User Based Collaborative Filtering Recommender System

Definition

Project Overview

Recommender systems have become a well studied problem now, and heavily integrated in many websites and apps we use daily,these systems tend to help users in their everyday decision making process from buying items to music listening by recommendations tailored differently to each user taste.
They are implemented in famous e-commerce websites like Amazon's famous "customers who bought this item also bought" section, in music we find Spotify's "Discover weekly" playlist which recommends a playlist curated differently to each user based on his taste, and even in movies where companies like Netflix recommend new movies you may like to watch,even made a 1M prize competition for improving its own recommender system, recommender systems have proven to be very important for these companies as they are an essential part for increasing their revenues and retaining customers.
There are different types for Recommender systems,this project is about implementing User Based Collaborative filtering recommender systems for movies, this type of systems is based on a simple idea which is people tend to ask their similar in taste friends for movies to watch, same idea implemented here but on a large scale of users, we find similar users in taste, we see what other items they like then recommend these items.
Item-based collaborative filtering is another famous type of collaborative filtering recommender systems called which is not the main goal here, but the idea is also the same,instead of finding similar users we find similar items to the current item.

Problem Statement

Given a dataset (MovieLens dataset) of movie ratings and the ID of each user our goal is to:

1. Identify similar users(nearest neighbours) that had similar preferences to the user we want to recommend movies to.
2. For every movie the user has not yet seen, a prediction is computed based on the ratings for these nearest neighbour users
3. Predictions produced are rounded to the nearest integer.

There are two assumptions here

1. If users had similar tastes in the past they will have similar tastes in the future
2. User preferences remain stable and consistent over time.

Metrics

I've worked with two metrics in this problem

1. Since this is a mainly a classification problem (Rating from 1 to 5), i used accuracy as a start to score the results of the classifier, accuracy is the number of correctly calculated movie predictions/the number of all predicted movies(right or wrong)

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

   After using accuracy i found that it doesn't tell me everything i need about evaluating the The data.
   In other words, accuracy gives a percentage of correctly rated movies, but what about ratings that are not a 100% correct but close to the correct ones they are still valid in the context of our problem, that was the problem with accuracy i wanted to know how much my predictions are far from the actual ratings that's why i calculated RMSE next.

2. Root mean squared error is then calculated, the reason to use it was to know the distance between the model's predicted ratings and the actual ratings and if the distance was less than a certain target/threshold then the model is accepted and if not further enhancements should be made.The difference here from accuracy is that accuracy calculates the percentage of right predictions, but ratings are not like classifying labels like in other problems, meaning that in other problems if we classified something to a wrong class then it's wrong, but in this problem predicted rating can be not equal to the true one but if it's close then it's acceptable as well, that's why i chose RMSE for this problem.RMSE is simply calculating the square root of the sum of differences between true and predicted ratings squared.

$$\mathbf{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Analysis

Data Exploration & Exploratory Visualization

Here the dataset used is MovieLens dataset.

	user_id	age	gender	occupation	zipcode	movie_id	title	release_date	video_release_date	IMDb URL	...	Horror	Musica
0	1	24	M	technician	85711	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	...	0	0
1	1	24	M	technician	85711	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	...	0	0
2	1	24	M	technician	85711	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	...	0	0
3	1	24	M	technician	85711	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	...	0	0
4	1	24	M	technician	85711	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	...	0	0

The above figure is a sample from the dataset, after reading three files(user,movie,ratings) and merging them, there are some features about user like ID,age and occupation, other features about the movie like ID, title and there is a categorical feature (genre) which is distributed across different columns where each column name is a category, there are some other irrelevant features for movies, like video release date and imdb url, these features are not important here as our algorithm calculates similarity between users and for calculating this similarity matrix we only use user ID, movie ID and movie ratings.It’s important to say that those unused features can be used in an expanded version of this problem, for example feature like release date is interesting as we may find that some users like modern movies more and do not like to watch old movies or vice versa so when predicting a movie rating we can integrate a weighting factor based on the movie release date preference for the current user.

First we start with some statistics about the data to understand it better:
Total number of users 943
Total number of features 30 (genre is distributed across columns,where each one is a dummy feature)
Total number of features 12 (if genres were combined into one column)
Number of Male users 670
Number of female users 273
Mean rating 3.52986
Standard deviation of ratings is 1.12567359914
Ratings can be an integer from 1 to 5

Then we calculate the Most rated movies and Top Rated movies

```
title
Star Wars (1977)                  583
Contact (1997)                    509
Fargo (1996)                      508
Return of the Jedi (1983)         507
Liar Liar (1997)                  485
English Patient, The (1996)       481
Scream (1996)                     478
Toy Story (1995)                  452
Air Force One (1997)              431
Independence Day (ID4) (1996)     429
dtype: int64
```

