I. Definition

Project Overview

I am movie lover, and I would love to have a system that can predict if I will like a move that I haven't seen.

I found a public dataset called MoviewLens (http://grouplens.org/datasets/movielens/) that I can use for this project. The data set claims to have 1 million ratings from 6000 users on 4000 movies.

The problem that I am trying to solve is predicting a rating, from 1 to 5, that I will give to a movie based on my ratings in other movies. The way I will attempt to solve this problem finding similar movies to the one I am trying to predict and give a prediction based on rating I gave to those movies that are similar.

Is important to notice that this project will only attempt to predict ratings given user-ids and a movie-ids as inputs. It will not provide a list of recommendations. Another thing to have in mind is that sometimes the system won't be able to predict a rating because it lacks the needed data (maybe you haven't given any other ratings, or there are no similar movies to the ones you have rated to make the prediction).

Problem Statement

More formally, I will try to predict the rating user "u" will give to movie "i" given by the formula:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}\, r_{uj}}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}$$

Figure 1. Rating prediction formula

Where:

$\hat{r}_{ui}$ *: Predicted rating given by user u to item i.*

$\mathcal{N}_u(i)$ *: Set of movies most similar to i that user u has rated.*

w_{ij} : Similarity weight between movie i and movie j.

r_{uj} : Rating given by user u to movie j.

$|w_{ij}|$: Absolute value of the similarity weight between movie i and movie j. The weight can be negative or can be positive. It would be positive if user

To solve this problem (ratings predictions), I will use an algorithm called **item-based collaborative filtering.** The way it works is by finding this similarity weights between each pair of movies by checking how similarly the community has rated two movies. To put an example, I will assume I will give a high rate to the movie “The Matrix” because I have enjoyed “Terminator” and “Die Hard” and these three movies have been rated in a very similar way by the community. After that , I can use the equation in Figure 1 to make predictions.

Metrics

The metric I will use to capture how similar two movies are (weights) is called **Pearson Correlation (PC).**

$$\mathrm{PC}(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}.$$

Figure 2. Pearson Correlation Formula

Where:

$\mathrm{PC}(i,j)$	: The Pearson Correlation between item i and item j.
$\mathcal{U}_{ij}$	: The set of users that have liked both item i and j.
r_{ui}	: The rating of the user u to item i
$\overline{r}_i$	: The mean rating of item i.
r_{uj}	: The rating of user u to item j.
$\overline{r}_j$	: The mean rating of item j.

The reasons for which I choose these metric over the others are:

- This metric (PC) takes into account the problem that if a movie has half ratings equal to 1 and the other half equal to 5, it will have a mean rating of 3 and if compared to other movie that has all of its votes equal to 3 may be considered identically in terms of how users rated them. So, with this formula we make sure that those two movies are not more similar than 2 movies that have all its ratings equal to 3.

- Models using this metric generally outperform models using other metrics (according to Francesco Ricci, Lior Rokach and Bracha Shapira,, 2015, Recommender System Handbook. Second Edition. London: Springer New York Heidelberg Dordrecht London, p. 54-56).

The metric I will use to evaluate how well the algorithm performed is the **Mean Absolute Error (MAE).** It will tell me how similar the predicted ratings and real ratings where. Here is the formula to calculate the MAE:

$$\mathrm{MAE}(f) = \frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} |f(u,i) - r_{ui}|$$

Figure 3. Mean Absolute Error (MAE)

Where:

$f(u,i)$: Prediction of the rating user u will give to movie i.

$|\mathcal{R}_{\text{test}}|$: Ratings in the test set.

MAE is part of the "Statistical accuracy" metrics, one of the 2 groups of metrics from which I could choose.

Statistical accuracy metrics reveals how good is the performance of the system by comparing the predicted score to the real score. In this group we can find the metrics Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), amongst other. I chose MAE because is the easiest to compute, is easy to interpret and the most commonly used.

The other group of metrics available is called "Decision support accuracy" metrics, and this group only takes into account if the prediction was satisfactory (good or bad), it doesn't take the predicted score vs the real score into account. Some of the metrics in this group are: reversal rate, weighted errors and ROC sensitivity.

Another metric that I will take into account is the **throughput,** which is basically how many estimations per seconds the model can do. This is not a priority for my model since I am not worrying about scalability right now, but it will be interesting to see how fast I can predict.

II. Analysis

Data Exploration

Raw data:

The raw data in the data set is contained in 2 files : /data/movies.dat, /data/ratings.dat.

All the info about the ratings is contained in the file /data/reatings.dat with this format:

UserID::MovieID::Rating::Timestamp

The information about the movies is contained in the file /data/movies.dat with this format:

UserID::Gender::Age::Occupation::Zip-code

Processed data:

My training and data sets will have 3 fields (Rating, MovieID, and User). I will use a library called Pandas to read and represent the data. To represent the data I will use the class Pandas.DataFrame which is like a database table with columns and rows, but it still has a numpy array on the inside.

