Movie Recommendation System
Final Project Report
CSC-522 Automated Learning and Data Analysis
NC State University

Anusha Balaji
Department of Computer Science
NC State University
abalaji@ncsu.edu

Raghavendra Prasad Potluri
Department of Computer Science
NC State University
rpotlur@ncsu.edu

Nisha Krithika Nagasimha
Department of Computer Science
NC State University
nknagasi@ncsu.edu

Ruthvik Mandava
Department of Computer Science
NC State University
rmandav@ncsu.edu

Nithya Kumar
Department of Computer Science
NC State University
nkumar8@ncsu.edu

Sahithi Guddeti
Department of Computer Science
NC State University
sguddet@ncsu.edu

***Abstract*—** Given the emergence of Big Data and "Internet of Things", the convenience of suggestions and recommendations is a granted among internet users. Even seemingly simple suggestions for what items to buy at the grocery store can go a long way to build customer satisfaction. Recommender systems, in general, are a type of information retrieval technology that improve access and proactively suggest relevant items to users by considering their explicitly mentioned preferences and past behaviors. It lets algorithm developers predict what a user may or may not like in any given item set like books, movies, etc. It is like a black box analysis where the user's gender, geography, interests, past behavior is analyzed and items are suggested for them. Hence, recommendation systems help users find information about products and services they are interested in which they may not have thought of before. The objective of this project is that we are trying to build a movie recommendation system using the MovieLens dataset and Apache Spark with its inbuilt packages to recommend movies for users to watch based on their movie ratings. A widely-used approach called collaborative filtering is going to be used as well. This will help us filter for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Apart from this, we have implemented a kNN algorithm using Euclidean distance measure to predict the ratings of new movies to users.

Keywords- recommendation systems; collaborative filtering; ALS; Apache Spark; kNN

I. INTRODUCTION

Recommendation systems are part of a subclass of information filtering system that seek to predict the rating or preference a user would give to an item. A recommender system is one of the major techniques that handle information overload problem of Information Retrieval by suggesting users with appropriate and relevant items. Due to the immense amount of information about movies and ratings, a movie recommender is useful and convenient for users as well as many businesses especially web services like YouTube, Amazon, Facebook [1].

Movie Recommendation System helps in finding movies that a user would like. It requires users to rate movies so that users can be profiled. Using the user profiles, the system recommends movies. But in our project, we will be trying out a different approach by implementing a movie recommendation system using the MovieLens dataset and Apache Spark with its inbuilt packages to recommend movies for users to watch. The system will be based on film preferences using collaborative filtering of members' movie ratings. Collaborative filtering is a widely-used approach to filter for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Since many movie applications on the market don't have an efficient tool to predict user behavior, we hope our movie recommendation system will be helpful.

By implementing a movie recommendation system, we aim to provide relevant and useful movie suggestions to our users based on their past ratings and ratings given by 'similar" users. "Similar" users are users that have given similar ratings to similar movies in the past like the current user. This will make our application more convenient for our users to not only watch movies they want, but also what they would want to watch based on their past behaviors. Therefore, the system would serve to improve movie applications' usability by making their services more user-friendly. Challenges [2] to this project would be:

1. Lack of Data – A recommendation system requires a lot of data to make effective recommendations.
2. Changing Data – It not only requires us to keep track of past behaviors but also current trends

which keep on changing with time. Also, since user preferences can change over time, our system must continually keep up with the changes.

II. Related work

In our research, we came across multiple effective ways to build recommendation systems including k-means clustering, decision trees based classification, and k nearest neighbors classification. As we further explored through more research and code implementation, we observed several strengths and weaknesses of each of these approaches however.

One of the first methods we were interested in at the beginning of the project was decision trees. Decision trees, in general, are a popular choice to implement recommendation systems because of their efficiency, interpretability, and flexibility in handling various attributes as input. We quickly realized decision trees have several benefits for recommendation systems, they are very complex to build as the number of attributes increases.

Traditionally, decision tree algorithms have only been as useful as predicting rating for one movie at once. In other words, the system had to traverse the tree every time to predict ratings for each movie of interest and then make recommendations based on the predicted ratings by the tree. In order to minimize on the run time complexity, the system can build many such trees either for each movie or for each user. Since our dataset is quite large, with thousands of users and movies, scaling became an issue of concern and we were hesitant to proceed with decision trees as our backbone technique.

However, a new method for constructing decision trees for recommendation systems is proposed by Meisels *et al* in their paper [3]. The proposed method includes two new major innovations including modifying the decision tree to produce lists of recommendations at its leaves and a better splitting method when constructing the tree. Though we decided not to proceed with decision trees because of runtime and memory complexity, we liked the improvisation on predicting several ratings and presenting the recommendations at once to the user using collaborative filtering. As presented in this paper, collaborative filtering can be very useful in finding similar users and basing our recommendations on the behaviors of such “similar” users. Several projects and papers we explored featured collaborative filtering as a very useful technique in implementing recommendation systems. Further details on how we employed collaborative filtering in our project follows in a later section.

