Tampere University

Toni Linnusmäki

Increasing e-commerce conversion rate with relevant search results using Elasticsearch

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
April 2020

Abstract

Toni Linnusmäki: Increasing e-commerce conversion rate with relevant search results using Elasticsearch
Master's thesis
Tampere University
Master's Programme in Computer Science
April 2020

Search engines have grown vastly and are now part of our everyday life by providing easier navigation on the internet. Besides, search engines are an integrated part of many websites, including e-commerce platforms. In this thesis, the focus was to increase the relevancy of the search results of an existing e-commerce platform and see how the conversion rate is affected.

Previous research has focused mostly on studying different ranking algorithms, but not many are included in the existing search engine frameworks. For this thesis, the purpose was to investigate how the methods from previous research could be utilized in an e-commerce platform with real customers. Furthermore, the concluded tests were over six weeks and included tens of thousands of customers.

The data that was collected during the testing phase of the implementation shows how the changes affect the conversion rate of the e-commerce platform. In addition to the conversion rate, the click-through rate was analyzed since it seemed to capture the changes better. Furthermore, the results include both measurements and analysis of how the different parts possibly affected the measurements.

While the results did not indicate that the conversion rate of the e-commerce platform was affected by the implementations, the results indicate that with a simple solution, the existing process could be replaced. In addition, the results indicated the need and direction for future development, which could have a further effect on the conversion rate of an e-commerce platform.

Keywords: search engine, relevant search results, e-commerce, Elasticsearch.

Contents

1 Introduction

Search engines have been a significant element of the internet for a long time, and some of the most visited websites in the world are search engine interfaces, like Google and Baidu [1]. By visiting different websites, people might interact with dozens of search engines, for example, while writing text to the address bar of a browser or looking for information about a topic of interest. In addition to interacting with search engines, people might not be aware that they are utilizing one of the most potent tools of parsing and matching text.

The history of search engines goes back decades, but the era of modern search engines can be thought to have begun when Brin and Page [3] introduced their search engine, Google. In addition, they proposed methods to index a vast number of documents, in their case web pages, and to use a ranking algorithm, PageRank, to rank the documents [3]. The usage of indexing and ranking algorithms has been the foundation for search engines ever since. While using ranking algorithms and the ranking search results by relevancy was not a new idea, it was revolutionary to utilize the PageRank ranking algorithm in search engines to provide more relevant search results to the queries of the users.

Besides handing most of the navigation on the internet, searching is a crucial part of an e-commerce platform. Sorokina and Cantu-Paz [16] introduced the usage of a search application in the largest e-commerce in the world, Amazon, and how the search application powers an extensive amount of sales of Amazon. Turnbull and Berryman [18] introduced text analysis and result ranking methods that can be efficiently utilized in a search engine to provide relevant search results to users.

While different relevancy methods and the relevancy of the search results has been investigated extensively in the past, little research has included the use of the relevancy methods in practice in search applications with real-world customers in e-commerce. The research has tended to focus on applying the methods in theory to a predetermined dataset to get measurable results as opposed to focusing on implementing the methods and measuring the performance in practice.

The primary focus of this thesis was to investigate how the relevancy of the search results affected the conversion rate of an e-commerce platform by implementing a selection of methods based on the literature. The conversion rate measures what proportion of customers who visited a website bought something, and it provided a generic measurement of how the implemented methods performed in providing relevant search results to customers. However, the relevancy of the search result is subjective since not all customers are searching for the same thing making the relevancy of a search result different for each customer.

The implementation of the methods in this thesis was done to an existing e-commerce platform, which serves tens of thousands of customers daily across the Nordic countries. Furthermore, a set of tests was performed in the platform to a set of customers by offering different search results and measuring how the implementation affected the conversion rate of the e-commerce platform. Furthermore, by making sure the set of customers was large enough and randomized, the relevancy of a different implementation could be compared against a baseline.

The significance of a search application to the conversion rate of the e-commerce platform is approximately 2.5x increase in conversion rate when customers interact with the search functionality on the website compared to customers who do not interact with the search while visiting the website [14]. On the other hand, the increase shows that a link exists between showing the search results shown to customers and the conversion rate of the e-commerce, on the other hand, the increase does not measure the relevancy of the search results.

So, depending on the size of the e-commerce organization, the increase in conversion rate can lead to a significant amount of revenue for the organization when customers use their search. Furthermore, in this thesis, the goal was to increase the conversion rate of the proportion of customers using the search application while visiting the website. Therefore, how the proportion of customers using the search can be increased was out of the scope of this thesis.

The current search application of the e-commerce platform uses Elasticsearch, which was the platform that the methods used in this thesis were implemented on. Elasticsearch is a distributed search and analytics engine that provides real-time search and text analysis tools in addition to document storage and indexing capabilities [4]. Turnbull and Berryman [18] introduced how to build a production scale search engine with Elasticsearch in their book. While building a large-scale search engine is out of the scope of this study, a selection of the text analyzing and search result ranking methods Turnbull and Berryman [18] proposed were utilized in this thesis. Therefore, understanding how the search engine performs in large scale search applications is crucial before implementing the methods from the literature in practice.

The tests for this thesis were done as A/B tests in a production environment to capture the effects that the different implementations had on the conversion rate and the click-through rate. However, tests performed for a few weeks might not have captured the full effect of this thesis since more relevant search results might have a lasting effect on the reputation of the organization, and the increase of a reputation is hard to measure with these kinds of tests. Generally, the more satisfied customers are with their experience on a website, the more likely they are to return to revisit the website.

The results showed that while the click-through rate of the search results was increased, it did not show a consistent increase in the conversion rate. Furthermore, indicating that while the relevancy of the search results was increased, the conversion rate was not affected by the change in relevancy. While the conversion rate was not consistently increased by the different implementations, they proved a need for future development and provided a direction for it.

2 Background

The following chapter acts as a detailed introduction of the necessary background from literature. First, different measurements that are utilized are introduced, both of which are generally used in e-commerce to measure success. In addition to the measurements, the section introduces the testing platform utilized to collect the measurements and to analyze the test results.

Second, the chapter introduces a search engine first at a more general level before introducing Elasticsearch, the search engine utilized in this thesis, in more detail. Lastly, the chapter introduces the ranking algorithms that served as a basis for the ones tested. In addition to the ranking functions, the chapter introduces what the relevancy of the search results means as the purpose is to try to increase the conversion rate by offering more relevant search results to users.

2.1 Measurements in e-commerce

The most popular metric to measure the success of an e-commerce platform is a metric called conversion rate, which provides a percentage of conversions compared to the total visitor count. Despite measuring the performance of a website accurately, a drawback of conversion rate is that a great deal of data is necessary for it to be an accurate measurement [2]. In addition, the overall conversion rate measures small changes poorly, since it measures the performance of the whole website, so additional measurements are necessary to measure more specific changes.

For instance, to measure the performance of a specific element on a website, the usage of the conversion rate might be redundant. Moreover, the specific element might not have a noticeable effect on it, thus making the conversion rate inaccurate measurement for that situation. For specific situations, there exists a metric called click-through rate (CTR), which can measure the performance a small change efficiently [12].

The following sections provide an overview of the different measurements used in e-commerce relevant to this thesis. First, Section 2.1.1 talks about the conversion rate in e-commerce, and second, Section 2.1.2 describes a metric called click-through rate. Lastly, Section 2.1.3 provides an overview of the test platform that was used

when collecting the results of the implementations.

2.1.1 Conversion rate

The overall conversion rate measures the proportion of orders to the website visitors [8]. For example, if a total of one hundred customers visit a website and two of them place orders, the conversion rate can be calculated with a formula

$$conversions \ / \ total \ visitors \ * \ 100\% = conversion \ rate$$

resulting in a conversion rate of 2%.

Although the conversion rate is not commonly the most accurate metric to be measured, it generally is the most significant for an e-commerce organization. The preceding example was with 100 visitors, which is a far too low number to measure conversion rate accurately [2]. Furthermore, in the example, one more converting customer would increase the conversion rate by 50%, making the measurement highly inaccurate.

It is important to note that there are other types of conversion rates in e-commerce [8]. The above conversion rate calculates the overall conversion rate of an e-commerce platform. In addition to placing an order, the conversions can be, for instance, form submissions, signing up for a subscription or events that product is added to cart [2]. Depending on the situation, other types of conversion measurements might be necessary to capture effects that might not be shown by the overall conversion rate. Furthermore, the same formula applies to the different types of conversion rates.

In this thesis, the measured conversion rate was the overall conversion rate of the e-commerce platform and the conversion rate of the proportion of customers who interacted with the search application. Furthermore, the conversion rates were used as a metric to measure the performance of the search applications compared to the baseline.

2.1.2 Click-through rate

Click-through rate (CTR) is a measurement used mostly to measure the performance of a specific element on a website, for example, an advertisement [12]. CTR is calculated similarly to the conversion rate, by dividing the conversions, which are the clicks on the element, by impressions, which means the number of visitors, who have seen the element [12].

While CTR measures the clicks on a single element accurately, the usage differs based on the situation. Generally, high CTR means that the element is highly

relevant. However, for instance, in an advertisement, ambiguous keywords might receive high CTR while not necessarily being relevant. [12]

As mentioned above, the CTR can measure more specific changes in the element. Therefore, it was utilized to measure the performance of the search result list with different implementations. Furthermore, to be more precise, to collect the number of clicks on the search results and the position, which product was clicked on. Since in each implementation, the product list is different, the CTR was able to capture the difference between implementations well and provide insight to the results.

2.1.3 Measurements with production A/B testing

The measurements were an essential part of determining the performance of the introduced methods. Furthermore, the measurements were collected by utilizing the Google Optimize platform to conclude A/B tests to calculate the measurements during the tests from the collected data. An A/B test is a randomized test where the test platform splits the website, or part of it, into two or more variants, and the platform redirects the traffic for the website into the different variants at random [7]. Therefore, when testing the different implementations, the user interface of the search that the customer sees does not change. Only the search results presented to the customer differ.

Google Optimize is a test environment, but in addition to operating the tests, it provides a tool to see a rough estimate of the results of the tests, based on the goals defined for the test, without extensive analysis [7]. The tool can give a quick estimate of the results, which is useful in determining if the implementation was performing as expected. Furthermore, the goals of the tests utilized in this thesis are introduced later in Section 4.2.

In order to reliably infer if the concluded tests have increased the conversion rate, a detailed analysis of the results of the tests was necessary. Therefore, Google Analytics was utilized for this since it provides a detailed report about the test variants based on the metadata collected on the website [6]. The organization already collects data about the performance of products and product lists, so utilizing the existing platform was a logical choice.

Generally, the metadata includes the click data when a customer browses the website, the purchase data, and the data, which products are shown on different pages to different customers. The collected metadata can be utilized in both the configuration and the analysis of the tests by setting up custom goals for tests. Therefore, the completion percentage of the custom goal during a test can be analyzed. The primary goals for e-commerce, for example, revenue and conversion, can be tracked with the completion of the custom goals. [6]

The above sections introduced the metrics used in this thesis and how the measurements are collected. Firstly, the conversion rate served as the primary measurement used in this thesis since it is generally the most utilized measurement in e-commerce. Secondly, the click-through rate was introduced, which was utilized to measure more specific changes. In this case, the performance of the search result list. Lastly, A/B tests were introduced and how they work and use the measurements to determine the best result from the test variants. The following sections move on to introduced search engines first on a more general level before introducing Elasticsearch, the search engine platform used, in more detail.

2.2 Utilization of a search engine

As a general term, a search engine means an application that performs a query from a user and presents the user with the results for the specified query. Furthermore, the era of modern search engines can be thought to have started when Brin and Page [3] created Google, now the largest search engine. Furthermore, Google has been a large part of making search engines part of our everyday life. To further understand how a search engine is utilized in this thesis, this section provides an overview in necessary detail. In addition, this thesis uses Elasticsearch as the search engine; it shares a considerable amount of similarities to the early search engine of Google introduced by Brin and Page [3]. Elasticsearch is introduced in detail in Section 2.3, but the general background of the search engines applies to it.

2.2.1 Overview

Generally, search engines must be able to search from extensive datasets efficiently. A design goal of Brin and Page [3] in 1998 was to build a large-scale search engine that would be able to index hundreds of gigabytes of data, in addition to performing hundreds of millions of queries per day. The above numbers have grown exponentially in the last two decades, and a present-day search engine might need to perform the same amount of queries every hour.

Searching from persistent storage or disk was not feasible, and a more efficient solution was necessary [3]. Thus, the design goals of Brin and Page [3] included the creation of a distributed indexing system, which they could efficiently store the documents. In addition, the indexing system needed to be able to process hundreds of gigabytes of data efficiently due to the queries from the users came in with a rate of hundreds to thousands in a second [3]. Furthermore, the preceding design goal set by Brin and Page [3] over two decades ago, describes an outline for many of the modern search engines, including Elasticsearch [18].

