Udacity

Recommendations using Collaborative Filtering on Amazon's Clothing Dataset

Machine Learning Nanodegree Capstone Project

Maher Malaeb
12/27/2016

1 TABLE OF CONTENTS

2 DEFINITION

2.1 PROJECT OVERVIEW

Recommendation systems play a vital role in our everyday lives. Each time Facebook suggests a new friend, Amazon a new product, Netflix a new movie, etc. there is a recommendation system behind the scenes that is working to generate the most relevant options for each user.

One of the most famous and powerful recommendation approaches is called collaborative filtering (CF). CF predicts items based on the history of ratings that a user gave and the history of rating given to an item. CF approaches has been heavily applied to different datasets in literature. The most famous 2 datasets are the Netflix and the MovieLense movies dataset. Both of these represent a set of movies and users and the ratings given by users to movies.

2.2 PROBLEM STATEMENT

Throughout this project we will use CF to predict the ratings that a user might give to a certain item based on the Amazon's rating data set (McAuley). The dataset is the actual ratings given to Amazon clothing, shoes, and jewelry category. It has a set of users and items and the ratings given by those users to some items. Our model will try to minimize the error between predicted ratings and actual ratings.

2.3 METRICS

2.3.1 ROOT MEAN SQUARE ERROR

Since we aim to predict the ratings that each user will give to each item, we want to minimize the difference between the predicted value and the actual value. A common metric for such problems is the root mean square error (RMSE).

The RMSE formula is defined as

$$RMSE = \sqrt{\sum_{t=1}^{n} \frac{(\hat{r} - r)^2}{n}}$$

$$where\ \hat{r} = predicted\ rating, r = actual\ rating, n = all\ elements\ with\ known\ values\ in\ array$$

2.3.2 FRACTION OF CONCORDANT PAIRS

Since this is a recommendation task, we don't only care to match the rating as close as possible (by minimizing RMSE) but also we care about ranking the items for a specific user in the correct order. i.e. if a user gave items A, B, C ratings of 5, 4 and 3 respectively our system should produce ratings for A higher than B and for B higher than C. There is a metric called Fraction of Concordant Pairs that is specifically designed to test this aspect.

For a certain user u, a pair of items is set to be a concordant pair if the order of the predicted rating is the same as the order of the actual rating. If the order is opposite, they are said to be a discordant pair. Mathematically, for a user u, the fraction of concordant pairs is computed as follows:

$$n_c^u = number\ of\ concordant\ pairs\ when\ comparing\ all\ pairs\ of\ items\ rated\ by\ user\ u$$

$$n_d^u = number\ of\ discordant\ pairs\ when\ comparing\ all\ pairs\ of\ items\ rated\ by\ user\ u$$

$$mean\ of\ the\ number\ of\ concordant\ pairs\ for\ all\ users: n_c = \frac{\sum_{u=1}^{k} n_c^u}{k}$$

$$mean\ of\ the\ number\ of\ discordant\ pairs\ for\ all\ users: n_d = \frac{\sum_{u=1}^{k} n_d^u}{k}$$

$$Fraction\ of\ concordant\ pairs: FCP = \frac{n_c}{n_c + n_d}$$

3 ANALYSIS

3.1 DATA EXPLORATION

The dataset used is formed is formatted in json. Where each entry represent s a reviewer's rating to an item. Below is a random sample entry.


```
{
	"reviewerID" : "AS2FY9P0LASZA",
	"asin" : "B002ATSG8C",
	"reviewerName" : "Casual Warrior",
	"helpful" : [0, 0],
	"reviewText" : "I only buy Dockers. I can but them with my eyes close.
	"overall" : 5.0,
	"summary" : "Dockers are the best",
	"unixReviewTime" : 1400198400,
	"reviewTime" : "05 16, 2014"
}
```


Since Collaborative filtering algorithms are based on the user-item ratings matrix, we need to build this table from the json file. From each entry we only care about the "reviewerID" which is a unique identifier for a user, "asin" which is a unique identifier for an item and "overall" which is the given rating. All the other fields are not useful for our collaborative filtering task so they will not be used in the implementation of the recommender system. The target prototype user-item ratings matrix is represented in Figure 1 - User-Item ratings matrix prototypeFigure 1 below where each row represent ratings given by a user and each column represents ratings given to an item. The zeros are unknown ratings.

	It1	It2	It3	It4	It5	It6	It7	It8	It9	...
U1	0	0	2	0	4	0	0	0	0	
U2	1	0	0	0	0	3	0	5	2	
U3	0	2	3	0	3	2	0	0	0	
U4	0	2	1	0	2	1	0	3	0	
U5	0	0	2	0	0	0	4	0	0	
U6	0	0	2	0	0	0	0	0	0	
U7	1	0	0	0	0	2	0	2	0	
U8	0	3	0	3	0	0	5	0	1	
U9	0	0	3	0	0	0	0	0	1	
U10	2	1	3	0	2	0	1	0	0	
U11	0	1	0	1	0	3	0	3	1	
...										

FIGURE 1 - USER-ITEM RATINGS MATRIX PROTOTYPE

Since collaborative filtering is based on the history of the user-item interactions, it is important that the users have rated several items and items have been rated by several users to produce meaningful results. In our dataset, each user has rated at least 5 items, and each items has been rated by at least 5 users as can be seen in Table 1 - basic dataset statisticsTable 1.

TABLE 1 - BASIC DATASET STATISTICS

	Minimum	Maximum	Mean	Standard deviation
number of items rated by user	5	136	7.075355	3.585205
number of users that rated an Item	5	441	12.09903	13.967952

From table we can see that the data varies widely where we have a user that rated 136 items and an item that has been rated by 441 users. The mean however, is much closer to the minimum and when taking the standard deviation into consideration and assuming normal distribution, we can conclude that more that 90% of the users gave between 5 and 14.24577 ratings (mean +- 2 standard deviation) and more than 90% of the items have been given between 5 and 40.03493 ratings.

