Master in applied sciences
Computer science and engineering
Year 2018-2019

Machine Learning

Third Project: movie rating prediction

ELEN0062-1
GEURTS Pierre
WEHENKEL Louis

MOITROUX Olivier
MIFTARI Bardhyl
HOCKERS Pierre

Liège

December 2018

Abstract

In this project, multiple databases were given containing informations about a set of movies and users as the well as the ratings each user gave for some movies. The goal was to predict the rating a user will give to a movie he has yet to see.

1 Introduction

To achieve this task, we came up with multiple ideas and ways to solve the problem. In the following, we explain how we proceeded from the start of the project until the final submission. We choose to work with regressors and not classifiers as we don't need to have a round number as rating for our prediction and the score is equal to the mean square error made (we minimize the difference between our output and the rating obtained by using a float instead of an integer). Furthermore, a classifier considers each "class" (rating = 1,2,3,4,5) as independent from one another. However, with a regressor we needed sometimes to do a scale back on values bigger than 5 or lower than 1. We submitted 4 files with this report:

- `MatrixFactorizationSVD.py`: Our final implementation
- `MatrixFactorizationSVD.py`: An overall good implementation using SGD that could have been improved if more time.
- `dataAnalysis.py`: A script to analyze the data.
- `FeatureSelectorOLD.py`: part of a code used to select and process some features. We don't submit the complete code as we don't use it for our final kaggle submission anyway.

2 Toy_example

In the toy example, the learning matrix was built as follows:

$$X_{LS} = \begin{bmatrix} r(u_1,m_1) & r(u_1,m_2) & ... & r(u_1,m_m) & r(u_1,m_1) & r(u_2,m_1) & ... & r(u_n,m_1) \\ r(u_2,m_1) & r(u_2,m_2) & ... & r(u_2,m_m) & r(u_1,m_2) & r(u_2,m_2) & ... & r(u_n,m_2) \\ ... & ... & ... & ... & ... & ... & ... & ... \\ r(u_n,m_1) & r(u_n,m_2) & ... & r(u_n,m_m) & r(u_1,m_m) & r(u_2,m_m) & ... & r(u_n,m_m) \end{bmatrix}$$

where r is the rating for a given (u_i, m_j) pair where u_i is the user 'i' and m_j is the movie 'j'. The learning matrix was then given to a *DecisionTreeRegressor* with as output for the learning matrix the rating each (u_i, m_j) pair received. Then, by using the model on the testing data, the toy_example submission was made. In the beginning, we just changed the regressor to see if this kind of data was enough for our model.

2.1 Testing with different algorithms

By testing different algorithms we obtained the results presented in table 1 on the Kaggle leaderboard.

As mentioned, the toy submit uses a *DecisionTreeRegressor*. By changing the regressor to a *RandomForestRegressor*, the score was divided by 2. *RandomForestRegressor* is a kind of

	DecisionTree	RandomForest	Adaboost	LogisticRegressor	LinearRegressor
score	2.45420	1.27805	1.54390	1.24437	0.97520

`Table 1` – Score of different models on toy example matrix representation

bagging which generates n *Decision Trees*, each one slightly overfitting the data. It takes the mean of all those decision trees to create the model. By definition, it behaves better than a simple *Decision Tree* does. We tried a 5 iterations Adaboost on a *Decision Tree* model, enhancing the results but not enough. We also combined Adaboost and *RandomForestRegressor* but the results were not conclusive and the computation time was too long (several hours). We turned to *Logistic* and *Linear Regressor*. *Linear Regressor* achieved promising results decreasing nearly by 2.5 the score obtained with a *DecisionTreeRegressor*. Enhancing the model with *Adaboost* and *Bagging* didn't bring much. We concluded that we reached the limits of what could be achieved by this representation.

3 Data analysis

Before even attempting to design an effective algorithm to predict users' ratings, we first made an analysis of the data available. In this way, we implemented a dedicated script, `data_analysis.py`, to better visualize all the data that we had to manipulate and to design the best possible approach. We got inspired by the good analysis done in [1] and tackle the learning curve of pandas thanks to the datacamp tutorial[6]. This script analyzes the data in several steps. For this purpose, we heavily used the library *pandas* and its useful class `dataframe`. Firstly, we analyzed the age repartition of the users data thanks to the histogram in figure 1[1].

