Comparative Analysis of Recommender System Algorithms Using API Dataset

Sandeep Kumar Ragila

ABSTRACT

Recommender systems are important for e-commerce ecosystem. A lot of applications use recommendation systems to recommend their products to users. Any application which has the information about the interaction between the user and their items, user likings, user ratings, user browsing history, text descriptions of items, can use these recommender systems to help its users to find their interested products. Some of the well-known applications are Amazon, Ebay, Netflix, Youtube, Spotify, etc.,

In this project, we will be using the concept of recommendation systems to recommend APIs to users. Also, we will compare the performance of different recommender algorithms and determine which algorithm works better with our dataset.

1. INTRODUCTION

There are several techniques to implement recommender systems. The popular three methods are the Collaborative Filtering, Content-Based Filtering and Hybrid recommender systems.

The Collaborative filtering technique is widely used because of the simple design and the procedure it takes to achieve recommender systems. It only takes user data, user likes, behavior and the similarity between users into consideration. Collaborative filtering needs a large amount of data in order to analyze the behavior of users and find their similarities. The advantage of using this approach is that it does not need to analyze the data associated with users or items. Also, it does not need to understand the type of item or the understanding of data behind it. The assumption behind collaborative filtering is that people who liked some items in the past might like the similar items in the future.

Content-Based filtering is also a popular technique used to find the similarities based on the content provided with the item or the user preference. The description of the user or the item description is used to analyze the keywords involved in the text and map these keywords to get the similarities. This approach needs techniques such as information retrieval and natural language processing. Some of the leading algorithms in this field are tf-idf algorithms, bag-of-words model etc.,

Hybrid recommender systems can be explained in terms of the combination of the Collaborative and Content-Based filtering techniques. We can combine the content-based after performing collaborative filtering or the other way. But, for this project, we will be focusing on the Collaborative filtering techniques and the different algorithms involved in the approach. We will be dealing with four different algorithms, namely, Random, Popular, Item-Based and User-Based recommendations. Also, we will look at their performances on our dataset and evaluate the algorithm based on some evaluation metrics.

A more detailed explanation is included in further sections. Project goal will be listed in the upcoming section. Dataset description section will have the details of the dataset and the structure of it. The following section will explain more about the data and some data analysis. Then we will look more into collaborative filtering and different steps involved to implement the algorithms. The future sections will discuss the experiments and results, t-Test evaluation and some Future work.

2. PROJECT GOAL

The goal of the project is to recommend APIs to developers who might be interested in. Developers use a lot of APIs and might not be aware of the APIs which are similar to what they have used. They search for APIs using different keywords. Although this approach is efficient, it is difficult when the dataset is very huge. To resolve this issue, I would like to analyze the behavior of the developer and recommend a new set of APIs which a user might be interested. I would also like to compare the performances of different recommender algorithms.

3. DATASET DESCRIPTION

For this project, the dataset was gathered from various sources and the major part of it was from programmableweb.com [1] website. ProgrammableWeb is a website which has all the information about the APIs. It is the world’s leading source in providing the information and news of web-based APIs. Users can sign up into the website and get the information of desired APIs. The website will also store the information of the users and their interested APIs. Although users can get the information by searching through its repository it might not be a feasible option for the users as they may
not always know what to search for. With this limitation, we can add a recommendation functionality where users will be shown with APIs they might be interested in based on their past behavior. To achieve that the dataset with the following information is extracted from the website.

Each data record has

- **Id**: A unique ID for each record
- **Name**: Name of the API
- **Description**: A short text describing the API
- **Primary Category**: A word that determines the API category
- **Secondary Category**: This is further used to categorize the API
- **Followers**: Usernames of the followers
- **Count of Followers**: The number of followers an API has

Figure 1 shows visually how the information is presented on the website. As we can see the Facebook API has a description of the API, count of followers, category and the user names of the followers.

The dataset collected was in CSV format, all the users are comma separated for each API, So, users are repeated for each API. To avoid this redundancy and get a well-represented dataset a Java program has been written to count the number of users and map them to the APIs. The program will take the original dataset and split each record based on the delimiter. At each stage, the program will check if the user is already recorded, if not the user is created and the API is added to the list or else the API will just be added to the existing user.

