KTH Computer Science
and Communication

Using text features of E-commerce products in recommendation systems

Evaluating vector representations of text features and using them in content-based filtering

BURE NOREUS

Master's Thesis at CSC
Supervisor: Pawel Herman
Examiner: Olov Engwall

2016-06-06

Abstract

Recommendation systems have great business value in all sorts of digital business. A content-based E-commerce recommendation system has been created. It has been analyzed if the text content in E-commerce products are sufficient for generating good recommendations. Different ways of vectorizing products and generating recommendations from those vectors have been evaluated. For product embedding, Bag-of-clusters and Bag-of-words have been tested. A novel way of performing recommendations called Local Product Clustering has been proposed.

It was found that Bag-of-words weighted with the term frequency-inverse document frequency scheme was the best way to represent the text features of the products. The Local Product Clustering was the best method for generating recommendations.The system has a precision of 3.24% and a recall of 9.89% for exact products. It was argued this recommendation task was very hard to evaluate and that precision and recall did not give a just interpretation of the qualitative results, which looked very promising.

Sammanfattning

Recommendationssystem har stort affärsvärde i många typer av digitala företag. Ett rekommendationssystem av typen *content-based filtering* för E-handel har skapats. Det har analyserats huruvida det är möjligt att använda enbart produkttexten hos E-handelsprodukterna för att skapa bra rekommendationer. Textkvantifieringsmetoderna Bag-of-words och Bag-of-clusters har undersökts för att vektorisera produkttexterna. Ett nytt sätt att generera rekommendationer, Local Product Clustering, har tagits fram.

Det upptäcktes att Bag-of-words med *term frequency-inverse document frequency*-vikter var den bästa kvantifieringsmetoden för produkttexten. Local product clustering var också den bästa metoden för att generera rekommendationerna. Systemet hade en precision på 3.24% och en recall på 9.89%. Det argumenterades att detta rekommendationsproblem var mycket svårt att utvärdera och att precision och recall inte gav en rättvis bild av de kvalitativa resultaten, som såg mycket lovande ut.

Contents

Acknowledgements

This thesis was done at the Swedish start-up Tipser. I would like to thank Jonas Sjöstedt, the CEO of Tipser of giving me the opportunity to work with his company. I would also like to thank As.Prof. Pawel Herman for his invaluable expertise and guidance during my master's thesis.

Chapter 1

Introduction

Since the late 90s and early 00s, Internet companies have sought to give their users a more personalized browsing experience. One way to do this is through a recommendation system. A recommendation system tries to understand what content individual users like and present them with relevant content. This has had many areas of application: Spotify and music, Amazon and books, YouTube and videos, Google and ads, Netflix and television. The examples are too many to enumerate. Today, it is hard to see E-commerce companies that do not have a "similar products" or an "other peope also liked" feature. It is hardly a surprise, considering how much content consumption increases thanks to recommendations. Reportedly, Netflix serves 75% of their videos from recommendations [Amatriain and Basilico, 2012]. YouTube produces a majority of their views from recommendations [Renjie Zhou, 2010].

Recommendation systems come from the field of Information Retrieval and were originally a subclass of information filtering systems. In 1992, Xerox built the first system using patterns in user data to make recommendations, also known as collaborative filtering.[Goldberg et al., 1992]. However, this system was not a recommendation system in the modern sense.

In 1994, the first recommendation system using user ratings was published by GroupLens, a research lab at the University of Minnesota [Resnick et al., 1994]. GroupLens has since become an important institution for research on recommendation system. It released MovieLens in 1997, the first ever recommendation system for movies. MovieLens is a non-commercial recommendation system built to advance research. The datasets from the system is publicly available to download. Those datasets are still often used as a benchmark in recommendation systems research papers.

Amazon is one of the pioneers of commercial recommendation systems. In 2001, Amazon issued a patent for their item-to-item collaborative filtering system. An-

other important milestone is the Netflix prize [Netflix, 2006]. In 2006, Netflix released a dataset of 100 480 507 ratings that 480 189 users gave to 17 770 movies. The competitors were asked to predict the users' ratings on 2 817 131 data points. The best team were given a $1M prize reward.

1.1 Problem

A common way to perform recommendations is to look for patterns in the user data (collaborative filtering). Unfortunately, there is not a lot of user data available in this case. It is difficult to recommend items that we have no user data on. This is known as the cold start problem [Lika et al., 2014]. Instead, the meta data of the items themselves can be used to produce recommendation similar to the items that the user has already picked (content-based filtering). The items being recommended are E-commerce products. To compute similarity between products, they need to quantified.

The problem here is to create a vector representation of E-commerce products and using that representation to generate relevant recommendations. This thesis investigates if it is possible to quantify products based on text features of products such as name, categories and description and make recommendations therefrom. The research question to be studied is *whether textual product features provide enough information to perform a successful content-based filtering.* The objective to answer this question is analyzing text vectorization techniques and analyzing recommendation strategies given those text vectors.

This problem is solved as a case study at a Swedish startup that has this very problem.

1.2 Tipser

Tipser, a Swedish E-commerce startup, wants to harvest the fruits of personalization as well. Tipser offers a platform for distributed E-commerce. That is, they let E-commerce companies distribute their sales to other websites. This is done via a convenient widget. Let's say a blogger wants to sell items on his/her website. That blogger could then easily copy+paste the Tipser widget onto his/her website and select items to sell. The items come from a long range of E-commerce companies that Tipser is collaborating with. Users create so called "collections" of items that the user's visitors can view. A concrete example of this is the following: Let's say Earl is running a E-commerce site selling beer and Buddy is running a blog about wine and beverages. If Earl is connected to Tipser, Buddy could put Earl's products on his blog via a convenient widget. This would let Buddy sell Earl's beer without having to deal with orders, shipping, warehousing etc. Buddy is only playing the role of a sales channel.

It is in Tipser's best interest that the users build collections and sell more items, since a commission is charged per sale. The catalogue from which user's can select items is huge. Tipser therefore believes that is hard for users to find what they like. To facilitate their selection, Tipser wants to show "These items are suggested for your collection"-style recommendations.

User's create "collections" of items that they like by browsing the Tipser catalogue. Usually, those collections have some underlying theme, e.g "Autumn Essentials", "For the beach" or "Living room furniture" etc. Tipser wants to provide suggested products that fit well into the collections. In other words, the system should predict which items a user is going to chose, before they do. Furthermore, the product catalogue is very large and users should not be inconvenienced by browsing a vast array of items to find what they are looking for.

1.3 Limitations

The only product features used were product name, product description and product categories. The methods used to vectorize those text features do not preserve the word order, which destroys information contained in the language. No collaborative filtering models are used, because of the lack of user data. Only neighborhood-based models are used to create the recommmendations. Hyperparameters are not exhaustively searched due to the high cost of cloud computing.

1.4 Ethical and Societal aspects

The most pressing ethical issue is privacy. This is true for every type of recommendation system, not only product recommendations. Google and Facebook have certainly received their fair share of criticism for recording implicit user data and using it to recommend ads.

Users are often critical towards models that use data the users have not given explicit consent for. E.g browsing history or purchases on other sites. The data used here is what products that a user has picked for their collection. This data should not be controversial, since the concept of collections could not work without data on what items the collections contain. In other words, if that data was not recorded, collections could not be built in the first place. Although the technology developed here could potentially be trained with "controversial" data, in this thesis it is not.

