Advanced Machine Learning: Recommender System

A. Reutov, E. Marquez, N. De Wit, N. Hussein
{aar1g14, esm1g14, ndw1u13, nmeh1n13}@soton.ac.uk

Abstract—We present in this report our effort in a competition [1] to build a recommender system able to predict user ratings for a given set of jokes.

While the problem can be classically solved as collaborative filtering, it can be modernly classified as a matrix completion problem. We first provide analysis of the given data, and describe our main methods (notably completion by averaging, KNN imputation, regression, and nuclear norm minimisation) individually before combining them. We then measure the performance of our team in the competition, analyse the results, and compare them to those of the other teams before presenting a conclusion.

Our key result is that, using a combination of these methods with weighted average and fine tuning, we secured the 1st place in the competition.

Index Terms—collaborative filtering, matrix completion

1 Introduction

Recommender systems make one of the major problem domains in machine learning. The system aims to predict ratings that users would give to certain features using the ratings of other users on the same features.

Classically, the problem is solved through collaborative filtering. In this method, we use the ratings we have to find patterns, then use these patterns to predict the missing ratings of other users. However, the state of the art techniques used to solve this problem are usually rather related to matrix completion methods. Although there are different approaches used in matrix completion, they all share the same goal: to reconstruct the full matrix depending on the existing user ratings, while minimising the error between the ratings in the original and the reconstructed matrix. Several techniques can be applied, the most notable of them being matrix factorisation (decomposition) and convex optimisation.

1.1 Problem Definition

The problem we will solve is thus a recommender system. We are given a training set, which is a $m * n$ matrix where $m = 24k$ and $n = 100$. The matrix represents the users' ratings (m observations) for a number of jokes (n features). Users rate how good or bad a joke is by attributing a value ranging between -10 and 10.

However, the training data is not complete. For a majority of users, there are missing ratings for some of the jokes. Specifically, these missing values account for 27.5% of the whole ratings in the training set. Missing ratings are represented by number 99.

We are also given a test set, which contains $3K$ lines of user ratings, for the same jokes as in the training set. Some of these ratings are missing and not required to be predicted (represented by number 99). Only 3 missing ratings per user are required to be predicted (represented by number 55) from this set. Figure 1.1 shows a diagram for the given training and test sets.

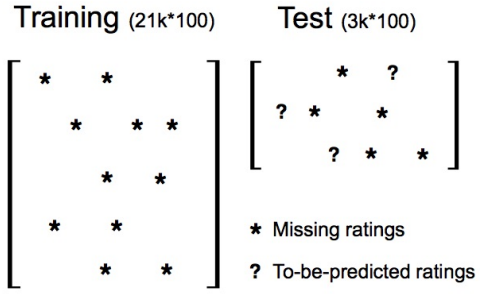

Fig. 1.1. Illustration of the given training and test data with missing user ratings.

1.2 Kaggle Competition

This problem is part of a running, in-class competition hosted on Kaggle [1]. This gives us the opportunity to submit the result of our work and compare it with other teams'.

1.3 Data Analysis

The dataset has some particular characteristics. Firstly, while the training set contains 21983 rows representing the different users (data vectors), the

This is a report for the module COMP6208:Adv. Machine Learning's coursework, MSc Artificial Intelligence, University of Southampton.

test set only contains 3000 users. As we have established, both training and test sets have 100 features (jokes).

Out of the 27.7% of missing values spread randomly in the training set, each user can have from 0 to 68 of them. Similarly, the test set has 27.1% of missing values, and those values can go from 3 to 69 per row. The 55 values in the training set (which represent 3% of it) are the only ones used to measure the estimation accuracy through the Kaggle website, but both 55 and 99 parameters are conceptually the same.

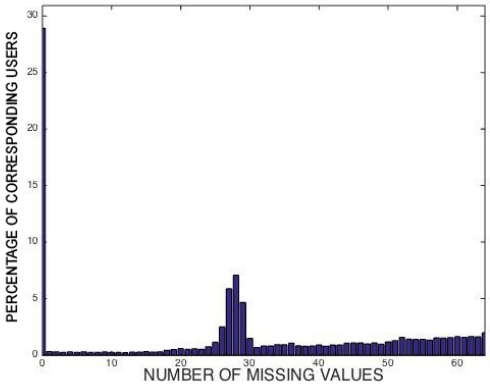

Fig. 1.2. Percentage of users having each number of missing values in the training set

Figure 1.2 shows the proportions in which the values are missing. It can be observed that 28% of the users in the training set have no missing values (around 6000 users have rated all possible jokes), which is an advantage that can be taken into account when testing or tuning need to be performed.

In order to estimate the amount of correlation in the dataset, we used Pearson's coefficient, which measures the linear correlation between variables. It was used on the rows that were complete in the training set, generating a 100x100 matrix (for the 100 jokes). It was observed that there was no good linear correlation (above $R = 0.75$) between jokes, meaning that linear models are not expected to provide good results.

