Machine Learning (CS-433) : Project 2
Recommender systems

Dylan Bourgeois, SCIPER 224797 , SMT
Antoine Mougeot, SCIPER 216839, SSC
Philippe Verbist, SCIPER 284393, SMT
EPF Lausanne, Switzerland

***Abstract*—After leaving the model in which content providers broadcasted manually curated content through select channels, we now find ourselves in a sea of information. Individual filtering of the information stream has become a necessary condition to the use of many services, from video streaming to online shoppping. Recommender systems offer exactly this personalization : by analyzing patterns of user and item interactions, they are capable of predicting what a user is most likely to enjoy or buy. In this project we focus on recommending a movie for a given user, more specifically predicting the rating a user would give to a particular movie. We first set a series of baselines as a reference for recommendation quality. We then build upon the literature to show the efficacy of matrix factorisation methods in predicting user behavior. Finally, we investigate the accuracy gain by using deeper neural networks to predict the ratings. These methods will show the highest accuracy, providing competitive results in the Kaggle competition.**

I. Introduction

Netflix Prize: In October 2006, Netflix launched its now famous competition : it challenged anyone to achieve a higher accuracy than their current recommendation system (Cinematch). The prize was $1M$\$ to be awarded to the team which had achieved the highest improvement. The winner was declared in June 2009, "BellKor's Pragmatic Chaos" having bettered the RMSE by over 10%. One of their main insights was to learn how to optimally blend different predictors to yield a larger more accurate predictor [9, 10].

Recommender systems: The Netflix Prize was a prime example of a *collaborative filtering* recommender system : the system tried to predict a rating for an item based on its interactions with users. This is the setting of this project. There are also content filtering methods, which look at patterns in the objects themselves. Nowadays though, most of the recommender systems are hybrid : the incorporate information both from user behavior and from content features.

In this project we have ratings for every user/item interaction : it is an *explicit* recommender system (as opposed to implicit systems which look purely at the interaction, or just at a binary like/dislike feature, as is the case on Netflix now [1]).

II. Data Description and exploration

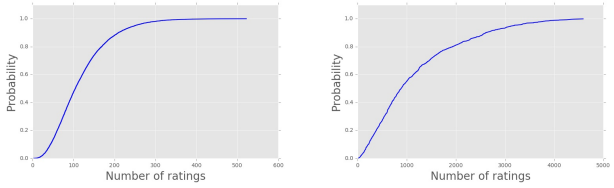
A. Dataset

This project is focused on predicting the ratings users give to a movie. We have at our disposal **1'176'952 ratings** sampled from **10'000 users** who rated **1'000 movies** on a integer scale from 1 to 5 (5 being the best grade). Users and movies are pseudonymized by an ID, and we have no additional metadata regarding these interactions. The data is considered dense for user/item interactions, most probably an artifact of the cleaning done before the data was provided to us : it has a sparsity of 11.8%. The test set of **1'176'952 ratings** is hidden from us and will be used to evaluate our predictions on Kaggle.

B. Data exploration

We suspect the data was cleaned before it was provided to us, as indicated by the low sparsity. We will confirm this intuition by checking the distributions of ratings over users and films :

Good participation of Users: We looked at the cumulative density of the number of ratings per user (Fig. II-B) and per movies (Fig. II-B) to make sure that there were no inactive users or movies which haven't been rated as could have damaged our prediction algorithms.



(a) (b)

Figure 1. Cumulative distribution function of the number of ratings per users (a) and per movie (b)

Absence of spammers: We looked at the distribution of the rating averages and variances to make sure that the data was free of uniform or random spammers.

C. User habits

Looking at the distribution of ratings averages (Fig. 2), we noticed that there were no spammers but that users had there own habits when it came to ratings movie. Some of

them might be systematically rating a movie lower or higher than others. We tried to take this effect into account in our models.

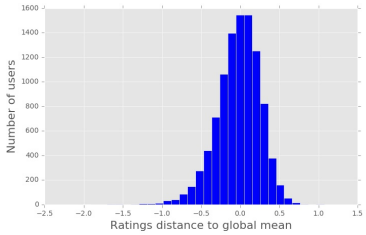


Figure 2. Distribution of the difference between user mean ratings and global mean

III. Models and Methods

A. Similarity based methods

We first evaluate baseline methods, designed through heuristics or through simple similarity matching. They should provide a good reference point for the efficiency of the more complex algorithms we design.

