Building a Sporting Goods Recommendation System

MIKAEL FLODMAN
Building a Sporting Goods Recommendation System

Ett Rekommendationssytem för Sportprodukter

Examensarbete inom datalogi, Civilingenjör Datateknik

Mikael Flodman, mflodman@kth.se

2015-06-08

Handledare: Stefan Arnborg
Examinator: Stefan Arnborg
Uppdragsgivare: Sportamore AB (Publ)
Abstract
This thesis report describes an attempt to build a recommender system for recommending sporting goods in an e-commerce setting, using the customer purchase history as the input dataset. Two input datasets were considered, the item purchases dataset and the item-category dataset. Both the datasets are implicit, that is not explicitly rated by the customer. The data is also very sparse that very few users have purchased more than a handful of the items featured in the dataset. The report describes a method for dealing with both the implicit datasets as well as addressing the problem of sparsity.

The report introduces SVD (Single Value Decomposition) with matrix factorization as a implementation for recommendation systems. Specifically implementations in the Apache Mahout machine learning framework.

Referat
Att rekommendera sportprodukter

Rapporten introducerar SVD (Single Value Decomposition) med matrisfaktorisering som en metod för att implementera rekommendationssystem. Specifikt implementerat med hjälp av maskininlärningsbiblioteket Apache Mahout.
# Table of Contents

Background.................................................................................................................. 3  
Sportamore.................................................................................................................... 3  
Recommendation rationale............................................................................................ 3  
Recommendation Systems............................................................................................. 4  

Uses Cases....................................................................................................................... 5  
Transaction transition page.......................................................................................... 5  
Index page personalization........................................................................................... 5  
Retention......................................................................................................................... 5  
Identification Problem................................................................................................. 5  

Input Data....................................................................................................................... 6  
Explicit vs Implicit data.................................................................................................. 6  
The Data.......................................................................................................................... 6  

Algorithm....................................................................................................................... 8  
What is a recommendation.............................................................................................. 8  
General Recommending................................................................................................. 8  

Common recommendation algorithms........................................................................... 8  
Collaborative Intelligence and Filtering........................................................................... 8  
Collaborative filtering: an overview................................................................................ 9  
Memory-based................................................................................................................ 9  
Model-based CF............................................................................................................... 10  
Matrix factorization, Sparsity and Dimensionality Reduction........................................... 10  
Latent factor models...................................................................................................... 10  
ALS-WR......................................................................................................................... 11  

Implementation and Technologies................................................................................ 13  
Mahout............................................................................................................................ 13  
Why Mahout................................................................................................................... 13  
Algorithms....................................................................................................................... 13  
Recommending with Mahout......................................................................................... 14  
SVDRecommender with ALSWRFactorizer.................................................................... 14  

Evaluation....................................................................................................................... 15  
Hidden and training data............................................................................................... 15  
Mean Rank....................................................................................................................... 15  

Results........................................................................................................................... 16  
Early implementations and tests.................................................................................. 16  
Results with ALS-WR recommender system.................................................................. 16  
Benchmark.................................................................................................................... 16  
Item dataset.................................................................................................................... 17  
Predicting item preference........................................................................................... 17  
Category dataset........................................................................................................... 18  
Predicting item category preference............................................................................ 18  
Benchmark comparison............................................................................................... 18  

Discussion...................................................................................................................... 19  
Popularity Recommender vs ALS-WR recommender..................................................... 19  
Data bias......................................................................................................................... 19  
Which is better............................................................................................................... 19  
One or several Algorithms............................................................................................ 19  
Choosing an algorithm................................................................................................. 19  
Item vs Category predictions....................................................................................... 20  
User experience............................................................................................................. 21  
Cold start problems..................................................................................................... 21  

Conclusion and Further Work....................................................................................... 22  

Literature........................................................................................................................ 23
Background

Sportamore

Sportamore AB is a publicly traded Swedish e-commerce company selling sporting goods to primarily the Nordic markets. The company was founded in 2009 and is active in Sweden, Denmark, Finland and Norway. The site is available under the domains sportamore.se, sportamore.dk, sportamore.fi and sportamore.no.

Recommendation rationale

The basic aim of a recommender system deployed on an e-commerce site is to increase customer conversions (paying transactions) by increasing the relevance of the displayed products with regard to the individual customer.