The testing of our performance on the datasets was done locally and remotely. We developed an algorithm to calculate the Root Mean Squared Error (RMSE) using an "unofficial" test set (by taking off some extra values from the training set), created only to locally test the different algorithms. On the other hand, the Kaggle website provided a fast response after a submission, allowing for remote testing directly on the test set. Surprisingly, the accuracy obtained from both tests showed significant differences. We infer that this gap comes from the peculiar behaviour of the users that were selected to constitute the test set. It was indeed detected that some users' ratings were not consistent or rational (all identical ratings, etc). This generates uncertainty and atypical points in the dataset, making harder to fit a model to it.

The next data characteristic was determined by plotting an histogram of the user ratings. It can be observed in Figure 1.3 that the users mostly rated between 0 and 7, and 63% of them rated positive values. Moreover, most of them did not use the entire range of rating values, but had fixed limits that depended on the user.

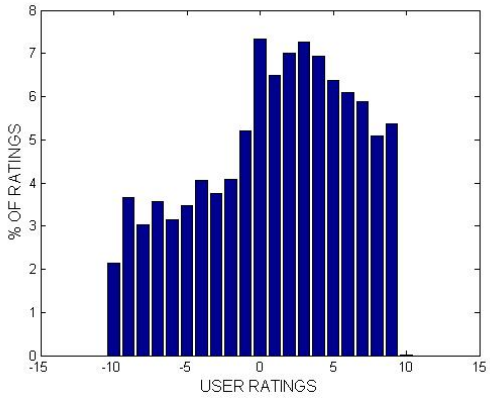

Fig. 1.3. Histogram of ratings density

2 METHODS

We chose the modern approach to solve the problem at hand: matrix completion. The main reason behind this is that the classical approaches (i.e. collaborative filtering) are not the most suitable for this dataset, since we are not producing predictions for the same set of users as in the training set, but rather working on a new set of users for which the available data is quite sparse, making it sensitive to the cold-start problem.

The following are the individual methods that were carried out to solve the problem. The relationships between them will be developed in the next section.

2.1 Completion by averaging

The first and most basic technique we applied was to average the available values in various ways in order to fill the missing matrix cells. Five different variations have been developed:

- **Joke average**: we give the average rating of the joke to every user that has yet to rate it (i.e. to each missing value that is located in the joke's column).
- **User average**: we rather give the average rating of a user to every joke that he has yet to mark.
- **Mixed average**: we balance the two previous averages ($\alpha * average + \beta * useraverage$) with empirical choice of the weights to balance the two aspects.
- **User offset**: we want to take into account a user's tendency to give higher or lower ratings in a more sensible manner. For each joke that a user has already rated, we calculate the difference between the joke's average rating and this user's rating, which gives an indicative offset for the user. We can then add that offset to the joke average.
- **Gaussian assumption**: following a suggestion from [9] for the Netflix Problem, we can consider that the values per user and per joke are taken from Gaussian distributions. This approach presents the advantage of not getting mislead by the cases where only few ratings are available.
  The variance in the ratings of each joke is calculated, and we obtain a ratio K for each variance with regard to all other jokes. The average rating of a joke is then not simply calculated through a simple $\frac{sumOfRatings}{numberOfRatings}$ formula, but by inserting that supplementary information:
  $$\frac{averageOverAllJokes * K + sumOfRatings}{K + numberOfRatings} \tag{2.1}$$
  The same is done for the user offset, as described in the previous point.

We found this last variation to be the most succesful one, building up on all the previous ones and taking into account an extra particularity of the dataset.

2.2 Singular Value Decomposition

A second technique, upon which many of the more advanced manipulation have been built, is the Singular Value Decomposition (SVD). SVD allows for the decomposition of a matrix into a less expensive representation. When truncated, that representation becomes a simplification of the model and can thus offer the generalisation that we seek.

A given matrix $\boldsymbol{A}$ of rank R can thus be decomposed as follows:

$$\boldsymbol{A} = \boldsymbol{U} * \boldsymbol{S} * \boldsymbol{V}^T \tag{2.2}$$

where $\boldsymbol{U}$ contains the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$, $\boldsymbol{S}$ the singular values of $\boldsymbol{A}$ and $\boldsymbol{V}$ the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$.

Once decomposed, the full matrix can be rewritten as the linear combination of the vector composing these submatrices:

$$\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{2.3}$$

However, with the values in the submatrices sorted in decreasing order, we obtain a situation where the significance of the n^{th} term becomes progressively lower. By limiting the sum to K values instead of R (with $K < R$), we thus can get an approximation of our original matrix, reduced to rank K.

The interpretation of this technique in our domain of application is quite intuitive: while in the full matrix of u users and j jokes, u independent rows of ratings are stored, the truncated SVD representation keeps only K different profiles of users in matrix $\boldsymbol{U}$ and K profiles of jokes in matrix $\boldsymbol{V}$. For example, if the ratings are correlated through the type of joke (group of black humour jokes, group of jokes using puns...), the SVD decomposition will give profiles where each vector in $\boldsymbol{U}$ indicates how much these users like or dislike every type of joke, and each vector in $\boldsymbol{V}$ shows the presence of that aspect in the joke. Both vectors are then related through the singular values contained in matrix $\boldsymbol{S}$.

Using this structure, the estimated rating for a joke by a particular user can be calculated through the dot product of the corresponding vectors in the submatrices.

