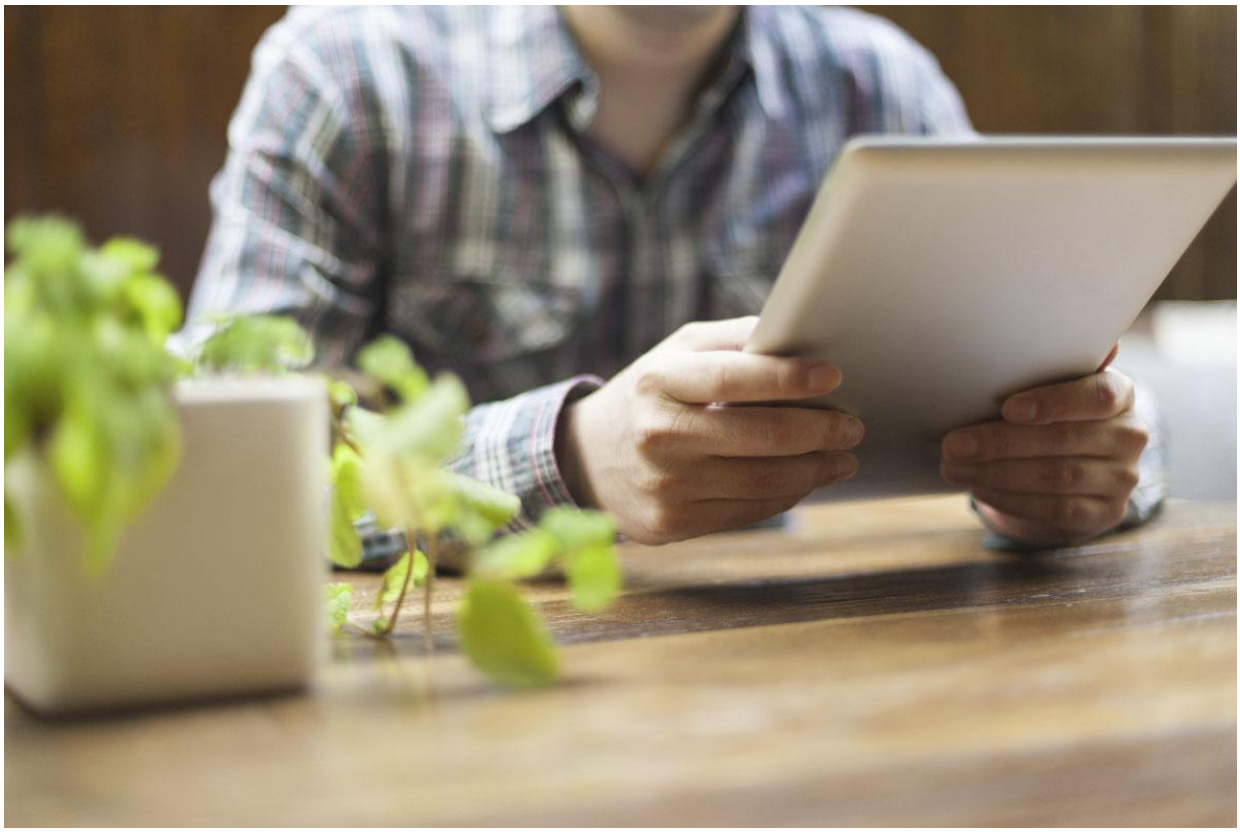
Recommending articles using implicit feedback data



Master's Thesis

Copenhagen Business School 2017

Programme: **MSc in Business Administration and Information Systems**

Student: **Mikkel Sikker Sørensen (Student number: 31986)**

Supervisor: **Daniel Hardt**

Date of submission: **01-11-2017**

Number of pages: **61**

Abstract

This paper explores the area of different recommender systems, their implications, and business relevance. The experiments conducted are based on a dataset provided by the Danish news media Jyllands-Posten.

The experiments seek to discover the possibilities for Jyllands-Posten to improve their subscribers online customer experience, utilizing the dataset provided with millions of data observations on thousands of articles.

Different recommender systems and approaches to developing them have been tested and discussed. A ranking factorization model based on a collaborative filtering approach performed the best out of the tested item-similarity and popularity-based models. The popularity-based approach served as a baseline for the other models performances since it is the most simple approach and its functions are similar to the solution that they have currently implemented. The popularity-based model recommends the same articles to all users accessing jyllands-posten.dk and thus lack the element of personalization. This could also be why the ranking factorization model outperformed the popularity-based model by about 10-15% in every evaluation.

The recommendation for implementing the ranking factorization model comes with a catch, which is that it only should start recommending to users exceeding the threshold of 80 read articles on jyllands-posten.dk. This is due to the fact that with 3 given recommendations and with a threshold of at least 80 read articles the recommendation system had an accuracy of 33,5% and thus it is fair to assume that at least one of the articles recommended, statistically, is a valuable recommendation.

By giving personalized recommendations to its users, Jyllands-Posten can improve their satisfaction as well as better utilize the long tail of having thousands of articles available on their site, that despite of their publishing date might be of relevance to some users.

Table of Contents

1 Introduction

This section provides an overview of the content presented throughout the paper and introduces the overall topics machine learning and the more specific techniques to build recommendation systems. Following is a problem definition that serves as the objective of the paper.

Related work inspiring and contributing to the development of this paper will be presented. Next, there will be a run through of the experiments; from data collection and modeling to the results from the final model output. The conclusion, discussion, and considerations about future work, will round of the paper.

1.1 Machine learning

We live in a world of data, which we constantly generate through our online behavior and use of smartphones, computers, smart TV's or other IoT devices, contributing to the ever-growing pool of data. In fact, the amount of data in the world is increasing exponentially, it is estimated that the total amount of stored data worldwide is doubled every 20 months. The challenge faced when getting access to all this data is the extraction of valuable and meaningful information. This challenge can be met by finding patterns and connections that are not predefined.

Machine learning is the art of extracting knowledge from data. Essentially, it is different algorithms that enable computers to learn from examples and experiences, rather than having to rely solely on hard-coded rules written by humans. Machine learning consists of several steps such as; collecting data, preprocessing data, training and testing models to compute and predict outcomes.

The model selection depends on the problem to be solved and the available data. Different machine learning algorithms excel at different problems. For instance, whether the dataset is labeled or unlabeled, explicit or implicit needs to be taken into account when deciding what algorithm to apply on the dataset. There are three overall types of machine learning problems, supervised-, unsupervised- and reinforcement learning.

Whether a supervised or unsupervised learning method applies to a given problem, depends on whether or not we have a target label in our data before building a predictive model, i.e., do we know beforehand exactly what we want to predict. If there is a defined label then it is a supervised learning problem, where several supervised learning algorithms can be used to make predictions. When the target label is numerical regression algorithms are appropriate to use. If the target variable is known but nominal instead of numeric a classification algorithm can be used, e.g., a decision tree.

Unsupervised learning comes to use when searching for patterns in data without a pre-specified target. Unsupervised learning usually entails some sort of clustering to find structure in the unlabeled data. There exists different algorithms for clustering data into groups, like the kNN that stands for k nearest neighbor and works by grouping instances by their nearest neighbors determined by a distance measure like euclidean- or cosine distance.

Reinforcement learning trains and optimizes parameters based on feedback that the model gets from the environment of which it is deployed. A well-known example of reinforcement learning is the Google DeepMind's Deep Q-learning, playing the Atari game Breakout. The model is not taught beforehand how to play the game, which clearly shows in the first many iterations, where the machine dies very early in the game. However as the model goes through iterations of playing games, it improves, and after just 120 minutes, it plays on an expert human level and after 240 minutes of playing it has figured out an effective tactic to beat the game. All with just the feedback of either gaining points or returning to 0 points when dying. All the model knows is that its goal is to maximize the point variable. The trained algorithm will be of use in predicting outcomes when given new relevant data.

Everyone experiences machine learning in action. Whether avoiding the receiving of spam emails from advertisement companies in their inbox or receive customized search engine results based on their preferences and historical search queries. Or getting recommendations for movies on Netflix, or products on Amazon based on previous behavior.

Furthermore, companies utilize machine learning technologies to aid in decision-making. This could, be an investment bank using machine learning to predict stock price development, to assist brokers in buying and selling decisions. Alternatively, banks using machine learning models in deciding whether a certain loan is profitable or not to provide a given customer is also a used implementation. Machine learning is also present in the health sector. They have it implemented in the form of image recognition to aid doctors in the diagnosis process from patient MRI scans, etc.

These are just a few examples of machine learning techniques being carried out by various institutions to illustrate how machine learning is deployed in many different fields. Looking just five or ten years into the future, the increasing amounts of data and continuous improvements in processing power, machine learning will become even more integrated into every aspect of business and our everyday lives than it is today.

1.2 Recommender systems

The following section will briefly introduce the concept of recommendation systems. Methods used for building recommendation systems will be further elaborated with concrete approaches in the section 'Theories for Recommendations'.

Recommender systems help users manage information overload. Finding information and making decisions where users lack the knowledge of, or about, a piece of information or product. One way to look at recommender systems is as data filters that fit data to individual user needs (Ullman, Jure, & Rajaraman, 2014). These needs could be something the users were not aware of beforehand.

The Xerox Palo Alto research center built the first recommender system in 1992 called Tapestry. Tapestry was built to handle the increasing number of emails that researchers of the center received. The recommender system used a collaborative filtering approach to find user similarities based on preferences (Goldberg, Nichols, Oki, & Terry, 1992).

Figure 1.1 represents a simplified overview of a recommendation system in action. The database contains data about users, items, and their interactions. Data extracted from the database is used to train a recommendation system, which then presents recommended items to the user through the user interface. Data generated by the user's interactions with the user interface is sent to the database to increase data that the recommendation model gets as training input and thereby improve future recommendations.

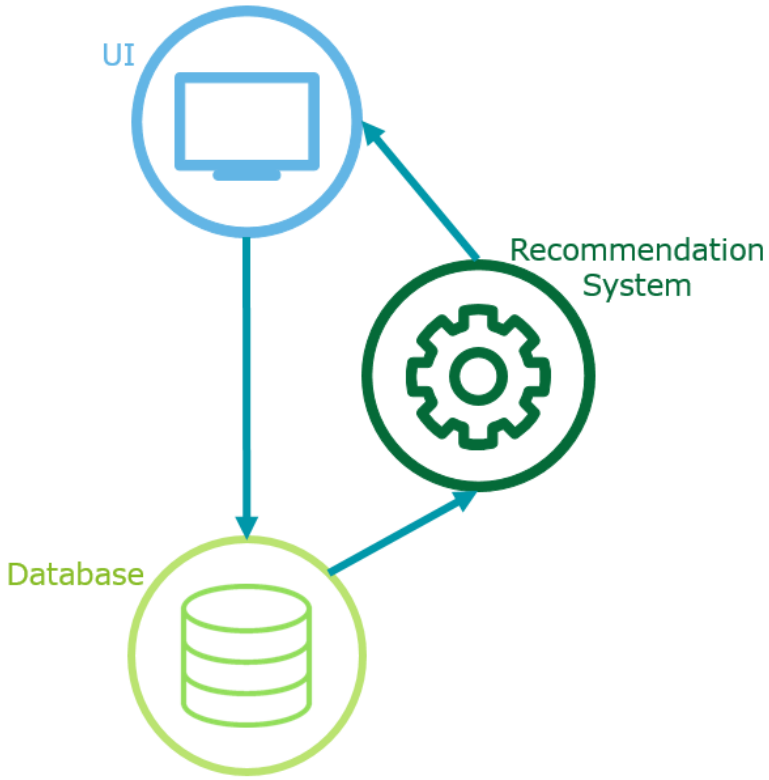


Figure 1.1

1.3 User and business implications of recommender systems

The recommendation of products is one of the oldest and most common sale approaches, recommending an item based on the wants and needs of a customer. Recommendations could also be an associated product to something the user has previously bought or looked at. Websites are the new salespersons and needs to adopt the recommendation of goods task, which was originally handled by sales employees.

The major user benefit of recommendation systems is the reducing of complexity in a world with an exponential amount of data. It is a way to receive prioritized information to ease decision-making. Most search engines are huge recommender systems. Google as an example, delivers personalized search results based on prior search queries.

The companies' benefits of using recommender systems are numerous. It provides an opportunity to improve communication with customers through personalized feeds, and serves as a method for pushing sales and thereby increasing revenue.

2 Problem definition and thesis objective

2.1 Business relevance

Consumers in today's world are flooded with choices, and the number of choices are increasing in every aspect of life. Online retailers offer a huge range of selections to fit consumer's individual tastes and needs (Koren, Bell, & Volinsky, 2009).

Matching the right users with right items is key to improve user experience and thereby overall satisfaction and loyalty (Koren, Bell, & Volinsky, 2009). Recommender systems can play a tremendous role in guiding consumers, narrowing choices down to the best-fit options and thereby leading to better decision-making (Gomez-Uribe & Hunt, 2016).

Recommender systems democratize access to the long tail of products, services, and information to users. Machines can learn from a lot of data and with useful predictions of valuable recommendations. They can give users and businesses the opportunities of utilizing the long tail (Gomez-Uribe & Hunt, 2016).

This new marketplace comes with a shift in demand from "hit", being the top most popular products to more niche products. The vast range of opportunities that technology has enabled for consumers fuels this shift (Gomez-Uribe & Hunt, 2016). Before music streaming services like Spotify, everybody listened to the same hits on the radio. Today, everybody creates his or her own playlists, not limited by genres, age or popularity. The same goes for TV, wherein the old days, before streaming. Everybody was more or less watching the same programs at the same time, or at least limited to what the channels chose to send and their time schedule. Today streaming services like Netflix or YouTube provides a much

broader range of different opportunities and the ability to offer niche movies without sacrificing any more popular movie.

As with the different online streaming services, news has also changed a lot. From the publisher having to choose what content to print into the newspaper, to having the ability to publish as many articles as they can produce in real-time on a website.
The news media has also changed a lot from the consumer's point of view. Before news online were available to everyone, the consumer had to choose what newspaper to buy and with that getting all of the content printed in that specific newspaper. Even though he or she is might only had interest in one or two of the articles printed within that newspaper.