This are the input data:

Input variable	**Values represented**	**Type**	**Desc**
y	Rating	Discrete	Number from 1 to 5 to describe how good the movie was., The bigger the number, the more you liked the movie.
X	MovieID	Discrete	Number that identify the movie.
	UserID	Discrete	Number to identify a user.

Table 1. Input variables.

This more metadata about the data:

Metric	**Ratings**
count	1.000209e+06
mean	3.581564e+00
std	1.117102e+00
min	1
25%	3
50%	4
75%	4
max	5
# of Users	6,040
Min. # of ratings by a single user	20
Max # of ratings by a single user	2314
Movies rated	3,706

Table 2. Statistics of the data set.

Comments:

As you can see this is a very simple data set and the data has been curated so there is no null value anywhere. In the same way , there are more users than movies rated. This is an important thing to mention in neighborhood algorithms (nearest neighbors) because this ratio should be taken into account when choosing between an user-based or item-based suggestion approach.

The reason why is because user-based suggestions are made based on which users have more similar taste to you and recommending items they have liked. Therefore, for user based you will need to compute the nearest neighbors of each user, and if there are more users than items (movies in this case), you should take into account that this approach will be more expensive than calculating the similarity between the items (item-based approach) since there are less items than users.

However, time and computational complexity are not the only things that needs to be considered. Each of these approaches have different characteristics to be considered, e.g., user-based usually can have more serendipitous (novel, different, interestingly new) recommendations since other users may have liked things that are very different than the ones you usually like. On the other hand, item-based has been proved to provide better accuracy in user ratings.

So, serendipitous recommendations is not something that I particular want for this system,since the scope of my project is prediction, not recommendation (although once you have prediction recommendation can be easily achieved). Also, in my data, I have 2 times more users than movies, therefore it will be faster to process item-based recommendations than user-based. Finally, for those reason, I choose item-based recommendations.

Exploratory Visualization

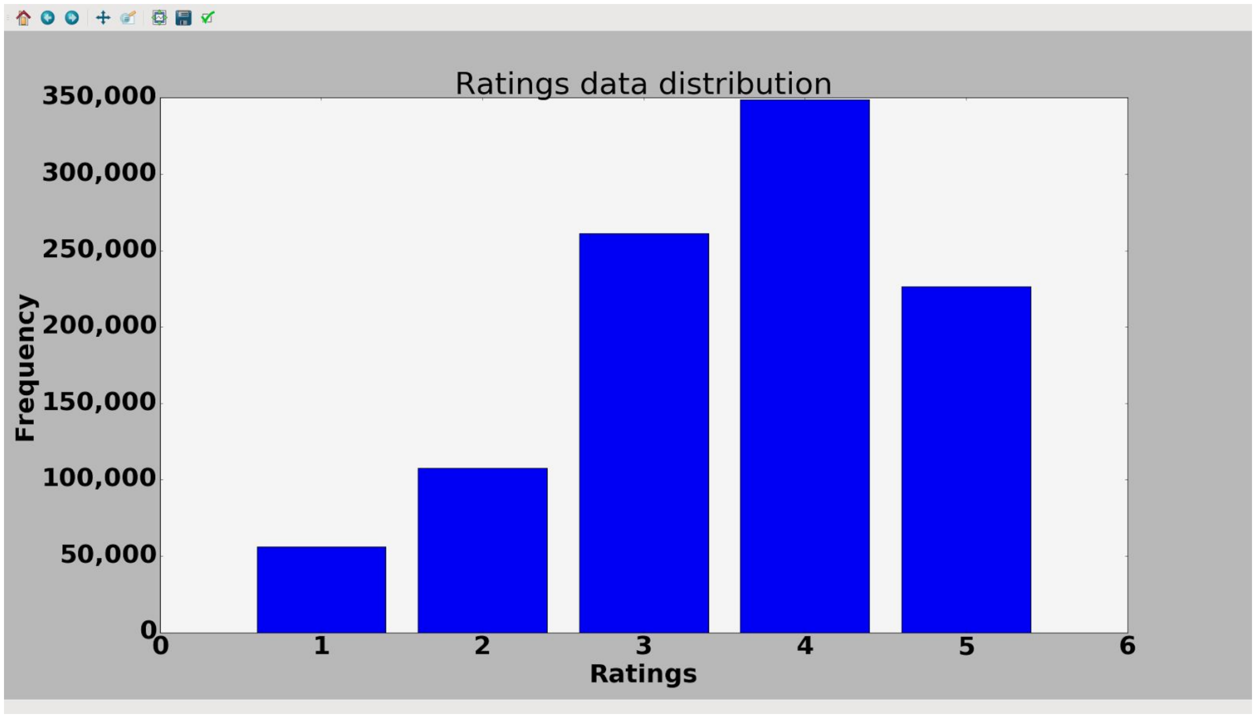


Figure 4. Ratings distribution

The figure 1 shows the distribution of the ratings of all movies. As shown in the picture, the most common score given to a movie is 4 and the least common is 1.

