What Should I Watch Next: A Movie Recommendation Study

Shruti Jagadeesh Bhat

Tri Han

Krishna Sahithi Manneti

Santa Clara University

Summer 2020

Acknowledgements

We would like to thank our parents,

our professor, and our classmates

for their continued support

Table of Contents

List of Figures

List of Tables

Abstract

Historically, when searching for the best option, one would typically refer to an expert's advice for guidance. Who is the best doctor for a specific disease? Which restaurant serves the kind of food that I like the most? All of these kinds of problems require the input and advice of someone with knowledge and expertise. What if we told you that with modern technology that is no longer the case?

In this paper, we address a similar problem. These kinds of questions and problems fall under a field of data mining called Recommendation Systems and it is the aim of this paper to propose such a system. Specifically, this paper aims to address the issue of recommending movies that are most likely to be enjoyable to watch for a user.

Though the existing traditional collaborative and content based approaches in recommender systems are effective, these methods have certain drawbacks like cold start, scalability and sparsity in collaborative approach and diversity problem in content based. In this project, we propose two hybrid models – SVD Content based model and LightFM model which resolves the issue with cold start, diversity problems and provide relevant high quality recommendations.

II. Introduction

In the field of data mining, there is an assortment of problems and applications that one can study. Some topics that are often studied include Finding Similar Items, Data Streams, Clustering, and Large-Scale Machine Learning. There is an endless array of problems and associated approaches to solving these problems. In this paper, we narrow the scope of this research to the area of data mining called recommender systems. In particular, we study the various approaches to building a recommendation system and how it is applied in the specific problem space of movie recommendation.

II.A. Objective

In this paper, there are several explicit objectives that we have. These objectives are (1) the study of recommendation systems and its application in movie recommendations, (2) the study of existing and state of the art approaches to understand the current state of affairs, and (3) to design and propose a recommendation system that is able to outperform existing methods in a particular scope. Below, we will briefly describe each of these objectives in a bit more detail.

The study of recommendation systems and its application in movie recommendations is important because it allows us to understand the baseline methods that are used to solve this problem. Underlying this objective is the study of collaborative filtering, content-based filtering, and hybrid systems that underpin the backbone of any recommendation systems. Without this knowledge, we are unable to grasp the more state of the art technology and approaches that have since replaced these core techniques.

The study of existing methods and current research to understand the current state is another important objective because it allows us to expand on the baseline knowledge and understand where the community stands in terms of progress. For instance, using only the collaborative and content-based filtering approaches, there are many known limitations and gaps that afflict those approaches. Through the years, researchers have built upon that knowledge to propose and design better methods that help to close those gaps. It is the study of this body of work that is crucial so that we are able to pick up as best as we can from where things currently stand.

The third objective of this paper is to propose a recommendation system that outperforms existing recommendations systems in a particular scope. What does this mean? By understanding the baseline methods and work since then, we are able to identify where there are still gaps in the approaches and then we are able to propose a solution that closes those gaps. In essence, we are contributing to this body of work to help move progress forward by proposing two hybrid models that solve the issues underlying the traditional methods.

II.B. What is the problem

To illustrate the problem, we will first describe a scenario before diving into an explanation. To begin, imagine yourself at home. You are sitting in front of the tv and you want to watch a movie. However, when you scroll through the list of available movies, there are so many and you have a hard time picking. A couple of the movies look promising but you wonder if there is anything better. There are other movies that you

know are certainly not your kind of movie. How can the system quickly and accurately recommend a list of movies that you are most likely to watch and enjoy? That is the problem that we are addressing in this paper.

Specifically, given a user's movie preferences and tastes, out of all possible movies, which one should we recommend that they will most likely enjoy watching? This is a simple problem statement. However, there are many nuances that we need to consider. For instance, what if there is no existing data about this user's preferences? Or, what if there are new movies that have no ratings yet? What if this user loves watching unique and niche kinds of movies that are not popular? Are we able to find and recommend those movies to this user? There is an endless list of considerations that we need to take into account such as those just mentioned. It is the aim of this paper to identify those problems and propose a solution that addresses each one.

II.C. Why is this project related to this class

This project is related to this class because of several reasons. The first reason is that this project is a specific study of a data mining problem. As listed above in the introduction, data mining problems span a wide range of topics, ranging from finding similar items to large scale machine learning problems. This particular project focuses on the study of recommendation systems, which is one of many subjects in data mining.

Taking this thought a step further, it might even be appropriate to say that this project spans across multiple areas of data mining, making it even more appropriate for this class. For instance, in this problem, we need to find similar items (or movies in this case) which is another field of study under data mining. Not only that, we may also need to apply

clustering algorithms in order to determine similar users and similar movies that we can then derive movie recommendations from. To gather the appropriate data to be used in this study, we may need to mine social-network graphs, which is yet another field of study in data mining. How will we synthesize all of this together? We will need to use large-scale machine learning algorithms in order to apply the concepts and derive recommendations to the user. This is why this project is applicable and related to this class.

Another reason why this project is related to this class is because it trains us to be effective researchers in the field of data mining, which allows us to be competitive in the real world after graduation. For those who seek to solve similar data mining problems post graduation, this project helps them understand the process of (1) understanding base knowledge, (2) studying state of the art research, and (3) proposing and applying new concepts to further the field of research. By going through this process and being familiar with it, upon post graduation, we will be able to solve any related problem under the recommendation system field of study by knowing what to look for and how to best approach the problem.

II.D. Why other approaches fall short

Studying the baseline methods and the contributions to this field of study, it is apparent that known approaches to recommendation systems are sufficient. They all perform the job reasonably well. However, where they fall short is often usually in a very niche and specific scope that is unique to each method. Where one method is strong, another

method may be weak, and vice versa. In order to understand these shortcomings, a brief description of known limitations for recommendation systems is presented below.

The first problem that some methods face is called the Cold Start problem (also known as the First Rater Problem). What is this problem exactly? Essentially, the reason for this problem is that systems that rely on historical data will not be able to recommend new users or items as accurately because there is no historical data to use for them. For example, let's imagine that a user enjoys watching the Avengers movie series. Oftentimes, the recommendation system does a good job of recommending similar action and superhero based movies to the user. However, imagine that a new Avengers movie has come out this year and there is no existing data for this movie. Intuitively, the system should recommend this movie to the user because it is part of the franchise. However, it does not! Why? There is no data to associate the user's preferences with this movie. This is known as the cold start problem and it is a limitation that is often faced by systems based on the collaborative filtering approach.

Another limitation that is often faced by systems is called the scarcity problem. What is the scarcity problem? This issue is when the user and movie matrix is sparse which makes movie recommendation challenging since it is harder to find similar users. This idea is still confusing so I will illustrate it with an example. Imagine there are 100 movies and imagine that there are 2 users, User A and User B. User A has watched movies 1 and 2 while User B has watched movies 3 and 4. If we were to construct a user by movie matrix for this illustration, one can see that most of the matrix is sparse. It is very difficult to find any correlation between these two users! Let's take it a step further. Let's imagine that both User A and User B like movie 5. However, that is only one movie out of 100 that

they share a common interest for. Does this single data point really tell us anything? This is the limitation of recommendation system called the scarcity problem. It is often faced by traditional collaborative filtering approaches and is a limitation that we hope to address in this paper.

Another limitation of recommendation systems is that the systems are often seen as a black box when it comes to providing recommendations. Why was a particular movie recommended? Why was another one not recommended? Although in practice, the final results matter more than the reasoning and underlying logic to coming up with these recommendations, it is still an important aspect to be cognizant of. Content based filtering approaches are able to provide transparency into this black box. However, other systems have a harder to show this same level of clarity.

Another limitation of recommendation systems is the challenge in finding the appropriate features to use in the model. For instance, what about a user is important enough so that it will accurately predict what kind of movies they will enjoy watching? Or, alternatively, what is it about a movie that will make it more appealing to a specific user or not? Coming up with these features is often challenging, difficult, and does not follow any one scripted approach. Often, it relies on intuition and heuristics. This is a challenge that content-based approaches suffer from.

Another problem faced by certain recommendation system approaches is overspecialization. What is this problem? This problem is essentially the tuning of the model to precisely recommend movies to a particular user based on their likes and preferences. However, because it is specially tailored for this user, there is no input or factor outside of this particular user that will influence the movies that are recommended.

Essentially, what ends up happening is that this user will be shown the same kinds of recommendations again and again without much variety. To illustrate this dynamic, imagine a user says they like comedy movies. For a particular recommendation system, it will accurately recommend and suggest comedies to this user. However, let's imagine that the user also likes action movies, but they never specify that! In effect, what ends up happening is that the user will never be shown an action movie to watch even though they do like it! This is what is known as overspecialization. Oftentimes, the kind of recommendation system that suffers from this problem is the content-based approach method.

One final known limitation of recommendation systems is what is known as popularity bias. Popularity bias is basically when popular items, with the highest and most ratings, bias the recommendation so that these kinds of movies are more often recommended than those movies with not as many ratings. Normally, recommending really popular movies is a good thing right? Well, in some cases, it is not. For instance, imagine a user who loves independent movies. These kinds of movies are not well known but to certain users they are their preferred kind of movie to watch. Due to popularity bias, it is possible that users with this preference will not be recommended independent films. Instead, they would recommend mainstream movies that are popular to the masses. Unfortunately for this particular user with niche preferences, they will not be recommended independent films as often as they would like. This is the issue that is caused by popularity bias. Oftentimes, recommendation systems that are based on collaborative filtering face this kind of problem.

II.E. Why our approach is better

Now that we have gone through the list of known limitations of existing recommendation system approaches, we now present an approach that we believe fills in the gaps mentioned above. Because every recommendation system has their own strengths, it would not be wise to dismiss them and just throw them out. Each one has a specific strength over the others. If each system has a problem that is unique to them and only them, how can we resolve it?

The most simple and straightforward suggestion is to combine these systems into one hybrid system, where the different approaches can overlap and the strengths of one system can make up for the weaknesses of another system. From the study of collaborative and content-based systems, it is apparent that both approaches have their own set of weaknesses. But when viewed at a high level, one can see that they complement each other well. The gap of one system is met by the strength of the other. With these two systems, there are still yet potential gaps that can be improved upon. In this case, we add another approach to this hybrid system that helps fill in this gap.

II.F. Statement of the problem

The problem that we want to address is simple: how do we provide a list of recommended movies that the user is most likely to enjoy watching given the common issues faced by existing models such as cold start, scarcity of data, etc.

II.G. Area or scope of investigation

Given the synopsis above, and after discussing all the different aspects of this undertaking, we want to clarify what will not be covered in this paper. Specifically, we will not be covering recommendation systems that involve recommending videos on social networks. For instance, let's take a look at YouTube. It is one of the largest video sharing websites (if not the largest) in the world. Although video recommendation may seem similar to movie recommendation, these two areas have two separate scopes and we will not be covering this kind of recommendation in this paper. Yes, the methods are similar but there are nuances to video recommendation on YouTube that needs to be addressed but is not covered in this paper.

Another scope that we will not be covering in this paper is streaming data. The data sets that we use in this paper is pre-fetched and stored in data storage (on disk for instance). We do not make use of streaming data to improve and provide recommendations in a real time fashion. We believe that there is no added benefit to providing recommendations using real time data, as most ratings of movies are already established and known in advance.

In this paper, we focus primarily on using existing movie data sets in order to provide users with accurate and reliable movie recommendations that they most likely and most probably enjoy.

III. Theoretical bases and literature review

III.A. Theoretical Background of the problem

With every research paper, the proposed algorithms are intended to make the system better. We referred to research papers as recent as 2018 to 2020 from IEEE and ACM conferences by researchers from different organizations. One of the major resources for data scientists, “kaggle.com,” is also our research pit, along with a couple of other website links mentioned in the reference section.

As we were researching these baseline models of Recommendation Systems, we found that the problems with collaborative filtering and content-based filtering are quite prevalent in the world of recommendation systems. Almost every research paper mentions the issue of not having sufficient data in the initial stage for prediction, especially when the recommendations are based on the history of user selections.

Collaborative Filtering (CF)

CF is carried out by considering the opinion of the popular crowd. Each user’s item preferences and user information are stored for say, ‘**n’** number of users and ‘**i’** number of items, multiple features about the products bought, are stored. A boolean matrix is created for each user-item combination for size **n x i**. For ‘**n+1’** th user, the previous data is used to recommend the product which users of similar tastes have bought so far.

Research stumbles when the data is freshly created, where there is not enough information about the previous users/items to make good predict. This is referred to as “cold start”. This is one of the issues that led us to pick this topic for our research.

There is another method which may seem to overcome this issue - Content based filtering (CB): In this filtering technique, each individual’s opinions about an item is treated independently, with no connection to the taste of other customers. This has its own merits and demerits. Merits being, the outliers have equal weightage in consideration, meaning the customers with unique tastes can be recommended with appropriate products to them, even if no one else preferred that item, as it is completely objective. This solves the cold start issue mentioned before. The demerits to this are the problem that we are trying to address in our proposal. The product details are still required before the system learns to recommend more, which is another version of cold start. This leads to unresolved issues in the research by only using these two filtering techniques.