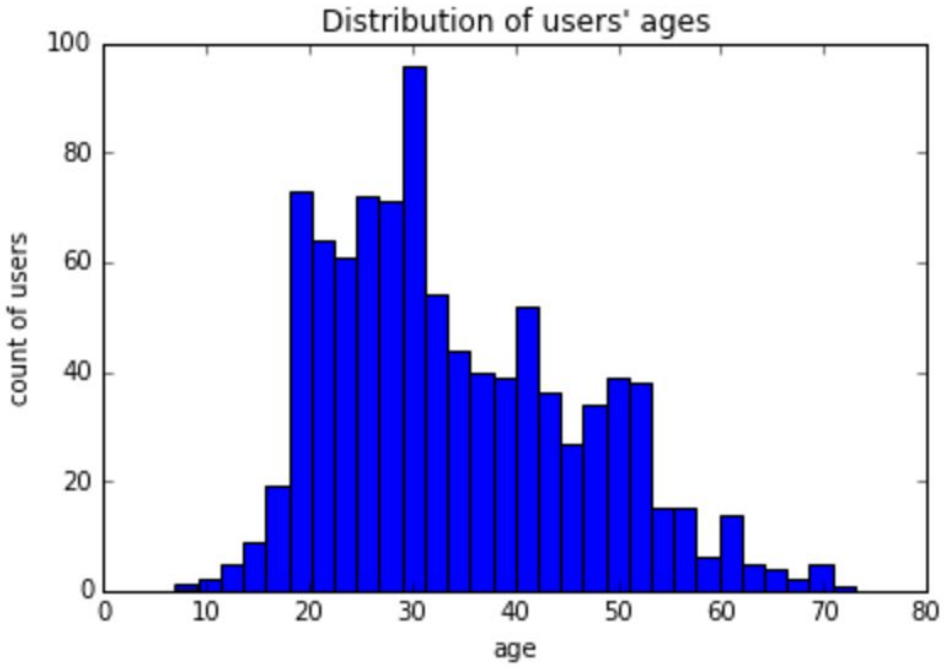
10 most rated movies in the dataset

```
dtype: int64
                                rating
                                  size      mean
title
Close Shave, A (1995)              112  4.491071
Schindler's List (1993)            298  4.466443
Wrong Trousers, The (1993)         118  4.466102
Casablanca (1942)                  243  4.456790
Shawshank Redemption, The (1994)   283  4.445230
Rear Window (1954)                 209  4.387560
Usual Suspects, The (1995)         267  4.385768
Star Wars (1977)                   583  4.358491
12 Angry Men (1957)                125  4.344000
Citizen Kane (1941)                198  4.292929
```

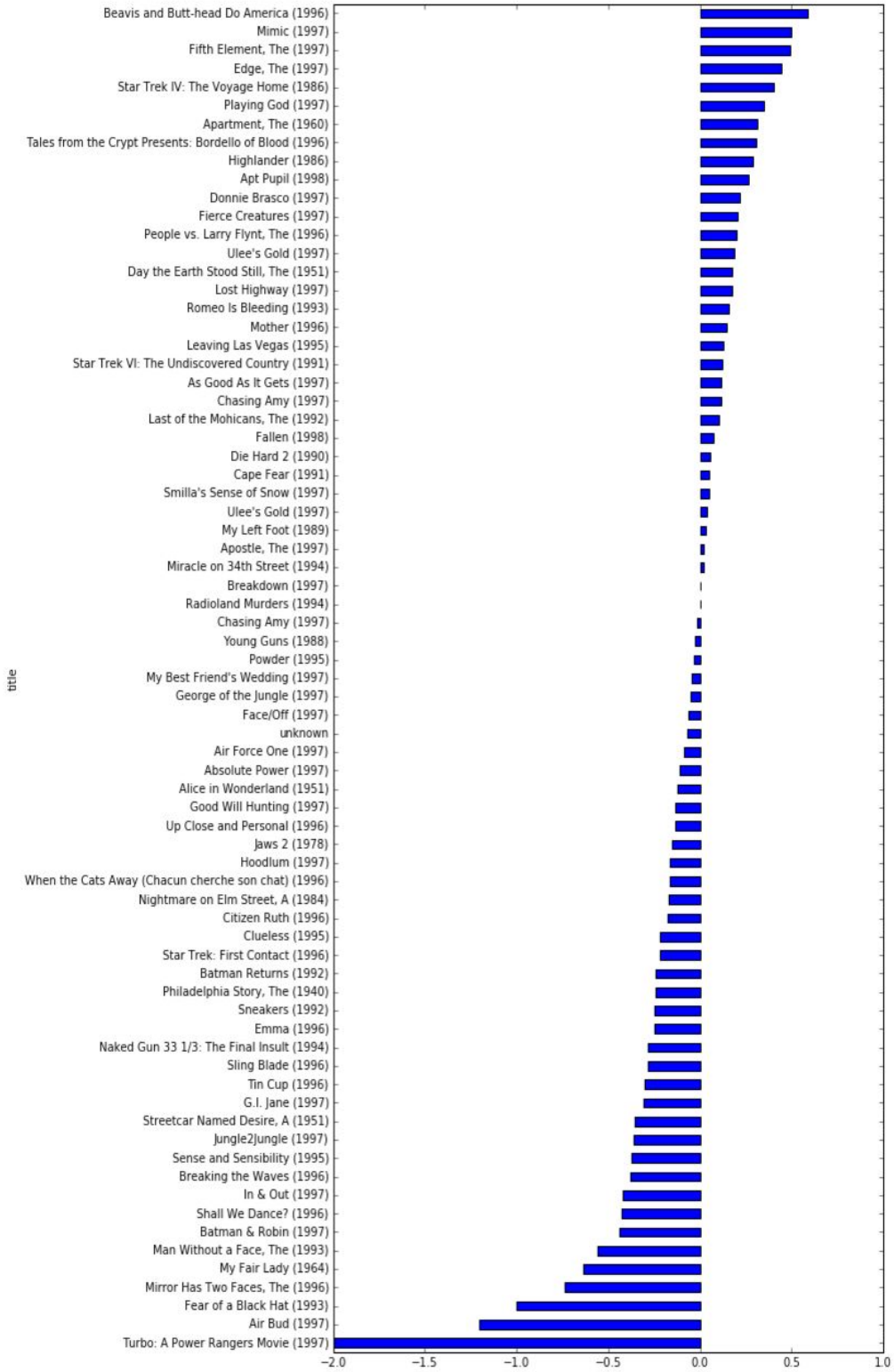
Top 10 rated movies in the dataset.