From the above data, we can notice that there are differences in the mean of and range of the users ratings and items ratings. This is explained by the checking Table 2 below where it shows that the number of items is lower than the number of users. Thus the number of ratings given by a user is naturally less than the number of ratings given to an item.

TABLE 2 - ITEM, USER AND RATINGS COUNT

number of unique items	23033
number of unique users	39387
total number of ratings	278677

Another important aspect to look at when dealing with a collaborative filtering problem is the sparsity of the user-item matrix. The sparsity is defined as the ratio of the number of known ratings over the total number of possible ratings. We can thus compute the sparsity of our matrix based on Table 2 above.

$$sparsity = \frac{\textbf{total number of ratings}}{\textbf{number of unique users} * \textbf{number of unique items}} = 0.00030718$$

3.2 EXPLORATORY VISUALIZATION

Figure 2 below represents the overall frequency of each rating. The histogram shows frequency of each rating whereas the pie chart shows the relative percentage distribution. It can be seen that high ratings (4 and 5) dominates low and middle ratings with 79.5 % of the total number of ratings. We can expect that our algorithm will give high ratings much more than low ratings to reduce the error with the actual data.

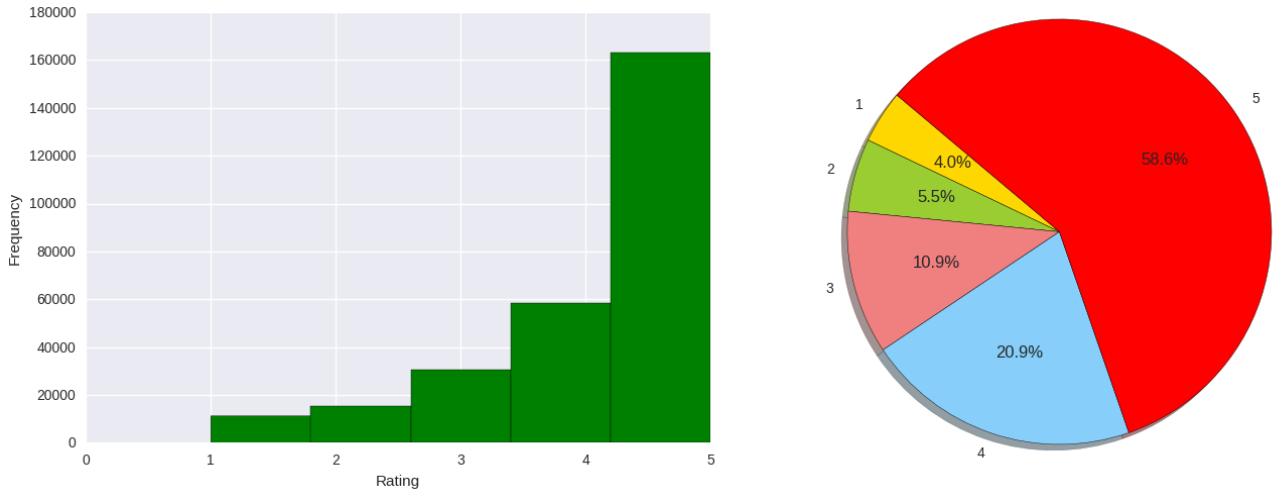

FIGURE 2 - RATINGS DISTRIBUTION

3.3 Algorithms and Techniques

3.3.1 Singular Value Decomposition (SVD)

Definition

SVD is a matrix factorization technique that is usually used to reduce the number of feature of a dataset by reducing the matrix from N space to K space where K < N. For the purpose of the recommendation system however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix built. The below is explained based on the paper by (Yehuda Koren, 2009)

Each item can represented with a vector q_i. Similarly each user can be represented by a vector p_u such that the dot product of these 2 vectors is the expected rating.

$$expected\ rating = \widehat{r_{ui}} = q_i^T p_u$$

q_i and $p_u$can be found in such a way that the square error difference between their dot product and the known rating is minimum.

$$minimum\ (p,q) \sum_{(u,i)\in K} (r_{ui} - q_i^T . p_u)^2$$

Regularization

For our model to be able to generalize well and not over-fit the training set, we introduce a penalty term our minimization equation. This is represented by a regularization factor λ multiplied by the square sum of the magnitudes of user and item vectors.

$$minimum\ (p,q) \sum_{(u,i)\in K} (r_{ui} - q_i^T . p_u) + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

To illustrate the usefulness of this factor imagine we have a case where $r_{ui}$is an extreme case (low rating given by a user to a movie with no other rating from this user). The algorithm will minimize the error by giving q_i a large value. This will cause all rating from this user to other movies to be very low. This intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided (Funk, 2006).

Bias terms

To reduce the error between the predicted and actual value the algorithm make use of some characteristics of the dataset. In particular for each user-item (u, i) pair we can extract 3 parameters. μ which is the average ratings of all items, b_i which is the average rating of item i minus μ and b_u which is the average rating given by user u minus μ which makes the expected rating

$$\widehat{r_{ui}} = q_i^T p_u + \mu + b_i + b_u$$

The final equation to minimize is thus (Hug, Matrix Factorization-based algortihms, 2015):

$$minimum\ (p,q,bi,bu) \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u - \mu - b_i - b_u) + \lambda(\|q_i\|^2 + \|p_u\|^2 + \|b_i\|^2 + \|b_u\|^2)$$

Stochastic Gradient Descent (SGD)

The above equation is minimized using stochastic gradient descent algorithm. From a high level perspective SGD starts by giving the parameters of the equation we are trying to minimize initial values and then iterating to reduce the error between the predicted and the actual value. Each time correcting the previous value by a change. This algorithm uses a factor called learning rate γ which determines the ratio of the old value and the new computed value after each iteration. Practically, when using high γ one might skip the optimal solution whereas when using low γ values a lot of iterations are needed to reach optimal value (Vryniotis, 2013).

