Bachelor Thesis
Intelligent Scoring Algorithm

Morten Sjælland
mosja14@student.sdu.dk
student no. 40 91 76

Thomas A. Lemqvist
thlem14@student.sdu.dk
student no. 41 02 61

6. Semester Software Engineering
University of Southern Denmark
Faculty of Engineering

Project Supervisor: Sanja Lazarova-Molnar
In Collaboration With: Hesehus A/S

May 31, 2017

Abstract

Within the area of e-commerce and online shopping, there is a need for precise and optimized search scoring algorithms. This is needed to precisely present customers with the products they are looking for and in this way differentiate from the competition.

This report presents the use of association rules mining as an approach to better recommend products to customers based on existing purchase patterns mined from previous orders. In this project, we use simulated product data and generate orders based on already defined purchase patterns. We then present an approach on how to actually utilize the mined association rules in a proof of concept system.

This report concludes, that we have been able to generate sensible association rules and use them for presenting customers with related products.

Keywords: Association Rules Mining, Association Rules, Scoring Algorithm, Elasticsearch, Market Basket Analysis

Signatures

"It is hereby solemnly declared that the undersigned single-handedly, and independently prepared this report. All quotations in the text are marked as such, and the report or substantial parts of it have not previously been subject to assessment."

Morten Sjælland	date	Thomas A. Lemqvist	date

Contents

Glossary

API Application Programming Interface, is a set of clearly defined methods of communication between various software components.

ARM Association Rules Mining.

data driven ranking A ranking based on automated score calculation from various aspects of the data on the products.

Empty Search String The empty search string means an empty string: ””.

GUI Graphical User Interface.

HTTP Hyper Text Transcript Protocol, a protocol used to communicate, and send data via the Internet.

index An index is used by Elasticsearch for storing documents to. An SQL analogy would be a database instance, as multiple documenttypes (sql tables), can be stored in an index, the documents are the counterpart to tuples, or rows in SQL.

MVC Model View Controller.

Normalization getting values of a dataset with different data ranges (i.e. value A spans from 1-10, and value B spans from 1.000-10.000) into the same range, for better comparability.

NoSQL A Database format which does not build on the database standard SQL, instead it commonly stores JSON objects.

RESTful Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet.

score A decimal number used as a Heuristic for search. i.e if a product has a high quality description, in our implementation, it will have a high score on the data quality score, which covers this aspect of the product.

scrape or Web-scraping is the action of pulling a lot of data from internet web-pages.

Acknowledgements

We would like to extend our thanks to our supervisor Sanja Lazarova-Molnar, for guidance in this project.

We would also like to thank our contacts at Hesehus, Bjarke and Lise, for the opportunity to work on this project, and for guidance on Elasticsearch.

Lastly we would like to thank the people who have helped us with proof reading of the report.

CD Contents

Attached to this report is a cd containing a readme file describing how to setup the system and prerequisites.

The CD has the following three folders:

Documents contains an digital version of the report in a .pdf format.

Source Code contains the source code for the project.

Tools contains two essential tools required to run the system. Kibana and Elastcsearch it self. Instructions for use is disclosed in the readme file in the source code folder.

1 : Introduction

This chapter will briefly introduce the report to the reader. It will walk through the background of the project, define the problem it strives to solve, and define the purpose and goal of the project. Finally, it will give an overview of the report structure.

Contents

1.1 Background

This project takes its starting point in a case presented by the company Hesehus A/S which specializes in development of e-commerce platforms for online shopping. In our modern time, online shopping is a popular form of shopping and a lot of different companies are already competing for customers. One of the things these companies can do to differentiate themselves from competition, is to aim for a high customer satisfaction, as happy and satisfied customers are more likely to return to the same company for their next purchases. A key aspect of any webshop is to make sure, that users are able to find the products they are looking for.

Presenting users with relevant products are exactly what Hesehus are hoping to achieve in the case that they have presented. In the case of Hesehus, when searching a database, matches are evaluated with a score, that indicates how good of a match they are. These scores can be manipulated to affect the ranking of matching products, and if this manipulation can be automated in a smart way, this could help improve the quality of the search. Another way of presenting users with relevant products is by looking at what products are typically bought together, and then suggest these products to the user. Both approaches are explained in much more detail in section 2.1.

1.2 Problem

The problem that this project tries to solve, is the fact that the search functionality, provided by Hesehus on their customers' e-commerce platforms, does not always return the most relevant matches. This means, that customers are not always presented with the products they are looking for, which can reduce user satisfaction and company revenue.

1.3 Purpose and Goal

This project aims to improve the search functionality used by Hesehus and help them present more relevant products to webshop customers.
The goal of the project is to do this, by enhancing Hesehus' existing search functionality, which is based on the search engine Elasticsearch. Also, this improved functionality is to be implemented in a proof of concept system demonstrating that it actually works operating on a database with data as close to real life data as possible. We want to improve the search functionality and present more relevant products by making the scoring algorithm more intelligent and present customers with relevant and related products by exploring what kind of products are typically bought together.

1.4 Delimitation

This project is not done on a live website with real users, products, and purchases. Instead, the project will be a proof of concept system, that runs on simulated data.

The system is programmed in C#, and operates on a locally hosted database with products. The ability to search these products will be based on the same search engine as the one Hesehus uses.

This project will focus primarily on suggesting related products to the customer, based on basket analysis.

1.5 Outline

The reader is first introduced to essential background knowledge in chapter 2. Also, related works are presented in this chapter. In chapter 3 the project requirements are defined. Likewise, aspects of the developed software is illustrated with a use case diagram from the perspective of the user. Next, chapter 4 presents the milestones in the project and a risk mitigation plan identifying and discussing chosen risks.

After these introductory chapters, chapter 5 outlines the design aspects of the software implementation by describing software architecture, data generation, and search result scoring using association rules. chapter 6 presents important parts of the code actualization in detail.

The last part of this report evaluates the created association rules and their use in chapter 7 both quantitatively and qualitatively. Lastly, the project is concluded in chapter 8 and results are highlighted. Potential future work directions are also presented in this last chapter.

2 : Background and Related Works

This chapter will introduce the essential background knowledge for this report about Elasticsearch and Association Rules Mining. It will introduce and analyze similar approaches done by other researchers in the related works section.

Contents

2.1 Background

This section describes how search functionality is facilitated by Elasticsearch. The basics of association rules mining is also presented alongside with the Apriori algorithm.

2.1.1 Elasticsearch

Elasticsearch is a search engine and is build on top of Apache Lucene[1]. Lucene is a full-text search engine library with a lot of features, however it can also be difficult to understand. Elasticsearch uses Lucene internally, but aims to make full-text searching easier and hide the complexities with a RESTful API.

Elasticsearch can run on your local computer, on a remote host, or distributed amongst multiple hosts, and communication can be via a web client, through a programming language, and also from the command line. [GT14]

Storing data

The data in Elasticsearch is stored as JSON documents. These documents are stored in a NoSQL database style in what is called an index, which bares much resemblance to a SQL database.

The RESTful API connects to an Elasticsearch cluster, which is a collection of nodes, and can have multiple nodes to scale as needed. These nodes are instances of Elasticsearch. They work together

[1] https://lucene.apache.org/

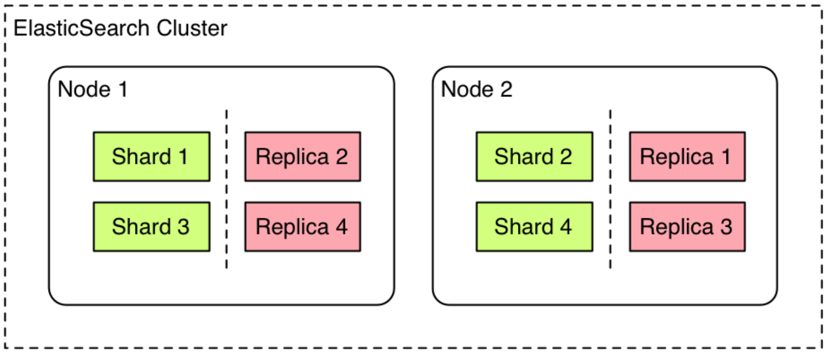


Figure 2.1: Example of an Elasticsearch setup
source: https://blog.liip.ch/archive/2013/07/19/on-elasticsearch-performance.html

to share workload and data evenly. Each of these Elasticsearch nodes contains shards, which are the workers in the system. There are two types of shards, the search shards, which are used for searching, using Lucene, and replica shards, which are backups of the search shards. The shards accesses the data through the index, and can perform full text search on the data stored within.

Figure 2.1 shows an example of an Elasticsearch cluster containing two nodes, which each has two shards, with one replica shard each. When searching this Elasticsearch cluster, the search request goes to whichever node is available, and the search is then performed by the shards inside the node.

Searching

When searching it is also possible to query documents for a specific word, a field, or the value of a specific field. For Elasticsearch to understand and interpret the document structure correctly a *mapping* is used. The mapping specifies what type the different fields in the document are, for example string, number, or boolean.

To improve the efficiency of searching through multiple documents, Elasticsearch can use different kinds of *analyzers*. These analyzers do two things. Firstly, they break down the text present in the document into more manageable individual terms. Secondly, they normalize the terms into more search friendly terms. [GT14, page 85-89]. For example, think of a sentence describing some sort of product. Firstly, this sentence could be broken down into single words. Secondly, typical stop words like "*a*", "*the*", and "*and*" are removed and the words are stemmed down to their base form. There are a lot of these analyzers which can be combined and customized as needed. Examples of build in analyzers include language analyzers customized to different languages.

2.1.2 Association Rules Mining

The idea of association rules mining is to find and understand associations between items in large data sets. It is also known as Market Basket Analysis, due to the fact that this technique is often applied to supermarket data. When applied to supermarket data, it can learn purchase patterns. [Lan15] The result of doing association rules mining is a collection of association rules that describes patterns found in the dataset. These rules have the general form $\{X\} \rightarrow \{Y\}$, where X on the left hand side is an itemset that leads to the itemset Y on the right hand side. An example of a rule that could appear in a dataset of supermarket purchases is shown below:

$$\{Chips,\ Dip\ mix\} \rightarrow \{Creme\ fraiche\}$$

This rule specifies that if chips and dip mix appear together in a purchase, it is likely that creme fraiche will also appear in this purchase. However, one of the challenges when using association rules mining is to actually identify useful and interesting patterns. How to measure the interest of a rule is described in further detail in subsubsection 2.1.2.

The Apriori property

A problem with using association rules mining is the amount of processing power required. This is clear as the number of potential itemsets to be evaluated grows exponentially with the number of items. This means that given k items that can be bought, 2^k potential rules exists. To counter this problem, a smart rule-mining algorithm uses the fact that a lot of potential combinations of items are rarely bought together. Therefore these items can be ignored when searching for rules to consider. This helps to greatly reduce both the number of potential rules and the computational time needed.
In practice, to limit the number of rules to consider, you will need to apply some sort of measurement. A widely used measurement is the *Apriori property* which dictates, that all subsets of a frequent itemset must also be frequent [Lan15]. To illustrate, consider the example set $\{car\ battery, hair\ spray\}$. This itemset can only be frequent enough to consider, if both hair spray and car batteries are sold frequently.

Rule interest

The interest of an association rule when using the Apriori algorithm is defined by two measures, *support* and *confidence* respectively. The support of an itemset or a rule measures how frequently an itemset occur in the data searched and is defined for an itemset X like below where N is the number of transactions in the data.

$$support(X) = \frac{count(X)}{N}$$

As an example if an item, say chips, occurs in 20 out of 30 purchases, the support of that item is $\frac{20}{30} = 0.67$. The confidence of a rule tells the proportion of purchases where an item or an itemset X or is purchased together with an item or itemset Y. The confidence of a rule $\{X\} \rightarrow \{Y\}$ is defined as the frequency of sets containing both X and Y divided by the support of the left hand side Y.

$$confidence(X \rightarrow Y) = \frac{support(X, Y)}{support(X)}$$

Returning to the chips example, say chips are bought together with dip mix 9 out of 30 purchases. The confidence of the rule $\{Chips \rightarrow Dipmix\}$ is calculated as $(9/30)/0.67 = 0.45$ Basically, the confidence of a rule is how accurate its predictive power is. A confidence of 0.45 means that 45 % of the time when chips are bought, dip mix is also bought.

By defining minimum thresholds for both the support and the confidence of a rule, the number of rules presented is greatly limited. However, defining these thresholds is a trade-off between getting only the most obvious rules and getting a lot of rules to look through.

A third measurement to use when considering the interest of a rule is called the lift of a rule. Lift measures how much more likely an item or an itemset is to appear in a transaction, given that another known item or itemset is already purchased, relative to its typical rate of purchase. [Lan15] It is defined as the confidence of the rule divided by the support of the right hand side of the rule like below.

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$$

To calculate the lift of the example rule $\{Chips \rightarrow Dipmix\}$ say dip mix is bought in 10 out of 30 purchases. The lift of the rule is then calculated to be $0.45/(10/30) = 1.35$. This means, having added chips to your basket you are 1.35 times as likely to also add dip mix. If the lift of a rule is greater than one it is a strong indicator that the rule is important and should be considered.Therefore, the lift of a rule is an obvious measure to consider, when presenting the rules found, such that the most interesting ones are presented first.

The algorithm

The Apriori algorithm is made out of the two following steps.

1. All itemsets that meet a given minimum support threshold are found
2. The algorithm generates rules from these itemsets and evaluates the rules with respect to a given minimum support and - confidence.

Step one is done in multiple iterations, each time finding itemsets of increasing size. For example, in iteration one, the algorithm finds itemsets of size 1, iteration two finds itemsets of size 2, etc. In general, iteration *i* finds itemsets of size *i*. The itemsets which exceed a minimum support threshold in iteration *i* are then joined to generate candidate itemsets for evaluation in iteration *i+1*. The algorithm stops when no new frequent items are generated in an iteration. As an example, say iteration 1 finds only two frequent itemsets meeting the minimum required support {*A*} and {*B*}. Iteration 2 then evaluates itemset {*A, B*} and if this itemset is not frequent, step one in the Apriori algorithm is over.

