Recommender Algorithms:

Search for the best performance recommender

Algorithm on Top-N Recommendation Tasks

Victor Naranjo

CSU Stanislaus

CS4960

Dr. Melanie Martin

Section 0.0 Introduction

In the past decade, e-commerce has become an integral part of our life from shopping for Christmas gifts to paying bills online. As the e-commerce environment has grown, it is now common to see products that are recommended specifically for the user's needs and interests. How can an online market recommend items specifically for you? To the un-informed, this may seem like magic. But to the informed computer scientist, a good guess can be made that there is a complex algorithm in the background doing all the work. Recommender systems are the algorithms being used to recommend items to users in any website.

These recommender systems filter through items on a specific data set; the datasets that we will be using are the Netflix dataset and the MovieLens dataset. The benefit of any online market having recommender systems is that it may increase sales by recommending items that are relative to the user. Therefore, the performance of recommender systems may play a very crucial role in sales. The goal is to find an algorithm that recommends the best item to the user, which is referred to as top-N items, through performance error metrics. We will evaluate the performance of filtering algorithms in the ultimate pursuit of the Top-N Recommender task. There are some basic terms and concepts to look at before we can understand how recommender systems work. Recommender systems fall under the discipline of science called artificial intelligence.

Section 0.1 Artificial Intelligence

The term artificial intelligence was coined in 1955 by John McCarthy and is defined as the science and engineering of making intelligent machines. But artificial intelligence goes much

further than just teaching machines to learn, it helps us to try and understand how we think, perceive, understand and predict. And as a result of these questions we try to understand how to build intelligent entities. Recommender systems are a type of machine learning tool which is a branch of artificial intelligence.

Section 0.2 Machine Learning

Machine learning is an algorithm that is implemented on a set of data to retrieve useful information. The data before it is analyzed is called, raw data, whereas, data that has been analyzed is called indexed data. This discipline of science is called machine learning because the algorithms that is being implemented is "learning" to make generalization from the source data. The objective of machine learning to generalize and learn from its experiences. The goal is for the machine to create a generalization of the input so that the learning machine can perform accurately and efficiently on new and unseen data. First, we "train" the learning machine, which we feed input into the learner so that it can make rules and generalizations about input. The action of implementing an algorithm on a source data is called a task. Machine learning tasks are usually broken down into 3 categories, supervised learning, unsupervised learning, and reinforcements learning. **Supervised learning** is when a computer is given a set of example inputs and expected outputs presented by the "teacher". What is expected is the computer to learn a general rule that maps input and outputs. **Unsupervised learning** is when a computer is not given any set of example input. The computer is left on its own to learn to structure the input; this is usually used to find hidden patterns in data. There is no error or reward system in place, this is what really distinguishes unsupervised learning from reinforcement and supervised learning. **Reinforcement learning** is when a computer program is presented with a dynamic situation in which it is given certain tasks that it must accomplish. The computer program does not have a teacher to tell it whether it is close to its goal or

not, a good example of this is a computer program learning to drive a vehicle. These three learning tasks are usually concerned with input. There are other categories that are concerned with the output. Recommender systems are very concerned with the output.

Tasks that are concerned with output are categorized into classification, regression, and clustering. **Classification**, inputs are provided and divided into the appropriate amount of classes, and the learner produces how to classify new inputs into the classes that were provided. An example of classification is spam filtering, where the input would be individual emails and the classes that the inputs can be classified as, are spam or not spam. The learner would be able to filter the mails into either one of the classes. **Regression** is considered supervised learning where the outputs are continuous rather than discrete. This task focuses on the relationship between a dependent variable and one or more independent variables. **Clustering**, is a type of unsupervised learning, much like classification inputs are grouped together but are not known beforehand. Cookies are used to store login information, browser history and other personal information from the user on both the end system and the back-end servers.