The purpose of recommendation systems in general is to make user consume more of whatever the system is recommending. YouTube recommends videos so that

people watch more ads, Spotify wants users to consume more music, etc. In the case of E-commerce, the goal is to make users purchase more products. One could argue that recommendation systems make the shopping experience smoother and compare this to how other inventions in retail and commerce have made people's lives easier. On the other hand, one could argue that recommendation systems are simply fueling consumerism and selling people goods that they don't really need.

Chapter 2

Background

2.1 Collaborative recommendations

The first technique is called collaborative filtering [Ochi, 2010]. This technique is based on the assumption that there is overlap in the interests of users. That is, a given user's interest for a given item is likely to be the same as other users with similar interests in other items. Consider figure 2.1. Randall likes pizza, salad and soda. Bubba likes pizza and salad but it is unknown whether Bubba likes soda or not. We can predict that Bubba likes soda, since collaborative filtering is based on the assumption that users' interests overlap.

Collaborative filtering does not take into account the products themselves. That is, our assumption of Bubba's interest in soda is only based on Randall liking the same products as him, not what those products actually are.

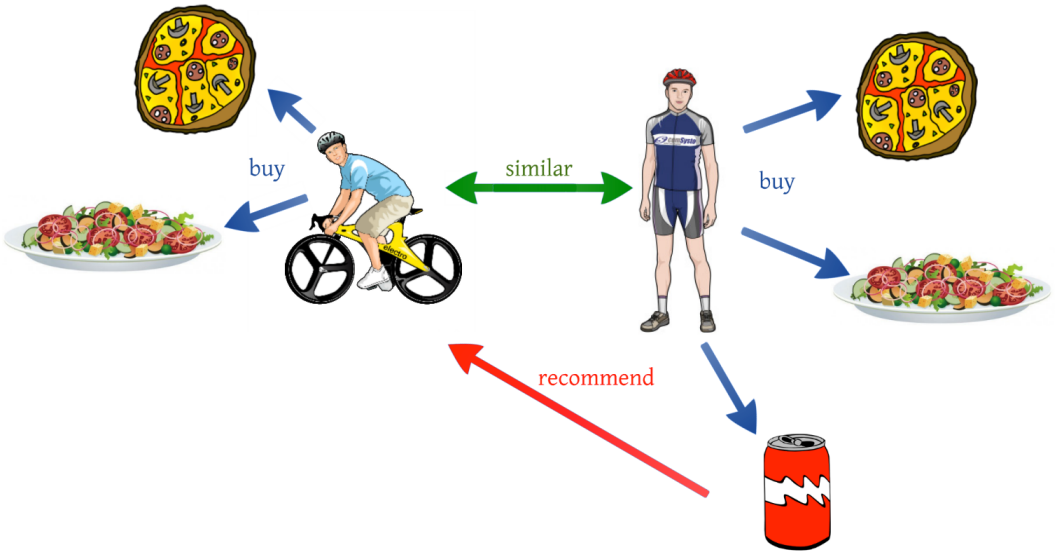

Figure 2.1: Visualization of collaborative filtering

The Netflix prize [Netflix, 2006] was a collaborative filtering contest. The training data set consisted of quadruples of the type *<movie id, user id, date of rating, rating>*. No contestants were able to see what actual movies were rated, only an anonymous id. This is a fundamental principle of collaborative filtering: the underlying content is not important.

The cornerstone of collaborative filtering is the item-user matrix. This matrix contains ratings made by users(rows) on items(columns). Thus, an item is defined by how users rated that item. Conversely, a user is defined by what items that user has rated. A rating may be ordinal (1 to 5 stars), binary (click/no click) or something else. Naturally, this matrix is very sparse since most users only interact with a few items.

According to [Lee et al., 2012], collaborative filtering can be divided into two approaches: neighborhood-based models and model-based models. When predicting the rating on an item by a user, neighborhood-based models hold the item-user matrix in memory and perform a neighbor-search to see how similar users rate that item. Model-based models fit a machine learning model to the item-user matrix and use that model to predict ratings.
The Tipser dataset contains 84 thousand binary(!) ratings on 230 thousands products from 4160 users. Compare this to the Netflix and MovieLens datasets: The Netflix and MovieLens datasets have a user/product ratio of 27 and 5, respectively. For Tipser, this ratio 0.02. Moreover, Netflix and MovieLens have a ratings/user ratio of 209 and 144, respectively. This ratio is only 20 for Tipser. Obviously, there is a huge lack of user data. This problem is known as the Cold start problem

[Lika et al., 2014]. It is very difficult to recommend a product that no user has liked yet.

2.2 Content-based recommendations

The second paradigm is called content-based filtering. This technique is entirely dependent on what the products are. It is based on the assumption that a user who likes a certain product is likely to like products similar to that product, as illustrated in figure 2.2). Randy likes soda. Because soda has similar features to duff beer, Randy will probably like duff beer.

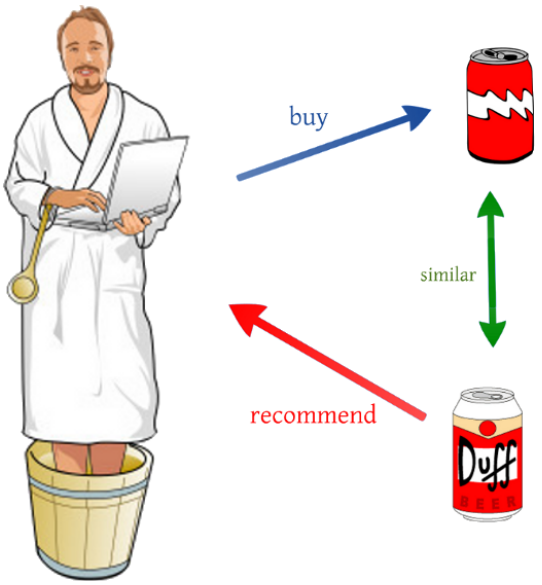

Figure 2.2: Visualization of content-based filtering

Accordingly to a state-of-the-art analysis [Lops et al., 2011], there are a three component that occur in most content-based recommendation systems:

2.2.1 Content analyser

Items in a recommendation system are mostly made up of unstructured information (text, music, video). The content analyser's job is to extract features from the content and create a structured representation of the items. A structured representation usually means a vector. There are no general state of the art models for the content analyser, because it entirely depends on the content and items at hand. The items in Tipser's case have semi-structured text features.

Bag-of-words

Other authors have successfully represented text features in recommendation systems with the Bag-of-Words model [Van Meteren and Van Someren, 2000], [Lops et al., 2011] and [Pazzani and Billsus, 2007]. Although the origins of Bag-of-words is not perfectly clear, early references to the approach can be found in [Harris, 1954]. Bag-of-words is a method of representing text as a fixed-length vector. A text is represented as a multiset of words. That is, a document is represented by its words and their counts. For example, consider the two following short text documents:

1. "I am not an apple, I am a pear"
2. "I might be an apple"

The documents would represented by the following vectors

$$\begin{pmatrix} I \\ am \\ not \\ an \\ a \\ apple \\ pear \\ might \\ be \end{pmatrix} D_1 = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} D_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{2.1}$$

Naturally, when representing many documents, the vectors' size will be the number of unique words in all documents. Thus, most entries in the vectors will be zero.

Term frequency weights

In text mining, using the raw term frequency is often not satisfactory since common words will have very high term frequencies. An often used method to prevent this is a weight scheme called term frequency-inverse document frequency or tf-idf for short. The term frequency and inverse document frequency statistics were invented by [Luhn, 1957] and [Jones, 1972] respectively. It it used by [Van Meteren and Van Someren, 2000], [Lops et al., 2011], [Pazzani and Billsus, 2007], for example.