Most and top rated movies are useful they are represented as an option for how the user want to browse movies on their website for example he can browse them by seeing top rated movies like in IMDB top 250 movies list

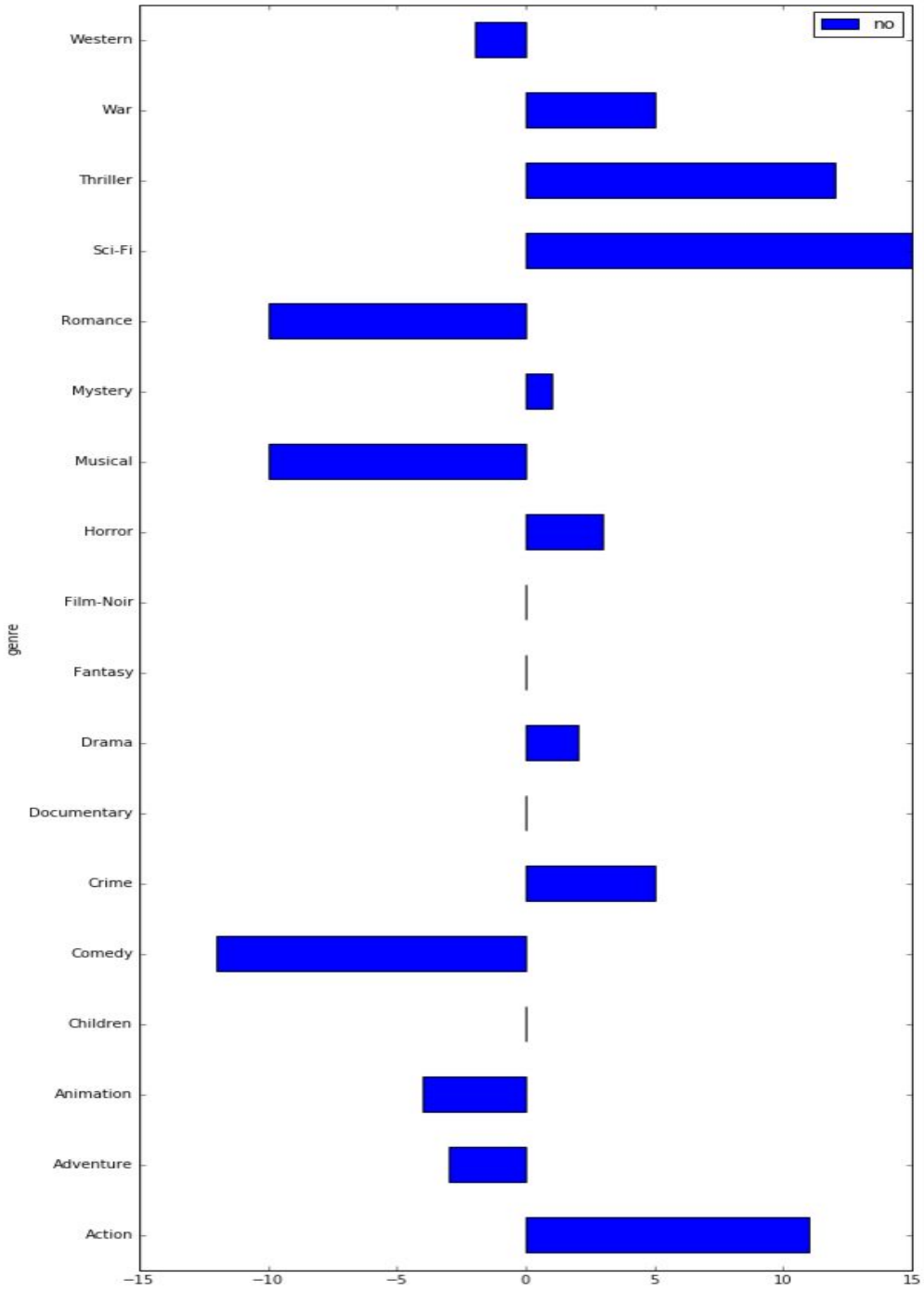
In order to understand the audience better i then plotted a histogram which shows a distribution of the users age, most users are about 30, after that it appears that much of the users are in their 20s, users older than 30 or younger than 20 are much less.



