Machine Learning
Project 2 : Recommender system

ML_budget_3000 : Louise Aubet, Alexandre Cadillon & Emma Hoggett
EPFL, Switzerland

***Abstract*—This project's goal is to build a movie recommendation system by using collaborative filtering machine learning techniques adapted to the provided dataset, such as matrix factorization and neural networks.**

The most accurate model was a combination of all models and obtained a RMSE of 1.017 on AICrowd.

I. Introduction

Recommending an item to a specific user is a major concern for online streaming platforms such as Netflix or Amazon Prime. The machine learning tools developed to solve this problem are called recommender systems. Nowadays, almost every major company use personalised recommendations to their users in order to enhance the quality of their services. A recommendation system is a system capable of learning the watching patterns and provide relevant suggestions. In this particular case, it uses collaborative filtering. This approach builds a model from user's past behaviour (i.e. items purchased or searched by the user) as well as similar decisions made by other users. This model is then used to predict ratings for items that user may have an interest in.

These tools can be generalised for other tasks such as recommending news articles, books, research articles, search queries, social tags etc.

II. Preliminary data analysis

The data set is composed of ratings from $N = 10000$ users for $D = 1000$ different movies. All ratings take integer values between 1 and 5. No additional information is known on the movies or users, other than their id number. The data set can be seen as sparse entries from a ratings matrix of size $N \times D$ and the goal of this project is to predict the whole matrix.

The rating spectrum $\{1, 2, 3, 4, 5\}$ is not symmetric, since the distributions of average rating per user and movie are note centred between 1 and 5, as seen in figure 1. This can affect the reliability of the predictions on the lower part of the spectrum and also create an unwanted bias favouring movies with good ratings.

All this leads to what is called *popularity bias*: popular items end up being recommended more often than niche products. This may pose a problem as it reduces the diversity of the recommendation engine, which is often a valued feature. The best recommender system will have to take into account this issue.

The data set is split into a train and a test set with sizes 80% and 20%. All results shown below are trained on the train set and evaluated on the test set, but the final submission on AICrowd is trained on 100% of the original data set.

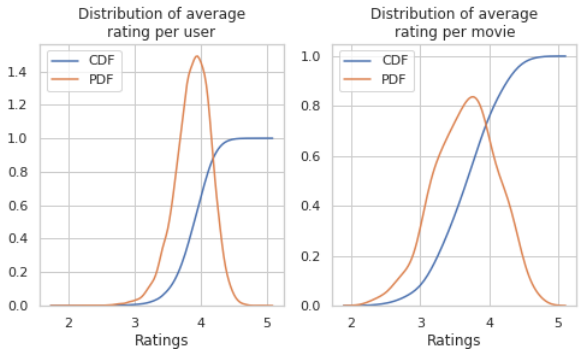

Fig. 1. Distribution of the average rating by user and by movie.

III. Baseline implementation

Three baseline models that use global, user and movie mean to predict ratings are implemented. As seen in table I, the most accurate model is the one using the item mean. This basic implementation already indicates how significant item bias is.

TABLE I
Baseline results : RMSE loss function

Method	Global mean	User mean	Item mean
RMSE[1]	1.1196	1.0967	1.0301

IV. Matrix factorisation

A. First approach

The idea behind matrix factorisation is to represent users and items in a lower dimensional latent space. This dimension reduction can be interpreted as if every user and every movie was being represented as a finite set of features. The algorithm's aim is to compute a full matrix of predicted ratings $\hat{R}$ of size $N \times D$. $\hat{R}$ is computed as the dot product between P and Q with sizes $K \times N$ and $K \times D$ respectively. K is a hyper-parameter representing the number of latent features. A large value of K may cause over-fitting. The prediction for user u and item i can be seen as $\hat{r}_{ui} = q_i^T p_u$, where q_i and p_u are two vectors of size K containing the latent features for u and i respectively.

[1]RMSE: root mean squared error

Only some sparse entries of the real ratings matrix R are available, so the error $e_{ui} = r_{ui} - \hat{r}_{ui}$ can only be computed for these known entries. The matrix factorisation algorithms use the squared errors of these known entries as a loss function and add L_2 regularisation terms with λ_q and λ_p constants.

$$\mathcal{L} = \sum_{(u,i) \text{ s.t. } r_{ui} \neq 0} (r_{ui} - \hat{r}_{ui})^2 + \lambda_q \|q_i\|^2 + \lambda_p \|p_u\|^2 \quad (1)$$

The optimisation of P and Q is done using either a classic SGD[2] method or ALS[3], which optimises P by keeping Q fixed and vice versa, successively.