In the original bag-of-words model, each vector cell is simply the number of occurrences of that word in the document. In the tf-idf model, each vector cell is a product of two metrics.

Term frequency is the number of occurrences of a word in a given document. Inverse document frequency is the natural logarithm of total number of documents

divided by the number of documents in which the term occurs. The purpose of this weighting is to discount words that are very frequent across all documents.

Bag-of-clusters

The bag-of-clusters model is a twist on the bag-of-words model. I was unable to find a case in which it was applied to recommendations, but researchers has used it in other domains such as movie review analysis [Kaggle, 2014], document categorization [Yanhong Yuan, 2014] and plagiarism detection [Duhaime, 2015]. In the bag-of-words representation, every word in the vocabulary has its own vector cell. That is, the vector length is the same as the size of the vocabulary. In the case of Tipser's product text features, the vocabulary has some tens of thousands of unique words and the text features do not have a lot of words. A bag-of-words representation of the product's text features will thus be extremely sparse.

Instead of using a bad-of-words representation, this thesis proposes a bag-of-clusters representation. First, all words in the vocabulary are mapped to vectors via the word2vec algorithm. The bag-of-clusters is created by clustering the vocabulary of the tens of thousands of unique words into K clusters. A bag-of-words model creates a vector in which each cell represents the number of occurrences for a specific word. A bag-of-clusters model, on the other hand, creates a vector in which each cell represents the number of occurrences of words belonging to a specific cluster. After creating the vectors, they are weighted with the tf-idf scheme.

This means that instead of creating a vector of W dimensions, where W is the number of unique words in the vocabulary, the bag-of-clusters model creates a vector of K dimensions, where K is the number of clusters.($K << W$) Naturally, this approach requires a high quality clustering. If words that are not semantically similar end up in the same cluster, this approach does not make a lot of sense. Getting a good quality clustering really depends on two things: An appropriate clustering algorithm and good word vectors.

2.2.2 Profile learner

After the content has been structurized, profiles are created for every user [Lops et al., 2011]. A profile is a model of a user's interest. Every item that a user has rated is collected and from them a profile is learned. This can be done in many different ways:

[Ng, 2012] suggests an approach based on optimization. The items are assumed to have a clearly defined feature vector $\vec{x}$. He describes a way to learn a profile vector $\vec{\theta}$ and a way to use that vector. To recommend items to a user, a rating for a given item is predicted by multiplying the user profile vector with the item vector as such: $\vec{\theta}^T\vec{x}$. After predicting a user's rating of all the items, the items with the highest rating can be shown to the user. To learn a user's profile vector, the

following optimization problem is solved:

$$\underset{\vec{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} (\vec{\theta}_i^T \vec{x}_i - y_i)^2 \tag{2.2}$$

Where N is the number of items and y_i is the user's real rating for the i:th item. The only rating given by Tipser' users is a binary one. Furthermore, Tipser users have only "rated" a tiny fraction of all items. This means that such an optimization would likely lead to a model that makes all-zeros predictions.

[Van Meteren and Van Someren, 2000] describes a profile learner for binary recommendation systems. The profile is calculated by simply doing a weighted averaged of all the item vectors that the user has rated positively. The average is weighted down by the time passed since rating, so that items rated a long time ago are discounted. Recommendations are generated by calculating similarity between items and the profile vectors and recommending the most similar items.

[Lops et al., 2011] suggest an approach for recommendation where items are rated as positive/negative. The paper suggests looking at recommendations as a classification problem. A machine learning classifier is trained for every user. This classifier can then later to be used to predict which items the user will find positive/negative. Classifiers such as naive bayes, SVM and ANN are suggested. In Tipser's data set, each colletion consists of 20 items on average. To train a classifier on such a small dataset would lead to huge overfitting problems. Furthermore, Tipser does not have any clear "liked" or "disliked" labels. Chosing an item suggests that a user likes that item, but not choosing an item does not mean that a user dislikes that item. Indeed, most items are never chosen by a given user.

2.2.3 Filtering component

When a user profile has been learned, the filtering components filters out items that are irrelevant for the user [Lops et al., 2011]. The filtering component is tightly linked to the profile learner and it is hard to give a general example of filtering components. If the profile learner is a classification model, the filtering component will be the loop running through the dataset and classifying unseen items. If the profile learner is a vector average, the filtering compoenent will be the loop running through the dataset and computing item similarities. Later it shall be shown that sometimes the filtering component and the profile learner can be the same thing.

Chapter 3

Method

This chapter will detail the path from product to recommendation. Figure 3.1 visualizes every part of the recommendation system. Features are first represented and then aggregated together into one representation. Many products then form one profile and this profile is used to create recommendations, which are to be evaluated.

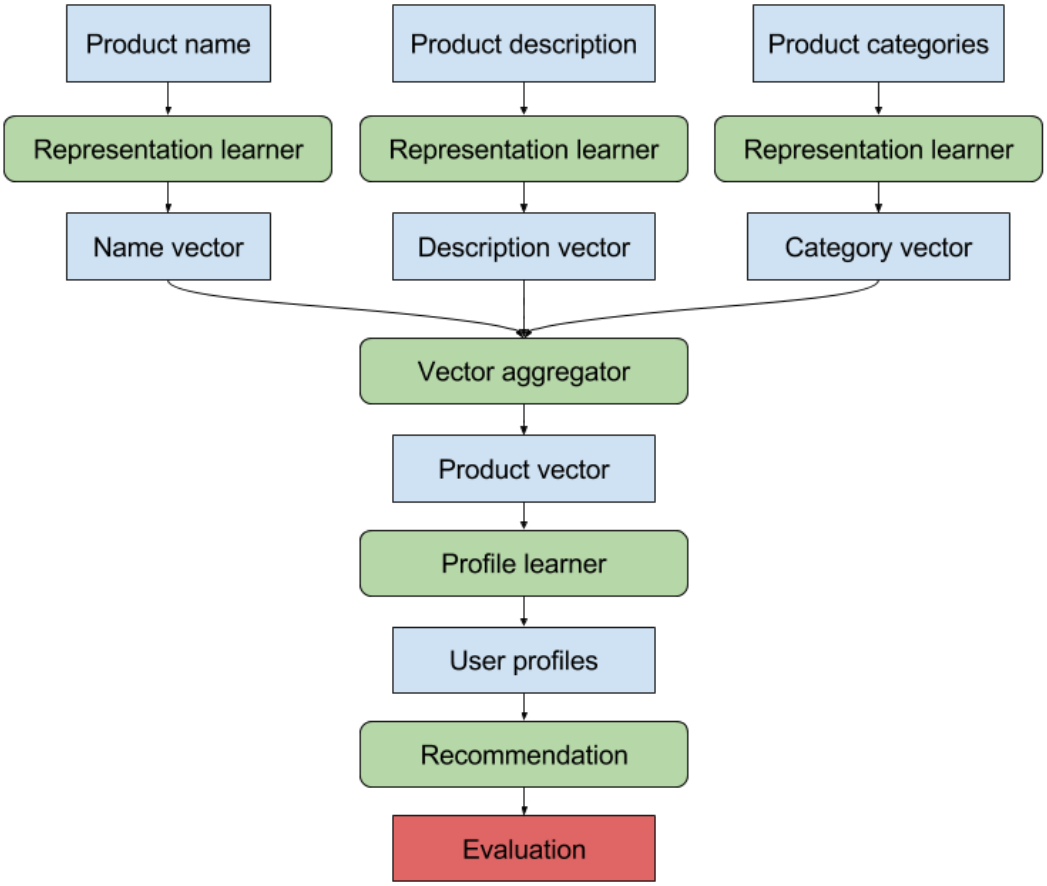

Figure 3.1: The machine learning pipeline used to create recommendations. Blue boxes are data sets, green boxes are procedures and red boxes are evaluations.