Based on the frequent itemsets found in step 1, association rules are generated from the possible subsets. For example, if the itemset $\{A, B\}$ had been found frequent in step 1, the potential rules $\{A\} \rightarrow \{B\}$ and $\{B\} \rightarrow \{A\}$ are evaluated with respect to a given minimum confidence, as described in subsubsection 2.1.2 [Lan15, chapter 8]

The Apriori algorithm was presented by Agrawal and Srikant in 1994, alongside with the pseudo code of the Apriori algorithm in algorithm 1. In the first pass, all itemsets of size 1 are found, denoted L_1. A following pass *k* use the previous pass *k-1* to generate candidate itemsets C_k. Next, the support of all candidates in C_k are counted, by scanning all transactions *t* in the database denoted *D* and checking if candidate *c* is a subset of the given transaction *t*. When no further itemsets are found, all found frequent itemsets are returned. [AS94]

```
Data: $L_1 = 1 - itemsets$
for $K = 2; L_{k-1}! = \emptyset; k++$ do
    $C_k = Apriori - gen(L_{k-1})$;
    for all transactions $t$ in $D$ do
        $C_t = subset(C_k, t)$;
        for all candidates $c$ in $C_t$ do
            c.support++
        end
    end
    $L_k = \{c \in C_k | c.support \geq min.support\}$
end
Result: $\cup L_k$
```

Algorithm 1: The Apriori pseudo code as presented in [AS94]

2.2 Related Works

This subsection will present examples of related work. In the project, we have researched this work and used it for inspiration.

2.2.1 On-Site search Design Patterns

To better understand how to actually use Elasticsearch for storing and searching data, we have studied the work of Dr. Martin Loetzsch and Krešimir Slugan in the article ”*On-Site Search Design Patterns for E-Commerce: Schema Structure, Data Driven Ranking & More*”. [LS]

In the article, Dr. Martin Loetzsch and Krešimir Slugan describe how they have configured Elasticsearch and used it for providing search functionality at contorion.de, an online e-commerce platform. The article also covers topics like how to show alternative search results when a user misspells a term, how to use search suggestions before the user is finished writing the whole search term, and how relevant products can be found using a full-text search. The article also has a very interesting part about how to rank products based on different kinds of attributes, which we have also applied in our scoring of the search results.

Building a search index

For products to be searchable using Elasticsearch a document defining the product must be present. This document could contain attributes like the product name, manufacturer, a description, a prize, and multiple other attributes. However, Loetzsch and Slugan suggests an approach somewhat different when creating these documents. Other than the basic attributes mentioned before, they add multiple other fields optimized for searching. Examples include suggestion- and completion terms that are suggested to the user when he or she is typing in a query. Static sorting is provided by defining a number_sort and a string_sort attribute, so that products are sortable by both name and price. Each document describing a product also contains information about what categories the product belongs to. Loetzsch and Slugan also uses two full-text search fields in the documents, full_text and full_text_boosted respectively. These fields contain all the text for which the product should be found during a free text search. Another important field defined by Loetzsch is what he calls ”facets”. Facets define all attributes for which search results can be filtered and grouped. A facet could be the price of the product, enabling a user querying the website to only see products in a specific price range. Lastly, a scores field is contained in each product document which indicate, amongst other things, the product quality, past sales, and product popularity. These result ranking scores are described in the next section Data-Driven ranking. To see the full document structure of the search index used in the article, see Appendix A.

Data-driven ranking

It is crucial that customers are presented with the most relevant search results when using a website, as this should increase customer satisfaction while also increasing company revenue. An approach recommended by Loetzsch and Slugan is to sort the search results based on predefined scores on each result. The scores used by Loetzsch on contorion.de are presented in Code snippet 2.1.

```
{
   "scores": {
      "top_seller": 0.91,
      "pdp_impressions": 0.38,
      "sale_impressions_rate": 0.8,
      "data_quality": 0.87,
      "delivery_speed": 0.85,
      "random": 0.75,
      "stock": 1
   }
}
```

Code snippet 2.1: The scores used on contrion.de

Each of the scores in Code snippet 2.1 aims to define how good a product is with respect to different parameters. Some of these scores are explained in detail in table 2.1.

Score	Description
top_seller	The top_seller score is based on how many units are sold for a specified period of time.
sale_impressions_rate	The conversion rate of the product. That is, how much the product sells with respect to how many times it is visited on the website
data_quality	As the name suggests, a score that describes the quality of the data of the product. This could be the descriptions, how many images are present, etc.
random	Used to boost new products, that might not have a high score in top_seller and sale_impressions_rate as they are completely new products
stock	Used to make sure, that products not currently in stock is not shown at the top of the search result. If the product is in stock, the score is 1, if not it is close to 0

Table 2.1: An explanation of some of the scores used by Loetzsch

These scores are present on all documents in the index, and should be updated regularly, to reflect the latest data. The approach to this updating taken by Loetzsch is to rebuild the search index and update the scores every night. A great thing about these scores is that you can of course come up with your own scores depending on what kind of parameters you want to take into account.

These scores can be utilized in Elasticsearch searched by using a scoring script as shown in Code snippet 2.2.

```
{
   "script_score": {
      "script": "
      (1 + _score ** 0.5)
      * doc['scores.stock'].value
      * (0.1 * doc['scores.random'].value
      + 0.3 * doc['scores.top_seller'].value
      + 0.1 * doc['scores.pdp_impressions'].value
      + 0.2 * doc['scores.sale_impressions_rate'].value
      + 0.1 * doc['scores.data_quality'].value
      + 0.3 * doc['scores.delivery_speed'].value)
      "
   }
}
```

Code snippet 2.2: Using the scores in Elasticsearch searches

Code snippet 2.2 shows the script used by Loetzsch. It takes into account the _score, a score used by Elasticsearch for evaluating how closely a document match a query. By multiplying it with the stock score, it is possible to eliminate any products that are not in stock, as this will result in a total score multiplied with a stock score close to zero. An important thing also mentioned in the article, is that these scores must be normalized to make any sense. This also enables us to weigh the different scores, which can also be seen in picture 2.2. The weights presented in the article are, however, random numbers replacing the real weight, as these are trade secret.

2.2.2 Market Basket Analysis

Market Basket Analysis is about discovering customer behavior trends and spot what kind of products are often purchased together as described in subsection 2.1.2. There has been a lot of research in this area, some of which are pointed out in this section.
One paper by Loraine Charlet Annie and Ashok Kumar explores the use of the K-Apriori algorithm to mine frequent itemset in retail organizations. [AK12] The K-Apriori algorithm is based on the Apriori

property, described in subsubsection 2.1.2. It categorizes customers in clusters and find the frequent itemsets based on customer transactions in these clusters. Using this approach on transaction data from a retailer called Anantha Stores, Ashok and Loraine conclude that the K-Apriori algorithm is capable of generating frequent itemsets and association rules. These rules can be used by the employees at Anantha Stores to manage placement of related products and goods in their stores. By grouping customers together and generate association rules separately, the rules become more specific and promotions can be targeted at specific customer groups. Lastly, the paper concludes that *"(...) it is shown that the market basket analysis using K-Apriori algorithm for Anantha stores improves its overall revenue."* [AK12]

Another paper by Harpreet Kaur and Kawaljeet Singh explores the use of association rules mining in a sports store. [MPA16] The biggest difference from Loraine Charlet Annie and Ashok Kumar's work [AK12], Kaur and Singh uses an algorithm called Frequent Pattern growth (FP-growth) instead of the Apriori. The FP-growth algorithm is presented in more detail in subsubsection 2.2.2. In their research paper, Harpreet Kaur and Kawaljeet Singh concludes that the FP-Growth algorithm efficiently mines association rules for the sport store, which can the use them to try and maximize their profit. [MPA16]

Although association rules are typically used for market basket analysis to increase revenue, it is possible to use it for a lot of other things. Examples include finding behavioral patterns leading to customers dropping their cellphone - or Internet service provider. Another example is using the rules to finding patterns, purchases, or medical claims that often occur in combination with credit card or insurance fraud. [Lan15]

Reducing required processing power using sampling

When doing market basket analyzing, using the Apriori algorithm is not very helpful for small datasets. [Lan15, chapter 8] Therefore, typically market basket analysis is done on large datasets which will require a significant amount of processing power. Besides processing power, scanning a large database also requires a lot of disk operations. An approach to minimize the required processing power and disk operations is proposed by Mohammed Javeed Zaki, Srinivasan Parthasarathy, and Wei Li in the paper titled Evaluation of Sampling for Data Mining of Association Rules. [ZPLO96].

The approach presented in this paper, is to use sampling instead of going over the total set of transactions, which will reduce the number of transactions considered. In their article they have used random sampling to create a smaller, more manageable, dataset from a database upon which the Apriori algorithm is run to mine association rules. Underlining a trade-off between the performance and the desired accuracy or confidence of the sample and the fact that a small sample size may generate false rules, the article concludes that *"(...) we claim that for practical purposes we can use sampling with confidence for data mining."* [ZPLO96] Reducing database passes can also be obtained by using alternatives to the Apriori algorithm, which are described in subsubsection 2.2.2

Another interesting performance optimization is parallelization as mentioned in [KK06], as this offer a greater capacity and speed. Multiple algorithms capable of parallelization exist, one of which is an algorithm called Fast Distributed Algorithm for Mining Association Rules (FDM), a parallelization of Apriori. The FDM algorithm enables multiple machines, each with its own database partition, to do association rules mining in cooperation and in parallel. [KK06]

Other Algorithms

This section describes two association rules mining algorithms, other than the Apriori algorithm. The algorithms described are the Frequent Pattern Growth algorithm and the TreeProjection algorithm.

Multiple alternatives to the Apriori algorithm have been proposed. One of them is the FP-Growth algorithm, which aims to reduce the number of passes over the database to only two passes as opposed to Apriori, which will pass over the database in every iteration. [KK06] The frequent pattern generation in the FP-Growth algorithm consists of two sub-processes. Firstly, the FP-algorithm generates a structure called an FP-Tree where each node represents an item bought and its count. That is, the number of times it is bought. Secondly, the frequent patterns are generated from this FP-tree. Each branch in the FP-Tree represents a different association which, like in Apriori, is evaluated with respect to some threshold. [FA] The result of the FP-Growth algorithm is a compressed representation of the original database, containing only the frequent items. [KK06]

Another example of an Apriori alternative is the TreeProjection algorithm. The idea behind the algorithm is to construct a lexicographical tree and from a large database create a smaller sub database which is based on the association rules already mined. The nodes in the created tree then represent each frequent itemset. [KK06]

An interesting approach supposed by [KK06] is the use of temporal association rules. The idea is, that they can be more useful than basic association rules, as they provide more information For example, they can take into account the hours of the day that a rule has the highest support, which will, of course, help retailers strategize sales and promotions even better.

We have gone through the theoretical knowledge about Elasticsearch and Association Rules Mining, and introduced works from other researchers, and analyzed how they have incorporated these techniques in their systems.

3 : Requirements

This chapter will introduce the requirements established for the system. Some of these requirements are then illustrated as use cases.

Contents

3.1 Requirements

This section will define and describe the requirements we have set up for the system, in form of functional - and non-functional requirements.

The requirements are based upon meetings with Hesehus, where we in collaboration have established what services the system should provide. The requirements will later be used for establishing milestones in the project.

3.1.1 Functional Requirements

The functional requirements define the services the system provides. We have specified the requirements in the user requirement format, this means that they are not very technical. They aim to be brief and easy to understand, without going into too much detail [Som10].

Functional Requirements

FR.1 The user must be able to query the products using a search phrase.

FR.2 Product Query Requirements
The user must be able to:

- FR.2.1 See products from the query.
- FR.2.2 Select products from the query.
- FR.2.3 Add products from the query to his basket.

FR.3 Basket Requirements
The user must be able to:

FR.3.1 See products in his basket.

FR.3.2 In- and Decrease the quantity of individual products in his basket.

FR.3.3 Remove a product from his basket.

FR.3.4 See the total Price of the products in his basket.

FR.4 The products should be contained in a search optimized index (search index)

FR.5 The search index should contain scores for various parts of the product, based on data on the product.

FR.6 Scores on the products:

FR.6.1 Data quality score, a score that represents the amount of data we have for each product.

FR.6.2 Stock score, a score that is equal to 1, if the stock is above a chosen minimum stock threshold, and 0.001 if its below. This is to score products which are out of stock lower.

FR.6.3 Top Seller score, a score based on sales of this product for the last two weeks.

FR.6.4 Visits score, a score based on how many visits, or clicks, a product has had.

FR.6.5 VisitConversion score, a score based on how many sales the product has pr visit.

FR.6.6 Random score, a score which allows new products, with no ”historical data”[1] to score higher.

FR.7 The system must be able to rebuild the search index on demand.
When rebuilding the search index the system must:

FR.7.1 Recalculate the scores, based on the latest data changes.

FR.7.2 Mine Association Rules, to update the rules.

FR.8 Association rules:

FR.8.1 The rules generated must be realistic.

FR.8.2 The rules should map categories together.

FR.8.3 The rules must contain support, confidence, and lift variables.

FR.8.4 The rules must be usable for recommendations, based on the basket.

FR.9 The search algorithm, must make recommendations based on items in the basket using association rules mining.

3.1.2 Non-Functional Requirements

The non-functional requirements set requirements for the system as a whole, the development process etc., not the functionality [Som10].

NFR.1 The implementation must make use of Elasticsearch.

NFR.2 The code language for the implementation must be C#.

NFR.3 We will have to generate our own test data, as Hesehus will not be able to provide any such data.

NFR.4 The system must apply artificial intelligence or statistical approaches, to improve the scoring algorithm.

NFR.5 The scores in the search index must be normalized, to ensure better comparability.

[1] sales and visits etc.

NFR.1 was set by Hesehus, as they wanted to see what we could come up with.

We chose to develop the system in C# because this is what Hesehus uses, for their platform. But as this is a proof of concept only, the choice of language does not matter as much, as if it was a production system.

3.2 Use Cases

This section will describe the actors and use cases in the system. The use cases are based on the requirements specified in the previous section. They were made to determine specifically what services the system should provide a user of the system, and to give a clear image of the features the system will have, from a users perspective.

