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 more straightforward 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 relevance of the search results of an existing e-commerce platform and analyze how the conversion rate is affected. Furthermore, the conversion rate describes the percentage of visitors that bought something from the total number of visitors on a website, and it is a widely used measurement in e-commerce.

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 conducted tests were over six weeks and included tens of thousands of customers.

The data that was collected during the testing phases of the proposed methods 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. 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 increased by the proposed methods, the results showed that with a simple solution, the existing process could be replaced. In addition, the results showed 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]. While using ranking algorithms and the ranking search results by relevance 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 relevance methods and the relevance of the search results have been investigated extensively in the past, research has included the use of the relevance methods in practice in search applications with real-world customers in e-commerce to a certain extent. 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 relevance of the search results affected the conversion rate of an e-commerce platform by implementing some methods based on the literature. The conversion rate measures the 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 relevance of the search result is subjective since not all customers are searching for the same thing making the relevance 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. The tests were 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 relevance of a different implementation could be compared against a baseline.

The impact of a search application to the conversion rate of 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]. However, it should be noted that even if a link exists between the search results and the conversion rate of e-commerce, it does not necessarily mean that the results are relevant.

The increase in conversion rate when the search is used can lead to a significant amount of revenue for the organization, depending on the size of the e-commerce organization. Additionally, the conversion rate of the e-commerce platform would increase if more customers were steered towards using the search application. However, in this thesis, only the increase of the conversion rate using relevant search results was studied. Some closely related research questions such as how the proportion of customers using the search can be increased are out of the scope of this thesis.

The current search application of the e-commerce platform uses Elasticsearch, which is the platform that the methods used in this thesis were implemented on. Elasticsearch is a large-scale distributed search and analytics engine that provides real-time search and text analysis tools in addition to document storage and indexing capabilities [4, 18]. While building a large-scale search engine is out of the scope of this study, a selection of text analyzing tools and search result ranking methods proposed by Turnbull and Berryman [18] 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 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, which 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, there was no consistent increase in the conversion rate, and, while the

relevance of the search results was increased, the conversion rate was not affected by the change in relevance. Although 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 an introduction of the necessary background from literature. First, two different measurements that are utilized are introduced, both of which are generally used in e-commerce to measure success. Second, the chapter introduces the testing platform utilized to collect the measurements and to analyze the test results. Third, 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 tested in this study and discusses what the relevance of the search results means when the aim is 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. 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].

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

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

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 adding a product to the shopping 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.

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. 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 metric used mostly to measure the performance of a specific element on a website, for example, an advertisement [12]. CTR is calculated by dividing the clicks on the element by impressions, which is the number of visitors who have seen the element [12]. Furthermore, CTR can be thought of as a type of conversion rate where conversions mean different things.

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 when testing the different methods. 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 Production A/B testing

Google Optimize platform was utilized to perform A/B tests during which conversion and click-through rates were measured to determine the performance of the tested method. 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. 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. Google Analytics was used 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 metadata about the performance of products and product lists, so choosing 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. For instance, the completion percentage of the custom goal during a test can be analyzed by calculating the completion percentage from the collected metadata. The primary goals for e-commerce, for example, revenue and conversion, can be tracked with the completion of the custom goals. [6]

2.2 Utilization of a search engine

As a general term, a search engine means an application that reads a query from a user and presents the user with the results for the specified query. The era of modern search engines can be thought to have started when Brin and Page [3] created Google, now the largest search engine. 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. Elasticsearch 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]. The preceding design goals 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 careful engineering work. 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, 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, which performs a search against the inverted index and yields the results for the query.

According to Turnbull and Berryman [18], a search engine consists of two major parts, the index time and the search time. The first three steps given above 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]. The following listings exemplify how the inverted index is created in Elasticsearch [18]. The listing 2.1 includes 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].*

The inverted index in Elasticsearch contains additional metadata necessary during the query evaluation [18]. The following list introduces the metadata necessary for this thesis collected during the indexing phase by Elasticsearch [18].

- *Document frequency* is a count of documents that contain the term, thus establishing a notion of the importance of the term. Generally, a high document frequency indicates that the term provides little value when determining the relevance of the document to a query. For instance, the term "the" is considered as a stop word since it has a high document frequency providing little value for the relevance calculation since it can be found in most English documents.

- *Term frequency* determines how many times a 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.

2.2.3 Index time

This section continues to describe what index time means and how the inverted index is created in 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]. The steps of a ETL pipeline for moving the data from persistent storage to the search engine are shown in Figure 2.1.