3.1 Representation learner

After the text features have been turned into sets of word vectors, they are passed to a representation learner [Figure 3.1]. Its task is to turn a vector set of varying size into a single vector of fixed length. This section evaluates the different approaches.

3.1.1 Bag-of-words

One major challenge when trying to represent text features is that they do not have a fixed length. Bag-of-Word is commonly employed to solve this problem, as explained in the background in this thesis, the bag-of-words model is used as a performance benchmark.

3.1.2 Bag-of-clusters

To create the word clusters in bag-of-clusters, word2vec is used. Invented by Tomas Mikolov, word2vec is a word embedding algorithm. It maps words to fixed-length vectors. Given a large amount of training data, the vectors capture semantic similarity very well when using the cosine similarity measure. That is, words with high cosine similarity also have high semantic similarity. For example, the vector for the word "Paris" is very close to the vector for the word "France", but not very close to the word "Doorknob".[Mikolov, 2013]

All that is needed is a very big corpus of sentences from a given language. Since no pretrained word2vec model was found in Swedish, one was trained. The model was trained on a big dataset made up of a wide array of text corpuses. The dataset that was used is approximately 5.8 billion words. It consists of the entire Swedish wikipedia, Swedish blogs from 2006 to 2014, subforums of the Flashback internet forum and the Familjeliv forum. The corpuses are listed in more detail in the appendix. The vector length is the default 100.

To cluster the vectors, K-means was used. K-means is a very popular clustering algorithm first proposed by Stuart Lloyd in the 50s. It was first published in 1982 [Lloyd, 1982]. The algorithm clusters points into a predefined number of clusters. The algorithm clusters data by iteratively changing cluster centers. Cluster centers are points in space that are initialized randomly and updated such that the following loss function is minimized:

$$L = \sum_{i=1}^{k} \sum_{x \in S_i} ||\vec{x} - \vec{\mu}_i||^2 \tag{3.1}$$

Where k is the number of clusters, S_i is the set of products that belong to cluster i and $\vec{\mu}_i$ is the mean of the points in cluster i. To find this minimum, the following steps are performed until convergence.

1. For each point, calculate the distance to every cluster center. The point is assigned to the cluster with the closest centroid.

2. Calculate new cluster centroids by averaging every point belonging to each cluster.

The algorithm is guaranteed to find a local minimum, but finding the global minimum is NP-hard. [Bottou and Bengio, 1995]

3.2 Vector aggregator

To create a final vector for the entire product, the vectors created by each feature is aggregated. To do this, two approaches are evaluated.

3.2.1 Weighted concatenation

Since each feature vector has its own size, the most simple aggregation is to concatenate the vectors. In this approach, each feature vector is weighted by multiplying every cell in that vector with a feature weight. The weights (name weight, category weight and description weight) are optimized with the Nelder-Mead method [Nelder and Mead, 1965]

3.2.2 Weighted sum

It is not possible to directly sum the feature vectors, since they have different clusters in their bag-of-clusters representation. To circumvent this problem, a vocabulary is created by taking the union of all feature vocabularies. A bag-of-clusters is created by clustering the words of the big vocabulary. Thereafter, the feature vectors have the same size and same clusters and can thus be summed. The vectors are weighted when creating this sum. The optimal weights are found with the Nelder-Mead method.

3.2.3 Nelder-Mead algorithm

The Nelder-Mead method is an optimization algorithm that can find the minimum of nonlinear functions for which no derivatives can be found [Nelder and Mead, 1965]. The main component of the method is a geometric shape called "simplex". A simplex is a generalization of a triangle. It's composed of n+1 points where n is the dimensionality of the optimization problem. Each point in the simplex is a potential solution candidate. In each iteration, the worst point is replaced by a new point calculated with the other points. The algorithm iterates until a local minimum is found.

3.3 Profile learner and Recommendation

To verify the quality of the product vectors, they need to be put to the test in their final application, i.e. recommendations. Creating content-based recommendations from product vectors can be done in many ways. Vector averaging, Top-X and Local clustering are evaluated in this thesis.

3.3.1 Vector average

The most simple user profile is the average of all product vectors that the user has liked. This operation is very cheap. Calculating the user profiles is $\mathcal{O}(MJ)$ where M is the number of user profiles and J is the average number of products per user. After calculating the profile vectors, the recommendations are simply the X closest product vectors to the profile vector. Creating the recommendations is therefore $\mathcal{O}(MN)$ where M is the number of users and N is the number of products.

3.3.2 Top-X

Top-X is to recommend the X closest products to each one of the products that the user has liked. During evaluation, the average number of input items is 10. The system recommends at most 30 items. If the number of input items are fewer than 30, each item will be randomly sampled without replacement to contribute with one recommendation. If all items have been sampled and there is still less recommendations than 30, the process is repeated. This approach is $\mathcal{O}(JMN)$ where J is the number of products per user used for recommendations. M is the number of users and N the number of products.

3.3.3 Local Clustering

Here an approach called Local Product Clustering is proposed. First cluster the products vectors into product clusters. The idea is that the product clusters will form meaningful "product groups".

Clustering is done locally in every user's product collection. That is, one clustering is performed for each user. Every clustering process is performed independently. The number of clusters depends on how many products a user has chosen. It is chosen so that the average number of products per cluster PPC is constant. That is, PPC is a hyperparameter for this model.

After clustering, the centroids of the clusters are used as representations of "themes" in the items that the users have picked. Recommendations are created by retrieving the X products closest to the centroid vectors.

For this approach to work, the clustering quality must be high. That is to say, the products inside each given cluster must be very similar. One cluster could solely contain mens swimwear, another could contain wooden tables and so forth. If these clusters are very noisy, i.e they contain just a random set of products, this approach would not work well.

K-means is used to perform the clustering here. To compare vectors, cosine measure is used instead of the more common euclidean distance.

3.4 Evaluation

To measure the recommendation performance quantitatively, the following prediction problem has been formulated:

Given half of the items that a user has picked, can the other half of the items be predicted? The algorithm is given half of the items that a user has picked and is asked to provide the 30 most relevant recommendations. The number 30 is selected

for UI/UX-purposes, but is arbitrary from an algorithmic standpoint. It is however needed to be sufficiently large, otherwise it is unlikely that the algorithm receives any true positives at all. To evaluate the 30 recommendations precision and recall are used. They are calculated as following [Gunawardana and Shani, 2009]:

$$Precision = \frac{|True\ positives|}{|True\ positives| + |False\ positives|} = \frac{|All\ True\ Positives|}{30 * |Users|} \quad (3.2)$$

$$Recall = \frac{|True\ positives|}{|True\ positives| + |False\ Negatives|} = \frac{|All\ true\ positives|}{|All\ predictable\ items|} \quad (3.3)$$

Where $|True\ positives|$ is the number of recommended items that were actually picked by the user, $|False\ positives|$ is the number of items that were recommended but were not picked by the user and $|False\ negatives|$ is the number of items that were actually picked by the user but were not recommended.

$$|True\ positives| = Number\ of\ recommended\ items\ that\ were\ actually\ picked\ by\ user \quad (3.4)$$

Choice of evaluation function
In environments where items are ranked (e.g 1-5 stars) it is common to try to predict that exact rating. In such cases, it is common to use a standard error measure like Mean Squared Error. The best example of this is the Netflix competition, which used Root Mean Squared Error [Netflix, 2006].