Today with online news sites, users can jump from site to site and read a given number of articles on each and publishers has to deliver news in almost real-time to remain relevant. Furthermore, publishers do not have a limit on the number of different articles it can publish. Generally, physical newspapers publish 50-100 articles a day, where online news sites publish up to a thousand, or more (Ullman, Jure, & Rajaraman, 2014).

In figure 2.1 an illustration of the long tail, also referred to as the new marketplace is presented. Popularity is on the x-axis and products on the y-axis. The range of most popular products, referred to as the head while all other products are called the long tail. The idea behind the concept is that physical retailers only have the storage capacity to display the most popular range of products, referred to as the head. On the other hand online retailers can tailor a more dynamic storefront, presenting the user with products that might not be in the top 100.000 of most popular books, but still relevant to the given user, based on valid recommendations. Thereby utilizing the long tail of products that physical merchants do not have the space to carry in their stores.

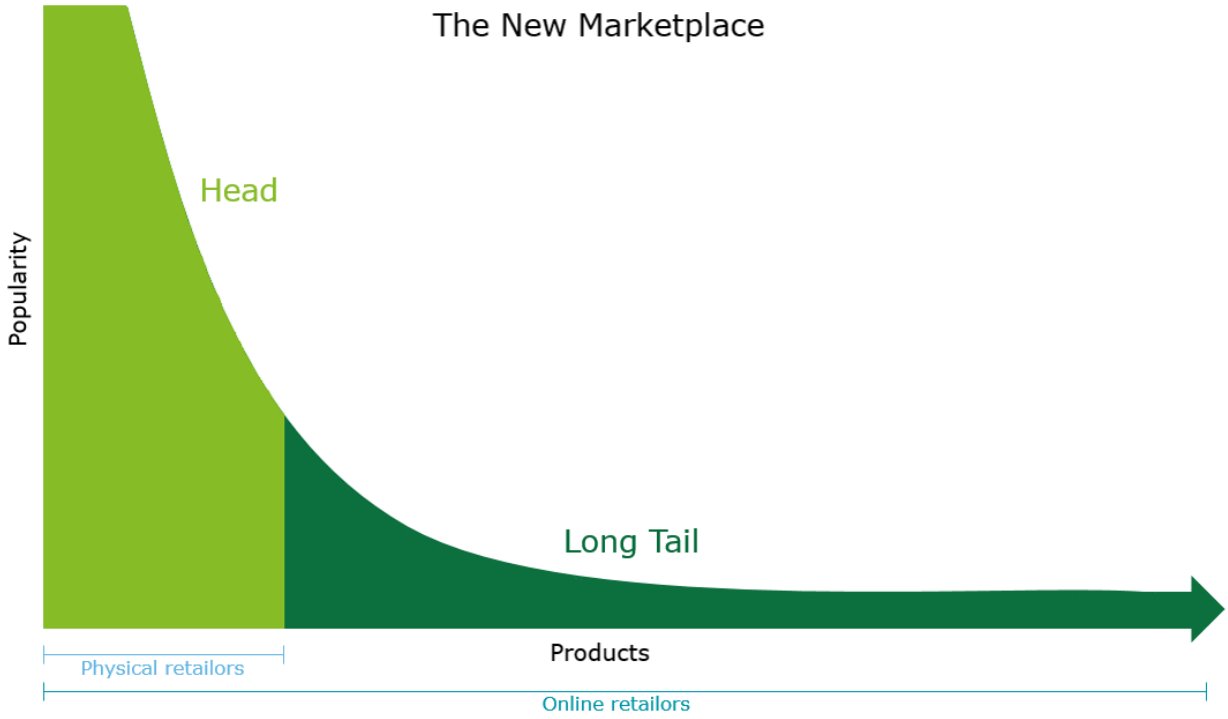


Figure 2.1

A good example of this is how Amazon has surpassed physical bookstores, partly due to their utilization of the long tail. Where physical bookstores have approximately 40-100 thousand books for sale, Amazon have approximately 3 million books available through their e-commerce website. One could argue that the users do not care about the remaining 2.9 million books that are not amongst the most popular and thus belongs to the long tail category. Brynjolfsson, Hu, and Smith wrote an article addressing this dilemma (Brynjolfsson, Hu, & Smith, 2006). Analyzing Amazons sales data, they discovered that 30-40% of the total sales

were from products considered as long tail items. They concluded that implementation of well-constructed recommendation systems increase the sale of niche products.

An example of such a real-life case from Amazon utilizing the long tail with a recommendation system is presented in the following paragraph.

The book 'Touching the Void' about mountain climbing was published a long time ago and was not a best-seller in its day. Many years later the book 'Into Thin Air' was released on the same topic.
Some users bought the two books together on Amazon, and their recommender system started recommending 'Touching the Void' to users who had bought or looked at 'Into Thin Air'. 'Touching the Void' eventually surpassed 'Into Thin Air' in the amount of books sold. This would not have been possible if not for the utilization of the long tail through recommender systems that Amazon practices. Physical bookstores did not keep 'Touching the Void' in their stores due to its low popularity, nor might have realized the valuable associated product recommendation that a recommendation system enabled Amazon to discover (Ullman, Jure, & Rajaraman, 2014).

2.2 Addressing chosen problem

The problem addressed in this paper is the exploration of data from the Danish newspaper Jyllands-Posten to improve their current recommendation approach. The approach implemented at the moment is a 'most popular' based article recommendation.

Screenshot 2.2 is taken from an article page at Jyllands-Postens website jyllands-posten.dk. In the marked red box is the currently implemented recommendation function. This function is based on the most read articles of all users on the site and thus is a popularity recommender.

Screenshot 2.2

A popularity model works well as a baseline model. However, it does not personalize the newsfeed for individual users. This paper will address how to personalize these recommendations for a higher prediction accuracy of

recommendations. Measured by the ratio between recommended and actually read articles.

Improving the recommendation function through a personalized newsfeed is not an end goal by itself. The improvement of recommendations is proposed for the purpose of improving the user experience. Matching users with the right items is key to improving user experience and in turn satisfaction and loyalty (Koren, Bell, & Volinsky, 2009). Satisfied and loyal customers will lead to an increase in paying subscribers as fewer users leave the site and the ones trying out the service are left more satisfied. The end goal of increasing revenue is realized through the increasing number of paying subscribers on jyllands-posten.dk.

2.3 Research Question

- How can Jyllands Posten improve their subscribers online customer experience?
    - How can user data be utilized to improve the customer experience?

With this research question, I wish to investigate how Jyllands Posten can improve their readers experience when visiting their website. How to leverage their collected user and article data from user behavior on their website, to deliver personalized news feeds for registered users.

3 Related work

This paper builds upon several articles on related problems to the research question of this paper. This related work is a gathering of several papers on recommendation system approaches and different implementations and implications in these contexts.

The topic of recommender systems and their application is addressed by Aher and Lobo (2013) in their article; *Combination of machine learning algorithms for the recommendation of courses in E-Learning System based on historical data*. The study addresses course recommendations on an E-learning platform. The approach is recommending courses to students based on data on other students' course selection and past attended courses. They test different combinations of methods without great results, until discovering a combined approach of the methods Simple K-means clustering and Apriori association rule algorithm, which performs well enough to recommend accurate and valuable course recommendation to students suited to their individual interests. They argue that their method could have great implication if integrated with MOOC (Massively Open Online Courses).

The approach to solving information overflow with the use of recommender systems is explored by Wang and David in their paper *Collaborative Topic Modeling for Recommending Scientific Articles* (2011). They propose implementing a hybrid recommender to recommend relevant scientific articles to users of an online community. Their hybrid recommender approach utilizes both articles contents and user ratings. Their findings concluded that the hybrid recommender approach they propose in their solution, outperforms traditional matrix factorization approaches while working well with new publications not yet rated by anyone from the community.

Another article assessing the approach to recommender system algorithms and their evaluation is Pabani and Deulgaonka in their paper *News Recommendation System Based on Multiple Classifiers* (2016). This paper assesses the recommendation of news articles using multiple classifiers. The experiments are conducted on a dataset from the BBC news website. Their different approaches to recommending using classifiers are K-means clustering, a Naïve Bayes probabilistic classifier and a Rank-Classifier using word frequency in articles to index them. With their findings which was based on evaluation through metrics, such as

precision and recall, they concluded that the Naïve Bayes classifier was the best performing on their dataset, but an incorporation of different classifiers was the optimal solution.

A commonly cited paper on the topic of recommender systems when using implicit feedback, is the paper; *Collaborative Filtering for Implicit Feedback Datasets* by Koren, Hu, and Volinsky (2008). They discuss the implications of using implicit feedback data, largely due to the fact of its lack of negative user feedback indication. The only feedback that can be considered negative are the items not yet consumed, which e.g. can be because of a deselection or unawareness of the item. These items are also the ones in where recommendations are to be found. There can be many reasonings behind an item not yet being consumed by a user; lack of knowledge about the existence of the item, low preference towards the item or difficulties finding a certain item. In their paper, they present a method of dealing with implicit feedback data by transferring user observations into positive and negative user and item pair preferences with different confidence levels. From this, a factor recommender model is developed, which competes in performance with the best explicit feedback recommenders.

Wang, Chuang, Hsu, & Keh (2004) wrote a paper studying the business implications of implementing recommendation systems. They implemented a personalized recommender in an e-commerce cosmetic business. Their findings concluded, after tracking performance for a year, that recommendations of similar items improved sales of recommended items with 9 percent. Meanwhile, recommendations across product categories did not make a difference. They state that domain knowledge is crucial when implementing recommender systems in an industry like cosmetics.

Another paper addressing the business implications of recommender systems is *The Value of Personalised Recommender Systems to E-Business: A Case Study*, by Dias, Locher, Li, and El-Deredy (2008). This paper explores the effects of recommender systems in e-commerce businesses. The effect of recommender systems is investigated through a case study in the e-commerce business LeShop.ch. Recommender systems were implemented in-store and at checkout, introducing customers to new and unpurchased items. They implemented the recommender systems in May 2006 and measured the effect of these until June 2008. Their findings concluded that the effect of implementing the recommender

systems indicated an increase in revenue superior to recent years. Furthermore, they discovered two key findings; to frequently retrain the model on new data to keep it updated, and that the benefits of implementing a recommender system extended beyond short-term revenue increase. The recommender systems not retrained frequently on new user and item data quickly dropped in performance. Secondly, that benefits from recommender systems extends beyond directly extra revenue generated. Users who were introduced to new products that they did not usually buy, not only possibly bought this product but also revisited the site to buy more related products from that category of products.

These former studies lays the foundation that this paper is based and tries to extend on. It draws on Aher and Lobo's (2013) approach to testing different recommendation methods to eventually finding the best performing recommender. Additionally Wang and David's (2011) considerations about information overflow as a problem for recommendation systems to solve. While using hybrid recommender approaches for dealing with collaborative filtering's cold start problem. On the field of recommending news articles, Pabani and Deulgaonkar's (2016) article worked as inspiration to the approach for recommending news articles based on users past reading history. Using evaluation metrics like precision and recall to evaluate models and exploring hybrid models composed of different recommendation methods. Koren, Hu and Volinsky's (2008) paper on how to handle implicit data such as; search patterns, browsing history and purchases. It contains techniques for for working with implicit data, e.g., transferring user observations into preference and confidence levels.

Looking into the business value of recommender systems, the paper from Wang, Chuang, Hsu, and Keh (2004), is considered when arguing the business implications of recommender systems. Considerations about the domain within the recommender system are operating and specific domain knowledge in that context, is important to acknowledge. Another paper used for assessing the business implications of recommender systems is the one of Dias, Locher, Li, El-Deredy, and Lisboa (2008). Considering how the introduction of articles from new categories might improve user experience, making them coming back to read more within the same category of articles. Remembering the importance of retraining recommender systems frequently

The way this paper differentiates and thereby contributes to the academic pool of value on the subject is by using implicit feedback data. Something very common in real-world applications, but less common in literary research. The method of altering thresholds to combine different recommender models. This paper introduces the concept of a hybrid recommendation system approach using different recommending methods for different data thresholds.

4 Theory and methodology

The following paragraphs introduce the different theories and procedures for building recommendation systems. Followed by an explanation of the specific concepts applied in the recommendation model building, described in the experiments section.

4.1 Theories for recommendations

There are two general approaches for recommending items to users. One is content-based relying on the content of items to be recommended, the other is collaborative filtering. It is based on historical user-generated data. Additionally, the two approaches can be combined into a hybrid model method, with the purpose of getting the best of both worlds. The following sections will elaborate on the technique and different features of each method.

4.1.1 Content-Based systems

Content-based recommender systems rely mainly on item content. Items with the most similar content to what the user has consumed in the past, but not yet consumed, is recommended to the user (Ullman, Jure, & Rajaraman, 2014).

The approach simplified is illustrated in figure 4.1.1.

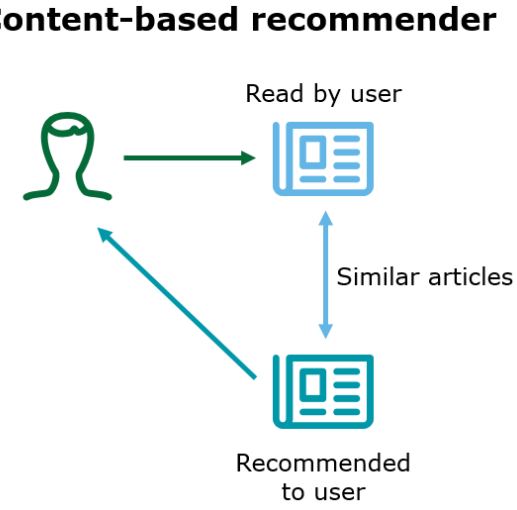


Figure 4.1.1

Diving deeper into the content-based approach a look at figure 4.1.2 gives a more detailed description of what actually is going on behind the scene. A user interacts with some articles; their feature values are stored in that user's user profile, this user profile is used for matching with content profiles, recommending articles with similar feature values as the ones contained in the user profile.

