Comparison of different techniques for predicting sparse data in Recommendation System

By

Kaitav Mehta

A Report Submitted to Dr. Z. Kobti in course Directed Special Studies for the Degree of Master of Science at the University of Windsor

ABSTRACT

In Recommendation Systems there is an open problem called as a Long-Tail problem, this report explains the long-tail problem in detail and briefly explains existing famous solution for it and concludes with experimental results of the comparison of different techniques and finally also explores Deep learning for the long-tail problem.

Table of Contents

LIST OF FIGURES

The Long Tail

“We are leaving the age of information and entering the age of Recommendation”

-Chris Anderson “The Long Tail”

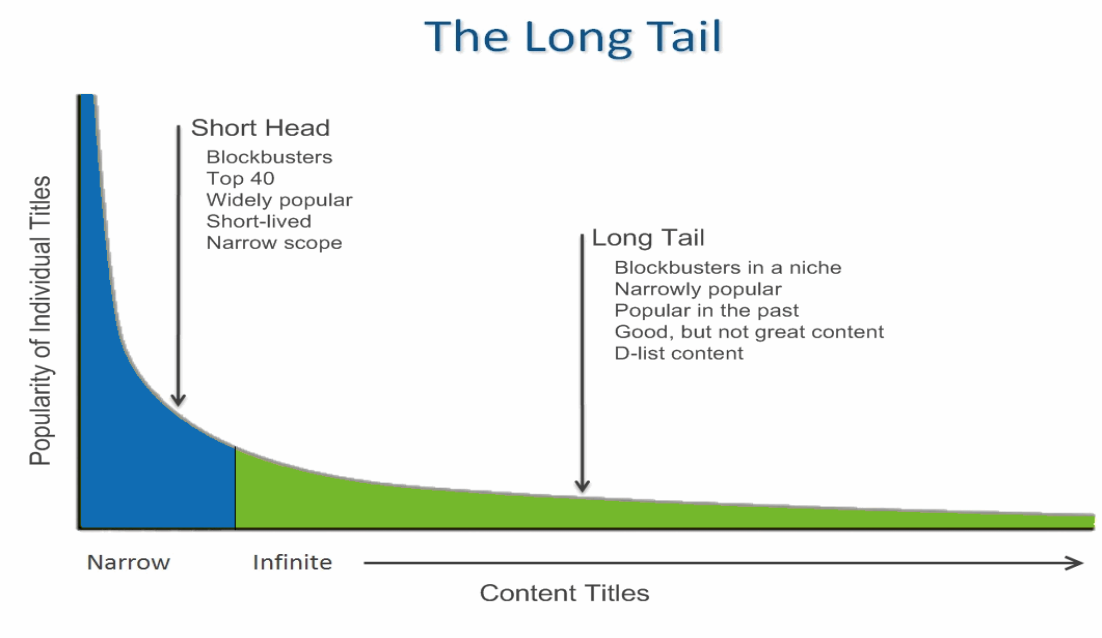


Figure 1 The long tail issue overview 9 [1]

Definition of Recommender System:

Recommender systems are systems that help users discover items they may like.

A recommender system is an Information Retrieval technology that improves access and proactively recommends relevant items to users by considering the users’ explicitly mentioned preferences and objective behaviors. A recommender system is one of the major techniques that handle information overload problem of Information Retrieval by suggesting users with appropriate and relevant items. Today, several recommender systems have been developed for different domains however, these are not precise enough to fulfil the information needs of users. Therefore, it is necessary to build high quality recommender systems. In designing such recommenders, designers face several issues and challenges that need proper attention. This document investigates and reports the current trends, issues, challenges, and research opportunities in developing high-quality recommender systems. If properly followed, these issues and challenges will introduce new research avenues and the goal towards fine-tuned and high-quality recommender systems can be achieved. [2]

Motivation: Why should we care about recommender systems?

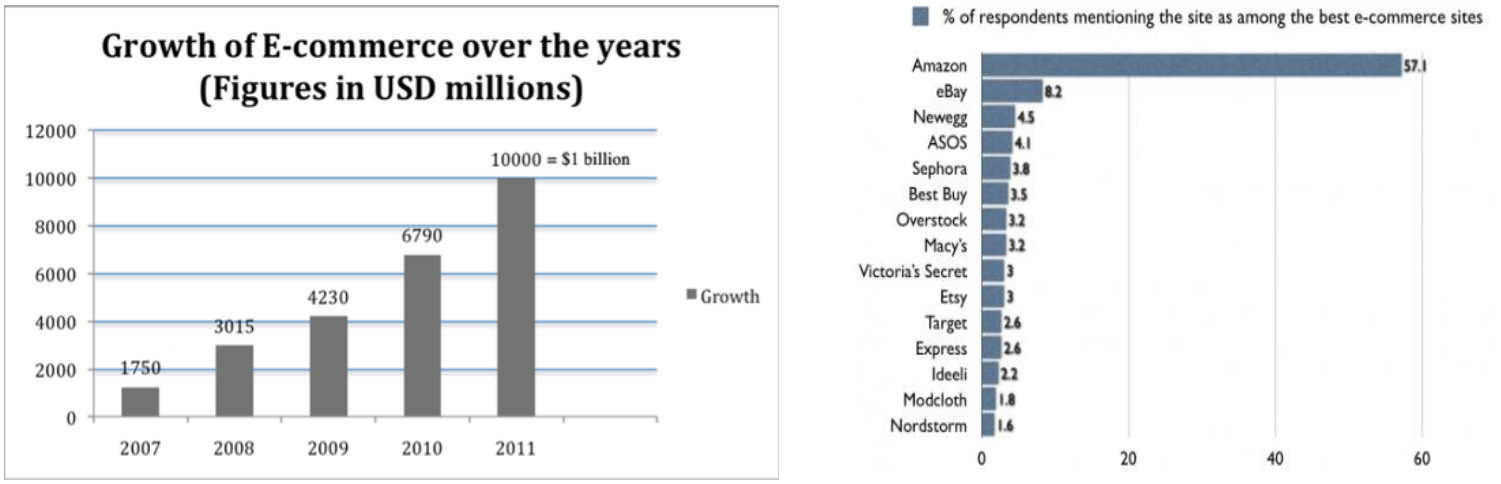


Figure 2 Growth of e-commerce bar graph comparison [1]

The key reason why many people seem to care about recommender systems is *money*. For companies such as Amazon, Netflix, and Spotify, recommender systems drive significant engagement and revenue. But this is the more cynical view of things. The reason these companies (and others) see increased revenue is because they deliver actual *value* to their customers – recommender systems provide a scalable way of personalizing content for users in scenarios with many items.

