U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

MSc

2.º
CICLO

FCUP
2015

U.PORTO

Mobile and Web Recommender System for Shopping

Luís Miguel Couto Moreira

FC

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

# Mobile and Web Recommender System for Shopping

Luís Miguel Couto Moreira
Dissertação de Mestrado apresentada à
Faculdade de Ciências da Universidade do Porto em
Ciência de Computadores
2015

# Mobile and Web Recommender System for Shopping

Luís Miguel Couto Moreira
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2015

**Orientador**
David Moura Ribeiro, MSc., Fraunhofer

**Coorientador**
João Pedro Pedroso, Prof. Doutor, FCUP

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

*Dedicated to my grandfather Manuel Couto Luiz and my dear friend Hugo Gouveia, although you both left you will always be with me*

# Acknowledgements

This work wouldn't be possible without the tremendous help of both of my advisors David Moura Ribeiro, Fraunhofer Portugal and João Pedro Pedroso, FCUP I want to thank them for their always helpful insights, patience and understanding.
I would also like to thank Nino Rocha from Fraunhofer Portugal for all is help with this thesis template and formatting.
I would also like to thank my parents José and Arminda, my brother José Luís and my girlfriend Catarina, I couldn't do this without all of you.
I couldn't end without thanking my friends Carlos Ferreira, Cristiano Monteiro, Mafalda Freitas, Mário Pinto, Miguel Lobo, Pedro Ramalho and Pedro Oliveira for all their support and for always being there.

## Abstract

Since the advent of digital point-of-sale systems in the early 90's, there has been an ever increasing interest in using the recorded transactions to obtain valuable information and relationships contained in the data. Today this field is incorporated in the broad research fields of data mining and machine learning.
With the transition from traditional brick and mortar retailers to e-commerce, this information has become easier to collect and explore.
At the same time, a typical online store contains thousands of products in its catalog, with this huge inventory it is impossible for a customer to known every single product and from there make the best purchasing decisions according to his/hers tastes. Recommendation systems were created to solve this problem.
They help customers by providing sensible product recommendations that the customer will appreciate, while providing to the stores potentially higher sales and lower costs related to product marketing.
In this thesis, recommendation systems will be explored with an emphasis on aided product replacement and periodically bought products recommendations, a prototype recommendation system that provides these features was developed and the system was tested on real transaction data provided by a large retail chain.
The system achieves encouraging results on traditional recommendations (used on aided product replacement) but the results of periodically bought products recommendations need further analysis.

## Resumo

Desde o aparecimento das caixas registadoras digitais no início dos anos 90, tem existido um interesse crescente em analisar a informação recolhida por estes sistemas, por forma a obter informações e relacionamentos importantes contidos nos dados. Esta área de estudo faz hoje em dia parte de dois grandes campos de investigação que são a prospeção de dados (do Inglês data mining) e aprendizagem automática (do Inglês machine learning).
Com a transição crescente das lojas de venda a retalho tradicionais para o comércio online, este tipo de informação tornou-se mais acessível e a sua posterior utilização mais facilitada.
Ao mesmo tempo, hoje em dia, o catálogo de uma loja online é composto por milhares de produtos diferentes, e com tanta escolha, torna-se impossível ao comum dos mortais conhecer todas as diferentes alternativas e daí escolher os produtos que melhor se adaptam às suas necessidades e gostos. Os sistemas de recomendação foram criados por forma a colmatar este problema.
Os sistemas de recomendação auxiliam os consumidores ao mostrar-lhes produtos pelos quais estes terão interesse, proporcionando ao mesmo tempo às lojas, maiores vendas potencias, bem como menos custos no marketing dos produtos.
Nesta tese, os sistemas de recomendação são explorados ao pormenor com ênfase no auxílio à recomendação de produtos substitutos, bem como à recomendação de produtos comprados periodicamente. Um protótipo foi desenvolvido que inclui estas funcionalidades e este foi testado em dados reais fornecidos por um grande retalhista.
O sistema demonstrou resultados encorajadores em recomendações tradicionais (substituição de produtos), enquanto que os resultados de produtos comprados periodicamente necessitam de ser melhor explorados.

# Contents

# List of Tables

# List of Figures

# Glossary

**ALS** Alternating Least Squares. 17, 37

**API** Application Programming Interface. 35

**CSV** A comma-separated values (CSV) is a file type that allows the storage of tabular data (strings/numbers) in plaint text. 35

**Fraunhofer AICOS** Fraunhofer Portugal Research Center for Assistive Information and Communication Solutions. 6

**GUI** Graphical User Interface. 6, 44, 50, 52, 53

**HDFS** Hadoop Distributed Filesystem. 28, 31

**HTTP** Hypertext Transfer Protocol. 33

**JDBC** Java Database Connectivity. 31

**JSON** JavaScript Object Notation. 33, 48

**JSP** JavaServer Pages. 50

**JVM** Java Virtual Machine. 41

**LDA** Latent Dirichlet Allocation. 20

**MBA** Market Basket Analysis. 6

**MLLib** Machine Learning Library. 37, 38

**MP** Marca Própria. 40

**POJO** Plain Old Java Object. 38

**PP** Primeiro Preço. 40

|  |  |
|---|---|
| **RDBMS** | Relational Database Management System. 31 |
| **RDD** | Resilient Distributed Datasets. 30, 38 |
| **REST** | Representational State Transfer. 33 |
| **RMSE** | Root Mean Squared Error. 39 |
| **SKU** | Stock Keeping Unit. 35, 36 |
| **SOAP** | Simple Object Access Protocol. 33 |
| **SQL** | Structured Query Language. 32 |
| **URI** | Uniform Resource Identifier. 33 |
| **VOD** | Video On Demand. 14 |
| **WSDL** | (Web Services Description Language. 33 |
| **WWW** | World Wide Web. 33 |
| **XML** | Extensible Markup Language. 33 |

# Chapter 1

# Introduction

## 1.1 Motivation

Currently customers are increasingly transitioning from shopping in traditional brick-and-mortar retail businesses in favour of e-commerce alternatives. With this transition comes the associated long tail phenomenon [9] where the number of products that are carried by a store is no longer limited by the physical space on the store's shelves, and thus the number of products carried can grow. Also, in the e-commerce world the cost of adding another product to the mix, or even ten more, is practically negligible.

Today's biggest online stores have a catalog size with an order of magnitude of hundreds of millions of products [44], with such an extensive catalog it is humanly impossible for customers to know every single product and from there make the best purchasing decisions according to their needs.

The long tail phenomenon concerns the fact that due to the increased product choice, the majority of sales cease to be concentrated on a small percentage of high selling products, to be more spread-out to a higher number of products with fewer sales each. These fewer sales of a higher number of items produces the so called long tail of the distribution.

Because of this, the need for recommendation systems is ever increasing and they are becoming an essential part of every user's online shopping experience.

From another perspective, recommendations have been proven to be a very effective way of increasing a stores' overall revenue [26].

From these two needs, recommendation systems were born. Recommendation systems aim to, through an assortment of techniques like data mining between others, provide to the user, at the right time, items that will be of his/hers interest.

This research area also has connections to Market Basket Analysis (MBA), a sub-area of affinity analysis in which one wishes to find co-occurrence of items, or putting it simply, items frequently bought together. This co-occurrence then allows for the implementation of various sales strategies like cross-selling or up-selling [51], building of loyalty programs, and even serves to influence brick-and-mortar store design.

## 1.2 Project Objectives

This master thesis was written in the context of a 6 months internship at Fraunhofer AICOS [4] in the months of March through September of 2015. The project was possible with the support of a major retail store chain company in Portugal, which provided customer retail transaction data for analysis and testing of the developed solutions. The project aimed to test different recommendation techniques using the retail chain data's and build a working recommendation system.

Two main objectives were first set. The first was to provide product recommendations specifically tailored for each user needs derived from his/her buying profile; these recommendations could also be limited to a product, subset such as a product category, or own brand/white label products. The second objective aimed to analyze each user's buying history to find periodic products (products that are recurrently bought within a well defined periodic interval) and then recommend these products at the right time (when the customer is most likely to accept the recommendation and buy the product).

The recommendation system should provide various web services that would allow product recommendations to be integrated into all sorts of clients, like an application on a smartphone device or a website.

Another objective was the development of a GUI prototype, for demonstration purposes.

## 1.3 Document Structure

The rest of this document is structured in the following way: In the next chapter, chapter 2 a review on the state of the art in the area of recommendation systems as well as some technologies used during the project development is made. In chapter 3, the developed work is described in detail and in chapter 4 some test results are shown. Finally, chapter 5 presents the conclusion and some future research directions.

Throughout this thesis, the terms product and item, and user and customer are used interchangeably.

# Chapter 2

# State of the Art

Every time we express/take an opinion about a product, being it an explicit opinion, like the one of fondness when we recurrently buy the same product, or an implicit opinion, when we constantly browse through the same product page without ever finalizing a buy, we build a preference profile that contains valuable information.

Recommendation systems try to harness the power of these opinions about products, being them positive or negative opinions, in order to present to the user other items that he/she may also like but hasn't yet discovered.

## 2.1 Recommendation Systems

Although every one of us has unique tastes, our tastes do follow patterns, if I like meat for a barbecue, with a high likelihood I will also like sausages and beer. Even more, with 7 billion people in the World [39], even the ones of us with the most obscure tastes are likely to find a very large group of peers that shares most of their preferences.

Recommendation systems seek to predict the opinion or the rating that a user would give to a product he/she isn't familiar with, through the power of aggregating the opinions of other users with similar tastes.

In this section a review on the state-of-the-art of recommendation systems and some technologies used in the project is presented.

### 2.1.1 Similarity Measures

To find these similar users to power the recommendations, recommendation systems need to use a measure of similarity. A measure of similarity seeks to quantify the similarity between two objects.

Before proceeding with a recommendation systems review, some similarity measures used in all of these systems need to be presented.

Example of a ratings matrix, in this case, of user's ratings of films:

**Table 2.1:** Film ratings by customers on a scale of 0-5

| **Customers** | **Films** | | | |
|---|---|---|---|---|
| | **Fight Club** | **Mad Max** | **The Hunger Games** | **The Godfather** |
| **John** | 4.8 | | 2.2 | |
| **Mary** | 4.5 | 4.0 | 3.9 | 4.3 |
| **Oliver** | 5.0 | | | |
| **Silvia** | 3.0 | 3.2 | 4.5 | 5.0 |
| **Sofia** | 2.0 | 2.5 | 4.5 | 2.8 |

(Missing values indicate no rating)

#### 2.1.1.1 Cosine Similarity

The cosine similarity measures the difference in orientation between two vectors in the same inner product space, it can be calculated through the following formula:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

It is not a measure of magnitude because two vectors with very different magnitudes but the same orientation will have a cosine similarity of 1. Two vectors with opposing directions will have a cosine similarity of -1 and two perpendicular vectors will have a cosine similarity of 0.

Take for example the ratings present in table 2.1, lets calculate the cosine similarity between Mary and Sofia:

$$similarity(Mary, Sofia) = \frac{(4.5 \times 2.0 + 4.0 \times 2.5 + 3.9 \times 4.5 + 4.3 \times 2.8)}{\sqrt{(4.5^2 + 4.0^2 + 3.9^2 + 4.3^2)} \times \sqrt{(2.0^2 + 2.5^2 + 4.5^2 + 2.8^2)}}$$
$$\approx 0.938$$

Instead of using the observed ratings directly, it is very common in recommendations systems to normalize the ratings first, this is done in order to account for rater bias.

Two types of rater bias can be considered, raters which consistently rate bellow average (called harsh/hard raters) and raters which tend to consistently rate above the average (called soft raters).

A popular normalization technique is to subtract each rating by the average rating given by the user. Using this technique, and starting from table 2.1, a new table, 2.2, can be calculated.

**Table 2.2:** Film ratings normalized

| **Customers** | **Films** | | | |
|---|---|---|---|---|
| | **Fight Club** | **Mad Max** | **The Hunger Games** | **The Godfather** |
| **John** | 1.300 | | -1.300 | |
| **Mary** | 0.325 | -0.175 | -0.275 | 0.125 |
| **Oliver** | 0 | | | |
| **Silvia** | -0.925 | -0.725 | 0.575 | 1.075 |
| **Sofia** | -0.950 | -0.450 | 1.550 | -0.150 |

(Missing values indicate no rating)

By normalizing by the mean, the results are called mean-centered and just by looking at the rating's signal one knowns if the user has a special preference for the item (rated above average) or not (rating equal or less than zero). Take note that by normalizing by the average value, the user ratings end up with a mean of zero.

Using the ratings on table 2.2 we get the following cosine similarity:

$$similarity(Mary, Sofia) \approx -0.753$$

#### 2.1.1.2 Pearson Correlation Coefficient and Distance

The Pearson product-moment correlation coefficient will give a value between [-1,1] of the correlation between two variables. It gives the linear dependence between the two variables, 1 signifies total positive correlation, 0 no correlation and -1 total negative correlation.

It can be calculated through the following formula:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

- cov is the covariance
- $\sigma_X$ is the standard deviation of $X$

The Pearson's distance can be calculated by the following formula:

$$d_{X,Y} = 1 - \rho_{X,Y}$$

Because $\rho_{X,Y}$ varies between [-1,1] the Pearson's distance varies from [0,2].

Again, using table 2.1 we get the following Pearson's distance:

$$\begin{aligned}\rho_{Mary,Sofia} &= \frac{\operatorname{cov}(Mary, Sofia)}{\sigma_{Mary}\sigma_{Sofia}} \\ &\approx \frac{-0.16875}{0.238 \times 0.939} \\ &\approx -0.753\end{aligned}$$

$$\begin{aligned}d_{Mary,Sofia} &= 1 - \rho_{Mary,Sofia} \\ &\approx 1 - (-0.753) \\ &\approx 1.753\end{aligned}$$

And if we use table 2.2 we get the following result:

$$\begin{aligned}\rho_{Mary,Sofia} &= \frac{\operatorname{cov}(Mary, Sofia)}{\sigma_{Mary}\sigma_{Sofia}} \\ &\approx \frac{-0.16875}{0.238 \times 0.939} \\ &\approx -0.753\end{aligned}$$

$$\begin{aligned}d_{Mary,Sofia} &= 1 - \rho_{Mary,Sofia} \\ &\approx 1 - (-0.753) \\ &\approx 1.753\end{aligned}$$

So, from these results we can see that contrary to cosine similarity, the Pearson's distance is not susceptible to the domain of the variables being used.

We can also verify that if the values are mean-centred, as in table 2.2, the cosine similarity (called in this case centred cosine similarity) is equivalent to the Pearson Correlation Coefficient.