A case can be made that the sheer volume of products present on a large e-commerce site hinders the customer from finding products of interest. For example in a regular retail store the customer would pass both, the socks, the functional running-wear and the GPS watches on his/her way to and from the wall with the running shoes. Most e-commerce sites try to accomplish the same situation as described above by presenting relevant products after an item is put in to the shopping cart or on up-sell pages displayed before entering the checkout page. An e-commerce site is in some sense both smaller and larger than the traditional retail store, larger in terms of volume of products available and smaller because there is no need to pass through the store to products of interest, therefore limiting exposure. The average time spent on an e-commerce site is also short compared to the regular retail store. The usual approach of of displaying items of interest or items on offer results in an exposure to a small selection of products in a relatively short time frame.

As an example putting a pair of running shoes in the shopping cart on www.sportamore.se will take the customer to a transitional page showing related products, the category relation is governed by a relation matrix and as the item was a pair of running shoes the related products might be be 4-5 of the most popular GPS watches. The selection is small and it is governed by the expected preference of all customers shopping for running shoes, and not the preferences of the particular customer. 4-5 items is not a big selection of the items available and the time allowed to capture customer interest is short.

So the question becomes, how to maximize the customer-to-product relevance in short time frame and with limited presentation area. A recommendation systems is an attempt to answer this question, by trying to predict user interest in an item based on the users previous behavior and the behavior of other similar users.
Recommendation Systems

The general reasons and rationals for recommendation systems were described in the previous section. I will continue with a brief overview of recommendation system techniques. A more involved dive into specific methods chosen for this master thesis, will follow in later sections.

The general scenario in which a recommendation system operates in has a list of $m$ users $u_n = \{ u_1, \ldots, u_m \}$ and a list of $n$ items $i_n = \{ i_1, \ldots, i_n \}$ and each user $u_i$ has a list of $k$ items $u_i = \{ i_1, \ldots, i_k \}$ each relation between a user $u$ and item $i_k$ in the users list of known items is assigned a value based either from the user rating of that item or from the users interaction with that item, purchases, clicks, view and so on. The recommendation systems goal is by using the underlying data recommend items that are relevant to a particular user [1].

There are three general types of recommendation systems. Collaborative Filtering (CF), Content based filtering and Hybrid filtering. [1][2]

Collaborative filtering approaches the problem without general knowledge of how the items relate to one another. The idea is to compare users ratings and/or behaviors towards an item and compare them to that of other users. For example if user $u_a$ buys items $\{ i_1, i_2, i_3 \}$ and user $u_b$ has historically bought $\{ i_1, i_2 \}$, a simplistic CF recommendation system would serve $i_3$ as a likely prediction of what $u_b$ would consider a good recommendation upon his return to the site/store. Challenges with using CF include sparsity, cold start, scalability, synonymy and others [1]. We shall return to Collaborative Filtering later in the text.

Content based filtering instead asks the question how does the item $i_a$ and $i_b$ relate to each other. The method tries to predict a users preference towards and item based on the attributes of that item compared to the attributes of other items that the user as expressed dislike or liking towards. That is to predict how the user will feel about a recommendation of item $i_x$ by comparing its attributes with the attributes of $i_a$ and $i_b$ previously rated by the user. Problems with this approach are among others that it will recommend things that are necessarily related to the item already in high regard by the user and might result in a repetitive result. For example in an e-commerce shop recommendations might be made to the user of items that already has already been purchased. Or at least items very similar to the already owned items, making the recommendation redundant.

Hybrid recommendation systems are efforts to combine the two aforementioned approaches to the problem. By looking at both users ratings and behavior and item attribute similarity to fine tune the recommendation [1].

When choosing either a collaborative, content or hybrid approach for producing recommendations account has to be taken for available data, computation capacity and scalability.
Uses Cases

Transactional transition page
After a user has put an item in the on-line shopping cart the user is presented with item recommendations on an transitional page before moving on to the checkout page. The items on display could benefit from a recommendation system, by displaying products that are statistically relevant to the product placed in the shopping cart. That is products that others have bought along side the chosen product. Personalization is also relevant that is to say the characteristics of the user could influence the recommended items.

Index page personalization
A recommendation system could be used to personalize the index landing page of the site. If the user has been identified and is known by the recommendation system, then top lists and offers presented on the index page could be tailored to the user.

Retention
User information and a recommendation system could be used for tailored retention efforts. Using a recommendation system items could be identified that has a higher probability of attracting the attention of the user. The predicted items could then be communicated in a news letter or other forms of user targeted communication.

Identification Problem
User-based recommender systems need user information in order to recommend items. Recommendations are made based on the users history in the system. Unless the user is identified or if the user is new to the system, then no recommendations can be calculated.