Another reason why data scientists specifically should care about recommender systems is that it is a true data science problem as the intersection between software engineering, machine learning, and statistics. As we will see, building successful recommender systems requires all of these skills (and more).

Also, Rich get richer concept, popular products get recommended more, while some products get lost in the Long Tail. Further Helping some of the amazing, yet not so popular products to reach the head and recommending different and diverse options for users to choose from.

Recommendation paradigms [3]

Depending on who you ask, there are between two and twenty different recommendation paradigms. The usual classification is by the type of data that is used to generate recommendations. The distinction between approaches is more academic than practical, as it is often a good idea to use hybrids/ensembles to address each method's limitations. Nonetheless, it is worthwhile discussing the different paradigms. The way I see it, if you ignore trivial approaches that often work surprisingly well (e.g., popular items, and "watch it again"), there are four main paradigms: collaborative filtering, content-based, social/demographic, and contextual recommendation.

Collaborative filtering is perhaps the most famous approach to recommendation, to the point that it is sometimes seen as synonymous with the field. The main idea is that you're given a matrix of preferences by users for items, and these are used to predict missing preferences and recommend items with high predictions. One of the key advantages of this approach is that there has been a huge amount of research into collaborative filtering, making it pretty well-understood, with existing libraries that make implementation fairly straightforward. Another important advantage is that collaborative filtering is independent of item properties. All you need to get started is user and item IDs, and some notion of preference by users for items (ratings, views, etc.).

The major limitation of collaborative filtering is its reliance on preferences. In a cold-start scenario, where there are no preferences at all, it can't generate any recommendations. However, cold starts can also occur when there are millions of available preferences, because pure collaborative recommendation doesn't work for items or users with no ratings, and often performs pretty poorly when there are only a few ratings. Further, the underlying collaborative model may yield disappointing results when the preference matrix is sparse. In fact, this has been my experience in nearly every situation where I deployed collaborative filtering. It always requires tweaking, and never simply works out of the box.

Content-based algorithms are given user preferences for items, and recommend similar items based on a domain-specific notion of item content. The main advantage of content-based recommendation over collaborative filtering is that it doesn't require as much user feedback to get going. Even one known user preference can yield many good recommendations (which can lead to the collection of preferences to enable collaborative recommendation). In many scenarios, content-based recommendation is the most natural approach. For example, when recommending news articles or blog posts, it's natural to compare the textual content of the items. This approach also extends naturally to cases where item metadata is available (e.g., movie stars, book authors, and music genres).

One problem with deploying content-based recommendations arises when item similarity is not so easily defined. However, even when it is natural to measure similarity, content-based recommendations may end up being too homogeneous to be useful. Such recommendations may also be too static over time, thereby failing to adjust to changes in individual user tastes and other shifts in the underlying data.

Social and demographic recommenders suggest items that are liked by friends, friends of friends, and demographically-similar people. Such recommenders don't need any preferences by the user to whom recommendations are made, making them very powerful. In my experience, even trivially-implemented approaches can be depressingly accurate. For example, just summing the number of Facebook likes by a person's close friends can often be enough to paint a pretty accurate picture of what that person likes.

Given this power of social and demographic recommenders, it isn't surprising that social networks don't easily give their data away. This means that for many practitioners, employing social/demographic recommendation algorithms is simply impossible. However, even when such data is available, it is not always easy to use without creeping users out. Further, privacy concerns need to be carefully addressed to ensure that users are comfortable with using the system.

Contextual recommendation algorithms recommend items that match the user's current context. This allows them to be more flexible and adaptive to current user needs than methods that ignore context (essentially giving the same weight to all of the user's history). Hence, contextual algorithms are more likely to elicit a response than approaches that are based only on historical data.

The key limitations of contextual recommenders are similar to those of social and demographic recommenders – contextual data may not always be available, and there's a risk of creeping out the user. For example, ad retargeting can be seen as a form of contextual recommendation that follows users around the web and across devices, without having the explicit consent of the users to being tracked in this manner.

Limitation of existing recommender system: [4]

- **Biased towards the old and have difficulty showing new**
    - Past behavior [of users] is not a good tool because **the trends are always changing**" (emphasis ours). Clearly an algorithmic approach will find it difficult if not impossible to keep up with fashion trends. Most fashion-challenged people – I fall into that category – rely on trusted fashion-conscious friends and family to recommend new clothes to them.
    - Item recommendations don't work because there are simply too many product attributes in fashion and each attribute (think fit, price, color, style, fabric, brand, etc) has a different level of importance at different times for the same consumer."

- **Changing User Preferences**
    - While today I have a particular intention when browsing e.g. Amazon – tomorrow I might have a different intention. A classic example is that one day I will be browsing Amazon for new books for myself, but the next day I'll be on Amazon searching for a birthday present for my sister (actually I got her a gift card, but that's beside the point).
    - On the topic of user preferences, recommender systems may also incorrectly label users

- **Unpredictable Items**

    - The type of movie that people either love or hate, such as Napoleon Dynamite. These type of items are difficult to make recommendations on, because the user reaction to them tends to be diverse and unpredictable.

- **Shilling Attacks**

    - What happens if a malicious user or competitor enters into a system and starts giving false ratings on some items either to increase the item popularity or to decrease its popularity. Such attacks can break the trust on the recommender

system as well as decrease the performance and quality of recommenders. This threat is of more concern in CF techniques but lesser threat to the item-based CF technique. There are different attack models like bandwagon, random, average, and reverse bandwagon attack. Attacks can be detected through different approaches like generic and model specific attributes, prediction shift, and hit ratio.
These types of attacks can be categorized by dimensions like intent of attacking, size of attack, and the required knowledge to start the attack. [5]