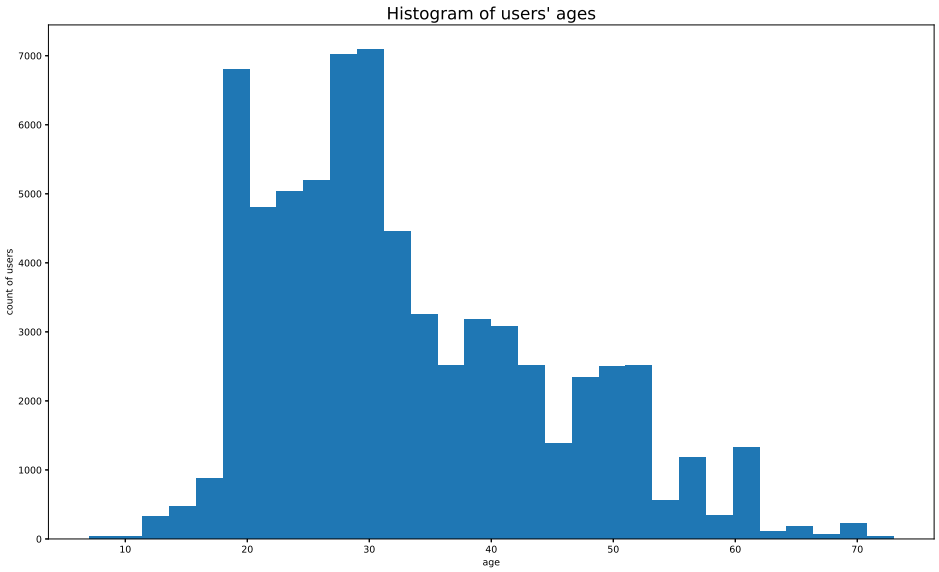
Next, we grouped the users by age categories in order to compute the number and mean of ratings per category. The result, as well as all the others, can be observed in console by running `data_analysis.py` or by taking a look in the appendix page 12. It turns out that children of 0 to 9 years old are the most generous in their ratings. The more critique ones are between 10 and 29 years with hint of generosity coming back between 50 and 59 years old.

We discovered that the data counts 69453 ratings with a global mean of 3.534 and a standard deviation of 1.124. The most watched movie has been watched 414 times with an impressive mean of 4.355. All the films have been watched at least once but their ratings are not relevant at all as can be inferred from the next print in console and which highlights the best rated movies. Actually, many films that have almost never been watched are considered as the best rated movies, so ratings should be taken with real care.

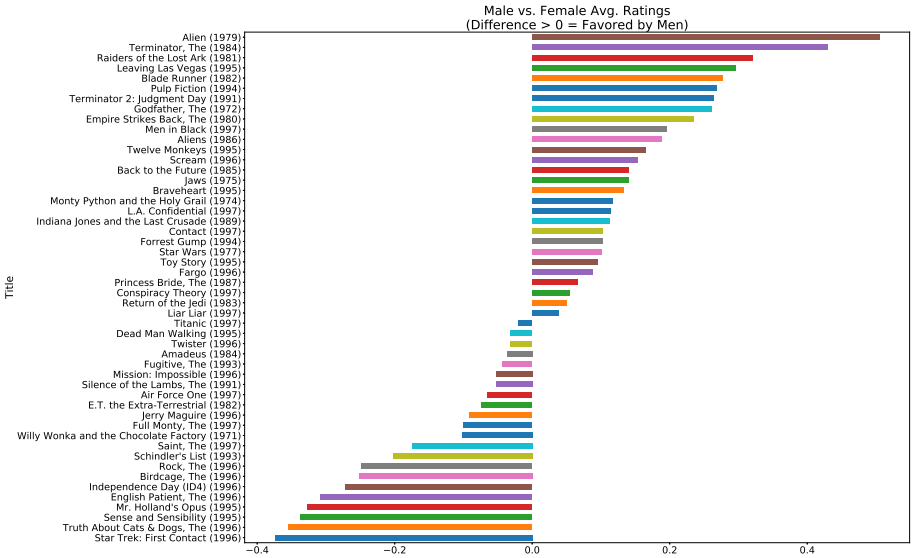
We then analyzed the impact of the sex on the ratings by showing the difference of the mean of ratings between males and females for each movies and plotted them (figure 2) for the 50 most watched movies.

Finally, we categorized the users by age categories (decades) in order to observe the impact of the age on the genre of movies watched. Quite unsurprisingly, no children watched horror films but the table didn't really highlight a general trend. A kind of feature selection was thus necessary to get the most interesting one.

[1]Increasing the font of the axis didn't work but the eps format allow lossless zoom.



`Figure 1` – User's age histogram



`Figure 2` – Delta of rating compared to average rating for males and females for the 50 most watched movies.

4 First attempt: raw features processing and selection

After realizing that the results we got from the simple application of common algorithms were not enough, we decided to switch for a more complex and thought-through method. We wanted to take into considerations the different parameters that might or should influence the rating of the users. There are 2 different things to do here : features selection and features extraction. Features selection means evaluating the most important and impactful features based on some criteria, while features extraction means transforming the full data to project it into a new feature space with lower dimension. PCA, and SVD are examples of this, as we will see later.