A search engine is a complex application to do simple string matching efficiently

on a large-scale. However, to perform in a simple task efficiently, a search engine requires a great deal of engineering work. In some cases, a simple spelling mistake can result in an empty set of documents, thus creating frustration towards the search engine. [18]

To simplify the complex application, Gan and Suel [5] broke down a web search engine to four different steps:

1. Collecting the data from a source.
2. Preprocessing the data to a specified format.
3. Indexing the documents and creating an inverted index to the search engine.
4. Processing queries and searching from the index.

Despite describing a web search engine, the preceding steps can be applied to various search applications. First, during the data collection, the search application in this thesis collects the data from persistent storage, which includes multiple databases and database tables. Second, since the data includes numerous unnecessary fields and only a selection of them is necessary, the collected data must be preprocessed. Third, the previous step yields a document with a specified schema from the data, that is indexed with Elasticsearch Index APIs. Indexing both stores the document and creates an inverted index from the document to the search engine. Lastly, the queries are processed and sent to the search engine. Furthermore, the search engine performs a search against the inverted index and yields the results for the query.

While the four steps describe a search engine in more detail, Turnbull and Berryman [18] stated that a search engine consists of two major parts, the index time and the search time. Moreover, the first three steps are considered as the index time, which is described in Section 2.2.2, and the fourth step is the search time, described in Section 2.2.4.

2.2.2 Creating the inverted index

The inverted index is a central data structure of a search engine enabling matching query terms to the documents swiftly. In worst cases, the fields in documents can contain an extensive amount of text, so finding a match from the documents can be costly. The inverted index is created to solve the problem of going through all the documents every time a query must be performed. [18]

The inverted index consists of two crucial parts a terms dictionary and a postings list [18]. Turnbull and Berryman [18] introduced the following listings describing how the inverted index is created in Elasticsearch. The listing 2.1 includes the example documents, which are indexed to a search engine.

```
0. One shoe, two shoe, the red shoe, the blue shoe.
1. The blue dress shoe is the best shoe.
2. The best dress is the one red dress.
```

Listing 2.1 *Example text documents with identifiers [18].*

During indexing, a term dictionary (2.2) is created by collecting all individual terms called tokens from the documents. Furthermore, each of the terms has an identifier, which maps the term to the posting list (2.3).

```
best  → 0      red   → 5
blue  → 1      shoe  → 6
dress → 2      the   → 7
is    → 3      two   → 8
one   → 4
```

Listing 2.2 *Term dictionary is generated from terms in the documents [18].*

```
0 → [1,2]      5 → [0,2]
1 → [0,1]      6 → [0,1]
2 → [1,2]      7 → [0,1,2]
3 → [1,2]      8 → [0]
4 → [0,2]
```

Listing 2.3 *Postings list is a mapping between the term dictionary and documents [18].*

In addition, the inverted index in Elasticsearch contains additional metadata necessary during the query evaluation [18]. Turnbull and Berryman [18] provided a full list of the metadata, and the following list introduces a selection of the metadata from the list necessary for this thesis.

- *Document frequency* is a count of documents that contain the term, thus establishing a notion of the importance of the term. Generally, the higher the document frequency indicates that the term provides little value when determining the relevancy of the document to a query. For instance, the term "the" has a high document frequency providing little value for the relevancy calculation.

- *Term frequency* determines how many times the term occurs in a document providing a notion of how well a query matches the document. In a nutshell, document 0 contains the term "shoe" four times and document 1 two times, making the document 0 twice as important match for the query term.

- *Term positions* provide the position where the term occurs in a document. It makes finding documents with phrase matching possible.
- *Term offsets* keep track of the start and end character offsets. For example, Google search provides highlights of the matches in the search result listing, which keeping track of the term offsets enables. The highlights provide feedback to the user on why the result matched the query.
- *Stored fields* contain the necessary fields used during searching. The fields in the inverted index are scrambled versions of the original document, and all fields presented back to the user must be separately saved to the stored fields. Therefore, the fields might require a great deal of disk space depending on the size of the fields.
- *Doc values* usually contain auxiliary values relevant to scoring the search results. For instance, the values used in sorting or boosting the results are stored in the doc values.

The above section provided an overview of the inverted index, the primary data structure of the search engine. In addition, the data saved to the inverted index was introduced, as it is referenced in later chapters. The following section continues to describe what index time means and how the inverted index is created in index time.

2.2.3 Index time

The index time can be viewed as an extracting, transforming, and loading information (ETL) pipeline [18]. Furthermore, ETL pipelines are referred to when moving data from storage to another while processing it in between [18]. In this thesis, the ETL pipeline steps for moving the data from persistent storage to the search engine are shown in Figure 2.1.

The steps of the pipeline are extraction, which collects the product data from the persistent storage. After the collection, the data is enriched with the additional data necessary to be searched. Furthermore, Figure 2.1 shows the description of the product and impressions, which is the number of times the product has been viewed, both utilized in searching. The former, when matching the search query to a product, the latter, by the ranking algorithm in the relevancy calculation.

The third step is analysis, which in Elasticsearch happens during the indexing. Before the metadata described in Section 2.2.2 is collected from the document, a set of analyzers are utilized to create better terms [4]. For instance, Figure 2.1 shows that all text from the document is lowercased, and the possessive ending from the title of Document 1 is removed [18]. Elasticsearch does this by default to normalize

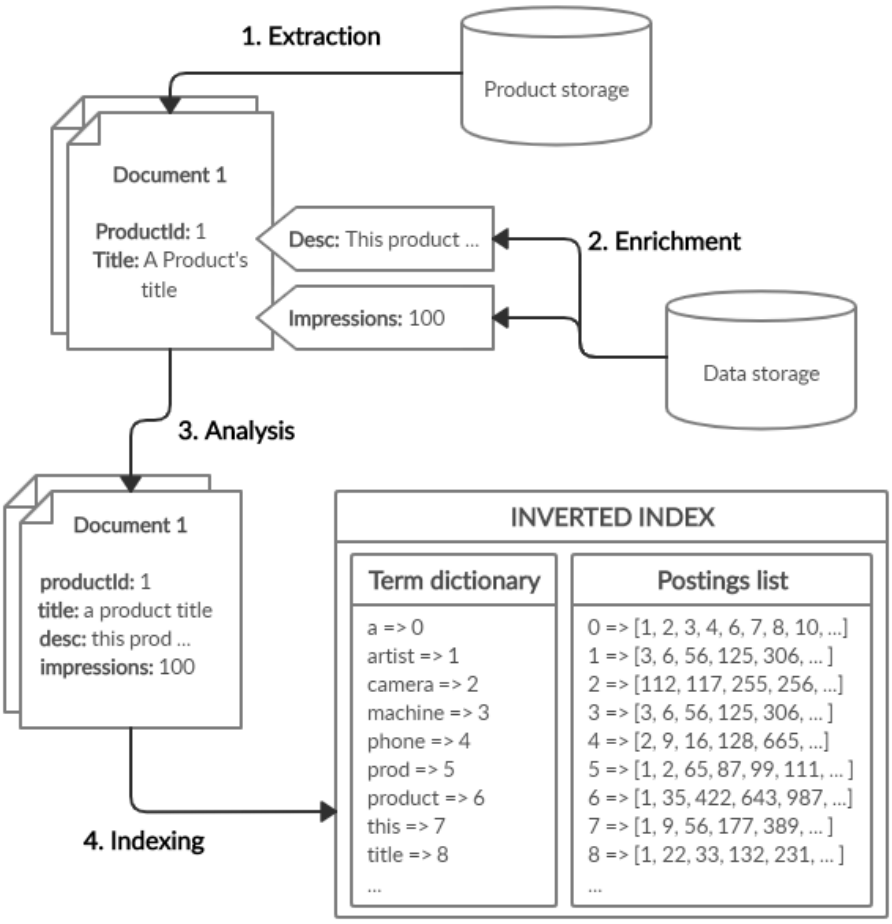


Figure 2.1 *Index time described as an ETL pipeline [18].*

the search terms to make the documents easier to find [4]. In an ideal case, after the analysis step, a term with plural should match the term without one, thus, making both documents match a query with the term. This thesis utilizes a set of analyzers to generate more generic search terms for specific fields. Furthermore, the analyzers used are introduced in Section 2.3.1.

The last step of the ETL pipeline is to index the analyzed documents. Furthermore, indexing is the process that saves the data to the inverted index and the storage of the search engine [18]. Generally, the focus during indexing is on the computation performance and resource management since the more the fields from the documents are analyzed, the slower the indexing becomes requiring more storage computation power and storage [18]. In Elasticsearch, the analysis and the indexing happen automatically when a document or a set of documents is posted to the Index API [4].

Searching from the field becomes available only when the field is indexed. Consequently, making the decision between which fields to index, which to store, or which fields both index and store. Storing the fields refers to saving original content to the metadata data structure introduced in Section 2.2.2. [18]

As mentioned above, the utilization of a search engine can be divided into two parts, index time and search time. The above section introduces what index time is and how the inverted index is created. Furthermore, the following section continues to introduce what search time is and how the documents are searched from the index.

2.2.4 Searching from the index

A search engine is not an intelligent machine; instead, it is merely a text-matching tool [18]. When a query is submitted to the search engine, it matches the individual terms from the query to the tokens in the inverted index, which consequently map to the documents in the index [18]. Inverted index functions like the index from the end of books, where the reader can look for a specific topic and find the relevant page number [18]. Searching for a specific subject from a book by going through page by page is not efficient. Instead, the relevant pages can generally be found from the index much faster. Furthermore, the same applies to the search engines; the abundance of data generally makes searching the documents one by one unfeasible.

After a term matches a document, a content score for the match is calculated based on the metadata values in the index [18]. The content score determines how good of a match the document is to the query. In addition to the content score, additional business scores can be calculated for the document to provide additional measurements of relevancy. Furthermore, the search engine sorts the resulting list in an order specified in the query, and generally, by default, the score determines the order of the resulting list. Thus, yielding in a list sorted by the relevancy of the search result.

While the previous section provided an overview of the basic functionalities of a search engine, the following section discusses how Elasticsearch works and how the functionalities are utilized in this thesis. First, the section provides an overview of Elasticsearch, before moving on the various text analysis utilized when implementing the new search functionalities to the current search application. Lastly, the section describes how the results are searched and scored in Elasticsearch before they are returned to the user.

2.3 Elasticsearch as a search engine

Elasticsearch provides real-time search and analytics for all types of data. Furthermore, the data is stored and indexed in a way that fast searches are supported. In addition, Elasticsearch provides a possibility to go far beyond simple information retrieval and supports different aggregations to fetch data efficiently. The distributed nature of Elasticsearch provides additional resiliency when the volume of the data

grows. Thus, providing an excellent platform to build a large-scale search application. [4]

The power of Elasticsearch comes from the optimally indexed data structures; for instance, text fields are by default indexed as inverted indices with the tokens collected from the fields of the documents. Additionally, Elasticsearch enables saving the data to different kinds of data structures depending on the use cases. For example, an analyzer can be applied to process data from a specific field, which creates an own data structure for the field with the data collected by the analyzer. [4]

The following part of this thesis moves on to describe in greater detail the text analysis tools provided by Elasticsearch utilized in this thesis. Furthermore, understanding the tools is necessary because they are used in the methods introduced in Section 4.

2.3.1 Text analysis

Text analysis in Elasticsearch means transforming text to tokens to the inverted index. Furthermore, choosing how the text is transformed determines the performance of the search application. During the analysis, the search engine converts the data from the documents into tokens using analyzers and stores the tokens in the internal data structures of the search engine. Besides utilizing analyzers to the documents while indexing, optionally, the queries can be analyzed with the same analyzers. [18]

As previously stated, search engines are merely trying to find exact matches the query from the inverted index. So, without good tokens in the inverted index, the document can be challenging to find with a query that might not exactly match the document. Thus, to make the documents more findable, Elasticsearch provides various analyzers by default in addition to the capability to create custom analyzers from different tokenizers and token filters [18].

In some languages, like Finnish, there are various inflected forms for a word, which can make matching the document to the query more difficult for the search. So the analyzers can help with making the search term as basic form as possible [18]. Since, generally, in e-commerce, the users might not use the most effective search terms to describe what they are searching for, instead of using technical terms, they use familiar terms. Furthermore, interacting with a search application that does not understand what is being asked from it can make users frustrated and might negatively affect their experience on the website [18].