Item-based recommender does not need user identifications but can recommend based on the current users interactions with items. Placing an item in an on-line shopping cart could trigger recommendations of items based on the chosen item. These recommendations would of course be based on the item and not the user. Removing the possibility to tailor the recommendations directly to the user.
Input Data

Explicit vs Implicit data

Most recommender system research concerns itself with explicit data, that is data where users give explicit information on item preference through ratings. This gives the recommender system input on both what the users like and what they dislike, and how much they like a certain item, if the ratings have a scale 1-5 for example. When explicit data is used it is usual to only account for rated user-item pairs and exclude unrated products.

Implicit data is data collecting implicit user interest in an item for example purchases, clicks in result lists, mentions or shares. Implicit data is inherently noisy, we cannot know if the user purchases an item for him/herself, it might be a gift for a friend. And a purchase does not necessarily indicate preference, the user might be unhappy with the purchase.

In Hu, Koren and Volinsky's paper Collaborative Filtering for Implicit Datasets [3] they introduce a way to look at implicit feedback data. Their approach splits the rating $r_{ui}$ by user $u$ of item $i$ into a preference $p$ and a confidence $c$. The preference $p$ is 1 if $r>0$ and 0 if $r=0$.

$$p_{ui} = \begin{cases} 1, & r_{ui} > 0 \\ 0, & r_{ui} = 0 \end{cases}$$

That means that if a user has shown interest in an item then that counts as a preference, to offset the previously discussed problems they add a confidence part $c$ for the user-item pair. The confidence $c_{ui}$ of the preference $p_{ui}$ is calculated as:

$$c_{ui} = 1 + \alpha r_{ui}$$

That gives a starting confidence for all $p_{ui}$ and bumps the confidence for the first recorded favorable user action. There main idea here is that a high value of $r_{ui}$ does not indicate high preference, the most loved item or category could be one that is only bought once. The frequency score of $r_{ui}$ however is indicative of how confident we can be in the indication of preference that value gives.

The Data

The available dataset at the time of this report is purchase history, and as Sportamore is a website selling sporting goods and this type of goods are usually not bought frequently, most users only purchase an specific item once. Add to this that most items are only available for one season that roughly corresponds to a quarter of a year. That means that the user $r_{ui}$ count for most items is 1 or 0, purchased once or not purchased at all. There are of course classes of items that are more frequently bought, mostly basic items such as socks.
In this report I will run tests on two cases. Predicting user item preference based on time purchase frequency and predicting user category preference based on frequency of in item-category purchases. The rational for switching to category is to predict user interest based on the type of item and then rank possible items from a demographic view point. If the recommendation predicts interest in rock climbing shoes. The system would recommend items from the category rock climbing shoes.

The data is structured as the purchase frequency per user and item pair.

User-item pair: (user id, item id, number of purchases )
User-category pair: (user id, category_id, number of in category purchases)
Algorithm

What is a recommendation
The general way that recommendation algorithms treat the recommendations is as predictions of how a user will rate a particular product. The statistical probability that a user will rate a product depending on the user's rating history and the rating history of users with similar tastes. In the scope of this thesis recommendations will be made on the prediction of how likely the user is to buy an item that is presented as a recommendation.

General Recommending
Given the above stated definition is true and workable then recommending items becomes a process which can be described as follows:

- Predict a set of items the user is most likely to buy or rate high.
- Remove any product the user has already bought or is similar enough to be considered redundant.
- Recommend the $n$ products with the highest probability for user approval.

If we switch to training category preferences from an implicit dataset, then the above method is amended as follows.

- Predict a set of item-categories that the user will be interested in buying items from.
- Select items from these categories based on some sort of ranking, perhaps over all popularity. Taking into account attributes of the user.
- Remove any items already bought by the user, or that are similar enough to bought items to be considered redundant.
- Recommend the top $n$ items remaining.

Common recommendation algorithms

Collaborative Intelligence and Filtering
Collective intelligence is defined by Wikipedia as “a shared or group intelligence that emerges from the collaboration and competition of many individuals”.

Collaborative Filtering (also defined by Wikipedia) is – “the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.”

Recommendation systems use collective intelligence to produce intelligent recommendations based
on the group behavior of all known users in the system. The most commonly applied method to achieve this is collaborative filtering [1] [2].

**Collaborative filtering: an overview**

Collaborative Filtering (CF) look at relationships between users and items. If two items $i_x$ and $i_y$ then occur frequently among a set of users, then a CF algorithm will consider these items similar in the sense that a user who has bought $i_x$ is probably interested in item $i_y$. Most Collaborative Filtering methods have been developed to consider explicit feedback [1] [3], that is feedback directly given by the user. Alterations and alternatives are available to account for implicit feedback.

Challenges for Collaborative Filtering are data sparsity, few recorded recommendations by users in a large user and item set [1]. The cold start problem is a variation of the sparsity problem, new items or users will have or have made few recommendations and therefore will not be classified as similar to other items and users.