Cosine similarities: An intuitive way of seeing a recommender system is to have it be highly confident that a user will like a movie that is similar to other movies he likes, or that someone who has similar viewing patterns as his own would like (resp. dislike). We will use cosine similarity $s(u,v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{||\mathbf{r}_u|| \cdot ||\mathbf{r}_v||}$ for this baseline, a common measure of distance in high dimensional spaces. Formally a predicted rating $\hat{r}_{ui}$ for a user u and an item i would be computed as:

$$\hat{r}_{ui} = \frac{\sum_v s(u,v) \cdot r_{vi}}{\sum_v |s(u,v)|} \quad (1)$$

Global mean/median: A very simple method is to take all the values available in the training set and take the mean or the median value. We use this value as the prediction, it gives us a first satisfying baseline that we can use to compare other models.

User/Movie mean/median: Another simple model is to compute the mean or the median for the other or the movie and take those values as a prediction.

Movie mean/median corrected: To elaborate the previous models, we can take into account the correction for users that we presented in the previous section : $d_u = \overline{U} - \overline{u} \quad \forall u \in U$, where $\overline{u}$ is the average rating of user u and $\overline{U}$ is the average of all $\overline{u}$.

This feature gives us the prediction of a user u on a movie m: $p_{m,u} = \overline{m} + d_u$ where $\overline{m}$ is the mean or the median of the ratings for movie m.

B. Linear Regression

Based on our own experiences with rating movies, we looked for a correlation between the number of ratings of a movie and its ratings. For our dataset, we found a correlation of 0.7. We used this to create a model where the prediction for a movie was the estimation of the rating by the linear regression.

C. Matrix Factorisation methods

Matrix Factorisation: The user-item interactions can be conveniently represented as number of users ($n_u = 10'000$) by number of items ($n_i = 1'000$) matrix, denoted R (size $n_u \times n_i$), where each entry is a rating 4. Matrix factorisation (MF) methods learn two new matrices, U and I, which when multiplied back together give an estimate of R, denoted $\hat{R} = U \cdot I \approx R$.

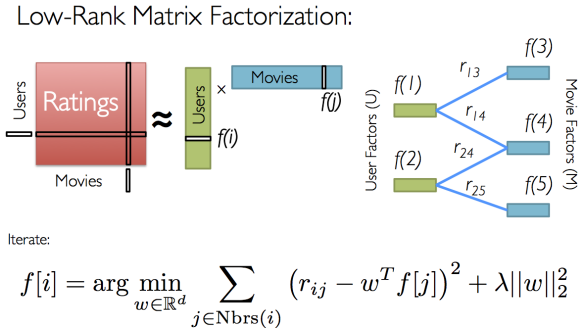


Figure 3. Matrix Factorisation method (Source)

U and I are $n_u \times L$ and $L \times n_i$, where L is the dimension of the latent space in which live the learned matrices. The latent spaces represent features of the users and the items, respectively. When the latent vectors are multiplied back together the prediction is a linear combination of these features (Figure 3).

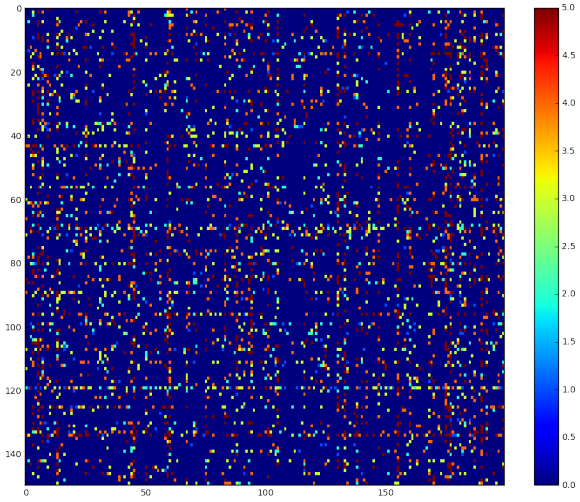


Figure 4. A subset of the user/item interaction matrix (150x200)

Learning the factorisation: The Surprise library (http://surpriselib.com/) was used to implement the factorization algorithms SVD, SVDpp and NMF. The default parameters are used except for SVD: $iter = 30$, $\gamma = 0.001$, $\lambda = 0.001$.