The main disadvantage of this method would be that SVD decomposition can only be applied on a full matrix: while it is a good way to obtain a generalisation of our data, the missing values must be initially filled through another method. In the same way, it is applicable on known users and jokes only, and cannot fill the missing data for a completely new user since it does not include the construction of a model.

2.3 Incremental SVD

The singular value decomposition can also be applied in an incremental process [9]. After computing a first SVD, we replace the missing values in the original matrix by the estimation coming from the

decomposed submatrices. The other values, that were originally filled, stay the same. We then feed this modified matrix back in the process and repeat these steps until the resulting matrix converges.

This method is sometimes presented as a type of Estimation-Maximisation algorithm [13], in which the maximisation step consists in calculating the decomposition and the estimation step fills the missing values with the ones calculated by approximation.

2.4 Singular Value Thresholding

The Singular Value Thresholding is a completion method using SVD that was first described by [10].

While the matrix completion problem can be formulated as a rank minimisation problem (one must find the matrix $\boldsymbol{X}$ that has the minimal rank while still giving the exact non-missing values), that formulation makes the problem NP-hard. Fortunately, a convex relaxation of the problem can give an approximation of it. The nuclear norm minimisation problem asks one to find the matrix $\boldsymbol{X}$ that has the minimal nuclear norm (sum of the singular values), while still giving the exact non-missing values [12].

The iterative algorithm SVT can then be used to solve this relaxation. Using two matrices X (containing the current estimations) and an initial Y (working matrix), it goes through two steps per iteration:

1) We shrink the $\boldsymbol{Y}$ matrix gotten in the previous step to get a new estimation matrix $\boldsymbol{X}$. The shrinkage operation of the $\boldsymbol{Y}$ matrix consists in compressing the middle part of the SVD decomposition of $\boldsymbol{Y}$ ($\boldsymbol{U} * \texttt{shrink}(\boldsymbol{S}) * \boldsymbol{V}$). Only the singular values of $\boldsymbol{Y}$ displayed in S that are bigger than a parameter τ are kept, while the others are replaced by zeros.
2) We recalculate the $\boldsymbol{Y}$ matrix based on the previous $\boldsymbol{Y}$, a parameter δ, and the difference between the non-missing values and the estimations we get through $\boldsymbol{X}$.

2.5 K-Nearest Neighbour

KNN is another simple technique for matrix completion. Using a particular kind of distance measure, it consists in finding the closest known entity and using its value to predict our target. While we mainly used MATLAB inbuilt function knnimpute (with Euclidean distance), we first tried another implementation as it gave more control over the process and even made our own, trying to combine some averaging and weighting ideas. For example, we chose not to use the closest neighbour's value directly, but to give a weighted sum of the closest neighbours with weights corresponding to the rate of similarity. Results of all our tries with this method were very similar in terms of accuracy (from 4.71 to 4.77), although the inbuilt function was considerably faster. In terms of performance, KNN gave a good overall result, and was later used by other methods to get a good initial guess for missing values.

2.6 Augmented Lagrange Multiplier

Augmented Lagrange Multiplier (ALM) is a fast and scalable approach for solving Robust PCA problem. It is able to recover a low rank matrix with some of its entities being corrupted. With a little change to the problem definition, it can thus be applied to matrix completion problem. It is also based on SVD, however it aims to update weights in an online fashion.

Authors of the paper [7] state three main properties of this algorithm:

- Superior convergence properties
- Easy-to-tune step size
- Convergence to optimal solution in a limited number of iterations

Open Source code provided with this paper [8] included three types of ALM algorithms. However, trying them out on our dataset with various parameters did not yield satisfactory results.

2.7 Nuclear Norm Basis Pursuit

In this method, we start by the given matrix of the user ratings $\boldsymbol{M}$. In order to recover matrix $\boldsymbol{M}$, we depend on all the available observations in it. Let $\boldsymbol{M}(\boldsymbol{\Omega})$ be the set of available observations in $\boldsymbol{M}$, where $\boldsymbol{\Omega}$ is the set of indices for these observations. If the available observations $\boldsymbol{M}(\boldsymbol{\Omega})$ are large enough, we are able to recover $\boldsymbol{M}$ by solving the following minimisation problem:

$$\begin{aligned} \text{minimise} \quad & \text{rank}(\boldsymbol{X}) \\ \text{subject to} \quad & \boldsymbol{X}_{ij} = \boldsymbol{M}_{ij} \quad (i,j) \in \boldsymbol{\Omega} \\ & \boldsymbol{X} \in \mathbb{R}^{n*n} \end{aligned} \tag{2.4}$$

According to the researchers E. Candès *et al.* [2], it is an NP-hard minimisation problem. The trick is thus to search for another representation for the term rank($\boldsymbol{X}$). As briefly mentioned before, that is when the nuclear norm (also known as the trace

norm or Ky-Fan n-norm) comes to the rescue. For a matrix $\boldsymbol{X}$ with SVD composition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, its nuclear norm can be proved as the following:

$$\begin{aligned}\|\boldsymbol{X}\|_* &= trace\left(\sqrt{\boldsymbol{X}^T\boldsymbol{X}}\right) \\ &= trace\left(\sqrt{(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T * (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)}\right) \\ &= trace\left(\sqrt{\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T * \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T}\right) \\ &= trace\left(\sqrt{\boldsymbol{V}\boldsymbol{\Sigma}^{\boldsymbol{2}}\boldsymbol{V}^T}\right)\end{aligned} \tag{2.5}$$