- **Scalability**
    - The rate of growth of nearest-neighbor algorithms shows a linear relation with number of items and number of users. It becomes difficult for a typical recommender to process such large-scale data. For example, Amazon.com recommends more that 18 million items to more than 20 million customers. Different techniques have been proposed including clustering, reducing dimensionality, and Bayesian Network. The problem can be addressed by using clustering CF algorithms that search users in small clusters instead of searching the entire database; by reducing dimensionality through SVD, by pre-processing that combines clustering and content-analysis with CF algorithms, and by using item classification with weighted slope one scheme in determining vacant ratings in the sparse dataset of CF systems. [5]
- **Limited Content Analysis and Overspecialization**
    - Content-based recommenders rely on content about items and users to be processed by information retrieval techniques. The limited availability of content leads to problems including overspecialization. Here, items are represented by their subjective attributes, where selecting an item is based mostly on their subjective attributes. Features that represent user preferences in a better way, are not taken into account. For many domains, content is either scarce such as books or it is challenging to obtain and represent the content such as movies. In such cases relevant items cannot be recommended unless the analyzed content contains enough information to be used in distinguishing items liked/disliked by the user. This also leads to representation of two different items with same set of features, where, e.g., well-written research articles can be difficult to distinguish from bad ones if both are represented with same set of keywords.
      Limited content analysis leads to overspecialization in which CB recommenders recommend items that are closely related to user profile and do not suggest novel items. In order to recommend novel and serendipitous items along with familiar items, we need to introduce additional hacks and note of randomness, which can be achieved by using **genetic algorithms** that brings diversity to recommendations
      being made. The problem is relatively small in CF recommenders where unexpected and novel items may get recommended. [5]
- **Context-Awareness**
    - From an operational point of view, context-awareness aggregates all categories that represent the setting in which recommender is deployed, e.g., the current location, the current activity, and the time. It is envisioned that the upcoming recommender systems will use contextual information obtained through mobile services infrastructure and will include the user's short and long term history, location, entries in the calendar, and the information that the user provides to social networks.

This can greatly affect the performance of recommenders. Finding out user preferences and context-related information is the key to come up with relevant recommendations for the user. Moreover, the performance can be improved if user context-related information can be found out in an unnoticeable way. For obtaining this unobtrusive preference elicitation, three basic methods can be used including detecting facial expressions, recording speech interpretation, and physiological signals analysis. [5]

- **The Long Tail / Lowest common denominator**
    - Television seldom broadcast Beethoven, because this is not something that common audience loves. On the other hand, television broadcasts sports events, because common people love sports events. Therefore, between Beethoven and sport, television will opt for sport. However, if a television has the possibility to broadcast the football match between two famous clubs or a match between two third-league clubs, the television will broadcast the former event rather than the latter one. Therefore, the media do not seek what is the lowest, but only what is the most common. The problem is that the most common is usually not a sophisticated debate or a high art.

Why is long-tail effect a problem in Recommender System?

The long tail is a problem but is also why many recommender systems exist. The items in the long tail are rare, obscure items, they are not very popular. Online vendors, unlike retailers can have these items in "stock", using an ample meaning for stock because frequently there's no physical entity to store at all.

The long tail is critical because if you recommend items in the long tail and the people like them then you are effectively doing "discovery" which is presenting to the users items he/she will like but wouldn't have discovered on his own. The critical thing is that as items in the long tail are very very abundant the recommender system if successful has the potential of returning a big income to the vendor.

The problem is that sometimes the algorithm will think that many items deserve recommendation, for example in the case of Amazon maybe the algorithm decides you might like 2073 books based on some algorithm. Then the problem becomes which items to present and in which order, a problem we call "learning to rank".

So the problem of the long tail is that learning to rank the items becomes critical. A secondary problem is diversity as you want to show items of different types or categories to the user, not all items in your recommendation should be similar. [6]

Business Perspective of long-tail issue

The long tail isn't always a problem. Indeed, there are plenty of businesses that do just fine with a short list of products that are all widely popular (consider McDonald's) or a set of products where they have nearly exclusive access to those products to sell (e.g., Nike sneakers).

The long tail effect becomes a problem—and and opportunity—when you reach a point where simply offering the same popular items is not a viable business strategy. This could be because there's a limit to how often consumers will select those popular items (i.e., you want a greater share of their dollar by having more choices to offer them), or it could be because you don't have exclusive rights to sell those

items, so you're effectively selling commodity goods, where the competition is nearly all on price (which drives down profits).

Two examples. Let's say you're Amazon, and you want people to shop with you rather than look to see if others have cheaper products. In that case, you want to carry as close to everything as possible, and you want to get as much from each customer by offering them things they wouldn't have otherwise considered. An effective recommender (along of course with other forms of competition like free/cheap/fast shipping) can help your customer loyalty, order size, and profits.

Or let's say you're an online news site. Most of the highly-popular news is probably non-exclusive (wire service reports—these days about Olympic medalists and Donald Trump). To get someone to come back to Joe's News rather than some other site, you need to interest them in some content that they can't find elsewhere.

Finally, the long tail is a social good opportunity as well. If you believe that a diversity of products, manufacturers, contents, etc., is good, then you also want to support a marketplace that makes it possible for that diverse set of products to find customers. Recommending in the long tail is one way to help avoid slipping towards a economy dominated by a few mega-players.

Different approach for solving Long Tail Problem

Figure 3 Long tail solutions [7]

Why Evolutionary computation for Recommender System? [8]

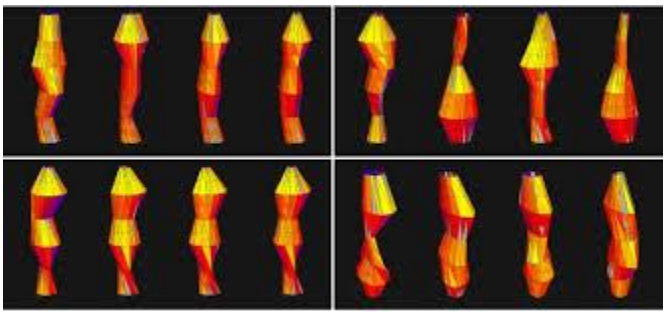


Figure 4 Evolutionary Computation [7]

To provide personalized recommendation, it is necessary to know the user preferences. User preferences can be seen as a mapping from the space of items to the space of preference values, indicating, for each item, how likely the user would prefer it. Thus, user preferences can be expressed by a prediction model whose parameters have to be learned from users' past feedbacks on a set of items. These models can be induced by machine learning (ML) techniques, adapted to a specific domain of recommendation and the available information.

The real-world performance of RS also largely depends on subjective, user-perceived criteria as occurs in the serendipity of user recommendations, favoring the use of multi-objective optimization techniques for learning user preferences.