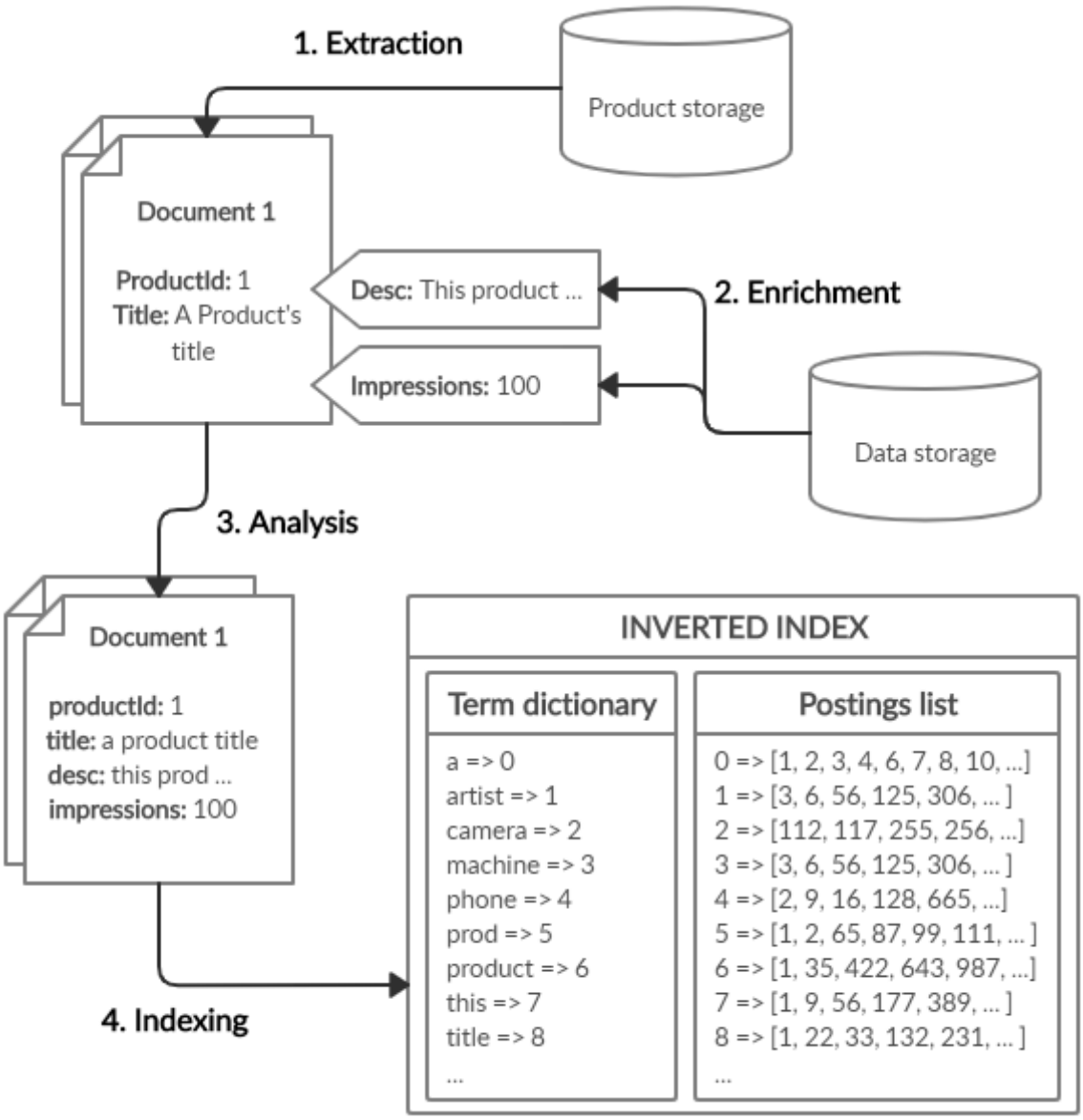


Figure 2.1 *Index time described as an ETL pipeline [18].*

In the extraction step, the product data is collected from the persistent storage. After the collection, the data is enriched with the description of the product and impressions, which is the number of times the product has been viewed. The former is utilized, when matching the search query to a product, the latter, by the ranking algorithm in the relevance calculation.

The analysis happens in Elasticsearch during the indexing. Before the metadata (see Section 2.2.2) is collected from the document, a set of analyzers are utilized to create better terms [4]. For instance, all text from the document is lowercased, and the possessive endings are removed [18]. 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 (Section 2.3.1)

to generate more generic search terms for specific fields.

The last step of the ETL pipeline is to index the analyzed documents. 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 a field becomes available only when the field is indexed. Therefore, a decision which fields to index, which to store, or which fields both index and store is necessary. Storing the fields refers to saving original content to the metadata data structure introduced in Section 2.2.2. [18]

2.2.4 Searching from the index

This section introduces search time and how the documents are searched 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 at the end of a book from which 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. 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 relevance. The search engine sorts the resulting list in an order specified in the query, and generally, the score determines the order of the resulting list. Thus, yielding in a list sorted by the relevance of the search result.

2.3 Elasticsearch as a search engine

This 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.

Elasticsearch provides real-time search and analytics for all types of data. 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. [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. 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 describes, in greater detail, the text analysis tools provided by Elasticsearch utilized in this thesis. An understanding of these 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, which are stored in the inverted index. 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, 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. Without good tokens in the inverted index, the document can be challenging to find with a query that might not exactly match the document. To make the documents more findable, Elasticsearch provides various analyzers by default and provides the capability to create custom analyzers from different tokenizers and token filters [18].

In some languages, such as Finnish, there are various inflected forms for a word, which can make matching the document to the query more difficult. The analyzers can help by stemming the search term, which means reducing the term into its root form [18]. For instance, stemming the terms *foxes* and *jumped* to root forms *fox* and *jump*.

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. By reducing all words from documents and search queries into root forms, the products better match the search terms used by the customers [18]. 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].