In summary, the creation of the tokens from the documents is a crucial part when creating a relevance-based search application. So, the following part focuses on why the token creation is an essential step in search engines and how tokens are generated

with analyzers provided by Elasticsearch relevant to this thesis. Additionally, a selection of the tokenizers and token filters is introduced since a set of custom analyzers had to be created to provide the functionality necessary for this thesis.

Extracting features from text

While token generation is a crucial part of a search application, it is essential to note that the tokens should not merely be the words from the text. Since the tokens are the parts that the search engine is going through when a query is received, the tokens should try to capture the meaning of the text instead of the text itself to make query matching more robust. Furthermore, tokens should try to anticipate the expectations of the user, which can lead to highly relevant and targeted results. [18]

Turnbull and Berryman [18] described the idea of feature modeling in their book. In feature modeling, the purpose is to take into consideration the intent of the user, in addition to the ideas conveyed in the indexed documents. Additionally, feature modeling ensures that the generated tokens represent the descriptive and meaningful features of the queries making understanding the information in the documents as well as the intent of the users. For instance, useful features take into account the pluralization and the parts of speech in addition to the specific language used in the domain context to make the query and the tokens match. The feature modeling is done both in index and search time since matching with features that are consistently generated becomes more straightforward. Furthermore, when the features capture both the meaning of the document and the query well, the matches become increasingly relevant. [18]

The feature modeling, among many other optimizations tasks, is a constant battle between precision and recall [18]. In a search application context, precision describes the percentage of documents in the results that are relevant to a query, and recall means the percentage of the relevant documents [18]. Precision is calculated with the following formula

$$precision = doc_{relevant}/(doc_{relevant} + doc_{irrelevant})$$

where the $doc_{relevant}$ describes the number of relevant documents in results, and $doc_{irrelevant}$ the number of irrelevant documents in the matches. Moreover, recall can be calculated with the formula

$$precision = doc_{relevant}/(doc_{relevant} + doc_{relevant_{nomatch}})$$

where the $doc_{relevant}$ describes the number of relevant documents in results, and

$doc_{relevant_{nomatch}}$ is the total number of relevant documents that do not match the query. Generally, higher precision means lower recall, which implies there is an optimal threshold that could be achieved with optimization. In an ideal case, both precision and recall would be 100%, but that is nearly impossible [18].

In this thesis, the trade-off between precision and recall appeared in one of the empirical tests chosen to measure the relevancy in the development process. Furthermore, the website has three distinct product categories that are standard washing machines, washing machines with dryers, and dishwashers. In Finnish, all terms *pyykinpesukone*, *kuivaava pesukone* and *astianpesukone* are separate product categories, and the query "pesukone" was quite common. However, most certainly, the major part of Finnish speakers would tell that "pesukone" does not mean *astianpesukone*, and most likely, the best match would be *pyykinpesukone*, or at least it should be included near the very top of the results. Therefore, teaching a search engine to know that the previous query should not only match the exact word found from the inverted index but to one word which does not start with the query itself can swiftly become an optimization problem.

In summary, this section introduced the importance of feature modeling in analysis and how it can be found in practice by giving an example of a problem found during development. Generally, similar problems during development are not easily solvable in practice but somewhat tedious and time-consuming to solve and might need additional feature extraction. Furthermore, the following sections introduce analyzers, which are the tools responsible for collecting the tokens from the documents.

Analyzers

In Elasticsearch, an analyzer is used in the text analysis step to extract tokens from the documents [4]. The usage of the analyzer is configurable, and multiple analyzers can be used for each field [18]. Even though there can be multiple analyzers defined for a field, only one can be utilized at a time because each of the analyzers creates a new inverted index from the field [18].

A drawback when creating analyzers is that configuring them requires the index re-creation, making changing the analyzers on an existing application complicated. For instance, if a search application uses index A, then a new index B must be created, and all documents must be re-indexed, after which the search application can be told to use the new index B. Furthermore, this can cause issues in some data-sensitive applications because of the index creation, and data replication is not instantaneous.

Analyzers allow normalization of tokens to create common representations to overcome the limitations of the byte-by-byte matching ability of the inverted index

[18]. Furthermore, analysis enables expressing the essential features of the search application as tokens [18]. The analyzers consist of tokenizers and token filters, and Elasticsearch provides a selection of built-in analyzers [4]. The following part introduced the built-in analyzers utilized in this thesis. However, a custom analyzer can be created and can be useful in specific situations in extracting the tokens from a field. For instance, parts from an email address can be extracted to create tokens from it, which can make searching for the email more efficient [18].

First, the standard analyzer is the default analyzer used by Elasticsearch, and it is used if none other is specified and is typically a first analyzer used in a search application [4, 18]. Furthermore, the standard analyzer divides input text into tokens on word boundaries, defined by the Unicode Text Segmentation (UTS) algorithm, and works well for most languages [4]. Besides creating tokens from the input, the analyzer converts all the tokens to lowercase [4]. Typically, the standard analyzer results have high precision, but low recall since it does not capture features of the text well, and user query must match the document exactly [18].

Secondly, the language analyzers work similarly to the standard analyzers. However, there are some minor differences between the languages provided by Elasticsearch by default. For instance, the English analyzer will convert the following sentence:

```
The QUICK brown foxes jumped over the lazy dog!
```

Listing 2.4 *Input for the English analyzer [4]*

into distinct tokens, convert all text to lowercase, remove the stop words ("the"), and additionally stem the words (foxes → fox, jumped → jump, lazy → lazi). The output of the analyzer is the following terms that are added to the inverted index: [4]

```
[ quick, brown, fox, jump, over, lazi, dog ]
```

Listing 2.5 *Output of the English analyzer [4]*

As shown in the example above, the language analyzers try to capture the meaning of the words and carry the meaning to the inverted index [18]. As previously stated, a search engine is merely a token-matching system, and it does not know about the meaning of the tokens, using analyzers to try to capture the meaning and the intent of the user, the token matching becomes more straightforward for the search engine [18]. However, the built-in analyzers might not be enough for specific application domains or might perform poorly on some languages. So, the following part introduces a selection of the tokenizers and token filters built-it in Elasticsearch, which are used to build custom analyzers to target specific issues.

Tokenizers

Tokenizers are the part of the analyzers that create the tokens from input words. A tokenizer receives a stream of characters and dismantles them to tokens and outputs a stream of tokens [4]. In addition, tokenizers are responsible for collecting the position of each term, and the start and the end character offsets of the original word that the term represents [4], so it could be saved to the inverted index as introduced in Section 2.2.2. The collected position is used in a phrase or word proximity queries, and the start and end character offsets are utilized in highlighting the matches in the search results [4]. The following part introduces the two main tokenizers, a standard tokenizer and an edge-N-gram tokenizer, both utilized in this thesis.

As explained earlier, the standard and language analyzers divide the words into tokens, both utilize standard tokenizer in doing that. Rather than creating the tokens, the analyzers utilize tokenizers and token filters in token creation. The standard tokenizer uses the UTS algorithm to divide the text, and also it removes punctuation symbols from the input. Furthermore, the standard tokenizer works well in most languages, and it is used as the default in Elasticsearch. [4]

The second tokenizer utilized is edge-N-gram tokenizer, that first divides the text into words when encountering specified characters, for instance, whitespace or punctuation. Then the tokenizer returns an N-gram of the single word anchored to the start of the word. Furthermore, N means the max amount of characters in the resulting output of the tokenizer. For instance, Listing 2.6 shows the input to the edge-N-gram tokenizer, and Listing 2.7 shows the output with $N = 5$, where the words are divided into tokens with a minimum of two characters and a maximum of five characters.

```
Quick Foxes.
```

Listing 2.6 *Input to the edge-N-gram tokenizer [4]*

```
[ Qu, Qui, Quic, Quick, Fo, Fox, Foxe, Foxes ]
```

Listing 2.7 *Output from the edge-N-gram tokenizer [4]*

Elasticsearch allows the minimum and maximum characters N to be configured along with the characters that should be included in the generated tokens. [4]

As was mentioned earlier, the built-in language analyzers perform poorly on some languages, which became a problem during the development of the new search functionalities. The Finnish language analyzer had a hard time capturing some of the plural forms in the Finnish language. For instance, capturing the features from the words *langaton* and *langattomat* was difficult, because of the addition ‘t‘ character added in the plural form of the word wireless.

In this thesis, an edge-N-gram tokenizer was utilized in generating the tokens from text to solve the described problem, which purpose was to mimic the behavior of the language analyzer and try to capture the meaning of the words by capturing the start characters. While the usage of edge-N-gram tokenizer increased the recall of the search results, it quickly decreased the precision since if the minimum length for a token were defined too low, the results would include almost all products. Before the generated tokens are ready and could be added to the inverted index, a set of token filters should be applied, which are introduced in the following part. They are an essential part of the analysis process since the tokenizers are not enough to generate consistent tokens to the inverted index. Since tokenizers do not modify the words, only divide them, the purpose of token filters is to create consistent tokens, which make the documents easier to find.

Token filters

The purpose of a token filter is to modify the stream of tokens output by a tokenizer. Furthermore, Elasticsearch provides various built-in token filters. The following part introduces a selection of the token filters, introduced in the documentation of Elasticsearch [4], relevant to this thesis and explains why those were utilized.

- **Lowercase token filter** modifies the stream of tokens and converts the characters to lowercase, making the tokens consistent and enabled case insensitive queries. A lowercase filter is in the core of every built-in analyzer Elasticsearch offers.

- **ASCII folding token filter** converts the alphabetic, numeric, and symbolic Unicode characters to ASCII characters if one exists. Utilizing it allowed the queries to match the words regardless if 'a' or 'ä' character was used. Generally, all search engines utilize a similar functionality to match documents regardless of the special characters.

- **Edge-N-gram token filter** implements the same functionality as the edge-N-gram tokenizer. The reason for using the token filter is if additional filters would be needed to be applied before the tokens are split to N-grams. For instance, the edge-N-gram from the end of the word must utilize the token filter, since the words must be reversed with a filter before generating the N-gram.

- **Conditional token filter** uses a condition and a list of subfilters and applies the subfilters to the token only if the token matches the condition; otherwise, returning the original token. For this thesis, a search time edge-N-gram filter was created, which used a condition to determine the length of the search

term. Furthermore, if the search term is longer than the average length search term, which was 6, a subfilter would apply an edge-N-gram token filter to the term.

- **Stemmer token filter** is used in the language analyzers to try to capture the meaning of the word by trying to collapse multiple forms of the same words into one. Furthermore, the token filter is available in multiple languages.
- **Stop token filter** uses a predefined set of stop words to filter out from the tokens. For instance, the words "and" and "the" do not bring meaning to the search since they are too common. So, a token filter removes those altogether from the inverted index to make it more compact.
- **Unique token filter** is used to remove the duplicate tokens from the token stream. So, if a text has multiple duplicate terms, which bring no value, then this filter could be applied. However, the usage of this token filter might affect the scoring of the documents, so this should be used only to bring additional tokens to the index from the fields, where duplicates are irrelevant.

This section has provided a detailed overview of the text analysis tools relevant to the thesis. The tools introduced here were used to build a custom analyzer that was designed to capture the features from the words similar to the language analyzer, but additionally to generate smaller tokens to make the matching easier. In addition to the usage of the text analysis tools in search time, the section that follows introduces the necessary search functionalities of Elasticsearch.

2.3.2 Searching with Elasticsearch

In addition to the text analysis tools, Elasticsearch supports complex queries, additionally providing analyzing capabilities to the queries [4]. The following part introduces briefly how the queries can be used to collect metadata about the results and what is the difference between the filter and the query context, before continuing to more specific parts of the search time.

The metadata about the results that match the query is collected with aggregations, also known as facets [4]. Furthermore, the typical use case for aggregations is the filters in lists on websites. For instance, generally, most websites have filters in the lists with products, like search result pages [18]. Furthermore, aggregations allow the query to collect data about the matching documents while searching the results from the inverted index without returning all the results to the end-user [18]. For instance, the number of products in individual categories can be collected and constructed to a search filter shown on the search result page. While, the aggregations are out of the scope of this thesis, since the current search application already

utilizes them, Turnbull and Berryman [18] recommended that every website should have a selection of filters in product lists. Furthermore, the filters provide useful feedback to the user about the results of the submitted query [18].

While querying in Elasticsearch, there are two different contexts, filter, and query. The first, filter context, answers the question, whether the document matches a query by returning a boolean value, which is used to filter out the documents from the results [4]. The other, query context, answers the question, how well does the document match a query, providing a relevance score about the match [4]. Furthermore, the relevance score, also known as the content score, is used by scoring functions to reflect the score for the content, and additional business scores can be added to it [4].

So, requirements a document must fulfill are essential to add as filters to the query; otherwise, the score can affect the end score. For instance, the current search application has multiple rules that a product must match before it should be in the search results; mostly, the rules define if a product is active. If the rules are constructed into the query rather than as a filter, it affects the relevance score, distorting the actual relevance score of the documents, which in some cases can lead to results that are not scored as they would typically be depending on the underlying rules.

While this overview provided a brief introduction to how searching works in Elasticsearch, the following part provides additional detail on the necessary subjects. Firstly, the query syntax differs from the syntax, traditionally used in information retrieval, secondly, how the text analysis tools introduced in Section 2.3.1. Lastly, the different scoring functions built-in to Elasticsearch and how those were utilized in this thesis to create a custom ranking algorithm for the search application.

Query syntax

Searching in Elasticsearch happens through the Query API, and under the hood, Elasticsearch uses Lucene [4]. Lucene is a search engine software library developed by Apache. Furthermore, the Lucene supports various types of data collection techniques in addition to the basic queries from Information Retrieval (IR) [18].

For instance, the basic boolean query operators (AND/OR/NOT) from IR in Lucene is implemented as Lucene operators (MUST/SHOULD/NOT). While the basic functionalities are like the basic boolean operators, the Lucene operators provide more flexibility than the basic methods. Because the individual operators describe the operation for individual terms instead of the relationship between the terms, the query with basic boolean operators is generally more complex than the same query with Lucene operators. [18]

Listing 2.8 shows a query with Lucene query syntax, where a matching document

must contain a term *cat*, should contain a term *black*, and must not contain a term *dog*. While the same query from Listing 2.8 can also be implemented with the basic boolean operators, Listing 2.9 shows that it becomes more complex and is not as readable as with the Lucene operators. [18]

```
black +cat dog
```

Listing 2.8 *Example query with the Lucene operators [18]*

```
(cat OR (black AND cat)) AND NOT dog
```

Listing 2.9 *Example query with the basic boolean operators [18]*

From a relevancy standpoint, the MUST operator provides more relevant results to the query yielding in documents that match all parts of the query [18]. However, the operator also would not return any results if a document does not contain all terms present in the query [18]. Since, showing search pages with zero results to customers in an e-commerce application should be avoided [13], the SHOULD operator might be more applicable in some cases, since it provides resiliency against spelling mistakes. Currently, the search application only utilizes MUST operators, so in this thesis, a test with SHOULD operators was concluded to confirm the presumption that the MUST provides more relevant search results in the current e-commerce search application.

Matching search terms to tokens

When a query is submitted, the same analyzers are usually applied to the search terms as were applied when the index was created, making the tokens consistent [18]. Analyzers used by default, both in index time and in search time, are standard analyzers for text fields, but the analyzer that should be used for a specific field can be changed when constructing the query. [4]. By default, if an analyzer is specified, the same analyzer is used both in index time and in query time. However, a different search time analyzer can be defined when creating the index, making the search engine use different analyzer for the search term as used while indexing the document [18].

While the reason for using the same analyzers is to keep the token generation consistent [18], in this thesis, a different search time analyzer was necessary for the custom analyzer to restrict the generation of the N-grams in the search terms. Most of the problems caused by using the language analyzers were that they did not capture the features from longer than average search terms well. So, the custom N-grams were only generated from longer terms in search time, while allowing the generation of smaller tokens in index time. Furthermore, when the N-grams were

generated from shorter words, the recall of the search increased. However, the precision of the search results decreased significantly as more irrelevant matches were included.

While the indexing has generated tokens to the inverted index, and no matter how good the feature modeling has been still without special handling of the search terms, it can only be utilized with full search terms. To enable the search-as-you-type feature or to search with incomplete search terms, a wildcard search is a useful tool [18]. It provides the possibility to add a '*'-sign to the end or the start of the search term, which matches one or more terms and enables the search with incomplete search terms or additionally can help to solve search problems like the one introduced in Section 2.3.1.

However, the overuse of wildcards can also introduce a relevancy problem, by increasing the recall and decreasing the precision, especially when wildcards are added to both ends of a shorter search term. The current solution used added a wildcard to the start and end of each term, which turned out to be necessary due to the current limitations of the content like the example in Section 2.3.1. So, to solve the problem of providing low precision, an implementation that boosts exact matches by adding a score to the matches that exactly matches the query. Furthermore, resulting in ranking the more relevant content closer to the top, thus, increasing the relevancy of the top search results.

In addition to the wildcard search, another functionality, which enabled the searching with incomplete terms, is fuzzy search [18]. Fuzzy search matches the tokens that are similar to the search term [4]. Furthermore, the similarity is calculated by calculating the one-character changes necessary to match the search term to a token in the inverted index [4]. To create matches fuzzy search creates a set of all possible variations from the search term within specified edit distance [4]. For instance, the variations include changing, removing, or inserting characters, or transposing two adjacent characters [4].

While the fuzzy search is designed to fix the typing errors of the search terms, there are other ways to factor in the typing errors, like the search term suggestion, which is included in this thesis instead of the fuzzy search. During the implementation, the fuzzy search was investigated. However, a similar problem with precision and recall arose as did with the edge-N-gram. If the fuzziness was included in shorter search terms, due to the static edit distance, too many irrelevant results were found. In addition, the fuzzy search cannot be used with the wildcard search [4], which the current solution relied on heavily. So, the fuzzy search was replaced with term suggestion, when no results were found, due to the time limitations caused by the above problem. The result of the term suggestion implementation showed that suggesting and automatically searching with a term was effective. Therefore, indicating that

the fuzziness might be beneficial and should be investigated further and probably included and tested in the future.

In summary, the above part introduced in necessary detail how the tokens used to find a match are generated and manipulated. While providing more relevant to the top, there is still a need to add business scores to the scoring of the documents. Thus, the following part introduces scoring functions that are provided by Elasticsearch and how this thesis utilized them to provide relevant results to the users.

Scoring documents

As a base in the content score calculation in Elasticsearch is the following formula

$$term\ frequency\ /\ inverse\ document\ frequency$$

to calculate how well a term matches a document [18]. Furthermore, both values are already collected to the inverted index during indexing, as introduced in Section 2.2.2. Optionally, the base calculation can be changed, but in this thesis, it was not necessary, so introducing it in more detail is not necessary.

Turnbull and Berryman [18] argued that relevant search results in the e-commerce context must not only meet the needs of the user but meet the needs of the business. Furthermore, to achieve this, a business score must be added to the document, in addition to the content score [18]. To apply additional scores to the documents, Elasticsearch provides built-in scoring functions that can be applied to create compound queries [4]. Moreover, the compound queries wrap other queries inside them to either combine the resulting scores or to switch from the query to the filter context [4].

This thesis utilizes two types of compound queries that are boolean queries and function score queries. First, the boolean queries are the default query type for combining multiple queries with the query clauses, *must*, *should*, *must_not*, and *filter* [4]. For instance, each of the query clauses is used in the current rules about product activity, mentioned earlier. In addition, the boolean queries execute *must* and *should* clauses in the query context, and *must_not* and *filter* clauses are executed in the filter context [4].

Second, the function score queries are utilized in the ranking function implemented in this thesis. Furthermore, it provides the ability to modify the content scores returned by the main query and take into account other configured functions when calculating new scores for the document [4]. Figure 2.2 provides an example of the functions score query that was tested during the development phase of this thesis. The function score query wraps the main query inside, seen on line 15 of the figure, and adds results from several mathematical functions to the resulting content

```
{ "function_score": {
    "functions": [
        {
            "field_value_factor": { "field": "viewScore", "missing": 0.0 },
            "weight": 1.0
        },
        {
            "field_value_factor": { "field": "orderScore", "missing": 0.0 },
            "weight": 1.0
        },
        {
            "weight": 1.0
        }
    ],
    "query": { "bool": { "must": [ ... ] } },
    "score_mode": "sum"
}}
```

Figure 2.2 *Example of a functions score query tested in this thesis.*

scores from the inner query.

In addition, the functions score query provides a possibility to define the scoring mode, which defines how the scores from the individual functions are combined [18]. Figure 2.2 shows the query has *sum* as the score mode, which calculates the scores for the documents with the following formula [18]

$$score_{total} = w_1 \times score_{views} + w_2 \times score_{orders} + w_3 \times score_{content}$$

The field value factor functions fetch a decimal value from the document and use it as the score for the functions; additionally, a missing value from the document is replaced with the value defined in the missing parameter [4].

In addition, the above formula uses weights for each value when calculating the total score, using weights enabled changing the emphasis on different functions more dynamic [18]. As was stated before, the ranking of the search results is an optimization problem, and finding the optimal threshold can be automated to find the optimal solution by testing different weights. Moreover, the individual weights are defined inside each of the function, the last weight with no function defined is the content weight w_3 presented in the formula above.

The previous section described how the business score is calculated and added to the content score and mentioned that the content score is a result of the inner query from Figure 2.2. This next part introduces an example of a query utilized in this thesis and summarizes the search functionalities introduced in this section. Figure 2.3 shows an example query that was used during development, which utilizes a query string query provided by Elasticsearch. Furthermore, the query string query allows the creation of complex queries that include wildcard characters, searching across multiple fields, and the usage of different functionalities.

Figure 2.3 shows the search across multiple fields as an array of field names

```
{ "query_string": {
    "analyze_wildcard": true,
    "fields": [
        "title^3",
        "manufacturerName^2",
        "shortDescription",
        "manufacturerName.edgeNGram",
        "categoryName.edgeNGram"
    ],
    "query": "*samsung* AND *puhelimet*",
    "tie_breaker": 0.3,
    "boost": 0.3
}}
```

Figure 2.3 *Example query used in this thesis.*

in the query. In addition, the query uses boosts on the individual fields *title* and *manufacturerName*. In Elasticsearch, the boost for a specific field is defined with a circumflex followed by the multiplier that multiplies the base content score for the field [18]. In addition to field boosts, the example query utilizes the previously introduced search time edge-N-gram analyzer to match the term to category and manufacturer of the product.

The search terms are shown in the query variable in Figure 2.3. Both terms are surrounded by the wildcard characters to allow the terms to be matched with characters missing from either end. A query must enable *analyze_wildcard* to utilize the wildcard search, since, by default, Elasticsearch does not do wildcard searches [4] since the computationally costly nature of the wildcard search [18].

As default queries use a *best fields* functionality, which means when searching across multiple fields, only the best matching field is included in the content score. However, the query in Figure 2.3 introduces a *tie_breaker* variable that softens the *best fields* functionality by introducing a weight that the other fields are multiplied with and included in the content score calculation [18]. Turnbull and Berryman [18] discovered that using only the *best fields* can lead to excluding some fields, especially when using boosts for individual fields. So, to include all the fields in the content score, the *tie_breaker* variable was introduced to the query, which provided better relevancy for the search results during the development phase.

This section has described the functionalities of Elasticsearch used in this thesis. The section started with an overview, before moving on to introduce the different text analysis tools provided by Elasticsearch. Lastly, the section described what happens during searching and how queries and scoring functions can be utilized to get a score of how well the query matches to the search results. Furthermore, using the score becomes increasingly relevant as the next sections introduce, what relevant search results are and what kinds of ranking algorithms have been introduced in previous literature.

2.4 Ranking the search results

Ranking algorithms are generally integrated part of a search engine, at least in the production scale engines, like Elasticsearch. Furthermore, ranking algorithms are also widely researched topic, and this following section introduces necessary ones which served as a base for the algorithm created for and utilized in this thesis.

Since the ranking algorithms are extensively studied field, and there are vast amounts of different solutions about the ranking of the search results from machine learning methods like learning to rank functions [9, 10] to how different e-commerce platforms have utilized different ranking functions [16, 17, 19]. However, this research mainly focuses on using the built-in methods of Elasticsearch when ranking the products by relevancy, rather than using complicated methods, due to the time constraints. While the need for more complicated ranking functions might be inevitable in the future, it was not necessary for the first iteration of the new search application.

The ranking algorithms used in this thesis are mainly based on the methods used by Turnbull and Berryman [18] in their book. They stated that the base score for a product comes from the content score, as mentioned previously, and that additional business scores are added to it [18]. In contrast, Long et al. [11] achieved an increase in revenue when including the number of transactions from the day before in indexing as the popularity measure for a product.

Furthermore, utilizing the popularity measure as one of the business scores, when calculating the score for the product in search time, is significant and matches the needs of the organization effectively. So, in this thesis, the proposed algorithm worked similarly, but instead of using the number of transactions as the metric to calculate the relevancy rating of a product, the number of product page visits was used. Furthermore, this provided much more data points to be used to calculate the relevancy rating.

This section gave an overview of the ranking algorithms relevant to this thesis. While there are a lot of complicated algorithms that could have been utilized, the choice to build an own ranking function with already existing data was sufficient for this scope. The following section continues to talk about the relevancy of the search results and how the relevancy can be measured in a way so the results can be compared.