Section 0.4 Cookies

Web browsers are constantly storing data about your activities on the internet by the use of cookies. As users browse the internet, there is constantly information being sent and received by the end user and the http servers. HTTP servers are designed to be stateless to permit engineers to develop high performance web servers, as a result these web servers can handle thousands of simultaneous TCP connections at once. Many e-commerce sites such as eBay and Amazon would to like to identify users to either restrict access or create specific content for that user. The way that this is accomplished is through the use of cookies. Cookies are defined in the Request for Comments (RFC) 6265, this allows web sites to keep track of user's activity and information. Servers store

massive amounts of personal data from users like you through cookies. This data initially is useless, but with data mining techniques we can extract useful data and use that to collect useful information about users and current trends. At this point, the web server do not know any personal information about the user. Web sites can easily by pass this obstacle by allowing the user to make an account with that web site.

When users make an account with a website, there is now a cookie referencing that user's credit card information, name, addresses and other personal information. Every page and item you look at will now be referenced by your cookie id. Recommender systems can use this information and begin to analyze your personality by items you look at. Cookies are very convenient for both users and web servers but are very controversial as they can possibly lead to an invasion of privacy. It is common practice for large companies to sell the information gathered to other companies who can than utilize the massive amounts of information and put it to practical use. How exactly is all this information sorted and used to recommend items to us?

Section 1.0 Recommender Systems

Several recommender systems show you the best bet recommendations rather than the predicted values. This approach is referred to as the **Top-N recommendation task**. The goal of such a task is to find an N amount of items which are found to be appealing to the user. The Top-N performance can be directly measured by accuracy metrics such as precision and recall. Many recommender tasks measure accuracy by trying to minimize RMSE (root-mean-square-error). RMSE is a measurement of the differences between values predicted by a task and the values that

are actually observed. There has been several evaluations of state-of-the art recommender systems that attempt to minimize RMSE that do not perform as expected in regard of Top-N recommendation task. This finding concludes that minimizing RMSE does not translate into accuracy improvements. In some instances, non-personalized algorithms can outperform and match some sophisticated algorithms. Another thing we must take into consideration is that a few top popular items (short-head) can skew the Top-N performance. Therefore the test set should be chosen carefully as to not bias accuracy metrics. We will look at two variants of recommender tasks so see if they outperform RMSE-oriented recommender algorithms in pursuit of the Top-N recommendation task

---

Section 1.1 how our datasets were obtained

The two datasets we will be using are from Netflix and MovieLens. Each dataset will be divided into two subsets. A training set M and a test set T. Our test set T will contain only 5-star ratings, which will give us the assumption that the test set will be relevant to our user. Netflix released a training data set when they announced their Netflix challenge. Their training set consists of 100 million ratings which we will refer to as our training dataset. Netflix also provided a validation set, we will refer to as our probe set, which contains 1.4 million ratings. Our training set M will consist of Netflix original training set and our test set T will contain all the 5-star ratings from the probe set, which we will denote as |T|=384,573. The following table shows the statistical properties of MovieLens and the Netflix Datasets.

Dataset	Users	Items	Ratings	Density
MovieLens Netflix	6,040 480,189	3,883 17,770	1,000,000 100,000,000	4.26% 1.18%

Our MovieLens dataset was created similar to as Netflix dataset. 1.4% of the ratings from the MovieLens dataset was randomly sub-sampled to create a probe set. Our training set M contains the rest of the ratings and our test set T contains all the 5-star ratings from our probe set. We will train the model over the ratings in M so that we can measure recall and precision.

Section 1.2 Training Our Models

There are 4 steps to take so that we can train the model over our ratings in training set M. For each item i rated 5-star ratings by user u in test set T.