The following section focuses on 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 custom analyzer 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]

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. When the features capture both the meaning of the document and the query well, the matches become increasingly relevant. The feature modeling is done in both the index and the search time, so the meaning of the term is captured consistently. [18]

The feature modeling, among many other optimization 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 in the results [18]. Precision is calculated with the following formula

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

where the $doc_{relevant}$ and $doc_{irrelevant}$ are the numbers of relevant and irrelevant

documents in the matches, respectively recall is calculated with the formula

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

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 and vice versa, 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 relevance in the development process. 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. By exactly matching "pesukone" to a term in the inverted index increases precision as only *kuivaava pesukone* category is included, but decreasing recall as the *pyykinpesukone* category is left out from the product lists. Vice versa, if "pesukone" matches all words that contain it, the precision is decreased, and recall increased as all three categories are included.

Teaching a search engine to know that a 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. Generally, similar problems are not easily solvable in practice but somewhat tedious and time-consuming to solve and might need additional feature extraction.

Analyzers

This section introduced different analyzers, which are the tools responsible for collecting the tokens from the documents. 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. Creating multiple indices 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]. 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].

In Elasticsearch, a standard analyzer is used if none other is specified, and it is typically the first analyzer used in a search application [4, 18]. 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].

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, and additionally stemming the words (see Section 2.3.1). The output of the analyzer is the following list of 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]. A search engine is merely a token-matching system that does not know about the meaning of the tokens [18]. The purpose of analyzers is to capture the meaning and the intent of the user to make better tokens [18]. The built-in analyzers may not be enough for specific application domains or may perform poorly on some languages. In Elasticsearch, the tokenizers and token filters, introduced in the following sections, facilitate the building of customer analyzers to target specific issues with the built-in analyzers.

Tokenizers

Tokenizers create the tokens from input words. A tokenizer receives a stream of characters as an input 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 to be saved to the inverted index (Section 2.2.2) [4]. The collected positions are 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]. Two different tokenizers, a standard tokenizer and an edge-N-gram tokenizer, were 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 it also removes punctuation symbols from the input. The standard tokenizer works well in most languages, and it is used as the default in Elasticsearch. [4]

The is edge-N-gram tokenizer 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. N means the maximum 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]

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 problem described above. The aim 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 searchable products. Before the generated tokens are added to the inverted index, a set of token filters should be applied.

Token filters

Token filters 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 by modifying the stream of tokens output by a tokenizer. Furthermore, Elasticsearch provides various built-in token filters [4]. Token filters relevant to this thesis are introduced below.

- **Lowercase token filter** modifies the stream of tokens and converts the characters to lowercase, making the tokens consistent and enabling 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 Scandinavian characters such as 'ä' instead of 'a' 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 need for using the token filter instead of tokenizer is if additional token filters are needed before splitting the tokens into 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 returning otherwise 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. If the search term was longer than the average length search term, which was 6, a subfilter applied 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. Since the usage of this filter may affect the scoring of the documents, it should be used only to bring additional tokens to the index from the fields, where duplicates are irrelevant.

2.3.2 Searching with Elasticsearch

In addition to the text analysis tools, Elasticsearch supports complex queries and provides analyzing capabilities to the queries [4]. This section 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]. A typical use case for aggregations is the filters in lists on websites. 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. Turnbull and Berryman [18] recommend that every website should have a selection of filters in product lists. However, the aggregations are out of the scope of this thesis, since the current search application already utilizes them. The filters also 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 filter context tells whether the document matches a query by returning a boolean value, which is used to filter out the documents from the results [4]. The query context tells how well the document matches a query, providing a relevance score about the match [4]. 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].

Since the relevancy score is calculated based on the query context, it is to differentiate the requirements a document must fulfill to the filter context. Otherwise, it can affect the relevancy score of the document in an unexpected way. For instance, the current search application has multiple rules that a product must match before it should be in the search results. Most of these rules define if a product is active. If the rules are constructed into the query rather than as a filter, they distort the actual relevance score of the documents, which in some cases can lead to unexpected scoring of the results.

Query syntax

Searching in Elasticsearch is made through the Query API, which relies on Lucene [4]. Lucene is a search engine software library developed by Apache. Lucene supports various types of data collection techniques in addition to the basic information retrieval queries [18].

The Lucene query syntax differs from the syntax, traditionally used in information retrieval. For instance, the basic boolean query operators (AND/OR/NOT) are 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*. The same query can also be implemented with the basic boolean operators. Listing 2.9 shows that the query 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 relevance 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 returns no 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, because it provides resiliency against spelling mistakes. Since the search application currently utilizes only MUST operators, 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 that were applied when the index was created, are usually applied to the search terms to make 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].

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 of the language analyzer were caused by not capturing the features from longer than average search term well. The custom N-grams were only generated from longer terms in search time while allowing the generation of smaller tokens in index time. 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.

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 introduce a relevance problem, where the recall increases, and the precision decreases, especially when wildcards are added to both ends of a shorter search term. The current solution adds 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. To solve the problem of providing low precision, an implementation that boosts exact matches by adding a score to them was created. The query ranked more relevant content closer to the top of the search results, thus, increasing the relevance of the top search results.

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]. The similarity is determined by calculating the one-character changes

necessary to match the search term to a token in the inverted index [4]. A fuzzy search creates a set of all possible variations from the search term within a specified edit distance and uses them to find matches from the inverted index [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, such as 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 it did with the edge-N-gram. If the fuzziness was included in shorter search terms, too many irrelevant results were found, due to the static edit distance. In addition, the fuzzy search cannot be used with the wildcard search [4], which the current solution relied on heavily. For these reasons, the fuzzy search was replaced with term suggestion, when no results were found. The results showed that suggesting and automatically searching with a term was effective. Therefore, the fuzzy search might be beneficial and should be investigated further and probably included and tested in the future.