Then i took a look at male vs female different average ratings on top 100 movies, from this i decided to take it up a notch and determine the genre of each movie and conclude which genre is better based on whether the user was male or a female.



Male vs. Female Avg. Ratings for top 100 movies



Male vs. Female Movie Genres

The above figure shows movie genres and which genre is better for males(right) and females(left), so for example females tend to love comedy,musical movies but males tend to like action,sci-fi movies more than females do.This results are not used in this project but it's very important, it can be used in so many different ways if you hold user data you can already recommend movies based on the above graph, or you can use the recommender system

implemented here and give movies with genres that males like for males more weight and give genres that female like for females more weight

Algorithms and Techniques

First i took a subset of the dataset discussed above, i only used in this recommender system 3 features, user ID , movie ID , movie rating as they are what we need for the equations below.

There are two sub parts for the recommender system and that's how the algorithm works

1. Calculate similarity between current user and all the other users
2. Calculate prediction for a movie

Similarity

First we calculate a similarity matrix between the current user that u want to recommend movies for and all the other users, this similarity matrix shows how different users are similar/close/distant to the current user.
There are different ways to calculate similarity Euclidean distance , Pearson similarity score, Cosine similarity measure.
In this problem we use Pearson similarity score which results from 1 to -1, where 1 means highly correlated and -1 means the opposite
The symbol r_a corresponds to the average rating of user a, $r_{i,j}$ means rating for user i on movie j

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}}$$

After some research, analyses show that for user-based recommender systems – and at least for the best studied recommendation domains ,the Pearson coefficient outperforms other measures of comparing users,the Pearson measure considers the fact that users are different with respect to how they interpret the rating scale. Some users tend to give only high ratings, whereas others will never give a 5 to any item. The Pearson coefficient factors these averages out in the calculation to make users comparable.

Prediction

To make a prediction for *an item*, we now have to decide which of the neighbors' ratings we shall take into account and how strongly we shall value their opinions, prediction is done according to the below equation with uses the similarity calculated from the previous section

$$pred(a, p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a, b)}$$

After calculating the prediction for movies which are not yet seen by our current users, then sorting them in descending order with high ratings first, we then can recommend the top predicted movies in this list.

Benchmarks

During model evaluation a RMSE less than 1 was the target, our model RMSE should not exceed this benchmark and if it exceeds enhancements should be made, a benchmark of maximum RMSE of 1 was chosen as an indicator of the maximum acceptable shift in predicted rating should not exceed one higher or lower than the actual rating.

Another benchmark during prediction, we exclude any user who has 0 to -1 similarity with the current user as they are not highly correlated with the user, and we only take into consideration users with 0 to 1 similarity as they are strongly correlated with the active user

Methodology

Data Preprocessing

There is no preprocessing done on the dataset, feature selection is done manually as the algorithm uses only user id, movie id and movie ratings.
The genres feature is a categorical feature with about 18 categories, they are already read as dummy variables meaning that they are already preprocessed by MovieLens, each category is represented in a column.

Implementation

First we read the data from three different files, these files represent user information, movie information, and ratings
Then these files were merged into one single variable.
After the analysis part of the data, we start with a small building block, by making a function which calculates pearson similarity between two users

```
def pearson_sim(user_a,user_b):
```

This function takes user a and user b and returns a similarity score between 1 and -1 where 1 means that they are very similar and -1 means they are not similar.
This is done first by calculating each user movie ratings mean ra , rb, then using these values and minusing them from each user's movie ratings we are able to calculate the pearson similarity according to this equation.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}}$$

The pearson_sim function is used in the below predict function.
Predict function takes user ID and movie ID and returns the predicted movie rating for this user

```
def predict(user_a,movie_id):
```

It loops on all users and calculates a similarity matrix using our previous similarity function between our current user and all the other users.
Then it predicts a rating using this equation

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

Both functions are commented in the code section in the iPython notebook for further information

Originally rating prediction was too slow, I've traced the code looking for code blocks of high time complexity that can be enhanced,

```
for indexA,valA in userA_movies_ratings.iterrows():
    for indexB,valB in userB_movies_ratings.iterrows():
        if valA["movie_id"] == valB["movie_id"]:
            tempA = valA["rating"]-userA_avg
            diffA.append(tempA)
            tempB =valB["rating"]-userB_avg
            diffB.append(tempB)
            diff_squaredA.append(pow(tempA,2))
            diff_squaredB.append(pow(tempB,2))
```