D. Blending

We created another model that combined, using Sequential Least-Square Programming (SLSQP), our predictions for the baselines and the models created using the Surprise library.

It allowed us a slightly better results but we chose to focus our efforts on our last model :

E. Deep recommender systems

1) Introduction: Matrix Factorisation methods extract features from the interaction matrix and predict the score through a linear combination of the learned factors. This is a simple regression model, but we can do better. In recent years, Artificial Neural Networks have seen great success in various domains of applications. They represent a more advanced form of prediction, but the general idea is similar : derive features from the input data and combine these features to make a prediction.

In this project we wanted to investigate these advances and apply at first some simple architectures to our problem in hopes of a better classification. After experiments in the regression setting, we settled on a multi-class classification problem : we predict the score as a category, generating the final predicted score by weighting each class by its confidence.

2) Learned features: The input layer of the network learns features of the input (user, item) pair that allow it to best classify the pair on the output : the error backpropagation helps not only learn a classifier but also an embedding.

We figured that this embedding might not be sufficient so we manually designed some additional representations of the data to feed to the classifier. These new embeddings can simply be concatenated to the input layer to give for a richer input that the network can learn from. In our method we computed four different embeddings :

- Locally Linear Embeddings (LLE) [12] which computes non-linear but neighborhood preserving embeddings
- Factor analysis (FA) where the embeddings reconstruct the original data through a linear combination with some gaussian noise
- Spectral Embeddings which learn a lower dimensional graph representation of the data, making it an obvious choice to model relationships between users or movies.
- Non-negative Matrix Factorisation (NMF) which we have seen in III-B.

3) Network configurations: One of the main issues with Neural Networks is the sheer amount of configurations they require. Once an architecture has been selected for the purpose, we still need to decide on many parameters.

Activation Function: The choice of a non-linearity is a common debate in the Deep Learning community, between the well understood and battle tested activations such as sigmoid or tanh, and the newer ones promising solutions to some of the shortcomings (Swish, ELU, SELU, PReLU, ...). In our case we chose to use the ReLU activation function : $f(x) = max(0, x)$. This is a good choice for quicker training (the gradient is simple 0 or 1 depending on the sign of the input), and helps learn sparse representations [4].

Minibatch: To accelerate the training of the networks, we resorted to using a mini-batch setting : we can compute the gradients on small subsets of the training data and update the model more often, but this leaves us vulnerable to a high variance in the gradient (causing unstable loss values). We settled on a standard size of 2048, a good trade-off between what our machine could handle and stabilizing the learning process.

Loss: Since we are learning a categorical classifier, a natural choice is to optimize for categorical cross-entropy, which takes into account the "proximity" of the predicted solution, especially since we are predicting categories. If we were to try regression for the prediction of the score, we would have used the Mean Squared Error (MSE) metric, a more expressive metric in continuous space than for classification accuracy.

Optimizer: We started by using the standard Stochastic Gradient Descent (SGD) but also experimented with some other optimizers, most notably Adam [8] which gave us the best results experimentally. The learning rate was tuned manually : if the network converged very quickly then started overfitting we would bring the learning rate down ($lr = 0.001$).

Regularization: Especially in a setting like our with sparse data, we need to be very weary of not letting our model overfit around the training data. The usual steps of creating a validation set were taken, but we also added a few additional regularization steps. First of all the embeddings have a regularization term : they are computed by minimizing a loss $\mathcal{L}$ which is composed of a reconstruction error $||\hat{r}_{ui} - U_u \cdot I_i||^2$ and a regularization term $\lambda \cdot ||\hat{r}_{ui}||^2$ (we set $lambda = 1e-4$).

A popular technique to reduce overfitting in deep networks is dropout [13] : by randomly pruning a portion of neurons at each pass, we force the network to distribute the information it learns across the networks. Otherwise we could end up reinforcing always a subset of neurons, drowning more subtle signals coming from other neurons.

Finally we add batch normalization to each layer and minibatch. this allows the network to be trained faster, by allowing for higher learning rates, but also acting as a regularizer [6].