Furthermore, there are other issues in collaborative filtering techniques that data scientists have faced, including scalability and sparsity. With the transition to big data, millions of users and products began to become a part of the recommendations, which extracted extreme high computing resources. The increase of data size over time led to the need for the reduction of data, which is done by a method called “Matrix Factorization”. Regarding scarcity, not all the users end up mentioning their opinions about the products they bought online. Even for the popular products, there are only a few ratings and fewer reviews. This brings down the quality of the recommendation systems as the user-item matrix is highly sparse, making the computing inefficient in terms of both hardware and software.

III.B. Related research to solve the problem, Merits and Demerits

Data Reduction

1. Memory based Collaborative Filtering

The researchers came up with a reduction technique to store only the relevant data according to the distance (or similarity) based neighborhood approach on the raw data, so that the further calculations can be done on the user's preferences alone. This technique is called **Dimensionality Reduction**.

Reasons to reduce dimensionality:

- Computational issues with large number of predictors
- Certain statistical methods like regression could not be applied when the density of observations per predictor is low.
- Noisy data provides less accurate results

In other words, only the tastes and preferences of the user is derived from the raw data using the Dimensionality Reduction aka, low-rank Matrix factorization.

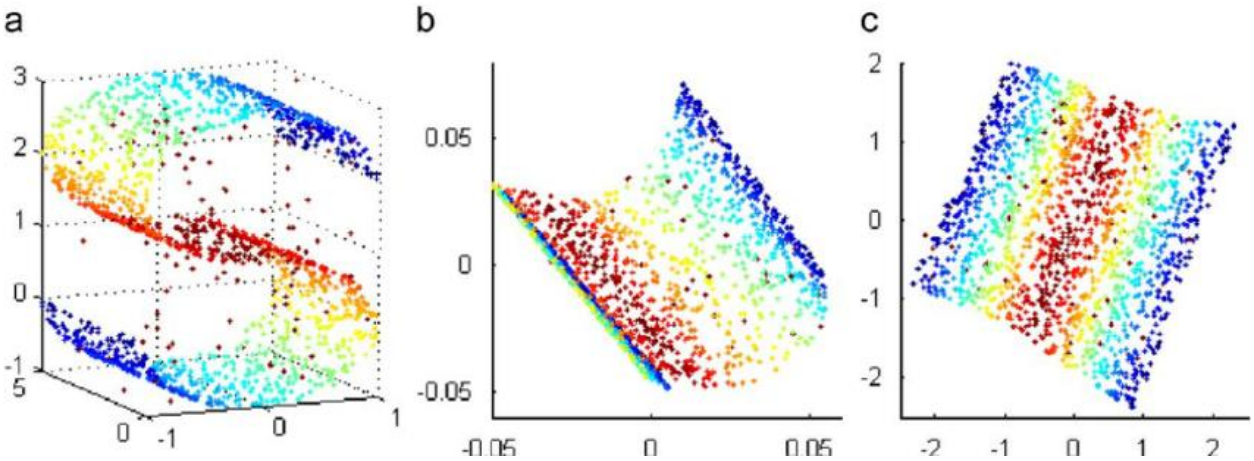

Figure 1: Example for Dimensionality Reduction: Image courtesy - "Local linear transformation embedding" research gate publication by Chenping Hou)

2. Model-based Collaborative filtering

This type of technique for data reduction is based on “Matrix Factorization”, an unsupervised learning method for latent variable decomposition and dimensionality reduction. In this technique, the user-item Matrix is decomposed into a product of two lower dimensionality rectangular matrices.

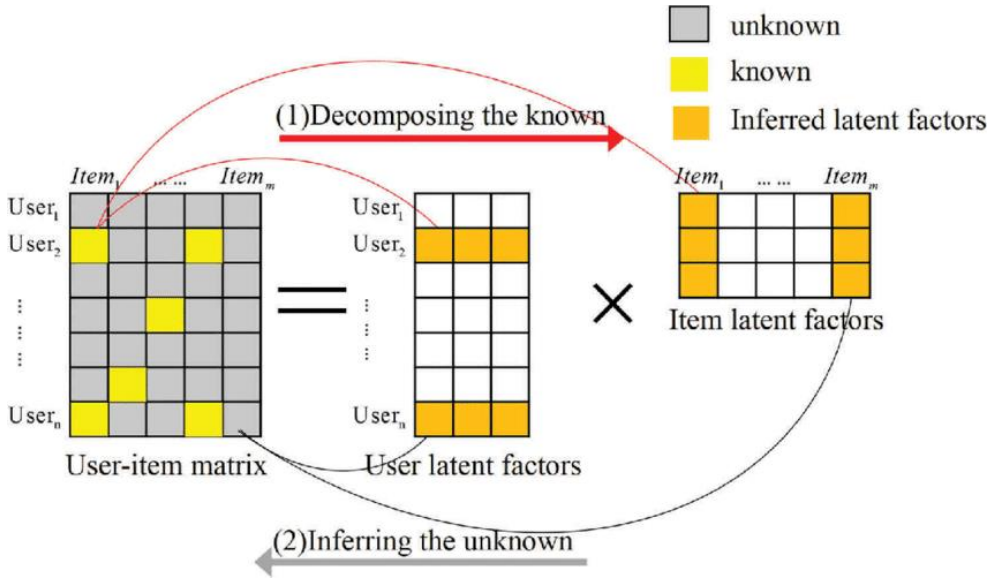

Figure 2: User-Item Matrix Factorization. Image courtesy - ResearchGate.net from “Integrating spatial and temporal contexts into a factorization model for POI recommendation” pub.)

In the recent years of research, both the techniques mentioned above have been combined to form a hybrid approach, with a better system to recommend products. There are various ways of combining these techniques, either making predictions separately and merging them together, or add one of the techniques to another, or combining CB and CF along with some other models too.

We, as a part of our proposal, intend to combine one of these hybrid techniques with other models which we researched upon, majorly, hybrid CB + SVD and LightFM models.

SVD model, or Singular Vector Decomposition model

SVD is a well-known Matrix factorization method, where a matrix A is decomposed into its best possible lower rank matrix approximation. Mathematically, it is decomposing the matrix into two unitary* matrices and a diagonal matrix.

(*In linear algebra, a complex square matrix U is unitary if its conjugate transpose U^T is also its inverse, that is, if where I is the identity matrix. Unitary matrix is also known as orthogonal matrix)

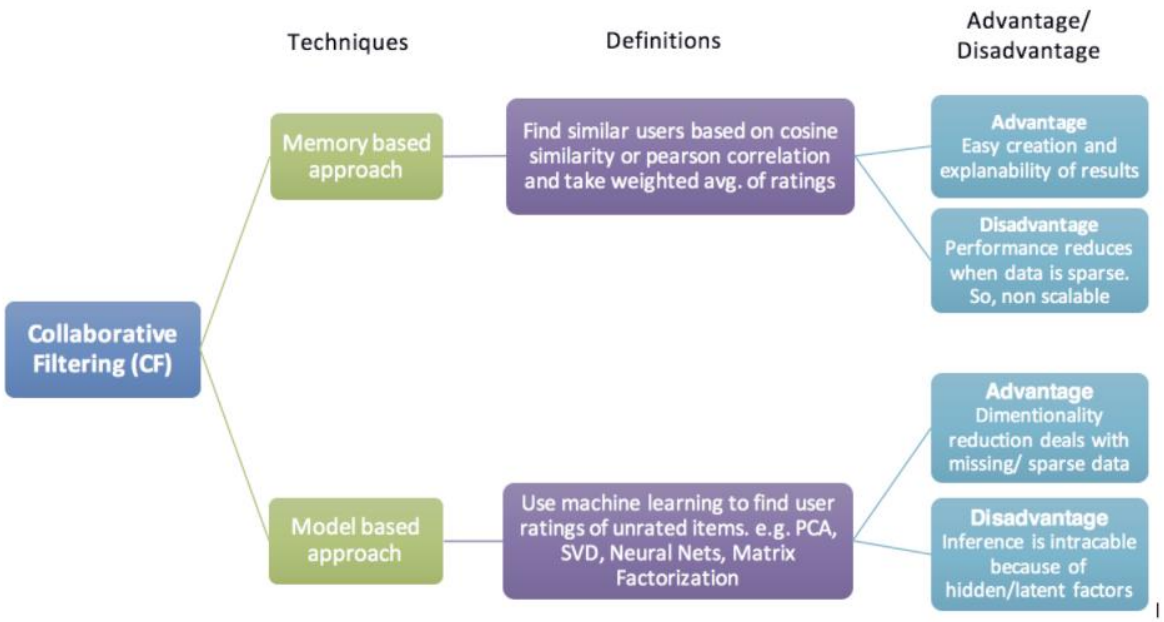

Figure 3: Advantages and Disadvantages of CF techniques. Image courtesy: "Movie Recommendation Systems" by Vivek dalal, Raj Sankhe, Tej Sankhe

In this model a matrix is decomposed into three other Matrices :

$A = USV^T$.....(1)

Where:

A is an m × n matrix

U is an m × r orthogonal matrix

S is an r × r diagonal matrix

V is an r × n orthogonal matrix

The diagonal matrix S is collapsed into a vector aij .

$$a_{ij} = \sum_{k=1}^{n} u_{ik} s_k v_{jk}$$ .....(2)

The variables {sk} are called “singular values” and are arranged in descending order.

$$s_{i+1} \leq s_i$$

The columns of U are called left singular vectors, while those of V are called right singular vectors.

Also since U and V are orthogonal,

$$U^T U = V V^T = I$$ .....(3)

I is called Identity Matrix (Only Diagonal values are 1, rest are 0)

Advantages

Decomposing matrix into relatively smaller vectors, it tackles the scalability problem by dimensionality reduction.

Latent Factor Model - A modern recommender system

So far the models that were discussed above were based only on the available data or the history of preferred items. In more recent research, another layer is added to the previous ones for optimized recommendations, where machine learning concepts have been introduced into this system to predict the missing ratings.

In the case of User-Item matrix R, we know that it is a sparse matrix with a lot of missing values and the Latent Factor Model is meant to predict them. If matrix R can be represented as $R = P \bullet Q^T$

where ,

$P \in m \times k$

$Q^T \in k \times n$

$R \in m \times n$,

Thus automatically able to generate all values of $rij \in R$, $rij = piqj^T$, which is achieved by the SVD as well.

SVD minimizes the error using the RMSE (Root Mean Squared Error), by applying the approximation,

$$\min_{P,Q} \sum_{(i,j)\in R} (r_{ij} - q_i p_j^T)^2$$

LightFM hybrid recommender model

Another proposed model is LightFM, which is one of the hybrid matrix factorization models, where users and items are represented in a linear combination of the content feature's latent factors.

Advantages

This model is better than the previous hybrid model discussed, as,

- It provided good results as the hybrid model of CB and CF, in that it performed well even for the cold start and sparse user-item matrix.

- When the collaborative information is sufficiently large in the training set or when the user features are available, it outperforms the content based (CB).
- it even performs as well as the collaborative matrix factorization model for the warm-start, dense user-item matrix (i.e. When the data is abundant)
- Another advantage of LightFM, it implements implicit feedback to train the system, hence performs well for such a scenario.
- Tested and used by many developers and brands resp.
- Model Evaluation metrics provision for evaluating the performance of the model
- Faster

III.B. Our solution to solve this problem

Importing the models discussed so far (ie. Hybrid recommendation systems for content filtering and Singular Value Decomposition, and Hybrid LightFM model), we intend to replace the baseline recommender systems with these hybrid systems to potentially create a more accurate recommendation system for the dataset chosen, for the cold start and sparse matrices. We intend to choose the best of both worlds and integrate in the hybrid CB - SVD model and Light FM models. We propose showing the results from the comparison of these two models and present them here in our paper.

III.C. Where our solution is different from others

Most proposed solutions either use the traditional baseline models to solve the problem. Because there are limitations to the baseline models, we will take it a step further by proposing hybrid models and we hypothesize that will deliver better quality of results in

terms of the relevance in the recommended items. While the cold start and scarcity issue still exists, we intend to reduce its effect on the recommendations.

IV. Hypothesis

Our hypothesis for this paper is such: to develop hybrid models, specifically the SVD Content based model and the LightFM model, that will achieve better results than baseline recommendation systems. We will test this by comparing our proposed hybrid models against the baseline models.

V. Methodology

V.A. How to generate/collect input data

For developing and evaluating our models in this project, we have used MovieLens dataset which was collected as part of GroupLens Project in University of Minnesota. This dataset contains comprehensive information on movies, users and the individual ratings for a movie by user. The dataset contains 100,836 ratings on 9742 movies by 610 users and contains 3683 tag applications. This data is generated between March 29, 1996 and September 24, 2018. Each user in this dataset rated at least 20 movies and is uniquely represented by an ID. The data is available in four files in csv format - links, movies, ratings and tags. Links file contains three fields which link to the popular open IMDB and TMDB databases and is used for getting more contextual information from the IMDB database. Movies file contains three fields related to movie information like movie name, and its genres. Ratings file contains ratings given by each user, and are on a 1-5 scale with half-star increments. Tags are the user generated metadata about movies.