### 2.1.2 Collaborative Filtering Based Recommenders

Collaborative filtering based as the name entails is a product recommendation technique that filters the candidate products (products that are available to be recommended) through some criteria, one can think of this criteria as the "Wisdom of the crowd".

Collaborative filtering explores similarities between customer's tastes or between the buying patterns of different products to generate recommendations.

There are two main collaborative filtering variants, the first being heuristic-style neighborhood based systems and model based systems.

#### 2.1.2.1 User-to-User Collaborative Filtering

User-to-User collaborative filtering (also called user-based collaborative filtering) is a neighborhood based collaborative filtering technique, where as the name entails, similarity between users is used in order to generate recommendations.

To generate recommendations to a target user, a group of $N$ most similar users is found, and to predict the rating of a new product (to the target user) an aggregation function of the ratings given by the similar users to that product is took. In the end, the item(s) that score the highest is(are) the recommendation(s).

Take the following example:

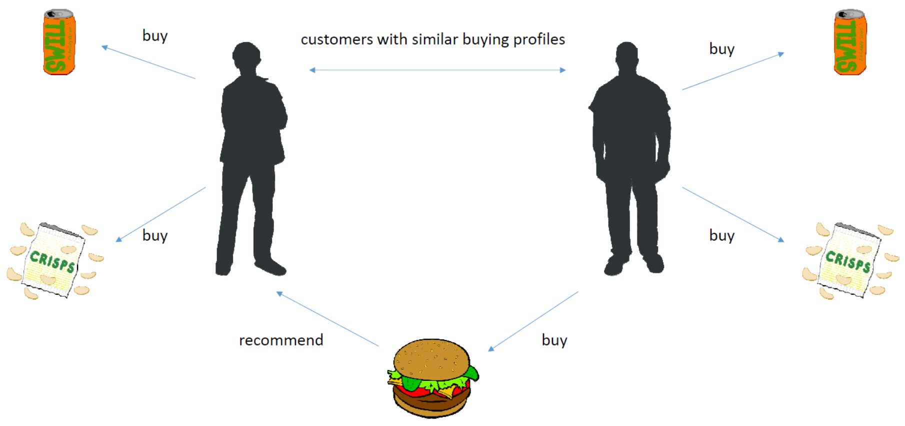


**Figure 2.1:** User-to-User recommendation example

Take the example given in figure 2.1, where the buying profile of both users is very similar, both have bought potato chips and a soda.

From this, when the user on the right buys an hamburger, since the two customers have very similar tastes, the recommendation system can act, and recommend the hamburger to the user on the left with a good assumption that the left user will appreciate the recommendation.

In general, the aggregation function of the ratings of the $N$ most similar users for an item can be as simple as averaging all the ratings or be more advanced by calculating giving weights to each rating (weighted average).

To calculate these weights two popular measures are for the weight to be the inverse of the distance between the target user ratings and the other user ratings, or, use another measure like tf-idf [42], [8] on the ratings of the two users in question.

#### 2.1.2.2 Item-to-Item Collaborative Filtering

Another neighborhood technique is Item-to-Item collaborative filtering (also called item-based collaborative filtering), where the similarity between items is used in order to generate recommendations.

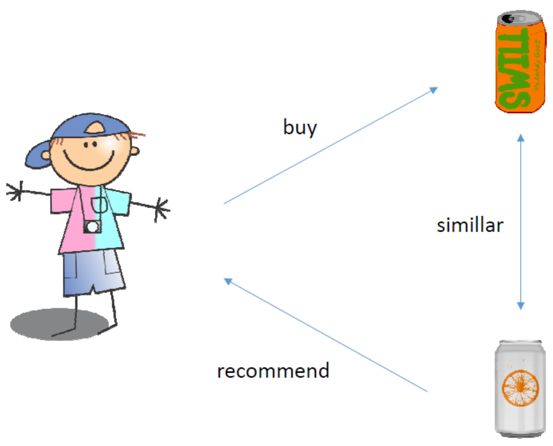
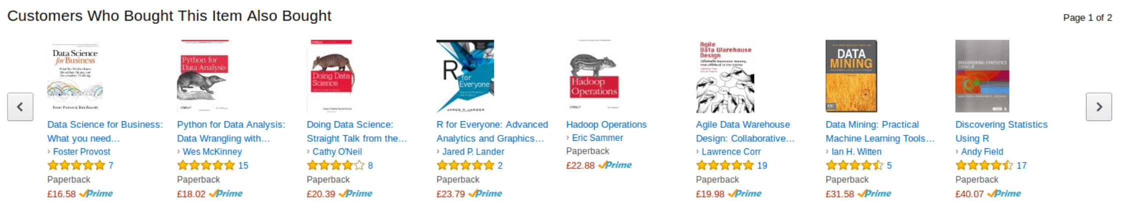


**Figure 2.2:** Item-to-Item recommendation example



Customers Who Bought This Item Also Bought

Page 1 of 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Data Science for Business: What you need… | Python for Data Analysis: Data Wrangling with… | Doing Data Science: Straight Talk from the… | R for Everyone: Advanced Analytics and Graphics… | Hadoop Operations | Agile Data Warehouse Design: Collaborative… | Data Mining: Practical Machine Learning Tools… | Discovering Statistics Using R |
| › Foster Provost | › Wes McKinney | › Cathy O'Neil | › Jared P. Lander | › Eric Sammer | › Lawrence Corr | › Ian H. Witten | › Andy Field |
| 7 | 15 | 8 | 2 | | 19 | 5 | 17 |
| Paperback | Paperback | Paperback | Paperback | Paperback | Paperback | Paperback | Paperback |
| £16.58 Prime | £18.02 Prime | £20.39 Prime | £23.79 Prime | £22.88 Prime | £19.98 Prime | £31.58 Prime | £40.07 Prime |

**Figure 2.3:** Item-to-Item recommendations at Amazon

Starting from an item, lets call it $I_0$, to generate a recommendation for $I_0$, the algorithm first finds all the cases where a customer has bought $I_0$ and another item $I_i$ and registers these cases in an Item-to-Item affinity matrix. The next step is taking the corresponding column/row for $I_0$ and calculate the similarity between it and all the products that were bought with it with for example the previously described cosine similarity. The most similar items are the algorithm's recommendations [36].

**Algorithm 1** Pseudocode for an Item-to-Item similarity computation algorithm

```
1: for each item $I_1$ in product catalog
2:     for each customer $c$ who purchased $I_1$
3:         for each item $I_2$ purchased by customer $c$
4:             Record that customer $c$ purchased both item $I_1$ and item $I_2$
5: for each item $I_2$
6:     Compute the similarity between $I_1$ and $I_2$
```

Example of a User-to-Item Matrix that could be used in a Video On Demand (VOD) movie rental website:

**Table 2.3:** N*M User-to-Item Matrix with ratings given to Movies by the Customers

| **Customers** | **Films** | | | |
|---|---|---|---|---|
| | **Fight Club** | **Mad Max** | **The Hunger Games** | **The Godfather** |
| **Paul** | 2.3 | | -0.3 | |
| **Mercy** | 2.0 | 1.5 | 1.4 | 1.8 |
| **Daniel** | 2.5 | | | |
| **Jane** | 0.5 | 0.7 | -0.5 | 2.5 |

(Missing values indicate no rating)

Ratings have been normalized to the scale [-2.5,+2.5]

By using algorithm 1 with cosine similarity measure, one would get the following Item-to-Item similarity matrix:

**Table 2.4:** M*M Item-to-Item Matrix with affinity on a scale [-1,1]

| **Films** | **Films** | | | |
|---|---|---|---|---|
| | **Fight Club** | **Mad Max** | **The Hunger Games** | **The Godfather** |
| **Fight Club** | 1.0 | 0.5 | 0.3 | 0.4 |
| **Mad Max** | 0.5 | 1.0 | 0.7 | 0.9 |
| **The Hunger Games** | 0.3 | 0.7 | 1.0 | 0.3 |
| **The Godfather** | 0.4 | 0.9 | 0.3 | 1.0 |

#### 2.1.2.3 Matrix Factorization Model Based Collaborative Filtering

Matrix factorization [35] methods are in the class of latent factor models, they infer both user and items characteristics through item rating history (for example customers' buys).

These latent factors (also called model features, and typically in the order of 20-100) are what describes an object in the model. For a product, factors could be color, smell, packaging, price, brand, etc, etc, and for customers the factors would equate to the affinity that the customer has to each of these product factors.

The model infers these factors automatically without explicit input and some of these factors could even be some uninterpretable dimension.

A matrix factorization model builds for each customer and for each product vectors with affinity factors.

The products affinity for user $u$ named $p_u$ is a vector of equal size to the number of items and measures $u's$ affinity to each product.
The item's $i$ affinity to each user corresponds to the vector $q_i$ of equal size to the number of users. Aggregation these vectors for all the items and users generates one matrix of factors for users $\{\forall p_u | u \in Users\}$ and another for items $\{\forall q_i | i \in Items\}$.

The matrix multiplication on column x row gives an approximation of the true rating (and equals the dot product of the two vectors):

$$\check{r_{ui}} = q_i^T p_u \approx r_{ui}$$

A matrix factorization model tries to reduce this difference between the true rating and the approximated rating for all products and users, this can be formulated as:

$$\min_{q*,p*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

The previous equation can be optimized through Alternating Least Squares (ALS), since both $q_i$ and $p_u$ are not know, ALS alternates between fixing one and then the other in order to optimize the approximation until it converges or there are very small changes to the matrices [18].

#### 2.1.2.4 Other Model Based Collaborative Filtering

The main difference between model based and neighborhood based recommendations systems is that model based recommenders build an actual model based on statistics or data mining techniques instead of an ad-hoc heuristic as the neighborhood based models. One of their advantages over heuristic-based collaborative filtering recommenders is that they can detect implicit user feedback instead of only explicit actions, model based recommenders have also been show to perform significantly better in some types of domains and because the computational-intensive part is the model creation, which can be done offline, they provide better online recommendation performance [43].

Unlike heuristic-based recommenders, model based techniques require a training and validation phase so as to construct the model that fits the best with the dataset at hand.

Some typical model based techniques include:

- Bayesian Networks [8], [12]
  In a Bayesian Network model, a conditional probabilistic model is built where each state corresponds to an item and each path leads to a choice of voting (purchasing/watching/reviewing) for an item or not. The model builds a decision tree based on the user's history and completes missing items based on other similar trees, attaching the right conditional probability to each state.
- Clustering
  An alternative model formulation is to cluster users and items with an algorithm like k-means or $k$-nearest neighbors.

  One common approach is to cluster users based on which movies they have watched and then cluster the movies based on who watched them. On the next step, the users are re-clustered based on the number of movies of each cluster that they have watched and the movies are also re-clustered based on the number of people in each cluster that as watched them [50].

  Then it is trivial calculating the probability of a user in class k liking a movies in class l, expressed as $p_{kl}$ and recommending the movies with higher probability.

#### 2.1.2.5 Problems With Collaborative Filtering Based Recommenders

One problem with collaborative filtering recommenders is the so called "Cold Start Problem" that occurs anytime a new user or product is added to the system, because the user hasn't given any ratings, nor has the product received any ratings, neighborhood based methods cannot recommend because they cannot calculate any similarity/distance.

Another problem is data sparsity, in a typical $N * M$ user-to-item matrix with thousands of rows and columns, and because a user usually only knows a small subset of the all catalog, the matrix is going to be very sparse with lots of combinations with no rating at all, this can be a challenge in algorithm design for recommendation correctness with less data and speed.

A third problem is data they tend to be biased towards recommending already popular items [17].

### 2.1.3 Content Based Recommenders

A content based recommendation system, recommends new products based on the products' characteristics/features.

To generate a recommendation, the customer's preferences are matched to the characteristics of the candidate products, the product(s) that most suit the customer preferences are recommended.

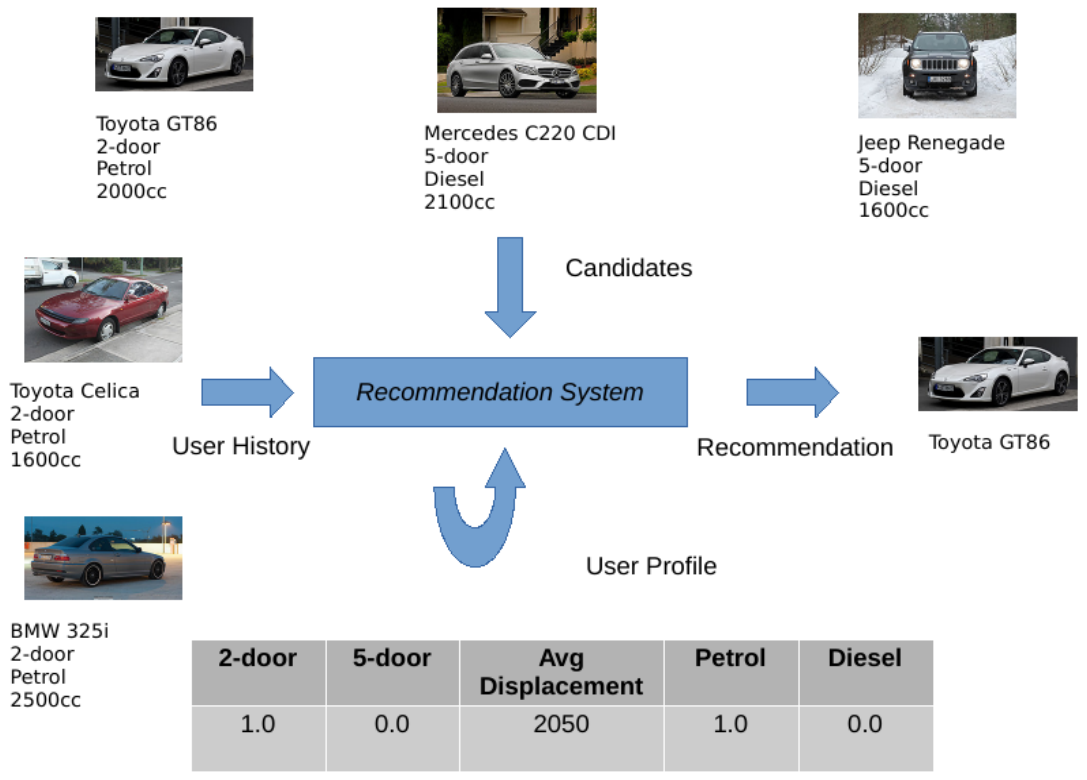


**Figure 2.4:** Content-based recommendation example [1]

The recommender builds for each customer a personalized preferences profile based on the features of the products that he/she has bought in the past [1].

Take the example first presented in the previous section of a VOD movie rental website.