On our mission to find “similar” users to make recommendations to our user, we began by utilizing unsupervised learning techniques. Specifically, we considered k-means clustering because it has several advantages including simplicity and performance. However, while doing a brief implementation of k-means, we found it challenging to find a distance measure that makes logical sense. Though we used Euclidean distance initially, we realized it may not be a useful metric to interpret the distance between two users, for example. Therefore, we felt the need to consider other techniques to better factor in user ratings and finding “similar” users. [4]

In that aspect, we wanted to consider some supervised machine learning in our project. By using a supervised learning algorithm, we can build a model for our system to make better predictions based on proximity measure. K-nearest neighbors is a technique in which the distance and similarity plays a major role. The distances between all the instances are calculated and the k nearest neighbors can be used to make a decision. We implemented k nearest neighbors technique in our system so that we can use the information of other users to predict ratings for movies. There is also a problem of dimensionality that comes with this approach [5]. Therefore, we explored through techniques like clustering to reduce space. Because our problem statement has emphasis on the other users’ choices, the similarities of the k nearest neighbors can be considered through distance. The user with the smallest distance contributes the most to the decision. This method is proposed in the paper [6].

The other related work [7] gives us the overview of various data mining techniques used in the context of Recommender Systems. It talks about preprocessing methods such as sampling and dimensionality reduction and classification techniques, including Bayesian Networks and Support Vector Machines. It introduces us to Recommender Systems and present cases where they have been successfully applied.

III. Methodology

Unsupervised Learning using K Means

One of the simplest unsupervised learning algorithms is the K-means that solves the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori.

It is a four step process which involves the foll. steps.,

i) Construct a user-movie matrix mapping the user Id, movie Id and rating

ii) Cluster the movies into genres using K Means

iii) Obtain the movies which a user has rated above a certain threshold

iv) Predict the top 5 genres for the user based on the movies obtained from step iii)

Collaborative filtering using Apache Spark

We tried to use a few existing ideas and implementations of Movie Recommendation Systems. One of the most widely used approach in Movie Recommender Systems is the Collaborative filtering. Collaborative filtering methods are based on collecting and analyzing a large amount of information on users’ behaviors, activities or preferences and predicting what users will like based on their similarity to other users. One key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself. [8] In a

narrow perspective, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. [9] So, a user who has liked some movie in the past and given high recommendations will like the movie in the future as well. Also, the users will like only similar kind of movies as they liked in the past. Here is the workflow of a collaborative filtering system:

1. A user rates movies. These ratings are viewed as representation of the user's interest in the movies.
2. The system matches the user's ratings against other users' ratings and finds the people with most "similar" tastes.
3. With similar users, the system recommends movies that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of a movie)

More technically from the system perspective,

1. Look for people who share the same rating patterns with the present user (the user whom the prediction is for).
2. Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user [10]

Collaborative filtering is implemented using the ALS (Alternating Least Squares). Spark MLib library implements it and ALS has the following parameters. [11]

numBlocks - the number of blocks used to parallelize computation (set to -1 to auto-configure).

rank - the number of latent factors in the model.

iterations - the number of iterations to run.

lambda - specifies the regularization parameter in ALS. The parameter is generally scaled based on number of ratings so that lambda is not dependent on the dataset. [12]

implicitPrefs - specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data.

alpha - a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations.

Our goal is to find out the best rank i.e number of latent factors of the model that forms the best recommendation engine. Hence the dataset is divided into training set, validation set and test set. For different values of rank such as 2,4,6 etc, validation set is used to predict the ratings from the model and sum of squared errors is calculated which forms as a performance metric for this project. For rank = 4, SSE is minimum and therefore finally is used to form the model again to predict the ratings on the test set and report the SSE of the model.

Supervised Learning using kNN

As said before, collaborative filtering helps to find out the results based on other users that have similar interests to the user that queried instead of using his/her own ratings which are very limited. Why KNN? KNN finds out the nearest K neighbours based on proximity measure selected. Our idea is that when there is need to predict rating for a movie that the user hasn't watched, we first pool out the users who had watched that movie, find out the k nearest neighbors from the pool to the user in query, and average out the ratings given by the k neighbors. Preprocessing and proximity measures are explained in the next section

With clustering added before performing KNN

Calculating the distance matrix and maintaining it for too many users (671 for our dataset and one million users for netflix data) is too time taking and inefficient. Also every time a movie rating is to be predicted, there needs sorting of distances to find out the nearest K neighbors. Therefore clustering either based on genres or distance measures would help to decrease time complexity by just allowing the engine to compute or sort the distances inside the cluster alone