1. 1000 additional items are randomly selected which are unrated by user u. we can make a safe assumption that the additional 1000 items are not relevant to user u.
2. Ratings for the 1000 additional items that were chosen at random and the test item i are predicted.
3. After the ratings are predicted, a ranked list is formed by ordering all the 1001 items according to the predicted ratings. The case where the test item i precedes all the 1000 additional items is the best result. When p=1, let p denote the rank of the test item i.
4. The way that we form a Top-N recommendation list is by picking the N top ranked items from the list. If $p \leq N$ we have a hit. When we have a hit the test item i is recommended to the user. When we don't have a hit we have a miss. When N = 1001 we always have a hit.

Since every test case will have 1001 items, we are guaranteed a hit.

Computation of recall can only assume a value of 0 or 1. Where 0 is a miss and 1 is a hit. Precision can have a value of either 0 or 1/N. Averages over all the test cases are defined as:

$$\text{Recall(N)} = \frac{\#hits}{|T|}$$

$$\text{Precision(N)} = \frac{\#hits}{N*|T|} = \frac{recall(N)}{N}$$

|T| represents the number of test ratings.

Section 1.3 Popular items vs. long tail items

It is well known, according to the long-tail distribution, that a majority of ratings are condensed in a small fraction of the most popular items.

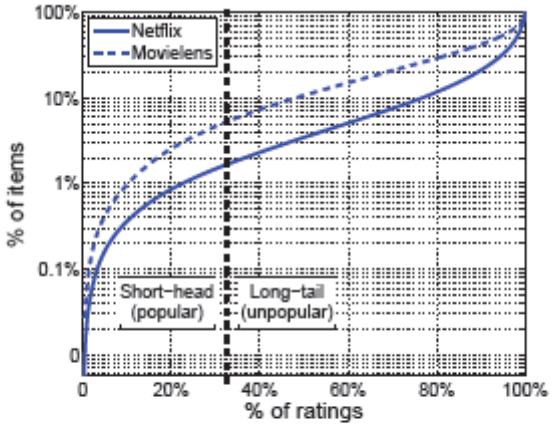


Figure 1: Rating distribution for Netflix (solid line) and Movielens (dashed line). Items are ordered according to popularity (most popular at the bottom).

In figure 1, empirical rating distributions of the Netflix and MovieLens datasets are plotted. The vertical axis contains items that are ordered according to their popularity with the most popular at the bottom. The short-head refers to the 1.7% of the most popular items in the Netflix dataset, the remaining set (about 98%) is referred to as the long tail.

Why is this figure relevant to us? By knowing what items are popular and unpopular it can help us determine what type of items to recommend. Obviously, recommending popular items is trivial. It does not take make to recommend the #1 selling item; it does not even require a complex algorithm. But being able to recommend items on the long tail adds novelty and thus promotes sales diversity. The algorithms we will look at we will be evaluating the accuracy of recommender algorithms suggesting non-trivial items. For this, we will further divide our test set into two more subset, T-head and T-long. T-head is in the short-head while items in T-long are in the long-tail of the distribution.

Section 2.0 Collaborative Algorithms

Most recommender systems utilize collaborative filtering algorithms. Collaborative filtering relies only on past user activity, where activity refers to the ratings given to items. Activity can also refer to other actions such as purchases, rentals and clicks but for this discussion we will refer to activity as ratings. There are two approaches to collaborative filtering we will look at: 1) the **neighborhood approach** and 2) the **latent factor approach**.

The neighborhood model is the most common approach to collaborative filtering and it is based on the similarity among either users or items. For example two users might be considered similar because they rate the same item with a similar rating. The latent factor approach models users and items as vectors in the same 'latent factor' space. In this latent factor space items and users are directly comparable by the proximity between these objects.

Section 2.1 Non-Personalized Models

Non-Personalized models are exactly what they sound like, they are not personalized for the user. Non-Personalized recommenders have a predefined, fixed list of items regardless of the preferences of the user. These models are less complex than personalized recommenders but are often utilized as baselines to more complicated filtering algorithms. An example of non-personalized recommender is a simple estimation rule called the **Movie Average algorithm**, which simply recommends Top-N items with the highest average rating. By this model, the rating of user u on item i is predicted by the rating given on item i by the community. As said before this type of model does not provide novelty to the recommendations and simply recommends items in the short head. Another type of non-personalized model is the **Top Popular** (TopPop) algorithm which recommends Top-N items with the highest popularity. Highest popularity is the same as the largest number of ratings.