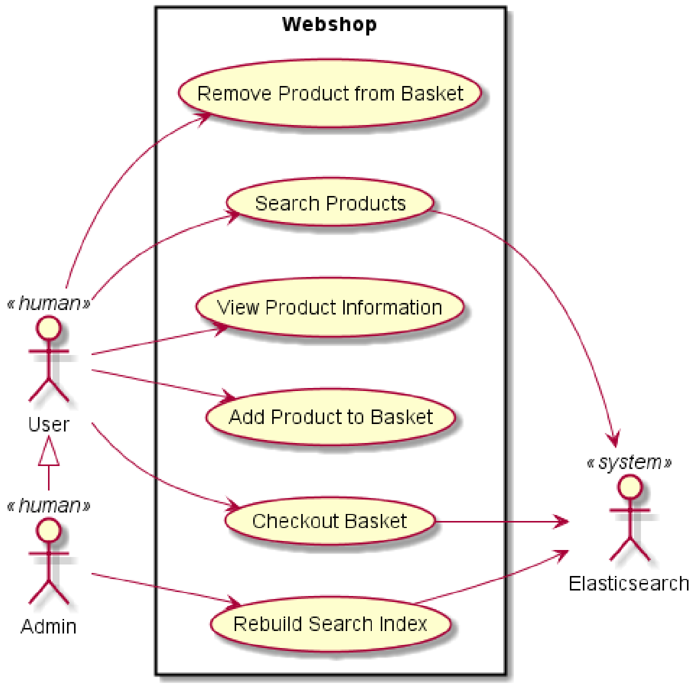


Figure 3.1: Use Case Diagram, which shows the use cases and actors of the system

Figure 3.1 shows the use case diagram, on which we can see the actors and the use cases of the system. It also shows what services the actors have access to.

3.2.1 Actors

Figure 3.1 shows the use case diagram for the system, and define three actors of the system: the User, Admin and Elasticsearch. These are defined as the following:

User is the customer, using the web-shop to purchase items with. He has access to the search functionality, and are able to manipulate his basket and purchase the items in his basket.

Admin has all the rights the customer has as well as the option to rebuild the search index from the database. Ideally over time this responsibility would be set to a timer, so that the search index would be rebuild on a nightly basis.

Elasticsearch is the Elasticsearch instance used by the application. It has the responsibility of serving the users with products from the search result. Also in our case, Elasticsearch has the responsibility of storing the data on the products[2].

3.2.2 Use Cases Description

Table 3.1 contains a description of the use cases shown in Figure 3.1. It adds more detail to the use cases, and makes it easier to measure if this use case has been implemented, and thereby if the requirement is fulfilled.

Table 3.1: Use Case Description Table

Use Case	Description
Search Products	Users can search the product catalog, using a search phrase in an input field.
View Product Information	The Search will yield the user products, and the user will have the possibility to view information, such as description and price, about the products.
Add Product to Basket	If a user wants to purchase a product, he has the possibility to add the product to his basket.
Remove Product from Basket	If the user regrets adding a product to his basket, he can then remove it from the basket again.
Checkout Basket	When the user has found all the items he wants to buy, he can purchase the items in the basket.
Rebuild Search Index	The Admin can update the product data on the search index, to accommodate changes in the dataset and update the scores based on recent product sales, views, etc.

We have established the requirements for the system, and set up some use cases.

[2] More on this in subsubsection 5.1.1

4 : Project Planing

This chapter lays out an overview of the project process by presenting major project milestones. It also discusses some highlighted risks identified in a risk analysis and what we have done to mitigate the effects of these risks.

Contents

4.1 Project Planning

This subsection presents the milestones of the project and outlines the project process. Throughout the project, regular meetings with the Hesehus representative have been scheduled every other week. Here, council has been given regarding problems, if any. Also, agreements have been made on what should be done for the next meeting. This has given the project a Scrum-like and agile structure, where we have been working on the agreed tasks for two weeks, presenting them to Hesehus, and agreeing on new ones.

4.1.1 Milestones

This section presents the project milestones, which provide an overview of what is to be done in the project. Table 4.1 lays out these milestones below.

Table 4.1: The Milestones in the Project

ID	Name
M1	Data acquisition
M2	Set up software infrastructure
M3	Provide search based on the Elasticsearch engine
M4	Implement association rules mining
M5	Implement genetic algorithms in the scoring algorithm

Milestone **M1** Data acquisition is the first milestone to achieve, as data is a fundamental part of the project. We needed to either find or generate data, as we otherwise would have no data to search through and try out the searching functionality on. However, this milestone was troublesome to achieve, as discussed in subsection 4.2.1. We ended up creating a small dataset of products to achieve this milestone. This is described in more detail in section 5.2.

Next, as the product of this project is a proof of concept system, milestone **M2** is to setup the infrastructure and implement a foundation that we can work on. The result of this is a mini version dummy web-shop where products can be searched, viewed, and bought. section 5.1 describes the developed infrastructure in greater detail.

Milestone **M3** is to actually implement a search functionality based on Elasticsearch. Again this is a fundamental part of the process, as we are building upon the core tools provided by Elasticsearch. As a part of this, data- and search indices need to be defined and setup in Elasticsearch which is examined in section 6.1.

M4 implementing association rules mining and using it in the search is a very important aspect of the project, as this is were we can actually provide some enhancement to the current searching experience on Hesehus' customer's websites. To achieve this, we need to be able to actually mine the rules from purchase transactions. Scoring and creating the search index with use of the found association rules are examined in section 5.3 and section 5.4. Mining the association rules and using them are described in further detail in chapter 6

Lastly, **M5**, an interesting way to artificially increase the intelligence of the scoring algorithm by applying genetic algorithms to determine what weights of the scores presented in subsection 2.2.1 are optimal for increasing revenue. However, we have been unable to implement this yet for reasons described in subsection 4.2

4.2 Risks

In the start of the project we carried out a risk analysis and did a risk mitigation plan. All of the risks identified have been given a severity level on the scale of 1-5 describing their possible influence if they occur. For example, a level 1 risk can easily be ignored, while a level 5 risk requires immediate attention. This severity is based on both the risk likelihood and the risk impact if it happens. The table used for this scoring and the full list of the risks identified can be found in Appendix B. The following three risks are highlighted and commented on in this subsection, as these have had a great influence on the project

1. r1 Hesehus is unable provide any test-data
2. r7 Associations rules mining fails to find any real associations in the dataset
3. r4 The database structure changes

4.2.1 r1 - Hesehus is unable provide any test-data

This is the risk that has had the biggest influence on the project. An overview of the risk is presented in table 4.2. As it can be seen from the table, this risk has a medium likelihood and a high impact resulting in an estimated severity level of 4 meaning that it is a pretty serious risk. If it occurs we will have to generate our own simulated data.

Early on in the project this risk became a reality, as we were not able to get any test data from Hesehus in the form of product information and purchase history to experiment on. As a result, we have had to follow the mitigation plan defined in table 4.2, that is, to create our own data based on an ER-diagram

Table 4.2: Risk one identified in the risk analysis

ID	r1
Risk	Hesehus is unable provide any test-data
Overview	We need data for testing, this data must be realistic, and have the correct attributes, to be usable in a real-world-scenario.
Likelihood	Medium
Impact	High
Severity	4
Mitigation plan	To mitigate this risk, we will ask Hesehus for a ER-diagram2, which we will use to generate dummy data, which follows this diagram. This can potentially lead to risk r2.

provided by Hesehus. As we wanted to have somewhat realistic data, we first decided to scrape the website komplett.dk for products using a the .NET code library HTMLAgilityPack[1]. It allowed us to traverse HTML documents on komplett.dk and use the product information to create our own product database. When implemented, this allowed us to generate hundreds of realistic products. We ended up using a lot of time on this, which could have been avoided if we had been provided test data. To make matters worse, the web-scraper stopped functioning due to it being blocked by komplett.dk. This meant, that we had to allocate extra work hours to rewrite the scraper to target wupti.com instead. It is due to this lack of provided data that requirement NFR.3 was introduced.

4.2.2 r7 - Associations rules mining fails to find any real associations in the dataset

Another risk that had a great influence on this project was risk r7, that we were unable to mine any meaningful association rules from the dataset. This risk, like risk r1, has an estimated severity level of 4, meaning that it had a lot of impact when it happened. As we were not able to get any transaction history from Hesehus to do association rules mining on, we had to generate a lot of simulated orders ourselves. Unfortunately, as we had a dataset of hundreds of products, the dummy orders that we made were too random, and association rules mining on these orders did not manage to find any meaningful associations between the products. We ended up creating a much smaller, more manageable, set of products, enabling us to have more control on what kind of orders were generated, which resulted in more meaningful associations. This process is described in greater detail in section 5.2.

4.2.3 r4 - The database structure changes

This risk has a severity level of 3 and also has a lower impact than risk r1 and r7. We started out by generating an Elasticsearch index describing our products. However, we found out that an extended index optimized for searching was needed and therefore we needed to structure this index also. These indices are described in subsection 5.1.3.

We have presented the project milestones and how these are to be achieved. Furthermore, we have discussed the impact of essential risks like not getting any test data, not finding any real association rules, and having the database structure change.

[1] HTTP://html-agility-pack.net/?z=codeplex

5 : Design

This chapter will introduce the design decisions made in the project. It will start with the system architecture decisions, then proceed to the data generation decisions made, and then go into generating the search index, using normalization and the Association Rules Mining. Lastly decisions made for the scoring with Association Rules are presented.

Contents

5.1 Architecture

This section will describe the design decisions made for the architecture of the system. It will describe the Model View Controller (MVC), and how we have implemented the MVC design pattern.

5.1.1 Model View Controller

We chose to implement the system, using the MVC design pattern, which divides the code structure into three blocks, the model, the view, and the controller. Using this design pattern, the data in the model is manipulated through the controller and presented to the user through the view. This design pattern also allows us to reuse the same data across multiple applications, other than the web-shop GUI, as we have had to create multiple tools for generating the test data[1] needed to evaluate the system. Another reason that we decided to use MVC is that it is very extensible, and it allows us to add more models to the system, without the need for major refactoring.

Model

The model in MVC is the data-source, and is where the data is stored. Our model has to represent the objects, which is necessary for an e-commerce platform. These objects include Products, Orders, Categories, Customers etc. Figure 5.1 shows a class diagram containing our model objects, from the

[1] This will be described in section 5.2

package Model.models. It shows the interface ”IElasticable” at the top, which ensures that the object has an Id, which is used as a key for storing the object in the database. The diagram is explained in detail in the breakdown below the diagram.

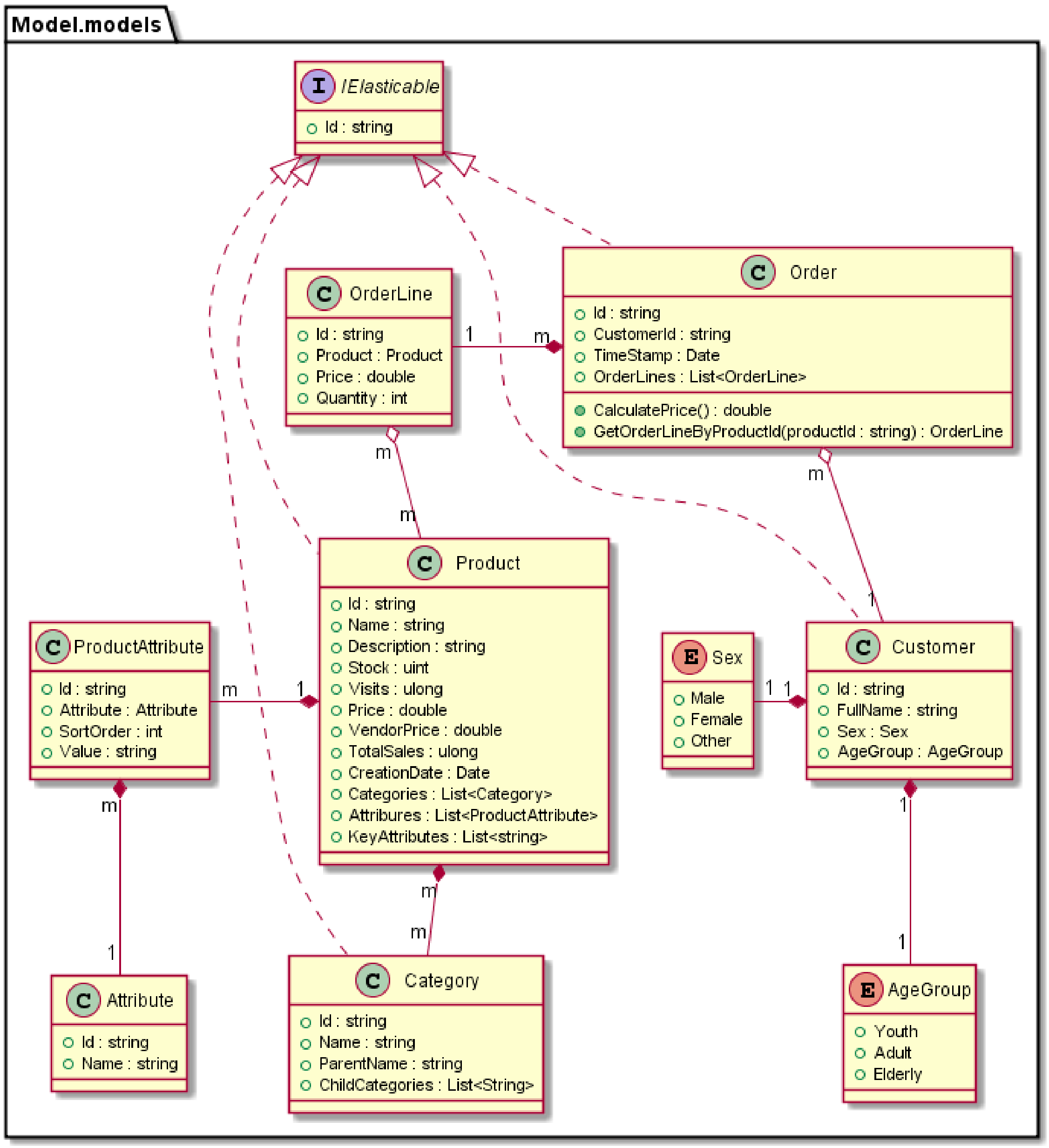


Figure 5.1: Class diagram of the Model

Breakdown of Figure 5.1

Product stores information about the product. It also keeps information on how many visits and sales the product has had. These attributes are used for generating scores for data driven ranking. The product has a list of Categories and a list of ProductAttributes.