A information retrieval measure has been used instead, since there is no intuitive rating on Tipser's items. I used precision and recall because it is a common evaluation metric in information retrieval and because a similar metric, Mean Average Precision, is often used in information retrieval competitions.[Kaggle, 2016]

Selecting a evaluation metric for this task was not trivial. The difficulties of evaluating this problem is discussed in the Discussion chapter.

3.5 Hyperparameter selection

To find the best final recommendation performance, every component of the machine learning pipeline has been analyzed on its own. This section describes the experiments and hyperparameters tested.

3.5.1 Representation learner

Two representation models have been used: bag-of-words with tf-idf weights and bag-of-clusters with tf-idf weights. Bag-of-words does not have any hyperparameters. Bag-of-clusters has one hyperparameter: The number of clusters, i.e the final dimensionality of the vectors. In the first experiment, each text feature (name,description,categories) is evaluated on its own. That is, the product vector is composed of only one feature at a time.

Number of clusters

For the different features, the following number of clusters were tested:

- Name: X (10% of data points),Y (35% of data points) and Z (50% of data points)
- Description: X (10% of data points),Y (35% of data points) and Z (50% of data points)
- Categories: X (10% of data points),Y (35% of data points) and Z (50% of data points)

10% was chosen because [Duhaime, 2015] had success with this cluster concentration when solving a problem very similar to this one. In the Swedish language, it does not seem right that every word has on average 10 synonyms. Therefore, a higher number of clusters were chosen as well: 35% and 50%.

3.5.2 Vector aggregator

The weights used in the vector aggregation are optimized using the Nelder-Mead method. This method does have hyperparameters, but the default parameters were not changed. ($\alpha = 1, \gamma = 2, \rho = 1/2, \sigma = 1/2$)

3.5.3 Local clustering

This approach has one hyperparameter: The number of local clusters in the set of items that each user has picked. Since the number of items that the users have picked k is not fixed, neither should the number of clusters be. The number of clusters were set individually so that the PPC was fixed. The following values for PPC were tested: 2,3,4 & 5. Less than 2 points per cluster does not make any sense and above 5 points per cluster is a bit much considering the fact that the input data was on average 10 points large.

Chapter 4

Results

4.1 Name, description and categories

These are some statistics about the text features: Product names are on average 3.64 words. The categories are usually one or two words and each product has 1-4 categories. On average, each product has 3.38 category words. The description is on average 34 words long. These numbers are interesting because it means that one should be able to retrieve a lot of information from the description compared to the other features.

The products do not contain a lot of repeated words. On average, the total amount of words is 41.8. 77% of words only occur once per product. 98% of words reoccur less than 5 times per product. For most non-zero cells, the the term frequency will be low.

4.2 Representation learner

In the first experiment, the performances when only using one of the three features were measured. I.e three experiments were performed, where the products were represented only by one feature. The vectorization was performed with Bag-of-words and Bag-of-clusters (K-means 10%, 35% and 50%)

Table 4.1: The performance when using only one of three text features. Since only one feature is used, no aggregation is made. The recommendation strategy is vector averaging.

Feature	Representation	Precision	Recall
Category	Bag-of-Words	1.11%	3.40%
Category	Bag-of-Clusters (10%)	0.61%	1.87%
Category	Bag-of-Clusters (35%)	0.80%	2.44%
Category	Bag-of-Clusters (50%)	0.84%	2.56%
Name	Bag-of-Words	2.29%	6.99%
Name	Bag-of-Clusters (10%)	1.61%	4.93%
Name	Bag-of-Clusters (35%)	2.00%	6.12%
Name	Bag-of-Clusters (50%)	2.02%	6.18%
Description	Bag-of-Words	1.42%	4.33%
Description	Bag-of-Clusters (10%)	0.47%	1.43%
Description	Bag-of-Clusters (35%)	1.00%	3.07%
Description	Bag-of-Clusters (50%)	1.06%	3.24%

From Table 4.1, it can be concluded that bag-of-clusters performed worse than bag-of-words. In theory, the bag-of-words model is sensitive to false negatives. That is, two very similar words are not matched because they aren't identical. To combat this, bag-of-clusters was introduced so that very similar words would map to the same cluster. The reason why this did not work is explained by the observation:

As the cluster size increased (i.e the number of clusters decrease) the performance deteriorated. The more words in each cluster, the more probable it was that the clusters contained outlying words. As the number of outliers increased, the quality of the clusters decreased.

As the number of clusters increase, the performance approaches that of the Bag-of-words. When K approaches the number of unique words, the Bag-of-clusters model itself approximately becomes the bag-of-words. When K is equal to the number of unique words, the models are equivalent. Had K-means not been significantly outperformed by the naive bag-of-words, other clustering approach might have been tested. Word2vec-vectors seemed to not form very good clusters.

4.3 Vector aggregation

In the second experiment, vector summation (summing the three feature vectors together) were compared to vector concatenation (concatenating the feature vectors into one long vector). Bag-of-words was used due to its slightly better performance

in the first experiment. The vectors were weighted before being summed/concatenated. The weights were found with the Nelder-Mead-algorithm. The recommendation strategy used was vector averaging. Table 4.2 and Table 4.3 show the results of these experiments.

4.3.1 Vector concatenation

Table 4.2: The features, represented by Bag-of-words and aggregated by vector concatenation, were weighted and evaluated with vector averaging as recommendation strategy. Nelder-Mead optimization was used to find the best weights. Each row in the table is a convergence point of the optimization. Every optimization run was initialized randomly and independently.

Run #	Precision	Recall	Optimal category weight	Optimal name weight	Optimal description weight
1	2.70%	8.25%	0.6093	0.7395	-0.0980
2	2.70%	8.26%	-1.832	2.205	-0.287
3	2.63%	8.04%	0.625	0.655	0.093
4	2.65%	8.11%	1.488	1.774	0.239
5	2.64%	8.06%	0.241	0.353	-0.046

Two important observations can be made from these results. First, some of the weights are negative. This is fine since the sign is irrelevant in the concatenation approach because the cells in the original feature vectors are independent. For example, if all the cells in the category "part" of the product vector are negative, the similarity will not be affected.

Second, if one looks at the optimal weights learned by the Nelder-Mead algorithm, an interesting pattern emerges. Every run of the algorithm converged to a different value. The proportions between the weights are very similar, however. There is a very simple theoretical explanation for this. When products are compared, they are compared with the cosine measure. The cosine measure is only dependent on the angle between vectors, not the magnitude of the vector. Therefore, it is the direction of the weight vector, not its magnitude, that determines the optimal aggregation.

4.3.2 Using vector summation

Table 4.3: The features, represented by bag-of-words and aggregated by vector sum, were weighted and evaluated with vector averaging as recommendation strategy. Nelder-Mead optimization is used to find the best weights. Each row in the table is a convergence point of the optimization. Every optimization run is initialized randomly and independently

Run #	Precision	Recall	Optimal category weight	Optimal name weight	Optimal description weight
1	2.78%	8.50%	2.6876	1.8612	2.2934
2	2.78%	8.50%	0.7636	0.4150	0.5640
3	2.77%	8.48%	0.8437	0.5269	0.6727
4	2.78%	8.50%	0.9379	0.5126	0.6932
5	2.77%	8.48%	0.8868	0.5522	0.7098

An interesting pattern emerges in the weights. The summation weights converge to different values, but the proportions between them is almost the same.

No conclusion can be made about the superiority of either method, but vector summation was used in the final experiment due to it being faster to calculate.

4.4 Profile learner & Recommendation strategy

Bag-of-words with summation aggregator was used in this final experiment. Five recommendation strategies have been tested. Since some of these have stochastic elements (random initialization), the models were run 5 times and the scores averaged.