Output layer: We have mentioned the fact that we are learning a categorical classifier, but the final result we need is a single score. A common method is to apply a softmax activation function, which squashes a high dimensional vector (the output of the last learning layer in this case) into n values which sum to 1. In our case, $n = 5$ since we are trying to predict a score from 1 to 5, and the values returned by the softmax can be interpreted as the confidence in each class [2]. Instead of simply choosing the class with the highest confidence, we will mix the predicted ratings p_i and their respective confidence s_i to obtain the predicted score for a user u / item i pair :

$$\hat{r}_{ui} = \sum_{i=1}^{5} s_i * p_i \tag{2}$$

4) Architectures: Unless otherwise specified, all the architectures described here subscribe to the configuration we have described in III-E2.

Shallow network: The first network we designed was a very simple feed-forward multi-layer perceptron, with 2 hidden layers (Fig. 5). A simple architecture, it still provided a good base for the design of the other networks.

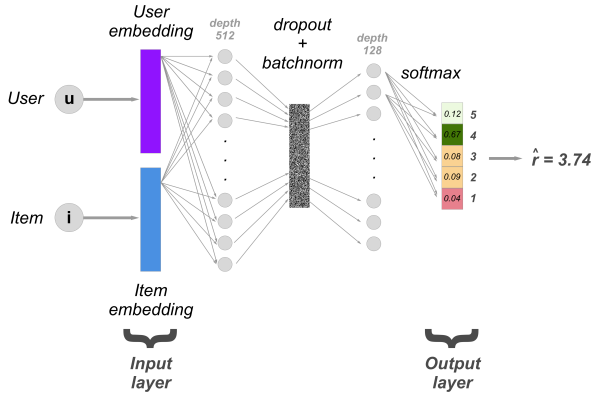


Figure 5. Shallow Network architecture

Dense network: The second approach we tried was a deeper, denser network. It has the same number of layers all throughout until the 5 category output (Fig. 6). In this iteration we show a version with 3 fully connected hidden layers. This architecture had more learning capabilities, but still fell short of yielding very good results.

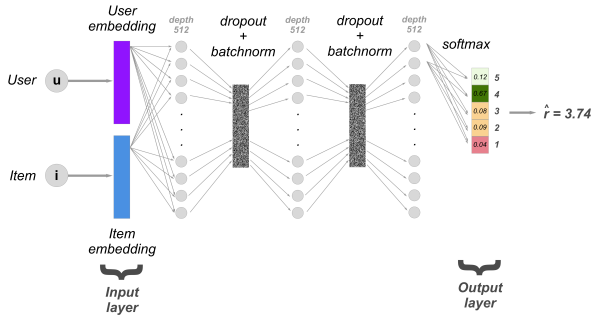


Figure 6. Dense Network architecture

Deep network: Our final choice settled on the largest and deepest network. In essence it's architecture is not particularly complex : it has 5 hidden layers, each half as large as the previous to learn more powerful features as the depth grows (Fig. 7).

F. Experimental methods

In this section we would like to mention some additional methods we came across in our research. Some led to solution stubs, others were just not adapted to our problem setting but could have been worth investigating but inspired us in our final solution. One particularly of interest to us

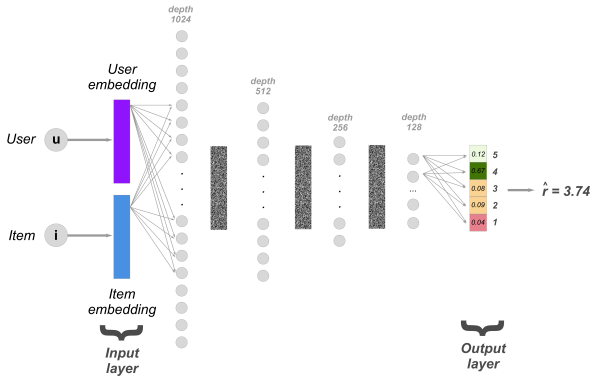


Figure 7. Deep Network architecture

would have been to apply work by van den Berg et al., Kalofolias et al. or Monti et al., as part of a new domain of deep learning on non-euclidian spaces, notably graphs [3]. The advantage of these methods is that they directly use the relationships (what we tried to emulate with similarities) between users and/or movies to predict the rating of a movie. This method has for example been successfully been applied to movie recommendation problems with datasets like the MovieLens 100K which is similar to our own [11].