Results are represented in table II, where ALS achieves the best RMSE. The hyper-parameters are different for each implementation and are optimised with a grid search.

TABLE II
MATRIX FACTORISATION RESULTS USING DIFFERENT OPTIMISER ($\gamma = 0.02$, # EPOCHS $= 20$).

Optimiser	SGD	ALS
Parameters	$K = 25$ $\lambda_p = \lambda_q = 0.07$	$K = 8$ $\lambda_p = \lambda_q = 0.081$
RMSE	1.0042	0.9918

B. Other Singular Value Decomposition methods

The MF[4] algorithm presented above is the simplest implementation of SVD[5]. During the Netflix Prize other optimised models based on SVD were implemented.

The method know as Funk MF adds overall (μ), user (b_u) and movie (b_i) bias terms to the dot product when computing the predicted rating: $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$. The regularisation term in the loss function also includes the L_2 norm of b_u and b_i. This model is less affected by the popularity bias mentioned above.

The method know as SVD++ is an extension of Funk MF that captures users' implicit ratings. The predicted rating is defined as $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$, where I_u is a set that contains all items for which user u provided an implicit preference and the y_j terms capture the implicit preference of item j by user u.

These methods were all implemented using SGD for optimisation. Results are represented in table III.
Note: The python library *Surprise* [5] is used.

TABLE III
COMPARISON BETWEEN SIMPLE MF, FUNK MF AND SVD++ ($K = 6$, CONSTANT REG. FACTOR $\lambda = 0.025$, LEARNING RATE $\gamma = 0.005$)

Method	Matrix factorisation	SVD	SVD++
RMSE	1.0123	1.0027	1.0001

[2]SGD: Stochastic Gradient Descent
[3]ALS: Alternating Least Squares
[4]MF: Matrix Factorisation
[5]SVD: Singular Value Decomposition

C. The Adam optimiser

Matrix factorisation was implemented using the neural network framework *Keras* [3]. The K latent features are contained in embedding layers which are then combined using dot product. RMSE is used for the loss function.

Three different optimiser are tested: Adam, Rectified Adam and AdaMax. As opposed to classic SGD which uses a constant learning rate, the Adam method computes individual adaptive learning rates to update the network weights. Rectified Adam introduces a term to rectify the variance of the adaptive learning rate in Adam. In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L_2 norm of their individual current and past gradients. The L_2 norm based update rule can be generalised to a L_p norm based update rule. AdaMax uses the L_∞ norm ($p \to \infty$) and is surprisingly stable. The results are summarised in table IV.

TABLE IV
DOT PRODUCT RESULTS WITH RMSE LOSS FUNCTION, USING DIFFERENT OPTIMISERS ($K = 25$, BATCH _ SIZE$= 1000$, #EPOCHS $= 20$)

Method	Adam	RAdam	AdaMax
RMSE	1.0187	1.0155	1.0059

D. Other improvements on the dot product implementation with Keras

A normalisation process can be implemented in three steps. The idea is to create an adjusted rating by subtracting off the overall mean rating $\bar{r}$, the mean rating for each item $\bar{r}_i$, and then the mean rating for each user $\bar{r}_i : \tilde{r}_{i,j} = \hat{r}_{i,j} - \bar{r} - \bar{r}_i - \bar{r}_u$. $\tilde{r} = 0$ means that user u's rating for item i is exactly what we would guess if the only known data were $\bar{r}$, $\bar{r}_i$ and $\bar{r}_u$. Any values above or below 0 indicate deviations in preference from this baseline. As a consequence, $\tilde{r}$ can be seen as the preference whereas the raw rating $\hat{r}$ is an actual rating.

Another improvement consists in adding a regularisation term. It is a hyper-parameter and can be optimised. The value used in the simulations is written in table V.

Also, another loss function can be used, the CCE[6] method. It is used in categorical classification problems where the output is seen as several distinct alternatives: in this case, the output is the probability that a particular rating is one of the categories in 1, 2, 3, 4, 5. Here, there are $C = 5$ different classes. The loss function is written :

$$CCE = -\sum_{i=1}^{5} y_{(u,i),c} \log(p_{(u,i),c}) \quad (2)$$

where y is the binary variable (0 or 1) that indicates if c is the correct class for element (u, i) and p is the predicted probability that (u, i) is in class c. Results are summarised in Table V. One can observe that normalisation doesn't improve the score as one could have expected. Also, the CCE loss

[6]CCE: Categorical Cross Entropy

function applied on that matrix factorisation problem gives a much less accurate prediction.