Category stores information about the category, and has a reference to its parent and child categories. This means, that the categories have a tree-formed hierarchical structure.

ProductAttribute and Attribute: are split into two, to make it easier to retrieve aggregates of the products, based on the attributes. They store information about the products i.e. size, weight, etc.

Order stores information about the order. It contains data about when the order was created, who purchased it, and what products was on it, in the form of OrderLines.

OrderLine stores information about a single product on the order. It contains a quantity of the product, the product itself, and the current price of the product. The price is kept for bookkeeping, as prices can change over time.

Customer Stores information about the Customer. The implementation of our customers is not very rich, and should contain more information in a real life system.

We have created an alternate Product, called *ProductSearchable*, which has been optimized for text search. It is this object that will be stored in the search index, and be used for the product search, ProductSearchable can be seen in Figure 5.2.

Figure 5.2 shows the ProductSearchable, which contains scores for the product, based upon the data in the regular product. It contains a reference to the product it is representing, and a list of facets, which are the attributes, stored in a way, that makes it easier to draw aggregates from. The field FullText contains everything that the product should match against when searching. This includes categories, description text, and key attributes as a string. Next, the FullTextBoosted field contains the name of the product, and is used for Elasticsearch to boost the score of matching text on this field. This means, that a search on the product name will yield a higher score than a search mathcing the product description.

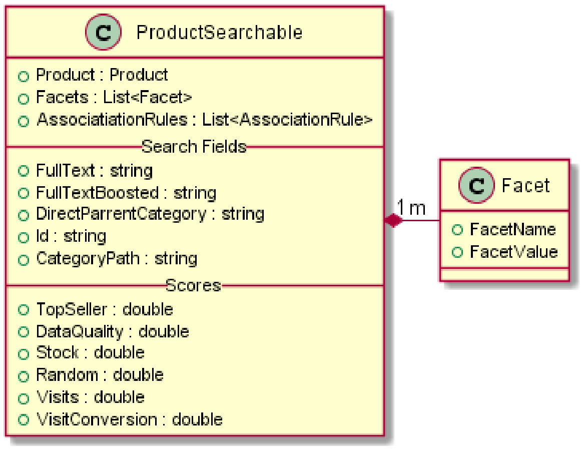
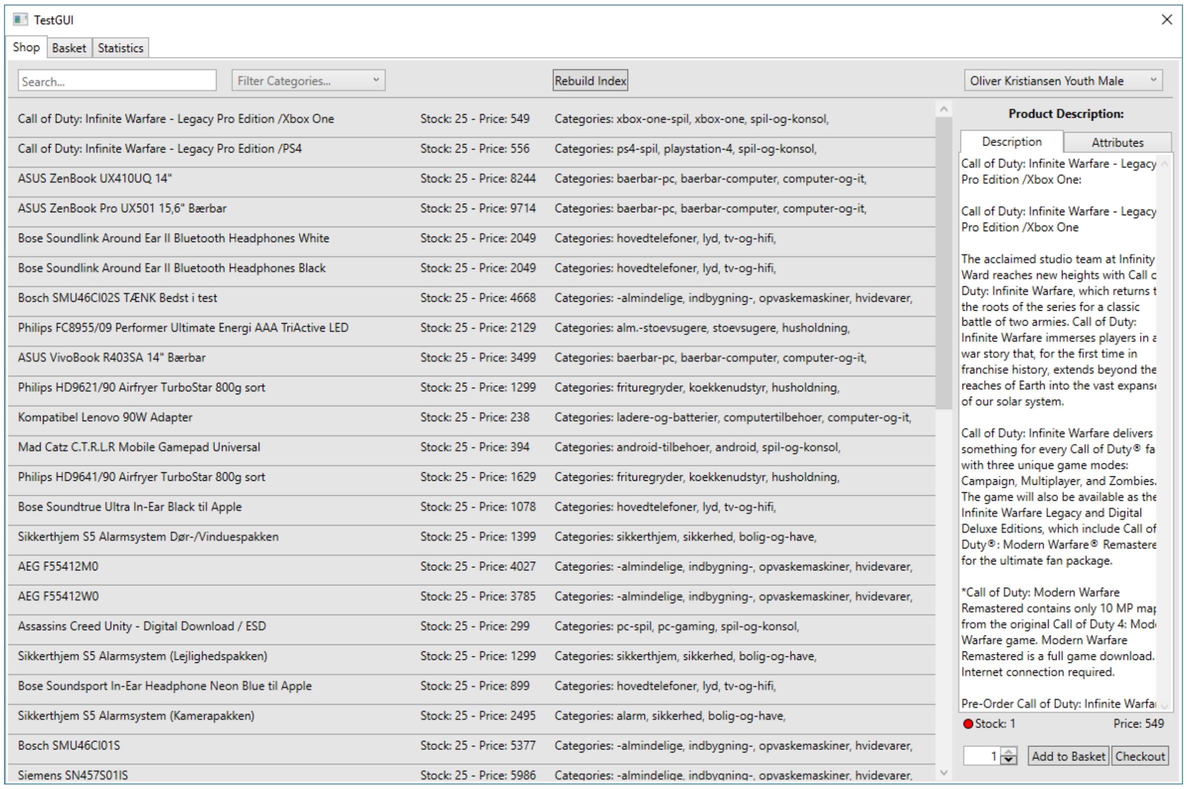


Figure 5.2: Class Diagram of the Models.Search package

View

The view in our MVC pattern is the Graphical User Interface (GUI) which has been made, based on the use cases, specified in section 3.2. The sole purpose of the View in the system is to test the functionality of the search algorithm. Figure 5.3 shows the user interface as presented to the user. The products presented in this picture are based on the data scraped from Wupti.com, as discussed in subsection 4.2.1. The focus of the project has not been to create a beautiful and smart GUI, hence the appearance in Figure 5.3.



Call of Duty: Infinite Warfare - Legacy Pro Edition /Xbox One	Stock: 25 - Price: 549	Categories: xbox-one-spil, xbox-one, spil-og-konsol,
Call of Duty: Infinite Warfare - Legacy Pro Edition /PS4	Stock: 25 - Price: 556	Categories: ps4-spil, playstation-4, spil-og-konsol,
ASUS ZenBook UX410UQ 14"	Stock: 25 - Price: 8244	Categories: baerbar-pc, baerbar-computer, computer-og-it,
ASUS ZenBook Pro UX501 15,6" Bærbar	Stock: 25 - Price: 9714	Categories: baerbar-pc, baerbar-computer, computer-og-it,
Bose Soundlink Around Ear II Bluetooth Headphones White	Stock: 25 - Price: 2049	Categories: hovedtelefoner, lyd, tv-og-hifi,
Bose Soundlink Around Ear II Bluetooth Headphones Black	Stock: 25 - Price: 2049	Categories: hovedtelefoner, lyd, tv-og-hifi,
Bosch SMU46CI02S TÆNK Bedst i test	Stock: 25 - Price: 4668	Categories: -almindelige, indbygning-, opvaskemaskiner, hvidevarer,
Philips FC8955/09 Performer Ultimate Energi AAA TriActive LED	Stock: 25 - Price: 2129	Categories: alm.-stoevsugere, stoevsugere, husholdning,
ASUS VivoBook R403SA 14" Bærbar	Stock: 25 - Price: 3499	Categories: baerbar-pc, baerbar-computer, computer-og-it,
Philips HD9621/90 Airfryer TurboStar 800g sort	Stock: 25 - Price: 1299	Categories: frituregryder, koekkenudstyr, husholdning,
Kompatibel Lenovo 90W Adapter	Stock: 25 - Price: 238	Categories: ladere-og-batterier, computertilbehoer, computer-og-it,
Mad Catz C.T.R.L.R Mobile Gamepad Universal	Stock: 25 - Price: 394	Categories: android-tilbehoer, android, spil-og-konsol,
Philips HD9641/90 Airfryer TurboStar 800g sort	Stock: 25 - Price: 1629	Categories: frituregryder, koekkenudstyr, husholdning,
Bose Soundtrue Ultra In-Ear Black til Apple	Stock: 25 - Price: 1078	Categories: hovedtelefoner, lyd, tv-og-hifi,
Sikkerthjem S5 Alarmsystem Dør-/Vinduespakken	Stock: 25 - Price: 1399	Categories: sikkerthjem, sikkerhed, bolig-og-have,
AEG F55412M0	Stock: 25 - Price: 4027	Categories: -almindelige, indbygning-, opvaskemaskiner, hvidevarer,
AEG F55412W0	Stock: 25 - Price: 3785	Categories: -almindelige, indbygning-, opvaskemaskiner, hvidevarer,
Assassins Creed Unity - Digital Download / ESD	Stock: 25 - Price: 299	Categories: pc-spil, pc-gaming, spil-og-konsol,
Sikkerthjem S5 Alarmsystem (Lejlighedspakken)	Stock: 25 - Price: 1299	Categories: sikkerthjem, sikkerhed, bolig-og-have,
Bose Soundsport In-Ear Headphone Neon Blue til Apple	Stock: 25 - Price: 899	Categories: hovedtelefoner, lyd, tv-og-hifi,
Sikkerthjem S5 Alarmsystem (Kamerapakken)	Stock: 25 - Price: 2495	Categories: alarm, sikkerhed, bolig-og-have,
Bosch SMU46CI01S	Stock: 25 - Price: 5377	Categories: -almindelige, indbygning-, opvaskemaskiner, hvidevarer,
Siemens SN457S01IS	Stock: 25 - Price: 5986	Categories: -almindelige, indbygning-, opvaskemaskiner, hvidevarer,

Call of Duty: Infinite Warfare - Legacy Pro Edition /Xbox One:

Call of Duty: Infinite Warfare - Legacy Pro Edition /Xbox One

The acclaimed studio team at Infinity Ward reaches new heights with Call c Duty: Infinite Warfare, which returns t the roots of the series for a classic battle of two armies. Call of Duty: Infinite Warfare immerses players in a war story that, for the first time in franchise history, extends beyond the reaches of Earth into the vast expans of our solar system.

Call of Duty: Infinite Warfare delivers something for every Call of Duty® fa with three unique game modes: Campaign, Multiplayer, and Zombies. The game will also be available as the Infinite Warfare Legacy and Digital Deluxe Editions, which include Call of Duty®: Modern Warfare® Remastere for the ultimate fan package.

*Call of Duty: Modern Warfare Remastered contains only 10 MP ma from the original Call of Duty 4: Mod Warfare game. Modern Warfare Remastered is a full game download. Internet connection required.

Pre-Order Call of Duty: Infinite Warfa

● Stock: 1 Price: 549

1

Figure 5.3: The view as presented to the user

Controller

The controller in the MVC pattern is responsible for manipulating the model, based upon inputs from the view. In our system we have multiple controllers, each with a different responsibility:

NestController is responsible for storing and retrieving objects from the *data index*, It is also responsible for rebuilding the *search index*, based on the data in the *data index*.[2]

SearchController is responsible for querying the products in the *search index*.

AssociationRulesController is responsible for generating the association rules, based on orders in the database.

The main responsibility of each of the controllers will be described later in this chapter in the sections 5.3 and 5.4.

5.1.2 Design of Association Rules Mining

We have chosen to make association rules based on categories, and not on individual products. This will increase the support of the rules, as multiple products will contribute to the support of a single category. This can potentially lead to some strange rules, if the categories are not specialized enough. If the categories are too broad, the rules generated might end up being too general and not useful for recommending products. Therefore, it is important that the categories are defined to be specific.

The Association Rules Mining introduces a new controller for mining the association rules from order data. A object for storing these association rules and a controller object are shown in Figure 5.4.

[2] The name comes from the .net package *Nest*, which is used to access Elasticsearch on the .net platform.

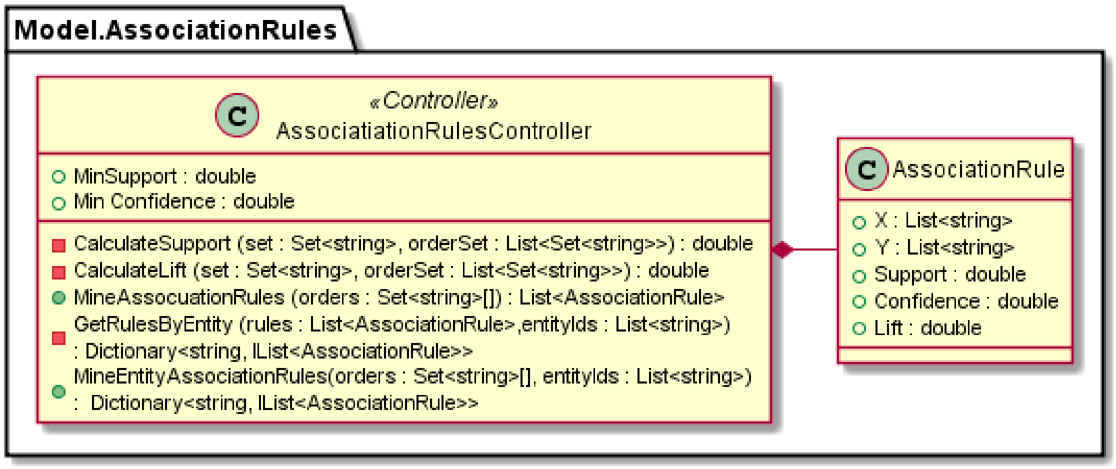

Model.AssociationRules
«Controller»
AssociatiationRulesController
MinSupport : double
Min Confidence : double
CalculateSupport (set : Set<string>, orderSet : List<Set<string>>) : double
CalculateLift (set : Set<string>, orderSet : List<Set<string>>) : double
MineAssocuationRules (orders : Set<string>[]) : List<AssociationRule>
GetRulesByEntity (rules : List<AssociationRule>,entityIds : List<string>) : Dictionary<string, IList<AssociationRule>>
MineEntityAssociationRules(orders : Set<string>[], entityIds : List<string>) : Dictionary<string, IList<AssociationRule>>
AssociationRule
X : List<string>
Y : List<string>
Support : double
Confidence : double
Lift : double

Figure 5.4: Class diagram of the Package Model.AssociationRules

The AssociationRule object contains the left-hand-side X and the right-hand-side Y of the rules[3] as a list of strings, containing the ID of associated categories. It also contains the Support, Confidence, and the Lift for the rule.

