Pattern Classification and Machine Learning, Project 2: Collaborative Filtering

Team "Going up": Audrey Loeffel, Maxime Coriou, and Romain Choukroun

***Abstract*—In this project we were given a set of data representing the ratings of movies by users. The aim was to implement a recommender system predicting future ratings made by users on any new movie in order to recommend them a set of movies to watch. We implemented different machine learning methods in order to solve this problem, following a few user-based recommendation techniques. Given a user-movie rating sparse matrix, we first tried out the K-Nearest Neighbors method, followed by many Collaborative Filtering method including Singular Value Decomposition and Matrix Factorization with Alternating Least Squares and Stochastic Gradient Descent.**

I. Data Format and Feature Engineering

The dataset consists of ratings by 10000 users on 1000 movies. Amongst the 10'000'000 possible entries, there are 1'176'192 ratings, so about 12% of the data is useful. Which means that we will be working with quite a large sparse matrix, fortunately, Scipy is perfectly designed to work with this kind of data. Feature engineering being less useful in the case of collaborative filtering in general and even more so when the metric for this project was precision (to the 10^{-5} decimal), we did not alter our data too much nor added any kind of new features. The only problem we can have with such a dataset would be its low-density with regard to the users that rate a handful of movies, or the movies that have been rated only a few times. Hence we only kept the data for each item or user for which we had 10 ratings or more.

II. Models

A. K-Nearest Neighbors

We started by working on a github library called Surprise [1] implemented in Python. One of the best methods provided by Surprise was the K-Nearest Neighbors (KNN) method. After playing a bit with the parameters of the library, we found that using $k = 4$ neighbors gave the best results. We used a similarity measure to estimate every rating, here the Mean Squared Difference similarity between all pairs of users (or items). After running multiple instances of the KNN algorithm, we quickly found that the Root-Mean-Square-Error we obtained was always slightly higher than 1. For this reason, we stopped pursuing the KNN route since we already had better results with the implementation of Alternating Least Squares of the 10^{th} Lab.

B. Collaborative Filtering with Singular Value Decomposition

Another method that gave us pretty good results was Collaborative Filtering using the Singular Value Decomposition (SVD) factorization. We trained our model on the same dataset as before, aiming to find the best set of parameters with which to run the SVD factorization. On the contrary to the following methods, taking a number of factors smaller than 10 surprisingly did not give us good results, we obtained better results around $k = 20$, k being the number of factors. By iterating over $n = 30$ epochs, with a regularizer of around $\lambda = 0.04$ and learning rate of $\gamma = 0.05$, we achieved a score slightly below 1. Same as previously, we had no point to investigate further the Singular Value Decomposition.

C. Collaborative Filtering with Non-Negative Matrix Factorization

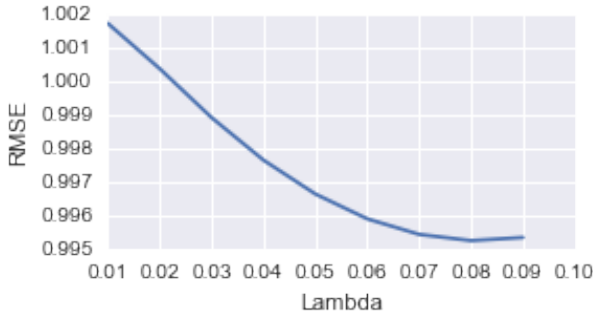
Our next idea was to use collaborative filtering with Non-Negative Matrix Factorization. This is quite straight forward, we need to find W and Z such that our data matrix $X \approx WZ^T$, where W is a $D \times K$ matrix, Z is a $N \times K$ and so, X is a $D \times N$ matrix (here D is the number of movies, N the number of users and K the number of features.
To be able to test our model, we start by splitting X into a training and a testing set. The next step is to optimize W and Z by minimizing a certain cost function using only the training set and then compute the Root-Mean Squared Error (RMSE, see equation 1) on the testing set using the newly found W and Z.

$$RMSE(W,Z) = \frac{1}{|\Omega|}\sum_{(d,n)\in|\Omega|}(x_{dn} - (WZ^T)_{dn})^2 + \frac{\lambda_w}{2}||W||^2 + \frac{\lambda_z}{2}||Z||^2 \tag{1}$$

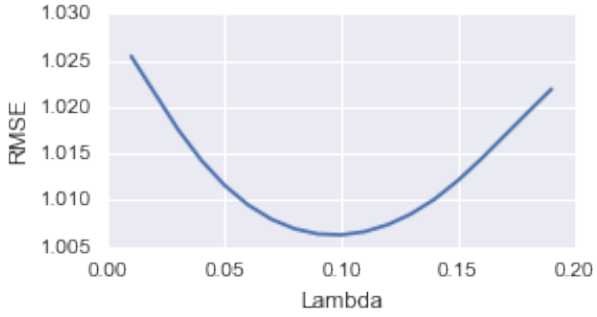
where Ω is the set of non zero entries in X. Here we use 10% of our data as testing data, hence 90% as our training data.
Let's now focus on the different methods we used in order to optimize the W and Z matrices.