4.1 Processing the features

Before starting the feature selection process itself, we did several modification to the dataset we received. Some modifications were due to simple reorganizing of the data, while others were due to problems inherent to the data we received. We classified men and women by values such as man = 1 and woman = 0. We also thought of considering the difference between the release date of a given movie and the birth date of a given person as someone born in the 60's is more likely to like a movie released while he was young than a youngster that was born years after the movie was released. While trying to process all the given data, some problems occurred :

- the "movie_data.csv" wasn't in UTF-8 format which generated several errors.We therefore modified a file to avoid this problem.
- the release dates of each movie were badly written and therefore difficult to extract (we took the year of the video release date)

In addition, in our will to simplify the representation of data, we thought of representing the genres of the movie by a binary number, where a 0 at an index would mean that the movie doesn't correspond to the movie genre corresponding to the same index in the genre vector, and a one would mean the opposite. However, this representation would work well for two types far from one another but badly for two movie types next to one another especially the one representing the first bit and the second one.The algorithm would try to find a correlation between the two as they are close to one another. We therefore dropped this idea. We also grouped people ages by decades for a first proof-of-concept.

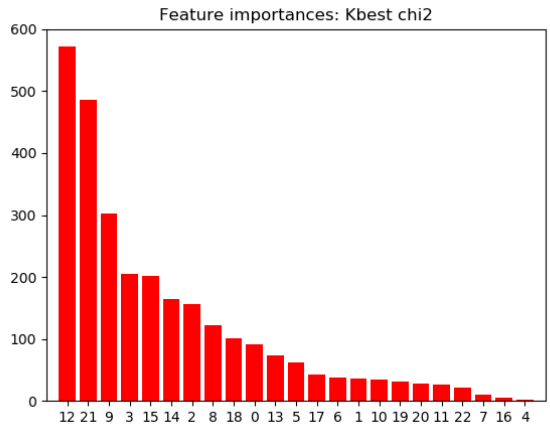
4.2 Selecting the 5 best features among all

We did this step in two times : first, we tried to simply "guess" which parameters would be useful, and extracted them using the *pandas* library. The results were not very conclusive from the start. We therefore didn't even submit it. We then used various algorithms available in the *scikit-learn* library to extract the k best features possible. Mainly, we used the `ExtraTreesClassifier` and `SelectKBest`. They gave us slightly different features. Let's note that we decided arbitrarily to stick to the 5 best features as we learned, by searching on the internet, that limiting the amount of features was good practice to limit the overfit. We summarized the features selected by the two methods in table (2) according to different error estimating methods. The Kbest features selection will attribute a score to each features and select the K features that got the best score. These scores are computed differently depending on what method is attributed. We tried chi square evaluation, f_classif and f_regression. $ExtraTreesClassifier$ works in a similar way, attributing scores to features according to their importance. This score is evaluated as follow : in a tree, the depth of a feature is linked to the amount of elements it has an influence on. This can be considered as an indicator of the importance of the feature. Since this class uses several trees on different parts of the train sample, one can average the importance of the features.

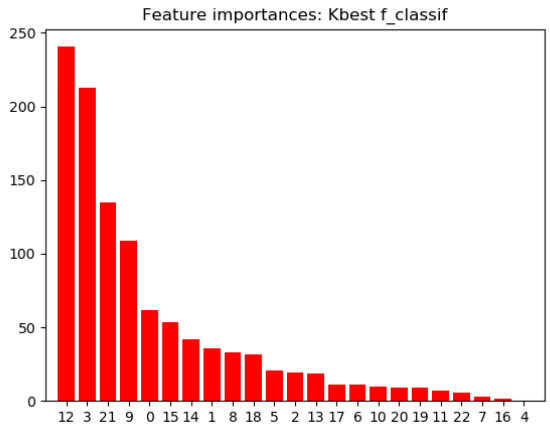
We also plot the score of each features (with standard deviation for random forest) for each technique. The results can be observed in figures (3a),(3b),(3c) for Kbest. Figure (3d) gives the random forest results.