By circularity of $trace$, we have:

$$\begin{aligned}\|\boldsymbol{X}\|_* &= trace\left(\sqrt{\boldsymbol{V}\boldsymbol{\Sigma}^{\boldsymbol{2}}\boldsymbol{V}^T}\right) = trace\left(\sqrt{\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^2}\right) \\ &= trace\left(\sqrt{\boldsymbol{\Sigma}^2}\right) = trace\left(|\boldsymbol{\Sigma}|\right)\end{aligned} \tag{2.6}$$

Therefore, the nuclear norm is defined as the sum of the singular values of the matrix $\boldsymbol{X}$.

$$\|\boldsymbol{X}\|_* = \sum_{k=1}^{n} \sigma_k(\boldsymbol{X}) \tag{2.7}$$

If our matrix $\boldsymbol{X}$ has rank r, the number of non-zero singular values is r as well. This means that instead of minimising the matrix' rank, we can minimise its norm. By substituting equation 2.7 in equation 2.4, we can express the modified minimisation problem as follows:

$$\begin{aligned}&\text{minimise} && \|\boldsymbol{X}\|_* \\ &\text{subject to} && \boldsymbol{X}_{ij} = \boldsymbol{M}_{ij} \quad (i,j) \in \boldsymbol{\Omega} \\ & && \boldsymbol{X} \in \mathbb{R}^{n*n}\end{aligned} \tag{2.8}$$

The researchers E. Candès *et al.* [2] did not stop at that point and took the optimisation of the problem one step further. The nuclear norm is reformed to reduce the complexity of the minimisation. The matrix $\boldsymbol{X}$ can be reformed as: $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{R}^T$, where $\boldsymbol{L}, \boldsymbol{R}$ are $n * k$ matrices. Many common factorisation methods can be applied here, but a common approach is to minimise the Frobenius-norm for these matrices. The minimisation problem is then described as:

$$\begin{aligned}&\text{minimise} && \frac{1}{2}\left(\|\boldsymbol{L}\|_F^2 + \|\boldsymbol{R}\|_F^2\right) \\ &\text{subject to} && \boldsymbol{L}\boldsymbol{R}^T = \boldsymbol{X} \\ & && \boldsymbol{X}_{ij} = \boldsymbol{M}_{ij} \qquad (i,j) \in \boldsymbol{\Omega} \\ & && \boldsymbol{L} \in \mathbb{R}^{n*k},\ \boldsymbol{R} \in \mathbb{R}^{n*k}\end{aligned} \tag{2.9}$$

where $\|.\|_F^2$ is the Frobenius-norm. With this step, we manage to reduce the decision variables from $n * n$ to $2nr$, where n is the length of matrix $\boldsymbol{X}$, r is it's rank.

We found a MATLAB toolbox [3] [4] that implements the nuclear norm minimisation as illustrated. We used it to predict the missing values of the test set with 3 different experiments:

1) Apply the nuclear norm minimisation on the test set alone.
2) Combine the training and test set into one matrix, then apply the nuclear norm minimisation on it.
3) Select the users from the training set that are relevant to the ones in the test set. With this step, we get rid of the training observations that are either irrational or irrelevant to the test observation. After that, the test set is added to the selected observations and we apply nuclear norm minimisation on the resulting matrix.

We found that the third experiment produces the best performance. The most likely reason for this difference is that getting rid of the observations with irrelevant ratings increased the method's accuracy.

2.8 Bayesian Probabilistic Matrix Factorisation

Another model used to solve this kind of problem is the low-dimensional factor model. Probabilistic matrix factorisation (PMF) is considered a probabilistic approach for this model. It can also be considered a graphical model.

While SVD converges by solving the sum-squared distances to the target matrix (which can be calculated by non-convex optimisation), PMF converges by solving sparse semi-definite programming. In addition, it is scalable to large datasets, where user ratings are in millions, which is considered one of the key advantages of the PMF. Another advantage is the ability of PMF to perform well on users with few ratings.

The researchers R. Salakhutdinov *et al.* [5] presented a Bayesian approach of the PMF model. Suppose we have a matrix $\boldsymbol{R}$ that represents the ratings of n users for m jokes. It can be factorised as: $\boldsymbol{R} = \boldsymbol{U}^T\boldsymbol{V}$ where $\boldsymbol{U}$ is the latent features for users, $\boldsymbol{U} \in \mathbb{R}^{d*n}$ and $\boldsymbol{V}$ is the latent features for jokes, $\boldsymbol{V} \in \mathbb{R}^{d*m}$. Using probabilistic linear model with Gaussian distribution, the conditional distribution of a user rating can be defined as:

$$p(\boldsymbol{R}|\boldsymbol{U}, \boldsymbol{V}, \sigma^2) = \prod_{i=1}^{n}\prod_{j=1}^{m}\left[\mathcal{N}(\boldsymbol{R}_{ij}|\boldsymbol{U}_i^T\boldsymbol{V}_j, \sigma^2)\right]^{\boldsymbol{I}_{ij}} \tag{2.10}$$