Is important to take into account that a score of 4 doesn't mean the same thing for two different users. Consider the scenario where there is very picky user, that only gives scores of 4 and 5 when his mind is blown away, otherwise he gives a score between 1 and 3. Now consider another user that is more easy to please, and usually gives scores between 4 and 5. The meaning of a rating of 4 is different for these two users. So they are not giving the same valuation to the movie, even if they are using the same number.

To solve this problem we can could normalize the scores of each user, dividing his ratings with his mean score. We could also take into account the standard deviation of his ratings and transform his ratings so they express better what the user meant to transmit with the rating. Nevertheless, that solution is used in the user-based context. We can do something similar when we are using item-based suggestions (like in this project), and that is normalizing the scores with the mean of the ratings of the item itself. The good news is that the Pearson Correlation (PC) coefficient already takes into account this normalization in its equation, so just using PC solves this problem.

Algorithms and Techniques

The algorithm I will be using to solve the problem of predicting the rating a user will give to a movie is called **item-based collaborative filtering.** It has 2 phases. First, using the ratings the community has given to the movies, you find which movies have been rated similarly (**calculating the Pearson Correlation coefficient between movies**). Secondly, you can predict the rating a user will give to a movie by doing a weighting average of user's rating to its nearest neighbors (the movies with the biggest PC with respect to the one we are predicting).

My workflow will be:

1) I am going to do is take a subset of the movies and pre-compute the a matrix of correlations between each movie pair. I will use the **Pearson Correlation (PC)** (**formula in figure 2**) formula to determine the similarity between 2 items. The bigger the Pearson Correlation, the more similar the 2 movies.

   Justification :

   The reasons for which I choose these metric over the others are:

   - This metric (PC) takes into account the problem that if an item has half ratings equal to 1 and the other half equal to 5, will have a mean rating of 3 and if compared to other item that has all of its votes equal to 3 may be considered identically in terms of how users rated them. So, with this formula we make sure that those two items are not more similar than 2 items that have all its ratings equal to 3. Not all the alternatives takes this into account.

   - Models using this metric generally outperform models using other metrics (according to Francesco Ricci, Lior Rokach and Bracha Shapira,, 2015, Recommender System Handbook. Second Edition. London: Springer New York Heidelberg Dordrecht London, p. 54-56).

2) I will use equation in figure 1 to compute the rating estimation between a user and a movie.

   Justification :

   This is the equation used for prediction in the algorithm that I chose (**item-based collaborative filtering)**

3) After that I will run k-fold cross validation for subsets of the data to find the best combination of the parameters. I will first run with a static value of **neighbors count,** and then see how the

train/test split percentage and **the number of ratings in the subsets** affects the **Mean Absolute Error (MAE)** (formula in figure 3) and the **throughput (predictions per second).** I will use the **Mean Absolute Error (MAE)** (formula in figure 3) and the **throughput (predictions per second** to guide my decision about which train/test split to use, and then I will use the whole dataset to test several values for **neighbors count.** With the results of the results of those steps I will decide which parameters to choose for the final model.

Justification:

I can choose between two groups of metrics : "Statistical Accuracy" and "Decision Support Accuracy". The Statistical Accuracy group shows how well the model predicted values in comparison to the real values using the any kind of rating originally in the data. On the other hand, the Decision Support group will evaluate how well the model predicted but it always supposes that there are only 2 possibles values for ratings, good or bad. Since my data has a categorical variable between 1 through 5 for ratings, the metrics of the Statistical Accuracy group will give me a better sense of how well the model is performing.

Inside the Statistical Accuracy group of metrics, I chose MAE because is easy to compute, is easy to interpret and the most commonly used.

4) After choosing the parameters I will train with all the data and see the final MAE and throughput.

Benchmark

I will choose an arbitrary benchmark for this experiment. I say the a model solves the problem successfully if it's MAE is less than 2. I am basing this benchmark on my opinion that a prediction that is not more than 2 points away than the reality is a good prediction. Since the MAE is the Mean Absolute Error, a MAE < 2 means that on average, the suggestions are less than 2 points away from the reality.

III. Methodology

Data Preprocessing

As mentioned in the data exploration section, the data was very curated. There will be no pre-processing since there are not nulls, no ratings outside defined valid range and the "normalization" of the ratings is taken care in by the Pearson Correlation formula.

Something I need to do with the data when before being able to make predictions, is reformat (pivot) it into a matrix of different dimensions. The correlation matrix will have dimensions #ofMovies x #ofMovies since its like a map between each movie pair. The training data will be pivoted to the dimensions #ofUsers x #ofMovies. This pivoting is used to facilitate the predictions calculation (see rating prediction equation) using matrices operations (dot products, etc), which make the calculations faster.