A table that depicts each movie characteristics could be:

**Table 2.5:** Movies depicted by gender as characteristics on a scale of [0-5]

| **Films** | **Genres** | | | | |
|---|---|---|---|---|---|
| | **Drama** | **Crime** | **Thriller** | **Adventure** | **Sci-Fi** |
| **Fight Club** | 5.0 | 0.0 | 0.0 | 3.0 | 0.0 |
| **Mad Max** | 0.0 | 0.0 | 4.0 | 4.0 | 4.0 |
| **The Hunger Games** | 0.0 | 0.0 | 0.0 | 4.0 | 4.5 |
| **The Godfather** | 5.0 | 4.5 | 0.0 | 0.0 | 0.0 |

A transaction like table of who watched what could look like:

[1]Car Images from the following Flickr users (in top down, left to right order) : 55391407@N03, 126433814@N04, pelice, nrmadriversseat, janitors, 55391407@N03. No changes were made to the images. CreativeCommons Attribution 2.0 Generic license.

**Table 2.6:** Movies watched by Customer, on a binary scale, 1 if watched 0 if not

| **Customers** | **Films** | | | |
|---|---|---|---|---|
| | **Fight Club** | **Mad Max** | **The Hunger Games** | **The Godfather** |
| **Paul** | 1 | 0 | 1 | 0 |
| **Mercy** | 1 | 1 | 1 | 1 |
| **Daniel** | 1 | 0 | 0 | 0 |
| **Jane** | 1 | 1 | 1 | 1 |

The customer profile preference for each genre could be calculated with the following formula:

$$P(g,c) = \frac{\sum_{i=1}^{k} w(c,i) * a(g,i)}{\sum_{i=1}^{k} w(c,i)}$$

where:

- $g$ is the movie genre
- $c$ is the customer id
- $w(c,i)$ is a boolean function that returns 1 if customer c watched movie i and 0 if not
- $a(i)$ is a function that returns the movie i affinity to the genre g.

The result would be:

**Table 2.7:** Customer profile based on watched movies characteristics

| **Customers** | **Genres** | | | | |
|---|---|---|---|---|---|
| | **Drama** | **Crime** | **Thriller** | **Adventure** | **Sci-Fi** |
| **Paul** | 2.50 | 0.00 | 0.00 | 3.50 | 2.25 |
| **Mercy** | 2.50 | 1.13 | 1.00 | 2.75 | 2.13 |
| **Daniel** | 5.0 | 0.00 | 0.00 | 3.00 | 0.00 |
| **Jane** | 2.50 | 1.13 | 1.00 | 2.75 | 2.13 |

In our example, the feature aggregation function is very simple, some systems use a weighted average with an item presentation algorithm like tf-idf (term frequency/inverse document frequency) representation to abstract item features, by giving weights to features when building a user profile [42], [8].

#### 2.1.3.1 Problems With Content Based Recommenders

A problem with content based recommenders is that because of the way they recommend based on the products' characteristics, to work effectively they require a detailed product database (with detailed descriptions for each product) and a solid hierarchical product categorization, this requires many man-hours to achieve which is not feasible for most organizations. An exception is that for recommenders in domains that deal with text based products (websites, books, newspapers, etc), a technique like Latent Dirichlet Allocation (LDA) can be used to automatically categorize products [7].

Another problem is that content based recommenders tend to recommend similar products to those that customer has already bought, this process is called overspecialization [1] and happens because products are only ranked highly if their characteristics approach the ones of the customer profile that was built with his/hers purchase history.

### 2.1.4 Hybrid Recommenders

Hybrid recommenders were developed because each recommender type suffer's from some type of problem/drawback. Hybrid recommenders combine the results of two or more recommendations systems into the final result.

When building a Hybrid recommender the most critical task is to find the best way combine the recommendations of the several recommenders integrated into the system. There are several variants of hybrid recommenders, of which the most important are:

- Cascade-Hybrid Recommenders
  In Cascade-Hybrid recommenders, the output of a first recommender serves as the input to the next one successively until reaching the last one. We can model the recommenders at each type as a filter where some candidate products are dropped following some criteria until the surviving ones reach the end of the system and become the recommendations [33, Chapter 5].
- Weighted-Hybrid Recommenders
  In Weighted-Hybrid recommenders, the recommendations of each included recommender are weighted and combined together to form a final list of recommendations. The simplest example could be a system that performs a linear combination of all the recommenders' output. A more advanced system could adjust the weights in training/validation phases to better adapt to the dataset at hand [10]. Another variant could be a voting hybrid recommender where all the recommenders reach a consensus on the recommendations to give [38].

Lately some more exotic Hybrid recommenders have been proposed like [46], [22] and [19], they apply more advanced methods to combine the recommendations from multiple sources.

## 2.2 Data Mining

### 2.2.1 Association Rules

In the early 90's, the first generation of recommendation systems used association rule learning to generate recommendations. Association rule learning scans the transaction history to find meaningful associations between items (products), it finds items frequently bought together.

These frequently bought together items are grouped into itemsets, each group of one or more items that are bought at the same time produce an itemset.

For example, for the transaction $T$ with 2 items such that $T = \{X, Y\}$ the following itemsets can be generated $\{(X), (Y), (X, Y)\}$, from this single transaction two 1-itemsets and one 2-itemsets can be derived.

#### 2.2.1.1 Algorithms

The first step of finding association rules consists of finding large itemsets (itemsets that exceed the predefined minimum support threshold, as opposed to the so called small itemsets which don't). Suppose that $D$ is a set of transactions, the support for the itemset $(X, Y)$ can be calculated by the following formula:

$$Supp(X \cup Y) = \frac{|\{T | X \cup Y \subseteq T\}|}{|D|}$$

A visualization of the itemset formation phase is presented in figure 2.5.

The second step is, for each large itemset $Y = I_1 I_2 ... I_k, k \geq 2$, generate all the possible association rules, which are at the most $k$ [2]. An association rule takes the form of an implication where the antecedent is a set of size $k - 1$ and the consequent is the remaining item in Y that is not in the antecedent. Suppose that two items $X, Y$ occur in transaction $T$, such that $X \cup Y \subseteq T$, an association rule in the form of $X \Rightarrow Y$ can be derived, it means that when $X$ occurs in a transaction, $Y$ also occurs. Suppose now that the transaction $T = \{milk, butter, bread\}$ because $|T| = 3$ the following three association rules can be derived:

- $\{milk, butter\} \Rightarrow bread$
- $\{milk, bread\} \Rightarrow butter$
- $\{butter, bread\} \Rightarrow milk$

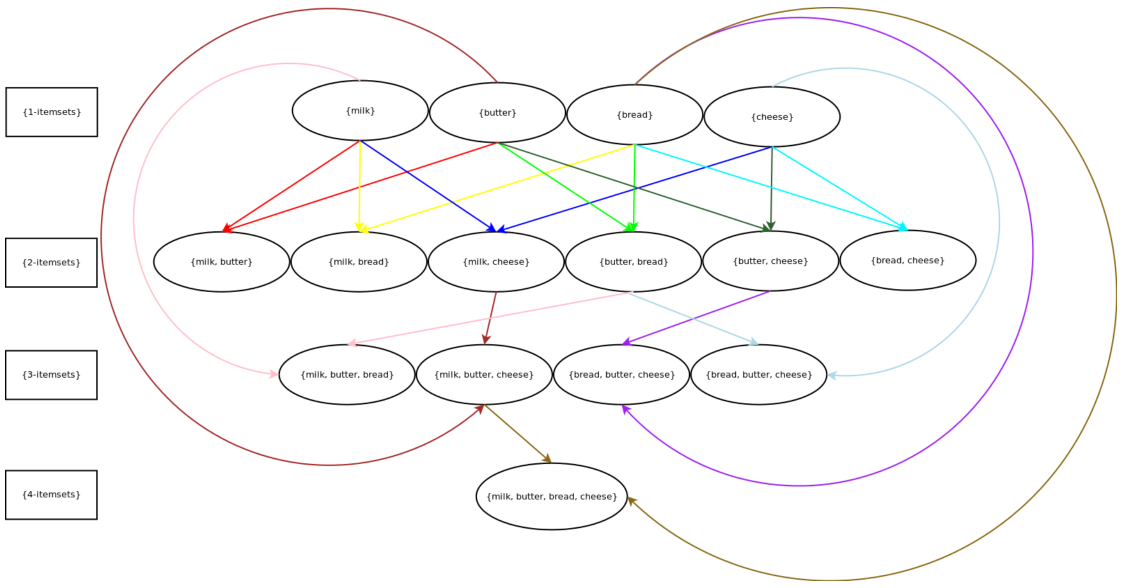


**Figure 2.5:** Candidate formation phases without cut from lack of support

The final step is to calculate the confidence of the derived association rules and see which ones equal or exceed the desired confidence threshold. The confidence of an association rule is the number of transactions where both the antecedent and the consequent appear together divided by the number of transactions that contain the antecedent. For the association rule $\{X, Y\} \Rightarrow Z$ the confidence can be calculated by the following formula:

$$Conf(\{X,Y\} \Rightarrow Z) = \frac{Supp(X \cup Y \cup Z)}{Supp(X \cup Y)}$$

The most common algorithm for mining association rules is the Apriori algorithm, it was created by Rakesh Agrawal and Ramakrishnan Srikant [3]. The Apriori algorithm, presented in pseudo-code form in algorithm 2, operates in two phases.

In the first phase, which is the most computationally costly, it calculates the large itemsets present in the transactions database. The first phase is a bottom-up search algorithm that starts with the large 1-itemsets as the candidates, at each loop the candidates are expanded by one element, if these new itemsets present a support that equals or exceeds the support threshold they are added to the result list and advance to the next loop as candidates.

The search space is cut by taking advantage of the fact that if an itemset is small (it's support is bellow the desired threshold) any other set that is constructed by expanding this itemset will also be small [3], as so, this itemset will not be expanded and added to the candidates list to analyze at the next loop run.

The second phase of Apriori just takes the result list of confirmed large itemsets, generates all the association rules based on them and outputs the rules whith confidence equal or greater than the predefined confidence threshold.

**Algorithm 2** Apriori algorithm taken from [3]

```
1: $L_1 = \{\text{large 1-itemsets}\}$
2: for $(k = 2; L_{k-1} \neq \emptyset; k++)$ do
3:     $C_k = apriori - gen(L_{k-1})$                     ▷ Candidates Generation
4:     for all transactions t $\in D$ do
5:         $C_t = subset(C_k, t)$                         ▷ Candidates contained in t
6:         for all candidates $c \in C_t$ do
7:             c.count++
8:         end for
9:         $L_k = \{c \in C_k | c.count \geq minsup\}$
10:    end for
11: end for
12: Answer = $\cup_k L_k$
```

The apriori-gen function, presented in pseudo-code in algorithm 3, is a helper function that generates the candidate itemsets by expanding by one item the previously found large itemsets.

**Algorithm 3** apriori-gen taken from [3]

```
1: function APRIORI-GEN($L_{k-1}$)
2:     insert into $C_k$                                  ▷ Candidates List
3:     select $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$
4:     from $L_{k-1}p$, $L_{k-1}q$
5:     where $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
6:
7:     Next, in the prune step, we delete all itemsets $c \in C_k$ such that some $(k-1)$-subset of c
   is not in $L_{k-1}$:
8:     for all itemsets $c \in C_k$ do
9:         for all $(k-1)$-subsets s of c do
10:            if $s \notin L_{k-1}$ then
11:                delete c from $C_k$
12:            end if
13:        end for
14:    end for
15: end function
16: Answer = $C_k$
```

The Apriori algorithm has since been superseded by faster algorithms such as Max-Miner [6] and FP-Growth [23] that achieve the same results but are more efficient with candidate generation.

#### 2.2.1.2 Using Association Rules To Generate Recommendations

In the context of recommendation systems, association rules can provide next-item recommendation, that is, they can answer the question "Taking into account the previous items bought by the customer, what is the item that he/she buys next?".

To recommend to a specific customer, a recommendation system can match the customer's current shopping cart (or sub-sets of it) with previously generated association rules, derived from the transactions of all the users, and chose the item to recommend as the consequent of the association rule with the higher confidence.

### 2.2.2 Statistics

Each customer can have hundreds of bought products in dozens of shopping visits, a product can have hundreds or even thousands of distinct buys, to further analyze the dataset at hand and during the recommendation system development some statistical tools were used.

#### 2.2.2.1 Z-test

The Z-test is used to verify if a test statistic (population sample) can be approximated by a normal distribution. It compares the samples' mean against the mean of the distribution.

First, the standard error of the mean is calculated:

$$SE = \frac{\sigma}{\sqrt{n}}$$

Where $\sigma$ is the standard deviation of the general population and $n$ is the population size.

Then, the Z-score is calculated:

$$z = \frac{M - \mu}{\text{SE}}$$

Where:
$\mu$ is the population mean
$M$ the sample mean.

Then, using a Z-score table for the normal distribution we find the probability of observing a normal value bellow our Z-score. If the result is statistically significant, we can reject with the previously calculated probability the null hypothesis $H_0$ that our sample is equal to a random sample of the population [47].

#### 2.2.2.2 Normality Tests

The normal distribution $N(\mu, \sigma^2)$ is used in many scientific fields to represent random variables whose underlying distribution is not known. The normal distribution has been shown to model well real world phenomena.

A standard practice is to compare the results of an experiment (our sample) to the values of a normal distribution and verify if our sample could be taken from a normal distribution or if it deviates from it significantly, to achieve this normality tests were developed. A normality test is used to determine if a dataset is well-modeled by a normal distribution or not, to determine this one tries to disprove with statistical significance the null hypothesis $H_0$ that the sample came from a normal distribution.

Three widely used normality tests are the Pearson chi-square normality test, for discrete variables, and the Kolmogorov–Smirnov test and the Cramér–von Mises criterion for continuous variables.

#### 2.2.2.3 Pearson Chi-Square Normality Test

The Chi-Square goodness-of-fit test can be used to find if a sample came from a population with a specific distribution, for example, it can be used to find the normality of a set of values, in that case it is called the Pearson chi-square normality test [13].

To calculate the test statistic, first, the sample is divided into $k$ bins and then the values distribution is compared to the cumulative distribution function for the normal distribution.

The test value is calculated by the following formula:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

where:

- $\chi^2$ is the Pearson's cumulative test statistic, which asymptotically approaches a $\chi^2$ distribution
- $k$ is the number of bins
- $O_i$ is the observed frequency for bin $i$
- $E_i$ is the expected frequency for bin $i$

The expected frequency is calculated by the following formula:

$$E_i = N(F(Y_u) - F(Y_l))$$

where:

- $F$ is the cumulative distribution function for the distribution being tested
- $Y_u$ is the upper limit for class $i$
- $Y_l$ is the lower limit for class $i$
- $N$ is the sample size.