Scalability is another issue that Collaborative Filtering faces [5] as the number of users and items are often large and resulting in a user-item matrix with very high dimensionality. Dimensionality reduction techniques, such as SVD Singular Value Decomposition can often deal with this and also produce good quality recommendations, at the cost of going through extensive matrix factorization steps [1].

Gray sheep is yet another problem. A gray sheep is a user whose opinions do not consistently agree or disagree with any group of users. Thus making recommending items problematic. Black sheep are users who's opinions do not coincide with anyone, making recommendations impossible [1].

Collaborative filters recommendation engines are usually classified as memory-based, model-based or hybrid [1].

**Memory-based**

Memory-based CF uses the entire, or a sample of, the user-item dataset to generate a prediction for a user's taste. A user is clustered into a group of users with similar interests. By identifying the correct group of similar users for an active user, a prediction of preference can be made based on the preferences of other users in the same group. A similarity is calculated between each pair of users based on a similarity measure. The users are then grouped into a neighborhood determined by either a similarity threshold or a K-nearest neighbors neighborhood where K is the number of users clustered into each group [1].

Advantages of the memory-based collaborative filtering techniques include easy implementation and it's simple to add new data. Disadvantages are prediction performance decreases when data is sparse, cannot recommend for new items and users, and limited scalability for large datasets. [1]

Common similarity measures used for calculating similarity between users are the Pearson correlation and vector cosine similarity [1][2].
Model-based CF

Model-based collaborative filtering uses models calculated or trained ahead of time. Using machine learning or data mining algorithms the system learns to recognize complex patterns. The patterns are then used to predict preferences from new data. The techniques used are among others Bayesian models, clustering models, and dependency networks. Other techniques include classification models if the user ratings are categorical, or regression models such as SVD if the ratings are numerical [1].

Matrix factorization, Sparsity and Dimensionality Reduction

Sparsity is a problem for most standard collaborative filter techniques. Data for e-commerce sites selling a large assortment of items is sparse. Most users have either rated or in implicit case only viewed/purchased a small portion of the total number of items, this results in a sparse item-user matrix. One way to deal with this problem is to employ algorithms that utilize matrix factorization, such as Singular Value Decomposition SVD or Primary Component Analysis PCA [3].

Matrix decomposition or factorization is a dimensionality reduction technique that factorize a matrix into a product of matrices. There exist many different decompositions, different ones finding use among a particular classes of problems.

Latent factor models

Latent factor models in are models that approach the problem of collaborative filtering differently than the discussed neighborhood models. Latent factor models aim to uncover latent features that explain observed ratings. Examples of these models are pLSA[3], neural networks, Latent Dirichlet Allocation and SVD models that based on Singular Value Decomposition of the user-item preference matrix, or observations matrix in the implicit data case.

The basic structure of a SVD model is that each user $u$ is associated with a user-feature vector $\vec{x}_u \in \mathbb{R}^f$, and each item $i$ with an item-features vector $\vec{y}_i \in \mathbb{R}^f$. The prediction in latent factor models is taken by the inner product, i.e., $r_{ui} = \vec{x}_u^T \vec{y}_i$. Most of the SVD methods put forward for explicit datasets suggest modeling only the observed ratings, while avoiding over-fitting by regularizing the model[1]. Such as:

$$\min_{\vec{x}, \vec{y}} \sum_{r_{ui} \text{ is known}} (r_{ui} - \vec{x}_u^T \vec{y}_i)^2 + \lambda (\|\vec{x}_u\|^2 + \|\vec{y}_i\|^2)$$

Here $\lambda$ regularizes the model. Usually parameters are found using stochastic gradient descent.
**ALS-WR**

Alternating Least Squares with Weighted Lambda Regularization or ALS-WR is proposed by Hu, Koren and Volinsky in their paper Collaborative Filtering for Implicit Datasets. It borrows from standard SVD models altering them to fit the implicit case. The name was not coined by the authors but the algorithm is commonly referred to as ALS-WR.

The main idea is to treat observed user interest, i.e. purchases, clicks, views as positive or negative preference associated with a varying degrees of confidence. For example purchasing an item once might be done without preference, but a second purchase indicates preference with a much higher degree of confidence.

The algorithm's only required input is the past behavior of users, which might be their previous transactions or the way they rate items. The ALS-WR approach is suited for implicit datasets, which makes it an appealing approach for the sparse implicit dataset used in this thesis.

In most collaborative filtering techniques for explicit datasets the unrated user-item pairs, which usually constitute the vast majority of the dataset, are treated as missing information and removed from the data set. In the implicit case we retain the zero action values as the missing positive actions might stem from other reasons than low preference, the user might be unaware of the existence of an item [3].