The above code block is in pearson_sim function which is called during prediction, this function calculates the pearson similarity between two users,it iterates through all user A movie ratings and for each rating it loops in all user B movie ratings so this code is O(n^2).
I've replaced it with the below code block,instead of iterating on all the rows in a naive way using iterrows, i used Pandas loc and isin functions, which are used for indexing and selecting data, i don't know the complexity time or the inner workings of the loc or isin functions and couldn't find information in the Pandas documentation or elsewhere, but since the above code block is already slow O(n^2) and is very basic way for iteration, i replaced it with the below code and confirmed which is better through testing, by running the prediction function and seeing how much it takes to predict a rating.
After making this change predictions now are faster than the first submission(and so i could work on large number of movies).
Below using Pandas loc and isin functions i've calculated common movies between user A and B in the first 2 lines then continued normally as the previous code block.

```
userA_common = userA_movies_ratings.loc[userA_movies_ratings['movie_id'].isin(userB_movies_ratings["movie_id"].values)] #user A ratings for common movies
userB_common = userB_movies_ratings.loc[userB_movies_ratings['movie_id'].isin(userA_movies_ratings["movie_id"].values)] #user B ratings for common movies

diffA = userA_common.rating - userA_avg #difference between user A ratings and user A average rating
diffB = userB_common.rating - userB_avg #difference between user B ratings and user B average rating
diffA = diffA.values
diffB = diffB.values
#calculate the square of differences
diff_squaredA = (pow(diffA,2))
diff_squaredB = (pow(diffB,2))
```

Normally we shouldn’t predict ratings for movies that are already rated by the active user, we should only predict ratings for movies the user has not yet seen and make recommendations from them, but for testing i’ve commented the following code which excludes already rated movies(this should be uncommented if we were in production) that’s because our testing data contains movies with their actual prediction for a given user, which means they are already rated by the user but we predict ratings for them anyway and compare with the actual rating to evaluate our model.

```
#if movie_id in userA_data.movie_id.values:
    #movie is already rated by the user
    #this is only commented here for testing purposes as we test on movie that's already rated by the user to be able to calculate errors
    #if this code went to production we should skip this loop as we shouldnt predict movies already rated
    #print "duplicate"
    #continue
```

Since the data is large (100 K ratings) I’ve tested the model on a subset of the data, 20,000 ratings,20% of the data, this subset is manually sliced by me from the original data, there are some testing files included in the dataset folder by MovieLens but they are the same about 20,000 ratings sliced from the original data file (Same results are driven if we used the testing data included by MovieLens).

This algorithm is considered a nearest neighbour algorithm in other words it’s a lazy learner, meaning that there is no training step or function approximation step,so there’s no training/testing data split, i worked on the whole dataset in my previous analysis section,and took a subset of the whole dataset about 20K as my testing set.
Since the model prediction is slow and takes some time, it should not be used in runtime probably will be used offline, store each user-movie rating, and in runtime we just only search and fetch the previously predicted ratings.

Refinement

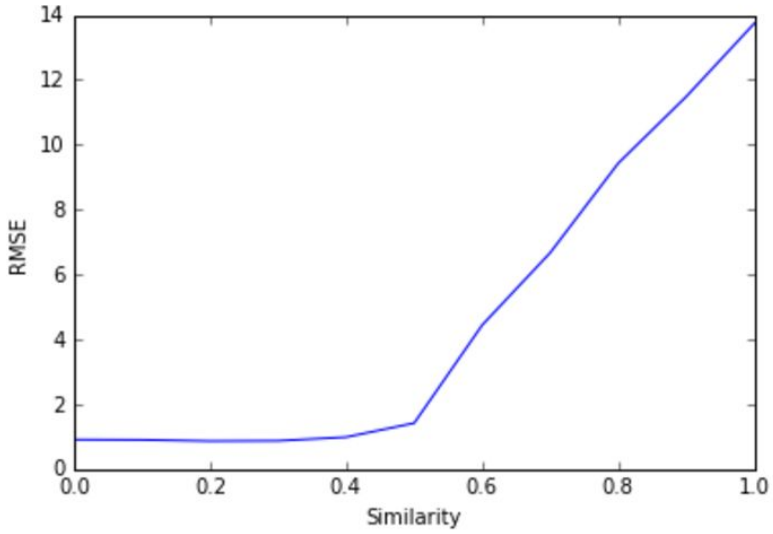
Originally i tested our model only on 20 movies which was not an accurate estimation of our model's RMSE and accuracy, below is the original result.

```
accuracy score 0.2
RMSE 0.95
```

To accurately evaluate our model i tested it on a 20% subset of the data (about 20K movie ratings) this way our model is exposed to large amount of data and we can evaluate it,the results improved more than before

```
accuracy score 0.4694
RMSE 0.8027
```

I've previously chosen a similarity equals to 0 or more and excluded 0 to -1 similarities as they are not highly correlated with our active user, i've evaluated our model for a small number of movie ratings (500 only as this task takes lots of time and it runs 11 times for each similarity variation) changing the range of similarity from 0 to 1 by a factor of 0.1 each time and see how increasing similarity threshold will enhance or worsen our model's predictions.



RMSE results when increasing similarity from 0 to 1

As shown in the above figure, we are starting with RMSE equals 0.91 then slightly decreasing until our RMSE will reach 0.86 at similarity 0.2, then it will increase again, based on these results i've chosen similarity threshold which produced lowest RMSE which equals 0.2 and applied it on our large testing set of 20K movies, which slightly enhanced our accuracy and

RMSE by a factor of 0.02, these results are still close to using threshold of 0 as previously shown.

```
accuracy score 0.4827
RMSE 0.7809
```

Since this algorithm is a nearest neighbour there is no training step and the prediction step is expensive as each time you want to make a prediction you are searching for the nearest neighbors in the entire dataset, there has been some slight code refinements as the algorithm was very slow like removing duplicates to loop on a much less data, and another code refinement as mentioned in the challenges in the above section for slightly faster prediction. There has been some refinements in how i evaluated the model written in the next section.