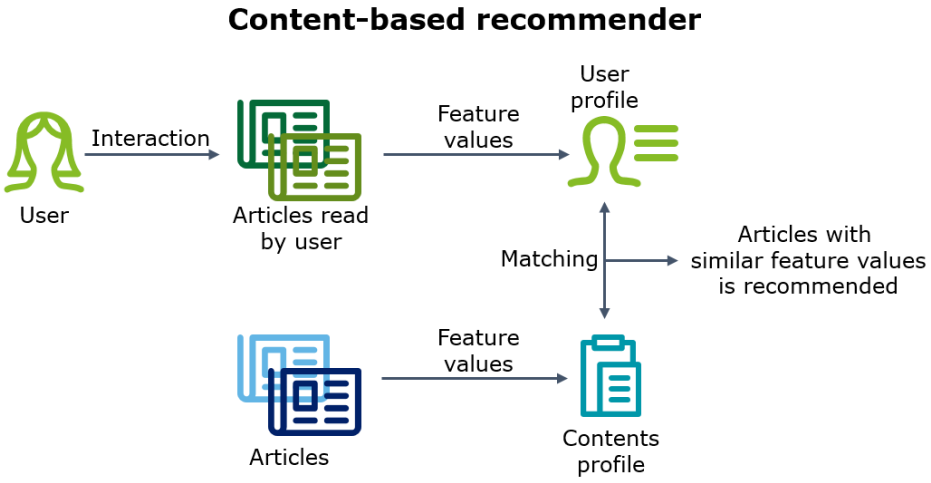


Figure 4.1.2

4.1.2 Collaborative-Filtering

The collaborative filtering approach is based on historical user-generated data, collaborative filtering algorithms can be user or item based. User-based approaches generate recommendations by finding similar users with the assumption that they have similar preferences. Item-based approaches use similarity between items, their closest K-neighbors to give recommendations (Ullman, Jure, & Rajaraman, 2014).

Collaborative filtering algorithms are generally more accurate than content-based approaches when scaled, but has a cold start problem (Koren, Bell, & Volinsky, 2009). The cold start problem is the inability to present new users with the valuable recommendation, with data being sparse. New items also need some attention to be considered in recommendations and thereby gaining traction.

A simple figure illustrating the user-based approach is presented in figure 4.1.3. The shared interest in items matches users and give recommendations based on items that one has consumed, which the other is yet to consume.

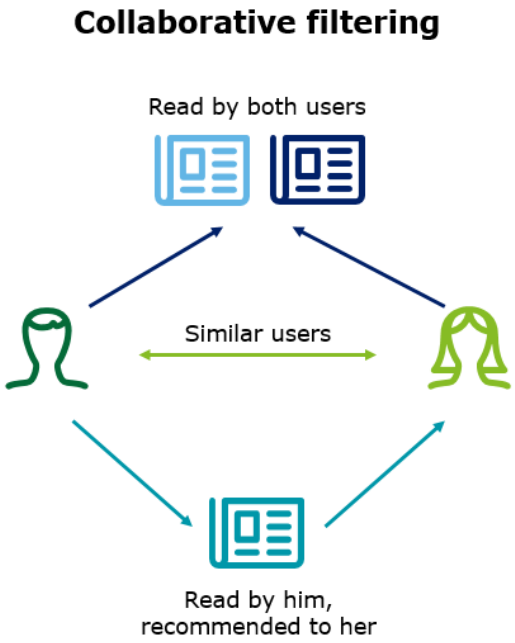


Figure 4.1.3

Going further into the user-based collaborative filtering method, figure 4.1.4. illustrates a simple example of how a correlation matrix extracts recommendations for a user, after matching with another user having similar interests.

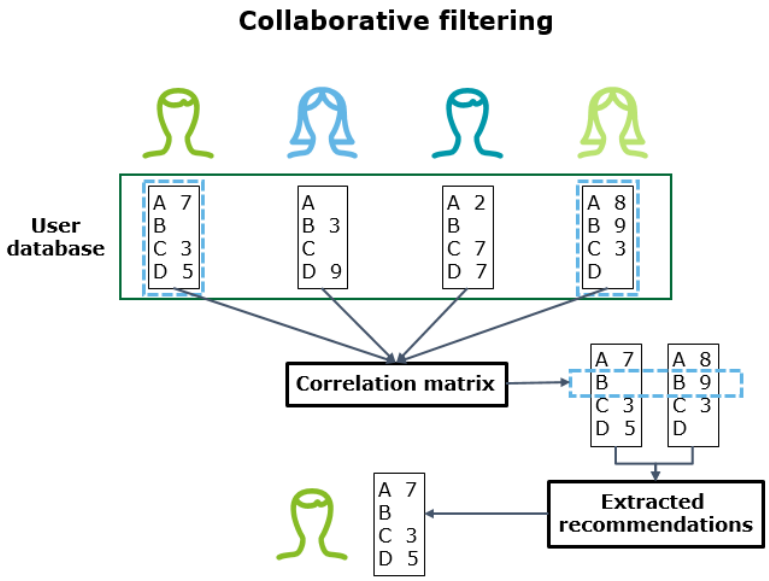


Figure 4.1.4

4.1.3 Hybrid recommenders

Hybrid recommenders blends methods of content-based and collaborative filtering recommenders, trying to tackle the downsides that each method faces. Content-based approaches perform well on new users but do not have the scalability when offered tremendous amounts of data that collaborative filtering has. On the contrary collaborative filtering models have a cold start problem and do not perform well on new users or items. A problem that could be dealt with by incorporating a content-based approach to accompany the collaborative filtering algorithm, thus working as a hybrid recommender system.

4.2 Applied theories

This research paper will focus on collaborative filtering due to it generally being more accurate than content-based approaches (Koren, Bell, & Volinsky, 2009). Furthermore, collaborative filtering provides good scalability while also efficient on sparse data. There are generally two different methods for collaborative filtering; neighborhood- and latent factor methods.

Neighborhood methods work by computing relations between items that tend to get similar ratings or users that tend to give similar ratings to similar items. Latent factor models map users and items into factors in obvious or uninterpretable dimensions. Factors of *u* and *i* mapped close to each other on dimensions would result in a recommendation.

One of the most successful latent factor models is matrix factorization. The approach characterizes both users and items as vectors inferred from rating or interactions patterns. Matrix factorization proved to be the most successful algorithm in the Netflix price challenge, surpassing neural network methods for making recommendations (Koren, Bell, & Volinsky, 2009).

The chosen algorithms for building recommenders in this paper are the neighborhood method; item similarity and latent factor model; factorization machine, built upon the matrix factorization approach. These algorithms are explained further in the following sections.

4.2.1 Popularity model

One recommender system approach is a simple popularity model. The popularity recommender always recommends the most popular items; hence, it does not individualize recommendations. Every user gets the same items recommended based on those items overall popularity, the most views, or ratings. This model can be valuable with new users, overcoming the cold-start problem of collaborative filtering algorithms since it does not differentiate recommendations for users that have generated a lot of data and completely new users. In these experiments, the popularity recommender model will act as a baseline model to which other models can be compared based on their performance.

4.2.2 Item similarity model

Another approach is the item similarity model; this model recommends items based on item-item similarity. This similarity is computed based on observations of the user interactions that these items have in common. In practice, the model ranks items according to their similarities to other items already purchased and viewed by the user in question. There are three different formulas for measuring these similarities: Cosine similarity, Pearson correlation similarity, and Jaccard similarity.

Since the data available for building this recommender system is implicit log data, of users views on articles, the choice of similarity metric needs to match the implicit data without any rating measures. Jaccard similarity only assesses whether a user and item interacted and do not evaluate on the possible rating given by the user for that item, contrary to Cosine and Pearson correlation similarity. Thus, the most applicable choice of metric is Jaccard similarity when dealing with implicit feedback data.

Equation 4.2.1 illustrates the formula for measuring Jaccard similarity. Here the similarity is measured for item i and j. U_i are the set of users who consumed item i, and U_j are the set of users who consumed item j. The intersection between U_i and U_j is denoted $U_i \cap U_j$ and discloses all items present in both sets, whereas the union denoted $U_i \cup U_j$ exposes those items represented in either set (Phillips, 2013).

$$\mathrm{JS}(i,j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

Equation 4.2.1

Equation 4.2.2 gives an example of a Jaccard similarity between i and j being calculated. Item i and j both have recorded four user interactions, of these users

two of them have intreacted with both items. This results in a Jaccard similarity of 33% between items i and j.

$$\begin{matrix} i & j \\ \{1,2,4,5\} & \{2,3,5,7\} \end{matrix}$$

$$= \frac{|\{2,5\}|}{|\{1,2,3,4,5,7\}|}$$

$$= \frac{2}{6} = \frac{1}{3} = 0.334$$

Equation 4.2.2

4.2.3 Ranking factorization model

One last approach used in our experiments is ranking factorization. The ranking factorization recommender works by learning latent factors for all users and items. Then using these latent factors to rank the highest recommended items based on the likelihood of detecting those user and item pairs.

The method behind the ranking factorization model is the factorization machine, introduced by Rendle (Rendle, 2010) which is a generalization of the matrix factorization approach. It works as matrix factorization, with the additional ability to learn latent factors for all variables including side features, which in our case could be the article categories data.

The model works by assigning the raw page view data (r_{ui}) into two separate magnitudes: preferences(p_{ui}) and confidence levels(c_{ui}), of distinct interpretations. These factors are used when training the model to be capable of predicting preferences and confidence levels for every possible user and item combination.

The internal coefficients of the recommender are trained from the pre-known interactions between users and items. Later the recommendations given are based on these interactions.

In the process of optimizing the model's internal coefficients, it learns from all users available in the dataset. Factorization machines break the independence of interaction parameters by factorizing these. Thus, data from one interaction contributes in the estimation of related interaction parameters. These aspects contributes to the models efficiency even when faced with data sets of high sparsity.

When the user and item factors are computed, the model can recommend for user u the K not yet viewed articles with the largest $\hat{p}_{ui}$. $\hat{p}_{ui}$ indicates the predicted preference of user u for item i (Koren, Hu, & Volinsky, 2008).

The ranking factorization model has two different choices for solvers when fed with implicit data. The first one is 'sgd', which is the stochastic gradient descent method. The other one is called implicit alternating least squares, or 'ials', an alternating least squares solver specifically designed for implicit data. The following paragraphs will go through the training of two different ranking factorization models with each different solver and with graphs illustrating their mean precision measure given the number of recommended items.

The IALS solver seeks to minimize the objective function. The goal is to make the sum of weighted squared differences between the true and predicted rankings as small as possible.

This training is done by computing on the P and Q matrices containing users u and items i. The process is executed by initializing Q with small random numbers and then for E number of iterations computing the P that minimizes f_R for fixed Q, followed by computing the Q that minimized f_R for fixed P (Takács & Tikk, 2012). With user or item factors set fixed, the cost function is quadratic. Thus an alternating least squares optimization works well (Koren, Hu, & Volinsky, 2008). f_R denotes the ranking objective function (Takács & Tikk, 2012).

Stochastic gradient descent computes an estimate of the loss, by taking the average loss of a little fraction of the training data. This fraction could be between 1-1000 training instances and picked at random, which is a crucial prerequisite for the method to work. Loss and derivative are computed from each fraction of training data. The computed derivative is treated as the pretended right direction for gradient descent, reducing the real loss. Sometimes this derivative might

increase the real loss and go opposite the right direction, this is however compensated for by doing a lot of iterations, taking small steps each time. This is possible due to the small cost of computing each step. Stochastic gradient descent does more iterations than the original gradient decent method but is still more efficient (Ruder, 2016). The method scales well with a lot of data and big models and is therefore a method used a lot in the area of deep learning.

5 Experiments

5.1 Overview

The following section will go through the whole process of creating a recommendation system. From data handling, collecting the dataset, data pre-processing and splitting data into train and test sets, to training and testing different recommender models, finding the optimal parameters and delivering actual individual article recommendations to users.

The recommender system models are build using Python while utilizing the libraries pandas and GraphLab Create. The Python package pandas is a data analysis toolkit providing a software library for data manipulation and analysis. The tool implements intuitive functions, making work with real-world data and different data analysis tasks simpler to perform (pandas, n.d.). The GraphLab Create library delivers a machine learning modeling toolkit, with a high-level programming interface for straightforward implementations of machine learning models and visualization of their output (Turi, n.d.).

The code example 5.1 imports these dependencies, giving them a shortened nickname for ease of use.

```
In [1]: #Import dependencies
        import pandas as pd
        import graphlab as gl
```

Code example 5.1

In these experiments the pandas package is mainly used for data importing and preprocessing, while the GraphLab Create package is used for making the test, train data split, building, training and evaluating different machine learning model approaches for recommender systems.

5.2 Dataset

The dataset consists of user logs collected since July 2015 from jyllands-posten.dk (Hardt & Rambow, 2017). The dataset includes 253,752,032 data observations on 89,605 different articles.

There are 89,179 distinct registered users, meaning that they have logged into Jyllands-Posten's website: jyllands-posten.dk.

For this research paper, the data observations are restricted to those generated from article clicks. Many data entries are from users entering the front page, clicking onto the premium page, etc. Since the purpose of this paper is article recommendations, the dataset is restricted to only include article views from registered users.

The dataset consists of implicit feedback data since there is no explicit feedback from users in the data; hence, the data available is extracted from user behavior. Observations about the users click views on different sites of the jyllands-posten.dk website are composing the feedback data. The challenge when dealing with implicit data is the lack of negative feedback. The data observations gives no indications about disliking any articles. The only thing to withdraw from the feedback is what the users probably liked, based on their viewing of different articles (Koren, Hu, & Volinsky, 2008).

Another challenge faced when using implicit data for building recommender systems are the lack of literature on the subject. This is strange since implicit data is more common to come across in real-world examples than explicit feedback data. There is a lot of implicit data to be collected by user behavior online, but limited explicit data from users actively giving feedback on a given product, service or good. Despite this being the case, the vast majority of literature and examples on the topic of recommender systems are conducted on explicit feedback. A reasoning might be the Netflix prize challenge from 2009 where the streaming company Netflix awarded $1 million dollars to the person or team that

could come up with a recommender system algorithm to beat theirs by at least 10% (Gomez-Uribe & Hunt, 2016). The economic incitement and attention inspired a lot of research in the field of recommender systems with explicit feedback data.

