VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A PROJECT REPORT (15CSS86)
on

"A Concise Study on Personalized Recommender Systems"

Submitted in Partial fulfillment of the Requirements for the VIII Semester of the Degree of

Bachelor of Engineering In
Computer Science & Engineering

By

AJAY M (1CR16CS011)

ANISHA RAO (1CR16CS019)

ATHREYA UPPILI (1CR16CS030)

CHAITHRA K K (1CR16CS036)

Under the Guidance of

Mrs. Poonam Vijay Tijare
Assistant Professor,
Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY
#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,

BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled **"A Concise Study on Personalized Recommender Systems"** carried out by **Mr. Ajay M**, USN **1CR16CS011**, **Ms. Anisha Rao**, USN **1CR16CS019**, **Mr. Athreya Uppili**, USN **1CR16CS030, Ms. Chaithra K K**, USN **1CR16CS036,** bonafide students of CMR Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2019-2020. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Mrs. Poonam Vijay Tijare	**Dr. Prem Kumar Ramesh**	**Dr. Sanjay Jain**
Assistant Professor	**Professor & Head**	**Principal**
Dept. of CSE, CMRIT	**Dept. of CSE, CMRIT**	**CMRIT**

DECLARATION

We, the students of Computer Science and Engineering, CMR Institute of Technology, Bangalore declare that the work entitled "**A Concise Study on Personalized Recommender Systems**" has been successfully completed under the guidance of Prof. Poonam Vijay Tijare, Computer Science and Engineering Department, CMR Institute of technology, Bangalore. This dissertation work is submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in Computer Science and Engineering during the academic year 2019 - 2020. Further the matter embodied in the project report has not been submitted previously by anybody for the award of any degree or diploma to any university.

Place:

Date:

Team members:

AJAY M (1CR16SCS011) _____

ANISHA RAO (1CR16SCS019) _____

ATHREYA UPPILI (1CR16SCS030) _____

CHAITHRA K K (1CR16SCS036) _____

ABSTRACT

Recommender systems are tools for interacting with large and complex information spaces. They are a type of information gathering/analysis system that aims to predict the rating that a particular user will give to an item. The main goal of such systems is to efficiently recommend relevant items of interest to their audience for prolonged interaction with the product and maximizing user satisfaction. They are mainly used in commercial applications, some highly popular examples being Netflix, Amazon etc. Recommender systems research has incorporated a wide variety of artificial intelligence techniques such as machine learning, data mining, user modeling, case-based reasoning, constraint satisfaction, etc. This project aims to provide a descriptive overview of the different types of recommender systems, their uses and the algorithms present in the field. We attempt to implement some of the algorithms and document the errors results and the conclusions we have observed.

ACKNOWLEDGEMENT

We take this opportunity to express our sincere gratitude and respect to **CMR Institute of Technology, Bengaluru** for providing us a platform to pursue our studies and carry out our final year project.

We have great pleasure in expressing our deep sense of gratitude to **Dr. Sanjay Jain,** Principal, CMRIT, Bangalore, for his constant encouragement.

We would like to thank **Dr. Prem Kumar Ramesh,** Professor and Head, Department of Computer Science and Engineering, CMRIT, Bangalore, who has been a constant support and encouragement throughout the course of this project.

We consider it a privilege and honor to express our sincere gratitude to our guide **Mrs. Poonam Vijay Tijare,** Assistant Professor, Department of Computer Science and Engineering, for the valuable guidance throughout the tenure of this review.

We also extend our thanks to all the faculty of Computer Science and Engineering who directly or indirectly encouraged us.

Finally, we would like to thank our parents and friends for all their moral support they have given us during the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

People rely on technology in almost every aspect of their lives. With the overload of information over the recent years, recommender systems have grown to become an important part of people's everyday lives. Online platforms are an absolute essential in this digital age. Increasing the utility of recommendation systems on these platforms has increased user interaction and is also a cost effective method of the same. Consumers expect a personalized experience and sophisticated recommendation systems to find relevant products and content, all to save consumers time and money[1]. Recommendation technologies are widely used to help people identify relevant products or services or information. YouTube, Amazon, Netflix and many other such web services are the famously known recommendation systems. The system is capable of suggesting a set of items to the users and recommending top items to the user. People are always provided with too many options to choose from, the recommendation system focuses on the user's best interest and suggests the best options by learning from the users.

There are several algorithms that have been developed over time to help improve the efficiency of these systems. Systems mainly can be categorized into Content Based and Collaborative-Filtering based.

Content Based Filtering

Content based recommender systems aim mainly to recommend items based on their similarity metrics. For example, in a movie recommender system where a user has recently seen a movie of horror genre, a simple content based system might recommend similar horror movies or movies by the same director or having the same actors etc. There are various measures available to compare and compute the similarity of two items including but not limited to cosine similarity, Pearson's coefficient, distance.

Collaborative Filtering

Collaborative Filtering deals with a user-item matrix in most cases and tries to compute the rating that a particular user will give to an item he has not yet interacted with. One way to do this can be to find other users with similar tastes to the user who's rating we are trying to predict and take a weighted average of those ratings. Till now our discussion was focused on explicit ratings, i.e. the ratings take on numbers or values in a given domain. There is also the existence of implicit ratings, which can take the form of whether the intended user has viewed, clicked or seen a particular item rather than explicitly providing a rating.

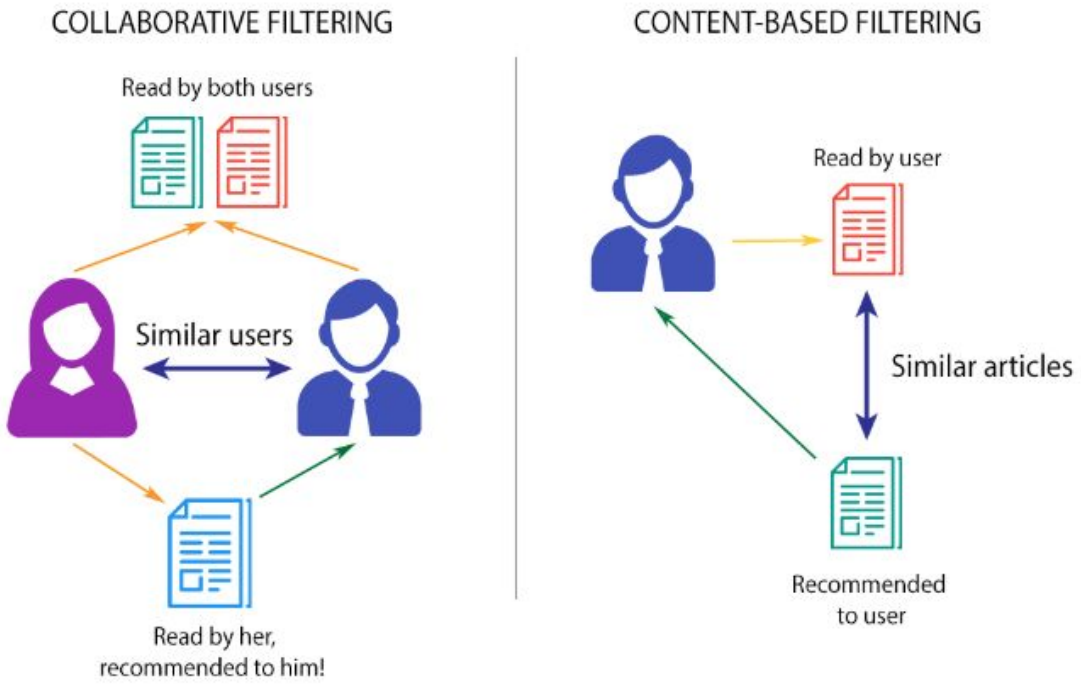

Fig 1.1 Types of Recommender Systems [2]

1.1 Relevance Of The Project

Recommender systems are still of prime importance today through their ability to provide a high return on investment when used wisely.