Adjustable parameters

One can conclude from the explanation of the algorithm above 3 major parameters that can be tuned to achieve better results. These are the regularization factor λ which has a default value of 0.02, the learning rate γ which defaults to 0.005 and the number of SGD iterations which defaults to 20.

3.3.2 BASELINE ESTIMATE

Definition

The baseline is a algorithm that is very similar to how the bias terms are added in the SVD. In particular, suppose that the average rating of all the dataset is μ, the expected rating is

$$\widehat{r_{ui}} = b_{iu} = \mu + b_i + b_u$$

where b_i is how much the item rating deviate from the overall average, and $b_u$how much the ratings given by a specific user deviate from the global average (Koren, 2010).

Baseline Configuration

The challenge in this algorithm is how to estimate the terms b_i& b_u to minimize the predictions error. And one of the ways this can be done is by using SVD in a similar manner as the one used in the SVD algorithm but to minimize the below formula

$$minimum\ (bi, bu) \sum_{(u,i)\in K} (r_{ui} - \mu - b_i - b_u)^2 + \lambda(\|b_i\|^2 + \|b_u\|^2)$$

here also we have the regularization term λ and the learning rate γ that are used in the computations as explained previously (Hug, similarities module, 2015)

Adjustable parameters

So the parameters to adjust are exactly the SVD algorithm parameters. These are λ defaulted to 0.02, γ to 0.005 and the number iteration SVD runs which is defaulted to 20 (Hug, similarities module, 2015).

3.3.3 CO-CLUSTERING

Co-Clustering definition

Normal clustering approaches put each element of a dataset in a separate cluster without duplication. i.e. An element can be in one and only one cluster. Co-Clustering is an algorithm that relaxes this constraint where a item can be in 2 clusters at the same time. For example suppose we have a dataset of animals and we are doing co-clustering on it. We can see that chicken appears in the “birds” cluster and in the “domesticated animals” cluster.

Co-Clustering for Collaborative filtering

In the context on CF, we divide users into k clusters and items into m clusters. A specific item or user in the user-item ratings matrix is in a user cluster and item cluster at the same time. Thus, the Co-Clustering part.

Predicting

As explained in (Thomas George) After dividing the items and users into clusters a prediction for the co-clustering algorithm is done as follows.

$$\widehat{r_{ui}} = (\mu_u - \mu_{uc}) + (\mu_i - \mu_{ic}) + \mu_{iuc}$$

Where

$$\mu_u = User\ average\ rating, \mu_{uc} = Average\ rating\ of\ user\ u\ cluster$$

$$\mu_i = Item\ average\ rating\ , \mu_{ic} = Average\ rating\ of\ item\ i\ cluster$$

$$\mu_{iuc} = Average\ rating\ in\ the\ Co-Cluster\ which\ is\ (User\ Cluster \bigcap Item\ Cluster)$$

Clustering Procedure

The next logical question is how can we find the user and item clusters to make our predictions accordingly. This is done by minimizing a loss function similar to previous algorithms

$$\sum_{items} \sum_{users} (r_{ui} - \widehat{r_{ui}})^2$$

New users and items

Note that unlike the previous 2 algorithms, this algorithm directly generalizes to new items and users. So if we are predicting a rating for a new user and a used item we return the average rating of the item. Similarly for a new item and old user, we return the rating average of the user. If both the item and the user are new we simply return the global average.

Adjustable parameters

For this algorithm, the user can choose the number of item clusters and the number of user clusters which are both defaulted to 3. The user can also choose the number of iteration the loss function minimization runs which is defaulted to 20 similar to previous algorithms (Hug, Co-clustering, 2015).

3.4 BENCHMARK

Since this dataset is not used in the research community for the purpose of collaborative filtering and thus there are no known published results, we will create our own benchmark. Creating a benchmark represents a more real world scenario where the data that one might be working with will be new data.

The benchmark used in our project is predicting a rating equal to the average rating of the dataset for all user-item pairs.

4 METHODOLOGY

4.1 DATA PREPROCESSING

As explained in the data exploration section, the data set is provided as a json file (fig XX). The first step was to parse this json file into a pandas data frame to facilitate working with data. Now that our data is in a data frame the next step was to remove the unnecessary features for our collaborative filtering problem. As a result we drop all the feature except the user identifier, item identifier and the rating. In particular we drop "reviewerName", "helpful", "reviewText" , "overall" , "summary", "unixReviewTime", "reviewTime" .Table 3 represents resulting data frame sample where reviewerID is unique user identifier, asin is the item identifier and overall is rating given by the user to the item.

TABLE 3 - SAMPLE FROM DATAFRAME

	reviewerID	asin	overall

0	A1KLRMWW2FWPL4	0000031887	5.0
1	A2G5TCU2WDFZ65	0000031887	5.0
2	A1RLQXYNCMWRWN	0000031887	5.0
3	A8U3FAMSJVHS5	0000031887	5.0
4	A3GEOILWLK86XM	0000031887	5.0

Other than dropping the irrelevant features, there is no need for more data cleaning. This is because our dataset is already formatted in a way where each user rated at least 5 items and each item has been rated by at least 5 users. This assures that the collaborative filtering algorithm has the correct input to be executed properly.

After having our data ready, we did some statistical analysis and exploration on the data some of which was reported in the data exploration part of this report

4.2 IMPLEMENTATION

4.2.1 LIBRARIES

One of the most time consuming and vital steps faced when targeting this problem is finding the right problem for the problem at hand. Throughout this phase, I researched and explored many libraries and actually experimented with few before reaching the final decision. The list of explored but not used libraries is represented in Table 4.