The Jyllands-Posten dataset is stored in a Postgres SQL database and to build the recommendation system different SQL queries were called on the database to extract the data needed for building and testing the different models.

Below is an example of the SQL query executed within the database, which contains various tables with everything from visit data to article categories and actual titles and text bodies on all published articles. The query selects sso_id's and content id's from the views_complete table, containing data of all page views, i.e., every time a user has opened up a web page on jyllands-posten.dk.

To exclude blank entities in the sso_id, the NOT IN ('NULL', 'NOTSET',' ') is included in the query. Furthermore, the only relevant instances to look at is articles, without considering page views of the front page or other pages within the website that are not articles. This is why the page_id LIKE '%.art%' is executed, to only return page_id rows with .art. in the name, which all articles have in their page_id name.

```
SELECT sso_id, content_id
FROM views_complete
Where sso_id NOT IN ('NULL', 'NOTSET','') AND page_id LIKE '%.art.%';
```

Query 5.2

Table 5.2 illustrates the output from the SQL query in ‘Query 5.2’ when called on the jp database. This output is stored in a comma-separated file (csv).

sso_id	content_id
af0f9208-0ecb-4a77-b451-672641a99705	9409190
d5cfd1aa-ffbf-4d8d-ada3-b9383e05ed15	9409171
5f9dbd21-356a-4eb4-a150-70f9a221cf02	9409190
d919e461-5f4b-4f57-8365-17da17585c74	9412675
bcdc0ba0-c1cd-48f6-9b4e-3f779abca6c8	9398666
f45bbed9-6aca-4c20-82cf-430f12e7f7bf	9391092
5a84f8d6-e12e-4e69-b472-1e63c01f5499	9401495
65442b8f-c662-4814-94fe-8eeae5b864a2	9398658
fdd87860-06bb-4c67-858d-81bb01cd9bef	8612221
400b2b63-d6c0-4b0c-acd7-5dd9fd435b39	9412666
347ff745-c5ea-433d-8f8e-ec435a0e5b80	9412675
cffcf618-dafd-427a-9048-6d7beab799f8	9398752
9e825915-dd39-499f-9539-26075f8a9db8	9398314

Table 5.2

5.3 Data preprocessing

Now that data is extracted from the database; the next step is data preprocessing to groom the dataset into something suitable for training and testing recommender models.

Code example 5.3.1 loads in the csv formatted and converts it to a pandas data frame containing the viewing data of all logged-in users. The data frame contains the columns ‘uid’, which is the user id and ‘aid’ short for the unique article id of each different articles. Calling the df.head() function returns an output visualizing the first five rows of the data frame with the defined column names from the line above.

In [2]:
```
#Load in dataset from csv file
df = pd.read_csv('jp_data.csv', sep=',')
df.columns = ['uid','aid']
df.head()
```

Out[2]:

	uid	aid
0	af0f9208-0ecb-4a77-b451-672641a99705	9409190
1	d5cfd1aa-ffbf-4d8d-ada3-b9383e05ed15	9409171
2	5f9dbd21-356a-4eb4-a150-70f9a221cf02	9409190
3	d919e461-5f4b-4f57-8365-17da17585c74	9412675
4	bcdc0ba0-c1cd-48f6-9b4e-3f779abca6c8	9398666

Code example 5.3.1

Now that the dataset has been loaded into data frames, some data preprocessing is performed to optimize later model performance.

Duplicates are removed to ensure that each user and item pair only appear once in the dataset. The removing of duplicates is executed in code example 5.3.2 with the df.drop_duplicates method.

In [3]:
```
#Remove dublicates from dataset
print('Number of rows: ' + str(len(df)))
print('Duplicated rows: ' + str(df.duplicated().sum()))
print('Drop duplicates')
df.drop_duplicates(inplace=True)
print('Remaining rows: ' + str(len(df)))
```

```
Number of rows: 9151970
Duplicated rows: 1831679
Drop duplicates
Remaining rows: 7320291
```

Code example 5.3.2

Another step in preprocessing the data set is setting the thresholds. Thresholds assess the minimum number of articles that a user has read and the minimum number of users to have read a certain article. Altering the thresholds determines the number of interactions that a user has to have in the dataset for the user to be contained in the train and test data that the recommender systems are trained

and tested on.

Code example 5.3.3 counts the interactions between uid and aid in a while loop. The minimum of uid and aid are set in the threshold functions parameters. The output at the bottom illustrates the users and articles included in the dataset before and after running the thresholds function.

```
In [4]: #Defining threshold function
def thresholds(df, uid_min, aid_min):
    n_users = df.uid.unique().shape[0]
    n_items = df.aid.unique().shape[0]
    print('Starting views info')
    print('Number of users: {}'.format(n_users))
    print('Number of articles: {}'.format(n_items))

    done = False
    while not done:
        starting_shape = df.shape[0]
        aid_counts = df.groupby('uid').aid.count()
        df = df[~df.uid.isin(aid_counts[aid_counts < aid_min].index.tolist())]
        uid_counts = df.groupby('aid').uid.count()
        df = df[~df.aid.isin(uid_counts[uid_counts < uid_min].index.tolist())]
        ending_shape = df.shape[0]
        if starting_shape == ending_shape:
            done = True

    assert(df.groupby('uid').aid.count().min() >= aid_min)
    assert(df.groupby('aid').uid.count().min() >= uid_min)

    n_users = df.uid.unique().shape[0]
    n_items = df.aid.unique().shape[0]
    print('Ending views info')
    print('Number of users: {}'.format(n_users))
    print('Number of articles: {}'.format(n_items))
    return df

In [5]: #Determining thresholds
df_th = thresholds(df,10,10)

Starting views info
Number of users: 89179
Number of articles: 89605
Ending views info
Number of users: 37563
Number of articles: 27919
```

Code example 5.3.3

When duplicates have been removed, and thresholds have been set the next step is to convert the data frame df into an SFrame, which is the format used to train recommenders in the GraphLab environment. Once converted, the SFrame is split into a train and test dataset for training and testing the recommender models. The random split is done by choosing 1000 random users and subsequently randomly taking 30% of those users' interactions for use in the test set. This is executed by the random_slip_by_user GraphLab function. The whole process is executed in code example 5.3.4.

```
In [6]: #Converting dataframe to SFrame
        #Splitting SFrame into train and test set
        sf = gl.SFrame(df_th)
        train, test = gl.recommender.util.random_split_by_user(sf,user_id='uid',item_id='aid',max_num_users=1000,
                                                               item_test_proportion=0.3)
        print(train)
        print(test)
```

```
This non-commercial license of GraphLab Create for academic use is assigned to miso12aj@student.cbs.dk and will expire on August
19, 2018.

[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: C:\Users\MIKSOE~1\AppData\Local\Temp\graphlab_server_15
04357852.log.0
```

uid	aid
af0f9208-0ecb-4a77-b451-67...	9409190
d5cfd1aa-ffbf-4d8d-ada3-b9...	9409171
5f9dbd21-356a-4eb4-a150-70...	9409190
d919e461-5f4b-4f57-8365-17...	9412675
bcdc0ba0-c1cd-48f6-9b4e-3f...	9398666
f45bbed9-6aca-4c20-82cf-43...	9391092
5a84f8d6-e12e-4e69-b472-1e...	9401495
fdd87860-06bb-4c67-858d-81...	8612221
347ff745-c5ea-433d-8f8e-ec...	9412675
cffcf618-dafd-427a-9048-6d...	9398752

```
[5793489 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

uid	aid
0d0f93dd-4955-4ab0-bc1d-78...	9397027
e7bb9cb4-aecf-4a45-be33-75...	9401884
7b273a36-fb30-4f7f-a7ae-3c...	9409186
ddd94d83-98f9-4efe-a1bd-53...	9397954
6b5c4118-6b19-4293-bb59-1c...	9404733
e1eb465a-7486-4dc7-a002-33...	9407391
ea169d2b-5edc-443b-be88-d9...	9408761
e7bb9cb4-aecf-4a45-be33-75...	9401803
9cc407c7-fadd-4312-adde-44...	9412141
ea169d2b-5edc-443b-be88-d9...	9409144

```
[107666 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

Code example 5.3.4

5.4 Training model

Using the Sframe ‘train’, the different models are initialized and trained on this data at this stage.

The ranking factorization recommender has an option optimized for ranking implicit data using the implicit matrix factorization model. This model addresses the unique characteristics of implicit data. To utilize the capabilities of this model feed in ‘user_id’ and ‘item_id’ data without any ‘target’ column when creating a ranking factorization recommender to choose the implicit matrix factorization model by default (Turi, 2016). The ‘target’ column is the one that would contain any available rating data in case of an explicit dataset.

The same goes for the item similarity model. Here the same user_id and item_id from the ‘train’ Sframe is fed into the item_similarity_recommender.create function. As discussed in the section theory and methodology the Jaccard similarity is the solver used in this model.

The popularity model is just trained to function as a baseline model for the other approaches to compare.

The create functions initializing and training the different recommender systems are displayed in code example 5.4.

```
In [7]: #Train recommendation models

        #Popularity model used as baseline
        popularity_model = gl.popularity_recommender.create(train, user_id='uid', item_id='aid')

        #Item similarity recommender
        item_sim_model = gl.item_similarity_recommender.create(train, user_id='uid', item_id='aid',
                                                               similarity_type='jaccard')
        #Ranking factorization recommender
        rank_fac_model = gl.ranking_factorization_recommender.create(train, user_id='uid',
                                                                     item_id='aid',solver='ials')
```

Code example 5.4

5.5 Testing models

This section will go through tests of different recommender models. Altering various parameters and comparing the results.

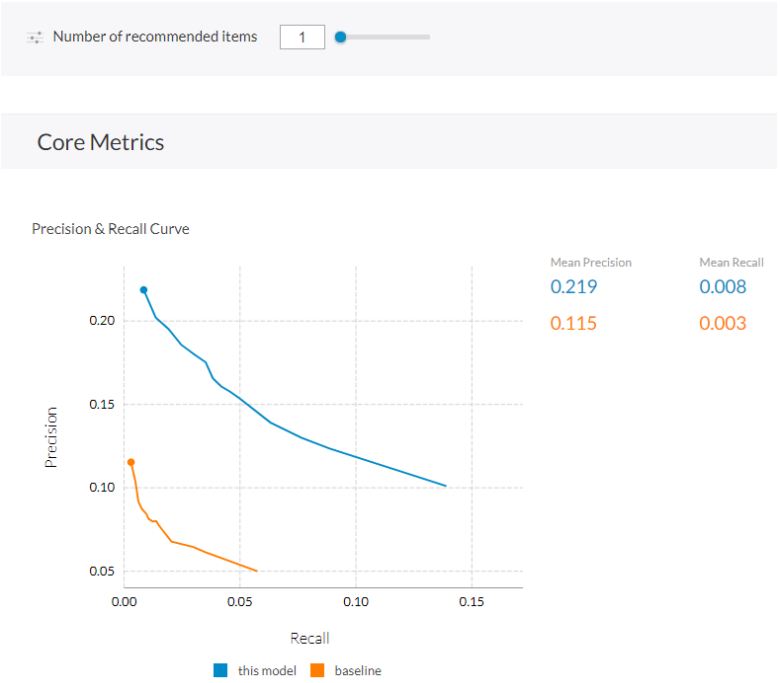
5.5.1 Evaluation metrics

The basic goal of the model is to deliver recommendations for users. To measure the performance of the model, it needs evaluation measures. The first tests are done using precision and recall.

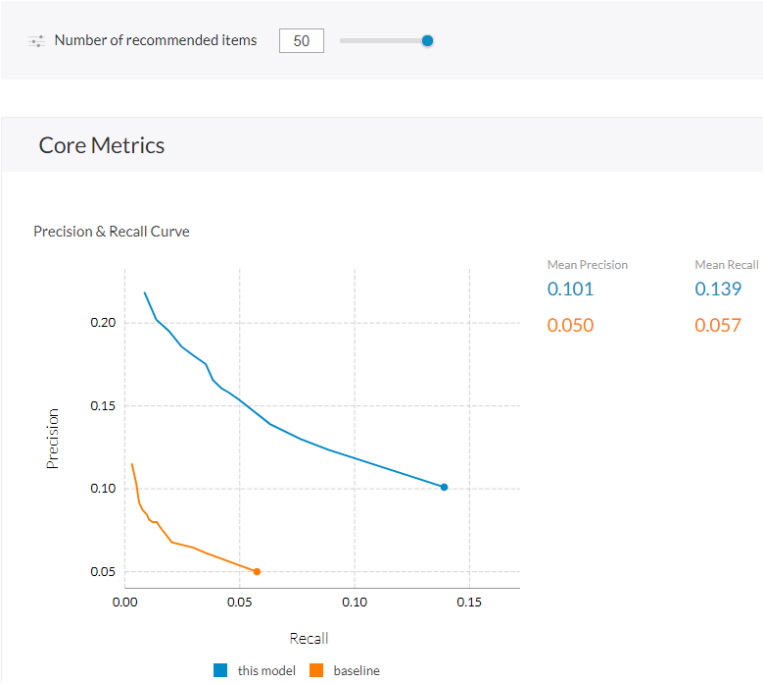
Precision is the definition of the measure; out of all the recommended items shown to the user, how many did the user actually like? Thus if our model recommends top 5 articles to a user based on the test set, and the user actually viewed 2, the precision calculated in this example would be 0.4 or 40%.

Recall is a measure of the ratio of items a user likes that actually were recommended by the model. Say a user likes 10 items and the recommender shows him 2 of these items, the recall would be 0.2 or 20%.

The two graphs 5.5.1 and 5.5.2 clearly illustrates is the inverse relationship between precision and recall as the number of recommended articles is altered.



Graph 5.5.1



Graph 5.5.2

Precision drops by more than 50% as we go from recommending 1 item to 50 items, while recall increase significantly.