- By providing better recommendations, the time a user spends on a site is maximised thereby directly improving the cart value and helping users to "buy" more.

- Maximises the engagement and satisfaction of the user.

- Unlike other applications that degrade with time, recommender systems have a high retention rate which implies they perform better over time as they have a larger subsection of data to operate upon.

- Timely recommendations (for say, medical items or research papers) can help save a lot of time and increase productivity.

As stated above the advantages go on and on. We as humans always tend to be choosy be it movies, food etc. It all boils down to personal preference and that is exactly where recommender systems scintillate. These systems generate user profiles by learning the user's preferences, collaborates these preferences with other users (Collaborative filtering) and runs various algorithms on these profiles as discussed in the future sections. A detailed study on such a powerful concept helps understand the behaviour of the users and tweak the algorithms accordingly. The ultimatum of recommender systems to cater the best possible suggestions to the users to their vacillating predilections.

Despite all these advantages, recommender systems are not without their fair share of disadvantages. For example, there are no accurate solutions to certain problems that plague it like cold start etc. and no "best algorithm" that improves accuracy as such. It is a growing field with much research being carried out every single day to improve it.

1.2 Scope of Project

Recommender Systems continue to be pivotal in machine learning and serve an important role in commercial applications. One of the reasons for widespread popularity of websites such as Amazon, Netflix, BestBuy are the strength of their recommender systems. Netflix had announced the 1-million-dollar prize for collaborative – filtering algorithms that improved the accuracy over their benchmark by a factor of 10%.

The business objective of recommender systems is to maximize user satisfaction. The user's satisfaction is not merely based on the accuracy of the predictions but on a variety of other factors such as the novelty of the predictions and their usefulness.

There is no perfect solution to recommender systems in a field where extensive research has been carried out. Even advanced techniques like autoencoders and restricted Boltzmann machines do not yield the supreme results that would be expected, mainly due to human factors and computing constraints.

There are also problems like the grey sheep, data sparsity and cold start issues and that still need to be addressed completely.

1.3 Approaches

Below is a table containing brief summaries of the various approaches explored during our research and also have implemented each of these approaches and documented the error results and conclusions.

Table 1.1 Summary of Various Approaches

Name of Approach	Description
Content Based Filtering	User Specific Recommendations based on item's features. If a person liked a particular book, a content based system might recommend others of a similar plot/genre/type.
Collaborative Filtering	Tries to recommend items by finding out what similar users like.
kNN	A distance metric that can be used to find out near or "similar" movies given a database of movies. Can also be used to find similar users
Clustering	The task of partitioning the data set into a few groups such that the members in each group have minimal differences or are most similar to one another. For example, k-means clustering tries to assign points to clusters based on mean
Singular Value Decomposition	It is a matrix structure used for recommendation where rows represent users, columns represent items and the value of the cells denote the ratings that the users have given to those items. It is a collaborative filtering technique that deals with matrix factorization and helps in reducing the number of dimensions.
Auto Encoders	A type of Artificial Neural Network used to learn efficient codings for the input data typically for dimensionality reduction. It tries to replicate the input as effectively as possible from the learnt representation while trying to minimise the error.
Restricted Boltzmann Machine	Given a matrix of users across rows and their ratings for movies across columns, it performs a binary prediction whether the user would like a movie or not. It would act as a sieve over other ML algorithms to cater better recommendations.

1.4 Datasets Used

Table 1.2: A summary of data sets typically used for recommendation systems

Dataset	Description
MovieLens[3]	Collection of movie ratings. Comes in 1M,10M and 20M ratings. The largest set uses 1.40.000 users and spans across 27000 movies.
Zomato Restaurants Data (Kaggle)	All metadata-rating and location information about restaurants fetched via Zomato's API.
Yelp Dataset (Kaggle)	A subset of Yelp's business, reviews and user data. It was originally put together for the Yelp Dataset Challenge for students to conduct research or analysis on Yelp's data and share their discoveries
Restaurant Data (Kaggle)	This dataset was used for a study where the task was to generate a top-n list of restaurants according to the consumer preferences.
Jester	List of various jokes and their rating
Book Crossing	A Book rating and metadata dataset
Last.fm	Dataset for songs and includes the top song in a playlist and the amount of times that song has been listened to.
OpenStreetMap	Contains map related data. Objects in the dataset include roads, buildings, points-of-interest, etc.
TMDB-5000 (Replacement by Kaggle)	Contains movie information relating to the metadata and credits.

1.5 Challenges faced by recommender systems

Few of the challenges[4] that are faced by systems include:

1. Cold Start — This problem can occur when a new user enters the recommendation system and the user's preferences are not known. This can be avoided by asking the user to indicate his preferences or interests the first time he signs up for the service.

2. Grey Sheep — This takes place when one person's tastes differ from that of the group thereby rendering the recommendations provided to him/her useless. The easy way to avoid this is by perusing collaborative filtering systems that provide recommendations based on the personal interests and profile of the user.

3. Synonymy — This problem can occur when two words or items have different ways of expression, but they point to the same thing. For example, action movie and action film mean the same but a rote learning or memory based approach to filtering systems will not be able to capture this semantic similarity. By using methods like SVD, this error can be averted.

4. Shilling attacks — This attack happens when a malicious user starts providing false ratings intentionally in order to sabotage the system and lower the trustworthiness / relevance of the items recommended. The remedy to this kind of attack involves identifying prediction shift, hit ratio etc

5. Sparsity — As every user tends to rate a very small set of all the available movies in the data set, most of the time the rating matrix is sparse, making it difficult in cases where algorithms that recommend items based on a similarity metric are used as there are few available ratings. Algorithms like SVD and some content based collaborative algorithms can counter this effect.

CHAPTER 2

LITERATURE SURVEY

This chapter discusses the various research papers involving the same topic and analyzes the approaches they used and the conclusions and accuracy results they observed.

2.1 Jiang Zhang, Yufeng Wang , Zhiyuan Yuan, and Qun Jin - Personalized Real-Time Movie Recommendation System: Practical Prototype and Evaluation - 2020

Jiang Zhang et al. have presented a personalized real-time movie recommendation system based on a CF algorithm called Weighted KM-Slope-VU. Firstly, a simple but high-efficiency recommendation algorithm is proposed, which exploits users' profile attributes to partition them into several clusters. For each cluster, a virtual opinion leader is conceived to represent the whole cluster, such that the dimension of the original user-item matrix can be significantly reduced, then a Weighted Slope One-VU method is designed and applied to the virtual opinion leader-item matrix to obtain the recommendation results.

Weighted KM-Slope-VU, the popular K-means algorithm is chosen to cluster users, for its simplicity and effectiveness. This method has significantly reduced the time complexity, also achieving comparable recommendation performance. The proposed method has achieved an average RMSE of 0.95062 and 0.94676 on 10K and 1M datasets respectively.[5]

2.2 Khamphaphone Xinchanang, Phonexay Vilakone, and Doo-Soon Park - Movie Recommendation Algorithm using Social Network Analysis to alleviate Cold-Start problem - 2019

Khamphaphone Xinchanang et al. have developed a movie recommendation algorithm using Social Network Analysis and collaborative filtering . This algorithm uses personal information of users such as age, gender, and occupation to make a relationship matrix between users, and the relationship matrix is applied to cluster

users by using community detection based on edge betweenness centrality. Then the recommended system will suggest movies which were previously interested by users in the group to new users.