TABLE 4 - EXPLORED LIBRARIES FOR CF

Library name	Purpose	Why not
Crab	User-based and item-based CF	No longer maintained, incomplete documentation
Nimfa	Implements NMF algorithms that can be used for recommending	General purpose with little focus on recommender systems
GraphLab	Good documentation for recommender systems	General purpose, needs registration
LightFM	Focus on specific algorithms set	Not enough documentation, not a vanilla CF approach

What was chosen is a library called Surprise which has the merits of all the above and not any of their drawbacks. Surprise, a python library specifically crafted towards a wide variety of collaborative filtering algorithms including SVD, Co-Clustering and Baseline estimate. It is aimed toward students and researches so it is very well documented, easy to use, works out of the box and consistently maintained and improved.

4.2.2 INPUT DATA

Preparing input data

Surprise automatically builds the user-item ratings matrix from a text file such as a csv, built of users, items and ratings entries. Thus it integrates easily with pandas. So to prepare the data for this library, all I had to do is to save the pandas data frame prepared in the data preprocessing phase as a csv file.

Cross validation

One of the interesting features of Surprise is that it makes it very easy and encouraged to use cross validation. We simply call a split function on our data after it is read from the csv file and give this split function the number of folds or parts we want our data to be divided. This will be used to evaluate results more accurately later on. Suppose the we split our data in n folds. Then when evaluating a model, we train it on n-1 sets and test it on the remaining 1 set. This process happens n times. Each time a different fold is chosen as a test fold and the rest as a training fold. The evaluation metric reported is then the average evaluation metric of all the n trials.

4.2.3 Benchmark Implementation

Surprise library has convenient methods that allow its user to implement their own prediction algorithms. As explained in there documentation page (Hug, How to build you own prediction algorithm, 2015) we create a class that calculates the mean of the training set. All the other methods implemented in Surprise that are used for the evaluation of the model will apply to the benchmark model.

4.2.4 Algorithms usage

Surprise library implements all the algorithms that were presented in previous sections out of the box. The same data and folds are used as an input to all the algorithms to insure there is no data bias.

As a start all the algorithms are used out of the box without changing any of the default parameters. Table 5 below represents all the algorithms along with the 2 error measures that were used

TABLE 5 - RMSE & FCP FOR DIFFERENT ALGORITHMS

Algorithm	RMSE	FCP
Benchmark	1.1037	0.000
SVD	1.0698	0.5239
Baseline Estimate	1.0517	0.5286
Co-Clustering	1.1833	0.5118

4.3 Refinement

4.3.1 Choosing the model

From the results in Table 5 above we can see that the best model in both RMSE and FCP measure is Baseline Estimate. On the other hand Co-Clustering is the worst where it does worse that the Benchmark. SVD results are very close to Baseline estimates, although they are little worse.

Although it is counter intuitive that Baseline algorithms are doing better than other more complex algorithms, this can be explained by the fact that our user-item ratings matrix is very sparse. As explained in the paper “A Comparative Study of Collaborative Filtering Algorithms” (Joonseok Lee, 2012) it has been proven that on sparse matrices baseline algorithms tend to give better results that other complex algorithms.

However, since the SVD is generally a more solid approach and since the results between the Baseline and SVD algorithms are not very different. The choice of the appropriate algorithm for this model was SVD.

4.3.2 Optimizing the model

As discussed in the planning phase, there are 3 factors to be varied for this algorithm which are the regularization factor, learning factor and the number of iterations. Surprise does not have an built in way to try the different combinations of all the input variables and choose the best model accordingly similar to the GridsearchCV of sklearn. To overcome this problem I created an analogous function which integrates well with surprise. This function returns the FCP and RMSE value of each combination of the input parameters.

Now that it is easy to vary the parameters of the SVD algorithm, the next step was to decide how we will actually vary the parameters. First of all we decided to vary the learning rate and give it values of 0.002, 0.005, 0.01, 0.015 and 0.02 and monitor both RMSE and FCP to check the variation of these measures with respect to the learning rate. Next we also studied the effect of the change of the regularization factor on the same measures. The regularization factors tried were 0.05, 0.1, 0.2, 0.4 and 0.6. Note that both learning rate and regularization factors varies from 0 to 1 and the learning rates are normally small (<0.1).

Finally we tested the model on 20 and 40 iterations. The aim of all these parameter variation is to achieve the highest FCP and lowers RMSE values in a reasonable performance constraint.

Figure 3 below represents some represents the variation of RMSE and FCP when we change the learning rate for different regularization factors.

Regularization factor effect

The most striking feature about this graph is that the plots of RMSE and FCP are almost parallel. So as we increase the regularization factor the RMSE graph moves down and the FCP graph moves up for all learning factors. Note that the change between 0.04 and 0.06 is very slight in the FCP score where the peak almost did not move. It is small in the RMSE scores. So this shows 0.6 is a good saturation value for regularization factor