Section 2.2 Neighborhood models

As mentioned above, neighborhood models predict their ratings among either users or items. So we can either have relationships of user-user or item-item. User-user relationship predict the rating by looking at users that are similar to user u. Item-item relationship predicts the rating for an item i based on ratings that user u has given on items similar to item i. Item-item relationship are most commonly preferred because it usually performs better in terms of RMSE as well as being more scalable. The reason that this is so is because is most cases there are more users than items. An advantage of the item-item relationship is that items can have a high relevance to the user by comparing it to items previously rated by the user. Also if we based our algorithm on a user-user relationship, we would have to account for the fact that there will be several new users every day making it more complicated. The definition of similarity is as follows: for item i and j, the similarity between them is the tendency of users to both items similarly.

In some cases where there is a sparse data set and you would still like to recommend items, there are ways of managing your recommender system. Let n_{ij} represent the number of common raters and s_{ij} represent the similarity between items i and j. d_{ij} will represent the shrunk similarity and

$$d_{ij} = \frac{n_{ij}}{n_{ij}+\alpha} s_{ij}$$

Where α represent as shrinking factor, and a typical value is 100.

We can further enhance this approach with the **kNN** approach (k-nearest neighborhood). This approach discards the items that are not relevant to the target item which will result in is a decrease of noise for improved quality recommendation. Let a predicted rating be represented by r_{ui}, k will represent the number of items rated by user u that are similar to item i. $D^k(u;i)$ will represent the set of similar items. To further improve quality recommendation it is highly advised to remove biases between the fundamental relations between items. Biases can include things such

item-effects which represents that fact that some items are more likely to receive a higher rating whether it is more of better quality or users rating highly. Another bias is the user-effect with states that it is natural for some user to rate higher than other user. That is an important bias to take into consideration because you cannot assume that the user rating have quality assurance expertise. b_{ui} will represent the bias associated with the ratings of user u on item i.

A residual rating is predicted by an item-item kNN method with is denoted by $r_{ui} - b_{ui}$ which represent the weighted average of the residual ratings of similar items:

$$\check{r}_{ui} = b_{ui} + \frac{\sum j \in D^k(u;i)^{d_{ij}(r_{uj}-b_{uj})}}{\sum j \in D^k(u;i)^{d_{ij}}}$$

This model is known as the **Correlation Neighborhood model** (CorNgbr)

As you can see in the CorNgbr model, the denominator forces the value to fall in the correct range of a 5 star rating which is typical of most star-rating systems. Keep in mind that we are not predicting star rating but rather creating a list for a Top-N recommendation task, therefore we can simplify the CorNgbr model by eliminating the denominator. An advantage of this is we will have a higher ranking system now which will result in improved quality recommendations. We will now rank items by:

$$\check{r}_{ui} = b_{ui} + \sum j \in D^k(u;i)^{d_{ij}(r_{uj}-b_{uj})}$$

This model is known as the **Non-Normalized Cosine Neighborhood** or NNCosNgbr for short and is a variant of the CorNgbr model. $\check{r}_{ui}$ does not represent a proper rating but only a means to form a ranked list. A proper definition of $\check{r}_{ui}$ is a metric for the association between user u and item i.

The best results in terms of accuracy were obtained when computing s_{ij} as the cosine similarity. The cosine similarity is a measurement of similarity between two vectors that measures the angle between the inner product spaces. Inner product space is simply a vector space with an additional structure called the inner product. In this case item i and j are both vectors and the inner

product associates each pair of vectors with a scalar quantity which results in the inner product of the vectors.