5.1.3 Persistence

The objects from our model are stored in an Elasticsearch index, which we call the *data index*. Elasticsearch is not intended as a main storage [Bra], however, as it is built as a NoSQL database, we have decided to use it instead of a conventional database. Instead of using Elasticsearch as the main storage, a structured database with more constraints could also be used for storing the objects.

The main purpose of Elasticsearch is, of course, searching and we have created a search optimized index, which we call the *search index*. The search index contains all the ProductSearchables, which are the search optimized products. The sole purpose for this index is to be used for product search, and it should be rebuilt regularly, to accommodate changes in the product data, such as sales, visits, views, etc.

That makes up the architecture of our system. In the following section we will describe how we have been generating test data for the system.

5.2 Data Generation

As Hesehus was unable to provide any test data for us, we have had to generate this data ourselves.

We have generated the data using a simple approach, where we created a very small amount of dummy-data, consisting of 11 different products.

[3] introduced in subsection 2.1.2

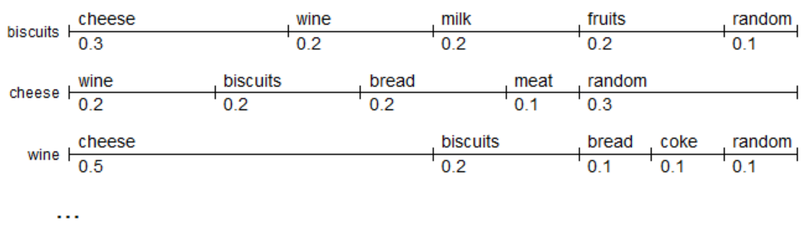


Figure 5.5: Purchase pattern format
Image of all probabilities can be found in Appendix C.

We created a bot which is able to generate orders based on preset purchase patterns for each of the products. These patterns are specified as a list of probabilities, for the bot to be jump from one product to another product. Figure 5.5 shows the format of these probabilities. Each line represents the purchase probabilities, given the product to the left is in the basket. It also contains the chance of buying a random product. This possibility is included to have some randomness in the orders. All the probabilities in the current order are combined, so the sum of all probabilities are always equals to one. This means that if more products leads to the same product, the probability of the bot choosing that product will rise.

5.3 Rebuilding the Search Index with Association Rules Mining

As mentioned, the search index should be rebuild, or updated, regularly, to reflect the latest data. This update includes change in informational data such as the product name and description, attributes etc., but also statistical data, such as sales, visits, and visit conversion. It also includes mining of association rules, based on order data from the database. It is therefore a great idea to update the search index regularly, so the scores and rules are always up to date with the latest data.

Figure 5.6 shows the update process with a sequence diagram. It starts with the Admin telling the system to rebuild the search index, which calls to the NestController[4]. The NestController then deletes the old search index, to make sure that the old data is gone. It then retrieves all the products and orders from the data index. The NestController then calls to the AssociationRulesController, to mine association rules, based on the orders in the system. The NestController then loops through each product, and creates a ProductSearchable for it, and calculates the normalized scores. The normalization is performed using the following approach $\forall\ x \in X\ :\ \frac{x}{Max(X)}$. This will result in a normalization between 0 and 1. The NestController then adds association rules, if any, to the ProductSearchable, and after having looped through all products, they are saved to the search index.

The search index is now rebuilt and running on updated data.

5.4 Scoring with Association Rules

This section will explain how we utilize these association rules in our scoring algorithm.

Figure 5.7 contains a sequence diagram, in which a customer uses the search functionality.

It starts with the customer using the search, with a string query. The system then passes this call too the SearchController, while also passing the customers basket. The SearchController starts by checking if the search string is empty and the basket is NOT. If either of these conditions are false it will do a normal search and not recommend any products. If the conditions are both true the SearchController

[4] introduced in subsubsection 5.1.1

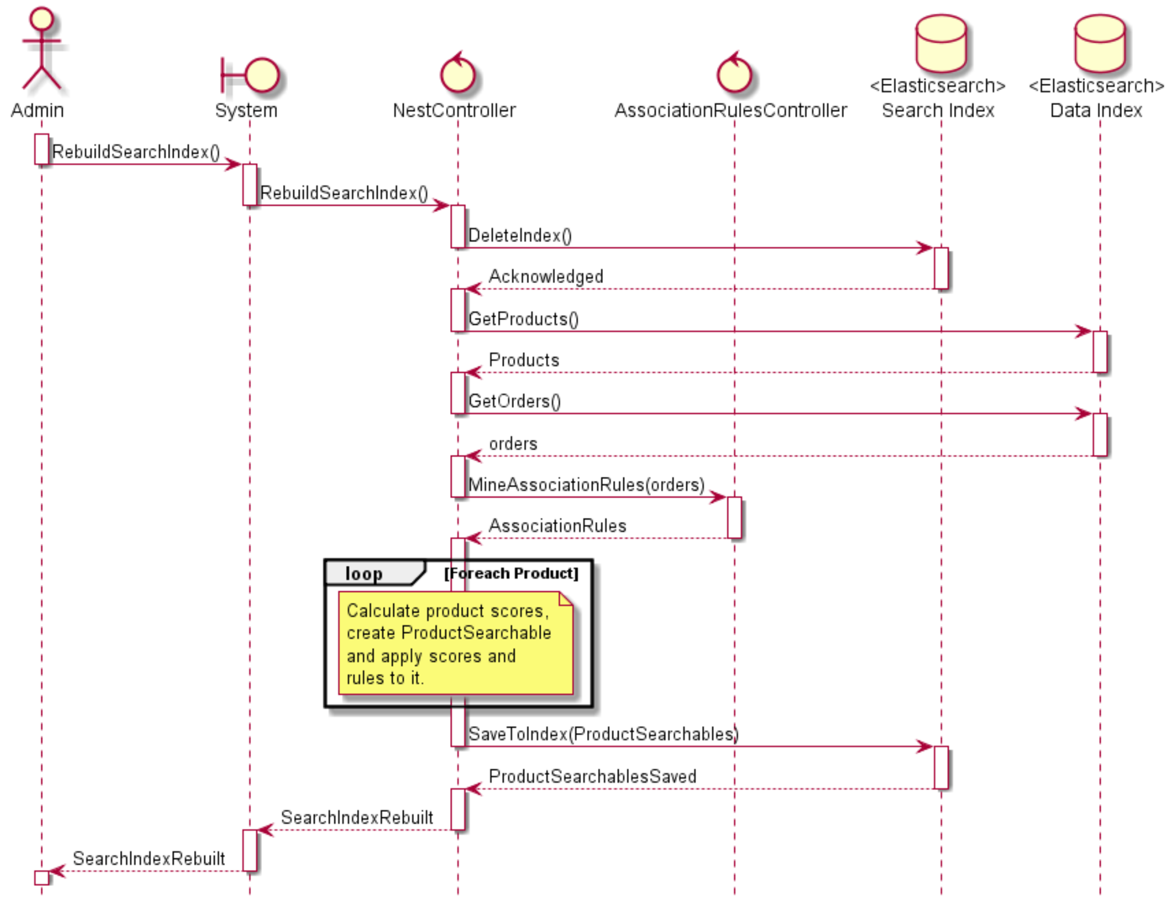


Figure 5.6: Sequence Diagram for rebuilding the Search Index

will check the contents of the basket, to see if any products in the basket match the right-hand-side of any rules. To check the rules, the SearchController first retrieves the ProductSearchable for each of the products in the basket from the search index, and adds all product rules to a list of rules. It then checks if any of the rules apply, by checking for each of the rules present in the basket, if the X of the rule is a subset of the basket as in $X \subset Basket$. If this is the case, it means that the rule apply, and the rule is added to a list of rules that apply. If no rules are found, the RuleQuery method will return null, and the query will do a normal search. If there are rules which apply the SearchController will find products which the rules leads to, and the rule query will return these.

We have described the system architecture and generated a small set of test data. Also, we have determined how the scoring algorithm should consider association rules and how these are to be stored and updated in the Elasticsearch search index.

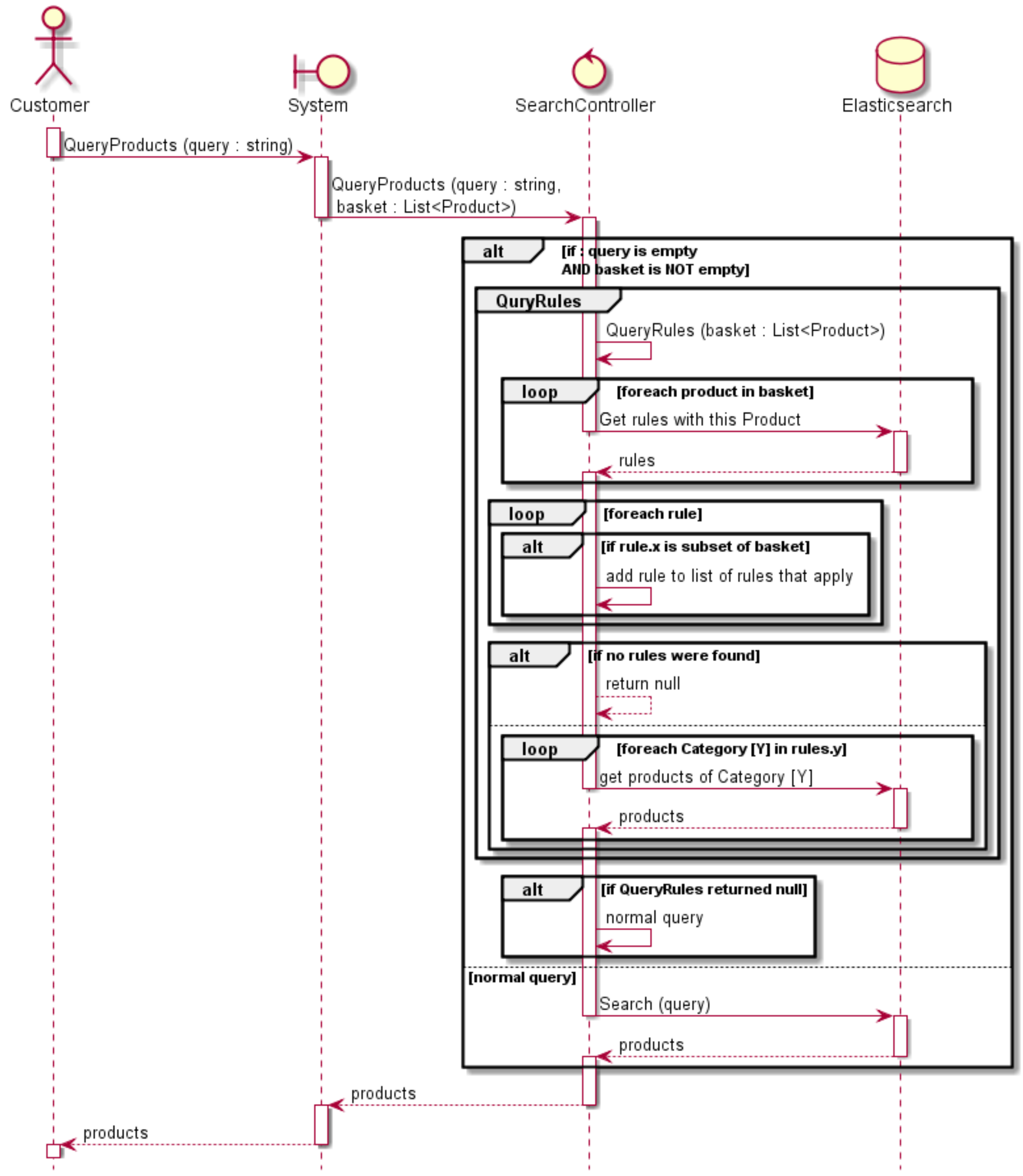

Customer
System
SearchController
Elasticsearch
QueryProducts (query : string)
QueryProducts (query : string, basket : List<Product>)
alt
[if : query is empty AND basket is NOT empty]
QuryRules
QueryRules (basket : List<Product>)
loop
[foreach product in basket]
Get rules with this Product
rules
loop
[foreach rule]
alt
[if rule.x is subset of basket]
add rule to list of rules that apply
alt
[if no rules were found]
return null
loop
[foreach Category [Y] in rules.y]
get products of Category [Y]
products
alt
[if QueryRules returned null]
normal query
[normal query]
Search (query)
products
products
products

Figure 5.7: Sequence Diagram for the Scoring with Association Rules

6 : Implementation and Algorithms

This chapter describes and discuss significant parts of the implementation of the software product. Firstly, the building of the search index is presented with a focus on the scores used for data driven ranking and how these are calculated. The mining of association rules is also presented. Secondly, it is presented how these scores and the mined association rules are used when searching.

Contents

6.1 Building the Search Index

This section describes how the custom scores of a product is calculated and normalized, and how association rules are mined and stored with the products.

6.1.1 Calculation and Normalization of Scores

This subsection presents the calculation of the scores presented in Table 2.1 in subsection 2.2.1. These scores are re-calculated each time the search index is rebuild, so that it is always up-to-date with the latest data. To recap, these scores are to influence the ranking of search result and in that way affect what products customers are presented.

```
foreach (var p in searchableProducts){
   p.DataQuality = Normalize(p.Product.DataLength, maxDataLength);
   p.Visits = Normalize((int) p.Product.Visits, (int) maxClicks);
   p.TopSeller = productsLast14Days.ContainsKey(p.Id) ? Normalize((int)
    ↪  productsLast14Days[p.Id], (int) maxSalesLast14Days) : 0.01;
   p.VisitConversion = p.Product.Visits <= 0 ? 0.01 : Normalize
    ↪  Normalize(p.Product.TotalSales /p.Product.Visits, maxVisitConversion);
}
```

Code snippet 6.1: A code snippet of the score updating - code has been omitted for readability

Code snippet 6.1 shows the essential code for updating the scores. An important thing to notice here, is that all the scores are normalized to a number between 0 and 1. The normalization is done by dividing the current value with the maximum value as shown in section 5.3.

As an example, consider the TopSeller score also seen in Code snippet 6.1. It is based on orders from the last 14 days. As all orders in the database has a creation date we can query the database for all orders from the last 14 days and count all occurring products. Now we can find both the maximum number of times a product occur and the number of times the current product occurs. By dividing the current product count with the maximum count a normalized score between 0 and 1 is achieved. Using this method of normalization, the best selling product will receive a TopSeller score of 1 while the worst selling product will have a TopSeller score of 0 or close to 0. The normalization is done so that the scores are comparable to each other and all in the same range so we avoid some products getting crazy high scores.