In addition to the MovieLens dataset, we also made use of IMDB's movie data set. Why the need for this data set? In the MovieLens data set, there was a file for movies. However, it lacked additional dimensional data such as director, actors, writers, Production Company, etc. By linking the MovieLens data set to IMDB's movie data set, we are then able to obtain dimensions, such as release year, country, language, director, writer, actors, etc, which will then be used in the content based recommender system. As an overview, the movies csv file has data on over 80k movies, more than enough for our purposes in this study.

V.B. How to solve the problem

Many techniques in data mining, information retrieval and machine learning exist and are widely used in recommender systems to get personalized recommendations. These traditional methods are effective, but each of these methods lack in different areas. For example, the cold start problem is not seen in a content based traditional approach while it is quite evident in a simple collaborative model. So combining these methods and building a hybrid model not only overcomes these drawbacks, but also provides recommendations of high quality.

V.B.I. Algorithm Design

In order to solve issues like cold start in existing collaborative filtering approaches and diversity problems in content based filtering techniques, we proposed two hybrid recommendation systems which take the best of each approach to provide high quality recommendations. One such hybrid recommendation model is a combination of a movie content based model and Singular Value Decomposition (SVD). Another proposed model is LightFM which is a hybrid matrix factorization model that denotes users and items as a linear combination of content features latent factors. In order to compare our proposed models, three baseline models are implemented. One is a simple content filtering model, and the other two models are based on collaborative filtering techniques using KNN and SVD++ models.

Hybrid content based filtering and Singular Value Decomposition model:

It is a combination of two standalone models – content based model and SVD model.

The content based model output is sent to the SVD model which gives a list of the recommended movies along with the user's likely ratings. The higher the user's predicted ratings are, the more relevant the recommendations for a particular user are generated.

Content based approach:

This is a data mining technique which finds similarity between movies based on contextual information on movies like genre, actor, director and recommends movies which are more similar to a movie that a user watched or liked. It mainly contains two steps.

- The first step is to find the relative importance of the movies with help of contextual information on movies using TF-IDF. These measures are found as there is no quantitative approach in the text.
- The second step is to find similarity between the movies.

In order to find the relative importance of a movie with the help of contextual information, term frequency (TF) and inverse document frequency (IDF) are used.TF gives the frequency of a tokenizer in a particular movie context while IDF is the inverse of the context frequency among the whole corpus.

$$TF(i,j) = \frac{\text{Term i frequency in document j}}{\text{Total words in document j}}$$

$$IDF(i) = \log_2\left(\frac{\text{Total documents}}{\text{documents with term i}}\right)$$

$$TFIDF\ score\ for\ term\ i\ in\ document\ j = TF(i,j) * IDF(i)$$

Cosine similarity is the cosine angle between two vectors of inner product space. In other words, it is the inner product of two vectors over the L2 norm of each of these vectors.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

This will give the list of movies which are highly similar and recommend them to the user.

Singular Value Decomposition:

This is a collaborative filtering technique which uses linear algebra concepts. This method helps in learning latent factors, highly scalable and works well even in the presence of noisy data. As per the SVD method, the user-movie-ratings matrix, M of dimensions, mxn, is decomposed into three matrices that are of lower dimensionality - U, ∑ and V. The rank of the matrix M is the largest number of rows or columns which are linearly independent with other rows or columns and is denoted by r. U is an orthogonal singular left matrix of dimensions mxr, contains users and its related latent factors. ∑ is an rxr diagonal matrix that describes latent factors strength or weights. V is an orthogonal singular right matrix of dimensions rxn, consisting of movies and its related features. U decomposed matrix tells us how much a user liked each feature while V tells how much a movie is related to that feature. The M matrix is to be approximated using the decomposed matrices which keeps top k most important features out of the total r features.

This approximated matrix is found which is an optimization problem where the root mean squared error between the actual ratings and predicted ratings is minimized using stochastic gradient descent.

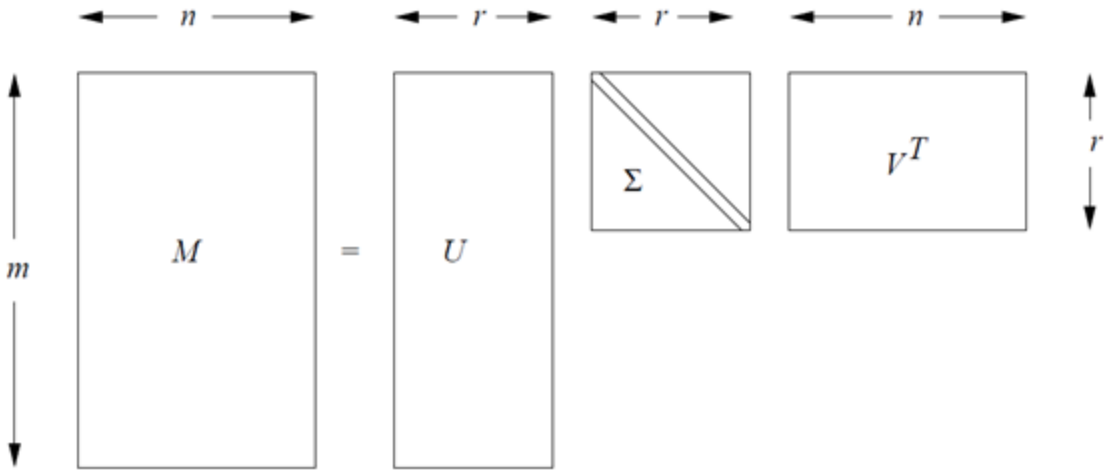

Figure 4: SVD Approach

Hybrid LightFM Model:

LightFM model is a hybrid matrix factorization model which represents the user and items as a linear combination of their content features' latent factors and solves the cold start and scarcity problems.

LightFM states that for the given user-item-rating matrix, let U be the user set, I be the item set, features related to users as F^U, and features related to items as F^I. The interactions whether positive or negative are stored in the user-interaction set $(u,i) \in UxI$ as S^+ , S^-. Each user in the set U is described using certain subset of features $f_u \subset F^U$ and each item in I is described using a subset of item features, $f_i \subset F^I$. The models' item and user features are translated to a latent dimension space and learns the embeddings respectively, e^I and e^U for all features. A bias term is associated with both item and user features, b^U and b^I. The latent representation of the user u and item i is equal to the sum of user features' latent factors and sum of item features latent factors respectively.

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U$$

$$\boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

The bias term for user and item is calculated as the sum of features biases respectively.

$$b_u = \sum_{j \in f_u} b_j^U$$

$$b_i = \sum_{j \in f_i} b_j^I$$

The model prediction for user u and item i is equal to sum of the dot product of user and item representations, user and feature biases.

$$\widehat{r}_{ui} = f\left(\boldsymbol{q}_u \cdot \boldsymbol{p}_i + b_u + b_i\right)$$

The function, f, is taken as sigmoid which gives the prediction in the range of 0 and 1.

$$f(x) = \frac{1}{1 + \exp(-x)}$$

The approximate predictions are achieved by maximizing the likelihood of data with respect to the parameters.

$$L\left(\boldsymbol{e}^U, \boldsymbol{e}^I, \boldsymbol{b}^U, \boldsymbol{b}^I\right) = \prod_{(u,i) \in S^+} \widehat{r}_{ui} \times \prod_{(u,i) \in S^-} (1 - \widehat{r}_{ui})$$

V.B.II. Language Used

For developing and implementing our models, python programming language is used.

V.B.III. Tools Used

For data preprocessing, standard libraries pandas and numpy are used. Matplotlib library is used for visualization and plotting graphs. Scikit learn, surprise library and lightFM are used for building our baseline and proposed hybrid models. Open source web application Jupyter notebook and Kaggle are used for creating our models using python programming language.

V.C. How to generate output

The dataset is split into two parts - training and testing. With training data, the proposed models are developed and tested with test dataset which provides the list of recommendations and the estimated ratings for each user. We also find the evaluation metrics like RMSE, precision @ k, recall @ k and AUC score, wherever possible for the models, to compare them.

V.D. How to test against hypothesis

The main aim of this project is to develop two hybrid models – Hybrid SVD Content based model and lightFM model that overcomes the issues with traditional approaches. We would test our proposed models and evaluate them against AUC score. Upon comparing these metrics in standalone traditional and hybrid models, it shows that the hybrid models outperform the standalone models and provide more quality recommendations tailored to each individual.

VI. Implementation

VI.A. Code (refer programming requirements)

Refer appendix and submit file for code

VI.B. Design document and flowchart

The generic approach involved in the two proposed models is shown in Figure 5.

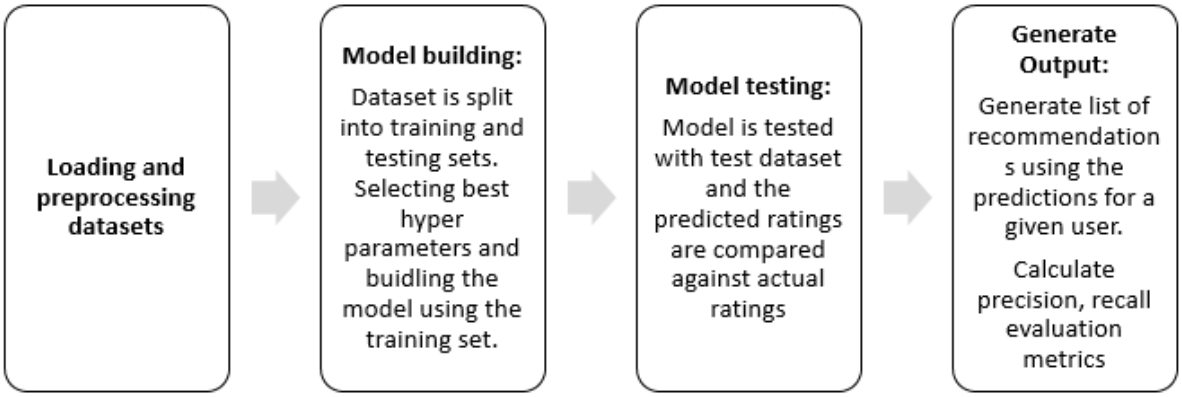

Figure 5: Generic approach for building the models

Hybrid content-based filtering and Singular Value Decomposition model:

The first proposed model implemented is a combination of two standalone models – content-based filtering model and SVD model. We have used surprise and sklearn libraries for building the models. Figure 6 shows the brief steps of the model.

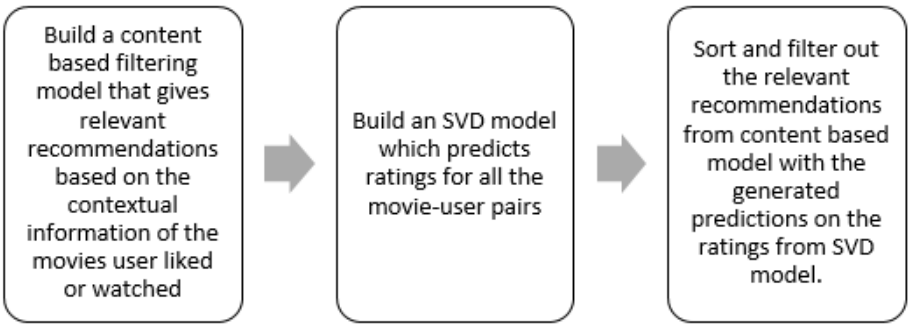

Figure 6: Hybrid SVD Content-Based Model Steps

Detailed steps of this model is as follows:

1. **Content-based model:**
    a. MovieLens and IMDB dataset is loaded from CSV files
        i. Specifically, load the movies, links, and IMDb movie data sets
    b. Join the three data sets so that you have the complete set of movie data in one format
    c. Extract the features of a movie that you will base similarity scores off of
        i. Specifically, in this implementation, extract the genre, language, director, writer, production company, and actors fields and combine them into a single token separated by a comma
    d. Use TF.IDF Vectorizer function provided by sklearn to generate a matrix of movies vs features. In this matrix, the rows correspond to a specific movie and the columns correspond to the feature that this movie may or may not have. If a movie does not have a specific feature (ie. The movie is not an action movie), then it will have a 0 for that feature
    e. Use the linear_kernel function provided by sklearn to then transform the matrix above into a similarity matrix, based on the cosine distance. The

resulting matrix is an n by n matrix, where the rows and columns correspond to movies and the value at (n,n) is the cosine similarity between those two movies, based on the features that they share

f. To generate a list of recommendations for a given user, the algorithm then performs the following steps
    i. Retrieve all the movies that this user has viewed
    ii. For every movie, use the matrix from step 5 to generate a list of similar movies
    iii. Sort the list of similar movies from largest to smallest based on the similarity score
    iv. Select the top n movies to recommend to the movie

2. **SVD model**:
    a. Movielens dataset is loaded and preprocessed by removing unwanted columns like timestamps.
    b. Convert the dataset into a matrix that contains users as the rows and movies as the columns with ratings as the data.
    c. Randomly initialize the user latent feature matrix and item latent feature matrix according to a normal distribution
    d. Split the dataset into train and test sets in 0.75-0.25 ratio with a random seed to replicate the same results over multiple function calls
    e. Assign some set of values for hyperparameters like the number of most significant features, number of epochs, learning rate, and regularization parameters.

f. Run the grid search to find the best hyperparameters.
g. Develop and train the model with the best hyperparameters until the loss is minimized or the number of epochs is reached.
    i. Perform stochastic gradient descent to minimize the loss and update the user and item features accordingly.
    ii. Calculate the root mean squared error between the actual ratings and predicted ratings in the utility matrix
h. Test the model for predicting the ratings and generate a list of recommendations for each user based on the estimated ratings

3. The recommendation list from the content-based model is passed to the SVD model to get the predicted ratings
4. Sorting and filtering are performed on the list to get more quality and relevant recommendations for each user.
5. Calculate precision and recall metrics for top n recommendations where the range of n is 2 to 20.