The Gaussian priors of latent feature vectors for the users and the jokes are given as the following:

$$p(\boldsymbol{U}|\sigma_U^2) = \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{U}_i|0, \sigma_U^2 \boldsymbol{I}) \\ p(\boldsymbol{V}|\sigma_V^2) = \prod_{j=1}^{m} \mathcal{N}(\boldsymbol{V}_j|0, \sigma_V^2 \boldsymbol{I}) \tag{2.11}$$

It is suggested that these priors have zero-mean and symmetric covariance (said to be spherical). Then, we can use MAP (maximum a posterior) as the learning algorithm. It is equivalent to minimising the sum of squared errors:

$$E = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} \boldsymbol{I}_{ij} \left(\boldsymbol{R}_{ij} - \boldsymbol{U}_i^T \boldsymbol{V}_j \right) \\ + \frac{\lambda_U}{2} \sum_{i=1}^{n} \|\boldsymbol{U}_i\|_F^2 + \frac{\lambda_V}{2} \sum_{j=1}^{m} \|\boldsymbol{V}_i\|_F^2 \tag{2.12}$$

where λ_U, λ_V are quadratic regularisation terms, $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$ and $\|.\|_F^2$ is the Frobenius-norm.

It is clear from the equation 2.10 that PMF can be considered a probabilistic approach to SVD. The same researchers suggested to use logistic function $g(x) = 1/(1+exp(x))$ when using the feature vectors of users and jokes. This is to prevent getting ratings from outside the valid range. The equation 2.12 conditional distribution can thus be written as:

$$p(\boldsymbol{R}|\boldsymbol{U}, \boldsymbol{V}, \sigma^2) = \prod_{i=1}^{n} \prod_{j=1}^{m} \left[\mathcal{N}(\boldsymbol{R}_{ij}|g(\boldsymbol{U}_i^T \boldsymbol{V}_j), \sigma^2) \right]^{\boldsymbol{I}_{ij}} \tag{2.13}$$

Luckily, the researchers offered their MATLAB implementation as open source for research [6]. We used their project in one of our experiments. Originally, the paper and the project of the researcher was built to solve the Netflix problem, but with a bit of effort we succeeded in modifying it in order to run on our problem's dataset.

2.9 User clustering

Data analysis revealed that users follow different patterns. We thought of clustering the users with respect to these distinctive patterns. The intuition behind this method was to minimise the number of observations, *i.e.* the number of users in the training set. Consequently, this would help in decreasing the required computation, and ruling out any irrational users.

We experimented several techniques to cluster the users. The first of them is K-means clustering. We empirically chose the hyper-parameters of the algorithm: the number of clusters/centers and the initial centers. After a few experiments, we concluded that K-means was not a good choice, because the number of missing ratings for certain users were over 50% of the ratings. Since we only depended on a small number of ratings for those users, we could not assign them to a specific cluster, as they seemed to fit in many of them simultaneously.

Another technique we used was pair-wise correlation to cluster the users. We built a matrix with pair-wise correlation coefficients between each two different users. Different methods to calculate the pair-wise correlation were experimented. For example, Pearson correlation is calculated as the following:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.14}$$

where x, y are different users, x_i are the ratings of user x, and y_i are the ratings of user y, $\bar{x}$ is the mean rating of user x and $\bar{y}$ is the mean rating of user y.

Another coefficient is the covariance correlation, calculated as the following:

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}} \tag{2.15}$$

where $C(i,j)$ is the covariance between users i and j.

After several experiments, we concluded that clustering the users by correlation coefficients was not an efficient idea. That is because these correlation techniques use linear ranking, while the users are non-linearly correlated.

2.10 Completion by Linear Regression

This method is based on a linear regression algorithm, using the part of the data that is complete (rows with no missing values). It aims to find the best linear combination of certain jokes to predict one missing value.

Since each user is unique, the training set was not static and changed according to the user's missing ratings. This way, each user would get a set of weights to determine each missing value. For example, if a particular row has 15 missing values, it must have 15 set of weights to calculate

each missing parameter. As it can be expected, this method is computationally expensive since there is a total of 696 845 missing values, which means as many pseudo-inverses.

The method starts by taking an user and extracting all of his missing values. It then extracts the same jokes from the whole local training set (the submatrix containing no missing parameters), generating a temporary training set for its process, and sets as the target the row of the rating to be predicted. Finally, a linear regression model is trained and one value is predicted. The iterative process then sets another value to predict and trains a different model.

At the end, both our initial training and test sets are completed, through successive predictions from these linear regression models.

We also tried using the same algorithm to first predict the missing values in the original training set, and only then using the now complete training set to train further models and fill the values in the test set. However, this generated worst results due to the fact that the error of the predicted values on the training set were carried out to the test set.

2.11 Radial Basis Functions

The main advantage of this method in comparison with the linear regression algorithm is that it can fit any function, and the non-linearity helps to manage the uncertainty in the dataset.

Firstly, we used an exponential function, as it is the most common one. The RBF model is then given by:

$$g(x) = \sum_{k=1} \lambda_k * \phi(|x - c_k|) \tag{2.16}$$

The training and test sets were concatenated in order to transform the problem into a matrix completion problem. The matrix was then divided into sections (N sections), grouping the rows with similar missing values, and a separate RBF model was trained for each section. Every section had its own set of targets.