The reason behind is that precision only considers the actual recommended articles and if this or these article(s) are in the selected number of

recommendations, whereas recall measures the ratio between recommended articles to the total amount of articles read. So increasing the articles attempted to recommend will always result in a higher recall, while precision will suffer tremendously.

As the goal of this research paper is to recommend a fixed number of articles. With recall, a user who reads 100 articles and only are recommended three, would only result in a 3% recall despite them all being correct. In comparison, a user who only read 10 articles, but got 3 recommendations and only one correct 'hit' would result in a recall of 10%. Taking this inverse relationship between precision and recall into consideration, the remainder of the experiments are done by only assessing the precision metric.

5.5.1.1 Comparing models

The popularity, item similarity, and ranking factorization models are compared using the compare_models GraphLab function, defining the models to compare, the metric to evaluate the models, in this case precision_recall, and feeding in the test data SFrame for the models to be evaluated on.

This process is displayed in code example 5.5.1. The output are three different tables of the different performance metric values at different recommendation cutoffs.

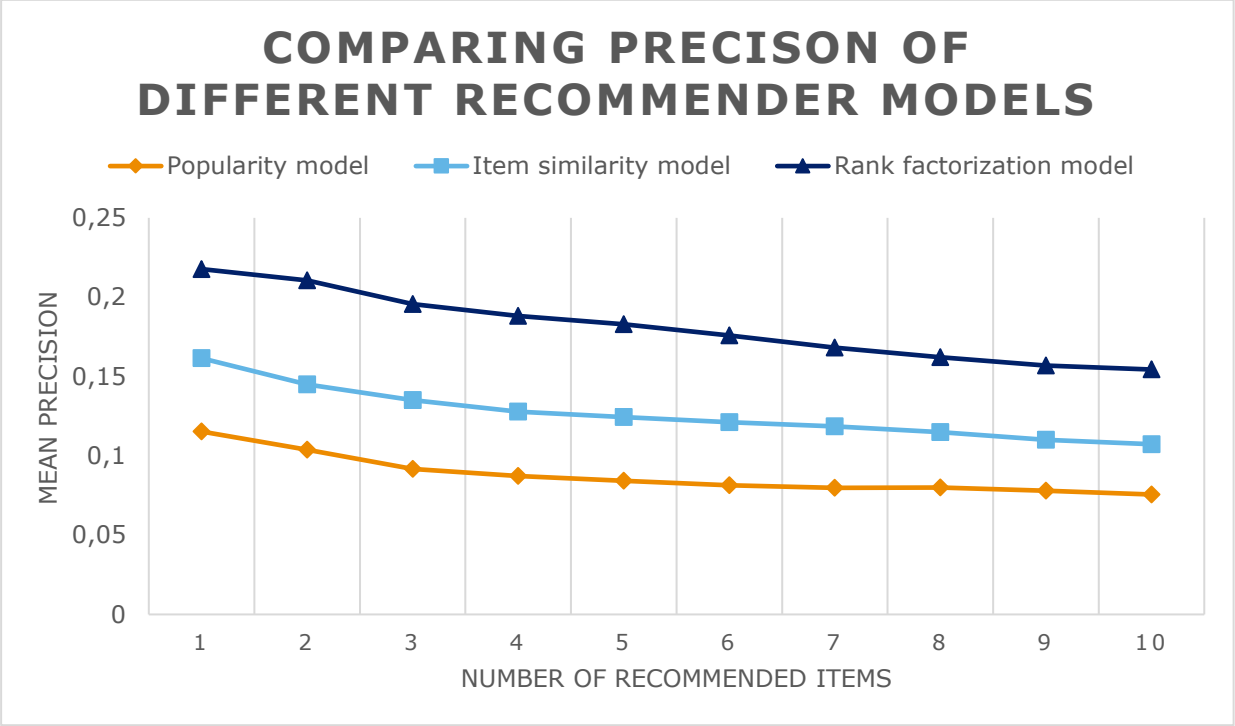
```
In [12]: #Comparing models on test set
         gl.recommender.util.compare_models(test,[popularity_model,item_sim_model, rank_fac_model],
                                            ['Popularity model','Item similarity recommender','Ranking factorization recommender'],
                                             metric='precision_recall')
```

```
PROGRESS: Evaluate model Popularity model

Precision and recall summary statistics by cutoff
+--------+-----------------+-------------------+
| cutoff |  mean_precision |    mean_recall    |
+--------+-----------------+-------------------+
|   1    |  0.115346038114 | 0.00301180256428  |
|   2    |  0.103811434303 | 0.00490184571683  |
|   3    | 0.0916081578068 | 0.00622095604541  |
|   4    | 0.0872617853561 | 0.0077303158753   |
|   5    | 0.0842527582748 | 0.00967831561167  |
|   6    | 0.0814108993648 | 0.0105677303102   |
|   7    | 0.0798108611549 | 0.0123400987594   |
|   8    | 0.0799899699097 | 0.0139531860326   |
|   9    |  0.07790036777  |  0.014829474799   |
|   10   | 0.0756268806419 | 0.0160217798077   |
+--------+-----------------+-------------------+
[10 rows x 3 columns]

PROGRESS: Evaluate model Item similarity recommender

Precision and recall summary statistics by cutoff
+--------+----------------+-------------------+
| cutoff | mean_precision |    mean_recall    |
+--------+----------------+-------------------+
|   1    | 0.16148445336  | 0.00572635707243  |
|   2    | 0.144934804413 | 0.0109166367509   |
|   3    | 0.135071882314 | 0.0155923192612   |
|   4    | 0.127883650953 | 0.0187903618421   |
|   5    | 0.124373119358 | 0.0228909189754   |
|   6    | 0.121196924106 | 0.0269081017044   |
|   7    | 0.118641639203 | 0.0292061502239   |
|   8    | 0.114969909729 | 0.0315686147982   |
|   9    | 0.110108102084 | 0.0336005484594   |
|   10   | 0.107321965898 |  0.035486047792   |
+--------+----------------+-------------------+
[10 rows x 3 columns]

PROGRESS: Evaluate model Ranking factorization recommender

Precision and recall summary statistics by cutoff
+--------+----------------+-------------------+
| cutoff | mean_precision |    mean_recall    |
+--------+----------------+-------------------+
|   1    | 0.217652958877 | 0.00823416119483  |
|   2    | 0.210631895687 | 0.0144199303669   |
|   3    | 0.195586760281 | 0.0202051768967   |
|   4    | 0.188064192578 | 0.0251467546235   |
|   5    | 0.18294884654  | 0.0305724520098   |
|   6    | 0.175860916082 |  0.035259551191   |
|   7    | 0.168218942542 | 0.0384894757358   |
|   8    | 0.16223671013  | 0.0416171196203   |
|   9    | 0.156803744567 | 0.0445738900361   |
|   10   | 0.154363089268 | 0.0480198035885   |
+--------+----------------+-------------------+
[10 rows x 3 columns]
```

Code example 5.5.1

This table data is visualized in graph 5.5.3. It shows obvious distinctions in precision performance between the three models. Both the item similarity model and ranking factorization model are outperforming the popularity baseline model by a great deal. For comparison; at 3 recommendations, the baseline model's precision is about 10%, the item similarity model is about 15%, and the ranking factorization model is about 20%.

Next, the ranking factorization model will be tested with different solvers to try and improve this best performing model.



Graph 5.5.3

5.5.1.2 Testing different solvers

The ranking factorization model has two different choices for solvers when fed with implicit data. The first one is 'sgd', which is the stochastic gradient descent method. The other one called implicit alternating least squares, or 'ials', an alternating least squares solver specifically designed for implicit data. The following paragraphs will go through the training of two different ranking factorization models with each different solver and with graphs illustrating their mean precision given the number of recommended items.

The IALS solver seeks to minimize the objective function. The goal is to make the sum of weighted squared differences between the true and predicted rankings as small as possible.

This training is done by computing on the *P* and *Q* matrices containing users *u* and items *i*. The process is executed by initializing *Q* with small random numbers and then for *E* number of iterations computing the *P* that minimizes f_R for fixed *Q*, followed by computing the *Q* that minimized f_R for fixed *P* (Takács & Tikk, 2012). With user or item factors set fixed, the cost function is quadratic. Thus an alternating least squares optimization works well (Koren, Hu, & Volinsky, 2008). f_R denotes the ranking objective function (Takács & Tikk, 2012).

Stochastic gradient descent computes an estimate of the loss by taking the average loss of a little fraction of the training data. This fraction should be between 1-1000 training instances and picked at random, which is a crucial prerequisite for the method to work. Loss and derivative are computed from each fraction of training data. The computed derivative is treated as the pretended right direction for gradient descent, reducing the real loss. Sometimes this derivative might increase the real loss and go opposite the right direction. However, this is compensated for by doing a lot of iterations, taking small steps each time. This is possible due to the small cost of computing each step. Stochastic gradient descent does more iterations than the original gradient decent method but is still more efficient (Ruder, 2016). The method scales well with a lot of data and big models and thus a method used a lot in the area of deep learning.

In illustration 5.5 the process of stochastic gradient descent is depicted. Taking a lot of small steps, not always optimal, but reach a loss minimum at the end of iterations.

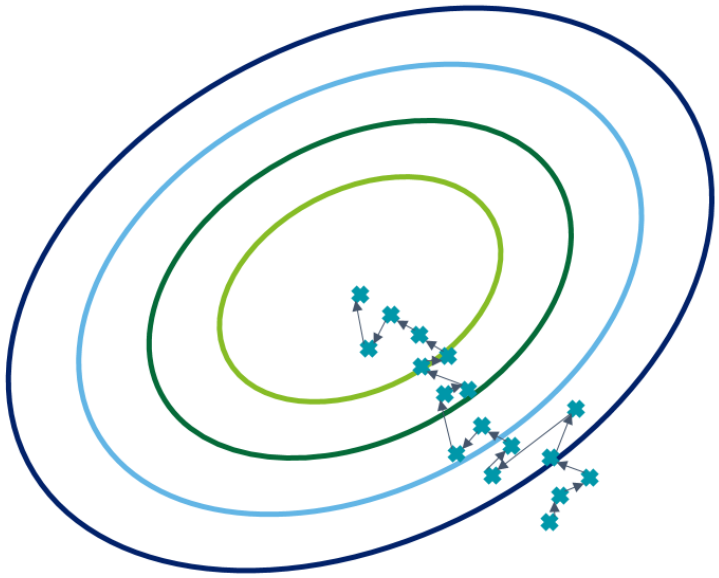


Illustration 5.5

Code 5.5.2 is extracted from the python notebook and is the part that initializes two different ranking factorization recommenders with the .create function. The solver is what differentiates these models initiated with the solver set to 'ials' for the rank_fac_model_ials model and 'sgd' for the rank_fac_model_sgd model.

```
In [7]: #Train ranking factorization recommendation models with different solvers

rank_fac_model_ials = gl.ranking_factorization_recommender.create(train, user_id='uid',
                                                                  item_id='aid',solver='ials')


rank_fac_model_sgd = gl.ranking_factorization_recommender.create(train, user_id='uid',
                                                                 item_id='aid',solver='sgd')
```

Code example 5.5.2

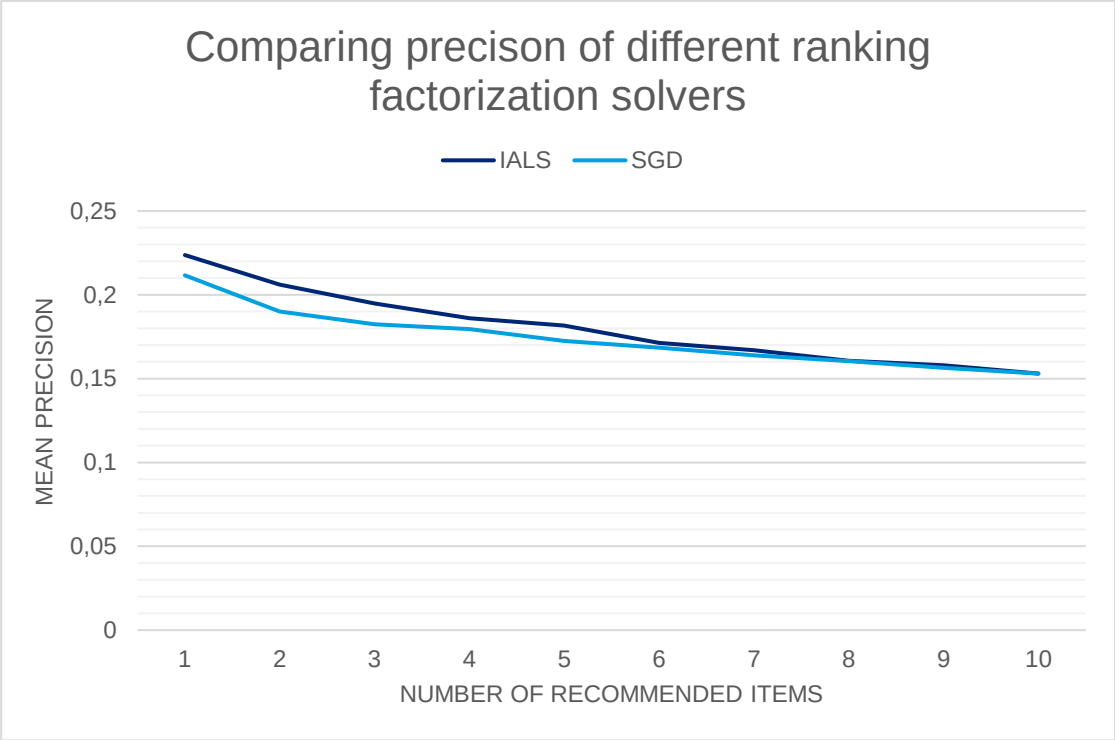
Code 5.5.3 compares the two models created on their performance on the test set. Using the metrics precision and recall to assess their relative performance, the output are two tables with precision and recall values for each model at different cutoffs. As discussed in the section 'evaluation metrics', precision is the only measurement relevant and hence recall is discarded for analysis.