Scoring documents

This section introduces scoring functions that are provided by Elasticsearch and how the score for the documents is calculated. As a base in the content score calculation in Elasticsearch is the following formula

$$term\ frequency\ \times\ inverse\ document\ frequency$$

to calculate how well a term matches a document [18]. A high score indicates that the word is important to a document. Both frequencies are already collected during indexing, as introduced in Section 2.2.2. Optionally, the base calculation can be changed, but in this thesis, it was not necessary.

Turnbull and Berryman [18] argue that relevant search results in the e-commerce context must not only meet the needs of the user but also meet the needs of the business. To achieve this, besides a content score, a business score must be added to the document [18]. To apply additional scores to the documents, Elasticsearch provides built-in scoring functions to create compound queries [4]. 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. The boolean queries are the default query type for combining multiple queries with the query clauses *must*, *should*, *must_not*, and *filter* [4]. The

```
1  { "function_score": {
2      "functions": [
3          {
4              "field_value_factor": { "field": "viewScore", "missing": 0.0 },
5              "weight": 1.0
6          },
7          {
8              "field_value_factor": { "field": "orderScore", "missing": 0.0 },
9              "weight": 1.0
10         },
11         {
12             "weight": 1.0
13         }
14     ],
15     "query": { "bool": { "must": [ ... ] } },
16     "score_mode": "sum"
17 }}
```

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

first three boolean queries map to the Lucene query syntax and last means to the filter context. For instance, each of the query clauses is used in the current rules about product activity, as mentioned earlier. In the boolean queries *must* and *should* clauses are executed in the query context, and *must_not* and *filter* clauses are executed in the filter context [4].

The function score queries are utilized in the ranking function implemented in this thesis. It provides the ability to modify the content scores returned by the main query and takes into account other configured functions when calculating new scores for the document [4]. Figure 2.2 provides an example of a function 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 of several mathematical functions to the resulting content scores from the inner query.

The function score query provides a possibility to define 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. A missing value from the document is replaced with the value defined in the missing parameter [4].

