Recommender System for Retail Industry

EASE CUSTOMERS’ PURCHASE BY GENERATING PERSONAL PURCHASE CARTS CONSISTING OF RELEVANT AND ORIGINAL PRODUCTS

FLORIAN CARRA
Recommender System for Retail Industry

Ease customers’ purchase by generating personal purchase carts consisting of relevant and original products

FLORIAN CARRA

Master’s Thesis at KTH CSC
Supervisor & Examiner: Jens LAGERGREN
Abstract

In this study we explore the problem of purchase cart recommendation in the field of retail. How can we push the right customize purchase cart that would consider both habits and serendipity constraints? Recommender Systems application is widely restricted to Internet services providers: movie recommendation, e-commerce, search engine. We brought algorithmic and technological breakthroughs to outdated retail systems while keeping in mind its own specificities: purchase cart rather than single products, restricted interactions between customers and products. After collecting ingenious recommendations methods, we defined two major directions - the correctness and the serendipity - that would serve as discriminant aspects to compare multiple solutions we implemented. We expect our solutions to have beneficial impacts on customers, gaining time and open-mindedness, and gradually obliterate the separation between supermarkets and e-commerce platforms as far as customized experience is concerned.
Contents

1 Introduction 1

I Materials and Methods 3

2 Data Handling 5
  2.1 Pruning Useless Information 5
  2.2 Structuring Data 5
  2.3 Clustering data 5
  2.4 De-identification 6

3 Content-Based Filtering 9
  3.1 Content-Based Filtering in three steps 9
    3.1.1 Item Profiling 9
    3.1.2 Profile Learner 10
    3.1.3 Decision Method 10
  3.2 Average Recommender 10
    3.2.1 Implementation Details 10
  3.3 Random Recommender 10
    3.3.1 Implementation Details 11

4 Neighborhood-Based Methods 13
  4.1 Implicit Scores 13
    4.1.1 Implicit Scores as opposed to Explicit Scores 13
    4.1.2 Temporal Dynamics 14
    4.1.3 Implemented Implicit Scores 14
  4.2 User-Item Matrix 14
  4.3 Similarity Measure 15
  4.4 User-Based Neighborhood 15
    4.4.1 Principle 15
    4.4.2 General Algorithm 16
    4.4.3 Prediction Method 16
    4.4.4 Implemented version of User-Based Neighborhood 17
  4.5 Item-Based Neighborhood 17
Chapter 1

Introduction

As the world tends to digitize and become more complex, we need tools to understand, sort and find the right item serving our purpose. That is the main relevance of Recommender Systems. Among the huge amount of available data and choices, such a system is made to point at the best alternative.

Nevertheless, the problematic of finding what suits you best is not new; it has just been sped up because of finer needs. Possibilities, expectations and necessities have exploded the last few decades. Because we can have direct access to movies, music or products, we expect them to be designed for us, otherwise item, and by extension the provider, is considered as useless.

This ascertainment is even more verified in the domain of retail industry. This domain is in perpetual reinvention, and refusing adaptation would mean a loss of competitiveness. Indeed physical (in opposition to numerical shops or e-commerce) shops have to cope with new competitors, the e-commerce actors. Amazon, ebay or PriceMinister and others have defined a new way of purchasing: faster, sometimes cheaper and often more targeted. On the last point, common retail industry has a role to play. By providing customized services, retailers would improve customer satisfaction, enhance their loyalty and thus increase their benefits. One possible application of such a strategy would be personalized recommendations of products thanks to a Recommender System.

Sharing its own specificity, we can only acknowledge the scope of possibilities of the retail industry, as far as recommendations are concerned. Even if they do not benefit from the countless opportunities of the Web, supermarkets and hypermarkets can rely on a non-negligible amount of clients and products information. Linked to strong computational architectures and improving algorithms, they can reveal a significant potential.

The following study proposes to bring a practical vision to the application of recommender systems in the domain of retail industry. Cash receipt data is being used to apply and test some of the state-of-the art algorithms in order to derive their utility and compare their performances.
How can we benefit from past purchases to help supermarkets' customers buy in a quicker and more efficient way? The objective is to build a purchase cart (also called basket) that would smartly fit to customer's expectations.

We will first display theoretical background necessary to fully understand each of the algorithms and utilized methods. The section will cover process of data cleaning, explanations of Recommender Systems algorithms: Content-Based Filtering, Collaborative Filtering & Advanced Collaborative Filtering. Last point being tackled in the section concerns evaluation methods that are being defined to compare, on the same basis, algorithms performances. The second step will consist in focusing on results we obtained by applying methods on data: Random Recommender, Average Recommender, k-Nearest-Neighbors Recommender, Item-to-Item Recommender & Alternating Least Squares Latent Factors Recommender. Finally, we will discuss results and derive specific utility of each solution.
Part I

Materials and Methods
Chapter 2

Data Handling

2.1 Pruning Useless Information

Initial data is atomic transactions from cash receipts. The first step is to clean and keep only useful information. The only necessary information in transaction is the customer identifier, the product identifier and its involved quantity and date. Furthermore, getting rid of contradictory samples such as null quantity or exotic values is inevitable.

2.2 Structuring Data

As previously explained, data is made of unique transactions that describe events relating a customer to a product bought in one or several exemplars at a specific date. As the final shape of recommendation is the purchase cart, atomic (a single product) transactions are being turned into baskets (group of products). The hypothesis taken here is that products bought by a customer on the same date belong to a single purchase cart. In other words, consideration is taken that a customer bought at most one basket per day. The final structure of purchase history is described in figure 2.1. Each customer is described as a set of dates, each referring to a bag of products that relate to a quantity. The corresponding structure in programming language python is dictionary.

2.3 Clustering data

As evidenced by stores like Amazon.com [10] or Walmart [21], data involved in retail industry (both offline and online) is in the range of millions. Millions of users, millions of single products. As recommendations have to be computed on near real-time and some models does not scale well, there exists techniques to reduce the input size of the model training: sampling, dimensionality reduction or clustering. Clustering methods can be applied to data in order to access group of users or products that share close features [17]. In the study case of Amazon.com [10], it is
explained that the idea behind clustering models is to minimized computations by acting in two steps. First, we create clusters, based on user similarity. The model is then trained for representative users of the cluster. Resulting recommendations are then applied to customers according to their representative cluster (or clusters in case of weighted clusters). By increasing the number of involved clusters, we gain in quality but computation time also increases. A number of clusters equal to the number of customers is the same as no clustering process.

A simple and direct clustering method to apply in the retail industry is products aggregation by categories. The clustering method does not rely on any algorithm. Indeed, as a matter of organization (supermarket shelves), products are already clustered by the retailer in different categories. For instance, the category “eggs” would gather all eggs boxes, no matter brand or size of the box. So by only keeping categories of products instead of products\(^1\), there is a drastic reduction in the number of items while not reducing quality of data description too much. As a consequence, both relevance and rapidity of the recommender will increase.