In [8]:
```
#Comparing models on test set
gl.recommender.util.compare_models(test,[rank_fac_model_ials, rank_fac_model_sgd],
                                   ['IALS','SGD'], metric='precision_recall')
```

```
PROGRESS: Evaluate model IALS

Precision and recall summary statistics by cutoff
+--------+----------------+------------------+
| cutoff | mean_precision |   mean_recall    |
+--------+----------------+------------------+
|   1    | 0.223671013039 | 0.00823336777339 |
|   2    | 0.206118355065 | 0.0138658727598  |
|   3    | 0.194918087596 | 0.0198684647166  |
|   4    | 0.186058174524 | 0.0245570601268  |
|   5    | 0.181745235707 | 0.0301988806553  |
|   6    | 0.17134737546  | 0.0329478744344  |
|   7    | 0.166929359507 | 0.0372512583146  |
|   8    | 0.160606820461 |  0.04078321057   |
|   9    | 0.157918199042 |  0.043669702432  |
|   10   | 0.15295887663  | 0.0470453384963  |
+--------+----------------+------------------+
[10 rows x 3 columns]

PROGRESS: Evaluate model SGD

Precision and recall summary statistics by cutoff
+--------+----------------+------------------+
| cutoff | mean_precision |   mean_recall    |
+--------+----------------+------------------+
|   1    | 0.211634904714 | 0.00734322878316 |
|   2    | 0.190070210632 |  0.013650608162  |
|   3    | 0.182547642929 | 0.0193107396767  |
|   4    | 0.179538615848 | 0.0250725106849  |
|   5    | 0.172517552658 | 0.0292951258976  |
|   6    | 0.16850551655  | 0.0344749039241  |
|   7    | 0.163920332426 | 0.0386457896785  |
|   8    | 0.160356068205 | 0.0426165829001  |
|   9    | 0.156357962777 | 0.0457779993902  |
|   10   | 0.15295887663  | 0.0489234024723  |
+--------+----------------+------------------+
[10 rows x 3 columns]
```

Code example 5.5.3

Graph 5.5.4 is created to give a better overview of the models, and their performances measured on precision.



Graph 5.5.4

Looking at the graph 5.5.4 the IALS solver clearly outperforms SGD up until the cutoff at 5 recommended items, thereafter the two solvers perform similar from cutoffs 6 to 10.

Due to this discovery, the IALS solver will be the one used going further into testing models with added side info and at different thresholds.

5.5.1.3 Adding side info

In an attempt to optimize recommender systems performance, side information is added to the recommender models to measure if this can improve precision.

Code example 5.5.4 loads in the csv file 'category.csv', which contains the data about the category related to each aid. In the same procedure as the data preprocessing, the csv file is loaded into a pandas data frame and subsequently converted to an SFrame.

In [8]:
```
#Load in side info
side_df = pd.read_csv('category.csv', sep=',')
side_df.columns = ['aid','category']
side_df.head()
```

Out[8]:

	aid	category
0	8798646	indland
1	9120180	indblik
2	9386039	aarhus
3	9096606	sport
4	9354492	article

In [9]:
```
#Converting dataframe to SFrame
side_sf = gl.SFrame(side_df)
print(side_sf)
```

```
+---------+---------------+
|   aid   |    category   |
+---------+---------------+
| 8798646 |    indland    |
| 9120180 |    indblik    |
| 9386039 |     aarhus    |
| 9096606 |     sport     |
| 9354492 |    article    |
| 7745211 |    livsstil   |
| 9337468 |    indblik    |
| 9204233 | international |
| 9086419 |    article    |
| 9213980 |    indland    |
+---------+---------------+
[129428 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

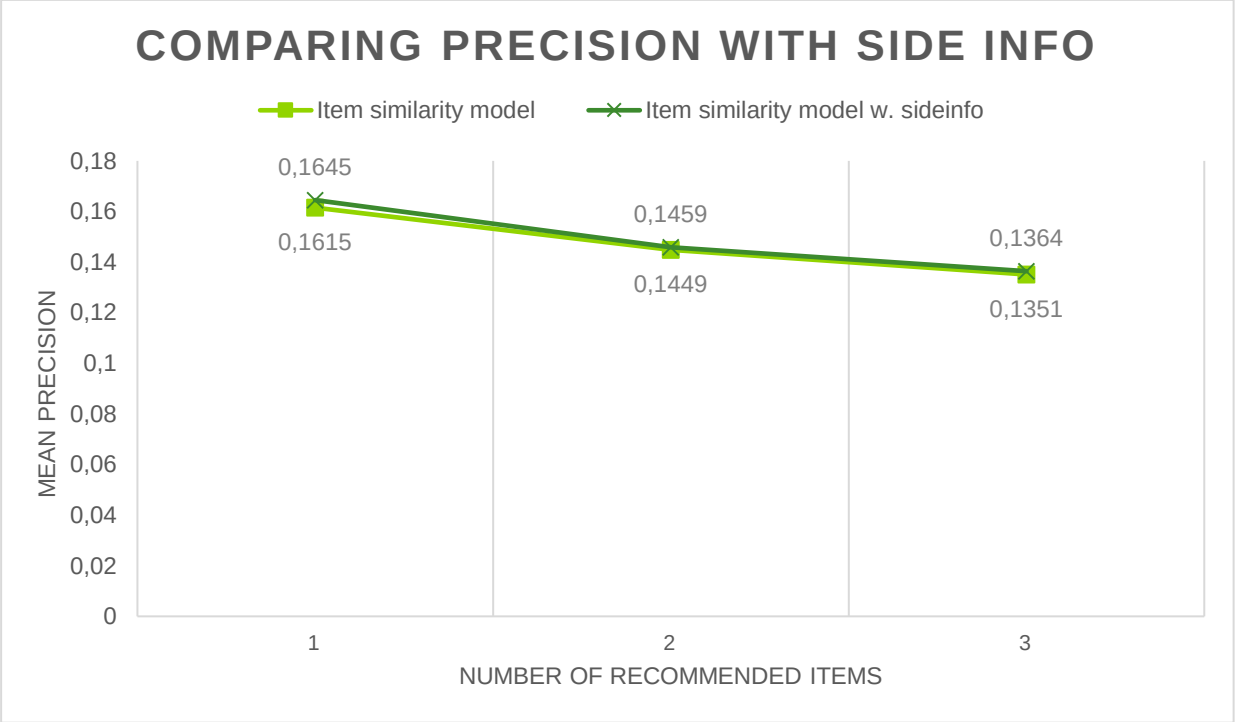
Code example 5.5.4

As in the code from the training model section, code example 5.5.5 initializes and trains models using the create function. The difference here is the item_data, which is set equal to the SFrame 'side_sf' just created.

In [10]:
```
#Train recommendation models with sideinfo for items

#Item similarity recommender with sideinfo for items
item_sim_model_sideinfo = gl.item_similarity_recommender.create(train, user_id='uid', item_id='aid',
                                                                item_data=side_sf, similarity_type='jaccard')

#Ranking factorization recommender with sideinfo for items
rank_fac_model_side = gl.ranking_factorization_recommender.create(train, user_id='uid', item_id='aid',
                                                                  item_data=side_sf,solver='ials')
```

Code example 5.5.5

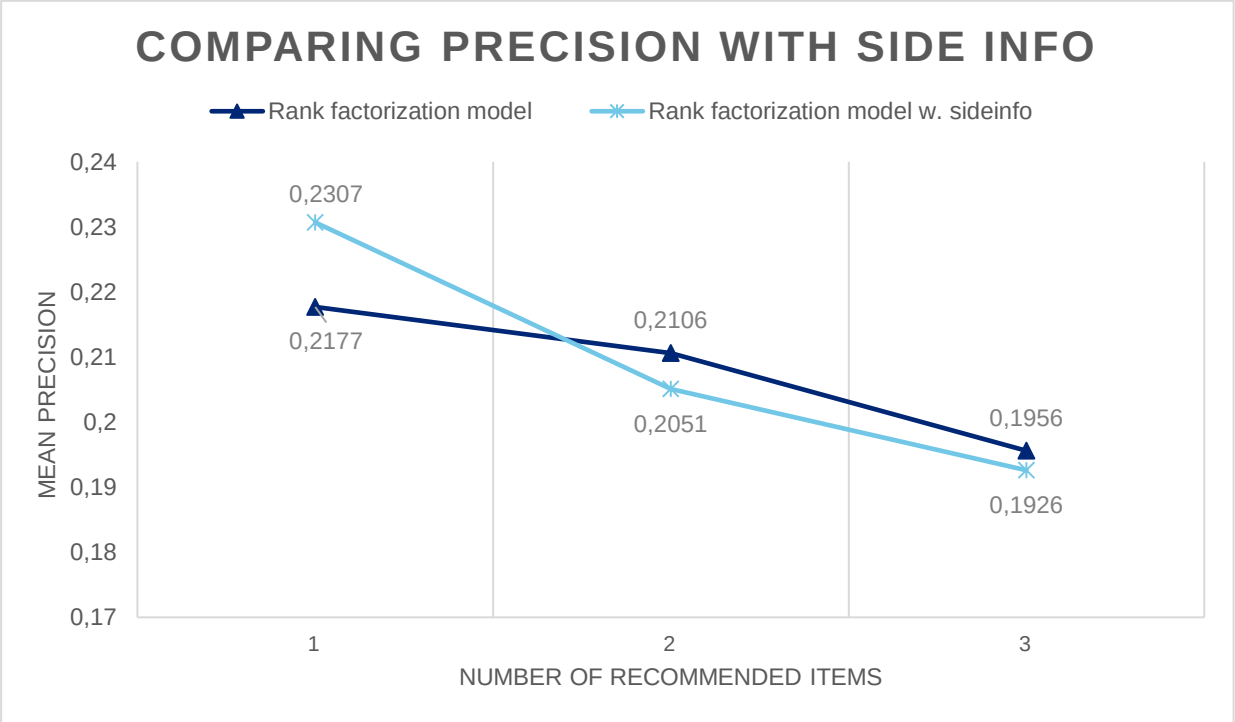
Graph 5.5.5 compares the item similarity models, with and without side information. They perform almost identically, with the side info recommender being slightly better on precision. However, nothing significant.



Graph 5.5.5

When comparing the ranking factorization models with and without side information, the model performance differs a bit more. The interesting observation from graph 5.5.6 is that the recommender with side information has the highest precision when only recommending one item, while the ranking factorization model without side information performs better when recommending 2 or 3 articles.

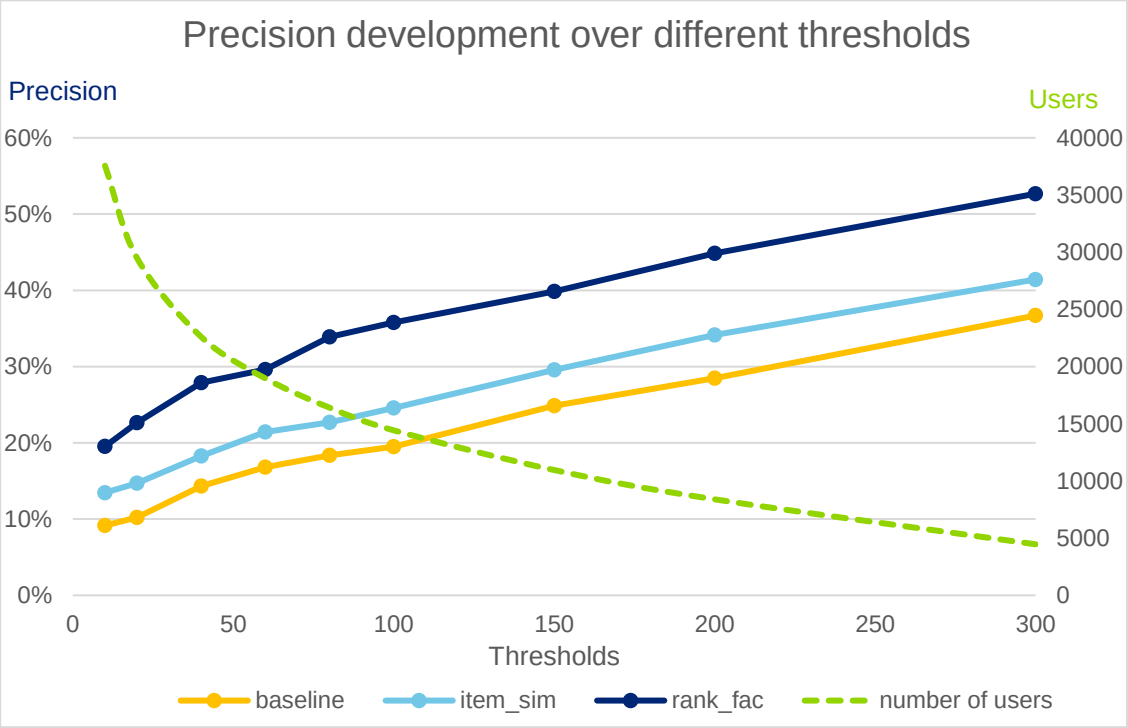
Since the goal is to recommend 3 articles to users, further experiments will be done on the ranking factorization model without side information.



Graph 5.5.6

5.5.1.4 Altering thresholds

The model performance differs a lot when altering the thresholds set in preprocessing of the dataset. As thresholds are set higher, precision improves, but at the cost of the number of users in the dataset. Graph 5.5.7 illustrates the effect that the value of thresholds has on the baseline, item similarity and ranking factorization models performance metric precision. The number of users left, qualifying for each threshold is represented by the green spaced line, the users metric is shown on the right y-axis.
All model precisions are measured given 3 item recommendations.



Graph 5.5.7

Choosing a threshold to use for building the recommendation system is tricky since it is a trade-off between the number of users available and model precision. Hence it is a consideration, assessing both measures. It can be concluded from the graph 5.5.7 that as more data about user activity is gathered, the recommendations made from the models are increasingly precise and thus valuable.

With the assessment that each user gets three recommendations and we wish at least one of the recommendations to be valuable to the user the choice of threshold was 80. With 80 as the threshold, making 3 recommendations for each user, a precision of 33.5% is reached, and it is safe to assume that in most cases, statistically, the recommender system will get at least one recommendation right. The threshold could be set even higher to improve precision further but at the cost of the number of users available to make recommendations for.

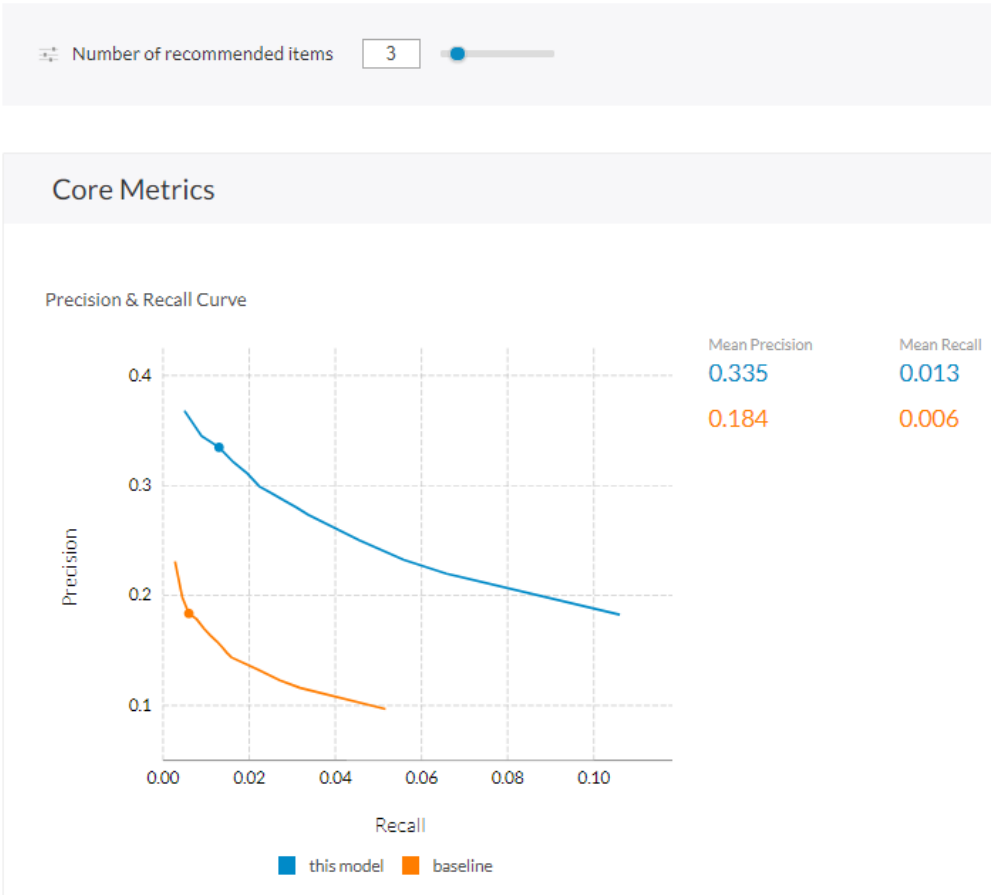
5.6 Results

The function overview in code example 5.6.1 is called to assess the final ranking factorization model compared to the baseline popularity recommender model and measured on the test set.