The above formula uses weights for each value when calculating the total score to enable dynamically changing the weight of the different functions [18]. The weights from the formula can also be seen in Figure 2.2. The individual weights are defined inside each function (Figure 2.2), and the last weight in the figure is the content weight w_3 presented. As was stated before, the ranking of the search results is an

```
{ "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.*

optimization problem, and finding the optimal threshold can be automated to find the optimal solution by testing different weights.

Figure 2.3 shows an example query that was used during development. This query utilizes a query string query provided by Elasticsearch. The query string query allows the creation of complex queries that include wildcard characters, searching across multiple fields, and the usage of different functionalities.

The example query defines a structured search across multiple fields as an array of field names. 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 a category and the manufacturer of the product.

The search terms are shown in the query variable. 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 [4]. Elasticsearch does not do wildcard searches by default because of 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. Utilizing the *best fields* functionality may hide other fields that may be relevant [18]. The query in Figure 2.3 introduces a *tie_breaker* variable that softens the *best fields* functionality by introducing a weight with which the other fields are multiplied before they are included in the content score calculation [18]. Including all fields in the content score provided better relevance for the search results during the development phase.

2.4 Ranking the search results

Ranking algorithms are generally an integral part of a search engine, at least in the production scale engines, such as Elasticsearch. Ranking algorithms are also a widely researched topic. This section introduces algorithms that served as a base for the algorithm created for and utilized in this thesis.

There are several different solutions about the ranking of the search results from machine learning methods, such as 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 relevance, 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], who 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.

Utilizing the popularity measure as one of the business scores, when calculating the score for the product in search time, matches the needs of the organization effectively. The algorithm worked similarly, but instead of using the number of transactions as the metric to calculate the relevance rating of a product as in [11], the number of product page visits was used. Furthermore, this provided much more data points to be used to calculate the relevance rating.

2.5 Relevance of the search results

This section introduces the relevance of the search results and how the relevance can be measured in a way so the results can be compared. To be able to serve customers relevant search results to the queries, it is necessary first to understand what relevance is. Turnbull and Berryman [18] defined the relevance 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 relevance 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.

Personalized search solves the problem described above, but since that is out of the scope of this thesis, more general measurements for the relevance were required. The definition given in [18] includes balancing the ranking regarding the needs of the business. This is reflected in the score calculation applied in this thesis. The score is based both on what is relevant to the query and on 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, relevance is purely based on how well the results match the query [18]. However, in practice, relevance 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.

To achieve some general relevance at first, a ranking algorithm that calculated scores for the document based on the popularity rating of the product was introduced. 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 using the data based on the behavior of customers should satisfy the needs of the customers.

3 Current search application

The purpose of the thesis was 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, the purpose of this chapter is to introduce the current search application and how it works from receiving the query from the customer to providing the search results back to the customer.

In addition, the 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]. The survey was also answered by the managers of the current search application responsible for keeping the search ranking up-to-date. The survey also collected the ideas on how the current setup could be improved and what is useful in the current setup.

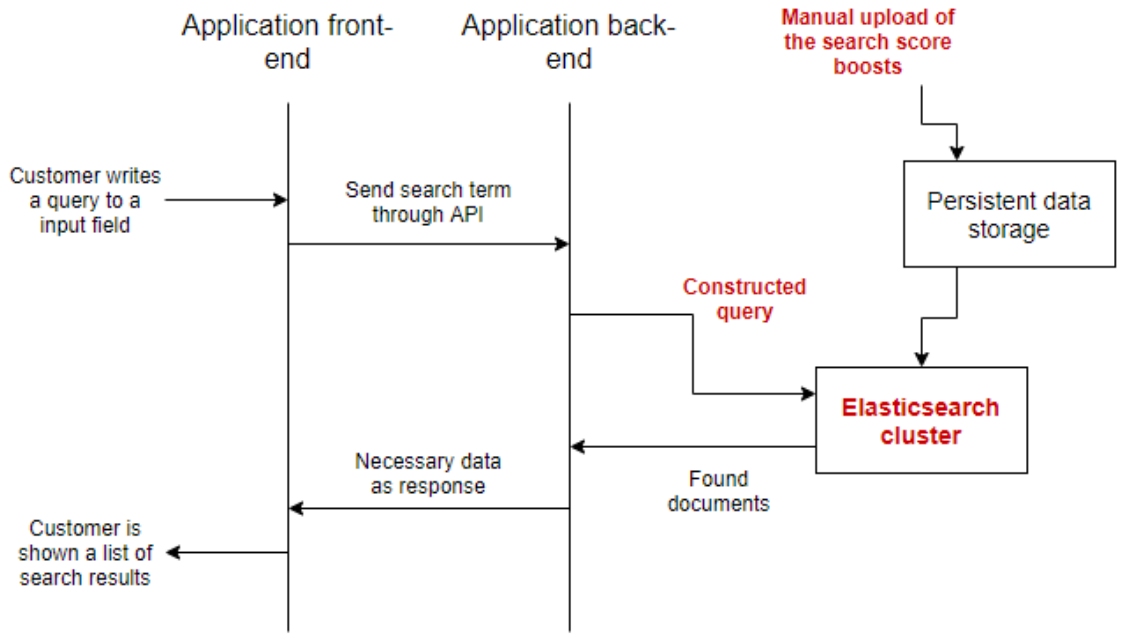


Figure 3.1 *Diagram of the current search application with targeted areas highlighted in red.*

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. The areas related to this thesis highlighted in red.

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. 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.

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.

The current solution indexes documents in an inefficient manner. While indexing a document in Elasticsearch is very fast, it still requires quite a lot of computing power when 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. These analyzers are most likely left there from some previous implementation.

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 seen in Figure 3.2. In some cases, the wildcards in both ends of the search term make sense but may produce inaccurate search results when 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 object shows that the query is 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. Elasticsearch, by default, matches to the best field only, the field with the highest content score. Consequently, boosting a field makes it more likely to be the best field if a match is found from it. It makes sense that a match found from the title of the product should be prioritized over a match in, for instance, sales arguments, it is difficult to say by how much it should be prioritized.

Figure 3.2 shows only a set of fields that were used. The query has some other fields, such as *searchTitle* and *searchTerms*, which indicate that there has been some idea to provide some additional content just for searching. However, the fields seemed either empty in most products or are just generated from other fields in runtime. Showing the urgent need for updating the search application 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.

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.

```
{ "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.*

3.2 Updating the search ranking scores

The current search application ranks the results with a value from a field called *scoreBoost*, shown in Figure 3.2. The field is precalculated and updated to the persistent storage, and it 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 in Figure 3.1, the process behind updating the score boosts field to the persistent storage is a manual process. The process of updating the score boosts is done manually by different individuals for four 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]. The different parameters have been chosen based on empirical tests and 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. The products that are in an upcoming campaign or are advertised may

need special attention [15]. For an expert user, the process takes anywhere from 30 minutes to two hours to complete [15]. 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, the top of the results consist of entirely of IPhones, due to the high field-specific boost shown in Figure 3.2. 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]

4 Methods

This chapter introduces the different implementations done to improve the relevance of the search results, and the tests concluded to measure the effect that each 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. The prerequisite for developing the methods was that if one were to succeed over the current process, the organization would put more effort into developing that method further.

4.1 Implementing an improved search

This section introduces how an improved 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 how the new ranking function was developed and how the score calculation implemented functioned. Since relevance is subjective and personalized search was out of scope, a general solution that tried to capture the general relevance of the search results was implemented.

4.1.1 Creating tokens from content

Here the process of generating tokens from the content of the products is described. As previously mentioned, the token generation is a crucial part of a search engine, which indeed became apparent during the development phase, where a few different options were tested and considered, before finding the best one.

The development was done on the Finnish website, where the product texts 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 was not sufficient. 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 (Section 2.3.2). The first thing where the development started was to investigate how more meaningful tokens could be generated.

A better Finnish analyzer was found, but it was not adopted because it would have been required to be installed in the existing production Elasticsearch cluster, and, furthermore, the analyzer was only for Finnish. The other Nordic languages would have needed a different solution. However, the lack of knowledge about the nuances of these 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 when the number of generated N-grams could be controlled. However, generating tokens with a minimum length of two was not enough to capture the meaning of the words, as quickly became apparent during development.

A similar problem arose during search time. If one of the N-grams 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, and neither should match the other one.