These are the main characteristics of the dataset that will tell us the size of the dataset. The dataset collected was

---

**Figure 1: API information**

---

**Facebook API - Followers**

The Facebook API is a platform for building applications that are available to the members of the social network of Facebook. The API allows applications to use the social connections and profile information to make applications more engaging, and to publish activities to the news feed and profile pages of Facebook, subject to individual user privacy settings. With the API, users can add social context to their applications by utilizing profile, friend, page, group, photo, and event data. The API uses RESTful protocol and responses are in JSON format.

**Figure 2: Users Sorted by Following Count**

---

**Figure 3: API vs Count of Followers**

---

4. **DATA ANALYSIS**

To get a good understanding of the dataset we need to look into the dataset more deeply. Below are some of the statistics collected from the dataset and some visualizations.

- **Total number of APIs**: 12922
- **Total number of Users**: 39596
- **Total number of Categories**: 389

---

Since the dataset is very large and the user-item interaction is very less for most of the data we need to set criteria
to reduce the dataset. The reduced dataset will better explain the behavior of the users which will be useful to gain accurate results. The base criteria for our dataset lie on two characteristics that are the User and the Item. I have chosen average as criteria to reduce the dataset. The two criteria’s finalized to reduce the dataset are

- For the User, the Average number of APIs each user follows should be atleast 5
- For the Items, the Average number of followers each API has should be atleast 7

With the above mentioned criteria the finalized dataset has the following information.

Dataset after reduction:
- Total number of Users: 904
- Total number of APIs: 2646

5. COLLABORATIVE FILTERING

Collaborative Filtering[7][6] is a popular technique used to build the recommendations. It uses the information of users and their interaction with the items to build the models. Using this information it will take a user-item binary matrix to calculate the similarity between different users or items and predict new items. This is the reason why it is called collaborative filtering. There are two types of collaborative filtering(CF) techniques. Memory-based CF and Model-based CF. I will be focusing on the Memory-based collaborative filtering in which the most popular neighborhood algorithms are the User-based CF and Item-based CF.

We will be comparing four collaborative filtering algorithms, that are: Random recommendations, Popular recommendations, User-based recommendations and Item-based recommendations[4]. In Random recommendations, we will choose a new set of random items and recommend them to the user. In Popular recommendations the items or the APIs which are not already in the users following list are chosen. These APIs are sorted according to the count of the followers. Top M recommendations from these APIs are recommended to the user. The idea behind Item-Based recommended suggestions is that if two items have been rated similarly by a set of users then the remaining users might also rate the item similarly. We calculate the similarity measures of all the items with respect to each other and recommend most similar items to a specific item. With User-Based the idea is that if two users have rated a set of items similarly then they might rate the other items similarly. We calculate the similarity between all the users with respect to each user. We then take the users who are similar to a specific user and recommend the items which are not in the target user following list. To achieve these algorithms R language is used. Recommender Lab is a package which is used to implement these algorithms.

There are four steps involved in memory-based CF as shown in Figure 4.

1. Generating a User-Item binary matrix
2. Choosing a similarity measure
3. Training the Model
4. Evaluating the Model

Figure 4: Collaborative Filtering Steps

5.1 Generate User-Item binary matrix

With the finalized dataset we need to create a user-item binary matrix which will be used as an input to the algorithms. I have written a program to take the dataset as an input and generate the matrix as the output. The size of the matrix created:

904 users * 2646 APIs = 2391984 cells

This matrix will have users on the y-axis and APIs on the x-axis. Each cell in the matrix will be a binary value i.e., either 0 or 1. The binary value 1 means that a user follows certain API. But if the cell has a value 0 it is known as missing value. In the next section we will see what a missing value is. When I looked into the matrix I found that most of the cells in the matrix are 0’s. This behavior is called Data Sparsity.