Let \( r_{ui} \) denote the observed positive action score made by user \( u \) for item \( i \). For implicit datasets it is natural to assign \( r_{ui} = 0 \) if no positive action is taken.

As we saw in the data discussion we introduce a confidence score based on \( r_{ui} \) calculated as:

\[
c_{ui} = 1 + \alpha r_{ui}
\]

as stated earlier this gives the \( c_{ui} \) a confidence value with a rate of increase controlled by the constant \( \alpha \) [3]. With the preference \( p_{ui} \) for item \( i \) and user \( u \) defined:

\[
p_{ui} = \begin{cases} 
1, r_{ui} > 0 \\
0, r_{ui} = 0 
\end{cases}
\]

this also gives a base confidence to \( p_{ui} = 0 \).

And as in the latent factor models the goal in ALS-WR is to discover vectors \( \mathbf{x}_u \in \mathbb{R}^f \) and \( \mathbf{y}_i \in \mathbb{R}^f \) for each item \( i \) and user \( u \) that will factor item-user preference. And the preference is assumed to be \( p_{ui} = \mathbf{x}_u^T \mathbf{y}_i \). The vector strives to map items and users into a common latent factor space where they then can be compared. This differs from matrix factorization in the explicit feedback case with two distinctions, varying confidence levels need to be accounted for and optimization needs to account for all possible item-user pairs, not only those corresponding to observed data. The factor-vectors are calculated by minimizing the following cost function.

\[
\min_{\mathbf{x}, \mathbf{y}} = \sum_{u,i} c_{ui} (p_{ui} - \mathbf{x}_u^T \mathbf{y}_i)^2 + \lambda (||\mathbf{x}_u||^2 + ||\mathbf{y}_i||^2)
\]

The exact value of \( \lambda \) is data-dependent and determined by cross validation.
The cost function contains $m \cdot n$ terms, where $m$ is the number of users and $n$ is the number of items. Many implicit datasets which include all $u, i$ pairs and not only observed ratings can number in the billions and therefore prevent most direct optimization techniques, such as stochastic gradient descent. Hu, Koren and Volinsky proposes an alternative efficient optimization process.

When either the user-features or the item-features are fixed the cost function becomes quadratic and can be calculated directly [3]. This leads to an alternating-least-squares optimization process, which also gives the algorithm its name. Alternating between re-computing user-features and item-features, and each step is guaranteed to lower the value of the cost function. To overcome the dense cost function and integrate the confidence levels. The following steps are taken.

Let us assume that all item-features are gathered in a $f \times f$ matrix $Y$. Before looping through all users, we compute the $f \times f$ matrix $Y^T Y$ in $O(f^2)$ time. For each user $u$, let us define the diagonal $n \times n$ matrix $C_u$ where $C_{ii} = c_{ui}$, we also define the vector $p(u) \in \mathbb{R}^n$ that contains all preferences made by $u$. Differentiation gives an analytic expression for $\vec{x}_u$ that minimizes the cost function:

$$\vec{x}_u = (Y^T C_u Y + \lambda I)^{-1} Y^T C_u p(u)$$

A bottleneck is found in calculating $Y^T C_u Y$, for if calculated naively will require $O(f^2)$ for each of the $m$ users. Here we use the fact that $Y^T C_u Y = Y^T Y + Y^T (C_u - I) Y$. Now $Y^T Y$ is independent of $u$ and can be precomputed [3]. As for $Y^T (C_u - I) Y$, only has $n_u$ non-zero elements, the non-zero $r_{ui}$ for user $u$. Typically $n_u$ is much smaller then $n$. Similarly $C_u p(u)$ also contains only $n_u$ non-zero values. This results in a computation of $\vec{x}_u$ performed in $O(f n_u + f^2)$. Here, we assumed $O(f^2)$ for the matrix inversion $(Y^T C_u Y + \lambda I)^{-1}$. This is run over each of the $m$ users, giving the total of $O(f m + f^2)$, where $N$ is the overall number of none-zero values of $r_{ui}$.

That is: $N = \sum_u n_u$. Typical values of $f$ vary between 20 and 200.

Computation of the user-features is followed with a computation of the item-features. Arrange all user-features within a $m \times f$ matrix $X$. First compute $X^T X$ in $O(f^2 m)$ time. For each item $i$ we define the $m \times m$ matrix $C_i$ so that $C_{ii} = c_{ai}$, and also the vector $p(i) \in \mathbb{R}^m$ that contains all the preferences for item $i$. Then solve:

$$\vec{y}_i = (X^T C_i X + \lambda I)^{-1} X^T C_i p(i)$$

Using the same technique as for the user-features described in previous sections. The algorithm calls for a few sweeps of re-computation, typically 10 sweeps [3].