2.4 De-identification

In the context of the symposium *The Privacy Paradox: Privacy and Its Conflicting Values*, Omer Tene and Jules Polonetsky wrote a paper stating privacy situation in the age of Big Data [19]. First finding is that anonymity is more of a concept than a property. What some called “anonymized” data can always be re-identified thanks to several methods: linking to external data for instance. Anonymization has to be seen as a de-identification process, with different options: pseudonymization, encryption, key-coding, data sharding. First evident positive impact of anonymization

\(^1\)From this point and for practicality, we will refer to category of products when speaking of products
2.4. DE-IDENTIFICATION

concerns security and ethics. As for every computing structure, data is supposed to move, being transferred between different instances and/or databases. Protection of customer anonymity is thus necessary. But another induced interest in anonymizing data is the storage optimization. Indeed, names and ID generated by retailer are often long and not storage-optimized. By attributing auto-incremented ID to both customers and products, we ensure that unicity is preserved while storage size of ID is minimal.
Chapter 3

Content-Based Filtering

Content-Based Filtering is the first approach that has been adopted to design Recommender Systems. One of the most efficient application of Content-Based Filtering is the Information Retrieval on the web [12]. Each item is a web page (or site) and users want the Recommender System to propose them the most suitable pages.

3.1 Content-Based Filtering in three steps

Content-Based Filtering recommenders rely on three steps: item profiling, the profile learner and the decision method [11] [9]. Item profiling is the process that permits to describe item/user in a discrete multi-dimensional space. The profile learner is the model training step. The model, one for each user, links user behavior to items description space. The decision method outputs recommendations, based on models results.

Content-Based Filtering may also designate the opposite process: user profiling, item profile learner and decision method. As a result the obtained model characterizes a product in the space of customer’s features

1 Because the few demographics features characterizing a customer could not efficiently discriminate the large amount of products (millions), we decide to skip this aspect of Content-Based Filtering.

3.1.1 Item Profiling

In a user-item relationship, it is useful to derive features that characterizes the relation. This is moreover essential in Content-Based Filtering. Focusing on items, aim is to represent each representative in a same space. Instead of describing an item by its “common” description - i.e. category, brand, price, size and other features...
the relation between user and item is only defined by their frequency of pairing. Item-User weight is all the more consequent as purchase is frequent.

### 3.1.2 Profile Learner

Now that products have been represented in a multidimensional space, we are able to build a model, based on user's preference. As it is the main idea of Content-Based Filtering, there will be a model for each user. This presents interesting advantages such as light models (in which only one user is taken into account, as opposed to Collaborative Filtering where the model is built for everyone) and user independence. But that also means many drawbacks: over-specialization on user profile or caveat of the new user (no data available). By describing a product from its palatability for a user, the model is being built in at the same time. This is thus a simple statistical model that states preferences for customers only based on their purchase history. Model is rather explanatory than predictive.

### 3.1.3 Decision Method

Once the model has been trained, the final step is to apply it to products for a specified customer. From the results, recommended products are chosen among high-rated products.

### 3.2 Average Recommender

Average Recommender broadly inspires from Content-Based Filtering and apply it to our data.

#### 3.2.1 Implementation Details

Average Recommender is a straightforward application of Content-Based Filtering that uses a single useful parameter:

- **basket size**: size of the recommended basket. The choice is given among:
  
  1. 10
  2. user average basket size
  3. overall average basket size

### 3.3 Random Recommender

The Random Recommender is a model that does not provide customized recommendations but baskets filled up with randomly picked products among all existing.
3.3. RANDOM RECOMMENDER

3.3.1 Implementation Details

As a really naive recommender, the Random Recommender enables only one simple tuning: the basket size. It follows a normal distribution:

\[ N(\text{average basket size}, \text{variance basket size}). \]
Chapter 4

Neighborhood-Based Methods

Neighborhood-Based methods is the first section of the more general family of Recommender Systems: Collaborative Filtering. Collaborative Filtering, as opposed to Content-Based Filtering, uses community behavior to derive individual preferences. The model is unique and global and enables to characterize every user. In the domain of Recommender Systems, this is the most used family of methods, as testifies the quantity of research papers. Neighborhood-Based methods are processed in 3 steps. Firstly the choice is made for the entity to compare: either users or items. Secondly, the definition of the similarity measure that will help finding nearest neighbors. Finally, a strategy is being defined in order to make recommendations to the customer out of its nearest neighbors.

We will first discuss about representation of data through Implicit Scores and User-Item Matrix. Then we tackle Similarity Measure since it is common to every Neighborhood-Based methods. In a fourth section, we will focus on both neighborhood-based approaches: user-based and item-based techniques. And we will finish this section by providing the possible prediction methods.

4.1 Implicit Scores

4.1.1 Implicit Scores as opposed to Explicit Scores

From a general point of view, Recommender Systems data inputs are classified into two families: either implicit or explicit. Explicit data is often formed of ratings. Either binary (like or dislike, as it is the case for YouTube thumbs up/down), or on a scale (from 1 to 5 for Netflix ratings), those ratings are precise and orderable. The opposite view is that such ratings are seen as obtrusive and require an effort from the user [14]. Added to that, explicit ratings are seen as biased [13] since every user gives a note according to their own defined scale: a rating of 4 over 5 given to a Netflix movie could mean a masterpiece for a user and only enjoyable for another. Explicit feedback is often preferred to implicit feedbacks as far as Recommender Models are concerned but the choice is not always given, this is the
case for real world stores. This preference mainly explains the fact that literature is way more abundant regarding explicit feedback. Nevertheless, for the domain we are specifically interested in, common retail industry, interactions between customers and recommender systems are really low. Explicit rating is just inapplicable. That leads us to the second category, implicit data. Defined as the observation of user behavior [7], it brings insight of the opinion a user shares towards an item.

4.1.2 Temporal Dynamics

Implicit observations are time-related. In online Recommender System, visiting a specific page one month ago should not have the same impact on recommendations than a page visited the current day. It is the same for retail industry: a product purchased one year before is less meaningful than another product bought recently. Such intuition is referred to as “temporal dynamics”. Koren, Bell and Volinsky [9] built a model for implicit data where implicit-based ratings depend on temporal parameters.

4.1.3 Implemented Implicit Scores

The objective of implicit scores is to relate a customer to a product through a score that includes aspects such as purchase frequency and recency.

Let $P_{cp}$ be the implicit preference of customer $c$ for product $p$.

$$P_{cp} = \sum_{k \in \Gamma_{cp}} \text{quantity}_{cpk} \times r^{\text{recency}(k)}$$

$\Gamma_{cp}$ is the set of dates when customer $c$ bought product $p$. $\text{quantity}_{cpk}$ is the quantity of product $p$ purchased by customer $c$ at date $k$. $r$ is a ratio such that $0 < r \leq 1$. $\text{recency}$ is a function that states the relative recency of purchase date, defined as follows:

$$\text{recency} : date \mapsto \text{daysBetween}(\max_{d \in D} d - date)$$

4.2 User-Item Matrix

Notion of user-item matrix is common to many Collaborative Filtering methods. This matrix represents (in 2 dimensions) the interactions between item and user in terms of preferences. This matrix is central to Neighborhood-Based methods. Figure 4.1 is the general representation of user-item matrix. $P_{ij}$ is the relation function that links user to item. Some are already fulfilled according to user's behavior (implicit scores). But most of $P_{ij}$ values are unknown and the goal of Recommender System is to predict them.
4.3 Similarity Measure

Neighborhood is a concept that cannot be evaluated without a similarity measure. Similarity $S$ between two objects characterizes their level of resemblance, and thus their neighborhood distance. The greater $S$ the nearer the two objects. In Recommender Systems, objects are items and/or users. The similarity between 2 items will be computed in the space of users (each component is the preference, if present, of the user for the specific item). Reciprocally, for users' similarity measure, the user is represented in the items space. There are two main similarity measures that are widely used in Recommender Systems: cosine similarity and Pearson Correlation [16].