Sparse Matrix is a concept where you have more number of missing values(0’s) than the positive values(1’s). If the matrix has more number of positive values then the matrix is considered as the dense matrix. According to this, the matrix which we have generated had 99.4% of data sparsity. Which means that we have less than 1% of user-item interaction. This behavior is common because, if we take an example of Amazon user-item database, the number of products in amazon.com is way too larger than the user base. Also, a user might not have interaction with all kinds of products. For this reason, 99.4% sparsity is considered good data.

5.2 Choosing a similarity measure

As we have created the user-item binary matrix, the next step is to choose a similarity measure. Similarity measures are used to measure the similarity between different users or items to predict a new item. There are different similarity measures available. The three major similarity measures are:

1. Pearson Coefficient
2. Cosine Similarity
3. Jaccard Index

Pearson Coefficient and Cosine Similarity are better applicable to numerical data i.e., real ratings or the ratings on a scale. Jaccard Index is a best suited for binary ratings. As our dataset has binary ratings we will be using Jaccard Index as our similarity measure. Jaccard Index does not take missing values while calculating the measure. It is the measure of the size of the intersection divided by the size of the union.

\[
\text{JaccardIndex} = \frac{|A \cap B|}{|A \cup B|}
\]

Let us see this with a small example. Below is the simple user-item interaction table.
In the above table, each column corresponds to an item and each row is a user. Cell value 1 represents that a user follows specific item. To calculate the similarity measure for user1 and user2 using Jaccard Index we need to take the sets of each user.

\[
\text{User1} = \{1,1,1,1\} \\
\text{User2} = \{0,1,0,0\}
\]

If we take the intersection of user1 and user2 we will have just 1 item in the set whose length is 1. The union of the set is all 1’s and the length of the set is 4.

\[
\text{Jaccard Index} = \frac{1}{4} = 0.25
\]

The above example shows us that Jaccard Index takes only positives into account and not the missing values. This feature is important to us because, if we take missing values into account we have an uncertainty in the similarity measure. We can see the missing values in three ways: A user might not be interested in the item, a user might not know about the item, a user might not need the item. Therefore, we need a measure which does not include missing values to calculate the similarity. Hence, we will choose Jaccard Index.

### 5.3 Training the Model

In order to train the model, we need to split the dataset into training and test modules. There are several techniques to split the dataset each with its own importance. We will focus on three important data splitting techniques and train the model with them. I will then compare the performance of the algorithms with each technique to decide which one works better with our dataset. The three techniques we will be using are:

1. **Percentage Splitting**: In this process, we divide the dataset into a certain amount of percentage for the training purpose and the remaining for the testing. The percentage which we choose is specific to the dataset and the model as different values might yield different results. We need to choose the best values which give us the highest accuracy. Generally, 66% of training and 34% testing is considered ideal split for most of the datasets.

2. **Bootstrap Sampling**: Bootstrapping is a technique to resample with replacement. The samples created after resampling can be used as training sets and the users not in the training set are used for testing. These samples may have duplicate information as we are replacing an item chosen. Selection of these items is random so we can have a large number of samples. The number of samples we create is a variable which we can set. Using this approach is highly recommended when we have a small dataset as we can resample and create a large training set.

3. **k-fold Cross Validation**: In the k-fold cross validation technique we divide the total dataset into k folds or chunks of equal size. Each time we keep hold of a single fold for testing and use remaining k-1 folds for training the model. We repeat this process k times. At the end, we perform an average of all the k results to get the overall measure for k value. The advantage of using this technique is that it ensures each user is present in the training set at least once. k value is chosen according to the error estimates.

### 5.4 Evaluating the Model

After creating the model the final step is to evaluate the model using some metrics. We will look at different evaluation metrics\[5\] to evaluate the recommender algorithms and see which metric fits our project.

1. **Mean Squared Error (MSE)**: It is the average of the squares of the error. MSE is useful when the data is numerical. It will calculate the difference between predictions and true ratings and take an average of the result. Different recommender systems use numerical data which is measured on a different scale. Hence, MSE will try to normalize the data and provide the error estimates in the form of percentages. MSE, therefore, cannot be used as a metric for our algorithms.