Once all the models were fully trained, they were used to predict the rest of the data. This generated $N - 1$ predictions per section, since RBFs are not used to predict values in the section from which they had been trained.

All the predictions for a single row were then averaged by a factor that depended on the similarities between sections (each sections' relationship had its own factor).

Finally, the test data is extracted from the concatenated matrix, giving us our result.

We later tried applying the same algorithm, but with different polynomial functions (instead of the exponential one).

One important parameter to take into account is the number of clusters (number of hidden units), which can diminish the generalisation error. Unfortunately there is no perfect algorithm to determine the right number of clusters. It was found empirically that 78 clusters gave the best result.

The results of these models were not successful. However, further experiments must be done in order to determine if there is a more adapted function to fit this particular dataset.

2.12 MLP Neural Networks

This method is very similar to the RBF method. The main differences between the two are that the number of units must be bigger than in the RBF, and that the training is performed faster, since is not done using the `CVX` toolbox but the in-built function in MATLAB.

As in the previous section, the dataset was divided into sections (this is done to diminish the amount of training networks), although ideally exactly one model per row should be trained —this is, however, computational inaccessible.

The results of this were not very satisfying, mainly because of the fact that the time complexity is proportional to the number of sections, and the number of sections is itself directly proportional to the accuracy.

In further work, more models could be trained using a GPU in order to improve the accuracy, or through a DNN instead of a single layer MLP.

3 Experiments

3.1 Individual Methods

The first step in our approach was to try as many methods for matrix factorisation and matrix completion as we could. In this step, we worked individually. Amongst the methods we tried, the most notable ones (whose concepts are developed in the previous section) were the following:

- Gaussian averaging
- SVD, Incremental SVD, SVT

- KNN imputation
- Nuclear norm basis pursuit (NNBP)
- Bayesian probabilisitic matrix factorisation
- User clustering
- Linear regression
- Radial basis function, MLP neural networks

After we came up with the best methods for matrix factorisation, we applied them. We found that KNN imputation and NNBP gave the best prediction results. We pushed the limit of these methods by fine-tuning their respective parameters. For the NNPB, we modified the smoothing parameter (μ) and the relaxation parameter (ϵ). For the KNN imputation, we mainly changed the number of means (k).

The result of this initial experiment brought us to the 9th place on Kaggle. After that, we struggled to get any higher using only individual methods. We thus concluded that a different approach needed to be followed, which took us to the next experiment.

3.2 Method Averaging

We found by empirically averaging the predictions of different methods that their combinations improved the total performance. We thus applied a boosting technique to find the best combination of weights to associate with each individual estimator. Considering boosting requires a way to calculate the error produced by the respective estimators, we took out some extra values from the original matrix in order to compare them with the predictions of each method. As a result, KNN and NNBP were the two methods that got the highest weights, and thus brought the biggest contribution to the resulting estimation.

We then pushed the general idea a little further by associating a different set of weights to each user, for higher flexibility and adaptation to the user's specificities.

This modification pushed our raking on Kaggle up from the 9th to the 4th place. Any further changes did not yield in any improvement, meaning that we reached the limit of this second experiment.

3.3 Normalisation

Early data analysis showed us that while the admissible values were between -10 and 10, most users did not actually reach these extremes in their ratings: some rate everything between smaller margins (for instance -5 and 5), others make a good audience and only put positive ratings, etc. We thus start by normalising the original values in each row with regards to the users habits. The missing values are then predicted through any of the above methods, and the result is finally de-normalised to match the users specificities again.

The normalisation process starts by calculating the ratio of the maximum possible range ([-10;10]) against the range that the user is actually using. This can be done via the following code:

```
for i=1:nbUsers
    Mx(i) = max(m(i,:));
    Mn(i) = min(m(i,:));
    Ratio(i) = (10-(-10)) / (Mx(i) - Mn(i));;
end
```

This ratio is used to scale the user's values up, in order for all of them to reach the same amount of variation in their ratings. In the same way, the values are shifted using the actual average in the user's ratings to match the center in the modified matrix.

```
for i=1:nbUsers
    for j=1:nbJokes
        % Rescale and shift the user's values
        m(i,j)= m(i,j)*Ratio(i) ...
                - (Mx(i)+Mn(i))/2;
    end
end
```

A special case must however be taken into account: users that are identified as having a variation of zero, and thus as putting the same rating for every joke, are considered irrelevant in the calculation. As soon as these users are detected, every missing value in their rows is replaced by the unique value they have used for all the jokes, and they are disregarded in the rest of the operations.

This algorithm thus takes into account different types irrational users. It resulted in bringing our ranking on Kaggle from the 4th to the 1st place.

4 Results

Figure 4.1 shows the diagram of our working process.