Cosine similarity is a characterization of the angle between 2 vectors. So users (user-based neighborhood) or items (item-based neighborhood) are seen as vectors. If we note $X$ and $Y$ the vector representing ratings/preferences of user in item space (or item in user space), we have:

$$
cos(X, Y) = \frac{\sum_i X_i \times Y_i}{\|X\| \times \|Y\|}
$$

But vectors are sparse since users did not try every product yet. So we need to take into account only the components that are common to both $X$ and $Y$. We note $U$ the set of features that are both present for $X$ and $Y$, the cosine similarity is as follows:

$$
S(X, Y) = \frac{\sum_{i \in U} X_i \times Y_i}{\sqrt{\sum_{i \in U} X_i^2} \times \sqrt{\sum_{i \in U} Y_i^2}}
$$

4.4 User-Based Neighborhood

4.4.1 Principle

User-Based Neighborhood is the original form of Neighborhood-Based methods. The idea is to bring closer users that share similar tastes. Predictions for a user
are then derived from its like-minded users’ preferences. In the case of User-Based Neighborhood, similarity has to be processed between users. In other words, the user-item matrix is considered as a set of rows and similarity is computed between those rows, as shown in Figure 4.2.

![Users Similarity](image)

**Figure 4.2: Users Similarity**

### 4.4.2 General Algorithm

If the goal is to make recommendations for user A, here is the pseudo-code that enables us to determine his nearest neighbors, in Algorithm 1:

**Data:** sparse user-item matrix  
**Result:** sorted list of customer A’s neighbors  
Create $L$ the list of tuples representing $(user, S(A, user))$;  

**for each user B other than A do**

- Compute $S(A, B)$;  
- Append $(B, S(A, B))$ to $L$;  

**end**  
Sort $L$ in descending order (by similarity)  

**Algorithm 1:** User-Based Neighborhood

### 4.4.3 Prediction Method

Once the model\(^1\) has been built, it is necessary to define a method that would predict ratings/preferences of the customer for products he has not been in contact with yet. Two main prediction methods are presented to us: regression and classification. Regression is best suited for ratings whose scale is relatively spread out [3]. Based on ratings/preferences of like-minded customers, initially missing preferences can

---

\(^1\)User Similarity Matrix
be derived. The most widespread method used in this case is the weighted sum\(^2\) [3].

\[
\hat{r}(x, a) = \frac{\sum_{y \in \text{neighborhood}(x) \cap \mathcal{U}} S(x, y) \times r(y, a)}{\sum_{y \in \text{neighborhood}(x) \cap \mathcal{U}} |S(x, y)|}
\]

The previous equation describes the predicted rating relating \(x\) to \(a\). \(x\) and \(y\) are customers whereas \(a\) is a product. \(r(y, a)\) is the initial implicit rating of \(y\) for \(a\). The implicit rating - or score - is also denoted \(P_{ya}\) and takes into account both frequency and recency of purchase. \(\mathcal{U}\) represents the set of customers for which there is an expressed relation with \(a\). The main parameter that can be tuned lies in \(\text{neighborhood}(x)\). Indeed it represents the level of neighborhood we want to consider in our computation. Often denoted “\(k\)” (as in k-Nearest-Neighbors), this parameter manages the number of nearest neighbors. The higher \(k\), the more accurate but the more computationally expensive calculus will be.

### 4.4.4 Implemented version of User-Based Neighborhood

One of the implemented Recommender relies on User-Based Neighborhood. The k-Nearest-Neighbors Recommender implements this solution by providing flexibility through parameters choice:

- **basket size**: size of the recommended basket. The choice is given among:
  1. 10
  2. user average basket size
  3. overall average basket size

- **\(k\)**: number of neighbors to consider when predicting relevant products

- **minimum number of common products**: co-purchased products quantity threshold below which 2 customers can not be considered as “neighbors” even if their computed similarity measure would be high

### 4.5 Item-Based Neighborhood

#### 4.5.1 Motivations

Even if User-Based Filtering presents good results, it implies some drawbacks. First, accuracy is not always optimal [16] [1]. More important, explaining a recommendation that relies on an anonymous user (or anonymous users) is almost impossible. With the Item-Based approach, improvements can be seen. As far as explanation is

\(^2\)There are other way of computing weights than simply applying similarity [1]
concerned, it turns out to be easier to explain to a customer that a product is rec-
ommended because it is close to another product he showed interest for previously.
Moreover, as similarity measures have to be computed between each possible pairs,
the number of combinations grows approximately as $n^2$, with $n$ the total number of
objects. So it could be wise to choose $n$ as the least between the number of items
and the number of users. In the case of retail industry, both are of the order of a
million, but perhaps customers outnumber products. As a consequence, item-based
approach is even more justified.

### 4.5.2 Principle

The item-user approach is the exact same as user-based Neighborhood, except that
similarity is computed between columns (Figure 4.3), as opposed to rows for users.

![Figure 4.3: Items Similarity](image)

This approach has mainly be used by Amazon.com, as explained by Linden, Smith
and York [10], the three of them being employed by the e-commerce giant actor.
Items are represented as vectors whose components are ratings/preferences of users.

### 4.5.3 Item-Item Similarity Algorithm

As presented in their paper [10], Amazon.com team designed an algorithm, in Al-
gorithm 2, that optimizes similarity computation between items. Indeed, our only
interest relies in items that have been purchased together at least once.
The main advantage of such a method is the offline computation. Such a com-
putation is allowed thanks to the static aspect of items [16], it is not the case for
user-based approaches. The whole computation that leads to similar-items table is
done offline. Indeed, from the algorithm introduced above, we can obtain similarity
between every pair of items (as soon as they have been bought together). The rec-
ommendation for a customer is then done according to his latest purchases: we read
4.5. ITEM-BASED NEIGHBORHOOD

Data: sparse user-item matrix

Result: items similarity matrix

for each item $I_1$ in product catalog do
  for each customer $C$ who purchased $I_1$ do
    for each item $I_2$, purchased by $C$, other than $I_1$ do
      Record that $I_1$ and $I_2$ have been purchased together: store $I_2$ in $L$;
    end
  end
  for each item $I_3$ in $L$ do
    Compute similarity $S(I_1, I_3)$;
  end
end

Algorithm 2: Item-Item Similarity

in the similar-items table the nearest products and make them a recommendation. The offline part is computationally expensive ($O(n_{products}^2 \times n_{customers})$ in the worst case scenario and $O(n_{products} \times n_{customers})$ in practice since data is sparse). But the online part is really quick ($O(n_{products})$).