Results

Model Evaluation and Validation

```
accuracy score 0.4827
RMSE 0.7809
```

In the first submission i used Accuracy to measure the performance of the algorithm but it results low accuracy (0.2) that's because i used very small data to test on, After testing on 20K Movies, accuracy increased to 0.48.
In this problem accuracy only doesn't indicate how our model is doing, because predicted ratings are acceptable if they were close to the true ratings and accuracy won't capture that. That's why i used RMSE next to see how my predictions are far from true,i also did some research to see if i'm on the right track, i found that mainly recommender systems are tested whether by A/B test some process with and without recommendation and observe changes in key metrics like clicks, purchases, views for example,or RMSE.
In the previous submission RMSE was approximately 1 (0.95) as mentioned i've tested on very small data(20 movies because the algorithm was too slow), after enhancing the complexity of the model, i've tested on 20K movies and the error was reduced to 0.78 .
This is an acceptable result as our target benchmark was not to exceed RMSE of 1 (as mentioned before).

The previous scores are driven when we try our model on the testing data subset we took from the original data.
There are testing data included by MovieLens which are the same, they are slices of the original whole data file i'm working with, i've also tested my model on one testing file from included by Movie lens(20K movies) and it produced the same results(shown below).

```
accuracy score 0.4724
RMSE 0.79245
```


```
{1: (272, 4.0), 2: (316, 4.0), 3: (355, 2.0), 4: (362, 4.0), 5: (457, 1.0), 6: (539, 2.0), 7: (683, 3.0), 8: (689, 4.
0), 9: (691, 4.0), 10: (712, 4.0), 11: (752, 3.0), 12: (754, 4.0), 13: (918, 3.0), 14: (923, 5.0), 15: (938, 2.0), 1
6: (948, 3.0), 17: (919, 4.0), 18: (973, 4.0), 19: (887, 4.0), 20: (934, 3.0), 21: (995, 2.0), 22: (1003, 3.0), 23:
 (1006, 3.0), 24: (1007, 5.0), 25: (969, 4.0), 26: (1016, 3.0), 27: (1017, 3.0), 28: (895, 4.0), 29: (1019, 4.0), 30:
 (1013, 2.0), 31: (1022, 5.0), 32: (1023, 2.0), 33: (895, 3.0), 34: (1024, 5.0), 35: (1025, 3.0), 36: (1026, 0.0), 3
7: (1027, 3.0), 38: (1037, 3.0), 39: (937, 4.0), 40: (1038, 2.0), 41: (1039, 4.0), 42: (1051, 3.0), 43: (1057, 3.0),
 44: (1058, 3.0), 45: (1061, 3.0), 46: (1062, 5.0), 47: (1022, 4.0), 48: (1065, 3.0), 49: (1083, 2.0), 50: (1084, 4.
0), 51: (705, 3.0), 52: (1086, 4.0), 53: (1087, 2.0), 54: (1088, 2.0), 55: (1089, 2.0), 56: (1092, 3.0), 57: (1096,
 3.0), 58: (1106, 4.0), 59: (1120, 3.0), 60: (1126, 4.0), 61: (1127, 3.0), 62: (1136, 3.0), 63: (1138, 3.0), 64: (114
1, 4.0), 65: (1142, 5.0), 66: (1016, 3.0), 67: (1095, 3.0), 68: (1089, 2.0), 69: (1144, 4.0), 70: (1146, 3.0), 71: (9
23, 5.0), 72: (1148, 4.0), 73: (1149, 4.0), 74: (1084, 4.0), 75: (1152, 3.0), 76: (1159, 3.0), 77: (1028, 3.0), 78:
 (1160, 4.0), 79: (1161, 3.0), 80: (887, 4.0), 81: (1059, 3.0), 82: (1164, 2.0), 83: (1165, 2.0), 84: (1047, 3.0), 8
5: (1174, 3.0), 86: (1176, 4.0), 87: (1190, 4.0), 88: (1191, 5.0), 89: (1119, 4.0), 90: (1206, 3.0), 91: (1192, 4.0),
 92: (1216, 3.0), 93: (934, 3.0), 94: (1226, 4.0), 95: (1231, 3.0), 96: (1232, 4.0), 97: (1126, 3.0), 98: (988, 2.0),
 99: (1132, 3.0), 100: (1238, 3.0), 101: (1132, 2.0), 102: (1240, 3.0), 103: (1089, 2.0), 104: (1241, 2.0), 105: (88
0, 3.0), 106: (1242, 4.0), 107: (1243, 3.0), 108: (931, 2.0), 109: (1245, 3.0), 110: (1250, 1.0), 111: (1024, 3.0), 1
12: (1106, 3.0), 113: (1252, 4.0), 114: (1104, 4.0), 115: (1073, 4.0), 116: (1258, 2.0), 117: (1165, 3.0), 118: (107
9, 3.0), 119: (1265, 3.0), 120: (924, 4.0), 121: (1266, 3.0), 122: (1268, 3.0), 123: (1269, 4.0), 124: (616, 4.0), 12
5: (1272, 2.0), 126: (1175, 3.0), 127: (901, 3.0), 128: (1221, 3.0), 129: (1176, 3.0), 130: (1280, 3.0), 131: (1281,
 3.0), 132: (1154, 3.0), 133: (902, 3.0), 134: (892, 3.0), 135: (1217, 2.0), 136: (1142, 5.0), 137: (1117, 4.0), 138:
 (742, 4.0), 139: (1233, 6.0), 140: (988, 2.0), 141: (1283, 3.0), 142: (895, 3.0), 143: (1038, 3.0), 144: (1286, 3.
0), 145: (1292, 2.0), 146: (1294, 3.0), 147: (937, 3.0), 148: (1149, 4.0), 149: (1296, 3.0), 150: (628, 4.0), 151: (1
299, 3.0), 152: (1301, 4.0), 153: (678, 2.0), 154: (945, 4.0), 155: (990, 2.0), 156: (806, 4.0), 157: (1302, 5.0), 15
8: (1303, 3.0), 159: (1278, 4.0), 160: (1223, 4.0), 161: (1266, 3.0), 162: (1047, 3.0), 163: (879, 3.0), 164: (1025,
 4.0), 165: (1119, 4.0), 166: (984, 3.0), 167: (1310, 0.0), 168: (1278, 3.0), 169: (879, 4.0), 170: (988, 2.0), 171:
 (1022, 3.0), 172: (1172, 3.0), 173: (1265, 3.0), 174: (1313, 3.0), 175: (869, 3.0), 176: (1097, 4.0), 177: (1218, 4.
0), 178: (1315, 3.0), 179: (1316, 2.0), 180: (1131, 4.0), 181: (1395, 1.0), 182: (864, 4.0), 183: (1217, 3.0), 184:
 (1398, 5.0), 185: (1020, 4.0), 186: (1399, 3.0), 187: (1119, 4.0), 188: (1263, 3.0), 189: (1404, 4.0), 190: (1313,
 3.0), 191: (900, 4.0), 192: (1405, 4.0), 193: (1407, 2.0), 194: (1412, 1.0), 195: (1418, 4.0), 196: (1241, 3.0), 19
7: (1420, 1.0), 198: (1245, 4.0), 199: (1354, 2.0), 200: (1419, 3.0), 201: (583, 3.0)}
```