Several interesting approaches using denoising autoencoders were found to yield promising results by considering the missing ratings as data the autoencoder should reconstruct [14] [15].

Finally we were interested by some approaches using Recurrent Neural Networks, which could capture the temporal dimension of the user's ratings (e.g. with session based recommendations [5] or the evolution of user preferences over time [17]).

IV. Results

In this section we present our results on the project. As mentioned earlier, the Deep Neural Network method was the one with the best results on this project. It allowed us to be in the provisional first place on the Kaggle competition for up until the last couple days, finishing in the top 3 (team name : Neural Netflix). The baselines gave us a good start for accuracy measure. The results are shown in I

Category	Method	RMSE
Similarity	Movie	3.4896
	User	3.1045
Mean	Movie	1.0294
	User	1.0850
Median	Movie	1.0978
	User	1.1398
Matrix Factorisation	NMF	1.0115
	SVD	1.0105
	SVDpp	1.0404
Blending	SLSQP	1.0046
Deep Recommender System	Shallow Network	0.97697
	Dense Network	0.97746
	Deep Network	0.97408

Table I
ACCURACY MEASURES (RMSE) FOR VARIOUS METHODS.

V. Discussion

Implementation details: For the simpler models and data explorations, we use the famous combination of jupyter notebooks, numpy and scikit-learn. For the neural network implementation we chose to use Keras for its simplicity : it wraps powerful computation through its Theano and Tensorflow backends, but presents a simple, declarative API which makes it very simple to try several architectures.

The models were trained locally on a `2.3 GHz` Intel Core i7 CPU, and later on a Tesla M60 GPU. We provide a Docker image (`dtsbourg/cs433-project`) that, thanks to Keras' wrapping of backends, can work on CPU or GPU and can be deployed anywhere.

Hurdles and joys: While the deep learning methods did end up winning in absolute scoring, they were considerably more intensive in terms of resources (time and computation) but also deceptively opaque : tuning parameters and selecting the right architecture is time-consuming mostly because of the non-linear relationships between the different choices (activation, loss, batch size, ...).

VI. Conclusion

Through this project we have been given an opportunity to study a large band of the state of the art methods in Recommender Systems, though more specifically in content filtering. They range from simple similarity based methods to large and complicated deep network architectures. Once again we realized that the determining factor in obtaining a satisfactory accuracy was feature engineering. Not only did the manually extracted features allow us to obtain a high score, but they are also at the heart of all the other methods that are being deployed today in academia and in the industry : incorporating more information always seems to help.

References

[1] Binary upvotes in netflix. https://media.netflix.com/en/company-blog/goodbye-stars-hello-thumbs.

[2] John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 211–217. Morgan-Kaufmann, 1990.

[3] M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34:18–42, July 2017. doi: 10.1109/MSP.2017.2693418.

[4] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

[5] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015. URL http://arxiv.org/abs/1511.06939.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[7] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix Completion on Graphs. *ArXiv e-prints*, August 2014.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42 (8):30–37, Aug 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263.

[10] Yehuda Koren. 1 the bellkor solution to the netflix grand prize, 2009.

[11] F. Monti, M.M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *CoRR*, abs/1704.06803, 2017. URL http://arxiv.org/abs/1704.06803.

[12] L. K. Saul and S. T. Roweis. An introduction to locally linear embedding. Technical report, 2000.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[14] F. Strub and J. Mary. Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs. In *NIPS Workshop on Machine Learning for eCommerce*, Montreal, Canada, December 2015. URL https://hal.inria.fr/hal-01256422.

[15] F. Strub, J. Mary, and R. Gaudel. Hybrid recommender system based on autoencoders. *CoRR*, abs/1606.07659, 2016. URL http://arxiv.org/abs/1606.07659.

[16] R. van den Berg, T. N. Kipf, and M. Welling. Graph Convolutional Matrix Completion. *ArXiv e-prints*, June 2017.

[17] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. Proceedings WSDM '17, pages 495–503, New York, NY, USA, 2017. ACM. doi: 10.1145/3018661.3018689. URL http://doi.acm.org/10.1145/3018661.3018689.