4.5.4 Implemented version of Item-Based Neighborhood

In our implemented version of Item-Based Neighborhood method, namely Item-to-Item Recommender, we propose some customization through parameters:

- **basket size**: size of the recommended basket. The choice is given among:
  1. 10
  2. user average basket size
  3. overall average basket size

- **ratio similar/favorite products**: ratio representing the number of products that are recommended as “similar” to customer’s favorite products

- **minimum number of co-buyers**: customers threshold below which 2 products can not be considered as “similar” even if their computed similarity measure would be high
Chapter 5

Advanced Collaborative Filtering - Latent Factors Models

The second main branch of Collaborative Filtering, after Neighborhood-based methods, is the Latent Factor Models. As explained, Collaborative Filtering presents a strong asset compared to Knowledge-Based Recommenders or even Content-Based Methods since it does not require external information, user past behavior is sufficient [7]. That encourages us to use such algorithms in the case of retail. Latent Factor Models are based on a description of users and items according to “latent factors”. Those factors are criteria that can discriminate the different elements. Instead of describing a movie according to its genre, actors, production date, etc..., the algorithm will find by itself the most relevant dimensions of discrimination for the movies. Resulting dimensions generally do not correspond to a human-understandable criterion. Techniques to determine latent factors are multiple: Probabilistic Latent Semantic Analysis (pLSA) [6], neural networks [15], Latent Dirichlet Allocation [2] and matrix factorization techniques [22][9][7]. In this section we will focus on the latter methods, that turns out be very popular in Recommender Systems thanks to its scalability and accuracy [8].

5.1 Matrix Decomposition

The basic idea behind Matrix Decomposition for Recommender System is to separate users from items. Customers and products are to be represented in a same space of latent factors. Nevertheless, the starting point of the Recommender System is a sparse matrix $M$, the User-Item Matrix, that contains partial relations between users and items. Each element of $M$ is supposed to be the preference the concerned customer presents for the specific product.

The aim is to split $M$ into two sub-matrices that will characterize in the one hand users ($U$) and in the other hand the items ($V$) such that their product is $M$.

Figure 5.1 shows a graphical view of the process. $M$ is the User-Item matrix for
n customers and m products. We decide to represent each entity in a space whose dimension is \( f \). \( x_k \) are customers and \( i_k \) are products. \((u_j, k)_{k \in [1, f]}\) is the representation of customer \( j \) in the latent factors space (of dimension \( f \)) whereas \((v_k, j)_{k \in [1, f]}\) is the representation of product \( j \). The principal interest in representing data in this shape is that \( \hat{r}(x_k, i_l) = u_{k, l} \cdot v_{l, f} \)

5.2 Optimization Problem

5.2.1 Definition

The problem of Matrix factorization is to seek \( U \) and \( V \) that best fit to \( M \). This is an optimization problem:

\[
\arg \min_{U, V} f(M, U \times V)
\]

\( f \) is the function to optimize and a part of the whole problem is to define the most suitable \( f \) function. A first approach would be to minimize the difference between \( M \) and the approximation of \( M \) in term of \( U \) and \( V \). It is of use to consider the
5.2. OPTIMIZATION PROBLEM

squared 2-norm.

$$\arg \min_{U,V} \| M - U \ast V \|^2$$

This optimization problem is solvable but not applicable to sparse matrix, which is the case of $M$. Instead we will consider only $M$ entries that we know.

$$\arg \min_{U,V} \sum_{(i,j) \in I} \| M_{i,j} - U_{i,.} \cdot V_{.j} \|^2$$

$I$ is the set of couples $(i, j)$ for which $r(i, j)$ is defined. It is clear that $I$ does not contain many couples. Indeed, in the retail application, $I$ is the set of (customer, product) for which there has been a relation, such as a purchase or an expressed preference. Over millions of products, a customer might be familiarized with approximately hundreds or thousands of them. That represents only 0.1% of $M$ entries... So in order to avoid overfitting, a penalty term will be associated to “complex” models [22]. One of the most common in that situation is to append a regularization factor, often referred to as $\lambda$ or Tikhonov regularization [20].

$$\arg \min_{U,V} \sum_{(i,j) \in I} \| M_{i,j} - U_{i,.} \cdot V_{.j} \|^2 + \lambda (\| \Gamma U U \|^2 + \| \Gamma V V \|^2)$$

This form of optimization is the most formal and basically applicable for Recommender Systems Matrix Decomposition. $\Gamma$ are matrices that enables to control what is considered as a “complex” model.

5.2.2 Alternating Least Squares Optimization

Once the optimization problem is established, the next step is to define a strategy to solve the equation while respecting the accuracy-computation trade-off.

The idea developed by the HP Labs team [22] within Netflix challenge is an answer to the current optimization problem. Instead of updating both $U$ and $V$ at the same step (as it is the case in Stochastic Gradient Descent [4]), they successively fix one element ($U$ or $V$) while updating the other. The update rule is based on Least Squares (hence the appellation Alternating Least Squares). By doing so, they turn the non-convex problem into a quadratic problem, which is a convex problem [22]. For instance, by fixing $V$:

$$U_{i,.} = \arg \min_{U_{i,.}} \sum_{(i,j) \in I} \| M_{i,j} - U_{i,.} \cdot V_{.j} \|^2 + \lambda (\| \Gamma U U \|^2)$$

The same goes with $V$ when $U$ is fixed. What is important to note here is that each row of $U$ and each column of $V$ can be optimized independently from other rows (for $U$) and columns (for $V$). Parallelization becomes possible.
Data: sparse User-Item Matrix
Result: User Matrix $U$ & Product Matrix $V$
Initialize matrix $M$ by assigning the average rating for the product as the first row, and small random numbers for the remaining entries;

while stopping criterion is not satisfied do
    Fix $V$, solve $U$ by minimizing the objective function (sum of squared errors);
    Fix $U$, solve $V$ by minimizing the objective function similarly;
end

Algorithm 3: ALS decomposition of User-Item Matrix

The main algorithm applied for Alternating Least Squares utilization is described in Algorithm 3.
The stopping criterion is either a maximum number of iteration or a small quantity $\epsilon$ below which the variation of $U$ and $V$ is considered null.

Time complexity for a step of Alternating Least Squares Optimization solver is $O(|\Omega|f^2 + (m + n)f^3)$ [7]. $f$ being the number of hidden factors, $\Omega$ the number of operation for one update, $m$ the number of products and $n$ the number of customers.

5.3 Implemented version of ALS

We used ALS method to build a Recommender that would rely on Latent Factors Model. The so-called ALS Recommender is customizable:

- **basket size**: size of the recommended basket. The choice is given among:
  1. 10
  2. user average basket size
  3. overall average basket size

- **rank**: number of latent factors, $f$, used for representing customers and products

- **iterations**: number of iterations before stopping

- **$\alpha$**: confidence in initial version of User-Item Matrix. $0 \leq \alpha \leq 1$

- **$\lambda$**: model complexity penalization factor
Chapter 6

Evaluation Methods

In order to evaluate a recommender relevance and compare it to another one, we need to define objective quantities that would characterize performances of such recommender.