IV. Experiments and Results

A. Experimental setup

i) PoC for K Means clustering to predict genres
ii) Collaborative filtering in Python with Spark's Alternating Least Squares implementation (from Spark MLlib)
iii) kNN implementation in R to predict ratings to a user

B. Datasets used

For our recommendation system, our training and testing set will be based on the dataset provided by MovieLens (https://grouplens.org/datasets/movielens/). For the experiments, we are using the small dataset [9] which has 100,000 ratings and 2,488 tag applications applied to 9,125 movies by 671 users. All ratings are contained in the file ratings.csv and movies are in movies.csv. Each line of ratings.csv after the header row represents one rating of one movie by one user, and has the following format: userId, movieId, rating, timestamp.

C. Implementation details

Unsupervised K Means Clustering in Python

1. Loading the dataset
2. Cluster movies based on genres using the function K Means provided by the sklearn library
3. Compute the user-movie matrix based on ratings
4. Obtain the top 5 genres for each user based on his ratings.
5. Recommend genres for all the users.

The code can be found here [13]

Collaborative Filtering

Preprocessing done

1. Loading and Parsing datasets
2. Parsing the movies and ratings files yields two RDDs:
   a. For each line in the ratings dataset, we create a tuple of (UserID, MovieID, Rating). We drop the timestamp because we do not need it for this recommender.
   b. For each line in the movies dataset, we create a tuple of (MovieID, Title). These are used at the final step to put out Movie names that are recommended instead of just movie IDs
3. Randomize the data
   Data is randomly split into training set, validation set and test set

Construction

1. Selecting ALS parameters using the small database
   a. Determine the best ALS parameters
   b. After training, our performance metric SSE is calculated using validation set and the final parameters that gave lower SSE are selected and used to form the model that is used to predict ratings for test set and report the SSE of the model..
2. Making recommendations
   a. Either a single movie is queried for a user to predict the ratings and sort it to recommend top rated movies by the model
   b. Or the package has a function to recommend top n movies which are cached until any change in the data or parameters.

The code can be found here [14]

K Nearest Neighbors without clustering and with clustering in R

Preprocessing done

1. Load the datasets from csv to data frames
2. Divide the dataset into training set, validation set and test set
3. Using training set, a matrix of n users * m movies is formed where each cell gives the rating or NA if the user hasn't rated
4. Need for standardizing the ratings:
   a. Each and every user have their own way of rating.
   b. A user who liked a movie very much might give ten out of ten points while another user has his/her rating to a movie as five.
   c. Therefore before finding out the distances between users, we standardized the ratings of each and every user
   d. We also store each user mean and standard deviation in a vector which are needed to give a final rating back while computing SSE

Implementation

1. Distance matrix is formed with different similarity measures such as Euclidean, Manhattan and Cosine Similarity.
2. The the ratings are predicted for validation set for different values of K using the movie user vector that has watched users list for each movie ID, and from distances retrieved only to those users, predicted rating is calculated by averaging out ratings given by top K similar users
3. The above implementation gave approximately equal SSE that we obtained from machine learning library sklearn
4. But the time complexity is huge because there needs a sorting of distances to all users in the pool selected.
5. We therefore applied basic K-means clustering to the users based on the distance matrix that we calculated in the step 1. Now the distances for any user are retrieved from the other users present in the current user's cluster and then top 'K' are chosen

The code can be found here [15]

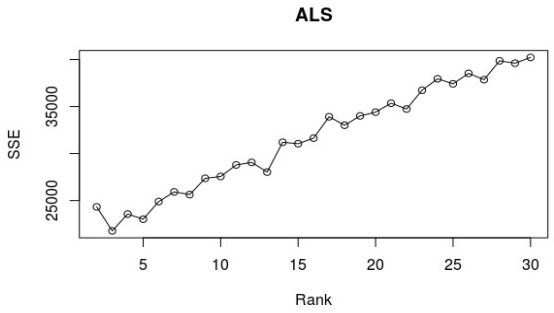
Results

Training Set - 60% - 60000 ratings, 671 Users, 9125 movies
Validation Set - 20% - 20000 ratings
Testing Set - 20% - 20000 ratings

Sum of squared errors reported in the following graphs are obtained on the validation set for different parameters chosen

1. *Collaborative Filtering*

From the figure below, we have lowest SSE for rank 3 and therefore recommend movies with chosen parameter.