	1st Feature	2nd Feature	3rd Feature	4th Feature	5th Feature
Kbest with chi2	Drama	War	Comedy	release date	Horror
Kbest f_classif	Drama	release	War	Comedy	age
Kbest f_regression	Drama	release	War	Comedy	Horror
Random forest	release date	occupation	age	Romance	gender

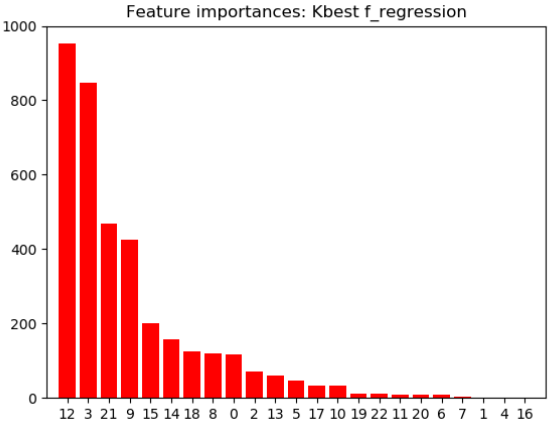
`Table 2` – Features selections results for different algorithm (and evaluations methods for Kbest)



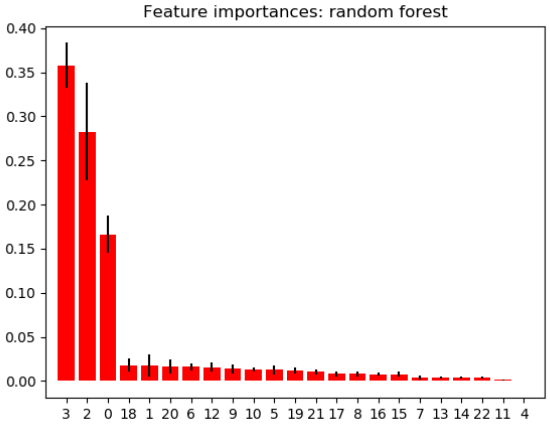
(a) KBest with chi-squared evaluation.



(b) KBest with `f-classic` evaluation.



(c) KBest with `f-regression` evaluation.



(d) Random forest selection.

`Figure 3` – Features score according to different selection techniques

4.3 Results with raw feature selection

In the end, we chose the best features selected by the random forest, since the previous experiments we came across on the internet seemed to get better results with it. The features we selected were therefore the release date, the occupation, the user's age, whether the movie belonged to the Romance genre or not, and the gender of the user. This didn't give good results neither, they were far worse than simpler method (for example our linear regressor that scored 0.97520), and never scored better than 1.60478(which was achieved with a `RandomForestRegressor` previously). There was a really slight improvement when using only

the three best features (still with random Forest) which gave 1.58220, but it was still bad.

4.4 Towards another approach

Due to the unsatisfying and disappointing results obtained with the above-mentioned approach, we decided to take a completely different approach to tackle the problem. An alternative to our first try would have been to stack horizontally all the (processed) features available to the raw matrix of ratings, and to train the algorithm like so. However, it is likely we would have encountered problems such as difficult model selection or tuning, over-fitting or too sparse learning matrix. Some temporal-effect features could also been added like scores depending on the number of ratings per user and so on.

Instead, we searched the internet in order to find inspiration to implement a technique that would use the correlation between users with similar behaviours and tastes. This kind of techniques turns out to be called collaborative-filtering and got impressive success in the Netflix prize challenge despite their relative simplicity. Modeling the bias on the ratings turns out to be an excellent idea to lower the error on the prediction.

A more theoretical explanation of the techniques we then used is available in the appendix at section 11.

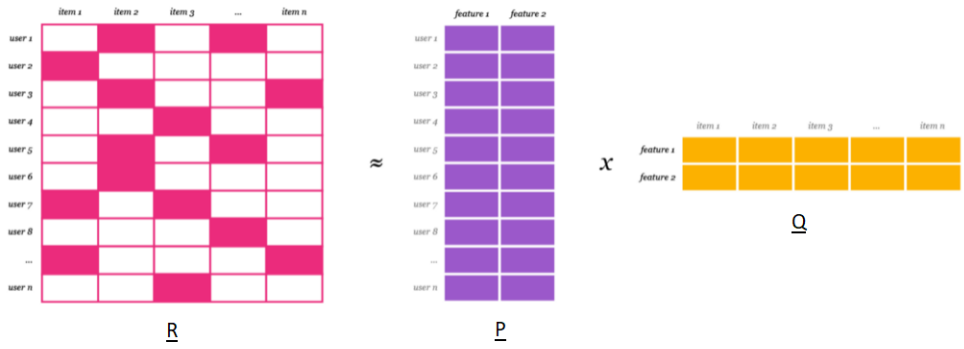
5 Second approach: SGD

We changed the way we proceeded. We consider each user as a linear combination of the other users and each movie as a linear combination of the other movies. We applied stochastic gradient descend on the following matrix, M,

$$M = \begin{bmatrix} r(u_1, m_1) & r(u_1, m_2) & ... & r(u_1, m_m) \\ r(u_2, m_1) & r(u_2, m_2) & ... & r(u_2, m_m) \\ ... & ... & ... & ... \\ r(u_n, m_1) & r(u_n, m_2) & ... & r(u_n, m_m) \end{bmatrix}$$

using the same notation as above but we generated the full matrix where we have the rating of each movie for each user (zero if no rating available). We obtained a score of 0.96137.