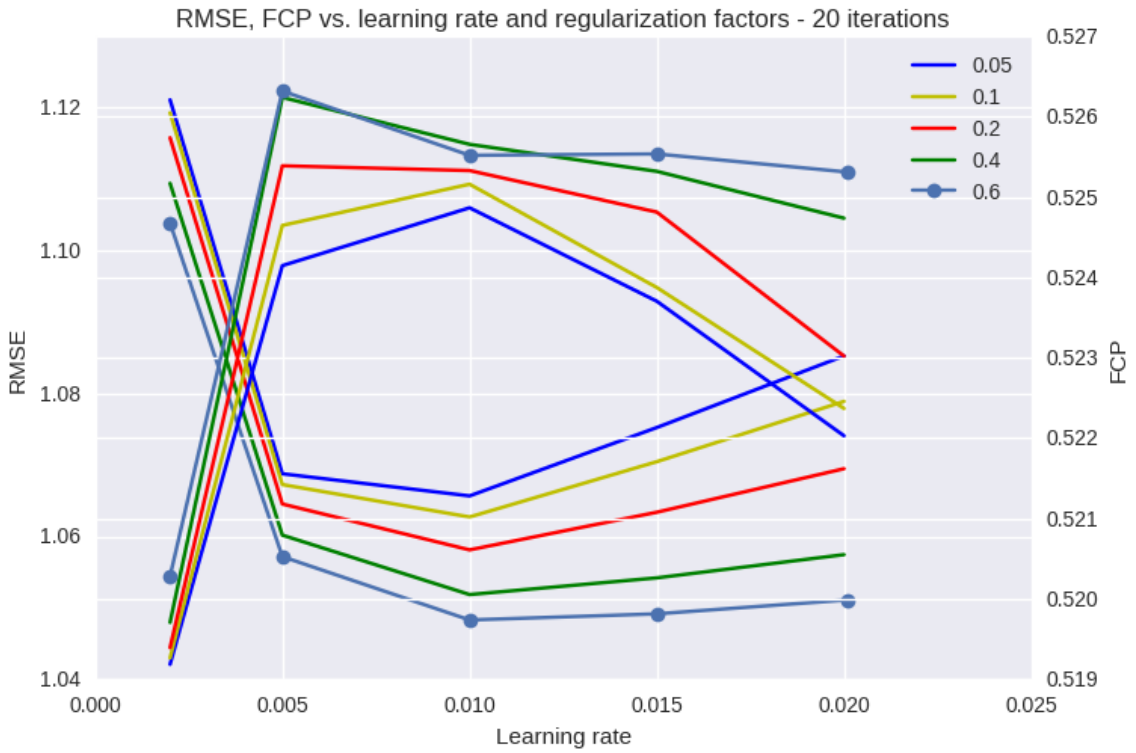

FIGURE 3 -SVD PARAMETERS VARIATION - 20 ITERATIONS

Leaning rate effect

The learning rate is more tricky where we can see that for very low learning rate of 0.002 shows very poor results. When increasing it to 0.005 the results of both RMSE and FCP are drastically improved with a peak performance on FCP. The peak performance of RMSE is at 0.01 learning rate. Afterwards, both RMSE and FCP get worse as we increase the learning rate.

Number of iterations effect

Next we do the same measurements but with 40 iterations instead of 20 for the stochastic gradient decent (SGD). The results are shown in Figure 4 below

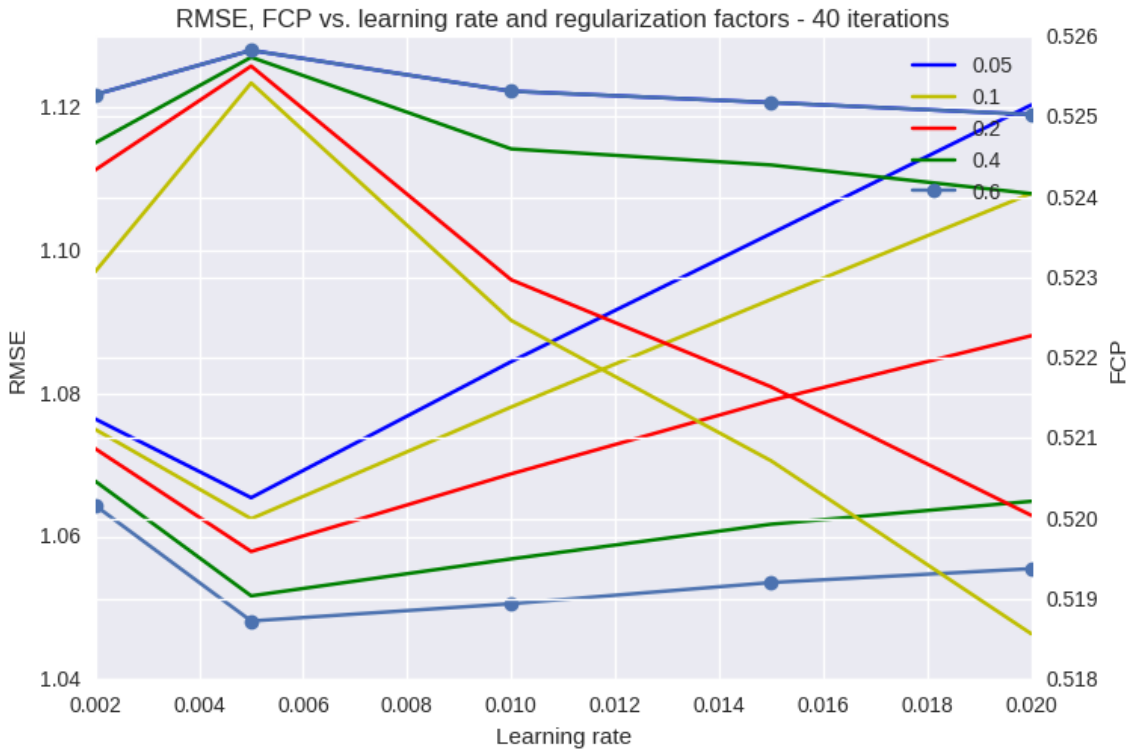

FIGURE 4 - SVD PARAMETERS VARIATION - 40 ITERATIONS

In this graph we can see a very similar behavior to the previous analysis with respect to the regularization factor. However something interesting happens in the analysis of the learning rates. We can see that when increasing the number of iterations the best performance for both FCP and RMSE was achieved at the learning rate of 0.05. This is due to the fact that when we decrease learning rates, the model needs more iterations to learn. When we gave it enough iterations, it achieved better results. We did not increase the number of iterations more because the tradeoff between performance and results improvement is not significant.