There is no optimal value for $k$, the number of bins, but for the chi-square approximation to be valid the expected frequency should be at least 5.

The null hypothesis is rejected, and therefore the sample doesn't follow a normal distribution, if the test value $\chi^2$ follows the following criteria:

$$\chi^2 > \chi^2_{1-\alpha,k-c}$$

where:

- $\alpha$ is the significance level
- $k - c$ are the degrees of freedom (c for the normal distribution is 3)

## 2.3 Technologies

The provided dataset was only a sample of the overall available data, this sample was in itself already of a large size and on the limit to what the called traditional technologies can handle. Because of this and the expected use case of the recommendation system, serving hundreds of recommendations a minute it was decided that big data technologies should be used on the project implementation.

### 2.3.1 Big Data

The term big data is used to refer to a myriad of things, the common denominator is that it applies to situations where huge amounts of data is generated, data that is so big that common applications/PC hardware cannot handle. Big data is mostly employed when talking about predictive analysis, the discipline that gathers meaningful information from past data and tries to extrapolate that knowledge into predicting future events. This predictive capacity is very useful and predictive analysis has become an indispensable tool in the areas of management, science, financial services between many others.

### 2.3.2 Parallel Computing

Parallel computing refers to the technique of splitting the computations between various processors/computers in order to obtain the final results faster. When building a parallel computing systems taxonomy there exist two separate classification models, systems differentiated by computation models and differentiated by communication model.

There are three distinct distributed systems communication paradigms: *shared memory systems*, *distributed memory systems* and hybrid systems called *distributed-hared memory* [24, page 7]. *Distributed memory systems* must communicative via message passing only, because, the various computing processes do not share the same physical computer and thereof cannot have access to the same memory space. *Shared memory systems* as the name entails communicate via shared memory space, this allows for lower parallelism costs but also has the cons of limiting the parallel processes to the same computer, *Distributed-shared memory* is a hybrid model where memory address space is provided via a logical overlay over the physical memory space, so all the processes can address the entire memory space in the cluster even if it resides in another computer.

In the realm of computation models, two alternative models exist: *functional parallel* and *data parallel* [24, page 7]. The first achieves parallelism by spawning multiple processes with different instructions (code) that relate to doing different tasks, whereas in the second model, *data parallel*, all the nodes run the same code and parallelization is achieved by chunking the input data into smaller pieces and sending each chuck of the data to a different node of the cluster.

#### 2.3.2.1 Apache Hadoop

Apache Hadoop was inspired by Google's advances on big data technologies, namely the computation model *MapReduce* [15] and the distributed filesystem *Bigtable* [11]. Hadoop was developed from the ground up to be run on commodity desktop hardware [52, page 42] because of it's relative cheapness while still providing a full fault-tolerant distributed filesystem and a parallel computing framework.

Hadoop Distributed Filesystem (HDFS) was designed with the intent of providing a unified filesystem for all the machines in a cluster to access. HDFS was designed to be tolerant to node failures without losing data and can be used to share very large files, files with sizes of megabytes, gigabytes and even terabytes, there are active Hadoop cluster which store petabytes of data [48]. The HDFS architecture is composed of typically one namenode (there can be two in certain configurations) and severall datanodes working as master and workers respectively. The namenode manages the filesystem namespace maintaining the filesystem tree and all the associated metadata whereas the datanodes keep the actual data stored in blocks like in a typical filesystem, the namenode knows the location of all the blocks for each file in the filesystem.

HDFS is built for a streaming like data access pattern, where a file is wrote once and read multiple times, so it focus on overall throughput sacrificing latency. HDFS is not appropriate for an application where lots of small files need to be write/read due to the above mentioned higher access/write latency and because all the filesystem metadata (names, etc) is kept in memory at all times by the namenode, with several thousand/millions of files the memory usage is significantly high.

MapReduce is a parallelization technique that happens in two phases, first the map phase and then the reduce phase. The two phases have key-value pairs as input and output and the user must provide suitable map and reduce functions. In the map phase the relevant input data is split into key-value pairs for better processing latter, here bad records are also dropped, at the end the record are grouped by key. Take for example the student's grades in several courses at a faculty, the goal is to find the highest grade for each course:

```
Input for map phase
(student number, course number, grade)                    ▷ format

$(1234, 101, 17)$
$(1235, 101, 15)$
$(1234, 102, 16)$
```

```
Ouput from map phase
(course number, grades)                                   ▷ format

$(101, [17, 15])$
$(102, [16])$
```

From the map phase, data is passed to the reduce phase which has to iterate through the grades for each course and find the higher value, here each course's grades can be analyzed in parallel. Taking the example above, the final result would look like:

```
Output from reduce phase (final result)
(course number, highest grade)                            ▷ format

$(101, 17)$
$(102, 16)$
```

On Hadoop a MapReduce job is a unit of work, it consists of the program itself, any input data and a suitable configuration file. This job is run by splitting it into tasks that are tracked by a number of tasktrackers whereas all the jobs are tracked by the jobtracker.

The flow of a typical MapReduce job is presented in figure 2.6, in the input phase the data is split into multiple bins that are mapped and sorted by mappers into key-value pairs. On the next phase, the values are then reduced on the reducers using the keys. The final result is then produced by merging the results of the various reducers.

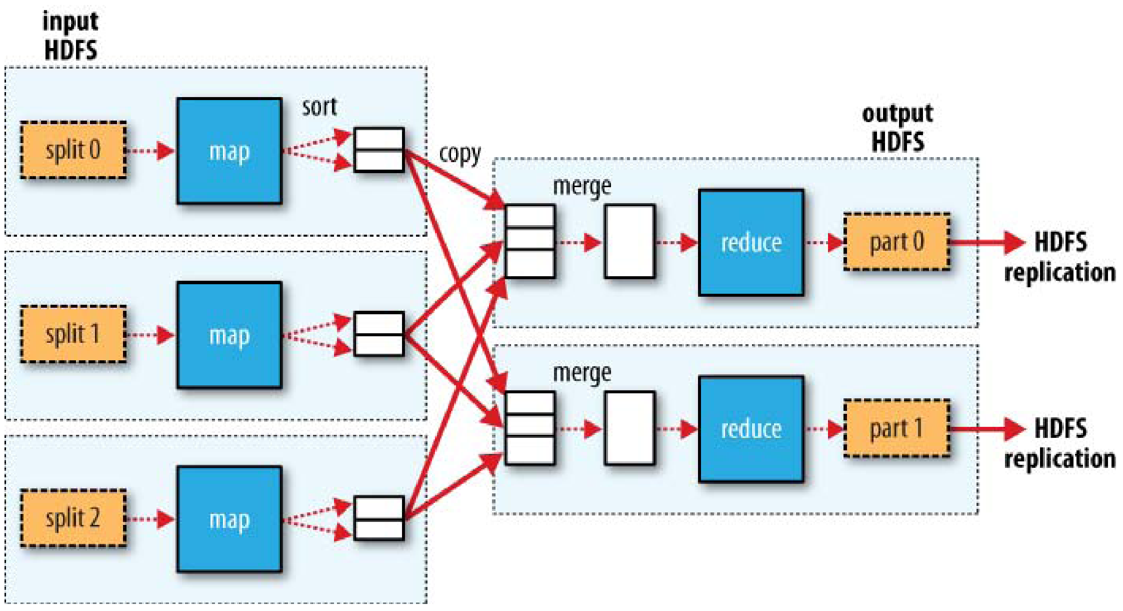


**Figure 2.6:** MapReduce data flow with multiple reducers - Image from [52, page 30]

#### 2.3.2.2 Apache Spark

Apache Spark was initially developed at the University of California, Berkley and is a cluster computing framework that provides primitives for fast on-memory data manipulation, which contrasts with Hadoop's MapReduce two stage disk based model. In some workloads Apache Spark as shown to be significantly faster than Hadoop's MapReduce [37].

At the heart of Spark's parallelism capabilities resides the notion of Resilient Distributed Datasets (RDD) which allow for fault-tolerant in-memory cluster computing [54]. RDD can be though of a database table that resides in memory, and can be operated on in parallel. They can hold any data type but are immutable, so any transformation on a RDD returns a new RDD [41].

There are two types of operations that can be done on a RDD, transformations and actions.

Some of the available transformations on RDD are *map* (returns a new RDD with a function applied to all elements of this RDD), *groupByKey* and *filter* (returns a new RDD with some elements of this RDD filtered by some criteria) between others. Some of the available actions are *count* (count the number of entries in this RDD), *first* (returns the first RDD element) and *collect* (returns RDD as a list object).

Spark also includes some useful libraries:

- *Spark Streaming* for processing real-time streaming data.

- *MLLib* a machine learning library, which includes recommendation systems implementation.
- *Spark SQL* which exposes Spark's datasets via Java Database Connectivity (JDBC) and lets the programmer query structured data inside Spark programs.
- *GraphX* which allows for graph representation and graph-parallel computation and includes some common algorithms.

Unlike Hadoop, Spark doesn't provide it's own distributed filesystem or security model so in a typical configuration it uses Hadoop's facilities like HDFS and YARN, a resource manager [14].

### 2.3.3 Distributed Databases

A distributed database is a type of database where the storage units are not directly connected to a single central machine but scattered on multiple machines that communicate over a network. These distributed machines can share the same physical location or not.

Depending on design a distributed database can achieve better performance and fault-tolerance than a single database system.

#### 2.3.3.1 Apache HBase

Apache HBase is a non-relational distributed database based on Google's BigTable [11], and has been adopted by the Apache Software Foundation. It runs on top of Hadoop's HDFS and provides fault-tolerant storing, querying and retrieval of data, it also uses Hadoop's MapReduce to process jobs in parallel.

HBase architecture was designed from the ground up to be distributed and linearly scalable by simply adding more nodes to the database cluster, and as Hadoop, it was designed to run on commodity hardware. HBase is specialized in efficiently querying huge amounts of sparse data, an instance where traditional Relational Database Management System (RDBMS) show weaknesses or in some cases don't work at all [32].

Data is stored in rows (identified by a primary key) within tables with labelled columns, columns are called column families and must be stated on the table schema during the table creation, then, each column family can cell can have multiple values, added during routine data insertion, the different values are identified by a column family qualifiers. General table architecture:

**Table 2.8:** HBase table architecture

| | **Column Family** | | **Column Family** |
|---|---|---|---|
| | **Column Family:Qualifier** | **Column Family:Qualifier** | |
| **Row Key** | Value | Value | Value |

Take for example a geographic information database, on a row with primary key 'Porto', we could have in column family 'Population' two values identified by two different qualifiers, for example '2011' and '2012', these two values would be identified by a string with the column family name as prefix, followed by a colon and with the qualifier as suffix.

Taking the above architecture image, a visual example is:

**Table 2.9:** Demographic HBase table example

| | **Population** | | **Number of Neighbouring Municipalities** |
|---|---|---|---|
| | **Population:2011** | **Population:2012** | |
| **Porto** | 237,591 | 230,298 | 4 |

Some other interesting features of HBase are:

- Every field in HBase is a byte array (except column family names), so there is extreme flexibility in what can be stored, from strings to ints, to longs and even serialized data structures.
- Naturally as a non-relational database, HBase does not support the common Structured Query Language (SQL) language but it does provide a shell, and shell commands can be scripted using Ruby [21].
- Rows are stored in sorted order and this sorting is made by the primary key byte value.

### 2.3.4 Web Services

A Web Service is a always-on, software function intended for machine-to-machine interaction, provided at a specific address via the Web [20]. A web service is typically described via the Extensible Markup Language (XML) based (Web Services Description Language (WSDL) a machine-readable file which specifies how to call the service and what data structures to expect to be returned. After accessing the WSDL file, the client calls the web service sending requests via Simple Object Access Protocol (SOAP) and after processing the request, the service responds again via SOAP. Some examples of common possible web services include a unit/currency converter, a weather reporting service or a Yellow Pages like service.

In today's World Wide Web (WWW) web services are ubiquitous.

#### 2.3.4.1 RESTful Web Service

Because of the complexity and performance overhead of traditional web services, much because of XML based protocols like SOAP, a back-to-basics movement surged in the early 2000's to simplify web services [16]. From this need, arised the definition of RESTful Web Service.

Representational State Transfer (REST) is the software architectural style of the WWW where actions, like those in Hypertext Transfer Protocol (HTTP) GET, POST, PUT, DELETE are applied to resources, for example GET /person/catarina/number.

A so called RESTful Web Service, adheres to the style represented by REST and also needs to meet some specific requirements like adhering to the Client-Server architecture, being stateless (no information about the client can be stored on the server) [49], cacheable (responses should identify themselves as cacheable when possible for performance improvement), be a layered system to improve scalability (the client should not be able to differentiate between the main server or a relay of it) and have a uniform interface. Contrary to SOAP based web services, RESTful web services don't have an official standard (because REST is an architectural style instead of a protcol like SOAP) but most web services use standards such as HTTP, Uniform Resource Identifier (URI), JavaScript Object Notation (JSON) and XML.

# Chapter 3

# Recommendation System Development

The first step during the development phase of the project was to compile some statistics from the dataset in use, and from there get a global picture of it's unique characteristics.

Certain characteristics like the number of unique products or unique customers could prevent some computation intensive recommendation techniques from being used.

For example, during this assessment, usage of association rules with the Apriori algorithm for next item recommendations generation was ruled out, because it's high memory requirements during the itemset creation step could not be accommodated with the available hardware.

## 3.1 Dataset

The dataset, provided by the project partner, consists of 6 months of real transaction data of the 5000 best customers (the ones with the higher number of transactions).

The period of the dataset is the months of December 2014 to June 2015, and apart from transaction data it also included, in a separate database table, product catalog information.

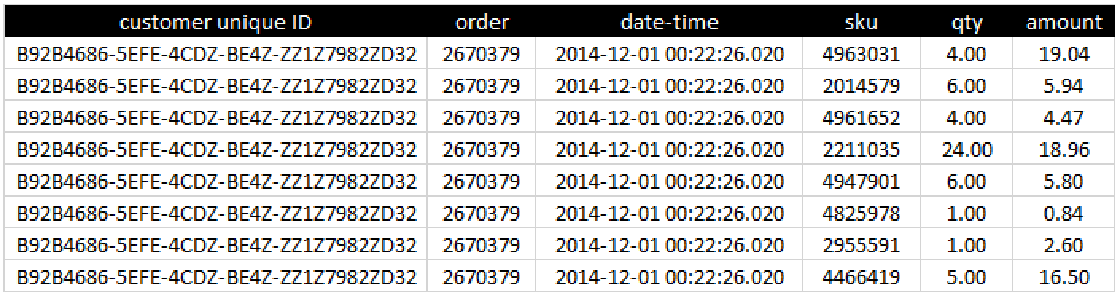
Some compiled dataset statistics are:

- 5,000 Customers
- 2,843 Product Categories
- 30,027 Products
- 73,030 Transactions
- Average of 14.61 Transactions/Customer
- Average of 25.27 Items/Transaction

- Average of 10.56 Products/Category
- Average of 206 Unique Bought Products/Customer

The dataset included two tables, one with the transactions and the other with catalog information (product categories and descriptions).

