed by the information and offerings presented online while searching for products.

In an attempt to lighten this information overload burden on consumers, there are several product search engines that aggregate product descriptions and price information from the Web and allow the user to easily query this information. Most of these search engines expect to receive the data from the participating Web shops in a specific format, which means Web shops need to transform their data more than once, as each product search engine requires a different format. Because currently most product information aggregation services rely on Web shops to send them their data, there is a big opportunity for solutions that aim to tackle this problem using a more automated approach.

This dissertation addresses key aspects of implementing such a system, including hierarchical product classification, entity resolution, ontology population and schema mapping, and lastly, the optimization of faceted user interfaces. The findings of this work show us how one can design Web product search engines that automatically aggregate product information while allowing users to perform effective and efficient queries.

ERiM

The Erasmus Research Institute of Management (ERIM) is the Research School (Onderzoekschool) in the field of management of the Erasmus University Rotterdam. The founding participants of ERIM are the Rotterdam School of Management (RSM), and the Erasmus School of Economics (ESE). ERIM was founded in 1999 and is officially accredited by the Royal Netherlands Academy of Arts and Sciences (KNAW). The research undertaken by ERIM is focused on the management of the firm in its environment, its intra- and interfirm relations, and its business processes in their interdependent connections.

The objective of ERIM is to carry out first rate research in management, and to offer an advanced doctoral programme in Research in Management. Within ERIM, over three hundred senior researchers and PhD candidates are active in the different research programmes. From a variety of academic backgrounds and expertises, the ERIM community is united in striving for excellence and working at the forefront of creating new business knowledge.

ERiM

ERIM PhD Series
Research in Management

Erasmus University Rotterdam (EUR)
Erasmus Research Institute of Management
Mandeville (T) Building
Burgemeester Oudlaan 50
3062 PA Rotterdam, The Netherlands

P.O. Box 1738
3000 DR Rotterdam, The Netherlands
T +31 10 408 1182
E info@erim.eur.nl
W www.erim.eur.nl