Project II Moscow

Wissam Allouchi **Abdelkoddous Badou**

Farah Bouassida
{wissam.allouchi,abdelkoddous.badou,farah.bouassida} @epfl.ch

Abstract

The second project of PCML course consists on applying machine learning techniques to two real-world problems. In this report, we describe the work we've done in order to tackle this problems. The first one is a prediction problem where we have to build a model able to produce accurate count prediction for given (user,artist) pair. The second is a classification problem where we also have to train a model that is able to recognize whether a person is present on an image or not.

1 Music Recommendation System

1.1 Introdution

To build this music recommendation system, we are given listening history of users for some musicians. Specifically, we know for each user a listening count for some of the artists that he listened to. By intuition, we know that each user cannot listen to all the artists which is an important information to take into account when training data i.e. a zero count does not mean anything. This will allow us to predict listening count of existing users (Weak generalization), for example artists that they didn't listened to, but this is not sufficient to predict counts for unknown users (Strong generalization). For this problem, we are also given social networking information about users so we can know who is friends with who.

1.2 Data exploratory and visualization

The provided dataset consists of a total of 15082 artists and 1774 users while we only have 60617 user-artist relations which is approximately 0.2% of the total number of user-artist pairs. Therefore, as we have noticed before a majority of potential counts are missing which is challenging since the given matrices are sparse. Then, some of the ML methods such as SVD are not applicable in our case. This is also a common issue in collaborative filtering that is known to use the aggregated behavior/taste of a large number of users to suggest relevant items to specific users. In order to explore this dataset, we started by investigating its distribution. We tested several transformations which allowed us to notice that the data has a log-normal distribution. Therefore, applying a simple logarithm (base 10) transformation would approximately give a normal distribution. This transformation is applied only to non zero values since logarithm of zero tends to infinity.

The histogram of the listening counts depicted in (*figure 1*) shows that a few artists have a huge count (of order 10^6) while the majority of artists have a very small count (between [0,10000]) compared to the maximum value.This already suggests that there will be some issues with predicting the high counts since they will act as outliers as they represent a small proportion of the data. We may need to do specific treatment to these 'outliers'.

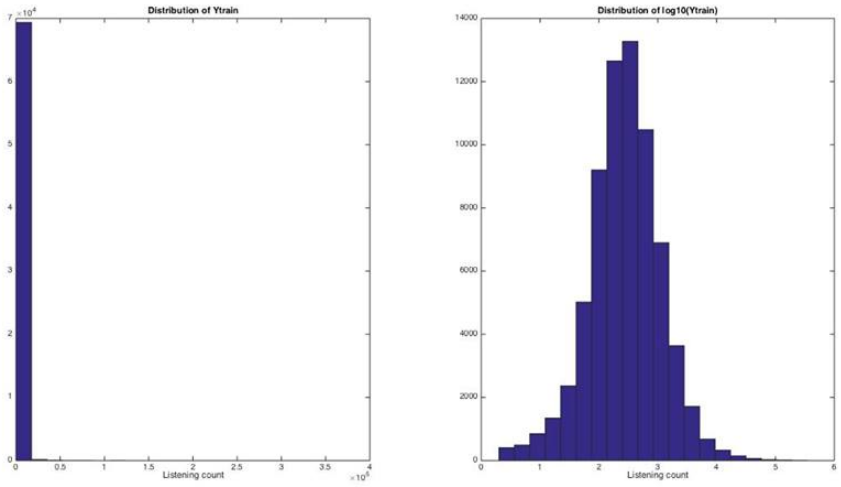


Figure 1: Data distribution before and after log transformation

1.3 Learning

1.3.1 Problem formulation

We are given the matrix Ytrain = $\{y_{ij}\}$ of size DxN where D is the number of users, N is the number of artists and y_{ij} is either integer number or missing (0 count). Our task is to estimate some of the missing values in the matrix Ytrain based on the known values, and the challenge is to be as accurate as possible. This translates into minimizing the Mean square error (MSE) of the model given by $L^2(y - y^m_{pred})$ where y^m_{pred} is the prediction given by model m.