For the above dataset we add a new user and some ratings as follows:

add_user=sc.parallelize([(0,260,4), (0,1,3), (0,16,3), (0,25,4), (0,32,4), (0,335,1), (0,379,1), (0,296,3), (0,858,5), (0,50,4)])

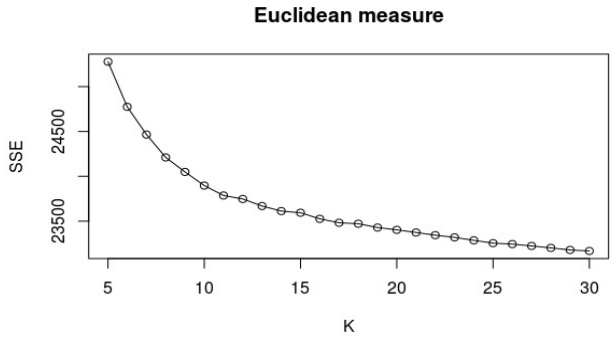
To get top 5 recommended movies from the model as:

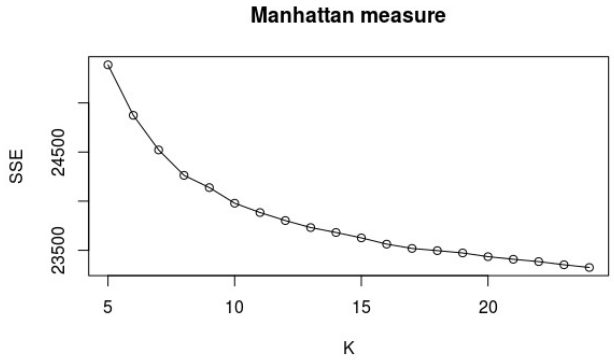
[u'Shoot the Moon (1982)', u'Bride & Prejudice (2004)', u'My Family (1995)', u'No Holds Barred (1989)', u'Bugsy (1991)']

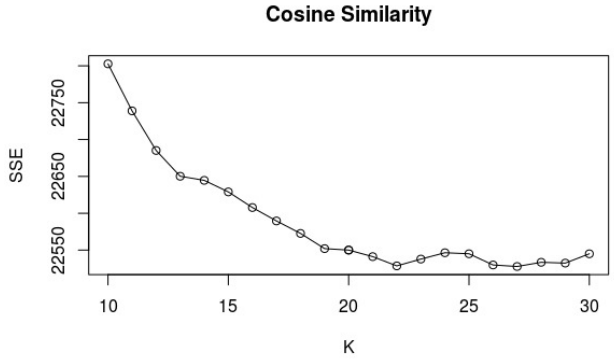
2. KNN - with different proximity measures

From the following graphs it is intuitive that after K=20, we get approximately equal SSE for the randomized dataset which explains that for this dataset, the predicted ratings when taken from average of all users is nearer to the actual rating. KNN worked fine as the SSE collected for different K and as well as for all proximity measures are equal to the ALS technique that is widely used.

Therefore future work is to make KNN scale to bigger dataset and bring clustering into picture to make it time efficient. As a stepping stone, we clustered the users based on distance metric and made code run in seconds but ended up with similar SSE.







V. CONCLUSIONS AND FUTURE DIRECTIONS

We have gained an insight into how a movie recommendation system works and the parameters, which include user ratings. Working with a large MovieLens dataset and using unsupervised learning method (KMeans) to understand the recommendation system, collaborative filtering and supervised (kNN) learning to implement the system, we learned to use Apache Spark and its inbuilt packages to handle the large dataset in building a decent movie recommender.

Our future aim is to build a movie recommendation model with sample data from the dataset and implement collaborative filtering technique with clustering techniques and decision tree model in python.

ACKNOWLEDGMENT

We would like to thank Dr. Raju for providing us with an opportunity to do this project and his valuable feedback. and guidance. We also thank the authors of the following resources for providing specific techniques to do our project.

REFERENCES

[1] http://dataaspirant.com
[2] http://readwrite.com/2009/01/28/5_problems_of_recommender_systems/
[3] A Decision Tree Based Recommender System. By Gershman, A., Meisels, A., Lüke, K.H., Rokach, L., Schclar, A. and Sturm, A., 2010, June. In IICS (pp. 170-179).
[4] http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap8_basic_cluster_analysis.pdf
[5] Effect of Dimensionality on kNN
[6] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5408784
[7] Data Mining Methods for Recommender Systems In Recommender Systems Handbook by Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, Josep M. Pujol edited by Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor. (2011), pp. 39-71.
[8] https://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering
[9] https://en.wikipedia.org/wiki/Collaborative_filtering
[10] https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
[11] https://link.springer.com/chapter/10.1007%2F978-3-540-68880-8_32
[12] http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
[13] https://github.ncsu.edu/rmandav/RecommendationSystem/tree/master/PoC_KMeans
[14] https://github.ncsu.edu/rmandav/RecommendationSystem/blob/master/MovieRecommend.ipynb
[15] https://github.ncsu.edu/rmandav/RecommendationSystem/blob/master/KNN-classifier.R