2.5 Relevancy of the search results

To be able to serve customers relevant search results to the queries, it is necessary first to understand what relevancy is. Turnbull and Berryman [18] defined the relevancy of the search results to be:

> Relevance is the practice of improving search results for users by satisfying their information needs in the context of a particular user experience, while balancing how ranking impacts our businesss needs.

In addition, Turnbull and Berryman [18] argued that relevancy is subjective depending on the person who is using the search engine. While two people can use the same search query, they value different things differently [18]. For instance, two customers, both of whom search with the term "samsung" might have different information needs. The first customer is searching for refrigerators, and the second is searching for phones. While both are equally relevant to the search term, both cannot be the first search result.

The solution which solves the example above is personalized search, but since that is out of the scope of this thesis, more general measurements for the relevancy were required. Turnbull and Berryman [18] stated in the definition of relevancy that the relevancy also includes balancing the ranking in regards to the needs of the business. Therefore, as previously mentioned, the score for the document comes from the content score; in this case, the score based solely on the query what is relevant and business scores, which again fills the needs of the organization. Furthermore, the business score is usually deciding factor which products to show on top of the search results if the query matches both equally.

In information retrieval, relevancy is purely based on how well the results match the query [18]. However, in practice, relevancy is more than satisfying the information needs; it also includes satisfying the needs of the business, which might depend on the context [18]. For instance, in an e-commerce context, for general search terms, showing the products that are out of stock might not usually be relevant. While a product that is not in stock might meet the requirements of the query well, it does not satisfy the needs of the business, since the product cannot be sold.

So, to achieve some general relevancy at first, a ranking algorithm that calculated scores for the document based on the popularity rating of the product. Furthermore, the popularity rating was calculated based on the number of visits on the product page in the previous few days. In theory, the product pages that customers visit are relevant to the customers, so in a way, using the data based on the behavior of customers should satisfy the needs of the customers. In addition to the popularity of the products, additional scores were added, and the scores were normalized, but that will be introduced in more detail in Section 4.1.3

The above section provided an insight into what relevant search results mean. Furthermore, defining and implementing a relevant search is a difficult task as the relevancy is not objective; instead, it changes depending on the user of the application. However, before introducing how this thesis tried to achieve relevancy, the following section describes the state of the current search application. Furthermore,

understanding the current state of the application is essential, so the changes proposed in the method section make sense.

3 Current search application

The purpose of the thesis is to increase the conversion rate of an e-commerce search application by providing more relevant search results. While the earlier chapter provided the necessary background information from the literature, the purpose of the following chapter is to introduce the current search application and how it functions. Generally, to improve an existing process, it is first important to understand the process. So, this chapter introduces how the current search application works from receiving the query from the customer to providing the search results back to the customer.

In addition, the following chapter introduces the process of managing the search ranking and monitoring the current search application. The conclusions in this chapter are based on a survey about the status of the current search [15]. Furthermore, the survey was answered by the managers of the current search application responsible for keeping the search ranking up-to-date. In addition, the survey was concluded to collect the ideas on how the current setup could be improved and what is useful in the current setup.

3.1 Performing query and receiving results

The current search application is on multiple websites of the organization across the Nordic countries. There is one back-end that handles all traffic and multiple Elasticsearch clusters for each website to hold the website specific data. Figure 3.1 describes the application from the input of the customer to the customer receiving the search results back. In addition, the figure shows the areas affected by this thesis highlighted in red. The application front-end uses an HTTP request to communicate the input query to the back-end application, which constructs a query from the input. Then the constructed query is sent to Elasticsearch cluster through the Query API, which returns a set of documents matching to the query which again is enriched with necessary data and sent back to the front-end to show the results to the customer.

The following part introduces how the current application creates an Elasticsearch index for the search application and how the query is currently created. Furthermore, the updating of the search score boosts is described in the next section. The parts that are not highlighted not necessary to introduce as those parts are not changed or affected by the implementation. While the implementation required some work in the front-end of the application to enable A/B testing, there is no need to go into detail how the application works other than the affected elements.

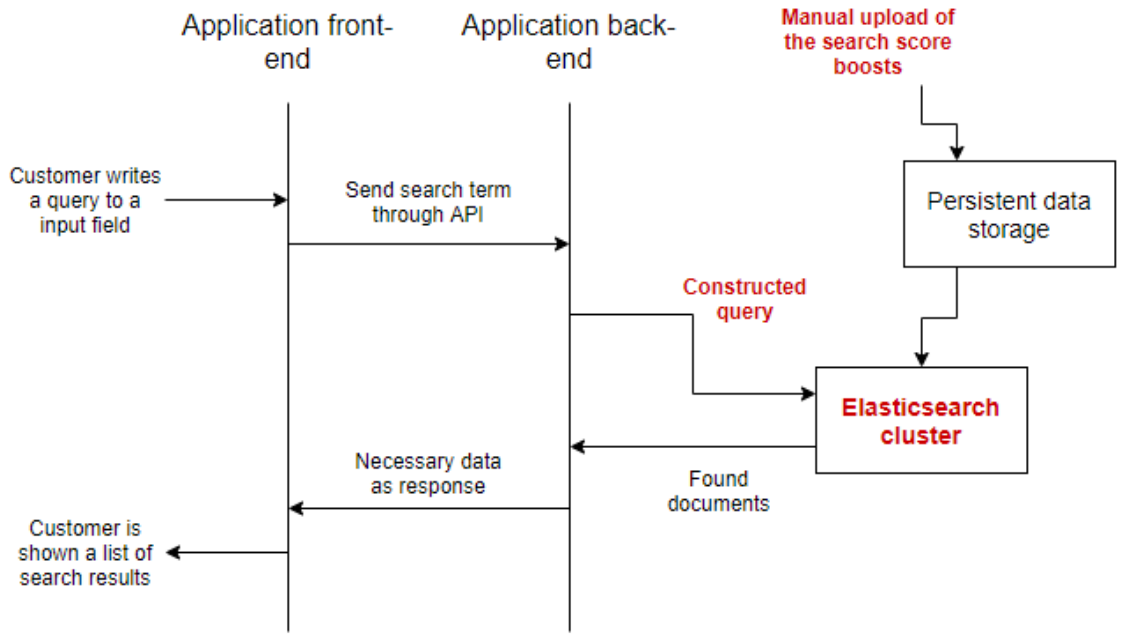


Figure 3.1 *Diagram of the current search application with targeted areas highlighted in red.*

As previously stated, Elasticsearch is a powerful tool when it comes to searching, in addition to providing many tools to make efficient queries to show relevant results to customers. However, the current search application has not been developed much in the last few years due to other projects taking priority over it. Therefore, it is missing out on some of the powerful tools provided by Elasticsearch. In addition to the implementation being outdated, it lacks automation in the business score calculation for the products, but the next section introduces that in more detail.

The current solution lacks efficiency when it comes to indexing the documents. While indexing a document in Elasticsearch is very fast, it still requires quite a lot of computing power when bulk indexing thousands of documents at the same time. The current search application utilizes only one analyzer, the language analyzer, in one field only. Furthermore, it contradicts with the function that creates the index for the product data, since it has multiple analyzers defined for multiple fields which are not utilized at all. Like previously mentioned, the analyzers used in text analysis require a full index re-creation, which can be quite inefficient to perform in a production application. So, the unutilized analyzers are most likely left there from some previous implementation. Furthermore, the re-creation of the index can be automated but usually can be performed manually by creating a parallel index with the data that the production environment is switched to use.

In the query construction, the application utilizes a wildcard search, which adds wildcard terms to both the start and the end of each search term in the query, as

```
{ "function_score": {
    "functions": [ { "field_value_factor": { "field": "scoreBoost" } } ],
    "query": {
        "query_string": {
            "fields": [
                "productId^15",
                "title^15",
                "searchTitle^5",
                "searchTerms^5",
                "manufacturerName^5",
                "salesArguments",
                "categoryName.finnish",
                ...
            ],
            "query": "*apple*"
        }
}}}
```

Figure 3.2 *Current query used by the search application.*

seen in Figure 3.2. Furthermore, in some cases, the wildcards in both ends of the search term make sense, but in some cases might provide inaccurate search results, at least, in cases where the search terms are short. For instance, the results for a query "tv" might be confusing since two letters might match many terms in the inverted index when used with wildcards. In summary, the lack of text analysis used and the lack of effort put into constructing the query is correlated with the lack of effort that has been put to developing the search application.

The problems with the current search application can be seen in the query shown in Figure 3.2. The query shows that the query terms are searched across multiple fields, and the fields have separate boosts defined to them. For instance, *title^15* in the query means that the content score calculated for the term matching to the title field should be multiplied by 15. In addition, Elasticsearch uses matches to the best field only, the field with the highest content score, as default, as mentioned before. So, adding a boost to a field makes it more likely to be the best field it a match is found from it. While making sense that if a match is found from the title of the product, it should be prioritized over a match in, for instance, sales arguments, it is difficult to say by how much should it be prioritized.

Figure 3.2 shows only a set of fields that were used. Furthermore, the query has some fields, like *searchTitle* and *searchTerms*, which indicate that there has been some idea to provide some additional content just for searching, but the fields seemed either empty in most products or are just generated from other fields in runtime. Again, showing the urgent need for updating the process to be more efficient. Furthermore, searching across multiple fields with wildcard search can result in some results which are hard to explain and might not seem at all relevant to the query. This behavior of odd results for queries can be observed when interacting with the search application for a while.

The above section provided an overview of how the current search application constructs the query used by the search application. The current solution is referenced in Chapter 4 when new solutions are introduced to make the application provide more relevant search results. In addition to the actual query object, Figure 3.2 shows a ranking function, the value of which is used to multiply the content score from the query object. The function on line 2 fetches a value from a field called *scoreBoost*, which is used in calculating the document score for each of the documents. The following section provides an overview of the process behind updating the score boost field for the products.

3.2 Updating the search ranking scores

As previously stated, an essential part of a search application is a ranking algorithm to order the search results in a specified order, in an ideal case by the relevancy. The current search application ranks the results with a value from a field called *scoreBoost*, shown in Figure 3.2. Furthermore, the field is precalculated and updated to the persistent storage, and the field is indexed with the document at the same time as other data. Thus, by using a precalculated field to rank the search results, the calculation necessary by the search engine lessens. Furthermore, there is no need to create a complicated ranking algorithm that the query uses to extract values from multiple fields to calculate the score for the document.

As shown already in Figure 3.1, the process behind updating the score boosts field to the persistent storage is a manual process. Furthermore, there is a total number of four different websites and Elasticsearch indices, one for each site. Currently, the process of updating the score boosts is done manually by different individuals for different websites. The boosts are calculated by using an Excel file, which has been created to allow boosting based on various parameters, including margin, stock levels, and sales, in addition to allowing different boosts based on category and brand of the product.

The process has been developed to keep boosting consistent across the different categories [15]. Furthermore, the different parameters have been chosen based on empirical tests and based on the expertise of the sales representatives across the categories [15]. The knowledge about the category has been a relevant part when the process was created since different parameters might vary across categories [15]. For instance, the importance of a stock level might differ between a product that is continuously in stock and a product that must be ordered when a customer places an order [15].

Since there are tens of thousands of products active in each of the websites, the usage of a manual process can be quite costly. However, not all products require to be separately boosted; some might just receive the boost based on their category or

brand. Mostly, the products that are in an upcoming campaign or are advertised might need special attention [15]. For an expert user, the process takes anywhere from 30 minutes to two hours to complete [15]. Furthermore, since the process is done by multiple individuals across the organization, the total cost in person-hours can be hard to measure.

The benefits of using a manual process are that detailed boosts can be achieved for specific situations, for instance, product campaigns [18]. While the creation of specific boosts for a set of products takes time, it can be beneficial to modify a specific boost for a product, if the parameters did not reflect the need well [15].

For instance, changing a boost to better match a query has been to introduce an additional boost for iPads and Macbooks to better match the search query "apple". While both are Apple products, neither mentions the word "apple" in the title and iPhones do, so naturally, due to the high field-specific boost shown in Figure 3.2 makes the top of the results consist almost entirely of iPhones. Therefore, a need for boosting iPads and Macbooks was necessary and could be achieved with an understanding of the domain and the manual process. [15]

This chapter has introduced the current search application, how it functions, and how the current search results are scored. The following chapter describes the methods used in this thesis to offer relevant search results back to the user. The methods introduced are based on the methods used previously on literature and the current manual process of boosting.