The efficiency of the SNA and CF method is compared with the normal CF method, kNN and CF method, and Density-based clustering method. The MAE was observed to be very less(3.55) using the SNA and CF method when compared with the other methods. But this value is a lot much when compared to the latest algorithms developed by others. [6]

2.3 Md. Akter Hossain, Mohammed nazim uddin - A Neural Engine for Movie Recommendation System - 2019

Md. Akter Hossain and Mohammed Nazim uddin proposed a Neural engine for movie recommendation system(NERS). In this system(NERS), they have incorporated data contents about the user's interests via a standard movie dataset that helps them make a neural engine called neural recommender (NR). Firstly, they use a collective dataset to predict movie outputs. Secondly, they developed explicit prediction models for different types of movies. This model helped both, the system and user, the flexibility to fetch information according to user expectations. Then, it uses two different types of clustering algorithms for evaluating their approach : Silhouette and DaviesBouldin measures and compares the performance with two proficient estimators.

At last, three estimators, mean square error (MSE), mean absolute error (MAE) and mean relative error (MRE), were exploiting to demonstrates prediction accuracy of NERS approach The MAE, MSE and MRE was calculated to be 1.97, 4.75 and 6.06% respectively which is quite a lot compared to many other approaches. [7]

2.4 Rahul Katarya, Om Prakash Varma - An effective collaborative movie recommender system with cuckoo search - 2017

Rahul Katarya and Om Prakash Varma [8] developed a movie recommendation system whose primary objective is to make suggestions through data clustering and computational intelligence. It uses k-means clustering algorithm along with cuckoo

search optimization algorithm applied on the Movielens datasets. The clusters are selected randomly at first then users are inspected one by one by calculating the differences in their ratings and the centroid of the clusters, and if their difference is smallest, then the user gets allocated to the cluster to which they are closest. Initially the k-means clustering algorithm is applied to Movielens 100K dataset. Next cuckoo search optimization algorithm is applied to the resultant of the k-means algorithm for optimizing the results. This approach was able to provide an MAE of 0.754 and RMSE of 1.266. [8]

2.5 Alif Azhar Fakhri, Z K A Baizal and Erwin Budi Setiawan - Restaurant Recommender System Using User-Based Collaborative Filtering Approach: A Case Study at Bandung Raya Region - 2019

Alif Azhar Fakhri, Z K A Baizal and Erwin Budi Setiawan proposed a recommendation system that implements a user-based collaborative filtering algorithm for recommending restaurants. If the user wants to find a restaurant recommended by another user, then the system will search the neighbors who have biggest similarity with that target user, The restaurants that have been given a rating by neighbors will be recommended to target users who have not rated that restaurant.

Similarity to find the proximity between users is calculated using two stages: 1) calculating the user similarity and 2) calculating the user attribute similarity. They found an MAE of 1.492 for calculation without user attributes and 2.166 for calculation with user attributes. Lesser the value of MAE, better the performance of the system. Hence, the authors concluded that the recommender system performs better without computing the user attribute similarity. [9]

2.6 Bhagyashree Basudkar, Shruti Bagayatkar, Meghana Chopade, Sachin Darekar - Restaurant Recommendation System Using Customer's Data Analysis - 2018

Bhagyashree Basudkar, Shruti Bagayatkar, Meghana Chopadeand Sachin Darekar attempt to understand, analyze and suggest restaurants to a particular user on the basis of user behavior and the restaurant rankings using Zomato's API. They have proposed an android and web application that focuses on user's behavior and generates

predictions based on the user's location and the popularity of the restaurant by using user ratings. Their application will also notify the user with the nearest restaurants when the user is in motion.

They have used content-based filtering as well as collaborative filtering to make the system more effective. The users' locations are tracked using gps. The recommender system adopted a user preference model by using the features of user's visited restaurants, and also the location information of users and restaurants in order to dynamically generate the recommendations. [10]

2.7 Ravinarayana A, Pooja M C and K Raghuveer - Using Clustered Database for Food Recommendation System - 2016

Ravinarayana A, Pooja M C and K Raghuveer attempted to create a food recommendation system that uses a content based filtering technique to recommend food items to the user. The system will suggest food items based on user inputs and also provide users a list of nearby restaurants. The system works as follows: the user is asked to input his/her favorite food items as well as the location in which he wants to find restaurants. In turn, a list of top ingredients of the users favorite food items is fetched. Clusters of food items are maintained along with the list of important keywords that belong to that respective cluster. They perform clustering by using the k-means method.

k-Means clustering was chosen to maintain the database as an attempt to increase the performance of existing systems. They concluded that their proposed system reduces the time taken to make comparisons for similarity in the database by 94%. The authors observed that the clustered database addresses the problem of dimensionality. In order to improve the accuracy even further, they will attempt to use a hybrid algorithm in future implementations. User specific recommendations can also be given using a reinforcement learning approach. [11]

2.8 Sumedh Sawant and Gina Pai - Yelp Food Recommendation Challenge -2013

Sumedh Sawant and Gina Pai used the Yelp dataset to implement a food recommendation system using various algorithms such as singular value

decomposition, weighted bipartite graph and hybrid cascade of kNN clustering. In order to evaluate and compare the performances of the difference implementations, they have chosen the metrics Root Mean Square Error and Mean Absolute Error.

They concluded that the implementation of the cascaded clustered multi-step weighted bipartite graph projection algorithm performed the best out of all the algorithms with RMSE of 1.09262 and MAE of 0.67548. In future implementation, they will attempt to augment the current analysis to include review text and user rating evaluations (whether other users thought a particular user's review was funny, useful, or helpful) as features in the prediction model. They will also explore further hybrid approaches and evaluate their performances to current implementations. [12]

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

3.1 Functional Requirements

- Users should be able to get a precise list of recommendations pertaining to his interest.
- Interface to the ongoing events, plays, movies.
- Performance improvement over time.
- Allow the user to change his preferences without impacting functionality.

3.2 Non-Functional Requirements

- Learnability
- Reusability
- Performance
- Reliability
- Correctness
- Privacy

3.3 Hardware Requirements

- **Processors:** Intel i3,i5,i7
- **Processor Speed:** 3.00GHZ
- **RAM:** 4GB
- **Storage:** 500GB

3.4 Software Requirements

- **Operating System:** Windows
- **Web Browser:** Google Chrome
- **RAM :** 1GB+
- **Python version :** 3.6.X

CHAPTER 4

IMPLEMENTATION

Various recommendation algorithms were implemented and their results and error metrics were noted.

4.1 Restaurant Recommendations

4.1.1 Content Based Approach using Cosine Similarity

In this implementation, the Zomato Dataset, containing restaurants across the world is considered. The dataset includes features such as city, locality, cuisine, aggregate rating as well as the total number of votes for each restaurant mentioned. In order to provide personalized recommendations to the user, the top, similar restaurants for a user are recommended by taking into consideration their location as well as the cuisine of the restaurant.

Using TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfv = TfidfVectorizer(min_df=3, max_features=5000,
analyzer='word', token_pattern=r'\w{1,}',
                      ngram_range=(1,3),
                      stop_words = 'english')
```

Using TF-IDF vectorizer, a maximum of 5000 features are extracted in the cuisine columns across all the restaurants. Sporadic terms are eliminated in order to get rich and meaningful features.

Cosine similarity is used to calculate the pairwise similarity between the restaurants. This similarity score calculated will be then used to determine the top ten most

similar restaurants with respect to the user's location and cuisine of the restaurant specified.

Code snippet

```
#Cosine Similarity
#Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
#Get indices
corpus_index=[n for n in data_sample['Split']]
indices=pd.Series(data_sample.index,index=data_sample['Restau
rant Name']).drop_duplicates()
 #index of the restaurant matchs the cuisines
idx = indices[title]
   #Aggregate rating added with cosine score in sim_score
list.
   sim_scores=[]
   for i,j in enumerate(cosine_sim[idx]):
        k=data_sample['Aggregate rating'].iloc[i]
        if j != 0 :
            sim_scores.append((i,j,k)
    #Sort the restaurant names based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: (x[1],x[2])
, reverse=True)
    # 10 similar cuisines
    sim_scores = sim_scores[0:10]
    rest_indices = [i[0] for i in sim_scores]
   data_x =data_sample[['Restaurant Name','Aggregate
rating']].iloc[rest_indices]
    data_x['Cosine Similarity']=0
    for i,j in enumerate(sim_scores):
        data_x['Cosine
Similarity'].iloc[i]=round(sim_scores[i][1],2)
    return data_x
restaurant_recommend_func('Connaught Place','Sbarro')
```

Cosine similarity is chosen as the similarity metric as it is proven to be better when compared with other similarity metrics like Euclidean distance and Pearson's correlation as it measures the similarity between items irregardless of their size. Cosine Similarity measures the cosine of the angle between two vectors (or

documents) projected in a multi-dimensional space. Therefore, even if two documents are far apart with respect to their Euclidean distance, there may be a chance they could have a smaller angle between them as smaller the angle, greater its similarity.

Table 4.1 Restaurant Recommendation Using Content Based Approach using Location

Restaurant ID	Restaurant Name	Aggregate Rating	Cosine Similarity
63	Pizza Hut	3.5	0.86
26	Caffe Tonino	3.9	0.79
32	Domino's Pizza	3.7	0.70
91	Ovenstory Pizza	0.0	0.70
24	Cafe Public Connection	3.7	0.33
12	The Immigrant Cafe	3.2	0.33
1	Attitude Kitchen & Bar	2.9	0.33
112	Smoke on Water	4.1	0.33
95	Ardor 2.1	4.1	0.28
94	Ambrosia Bliss	4.0	0.28

A reason why content based systems are useful and easy to implement is because they do not require any data about other users using the system which makes it easier for the system to scale to a large number of users. Another advantage of this approach is that the recommendations made for the target user are specific to that user only. For this reason, content based systems work well if a user has an interest that is not very common among all users.

Content based systems also come with their limitations. Implementing such a system requires extensive knowledge of that particular domain. In addition to this, these systems only provide recommendations based on the user's current likes and has no way of expanding the user's interest beyond this. Due to these disadvantages, content based systems are not used in all situations.

4.1.2 kNN Item Based Collaborative Filtering

Collaborative Filtering is a machine learning technique that is used to make predictions based on the past behavior of users. It can be either item-based or user-based. The bottleneck in traditional collaborative filtering algorithms is searching for neighbors among a large user population of potential neighbors. Item-based filtering algorithms avoid this bottleneck by exploring the relationships between items ratings instead of the relationships between users as user behavior tends to be dynamic in nature whereas items' ratings tend to remain static. Item-based algorithms provide recommendations for users by finding items that are similar to other items the user has liked in the past.

An item based collaborative filtering technique using k-Nearest Neighbors algorithm is implemented. kNN algorithm is a supervised machine learning technique that uses labeled input data for learning in order to make predictions on new, unlabeled data.

For the implementation, Restaurant Dataset (Kaggle) [13] which contains user ratings for particular restaurants is used. Then combine all the ratings given by users for a particular restaurant in order to find out the total number of ratings each restaurant has.

A 'popularity threshold' is considered in order to recommend restaurants having more number of ratings than the 'popularity threshold'. The threshold we have considered in '25'. Then a pivot table is created containing the ratings given by each user for a particular restaurant. The table has restaurant names are the indices and the user id as the columns. This table is then converted into an array matrix by importing the csr.matrix library from the scipy.sparse package. The metric chosen for calculating the distance between items is Cosine Similarity and the number of neighbors chosen is 5.

Code Snippet

```
popularity_threshold = 25
rating_popular_rest=rating_with_totalRatingCount.query('total
RatingCount>= @popularity_threshold')
rating_popular_rest.head()
```

```
rest_features_df=rating_popular_rest.pivot_table(index='name'
,columns='userID',values='rating').fillna(0)
rest_features_df.head()
from scipy.sparse import csr_matrix
rest_df_matrix = csr_matrix(rest_features_df.values)
from sklearn.neighbors import NearestNeighbors
model_knn = NearestNeighbors(metric = 'cosine', algorithm =
'brute')
model_knn.fit(rest_df_matrix)
query_index = np.random.choice(rest_features_df.shape[0])
print(query_index)
distances,indices=model_knn.kneighbors(rest_features_df.iloc[
query_index,:].values.reshape(1, -1), n_neighbors = 4)
```

Figure 4.1 Output for kNN Item Based Collaborative Filtering

```
Recommendations for Cafeteria y Restaurant El Pacifico:

1: La Cantina Restaurante, with distance of 0.6580072641372681:
2: puesto de tacos, with distance of 0.7170524597167969:
3: Tortas Locas Hipocampo, with distance of 0.7254151701927185:
```

The surprise module is used to calculate the RMSE and the MAE by importing KNNBasic.

Table 4.2 Evaluating RMSE, MAE of algorithm KNNBasic on 2 splits

	Fold 1	**Fold 2**	**Mean**	**Std**
RMSE	0.8218	0.8324	0.8271	0.0053
MAE	0.6584	0.6636	0.6610	0.0026
Fit time	0.03	0.05	0.04	0.01
Test time	0.0	0.0	0.0	0.0

The advantage of using a kNN algorithm based approach is that it does not require any training before making predictions and adding new items does not affect the accuracy of the algorithm. Another reason as to why kNN is a popular approach is

that it is relatively simpler to implement as it considers only two parameters: the value chosen for 'k' and the distance measured between items.

Item based Collaborative filtering algorithms are advantageous because the average rating of items don't change as often so the user-item matrix does not need to be computed frequently leading to lower computation costs.

4.2 Movie Recommendation

4.2.1 Built-in Algorithms

Code Snippet

```
from surprise import SVD , SVDpp , NMF , KNNBasic ,
NormalPredictor
from surprise import Dataset
from surprise.model_selection import cross_validate
#Base-lines
data = Dataset.load_builtin('ml-100k') #Movielens 100k
Dataset- preloaded
algorithms = [SVD() , KNNBasic() , NormalPredictor()] #SVD++
, Matrix Factorization Models
for algo in algorithms:
    cross_validate(algo, data, measures=['RMSE', 'MAE'],
cv=5, verbose=True)
```

The surprise module is a specialized module built to provide users complete control over their recommender systems research and experiments as well as a method for evaluation of such systems. Another advantage of using this module is that few datasets such as MovieLens are built-in along with implementations of popular algorithms.

A few of the built-in popular algorithms like SVD, kNN are run on the movielens 100k dataset to establish baseline metrics. 5-fold cross validation was conducted and the mean is documented.

Table 4.3 Evaluation results of Built-in algorithms

Built-in Algorithm	Root Mean Square Error(RMSE)	Mean Absolute Error(MAE)
SVD	0.9364	0.7385
kNN	0.9790	0.7730
Normal Predictor	1.5233	1.2232

4.2.2 Mean and Random Measures

Using the surprise module, one can implement and execute one's own algorithms by implementing two of the methods fit and estimate derived from AlgoBase as shown below:

Code Snippet