It will create two vectors p and q filled with random numbers from a normal distribution that will approximate the ratings we already know by minimizing the error between those ratings and a linear combination of their n_factors and therefore also approximate the values we don't know (zeros) in the rating matrix.



`Figure 4` – Stochastic gradient descent approximation of the matrix R

However, the matrix's sparcity is still a problem if we want to enhance our model. We found two ways of handling this problem :

- Use an heuristic to approximate each rating
- consider sub-matrices for each rating

5.1 Heuristic

Inspired by [3] and [5], we considered that each rating can be separated in multiple parts:

- M = mean (mean of all the rating for all the movies)
- $user_c$ = user contribution (difference between M and the mean of the values the user i gives)
- $movie_c$ = movie contribution (difference between M and the mean of the ratings given to the movie j)
- $usermovie_c$ = user-movie interaction

The mean, user contribution and movie contribution can be calculated easily thanks to the panda library(with a Dataframe). We created specific functions to find those contributions. We get the user-movie interaction as follows :

$$M - user_c - movie_c = usermovie_c$$

We trained an SVD algorithm on the $usermovie_c$ obtained for the learning set and predict it for the training data. Then, we rebuild our rating by doing the steps backwards. The score obtained was 0.95097. This method seemed promising. However, we discovered the Surprise Library which has an SVD algorithm with SGD already integrated at this point, so our functions were not useful anymore. This library was specialy designed for movie ratings and several algorithms were tested.

5.2 Sub-matrices

For each user in the testing matrix, we create the following matrix,

$$M = \begin{bmatrix} r(u_1, m_1) & r(u_1, m_2) & ... & r(u_1, m_m) \\ r(u_2, m_1) & r(u_2, m_2) & ... & r(u_2, m_m) \\ ... & ... & ... & ... \\ r(u_n, m_1) & r(u_n, m_2) & ... & r(u_n, m_m) \end{bmatrix}$$

where u_1 is the user considered, u_2 *to* u_n are the n-1 users who watched most movies in the set of movie $[m_1...m_m]$. The set of movies considered is the sum of the set of movies rated by u_1 and the set of movies a rating was expected. We applied SGD but the results were not conclusive ($rmse = 2.46314$ on public leaderboard) as the values generated were too random. Furthermore, some users didn't watch enough movies or only watched movie rated by few(sub-matrix is still sparse), which makes this method error-prone.

6 Third and best approach: SVD_{++} unbiaised

Next, we started to use the SVD decomposition. We first began by implementing all the functions to remove the bias in the rating for better training (and to to put it back afterwards). However, during this process, we found the perfect library to suit ours needs: *Surprise*. Indeed this function already offer an *SVD* algorithm with rating-bias removal. We achieved great results with SVD with a RMSE of 0.87154 in the public leaderboard. Finally, we combined SVD_{++} with *SGD* and cross-validation to obtain our best score of 0.85003 in the public leaderboard. SVD_{++} is the singular value decomposition ++ algorithm (same as SVD but incorporates implicit feedback). SVD_{++} achieved the best score with the features exposed in table 3:

Parameters	n_factors	n_epochs	lr_all	reg_all
SVD_{++}	30	50	0.006	0.09

`Table 3` – Best tuned parameters for SVD_{++}

RMSE	Pub. lead.	Priv. lead.	Avrg. on 5 c-v trainset
SVD_{++}	0.85033	0.83022	0.9350

`Table 4` – Results with the tuned parameters of table 3

where n_factors is the number of factors used to compute the matrix P and Q, n_epochs is the number of iterations in the SGD, lr_all is learning rate and reg_all is the regularization term for all parameters. Tuning the parameters was done thanks to the class `gridSearchCV`[2] which takes as an input a set of parameters for optimization. Each time, the current tuple of parameters was scored with root mean square error averaged on 5 folds of the training set. One single training already took 11 minutes 08 seconds, so it unsurprising than finding the right tuned parameters took some hours of computing. On the other hand, SVD was considerably faster with training time of more or less 1 minute and 20 seconds. Let's note that we stored

[2]cross-validation for grid search

all the scoring information (standard deviation, ...) of each optimization iteration in a pandas dataframe. This dataframe can then be analyzed to better understand the behaviour of each parameter. We only used this dataframe one or two times in debug mode on pycharm, especially to check the standard deviation of the root mean squared error over the cross-validation process.

7 Fourth but very good approach: KNN unbiaised

In order to implement another collaborative filtering, we also tried to feed to the `gridSearchCV` with several K-NN algorithms already implemented in the *Surprise library*.