To solve these problems, a custom edge-N-gram analyzer was created (Figure

```
{ "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.*

4.1). The custom analyzer includes two different edge-N-gram analyzers. The first one, *edge_ngram_analyzer*, is used during the indexing of the documents, and the second one, *search_term_over_6*, 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.

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. The search time analyzer uses a conditional token filter, which determines if an N-gram should be created or not. 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. In the 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.

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.

Furthermore, for longer search terms, the analyzer tried to capture the meaning of the terms and match the generated tokens to the inverted index.

4.1.2 Replacing the manual score boosts

The manual process of updating the score boosts was replaced by an automated process to create a more general solution. As the aim was not to create separate search strategies based on, for instance, the behavior of a customer, a precalculated boost value 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 relevance.

To answer the question, what customers see relevant, a popularity rating was introduced. The rating includes only a normalized value of the number of product page views from the previous few days. Since the data was collected and stored by Google Analytics and was not available during indexing, a new integration was required to import the data. A new task was created to import the data every morning from the previous day and to saves the data to persistent storage, where from the indexing process can fetch the data and calculate the value for the boost.

Calculating the boost based on the previous days may not be the most accurate method, at least in e-commerce, since many factors affect the popularity of a product from different trends to many campaigns and promotions. To keep the rating as updated as possible, a custom function, shown in Figure 4.2, was introduced. The function receives an ordered list of the number of views and weight as input and 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 to allow the newer product page views to affect the resulting rating more than the older dates.

After calculating the decaying average, the function uses a natural logarithm to flatten the products with a very high number of views. A very high view count is not that meaningful since it may negatively affect the ranking, by overriding the content score altogether. 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. Normalization also prevents high business scores, which might cause the content score to lose meaning in the ranking functions.

4.1.3 Ranking algorithm

This section introduces how the new popularity rating was utilized in the ranking algorithm, which determines the position of the products in the search results. In