6.1 Offline Evaluation

The main distinction to be made concerning recommender evaluation methods is between online and offline evaluations. Indeed, the two methods differ in term of involved actors. Online evaluation is a constant cycle between users and model: users provide their data to the model which improves itself and then provide a version of recommender to users. The users test and evaluate the recommender/recommendations and give their updated data to the model which learns from the feedbacks. And so on. Whereas offline evaluation is a one way ticket for data that comes from users once and are utilized by the model to both build and evaluate its performances.

There are multiple strategies to combine offline and online evaluations in order to optimize relevance of recommender. But in the case of retailer, in the current study, a non-negligible constraint is the data procuration. Indeed, dataset is static and interaction with real customers is reduced to zero. That is why the study merely relies on offline evaluation to analyze performances of the different versions of recommender.

6.2 Evaluation Strategy

6.2.1 Dataset Split

The input data is the set of baskets for every customer over the 2-year purchase period. As the dataset is used for both training and evaluation of performances, a split is processed to separate data into 2 subsets. The training set will serve as purchase history whereas testing set will be used as comparator to recommendations.
CHAPTER 6. EVALUATION METHODS

<table>
<thead>
<tr>
<th></th>
<th>total number of baskets</th>
<th>number of baskets in the training test</th>
<th>number of baskets in the testing test</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer 1</td>
<td>100</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>customer 2</td>
<td>20</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>customer 3</td>
<td>4</td>
<td>30</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 6.1: Examples of train/test distribution, for ratio = 3

In order to characterize proportions of data in the training set and testing set, we rely on a ratio defined as:

\[
\text{train/test ratio} = \frac{\text{nb baskets in training set}}{\text{nb baskets in testing set}}
\]

Splitting has to be considered in term of baskets. Indeed, every customer has baskets in both training and testing tests. For instance, a ratio of 3 would lead to the following distribution presented in Table 6.1.

6.2.2 Evaluation Scope

Evaluation is conducted over testing set baskets. With the intention of getting an insight as accurate as possible, we compare every basket part of recommendations to the corresponding basket in the testing set. Indicators are then computed as the mean over all baskets.

6.3 Indicators

6.3.1 Confusion Matrix

A common way of evaluating performances of a model that outputs sets is to build a confusion matrix. That enables us to characterize quantities such as False Positive (wrongly recommended), True Positive (relevant recommendation), False Negative (omitted recommendation) and True Negative (suitable omission).

Based on the confusion matrix, we are now able to compute three indicators of relevance:

\[
\begin{align*}
\text{Precision} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \\
\text{Recall} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \\
f\text{-measure} &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\end{align*}
\]

As far as those indicators are concerned, the higher the better. Precision corresponds to the opposite of the "risk" taken among recommendations. Whereas recall
6.3. INDICATORS

corresponds to the proportion of unveiled truth in testing set. F-measure is a mixture of both previous indicators and offers a good appreciation of model relevance.

6.3.2 Novelty

As Confusion Matrix based indicators only evaluate accuracy of recommendations and our aim is not only to decrypt past behavior, it is necessary to define a quantity that would quantify the degree of serendipity in the recommendations. This quantity we call novelty is nothing but the proportion of currently unseen products in the recommendation.

\[ \text{Novelty} = \frac{\text{discovered products}}{\text{recommended products}} \]

A high novelty is not synonym of good performances since customer is not likely to buy only new products. As the opposite, a recommender that provides low or null novelty recommendations does not fulfill objectives we initially set.

Among the new products, we are also interested in products that have effectively been purchased by the customer.

\[ \text{Relevant Novelty} = \frac{\text{discovered products effectively purchased}}{\text{discovered products}} \]

The higher relevant novelty is, the best. Nevertheless, since we do not interact with the customer (offline evaluation), relevant novelty is likely to be really low. Indeed, as the customer is not aware of the recommendation yet, he is not able to analyze a recommended product and find out if this is a relevant product.
Chapter 7

Distributed Architecture

Besides the algorithmic aspect of Recommender Systems, such services need to rely on adapted and adaptable infrastructures. In retail, involved components are numbered in millions: millions of customers, millions of products... Furthermore, quality of service requires near-real-time systems that are able to take into account every evolution in data. One of the accepted solution towards the problem is Big Data. Clusters of machines permits a distribution of both data and its computation through distributed algorithms.

7.1 Hadoop and MR

Apache™ Hadoop® [5] is a framework that aims to offer processing tools that are distributed, scalable and fault-tolerant. Main modules of Hadoop include YARN (job scheduler and resource manager), Hadoop Distributed File System (HDFS™) and MapReduce. Hadoop runs on cluster of machines, hence the need of distributed algorithms and softwares. In Figure 7.1, we make the distinction between Master Node and Slave Nodes.

One of the main solution to benefit from such a distributed structure is the MapReduce solution. Figure 7.2 illustrates how MapReduce works in 3 main steps:

- Map: data is sorted depending on the Map instruction
- Shuffle: Map outputs are re-distributed across the cluster
- Reduce: data is sorted such that “similar” data is gathered and processed as a single entity

7.2 Apache Spark

Apache™ Spark® [18] is a project that comes on top of Hadoop solution. By providing a new distribution model, it outperforms the standard Map Reduce scheme.
It is a scalable in-memory computation process that implements an advanced Directed Acyclic Graph for task scheduling. Spark comes with a Machine Learning Library called *MLLib*. *MLLib* implements distributed versions of Machine Learning algorithms such as K-Means, Random Forest or Alternating Least Squares for Collaborative Filtering.
Part II

Results
Chapter 8

Data Exploration

8.1 Data Scope

Involved data can be summarized with following figures:

- 1990 unique customers
- 13,850 categories of products
- from 2012-01-02 to 2014-01-31
- 3,887,297 unique transactions defined as events (date, customer, product, quantity)

8.2 Storage Optimization

Along the different steps of data cleansing, we were able to diminish disk space occupied by transactions-related data. As a reminder, cleaning process includes pruning useless information, structuring data, clustering data and de-identification. The storage optimization is described in Table 8.1.

<table>
<thead>
<tr>
<th>Raw Transactions</th>
<th>Cleaned Transactions</th>
<th>Cleaned Baskets</th>
<th>Anonymous Baskets</th>
</tr>
</thead>
<tbody>
<tr>
<td>316.8 Mb</td>
<td>117.5 Mb</td>
<td>40.2 Mb</td>
<td>26.6 Mb</td>
</tr>
</tbody>
</table>

Table 8.1: Storage Optimization

8.3 Typical Customer Behavior

In order to fully perceive shape of involved data, plots are being made. Figure 8.1 depicts distribution of baskets among customers, with an overall mean of 103 purchase carts for the typical customer between 2012-01-02 and 2014-01-31.
Figure 8.2 depicts distribution of basket constituting products among customers, with an overall mean of 18.7 products in the typical purchase cart.

Figure 8.3 depicts distribution of categories of products that are familiar to customers, with an overall mean of 592 known categories per customer between 2012-01-02 and 2014-01-31.

Figure 8.4 depicts temporal distribution of customers, the daily average number of customers being 270.