4 Methods

This chapter introduces the different implementations done to improve the relevancy of the search results, in addition to the tests concluded to measure the effect that the implementation had. The purpose of the implementations was to provide a general direction for future development while providing some robust data on the performance of the current search application for the organization. Furthermore, the prerequisite for developing the methods was if one were to succeed over the current process, the organization would put more effort into developing that method further.

4.1 Implementing a relevant search

The purpose of this section is to introduce how the relevant search was implemented in this thesis. First, the section provides an overview of the interpretation of how to achieve more relevant search results. Second, the section introduces the methods that were used in creating tokens while indexing. Lastly, the section describes how the popularity data was collected to replace the manual process of uploading the score boost, before moving on to introduce the new ranking function was developed

and how the score calculation implemented functioned.

Since relevancy is subjective and personalized search was out of scope, as previously mentioned, a general solution that tried to capture the general relevancy of the search results. Therefore, the purpose was not to show different search results based on the customer who is interacting with the search application or based on the previous products the customer has viewed. The solution that was developed by the following parts did not take into account any personalized information; instead, it tried to achieve good relevancy by using the popularity data for products.

4.1.1 Creating tokens from content

The following part describes the process of generating tokens from the content of the products. As previously mentioned, the token generation is a crucial part of a search engine, which indeed became apparent during the development phase. Furthermore, during development, there were a few different options tested and considered, the following part goes through the process of choosing the best one based on the empirical tests concluded during development.

The development was done on the Finnish website, where the contents of the products are in Finnish. Due to the nature of Finnish language having many inflected forms of words, it quickly became apparent that only the language analyzer. Furthermore, this was a case mentioned previously, where the language analyzer could not interpret the meaning from the words and, while generating good enough tokens for most cases, failed to do so with longer words. Thus, the first thing where the development started was to investigate how more meaningful tokens could be generated.

Since the development was done in Finnish and Elasticsearch supports custom plugins, a new Finnish analyzer was found that could solve the problems since it seemed to capture the meaning from even more difficult words well. However, the decision to not use it was done, since, first, it would have required the be installed in the existing production Elasticsearch cluster, and second, the language analyzer was only for Finnish, so other Nordic languages would need to use a different solution. Furthermore, the lack of knowledge about the nuances of other Nordic languages made it impossible to determine how well the language analyzers were performing in practice.

However, by looking at how the language analyzer worked at a lower level, it became clear that an edge-N-gram analyzer could be utilized to achieve similar behavior as the language analyzer. In addition to the similar behavior, the edge-N-gram could capture the meaning from a word better as the amount of N-grams that were generated could be controlled. However, capturing meaning from the words did not equal to generating tokens with a minimum length of two, as quickly became

clear during development.

In addition, a similar problem arose during the usage of an analyzer during search time since if one of the N-grams generated matched a token in the inverted index, the document was considered a match. While this is the expected and wanted result, it meant that when generating small N-grams, the precision of the search results suffered greatly while the recall went up. The low precision can be observed in Listing 4.1, by generating edge-N-grams with a minimum length of two from the words “puhelimet“ and “punainen“.

```
[ pu, puh, puhe, puhel, puheli, puhelim, ... ]
[ pu, pun, puna, punai, punain, punaine, ... ]
```

Listing 4.1 *Output from the edge-N-gram analyzer with a minimum length of two.*

As the first term translates to *phones* and the second term to *red*, the words mean entirely different things, so by querying either should not match to the other one.

To solve similar problems, as introduced above, a custom edge-N-gram analyzer was created, as shown in Figure 4.1. Furthermore, the custom analyzer includes two different edge-N-gram analyzers, the first, *edge_ngram_analyzer*, which is used during indexing of the documents, and the second, *search_term_over_6*, which is used during search time. Both analyzers utilize the same tokenizers and the first two token filters to keep the generated tokens consistent, first by converting the words to lowercase and then converting them to ASCII characters.

Then, the index time analyzer creates N-grams with a minimum token length of two and a maximum token length of 15 to try to capture as much meaning from the words as possible. While, the search time analyzer uses a conditional token filter, which determines if an N-gram should be created or not. Since the problem in Listing 4.1, the search time tokenizer only generates N-grams if the search term is over six characters long. The threshold of six letters was achieved by calculating the average length from the queried search terms over one month. In addition, six letters seemed to capture the meaning from the words during development well enough that it was chosen. However, in future development, the process of choosing the threshold could be automated to find the optimal threshold while balancing precision and recall of the search results.

In summary, by using different analyzers during index time and search time allowed to create shorter N-grams to allow matching with shorter search terms. In addition, the usage of an edge-N-gram analyzer in search time only when the search term exceeded the configured threshold allowed the exact matching of the shorter search terms. In addition to allowing to try to capture the meaning of the longer ones and match the generated tokens to the inverted index.

```
{ "analysis": {
    "filter": {
        "edge_ngram_token_filter_5_12": {
            "type": "edge_ngram",
            "min_gram": "5",
            "max_gram": "12"
        },
        "edge_ngram_token_filter": {
            "type": "edge_ngram",
            "min_gram": "2",
            "max_gram": "15"
        },
        "search_term_over_6": {
            "filter": [ "edge_ngram_token_filter_5_12" ],
            "type": "condition",
            "script": { "source": "token.getTerm().length() > 6" }
        }
    },
    "analyzer": {
        "edge_ngram_analyzer": {
            "filter": [ "lowercase", "asciifolding", "edge_ngram_token_filter" ],
            "type": "custom",
            "tokenizer": "standard"
        },
        "search_term_over_6": {
            "filter": [ "lowercase", "asciifolding", "search_term_over_6" ],
            "type": "custom",
            "tokenizer": "standard"
        }
}}}
```

Figure 4.1 *Different analyzers for index time and search time.*

The previous part introduced how the text analysis tools provided by Elasticsearch were utilized to try to generate meaningful tokens. In addition, a customer edge-N-gram analyzer was introduced to generate different tokens during indexing and searching. The following part continues to introduce how the manual process was replaced by an automatic one.

4.1.2 Replacing the manual score boosts

As precious stated, a more general solution was needed, so the purpose of this part is to introduce how the manual process of updating the score boosts was replaced by an automated process. As the purpose was not to create separate search strategies based on, for instance, the behavior of a customer, a precalculated boost value as the current solution had was enough. Furthermore, the current solution used many parameters when calculating a new boost value for a product, and most of them seemed unnecessary and had little to do with relevancy.

So, to answer the question, what customers see relevant, a popularity rating was introduced. Furthermore, the rating just included a normalized value of the number of product page views from the previous few days. However, since the data was not available during indexing, but was collected and stored by Google Analytics, a new integration was required to import the data. Thus, a new task was created

```python
from functools import reduce
import numpy as np

def boost_rating(view_per_date, weight):
    def weighted_average(value1, value2, weight):
        return value1 * (1 - weight) + value2 * weight

    decayed = reduce(lambda acc, x: weighted_average(acc, x, weight), view_per_date)

    return np.log(decayed + 1)
```

Figure 4.2 *Function to calculate boost rating based on the number of views.*

which imports the data every morning from the previous day and saves the value to persistent storage, where then the indexing process could fetch the data and calculate the value for the boost.

Calculating data based on the previous days might not be the most accurate at least in e-commerce since many factors affect the popularity of a product from different trends to many campaigns and promotions. So, to keep the rating as updated as possible, a custom function, shown in Figure 4.2, was introduced. Furthermore, the function receives an ordered list of the number of views and weight as input, and then it calculates the decaying average starting from the first item on the list. The function weighs the items at the start of the input less than the last items, so this allows the newer product page views to affect the resulting rating more than the older dates.

In addition, after calculating the decaying average, the function uses a natural logarithm to flatten the products with a very high number of views. Furthermore, a very high view count is not that meaningful since it might negatively affect the ranking, as it might override the content score altogether, and it does not capture the popularity that much. For instance, saying that a product with 10000 views is ten times more popular as a product with 1000 views might not be accurate. The last step of the score calculation is to normalize the rating, given by the function, across all products to a value between zero and one. Furthermore, this prevents high business scores, which might cause the content score to lose meaning in the ranking functions.

The previous part introduced the new process introduced to replace the current manual process altogether. The process included an integration to import data about the number of product page views to the persistent storage and the number of views to be used to calculate a rating for the products in the index. The following part continues to introduce how the previous rating is utilized in the ranking algorithm, which determines the position of the products in the search results.

4.1.3 Ranking algorithm

The purpose of this section is to introduce the ranking algorithm used in the implementations. In total, there were two different ranking algorithms created for this thesis, both utilizing the calculated popularity rating, introduced in the previous section. This section will introduce both algorithms and provide the reasoning for developing the first one further.

The precalculated popularity rating serves as the base for the ranking algorithms. Furthermore, the first iteration of the ranking algorithm only used the popularity rating as the business score. However, after the first test concluded, it became clear that it did not perform as expected since the conversion rate went down. Since the first algorithm did not include anything else other than the popularity score as business scores, the algorithm captured the intent of users well but did not meet the needs of the business. Furthermore, the top products in search lists were relevant, but there were multiple cases that the product was not in stock. Thus, providing a relevant search result back to the user, but the search result was still not buyable, explaining the drop in conversion rate.

The second algorithm created was similar to the first one with an additional boost if the product was in stock. The second algorithm uses the following formula to calculate the score for a product

$$score_{product} = 1 \times score_{popularity} + 0.3 \times score_{stock} + 1 \times score_{content}$$

where the popularity, stock, and content score are first multiplied with weight values, and then the sum of all the scores if the calculated score for the product. The $score_{stock}$ is a value of 1 or 0 depending on if the product has stock and customers can buy it. The weight values for the individual parameters are a result of the empirical tests and should be a subject for future development.

The value that changes the most in the ranking algorithms is the content score $score_{content}$. Furthermore, the base content score is calculated in Elasticsearch by dividing the term frequency with the inverse document frequency. However, in this case, the first concluded test showed that the content score calculation in the first algorithm could be improved since it did not capture exact matches well due to the wildcard nature of the query.

So, the second ranking algorithm, in addition to the updated business score, included multiple queries to boost the exact matching results better. Furthermore, multiple queries are shown in Figure 4.3. Using multiple queries allowed the first query to use wildcard characters both at the start and at the end of the search term, the second query to utilize the wildcards at the end of the search term. The last query did not use any wildcards, so the exact matches received a small boost if they

```
[{ "query_string": {
    "boost": 1.0,
    "fields": [
        "title^2",
        "manufacturerName^2",
        "categoryName",
        "shortDescription",
        "salesArguments"
    ],
    "query": "*bose*",
    "tie_breaker": 0.3
}},
{ "query_string": {
    "boost": 0.3,
    "fields": [
        "title^2",
        "manufacturerName^2",
        "title.edgeNGram",
        "manufacturerName.edgeNGram",
        "categoryName.edgeNGram", ...
    ],
    "query": "bose*",
    "tie_breaker": 0.3
}},
{ "query_string": {
    "boost": 0.05,
    "fields": [
        "title^2",
        "manufacturerName^2",
        "title.edgeNGram",
        "manufacturerName.edgeNGram",
        "categoryName.edgeNGram", ...
    ],
    "query": "bose",
    "tie_breaker": 0.5
}}]
```

Figure 4.3 *Three parallel queries used across different fields.*

matched the third query also.

Elasticsearch allows the usage of multiple queries at the same time withing a *should* clause, so if a product matches any of the three queries, it will be shown in the search results. Furthermore, the only difference is if the product matches multiple queries, then the content score for the product will increase based on how well it matches to the query. So, each query has a *boost* variable defined, which allows using different weights for each query. As in this setup, the exact queries did not make sense to be boosted highly since while increasing precision and decreasing recall. The selected boosts for each of the queries were the result of the empirical tests done during development.

In addition, each of the queries defines a *tie_breaker* value, which tells Elasticsearch that instead of using the *best fields* functionality search across all fields. Furthermore, the scores from all fields are utilized in the content score calculation, as introduced in Section 2.3. The utilization of all fields allowed the content score to not only include the best matching field to reflect the content of the document better. It seemed to provide better search results during the empirical tests, but in the end, using all fields depends on the content in the fields. Furthermore, if the content is very different in the fields, then the usage of only the *best fields* might be justified.

The previous section introduced different methods for this thesis. Furthermore, the new methods proposed included new analyzers to generate meaningful tokens, the new process to replace the manual process. In addition to a new ranking function, which utilized the calculate popularity rating for the product and used multiple queries to manipulate the content score calculation. The following sections continue

to explain how the tests were concluded and how the test results were analyzed before the results are introduced.

4.2 Testing the search relevancy

The purpose of this section is to introduce the testing methods used and introduce how the test setup worked. Since the tests were concluded with Google Optimize, which allows running different kinds of tests. Moreover, in this thesis, simple A/B tests were sufficient enough. Google Optimize requires a script to be loaded on page load, and it uses a cookie to determine which variant the customer belongs to.