Before any work could be done, we had to familiarise ourselves with the topic by reading related articles and publications. Almost at the same time we started to analyse our dataset to determine its features and points of interest. The goal was to find any regularities that might be useful for prediction. The next step was to work in parallel by implementing and analysing different techniques and approaches. When we finally got an idea of which methods worked and which ones did not, we

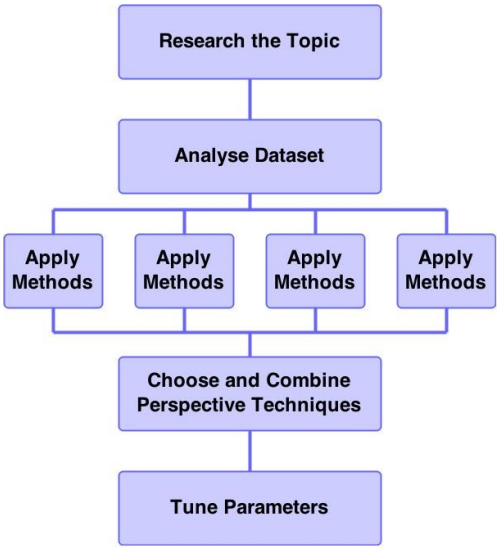

Fig. 4.1. Working methodology diagram

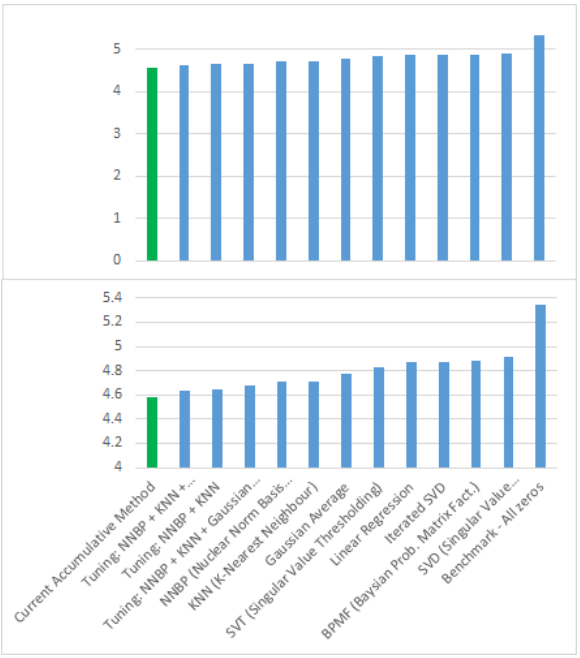

Fig. 4.2. Performance of our methods

began to assemble a combined technique and tune its parameters to maximise performance.

TABLE 4.1
Performance of used methods, the best one being at the top.

Method	RMSE Error
Current accumulative method	**4.58077**
Tuning of NNBP + KNN + Normalisation	4.63962
Tuning of NNBP + KNN	4.64397
Tuning of NNBP + KNN + Gaussian Avg.	4.67299
NNBP (Nuclear Norm Basis Pursuit)	4.70701
KNN (K-Nearest Neighbour)	4.71361
Gaussian Averaging	4.77293
SVT (Singular Value Thresholding)	4.83009
Linear Regression	4.87069
Iterated SVD	4.87156
BPMF (Baysian Prob. Matrix Fact.)	4.88315
SVD (Singular Value Decomposition)	4.91119
Benchmark - All zeros	**5.34441**

Table 4.1 summarises our results with the different methods and their combinations. The performance measure is the root mean squared error on the Kaggle test set.

Figure 4.2 gives a better visualisation of relative performance. The upper part shows how far are all of the results from our target, and the bottom part shows relative comparison within a closer scale. While having a RMSE of 4.5 where random filling gives only 5.3 can be considered a disappointing result, we believe this limited performance to be due to the nature of the dataset. We can observe that a simple KNN imputation was able to give a result that outperformed several state of the art methods on this given problem, and more successful methods actually relied on the results of that KNN.

As this task is a running Kaggle competition, our team participated to see how well our final method performed compared to other participants. Out of 63 teams and around 1000 total submissions, our method currently holds the best result. Table 4.2 displays the results of our submissions, and figure 4.3 shows our position with respect to ten best teams in the competition.

TABLE 4.2
Team ranking and performance on Kaggle competition.
Kremlin is our team.

Rank	Method	RMSE Error
1st	**Kremlin**	**4.58077**
2nd	Jules	4.64266
3rd	Igii	4.64298
4th	Unknown	4.64339
5th	Darshan Singh	4.65100

Such density of results both between the techniques we tried and other team's performance in the competition, along with a large gap between the target and best prediction indicates the properties of the dataset itself.

We believe it to be very noisy and to possess a very low correlation between features (jokes).

Another problem is heterogenity of users. Some of them used the whole [-10;10] margin, some used only a fraction of it ; for some users we had all

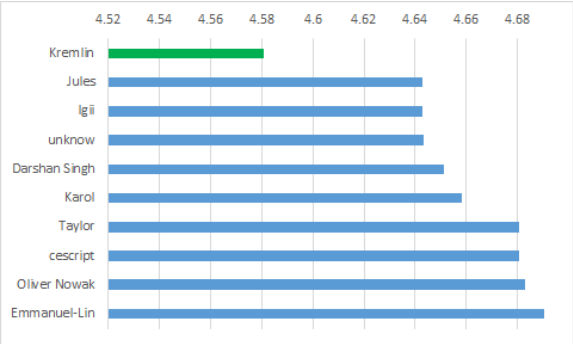

Fig. 4.3. Leaderboard from the Kaggle Website (relative scale)

the features and for others only a small subset. We also found users that were behaving irrationally by giving same ratings to every joke. All that combined resulted in a very hard to predict dataset (we estimate the best possible RMSE to be around 4).