4.3.3 Final model

Thus the final model is an SVD model with 40 SGD iterations a learning rate of 0.05 and a regularization factor of 0.6

5 Results

5.1 Model Evaluation and Validation

5.1.1 Preparing the data

To evaluate the final model we studied how it performs on the test set compared to the actual rating. To do this, we saved all the predictions of the SVD algorithm in a pickle file and then we read the file into a pandas data frame which. The data frame contained the user id, item id, actual rating, predicted rating as seen in Table 6 below

TABLE 6 - TEST SET SAMPLE PREDICTIONS FROM DATAFRAME

	user_id	item_id	actual_rating	predicted_rating
0	A1121T1I50825Z	B008POAUA8	1	4.16594
1	A1XQ0F01CF84Y3	B008CIQZ5G	5	4.420822
2	A2Y0TX9P5GIHRY	B0002MB9GK	4	4.531579
3	A1Q4V5G4S00Q7M	B000GB1R7S	5	4.949796
4	A2KCVT2709IED0	B002QAK32C	4	4.55658

next we calculate the Mean absolute error (MAE) for each entry along with the number of ratings the user gave and the number of ratings the item got in the training set. Thus the sample becomes as shows in Table 7 below.

TABLE 7 - TEST SET SAMPLE PREDICTIONS FROM DATAFRAME WITH COMPUTED STATISTICS

	user_id	item_id	actual_rating	predicted_rating	user_num_of_ratings	item_num_of_ratings	mae
0	A1121T1I50825Z	B008POAUA8	1	4.16594	7	5	3.16594
1	A1XQ0F01CF84Y3	B008CIQZ5G	5	4.420822	32	3	0.579178
2	A2Y0TX9P5GIHRY	B0002MB9GK	4	4.531579	3	4	0.531579
3	A1Q4V5G4S00Q7M	B000GB1R7S	5	4.949796	24	16	0.050204
4	A2KCVT2709IED0	B002QAK32C	4	4.55658	3	14	0.55658

5.1.2 BEST AND WORST PREDICTIONS ANALYSIS

It is useful to study the best predictions and worst predictions of the model to understand when the model performs well and when it performs poor.

To do the best prediction analysis we took the sample where the MAE is minimum, which is zero in the data set. Then we calculated the average ratings given by the user and the average ratings given to the item in the training set.

TABLE 8 - BEST PREDICTION

Actual rating	5
Average rating given to the item	5
Number of ratings given to item	5
Average rating given by the user	5
Number of ratings given by user	6
Predicted rating	**5**

As can be seen in Table 8 above, since the SVD depends on the user and item vectors in the training set, and all the ratings and given by the user and given to the item is 5, thus it is reasonable that our model will predict a rating of 5 which corresponds to the actual value. It is to be noted also that as the number of items given by a user and to an item increase, the actual trust that is given to the results increase and the prediction are no longer a matter of chance.

On the other hand to do the worst prediction analysis, we took a sample where MAE is maximum with predicted rating of 5 and an actual rating of 1.

TABLE 9 - WORST PREDICTION

Actual rating	1
Average rating given to the item	5
Number of ratings given to item	2
Average rating given by the user	5
Number of ratings given by user	2
Predicted rating	**5**

Same analysis apply to the best case scenario above where our model made a reasonable prediction of 5 because all it saw is an average item and user rating of 5 on the training set as seen in Table 9. However, the actual number of training points for this user item pair is very low with 2 ratings each which makes prediction not trustworthy and thus the poor prediction.

5.1.3 OVERALL PERFORMANCE

The next step is to study the average performance of our model. To do this we studied what is the average MAE for each rating separately. The results are reported in Table 10 below

TABLE 10 - AVERAGE PER RATING

Actual rating	Predicted rating Average	MAE average
1	3.923888	2.923888
2	4.020919	2.020919
3	4.096324	1.096324
4	4.19345	0.19345
5	4.357332	0.642668

Knowing that the global average of the train set is 4.24591520858, we can see that the average predicted rating are all around this value. Although low ratings have lower overall average and high ratings have higher average, the values of the average shows a poor performance for the model on low ratings. Our model is very optimistic and tends to always give high ratings. This is not evident from looking at the RMSE alone due to the fact that the number of high ratings (4 and 5) is much larger than the number of low ratings and thus the small error of high rating hides the error on small ratings.

The overall performance analysis shows that the model is not very trust worthy because the model will predict high ratings and recommend items that the user might actually hate.

5.2 JUSTIFICATION

Going back to the global RMSE on the dataset, we can see that our model did better than the benchmark model improving the RMSE error by 5% and having a good FCP value as seen in Table 11.

TABLE 11 - SVD VS. BENCHMARK

	benchmark	SVD	% improvement
RMSE	1.1037	1.0482	5%
FCP	N/A	0.5258	N/A

The 5% change is not a bad improvement in the context of RMSE because in collaborative filtering, even small improvements is reflected by much better top–N recommendations (Wikipedia, 2016). The most famous example of improving RMSE values is the Netflix prize where the DVD rental company Netflix, gave a prize of 1 million dollars to improve RMSE ratings by 10% on its dataset (Wikipedia, 2016)

Nevertheless, the final solution is still not strong enough to solve the recommendation problem on this particular dataset. We cannot deploy this model with a big error on low rating to actually recommend items to user. The large errors is due to 2 main factors which is the sparsity of the matrix. The more ratings we have the better the performance of the SVD as explained in the best and worst predictions above. The second reason for bad results comes from the fact that the ratings are not balanced and we have far more high ratings than low ratings. Working on balanced dataset could improve performance and remove the bias of the model.

6 CONCLUSION

6.1 FREE-FORM VISUALIZATION

The histogram in Figure 5 shows the frequency of the predicted and the actual ratings. It confirms the analysis that we did previously where we said that our model does not predict low ratings. The predicted ratings (in blue) are almost non-existent for rating 1 and 2. Another interesting aspect of the graph is that both predicted and actual ratings have very close means around 4.25, however the standard deviation of the 2 is very far apart (lower standard deviation) where the standard deviation of the actual data is much larger the the predicted data. This explains why the predicted values are more centered around the mean.