SVD++ Model: This is one of the baseline models which is an extension of the SVD model that uses implicit ratings. We have used the surprise library for building the model, generate a list of recommendations to users, and to calculate precision and recall metrics. The parameters used for this model are similar to the proposed models to have an unbiased comparison with the hybrid model.

KNN Model: KNN Model is another baseline traditional collaborative model that is developed for comparing the model with proposed models. KNN Model is built using sklearn and surprise libraries. The steps involved in this model are as follows:

1. Load the MovieLens rating file
2. Split the data into a training set and a testing set
3. Create a KNN model that is provided by the surprise library
4. Fit the model with the rating data
5. Generate predictions using the fitted model
6. Run evaluation functions to gather performance metrics

LightFM Model:

This model is a combination of content-based filtering and collaborative filtering baseline models. It includes implementation of BPR(Bayesian Personalised Ranking pairwise loss) and WARP(Weighted Approximate-Rank Pairwise loss) loss functions. We used WARP as Maximizes the rank of positive examples by repeatedly sampling negative examples until rank violating one is found, which is best suited for the current context of recommendation systems.

This model also follows the generic approach as shown in the figure of the flowchart at the beginning of this section.

1. The data fetching and preprocessing is done using the fetch_movielens library provided by the lightfm package, by also mentioning the threshold for ratings.
2. The data is then split into train and test sparse matrices, also while preprocessing, along with the item feature matrix.

3. The train data is fit into the lightfm model, along with some hyperparameters like loss, number of epochs, learning rate, number of samples, number of threads as required.
4. A recommender function is defined which takes the training data and tests it against that data. The user ratings above the threshold mentioned while fetching the data are considered as known positives, which is used to display the recommendation for movies as well as the known positives as a table.
5. Further, some more evaluation metrics functions are defined in the lightfm model by varying all the hyperparameters among a range of data by trying out random combinations. The evaluation metrics for all the combinations, including the best is displayed.

VII. Data Analysis and Discussion

VII.A. Output Generation

For all the proposed and baseline models, the output is the list of top 10 movie recommendations for each user. To evaluate all the models, we are using AUC score. For lightFM, AUC score is the best metric used for comparison. For other models, we found RMSE, precision @5, precision @10, recall @5, recall@10 metrics to compare and evaluate the models. In information retrieval, these are some of the widely used evaluation metrics for recommender systems. For a user, the metric precision @ k is equal to the fraction of retrieved recommendations that are relevant to the query.

$$\text{Precision @ k} = \frac{|\{\text{Recommended items that are relevant}\}|}{|\{\text{Recommended items}\}|}$$

Recall @ k is the fraction of the relevant documents that are successfully retrieved.

$$\text{Recall @ k} = \frac{|\{\text{Recommended items that are relevant}\}|}{|\{\text{Relevant items}\}|}$$

In the recommended items, a movie is relevant if its true rating is greater than threshold and a movie is recommended if the predicted rating is greater than threshold and is among the top k estimated ratings.

VII.B. Output Analysis

VII.B.I. Hybrid genre based content filtering and Singular Value Decomposition model:

For the first proposed hybrid content-based and SVD model, we have used grid search for choosing the best hyper parameters and their values for the model. Table 1 shows the different hyper parameters and their values. The model is trained with the best values highlighted in bold.

Table 1: Parameters used in Gridsearch for the hybrid SVD Content-based model

Hyper Parameter	**Value**
Number of features	10,20,25,30,35,40,50,**100**
Number of epochs	10,20,30,40,**50**
Learning rate	0.9,0.09,0.009,0.1,**0.01**,0.001,0
Regularization parameter	0.9,0.09,0.009,**0.1**,0.01,0.001,0

We compared each hyper parameter with RMSE to see how the presence or absence of hyper parameter and its value affects the RMSE. Figure 7 shows how the RMSE varies with respect to the number of features for test data. The root mean square error between the estimated and actual ratings did not affect much with respect to different number of factors. When the number of factors is 100, the RMSE is 0.852, is almost flat

and did not change much as the factors keep on increasing. Figure 8 shows how the

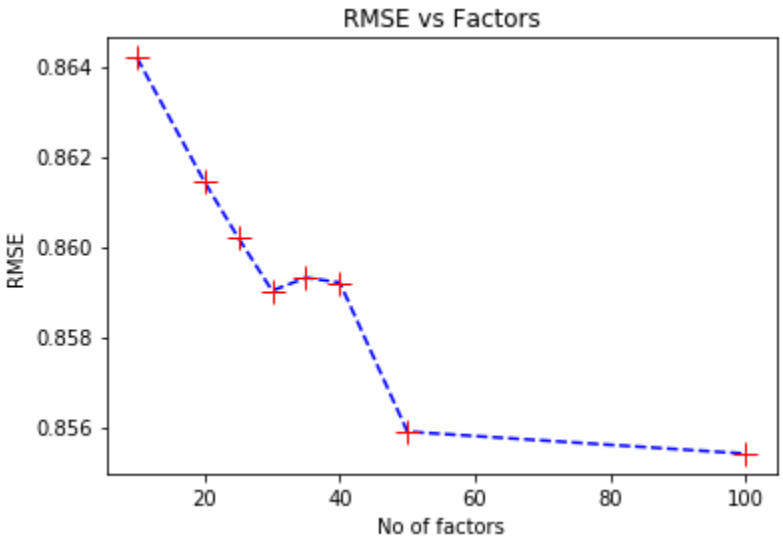

Figure 7: Varying RMSE with increase of factors

RMSE varies in multiple epochs.

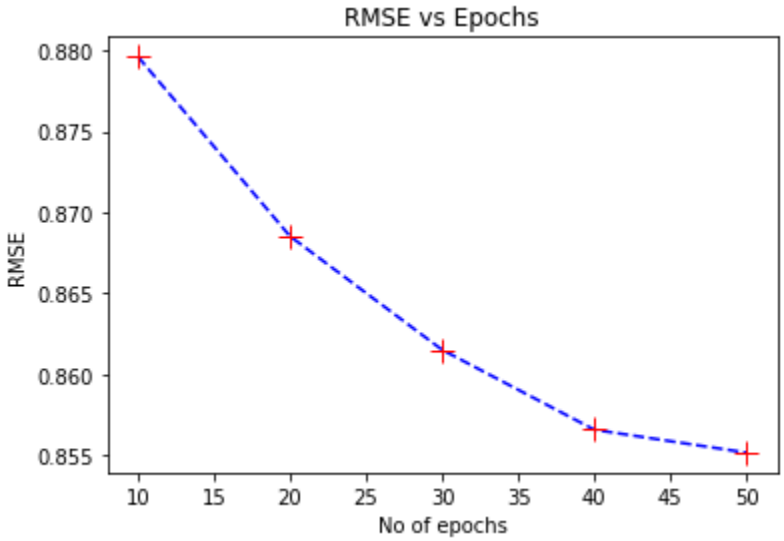

Figure 8: RMSE Variation with increasing epochs

With respect to epochs, the RMSE did not change much similar to the factors. As the number of epochs keep on increasing, RMSE almost went flat and did not have much difference. Figure 9 shows the effect of learning rate with RMSE. The graph shows that without any learning rate, the RMSE is pretty high and the presence of learning rate made

a difference. However, as the learning rate increased, the RMSE also started increasing. At learning rate 0.01, the RMSE is minimum which is used for training the model.

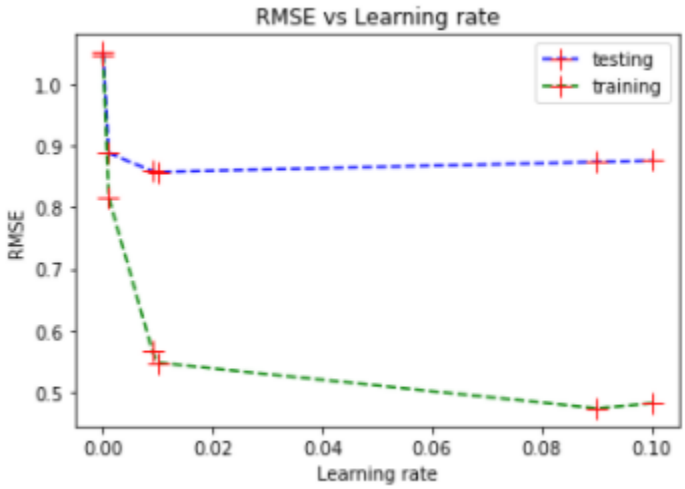

Figure 9: Effect of learning rate on RMSE

The effect of the presence of regularization is seen in the figure 10. For training data, the data is overfitting without regularization and RMSE is increasing as the parameter value keeps on increasing. For testing dataset, the RMSE starts decreasing and becomes flat at regularization parameter 0.10.

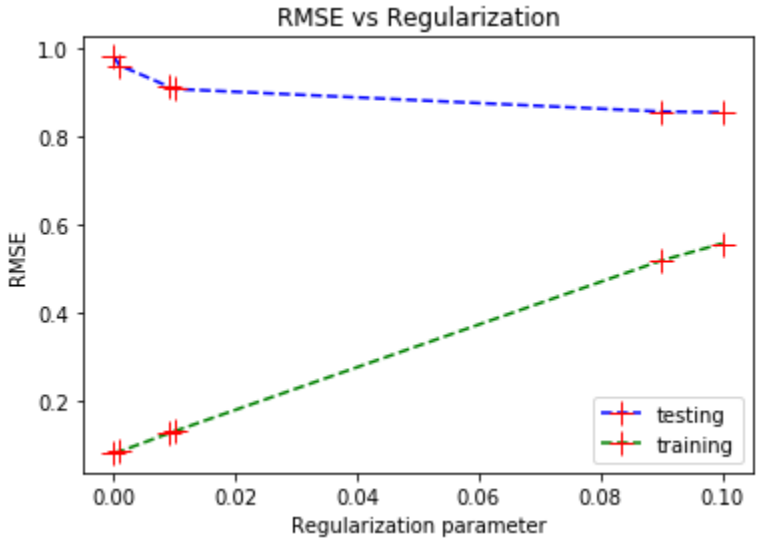

Figure 10: Effect of regularization

The evaluation of the model using precision @ K and recall @ K starting from top 5 to 10 recommendations is shown in figure 11. The model achieves a precision of 0.91 and a recall of 0.56 for the top 10 recommendations.

```
{'topN': 5, 'Precision': 0.9266666666666675, 'Recall': 0.3751735766282603}
{'topN': 6, 'Precision': 0.9216939890710381, 'Recall': 0.4245747204438231}
{'topN': 7, 'Precision': 0.9191959406713495, 'Recall': 0.46854289918835484}
{'topN': 8, 'Precision': 0.9185519125683059, 'Recall': 0.5043262358985325}
{'topN': 9, 'Precision': 0.9168214936247734, 'Recall': 0.535795785503344}
{'topN': 10, 'Precision': 0.9151092896174856, 'Recall': 0.5616277574566677}
```

Figure 11: Precision@k and Recall@k scores for hybrid SVD content model

The content-based model retrieves a list of recommendations based on cosine similarity for all users. The final output of the model is the list of relevant recommendations from content based model with the estimated ratings coming from the SVD model which is sorted in descending order.

	userId	movieId	rating	title	genres
3591	60	527	5.0	Schindler's List (1993)	Drama|War
17079	60	58559	5.0	Dark Knight, The (2008)	Action|Crime|Drama|IMAX
45373	60	858	5.0	Godfather, The (1972)	Crime|Drama
2763	60	362	4.0	Jungle Book, The (1994)	Adventure|Children|Romance
15385	60	3386	4.0	JFK (1991)	Drama|Mystery|Thriller
16328	60	318	4.0	Shawshank Redemption, The (1994)	Crime|Drama
22569	60	2150	4.0	Gods Must Be Crazy, The (1980)	Adventure|Comedy
33315	60	783	4.0	Hunchback of Notre Dame, The (1996)	Animation|Children|Drama|Musical|Romance
51022	60	6016	4.0	City of God (Cidade de Deus) (2002)	Action|Adventure|Crime|Drama|Thriller
63673	60	3424	4.0	Do the Right Thing (1989)	Drama