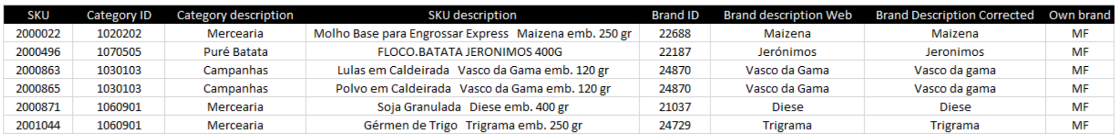
In the transactions table, each transaction was described by one or more lines, one line corresponding to each product bought. In figure 3.1 an example transaction taken from the dataset, in this case with 8 products, is presented.



| customer unique ID | order | date-time | sku | qty | amount |
|---|---|---|---|---|---|
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 4963031 | 4.00 | 19.04 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 2014579 | 6.00 | 5.94 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 4961652 | 4.00 | 4.47 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 2211035 | 24.00 | 18.96 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 4947901 | 6.00 | 5.80 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 4825978 | 1.00 | 0.84 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 2955591 | 1.00 | 2.60 |
| B92B4686-5EFE-4CDZ-BE4Z-ZZ1Z7982ZD32 | 2670379 | 2014-12-01 00:22:26.020 | 4466419 | 5.00 | 16.50 |

**Figure 3.1:** Extract of an entire transaction

The second table which stores information about the product range, has a line for each product (identified by its Stock Keeping Unit (SKU)) and typical information includes the product name and category information.



| SKU | Category ID | Category description | SKU description | Brand ID | Brand description Web | Brand Description Corrected | Own brand |
|---|---|---|---|---|---|---|---|
| 2000022 | 1020202 | Mercearia | Molho Base para Engrossar Express Maizena emb. 250 gr | 22688 | Maizena | Maizena | MF |
| 2000496 | 1070505 | Puré Batata | FLOCO.BATATA JERONIMOS 400G | 22187 | Jerónimos | Jeronimos | MF |
| 2000863 | 1030103 | Campanhas | Lulas em Caldeirada Vasco da Gama emb. 120 gr | 24870 | Vasco da Gama | Vasco da gama | MF |
| 2000865 | 1030103 | Campanhas | Polvo em Caldeirada Vasco da Gama emb. 120 gr | 24870 | Vasco da Gama | Vasco da gama | MF |
| 2000871 | 1060901 | Mercearia | Soja Granulada Diese emb. 400 gr | 21037 | Diese | Diese | MF |
| 2001044 | 1060901 | Mercearia | Gérmen de Trigo Trigrama emb. 250 gr | 24729 | Trigrama | Trigrama | MF |

**Figure 3.2:** Product range database sample

### 3.1.1 Database preparation

The next step was to import the two tables into an HBase server, the database was received in Excel's .xlsx format from a database dump, the data was then exported to CSV format files and a Java application was written to take care of inserting the data into HBase.

This developed application named *pt.fraunhofer.recommender.spark* uses the library *hbase-client* [25] to interact with the HBase server using a client Application Programming Interface (API).

The program first creates three tables, one for holding the actual transactions, another to hold the category data and a third one that maps the customer ID hash string present in the dataset into an integer ID that is suitable for internal Apache Spark usage.

The first created table holds the transactions, one transaction per row and its structure is presented in table 3.1.

**Table 3.1:** Transactions table

| | customer ID | dateTime | order | num Of Products | sku | | quantity | | amount | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Qualifier* | | | | | 1 | 2 | 1 | 2 | 1 | 2 |
| e1fe9f... | 1 | 2014-12-01 00:00:26.540 | 2790485 | 2 | 30293 | | 2.0 | | 18.73 | |
| | | | | | | 22569 | | 1.0 | | 0.99 |

(The row key is a SHA-1 hash of the string
2790485_2014-12-01 00:00:26.540 as in order_dateTime)

Please note that for each row of the table, the column families: sku, quantity and amount, each have column qualifiers from the range [1..numOfProducts] in order to store the different items bought in the transaction, as well as the quantity and the sub-total in Euros of that product.

The category table structure is presented in table 3.2.

**Table 3.2:** Categories table

| | name | brandID | brandName | brandCategory | categoryID | categoryName |
|---|---|---|---|---|---|---|
| 2000022 | Molho Base para Engrossar Express Maizena emb. 250 gr | 22688 | Maizena | MF | 1020202 | Mercearia |

(The row key is the product's SKU)

The map table structure is presented in table 3.3.

**Table 3.3:** Map table

| | customerID |
|---|---|
| 1 | ZF48EEZ5-FB8F-44ZB-99B2-4E21Z783414E |

(The row key is the customer id has an integer)

## 3.2 Product Recommendations

The most important design choice when building a product recommendation system is the algorithm that powers the recommendations.

From the state of the art review it was decided that a collaborative filtering matrix factorization model (reviewed in 2.1.2.3) would bring the best recommendation results. This is due to the fact that each user is only familiar with a small portion of the total 30,027 products (in fact less than 0.7%, see 3.1) and from there follows that the user-to-item affinity matrix is very sparse.

Any neighbourhood-based model would have difficulties in finding users with approximate tastes due to the sparseness of the affinity matrix.

As it stands, Apache Spark's Machine Learning Library (MLLib) already comes with an implementation of a collaborative filtering matrix factorization model with optimization through ALS. Using this library allowed for a fast implementation of the recommender system with the advantage that since Spark is open-source software, if need be, small changes could be made for it to meet specific needs.

As Apache Spark is a parallel computing framework, this matrix factorization model is already parallel computing capable.

The followed approach was to prepare the input data for the model according to our needs, train and test various models with different parameters to find the parameters that allowed for the best recommendation performance and from there generate the product recommendations.

A matrix factorization model predicts a rating for each product not yet known to a customer, and doesn't generate a recommendation directly. So, after predicting the rating for the desired products, to generate a recommendation, the products with the highest predicted rating for that customer are recommended.

### 3.2.1 Model Building

The first step in building the model was to create a user-to-item affinity matrix, this matrix was then stored in an HBase table for future usage.

This type of affinity matrix has a row for each user and a column for each item. Since our analysis is solely based on the customer's purchase history, if a customer has never purchased a particular product there is no available information about the customer's affinity to that product.

In the used dataset, with as much as 30,027 SKUs any customer has only purchased a small portion of the entire product catalog, what this means is that in practice this matrix is very, very sparse. In this case the maximum matrix size is $30,027SKUs * 5000users = 150,135,000$ but in reality from the purchase history only $1,016,732$ affinity values can be deduced, obviously with so much empty cells, this matrix was stored as sparse .

Two methodologies were tested when building this user-to-item matrix, the first was with boolean values *TRUE* (1) if the customer had bought at least one time that particular product or *FALSE* (0) if not, as the matrix is stored sparsely only the TRUE values are actually stored.

The second strategy was to calculate the following ratio on a continuous scale from 0 to 1:

$$Affinity(c, i) = \frac{NumBuys(c, i)}{TotalNumBuys(c)}$$

This affinity is the ratio of purchases by the client $c$ that included the item $i$, it can be 1 if in all of his purchases customer c has always bought product i.

The next step was to train the actual models in Apache Spark using MLLib, for this, all the affinity values needed to be loaded into Spark's internal data structures. Spark provides a Plain Old Java Object (POJO) to store this affinity value *Rating* which underneath is a Scala tuple of the form (user, product, rating). These *Rating* objects were then stored in Spark's RDD structures.

The parameters to train the MatrixFactorizationModel are *rank*, *numUserBlocks* and *numItemBlocks* [5].

A total of twelve different models were trained and validated. The RDD with rating data was separated into train and validation RDDs on 80% / 20% percentages and a matrix factorization model was trained for each combination of the training parameters:

**Code 3.1:** Training MatrixFactorizationModel

```
MatrixFactorizationModel model = ALS.train(JavaRDD.toRDD(training),
    rank, numIter, lambda);
double currRMSE = computeRMSE(model, validation);
```

For each actual rating present in the validation dataset, an approximated rating was generated by the model and the two were then compared and the Root Mean Squared Error(RMSE) was calculated, the actual RMSE results varied with the parameters used but were between 0.07 and 0.14 with ratings on a scale of 0-1.

The model which achieved the lowest aggregated RMSE was chosen.

### 3.2.2 Generating Recommendations

With a MatrixFactorizationModel created, generating recommendations is fairly simple as demonstrated on code 3.2.

**Code 3.2:** Generating Ratings

```
int user = 1;
int sku = 2000022;
double predictedRating = model.predict(user, sku);
```

The predicted rating was then pre-computed for all user/product pairs and then stored with indexation by product category in an HBase table.

By doing this all the recommendations are generated offline, to then give a recommendation to a user, the system gets the recommendations from a certain product category from HBase and ranks them by descending predicted rating, then the top-k recommendations are given to the user.

This solution provides great online recommendation performance, recommendations are just a database query away, but is very taxing when the recommendations are first being generated for every user and every product category $5000\ \textit{users}\ \times 2843\ \textit{product categories} = 14,215,000$ different recommendations.

Another alternative would be to generate the recommendations online, i.e. when the system is requested $K$ number of recommendation from a certain category, it calls the MatrixFactorizationModel to predict the rating of all the products of that category and returns the top-K ranked ones.

This may not be viable due to the time taken to predict these ratings.

### 3.2.3 Own Brand Products

Nowadays, own brand products (also named white label products) are less and less of a novelty and are becoming an established alternative in the groceries market, with their market share in each product category increasing year over year. Our project partner wanted to have a solution where the customers affinity for only own brand products was predicted and then the most liked products were recommended to the customers.

Own brand products normally provide a lower cost alternative to brand products but product quality has a high variation.

In order to generate recommendations of only own brand products, these products need to be filtered from the rest of the product catalog, this filtering can be made in two stages, the products can be filtered before predicting the rating with the *MatrixFactorizationModel* or they can be filtered when retrieving the generated recommendations from HBase. The first solution was chosen.

To filter only own brand products it is as simple as only choosing the products with the column *brandCategory*, as in table 3.2 , with values either Marca Própria(MP) or Primeiro Preço(PP).

#### 3.2.3.1 Shopping cart replacement

A feature that was envisioned as being able to demonstrate the power of recommendations was that of shopping cart replacement.

Starting from a complete shopping cart, each product is replaced (where an alternative exists) with the alternative highest rated own brand product of the same category. In the end a, hopefully cheaper, replacement cart is presented to the costumer.

A use case for this feature, would be that at checkout the customer would be recommended alternative products, that he/she could accept for added savings.

The major challenged in producing this demonstration feature is that certain product categories encapsulate different products that are not a suitable replacement to each other, take for example cookies, there exist a myriad of different chocolate cookies and in the used dataset very different chocolate cookies can belong to the same product category. Two other problems are those of product quantity, for example a juices product category has products that come in 1x1L or 3x200mL packages, these are not suitable replacements for each other, and the same is true for other product characteristics like flavour or aroma, a clothes detergent with an aroma of oceanic breeze can be in the same product category as a detergent with orange aroma.

These challenges require major work to overcome and don't belong on the context of this thesis so they were not tackled and are proposed as future work.

Even then a working interface was produced that provides suitable alternatives in most of the cases.

### 3.2.4 Association Rules

Since association rules are still today used in some areas of recommendation systems [53], [45], it was decided to test how well they would fare in the used dataset.

For this, the first step was to implement the Apriori algorithm in Java, this proved to be quite a difficult task because of the dataset size. Particularly, due to the high number of products, at each iteration, the number of candidate itemsets tends to grow a lot and fill all the available heap space of the Java Virtual Machine(JVM).

Taking into account the statistics presented in 3.1 the first run of the algorithm will start with 30,027 1-itemset candidates and with the same statistics it is possible to calculate that there are approximately ${}^{30{,}027}C_2 \approx 4.5 * 10^8$ possible unique 2-itemsets, assuming the optimum case where each item can be stored using 15 bits ($2^{15} - 1 > 30,027$), a 2-itemset would occupy 30 bits, and so, all the possible 2-itemsets would take $4.5 * 10^8 * 30 bits \approx 1.7$ gigabytes, assuming that no itemsets are dropped due to lack of support, the next phase (3-itemsets) would need 25 terabytes of memory.

Even thought candidate itemsets will be dropped due to lack of support and consequently won't pass to the next iteration, the presented memory consumption is the lowest possible, a typical implementation in Java would probably require 30% to 40% more memory than the numbers given above.

For this reason, no further advances could be made with association rules.

## 3.3 Periodically Bought Products

Periodically bought products for a certain user can be described as those products which the customer buys on a regular basis and within a well defined period between successive buys.

The number of days between each successive buy of the same product by a customer is called the buying period. The periods are assumed to be discrete, so, even if a period accounts for $N$ days plus some hours, it is rounded to the nearest integer number of days $N$.

**Table 3.4:** Example of period calculation

| Buying Dates | Period |
|---|---|
| 2 April, 2015 | |
| 12 April, 2015 | 10 days |
| 22 April, 2015 | 10 days |
| 2 May, 2015 | 10 days |
| 6 May, 2015 | 4 days |

Due to different tastes, socio-economic conditions, between others factors, a product can be periodic for a certain user and not for another, also it can be assumed that globally some products are periodic (bread, detergent) while others certainly are not (cutlery, tableware).

### 3.3.1 Motivation and Objectives

Imagine that the recurrently bought common wear products of a particular customer could be recommended to him/her at the right time, and he/she could easily add them to the shopping cart, this feature tries to achieve just that, and this could save the customer a lot of time and sometimes even frustration (who hasn't arrived home to only there realize that he/she forgot to buy X, Y product at the store?).

The objective of this feature is to obtain a list of periodically bought products for each customer as well as finding a possible future buying date. The use-case for this service could be the following, a customer opens a mobile application/website, and before starting his shopping experience, he is presented with his periodically bought products that he might be interested in buying in the near future and from right there he can easily add them to this shopping cart.

The first step in developing this feature was to analyze the periodicity of the buys of certain key products, those which were suspected to bought periodically and trying to find some insights from there.