The calculation of the other scores presented in in Code snippet 6.1 follows the same approach as the topSeller score. The maximum value is found and used to normalize the current value.

When calculated, all scores are updated in the search index.

6.1.2 Mining Association Rules

After generating a lot of orders as described in section 5.2 association rules are mined based on these orders. For this mining of association rules we have used a third part library provided in the Accord.NET framework. The Accord.NET framework provides tools for machine learning, statistical analysis, etc. and consist of multiple different libraries from which we have used the library called Accord.MachineLearning.Rules[1]. This library provides the Apriori algorithm to mine association rules, which has made it an obvious choice for us. We chose to use a third party library for the Apriori implementation, as we focus more on using the rules it generates.

Code snippet 6.2 shows a snippet of how we use this library to mine association rules based on a set of orders. First, a new instance of the Apriori object is instantiated with given minimum support and - confidence values specified. Next, the library offers a simple way to mine the rules by calling apriori.learn() and passing the set of orders as a parameter. The result is a collection of rules that we create with support, confidence, lift, the right-hand-side denoted X and the left-hand-side denoted Y. Note that support is calculated as the rule support divided by the number of orders. This is due to the fact, that we want to use support as a percentage of the orders, as it is a more relative measure, while the Accord.MachineLearning.Rules returns support as the number of times an association is present. We also calculate the lift of the rule as defined in subsubsection 2.1.2. Lastly, the left- and right-hand-side is added to the AssociationRule object and a list

[1] HTTP://accord-framework.net/docs/html/R_Project_Accord_NET.htm

```
public IList<AssociationRule> MineAssociationRules(orders){
   var apriori = new Apriori<string>(minSuport, minConfidence);
      var ARrules = apriori.learn(orders);
      var resultRules = new List<AssociationRule>();
      foreach (var rule in ARrules.Rules)
         resultRules.add(new AssociationRule
         {
            Support = rule.Support / orders.Count(),
            Confidence = rule.Confidence,
            Lift = rule.Confidence / rule.Y
            X = rule.X,
            Y = rule.Y
         });
      return resultRules;
   }
```

Code snippet 6.2: A code snipped of the method MineAssociationRules(orders)

After having mined all association rules from the orders in Code snippet 6.2, to actually use them in a search we want to find the association rules belonging to any given product. A snippet of this code is presented in Code snippet 6.3. The method takes a list of all rules and one or more ids that we want to find the related association rules for. The method returns a dictionary with the product id as key and a list of related association rules as value. The method loops through all rules for all ids. If the rule left-hand-side does not contain the product id, the loops skips onto the next id in the list. If however, the left-hand-side of the rule does contain the product id, the entityRules dictionary is updated with the product id and the related rule is added to the list. This dictionary is returned when the loop has run through all the ids given.

```
public Dictionary<string, IList<AssociationRule>> GetRulesByEntity(rules, ids){
   var entityRules = new Dictionary<string, IList<AssociationRule>>();
   foreach (var id in ids)
      foreach (var rule in rules)
      {
         if (!rule.X.contains(id))
            continue; //continue inner loop
         if (entityRules.ContainsKey(if))
            entityRules[i].Add(rule);
         else
            entiryRules.Add(id, new List<AssociationRule> {rule});
      }
   return entityRules;
}
```

Code snippet 6.3: A code snippet of the method GetRulesByEntity(rules, productIds)

From the code presented in Code snippet 6.3 we now know which association rules any given product category is present in. To actually utilize this knowledge, the document for every object is updated and the related product categories are added to the index. An example could be the rule $\{cremefraiche, snacks\} \rightarrow \{dipmix\}$ which is stored in the document for all products belonging to either the snacks or the creme fraiche category like shown in Code snippet 6.4.

```
{
   "score": 0.7828665030445657,
   "x": [
      "creme fraiche",
      "snacks"
   ],
   "y": [
      "dip mix"
   ],
   "support": 0.137,
   "confidence": 0.8404907975460123,
   "lift": 3.7272319181641342,
}
```

Code snippet 6.4: A highlight from the product search index showing how the association rules are stored

Code snippet 6.4 shows both the left-hand-side of the rule which is creme fraiche and snacks and the right-hand-side which is dip mix. It also stores the support, the confidence and the lift of the rule. Another thing stored is the rule score, which we have defined as a weighted score based on support, confidence, and lift.

6.2 The Scoring Algorithm

This section describes how the scores calculated in subsection 6.1.1 are used to rank search hits and how we use the association rules mined in subsection 6.1.2 to recommend related products.

6.2.1 Custom Scoring

We use a custom scoring script for ranking search results in extension to what Elasticsearch is already doing out of the box. A snippet of the method QueryProducts(query, basketList) responsible for handling querying logic is shown in Code snippet 6.5.

```
QueryProducts(string query, IList<Product> basketList)
   (...)
   Fields(f => f
      .Field(fff => fff.FullText)
      .Field(fff => fff.FullTextBoosted)
      .Field(fff => fff.AllParentCats)
      .Field(fff => fff.Id)
   )
   .Analyzer("danish")
   .Type(TextQueryType.CrossFields)
   .Query(query)
```

```
   .Script(script => script
      .Inline("(1 + Math.pow(_score, params.docweight)) * " +
              "doc['stock'].value * " +
              "(" +
              "  doc['topSeller'].value * params.tsweight" +
              "+ doc['dataQuality'].value * params.dqweight" +
              "+ doc['visits'].value * params.vweight" +
              "+ doc['visitConversion'].value * params.vcweight" +
              "+ doc['random'].value * params.rweight" +
              ")"
      )
   )
```

Code snippet 6.5: Snippet of the method QueryProducts. Some code has been omitted for readability

The method starts out by checking if the basket of the user already contains any items, and if so, if these are part of any association rules that can be used for recommending products. Products are only recommended, if the passed query string is empty. If the query string is not empty, the method continues without recommending any products. The recommendation of products are described further in subsection 6.2.2.

In the first part of the snippet, it is defined what fields in the search index Elasticsearch should search. As it can be seen in Code snippet 6.5 we are looking at the fields full text, full text boosted, parent categories and product id. The typical job of an analyzer is to break down the text into more manageable terms, and this can be done in several different ways. Elasticsearch provides a lot of different analyzer out of the box[2], and we have specified it to be the danish analyzer, as the product data is all in danish.

In the next part of Code snippet 6.5 the custom scoring algorithm is applied. It takes into account the _score variable which is created by Elasticsearch based on how good of a match the hit is from searching the specified fields. However, it also takes into account all the custom scores discussed in subsection 6.1.1. In the omitted code the weight are defined in the params object. As of the current implementation, all these weights have a value of 0.2. These weight can however, be changed and manipulated to try and come up with an even better scoring of the search results. A thing to notice here, is the fact that the stock score is multiplied while the other scores are added together. This is done, as stock will have a value close to zero if the product is not in stock, and a value of 1 if the product is in stock. In this way, products not in stock will always score very low. On the other hand, products in stock will have a score of 1 which does not affect the overall scoring at all.

6.2.2 Association Rules Recommendations

Recommended products are shown, when the user has already added products to her basket that are part of one or more rules and enters an empty search string. Code snippet 6.6 shows the check, that determines if the basket should be checked for any containing rules.

```
QueryProducts(string query, IList<Product> basketList)
    if (basketList != null && basketList.Count > 0 && query == string.Empty)
   {
      var rulesQuery = RulesQuery(basketList);
      // if any rules are found, return fitting products
      if (rulesQuery != null)
         return rulesQuery;
   }
```

Code snippet 6.6: The check in the start of QueryProducts method. If any rules are found, recommended products are returned.

If products are present in the basket and the search string is empty, method RulesQuery(basketList) is called. A snippet of this method is shown in Code snippet 6.7. Again, some code has been omitted for readability. First, all product search indexes are found based on the ids of the products in the basket. Next all products the basket are iterated and it is checked, if their rules left-hand-side denoted x is a subset of the basket. For example, consider the association rule from Code snippet 6.4 $\{cremefraiche, snacks\} \rightarrow \{dipmix\}$ and creme fraiche and some sort of snacks are already in the basket. When going over creme fraiche, it is found out, that the rule left-hand-side $\{cremefraiche, snacks\}$ is in fact a subset of the basket, and the rule is added to the rules dictionary.

[2] HTTPs://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html

```
RulesQuery(IList<Product> basketProducts)
   var productSearchables = new List<ProductSearchable>();
   foreach (var product in basketProducts)
      productSearchables.Add(_client.Get(new DocumentPath<ProductSearchable>(product.Id);
      (...)
   var rules = new Dictionary<string, AssociationRule>();
   foreach (var product in productSearchables)
      foreach (var rule in productSearchable.AssociatedCategories)
      if (rule.X.IsSubsetOf(basketProducts.Select(p => p.Categories.First().Id).ToList()))
      if (!rules.ContainsKey(rule.Y.Aggregate(string.Empty, (a, b) => a + ":" + b)))
      rules.Add(rule.Y.Aggregate(string.Empty, (a, b) => a + ":" + b), rule);
```

Code snippet 6.7: A code snipped of the method RulesQuery(basketList)

When all rules from the basket are found, the rules are sorted with respect to their rule score, which is a weighted score based on the rule support, confidence, and lift. This is done so that products recommended based on the best rules are presented first. Based on the found rules, the product database is queried again, once for each rule, for products fulfilling the right-hand-side of the rule, as shown in Code snippet 6.8.

This search is close to identical to the search in Code snippet 6.5, however there are two major differences. First of all, the QueryProducts method from Code snippet 6.5 queries multiple fields like full text, id, and full text boosted, whereas the QueryProducts method only queries products whose parent category match the right-hand-side of the rules considered. This means, that the recommended product is the best scoring product that match category of the rule's right-hand-side. Secondly, the return size of the query, that is, the number of hits, are set to a default in the QueryProducts method. In the QueryProducts method, this return size for each rule is set to the default divided by the number of rules divided by the number of categories on the right-hand-side of the rule.

```
RulesQuery(IList<Product> basketProducts)
(...)
 foreach (var rule in rules.Values.OrderByDescending(r => r.Score))
     foreach (var catId in rule.Y)
     (...)
     .Query(qf => qf
     .Match(m => m
     .Field(f => f.DirectParentCatId)
     .Query(catId)
     (...)
     .Size(defaultReturnSize / rules.Count / rule.Y.Count)
```

Code snippet 6.8: A code snipped of the method RulesQuery(basketList)

We have presented how the product scores used for search ranking are calculated and used on search time. Also, we have discussed how association rules based on the orders are mined and how these rules are used to recommend products.

7 : Evaluation

This chapter will evaluate the important parts of the system, which we have designed and implemented in the prior chapters. We will look at the Association Rules, that are generated, both in a quantitative and a qualitative perspective, and we will evaluate how we score with the association rules.

Contents

7.1 Evaluation of the Association Rules Mining

This section evaluates the Association Rules Mining (ARM) based on the rules it produces. An Association Rule has three important variables, support, confidence, and lift, all of which have been described in subsubsection 2.1.2. The Apriori algorithm we use, has two parameters, the *minimum support* and *minimum confidence*. The minimum support dictates how frequent the items has to be, to be considered for rules, and the minimum confidence dictates how strong the rules must be to be selected.

7.1.1 Quantitative Evaluation of the Association Rules

In this section we will evaluate the Association Rules based on the data generated by the bot, described in section 5.2. We set the bot up to generate 2000 orders based on the specified purchase patterns, and run ARM on these orders, with various presets of minimum support and - confidence. This process is repeated multiple times, to get a more precise average of the rules.

Number of Rules Generated

A way to evaluate the ARM is to look at how many rules it generates. So we set up the bot to generate orders, and rules with a varying minimum support in the range 0.001 - 0.2[1] and a consistent minimum confidence at 0.4. We did this to evaluate the influence of the minimum support parameter, and to see how many rules we could get rid of, by adjusting the minimum support.

[1] these numbers are proportions and mean that the item must present in 20% of all orders, to be considered in a rule.

Figure 7.1 shows a stair diagram for the number of rules, for each of the Apriori parameter presets. It starts off by generating more than 1000 rules, with a minimum support of 0.001, which is a very high number of rules created, given that there are only 11 products to consider. As the minimum support rises, Figure 7.1 shows that the number of rules decreases, and flattens out around a minimum support of 0.15.

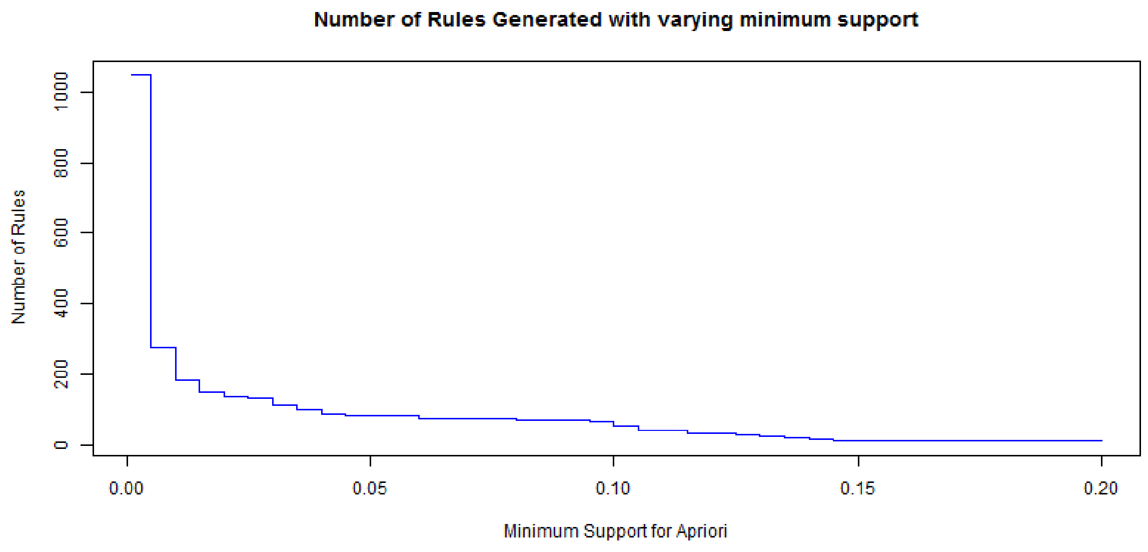


Figure 7.1: Number of rules generated by the ARM with varying Minimum Support Values

But do the rules make any sense? We do not have any way of quantitatively tell if the rule makes any sense, as this is a subjective measure, and can therefore not be quantitatively measured. The rules will be evaluated qualitatively in subsection 7.1.2.