Figure 12: Top 10 highly rated movies by user 60

Figure 12 shows a list of top 10 movies that are highly rated and liked by user 60. We can depict from the list that the user generally enjoys movies that are related to Drama, Action, Crime genres. The output of the hybrid model is shown in figure 13, the movie recommendations for a user 60, and the likely ratings user would provide. We can see

that the recommendations are mostly from the Action, Crime, and Drama genre and the model is producing more quality and relevant recommendations.

	userID	movieId	title	genre	predicted rating
15	60	85342	Elite Squad: The Enemy Within (Tropa de Elite ...	Action, Crime, Drama	4.312821
44	60	457	Fugitive, The (1993)	Action, Crime, Drama	4.144130
4	60	33794	Batman Begins (2005)	Action, Adventure	4.054929
71	60	2583	Cookie's Fortune (1999)	Comedy, Drama	4.025501
109	60	62293	Duchess, The (2008)	Biography, Drama, History	4.023580
111	60	2761	Iron Giant, The (1999)	Animation, Action, Adventure	4.008371
9	60	109487	Interstellar (2014)	Adventure, Drama, Sci-Fi	4.006568
41	60	52867	Ex, The (2007)	Comedy, Romance	3.992800
5	60	3147	Green Mile, The (1999)	Crime, Drama, Fantasy	3.963844
149	60	63033	Blindness (2008)	Drama, Mystery, Sci-Fi	3.956392
82	60	93022	Miss Nobody (2010)	Comedy, Crime	3.951273

Figure 13: Top 10 recommendations for user 60 by SVD content based model

VII.B.II. SVD++ Model:

The model is developed using learning rate and regularization parameters to avoid overfitting. For test dataset, the RMSE between actual and estimated ratings achieved is 0.90. The model is evaluated using precision @ 5, precision @ 10, recall @5 and recall @10 metrics. We have achieved a precision of 0.90 and recall of 0.56 for top 10 recommendations. The output of the model is shown in figure 14 for user 10

	user	movieId	predicted_rating	title	genres
4228	10	2571	4.079510	Matrix, The (1999)	Action|Sci-Fi|Thriller
15727	10	4995	3.978176	Beautiful Mind, A (2001)	Drama|Romance
1656	10	356	3.950103	Forrest Gump (1994)	Comedy|Drama|Romance|War
2741	10	104374	3.949403	About Time (2013)	Drama|Fantasy|Romance
19986	10	81847	3.920703	Tangled (2010)	Animation|Children|Comedy|Fantasy|Musical|Roma...
2084	10	68954	3.910650	Up (2009)	Adventure|Animation|Children|Drama
8053	10	6377	3.899599	Finding Nemo (2003)	Adventure|Animation|Children|Comedy
19967	10	70183	3.765376	Ugly Truth, The (2009)	Comedy|Drama|Romance
9710	10	7293	3.743389	50 First Dates (2004)	Comedy|Romance
6359	10	5952	3.734409	Lord of the Rings: The Two Towers, The (2002)	Adventure|Fantasy

Figure 14: Top 10 recommendations for user 10 by SVD++ model

Figure 15 shows the precision and recall metrics starting from top 5 to top 10 recommendations.

```
{'topN': 5, 'Precision': 0.9102459016393452, 'Recall': 0.3756430909688781}
{'topN': 6, 'Precision': 0.9116120218579233, 'Recall': 0.4283813164584985}
{'topN': 7, 'Precision': 0.9100897736143627, 'Recall': 0.4732680894681844}
{'topN': 8, 'Precision': 0.9089773614363774, 'Recall': 0.5099375754925136}
{'topN': 9, 'Precision': 0.9074290918553217, 'Recall': 0.5423612056964938}
{'topN': 10, 'Precision': 0.905042935206869, 'Recall': 0.5688539873317746}
```

Figure 15: Precision@k and Recall@k scores SVD++ model

VII.B.III. KNN Model

Similar to the SVD++ model, this model is tuned with various parameters. It is also evaluated based on the same metrics. For instance, we've obtained RMSE score, precision@, and recall@ scores so that the model itself can be compared against other baseline models as well as the proposed hybrid models. Below is a brief analysis on the output.

Similar to SVD++, we show here the top 10 rated movies that user 10 rated. For this user, it seems like they enjoy Comedy, Action, and Adventure movies.

	movieId	title	genres
4948	7458	Troy (2004)	Action|Adventure|Drama|War
5227	8533	Notebook, The (2004)	Drama|Romance
5332	8869	First Daughter (2004)	Comedy|Romance
5917	33794	Batman Begins (2005)	Action|Crime|IMAX
7156	71579	Education, An (2009)	Drama|Romance
7371	79091	Despicable Me (2010)	Animation|Children|Comedy|Crime
7466	81845	King's Speech, The (2010)	Drama
7768	91529	Dark Knight Rises, The (2012)	Action|Adventure|Crime|IMAX
7955	96079	Skyfall (2012)	Action|Adventure|Thriller|IMAX
9006	140110	The Intern (2015)	Comedy

Figure 16: Top 10 highly liked movies by user 10

Now, if we use the KNN model to suggest the top 10 movies that this user may like, it will present the following list. The recommended movies seem to be very much aligned with what the user enjoys. It returns many Comedy, Action, and Adventure movies. How does this model actually perform?

	movieId	title	genres
506	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical
514	597	Pretty Woman (1990)	Comedy|Romance
694	912	Casablanca (1942)	Drama|Romance
946	1247	Graduate, The (1967)	Comedy|Drama|Romance
3157	4246	Bridget Jones's Diary (2001)	Comedy|Drama|Romance
3287	4447	Legally Blonde (2001)	Comedy|Romance
3640	4995	Beautiful Mind, A (2001)	Drama|Romance
4137	5952	Lord of the Rings: The Two Towers, The (2002)	Adventure|Fantasy
4644	6942	Love Actually (2003)	Comedy|Drama|Romance
4799	7151	Girl with a Pearl Earring (2003)	Drama|Romance

Figure 17: Top 10 recommendations for user 10 by KNN model

Coming to the KNN model evaluation, it has an RMSE score of 0.985, which is a bit higher than what the other models have shown. When we look at its precision@k and recall@k scores, we see that it performs reasonably. The precision and recall are not as good but decent when compared to other models.


```
[{'topN': 2, 'Precision': 0.9377049180327869, 'Recall': 0.14798911497140263},
 {'topN': 3, 'Precision': 0.9229508196721311, 'Recall': 0.212382472826808},
 {'topN': 4, 'Precision': 0.9174863387978142, 'Recall': 0.2756918521553819},
 {'topN': 5, 'Precision': 0.917732240437159, 'Recall': 0.3350062804028265},
 {'topN': 6, 'Precision': 0.912486338797814, 'Recall': 0.3851047082765852},
 {'topN': 7, 'Precision': 0.9069437939110057, 'Recall': 0.42978142646920114},
 {'topN': 8, 'Precision': 0.9051288056206088, 'Recall': 0.4693828466156005},
 {'topN': 9, 'Precision': 0.9023510278428328, 'Recall': 0.5039661088855079},
 {'topN': 10, 'Precision': 0.9006206088992961, 'Recall': 0.5335213639205003},
 {'topN': 11, 'Precision': 0.9000989993612958, 'Recall': 0.55893545466625},
 {'topN': 12, 'Precision': 0.899937548790008, 'Recall': 0.5826123028545311},
 {'topN': 13, 'Precision': 0.8981300666546567, 'Recall': 0.6025723699086144},
 {'topN': 14, 'Precision': 0.896013330931363, 'Recall': 0.6207243668580433},
 {'topN': 15, 'Precision': 0.8956698492764072, 'Recall': 0.6383959535479563},
 {'topN': 16, 'Precision': 0.893934876598811, 'Recall': 0.6536265427584277},
 {'topN': 17, 'Precision': 0.8925004503693013, 'Recall': 0.6670318922052259},
 {'topN': 18, 'Precision': 0.8919486449476819, 'Recall': 0.6797862264032566},
 {'topN': 19, 'Precision': 0.8909180630284023, 'Recall': 0.6912772119795995}]
```


Figure 18: Precision@k and Recall@k scores for KNN model

The KNN Model performs decently. However, it can already be seen that other models can perform much better.

VII.B.IV. LightFM Model

LightFm library is developed based on the sklearn library, with a scope of creating interaction matrices, evaluation metrics and many more. Also contains a large set of datasets related to the movie rating, called ‘Movielens’ dataset.

It fetches input from movielens, and divides into train and test sparse matrices.There are fitted into the lightfm model, tweaking hyperparameters. The model is further used to predict the recommendations based on the known likes of the user. Output of Known positives are shown on the right side in Figure 19 shows the known likes or (known positives) of the user.

Left side image in Figure 19 shows the output dataframe of lightfm model for the recommended movies based on those likes.

Evaluation of LightFM

We calculated Precision@k, Recall@k metrics provided by lightfm library and AUC score evaluation metrics provided by the sklearn.metrics is applied on the lightfm model.

We used AUC score to compare it with other models as it is the recommended metric for evaluation.We see that the model performed better than the other models with respect to AUC score.

Values : Train set = 0.94, Test_set = 0.91

Top-10 recommendations are taken into account to compute true positive rate and false positive rate?

All the ratings treated as binary for the AUC score in LightFM.

	User	Top Movies Recommended
0	2	Titanic (1997)
1	2	L.A. Confidential (1997)
2	2	Air Force One (1997)
3	2	Contact (1997)
4	2	Cop Land (1997)
5	2	Devil's Advocate, The (1997)
6	2	English Patient, The (1996)
7	2	G.I. Jane (1997)
8	2	Seven Years in Tibet (1997)
9	2	Good Will Hunting (1997)
10	10	Star Wars (1977)
11	10	Return of the Jedi (1983)
12	10	Raiders of the Lost Ark (1981)
13	10	Princess Bride, The (1987)
14	10	Monty Python and the Holy Grail (1974)
15	10	Back to the Future (1985)
16	10	Empire Strikes Back, The (1980)
17	10	Fargo (1996)
18	10	Indiana Jones and the Last Crusade (1989)
19	10	Toy Story (1995)
20	60	Contact (1997)
21	60	L.A. Confidential (1997)
22	60	English Patient, The (1996)
23	60	Air Force One (1997)
24	60	Titanic (1997)
25	60	Full Monty, The (1997)
26	60	Scream (1996)
27	60	Good Will Hunting (1997)
28	60	Conspiracy Theory (1997)
29	60	Game, The (1997)

	User	Known Positives
0	2	Return of the Jedi (1983)
1	2	Event Horizon (1997)
2	2	Chasing Amy (1997)
3	2	Starship Troopers (1997)
4	2	Hoodlum (1997)
5	2	Ulee's Gold (1997)
6	2	Devil's Advocate, The (1997)
7	2	Schindler's List (1993)
8	2	Paradise Lost: The Child Murders at Robin Hood...
9	2	Mother (1996)
10	10	Babe (1995)
11	10	Dead Man Walking (1995)
12	10	Mr. Holland's Opus (1995)
13	10	Braveheart (1995)
14	10	Rumble in the Bronx (1995)
15	10	Birdcage, The (1996)
16	10	Apollo 13 (1995)
17	10	Batman Forever (1995)
18	10	Strange Days (1995)
19	10	To Wong Foo, Thanks for Everything! Julie Newm...
20	60	Contact (1997)
21	60	Full Monty, The (1997)
22	60	Air Force One (1997)
23	60	Rainmaker, The (1997)
24	60	Game, The (1997)
25	60	Tomorrow Never Dies (1997)

Figure 19: Top 10 recommendations for the users; known positives - the movies liked by the users

VII.C. Compare Output Against Hypothesis

The hypothesis of our project is to develop two hybrid models – SVD Content based model and LightFM model which outperform the traditional baseline approaches. All the

models are compared against AUC Score since this is the recommended metric for comparison with respect to LightFM model. The other three models are compared with three metrics – Precision @5, Recall @5, and AUC Score evaluation metrics. The Table 2 shows comparison of all models with AUC score while table 3 shows all the metric scores calculated for models except LightFM.

Table 2: Evaluating models with AUC Score

Method	Type of model	AUC Score
KNN	Baseline	0.68
SVD++	Baseline	0.73
Hybrid SVD Content-Based	**Hybrid**	0.77
LightFM	**Hybrid**	**0.92**

On seeing the evaluation metrics with respect to all models, in a nutshell, we can conclude that the proposed models perform better than the baseline models. The best values achieved are highlighted in bold. When we compare the hybrid SVD Content based model with the baselines, the hybrid model performs better with respect to precision and recall which retrieves most of the relevant recommendations. Although the recall metric is similar to SVD++ model, the precision and AUC score is higher for the proposed model.