2. **Precision**: \[2\] is the fraction of retrieved instances that are relevant. In the retrieved items we might have false-positives which are also included. Here, missing values are considered to be false-positives. Precision can be better explained with the following formula.

\[
\text{Precision} = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}\}|}{|\{\text{retrieved}\}|}
\]

But, we cannot use precision as our evaluation metric. The reason being, as the denominator in the above equation, has total retrieved which might include the missing values. As discussed previously missing values are highly uncertain and for that reason it might not be the true measure.

3. **Recall**: It is the fraction of relevant items that are retrieved. The recall is a true measure as there are only true-positives and false-negatives included. It is good compared to precision as false-negatives are better than false-positives. The formula for Recall is as follows.

\[
\text{Recall} = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}\}|}{|\{\text{relevant}\}|}
\]

In the above equation, we have relevant items from the retrieved as the numerator, divided by the total number of relevant items. In this case, we only take the positive values into consideration and missing values are ignored. Hence, Recall is the evaluation metric we will be using to evaluate the performance of algorithms.

As we look closely into the metric there is a flaw in the metric. If we recommend all the APIs in the dataset and calculate the recall we would always get a recall value of 1. For example, if we have 10 items in total
in our dataset and a user follows 5 items. If we recommend all the 10 items, the total number of relevant from the retrieved are 5 and the relevant items known are 5. So the recall value is 1 no matter how many APIs a user follows. But, is this a good system? No. A better system is when we recommend fewer items in which most of the items are relevant. For this reason, we will be using a modified version of the metric. That is Recall @ M (Recall at M). Where M is the number of recommendations.

\[
\text{Recall}_@M = \frac{|\{\text{relevant}\} \cap \{\text{M items retrieved}\}|}{|\{\text{relevant}\}|}
\]

As we increase the M value i.e., as we increase the number of recommendations we will get a better recall value. But, after a certain point, the growth will be stable because the number of relevant items might not be very high compared to the previously retrieved items. So, we should choose an M value where the recall is high.

6. COMPARATIVE ANALYSIS

Below are the experiments conducted on all the four algorithms listed on three different data portioning techniques. Each graph is plotted between Precision and Recall. The X-axis is precision and Y-axis is recall.

6.1 Percentage Splitting

As discussed above, percentage splitting is a technique to divide the dataset into training and test modules. Different values of percentages have been tried to check the accuracy of the model. The best results were yielded with the below parameters.

- Training Set: 90%
- Testing Set: 10%
- Given: -1
- Number of runs(K): 1
- Number of recommendations(M values): 100, 150, 200, 250, 300, 350, 400, 450, 500, 520, 540

The first two parameters are self-explanatory, the third parameter is Given. In order to test the performance of the model, we need to withhold some of the known items to test against the algorithm. For this reason, the depending upon the given value the algorithm randomly chooses ‘x’ (Given value) number of items and does not include in the training set. Now the model is tested and checks if it predicted the withheld item values. This is how the accuracy of the model is tested. A Given value of 1 means that the model will keep hold of 1 item from the dataset. Value -1 means that each time a random number of items are withheld. For the percentage splitting technique, we have withheld 1 item to test. The next parameter is the K value which is the number of runs the model performs. The last parameter is the number of recommendations(M value) in the Recall @ M measure. We have tested with a different set of M values and the above chosen values were yielding the best results. The resulting graph with above mentioned parameters is shown in Figure 5.

As we see in Figure 5 User-Based recommendations outperformed other algorithms. The highest accuracy we have achieved is 61.53%. Each data point in the graph is an average measure of the model obtained with certain M value. As we increase the M value at a certain point i.e., at M value = 500 the graphs starts to behave linearly. This is because, no matter how much you increase the number of recommendations after a threshold the user might get most of the recommendations which are not of his/her interest.