1.3.2 Weak generalization

To tackle this problem, we first applied our baseline method consisting of simply assigning for each (user,artist) pair the average listening count for that artist : y^{ij}_{pred} = mean(y_{ij}) for i=1..D. Using this approach we estimate that the test error would be of order 0.83. Now, we try to improve our previous results using a more complex method. We decided to try a Low rank approximation of the user-artist count matrices. This allows us to model both users and musicians by expressing them in a lower dimensional feature space. We followed the model described by NETFLIX paper [6] namely Alternative Least Squares with Weighted-λ-Regularization (ALS-WR) algorithm. In order to apply this algorithm, we need to preprocess our data to make it as Gaussian as possible. Another issue arises when we split our data into training and test sets. We observe that some of artists in the training set does not have any listening count. Therefore, we make sure that we remove these artists from the matrix we feed to ALS algorithm to avoid computational issues. Furthermore, we also noticed that when we normalize the data, the prediction on the test set becomes extremely bad. This is why we decided to drop this step from the data preprocessing steps. The only tunable parameters of ALS-WR algorithm are the regularization term λ and the number of hidden variables N_f. We fix them using a cross validation procedure, where we choose λ^* and N^*_f to be the parameters that minimize the mean test error. Note that in our case, since the RMSE does not give a good insight of the performance of our prediction, we rather use the following error measure :
ϵ = (1/N)$\sum y_{ij} \in testset \parallel \log(y_{ij}) - \log(y_{pred}) \parallel$.
This formula capture how close is our prediction to the actual count.

The (*figure 2*) illustrate a strange behaviour : for a fixed λ, we observe that the error increases monotonically with the number of features N_f. This behavior can be explained by the fact that the low rank approximation remove the unnecessary components of our 'signal' which translates by attenuating the noise. So, when we increase the number of hidden variables, we are adding more and more noise. Therefore, the error should increase as well. In the (*figure 3*), we plot (on the

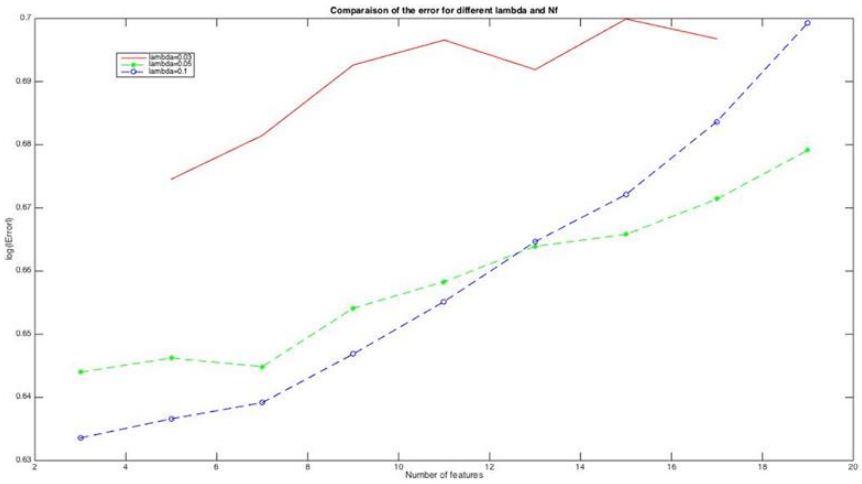


Figure 2: Error with regard to λ

left) the prediction for a random data test set versus the actual counts. On the right, we have the histogram of capturing the distribution of the error that our algorithm makes. From the scatter plot, we can identify almost three clusters of points. The first one can be represented by the points that are around the axis y = x. For this points, the predicted count is close to the actual count therefore, the ALS-WR algorithm have a good performance. The second cluster can be represented by points that are above the axis y = x and in this case, the ALS-WR algorithm predicts much higher counts compared to the actual ones. Finally, the last cluster can be represented by points that are fare away bellow y = x which corresponds to extremely large listening counts for which ALS-WR algorithm is unable to predict good estimation. This suggests that we can build three different models to fit