Table 3: Metric scores for models except lightFM

Method	Type of model	Precision@5	Precision@10	Recall @5	Recall @10	AUC Score
KNN	Baseline	0.91	0.9	0.33	0.53	0.68
SVD++	Baseline	0.91	0.9	0.37	0.56	0.73
Hybrid SVD Content-Based	**Hybrid**	**0.92**	**0.91**	**0.37**	**0.56**	**0.77**

VIII. Conclusions and Recommendations

VIII.A. Summary and Conclusions

In this study, we have come to the conclusion that hybrid recommender systems outperform their more basic content based and collaborative filtering counterparts. How did we come to this conclusion? We first did research on existing technologies out there. Then, we figured out what were the base models (content based and collaborative filtering models) and the enhancements that have been made to these models since. From there, we mixed and mashed several different approaches, including content based and SVD, latent factorization through LightFM, etc. By combining the different approaches, we were able to implement models that outperformed their basic counterparts.

VIII.B. Recommendations for Future Studies

The benefit of combining different approaches into one model was that the strength of one model covered the weakness of another model. For instance, cold start is a common issue for collaborative filtering models. As a reminder, cold start problem is where there is no historical data for the user or the movie and so the model has nothing to base its predictions off of. Although this is a limitation of the collaborative model, it is not one of the content-based recommender models. It is not just a cold start that is addressed through a hybrid approach. Other issues such as sparsity of data and also implicit features are issues that are tricky to handle in regular recommender systems. That is why we

incorporated SVD as well as LightFM which uses latent factorization to incorporate implicit features.

That said, there are always issues that can still be addressed. There are always limitations to the current state of recommender systems that can be researched and addressed. For instance, localization is one aspect of recommender systems that can be looked into more. It is possible that the tastes of individuals in one particular region or state can be drastically different than the tastes of individuals in another region. If the recommender systems are trained using all individual's data without considering region, the recommendations may not be as accurate as it can be. By creating tailored recommendation systems for specific geographies, it is possible to enhance the existing state of recommendation systems even further.

IX. Bibliography

[1]. Harper, F. Maxwell, and Joseph A. Konstan. "The movielens datasets: History and context." *Acm transactions on interactive intelligent systems (tiis)* 5, no. 4 (2015): 1-19.

[2]. Hug, Nicolas. "Surprise: A Python library for recommender systems." *Journal of Open Source Software* 5, no. 52 (2020): 2174.

[3]. Kula, Maciej. "Metadata embeddings for user and item cold-start recommendations." *arXiv preprint arXiv:1507.08439* (2015).

[4]. Guan, Xin, Chang-Tsun Li, and Yu Guan. "Matrix factorization with rating completion: An enhanced SVD model for collaborative filtering recommender systems." *IEEE access* 5 (2017): 27668-27678.

[5]. Kumar, Rajeev, B. K. Verma, and Shyam Sunder Rastogi. "Social popularity based SVD++ recommender system." *International Journal of Computer Applications* 87, no. 14 (2014).

[6]. Shani, Guy, and Asela Gunawardana. "Evaluating recommendation systems." In *Recommender systems handbook*, pp. 257-297. Springer, Boston, MA, 2011.

[7]. Rajaraman, Anand, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[8]. Pochetti, Francesco. “Bridging Recommender Systems and Dimensionality Reduction.” *Francesco Pochetti*, 2017, francescopochetti.com/bridging-recommender-systems-dimensionality-reduction/.

[9]. Schröder, Gunnar, Maik Thiele, and Wolfgang Lehner. "Setting goals and choosing metrics for recommender system evaluations." In *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*, vol. 23, p. 53. 2011.

Web References:

[10]. https://making.lyst.com/lightfm/docs/home.html

[11]. https://stackoverflow.com/questions/49896816/how-do-i-optimize-the-hyperparameters-of-lightfm

X. Appendices

X.A. Program source code with documentation

BaselineKNN.py

```
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import cross_validate
from surprise.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import pandas as pd
from collections import defaultdict
import numpy as np
from surprise import KNNBasic

df_ratings = pd.read_csv('input/ratings.csv')
df_movies = pd.read_csv('input/movies.csv')
df_ratings = df_ratings.drop(columns='timestamp')

# loading the data
train_split, test_split = train_test_split(df_ratings, test_size =
0.3, random_state=42)
reader = Reader(rating_scale=(1,5))
train_build = Dataset.load_from_df(train_split, reader)
test_build = Dataset.load_from_df(test_split, reader)
trainset = train_build.build_full_trainset()
testset = test_build.build_full_trainset().build_testset()

model = KNNBasic(k=50,min_k=20)
model.fit(trainset)
predictions = model.test(testset)
accuracy.rmse(predictions, verbose=True)

metrics=[]
true_pos_array = []
est_array = []
for rating_threshold in np.arange(0,5.5,0.5):
    truePositives = 0
    trueNegatives = 0
    falseNegatives = 0
    falsePositives = 0
    accuracy =0
    precision =0
```

```
    recall =0
    f1_score = 0
    for uid,_, true_r, est, _ in predictions:
        if(true_r >= rating_threshold and est >= rating_threshold):
            truePositives = truePositives + 1
            true_pos_array.append(true_r)
            est_array.append(est)
        elif(true_r >= rating_threshold and est <= rating_threshold):
            falseNegatives = falseNegatives + 1
        elif(true_r <= rating_threshold and est >= rating_threshold):
            falsePositives = falsePositives + 1
        elif(true_r <= rating_threshold and est <= rating_threshold):
            trueNegatives = trueNegatives + 1
        if(truePositives > 0):
            accuracy = (truePositives + trueNegatives ) /
(truePositives + trueNegatives + falsePositives + falseNegatives)
            precision = truePositives / (truePositives +
falsePositives)
            recall = truePositives / (truePositives + falseNegatives)
            f1_score = 2 * (precision * recall) / (precision + recall)

metrics.append([rating_threshold,truePositives,trueNegatives,falsePosi
tives,falseNegatives,accuracy,precision,recall,f1_score])
    metrics_df = pd.DataFrame(metrics)
    metrics_df.rename(columns={0:'rating_threshold',
1:'truePositives', 2: 'trueNegatives', 3: 'falsePositives',
4:'falseNegatives', 5: 'Accuracy', 6: 'Precision', 7:'Recall', 8:'F1
Score'},inplace=True)
true_bin_array =[]
for x in true_pos_array:
    if x >= rating_threshold:
        x = 1
    else:
        x = 0
    true_bin_array.append(x)
auc_score =
roc_auc_score(true_bin_array,est_array,multi_class='raise',average='ma
cro')
print('AUC Score: ',auc_score)

def get_precision_recall_at_n(predictions,topn,rating_threshold):
    all_actual_predicted_list = defaultdict(list)
    precision = dict()
    recall= dict()
    no_of_relevant_items = 0
    no_of_recommended_items_at_top_n = 0
    no_of_relevant_recommended_items_at_top_n = 0
    for uid, iid, true_r, est, _ in predictions:
        all_actual_predicted_list[uid].append((est, true_r))
    for uid, user_ratings in all_actual_predicted_list.items():
        user_ratings.sort(key=lambda x: x[0], reverse=True)
```

```
        no_of_relevant_items = sum((true_r >= rating_threshold) for
(_, true_r) in user_ratings)
        no_of_recommended_items_at_top_n = sum((est >=
rating_threshold) for (est, _) in user_ratings[:topn])
        no_of_relevant_recommended_items_at_top_n = sum(((true_r >=
rating_threshold) and (est >= rating_threshold)) for (est, true_r) in
user_ratings[:topn])

        precision[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_recommended_items_at_top_n if no_of_recommended_items_at_top_n
!= 0 else 1
        recall[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_relevant_items if no_of_relevant_items != 0 else 1

    return precision, recall

rating_threshold=3
precision_recall_at_n = []
all_precision = 0
all_recall = 0
for topn in range(2,20):
    precision, recall =
get_precision_recall_at_n(predictions,topn,rating_threshold)
    precision_at_n = sum(prec for prec in precision.values()) /
len(precision)
    recall_at_n = sum(rec for rec in recall.values()) / len(recall)
    precision_recall_at_n.append({'topN' : topn, 'Precision' :
precision_at_n, 'Recall': recall_at_n})
print(precision_recall_at_n)

userId = 10
# Display top rated movies
def get_top_n_rated_movies_for(userId, n=10):
    r = df_ratings[df_ratings['userId'] == userId]
    r_sorted = r.sort_values('rating', ascending=False)
    top_rated_movies = r_sorted.head(n)
    movie_info =
df_movies[df_movies['movieId'].isin(top_rated_movies['movieId'])]
    return movie_info

top_rated_movies = get_top_n_rated_movies_for(userId)
print(top_rated_movies.head(20))

# Display predictions

def get_top_n_recommended_movies_for(userId, n=10):
    movies = list(filter(lambda p: p[0] == userId, predictions))
    movies_sorted = sorted(movies, key=lambda p: p.est, reverse=True)
    movieIds = [p.iid for p in movies_sorted]
    return df_movies[df_movies['movieId'].isin(movieIds)][:n]
```

```
recommended_movies = get_top_n_recommended_movies_for(userId)
print(recommended_movies.head(20))
```

BaselineSVD++.py

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from surprise import SVDpp
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from collections import defaultdict
from surprise.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

#reading files
df_ratings = pd.read_csv('input/ratings.csv')
df_movies = pd.read_csv('input/movies.csv')
df_ratings = df_ratings.drop(columns= 'timestamp')
print(df_movies.head(5))
print(df_ratings.head(5))
#splitting data into train and test sets
train_split, test_split = train_test_split(df_ratings, test_size =
0.25, random_state = 20)
print("Training data size:", train_split.shape)
print("Test data size:", test_split.shape)
#reader for parsing the ratings file
reader = Reader(rating_scale=(1, 5))
#building the train and test set, loading the data from dataframe
train_build = Dataset.load_from_df(train_split, reader)
test_build = Dataset.load_from_df(test_split, reader)
trainset = train_build.build_full_trainset()
testset = test_build.build_full_trainset().build_testset()
print("Test set size:", len(testset))
#model building
#takes in factors, epochs, learning rate and regularization parameter
model = SVDpp(n_factors=20,n_epochs=5,lr_all=0.09,reg_all=0.5)
model.fit(trainset)
#making predictions
predictions = model.test(testset)
#calculating rmse
accuracy.rmse(predictions, verbose = True)
#Save all the predicted ratings and convert it to a dataframe
all_recommendations_list = defaultdict(list)
all_recommendations_df = pd.DataFrame([])
for uid, iid, true_r, est, _ in predictions:
    all_recommendations_list[uid].append((iid, est))
```

```
    all_recommendations_df =
all_recommendations_df.append(pd.DataFrame({'user': uid, 'movieId':
iid, 'predicted_rating' : est}, index=[0]), ignore_index=True);
print(all_recommendations_df.head(5))
print(all_recommendations_df.shape)
#Merging with movies file to get genre, title information for
predictions
all_recommendations_df_details =
pd.merge(all_recommendations_df,df_movies, on='movieId', how='inner')
print(all_recommendations_df_details.head(5))
#List of top n recommendations list as per SVD++
def get_top_n_recommendation_list_df(all_recommendations_df_details,
n=10):
    top_n_recommendations_df =
all_recommendations_df_details.sort_values(['user','predicted_rating']
,ascending=[True, False])
    return top_n_recommendations_df
top_n_recommendations_df =
get_top_n_recommendation_list_df(all_recommendations_df_details, 10)
print(top_n_recommendations_df.head())
metrics=[]
true_positives_array = []
est_array = []
for rating_threshold in np.arange(0,5.5,0.5):
    truePositives = 0
    trueNegatives = 0
    falseNegatives = 0
    falsePositives = 0
    accuracy =0
    precision =0
    recall =0
    f1_score = 0
    for uid,_, true_r, est, _ in predictions:
        if(true_r >= rating_threshold and est >= rating_threshold):
            truePositives = truePositives + 1
            true_positives_array.append(true_r)
            est_array.append(est)
            #here
        elif(true_r >= rating_threshold and est <= rating_threshold):
            falseNegatives = falseNegatives + 1
        elif(true_r <= rating_threshold and est >= rating_threshold):
            falsePositives = falsePositives + 1
        elif(true_r <= rating_threshold and est <= rating_threshold):
            trueNegatives = trueNegatives + 1
        if(truePositives > 0):
            accuracy = (truePositives + trueNegatives ) /
(truePositives + trueNegatives + falsePositives + falseNegatives)
            precision = truePositives / (truePositives +
falsePositives)
            recall = truePositives / (truePositives + falseNegatives)
            f1_score = 2 * (precision * recall) / (precision + recall)
```