1) Alternative Least Squares with Spark MLlib: We found the amazing Spark MLlib[2] library that we used to train a model with ALS. After testing our model with different values for the regularizing parameter and the number of features, we achieved a RMSE of 0.99524 on the testing set with $k = 3$ as the number of features and $lambda = 0.08$.
Their ALS model didn't allow to set a different regularizer value to the user and the item; the same λ is applied in both regressions. Therefore it limits the possible range of optimization that we could reach to minimize the RMSE.

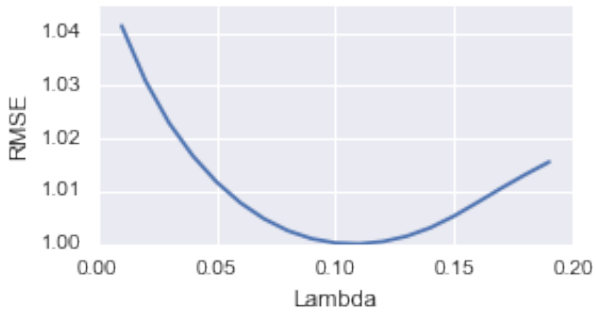
The figures 1 below show the RMSE in function of lambda for $k \in \{3, 5, 7, 8\}$. As we can see, the smaller RMSE is obtained with $k = 3$ and $lambda = 0.08$.



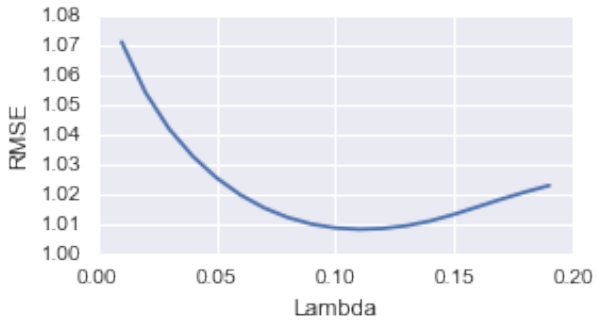
(a) 3 Features



(b) 5 Features



(c) 7 Features



(d) 9 Features

Fig. 1. RMSEs in function of the number of features and the regularizer λ for ALS implemented with MLlib

After submitting our predictions in the Kaggle Competition, we only achieved a score slightly greater than 1. Since this result was far from our previous results and since it would have been difficult to optimize this limited model, we decided to stop investigating further and try another method.

2) Alternative Least Squares: Moving on to our own implementation of the Alternating Least Squares by following a few good online articles ([3]), we managed to implement the ALS agorithm with missing entries.
Two things immediately shocked us. The first one being that it was converging way faster than our best method that we will discuss more in details in the next section. The second one was how "accurate" it was compared to the previously tried out methods. First, let's take a look at the number of features we used in the following graph 2:

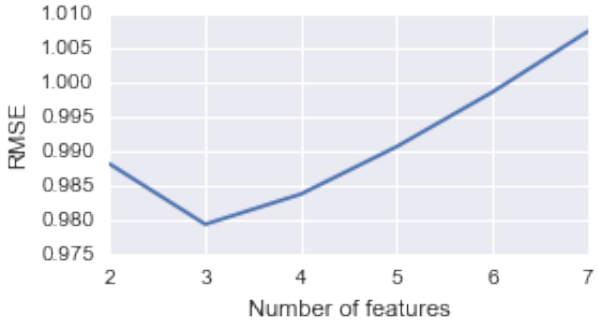


Fig. 2. Minimum testing RMSE in function of the number of features k.

Same as the previous ALS method, we observe that a really low number of features gives us the best score, which is in accordance with the method itself, since we aim to update the items matrix W and the user matrix Z always one after the other. So in between runs, each matrix update will be used in order to compute the update of the other. The beauty of this is that we sacrifice precision for a huge gain in computational cost. And the best Root-Mean Squared error that we obtained on our testing set was around 0.9798, while our best Kaggle score was 0.98916, which was a decent score. However the aim of this project was primarily to compete against one another in order to get the best score. In that regard, this result was pretty good, but still not enough. In fact we had to sacrifice computational cost in favor of precision to rank higher. Which brings us to the final algorithm, Stochastic Gradient Descent.