### 3.3.2 Analytical Tools

Analytical tools allow through visualization for a better understanding of the data at hand, particularly like in this case when the dataset is very large, that could not be achieved otherwise.

In the frequently bought products case some important properties to be analyzed included the product quantities bought in each sale and the period of days between successive buys of the same product.

To aid in this visualization, some charts were constructed as part of the general GUI using the Javascript libraries Chart.js [28] and D3.js [29].

A web page was developed which for every customer bought products allowed the visualization of the buying dates and respective amount of product bought at each of the dates, visualization of the buying periods and the normalized buying periods that will be explained latter.

The first chart draws the quantity of each product that a customer has bought during the dataset time frame. This chart has on the x-axis each buy (identified by the buying date) that the customer has made of the product, and on the y-axis the quantity of product that was bought on each of these buys. An example of this chart is presented in figure 3.3.

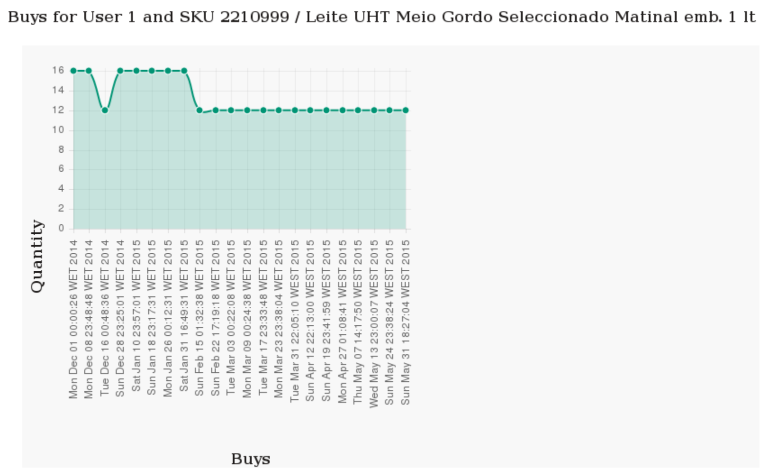


**Figure 3.3:** Buying Quantities Visualization

The chart in figure 3.3 serves as an indication on the regularity of the quantities bought by the customer. A chart with more spikes indicates lower regularity, a straight line indicates perfect regularity.

The second chart shows the variation of periods between consecutive buys of the same product by the customer. On the x-axis each point indicates a period serving the number as a merely temporal indication (e.g. period 0 encapsulates the period in days between buys 1 and 2, and period 10 encapsulates the period in days between buys 11 and 12), the y-axis represents the number of days in each period.

Since the periods are the number of days between successive buys, the number of periods (and hence data points in this chart) is $numBuys - 1$. An example of this chart is presented in figure 3.4.

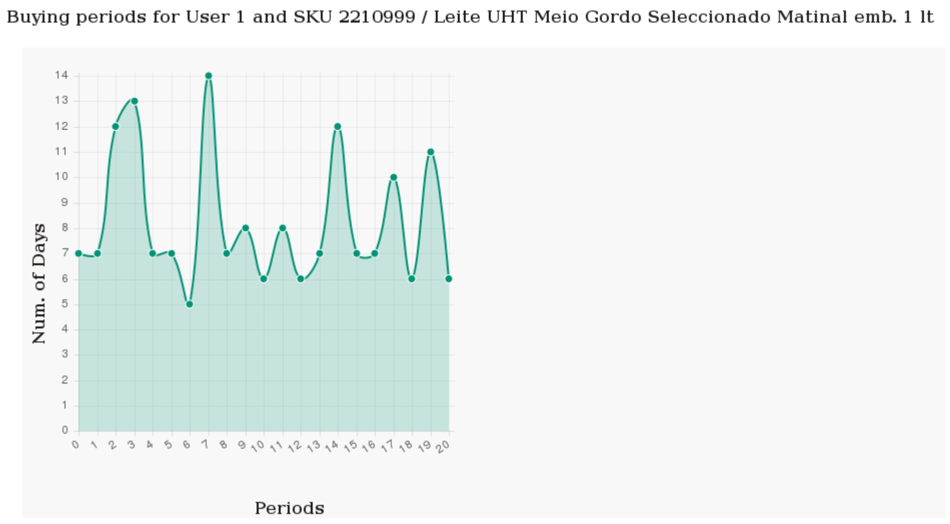


**Figure 3.4:** Buying Periods Visualization

Visualizing the previous two charts, it becomes clear that the period between buys depends on the quantity of product that is bought on the first visit of the period.

A third chart was produced which combines the results of the previous two charts to give a normalized view of the buying periods, this normalization actually gives the product consumption ratio with the formula:

$$\text{Product Consumption Ratio} = \frac{\text{Bought Quantity}}{\text{Period}}$$

This chart has on the x-axis the periods identified by their number (as in the previous chart) and on the y-axis the quantity bought in the first visit of the period divided by the number of days in the period. An example of this chart is presented on figure 3.5.

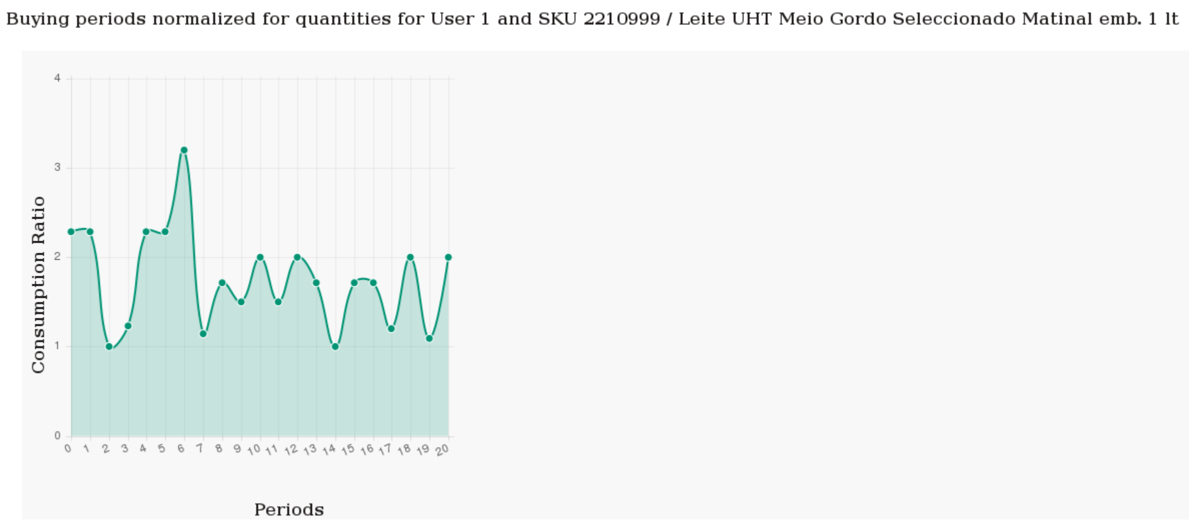


**Figure 3.5:** Normalized Periods Visualization

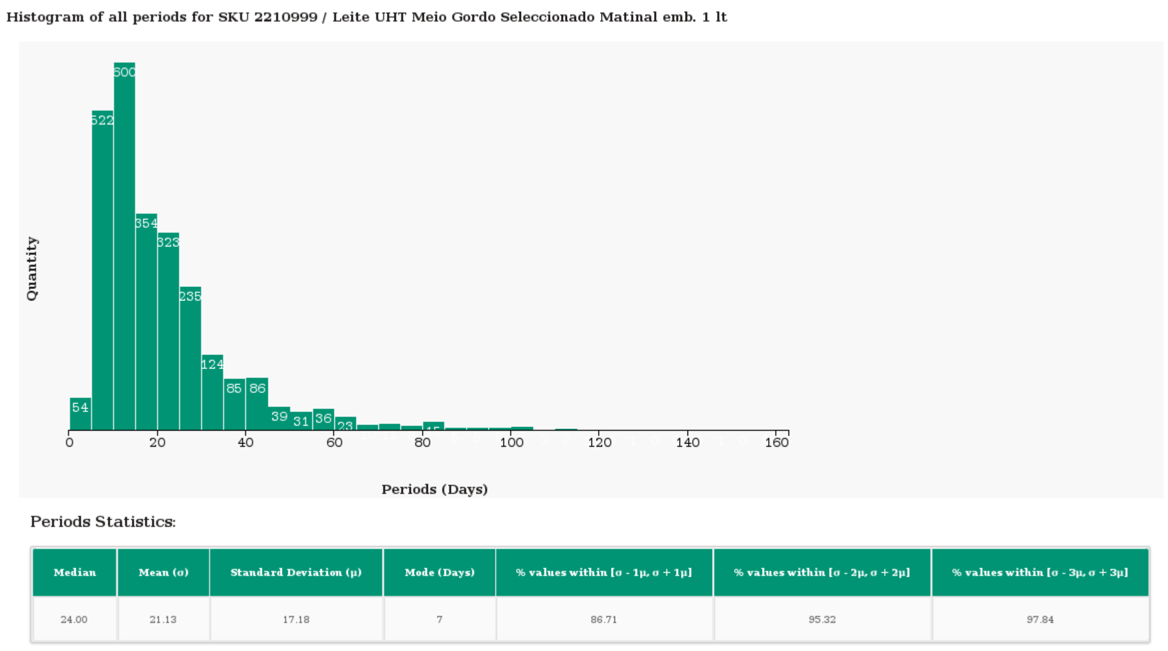
Analyzing the chart present in figure 3.5, one can assess that in this case, during the period of the first to the second buy, the product consumption ratio was approximately 2.3 cartons of milk per day.

### 3.3.3 Concept

Getting back to the initial proposition that products can be periodic for certain users while not being for others, the first step to finding periodically bought products was to analyze all the periods between buys for all products in order to assess if globally the product could be considered to be bought periodically.

For this global analysis the distribution function of all the periods was drawn via an histogram and the periodically bought products were expected to have a high concentration of values in a few buckets.

All the buying periods for a certain product were retrieved from HBase and collected to draw the histogram:


Histogram of all periods for SKU 2210999 / Leite UHT Meio Gordo Seleccionado Matinal emb. 1 lt

Periods Statistics:

| Median | Mean (σ) | Standard Deviation (μ) | Mode (Days) | % values within [σ - 1μ, σ + 1μ] | % values within [σ - 2μ, σ + 2μ] | % values within [σ - 3μ, σ + 3μ] |
|---|---|---|---|---|---|---|
| 24.00 | 21.13 | 17.18 | 7 | 86.71 | 95.32 | 97.84 |

**Figure 3.6:** Example of a global periods graphic with 40 bins

Through evaluation of dozens of distributions of several products' periods it was discovered that the distribution tends to follow the pattern present in the above chart 3.6 . Another characteristic is that the distribution of this variable tends to be positively skewed (right-skewed/right-tailed), this means that the mass of the distribution is to the left of the mean.

The periods' distribution was tested for normality for all the products using the previously referenced *Pearson Chi-Square Normality Test* (see 2.2.2.3) but it was found that with exception to products with very few periods (products with only 2 periods) the distribution did not follow a normal distribution.

### 3.3.4 Solution

Taking into account the observed line of the distribution function of the periods between buys for several products, periodically bought products were defined as those with a big mass of the distribution around the mean.

This means finding products which have lots of periods in and around the average of all the periods.

The logic for this definition is that if the mass of the distribution function of the periods around the mean is very high it follows that these products are bought within a well defined interval.

The *Three Sigma Rule* [40] is an empirically observed rule that states that for a normal distribution, assuming that $\mu$ is the distribution mean and $\sigma$ is the distribution standard deviation, any value $x$, sampled from that distribution, follows the following rules:

- $P(\mu - \sigma \ \leq x \leq \mu + \sigma) \approx 0.6827$
- $P(\mu - 2\sigma \ \leq x \leq \mu + 2\sigma) \approx 0.9545$
- $P(\mu - 3\sigma \ \leq x \leq \mu + 3\sigma) \approx 0.9973$

The *Three-sigma rule of thumb* says that even for non-normal distributions 98% of the values should be within $[\mu - 3\sigma, \mu + 3\sigma]$ [40].

Even though it was observed that most product periods distributions did not follow a normal distribution, we defined a product as periodic if the observed value of the mass around $\mu \pm \sigma$ was above a certain threshold (instead of it being above 68.27% if it was tje case that it followed a normal distribution).

### 3.3.5 Generating Frequent Products

A web service was developed that when invoked with a customer ID and a set of products to analyze, returns the products from the set (given as an argument) that are periodic as well as the expected next buying interval start and end dates. Note that this service could hypothetically be called with a set composed of all the products in the catalog for it to analyze all the products, and return only the relevant ones to the given customer.

The web service implementation relies on the previously mentioned Jersey [30] library.

The implementation receives a request passed as an object of the class *FrequentProductRequest* encoded as a JSON object, this request includes the customer ID, the products tp analyze and the desired threshold.

The service then for each product retrieves the periods mean $\mu$, standard deviation $\sigma$ and the number of periods that fall within $\mu \pm \sigma$ for the customer ID purchases. If the percentage of periods that fall within $\mu \pm \sigma$ is above the threshold present in the request, then, the product is considered to be purchase periodically, the next purchase interval is calculated and it is added to the result list to be returned.

**Code 3.3:** Calculating the number of periods within the specified interval

```
periods = (double[]) oisPeriods.readObject();
int count = 0;
double minValue = periodsMean - periodsStdDev;
double maxValue = periodsMean + periodsStdDev;

for(double period : periods) {
   if(period >= minValue && period <= maxValue)
```

```
            count++;
        }
        userPercentage = ((double)count/(double)periods.length)*100;
```

To estimate the next purchase interval for the customer, the last buy date of the product is retrieved from HBase and the next purchase interval is estimated as follows:

$$[(lastBuyDate + \mu \text{ days}) - \sigma \text{ days}, (lastBuyDate + \mu \text{ days}) + \sigma \text{ days}]$$

Another option to estimate the next buying interval would be to use the last buy date and the mode instead of the mean:

$$[(lastBuyDate + mode \text{ days}) - \sigma \text{ days}, (lastBuyDate + mode \text{ days}) + \sigma \text{ days}]$$

### 3.3.6 Web Services

Several web services were needed during the development of this solution and thus were developed as part of the main server program, the services include the following:

| Path | HTTP Method | Receives | Description |
|---|---|---|---|
| getUserBoughtSKUs | GET | userID, minNumBuys | Service returns a list of products bought at least minNumBuys times by user identified by userID |
| getUserFrequentProducts | POST | FrequentProductRequest | Service scans the buys of the products contained in FrequentProductRequest by user also contained in FrequentProductRequest and returns those found to be bought periodically as a list of FrequentProducts |
| getAllSKUs | GET | - | Service returns a JSONArray with a JSONObject with name and sku properties for each product in the catalog |

| Path | HTTP Method | Receives | Description |
|---|---|---|---|
| getGlobalProductStatistics | POST | JSON Array | Service returns a list with a ProductStatistic object with several product statistics for each product included in the request as an object |

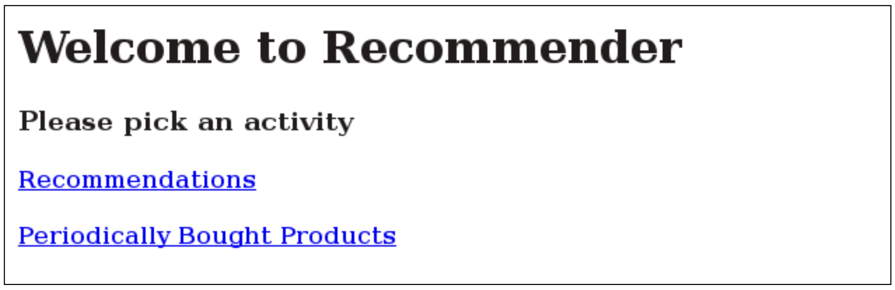
## 3.4 Graphical User Interface

During the project development a GUI was developed in order to aid in data analysis, given the size of the dataset, and for solution demonstration purposes.