The described process scales linearly with the size of data. And the recommendation is performed by recommending to user $u$ the $k$ items with the highest value of $\hat{p}_{ui} = \vec{x}_u^T \vec{y}_i$, where $\hat{p}_{ui}$ symbolizes the predicted preference of user $u$ for item $i$.

The Hu, Koren and Volinsky's ALS-WR algorithm has the property that it models implicit feedback to better reflect is nature. And the algorithm addresses all possible user-item pairs ($n \times m$) in linear running time, exploiting the algebraic structure of the variables [3].
Implementation and Technologies

There are a few available open source implementations of recommender systems. Ranging from hosted solutions to program libraries implementing algorithms and back-ends.

One of the more widely used is Mahout from the Apache foundation. Others include Crab, LensKit and Easyrec [6]. There are also commercial offerings available [6]. Due to the limited scope this report will focus on Mahout.

Mahout

Mahout is a Java library implementing Machine Learning Techniques. There is wide use of Mahout as a recommender [7]. The current manifestation of Mahout is the result of a merge between two projects, an earlier version of Mahout focused on machine learning algorithms and Taste a recommendation system library [7]. Mahout is an Apache project and is therefore readily available under open source licensing. Usages include clustering, classification, recommendation and frequent items set mining.

Why Mahout

Mahout is widely used and is actively developed, available under Apache licensing. Making Mahout powerful and affordable. The possibility of adding a clustering backends such as Hadoop or Spark gives it scalability for production use on relatively large sites.

Mahout is used by Foursquare implementing a recommender system. And by Twitter for user interest recommendation [8].

Algorithms

Mahout includes a wide variety of techniques and algorithms for clustering, classification and recommendation. The inclusion of the Taste project into Mahout has added many algorithms for use in recommendation systems.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Clustering</th>
<th>Classification</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-based CF</td>
<td>Canopy</td>
<td>Logistics Regression</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>Item-Based CF</td>
<td>K-Means</td>
<td>Bayesian Random Forests</td>
<td>Perceptrons</td>
</tr>
<tr>
<td>SlopeOne Recommender</td>
<td>Fuzzy K-Means</td>
<td>Hidden Markov Models</td>
<td>Neural Networks</td>
</tr>
<tr>
<td>Singular Value</td>
<td>Latent Dirichlet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decomposition-based</td>
<td>Allocation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td>MinHash Clustering</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Others: Evolutionary Algorithms, Dimensionality reduction techniques (SVD, PCA) and so on.
Recommending with Mahout

Mahout is a Java library and is most easily installed as a Apache Maven dependency. Mahout can be found at mahout.apache.org.

Building a recommender in Mahout [7].

1. Map data (file, database) to a DataModel class.
2. Tuning recommender component classes.
   1. Similarity measure, neighborhood, matrix factorizers.
4. Compute recommendation scores.
5. Recommend products by querying the recommender object by user or item depending on implementation.

SVDRecommender with ALSWRFactorizer

The recommender implementation that was used in this thesis work is the ALS-WR matrix factorization algorithm described in earlier sections. This algorithm is implemented in the ALSWRFactorizer class in Mahout and can be used as component in the SVDRecommender class.

The ALSWRFactorizer class is a non-distributed implementation of ALS-WR that utilizes multi-threading. Distributed implementations are also available if one wishes to use Mahouts distributed computing back-ends.

The SVDRecommender class provides an interface to build recommenders based on matrix factorization. SVD recommenders project the user and item relation on to a feature vector space. Several implementations of factorizers are available in Mahout; SGD, SVD++, ALSWR and parallel implementations of SGD and ALS-WR.
Evaluation

The scope of this thesis does not allow for on-line evaluation of the performance of the chosen recommender implementation. Therefore an off-line metric is needed for evaluation purposes both of algorithms and for parameter cross validation.

Hidden and training data

To be able to test the prediction performance of a recommender system we split the available data randomly into two sets. A training dataset and a hidden testing data set. The algorithm is then trained with the training data and evaluated using the data that was hidden from the system [1] [3] [4].

Mean Rank

Most off-line evaluation performed in recommender system research use precision metrics, such as the mean squared error between the predicted rating of item $i$ by user $u$ and the actual rating made by the user $u$ of item $i$. This is not useful with implicit data as we are looking at purchase frequencies and not explicit item ratings. In the implicit case a recall-based metric is more appropriate. Hu, Koren, Volinsky proposes a recall metric for the implicit case, mean rank [3] [4].