Table 4.4: Evaluation of the recommendation strategy. The features category,name and description were aggregated with a weighted sum. The weights, obtained from table 4.3 , were 2.6876,1.8612 & 2.2934 respectively.

Strategy	Precision	Recall
Vector average	2.78%	8.50%
Top-X	3.05%	9.32%
Local clustering(productsPerCluster=2)	3.10%	9.46%
Local clustering(productsPerCluster=3)	3.24%	9.89%
Local clustering(productsPerCluster=4)	3.16%	9.65%
Local clustering(productsPerCluster=5)	3.08%	9.41%

In theory, the local clustering strategy will converge to the Top-X strategy when the number of clusters approaches the number of products. If the number of clusters equals the number of products, each product will be a centroid and this is equivalent to Top-X.

4.5 Interpreting results

First of all, the average collection has 20 items in it. Of those, half are used to create a user profile. The other half are predicted. Now, if only 10 items are predictable and 30 recommendations are made, it is on average theoretically impossible to to get higher precision than 33%. The choice is however justified by the fact it otherwise would be very unlikely to get any true positives. It is therefore expected that this recommendation task will get a lower precision than the average information retrieval task. Nevertheless, a precision of 3.24% may seem low. After all, it means that only 1 in 30 recommends are "good" and the rest are "bad". With the following figures, it is illustrated that precision is a slightly flawed measure for this recommendation task.

Figure 4.1: The top 4 products are the raw input products picked by the user. The bottom products are recommendations given the input products. Local clustering is used to create the recommendations.

Figure 4.2: The top 5 products are the raw input products picked by the user. The bottom products are recommendations given the input products. Local clustering is used to create the recommendations.

As we can see, the recommendations are very relevant to the input. In figure 4.1, the recommendations reflect the user's interest in low-top shoes and workout-leggings. The recommendations in 4.2 capture the user's interest in brown/black leather shoes and natural-colored party tops.

Arguably, most of these recommendations can be considered "good" or "relevant". During evaluation, however, none of the recommendations received any points during evaluation. Thus, both recommendations scored a 0% precision. Consider the following recommendations:

Figure 4.3: The top products are the raw input products picked by the user. The middle products are the recommendations. The bottom products are also picked by the user, but the system does not know it. The system is trying to predict the output products. It manages to predict one of them, marked with a green square. For brevity, two recommendations are omitted.

In figure 4.3, only one of the recommendations match the actual input There are, however, a lot of recommendations that are relevant, even though they do not actually match the output. These relevant but nonmatching products illustrate the difficulty in evaluating this problem. It is therefore hard to interpret the quantitative results, but it is however possible to say that the qualitative results look very promising.

A note on homogenity
Tipser's users do pick items very homogeneously, as shown by the figures. It is not easy to visualize all the item collections that users have created, but in most cases, user do bundle similar products together.

4.6 Comparing results to other studies

The system has a precision of precision of 3.24% and a recall of almost 10%. To put it into context, it is compared to other recommendation tasks. There have been benchmarks of Top-N recommendendations on the Netflix and MovieLens datasets [Cremonesi et al., 2010]. On the MovieLens dataset, the author is able to get a 22% recall and 11% precision when retrieving 5-star movies. On the Netflix dataset, the numbers are 18% and 9% , respectively. Does this mean that Tipser's recommendation system is worse? It is hard to compare. Netflix and MovieLens have many more ratings made per user on a much smaller set of items. Guessing randomly

would give a lower performance on the Tipser recommendation task. Therefore, Tipser's recommendation problem is probably harder to solve.

[Karypis, 2001] does an excellent comparison of top-N recommendation models on a different set of recommendation tasks: movies, e-commerce products, mail-orders, credit card purchases and resumés. For the same model, the best recalls are 25%, 7%, 53%, 18% and 34%, respectively. The point is, it is not meaningful to compare two entirely different recommendation tasks.

Chapter 5

Discussion

5.1 Representation learner

The description feature is surprisingly *not* the best performing feature, when using only one feature to represent the products. One might believe that the description is best, due to its length. In fact, it is not much better than the category feature. The reason for this is likely the description's susceptibility to noise. 98% of words are used less than 5 times. Therefore, the tf-idf weights are going to be mostly determined by the idf-weights. Noise can arise when descriptions contain uncommon (high idf) words that carry no information about the product. Such words would increase the cosine similarity between products that have no semantic similarity. If an E-commerce vendor use similar meta descriptions for many of its products, those may also cause false positives.

The information is more concentrated in the name. The product name is shorter (3.6 words) and often very descriptive. Moreover, the name is very reliable for similarity. For example, if a name contains the word "bikini", it is very likely that the products with similar names are swimwear.

The information is also very concentrated in the categories, but they have another issue: Every E-commerce company has its own taxonomy. A category containing swimwear might called "Beachwear" by one merchant and "Swimsuits" by another. Indeed, the only words shared between categories are common words that do not carry much information. Therefore, it may be hard to find similarities across merchants. Furthermore, product names carry a lot of information that is not conveyed by product categories. For example, product names often mention what brand or specific piece of clothing the product is.

5.2 Vector aggregator

Interestingly, the vector summation dominated the vector concatenation. There are a few disadvantages with the concatenation aggregator. A given word has a three cells in the vector, one for each feature. If product X has a given word in its name and product Y has the same word in its description, there will be no similarity.

Concatenated vectors are also susceptible to noise. If two products with very short descriptions are compared, accidental matches in the name/category can outweigh the lack of similarity in the description. The vector summation method has the opposite problem. If the descriptions are long, any eventual matches in category/name might be drowned.

An interesting observation is that in the weighted summation. The category weight is the highest and the name weight is the lowest. In the concatenation case the name weight is the highest and the description weight is the lowest. Furthermore, when the features are tested independently, the name feature gives the highest performance and the category feature the lowest. How can it be that the weights do not reflect the independent performance of the features?

A probable explanation could be that when the words are combined, the vectors play different roles than they do apart. The description might be weighted high if words occur both in the description and the name. The categories may be weighted high because they counteract the noise in the description.

5.3 Recommendation strategy

5.3.1 Vector averaging

The loss of information when averaging a set of product vectors is likely the main reason for this strategy's low performance. If one takes two vectors of products that are very similar, the result will behave well. On the other hand, if one takes two vastly different products, the average of them might be something different. Consider the two following recommendation situations:

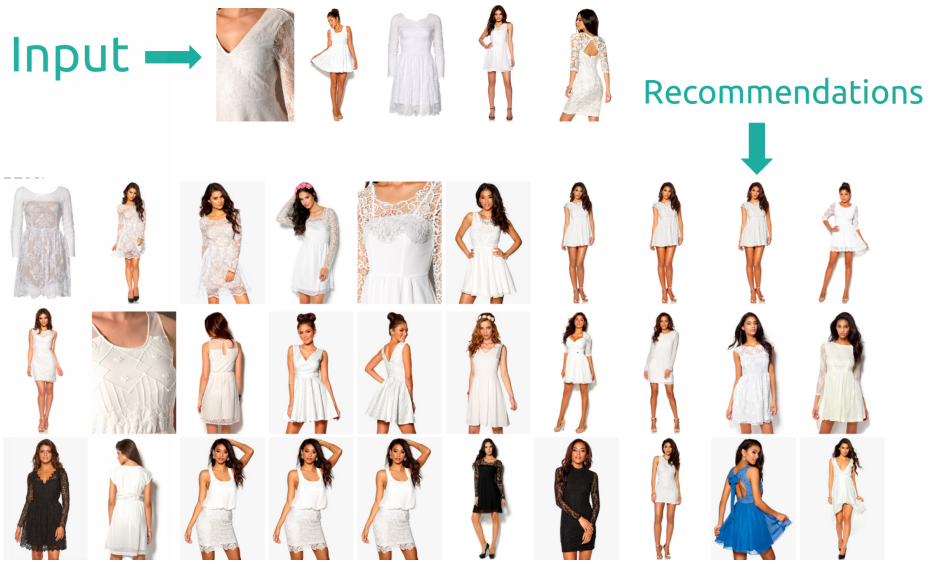

Figure 5.1: The top 4 products are picked by the user. The bottom products are recommendations given the input products. Vector averaging is used to create the recommendations.