We do however have the possibility to evaluate the quality of the rules, statistically, based on the order data. The confidence variable on the rules tells us, theoretically, how accurate the rule is, as it covers the probability that product Y is bought given product X is also bought.

Figure 7.2 shows a graph of the mean confidence for all the rules at each of the minimum supports. We see that as the minimum support increases, so does the mean confidence, which means that our rules become statistically more accurate, based on the order data.

This means that by removing a lot of the rules, we have increased the average confidence for our rules, and thereby increased the quality of the rules.

7.1.2 Qualitative Evaluation of the Association Rules

The qualitative evaluation approach is necessary, as rules have subjective measures that tell if a rule makes sense or not. This is very subjective, and can change based on location and culture. Some countries can have purchase patterns, which does not make sense in another country. And therefore we have to evaluate the rules subjectively.

In the evaluation we will discuss whether rules makes sense. We will start with an example of a bad rule and discuss, why this rule is not any good, and then go through an example for a good rule.

Code snippet 7.1 shows an example of a bad rule, that has been generated without a minimum support (0) and a minimum confidence of 0.4. $\{Biscuits, Cheese, Milk, Snacks\} \rightarrow \{Bread\}$. This rule would have been discarded by the mining algorithm if a reasonable minimum support had been

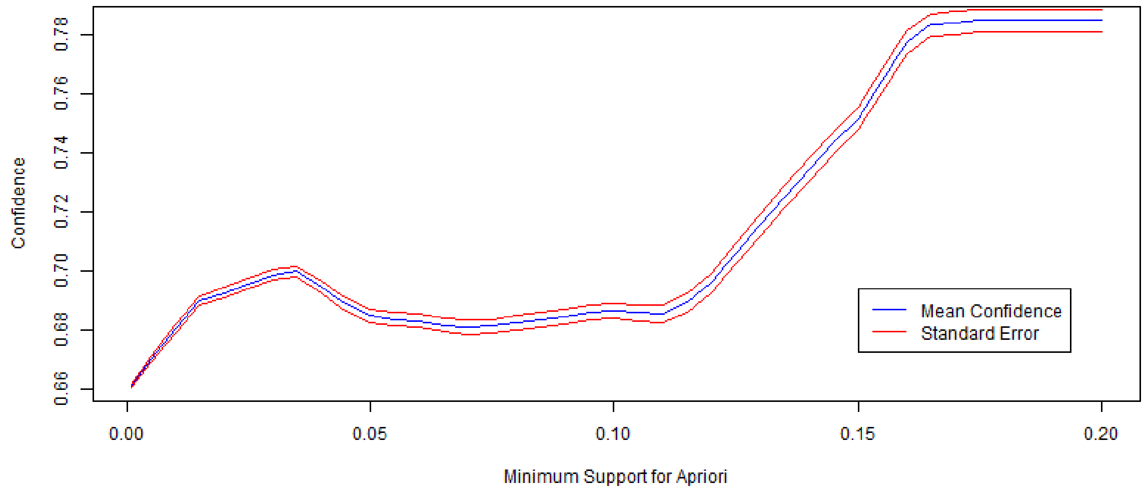


Figure 7.2: Graph of the Mean Confidence for the Association Rules for varying Minimum Support on the X-axis

```
{
  "x": [
    "biscuits",
    "cheese",
    "milk",
    "snacks"
  ],
  "y": [
    "bread"
  ],
  "support": 0.0005,
  "confidence": 0.4,
  "lift": 0.8756567425569177
}
```

Code snippet 7.1: An example of a bad Rule

applied. While these products might be purchased together regularly, we can not really determine that this rule is realistic, as the products does not really fit together, and the combination of them seem a bit random.

Another thing to add is that only two of the X items ($Cheese$ and $Milk$) leads to the $Bread$, as defined in the purchase patterns in Appendix C, and the addition of biscuits and snacks seem unnecessary. The statistics of the rule are very bad as well, as the products are not very frequent (very low support) and the confidence only just passes the minimum confidence of 0.4. The lift suggest that people are actually less likely to purchase bread after adding the products of X to the basket, as it is below 1.

Code snippet 7.2 shows an example of a good rule $\{Creme\ Fraiche,\ Snacks\} \rightarrow \{Dip\ Mix\}$. It makes sense that the customer is going to need dip mix, when he is about to purchase snacks and creme fraiche, so that he can have dip for his snacks. This rule would also be realistic in the real world, and it is also present in the purchase patterns[2], which the bot uses for creating the orders. The statistics of the rule, also suggests that this is a good rule, as we see a high support, confidence, and lift.

[2] Purchase patterns can be seen in Appendix C

```
{
  "x": [
    "creme fraiche",
    "snacks"
  ],
  "y": [
    "dip mix"
  ],
  "support": 0.137,
  "confidence": 0.8404907975460123,
  "lift": 3.7272319181641342
}
```

Code snippet 7.2: An example of a good Rule

7.2 Evaluation of the Association Rules in the Scoring Algorithm

This section will evaluate the functionality of the scoring algorithm, when using association rules, both qualitatively, but also in terms of scalability.

7.2.1 Qualitative Evaluation of the Scoring

This section will evaluate the experience of searching with association rules. With the limited data we have generated, it is difficult to evaluate the user experience.

At the moment if you add an item to the basket, which fulfills a rule, and search for the empty search string, it will only show products of the rule. This might confuse new users of the system.

This could be solved, by adding the suggested associated products to a separate suggestion field, instead of presenting them with the other products. However, this system has been developed only to evaluate the rules, and the scoring of these rules. Therefore the user experience is not the main focus of the GUI.

We are able to present products, based on the association rules, and if the rules makes sense, the products presented would also make sense. The user experience would most likely be better, if we had a wider range of products in our dataset, as it would be able to suggest multiple products from the same category.

Figure 7.3 shows the presented items, when we have added Chips (from the category "*Snacks*") to the basket. The products presented are based on the mined rules:

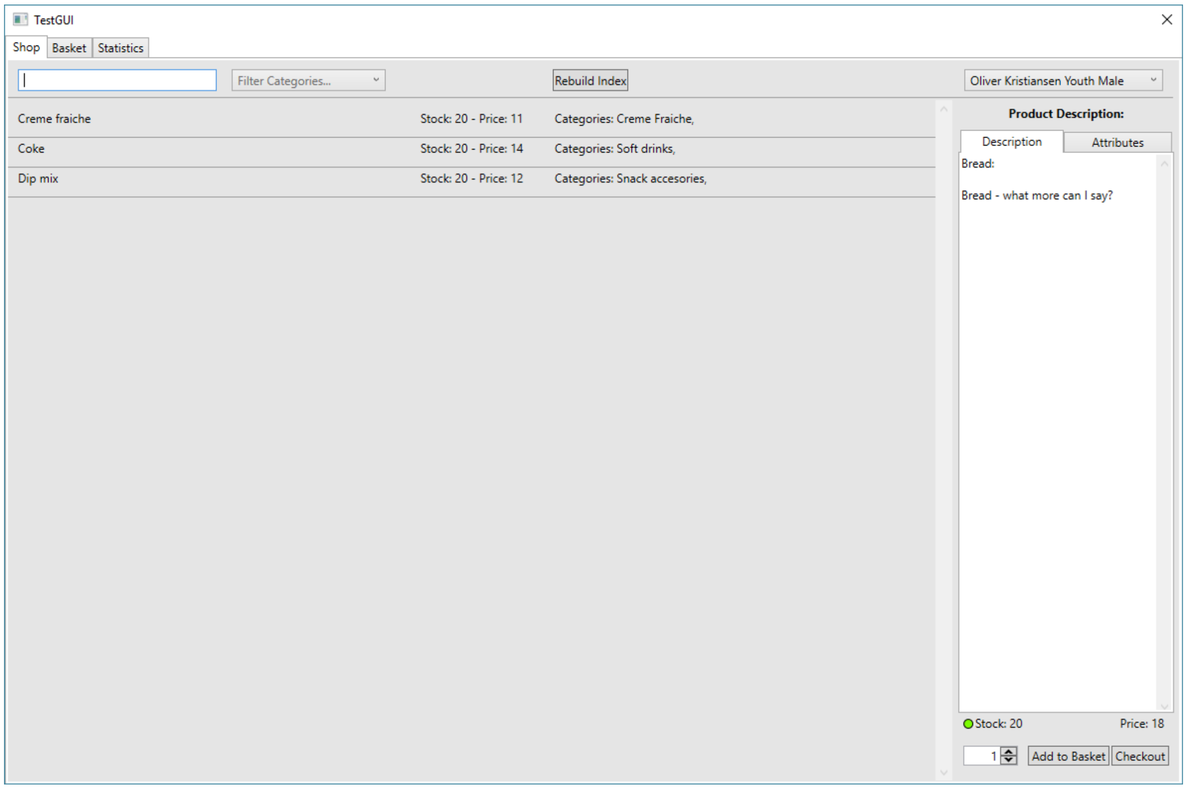
$$R1 = \{Snacks\} \rightarrow \{Creme\ Fraiche\},\ ruleScore = 0.50$$
$$R2 = \{Snacks\} \rightarrow \{Coke\},\ ruleScore = 0.53$$
$$R3 = \{Snacks\} \rightarrow \{Dip\ Mix\},\ ruleScore = 0.63$$

The order of the products are also influenced by the products own score, $R3$ has the highest ruleScore[3] of the three, and it is ranked the lowest, due to a lower product score. The product score is based on the scores on the product, such as sales, and data quality.

7.2.2 Scalability

The scoring of association rules adds a few steps to the process of querying, as it has to retrieve the products from the basket to check if any rules apply. This means that it has to do a HTTP call for each product in the basket, and another HTTP call for each rule, which applies. This will impact the running

[3] ruleScore is described in subsection 6.1.1



Creme fraiche Stock: 20 - Price: 11 Categories: Creme Fraiche,
Coke Stock: 20 - Price: 14 Categories: Soft drinks,
Dip mix Stock: 20 - Price: 12 Categories: Snack accesories,

Bread:

Bread - what more can I say?

Stock: 20 Price: 18

1

Figure 7.3: The GUI with when a Chips item has been added to the Basket

time for the scoring algorithm, as it does not scale very well in situations with large baskets and many applicable rules, because of the increasing number of HTTP calls.

We have tested this, by querying for the empty string, with a varying basket size between 0 and 11 products, and repeated this process 1000 times, to get a better average.
This test was conducted on a localhost network, and the timing should vary much from a distributed setup because of network latency.

We have tried to see what impact the extra HTTP calls have on the query time, and plotted it to the graph in Figure 7.4. We can see that as the amount of products in the basket increases, so does the query time.

This means that the larger the basket, the longer time it will take for the query to complete. The amount of rules fulfilled in the basket, also impacts the query time, as each rule also require a HTTP call to find products from the category in Y.

However this issue, could potentially be resolved with an approach, in which the *AssociationRule* object would store a number of *ProductSearchables* from the categories it leads to. It would also be better if the basket would contain *ProductSearchable* objects instead of *Product* objects, as they contain the association rules needed.
This approach would increase the time it takes to build the search index, as we would have to query products for each association rule generated.
In a real world scenario a longer index build time is to prefer, if it can make querying quicker. Index building is done only once in a while, while thousands of users is affected by slow query times each and every day.
The complexity and file size of the *ProductSearchable* objects would also increase a lot, as it would need to store information about multiple other products now.
However, it would be worth it when the number of HTTP calls becomes an issue. We could potentially

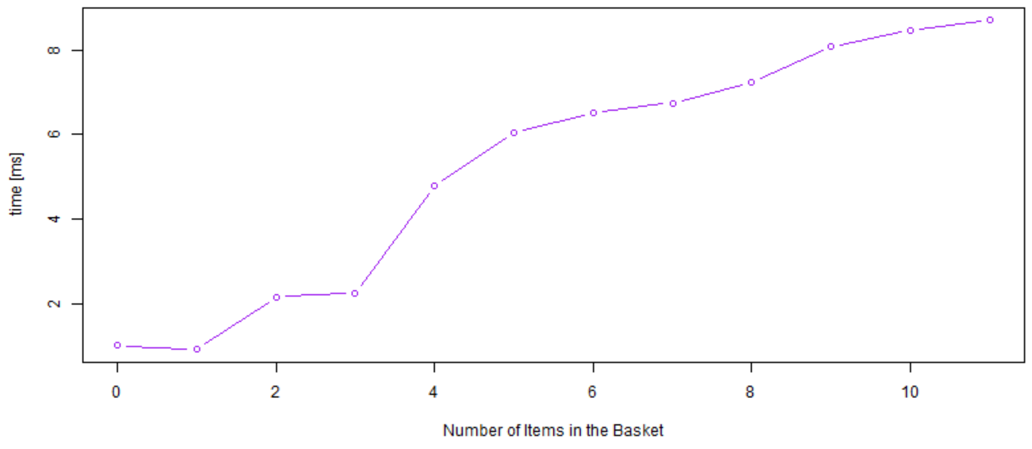


Figure 7.4: Query time for increasing amount of Products in the Basket

decrease the number of calls needed, for the association rules query to 0. This is possible because the data needed from the search index, already has been prepared when the search index was built.

As we have been running the system locally, we have not focused on reducing the amount of HTTP calls needed. But in a real world scenario, where the system would be distributed, it would be beneficial to implement it this way, as HTTP calls are expensive latency-wise and should be limited.

We have evaluated the accuracy of the Association Rules Mining, and found that by increasing the minimum support, we get a lower number of rules, with a higher average confidence. Through subjective measures we have concluded that the rules with a high confidence seem to make sense and are also presented correctly in the GUI. Lastly we have evaluated that our scoring algorithm does not scale very well, in regard to the amount of HTTP calls needed.

8 : Conclusion and Future Works

The first part of this chapter presents the project conclusion. The second part of this chapter points out directions for future work.