The second issue concerns more complex recommendation scenarios, like event recommendation. This implies that the user preference model may be hard to express in an analytic or formalized way, therefore should be treated as a black-box function. This additionally implies that the parameters of this model are hard to be learned by traditional optimization techniques.

One way to approach the previous issues would be to employ techniques based on multi-objective optimization of (expensive) black-box. Evolutionary Computing (EC) would be a reasonable choice,

since EC have been successfully used for optimization of real-parameter problems multi-objective optimization tasks and dynamic optimization problems for changing environments.

Some of the analyzed approaches are able to deal with various aspects of RS. This makes the EC-based recommendation techniques worth of further investigation.

The main aim of a RS is to provide decision support for the user in the process of search. However, user's decisions are usually chosen by considering, or rather, aggregating multiple criteria. The ability of EC techniques to deal with multi-objective optimization problems is one of the reasons for further research on using EC in recommendation.

Another reason is concerned with complex recommendations with which, for example, a tourist RS has to deal. Let's imagine a system for recommending a suitable sightseeing route for the user by taking into account a lot of constraints such as types, opening hours and prices of attractions, weather conditions, distances between attractions and the context of the user, etc. Such a recommendation model may be not easy to express in a nice, analytical way, and also, the optimization of its parameters can be a very hard task. EC techniques are not only able to encode complex scenarios but are also suitable to find sub-optimal solutions of hard optimization problems in a reasonable time.

The solutions generated by EC are inherently more diversified due to the stochastic processes driving the search. Such randomness can be helpful in exploring unpopular items, increasing serendipity and novelty of recommendations or addressing the so-called long-tail problem discussed in Yin et al. (2012) which is still challenging.

Strengths of EC based Recommendation systems

EC is suitable for multi-objective optimization problems, Ability of EC to optimize black-box functions

Weakness

Time complexity of EC techniques

Opportunities

Complex recommendation scenarios Models focusing on novelty

Diversity and Serendipity

Long-tail Recommendation

Collaborative Filtering for long-tail

collaborative filtering models which are based on assumption that people like things similar to other things they like, and things that are liked by other people with similar taste.

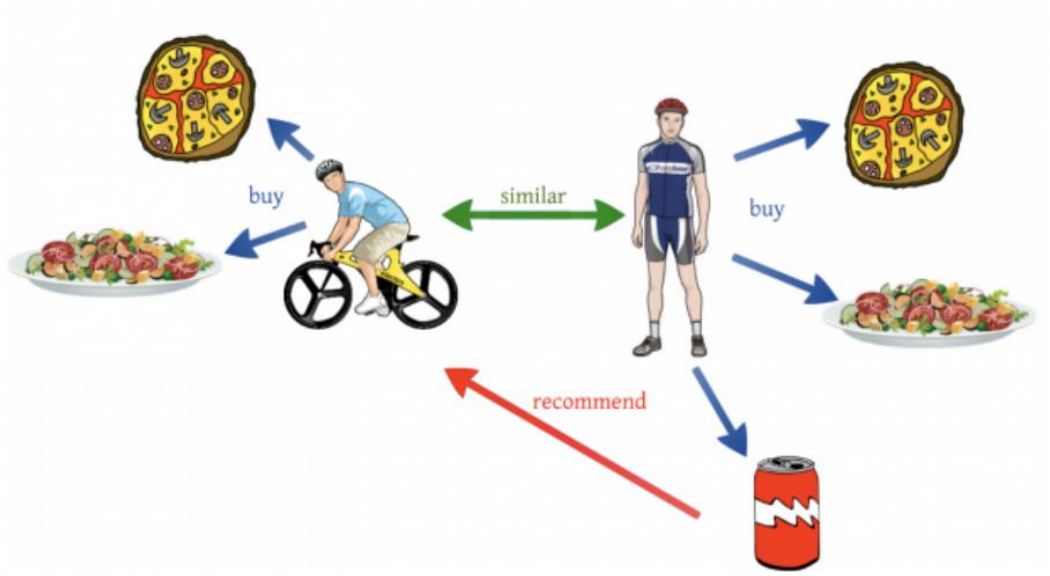


Figure 5 Collaborative Filtering [7]

Types of collaborative filtering techniques

1) Memory based
2) Model based

- Matrix Factorization
- Clustering
- Deep Learning

A lot of research has been done on collaborative filtering (CF), and most popular approaches are based on **low-dimensional factor models**. The CF techniques are broadly divided into 2-types:

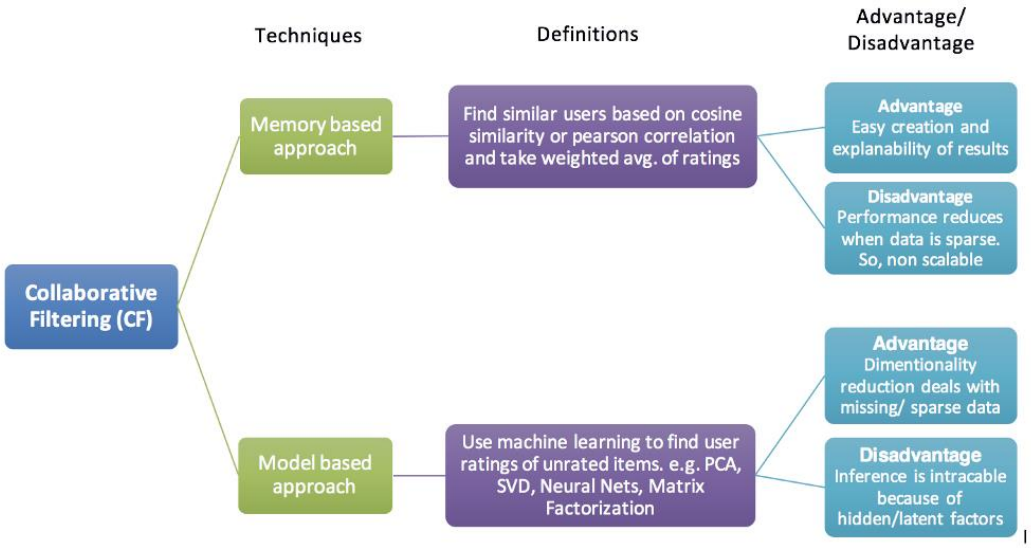


Figure 6 Collaborative Filtering techniques

1. Memory based approach:

Memory-Based Collaborative Filtering approaches can be divided into two main sections: user-item filtering and item-item filtering. A **user-item filtering** takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. In contrast, **item-item filtering** will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

Item-Item Collaborative Filtering: “Users who liked this item also liked …”
User-Item Collaborative Filtering: “Users who are similar to you also liked …”

The key difference of memory-based approach from the model-based techniques is that we are not learning any parameter using gradient descent (or any other optimization algorithm). The closest user or items are calculated only by using **Cosine similarity or Pearson correlation coefficients**, which are only based on arithmetic operations.