All development was made using Web technologies. The main web pages were developed using JavaServer Pages (JSP) which is a technology that all ows for dynamic web pages creation using a mix of Java language and HTML code in the same file, Javascript was also used for web service calls through jQuery [34] and chart drawing with the libraries Chart.js [28] and D3.js [29].

All the code ran on Apache Tomcat [27] which is a web server and servlet container, a Java servlet is a Java program that allows for the implementation of a request-response methodology via extending the server capabilities.

The index web page has the following look:


Welcome to Recommender

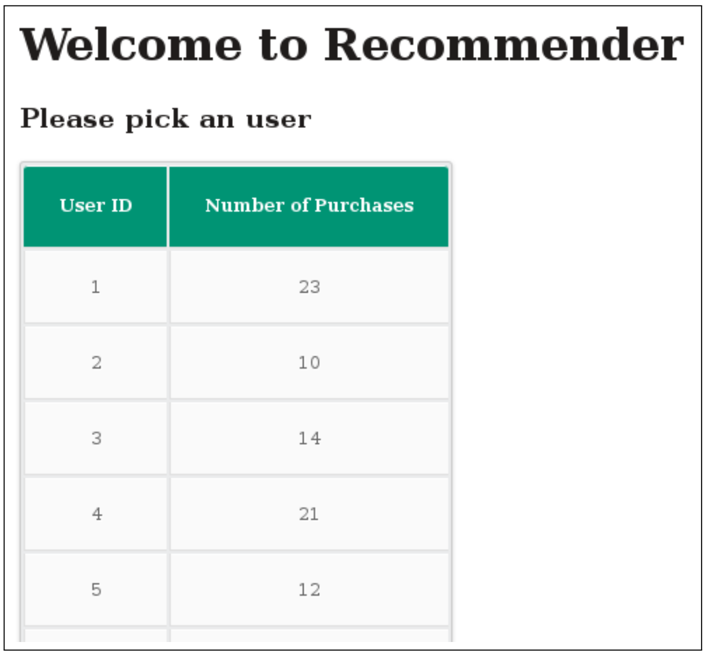
Please pick an activity

Recommendations

Periodically Bought Products

**Figure 3.7:** Index

Choosing Recommendations leads to the next page, with various customers identified by respective ID and number of transactions (an entire purchase made in the online store):



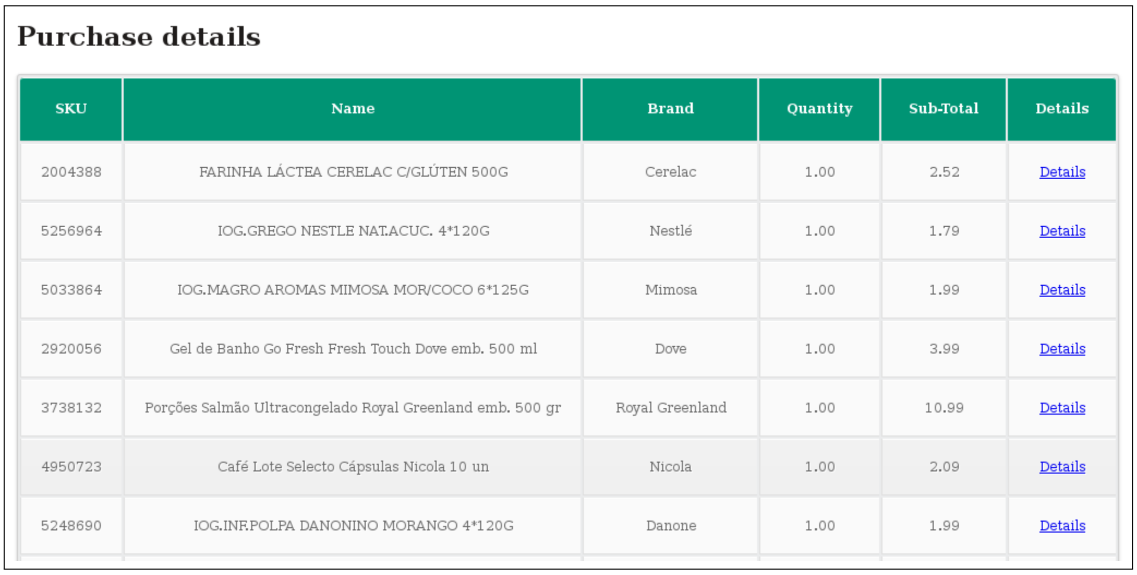
Welcome to Recommender

Please pick an user

| User ID | Number of Purchases |
|---|---|
| 1 | 23 |
| 2 | 10 |
| 3 | 14 |
| 4 | 21 |
| 5 | 12 |

**Figure 3.8:** Recommendations index with users listed

Clicking on a row leads to the next web page with details about each transaction, one row for each product:

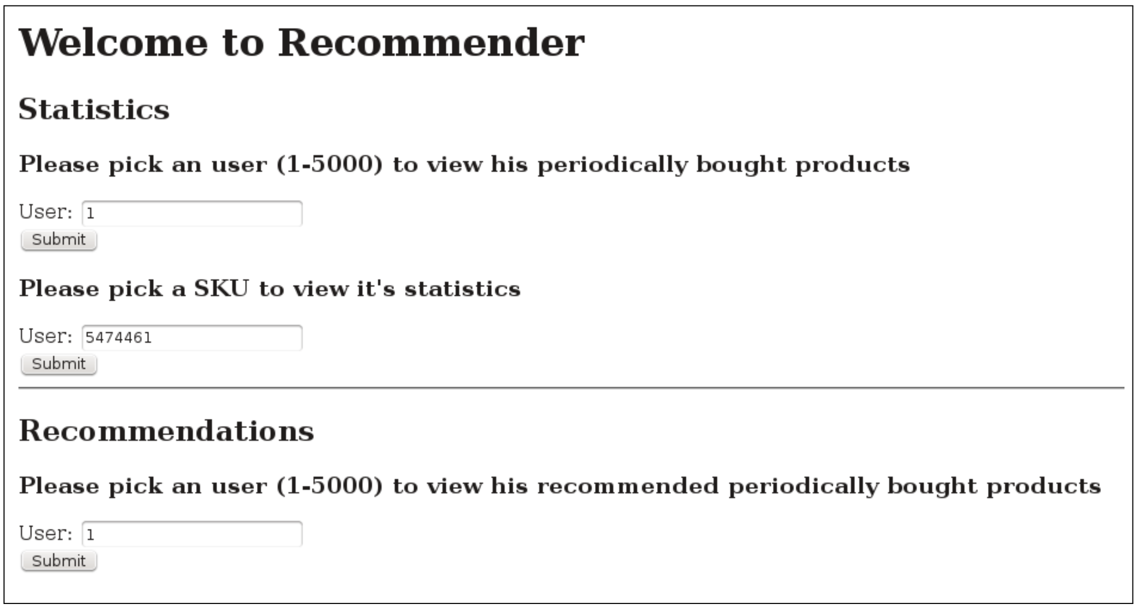


Purchase details

| SKU | Name | Brand | Quantity | Sub-Total | Details |
|---|---|---|---|---|---|
| 2004388 | FARINHA LÁCTEA CERELAC C/GLÚTEN 500G | Cerelac | 1.00 | 2.52 | Details |
| 5256964 | IOG.GREGO NESTLE NAT.ACUC. 4*120G | Nestlé | 1.00 | 1.79 | Details |
| 5033864 | IOG.MAGRO AROMAS MIMOSA MOR/COCO 6*125G | Mimosa | 1.00 | 1.99 | Details |
| 2920056 | Gel de Banho Go Fresh Fresh Touch Dove emb. 500 ml | Dove | 1.00 | 3.99 | Details |
| 3738132 | Porções Salmão Ultracongelado Royal Greenland emb. 500 gr | Royal Greenland | 1.00 | 10.99 | Details |
| 4950723 | Café Lote Selecto Cápsulas Nicola 10 un | Nicola | 1.00 | 2.09 | Details |
| 5248690 | IOG.INF.POLPA DANONINO MORANGO 4*120G | Danone | 1.00 | 1.99 | Details |

**Figure 3.9:** Transaction details example with one bought product per row

If in the index page, one chooses to click on Frequency it is lead to the following page:


Welcome to Recommender

Statistics

Please pick an user (1-5000) to view his periodically bought products

User: 1

Submit

Please pick a SKU to view it's statistics

User: 5474461

Submit

Recommendations

Please pick an user (1-5000) to view his recommended periodically bought products

User: 1

Submit

**Figure 3.10:** Frequency index

Apart from the pages shown in this section, the rest of the interface was shown on the previous sections, a flowchart of the entire GUI is:

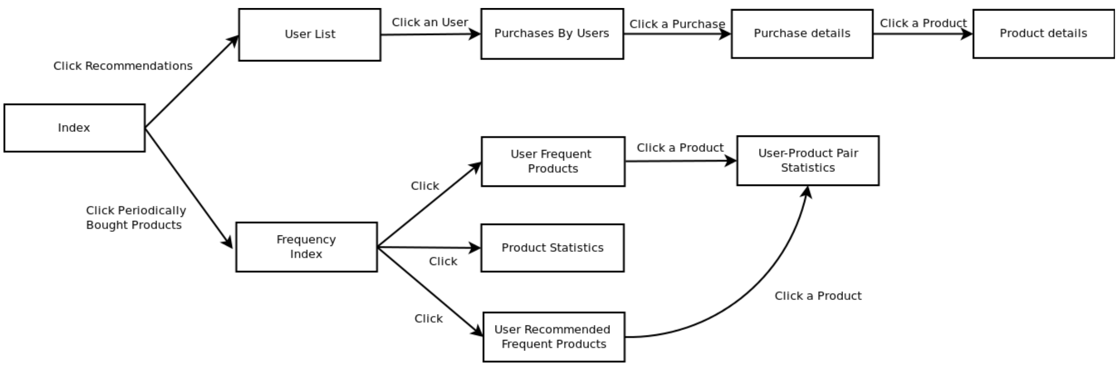


**Figure 3.11:** GUI Flowchart

# Chapter 4

# Tests and Results

To find the validity of the proposed solutions, some tests were run in order to measure the obtained accuracy.

## 4.1 Product Recommendations

After much thought a conclusion was reached that without assessing how much recommendations were helpful to users with some type of survey, it would be very difficult to measure the true usefulness of the recommendations in the user's overall shopping experience.

This survey could be made with a questionnaire that the users would be asked to fill after using the system.

Another alternative would be to use A/B testing, i.e. use a control group, and measure the rate of acceptance of the recommendations as the users make their shopping. Or as alternative, also use A/B testing but measure if on average the group that was presented with recommendations spent more when compared to the control group (which would not be presented with recommendations).

As both of these tests would take more time than was available in the context of this internship, another route had to be taken. A way to measure the effectiveness of the recommendations would be to measure the rank percentile of the recommendations, this idea was taken from [31].

The recommendations average rank percentile can be calculated by averaging the rank percentile calculation for every user that is tested.

To test a user, one proceeds in the following way, the rating that the user would give to every product is predicted, and then the products are sorted by descending predicted rating. The average rank percentile for the user of the observed buys in the test dataset (these are the products actually bought by the customer) is then taken, accumulating the multiplication of each observed product buy rank percentile in the recommendations list by the predicted rating and diving in the end this result by the summation of all the predicted ratings used.

The average rank percentile formula can be more clearly presented using mathematical notation as:

$$\overline{rank} = \frac{\sum_{u,i} (r^t_{u,i} * rank_{u,i})}{\sum_{u,i} r^t_{u,i}}$$

The objective is to have the products bought at the top of the recommendation list, so as to lower their rank percentile, and thus the final result. So, by following the same logic, a lower final result is better than a higher one.

A random recommender (one that picks products at random) has an expected average rank percentile of 50%. This can serve has a simple baseline that the recommender has to beat.

The dataset was split between 4 months of training data and 2 months of test data. The model was then trained with various number of features (latent factors), and the average results of 3 runs, each with 10 users, are presented in table 4.1.

**Table 4.1:** Average rank recommendation results

| Number of Model Features | Average Recommendation Rank |
|---|---|
| 12 | 12.50% |
| 20 | 12.00% |
| 25 | 11.50% |

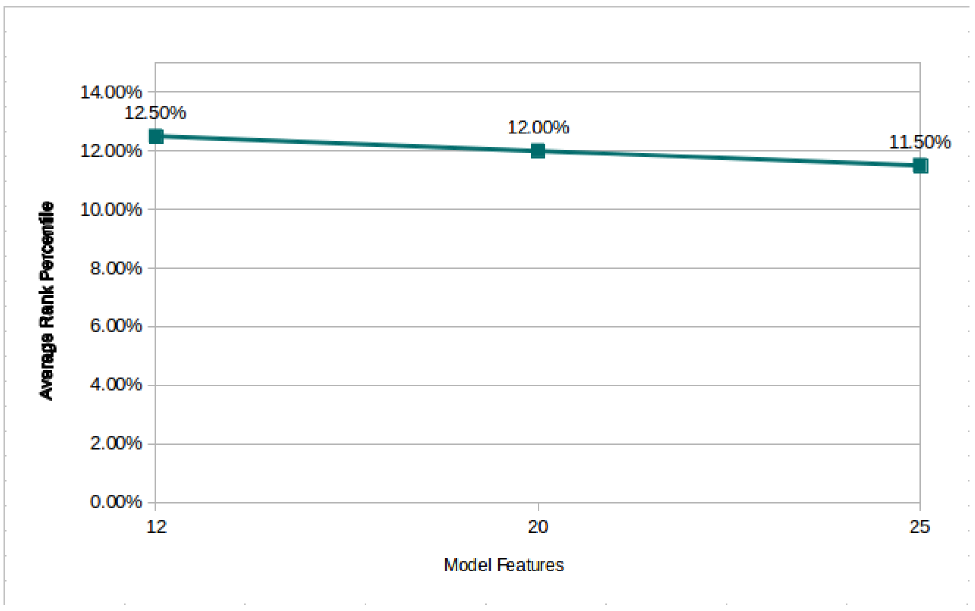


**Figure 4.1:** Rank percentile results

The results are encouraging, being almost on par with the results presented in [31]. It can be argued that if the models could be constructed with a higher number of features, the results would be even better.