6.2 Bootstrap Sampling

Bootstrap sampling is a technique of sampling with replacement. The parameters used to obtained the result for Bootstrap sampling are

- Number of samples: 10
- Given: -1
- Number of runs(K): 10
- Number of recommendations(M values): 100, 150, 200, 250, 300, 350, 400, 450, 500, 520, 540

I have tried a different number of samples i.e., 5, 10, 20, 30 and tested with given values 1, -1, 10. The best results were obtained with the values 10 for k and -1 for given. Each sample will have random items included which may be duplicated. Figure 6 show that the User-Based recommendations perform well but not as much as compared to the percentage splitting technique. This is because, as the data samples have a lot of duplicates when we resample the results may vary each time. As we run the model again the results might be different but not significantly different. The highest accuracy we have achieved using Bootstrap sampling is 56.66. Item-Based recommendations performed well initially but as we increased the number of recommendations it behaves similarly to the Random recommendations. This is the drawback for the Item-Based recommendations. As we increase the number of recommendations the accuracy will remain constant but the system will perform poorly.
6.3 k-Fold Cross Validation

k-Fold Cross Validation is a technique of dividing the entire dataset into k equal sized folds. This is different from Bootstrap sampling in a sense that cross validation does not have duplicates. The parameters to obtain the Figure 7 are

- Number of Folds: 10
- Given: -1
- Number of runs(K): 10
- Number of recommendations(M values): 100, 150, 200, 250, 300, 350, 400, 450, 500, 520, 540

k-Fold Cross Validation clearly performed better than other techniques with our dataset. This behavior is expected as opposed to using a single run percentage splitting or random resampling techniques. Because, if you have a single fold with 90% training we are not left with a lot of data to test the model. Also, the remaining dataset may be biased towards the model giving us the variant results. This approach can be mitigated using the k-Fold Cross-Validation as each user will at least be once in the training set. Each fold will be evaluated separately and the average of the results will be taken for each fold. This process is repeated K times for K folds. Each time K-1 folds will be in training and 1 fold is given to testing. With this approach, the variance is not high and the results obtained will not be biased. Thus, K-Fold Cross Validations works better than other techniques.

In Figure 7 User-Based gained better accuracy than the other algorithms compared. The model was tested with k values 5, 10, 20, 30 and the given values of 1, -1. The highest accuracy obtained is 69.42%.

7. T-TEST

t-Test is a statistical measure to find out if two sets of data are significantly different or same. It can be calculated using the following formula.

\[
t = \frac{\text{Mean}(X_1) - \text{Mean}(X_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}
\]

Where t is the t-test value, Mean(X1) is the mean of the dataset X1, Mean(X2) is the mean of dataset X2, s1 and s2 are the standard deviations of dataset X1 and X2 respectively and finally n1 and n2 are the number of values in the datasets.

With this formula, we have calculated the t-value between the techniques Bootstrap sampling and Cross-Validation. As we know that each datapoint is the average value of the model we are taking into account the highest accuracy we have achieved in the Figures 6 and 7. The t-Test has been conducted between the two points and the results yielded a t-value of 1.772 and a p-value of 0.092 for TPR(True positive rate). For the FPR(False positive rate) we have got a p-value of 0.99 and t 0.

For TPR, since the p-value is 0.092 we can reject the null hypothesis with a confidence of 90.8% and conclude that the two techniques are significantly different with respect to TPR. For FPR, the p-value is 0.99(t 0), hence, we can conclude that the two techniques are almost similar.

8. TECHNOLOGIES AND TOOLS USED

- Java
- R
- RecommenderLab
- Tableau
- Excel
9. FUTURE WORK

In the technique discussed above, as the user or item base increases the dataset increases widely. With the increase in the dataset, the user-item binary matrix would be significantly large which makes the computation complex. Due to this computational complexity, we can make use of parallel computing by using multiple systems and get the recommendations in real time.

As we have seen one of the techniques to develop recommender systems, there are several other approaches. Collaborative filtering is categorized into two types. Neighborhood-based algorithms which are the Item-Based and User-Based as discussed in this report the other approach is the Latent factor models. The idea behind Latent factor models is that users like items based on some hidden factors. We need to use these hidden factors to find the similarity between uses. This approach is also called Matrix Factorization[3].

The other area of development is the content based filtering where we can use information retrieval and natural language processing techniques to find the similarities between users. The dataset we have collected has the metadata about each API we can use this data and perform the text analysis.

10. REFERENCES