A common distance metric is **cosine similarity**. The metric can be thought of geometrically if one treats a given user’s (item’s) row (column) of the ratings matrix as a vector. For user-based collaborative filtering, two users’ similarity is measured as the **cosine of the angle between the two users’ vectors**. For users u and u′, the cosine similarity is:

$$sim(u, u') = cos(\theta) = \frac{\mathbf{r}_u \cdot \mathbf{r}_{u'}}{\|\mathbf{r}_u\| \|\mathbf{r}_{u'}\|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{u'i}^2}}$$

[9]

We can predict user-u's rating for movie-i by taking weighted sum of movie-i ratings from all other users (u′s) where weighting is similarity number between each user and user-u002E

$$\hat{r}_{ui} = \sum_{u'} sim(u, u') r_{u'i}$$

[9]

We should also normalize the ratings by total number of u′ (other user's) ratings.

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u, u') r_{u'i}}{\sum_{u'} |sim(u, u')|}$$

[9]

Final words on Memory-based approach: As no training or optimization is involved, it is an easy to use approach. But its performance decreases when we have sparse data which hinders scalability of this approach for most of the real-world problems. [10]

2. Model based approach

In this approach, CF models are developed using machine learning algorithms to predict user's rating of unrated items. As per my understanding, the algorithms in this approach can further be broken down into 3 sub-types.

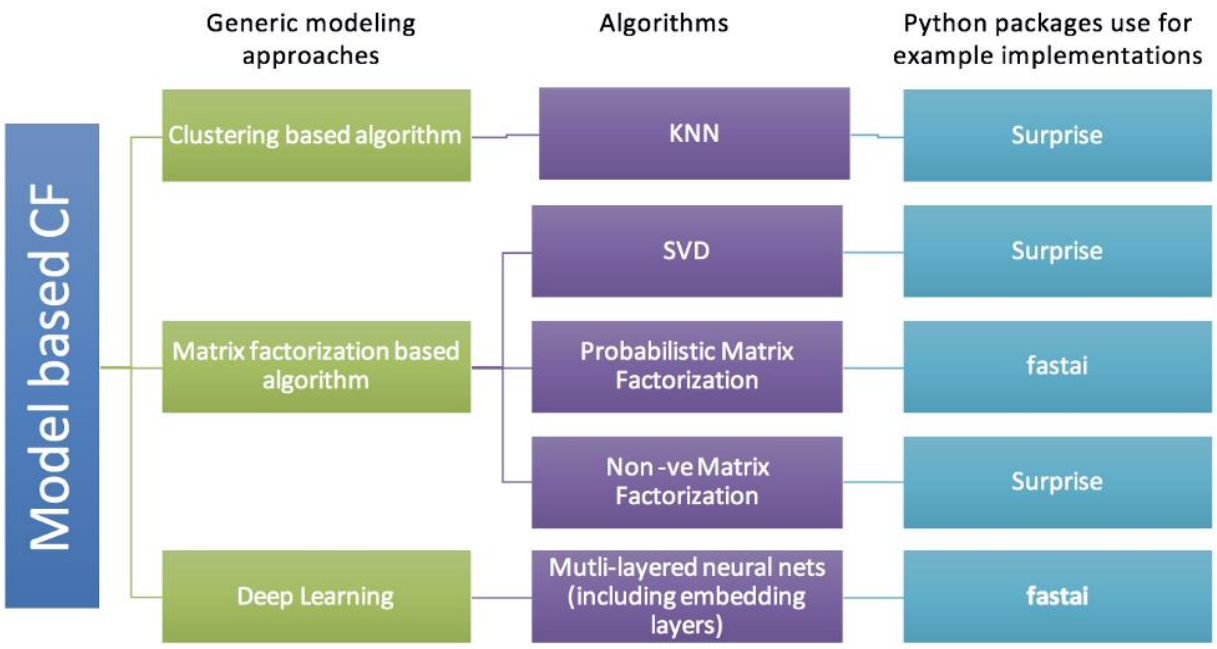


Figure 7 Model Based CF

Clustering Based Algorithm

Clustering based algorithm (KNN):

The idea of clustering is same as that of memory-based recommendation systems. In memory-based algorithms, we use the similarities between users and/or items and use them as **weights** to predict a rating for a user and an item. The difference is that the similarities in this approach are calculated based on an unsupervised learning model, rather than Pearson correlation or cosine similarity. In this approach, we also limit the number of similar users as **k**, which makes system more scalable.

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) [11]:

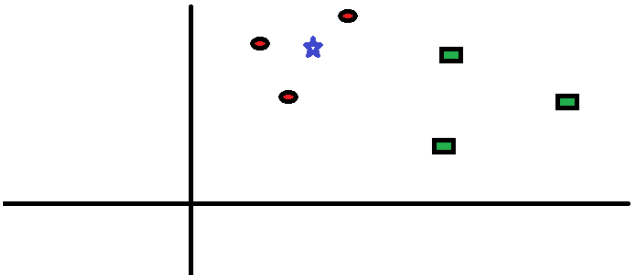


Figure 8 KNN example 1 [12]

You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The “K” is KNN algorithm is the nearest neighbors we wish to take vote from. Let’s say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:

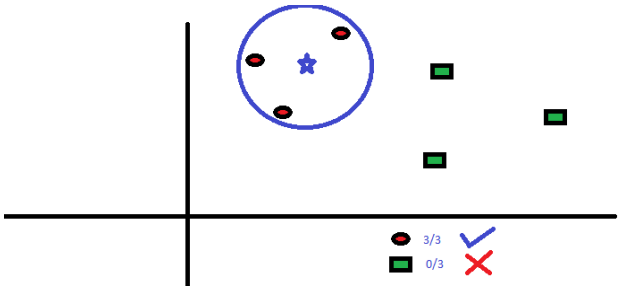


Figure 9 KNN example 2 [12]

The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm.