```
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise import AlgoBase
import random
#Dumb Sample Model to get started
class ZeroAl(AlgoBase):

    def __init__(self):
        AlgoBase.__init__(self)

    def fit(self , trainset):
        AlgoBase.fit(self , trainset)
        return self

    def estimate(self, u, i): #Have to predict rating
        return 0
    #If we return 0 , the maximum deviation will be there
from rating as ratings are from 1-5

data = Dataset.load_builtin('ml-100k')
algo = ZeroAl()

cross_validate(algo, data, verbose=True)
```

In order to build our recommendation algorithm with surprise, one needs to encapsulate the idea into a class while providing definitions for three methods - init , fit and estimate.

__init__() - This is a basic python constructor that instantiates the super class AlgoBase

fit(self , trainset) - This method is called once on the entire training set. Here, weights are learned and features are processed etc.

estimate(self , u , i) - This method takes in two parameters: the User Id and Item Id. It is called for every row of the test set. Based on the learning done in the fit method, this method is expected to return a numeric value of the rating which this user would have provided for this particular item. In the case of movies, this method is called for every (User Id , Item Id) in the testing set and outputs a numeric value (1-5) the particular user is expected to provide for the movie under consideration based on a variety of factors that may include history, interests etc.

The mentioned model is built to serve the purpose of proving that with surprise, building models are very simple and also to provide an example of a bad algorithm that should be steered clear of. By mindlessly returning 0 regardless on what userId and movieId is being tested, we achieve the most deviation from the observed ratings providing us with an abysmal RMSE of 2.7690 and MAE of 2.5299 as expected

So to summarize, in order to build algorithms in surprise, a dataset must be loaded, implement the algorithm / use built-in ones and then it gives the error estimates. In case, the need arises to test our own model, one must first implement the fit and estimate methods. The fit method is called once on the training set and the estimate method is called for every row of the test set and expects the predicted rating for given user id (u) and item id (i) taken as parameters. (Reminisce that the data set is organized into User Id, Movie Id, Rating and Time Stamp) Now once we get this out of the way, we can continue on our road of experimenting with various other simple ideas slowly progressing in difficulty and improving the accuracy.

First we try to find the RMSE and MAE in two cases, when we return the mean of all the ratings and when we randomly return an acceptable value

Table 4.4 How accurate is returning the mean of all ratings or returning a random value

Algorithm	RMSE	MAE
Mean of all ratings	1.1257	0.9447
A random rating value(1-5)	1.8869	1.5129

4.2.3 Age-based Clustering

Here, classification of people is done according to their age into certain age groups and try to ascertain the rating they would provide to a particular movie based on the cluster in which they belong. Some Exploratory Data Analysis on the data reveals that the youngest person in the data set is 7 years old and the oldest person is 73 years old. Based on this, the age groups that are identified are 7-17, 18-29, 30-40, 41-50, 51-60 and 61-73. The reasoning behind this approach is the idea that few movies are targeted towards certain age groups, taking the average of the ratings of other users in a similar age group for a particular movie, the ratings are computed. However, it must be noted that this isn't a foolproof method as an individual's taste may differ f
rom the group/cluster allotted to him and few movies are marketed as being for all age groups or there may be a few outliers. It should also be kept in mind that after analyzing the data, it is found that the most common rating users provided to a movie was 4. A few variations of this method were tried, one involving taking just the average of all ratings of users in the same cluster as the one to be predicted irrespective of the rating as in if the rating of a user is to be predicted who happens to fall in cluster 3 (30-40), the average of all ratings in that age group is returned. The second method takes into consideration the movie too, as in if we are trying to predict the rating a user(u) would give to a movie(m) and the user lies in the group 2(18-29), the average rating of movie m in that age group is returned. It should be noted that these two algorithms from scratch to the best of our ability.

Code Snippet

```
user= files_path+'u.user'
age = {}
#Some EDA
maximumAge = -1
minimumAge = 2**15
for line in open(user):
    array = line.split('|')
    age[(int(array[0]) - 1)] = array[1] #age[userId] = age of
that person
    if int(array[1]) > maximumAge:
        maximumAge = int(array[1])
    if int(array[1]) < minimumAge:
        minimumAge = int(array[1])
print('The oldest person in the dataset is ' , maximumAge , '
years old')
print('The youngest person in the dataset is ' , minimumAge ,
' years old')

#A short sample snippet deciding which people belong to
particular clusters
def fit(self, trainset):
        AlgoBase.fit(self, trainset)
        self.count+=1
        if self.count==1:
            c = [(userId, movieId, rating)for (userId,
movieId, rating) in self.trainset.all_ratings()]
            for userId , movieId , rating in c:
                ageOfUser = age[userId]
                ageInt = int(ageOfUser)
                if ageInt >=1 and ageInt <= 6:
                    rateAge[0] = rateAge.get(0 , 0)+rating
                    ratingCount[str(0) +" "+str(rating)] =
ratingCount.get(str(0) +" "+str(rating) , 0)+1
                    countAge[0] = countAge.get(0 , 0)+1
                elif ageInt >=7 and ageInt <= 17:
                    rateAge[1] = rateAge.get(1 , 0)+rating
                    ratingCount[str(1) +" "+str(rating)] =
ratingCount.get(str(1) +" "+str(rating) , 0)+1
                    countAge[1] = countAge.get(1 , 0)+1
```

```
#Similar short sample snippet for classification - complete
code not displayed
def estimate(self, u, i):
        ageInt = int(age[u])
        cluster = 0
        self.count2+=1

        #Cluster Finding Code here (not shown)
        Rating = rateAge.get(cluster)/countAge.get(cluster)

        ratings = ["1" , "2" , "3" , "4" , "5"]
        estimatedRating = Rating
        maxCount = -1
        for rating in ratings:
            key = str(cluster)+" "+str(float(rating))
            if ratingCount.get(key , 0) > maxCount:
                maxCount = ratingCount.get(key, 0)
                estimatedRating = int(rating)
        prob = random.randint(0 , 1)
        if prob==0 or prob==1: #50% of the time , by making
this 100% , RMSE becomes 1.21 but MAE is 0.89 always
            return estimatedRating
        else:
            return Rating
```

Here, the above code shows the general idea of the described approach. By splitting people according to their age groups, an attempt was made to find similar users in the target user's age group with similar interests in order to recommend relevant movies.

The approaches tried included trying out the age-metric alone, supplementing the age data with the movie information and trying to return the most common rating within a given cluster.

Table 4.5: Analysis of clustering algorithms based on age

Technique	RMSE	MAE
Age-Based	1.254	0.9436
Age-Based along with movie	1.26	1.00
Most common rating	1.2199	0.8942

4.2.4 Genre Based Recommendation

In accordance with the movie lens dataset, the movies.csv file provides movie id, movie name and genres of the movie. The user's previous ratings on the movies are grouped into three groups i.e. ratings 3, 4 and 5 respectively.

Code Snippet

```
rating3 = {}
rating4 = {}
rating5 = {}
r3=training_set[training_set[2]==3]
r4=training_set[training_set[2]==4]
r5=training_set[training_set[2]==5]
# Rating 3
for i in range(len(r3)):
    movieid = r3.iloc[i,1]
    userid = r3.iloc[i,0]
    try:
        if movieid <= len(movies):
            genres = movies.loc[movieid,2].split('|')
            for genre in genres:
                if (userid,genre) in rating3:
                    rating3[(userid,genre)] += 1
                else:
                    rating3[(userid,genre)] = 1
    except:
        pass
# Similarly for rating 4 and 5
```

To predict how the user would rate an unprecedented movie, the genres of the movies the user rated previously is stored and the frequency of each genre occurring in each group is calculated. For instance, if the user rated 5 for Action movies 10 times, rated 4 three times, then the user is most likely to rate the action movie 5 again. Considering the genres comprised in a movie, the mean of ratings cohering with the frequency is computed. The error metrics evidently proves a decent score for a genre based proposed approach.

Table 4.6: Analysis of genre based clustering

Technique	RMSE	MAE
Genre	1.45199	1.14648
Genre with supplemented rating	1.24133	0.90754

4.2.5 Cosine Similarity

Cosine similarity is a metric that is generally used to determine how similar documents are based on their content. Formally, it is a measure of similarity between two non-zero vectors of an inner product space.

Code Snippet

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidfObj = TfidfVectorizer(stop_words='english')
movies['overview'] = movies['overview'].fillna('')
tfidf_matrix = tfidfObj.fit_transform(movies['overview'])

from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
indices = pd.Series(movies.index,
index=movies['title']).drop_duplicates()
```