```
metrics.append([rating_threshold,truePositives,trueNegatives,falsePosi
tives,falseNegatives,accuracy,precision,recall,f1_score])
    metrics_df = pd.DataFrame(metrics)
    metrics_df.rename(columns={0:'rating_threshold',
1:'truePositives', 2: 'trueNegatives', 3: 'falsePositives',
4:'falseNegatives', 5: 'Accuracy', 6: 'Precision', 7:'Recall', 8:'F1
Score'},inplace=True)
true_bin_array =[]
for x in true_positives_array:
    if x >= rating_threshold:
        x = 1
    else:
        x = 0
    true_bin_array.append(x)
auc_score =
roc_auc_score(true_bin_array,est_array,multi_class='raise',average='ma
cro')
print('AUC Score: ',auc_score)
#Calculate precision and recall at n
def get_precision_recall_at_n(predictions,topn,rating_threshold):
    all_actual_predicted_list = defaultdict(list)
    precision = dict()
    recall= dict()
    no_of_relevant_items = 0
    no_of_recommended_items_at_top_n = 0
    no_of_relevant_recommended_items_at_top_n = 0
    for uid, iid, true_r, est, _ in predictions:
        all_actual_predicted_list[uid].append((est, true_r))
    for uid, user_ratings in all_actual_predicted_list.items():
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        no_of_relevant_items = sum((true_r >= rating_threshold) for
(_, true_r) in user_ratings)
        no_of_recommended_items_at_top_n = sum((est >=
rating_threshold) for (est, _) in user_ratings[:topn])
        no_of_relevant_recommended_items_at_top_n = sum(((true_r >=
rating_threshold) and (est >= rating_threshold)) for (est, true_r) in
user_ratings[:topn])
        precision[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_recommended_items_at_top_n if no_of_recommended_items_at_top_n
!= 0 else 1
        recall[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_relevant_items if no_of_relevant_items != 0 else 1
    return precision, recall
rating_threshold=3
precision_recall_at_n = []
for topn in range(2,20):
    precision, recall =
get_precision_recall_at_n(predictions,topn,rating_threshold)
    precision_at_n = sum(prec for prec in precision.values()) /
len(precision)
```

```
    recall_at_n = sum(rec for rec in recall.values()) / len(recall)
    precision_recall_at_n.append({'topN' : topn, 'Precision' :
precision_at_n, 'Recall': recall_at_n})
for n in range(3,9):
    print(precision_recall_at_n[n])
#get user high rated and liked movies
all_movie_df_details = pd.merge(df_ratings,df_movies, on='movieId',
how='inner')
all_movie_df_details =
all_movie_df_details.sort_values(['userId','rating'],ascending=[True,
False])
print(all_movie_df_details.loc[all_movie_df_details['userId'] ==
10].head(10)) #user 10 top 10 rated movies
#user 10 top 10 movie recommendations list
print(top_n_recommendations_df.loc[top_n_recommendations_df['user'] ==
10].head(10))
```

ProposedHybridSVDContent.py

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from surprise import SVD
from surprise import Dataset
from surprise import Reader
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from surprise import accuracy
from collections import defaultdict
from surprise.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from surprise.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

#Content Based model
#Reading datasets
movies = pd.read_csv("input/movies.csv")
print(movies.head(5))
links = pd.read_csv("input/links.csv")
print(links.head(5))
# converting the imdbId field to be joinable later
links['imdbId2'] = links['imdbId'].map(str).apply(lambda s: "tt0" + s
if len(s) == 6 else "tt" + s)
print(links.head(5))
imdb_movies = pd.read_csv("input/IMDb movies.csv")
print(imdb_movies.head(5))
```

```
# joining data sets to obtain contextual information on movies (ie.
director, actor, production_company)
movies_links = pd.merge(movies, links, how='inner', on='movieId')
print(movies_links.head(5))
movies_complete =
pd.merge(movies_links,imdb_movies,how='inner',left_on='imdbId2',right_
on='imdb_title_id')

def generate_similarity_matrix():
    mc = movies_complete
    # filling in missing data
    mc['genre'].fillna("No genre")
    mc['language'].fillna("language")
    mc['director'].fillna("director")
    mc['writer'].fillna("writer")
    mc['production_company'].fillna("production_company")
    mc['actors'].fillna("actors")
    movies_complete["tokens"] = mc['genre'] + "," + mc['director'] +
mc['writer'] + "," + mc['actors'] + "," + mc['production_company']
    v = TfidfVectorizer(token_pattern = '[a-zA-Z0-9\s]+')
    tfidf_movies_context_matrix =
v.fit_transform(movies_complete['tokens'].values.astype('U'))
    cos_sim_matrix = linear_kernel(tfidf_movies_context_matrix,
tfidf_movies_context_matrix)
    return cos_sim_matrix
cos_sim_matrix = generate_similarity_matrix()
ratings = pd.read_csv("input/ratings.csv")
def get_movies_watched_by(userID):
    user_filter = ratings['userId'] == userID
    movies_watched = ratings[user_filter]
    return movies_watched
def get_movie_recommendations_for(userID):
    # get the list of movies that this user has watched
    movies_watched = get_movies_watched_by(userID)
    df_movies_watched = pd.DataFrame()
    for index, row in movies_watched.iterrows():
        i = movies_complete[movies_complete['movieId'] ==
row['movieId']].index
        df_movies_watched =
df_movies_watched.append(movies_complete.loc[i])

    # get similar items based on those movies
    similar_movies = []
    for index, row in df_movies_watched.iterrows():
        # generate top n similar items and add to the similar_movies
list
        sim_movies = list(enumerate(cos_sim_matrix[index]))
        sim_movies_sorted = sorted(sim_movies, key=lambda
movieid_score_tuple: movieid_score_tuple[1], reverse=True)
        similar_movies = similar_movies + sim_movies_sorted[1:11]
    # order the list from highest similarity to lowest similarity
```

```
    # recommend the top 10 movies
    similar_movies = sorted(similar_movies, key=lambda
movieid_score_tuple: movieid_score_tuple[1], reverse=True)
    top_10_recommendations = similar_movies
    # convert the list of movie indexes to movie id
    results_2 = list()
    for movie_score in top_10_recommendations:
        movie_index = movie_score[0]
        movie_score = movie_score[1]
        movieId = movies_complete.iloc[movie_index]['movieId']
        movieTitle = movies_complete.iloc[movie_index]['title_x']

        genre = movies_complete.iloc[movie_index]['genre']
        language = movies_complete.iloc[movie_index]['language']
        director = movies_complete.iloc[movie_index]['director']
        writer = movies_complete.iloc[movie_index]['writer']
        production_company =
movies_complete.iloc[movie_index]['production_company']
        actors = movies_complete.iloc[movie_index]['actors']
        t = (userID, movie_index, movie_score, movieId, movieTitle,
genre, language, director, writer, production_company, actors)
        results_2.append(t)
    df_results_2 = pd.DataFrame(results_2, columns=['userID',
'movie_index', 'score', 'movieId', 'title', 'genre', 'language',
'director', 'writer', 'production_company', 'actors'])
    # need to remove duplicates by only keeping the movie with the
highest score
    df_results_2 =
df_results_2[df_results_2.groupby(['movieId'],sort=False)['score'].tra
nsform(max) == df_results_2['score']]
    return df_results_2
results = get_movie_recommendations_for(60)
print(results.head(20))

#SVD Model
#reading files
df_ratings = pd.read_csv('input/ratings.csv')
df_movies = pd.read_csv('input/movies.csv')
df_ratings = df_ratings.drop(columns= 'timestamp')
print(df_movies.head(5))
print(df_ratings.head(5))

#splitting data into train and test sets
train_split, test_split = train_test_split(df_ratings, test_size =
0.25, random_state=20)
print("Training data size:", train_split.shape)
print("Test data size:", test_split.shape)
#reader to parse the ratings
reader = Reader(rating_scale=(1, 5))
#Train and test set
train_build = Dataset.load_from_df(train_split, reader)
```

```
test_build = Dataset.load_from_df(test_split, reader)
trainset = train_build.build_full_trainset()
testset = test_build.build_full_trainset().build_testset()
print("Test set size:", len(testset))

#Gridsearch to select best parameters
number_of_factors_list = [10,20,25,30,35,40,50,100]
number_of_epochs_list = [10,20,30,40,50]
learning_rate_list = [0.9,0.09,0.009,0.1,0.01,0.001]
regularization_parameter_list = [0.9,0.09,0.009,0.1,0.01,0.001]
hyper_parameters_set = { 'n_factors': number_of_factors_list,
'n_epochs': number_of_epochs_list, 'lr_all':
learning_rate_list,'reg_all': regularization_parameter_list}
trained_model = SVD
best_model_selection =
GridSearchCV(trained_model,hyper_parameters_set,measures=['rmse'],
cv=4)
best_model_selection.fit(train_build)
print("Best hyperparameters:
",best_model_selection.best_params['rmse'], "to achieve minimum RMSE:
" ,best_model_selection.best_score['rmse'])

#Factors vs RMSE
validationset = trainset.build_testset()
training_rmse = []
testing_rmse =[]
number_of_factors_list = [10,20,25,30,35,40,50,100]
for factor in number_of_factors_list:
    model = SVD(n_factors=factor,n_epochs=50,lr_all=0.01,reg_all=0.1)
    model.fit(trainset)
    training_predictions = model.test(validationset)
    training_rmse.append(accuracy.rmse(training_predictions))
    test_predictions = model.test(testset)
    testing_rmse.append(accuracy.rmse(test_predictions))
plt.figure(0)
plt.plot(number_of_factors_list,testing_rmse, 'b+--', markersize=12,
markeredgecolor='r',label='testing')
plt.plot(number_of_factors_list,training_rmse, 'g+--', markersize=12,
markeredgecolor='r',label='training')
plt.xlabel('No of factors')
plt.ylabel('RMSE')
plt.title("RMSE vs Factors")
plt.legend()

#Epochs vs RMSE
training_rmse = []
testing_rmse =[]
number_of_epochs_list = [10,20,30,40,50]
for epoch in number_of_epochs_list:
    model = SVD(n_factors=100,n_epochs=epoch,lr_all=0.01,reg_all=0.1)
    model.fit(trainset)
```

```
    training_predictions = model.test(validationset)
    training_rmse.append(accuracy.rmse(training_predictions))
    test_predictions = model.test(testset)
    testing_rmse.append(accuracy.rmse(test_predictions))
plt.figure(1)
plt.plot(number_of_epochs_list,testing_rmse, 'b+--', markersize=12,
markeredgecolor='r',label='testing')
plt.plot(number_of_epochs_list,training_rmse, 'g+--', markersize=12,
markeredgecolor='r',label='training')
plt.xlabel('No of epochs')
plt.ylabel('RMSE')
plt.title("RMSE vs Epochs")
plt.legend()

#Effect of learning rate with respect to RMSE
training_rmse = []
testing_rmse =[]
learning_rate_list = [0,0.001,0.009,0.01,0.09,0.1]
for lr in learning_rate_list:
    model = SVD(n_factors=100,n_epochs=50,lr_all=lr,reg_all=0.1)
    model.fit(trainset)
    training_predictions = model.test(validationset)
    training_rmse.append(accuracy.rmse(training_predictions))
    test_predictions = model.test(testset)
    testing_rmse.append(accuracy.rmse(test_predictions))
plt.figure(2)
plt.plot(learning_rate_list,testing_rmse, 'b+--', markersize=12,
markeredgecolor='r',label='testing')
plt.plot(learning_rate_list,training_rmse, 'g+--', markersize=12,
markeredgecolor='r',label='training')
plt.xlabel('Learning rate')
plt.ylabel('RMSE')
plt.title("RMSE vs Learning rate")
plt.legend()

#Regularization parameter and RMSE
training_rmse = []
testing_rmse =[]
regularization_parameter_list = [0,0.001,0.009,0.01,0.09,0.1]
for reg in regularization_parameter_list:
    model = SVD(n_factors=100,n_epochs=50,lr_all=0.01,reg_all=reg)
    model.fit(trainset)
    training_predictions = model.test(validationset)
    training_rmse.append(accuracy.rmse(training_predictions))
    test_predictions = model.test(testset)
    testing_rmse.append(accuracy.rmse(test_predictions))
plt.figure(3)
plt.plot(regularization_parameter_list,testing_rmse, 'b+--',
markersize=12, markeredgecolor='r',label='testing')
plt.plot(regularization_parameter_list,training_rmse, 'g+--',
markersize=12, markeredgecolor='r',label='training')
```

```
plt.xlabel('Regularization parameter')
plt.ylabel('RMSE')
plt.title("RMSE vs Regularization")
plt.legend()

#Building model using the best parameters from gridsearch
model = SVD(n_factors=100,n_epochs=50,lr_all=0.01,reg_all=0.1)
model.fit(trainset)
predictions = model.test(testset)
accuracy.rmse(predictions, verbose = True)

#Save all the predicted ratings and convert it to a dataframe
all_recommendations_list = defaultdict(list)
all_recommendations_df = pd.DataFrame([])
for uid, iid, true_r, est, _ in predictions:
    all_recommendations_list[uid].append((iid, est))
    all_recommendations_df =
all_recommendations_df.append(pd.DataFrame({'user': uid, 'movieId':
iid, 'predicted_rating' : est}, index=[0]), ignore_index=True);
print(all_recommendations_df.head(5))
print(all_recommendations_df.shape)

#Append movie info to the predictions
all_recommendations_df_details =
pd.merge(all_recommendations_df,df_movies, on='movieId', how='inner')
print(all_recommendations_df_details)
#top n recommendations list
def get_top_n_recommendation_list_df(all_recommendations_df_details,
n=10):
    top_n_recommendations_df =
all_recommendations_df_details.sort_values(['user','predicted_rating']
,ascending=[True, False])
    return top_n_recommendations_df
top_n_recommendations_df =
get_top_n_recommendation_list_df(all_recommendations_df_details, n=10)
print(top_n_recommendations_df.head())

#Hybrid model
def hybrid_model(userID):
    content_recommendations_list =
get_movie_recommendations_for(userID) #list of movies for that user
    content_recommendations_list=
content_recommendations_list[['userID','movieId', 'title', 'genre']]
    for key, columns in content_recommendations_list.iterrows():
        #key is the index of the dataframe, columns are movieid, title
and genre
        predict = model.predict(userID, columns["movieId"])
#predicting the rating based on svd model
        content_recommendations_list.loc[key, "predicted rating"] =
predict.est #adding a column svd rating and adding prediction value
```

```
    return content_recommendations_list.sort_values("predicted
rating", ascending=False).iloc[0:11] # return only first 10 movies
based on ratings

#calculate evaluation metrics
metrics=[]
true_positives_array = []
est_array = []
for rating_threshold in np.arange(0,5.5,0.5):
    truePositives = 0
    trueNegatives = 0
    falseNegatives = 0
    falsePositives = 0
    accuracy =0
    precision =0
    recall =0
    f1_score = 0
    for uid,_, true_r, est, _ in predictions:
        if(true_r >= rating_threshold and est >= rating_threshold):
            truePositives = truePositives + 1
            true_positives_array.append(true_r)
            est_array.append(est)
        elif(true_r >= rating_threshold and est <= rating_threshold):
            falseNegatives = falseNegatives + 1
        elif(true_r <= rating_threshold and est >= rating_threshold):
            falsePositives = falsePositives + 1
        elif(true_r <= rating_threshold and est <= rating_threshold):
            trueNegatives = trueNegatives + 1
        if(truePositives > 0):
            accuracy = (truePositives + trueNegatives ) /
(truePositives + trueNegatives + falsePositives + falseNegatives)
            precision = truePositives / (truePositives +
falsePositives)
            recall = truePositives / (truePositives + falseNegatives)
            f1_score = 2 * (precision * recall) / (precision + recall)

metrics.append([rating_threshold,truePositives,trueNegatives,falsePosi
tives,falseNegatives,accuracy,precision,recall,f1_score])
    metrics_df = pd.DataFrame(metrics)
    metrics_df.rename(columns={0:'rating_threshold',
1:'truePositives', 2: 'trueNegatives', 3: 'falsePositives',
4:'falseNegatives', 5: 'Accuracy', 6: 'Precision', 7:'Recall', 8:'F1
Score'},inplace=True)
true_bin_array =[]
for x in true_positives_array:
    if x >= rating_threshold:
        x = 1
    else:
        x = 0
    true_bin_array.append(x)
```

```
auc_score =
roc_auc_score(true_bin_array,est_array,multi_class='raise',average='ma
cro')
print('AUC Score: ',auc_score)

#calculate precision @ k and recall @ k
def get_precision_recall_at_n(predictions,topn,rating_threshold):
    all_actual_predicted_list = defaultdict(list)
    precision = dict()
    recall= dict()
    no_of_relevant_items = 0
    no_of_recommended_items_at_top_n = 0
    no_of_relevant_recommended_items_at_top_n = 0
    for uid, iid, true_r, est, _ in predictions:
        all_actual_predicted_list[uid].append((est, true_r))
    for uid, user_ratings in all_actual_predicted_list.items():
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        no_of_relevant_items = sum((true_r >= rating_threshold) for
(_, true_r) in user_ratings)
        no_of_recommended_items_at_top_n = sum((est >=
rating_threshold) for (est, _) in user_ratings[:topn])
        no_of_relevant_recommended_items_at_top_n = sum(((true_r >=
rating_threshold) and (est >= rating_threshold)) for (est, true_r) in
user_ratings[:topn])
        precision[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_recommended_items_at_top_n if no_of_recommended_items_at_top_n
!= 0 else 1
        recall[uid] = no_of_relevant_recommended_items_at_top_n /
no_of_relevant_items if no_of_relevant_items != 0 else 1
    return precision, recall
rating_threshold=3
precision_recall_at_n = []
for topn in range(2,20):
    precision, recall =
get_precision_recall_at_n(predictions,topn,rating_threshold)
    precision_at_n = sum(prec for prec in precision.values()) /
len(precision)
    recall_at_n = sum(rec for rec in recall.values()) / len(recall)
    precision_recall_at_n.append({'topN' : topn, 'Precision' :
precision_at_n, 'Recall': recall_at_n})
for n in range(3,9):
    print(precision_recall_at_n[n])

#get user liked and high rated movies
all_movie_df_details = pd.merge(df_ratings,df_movies, on='movieId',
how='inner')
all_movie_df_details =
all_movie_df_details.sort_values(['userId','rating'],ascending=[True,
False])
print(all_movie_df_details.loc[all_movie_df_details['userId'] ==
60].head(10))
```

```
#output of hybrid model which shows recommendations from user 60
print(hybrid_model(60))
```