The surprise library gives the following set of built-ins algorithms:

Movielens 100k	RMSE	MAE	Time
SVD	0.934	0.737	0:00:11
SVD++	0.92	0.722	0:09:03
NMF	0.963	0.758	0:00:15
Slope One	0.946	0.743	0:00:08
k-NN	0.98	0.774	0:00:10
Centered k-NN	0.951	0.749	0:00:10
k-NN Baseline	0.931	0.733	0:00:12
Co-Clustering	0.963	0.753	0:00:03
Baseline	0.944	0.748	0:00:01
Random	1.514	1.215	0:00:01

`Figure 5` – Algorithms' root mean square error, mean absolute error and computation time on a 5-fold cross-validation procedure.

As our ratings contain a little more than 69k movie ratings theses algorithms seemed very efficient. We can clearly see that KNN baseline, SVD++ and SVD are the most efficient ones.

In the surprise library, multiple KNN algorithms were available but the one that worked the best for our particular database was KNN baseline. All in all , this is our second choice method. Tuningg $KNN_{baseline}$ on 12 folds took only 1 minute and 47 seconds which is far less than SVD_{++} and even SVD that were both trained on 5 folds and with fewer tuning parameters.

Parameters	k	Similarity measure	User/item based C.F.
$KNN_{baseline}$	41	Pearson	Item based

`Table 5` – Best tuned parameters for $KNN_{baseline}$

RMSE	Pub. lead.	Priv. lead.	Avrg. on 12 c-v trainset
$KNN_{baseline}$	0.86768	0.84704	0.9281

`Table 6` – Results with the tuned parameters of table 3

Finally, it turned out that item-based collaborative filtering offered better results than user-based collaborative filtering on each test performed (see table 6). Other similirity measures have been tried like , for example, the conventional cosine similarity.

8 Improvements

Lots of improvements can be found. First, we could combine our final model with another that works completely differently, one that is only based on user and movie features to find the ratings. Such an hybrid approach of content-based filtering and collaborative filtering may give even better results. Secondly, some ideas we had were not totally explored such as making sub-matrices easier to optimize for SGD. Thirdly, we could work on a much more sophisticated heuristic than the simple one we implemented. This report could also incorporate bar graphs to represent the performance of all our tests as well as the associated standard deviation on rmse. We could have detailed further the set of parameters that have been tested and plot the rmse according to them. This is however beyond the scope of this third project.

9 Conclusion

For this project, we first started by playing around the `toy_example` code and tweaked one or two models. We also investigated some way to improve the score with bagging and adaboost techniques. After reaching the limit of this implementation, we dived into an analysis of the data to orient ourselves towards a good way of apprehending the problem. Making a raw feature selection by checking the correlation between each features didn't bring satisfying results. We then oriented ourselves towards collaborative filtering techniques. More specifically, we discovered that removing the bias in the rating had a huge impact on the prediction. After some tests with SGD, we discovered we were on the right path. Matrix factorization was finally the right solution and best results were obtained by training on SVD_{++} on un-biaised ratings despite a longer training time. Top-k collaborative filtering with k-nn methods also scored very well with a much faster training time, making it our second best choice. However, these methods seemed to reach their limits and other methods like, perhaps an hybrid approach of content-based filtering and collaborative filtering could offer even better results. Our submission with SVP_{pp} ranked no 7 and no 10 in the private and public leaderboard respectively.

10 References

[1] Greg Reda, *Using pandas on the MovieLens dataset* October 2013

[2] Ethan Rosenthal, *Intro to Recommender Systems: Collaborative Filtering*, November 2015

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. *Matrix factorization techniques for recommender systems.* 2009.

[4] Simon Funk approach on sparsity problem. Blog

[5] Edwin Chen. Blog on winning the netflix prize

[6] Datacamp tutorial on pandas

[7] *Scikit-learn* library documentation

[8] *Surprise* library documentation by Nicolas Hug

[9] Pandas dataframe manipulation tutorial

[10] Nicolas Hug blog on matrix factorisation

Appendix

11 Collaborative filtering in a nutshell

For this project, we focused on three categories of collaborative filtering: user-collaborative, item-collaborative filtering and matrix factorization. The first one aims at building a similarity matrix between any two pairs of users, while in the second case, the matrix consist of some distance metric that measure the similarity between any two pairs of items (movies in our case). The third one aims at determining the preferences of a user based on a small number of hidden factors.