The simplest form of a cosine similarity metric is described above. In this code, similar movies are recommended to the users on the basis of how similar their plot

vectors are. This could be made more intuitive by using the features of the movie like Director , Actor , Genre etc as shown in the sample (not complete code) below:

Code Snippet

```
from ast import literal_eval

features = ['cast', 'crew' , 'keywords' , 'genres'] #First
two are in one dataset
movies['cast'] = credz['cast'].apply(literal_eval)
movies['crew'] = credz['crew'].apply(literal_eval)

import numpy as np
def get_director(x):
    for i in x:
        if i['job']=='Director':
            return i['name']
    return np.nan
#Trying to add more weight to the director or a specific
feature
def weighted_feature(x , feature='director' , times=100):
    feature_list = ['keywords' , 'cast' , 'director' ,
'genres']
    string = ' '
    for feat in feature_list:
        if feat!=feature:
            string+=' '.join(x[feat])+' '
        else:
            for j in range(times):
                string+=''.join(x[feat])+' '
                #print(string)
    string.rstrip()
    return string
movies['weighted_director1'] = movies.apply(weighted_feature
, axis=1)
```

The idea behind this is that we can use a combination of different features to guide our recommendations, not only plot and to provide more weight to a certain feature, we use the weighted_feature function which operates by appending the specified

feature given number of times to the feature string. This works because instead of plot, our target feature string is now a space delimited concatenated string containing data like director, actor, genre, etc. By appending one string many times to that string, the relative importance of that particular feature is increased.

This is all done on the TMDB-5000 data set provided by Kaggle. But this helps us for content-based filtering. In order to utilise this approach for collaborative filtering, we need to modify it a bit.

A sort of hybrid approach is utilized by including two data sets - The Kaggle TMDB replacement dataset which has information regarding 5000 movies and the existing MovieLens data set. Cosine similarity measures were used in order to identify similar movies based on how similar the plot vectors of two movies are providing more weight to the "director" factor thereby getting movies with similar plot along with good movies from the same director. Apart from this a different similarity measure was also tried where the metric of comparison wasn't weighted in terms of director but rather was considered to be a mixture of actors , genre and plot.

So our idea involved fetching similar movies (in the other data set) to the one we're currently trying to predict (in MovieLens) and our estimated predicted was the weighted average of these

Table 4.7: Evaluation of Cosine Similarity for Collaborative Filtering

Metric	**RMSE**	**MAE**
Similarity of Plot (giving director weight)	1.2323	0.9433
Mixture of plot , genres , actors etc.	1.2208	0.9355

4.2.6 k-Nearest Neighbors

This machine learning algorithm is used to find clusters of similar users based on common movie ratings, and make predictions using the average rating of top-k nearest neighbors. Using the MovieLens 2M dataset from GroupLens, a ratings matrix is constructed, with the matrix having one row for each movie and one column for each user. Then, the k items that have the most similar user engagement vectors are found. This algorithm uses Brute force implementation to compute the nearest neighbors and cosine metric to calculate the cosine similarity between rating vectors.

Code Snippet

```
from sklearn.neighbors import NearestNeighbors model_
knn = NearestNeighbors(metric = 'cosine', algorithm =
'brute')
model_knn.fit(user_rating_matrix)
```

Next, the closeness of instances are determined by calculating the distance. Then the algorithm classifies an instance by finding its nearest neighbors, and picks the most popular class among the neighbors.

Code Snippet

```
query_index = np.random.choice(user_rating_pivot.shape[0])
distances, indices = model_knn.kneighbors(user_rating_pivot.
iloc[query_index,:].values.reshape(1,-1), n_neighbors = 6)
for i in range(0, len(distances.flatten())):
if i == 0:
print('Recommendations for {0}:\n'.format(
user_rating_pivot.index[query_index]))
else:
print('{0}: {1}, with distance of {2}:' .format(i,
user_rating_pivot.index[indices.flatten()[i]],
distances.flatten()[i]
```

4.2.7 Support Vector Decomposition

Inspiration was drawn from Nicolas' article[22] on matrix factorization for this section and an attempt was made to implement SVD from scratch with stochastic gradient descent for optimization. The problem involves finding the two matrices (p , q) whose product gives us the rating matrix (R). To this end the equation(1) has to be minimised.

$$\sum_{r_{ui} \in R} (r_{ui} - p_u.q_i)^2 \quad \text{---------- (1)}$$

The vector p(u) is used to represent the affinity of the user towards the latent factors. If the factors are Action, Romance, Horror and let's say a particular user (say, Raul) is prone to those factors. In other words, an effort is being made to decompose the user as 10% , 30% , 60% etc. which means that this user particularly likes Horror.

The vector q(i) is used to represent the affinity of the items towards these same latent factors. For example if the example of movie "Shining" is considered with respect to the factors like Comedy, Action, Horror , the matrix might be like 0% , 10% , 80% indicating that this particular movie has an affinity for Horror.

Now if an estimation of Raul's rating for "Shining" was needed to be made, the product of the two matrices is necessary to be found but here it is reasonable to assume that he would provide a high rating for this movie.

Gradient Descent is used as an optimisation technique used to minimise a given function by iteratively moving in the direction of steepest descent (-ve gradient)

In this case , p(u) denotes the row vector of the matrix p and q(i) denotes the column matrix at position i. Assuming a stochastic gradient descent approach to minimize the above expression by starting with random values for p and q and for a given number of epochs updating those parameters by subtracting the product of the derivative and learning rate. One of the defining characteristics of the rating matrix is that it is most of the time sparse. We assume that SVD will help identify the latent factors and the corresponding strength of each factor (In a simple sense in the case of movies, it can be stated as how much a user is prone to some factors like action, comedy etc and

how much the strength of that factor is in each movie in the dataset. Thus, the multiplication of the two factored matrices would provide a way of constructing back the matrix and guessing the values which is not known).

Code Snippet

```
#Inspired by Nicolas Hug's Blog
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise import AlgoBase
import random
import numpy as np

class SVDIn(AlgoBase):
    p =  []
    q =  []
    c = 0
    est = 4
    learningRate = 0.001
    noOfFactors = 10
    def __init__(self , learningRate=learningRate ,
noOfFactors=noOfFactors):
        AlgoBase.__init__(self)
        self.learningRate = learningRate
        self.noOfFactors = noOfFactors

    def fit(self, trainset):
        AlgoBase.fit(self, trainset)
        self.trainset = trainset
        p = np.random.normal(0 ,  0.1 ,
(self.trainset.n_users, self.noOfFactors)) #Normal
distribution pick
        q = np.random.normal(0 , 0.1 , (self.trainset.n_items
, self.noOfFactors))
        self.c+=1
        if(self.c==1):
            print(len(q) , len(q[0]) , len(p)  , len(p[0]))
        for epoch in range(15):
            for u, i, rating in trainset.all_ratings():
                estimatedRating = np.dot(p[u] , q[i])
```

```
                self.c+=1
                if(self.c==2):
                    print(p[u])
                    print(q[i])
                error = rating - estimatedRating
                p[u]+=self.learningRate * error * q[i]
                q[i]+=self.learningRate * error * p[u]
        self.p = p
        self.q = q
        return self

    def estimate(self, u, i): #Have to predict rating
        if self.trainset.knows_user(u) and
self.trainset.knows_item(i):
            return np.dot(self.p[u] , self.q[i])
        else:
            return self.est   #Or Global Mean
data = Dataset.load_builtin('ml-100k')
algo = SVDIn()
cross_validate(algo, data, cv = 5 , verbose=True)
```

The main SVD part takes up less than 15 lines of the above snippet. It should be noted this simple attempt at SVD performs well enough to beat some of the built-in algorithms reaffirming the strength of matrix based factorization models.