Section 3.0 Latent Factor Models

As previously stated the 'Latent factor approach' models users and items as vectors in the same 'latent factor' space. An advantage of this is that user and items are directly comparable according to their proximity in said space. A state of the art recommender system that we will be looking at is the **SVD model** (Singular Value Decomposition) which is based on factoring the user-item ratings matrix. The basic idea behind the SVD model is to factorize the user-item rating matrix to a product of two lower rank matrices. The two lower rank matrices consist of the 'user-factor' matrix and the 'item-factor' matrix. These matrices are traits of item 'i' and user 'u', where user u is represented with an f-dimensional user-factor ($\mathbf{p}_u \in R^f$) and item 'i' is represented with an item-factor vector. The prediction rating is computed by the following formula

$$\check{r}_{ui} = b_{ui} + \mathbf{p}_u \mathbf{q}_i^T$$

The SVD model that was used in these experiments are called **Asymmetric-SVD** (AsySVD). It can reach RMSE value of 0.9000 on the Netflix Dataset. We will be comparing AsySVD with **SVD++** which is the highest optimized algorithm in terms of RMSE, but the disadvantage of SVD++ is that users are not represented by item features.

Section 3.1 PureSVD

Since our goal is to find the Top-N Recommendation Task, we are most interested in a ranked list not actual rating predictions. This results in another benefit for us because it grants us more flexibility when there are missing values in our rating matrix.

Our user rating matrix **R** is computed by the factorization of

$$\check{R} = U * E * Q^T$$

Where,

U is a n * f orthonormal matrix

Q is a m * f orthonormal matrix

E is a f * f diagonal matrix which contains the first f singular values.

To make this a little easier to read let us define P = U * E, the u-th row of P is the user factor vector p_u and the i-th row of Q is the item factor vector q_i. Since P = U * E, P also equals R * Q. We can now rewrite $\check{r}_{ui}$ as follows.

$$\check{r}_{ui} = r_u * Q * \mathbf{q}_i^T$$

As mentioned earlier, $\check{r}_{ui}$ does not represent a rating value but is an association measure between user u and item i. This notation represents PureSVD.

Section 4.0 Results

Thus far we have evaluated 6 Recommender algorithms.

1. MovieAvg
2. TopPop
3. CorNgbr

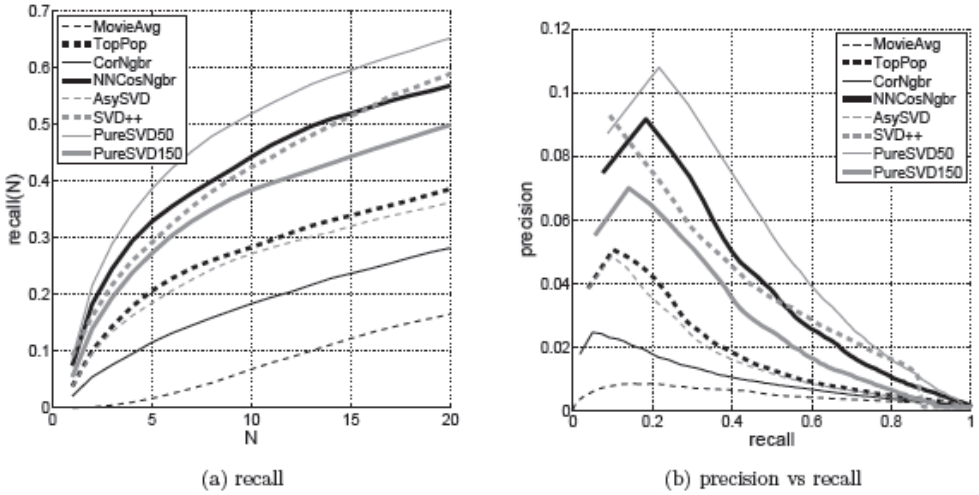
4. NNCosNgbr
5. AsySVD
6. 200-D SVD++

Algorithms 2, 4 and PureSVD cannot be measured in terms of RMSE, but all the other algorithms can. Here are the RMSE scores that were obtained:

Algorithms	RMSE Score
MovieAvg	1.053
CorNgbr	0.9406
AsySVD	0.9000
SVD++	0.8911

Section 4.1 MovieLens Dataset Results

The following figure shows the performance of the algorithms on the MovieLens dataset with the short-head data include.