```
In [26]: #view examples of recommendations based on test set
         view = rank_fac_model.views.overview(validation_set=test,
                                              baseline=popularity_model,
                                              item_data=title_sf,
                                              item_name_column='title')
         view.show()
```

Code example 5.6.1

Graph 5.6 illustrates the model performance measured on precision. As clearly stated in the graph, the ranking factorization model outperforms the baseline model by far. The precision at 3 recommendations for the ranking factorization model is 33.5%, well above in comparison to the popularity model with 18.4% precision.



Graph 5.6

To display the recommendations in a more meaningful way, another csv file containing the title of the articles is loaded in as the SFrame title_sf in code example 5.6.2.

```
In [24]: title_sf = gl.SFrame.read_csv('aid_titles.csv',
                                column_type_hints={'content_id':int,'title':str})
         title_sf.rename({'content_id': 'aid'})
         print(title_sf)

Finished parsing file C:\Users\miksoerensen\Documents\Python Scripts\aid_titles.csv

Parsing completed. Parsed 100 lines in 0.172494 secs.

Finished parsing file C:\Users\miksoerensen\Documents\Python Scripts\aid_titles.csv

Parsing completed. Parsed 115540 lines in 0.126351 secs.

+---------+------------------------------+
|   aid   |             title            |
+---------+------------------------------+
| 4676534 | Naturkatastrofer tog 30.00... |
| 4680165 | Tysk spøgelses-fly hentet ... |
| 6216237 | Sådan fungerer EU's handel... |
| 4656811 | Vandt mio. i usædvanlig he... |
| 7969376 | Syg modelbranche? Model me... |
| 8766281 | Fusk med forskningsmidler ... |
| 4571228 |         Lars Skov (DF)       |
| 4955783 | Google fejrer eventyrlige ... |
| 3277287 |  Nyt navn til Avedøre-lejren  |
| 8123674 | IS og al-Qaeda må se langt... |
+---------+------------------------------+
[115540 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

Code example 5.6.2

Each title is assigned a certain article id as illustrated in code example 5.6.2.

Following are a few examples of real recommendations made by the ranking factorization recommender system on randomly chosen users.

Recommendation example 5.6.1 is displaying three recommended articles to a user who have read 116 articles previously. The recommendation suggestion seems to be reasonably within the same ballpark, with at least two dealing with the topic of some relationship advice.

Focus User

	uid	_num_items
	77225ad5-8c23-4212-	116

Top Recommended Items

Score	title	aid	rank
0.31	Nødråb til sexologen: Hvordan får jeg liv i ham sofadyret?	9122003	1
0.30	Når kærligheden kollapser – sådan undgår I at gå fra hinanden	9023125	2
0.29	Ryk i hemmelig snor skruede tiden 70 år tilbage	8356670	3

Recommendation example 5.6.1

Recommendation example 5.6.2 has three very high precision rated articles for a user who have read 626 articles previously. The user is presented with articles on war and political issues, which is probably something that the user has read a lot about previously.

Focus User

	uid	_num_items
	dd3cfd20-504f-4320-9b6c-6f8da167e117	626

Top Recommended Items

Score	title	aid	rank
0.90	Her vil Trump komme til kort	9404744	1
0.71	Kreml og medier vil lokke russerne til at blive hjemme, men flere og flere rejser mod vest	9095476	2
0.70	Russisk hangarskib er i så dårlig stand, at det sejler med en slæbebåd ved sin side	9095190	3

Recommendation example 5.6.2

The articles presented in recommendation example 5.6.3 also seems reasonably aligned with two of them addressing some navy stories and one about formula 1.

Focus User

	uid	_num_items	
	1ab695be-688d-4a78-908b-9b41d7f197c8	154	

Top Recommended Items

Score	title	aid	rank
0.60	Russisk hangarskib er i så dårlig stand, at det sejler med en slæbebåd ved sin side	9095190	1
0.52	USA's flåde tager nyt superskib i brug	9079587	2
0.50	Magnussens holdkammerat frustreret: Meget hurtigere end Kevin	9098600	3

Recommendation example 5.6.3

The last recommendation example 5.6.4, also has some articles along the same lines. The amazing observation is that the system actually recommends the top two articles concerning the same person, without giving any text input data to train on. The algorithm does not know that the same name are present in the top two recommendations, these recommendations are based on a or some similar user(s) probably reading both articles about Anja Andersen.

Focus User

	uid	_num_items	
	60eee81e-4d45-4cab-99b0-4aecb8bba0ba	362	

Top Recommended Items

Score	title	aid	rank
0.62	Anja Andersen afslører hemmeligheder om Wilbek-relation	9082842	1
0.53	Anja Andersen fandt sin motivation i faderens vrede	9080526	2
0.52	Skil dig ud	9092812	3

Recommendation example 5.6.4

To conclude it seems like the recommendations made by the recommender system makes logically sense along with the statistical argument that at least one of them should be appropriate on average.

6 Conclusion

The proposed approach to improving the online experience of users on jyllands-posten.dk is by giving valuable recommendations to users, by utilizing user data and recommendation systems.

The proposed recommender system solution is a ranking factorization recommender as it have proved to be the best performing model on the data available. The models performance is way above baseline when implemented on users with a minimum amount of historical data, and the recommender system precision improves as user data accumulates, through the interaction with content on jyllands-posten.dk.

With the implementation of the recommender system on the jyllands-posten.dk website, Jyllands-Posten will be better capable of utilizing the long-tail of their published content. Of course, some news are only relevant the day or week of it being published. However, some articles debating different subjects or analyzing a certain situation can be relevant for years. This is where the recommendation system really will improve how the users experience is on jyllands-posten.dk in what content they are presented with. Preferably more relevant to them being a user specific recommendation than some generic recommendation of content that tries to fit to everyone's taste.

The proposed recommendation system solution is only applicable to users logged on to the website for now, because of the difficulties in tracking not logged-in users from session to session. However, solving this logging of non-users through cookies could widen the utilization of the recommendation system to more users visiting jyllands-posten.dk both users and non-users.

6.1 Discussion

Evaluation of the model performance are based on predictions of recommendations based on what the users already consumed, evaluated on the examples hidden in the test set, emulating the future consuming behavior of users. However, the purpose of recommender systems can be to guide customers towards something they would not have thought of consuming if not

recommended. Thus recommending something out of the scope of the usual user consumption behavior.

Although there are many upsides to personalizing the user experience for people online, such as relevance, better-targeted ads, and valuable recommendations, as addressed throughout this paper, there are also ethical issues with personalizing the web that requires attention. The activist Eli Pariser calls this problem 'The Filter Bubble', the term describes the situation we live in where everyone lives in their own online universe. A universe where you do not actively choose the content that you see, and you do not see what is edited out before the information is presented to you. Examples of this are how different people can make the exact same search query on Google and get different results, and the Facebook newsfeed where filtering enables, that the information presented is from the people whom you interact the most with, leaving you no clue about what other friends or less active Facebook users might be posting. When asked about the Facebook news feed Mark Zuckerberg, the founder of Facebook, said this: "A squirrel dying in front of your house may be more relevant to your interests right now than people dying in Africa." (Pariser, 2011).

There is a difference between what users wants to see vs. what they need to see. Eli Pariser calls this a balance in the information diet. People need to be fed information vegetables along with information junk food. A way to overcome this issue could be with a focus on training these new algorithms serving as information gatekeepers to give individuals more control and transparency, and encode a sense of societal responsibility. This could be presenting information to users that are from another point of view, uncomfortable, relevant, important, and challenging along with the information that the person usually consumes.

6.2 Future work

For future work, a merging of a collaborative filtering recommender and a content-based approach, creating a hybrid recommender, could overcome the cold-start issues of a collaborative filtering model. Thus being applicable to implement in the recommendation of articles to users or non-users from the first interaction with jyllands-posten.dk.

Another approach could be to do some natural language processing on the titles or text corpuses of the articles, to extract themes or features for use in the building of the recommendation system model.

7 References

Aher, S. B., & Lobo, L. (2013). Combination of machine learning algorithms for recommendation of courses in E-Learning System based on historical data. *Knowledge-Based Systems*, 14.

Brynjolfsson, E., Hu, Y. "., & Smith, M. D. (2006). From Niches to Riches: The Anatomy of the Long Tail. *MIT Sloan Management Review* , 12.

Dias, M. B., Locher, D., Li, M., El-Deredy, W., & Lisboa, P. J. (2008). The Value of Personalised Recommender Systems to E-Business: A Case Study. *Proceedings of the 2008 ACM conference on Recommender systems*, 291-294.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information Tapestry. *Association for Computing Machinery* , 1-10.

Gomez-Uribe, C. A., & Hunt, N. (2016). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems (TMIS)* , 19.

Hardt, D., & Rambow, O. (2017). Predicting User Views in Online News.

Koren, T., Hu, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. *ICDM '08 Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 263-272.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *IEEE Computer Society*, 42-49.

Pabani, V., & Deulgaonkar, A. (2016). News Recommendation System Based on Multiple Classifiers. 1-6.

pandas. (n.d.). *GitHub*. Retrieved from pandas: https://github.com/pandas-dev/pandas

Pariser, E. (2011). *The Filter Bubble - How the new personalized web is changing what we read and how we think.* New York: The Penguin Press.

Phillips, J. M. (2013). *Jaccard Similarity and Shingling.* Retrieved from School of Computing - the University of Utah: https://www.cs.utah.edu/~jeffp/teaching/cs5955/L4-Jaccard+Shingle.pdf

Rendle, S. (2010). Factorization Machines. *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*.

Ruder, S. (2016, Jan 19). *ruder.io.* Retrieved from Stochastic gradient descent: http://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent

Takács, G., & Tikk, D. (2012). Alternating least squares for personalized ranking. *Researchgate*, 10.

Turi. (2016, July 6). *GraphLab Create API*. Retrieved from graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender: https://turi.com/products/create/docs/generated/graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender.html#graphlab.recommender.ranking_factorization_recommender.RankingFactorizationRecommender

Turi. (n.d.). *Turi.com*. Retrieved from Turi Machine Learning Platform User Guide: https://turi.com/learn/userguide/

Ullman, J. D., Jure, L., & Rajaraman, A. (2014). *Mining of Massive Datasets.* Cambridge University Press.

Wang, C., & David, B. M. (2011). Collaborative Topic Modeling for Recommending Scientific Articles. *Colombia*, 9.

Wang, Y.-F., Chuang, Y.-L., Hsu, M.-H., & Keh, H.-C. (2004). A personalized recommender system for the cosmetic business. *Expert Systems with Applications*, 427-434.