Table 4.8: Evaluation for various number of latent factors

No. of factors	RMSE	MAE
10	0.9598	0.7521
15	0.9600	0.7534
20	0.9587	0.7522
50	0.9696	0.7614
100	0.9807	0.7696

In the above experiments, learning rate was fixed to 0.01 and the number of epochs were 10, now the error estimates are calculated if factors are 20 and learning rate is slowly varied.

Table 4.9: Error estimates for varying learning rates

Learning Rate	RMSE	MAE
0.001	1.5000	1.1771
0.01	0.9602	0.7501
0.02	1.0000	0.7776

This takes a large amount of time if the number of epochs are very large as it is implemented in the fit method and for each iteration the entire training set is processed. Large nudges to the learning rate are avoided so as to prevent overshooting the minima. The best learning rate seems to be 0.01.

4.2.8 Neural Networks / Auto Encoders

An autoencoder is a specialised type of Artificial Neural Network, typically used for dimensionality reduction that tries to learn efficient encodings for the input. There are various kinds like Sparse, De-Noising , Stacked , Contractive etc. [21]

The main part of the code is described below:

```
class StackedAutoencoders(nn.Module):
    def __init__(self ,  ):
        super(StackedAutoencoders , self).__init__()
        self.firstConn = nn.Linear(noOfMovies , 20)
        self.secondConn = nn.Linear(20 , 10)
        self.thirdConn = nn.Linear(10 , 20)
        self.fourthConn = nn.Linear(20 , noOfMovies)
        self.act = nn.Sigmoid()
    def forward(self , x): #Input
        x = self.act(self.firstConn(x))
        x = self.act(self.secondConn(x))
        x = self.act(self.thirdConn(x))
        x = self.fourthConn(x)
```

```
        return x
obj = StackedAutoencoders()
criteria = nn.MSELoss()
optimizer = optim.RMSprop(obj.parameters() , lr=0.01 ,
weight_decay=0.5) #Learning Rate
noOfepochs = 300
xd = 0
for iteration in range(1, noOfepochs+1):
    trainLoss = 0
    noOfProper = 0.0
    for userId in range(noOfUsers):
        currentInput =
Variable(trainSet[userId]).unsqueeze(0)
        target = currentInput.clone()
        if torch.sum(target.data > 0) > 0:
            predictedRating = obj(currentInput)
            target.require_grad = False
            predictedRating[target == 0] = 0
            loss = criteria(predictedRating , target)
            mean_correct = noOfMovies
/float(torch.sum(target.data > 0) + 1e-10)
            loss.backward()
            trainLoss+= np.sqrt(loss.data * mean_correct)

            noOfProper+=1.0
            optimizer.step()
    print('Iteration : '+str(iteration) + '
loss:'+str(trainLoss / noOfProper))
```

Using the existing files in the movie lens data set for cross validation (u1.base, u1.test, ... u5.base, u5.test), a stacked autoencoder with pytorch is constructed and trained on the u4.base and tested on u4.test. Running the SVD built-in on the mentioned files we get an RMSE of 0.9337. The auto-encoder results are as follows:

Table 4.10: Neural Network Evaluation using RMSprop as optimiser

No. of Epochs	RMSE
50	0.9754
100	0.9409
150	0.9419
200	0.9450

We have tried to use other optimisers like Adagrad and SGD but RMSprop seems to give us the best output with the least hassle.

4.2.9 Restricted Boltzmann Machine

Restricted Boltzmann Machine (RBM) is a probabilistic graphical artificial neural network model that learns from probability distribution over a set of inputs. RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph: a pair of nodes from each of the two groups of units (commonly referred to as the "visible" and "hidden" units respectively) may have a symmetric connection between them; and there are no connections between nodes within a group. By contrast, "unrestricted" Boltzmann machines may have connections between hidden units. This restriction allows for more efficient training algorithms than are available for the general class of Boltzmann machines, in particular the gradient-based contrastive divergence algorithm.

This algorithm uses the movie lens 100k ratings dataset and is trained and tested on the train-test splits (u1.base, u1.test, ..., u5.base, u5.test) The algorithm performs binary classification i.e., if user likes the movie it returns 1 else 0. The ratings dataset is pre-processed to achieve a matrix having users as rows and movies as columns. If the movie is rated below 3, it implies the user didn't enjoy the movie but if he/she rated 3 and above they enjoyed it. So ratings of 1 and 2 are made 0 in the matrix while the ratings 3, 4 and 5 are given 1. Note that a rating of -1 is used if the user hasn't rated the movie.

Code Snippet

```
class RBM():
    def __init__(self, nv, nh):
        self.W = torch.randn(nh, nv)
        self.a = torch.randn(1, nh)
        self.b = torch.randn(1, nv)
    def sample_h(self, x):
        wx = torch.mm(x , self.W.t())
        activation = wx + self.a.expand_as(wx)
        p_h_given_v = torch.sigmoid(activation)
        return p_h_given_v, torch.bernoulli(p_h_given_v)
    def sample_v(self, y):
        wy = torch.mm(y , self.W)
        activation = wy + self.b.expand_as(wy)
        p_v_given_h = torch.sigmoid(activation)
        return p_v_given_h, torch.bernoulli(p_v_given_h)
    def train(self, v0,  vk, ph0, phk):
        self.W += (torch.mm(v0.t(), ph0) - torch.mm(vk.t(),
phk)).t()
        self.b += torch.sum((v0 - vk), 0)
        self.a += torch.sum((ph0 - phk), 0)
```

This class is used to sample the hidden and visible nodes using Gibbs sampling technique. Pytorch libraries have been used for sampling probability distributions. K step contrastive divergence is performed to minimize the overall cost function. Computing gradients are computationally expensive and hence techniques like Contrastive Divergence have been employed to approximate the likelihood gradient. Training has been done for various epochs on u5 train-test splits as seen in the results below:

Table 4.11: RBM evaluation on different number of epochs

No. of Epochs	RMSE	MAE
10	0.41866	0.23041
20	0.41144	0.22523
50	0.40545	0.22369

75	0.40373	0.21915
100	0.41502	0.23174

75 epochs seem to be giving the best results overall. Now we try to tweak the batch size which determines for how many input rows supplied to the neural network, the weights and bias get updated.

Table 4.12: RBM evaluation on various batch sizes

Batch Size	**RMSE**	**MAE**
1	0.44339	0.25367
10	0.43074	0.24167
20	0.42073	0.23402
50	0.41354	0.22840
75	0.41669	0.22786
100	0.41304	0.22756
200	0.41952	0.23083

Batch size of 100 works well to produce the best results as seen. From the results it can be concluded that 75% of the times i.e. 3 out of 4 times given a movie, the algorithm is able to predict whether the user will like the movie or not. This algorithm finds correlations purely based on user ratings. Hence, it can be used as a good filtering technique on some other base algorithms like auto encoders etc where we curate a list of movies for the user and further sift out the movies the user will enjoy and recommend them.

CHAPTER 5

RESULTS AND DISCUSSION

This is a summary of the approaches that is used and the error metrics associated.