Figure 5.2: The top products are picked by the user. The bottom products are recommendations given the input products. Vector averaging is used to create the recommendations.

In figure 5.1 we see that the recommendations reflect the input very well. The input is very homogenous, only consisting of white dresses. In figure 5.2, we can see that a lot of information has been destroyed in the vector averaging. The input contains a diverse array of kitchen appliances, yet the recommendations contain

only cutting boards.

To understand this, one needs to understand the way a tf-idf vector is created:

The reason the tf-idf vector works well because words specific to a certain product type are weighted high and ubiquitous words are weighted low. Consider the figure below.

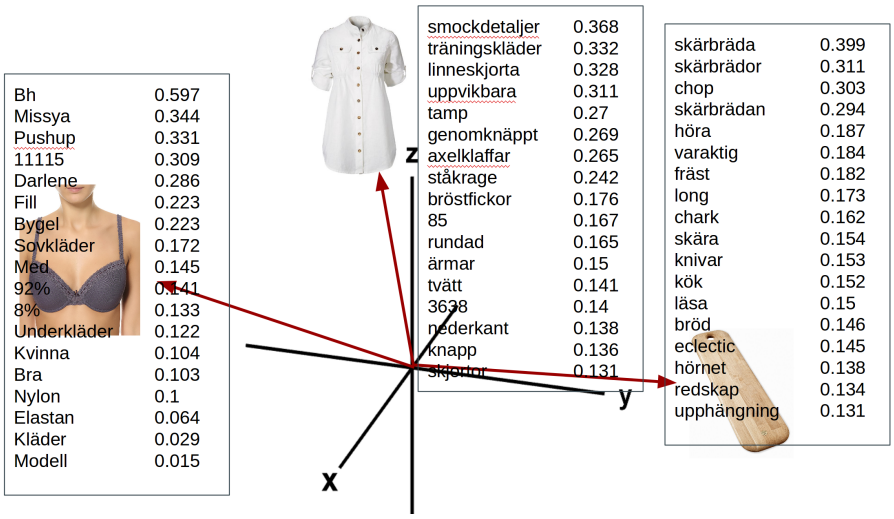

Figure 5.3: The underlying words of three very differentproducts. The boxes contain the words with the highest weights.

When these three vectors are averaged, the following vector is created:

bh	0.342
skärbräda	0.228
smockdetaljer	0.211
missya	0.197
träningskläder	0.19
pushup	0.19
linneskjorta	0.188
uppvikbara	0.178
skärbrädor	0.178
11115	0.177
chop	0.174
skärbrädan	0.169
darlene	0.164
tamp	0.155
genomknäppt	0.154
axelklaffar	0.152
med	0.14
ståkrage	0.139
fill	0.128
bygel	0.128
höra	0.107
varaktig	0.106
fräst	0.104

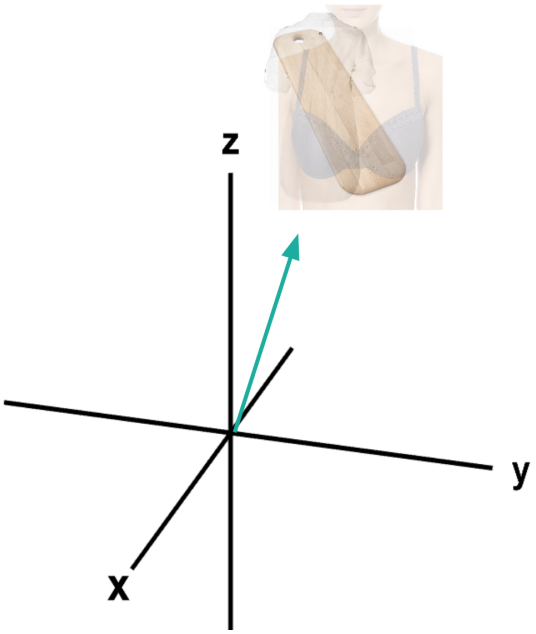

Figure 5.4: After averaging the vectors for a bra, a cutting board and a linen shirt, this vector is created. The box shows the words with the highest weights.

As figure 5.4 shows, the mean vector is comprised of a mixture of words from the constituent vectors. The words from the linen have all but vanished in comparison, since they had smaller weights to begin with. If two products are averaged, the one with higher word weights will outweigh the other. The results of this outweighing can be seen in figure 5.2. A more obvious results is shown in the figure below.

Figure 5.5: The vectors of the top vectors are averaged and through the mean vector the bottom products are recommended.

In figure 5.5 it is clear to see that the linen shirt's influence on the recommendations is gone. The cutting boards are also underrepresented.

When averaging a lot of product vectors noise will be introduced, as seen in 5.2. Indeed, if one would averaging infinitely many product vectors, one would get a word distribution of the language that describes products.

5.3.2 Top-X

A benefit of this approach is not averaging vectors. No information is destroyed via aggregation. On the other hand, no general theme is captured, since recommendations are based on individual products and no profile is created.

5.3.3 Local clustering

This approach does destroy information via aggregation, but much less aggressively than the first approach. In this strategy, recommendations are created by picking nearest neighbors to cluster centroids. K-means centroids are averages of vectors.

The centroids will probably have higher quality than the vector averages of entire users. A user may have rated a diverse array of items, making the averages noisier. Secondly, the vectors in a cluster are likely already close to each other, due to the nature of clustering. Furthermore, the number of clusters is dynamically set so that the average number of products per cluster will be constant. This way,

clusters can be kept small so that centroids do not destroy too much information.

This approach really depends on the quality of the clusters. If clusters are noisy and heterogeneous, averaging them might cause the problem we see in Figure 5.5. Below are some examples of the clustering.

Disclaimer: It is very important to note that these products are clustered by the their text features and not by their images. The images are merely used as they are easier to interpret. Obviously it is desirable that products look relevant because this is what determines the end user's first impression of the recommendations.

Figure 5.6: The black boxes are product clusters made with kmeans.

Figure 5.7: The black boxes are product clusters made with kmeans.

Figure 5.8: The black boxes are product clusters made with kmeans.

Each picture, figure 5.6, figure 5.7, figure 5.8 is the products from separate users.

5.4 Evaluation

It has been shown that the recommendation precision is 3.24%. I argue that the system is not as bad as the precision indicates and that there is an inherent prob-

lem with evaluation with recommendation task. The problem is best described by Figure 4.3. It shows a handful of good-enough recommendations of which only one gives credit in the evaluation. The inherent problem of this recommendation task is the incredible difficulty in defining a evaluation function such that "good-enough" recommendations receive credit.