Google Optimize offers a possibility to modify the part of a website while configuring the test, which allowed a custom tag to be added when a customer loads the page of a website and loads the Google Optimize script. So by injecting custom HTML tag to the webpage shown to the customer, the front-end JavaScript code could then determine which variant the customer belonged to and set a custom cookie according to the variant.

When a customer interacted with the search input field and queried something, the receiving back-end API determined by looking at the cookies of the customer which variant the customer belonged to and constructed the query differently based on the variant in the cookies. Furthermore, after constructing the query and posting it through the Query API to the Elasticsearch search engine, the results were handled independently of the variant the customer belonged to. So, only by constructing the query differently based on a cookie value, different results could be shown to the customers, by using the same code as before.

While there were four different variants, only one was used by a specific website along with the baseline variant. Since the tests were concluded in three different websites across Nordic countries, and the codebase is the same across the website back-ends, there was a need for a total of four variants. Each variant had different functionality enabled, for instance, the variant B had only the new ranking algorithm which used the popularity rating and stock availability, and the variant C had the new queries for calculating the content score, but used the old *scoreBoost* field to calculate the business scores. Furthermore, the variant D had both enabled, and the variant A was the baseline with nothing new features enabled. The specific test setups across the variants are introduced in detail in Section 5.

This section provided an overview of how the testing setup worked through the third party Google Optimize software. Furthermore, by creating a cookie to the browser of the customer, the back-end code could construct the Elasticsearch query differently based on the variant. The following section introduces how the results from the tests were analyzed before moving on to the actual results.

4.3 Analyzing the results

This section introduces how data from the tests were collected and analyzed using Google Analytics. Furthermore, Google Analytics is a powerful analytics tool that provides the collection and analysis for organizations. While Google Analytics collects the data, Google Optimize adds an identifier to the collected data about what variant the customer has belonged to during the test phase. Both products are operated by Google and are used in many organizations across the world.

The measurements introduced in Section 2.1 were used as the primary measurements in this thesis. However, for a more in-depth analysis of the results, Google Analytics provides a variety of functions to analyze the collected data. The measurements were collected based on the time frame of the tests and the identifier added by the Google Optimize. A more in-depth analysis of the results was not necessary to understand the effects of the concluded tests. The collected measurements captured the behavior of the customers well, and the result reflects the expectations moderately.

Since the collected data was cross three different websites, they were easy to differentiate since, for each website, there is an own Google Analytics project. The results were collected based on the test version that was concluded, and based on the website, it was concluded in. The following section will introduce the results in detail and provide an analysis based on the results.

5 Results

This chapter introduces the individual tests and what was included in each of them, in addition to the results collected from the tests. The following results use both measurements, the conversion rate, and the click-through rate as previously introduced. Furthermore, the conversion rates in the following tables are the conversion rate of the customers that used the search application. Also, the CTR shown in the first three tests is measured in the search result page from the subset of customers that did not use any filters when they visited the page, and in *Test 4*, the click-through rate included all customers. In the following descriptions, *Variant A* is not referred to since it served as the baseline in all the concluded tests.

The results for the first three tests include the three different variants per test, as each variant constructed the query differently. *Test 4* includes the measurements across different websites since the last test concluded included the best parts from the previous three tests. Furthermore, the fourth test was done in three different countries to see if the same solution could be applied across the Nordic countries.

Test 1 included the first iteration of the ranking algorithm that boosted the products purely based on the popularity rating. In addition, the test included a

query that was the first iteration of the one previously introduced in Figure 4.3. The query worked similarly and had the custom edge-N-gram analyzers with a wildcard character at the start and the end of the search terms. The query was not separately introduced in Section 4.1.3 since the query was similar to the first query from the introduced multiple queries setup. In *Test 1*, *Variant B* included only the new query, *Variant C* included only the new ranking function, and *Variant D* included both the new query and the new ranking function.

Figure 5.1 shows that *Test 1* increased the CTR of the top products when the new ranking function was utilized, in *Variant C* and *Variant D*. However, the usage of the new ranking function decreased conversion rate, shown in Figure 5.1, most likely due to the ranking algorithm not boosting the products that are in stock over the ones that are not. While the conversion did not increase with the new ranking function, the effect of the new query can be seen with *Variant B*. Furthermore, the conversion rate slightly increases, and also the CTR of the top of the product list seems to be increased slightly. The usage of the edge-N-gram analyzers in the new query for more meaningful token generation seemed to make the CTR increase slightly.

In *Test 2*, both the ranking algorithm and the query were upgraded. While the ranking algorithm in the previous test did not include the stock status of a product, a value that was calculated was added to the index and used by the ranking algorithm. Furthermore, *Variant B* and *Variant D* utilized the new ranking algorithm. The previous query was updated to the one shown in Figure 4.3, and it was utilized by *Variant C* and *Variant D*.

Figure 5.2 shows, again, an increase when the ranking algorithm was utilized. However, the multiple query setup did not seem to have an actual effect, although it seemed to perform considerably better during the development. The effect *Test 2* had on conversion rate can be seen in Figure 5.2. Furthermore, the conversion rate seems to be slightly increased in all variants.

The result did not meet the expectation set before the test, as the first test showed that conversion decrease as the products might have been out of stock. However, the expectation was that the conversion rate would be increased since the new ranking algorithm boosted the products in stock higher. In addition, it seems that the usage of the analyzer in the specific fields is not optimal, as it seems to increase the CTR, but not the conversion rate.

While the previous two tests included three different variables, *Test 3* only included two. Both *Variant C* and *Variant D* included the same ranking function and multiple queries as *Test 2*. However, instead of using *MUST* clauses with multiple search terms, *SHOULD* clause was used. Furthermore, this meant that by searching with multiple search terms, all matches that matched any of the search terms

were included, but the results that matched several terms were boosted accordingly higher.

Figure 5.3 shows that the CTR of the search result list on the subset of customers who did not use the product filters after searching did not change significantly. Still, the conversion rate was increased in both different variants, as shown in Figure 5.3. The described behavior did not meet expectations, as the same ranking algorithm was utilized as in *Test 2*, the expected result was that the CTR would have been at a similar level as well.

The fourth and final test was concluded based on the previous tests, introduced above. In addition to the testing setup used in *Test 2*, this test included a functionality that, while querying the results, queried a suggested search term at the same time. Then, if the query did not match any products, the query would be rerun with the suggested word. Furthermore, meaning that, for instance, if the customer misspelled a word and it resulted in zero results, the suggested word would be searched with, and results with the correct term could be returned.

Furthermore, this functionality was designed to be similar to fuzzy search and was easier to implement, due to the time limitations, as using the fuzzy search. For the future development, the fuzzy search is most likely investigated properly, as fuzzy search does not work with wildcard searches, which were utilized by the queries, it could not be implemented for this thesis.

Test 4 the three variants again, but in this setup, all the variants utilized the same features. In this test, the three variants mean three different websites in three different countries. Furthermore, since the organization has multiple websites across the Nordic countries, it was crucial to see if the solution proposed by this thesis performed well in all of the countries.

The results, in Table 5.1, show a remarkable increase in the CTR of the search result list. Therefore, indicating that the relevancy of the search result was increased in *Test 4*, at least when measured with CTR. However, the conversion rate was not increased by *Test 4* that much, and in *Variant D* even decreased. While the solution in *Test 4* seemed to satisfy the needs of the customers, at least better, the solution did not seem to capture the needs of the organization, as the number of sales was not affected that significantly.

This chapter introduced the four different tests that were concluded for this thesis. Each test had separate variants, which were measured from different websites in different countries. In summary, the conclude test showed an increase in the click-through rate of the search product lists. However, the primary measurement, the conversion rate, was not significantly affected by the concluded tests. The following chapter includes the conclusions that could be drawn from the results, in addition

to including subjects for future development.

6 Conclusions

The purpose of this chapter is to summarize the goals of this thesis and identify some problems discovered during the tests. The chapter summarizes the research goal of this thesis in addition to the additional conclusions drawn from the results. The conclusions each include the subjects that could be researched further to improve the solution proposed in this thesis if some problems were found with the chosen implementation.

6.1 The effect on conversion rate

The primary purpose of this thesis was to increase the conversion rate of an existing e-commerce platform by offering more relevant search results to users. However, while the concluded tests did not indicate a significant increase in the conversion rate, they still indicated that there might be room to improve the solution, as the conversion rate still was affected by different tests. While there were a lot of useful tools provided by Turnbull and Berryman [18] in their book, using them, in practice, in a large scale search engine with multiple languages proved to be a difficult task.

Furthermore, this thesis still proved that the chosen direction was a success, and the development in the future should focus on the subjects introduced in this thesis. When it comes to increasing the conversion rate, much more work needs to be put into investigating how the conversion rate could be increased in the search result list. Since customers seemed to find interesting products according to the tests, but still, that did not translate into the conversion rate.

6.2 Relevancy of the search results

The other main point of this thesis was the relevancy of the search results and how it affected the conversion rate of the e-commerce platform. Based on the concluded tests, the conversion rate was not affected significantly by the different implementations. However, as previously stated, the customers seemed to find interesting products, at least when measured with the click-through rate. Based on the concluded tests, there seemed to be little correlation with the relevancy of the search results and the conversion rate in the specific e-commerce platform.

However, in literature, the conversion rate was affected by the search results. So, in this case, the problem, most likely, was that either the customers that interacted with the search are not that keen on buying products or that they are searching for information about a product. Which again, could be explained with the lack of development the search application has seen, which could have affected the behavior

of the customers, or that the products offered by the e-commerce platform do not satisfy the needs of the customer base well.

In addition, during development was found that the content in the products is not very searchable. For instance, the example with "pesukone" introduced previously. Furthermore, meaning that future development could also focus on generating meaningful tokens. For instance, mining content from the web about the products and using the previous search terms as a base, the already existing *searchTerms* field could be filled with actually searchable content.

In addition to generating content for the search engine, with properly generating the search token, the usage of wildcards in search could be made obsolete. Well generated tokens enable the usage of fuzzy search instead of the wildcard, which again allows the search engine to match words even though the customer has misspelled it. Furthermore, it could drastically increase the usability of the search application in the future as there would not need to write the search term correctly.

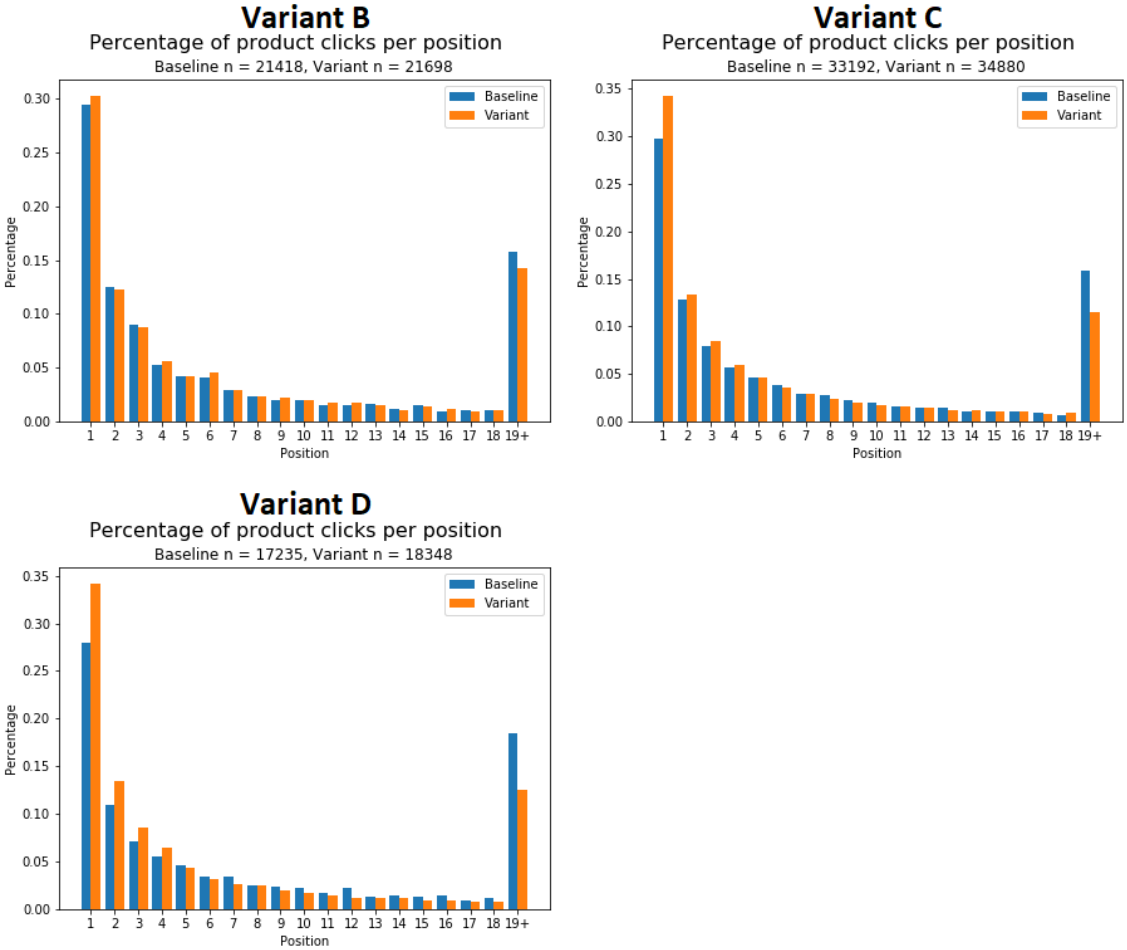
6.3 Testing the implementations

In this thesis, four different tests were concluded over six weeks. With the four tests, only a handful of different implementations could be tested, meaning that the testing of different implementation is prolonged. While the results from A/B tests are reliable, the time used to conclude them is not feasible.