Table 5.1: Overall Results

Approach	**RMSE**	**MAE**
Mean	1.1257	0.9447
Random	1.8869	1.5129
Age Cluster	1.2199	0.8942
Genre Cluster	1.2413	0.9075
Cosine Similarity	1.2208	0.9355
SVD	0.9587	0.7522
Auto encoders	0.9409	-
Restricted Boltzmann Machine	0.4037	0.2191

The best values for each algorithm have been noted. Each algorithm has been run on the dataset multiple times, each time changing some parameters in order to report the best findings.

RBM gives the least error values however RBM doesn't fully qualify as a recommendation algorithm in the implementation done. It classifies whether the user would like a movie or not (binary prediction). The future scope of this algorithm would be to predict ratings 1 through 5 like other neural network models (eg: auto-encoders).

The best approaches for recommendation systems from our results are SVD , RBM and Auto Encoders (AE was run on u5 base, test of the cross validation set for MovieLens). These approaches seem to be the best chance at rivalling the best

algorithms of the surprise library. However, out-classing those algorithms is still very much a task in progress.

Restaurant recommendations using content based approaches gave commendable results however, in the absence of an evaluation metric like RMSE or MAE, it cannot be compared with predefined results. Instead, considering offline evaluation is a viable option for evaluation.

Till here, the approaches proposed have been compared to baseline/standard models in the field. Now, it is time to look at how these fare when compared to the best methods. [14]

A quick search on the internet reveals that the best RMSE achieved on the MovieLens 100K dataset seems to be 0.905 followed by approaches achieving 0.929 and 0.945

The best approach [15] is based around Graph Convolutional Matrix Completion which deals with recommender systems as a problem of link prediction in graphs. Based on the recent progress in deep learning on graphs, the authors propose a graph based AE (Auto-Encoder) Structure based on differential message passing. The authors claim that if supplementary information or structured data (social network) is provided to their models, it beats almost all state of the art models. They approach the problem by representing the movie data as a bipartite graph between user and items with edges having the value of rating.

Another approach[16] boasting the same RMSE of 0.905 does not make use of extra information unlike the previous method. The paper deals with inductive matrix completion. The authors state that most matrix factorization models- breaking down the matrix into the product of low-dimensional latent embeddings is transductive and cannot be generalized beyond the given training set and inorder to create inductive models, most people make use of additional information. The authors of the paper under discussion try to create an inductive state of the art recommendation model using only the data available by using a graph neural network (GNN) based on 1-hop subgraph pairs (user, item) and map these subgraphs to their ratings. They successfully implement this method and show that the inductive model they had trained on the MovieLens Dataset generalises very well and shows good results on the

Douban Dataset establishing the fact that the model can pick up user-item interactions it hasn't seen before.

The next approach[17] has an RMSE of 0.91 and is based on deep learning again to model the interaction between two sets. They state that the canonical representation of such interactions is a matrix with a property - permuting the rows or columns does not change the meaning of the encoding (exchangeability). They propose that such models be Permutation-Equivariant - same predictions across such permutations. The authors propose a parameter-sharing scheme and prove that it cannot be more expressive without violating the previous condition and show that the system has good generalization (Model trained on movies gave good predictions for music etc)

It should be noted that most of the approaches above either used Matrix Factorization methods or graphs and deep learning like Neural Networks , Auto encoders etc. The RMSE of our best approaches (SVD , RBM and AE) are not too far off from state of the art models.

CHAPTER 6

CONCLUSION

An attempt has been made to study existing literature surveys and research papers to obtain knowledge about the various types of recommender systems, their uses, the algorithms implemented and the challenges faced in each of them. Using this knowledge, implementation of some of the known algorithms as well implementation of our own techniques was carried out. The errors obtained for each algorithm and approach have been documented extensively.

Altogether, this paper has provided an elaborate overview to those entering the field of recommendation systems and arm them with more than sufficient intuition to select the best algorithm suited for their task.

6.1 Future Scope

One of the main goals moving forward is towards developing a recommendation algorithm that would decisively beat the existing baseline measures.

Apart from that, a few ideas for future scrutinization include:

1. Exploring the possibility of probing Genetic Algorithms for selecting population size and features and adopting at each iteration the sets with most potential.
2. How to be more informed while predicting the rating of movies? Trying out ensemble algorithms, a mix of different algorithms taking the majority vote in case of a mismatch in case of collaborative and content filtering algorithms.
3. Is it possible to recognize enough outliers (people who always give high ratings, people who like one type of movie, people who are swayed by trends) to radically change the performance accuracy?
4. Explore the accuracy of graph based models. How to intelligently find similar users? Can giving weight to different parameters play a role?

REFERENCES

[1] How to build a Recommendation Engine quick and simple: https://towardsdatascience.com/how-to-build-a-recommendation-engine-quick-and-simple-aec8c71a823e accessed on 16/05/2020 9:47 AM

[2]https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd accessed on 16/05/2020 10:00 AM

[3] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets:

History and Context. ACM Transactions on Interactive Intelligent

Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.

DOI=http://dx.doi.org/10.1145/2827872

[4] Khusro, Shah & Ali, Zafar & Ullah, Irfan. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. 10.1007/978-981-10-0557-2_112.

[5] Jiang Zhang, Yufeng Wang , Zhiyuan Yuan, and Qun Jin - (Personalized Real-Time Movie Recommendation System: Practical Prototype and Evaluation - 2020

[6] Khamphaphone Xinchanang, Phonexay Vilakone, and Doo-Soon Park - Movie Recommendation Algorithm using Social Network Analysis to alleviate Cold-Start problem - 2019

[7] Md. Akter Hossain, Mohammed nazim uddin - A Neural Engine for Movie Recommendation System - 2019

[8] Rahul Katarya, Om Prakash Varma-An effective collaborative movie recommender system with cuckoo search - 2017

[9] Alif Azhar Fakhri, Z K A Baizal and Erwin Budi Setiawan - Restaurant Recommender System Using User-Based Collaborative Filtering Approach: A Case Study at Bandung Raya Region - 2019

[10] Bhagyashree Basudkar, Shruti Bagayatkar, Meghana Chopade, Sachin Darekar - Restaurant Recommendation System Using Customer's Data Analysis - 2018

[11] Ravinarayana A, Pooja M C and K Raghuveer - Using Clustered Database for Food Recommendation System - 2016

[12] Sumedh Sawant and Gina Pai - Yelp Food Recommendation Challenge -2013

[13] https://www.kaggle.com/uciml/restaurant-data-with-consumer-ratings , accessed on 16/05/2020 10:25 AM

[14] MovieLens 100K Leaderboard - https://paperswithcode.com/sota/collaborative-filtering-on-movielens-100k Accessed on 20/05/20 10:00 PM

[15] Rianne van den Berg, Thomas N. Kipf, & Max Welling. (2017). Graph Convolutional Matrix Completion.

[16] Muhan Zhang, & Yixin Chen. (2019). Inductive Matrix Completion Based on Graph Neural Networks.

[17] Jason Hartford, Devon R Graham, Kevin Leyton-Brown, & Siamak Ravanbakhsh. (2018). Deep Models of Interactions Across Sets.

[18]https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c, accessed on 10/04/2020 10:00 AM

[19]https://towardsdatascience.com/k-nearest-neighbours-introduction-to-machine-learning-algorithms-18e7ce3d802a, accessed on 03/05/2020 12:30 AM

[20] Alka Lamba , Dharmender Kumar (2016). Survey on KNN and Its Variants

[21] Autoencoders - https://www.jeremyjordan.me/autoencoders/ accessed on 12/04/20 09:00 AM

[22] Understanding matrix factorization - http://nicolas-hug.com/blog/matrix_facto_1 accessed on 12/04/20 4:00 PM