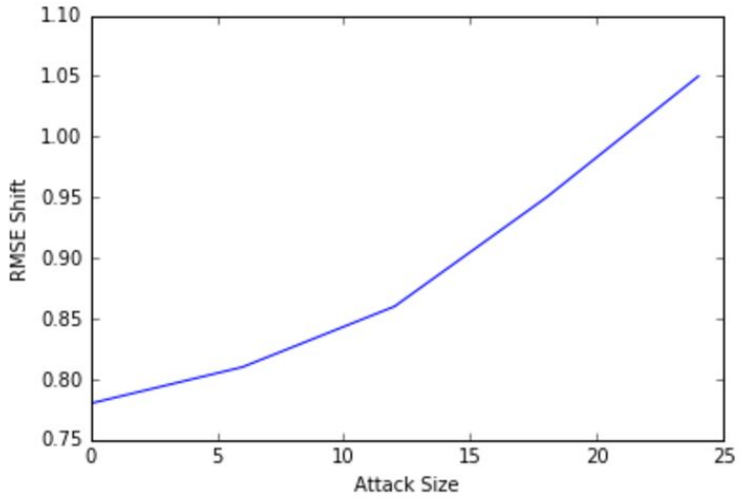

Final output user_id:(movie_id,rating)

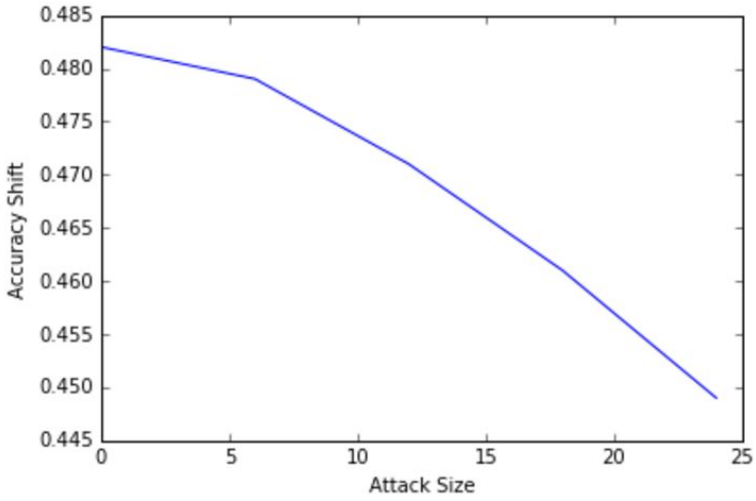
We can easily sort the output in the previous figure in descending order based on the rating and recommend them from top to bottom for our user or recommend only movies of average or high ratings (3 or more).