The system would not receive credit if a user was looking for a blue shirt and the system recommended a shirt in a slightly lighter blue. Likewise, if the system recommends a chair when the user is looking for a blue shirt, it would not receive credit. Arguably, the system should receive some credit when recommending a very similar item to what the user was looking for. For any given item in Tipser's catalogue, there are a dozen of very similar items. Thus, it is likely that the system recommends the "wrong" blue shirt, even if it is recommending blue shirts. Receiving credit from in the evaluation is quite stochastic since there are many "wrong" but very similar items to the "correct" item. [Cremonesi et al., 2010] reduces the variance by putting the products the system is trying to predict (the output) and 1000 other randomly sampled items into one set and searching only that set when generating recommendations. This makes the search set artifically smaller and increases the probability that the "correct" item is picked. This is making the problem unrealistically easy, because in a realistic scenario, one has to search the entire product catalogue. It is also unwise to optimize a system under artificial conditions, as it may not generalize to the production environment.

In section 4.6 it is mentioned that [Karypis, 2001] compared recommendations of movies, e-commerce products, mail-orders, credit card purchases and resumés. Karypis showed how one model can give vastly different recalls depending on the recommendation task at hand. This points out the meaninglessness of comparing the performance between recommendation tasks.

For there to be a meaningful comparison, two things need to hold. First, all recommendation tasks must be equally difficult for a random predictor. If one problem contains less negative answers, obviously that will be easier to solve.

Secondly, the items recommended must be very similar. It does not make any sense to compare music recommendation with clothing recommendation, because they are consumed differently. This is especially important when doing content-based recommendations. The content of songs or movies might be much more difficult to quantify than the content of products. Even product recommendation systems between themselves might be hard to compare if one system has products with great descriptions and the other does not.

Finally, precision and recall should be used as an indicator, not ground truth. Tipser needs to deploy the system live to find out if it is worthwhile. When deploying live, Tipser can see how many items are being selected from recommendations and how

many of those items are later being sold.

Chapter 6

Conclusion

The aim of this thesis was to find out whether textual product features provide enough information to perform a successful content-based filtering. This was done by analyzing text vectorization techniques and analyzing recommendation strategies given those text vectors.

For vectorizing the text features, it was shown that Bag-of-words performed best. Despite having a very well trained word2vec-model (5.8 billion words), the word clusters were found to be too noisy.
It was not possible to determine whether vector summation or vector concatenation was better for aggregating text feature vectors.

It was found that the best way to create recommendations was to cluster each user's products so that the average product per cluster is 3. Thereafter, recommendations were generated by retrieving the products closest to the centroids.

The best model was able to retrieve products with a 3.24% precision and 9.89% recall. Finally it was argued that this recommendation task is very hard to evaluate, making precision and recall misleading. The metrics made it hard to draw any conclusion from the quantitative results, but the qualitative results looked very promising.

Bibliography

[Amatriain and Basilico, 2012] Amatriain, A. and Basilico, J. (2012). Netflix recommendations beyond 5 stars. Available at http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html retrieved 2016-06-01.

[Bottou and Bengio, 1995] Bottou, L. and Bengio, Y. e. a. (1995). Convergence properties of the k-means algorithms. *Advances in neural information processing systems*, pages 585–592.

[Cremonesi et al., 2010] Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM.

[Duhaime, 2015] Duhaime, D. (2015). Clustering semantic vectors with python. Available at http://douglasduhaime.com/blog/clustering-semantic-vectors-with-python retrieved 2016-05-20.

[Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.

[Gunawardana and Shani, 2009] Gunawardana, A. and Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, 10:2935–2962.

[Harris, 1954] Harris, Z. (1954). Distributional structure. *Word*, pages 142–162.

[Jones, 1972] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

[Kaggle, 2014] Kaggle (2014). Bag of words meets bags of popcorn. Available at https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-3-more-fun-with-word-vectors , retrieved 2016-06-01.

[Kaggle, 2016] Kaggle (2016). Mean average precision. Available at https://www.kaggle.com/wiki/MeanAveragePrecision, retrieved 2016-09-29.

[Karypis, 2001] Karypis, G. (2001). Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM.

[Lee et al., 2012] Lee, J., Sun, M., and Lebanon, G. (2012). A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*.

[Lika et al., 2014] Lika, B., Kolomvatsos, K., and Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4, Part 2):2065 – 2073.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Lops et al., 2011] Lops, P., De Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer.

[Luhn, 1957] Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

[Mikolov, 2013] Mikolov, Tomas, e. a. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pages 3111–3119.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

[Netflix, 2006] Netflix (2006). Netflix prize. Available at http://www.netflixprize.com/rules. Retrieved 2016-06-01.

[Ng, 2012] Ng, A. (2012). Content based recommendations. Available at https://www.youtube.com/watch?v=i6u5ykEHSP8 2016-08-08.

[Ochi, 2010] Ochi, Paloma, e. a. (2010). Predictors of user perceptions of web recommender systems: How the basis for generating experience and search product recommendations affects user responses. *International Journal of Human-Computer Studies*, 13(8):472–482.

[Pazzani and Billsus, 2007] Pazzani, M. J. and Billsus, D. (2007). *The Adaptive Web: Methods and Strategies of Web Personalization*, chapter Content-Based Recommendation Systems, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Renjie Zhou, 2010] Renjie Zhou, Samamon Khemmarat, L. G. (2010). The impact of youtube recommendation system on video views. Available at http://conferences.sigcomm.org/imc/2010/papers/p404.pdf , retrieved 2016-05-30.

[Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA. ACM.

[Van Meteren and Van Someren, 2000] Van Meteren, R. and Van Someren, M. (2000). Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pages 47–56.

[Yanhong Yuan, 2014] Yanhong Yuan, Liming He, L. P. Z. H. (2014). A new study based on word2vec and cluster for document categorization. *Journal of Computational Information Systems 10: 21 (2014) 9301–9308.*

Appendix A

Word2vec corpus

The word2vec was trained on the following corpuses, collected by Gothenburg University:

1. https://spraakbanken.gu.se/eng/resource/familjeliv-kansliga
2. https://spraakbanken.gu.se/eng/resource/familjeliv-foralder
3. https://spraakbanken.gu.se/eng/resource/flashback-samhalle
4. https://spraakbanken.gu.se/eng/resource/familjeliv-gravid
5. https://spraakbanken.gu.se/eng/resource/sou
6. https://spraakbanken.gu.se/eng/resource/flashback-politik
7. https://spraakbanken.gu.se/eng/resource/familjeliv-medlem-vantarbarn
8. https://spraakbanken.gu.se/eng/resource/familjeliv-medlem-allmanna
9. https://spraakbanken.gu.se/eng/resource/familjeliv-medlem-foraldrar
10. https://spraakbanken.gu.se/eng/resource/flashback-vetenskap
11. https://spraakbanken.gu.se/eng/resource/familjeliv-allmanna-samhalle
12. https://spraakbanken.gu.se/eng/resource/flashback-kultur
13. https://spraakbanken.gu.se/eng/resource/flashback-dator
14. https://spraakbanken.gu.se/eng/resource/flashback-droger
15. https://spraakbanken.gu.se/eng/resource/flashback-sport
16. https://spraakbanken.gu.se/eng/resource/bloggmix2011
17. https://spraakbanken.gu.se/eng/resource/bloggmix2010

18. https://spraakbanken.gu.se/eng/resource/bloggmix2012
19. https://spraakbanken.gu.se/eng/resource/bloggmix2009
20. https://spraakbanken.gu.se/eng/resource/bloggmix2013
21. https://spraakbanken.gu.se/eng/resource/bloggmix2008
22. https://spraakbanken.gu.se/eng/resource/bloggmix2014
23. https://spraakbanken.gu.se/eng/resource/bloggmix2007
24. https://spraakbanken.gu.se/eng/resource/bloggmix2006
25. https://spraakbanken.gu.se/eng/resource/wikipedia-sv