The competition is still running, which enables our team to submit the predictions, view the result and compare it with the results of the other teams. At the time of report submission, our team ranked the 1st with safe margin from the 2nd.

5 Conclusion

The task of dealing with missing data is a classical machine learning problem. The main goal for our team in this work was to learn how to address it. Upon completing this project, we drew several conclusions.

1) None of the methods, no matter how novel or claimed to be powerful, provided good results on their own. Only by composing techniques with boosting and directly addressing dataset issues were we able to achieve high performance.
2) For this particular task, a lot of efforts in implementation of various techniques yielded small overall changes to the results. This made properties other than accuracy (speed, simplicity of the model, generalisation capabilities) play a greater role.
3) Solving such task is a multistep process, where each of the steps is equally important. This means that it is impossible to get a good result by just applying a powerful machine learning technique to the given dataset.
4) Any information about the dataset and how it was obtained can be used to boost performance. Examples are our normalisation/denormalisation process for users who used a fraction of [-10;10] scale, as well as dealing manually with irrational ratings. Moreover, any patterns in how the dataset was assembled could lead to patterns within the dataset that need to be addressed.

At the end, the main achievement of our work is the experience with a real world machine learning problem. Unlike benchmark tasks that we dealt with during our classes, this one came with its own interesting characteristics and challenges. Moreover, this problem being a running Kaggle competition provided us with additional interest and motivation.

References

[1] Kaggle.com, 'Collaborative Filtering — Kaggle in Class', 2015. [Online]. Available: http://inclass.kaggle.com/c/uci77B/. [Accessed: 06- Apr- 2015].
[2] E. Candès and B. Recht, 'Exact matrix completion via convex optimization', *Commun. ACM*, vol. 55, no. 6, p. 111, 2012.
[3] S. Becker, E. Candès and M. Grant, 'Templates for convex cone problems with applications to sparse signal recovery', *Mathematical Programming Computation*, vol. 3, no. 3, pp. 165-218, 2011.
[4] CVX Research, 'Matrix Completion — CVX Research, Inc.', 2012. [Online]. Available: http://cvxr.com/tfocs/demos/matrixcompletion/. [Accessed: 06- Apr- 2015].
[5] R. Salakhutdinov and A. Mnih, 'Bayesian probabilistic matrix factorization using Markov chain Monte Carlo', *Proceedings of ICML*, 2008.
[6] Toronto University, 'Bayesian Probabilistic Matrix Factorization', 2015. [Online]. Available: http://www.cs.toronto.edu/~rsalakhu/BPMF.html. [Accessed: 06- Apr- 2015].
[7] Z. Lin, M. Chen, L. Wu, and Y. Ma, 'The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices', *UIUC Technical Report* vol. 9, pp. 2215, November 2009.
[8] University of Illinois, 'Augmented Lagrange Multiplier (ALM) Method', 2015. [Online]. Available: http://perception.csl.illinois.edu/matrix-rank/sample_code.html. [Accessed: 15- Mar- 2015].
[9] Sifter.org, 'Netflix Update: Try This at Home', 2015. [Online]. Available: http://sifter.org/~simon/journal/20061211.html. [Accessed: 01- Apr- 2015].
[10] JF. Cai, E. Candès and Z. Shen, 'A singular value thresholding algorithm for matrix completion', *SIAM Journal on Optimization*, 2010.
[11] E. Candès and B. Recht, 'Exact matrix completion via convex optimization', *Commun. ACM*, vol. 55, no. 6, p. 111, 2012.
[12] L. Wang, J. Hu and C. Chen, 'On Accelerated Singular Value Thresholding Algorithm for Matrix Completion', *AM*, vol. 05, no. 21, pp. 3445-3451, 2014.
[13] M. Kurucz, A. Benczúr and K. Csalogány, 'Methods for large scale SVD with missing values.' *Proceedings of KDD Cup and Workshop*, vol. 12. 2007.

APPENDIX

Contributions

Nathalie

Nathalie analysed and applied the following techniques: completion by averaging, singular value decomposition and other matrix factorisation methods, its iterated version, and singular value thresholding. She took part in the process of tuning parameters of the different methods, normalisation of the data, treatment of irrational users, result analysis, and boosting.

Alexey

Alexey analysed and applied several forms of KNN imputation, augmented Lagrange multipliers (3 versions), accelerated proximal gradient, alternating direction method, subspace evolution and transfer algorithms, took part in normalising the data, tried several hypotheses related to correlation between subsequent ratings, participated in results analysis.

Enrique

Enrique performed initial data analysis, and applied the following methods: completion by Linear Regression, two RBF models, and MLP neural networks.

Noureldien

Noureldien took part in data analysis: user clustering and feature selection by correlation. He applied the following techniques: nuclear norm pursuit, Bayesian probabilistic matrix factorisation and averaging by boosting. He also tested methods using LMaFit library.