Figure 8.1: Average number of purchase carts by customer

Figure 8.2: Average number of products in the purchase cart for each customer
8.3. TYPICAL CUSTOMER BEHAVIOR

Figure 8.3: Number of purchased categories of products for each customer

Figure 8.4: Number of purchase carts (equivalently customers) for each day of the considered period
Chapter 9

Random Recommender

Random Recommender is the most naïve version of a recommender. It is used as baseline for comparison.
Each basket is built as a random pick among all existing products.

9.1 Benchmark

Table 9.1 gathers parameters sets that have been tested for the Random Recommender.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>average basket size</td>
<td>6 9 12 15 17 18 21 24 27 17</td>
</tr>
<tr>
<td>variance of basket size</td>
<td>4 4 4 4 4 4 4 4 4 0</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3 3 3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td>average basket size</td>
<td>17 17 17 17 17 17 17 17 17 17</td>
</tr>
<tr>
<td>variance of basket size</td>
<td>1 2 3 5 6 4 4 4 4 4</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3 3 3 3 3 2 5 10 100 1000</td>
</tr>
</tbody>
</table>

Table 9.1: Random Recommender Benchmark

9.2 Graphs

The novelty is basically constant over all configurations, around 95%.
The same goes with relevant novelty that reaches around 0.65%.
Figure 9.1: Plots of impact of basket size parameters over performances of Random Recommender - (a) mean ; (b) variance

Figure 9.2: Evolution of performances with train/test ratio
Chapter 10

Average Recommender

The Average Recommender has to be considered as the Content-Based Filtering Recommender in term of users.

10.1 Benchmark

Table 10.1 gathers parameters sets that have been tested for the Average Recommender.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>average</th>
<th>random</th>
<th>average</th>
<th>top 10</th>
<th>average</th>
<th>average</th>
<th>average</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>basket size strategy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>train/test ratio</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 10.1: Average Recommender Benchmark

10.2 Graphs
CHAPTER 10. AVERAGE RECOMMENDER

Figure 10.1: Impact of basket size strategies over performances

Figure 10.2: Evolution of performances with train/test ratio
Chapter 11

k-Nearest-Neighbors Recommender

User k-Nearest-Neighbors Recommender is a weighted sum of preferences of k Nearest Neighbors for every customer. By neighbors we mean here users that share a high similarity as far as products’ purchase habits are concerned.

11.1 Benchmark

Table 11.1 gathers parameters sets that have been tested for the user k-Nearest-Neighbors Recommender.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0  1  2  3  5  10  20</td>
</tr>
<tr>
<td>minimum number of common products</td>
<td>200  200  200  200  200  200  200</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average average average average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3  3  3  3  3  3  3</td>
</tr>
<tr>
<td>k</td>
<td>1  1  1  1  1  1  1</td>
</tr>
<tr>
<td>minimum number of common products</td>
<td>0  5  10  20  50  100  500</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average average average average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3  3  3  3  3  3  3</td>
</tr>
<tr>
<td>k</td>
<td>1  1  1  1  1  1  1</td>
</tr>
<tr>
<td>minimum number of common products</td>
<td>200  200  200  200  200  200  200</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average random top 10 average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>2  3  3  5  10  100  1000</td>
</tr>
</tbody>
</table>

Table 11.1: user k-Nearest-Neighbors Recommender Benchmark
11.2 Graphs

Figure 11.1: Impact of basket size strategies over performances

Figure 11.2: Evolution of performances with k
11.2. GRAPHS

Figure 11.3: Evolution of performances with minimum number of common products

Figure 11.4: Evolution of performances with train/test ratio
Chapter 12

Item-to-Item Recommender

Item-to-Item Recommender is the second Recommender based on Collaborative Filtering Neighborhood methods. It uses similarity between products to push similar products along with favorite products.

12.1 Benchmark

Table 12.1 gathers parameters sets that have been tested for the Item-to-Item Recommender.

12.2 Graphs
### Parameter Name

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio similar/favorite products</td>
<td>0 0.1 0.25 0.33 0.5 1 2 1</td>
</tr>
<tr>
<td>minimum number of co-buyers</td>
<td>400 400 400 400 400 400 400 1</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average average average average average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td>ratio similar/favorite products</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>minimum number of co-buyers</td>
<td>5 10 20 50 100 200 300 500</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average average average average average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3 3 3 3 3 3 3 3</td>
</tr>
<tr>
<td>ratio similar/favorite products</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>minimum number of common products</td>
<td>1000 400 400 400 400 400 400 400</td>
</tr>
<tr>
<td>basket size strategy</td>
<td>average random top 10 average average average average average</td>
</tr>
<tr>
<td>train/test ratio</td>
<td>3 3 3 2 5 10 100 1000</td>
</tr>
</tbody>
</table>

Table 12.1: Item-to-Item Recommender Benchmark

![Influence of basket size strategy over indicators](image.png)

Figure 12.1: Impact of basket size strategies over performances
12.2. GRAPHS

Figure 12.2: Evolution of performances with ratio similar/favorite products

Figure 12.3: Evolution of performances with minimum number of co-buyers
Figure 12.4: Evolution of performances with train/test ratio
Chapter 13

Alternating Least Squares Latent Factors Recommender

ALS Recommender is classified as a Collaborative Filtering tool and relies on matrix decomposition approach. Optimization method used is Alternating Least Squares.

13.1 Benchmark

Table 13.1 gathers parameters sets that have been tested for the ALS Recommender.

13.2 Graphs
### Table 13.1: ALS Recommender Benchmark

| Parameter Name               | Configurations | Configurations | Configurations | Configurations | Configurations | Configurations | Configurations | Configurations | Configurations |
|-----------------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| ALS rank                    | 8              | 10             | 12             | 10             | 10             | 10             | 10             | 10             | 10             | 10             |
| number of ALS iterations    | 10             | 10             | 10             | 5              | 15             | 20             | 10             | 10             | 10             | 10             |
| ALS alpha                   | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0              | 0.25           |
| ALS lambda                  | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              |
| basket size strategy        | average        | average        | average        | average        | average        | average        | average        | average        | average        | average        |
| train/test ratio            | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              |
| ALS rank                    | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             |
| number of ALS iterations    | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             |
| ALS alpha                   | 0.5            | 1              | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           |
| ALS lambda                  | 0              | 0              | 0.5            | 2.5            | 5              | 7.5            | 10             |               |               |               |
| basket size strategy        | average        | average        | average        | average        | average        | average        | average        | average        | average        | average        |
| train/test ratio            | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              | 3              |
| ALS rank                    | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             |
| number of ALS iterations    | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             | 10             |
| ALS alpha                   | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           | 0.75           |
| ALS lambda                  | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              |
| basket size strategy        | random         | top 10         | average        | average        | average        | average        | average        | average        | average        | average        |
| train/test ratio            | 3              | 3              | 2              | 5              | 10             | 100            | 1000           |               |               |               |
13.2. GRAPHS

Figure 13.1: Impact of basket size strategies over performances

Figure 13.2: Evolution of performances with train/test ratio
CHAPTER 13. ALTERNATING LEAST SQUARES LATENT FACTORS RECOMMENDER

Figure 13.3: Plots of impact of ALS parameters on performances - (a) rank ; (b) number of iterations ; (c) alpha ; (d) lambda
Part III

Discussion
Chapter 14

Impact of parameters over Recommenders behavior

Relying on results for each Recommenders on their corresponding configurations, we determine influence of each parameter in the overall behavior of the Recommender.