8.1 Conclusion

This report has documented the implementation of a more intelligent scoring algorithm to present users with more relevant products. The report suggests the use of association rules mining based on order data to determine purchase patterns and relations between product categories. The implementation utilizes the search engine Elasticsearch and the Apriori algorithm to mine association rules. Products are then suggested based on these rules.

The project has been evaluated using both quantitative and qualitative measures. Through the quantitative evaluation it is concluded, that a higher minimum support value will result in a decreased amount of rules, with a higher average confidence. It is concluded, that rules with a high support and confidence seem to make sense, based on the qualitative evaluation presented.

It is concluded that we, with the use of association rules mining, are able to present customers with relevant products based on purchase patterns.

8.2 Future Works

Genetic Algorithm for Scoring Weight Optimization

Future work should include the use of genetic algorithms to find the most optimal weighing of the scores that determines the rank of a product. The modified weights could be evaluated with regard to a lot of different measurements. Obvious measures include total revenue, total sales, or customer satisfaction, which will of course need to be evaluated both before and after modifying the weights.

Personalized of Association Rules

In the future, it would be interesting to take into account that not all users have the same needs and buy the same items. Therefore, it could make sense to define a number of user groups and mine association rules specifically within these groups. An example of a group could be women in the age group $18 - 30$. Using this approach, users could be presented with more relevant products.

Bibliography

[AK12] Loraine Charlet Annie and Ashok Kumar. Market basket analysis for a supermarket based on frequent itemset mining. *IJCSI International Journal of Computer Science*, Vol. 9, Issue 5(3):1694–0814, September 2012.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Journal of Computer Science and Technology*, 15(6):487–499, 1994.

[Bra] Alex Brasetvik. Elasticsearch as a nosql database. https://www.elastic.co/blog/found-elasticsearch-as-nosql.

[FA] Jimmy Solano Felipe Alfaro. Apriori vs fp-growth for frequent item set mining. http://singularities.com/blog/2015/08/apriori-vs-fpgrowth-for-frequent-item-set-mining.

[GT14] Clinton Gormley and Zachary Tong. *Elasticsearch*. O'Reilly Media, 2014.

[KK06] Sotiris Kotsiantis and Dimitris Kanellopoulos. Association Rules Mining : A Recent Overview Basic Concepts & Basic Association Rules Algorithms. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[Lan15] Brett Lantz. *Machine Learning with R - Second Edition*. Packt Publishing, 2015.

[LS] Martin Loetzsch and Krešimir Slugan. On-site search design patterns for e-commerce: Schema structure, data driven ranking and more. http://project-a.github.io/on-site-search-design-patterns-for-e-commerce/.

[MPA16] Nidhi Maheshwari, Nikhilendra K. Pandey, and Pankaj Agarwal. Article: Market basket analysis using association rule learning. *IJCA Proceedings on Recent Trends in Future Prospective in Engineering and Management Technology*, RTFEM 2016(2):20–24, July 2016. Full text available.

[Som10] Ian Sommerville. *Software Engineering*. Pearson, 2010.

[Wys14] Robert K. Wysocki. *Effective Project Management: Traditional, Agile, Extreme*. Wiley, 7th edition, 2014.

[ZPLO96] Mohammed J Zaki, Srinivasan Parthasarathy, Wei Li, and Mitsunori Ogihara. Evaluation of sampling for data mining of association rules. Technical report, University of Rochester, Rochester, NY, USA, 1996.

A : Example of an Search Index Structure

```json
{
  "type": "staple",
  "search_result_data": {
    "sku": "SP11968",
    "name": "Fortis Fäustel, mit Eschen-Stiel",
    "preview_image": "faeustel-din6475-2000g-eschenstiel-fortis-21049292-0-JlHR5nOi-l.jpg",
    "number_of_products": "4",
    "final_gross_price": "822",
    "final_net_price": "691",
    "base_gross_price": null,
    "base_price_unit": null,
    "url": "/handwerkzeug/fortis-faeustel-mit-eschen-stiel-SP11968"
  },
  "search_data": [
    {
      "full_text": " 21049289  4317784792714 04317784792714 Fäustel DIN 6475<br><br>Stahlgeschmiedet, Kopf schwarz lackiert, Bahnen poliert, doppelt geschweifter Eschenstiel mit ozeanblau lackiertem Handende. SP11968 SP11968",
      "full_text_boosted": " Fortis Fäustel DIN6475 1000g Eschenstiel FORTIS 1000 Fäustel Handwerkzeug Hammer Fäustel Fortis Fäustel, mit Eschen-Stiel Fortis Fäustel, mit Eschen-Stiel",
      "string_facet": [
        {
          "facet-name": "manufacturer",
          "facet-value": "Fortis"
        },
        {
          "facet-name": "hammer_weight",
          "facet-value": "1000"
        }
      ],
      "number_facet": [
        {
          "facet-name": "final_gross_price",
          "facet-value": 822
        }
      ]
    },
    {
      "full_text": " 21049290  4317784792721 04317784792721 Fäustel DIN 6475<br><br>Stahlgeschmiedet, Kopf schwarz lackiert, Bahnen poliert, doppelt geschweifter Eschenstiel mit ozeanblau lackiertem Handende. SP11968 SP11968",
      "full_text_boosted": " Fortis Fäustel DIN6475 1250g Eschenstiel FORTIS 1250 Fäustel Handwerkzeug Hammer Fäustel Fortis Fäustel, mit Eschen-Stiel Fortis Fäustel, mit Eschen-Stiel",
      "string_facet": [
        {
          "facet-name": "manufacturer",
          "facet-value": "Fortis"
        },
        {
          "facet-name": "hammer_weight",
          "facet-value": "1250"
        }
      ],
      "number_facet": [
        {
          "facet-name": "final_gross_price",
          "facet-value": 1020
        }
      ]
    },
    {
      "full_text": " 21049291  4317784792738 04317784792738 Fäustel DIN 6475<br><br>Stahlgeschmiedet, Kopf schwarz lackiert, Bahnen poliert, doppelt geschweifter Eschenstiel mit ozeanblau lackiertem Handende. SP11968 SP11968",
      "full_text_boosted": " Fortis Fäustel DIN6475 1500g Eschenstiel FORTIS 1500 Fäustel Handwerkzeug Hammer Fäustel Fortis Fäustel, mit Eschen-Stiel Fortis Fäustel, mit Eschen-Stiel",
      "string_facet": [
        {
          "facet-name": "manufacturer",
          "facet-value": "Fortis"
        },
        {
          "facet-name": "hammer_weight",
          "facet-value": "1500"
        }
      ],
      "number_facet": [
        {
          "facet-name": "final_gross_price",
          "facet-value": 1039
        }
      ]
    },{
      "full_text": " 21049292  4317784792745 04317784792745 Fäustel DIN 6475<br><br>Stahlgeschmiedet, Kopf schwarz lackiert, Bahnen poliert, doppelt geschweifter Eschenstiel mit ozeanblau lackiertem Handende. SP11968 SP11968",
      "full_text_boosted": " Fortis Fäustel DIN6475 2000g Eschenstiel FORTIS 2000 Fäustel Handwerkzeug Hammer Fäustel Fortis Fäustel, mit Eschen-Stiel Fortis Fäustel, mit Eschen-Stiel",
      "string_facet": [
        {
          "facet-name": "manufacturer",
          "facet-value": "Fortis"
        },
        {
          "facet-name": "hammer_weight",
          "facet-value": "2000"
        }
      ],
      "number_facet": [
        {
          "facet-name": "final_gross_price",
          "facet-value": 1194
        }
      ]
    }
  ],
  "completion_terms": [
    "Fortis",
    "1000",
    "1250",
    "1500",
    "2000",
    "Fäustel",
    "Handwerkzeug",
    "Hammer",
    "Fäustel"
  ],
  "suggestion_terms": [
    "Fortis Fäustel, mit Eschen-Stiel"
  ],
  "number_sort": {
    "final_gross_price": 822
  },
  "string_sort": {
    "name": "Fortis Fäustel, mit Eschen-Stiel"
  },
  "scores": {
    "top_seller": 0.91,
    "pdp_impressions": 0.38,
    "sale_impressions_rate": 0.8,
    "data_quality": 0.87,
    "delivery_speed": 0.85,
    "random": 0.75,
    "stock": 1
  },
  "category": {
    "direct_parents": [
      "bpka"
    ],
    "all_parents": [
      "bost",
      "boum",
      "boun",
      "bpka"
    ],
    "paths": [
      "boum-boun-bpka"
    ]
  },
  "category_scores": {
    "number_of_impressions": 265,
    "number_of_orders": 23
  }
}
```

B : Risk Mitigation Plan

Table B.1: The matrix used for calculating the severity of a risk

Impact	High	3	4	5
	Medium	2	3	4
	Low	1	2	3
		Low	Medium	High
		Likelihood		

ID	r1
Risk	Hesehus is unable provide any test-data
Overview	We need data for testing, this data must be realistic, and have the correct attributes, to be usable in a real-world-scenario.
Likelihood	Medium
Impact	High
Severity	4
Mitigation plan	To mitigate this risk, we will ask Hesehus for a ER-diagram2, which we will use to generate dummy data, which follows this diagram. This can potentially lead to risk r2.

ID	r2
Risk	Hesehus will not provide an ER-diagram of their data
Overview	If Hesehus do not provide any test data, we need to generate it ourselves. An ER-diagram is then needed to structure our data around to make it more realistic..
Likelihood	Low
Impact	Medium
Severity	2
Mitigation plan	We should figure out the key objects needed for the system and attributes for these, and consult Hesehus to determine if the identified attributes are optimal for this scenario.

ID	r3
Risk	Hesehus will not provide their current scoring algorithm
Overview	The current scoring algorithm is an obvious starting point for this project
Likelihood	Low
Impact	High
Severity	3
Mitigation plan	We are going to implement our own scoring algorithm

ID	r4
Risk	The database structure changes
Overview	Happens if we discover that the current database structure is not optimal for the implementation
Likelihood	Medium
Impact	Medium
Severity	3
Mitigation plan	We must refactor the database structure and re-generate test data or patch the current test data to fit the new database structure

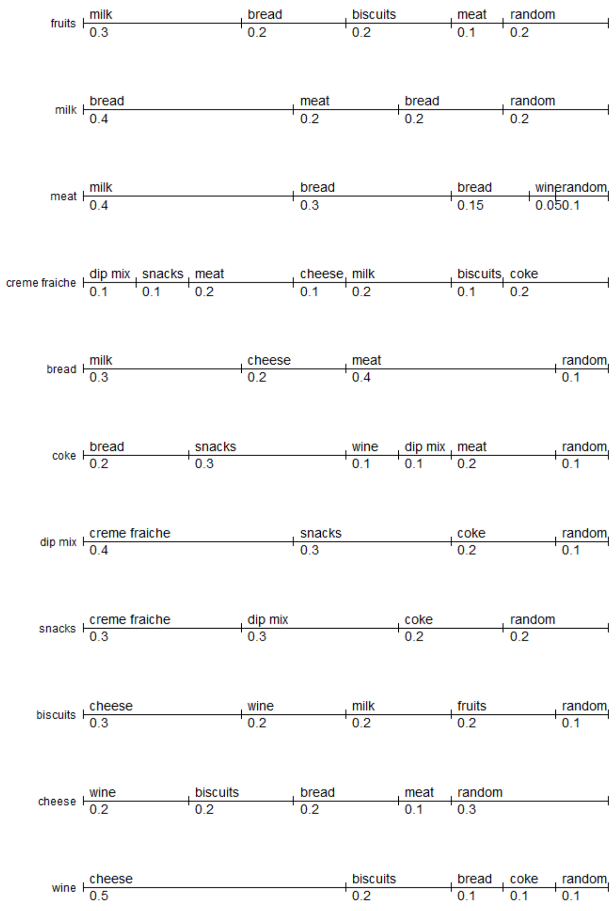
ID	r5
Risk	The currently used Elasticsearch implementation is too complex
Overview	The time taken to understand the structure of Elasticsearch takes more time than estimated
Likelihood	Low
Impact	High
Severity	3
Mitigation plan	We need to reschedule and take the time needed to understand Elasticsearch as it is at the core of this project

ID	r6
Risk	The custom scoring for Elasticsearch does not provide enough extendibility
Overview	We are unable to integrate a more intelligent scoring with Elasticsearh
Likelihood	Low
Impact	High
Severity	3
Mitigation plan	We need to create a software layer on top of Elasticsearch that implements the intelligent scoring and recommendations, while still utilizing the core tools that Elasticsearch provides

ID	r7
Risk	Associations rules mining fails to find any real associations in the dataset
Overview	We define real associations as products that could also be associated in the real world. An association from a computer to a bag of diapers would most likely be a false positive.
Likelihood	Medium
Impact	High
Severity	4
Mitigation plan	We need to adjust acceptance margins in the algorithm to sort out false positives.

ID	r8
Risk	The time frame is exceeded due to unexpected events or bad planning
Overview	A process like implementation or testing exceeds the estimated time. Unexpected outside events which we have no control over influence the schedule
Likelihood	Medium
Impact	High
Severity	4
Mitigation plan	We need to reevaluate the time schedule and allocate more working hours based upon need

C : Proportions for Purchase Patterns


fruits: milk 0.3, bread 0.2, biscuits 0.2, meat 0.1, random 0.2
milk: bread 0.4, meat 0.2, bread 0.2, random 0.2
meat: milk 0.4, bread 0.3, bread 0.15, wine 0.05, random 0.1
creme fraiche: dip mix 0.1, snacks 0.1, meat 0.2, cheese 0.1, milk 0.2, biscuits 0.1, coke 0.2
bread: milk 0.3, cheese 0.2, meat 0.4, random 0.1
coke: bread 0.2, snacks 0.3, wine 0.1, dip mix 0.1, meat 0.2, random 0.1
dip mix: creme fraiche 0.4, snacks 0.3, coke 0.2, random 0.1
snacks: creme fraiche 0.3, dip mix 0.3, coke 0.2, random 0.2
biscuits: cheese 0.3, wine 0.2, milk 0.2, fruits 0.2, random 0.1
cheese: wine 0.2, biscuits 0.2, bread 0.2, meat 0.1, random 0.3
wine: cheese 0.5, biscuits 0.2, bread 0.1, coke 0.1, random 0.1