```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.*

total, there were two different ranking algorithms created for this thesis, both utilizing the calculated popularity rating that was 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. The first iteration of the ranking algorithm only used the popularity rating as the business score. 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. The top products in search lists were relevant, but there were multiple cases that the product was not in stock. A relevant search result was provided back to the user, but the search result was not buyable, which explained the decrease of the 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 is calculated as the 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 $score_{content}$. 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.

```
[{ "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.*

The second ranking algorithm, in addition to the updated business score, included multiple queries to boost the exact matching results better. An example of multiple queries is 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 to add a small additional boost to the exact matches.

Elasticsearch allows the usage of multiple queries at the same time within a *should* clause. If a product matches any of the three queries, it will be shown in the search results. The only difference is if the product matches multiple queries, the content score for the product will increase based on how well it matches to the query. Each query has a *boost* variable defined, which allows using different weights for each query. In this setup, the exact queries did not receive a high boost value since it quickly increased precision and decreased recall too much. The selected boosts for each of the queries were the result of the empirical tests done during development.

Each of the queries defines a *tie_breaker* value, which tells Elasticsearch to search across all fields and to utilize all scores in the content score calculation (Section 2.3). Utilizing all fields instead of the best matching field reflected the content of the document better. This approach 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* may be justified.

4.2 Testing the search relevance

The tests were concluded with Google Optimize, which allows running different kinds of tests. In this thesis, simple A/B tests were sufficient enough. Google Optimize requires a script to be loaded on the page, 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. By injecting custom HTML tag to the webpage shown to the customer, the front-end JavaScript code can determine which variant the customer belongs 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 via cookies which the variant of the customer and constructed the query differently based on the variant. 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. Only by constructing the query differently based on a cookie value, different results could be shown to the customers, without any modifications to the code.

Only one of the different variants 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, *Variant B* had only the new ranking algorithm, which used the popularity rating and stock availability. *Variant C* had the new queries for calculating the content score but used the old *scoreBoost* field to calculate the business scores. *Variant D* had both enabled, and *Variant A* was the baseline with nothing new features enabled. The specific test setups across the variants are introduced in detail in Section 5.

4.3 Analyzing the results

Google Analytics is a powerful analytics tool that facilitates the collection and analysis of data for organizations. In this study, Google Analytics collects the data and Google Optimize added variant identifiers to the data. Both products are operated by Google and are used in many organizations across the world.

The metrics introduced in Section 2.1 were used as the primary measurements in this thesis. For an in-depth analysis of the data, Google Analytics provides a variety of functions. The measurements were collected based on the time frame of the tests and the identifier added by the Google Optimize. A closer 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 results reflect the expectations moderately. Although the data was collected across three different websites, the results were easy to differentiate since, for each website, there was an own Google Analytics project.

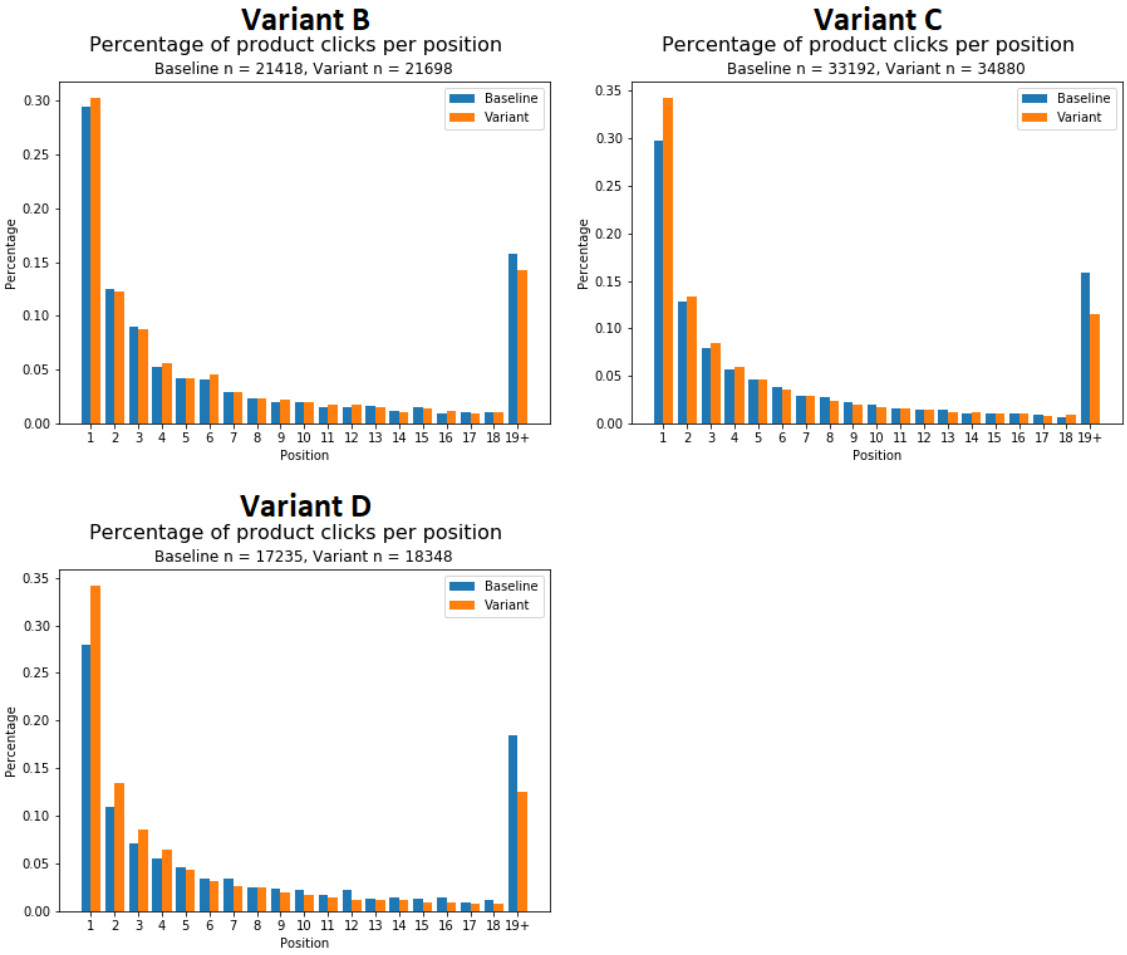
5 Results