14.1 Random Recommender

Random Recommender is the baseline model and, as such, does not provide sensational results.

14.1.1 Novelty Derivation

As far as novelty is concerned, obtained value can be found by analyzing data:

\[
\text{novelty} = 1 - \frac{\text{number of purchased products}}{\text{overall number of products}} = \frac{13850 - 592}{13850} \approx 0.957
\]

14.1.2 Erratic Behavior

With variance and mean of basket size evolving, we observe an erratic behavior of performances. That is especially true for variance. This behavior is obviously related to the randomness that characterizes the model.

14.1.3 Boosting performances

When average basket size grows, we can see that precision seems to stabilize while recall, and thus f-measure, grows. That is explained by the fact that boosting basket size induces a higher number of recommended products. That implies more relevant products (true positive) but also more irrelevant products (false positive). As ratio true positive / false positive is conserved (due to random, or gaussian, distribution), precision remains still while recall increases.
14.2 Average Recommender

Average Recommender is merely a statistical explanatory tool, that focus of individual purchase history. That is why novelty is null.

14.2.1 Basket Size Strategy

As for the remaining Recommenders (Average, user K-Nearest-Neighbors, Item-to-Item and ALS), strategy of basket size as little but non-negligible impact on performances. In the top 10 strategy, the restricted number of products per basket (only 10) implies a better precision but a worse recall. Despite their structural discrepancy, average (constant size for a given customer) and random (gaussian distribution for every basket) share very similar behavior as far as performances are concerned.

14.2.2 Train/Test Ratio

The higher train/test ratio, the more accurate the recommendations. Indeed, by increasing the ratio, we increase training period while diminishing testing period. That means that first, model training is more efficient and second that customer behavior is less likely to evolve over the short period of recommendations, compared to the training period.

14.3 User k-Nearest-Neighbors Recommender

User k-Nearest-Neighbors is an implementation of neighborhood basic approach.

14.3.1 Neighborhood Scope

First observation is that $k = 0$ reverts to Average Recommender, since we do not benefit from others’ preferences.

Confusion matrix related indicators - namely precision, recall & f-measure - shows an exponential decay with an asymptote at about 10%. The same goes with relevant novelty with a stabilization at about 6%. On the contrary, showing an anti-symmetric evolution with other indicators, novelty displays a logarithmic growth. A trade-off needs to be found between novelty and the other indicators. A wise decision would be to choose 1 for the value of $k$.

14.3.2 Products Support

Precision, Recall & f-measure evolution with respect to minimum support of jointly purchased products is a convex function, with a minimum at 100. Likewise, Novelty is a concave function of products support, and reaches is maximum for 100. Since
we want to maximize performances while not introducing too much novelty, we should escape from 100 for the minimal products support. Relevant novelty reaches a maximum at 200. This seems to be the optimized configuration.

Boosting the minimal support for commonly purchased products between two customers is synonym of greater accuracy. Indeed, by comparing customers over the greatest amount of preferences we obtain a precise measure of similarity. Nevertheless, pushing the support too high backfires the phenomenon. A value of 500, close to the average number of known products per customer (592), implies that candidates for neighborhood are rare.

14.3.3 Train/Test Ratio
As the train/test ratio grows we observe a drop in (relevant) novelty performances. This is explained by the fact that pushing unseen products is harder as training period expands, and testing diminishes.

14.4 Item-to-Item Recommender

Item-to-Item Recommender is the item-based neighborhood method implementation. This strategy is mainly known because of Amazon “customers who bought this item also bought...”.

14.4.1 Similar/Favorite Ratio
A Recommender with ratio at 0 is equivalent to Average Recommender since no new products are proposed.

With the similar/favorite products ratio growing, we observe a logically relatively linear growth of novelty but a decrease in standard indicators, namely precision, recall and f-measure. At the same time, relevant novelty displays a concave curve that maximizes for a ratio of 1.0.

14.4.2 Customers Support
Minimal customers support for products aims to restraint products pairs candidate for similarity measure. Growing the minimal support results in improving model accuracy. Standard indicators grow bigger, along with relevant novelty while novelty decreases.

We observe a fracture between 500 and 1000. This means that there are no products that have been bought by the same customers over the training period. Indeed, with
1989 customers, for a given pair of products, it would require almost one customer over two that have already purchased both specific products.

### 14.4.3 Train/Test Ratio

As for User kNN Recommender, novelty diminishes when train/test ratio grows. This behavior is increased in the case of relevant novelty.

### 14.5 ALS Recommender

ALS Recommender is the Recommender that benefit from state-of-the-art Collaborative Filtering method: latent factors based on matrix factorization. This specific method has been democratized by the Netflix Challenge.

#### 14.5.1 Intermediate Performances

Standard indicators - precision, recall and f-measure - displays relatively low scores (approximately 6%). Nevertheless, relevant novelty performances are the best (around 20%), compared to other solutions. Furthermore, the also high novelty (around 20%) corroborates the interest of ALS Recommender for pushing interesting new products. Concretely, it means that each 25 recommended products, there is averagely 1 product that was unknown but still bought. Without the customer being aware of the recommendation!

#### 14.5.2 ALS parameters

It appears that confidence in initial preferences does not need to be high (0.25) for the Recommender to perform best. For the data we used to train our model, we observe that penalizing the complexity is not relevant. Even more, it is detrimental to performances. Indeed, matrices being consequent (thousands of customers and tens of thousands products), applying a penalty to their norm is not relevant.
Chapter 15

Running Time Discussion

Along with performances discrepancies, all versions of Recommender Systems that have been implemented provide different computational time, depending on their algorithmic structure.

**Random** and **Average** are models that are computed on-the-fly, along with their associated predictions/recommendations. Because those models are naïve and process each customer (**Average**) or product (**Random**) independently, computation is fast. Approximately 15 seconds for all customers and thus around 0.0075 seconds per customer are sufficient to recommend, from scratch, “elaborated” baskets.

**Item-to-Item** is processed in two distinct steps. The first step consists in deriving preferences from purchase history and building corresponding items similarity matrix. This step is done offline and can be easily distributed over multiple machines on a cluster. Furthermore, this step is the model training step and does not need to be dynamically updated. For the two previous reasons - namely distributed algorithm and batch model training - computational time is not relevant. Frequency of model update would be more interesting. The second step is the prediction step. This online step needs to be optimized for requests to be answered in almost real-time. The predictions for all customers take approximately 40 seconds, an average of 0.0201 second per customer.

**User k-Nearest-Neighbors** uses the same 2-step scheme as **Item-to-Item**. The predictions for all customers take approximately 32 seconds, an average of 0.0161 second per customer.