(a) recall (b) precision vs recall

Figure 4.1.1 Experiment with short-head included

Recall of AsySVD in terms of top-N Performance:

When N = 10, where N represents the number of items recommended, the recall is about 0.28 which represents that the AsySVD model has a 28% probability of recommending the user a relevant item.

Recall of TopPop in terms of Top-N Performance:

When N = 10, the average recall is 0.29, which is very close to a state of the art algorithms AsySVD. Remember that TopPop is a non-personalized algorithms.

The most accurate algorithms were NNCosNgbr and PureSVD

When N = 10, a recall was computed of 0.44 and 0.52, respectively. As the figure above shows, PureSVD has outperformed all other algorithms, and note that the algorithms CorNgbr is one of the most widely used recommender algorithms today. The strange and unexpected results of the TopPop algorithm has produced reason for a second experiment to take place. The figure below shows the second experiment with the short-head data excluded.

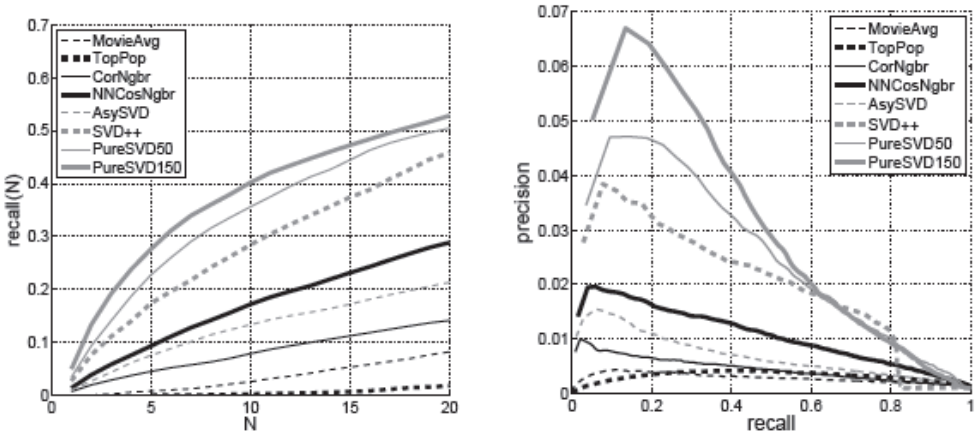
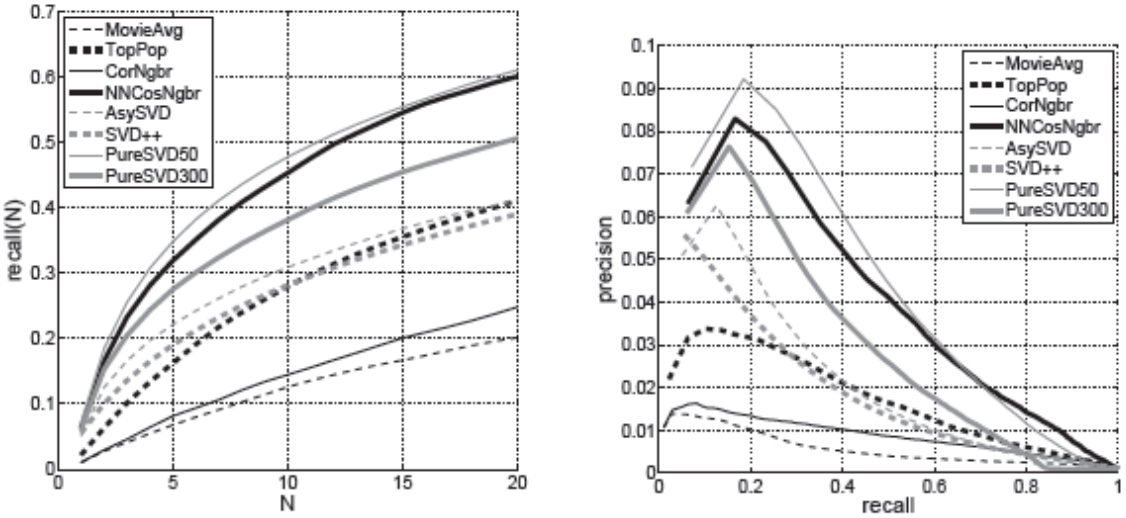