This chapter introduces the individual tests, what was included in each of them, and the results collected from the tests. The following results use both the conversion rate and the click-through rate (2.1). The conversion rates in the following tables are the conversion rate of the customers that used the search application. The click-through rate (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. In *Test 4*, the click-through rate included all customers who visited the search result page. 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



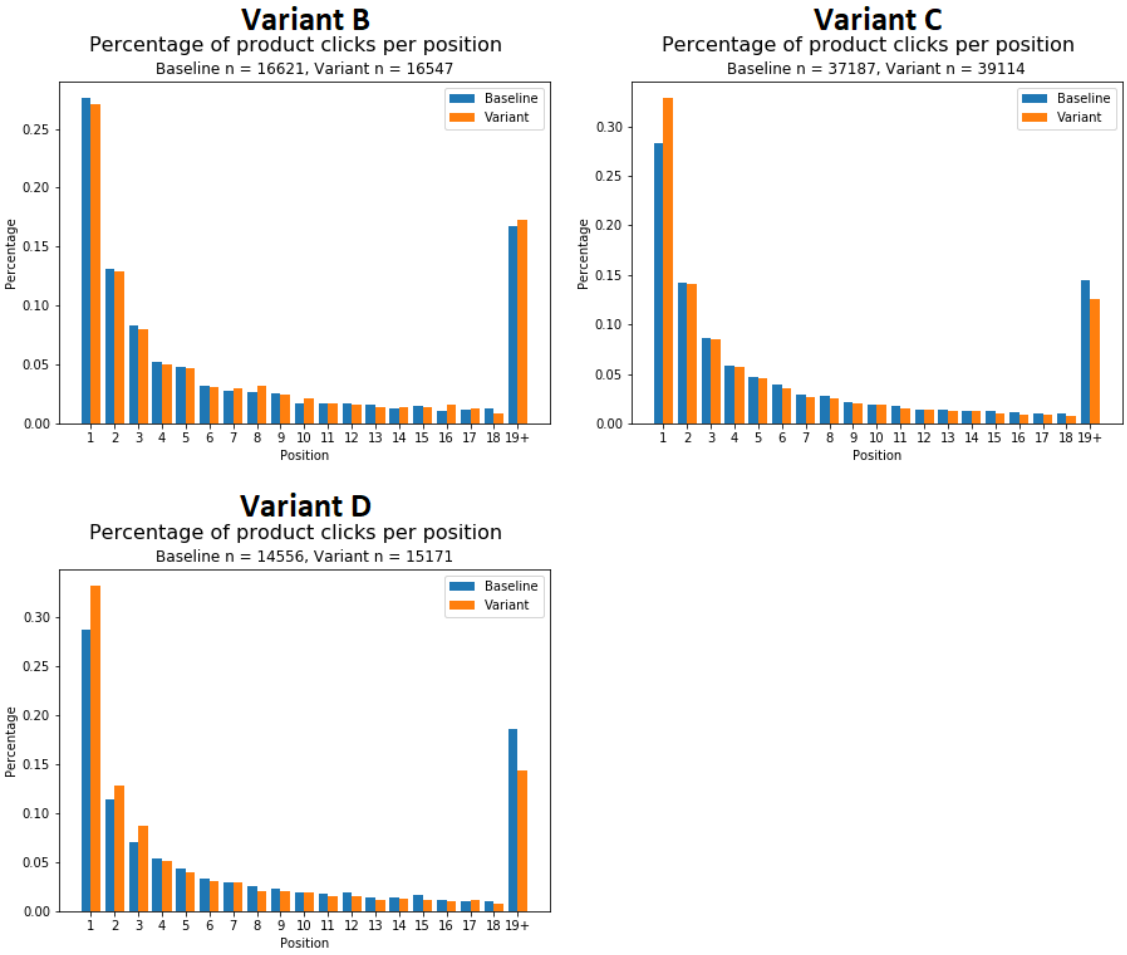
Test 1: Conversion rate from search						
	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
Sessions	59 359	58 587	85 764	85 511	55 246	55 444
Conversion rate	3,32%	3,43%	2,37%	2,19%	1,71%	1,52%

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

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



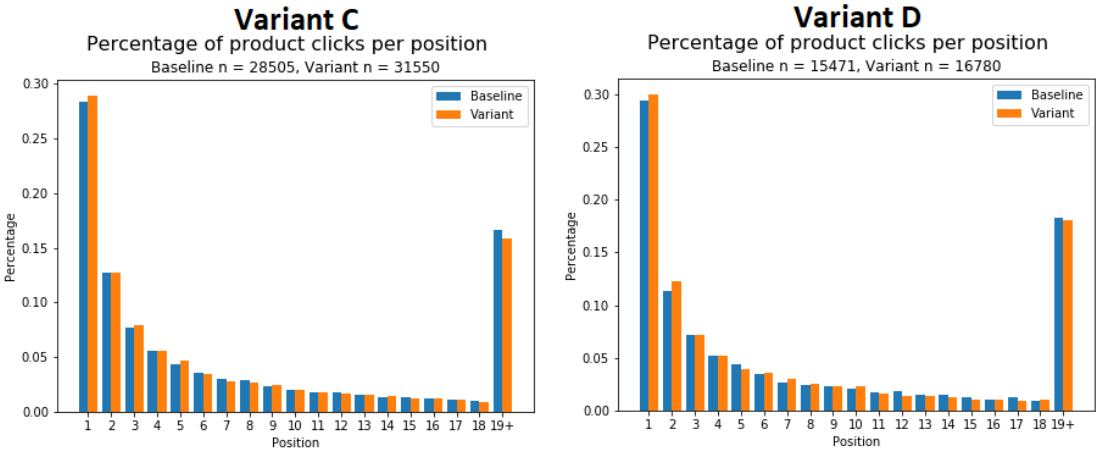
Test 2: Conversion rate from search						
	Variant B		Variant C		Variant D	
	Baseline	Variant	Baseline	Variant	Baseline	Variant
Sessions	47 884	47 866	89 820	88 990	45 472	46 414
Conversion rate	3,04%	3,20%	4,84%	4,95%	1,45%	1,46%

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

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. This meant that by searching with multi-



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.*

ple 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 because 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. For instance, if the customer misspelled a word resulting in zero results, the suggested word would be searched with, and results with the correct term could be returned.

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

Test 4 had the three variants, but in this setup, all the variants utilized the same features. In this test, the three variants mean three different websites in three

Test 4: CTR based on the product position						
Result 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%

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%

Figure 5.4 *Test 4: CTR and conversion rates in the different websites.*

different countries. 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 5.4, show a remarkable increase in the CTR of the search result list. This indicates that the relevance 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*, it even decreased, as shown in Table 5.4. While the solution in *Test 4* seemed to satisfy the needs of the customers, 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.

6 Conclusions

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 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], using them, in practice, in a large-scale search engine with multiple languages proved to be a difficult task.

This thesis 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, but that did not translate into the conversion rate.

6.2 Relevance of the search results

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

While previous studies [11, 18], the conversion rate can be affected by the search results. There are some possible reasons why the increase in relevance did not increase the conversion rate. The customers may have been searching for information about products and not that interested in buying them. Furthermore, the behavior could be explained with the lack of development the search application has seen, which could have affected the behavior of the customers. The explanation could also be that the products offered by the e-commerce platform do not satisfy the needs of the customer base well.

In addition, during development, it was found that the content in the products is not very searchable. The future development could 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. A fuzzy search 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.

In search application development, the goal is to fail fast [18], 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 directly comparable between each other.

In the future, a testing platform 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 in a matter of hours with a testing platform that simulates real customers. While the results might not have been as precise, 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.

While the results of the concluded tests were not optimal as the conversion rate was not increased, 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.

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 International AS. *Internal Analytics of Power*. Internal document (Not public). Checked 20.01.2020.

[15] Power International AS. *Survey on the current status of the search application*. Internal document (Not public). 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.