3) Stochastic Gradient Descent: The other possibility is to minimize this cost function using SGD.
We first need to derive the gradient with respect to W (g_w) and the gradient with respect to Z (g_z). Then, we compute the new $W = W - \gamma g_w$ and $Z = Z - \gamma g_z$ (γ being the learning rate). We repeat this step up until convergence of the training Root-Mean Squared Error. Once it has converged, we compute the testing error with the new matrices W an Z.
We can now test our model with different set of parameters k, λ_u and λ_i. If we take a look at Fig. 3, we can see a curve representing the minimum testing RMSE for each k.

Obviously, our best case is with $k = 9$ features. So, let's focus on this case by looking at Fig.4, the reverse normalized

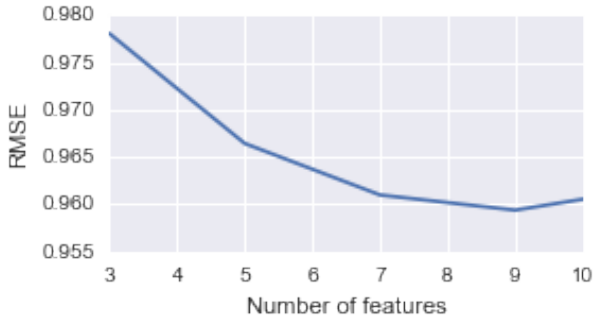


Fig. 3. Minimum testing RMSE in function of the number of features k.

testing RMSE for $k = 9$ and the different values of the two lambdas.

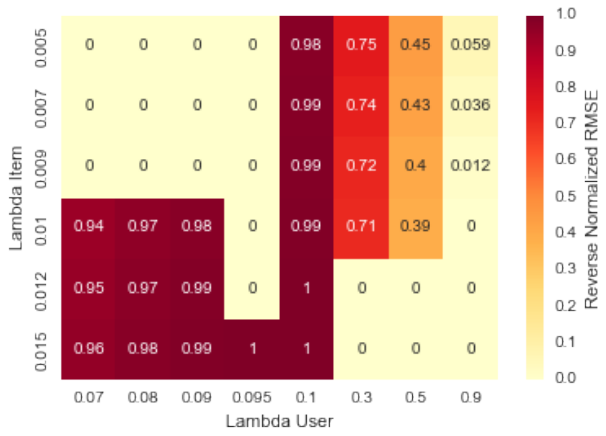


Fig. 4. Reverse Normalized RMSE in function of λ_i and λ_u for $k = 9$

As we reverse the RMSE, our best results are now the one with the closest score to 1. We see on Fig.4 that the concerned values are the ones with their regularizers around $\lambda_u = 0.1$ and $\lambda_i = 0.015$. Actually, after running a few more iteration around them, we get a more precise grasp on the best parameters, which are $\lambda_u = 0.1$ and $\lambda_i = 0.016$, achieving a RMSE of 0.9593 on the testing set.
Now that we have the best parameters for our model ($k = 9$, $\lambda_u = 0.1$ and $\lambda_i = 0.016$) we can simply train it on the full dataset and do the predictions. This technique resulted as being the best for us.

III. Conclusion

We submitted the prediction of our best Stochastic Gradient Descent model and got a Root-Mean Squared error of 0.97803 on Kaggle. This result is the best one we could achieve, but finding the model was really costly in terms of computation. As our goal in this project was to achieve the best possible score we used SGD but practically, to be computationally efficient, we should use ALS. Indeed the score with this method is not far from the one we got for SGD (0.98916) and ALS is way faster to train than SGD, about 3 minutes for a single set of parameters for ALS against more than 20 minutes for SGD. Also, as we spent a lot of time trying to optimize SGD, we couldn't spend more time looking at external libraries and maybe we could have found some better methods.
We also stumbled against a few datasets that contained more information than our own, the movielens dataset is a great example of that. In fact in addition to the ratings, it also contained the time-stamp for each rating. Which is quite a big deal, since we would've been able to focus on methods related to clustering way more in this case.
That made us realize that this kind of optimization problems are central to a lot of industries nowadays. The company Netflix (that came up with a challenge extremely close to this project a few years back) is an obvious candidate to this description, their products and productions are solely based on analysis of the "users" or "consumers" behaviors and likings. Finally, as a humble suggestion for next year's project, it would've been great to have access to a few powerful computers (from the lab or maybe elsewhere) in order to have more time to focus on feature engineering or maybe different models and/or external libraries. Regardless it was awesome working on that projects and we had loads of fun. We wish you a merry christmas and a happy new year !

References

[1] http://surpriselib.com/
[2] http://spark.apache.org/docs/latest/ml-guide.html
[3] https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea