Machine Learning Project
Recommender System

Armand BOSCHIN Guirec MALOISEL Adrian VALENTE

***Abstract*—In this paper, we propose a model a recommender system. The aim is to predict the rating an item is going to get from a user. The structure of the paper is the following : after a short introduction in section I, we present the data set in section II, the algorithms used in section III and eventually we discuss the results in the last section. Particularly, we show that blending the predictions of various models including matrix factorization and collaborative filtering can lead to good performance.**

I. INTRODUCTION

The goal of this project is to build a model to predict the ratings of items by users. This is a common problem especially for web companies providing content on demand such as Netflix, Deezer or Spotify. In 2008, was even launched the Netflix Prize, a competition hosted by Netflix with a $1'000'000 grand prize.

Here, in order to compare the performance of the various models presented, we use the root mean squared error (RMSE) :

$$RMSE = \sqrt{\frac{1}{|\Omega|} \sum_{(i,u)\in\Omega} (\hat{r}_{iu} - r_{iu})^2} \tag{1}$$

where Ω is the set of couple (i, u) of items and users.

II. DATA SET

The data comes as a sparse matrix R whose non-zero entries r_{iu} are integers in $[1..5]$ corresponding to the rating of a user u for an item i. R has a size 10000×1000, its rows represent the items, and its columns represent the users. R has only 1,176,952 non-zero entries, which is approximately 12% of the total. Our objective is to design a model capable of predicting accurately the missing ratings (accurately meaning with the lowest RMSE possible).

In the following we note $\mathcal{U}$ the set of users (of size $m = 1000$) and $\mathcal{I}$ the set of items (of size $n = 10000$).

III. DESCRIPTION OF MODELS USED

A. Baselines

In order to assess the efficiency of our models, we start by defining some baselines:

- Global mean: predict for each $i \in \mathcal{I}$, $u \in \mathcal{U}$, $\hat{r}_{iu} = \mu$ where μ is the mean of all non-zero ratings:

$$\mu = \frac{1}{N} \sum_{i\in\mathcal{I}} \sum_{u\in\mathcal{U}} r_{iu}$$

where N is the number of non-zero ratings.

- User mean: predict for $i \in \mathcal{I}$, $u \in \mathcal{U}$, $\hat{r}_{iu} = \mu_u$ where μ_u is the mean of non-zero ratings for user u:

$$\mu_u = \frac{1}{N_u} \sum_{i\in\mathcal{I}} r_{iu}$$

where N_u is the number of non-zero ratings for user u.

- Item mean: predict for $i \in \mathcal{I}$, $u \in \mathcal{U}$, $\hat{r}_{iu} = \mu_i$ where μ_i is the mean of non-zero ratings for item i:

$$\mu_i = \frac{1}{N_i} \sum_{u\in\mathcal{U}} r_{iu}$$

where N_i is the number of non-zero ratings for item i.

B. Matrix factorization model

The first model that we use approximates the matrix of ratings by a product of two thin matrices. It is a model that has received attention during the Netflix Prize as one of the best predictors. [5]

Formally, we fix a hyper-parameter k called the number of latent features, and we compute two matrices P of size $k \times n$ and Q of size $k \times m$ such that our predictions are

$$\hat{R} = P^T Q$$

and such that the RMSE is minimized over train instances, which is equivalent to minimizing the sum of squares:

$$\begin{aligned}\mathcal{L}(\mathcal{P}, \mathcal{Q}) &= \sum_{(i,j)\in Train} (\hat{r}_{iu} - r_{iu})^2 \\ &= \sum_{(i,j)\in Train} (\mathbf{p}_i^T \mathbf{q}_u - r_{iu})^2\end{aligned}$$

The rationale behind this model is that we can represent the tastes of each user by k features, and similarly represent characteristics of each item by k features. Thus, computing a scalar product $\mathbf{p}_i^T \mathbf{q}_u$ gives us the affinity between the user u and the item i, thus an estimation of the rating.

In order to avoid overfitting, we also added a regularization term to our loss function:

$$\begin{aligned}\mathcal{L}(\mathcal{P}, \mathcal{Q}) = \sum_{(i,j)\in Train} (\mathbf{p}_i^T \mathbf{q}_u - r_{iu})^2 + \lambda_{\mathcal{I}} \|P\|_{Frob}^2 \\ + \lambda_{\mathcal{U}} \|Q\|_{Frob}^2\end{aligned}$$

There are two main algorithm for matrix factorization. Here are the details of the two.

1) Stochastic Gradient Descent (SGD): The goal is to minimize $L(P, Q)$ by conduction a stochastic gradient descent over all non-zero ratings. That is, at each iteration a random (i, u) among all the non-zero rations is selected and the following update is executed :

$$\mathbf{p}_i \leftarrow \mathbf{p}_i - \gamma((\hat{r}_{iu} - r_{iu})\mathbf{q}_u + \lambda_{\mathcal{I}}\mathbf{p}_i)$$

$$\mathbf{q}_u \leftarrow \mathbf{q}_u - \gamma((\hat{r}_{iu} - r_{iu})\mathbf{p}_i + \lambda_{\mathcal{U}}\mathbf{q}_u)$$

where γ is the step of the descent, and $\lambda_{\mathcal{I}}$ and $\lambda_{\mathcal{U}}$ are the penalization coefficients (those should be chosen by cross-validation).

2) Alternating Least Squares (ALS): This method successively solves analytically the least-squares problem for P and Q. More precisely, at each iteration the ALS algorithm computes the updates:

$$\mathbf{p}_i \leftarrow (\tilde{Q}\tilde{Q}^T + \lambda_{\mathcal{I}} I_k)^{-1}\tilde{Q}\tilde{\mathbf{r}}_i$$

(where $\tilde{Q}$ represents the matrix Q stripped of the columns u for which $r_{iu} = 0$, and $\tilde{r}_i$ is the vector of non-zero ratings r_{iu}) and

$$\mathbf{q}_u \leftarrow (\tilde{P}\tilde{P}^T + \lambda_{\mathcal{U}} I_k)^{-1}\tilde{P}\tilde{\mathbf{r}}_u$$

(where $\tilde{P}$ and $\tilde{r}_u$ are similarly defined).
Once again $\lambda_{\mathcal{I}}$ and $\lambda_{\mathcal{U}}$ are the penalization coefficients and should be chosen by cross-validation.

C. Collaborative filtering

Collaborative filtering is based on the observation that if someone enjoys a film, people who enjoyed the same type of films will probably enjoy this one too. Thus, collaborative filtering consists in comparing different users, by somehow measuring their similarity of tastes, and then predict the rating of a given movie i by a user u as a function of the ratings of the most similar users u' to u who also rated i. This process is the user-based collaborative filtering, but an item-based version can also be derived in a symmetric fashion.
To be able to compare the ratings of different users, we use normalized ratings on a per-user basis. More precisely, we compute as a pre-processing step:

$$\bar{r}_{iu} = \frac{r_{iu} - \mu_u}{\sigma_u}$$

where

$$\mu_u = \frac{1}{|\mathcal{I}_u|}\sum_{j \in \mathcal{I}_u} r_{ju} \quad \text{and} \quad \sigma_u = \sqrt{\frac{1}{|\mathcal{I}_u|}\sum_{j \in \mathcal{I}_u}(r_{ju} - \mu_u)^2}$$

are respectively the mean and the standard deviation of the ratings given by u, $\mathcal{I}_u$ being the set of the items rated by u.
We then define the similarity measure between two users u and u' as:

$$sim(u, u') = \sum_{i \in \mathcal{I}_{uu'}} \bar{r}_{iu}\bar{r}_{iu'}$$

where $\mathcal{I}_{uu'}$ designates the set of items that have been rated both by u and u'.
Thanks to the normalization process, this is a slightly tweaked correlation measure : the positive contributions to the similarity come from the films u and u' both rated above (or both rated below) their own average rating. This measure also has the propriety, as the sum is not normalized, that users who give similar ratings on a larger number of films will be deemed as more similar (their similarity can be more trusted).
Computing this similarity provides us with a graph structure on the set of users. We connect any two users which have at least one common rated film, and whose similarity is positive. We weight the edge between them by their similarity. To predict a rating $\hat{r}_{iu}$, we use a variation of the k-nearest neighbors algorithm. More precisely, we look at the set $\mathcal{N}^{(i)}(u)$ of the neighbors of u in the graph, having also rated the film i. Among these neighbors, we retrieve the k most similar to u, forming the set $\mathcal{N}_k^{(i)}(u) \subset \mathcal{N}^{(i)}(u)$. We then predict the *normalized* rating of i by u as the weighted average of their normalized scores, the weights being similarities with u :

$$\hat{\bar{r}}_{iu} = \frac{\sum_{u' \in \mathcal{N}_k^{(i)}(u)} sim(u, u') \cdot \bar{r}_{iu'}}{\sum_{u' \in \mathcal{N}_k^{(i)}(u)} sim(u, u')}$$

If not enough neighbors (less than k) have seen the film i, the set $\mathcal{N}_k^{(i)}(u)$ will simply be formed of less than k elements, which doesn't change the averaging process, unless in the unlikely event that no neighbor of u has seen i, in which case we set $\hat{\bar{r}}_{iu} = 0$.
To get the final prediction $\hat{r}_{iu}$, we have to reverse the normalization process and go back to the user u's personal grading scale. This is simply done by setting

$$\hat{r}_{iu} = \mu_u + \sigma_u \hat{\bar{r}}_{iu}$$

(thus adding the notion of a *user bias* to our prediction). Note that when we have no information from the neighbors, ie when $\mathcal{N}_k^{(i)}(u) = \varnothing$, we get $\hat{r}_{iu} = \mu_u$, which is reasonable.
As aforementioned, we can apply the whole process to build a dual model, which is item-based. For this, we keep the normalization process centered on users, as it is our intuition that users have their personal grading scale (summed up by a mean and standard deviation), but that this doesn't really apply to items. The other operations (computation of the similarities, prediction using nearest neighbors) are done by switching users and films, which leads to a different prediction model.

D. Blending

Being inspired by the nice story telling that the best scores for the Netflix Prize were achieved by teams who *merged* there predictions, we decided to try the idea called blending [4]. In our case, we wish to combine the

predictions of 7 models: the global mean, the user mean, the item mean, SGD and ALS matrix factorization, user-based and item-based collaborative filtering. To do this, we train a simple linear model to map the ouputs of these 7 models to a single prediction. This looks like a weighted average of the predictions of each model. More formally, we build a matrix X with 7 columns and one line for each (i, u) pair such that a rating exists in the training data, and a column vector $\mathbf{y}$ containing the corresponding ratings. The goal is to find the least squares problem :

$$\begin{pmatrix} \hat{r}_{1,1} & \dots & \hat{r}_{1,7} \\ \vdots & & \vdots \\ \hat{r}_{N,1} & \dots & \hat{r}_{N,7} \end{pmatrix} \cdot \mathbf{w} = \mathbf{y}$$

where $\hat{r}_{i,j}$ is the estimation of the i-th rating by the j-th model. This can be solved using penalization on w and then solving a Ridge regression problem.

IV. Experiments and results

A. Baselines

Though the baseline algorithm are quite simple, they perform not that bad.

Model	RMSE on Kaggle	Training time
Global Mean	1.11785	$\sim$ 15s
User Mean	1.02982	$\sim$ 2 min
Item Mean	1.09267	$\sim$ 30s

B. SGD

In order to tune the penalization hyper parameters, we used a k-fold cross validation with $k = 3$. The obtained curves are in figure 1. The best values found are $\lambda_{\mathcal{I}} = 0.25$ and $\lambda_{\mathcal{U}} = 0.011$. The tuning of k gave $k = 25$. The following array gives the performance of this model trained during 60 epochs with the parameters given above. We choose 60 epochs empirically.

Model	RMSE on Kaggle	Training time
SGD	1.00014	$\sim$ 30 min

C. ALS

Just like for the SGD model, the penalization coefficients were tuned using k-fold cross validation with $k = 3$. The resulting curves are in figure 2. The values giving the best results are $\lambda_{\mathcal{I}} = 0.575$ and $\lambda_{\mathcal{U}} = 0.014$. The tuning of k gave $k = 20$. The model was trained until the improvement between two iterations was less than 10^{-6}.

Model	RMSE on Kaggle	Training time
ALS	0.98194	$\sim$ 15 min

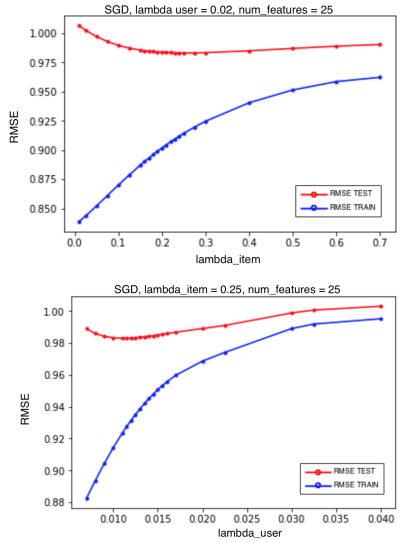

Fig. 1. Cross-validation plots of the SGD penalization coefficients.

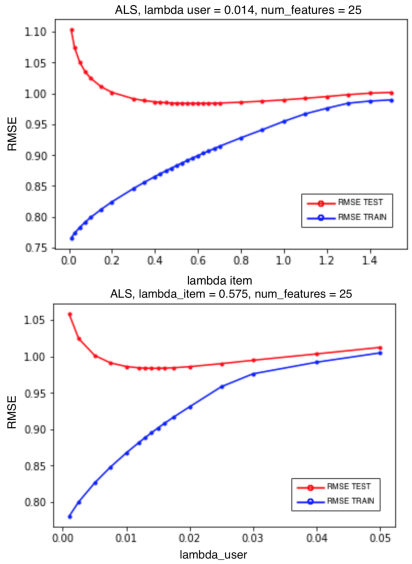

Fig. 2. Cross-validation plots of the ALS penalization coefficients.

D. Collaborative Filtering

The training of the Collaborative filtering model is quite long as we have to build a graph encoding similarity of each item/user with its peers. The computation time given below is indicative and highly depends on the machine it is trained on.

Let's note that because the two models (item and user based) are really long to train, we could not do as much tuning as we would have liked. That is why the RMSE is not as satisfying as what we had hopped for.

Model	RMSE on Kaggle	Training time
User based CF	0.99373	$\sim$ 2 h
Item based CF	0.99957	$\sim$ 7-8 h

E. Blending

Blending all the models together did not produce the best results. We tried all the picks of models possible and only two combinations actually improve the prediction. The first one is the blending of the ALS alone, that correspond actually to a dilatation of the result (multiplication by a unique factor). The second one is the blending of ALS matrix factorization, user-based and item-based collaborative filtering.
The times given in the following array correspond to a computation where each model has already been trained and has already done its predictions.

Model	RMSE on Kaggle	Training time
Blending	0.97747	$\sim$ 30 s
Dilatation of ALS	0.97700	$\sim$ 30 s

We believe that this odd result regarding blending is only happening because we did not manage to accurately tune the collaborative filtering model.
Eventually the model that performed the best is an ALS matrix factorization with the following parameters, dilated by a blending coefficient :

Parameter	Value
num_features	20
$\lambda_{\mathcal{I}}$	0.575
$\lambda_{\mathcal{U}}$	0.014
stop_criteria	1e-5
dilatation coefficient	1.02774082

And the performances : (the training time is from scratch, in the provided code, some the pre trained ALS model is provided).

Model	RMSE on Kaggle	Training time
Final Model	0.97700	$\sim$ 25 min

V. Conclusion

In this paper we showed that blending different models together can significantly reduce the prediction error. We noted that the performance of the SGD and the ALS algorithm are quite different even if they try to solve the same optimization problem. We also noted that tuning a model is really important and can have a huge computational cost. We see the interest of parallelization. An improvement to our work could be a finer tuning of the models because as we have seen this seems to be a limiting factor here.

References

[1] Edwin Chen. Winning the netflix prize: A summary. http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/, 10 2011.

[2] Simon Funk. Netflix update: Try this at home. http://sifter.org/simon/journal/20061211.html, 2006. Accessed: 2006-12-11.

[3] Martin Jaggi and Mohammad Emtiyaz Khan. Matrix factorizations course notes. 2016.

[4] Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. pages 693–702, 2010.

[5] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. 42(8):30–37, 2009.