8 Appendix

8.1 Full code from final run

In [1]:
```
#Import dependencies
import pandas as pd
import graphlab as gl
```

In [2]:
```
#Load in dataset from csv file
df = pd.read_csv('jp_data.csv', sep=',')
df.columns = ['uid','aid']
df.head()
```

Out[2]:

	uid	aid
0	af0f9208-0ecb-4a77-b451-672641a99705	9409190
1	d5cfd1aa-ffbf-4d8d-ada3-b9383e05ed15	9409171
2	5f9dbd21-356a-4eb4-a150-70f9a221cf02	9409190
3	d919e461-5f4b-4f57-8365-17da17585c74	9412675
4	bcdc0ba0-c1cd-48f6-9b4e-3f779abca6c8	9398666

In [3]:
```
#Remove dublicates from dataset
print('Number of rows: ' + str(len(df)))
print('Duplicated rows: ' + str(df.duplicated().sum()))
print('Drop duplicates')
df.drop_duplicates(inplace=True)
print('Remaining rows: ' + str(len(df)))
```

```
Number of rows: 9151970
Duplicated rows: 1831679
Drop duplicates
Remaining rows: 7320291
```

In [4]:
```
#Defining threshold function
def thresholds(df, uid_min, aid_min):
    n_users = df.uid.unique().shape[0]
    n_items = df.aid.unique().shape[0]
    print('Starting views info')
    print('Number of users: {}'.format(n_users))
    print('Number of articles: {}'.format(n_items))

    done = False
    while not done:
        starting_shape = df.shape[0]
        aid_counts = df.groupby('uid').aid.count()
        df = df[~df.uid.isin(aid_counts[aid_counts < aid_min].index.tolist())]
        uid_counts = df.groupby('aid').uid.count()
        df = df[~df.aid.isin(uid_counts[uid_counts < uid_min].index.tolist())]
        ending_shape = df.shape[0]
        if starting_shape == ending_shape:
            done = True

    assert(df.groupby('uid').aid.count().min() >= aid_min)
    assert(df.groupby('aid').uid.count().min() >= uid_min)

    n_users = df.uid.unique().shape[0]
    n_items = df.aid.unique().shape[0]
    print('Ending views info')
    print('Number of users: {}'.format(n_users))
    print('Number of articles: {}'.format(n_items))
    return df
```

In [5]:
```
#Determining thresholds
df_th = thresholds(df,80,80)
```

```
Starting views info
Number of users: 89179
Number of articles: 89605
Ending views info
Number of users: 16406
Number of articles: 12663
```

In [6]:
```
#Converting dataframe to SFrame
#Splitting SFrame into train and test set
sf = gl.SFrame(df_th)
train, test = gl.recommender.util.random_split_by_user(sf,user_id='uid',item_id='aid',max_num_users=1000,
                                                       item_test_proportion=0.3)
print(train)
print(test)
```

```
This non-commercial license of GraphLab Create for academic use is assigned to miso12aj@student.cbs.dk and will expire on August
19, 2018.
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: C:\Users\MIKSOE~1\AppData\Local\Temp\graphlab_server_15
04357852.log.0
```

```
+------------------------------+---------+
|              uid             |   aid   |
+------------------------------+---------+
| af0f9208-0ecb-4a77-b451-67... | 9409190 |
| d5cfd1aa-ffbf-4d8d-ada3-b9... | 9409171 |
| 5f9dbd21-356a-4eb4-a150-70... | 9409190 |
| d919e461-5f4b-4f57-8365-17... | 9412675 |
| bcdc0ba0-c1cd-48f6-9b4e-3f... | 9398666 |
| f45bbed9-6aca-4c20-82cf-43... | 9391092 |
| 5a84f8d6-e12e-4e69-b472-1e... | 9401495 |
| fdd87860-06bb-4c67-858d-81... | 8612221 |
| 347ff745-c5ea-433d-8f8e-ec... | 9412675 |
| cffcf618-dafd-427a-9048-6d... | 9398752 |
+------------------------------+---------+
[5793489 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
+------------------------------+---------+
|              uid             |   aid   |
+------------------------------+---------+
| 0d0f93dd-4955-4ab0-bc1d-78... | 9397027 |
| e7bb9cb4-aecf-4a45-be33-75... | 9401884 |
| 7b273a36-fb30-4f7f-a7ae-3c... | 9409186 |
| ddd94d83-98f9-4efe-a1bd-53... | 9397954 |
| 6b5c4118-6b19-4293-bb59-1c... | 9404733 |
| e1eb465a-7486-4dc7-a002-33... | 9407391 |
| ea169d2b-5edc-443b-be88-d9... | 9408761 |
| e7bb9cb4-aecf-4a45-be33-75... | 9401803 |
| 9cc407c7-fadd-4312-adde-44... | 9412141 |
| ea169d2b-5edc-443b-be88-d9... | 9409144 |
+------------------------------+---------+
[107666 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

In [7]:
```
#Train recommendation models

#Popularity model used as baseline
popularity_model = gl.popularity_recommender.create(train, user_id='uid', item_id='aid')

#Item similarity recommender
item_sim_model = gl.item_similarity_recommender.create(train, user_id='uid', item_id='aid',
                                                       similarity_type='jaccard')
#Ranking factorization recommender
rank_fac_model = gl.ranking_factorization_recommender.create(train, user_id='uid',
                                                             item_id='aid',solver='ials')
```

In [8]:
```
#Load in side info
side_df = pd.read_csv('category.csv', sep=',')
side_df.columns = ['aid','category']
side_df.head()
```

Out[8]:

	aid	category
0	8798646	indland
1	9120180	indblik
2	9386039	aarhus
3	9096606	sport
4	9354492	article

In [9]:
```
#Converting dataframe to SFrame
side_sf = gl.SFrame(side_df)
print(side_sf)
```

```
+---------+---------------+
|   aid   |    category   |
+---------+---------------+
| 8798646 |    indland    |
| 9120180 |    indblik    |
| 9386039 |     aarhus    |
| 9096606 |     sport     |
| 9354492 |    article    |
| 7745211 |    livsstil   |
| 9337468 |    indblik    |
| 9204233 | international |
| 9086419 |    article    |
| 9213980 |    indland    |
+---------+---------------+
[129428 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

In [10]:
```
#Train recommendation models with sideinfo for items

#Item similarity recommender with sideinfo for items
item_sim_model_sideinfo = gl.item_similarity_recommender.create(train, user_id='uid', item_id='aid',
                                                                item_data=side_sf, similarity_type='jaccard')

#Ranking factorization recommender with sideinfo for items
rank_fac_model_side = gl.ranking_factorization_recommender.create(train, user_id='uid', item_id='aid',
                                                                  item_data=side_sf,solver='ials')
```

In [11]:
```
#Comparing models on test set
gl.recommender.util.compare_models(test,[popularity_model,item_sim_model, rank_fac_model],
                                   ['Popularity model','Item similarity recommender','Ranking factorization recommender'],
                                    metric='precision_recall')
```

```
PROGRESS: Evaluate model Popularity model

recommendations finished on 1000/1000 queries. users per second: 3636.42


Precision and recall summary statistics by cutoff
+--------+----------------+-------------------+
| cutoff | mean_precision |    mean_recall    |
+--------+----------------+-------------------+
|   1    |     0.231      | 0.00277468976843  |
|   2    |     0.1985     | 0.00445465457473  |
|   3    | 0.183666666667 | 0.00598691817635  |
|   4    |    0.17825     | 0.00783573031927  |
|   5    |      0.17      | 0.00946724899886  |
|   6    | 0.163833333333 | 0.0109520424357   |
|   7    | 0.158428571429 | 0.0124549832648   |
|   8    |     0.153      |  0.013734507417   |
|   9    |     0.148      | 0.0148023090111   |
|   10   |     0.1437     | 0.0158686377195   |
+--------+----------------+-------------------+
[10 rows x 3 columns]

PROGRESS: Evaluate model Item similarity recommender

recommendations finished on 1000/1000 queries. users per second: 2739.78


Precision and recall summary statistics by cutoff
+--------+----------------+-------------------+
| cutoff | mean_precision |    mean_recall    |
+--------+----------------+-------------------+
|   1    |     0.273      | 0.00346292276372  |
|   2    |     0.245      | 0.00608847734124  |
|   3    |     0.227      | 0.00833849169424  |
|   4    |    0.21925     | 0.0108898562453   |
|   5    |     0.2142     | 0.0132930447552   |
|   6    | 0.207166666667 | 0.0152907984703   |
|   7    | 0.202285714286 | 0.0174117502335   |
|   8    |    0.19575     | 0.0191640268721   |
|   9    |     0.191      | 0.0209828125823   |
|   10   |     0.1882     | 0.0229118340791   |
+--------+----------------+-------------------+
[10 rows x 3 columns]


  PROGRESS: Evaluate model Ranking factorization recommender

  recommendations finished on 1000/1000 queries. users per second: 2666.67


  Precision and recall summary statistics by cutoff
  +--------+----------------+-------------------+
  | cutoff | mean_precision |    mean_recall    |
  +--------+----------------+-------------------+
  |   1    |     0.368      | 0.00495668044769  |
  |   2    |     0.345      | 0.00893875058652  |
  |   3    | 0.334666666667 | 0.0129899060278   |
  |   4    |     0.321      | 0.0163879832335   |
  |   5    |     0.3108     | 0.0196033536602   |
  |   6    | 0.299166666667 | 0.0223510297445   |
  |   7    | 0.292571428571 | 0.0253749675013   |
  |   8    |     0.286      | 0.0282352584414   |
  |   9    |      0.28      | 0.0310136378703   |
  |   10   |     0.2734     | 0.0336533837746   |
  +--------+----------------+-------------------+
  [10 rows x 3 columns]
```

In [12]:
```
#Comparing models with sideinfo on test set
gl.recommender.util.compare_models(test,[item_sim_model_sideinfo, rank_fac_model_side],
                                   ['Item similarity recommender with sideinfo',
                                    'Ranking factorization recommender with sideinfo'],
                                    metric='precision_recall')
```

```
PROGRESS: Evaluate model Item similarity recommender with sideinfo

recommendations finished on 1000/1000 queries. users per second: 1165.48


Precision and recall summary statistics by cutoff
+--------+----------------+------------------+
| cutoff | mean_precision |   mean_recall    |
+--------+----------------+------------------+
|   1    |     0.273      | 0.00346292276372 |
|   2    |     0.2445     | 0.00604781395058 |
|   3    | 0.226666666667 | 0.00834274650617 |
|   4    |    0.21875     | 0.0108496890407  |
|   5    |     0.2132     | 0.0132064623639  |
|   6    |     0.207      | 0.0152595305296  |
|   7    | 0.202428571429 |  0.01741767692   |
|   8    |    0.195625    | 0.0191681624534  |
|   9    | 0.191222222222 | 0.0209916445376  |
|   10   |     0.1879     | 0.0229097570122  |
+--------+----------------+------------------+
[10 rows x 3 columns]

PROGRESS: Evaluate model Ranking factorization recommender with sideinfo

recommendations finished on 1000/1000 queries. users per second: 4566.17


Precision and recall summary statistics by cutoff
+--------+----------------+------------------+
| cutoff | mean_precision |   mean_recall    |
+--------+----------------+------------------+
|   1    |      0.38      | 0.0050505378568  |
|   2    |     0.347      | 0.00915578294338 |
|   3    | 0.331666666667 | 0.0128151775032  |
|   4    |     0.3205     | 0.0162816087954  |
|   5    |     0.3114     | 0.0195203846909  |
|   6    | 0.300333333333 | 0.0227294047814  |
|   7    | 0.293285714286 | 0.0258649943976  |
|   8    |     0.2875     | 0.0287559685376  |
|   9    | 0.281333333333 | 0.0314755831239  |
|   10   |     0.2744     | 0.0340233368816  |
+--------+----------------+------------------+
[10 rows x 3 columns]
```

In [24]:
```
title_sf = gl.SFrame.read_csv('aid_titles.csv',
                              column_type_hints={'content_id':int,'title':str})
title_sf.rename({'content_id': 'aid'})
print(title_sf)
```

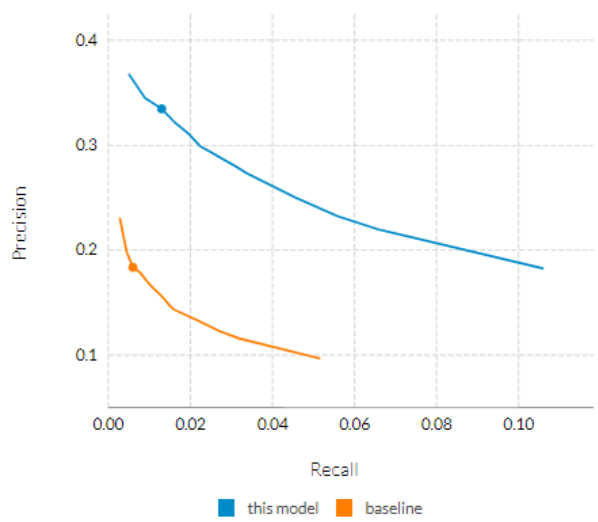
```
Finished parsing file C:\Users\miksoerensen\Documents\Python Scripts\aid_titles.csv

Parsing completed. Parsed 100 lines in 0.172494 secs.

Finished parsing file C:\Users\miksoerensen\Documents\Python Scripts\aid_titles.csv

Parsing completed. Parsed 115540 lines in 0.126351 secs.

+---------+------------------------------+
|   aid   |             title            |
+---------+------------------------------+
| 4676534 | Naturkatastrofer tog 30.00... |
| 4680165 | Tysk spøgelses-fly hentet ... |
| 6216237 | Sådan fungerer EU's handel... |
| 4656811 | Vandt mio. i usædvanlig he... |
| 7969376 | Syg modelbranche? Model me... |
| 8766281 | Fusk med forskningsmidler ... |
| 4571228 |         Lars Skov (DF)       |
| 4955783 | Google fejrer eventyrlige ... |
| 3277287 |  Nyt navn til Avedøre-lejren  |
| 8123674 | IS og al-Qaeda må se langt... |
+---------+------------------------------+
[115540 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

In [26]:
```
#view examples of recommendations based on test set
view = rank_fac_model.views.overview(validation_set=test,
                                     baseline=popularity_model,
                                     item_data=title_sf,
                                     item_name_column='title')
view.show()
```

Number of recommended items 3

Core Metrics

Precision & Recall Curve



	Mean Precision	Mean Recall
this model	0.335	0.013
baseline	0.184	0.006