TABLE V
DOT PRODUCT RESULTS WITH DIFFERENT LOSS FUNCTIONS
($K = 25$, BATCH _ SIZE= 1000, #EPOCHS = 20)

Method	RMSE	RMSE & normalisation	CCE
Parameters	opt = AdaMax $\lambda = 1 \cdot 10^{-6}$	opt = AdaMax $\lambda = 1 \cdot 10^{-6}$	opt = SGD $\lambda = 1 \cdot 10^{-5}$
RMSE	1.0007	1.1488	1.3947

V. NEURAL NETWORK

Neural networks can also be implemented for this task extending some of the main concepts seen in matrix factorisation methods.

The first layer of the neural network will be an embedding layer that, just as in matrix factorisation, will contain latent features. The model no longer has the constraint of having the same number of latent features for both users and movies.

If the output (i.e. the predicted rating) is considered a categorical value, the last layer of the neural network will necessarily have 5 neurons with a softmax activation function.

There are two ways to go, either a shallow network with a high number neurons in each layer or a deep network with more layers but less neurons in each one. The former tends to do better at memorisation tasks, whereas the latter reduces variance and performs better when it comes to generalisation.

Shallow Feed-forward network with two hidden layers with 512 and 128 neurons. The dropout is set to 0.5.

Dense Same structure as for the shallow network with no dropout.

Deep Feed-forward network with five hidden layers, each half of the size of the previous one. The dropout is set to 0.5.

All these models are trained using CCE as the loss function and AdaMax as the optimiser.

The results are contained in table VI. The three networks achieve similar results. In comparison with previous methods, the RMSE loss function on the test set is much higher. The neural networks implemented don't seem to be suitable models for the recommender system.

TABLE VI
NEURAL NETWORKS RESULTS WITH THREE DIFFERENT STRUCTURES
(BATCH_SIZE= 20480, #EPOCHS = 20)

Type	Shallow	Dense	Deep
Parameters	#features = 48	#features = 256	#features = 48
RMSE	1.2622	1.2717	1.2741

VI. k-NEAREST NEIGHBOURS

k-Nearest Neighbours [5] is a method that computes predictions from the existing predictions of the k nearest neighbours. The following equation corresponds to the user based k-NN algorithm. A similar equation can be applied for an item based algorithm.

$$\hat{r}_{ui} = \frac{\sum_{j \in N_k^i(u)} sim(u,v) \cdot r_{vi}}{\sum_{j \in N_k^i(u)} sim(u,v)} \tag{3}$$

k is the number of nearest neighbours considered. The similarity function sim is a hyper-parameter that introduces the similarity measure. It can be understood as the inverse of distance metrics: the similarity takes on large values for similar objects and either zero or a negative value for very dissimilar objects. There exist different types of similarity functions. The Pearson baseline is a similarity function that compute the linear correlation between all pairs of users (or items). MSD represents the Mean Square Difference between all pairs of users (or items). k-NN with means is the same as above, except that the mean rating μ_u/μ_i of the user/item studied is taken into account. Finally, the k-NN with Z score is computed, which correspond to the normalisation of the equation 3. Results are given in Table VII. k-NN with means and with Z score achieve a lower RMSE than basic k-NN. This is not unexpected since the existence of user and item bias was already noted. Furthermore, in the case of k-NN with means, the item based method achieves better results.

TABLE VII
k-NEAREST NEIGHBOURS METHODS: UNSPECIFIED FEATURES ARE SET AS DEFAULT. [5].

Method	Similarity options	Features	Test RMSE
k-NN Basic	name: MSD user based	$k = 253$	1.0157
	name: MSD item based	$k = 23$	1.0235
k-NN with Means	name: Pearson baseline user based	$k = 500$	0.9993
	name: Pearson baseline item based	$k = 108$	0.9910
k-NN with Z score	name: Pearson baseline user based	$k = 500$	0.9979
	name: Pearson baseline item based	$k = 108$	0.9912

VII. SLOPE ONE

Slope One is another method from the *Surprise* library [5]. This model computes predictions $\hat{r}_{ui}$ for a user u and an item i as followed :

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i,j) \tag{4}$$

μ_u is the average rating of user u.
$R_i(u)$ is a set of items j rated by user u that were also rated by one of the users that rated item i.
$\text{dev}(i,j)$ is the average difference between the ratings of item i and those of item j :

$$\text{dev}(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj} \tag{5}$$

where U_{ij} is the set of all users that rated both items i and j.

This is a simple model that doesn't require any hyper-parameters. Table VIII shows the result.