Pivoted data example:

	User 1	**User 2**	**... User N**
Movie 1	Rating User 1 gave to Movie 1	Rating User 2 gave to Movie 1	Rating User N gave to Movie 1
Movie 2	Rating User 1 gave to Movie 2	Rating User 2 gave to Movie 2	Rating User N gave to Movie 2
. . . Movie M	Rating User 1 gave to Movie M	Rating User 2 gave to Movie M	Rating User N gave to Movie M

Table 3. Pivoted Data format.

Correlation Matrix format:

	Movie 1	**Movie 2**	**... Movie M**
Movie 1	PC coefficient between Movie 1 and Movie 1 (always 1)	PC coefficient between Movie 2 and Movie 1	PC coefficient between Movie M and Movie 1
Movie 2	PC coefficient between Movie 1 and Movie 2	PC coefficient between Movie 2 and Movie 2 (always 1)	PC coefficient between Movie M and Movie 2
. . . Movie M	PC coefficient between Movie 1 and Movie M	PC coefficient between Movie 2 and Movie M	PC coefficient between Movie M and Movie M (always 1)

Table 4. Correlation Matrix format.

After pivoting the data in the above formats, you will end up wit a lot of these positions in NaN (Not a number), because, maybe user X has not rated movie Y, but still there is a place in the matrix where that rating belongs. I have read about using a sparse matrix for saving memory in these cases where we have a lot of NaN and 0, but I won't try to optimize prematurely.

For calculation purposes, all empty values (NaN) will be set to 0 when predicting. Doing this will allow us to perform dot products and other matrix operations on the data to achieve the prediction equation without errors and without affecting the final value.

Implementation

The implementation can be divided in the next stages:

1) Bootstrapping
2) Training and Pearson Correlation matrix calculation
3) Predictions
4) Analysis of results

Bootstrapping

1) I use the scripts/main.py module to bootstrap the process. In here we read the raw data from the file with the ratings into a Pandas.DataFrame. I declare a list of **ratings numbers to slice from the total dataset (the subset size)** , **test/train split percentage and K values** to test in the grid search and trigger the training.

2) I send the raw data and params to the scripts/metrics_calc.py module. This slices the data in subsets and divide them further into specified train/test splits and triggers the grid search for each of those slices with the specified values of K. Also it saves the results of the metrics and logs it to pandas.DataFrame objects that will be used for reports and graphics.

Training process and Pearson Correlation matrix calculation:

1) The metrics_calc module creates sklearn.grid_search.GridSearchCV objects with the params combination and the subsets and uses the BaseEstimator class in the scripts/estimator.py, which is my implementation of the pandas.base.BaseEstimator for this problem, as the estimator class for the GridSearchCV to train the model.

2) The estimator.RatingsEstimator receives neighbors_count as a param, and trains on the provided data by calculating the Pearson Correlation matrix between the movies using the method pandas.Dataframe.corr. Also, it pivots the training data and keeps both the pivoted data and correlation matrix for future predictions.

Difficulties in this step:

I was having a lot of trouble calculating the Pearson Correlation matrix at the beginning. I didn't know initially that Pandas library provided the pandas.Dataframe.corr method for doing this so I was trying to do my own implementation. I found a way to calculate the numerator of the equation with dot products but couldn't find an efficient way to predict the denominator of the equation, so I gave for loops a try. As suspected, looping through a very big matrix like this one was too much and the program was taking too long to finish (I left it running for 8 hours and didn't finish). After investigating a little more, I found the method pandas.Dataframe.corr which solved my problem.

Predictions

1) The prediction is implemented in the method 'predict' of the estimator.RatingsEstimator. When you call that method, the input data is pivoted and merged with the pivoted data from the training process. After that, only the Pearson Correlation of the K nearest neighbors are kept in the correlation matrix, the others are set to 0. Finally, matrix operations are performed between between the merged pivoted data and the correlation matrix to calculate the rating prediction equation.

Analysis of results

1) The logs of the process are kept. The idea was to first keep the "Neighbors count" static and vary the other parameters (test set percentage, size of subset) and see how the MAE and Throughput varies with those. After that we which values of "test set percentage",
"size of subset" to choose, and then run the full data set with multiple values of K, and see which K is the best.

After runs are made we will generate graphics for:

a) How the MAE changes with the train/test split
b) How the Throughput changes with the model size
c) How the MAE changes with the model size
d) How the MAE changes with the Neighbors count.

Refinement:

Choosing train/test split percentages:
We created several models and tested them with several subsets of data splitted in different train and test sets of different sizes, in order to produce the best model possible.

As we can see in Figure 5, the Mean Absolute Mean (MAE) changed with the test percentage (train/test split). It seems like as test percentage increases, the MAE also increases. Three of the four models performed better with a train set percentage of 0.3

Because of the results of this experiment we will choose a **test_set_percentage of 0.3** for the final model.