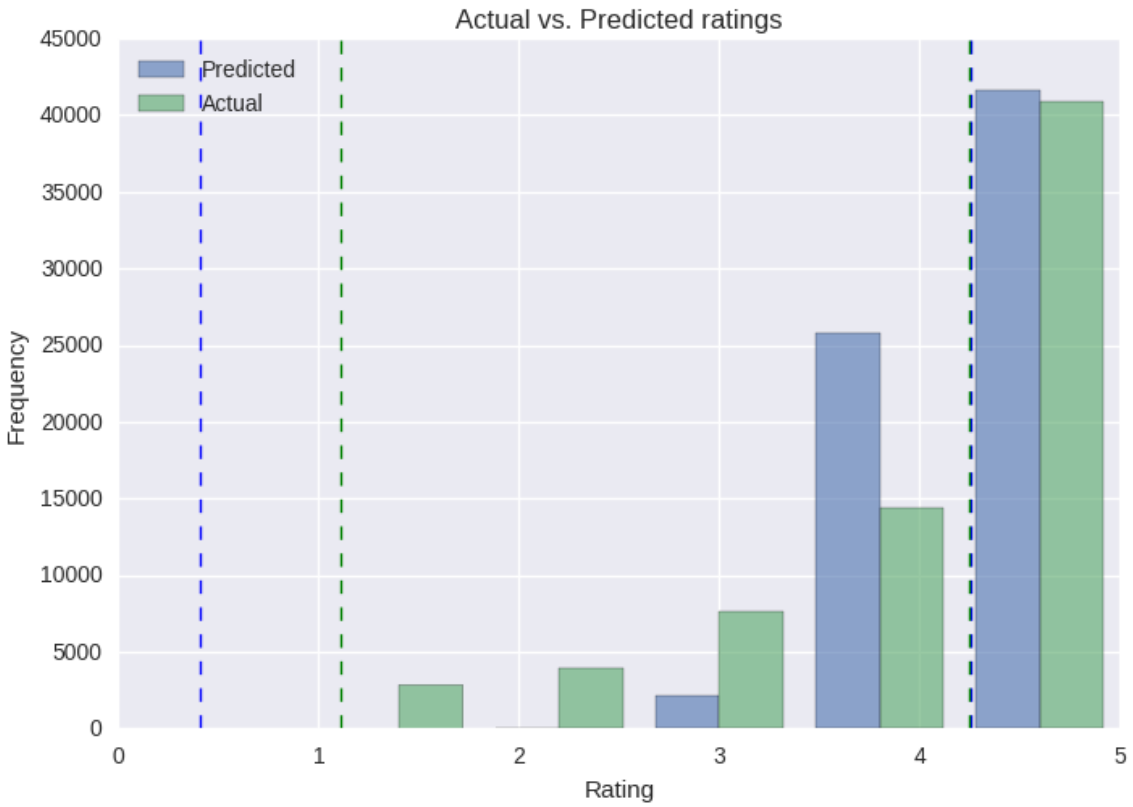

FIGURE 5 - ACTUAL VS. PREDICTED DISRIBUTION - 5 BINS

To understand better the relationship between predicted and actual values, we increase the number of bins of the histogram as shown in Figure 6 where an interesting property of the predicted values emerge. The SVD generated a normal distribution of the ratings whereas the actual ratings are not normally distributed. It can be concluded that to have SVD perform better we should apply it on a normally distributed dataset

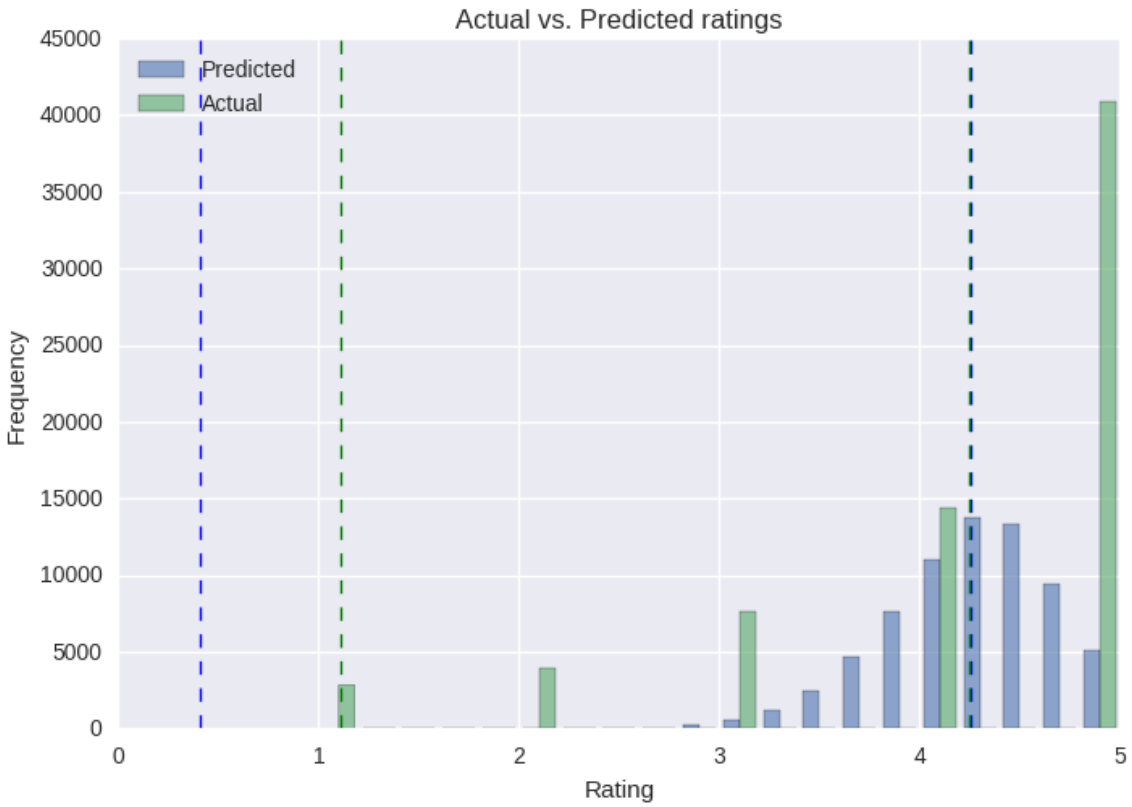

FIGURE 6 - ACTUAL VS. PREDICTED DISRIBUTION - 20 BINS

6.2 REFLECTION

6.2.1 DATASETS

After deciding that for the capstone project I want to work on a recommendation systems I started the process by searching for datasets are relevant to such problems. The idea was that I want something new that hasn't been widely studied to mimic real world situations as much as possible. The dataset explored were different in 2 axis. The first axis was the category of data such as books, movies, items, etc. And the second axis the structure of the dataset. Whether it is only ratings or items metadata. After a lot of research the choice fell on the Amazons Clothing ratings dataset because it is recent, it is extracted from real data and it is pre-processed so it reduces pre-processing time. In addition, it hasn't been widely studied.