Figure 4.1.2 Experiment with short-head excluded

In this experiment, all popular items have been excluded. The results of this experiment are as expected, since we exclude the short head the recall and precision of the TopPop algorithm has greatly reduced. Even with the short-head excluded, PureSVD has still outperformed all other algorithms when N=10 recall is about 40% and SVD++ is the best algorithms in terms of RMSE.

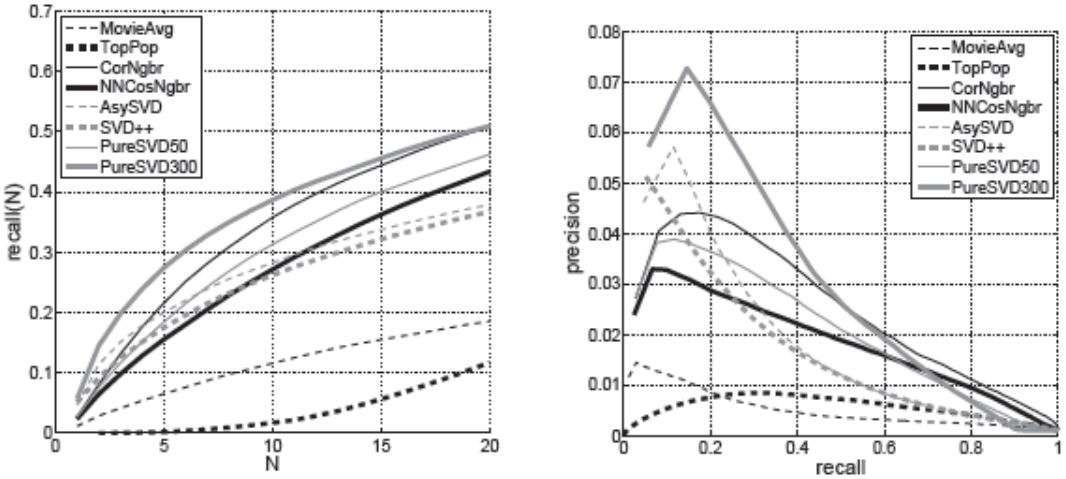
Section 4.2 Netflix Results

The following figure shows the performance of the algorithms on the Netflix dataset with the short-head data include.



(a) recall (b) precision vs recall

Figure 4.2.1 Experiments with short-head data included.



(a) recall (b) precision vs recall

Figure 4.2.2 Experiments with short-head data excluded.

As with the MovieLens results, the non-personalized algorithm TopPop showed very promising results with the Netflix dataset. It outperformed the widely popular CorNgbr but as we can predict from the MovieLens results, the algorithms AsySVD and SDV++ outperformed TopPop. PureSVD, as in the MovieLens Results, was the best non-RMSE oriented album in terms of recall and precision. The CorNgbr algorithms performed better than expected when the short head was excluded but it still under-performed many of the other algorithms. When excluding the short head, many of the algorithms performance in terms of precision and recall decreased whereas CorNgbr appeared to be more accurate. It might be for this reason why the algorithm CorNgbr is so widely used.