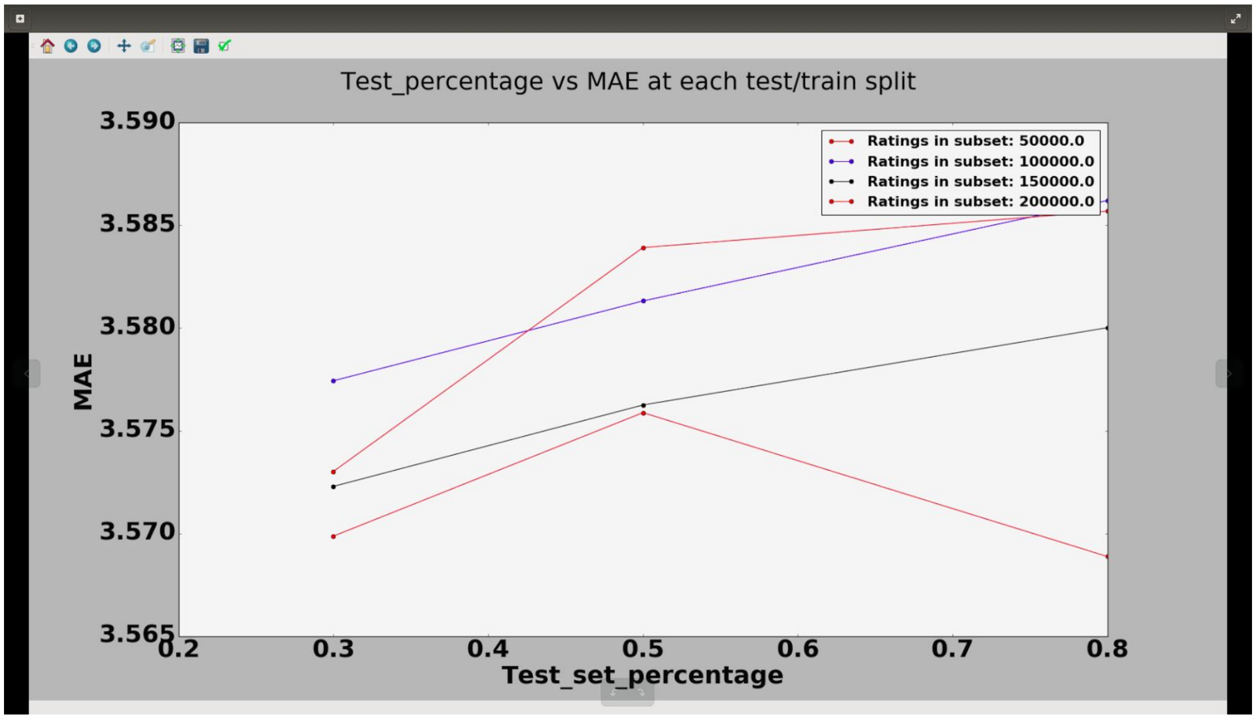


Figure 5. Test set percentage vs MAE

MAE variation with Ratings in subset:

As seen in Figure 6, MAE tends to decrease as more data is added. This makes sense, since the module will have more and better neighbors and more ratings to do better predictions.