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} \text{sim}(u,v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} \text{sim}(u,v)}$$

[11]

Figure 10 KNN equation

The actual number of neighbors that are aggregated to compute an estimation is necessarily less than or equal to k. First, there might just not exist enough neighbors and second, the sets Nki(u) and Nku(i) only include neighbors for which the similarity measure is positive. It would make no sense to aggregate ratings from users (or items) that are negatively correlated. For a given prediction, the actual number of neighbors can be retrieved in the 'actual_k' field of the details dictionary of the prediction.

Matrix Factorization based algorithm

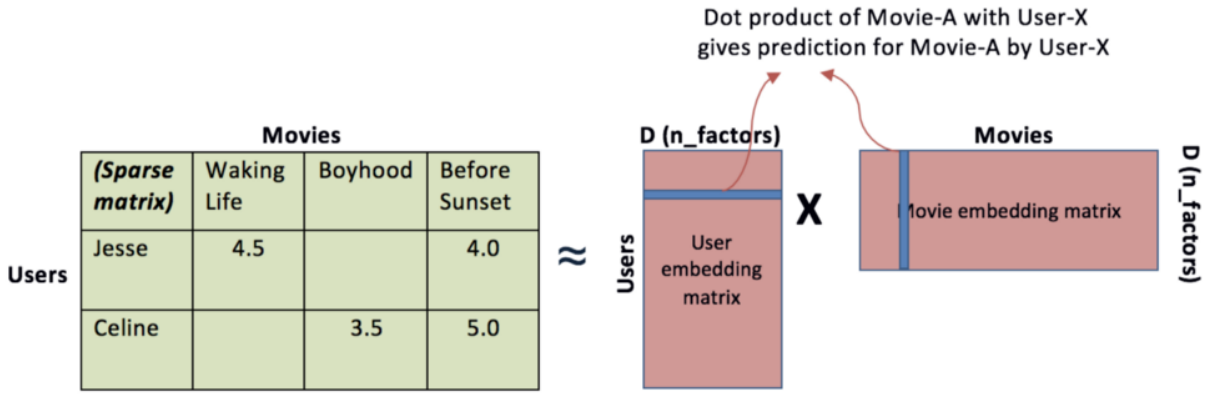


Figure 11 Matrix Factorization [10]

The ratings matrix is equal to the result of multiplying the user attributes matrix by the movie attributes matrix, we can work backwards using matrix factorization to find values for U and M. In code, we use an algorithm called, low rank matrix factorization, to do this. Let's look at how this algorithm works. Matrix factorization is the idea that a large matrix can be broken down into smaller matrices. So, assuming that we have a large matrix of numbers, and assuming that we want to be able to find two smaller matrices that multiply together to result in that large matrix, our goal is to find two smaller matrices that satisfy that requirement.

There are standard ways to factor a matrix, such as using a process called, singular value decomposition. But this is a special case where that won't work. The problem is that we only know some of the values in the large matrix. Many of the entries in the large matrix are blank, or users haven't yet reviewed particular movies. So, instead of trying to directly factor the ratings array into two smaller matrices, we'll estimate values for the smaller matrices using an iterative algorithm. We'll guess and check until we get close to the right answer.

Here's how it works. First, we'll create the U and M matrices, but set all the values to random numbers. Because U and M are full of random numbers, if we multiply U and M right now, the result will be random. The next step is to check how different our calculated ratings matrix is from the real ratings matrix with the current values for U and M. But we'll ignore all the spots in the ratings matrix where we don't have data, and only look at the spots where we have actual user reviews. We'll call this difference the cost. The cost is how wrong we are.

Next, we'll use a numeric optimization algorithm to search for the minimum cost. The numerical optimization algorithm will tweak the numbers in U and M a little at a time. The goal is to get the cost function a little closer to zero at each step. The function we'll use is called fmin_cg. It searches for the inputs that make a function return the minimal possible output. It's provided by the Surprise library. Finally, the fmin_cg function will loop hundreds of times until we get the cost as small as possible. When the value of the cost function is as low as we can get it, the final values of U and M at that point are what we'll use.

But since they're just approximations, they won't be exactly perfect. When we multiply these U and M matrices to calculate movie ratings,and check it against the original movie ratings, we'll see that there's still a bit of difference. But as long as we get pretty close, the small amount of difference won't matter. [13]

Matrix factorization can be done by various methods and there are several research papers out there. In next section, there is python implementation for orthogonal factorization (SVD) or probabilistic factorization (PMF) or Non-negative factorization (NMF).

SVD

SVD in the context of recommendation systems is used as a collaborative filtering (CF) algorithm. For those of you who don't know, collaborative filtering is a method to predict a rating for a user item pair based on the history of ratings given by the user and given to the item. Most CF algorithms are based on user-item rating matrix where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items.

SVD and Matrix factorization

SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from N to K where K < N. For the purpose of the recommendation systems however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix. From a high level, matrix factorization can be thought of as finding 2 matrices whose product is the original matrix.

Each item can be represented by a vector `*qi*`. Similarly each user can be represented by a vector `*pu*` such that the dot product of those 2 vectors is the expected rating

$$expected\ rating = \hat{r}_{ui} = q_i^T p_u$$ [14]

It is a form of factorization!!

`*qi*` and `*pu*` can be found in such a way that the square error difference between their dot product and the known rating in the user-item matrix is minimum

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2$$ [14]

Regularization

For our model to be able to generalize well and

not over-fit the training set, we introduce a penalty term to our minimization equation. This is represented by a regularization factor λ multiplied by the square sum of the magnitudes of user and item vectors.

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

[14]

To illustrate the usefulness of this factor imagine we have an extreme case where a low rating given by a user to a movie with no other rating from this user. The algorithm will minimize the error by giving `*qi*`a large value. This will cause all rating from this user to other movies to be very low. This is intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided

NMF

A collaborative filtering algorithm based on Non-negative Matrix Factorization.

This algorithm is very similar to SVD. The prediction r^ui is set as:

$$expected\ rating = \hat{r}_{ui} = q_i^T p_u$$

The optimization procedure is a (regularized) stochastic gradient descent with a specific choice of step size that ensures non-negativity of factors, provided that their initial values are also positive.