The number of model features influences heavily the computation times (in model creation and recommendations calculation). Because of this models with a higher number of features could not be tested in a timely manner.

## 4.2 Periodically Bought Products

In the case of periodically bought products, a perfect solution would be able to pinpoint the next exact date of when a customer would buy each of his/hers periodically bought products. This would allow the recommendation system to, near that date, show the recommendation to the customer in order to facilitate the customer's buying experience while still adding overall value to the store by assuring that the customer is given the choice to purchase a product that he/she is very receptive to.

So, the most important metric is the deviation of the predicted date to the actual next buying date, if the deviation from the real date is too wide the system looses it's usefulness by not being able to recommend the products at the right time.

To test the usefulness of the predictions, a Java program was written that for $N$ randomly sampled users, requests their frequent products and respective next predicted buying dates and compares these dates with the real buying dates.

The tester program first retrieves for each customer the products bought at least three times, and then, for each of these products, finds if it's bought periodically and the next predicted buying date.

As previously said, the time frame of this internship didn't allow for a live test of the predictions/recommendations.

### 4.2.1 Results with 4 months of training and 2 months of testing data

The test methodology consisted of splitting the original dataset into four months of training data and two months of test data and then analyze the prediction accuracy by measuring the global deviation of the predicted next buying date and the observed date on the test part of the dataset. This splitting was chosen in order to have enough data for the predictor to work from but to also give it a fair chance of guessing the next buying date by having a large enough possible interval.

The first test was run with a threshold of 85% to consider a product as being bought frequently, what this means is that for the specific customer, 85% of his/hers buying periods for this product are within $mean \pm$ *standard deviation*.

The next predicted buying date was calculated by adding to the last recorded buying date, the mode of the periods, i.e. the most frequent buying period for that particular customer/product pair:

$$NextBuyDate = LastBuyDate + mode(customer, product)$$

With a dataset division of 4 months of training and 2 months of testing. The results were:

**Table 4.2:** 1st Formula results with 4months/2months, threshold = 85%

| Num. of Customers | Avg. Num. of Products | Average Deviation (Days) | % of exact hits |
|---|---|---|---|
| 200 | 4.4 | 27.99 | 1.88 |
| 500 | 4.4 | 28.05 | 2.59 |
| 1000 | 4.6 | 27.66 | 2.45 |

(Average of 5 runs for each number of customers)

The following test was run with another formula to predict the next buying date:

$$NextBuyDate = LastBuyDate + mean(customer, product)$$

The results were:

**Table 4.3:** 2nd Formula results with 4months/2months, threshold = 85%

| Num. of Customers | Avg. Num. of Products | Avg. Deviation (Days) | % of exact hits |
|---|---|---|---|
| 200 | 4.2 | 23.86 | 2.33 |
| 500 | 4.2 | 23.95 | 2.74 |
| 1000 | 4.8 | 23.70 | 2.31 |

(Average of 5 runs for each number of customers)

The next test was also run with a threshold of 85% but the next predicted buying date was calculated by adding to the last recorded buying date, the mean of the buying periods of that particular customer/product combo minus half of the standard deviation, i.e.:

$$NextBuyDate = LastBuyDate + mean(customer, product) - stdDev/2$$

Halving the standard deviation was somewhat of a naive choice, but the idea behind it was to encapsulate the deviations below and above the mean but not to introduce a number that would disrupt the calculated next buying date. The results are presented in table 4.4.

**Table 4.4:** 3rd Formula results with 4months/2months, threshold = 85%

| Num. of Customers | Avg. Num. of Products | Avg. Deviation (Days) | % of exact hits |
|---|---|---|---|
| 200 | 4.6 | 25.24 | 2.62 |
| 500 | 4.4 | 24.99 | 2.48 |
| 1000 | 4.4 | 24.80 | 2.58 |

(Average of 5 runs for each number of customers)

A third test was run where the required threshold was lowered to 60% and the predicted next buying date was also calculated using the previous formula. The results are presented in table 4.5.

**Table 4.5:** Periodically Bought Products Results, 4months/2months, threshold = 60%

| Num. of Customers | Avg. Num. of Products | Avg. Deviation (Days) | % of exact hits |
|---|---|---|---|
| 200 | 9.2 | 21.00 | 3.70 |

(Average of 5 runs for each number of customers)

### 4.2.2 Results with 5 months of training and 1 month of testing data

A final test was run with the required threshold still set at 60% and the same formula for next buying date prediction:

$$NextBuyDate = LastBuyDate + mean(customer, product) - stdDev/2$$

But this time with 5 months of training and 1 month of testing data. The results are presented in table 4.6.

**Table 4.6:** Periodically Bought Products Results, 5months/1month, threshold = 60%

| Num. of Customers | Avg. Num. of Products | Avg. Deviation (Days) | % of exact hits |
|---|---|---|---|
| 200 | 8.6 | 17.09 | 4.34 |
| 500 | 8.2 | 17.27 | 4.22 |
| 1000 | 8.4 | 17.07 | 4.56 |

(Average of 5 runs for each number of customers)

### 4.2.3 Results Discussion

Overall, the developed recommendation system shows promising results, based on the good percentage of exact next buying date hits, which is a metric that is very difficult to predict.

Two formulas obtain good results on next buying date predicting, being them:

$$NextBuyDate = LastBuyDate + mean(customer, product)$$

$$NextBuyDate = LastBuyDate + mean(customer, product) - stdDev/2$$

The second formula trades a higher average rate of exact hits (0.16%) for a lower overall accuracy, with an average rise of 1.2 days on the next buying date prediction.

Comparing the results on table 4.5, to the results on table 4.4, the average number of found periodically bought products doubles because of the lowered threshold required to consider a product as periodically bought.

The average deviation is lower and the percentage of exact hits rises, which are both good results, but it's conjectured that this happens because of the previously referenced doubling of the number of products and not because the products that fit on this lower threshold allow for a better next date prediction.

Because of the fact that some of the predicted next buying dates fall beyond the scope of the used dataset, thus making our data right censored, the fact that the system returns more recommended products leads to better prediction results simply because some products will always be discarded, in both cases, with this taking a higher impact on the case where less products are retrieved already.

But even with this assumption, these results warrant further testing to see if it holds true.

Compared to the results on table 4.5, the results on table 4.6 are very promising because it seems to show that the recommendation system behaves better with more training data.

Some trends observed in the test results warrant further testing, if possible with a larger dataset with more months of transactions, to see the actual effect on some parameters used in the prediction.

# Chapter 5

# Conclusion

This thesis details the work performed during the internship, from the state-of-the-art review, to the actual system development to the performed tests and results.

When the project ended the following objectives had been achieved:

- Product recommendations
- Product recommendations of only own brand/white label products
- Complete cart replacement with own brand/white label alternative products
- Product statistics and data visualization
- Periodically bought products discovery with next buying date prediction

Recommendations are a powerful tool for both the customer and the retailer, when correctly used they can help the customer discover new products that he/she will love but otherwise wouldn't have known about, from the retailer side, recommendations and their associated data can provide great insights into the inner workings of consumers taste, as well as helping the retailer focusing his sale effort on the products that really matter to the consumer. In our tests, the developed solution seems to provide meaningful recommendations but further testing is needed to correctly assess its effectiveness.

## 5.1 Future Work

Some things need to be tested more thoroughly:

- The time taken to generate recommendations on more powerful hardware
- Testing different metrics for user-to-item affinity

- Validate the recommendations results

As future work, apart from an A/B field test to validate the results, the following features could also be added:

- Recommender that learns the effectiveness of the recommendations, so as to not repeat unwanted recommendations
- When generating recommendations, also take into account the customer behavior in the online store instead of only focusing on the buying history.
- Employ data mining techniques like linear regression, ANNs, etc to more accurately predict the next buying date

# Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

[3] Rakesh Agrawal and Ramakrishnan Skikant. Fast Algorithms for Mining Association Rules. *Sdm*, pp:478–489, 2010.

[4] Fraunhofer AICOS. `http://www.fraunhofer.pt/en/fraunhofer_aicos/home.html/`, September 2015.

[5] ALS.train(). `https://spark.apache.org/docs/latest/api/java/org/apache/spark/ml/recommendation/ALS.html#train%28org.apache.spark.rdd.RDD,%20int,%20int,%20int,%20int,%20double,%20boolean,%20double,%20boolean,%20org.apache.spark.storage.StorageLevel,%20org.apache.spark.storage.StorageLevel,%20int,%20long,%20scala.reflect.ClassTag,%20scala.math.Ordering%29`, September 2015.

[6] Roberto J. Bayardo. Efficiently mining long patterns from databases. *ACM SIGMOD Record*, 27(2):85–93, 1998.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2012.

[8] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, 461(8):43–52, 1998.

[9] Erik Brynjolfsson, Yu Jeffrey Hu, and Michael D Smith. From Niches to Riches : Anatomy of the Long Tail From Niches to Riches : Anatomy of the Long Tail. *MIT Sloan Management Review*, 47:67, 2006.

[10] Robin Burke. Hybrid Recommender Systems: Survey and Experiments. *The adaptive web*, pages 377–408, 2007.

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah a. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable. *ACM Transactions on Computer Systems*, 26(2):1–26, 2008.

[12] Yung-hsin Chen and Edward I George. A bayesian model for collaborative filtering. *Direct*, (1), 1999.

[13] Pearson chi-square normality test. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm`, September 2015.

[14] Is Hadoop dead and is it time to move to Spark? `https://www.quora.com/Is-Hadoop-dead-and-is-it-time-to-move-to-Spark/answer/Sean-Owen`, September 2015.

[15] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplied Data Processing on Large Clusters. *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pages 137–149, 2004.

[16] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000.

[17] Daniel M Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. *Proceedings of the 8th ACM conference on Electronic commerce EC 07*, 55:192, 2007.

[18] Alternating Least Squares Method for Collaborative Filtering. `http://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/`, September 2015.

[19] Anna Gatzioura and Miquel Sànchez-Marrè. A Case-Based Recommendation Approach for Market Basket Data. *IEEE Intelligent Systems*, 2015.

[20] Web Services Glossary. `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice`, September 2015.

[21] Apache HBase Reference Guide. `http://hbase.apache.org/book.html#scripting`, September 2015.

[22] Asela Gunawardana and Christopher Meek. A unified approach to building hybrid recommender systems. *Proceedings of the third ACM conference on Recommender systems RecSys 09*, 27(5):117, 2009.

[23] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[24] Salim Hariri. *Tools and environments for parallel and distributed computing*. J. Wiley, Hoboken, N.J, 2004.

[25] hbase-client Library. `https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/package-summary.html`, September 2015.

[26] Andres Hervas-drane. Word of Mouth and Recommender Systems : A Theory of the Long Tail. *Business*, pages 1–48, 2008.

[27] Apache Tomcat Homepage. `http://tomcat.apache.org/`, September 2015.

[28] Chart.js Homepage. `http://www.chartjs.org/`, September 2015.

[29] D3.js Homepage. `http://d3js.org/`, September 2015.

[30] Jersey Homepage. `https://jersey.java.net/`, September 2015.

[31] Yifan Hu, Chris Volinsky, and Yehuda Koren. Collaborative filtering for implicit feedback datasets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, (July):263–272, 2008.

[32] Adam Jacobs. The Pathologies of Big Data. *Queue*, 7(6):10, 2009.

[33] Dietmar Jannach. *Recommender systems : an introduction*. Cambridge University Press, New York, 2011.

[34] jQuery Homepage. `https://jquery.com/`, September 2015.

[35] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):42–49, 2009.

[36] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[37] Spark officially sets a new record in large-scale sorting. `https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`, September 2015.

[38] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408, 1999.

[39] Current World Population. `http://www.worldometers.info/world-population/`, September 2015.

[40] Friedrich Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):pp. 88–91, 1994.

[41] Resilient Distributed Datasets RDDs. http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds, September 2015.

[42] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[43] B Sarwar, G Karypis, J Konstan, and J Riedl. Item-Based Collaborative Filtering Recommendation Algorithms. *Proceedings of the 10th international conference on World Wide Web. ACM, 2001.*

[44] Paul Cole (SellerEngine). Amazon.com catalog blows past 200m items. http://sellerengine.com/amazon-com-catalog-blows-past-200m-items/, April 2015.

[45] Djoni Haryadi Setiabudi, Gregorius Satia Budhi, I. W J Purnama, and Agustinus Noertjahyana. Data mining market basket analysis' using hybrid-dimension association rules, case study in Minimarket X. *Proceedings of the International Conference on Uncertainty Reasoning and Knowledge Engineering, URKE 2011*, 1:196–199, 2011.

[46] Ya-Yueh Shih Ya-Yueh Shih and Duen-Ren Liu Duen-Ren Liu. Hybrid Recommendation Approaches: Collaborative Filtering via Valuable Content Information. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 00(C):1–7, 2005.

[47] Richard Sprinthall. *Basic statistical analysis*. Pearson Allyn & Bacon, Boston, 2012.

[48] Scaling Hadoop to 4000 nodes at Yahoo! https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html, September 2015.

[49] Relationship to the World Wide Web and REST Architectures. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest, September 2015.

[50] L H Ungar and D P Foster. Clustering Methods For Collaborative Filtering. *Proceedings of the Workshop on Recommendation Systems*, pages 1–16, 1998.

[51] A Beginner's Guide To Upselling and Cross-Selling. http://www.forbes.com/sites/chuckcohn/2015/05/15/a-beginners-guide-to-upselling-and-cross-selling/, September 2015.

[52] Tom White. *Hadoop : the definitive guide*. O'Reilly, Farnham, 2010.

[53] Show-jane Yen, Chia-ching Chen, and Yue-shi Lee. A Fast Algorithm for Mining High Utility Itemsets. pages 90–99, 2012.

[54] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, and Ankur Dave. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2, 2012.