**ALS** is also a 2-step method. In a first step we compute initial preferences from customers purchase history. In the same step we train the model by decomposing user-item matrix. This is done offline, in a distributed computing environment. In the second step, we apply the model by reading matrices product. This step takes 16 seconds for all customer, that is to say an average of 0.0080 second per customer.
The three Collaborative Filtering implemented Recommenders - **User kNN**, **Item-to-Item** and **ALS** - require a batch model computation which is processed on a distributed computing environment. So we need to define frequency of update. It could go from once a month for static systems to once an hour for volatile systems. The lack of adaptability could be a substantial handicap if it were not used in offline retail industry, where time scale is of the order of day.
Chapter 16

Recommenders Performances Comparison

After analyzing results of different Recommenders over all their configurations, we gathered best-case sets of parameters for each. That will be useful to compare Recommenders with each other.

16.1 Performances

Table 16.1 presents performances for Recommenders that have been optimized.

<table>
<thead>
<tr>
<th>Performance</th>
<th>Random</th>
<th>Average</th>
<th>kNN</th>
<th>Item-to-Item</th>
<th>ALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>basket size strategy</td>
<td>random</td>
<td>random</td>
<td>average</td>
<td>average</td>
<td>random</td>
</tr>
<tr>
<td>precision (%)</td>
<td>0.12</td>
<td>16.28</td>
<td>13.51</td>
<td>15.50</td>
<td>6.27</td>
</tr>
<tr>
<td>recall (%)</td>
<td>0.18</td>
<td>17.67</td>
<td>14.55</td>
<td>16.48</td>
<td>6.48</td>
</tr>
<tr>
<td>f-measure</td>
<td>0.14</td>
<td>16.95</td>
<td>14.01</td>
<td>15.97</td>
<td>6.37</td>
</tr>
<tr>
<td>novelty (%)</td>
<td>95.59</td>
<td>0</td>
<td>27.53</td>
<td>6.92</td>
<td>19.26</td>
</tr>
<tr>
<td>relevant novelty (%)</td>
<td>0.63</td>
<td>0</td>
<td>8.14</td>
<td>21.69</td>
<td>19.28</td>
</tr>
<tr>
<td>computational time (sec)</td>
<td>16.36</td>
<td>14.39</td>
<td>32.66</td>
<td>39.65</td>
<td>16.29</td>
</tr>
</tbody>
</table>

Table 16.1: Comparison of Recommenders Performances
CHAPTER 16. RECOMMENDERS PERFORMANCES COMPARISON

Figure 16.1: Comparison of Recommenders performances

16.2 Graph

16.3 Pros and Cons of each solution

As a summary of Recommenders comparison, Table 16.2 gathers major points of each version of Recommender Systems.

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>Trust, Fidelity &amp; Transparency</td>
<td>No Serendipity</td>
</tr>
<tr>
<td></td>
<td>Limited Computational Time</td>
<td>User Cold Start</td>
</tr>
<tr>
<td>User kNN</td>
<td>High Novelty</td>
<td>User Cold Start</td>
</tr>
<tr>
<td></td>
<td>Offline Scalability</td>
<td>Static Model</td>
</tr>
<tr>
<td>Item-to-Item</td>
<td>Explained Recommendations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Balanced Recommendations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Offline Scalability</td>
<td>Static Model</td>
</tr>
<tr>
<td></td>
<td>Relevant Serendipity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New User Integration</td>
<td></td>
</tr>
<tr>
<td>ALS</td>
<td>Excellent Serendipity</td>
<td>Cold start Issue (both customers &amp; products)</td>
</tr>
<tr>
<td></td>
<td>Offline Scalability</td>
<td>Static Model</td>
</tr>
</tbody>
</table>

Table 16.2: Recommenders Pros and Cons

**Random** Recommender is the baseline recommender and, as such, embodies no interesting solution.
16.3. PROS AND CONS OF EACH SOLUTION

**Average** Recommender can bring trust, fidelity and transparency as indicators such as precision, recall and f-measure depict. On the other side, proposing only known products prevents from serendipity. As explained before, the naivety of model implies that not much time is required to perform adequate recommendations. Final point is User Cold Start, it denotes that complexity to train a model for a user that provides little or no purchase history.

**User kNN** Recommender provides a high novelty while partially relevant. As a two-step model, it shows difficulty to adapt to fast changes on customer data. Nevertheless, a solution has been found to make offline computation scalable. As for **Average** Recommender, introducing a new customer may be problematic because similarity with other customers is impossible, at least difficult, to derive.

**Item-to-Item** Recommender presents the same issue as User kNN as far as model is computed. Training model is not incremental but can be computed on distributed environment. Main advantages of recommendations in **Item-to-Item** Recommender are their ease to be explained and their novelty/favorite trade-off in composition. Indeed, a novel product always come along with a favorite product. It turns out serendipity is relevant since customers actually buys what recommendations predicted. As a discriminating element compared to previous recommenders, **Item-to-Item** deals efficiently with new customers. As recommendations are pushed on the basis of products comparison, a customer can be provided with recommendations as soon as we a first sample of data on the new customer.

**ALS** Recommender displays a substantial interest in serendipity. Relevant new products score is the higher than any other implemented solution. As any “complex” model, the 2-step process prevents from sticking to every change in products and customers data. The scalable first to comes to counterbalance this finding. Finally, cold start issue is particularly problematic in such complex models. This collaborative filtering model is built as a whole, benefiting from every customer and product. When inserting a new customer or a new item, it requires time to collect useful interactions. Before consistent data is collected, weight of such customers and products are too low to be sufficiently considered in the final model.
Chapter 17

Conclusion

In retail industry, providing customized services is a necessity in order to develop customer loyalty. Customization may take the form of recommended purchase carts. Such carts should relay past behavior, as methods based on pure descriptive statistics do, but also provide serendipity, as matrix factorization based models enable. Merging both expectations seems to be the key for an optimized experience. That is what Neighborhood-Based models permit, especially when relying on items similarity.

Among the five implemented models - Random, Average, k-Nearest-Neighbors, Item-to-Item and Alternating Least Squares, two stand out. ALS splits both concepts of customers and products in a multidimensional space and thus decorrelates the two entities. As a consequence, unknown relations between customers and products can be unveiled and new products can be pushed. As evidenced by our current offline analysis, almost 1 predicted product out of 25 is an unseen yet effectively bought item. On the other hand, Item-to-Item shows balanced yet relevant results. As the model relies on similarity, we are able to explain recommendations as group of close products. Furthermore, Item-to-Item is intrinsically and only built on items information. As a consequence, the difficulty to recommend items to new customers, known as the Cold Start Issue, does not concern this specific model.

Incorporating such solutions into large actors of retail industry would have double benefits that are not highlighted at their true value. First, the more entities (products and customers), the more relevant. Collaborative filtering methods permit to expose unpalpable relations. Second, adding a constant and reciprocal feedback between system that provides recommendations and customers that obtain them would multiply performances. Recommender system evolves with customers reactions and serendipity is accurately controlled.

Not only millions of customers from several major retailers could benefit from such customization but also some actors of e-commerce. Provided they share similarities with retail such as large panel of products and customers, high frequency of purchase and multiple-products purchases, we could transpose the developed solution to e-commerce. Platform providers such as Hybris, Woocommerce or Magento...
CHAPTER 17. CONCLUSION

gather such candidates, and by easing plugin integration, they enable our solution to potentially concern additional millions of customers.
Bibliography