TABLE VIII
SLOPE ONE RESULTS: NO HYPER-PARAMETERS

Method	Slope One
RMSE	1.0002

VIII. CO-CLUSTERING

Co-clustering [5] is a method that does a simultaneous clustering on rows and columns. This means that user's and item's characteristics are taken into account and that both parameters have a joint interaction. Even if two users match perfectly in their preferences, it is unlikely that they will both see the same movies. The predictions can be computed with the following formula :

$$\hat{r}_{ui} = \overline{C}_{ui} + (\mu_u - \overline{C}_u) + (\mu_i - \overline{C}_i) \tag{6}$$

where $\overline{C}_{ui}$ is the average rating in co-clusters and $\left(\mu_{u/i} - \overline{C}_{u/i}\right)$ is the mean rating per user/item minus the average of users'/items' clusters. Table IX summarises the optimised hyper-parameters of this model and the resulting RMSE.

TABLE IX
RESULTS OF CO-CLUSTERING METHOD.

Method	Features	RMSE
Co-Clustering	#clusters $u = 2$ #clusters $i = 19$ #epochs = 30	1.0049

IX. BLENDING OF MODELS

The blending method consists in taking the output of multiple models and combine them to make the final prediction. The combination is done as follows (using the above mentioned train-test split):

1) Selected models are trained on the train set.
2) Predictions are made on the test set.
3) A ridge regression is trained with all the predictions on the test set as x and the real ratings on the test set as y. The regularisation parameter α is determined by 3-fold cross-validation on the test set ($\alpha = 1.0$).
4) Selected models are trained once again but this time using 100% of the data.
5) Predictions are made on the submission set (whose real ratings are unknown) and the ridge regression is used to make the final prediction.

The weights found for each method in the ridge regression are summarised in table X. The final prediction achieves a RMSE of 1.017 on AICrowd. Note that this RMSE value is higher than all the previously estimated ones because the results of the final prediction had to be rounded to the nearest integer value. This was the submission format imposed by AICrowd.

TABLE X
RESULTS OF BLENDING METHOD.

Method	Weight	Method	Weight
Global Mean	0.5906	k-NN Zscore User	0.1881
User Mean	-0.6551	k-NN Zscore Item	0.1061
Item Mean	-0.5989	Co Clustering	-0.0044
MF ALS	0.4067	Slope One	0.1961
MF RMSE	-0.2107	MF (surprise)	-0.0417
k-NN Basic User	0.1404	Funk MF	0.0160
k-NN Basic Item	0.0920	SVD++	0.1875
k-NN Means User	0.2343	Shallow NN	0.1294
k-NN Means Item	0.0683	Deep NN	0.1680

X. IMPROVEMENTS

Hyper-parameter tuning is a work in progress and grid searches used could still be refined.

Popularity bias is a serious issue for any recommender system : it is prone to recommend popular items. Also, it fails to recommend new or less-known items because these items have either none or very little interactions. Some methods factored this in such as some of the SVD methods, k-NN and SlopeOne but other implementations could be used. A personalised re-ranking approach to increase the representation of less popular items in recommendations could be done as post-processing [6].

XI. CONCLUSION

The combined model scores a RMSE of 1.017 on the AICrowd platform. This work showed how effective ensemble learning is. Each specific model captures better a different aspect of the data set, i.e. baseline models capture the core statistical properties, matrix factorisation models discover the underlying latent features and neighbourhood models capture similarity relations. Thus, a combined model allows for a much more flexible model that can make use of various aspects of the data set.

REFERENCES

Sources verified on December 19, 2019:

[1] Martin Jaggi & Rüdiger Urbanke, *Class Project 2*, EPFL, https://github.com/epfml/ML_course/blob/master/projects/project2/project2_description.pdf

[2] Martin Jaggi & Rüdiger Urbanke, *Problem set 10 - Matrix factorisation and recommender systems*, EPFL, November 2017, https://github.com/epfml/ML_course/blob/344456134d9b217798c4050e62c3be4d3de96c1c/labs/ex10/exercise10.pdf

[3] Keras library documentation, *About Keras models*, https://keras.io/models/about-keras-models/

[4] Less Wright, *New State of the Art AI Optimizer: Rectified Adam (RAdam)*, https://medium.com/@lessw/new-state-of-the-art-ai-optimizer-rectified-adam-radam-5d854730807b, August 2019

[5] Surprise library, *prediction_algorithms package*, https://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html

[6] Himan Abdollahpouri and al., *Managing Popularity Bias in Recommender Systems with Personalized Re-Ranking*, University of Colorado Boulder, August 2019