6.2.2 RECOMMENDATION SYSTEMS

Exploring recommendation approaches went in parallel with exploring datasets because each dataset needs a specific recommendation approach. I explored Popularity based recommendations and personalized recommendations using collaborative filtering and content based filtering. The choice fell on collaborative filtering since it usually produce better results, ratings dataset that collaborative filtering use are widely available and the algorithms behind collaborative filtering are based on machine learning.

6.2.3 METRICS AND LIBRARIES

After deciding on the collaborative filtering I began the research on collaborative filtering libraries in python. A lot of libraries were explored and each library had its own algorithms that optimizes its own

metrics. At this stage I have decided to treat the problem as a best predictions to ratings problem and thus Metrics such as RMSE and MAE were chosen over other metrics such as precision and recall for top-N recommendations. Accordingly, the library chosen was based Surprise which has algorithms that does this task in particular and optimize such metrics

6.2.4 Implementation process

Next I started with the implementation where the first step was to explore the data to have a clear picture of what I am working on. After doing some statistical analysis on the data, the next step was to work with Surprise library where I thoroughly read documentation, explored some implementation and applied some of the algorithms to my dataset as a proof of concept that it works.

Next, I went back to research to understand the selected algorithms well where I read some papers on the topics. Based on my readings and the results of the algorithm on the dataset, I chose SVD to optimize and adopt as the final solution. In the optimizing task I had to borrow some ideas from sklearn and try to apply them to the project at hand to find the best set of parameters. I also spent a fair amount of time in the visualization part trying to study the effect of each factor on the performance.

The last step was to evaluate the final model, understand its pros and cons and understand why it behaves the way it does to see how can we improve on it in the future.

6.2.5 Overall Reflection

Since the topic of recommendation system is totally new to me that was not explored in the courses and projects, the project took a big amount of time to complete. Much more than what is predicted by Udacity. However, it was very educational where I have learned a lot of key skills. Both soft skills and technical skills. I have learned a lot about recommendation systems and the different algorithms and approaches. I have learned how to deal with real world data. I practiced reading research papers and extracting high level concepts from all the math. Finally one of the most important points I learned is the importance of finding the right library with the right documentation and community support and not to re-invent the wheel. On the soft skills part, I have experienced the importance of persistence to achieve a task, the importance of staying motivated.

One of the most difficult aspects of this project was finding the right dataset. The dataset I chose, was a very tough because of its sparsity and thus the improvements and predictions our model achieved wasn't radical. For the future I would have chosen the dataset more wisely because having good data is one of the key aspect of any machine learning problem. Algorithms alone are not enough.

6.3 Improvement

There are several directions that can be explored to improve this model. First of all we can work with algorithms that consider non-rated items as negative ratings and thus solve the sparsity problem. It could be interesting also to explore ensembled solution where a certain prediction comes from several algorithms. One other direction that we can explore is targeting recall and precision metrics instead of predicting ratings. Such systems and metrics are said to perform better when dealing with recommending top-N items. Most of the times we do not care about predicting low ratings correctly.

7 WORKS CITED

Funk, S. (2006, December 11). *Monday, December 11, 2006. Netflix Update: Try This at Home*. Retrieved December 21, 2016, from Project Sifter: http://sifter.org/~simon/journal/20061211.html

Hug, N. (2015). *Co-clustering*. Retrieved December 27, 2016, from Welcome to Surprise' documentation!: http://surprise.readthedocs.io/en/latest/co_clustering.html

Hug, N. (2015). *How to build you own prediction algorithm.* Retrieved December 26, 2016, from Welcome to Surprise' documentation!: http://surprise.readthedocs.io/en/latest/building_custom_algo.html

Hug, N. (2015). *Matrix Factorization-based algortihms*. Retrieved December 22, 2016, from Welcome to Surprise' documentation!: http://surprise.readthedocs.io/en/latest/matrix_factorization.html#unbiased-note

Hug, N. (2015). *similarities module*. Retrieved December 22, 2016, from Welcome to Surprise' documentation!: http://surprise.readthedocs.io/en/latest/similarities.html

Joonseok Lee, M. S. (2012). *A Comparative Study of Collaborative Filtering.*

Koren, Y. (2010, January). Factor in the Neighbors: Scalable and Accurate Collaborative Filtering. *ACM Transactions on Knowledge Discovery from Data, Vol. 4, No. 1, Article* .

McAuley, J. (n.d.). *Amazon product data*. Retrieved November 20, 2016, from Amazon product data: http://jmcauley.ucsd.edu/data/amazon/

Thomas George, S. M. (n.d.). A Scalable Collaborative Filtering Framework based on Co-clustering.

Vryniotis, V. (2013, October 27). *Tuning the learning rate in Gradient Descent*. Retrieved December 21, 2016, from DatumBox: http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/

Wikipedia. (2016, December 18). *Netflix Prize.* Retrieved December 26, 2016, from Wikipedia: The Free Encyclopedia: https://en.wikipedia.org/wiki/Netflix_Prize

Yehuda Koren, R. B. (2009). Matrix Factorization Techniques for Recommender Systems. *IEEE computer society* , 42-49.