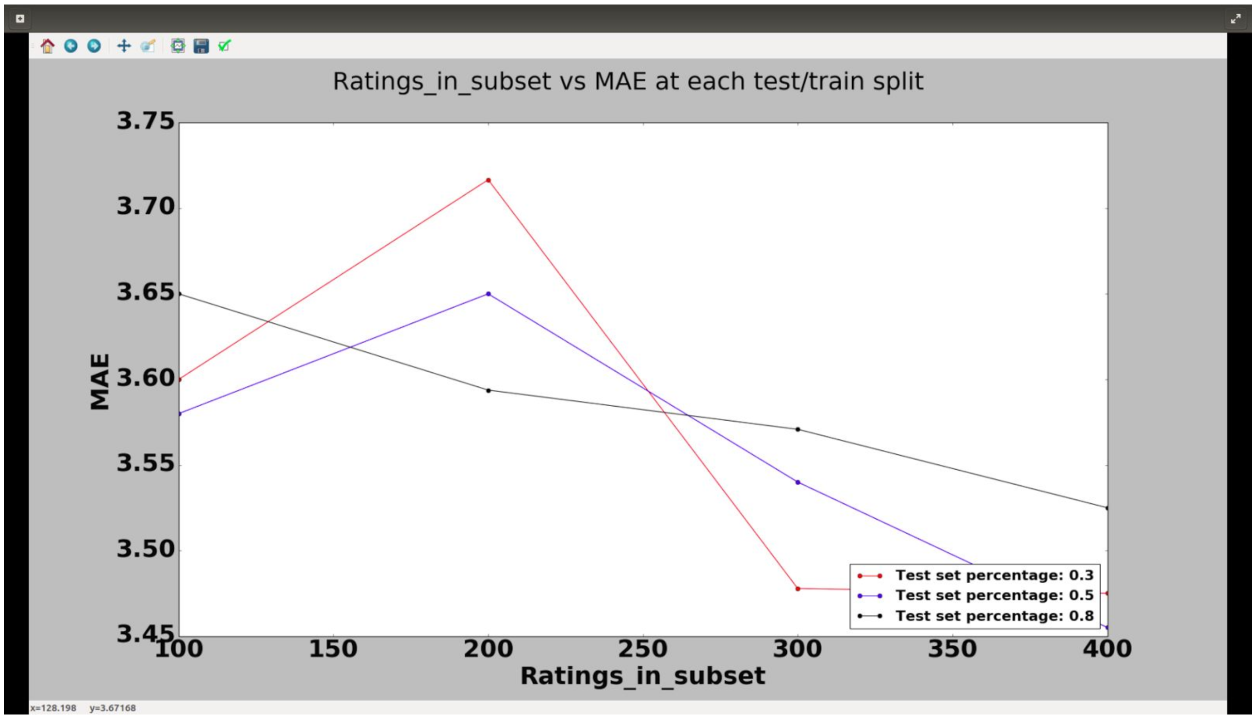


Figure 6. Ratings in subset vs MAE

Throughput variation with the subset size:

The throughput behaves as expected, the more big the data set, the more slow it will be. This is because the correlation matrix and the pivoted data matrix will get bigger.

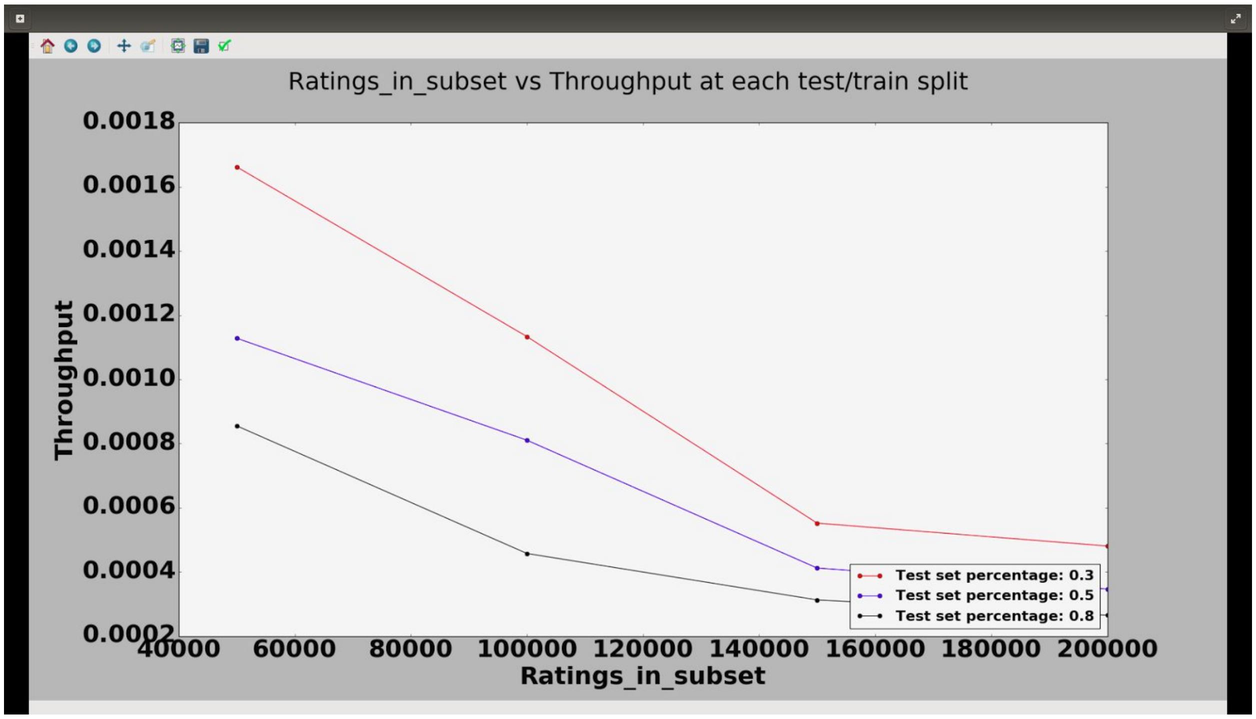


Figure 7. Ratings in subset

IV. Results

Model Evaluation and Validation:

These are the values for the final model:

Best Neighbors Count	**Best test set percentage**	**MAE**	**Throughput**
200	30 %	3.572689	0.00023

Table 5. Best values for model.