ProposedLightFM.py

```
import numpy as np
import pandas as pd
from lightfm.datasets import fetch_movielens
from lightfm import LightFM
from lightfm.evaluation import precision_at_k
from lightfm.evaluation import recall_at_k
from lightfm.evaluation import auc_score
from lightfm.data import Dataset

from lightfm.cross_validation import random_train_test_split
import scipy.sparse as sp
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
import itertools

# movielens dataset with 100k movie ratings from 1k users on 1700 movi
es
data = fetch_movielens(min_rating=2.5)
print(data)

train_set = data['train']
test_set = data['test']

# create hybrid model, CB+CF
model = LightFM(learning_rate=0.05,loss='warp')

#train model
model.fit(data['train'],epochs=20,num_threads=2)


# Results

result=[]
known_values =[]
k = 10
userID_list = [2,10,60]
known_positives =[]
top_movies = []
def lightfm_recommender(model,data,user_ids):
     # of users and items usinf shape
    no_users,no_movies = data['train'].shape

#     generate recommendations for each user we input
    for user_id in user_ids:
#         movies already liked by user so far
        known_positives = data['item_labels'][data['train'].tocsr() [u
ser_id].indices]
#         movies our model predicts they will like
        scores = model.predict(user_id, np.arange(no_movies))
#      rank them in order of most liked to least
```

```
        top_movies = data['item_labels'][np.argsort(-scores)]
#       print out the results
        print("User %s" % user_id)
#         userID_list.append(user_id)
        print("     Known positives:")

        for x in known_positives[:k]:
            print("        %s" % x)
            known_values.append([user_id,x])

        print("     Recommended:")
        for x in top_movies[:k]:
            print("        %s" % x)
            result.append([user_id,x])
    return known_values,result

known_values, result = lightfm_recommender(model, data, userID_list)

known_values_df = pd.DataFrame(known_values)
known_values_df.rename(columns={0:'User',1:'Known Positives'},inplace=
True)
result_df = pd.DataFrame(result)
result_df.rename(columns={0:'User',1:'Top Movies Recommended'},inplace
=True)

result_df

known_values_df


# Evaluation
from lightfm.evaluation import precision_at_k
train_p_at_k = precision_at_k(model, data['train'], k=5).mean()
test_p_at_k = precision_at_k(model, data['test'], k=5).mean()
print("PRECISION@K: Train precision: %.4f" % train_p_at_k)
print("PRECISION@K: Test precision: %.4f" % test_p_at_k)

train_auc = auc_score(model, train_set).mean()
test_auc = auc_score(model, test_set).mean()
print('AUC: train %.4f, test %.4f.' % (train_auc, test_auc))

train_recall = recall_at_k(model, train_set).mean()
test_recall = recall_at_k(model, test_set).mean()
print('RECALL@K: train %.4f, test %.4f.' % (train_recall, test_recall)
)

def sample_hyperparameters():
    """
    Yield possible hyperparameter choices.
    """

    while True:
        yield {
            "no_components": np.random.randint(16, 64),
            "learning_schedule": np.random.choice(["adagrad", "adadelt
a"]),
            "loss": np.random.choice(["bpr", "warp", "warp-kos"]),
            "learning_rate": np.random.exponential(0.05),
```

```
            "item_alpha": np.random.exponential(1e-8),
            "user_alpha": np.random.exponential(1e-8),
            "max_sampled": np.random.randint(5, 15),
            "num_epochs": np.random.randint(5, 50),

        }


def random_search_auc(train, test, num_samples=10, num_threads=1,k=5):

    for hyperparams in itertools.islice(sample_hyperparameters(), num_
samples):
        num_epochs = hyperparams.pop("num_epochs")
        model = LightFM(**hyperparams)
        model.fit(train, epochs=num_epochs, num_threads=num_threads)
        score = auc_score(model, test, train_interactions=train, num_t
hreads=num_threads,check_intersections=False).mean()
        hyperparams["num_epochs"] = num_epochs
        yield (score, hyperparams, model)


if __name__ == "__main__":


    (score, hyperparams, model) = max(random_search_auc(train_set, tes
t_set, num_threads=2), key=lambda x: x[0])

    print("Best AUC score {} at {}".format(score, hyperparams))
```

#Precision@K:

```
def random_search_precision_at_k(train, test, num_samples=5, num_threa
ds=2,k=5):

    for hyperparams in itertools.islice(sample_hyperparameters(), num_
samples):
        num_epochs = hyperparams.pop("num_epochs")

        model = LightFM(**hyperparams)
        model.fit(train, epochs=num_epochs, num_threads=num_threads)

        p_at_k_score = precision_at_k(model, test_interactions=test, t
rain_interactions=train, num_threads=num_threads,k=k,check_intersectio
ns=False,preserve_rows=True).mean()

        hyperparams["num_epochs"] = num_epochs

        yield (p_at_k_score, hyperparams, model)


if __name__ == "__main__":


    (p_at_k_score, hyperparams, model) = max(random_search_precision_a
t_k(train_set, test_set, num_threads=2,k=5), key=lambda x: x[0])

    print("Best Precision@k score {} at {}".format(p_at_k_score, hyper
params))
```

#Recall@K

```
def random_search_recall_at_k(train, test, num_samples=10, num_threads
=1,k=5):


    for hyperparams in itertools.islice(sample_hyperparameters(), num_
samples):
        num_epochs = hyperparams.pop("num_epochs")

        model = LightFM(**hyperparams)
        model.fit(train, epochs=num_epochs, num_threads=num_threads)

        r_at_k_score = recall_at_k(model, test, train_interactions=tra
in, num_threads=num_threads, k=k).mean()

        hyperparams["num_epochs"] = num_epochs

        yield (r_at_k_score, hyperparams, model)


if __name__ == "__main__":


    (r_at_k_score, hyperparams, model) = max(random_search_recall_at_k
(train_set, test_set, num_threads=2), key=lambda x: x[0])

    print("Best Recall@k score {} at {}".format(r_at_k_score, hyperpar
ams))
```

X.B. Input/Output Listing

Input:

- Movielens data set

  Movies file:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Ratings file:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Links:

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

- IMBD data set

Movies file:

	imdb_title_id	title	original_title	year	date_published	genre	duration	country	language	director	...	actors	description	avg_vote	votes	budget	usa_gross_income	worlwide_gross_income	metascore	revi
0	tt0000574	The Story of the Kelly Gang	The Story of the Kelly Gang	1906	1906-12-26	Biography, Crime, Drama	70	Australia	NaN	Charles Tait	...	Elizabeth Tait, John Tait, Norman Campbell, Be...	True story of notorious Australian outlaw Ned ...	6.1	537	$ 2250	NaN	NaN	NaN	
1	tt0001892	Den sorte drøm	Den sorte drøm	1911	1911-08-19	Drama	53	Germany, Denmark	NaN	Urban Gad	...	Asta Nielsen, Valdemar Psilander, Gunnar Helse...	Two men of high rank are both wooing the beaut...	5.9	171	NaN	NaN	NaN	NaN	
2	tt0002101	Cleopatra	Cleopatra	1912	1912-11-13	Drama, History	100	USA	English	Charles L. Gaskill	...	Helen Gardner, Pearl Sindelar, Miss Fielding, ...	The fabled queen of Egypt's affair with Roman ...	5.2	420	$ 45000	NaN	NaN	NaN	
3	tt0002130	L'Inferno	L'Inferno	1911	1911-03-06	Adventure, Drama, Fantasy	68	Italy	Italian	Francesco Bertolini, Adolfo Padovan	...	Salvatore Papa, Arturo Pirovano, Giuseppe de L...	Loosely adapted from Dante's Divine Comedy and...	7.0	2019	NaN	NaN	NaN	NaN	
4	tt0002199	From the Manger to the Cross; or, Jesus of Naz...	From the Manger to the Cross; or, Jesus of Naz...	1912	1913	Biography, Drama	60	USA	English	Sidney Olcott	...	R. Henderson Bland, Percy Dyer, Gene Gauntier,...	An account of the life of Jesus Christ, based ...	5.7	438	NaN	NaN	NaN	NaN	