11.1 Our first implementation: Matrix factorization

Let's consider the matrix of ratings R. Each row of the matrix corresponds to a given user, and each column corresponds to a given item. Of course, the matrix is (very) sparse and the goal is to guess the empty slots. To condense the information, we can factorize the matrix R with SVD (Singular Value Decomposition), a dimensionality reduction technique. Matrix reduction limits the accuracy-loss on results when the data is noisy and may allow the use of statistical methods that can't be used if not enough observations are available. SVD is a technique that decomposes the matrix R into the best lower rank approximation of the original matrix R. Like stated by Nicolas Hugg, PCA on the matrix R will give us typical users. PCA on the matrix R^T will give us typical movies. SVD is PCA on R and R^T, in one shot that gives U and M by factorizing R into three matrices:

$$R = M\Sigma U^T \tag{1}$$

such that a row p_u of M represents the affinity of user u for each of the latent factors and a column q_i of U^T represents the affinity of the item i for the latent factors. The number of latent factor thus become a tuning parameter for us. Ignoring the diagonal matrix Σ for the seek of simplicity of the explanation, the rating can then be obtained by making the dot product:

$$r_{ui} = p_u \cdot qi \tag{2}$$

, r_{ui} being the rating for the pair $(u = user_{id}, i = movie_{id})$.

Two techniques can be used to tackle the sparcity problem: filling the missing entries of R with some simple heuristic or finding all the vectors p_u and q_i such that $r_{ui} = p_u \cdot q_i$ for all u and i. Basically, we want to optimize the non-convex optimization problem

$$min_{p_u,q_i} \sum_{r_{ui}\in\mathbb{R}} (r_{ui} - p_u \cdot q_i))^2 \tag{3}$$

proposed by [5]. This can be done via Stochastic Gradient Descent.

11.2 User- and item-based filtering

a General explanation

Let's consider a particular user. The goal is to find users that are similar to that user based on similarity of ratings, so as to be able to predict correctly the future rating.[2] Formally, we

predict that a user's rating u for movie i is given by the weighted sum of all other users' ratings for item i where the weighting is computed thanks to a similarity function that takes as input each user and the considered user u, normalized by the number of $r_{u'i}$ ratings.

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u, u') r_{u'i}}{\sum_{u'} |sim(u, u')|} \tag{4}$$

Likewise, the item-similarity matrix will measure the similarity between any two pairs of items. Item-based filtering has, however, two advantages over the user-based filtering: it scales better with time (movie categories doesn't change in opposition to taste f.e.) and it may be computationally cheaper because there are less movies than items. It turns out that item-based filtering worked the best for us.

b Bias subtracted collaborative filtering

Subtracting each user's average rating when summing over similar user's ratings can reduce the bias as not all users are equal. Some may be generous while some may be very critique. Thus, to reduce the prediction error, it is a good idea to train the model on a as unbiaised as possible rating. Then, the bias of the user (here, simply r_u) is added back at the end.

$$\hat{r}_{ui} = \overline{r_u} + \frac{\sum_{u'} sim(u, u') r_{u'i} - \overline{r_{u'}}}{\sum_{u'} |sim(u, u')|} \tag{5}$$

where $\overline{r_u}$is user u's average rating.

c Our second implementation: combining top-k and bias-substracted filtering

A popular similarity function used to build similarity matrices is the cosine similarity that uses the cosine of the angle between the two users' vectors. However, our experiments highlighted the fact that the pearson similarity is always better. In order to implement user- and item-based filtering, we used the *Surprise* library, and more specifically, a k-nearest neighbors-based technique that took into account a bias evaluation.

12 Output in console of `dataAnalysis.py`

`dataAnalysis.txt`

```
------------
Age of user analysis:
------------

Plotting age distribution:


Age categories:

                rating
                  size      mean
age_categories
0-9                 31  3.838710
```

```
10-19            5678  3.491899
20-29           27466  3.475643
30-39           17936  3.556590
40-49           10382  3.589963
50-59            6021  3.641920
60-69            1794  3.630992
70-79             145  3.613793
------------
Rating analysis:
------------

            rating
count  69453.000000
mean       3.533843
std        1.124049
min        1.000000
25%        3.000000
50%        4.000000
75%        4.000000
max        5.000000
------------
Movie analysis:
------------

Most watched movies:

                                rating
                                  size      mean
movie_title
Star Wars 1977                     414  4.355072
Fargo 1996                         361  4.177285
Contact 1997                       347  3.847262
Liar Liar 1997                     343  3.148688
Return of the Jedi 1983            343  4.037901
Scream 1996                        333  3.474474
English Patient, The 1996          310  3.641935
Independence Day ID4 1996          308  3.405844
Air Force One 1997                 304  3.598684
Toy Story 1995                     300  3.936667
...                                  ...       ...
Turning, The 1992                    1  3.000000
Beans of Egypt, Maine, The 1994      1  4.000000
Scarlet Letter, The 1926             1  2.000000
When Night Is Falling 1995           1  3.000000
Time Tracers 1995                    1  2.000000
Babyfever 1994                       1  1.000000
Golden Earrings 1947                 1  3.000000
BGood Man in Africa, A 1994          1  2.000000
Salut cousin! 1996                   1  1.000000

[1520 rows x 2 columns]

Less watched movies:

movie_title
Dream Man 1995                          1
Naked in New York 1994                  1
Ill Gotten Gains 1997                   1
Two Bits 1995                           1
I Can't Sleep J'ai pas sommeil 1994     1
```

```
Country Life 1994                          1
Turning, The 1992                          1
Hunted, The 1995                           1
Sexual Life of the Belgians, The 1994      1
For the Moment 1994                        1
Total Eclipse 1995                         1
All Over Me 1997                           1
Beyond Bedlam 1993                         1
Time Tracers 1995                          1
Simple Wish, A 1997                        1
BAmerican Dream 1990                       1
American Strays 1996                       1
Big Squeeze, The 1996                      1
Golden Earrings 1947                       1
dtype: int64
```