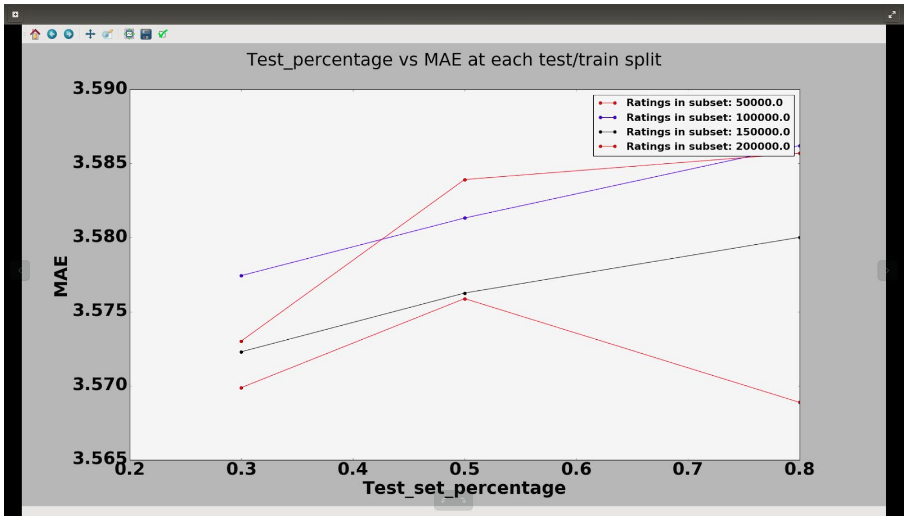
These values of (except the best test set percentage) where obtained from running the entire dataset dataset in a 3-fold cross validation of the following values:

Neighbors Count	Test set percentage	MAE	Ratings in subset
25	30 %	3.572689	1,000,209
50	30 %	3.572689	1,000,209
75	30 %	3.572689	1,000,209
100	30 %	3.572689	1,000,209
125	30 %	3.572689	1,000,209
150	30 %	3.572689	1,000,209
175	30 %	3.572689	1,000,209
200	30 %	3.572689	1,000,209
6000	30 %	3.582269	1,000,209

Table 5. Last grid search.

It captured my attention that the MAE is almost not varying with the neighbors count. I only saw the difference when I increased the value a lot, and by doing that the MAE got worse. When I debugged I found out that most of the values in the in the pivoted values matrix and correlation matrix are 0, so when I perform the dot product, they cancel each other. It doesn’t matter if I take 50 or 200 as the nearest neighbors if the user doesn't have any rating for that movie. A very big value of K increases the chance of finding a neighbor that was rated by the user, but then there are a lot of negative correlations, and when those are taken into account they increased the MAE.

The best test set percentage was chosen earlier by running small subsets and choosing which was the train/test percentage to use:



Is a little difficult to test do a sensibility test in this context. As far as I understand, a sensibility test consist in varying the input variables and see how much the output is changed and which variable caused the most change. In this case the input data is a user_id and movie_id, and the output is rating. I tried to send movies_ids and user_ids that didn't exist and the out of the model is always 0 when either one of them doesn't exist.

Justification:

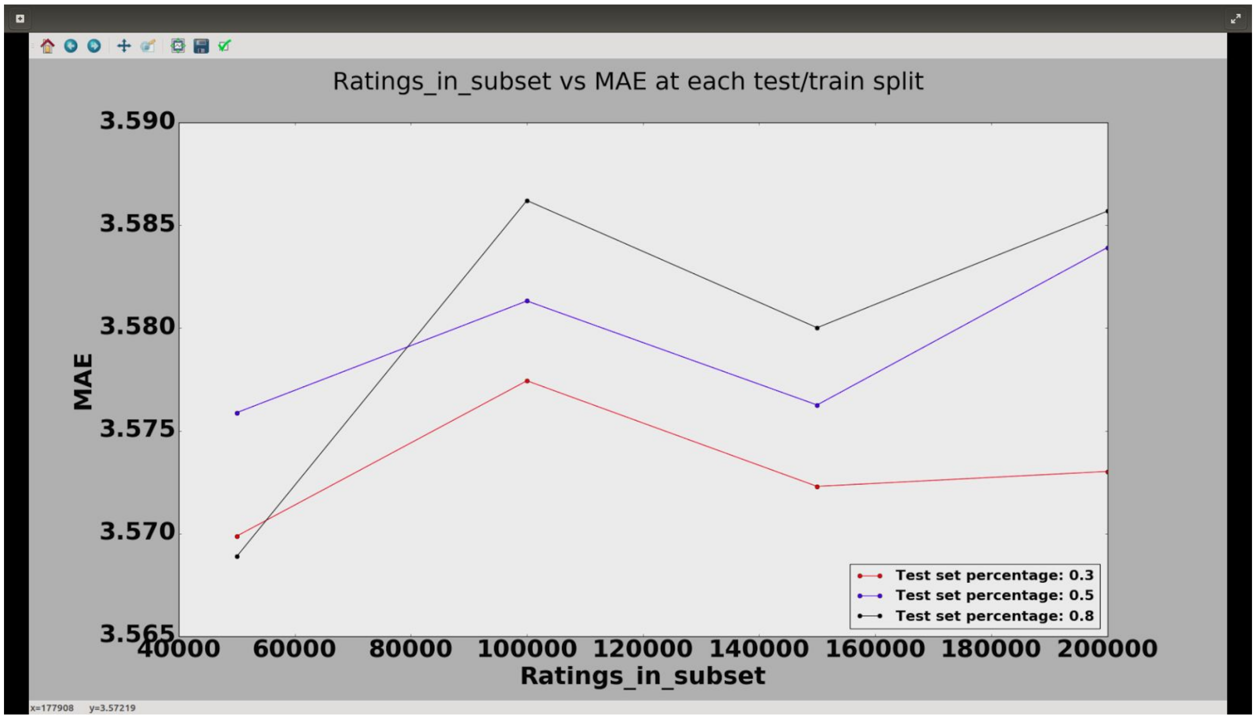
If we test the MAE obtained in the result (x) in the benchmark condition we defined in the benchmark section (MAE < 1.5 is acceptable) , we will conclude that we couldn't make a model that satisfies the benchmark, and that it **doesn't solve** the problem for what it was created.