At each step of the SGD procedure, the factors ff or user uu and item ii are updated as follows:

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \in I_u} q_{if} \cdot r_{ui}}{\sum_{i \in I_u} q_{if} \cdot \hat{r_{ui}} + \lambda_u |I_u| p_{uf}}$$

$$q_{if} \leftarrow q_{if} \cdot \frac{\sum_{u \in U_i} p_{uf} \cdot r_{ui}}{\sum_{u \in U_i} p_{uf} \cdot \hat{r_{ui}} + \lambda_i |U_i| q_{if}}$$

[14]

where λu and λi are regularization parameters.

This algorithm is highly dependent on initial values. User and item factors are uniformly initialized between init_low and init_high .

Practical Implementations:

In this section, I explain the experimental settings used for comparing different techniques for predicting sparse data and giving recommendation, including an overview of the data used, selected variables, data predicting methods and performance measurements.

Surprise package:

I have used Surprise package for experimental test of this report. This package has been specially developed to make recommendation based on collaborative filtering easy. It has default implementation for a variety of CF algorithms. [14]

Datasets used for Experiments:

I have used two popular datasets in our study MovieLens [15] and BookCrossing [16] and Hosptial Ratings dataset [17]. The MovieLens dataset contains 100,000 ratings on the scale of 1 to 5 from 943 customers on 1682 movies. The BookCrossing dataset contains 1,149,780 ratings on the scale of 1 to 10 from 278,858 customers on 271,379 books. The Hospital Service data sets consists of 1000 different user's ratings for 1000 different hospitals and it's services. Hospital Compare is a consumer-oriented website that provides information on how well hospitals provide recommended care to their patients. Hospital Compare allows consumers to select multiple hospitals and directly compare performance measure information related to heart attack, heart failure, pneumonia, surgery and other conditions.

	userId	hospitalId	rating
0	10005	1	3
1	10005	2	3
2	10032	3	4
3	10095	4	0
4	10131	5	3

Figure 12 Head view of ratings table of Hospital Service Dataset

Performance Measurement:

Mean Absolute Error (MAE) [18] and Root Mean Square Error (RMSE) [19] as measures. After building the model, we predict the unknown rating on the holdout sample and calculate the error rates as:

$$(1)\ RMSE = \sqrt{\frac{\sum_{i=1}^{n} e_i^{\,2}}{n}} \qquad (2)\ MAE = \frac{\sum_{i=1}^{n} |\, e_i \,|}{n}$$

Figure 13 Performance measurement equations

Experimental Results:

1) GA based Movie Recommender System

   I have created a movie recommendation system where a clustering-based approach is used to find the nearest neighbors, for every user there will a set of other users which will be predicted using Genetic algorithm. The technology I have used here is Java [20]. There are three folds for creating each neighbor and I have divided datasets into training and testing sets. And based on testing sets data and MEA comparison technique I have created the below result table.

```
Mean Absolute Error for fold 1: 1.1986642453097094
Mean Absolute Error for fold 2: 1.1169865412859585
Mean Absolute Error for fold 3: 1.1439829070200882
Mean Absolute Error for the System: 1.153211231205252
```

Figure 14 Results of GA based recommendation system

2) Movie Recommendation system

   I have used surprise-scikit python based framework for creating movie recommendation systems. Compared cosine-similarity, SVD, NMF and KNN optimization algorithms. Below are the results displayed comparing RMSE and MAE for every algorithm.

   Result of SVD :

   Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9423	0.9427	0.9382	0.9349	0.9428	0.9402	0.0032
MAE (testset)	0.7443	0.7433	0.7365	0.7361	0.7417	0.7404	0.0034
Fit time	4.64	4.99	4.70	4.87	4.64	4.77	0.14
Test time	0.13	0.13	0.10	0.13	0.13	0.13	0.01

   Result of KNN:

   Evaluating RMSE, MAE of algorithm KNN on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9796	0.9806	0.9822	0.9823	0.9886	0.9827	0.0032
MAE (testset)	0.7752	0.7717	0.7734	0.7769	0.7776	0.7750	0.0022
Fit time	0.47	0.48	0.48	0.48	0.48	0.48	0.01
Test time	2.77	2.82	2.76	2.73	2.83	2.78	0.04

Result of NMF:

Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9690	0.9683	0.9622	0.9726	0.9714	0.9687	0.0036
MAE (testset)	0.7615	0.7601	0.7558	0.7628	0.7611	0.7602	0.0024
Fit time	4.91	4.95	4.89	4.86	4.87	4.89	0.03
Test time	0.11	0.15	0.11	0.12	0.09	0.12	0.02

3) Book Recommendation system

I have used surprise-scikit python based framework for creating book recommendation systems. Compared cosine-similarity, SVD, NMF and KNN optimization algorithms. Below are the results displayed comparing RMSE and MAE for every algorithm.

Result of SVD :

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8943	0.8896	0.9021	0.8811	0.8933	0.8921	0.0068
MAE (testset)	0.7000	0.6977	0.7069	0.6920	0.6998	0.6993	0.0048
Fit time	4.89	4.84	4.84	4.85	4.84	4.85	0.02
Test time	0.15	0.14	0.14	0.15	0.15	0.15	0.00

Result of KNN:

Evaluating RMSE, MAE of algorithm KNN on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9384	0.9447	0.9376	0.9369	0.9398	0.9395	0.0028
MAE (testset)	0.7463	0.7506	0.7443	0.7425	0.7425	0.7452	0.0030
Fit time	1.24	1.21	1.57	1.26	1.26	1.31	0.13
Test time	2.06	2.53	2.28	2.06	2.11	2.21	0.18

Result of NMF:

Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9308	0.9347	0.9350	0.9383	0.9390	0.9356	0.0029
MAE (testset)	0.7226	0.7267	0.7284	0.7311	0.7300	0.7277	0.0030
Fit time	5.81	5.62	5.57	5.57	5.60	5.63	0.09
Test time	0.13	0.12	0.18	0.12	0.13	0.14	0.02

4) Long tail problem in Hospital Recommendation system

Result of SVD :

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.5043	1.5211	1.5045	1.4947	1.5336	1.5116	0.0139
MAE (testset)	1.2950	1.3184	1.2998	1.2839	1.3280	1.3050	0.0160
Fit time	0.22	0.23	0.23	0.24	0.22	0.23	0.00
Test time	0.00	0.00	0.00	0.01	0.00	0.01	0.00

Result of KNN :