Mahout supplies a range of recommender evaluation classes most built around various precision metrics. A pure mean rank class however is not available. But is fairly simple to implement.

1. First we remove a number of data points from the training data. A few hundred user-item pairs. This is done randomly. Effectively hiding the data from the algorithm. Making sure to save the hidden data, for testing.

2. Train the algorithm, with the training dataset.

3. For each user-item pair in the hidden dataset.
   1. Retrieve the recommendation score for all items and sort them from most relevant to least relevant to the current user.
   2. Find the position of the current test item in the list of recommendations. And calculate the relative rank $r$ as $r = \text{position} / \text{total number of items}$.

4. When all the ranks have been calculated calculate the mean rank for the test run.
Results

Early implementations and tests

Early in my work I made attempts with neighborhood collaborative filtering techniques both Item-based and User-based with several different similarity measures. These techniques do not lend themselves well to implicit datasets, and have issues with data sparsity. These implementations were initially built in Python giving both performance issues as well as sparsity issues. An implementation of standard Collaborative filtering techniques where made with Mahout, lessening the performance issues. Unfortunately poor results were achieved with these methods, both due to sparsity and the implicit structure of the data. To alleviate the sparsity problem I instead turned to matrix factorization and SVD recommender implementations in Mahout. Because of the implicit structure of the dataset ALS-WR factorization was chosen among the available methods.

Results with ALS-WR recommender system

I have evaluated the ALS-WR recommender using two different datasets. The category dataset containing purchase frequency of item categories. With this dataset the algorithm tries to predict in which item-category the users next purchase will be. The other case using the item dataset containing purchase frequency of items, the algorithm tries to predict which exact item the test user will purchase.

Benchmark

As a benchmark a simple popularity based recommender was implemented. This recommender counts the number of purchases of an item or item-category independent of the current user. The same mean rank evaluation was applied.

1. Randomly select a number of users and hide their data from the algorithm.
2. Order all items or categories from most frequently purchased to least frequently purchased.
3. For each user in the hidden data set:
   1. Find the item or category in the global popularity list.
   2. Calculate the relative rank for that item or category.
4. Calculate the mean rank for the test run.
**Item dataset**

**Predicting item preference**

Best result for the item set was achieved with a lambda $\lambda=1000$ and alpha $\alpha=80$. The chart below shows a lambda search with 15 features for the item and user vectors and 15 iterations of the algorithm.

![Lambda Item-Dataset](image)

For the item prediction algorithm the best mean rank value was achieved with 35 features and 15 iterations of the algorithm. The results are graphed in the figure below.

![Mean Rank %](image)

With $\lambda=1000$, $\alpha=80$, 35 features and 15 iterations the item ALSWR recommender achieved a 13%
mean rank. That is the test item was generally found among the top 13% of the total item list ordered from most relevant to least relevant for the test user.

**Category dataset**

**Predicting item category preference**

For the category dataset best mean rank result was achieved with a lambda $\lambda=2200$ and alpha $\alpha=80$. Using 15 iterations and 15 features the algorithm gives a mean rank of around 9% using $\lambda = 2200$. Below the test results for different $\lambda$-values are shown.

The algorithm score a 8.5-9% mean rank when predicting user preference of categories meaning that the hidden item category was generally found among the top 9% of the ordered list of all categories.

**Benchmark comparison**

When comparing the ALS-WR Recommender results to the popularity benchmark results only the category recommender outperforms the benchmark. As can be seen in the table below the ALS-WR recommender is on par with the popularity benchmark.

<table>
<thead>
<tr>
<th></th>
<th>ALS-WR</th>
<th>Popularity benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category Dataset</td>
<td>9%</td>
<td>11%</td>
</tr>
<tr>
<td>Item Dataset</td>
<td>13%</td>
<td>13%</td>
</tr>
</tbody>
</table>
Discussion

Popularity Recommender vs ALS-WR recommender
As seen in result section of this thesis the mean rank results compared with plain popularity recommendation is discouraging. With no significant decrease in mean rank using the algorithm for item recommendations compared to the popularity benchmark. The accuracy of the popularity benchmark is somewhat surprising.

Data bias
The current method on Sportamore's site to generate top lists and recommendations are based on the popularity of items and categories. As items are more frequently shown to the user based on item popularity we have no way of knowing if the user bought the item due to actual preference or due to the fact that no more relevant item could be found or was presented. There could be made an argument that off-line testing method is unfair to the ALSWR recommender due to the implemented popularity sorting of the site. The test users have all had their decision influenced by the popularity sorting, which might skew the data towards a better score for the benchmark.