I debugged my application carefully and it seems to be doing what it should. I found several factors that could be preventing my model from being better in the training dataset itself. For example a found out that either the user did not have ratings on the neighbors movies or the correlation index between the movies was zero. Since there is a product between these two in the rating prediction equation, and if either one is zero that is enough to make that neighbor have no effect in the prediction.

In the same way, we have a lot of negative and 0 Pearson Correlation Coefficients between many of the movies, and with the lack of good neighbors, these coefficients are taken into account for predictions, affecting negatively the predictions.

V. Conclusion

Free form visualization:



One of the things I noticed is that, I can obtain a very good performance without training the whole data. The MAE of the best model, using all the available data (1,000,209 ratings) was 3.5726. But as seen in the above figure, I obtained a MAE of around 3.57 by only around 40% of the data as training (400,000/1,000,209 = 0.3999).

That doesn't mean that taking less data into consideration is better. But if I have a little computational resources and/or time, I can obtain a model that performs acceptably in less time.

Reflection:

This problem consisted in obtaining a model that can predict the ratings that a user will give to a movie and that performs better than MAE < 1.5. The plan was to use item-based collaborative-filtering to achieve that. This algorithm consist in calculating similarities between movies and with the movies that are most similar to the one we are going to predict, we can do a weighted average of the user´s ratings to those other movies and use that as the prediction.

We use the Pearson Correlation coefficient to represent the similarity between two movies. This metric goes from -1 to 1, and when is 1, it means that the two movies are linearly and positively correlated, having a score of -1 means that they are inversely correlated, and 0 means no correlation. The PC are calculated based on how similar the community has voted in these two movies.

After obtaining the pearson correlation values, we can choose the K nearest neighbors of a movie by selecting the movies with the highest PC coefficient between them. We can take the K neighbors and to a weighted average of the ratings of the user on those K neighbors and use that as our prediction.

In the implementation we encountered several challenges, especially with performance. At the beginning, we didn't know an easy way to calculate the pearson correlation. The numerator of the equation could be easily computed by a dot product between matrices, but I was having trouble to come up with an easy way of calculating the denominator. I tried to implement the pearson correlation index calculation by traversing the pivoted data matrix with 2 for loops, but that was taking too much time (I leave the program running the whole night and it never finished). After investigating a little bit more, I found the function pandas.DataFrame.corr() which calculates the Pearson Correlation between each pair of column in a DataFrame. Using this function on the pivoted data matrix I obtained the correlation matrix and the performance problem ended.

At the end, the model didn't satisfy my expectations of having a MAE < 2. I debugged my application carefully and it seems to be doing what it should. Something that I did found out, is that sometimes negative Pearson Correlation affects the too much the much the prediction, which otherwise will be a better one. Likewise, I found out that many of the ratings by users met with a Pearson Correlation of 0 or NaN in the dot product between the pivoted data matrix and the correlation matrix used for prediction. This caused a lot predictions to be 0 which produces a very high error. I think this problems can be addressed by getting more data.

Improvement:

We could improve several things. Firstly the MAE is higher than the desired one. It could be possible to get a better MAE by getting more records, because many times when the ratings by users met with a Pearson Correlation in the dot product between the pivoted data matrix and the correlation matrix used for prediction, either the rating or the correlation are 0 and doesn´t help in the prediction.

On the other hand, I could also improve the throughput. I could do this by reducing the size of the pivoted data matrix and the correlation matrix, to just contain the rows and columns that represents the movies that are the K nearest neighbors. In my algorithm, I do this partially, but only in the columns of the correlation matrix, I could also eliminate rows. What I do in the rows is, set to zero the correlations between the movie I am trying to predict and the ones that are not the in the K nearest neighbors. That way the prediction is not affected by those dissimilar movies, but the dot product will still need to calculate the operation for the rows and columns in zero.