Best rated movies:

movie_title	rating size	rating mean
Star Kid 1997	3	5.000000
Visitors, The Visiteurs, Les 1993	1	5.000000
Saint of Fort Washington, The 1993	2	5.000000
Santa with Muscles 1996	1	5.000000
Prefontaine 1997	2	5.000000
Perfect Candidate, A 1996	2	5.000000
Pather Panchali 1955	5	4.800000
Two or Three Things I Know About Her 1966	3	4.666667
Close Shave, A 1995	72	4.541667
Shall We Dance? 1996	28	4.535714
...	...	...
Mighty, The 1998	2	1.000000
Gordy 1995	3	1.000000
Amityville 1992: It's About Time 1992	4	1.000000
Sexual Life of the Belgians, The 1994	1	1.000000
Amityville: Dollhouse 1996	3	1.000000
Amityville: A New Generation 1993	4	1.000000
Sliding Doors 1998	2	1.000000
Theodore Rex 1995	3	1.000000
For the Moment 1994	1	1.000000
American Strays 1996	1	1.000000

[1520 rows x 2 columns]

movie_title	rating size	rating mean
Close Shave, A 1995	72	4.541667
Shawshank Redemption, The 1994	183	4.535519
Casablanca 1942	162	4.524691
Schindler's List 1993	200	4.435000
Wrong Trousers, The 1993	88	4.431818
Rear Window 1954	148	4.418919
Star Wars 1977	414	4.355072
One Flew Over the Cuckoo's Nest 1975	180	4.338889
Usual Suspects, The 1995	175	4.337143
Silence of the Lambs, The 1991	286	4.321678
To Kill a Mockingbird 1962	159	4.320755

```
12 Angry Men 1957                                      89  4.314607
North by Northwest 1959                               130  4.300000
Titanic 1997                                          232  4.288793
Vertigo 1958                                          120  4.266667
Dr. Strangelove or: How I Learned to Stop Worry...    138  4.260870
Good Will Hunting 1997                                135  4.259259
Secrets & Lies 1996                                   112  4.258929
Empire Strikes Back, The 1980                         262  4.251908
Raiders of the Lost Ark 1981                          287  4.247387
------------
Gender analysis:
------------

Movie delta for gender:

gender                                        F         M      diff
movie_id movie_title
1        Babyfever 1994              0.000000  1.000000  1.000000
2        Money Train 1995            2.333333  2.571429  0.238095
3        Good Man in Africa, A 1994  0.000000  2.000000  2.000000
4        Vertigo 1958                4.035714  4.336957  0.301242
5        Paris Is Burning 1990       3.833333  3.769231 -0.064103
Plot movie delta for gender:

                unknown  Action  Adventure  Animation  Children  Comedy  \
age_categories
0-9                   0      13         12          1         4      14
10-19                 0    1637        849        274       580    1777
20-29                 5    7629       3978       1094      1884    8450
30-39                 0    4540       2565        667      1334    5391
40-49                 0    2427       1376        337       768    2966
50-59                 0    1265        649        116       285    1541
60-69                 0     322        171         38       106     480
70-79                 0      24          8          4        11      64

                Crime  Documentary  Drama  Fantasy  Film-Noir  Horror  \
age_categories
0-9                 3            0      7        0          0       0
10-19             463           40   1929       99         55     382
20-29            2290          193  10273      376        393    1632
30-39            1369          134   7134      221        342     984
40-49             777           83   4436      149        209     447
50-59             508           62   2893       67        149     238
60-69             152           18    891       22         46      56
70-79              10            0     64        1          3       3

                Musical  Mystery  Romance  Sci-Fi  Thriller   War
age_categories
0-9                   2        1        8       7         6     4
10-19               314      252     1000     725      1388   400
20-29              1303     1329     5221    3840      6321  2352
30-39               925      882     3565    2369      3789  1790
40-49               521      628     2129    1204      2139  1083
50-59               262      408     1233     594      1226   649
60-69               104      103      347     145       341   179
70-79                 7       15       38      12        31    13
```