There is a problem with recommendation systems is that our collaborative filtering model assumes that all the ratings given for users are the true ratings.
This problem can cause a defect in our system as many attackers can use this problem to their advantage, in other words there can be an attack profile strategy which creates attack profiles and start to shift the prediction of the recommendation system, they can shift the prediction towards high ratings for certain products (Product push), or low ratings for certain products(Product nuke) or just randomly rate items for the sake of decreasing our model's accuracy

There are many kinds of attacks I've chosen to test our system on Bandwagon Attack, this attack chooses a set of popular items and give them the maximum ratings while remaining items(filler items) are given random ratings,by choosing to rate popular items this increases our probability for the attack user profile to be more correlated with many users in our database and to avoid detection, this attack also is similar to Random attack(which randomly predicts all

items) the only difference is it chooses a set of popular items to rate plus the randomly rated items.

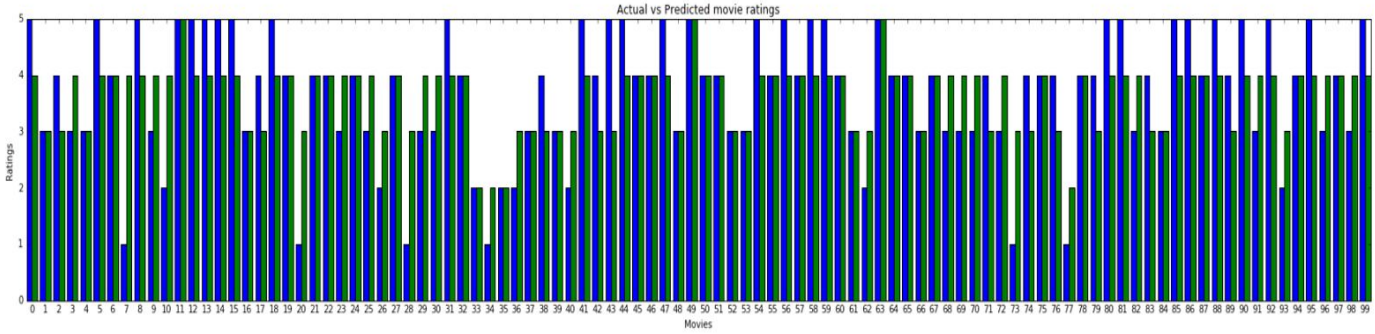




This Bandwagon attack uses a single popular item and 3% filler size (3% of the number of movies), This attack was run on our 20K testing set, different number of attack users were created (0%,3%,6%,9%,12% of the number of current users), the above results shows that our model's RMSE slightly increase as we increase the number of attack profiles in the system,and accuracy decreases slightly as well. There are some suggestions to decrease the effect of attack profiles in the improvements section below.

Conclusion

Free-Form Visualization



Actual ratings are represented by blue lines and Predicted ratings are represented by green lines

The above figure shows the actual and predicted ratings for 100 movies, in general they may be equal or slightly different by a scale of 1 rating (according to the previous evaluation results)
This figure shows a glimpse of why we allowed for a slight deviation of maximum 1 rating and why i chose RMSE of max 1 on the whole dataset, it's about the flow of both graphs,the flow is close where high ratings are predicted as high and and low ratings are predicted low (there is no large gap between ratings) that's what's important here, and that's what our evaluation results of RMSE less than one means.

Reflection

In this Recommender system we calculated correlation between a user and each other user in the system, these weights represent similarity and how much should a user value other user's opinion, based on these weights we predict the estimated rating of a movie for a certain user.
Opinions of users with high weights will be much more important than opinions of users of low weights and that was demonstrated in the prediction equation.
The goal of this project was to predict movie ratings for user, with these predictions many things can be accomplished and will be discussed in the improvements section
Other than the recommender system, user can query most rated items or top rated items, they sometimes are useful and needed in some applications
The algorithm is also slow and not real time, meaning that these predictions should be calculated and stored offline, and w query for the corresponding previously calculated prediction when needed.

Improvement

- Improve the time complexity of the algorithm to run faster.
- Integrate gender in the algorithm, create weights from the graph “Male vs. Female Movie Genres” and integrate these weights in the prediction function,where we will multiply by a weight based on user gender and movie genre as we discovered that some movies genres are strongly correlated with males while others are strongly correlated with females.
- Store calculated predictions offline, and only query for a user-movie prediction when needed.
- Test our model’s robustness on the whole testing data.
- Clustering can be used as a way of pre-filtering our users, looking for patterns for detecting outliers (possibly attack profiles) and excluding them.
- Increasing similarity threshold may decrease the effect of attack profiles on the predicted ratings if they were not highly correlated enough with our active user.
- If there is no sufficient data on our user or our user is new to our model(Cold start problem)
    - We can recommend Top rated movies but not the 10 most top rated, we go further down in the list
    - We can recommend also based on gender
- Implement item based collaborative filtering
    - Since we scan very large number of users, the need to scan this vast number of potential neighbors makes it impossible to compute predictions in real time
    - The main idea of item based CF is to calculate similarity between items not similarity between users, it’s an interesting problem and similar to this one.

References

Recommender Systems: An introduction by Dietmar Jannach
http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify
http://shop.oreilly.com/product/9780596529321.do
http://www.slideshare.net/neilhurley/tutorial-on-robustness-of-recommender-systems