Evaluating RMSE, MAE of algorithm KNN on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.5258	1.4789	1.5320	1.4839	1.5384	1.5118	0.0252
MAE (testset)	1.3144	1.2745	1.3245	1.2721	1.3413	1.3054	0.0275
Fit time	0.20	0.22	0.21	0.21	0.22	0.21	0.01
Test time	0.01	0.01	0.01	0.01	0.01	0.01	0.00

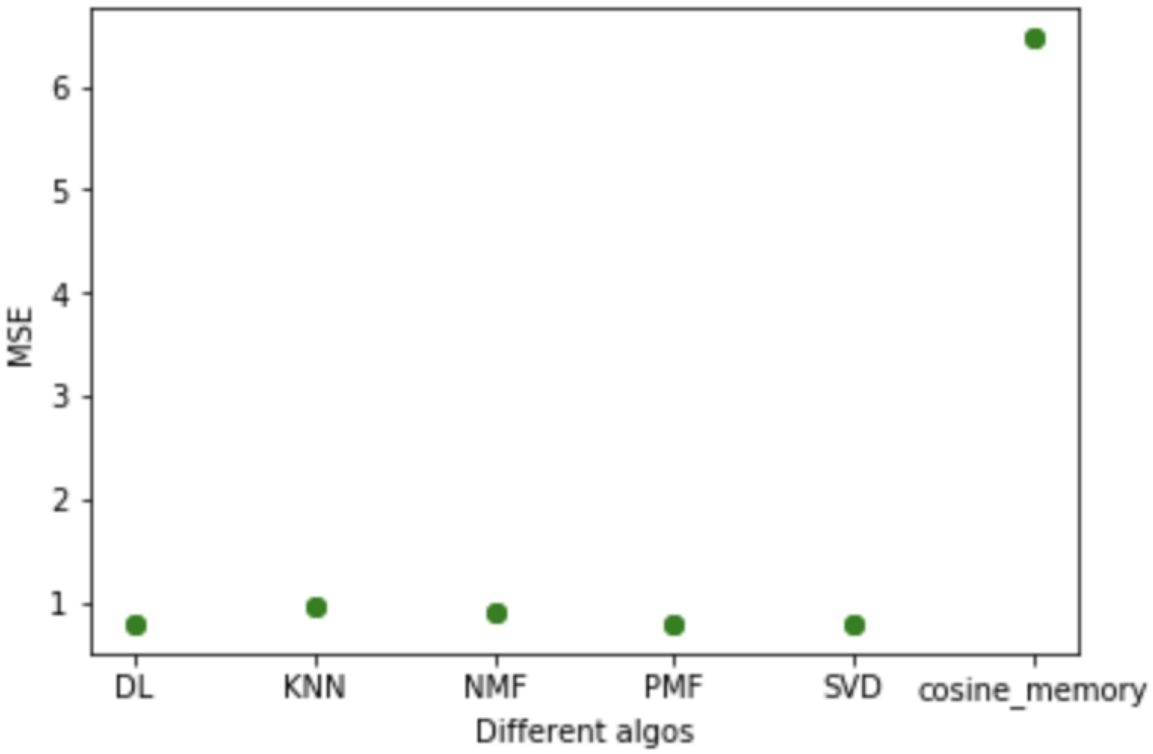
Result of NMF :

Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.5194	1.5194	1.5151	1.4946	1.5117	1.5121	0.0092
MAE (testset)	1.3146	1.3096	1.3107	1.2925	1.2985	1.3052	0.0083
Fit time	0.35	0.36	0.34	0.35	0.36	0.35	0.01
Test time	0.00	0.00	0.00	0.00	0.01	0.01	0.00

Discussion & Conclusion

Below is the plot of MSE obtained from different approaches on MovieLens 100k data.



SVD and Neural net (DL) give the best results. Neural net implementation will also perform well on imbalanced data, with infrequent users unlike other MF algorithms.

It can be useful to built customer targeted recommendation system for your products/ services. Most easiest and well-researched method out there is collaborative filtering.

Future Work

In future the long-tail problem can be further optimized using Deep learning methodologies. Also matrix factorization based collaborative filtering can be tested with Evolutionary algorithms for unstructured data types.

References

[1] SudeshnaPAl. [Online]. Available: https://github.com/sudeshnapal12/Long-Tail-Recommendation.

[2] [Online]. Available: https://en.wikipedia.org/wiki/Recommender_system.

[3] Z. A. a. I. U. Shah Khusro, "Recommender Systems: Issues, Challenges, and Research Opportunities," 2016.

[4] R. Macmanus, January 2009. [Online]. Available: https://readwrite.com/2009/01/28/5_problems_of_recommender_systems/.

[5] Z. A. a. I. U. Shah Khusro, "Recommender Systems: Issues, Challenges, and Research Opportunities," in *Springer*, 2016.

[6] L. Argerich, "Quora," 22 May 2016. [Online]. Available: https://www.quora.com/Why-is-long-tail-effect-a-problem-in-recommender-systems. [Accessed 21 7 2018].

[7] [Online]. Available: https://images.google.com/.

[8] T. Horváth, "Evolutionary computing in recommender systems: a review of recent research," 2016.

[9] "Surprise for Recommendation System Python framework," [Online]. Available: https://surprise.readthedocs.io/en/stable/index.html.

[10] "Prince Grover," December 2017. [Online]. Available: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0.

[11] [Online]. Available: https://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms.knns.KNNBasic.

[12] [Online]. Available: https://www.youtube.com/watch?v=UqYde-LULfs.

[13] "Lynda," [Online]. Available: https://www.lynda.com/Data-Science-tutorials/How-matrix-factorization-works/563030/600827-4.html?autoplay=true.

[14] "Surprise python framework for Recommendation System," [Online]. Available: https://surprise.readthedocs.io/en/stable/index.html.

[15] [Online]. Available: http://movielens.umn.edu..

[16] [Online]. Available: http://www.bookcrossing.com/.

[17] Aswathi, "Kagle datasets," 2017. [Online]. Available: https://www.kaggle.com/aswathi15/hospital-compare-dataset-analysis/notebook.

[18] [Online]. Available: https://en.wikipedia.org/wiki/Mean_absolute_error.

[19] [Online]. Available: https://en.wikipedia.org/wiki/Root-mean-square_deviation.

[20] S. sethia. [Online]. Available: https://github.com/sidharthsethia/Genetic-Algorithm-based-Recommender-System-.

[21] [Online]. Available: https://images.google.com/.