Section 4.3 Discussion of PureSVD

Results from both MovieLens and Netflix dataset have made it clear that PureSVD has outperformed all other algorithms and it is a non-RMSE oriented algorithm. This was a shocking result and is very good news for practitioners since it offers so many benefits. These benefits include:

1. PureSVD being very easy to code and implement and there is a very small learning curve.
2. Relies on off-the-shelf optimized SVD packages.
3. Has the ability to represent the user as a combination of item features.

PureSVD can also be configured to deal with long tail item accurately, by raising the dimensionality of the PureSVD model we can increase the accuracy. Our first latent factor captures properties of the short head and the additional latent factors capture the properties of the long tail. Therefore the practitioners could anticipate how to deal with unpopular items. One reason why PureSVD has outperformed the other algorithms is because while testing for RMSE, there are

several things that cannot be tested. As stated earlier when testing for RMSE we can only use the ratings that the user has rated on items. But when we don't look towards RMSE we can start to evaluate and predict ratings for items, even if the user has never rated an item.

Section 5.0 Conclusion

When considering Recommender Algorithms, evaluations of these recommenders have been divided between accuracy metrics and error metrics. RMSE is by a longshot the most popular because the mathematical convenience and fitness with formal optimization methods. But as seen through these experiments, they are not as good as other algorithms in Top-N Recommendations. RMSE is better served as a proxy for Top-N accuracy. Despite the downfall of RMSE oriented algorithms, several non-RMSE oriented recommenders that were shown have proven to be competitors in the recommender scene. It is very easy to create a biased experiment with non-personalized algorithms but the experiments shown have made the results unbiased since two experiments were done, one with the top-head data included and another with the top-head data excluded. This has allowed us to find correct values of non-personalized algorithms, as well as displaying a good behavior for the widely used correlation-based kNN approach which is known for exhibiting poor results. The results of these experiments were threefold. We found that there is no trivial relationship between error metrics and accuracy metrics. Proposal of a careful construction of the test set to avoid biasing accuracy metrics and new variants of algorithms that improve Top-N performance were introduced.

Works Cited

Amin Javari and Mahdi Jalili. 2014. Accurate and novel recommendations: An algorithm based on popularity forecasting. ACM Trans. Intell. Syst. Technol. 5, 4, Article 56 (December 2014), 20 pages. DOI: http://dx.doi.org/10.1145/2668107

Gedikli, F. and Jannach,D. 2013. Improving recommendation accuracy based on item-specific tag preferences. ACM Trans. Intell. Syst. Technol. 4, 1, Article 11 (January 2013), 19 pages. DOI = 10.1145/2414425.2414436 http://doi.acm.org/10.1145/2414425.2414436

Panagiotis Adamopoulos and Alexander Tuzhilin. 2014. On unexpectedness in recommender systems: Or how to better expect the unexpected. ACM Trans. Intell. Syst. Technol. 5, 4, Article 54 (December 2014), 32 pages.
DOI: http://dx.doi.org/10.1145/2559952

Adomavicius, G. and Zhang, J. 2012. Stability of recommendation algorithms. ACM Trans. Inf. Syst. 30, 4, Article 23 (November 2012), 31 pages.
DOI = 10.1145/2382438.2382442 http://doi.acm.org/10.1145/2382438.2382442

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin. "Performance of Recommender Algorithms on Top-n Recommendation Tasks." *Proceedings of the Fourth ACM Conference on Recommender Systems - RecSys '10* (2010): n. pag. Web.

Vargas, Saul, and Pablo Castells. "Improving Sales Diversity by Recommending Users to Items." *Improving Sales Diversity by Recommending Users to Items*. N.p., n.d. Web. 09 Apr. 2015.

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995. Print.

Kurose, James F., and Keith W. Ross. *Computer Networking: A Top-down Approach*. Boston: Pearson/Addison Wesley, 2008. Print.