Which is better
As with all recommender systems the actual performance of the ALS-WR recommender in a production environment can not be known beforehand. The popularity benchmark method will present the same items to all users and as stated earlier the training data and the testing data will reflect existing buying patterns which is already governed by item popularity. There might be a case to make that recommending from the ALS-WR recommender will actual produce a better prediction of user preference than popularity. The recommender might recommend items that otherwise would be hidden to the user due to lower popularity.

Using the ALS-WR recommender could also in time change the overall buying pattern which in turn would change the algorithms score compared to the popularity benchmark. The only way to know is to test it in production with real users.

One or several Algorithms
Erik Bernhardsson's thesis work done for Spotify [4] exemplifies an attempt to use several algorithms to create an optimal recommendation pipeline. This might be a way forward to increase the overall prediction performance.

Choosing an algorithm
There are many algorithms available and choosing which one is the right fit for the available data is not a trivial task. For the case in this report with implicit purchase data, the choice of ALS-WR came about after testing several neighborhood algorithms on the large and sparse dataset for item purchasing, resulting in poor or no results. Utilizing these traditional methods proved better with the
category data, probably because the sparsity was a lot less due to the greatly reduced 'item' dimension. The sparse dataset turned me towards factorization methods, and particularly the SVD variety, after reading of both better predictions, performance but also sparsity reduction. The ALS-WR seemed to be a good fit for the data, and yielded results which were at least more intuitive.

**Item vs Category predictions**

The sporting goods business is both seasonal and the lifetime of a unique item is not very long, ranging from a season to a few years. There is also a basic assortment with long lifetime, and iconic items with an endless lifetime, Chuck Taylors from Converse for example. So it stands to reason that if one were to use all available data to build a recommender then the data would include a lot of “dead” items, and it would be skewed towards recommending iconic or basic items with long lifetimes. This does not necessarily have to effect the recommendations negatively depending on the goal. It could also stand to reason that most long lived products are desired by the users, and therefor recommending them might drive sales. But if the goal was to show items to users that they otherwise might have missed then showing basic assortment such as socks or iconic products may be counter-productive.

If we instead consider using category recommendation, then we have different issues. The recommender would recommend other categories that are 'similar' based on purchasing similarity. If we get results of the form pants are similar to sweatpants then we have no reason to use it, we already know this. The gain would be if we could produce results that users buying running shoes should be presented with wind jackets or another category which is not self evident. The problem than becomes which item within that category should be presented. The obvious way might be to recommend the most popular item that has overlapping features with the user. For example if the recommender predicts that the users should be interested in buying pants, popular pants with the correct gender might be presented. But again the popularity of some items might hide other more relevant items.

Both the category and the item approach, has the problem of seasonality. Wind jackets might be very relevant to a user during fall or spring but completely irrelevant during summer or winter. An attempt to remove this problem might be to train the algorithm with seasonal relevant data. In the category case using last years purchase history for the current season might work to alleviate the problem. In the item case the problem with dead items remain, so we have a new kind of cold start problem in the beginning of each season. This might be exaggerated as the start of each season is not clean cut, and winter items are still selling in the beginning of spring. Thus employing a rolling data collection and training, for example using the last three months purchase history might be enough. The problem remains if there is a sudden shift in the weather early in the season making the recommended items seem odd. This should of course correct itself if training is fairly regular.
User experience

All recommender systems suffer from perceived relevance, that is to say a difference between statistical predictions and user expectations. For example recommending children's boots when buying running shoes might be correct given the dataset at hand but could potentially be perceived as weird by the user.

Many papers on the subject of recommender systems discuss the problem of telling the user why a certain item is recommended. Telling the user might build trust in the system, but it might also raise questions of privacy. Hu, Koren and Volinsky discusses using the ALS-WR algorithm to explain how the recommendations are made [3].

Cold start problems

The ALS-WR algorithm can only recommend for known users as there is no calculated user-feature vector for a unknown user. This makes the ALS-WR recommender useful for retention recommendation to known users, but not for on site recommendations when the user is unidentified. Using an Item-based recommender could address this issue, being able to recommend solely on item similarity.
Conclusion and Further Work

The performance of the resulting algorithm is underwhelming, and further testing is needed to evaluate the production value of using the ALS-WR recommender. Further work should also be done in refining the data collection, adding other sources of user preference for items might improve the results. Page views and on-site behavior might increase confidence in user-item preference scores. The performance of the popularity benchmark gives indications that either the users buy what is presented to them, or they share very similar interests. Deploying the recommender in production presented to a limited number of users in an A/B-test, could shed further light on business value, and give further information on the recommender's true performance versus the popularity benchmark.
Literature