As Turnbull and Berryman [18] said that with search application development, the goal is to fail fast, meaning that a new implementation should be tested, and if not working as expected, it should be disregarded. Using production A/B tests in a fail-fast development is not possible since the tests need to run for days to get meaningful results. In addition, since the data was collected in different periods and from different countries, the tests are not comparable between each other.

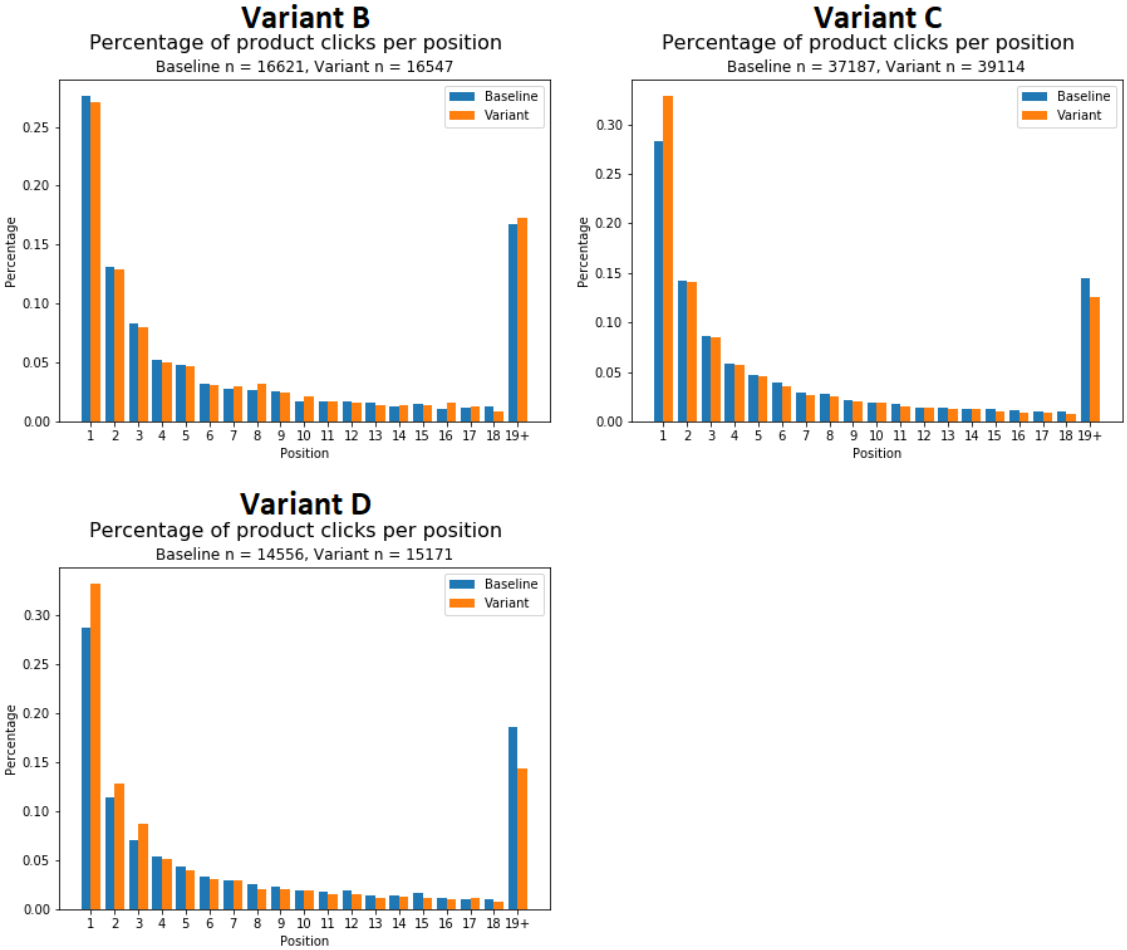
In the future, a testing platform purposefully built for testing search engines could be beneficial to run simulated tests to see what is working. The tests concluded for this thesis could have been done possibly in a matter of hours with a testing platform that simulates real customers. While the results might not have been as precise, still instead of waiting for results from the tests, more development could have been done. In addition, the simulated platform could provide comparable results across different tests since the data used could stay more or less the same.

This chapter provided an overview of the conclusions drawn from the results and included subjects that could be developed in the future. While the results of the concluded tests were not optimal as the conversion rate was not increased, a much was learned during the development. Therefore, this thesis proved to be an excellent learning experience in how search engines function at a lower level, in addition to how Elasticsearch performs as a production scale search application.



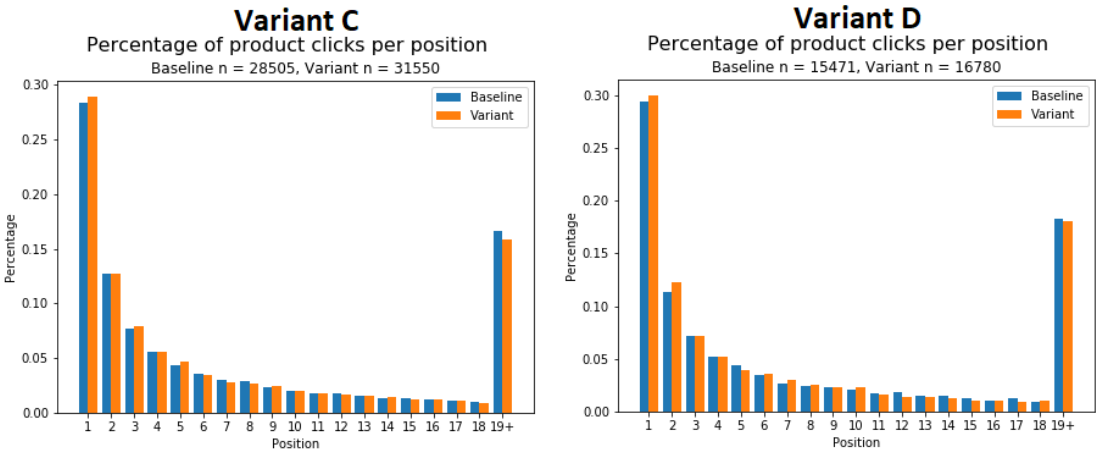
Test 1: Conversion rate from search						
	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
Sessions	59 359	58 587	85 764	85 511	55 246	55 444
Conversion rate	3,32%	3,43%	2,37%	2,19%	1,71%	1,52%

Figure 5.1 *Test 1: CTR and Conversion rates across variants.*



Test 2: Conversion rate from search						
	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
Sessions	47 884	47 866	89 820	88 990	45 472	46 414
Conversion rate	3,04%	3,20%	4,84%	4,95%	1,45%	1,46%

Figure 5.2 *Test 2: CTR and Conversion rates across variants.*



Test 3: Conversion rate from search				
	Variant C		Variant D	
	Baseline	Variant	Baseline	Variant
Sessions	79 894	81 279	52 575	51 814
Conversion rate	2,27%	2,35%	1,37%	1,44%

Figure 5.3 *Test 3: CTR and Conversion rates across variants.*

Test 4: CTR based on the product position						
List position	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
1	21,99%	28,59%	19,92%	25,19%	21,98%	27,11%
2	14,32%	16,65%	12,71%	16,90%	12,38%	16,42%
3	11,28%	12,29%	10,14%	12,39%	10,58%	12,06%
4	8,15%	9,10%	8,55%	9,77%	8,14%	9,31%
5	6,92%	8,14%	7,51%	8,91%	6,74%	7,93%
6	6,43%	7,37%	6,29%	7,86%	6,19%	6,88%
7	6,46%	6,74%	5,95%	7,19%	6,01%	6,25%
8	6,05%	6,33%	5,87%	7,12%	5,59%	5,87%
9	5,93%	6,28%	5,78%	6,75%	5,70%	5,37%
10	6,19%	5,94%	5,69%	6,68%	4,80%	5,44%

Table 5.1 *Test 4: CTR in the different websites.*

Test 4: Conversion rate from search						
	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
Sessions	119 609	119 535	189 661	192 392	105 572	108 762
Conversion rate	3,89%	3,95%	2,78%	2,96%	2,47%	2,41%

Table 5.2 *Test 4: Conversion rates in the different websites.*

7 References

[1] Amazon Alexa. *The top 500 sites on the web*. Available: http://www.ctan.org/pkg/listings. Checked 20.01.2020.

[2] Aden Andrus. *What is Conversion Rate? How to Calculate and Improve Your Conversion Rate*. Available: https://www.disruptiveadvertising.com/conversion-rate-optimization/conversion-rate/. Checked 15.02.2020. 2018.

[3] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X. URL: http://www.sciencedirect.com/science/article/pii/S016975529800110X.

[4] Elasticsearch. *Elasticsearch introduction*. Available: https://www.elastic.co/guide/en/elasticsearch/reference/6.8/elasticsearch-intro.html. Checked 20.01.2020. Elasticsearch B.V.

[5] Qingqing Gan and Torsten Suel. "Improved Techniques for Result Caching in Web Search Engines". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW 09. Madrid, Spain: Association for Computing Machinery, 2009, pp. 431–440. ISBN: 9781605584874. DOI: 10.1145/1526709.1526768. URL: https://doi-org.libproxy.tuni.fi/10.1145/1526709.1526768.

[6] Google. *Analytics Tools & Solutions for Your Business*. Available: https://marketingplatform.google.com/about/analytics/. Checked 20.01.2020. Google Inc.

[7] Google. *Optimize Resource Hub*. Available: https://support.google.com/optimize/topic/9340207. Checked 20.01.2020. Google Inc.

[8] Naveen Gudigantala, Pelin Bicen, and Mike Eom. "An examination of antecedents of conversion rates of e-commerce retailers: MRN MRN". In: *Management Research Review* 39.1 (2016), pp. 82–114.

[9] Changsung Kang et al. "Learning to rank related entities in Web search". In: *Neurocomputing* 166 (2015), pp. 309–318.

[10] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. "On Application of Learning to Rank for E-Commerce Search". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR 17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 475–484.

[11] Bo Long et al. "Enhancing Product Search by Best-Selling Prediction in e-Commerce". In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. CIKM 12. Maui, Hawaii, USA: Association for Computing Machinery, 2012, pp. 2479–2482.

[12] Melissa Mackey. *What Is Click-Through Rate & Why CTR Is Important*. Available: `https://www.searchenginejournal.com/ppc-guide/click-through-rate-ctr/`. Checked 22.02.2020. 2019.

[13] Greg R. Notess. "Tips for Avoiding, or Celebrating, Zero Search Results". In: *Online Searcher* 40.5 (2016), pp. 54–56.

[14] Power Internation As. *Internal Analytics of Power*. Checked 20.01.2020.

[15] Power Internation As. *Internal survey on the current status of the search application*. Checked 28.03.2020.

[16] Daria Sorokina and Erick Cantu-Paz. "Amazon Search: The Joy of Ranking Products". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR 16. Pisa, Italy: Association for Computing Machinery, 2016, pp. 459–460.

[17] Mohammed Zuhair Al-Taie, Siti Mariyam Shamsuddin, and Joel Pinho Lucas. "Predicting the Relevance of Search Results for E-Commerce Systems". In: *International Journal of Advances in Soft Computing & Its Applications* 7.3 (2015), pp. 85–93.

[18] Doug Turnbull and John Berryman. *Relevant Search: With applications for Solr and Elasticsearch*. USA: Manning Publications, 2016.

[19] Liang Wu et al. "Turning Clicks into Purchases: Revenue Optimization for Product Search in E-Commerce". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR 18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 365–374.