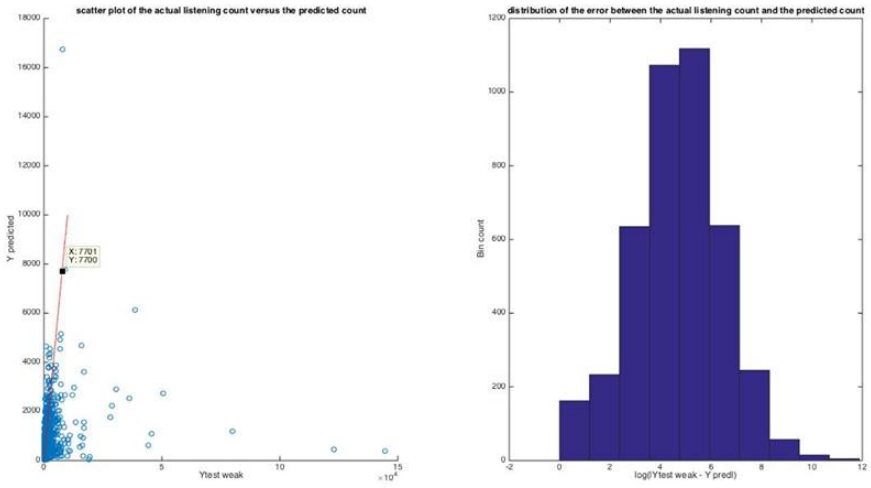


Figure 3: Scatter-error distribution

our data. One model to fit average listening counts, an other one to fit extremely large counts and a third one to fit very low counts. To do so, we rely on Expectation-Maximization algorithm to cluster the (user,artist) pairs into these three categories and given this categorization, use the appropriate ALS-WR model to predict. Unfortunately, due to lack of time and other implementation troubles, we were not able to test and fully implement this method. As we can see in the (*figure 4*), we

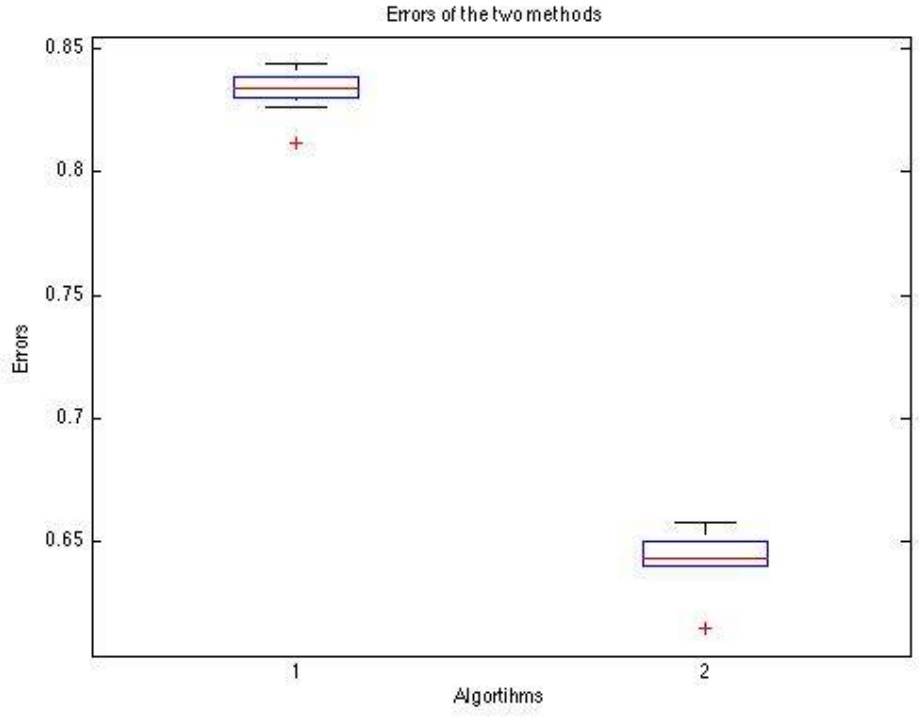


Figure 4: Weak error boxplot

make a comparison between the ALS-WR approach and the baseline approach. This figure was generated by running a 10-fold cross validation procedure. Therefore, ALS-WR outperforms the naive approach with an error of 0.6309 for $\lambda = 0.1$ and $N_f = 3$.

1.3.3 Strong generalization

Now we try to predict listening counts for users we did not train our model on. This is also known as the Cold start problem. Our baseline approach is to assign to each user-artist pair the average listening counts for that artist. $y_{pred}^{new,j}$ = mean(y_{ij}) for all users i in the training data set. This approach give a mean log error estimate of 1.93 using 10-fold cross validation. Next we try a median approach i.e. $y_{pred}^{new,j}$ = median(y_{ij}) for all users i in the training set. This improves our error estimate by 9% (the new error estimate is now 1.75). Now we will make use of the friendship information that we have. We first assign for a new user the average listening count of all his friends for a given artist i.e. $y_{pred}^{new,j}$ = mean(y_{ij}) for all i in the friendship network of the new user. This approach does not improve our test error estimate, but rather make it worse (error estimate is 3.75). We also try a friendship median approach where we assign for a new user the median count of all his friends but it does not improve our error estimate (3.7) We summarize the results in (*figure 5*). Note that we compute the mean and median only on the non zero-values, otherwise our error estimation is very large.

1.4 Final model

In this section, we describe the model we've chosen to solve the two problems presented above. For the weak generalization, we achieved the best performance using ALS-WR algorithm with parameters λ=0.1 and N_f = 3 where λ is the regularization parameter and N_f is the number of hidden variables. This was obtained using k-fold cross-validation technique as explained above. For the second problem, namely strong generalization, we obtained the best results using the median ap-

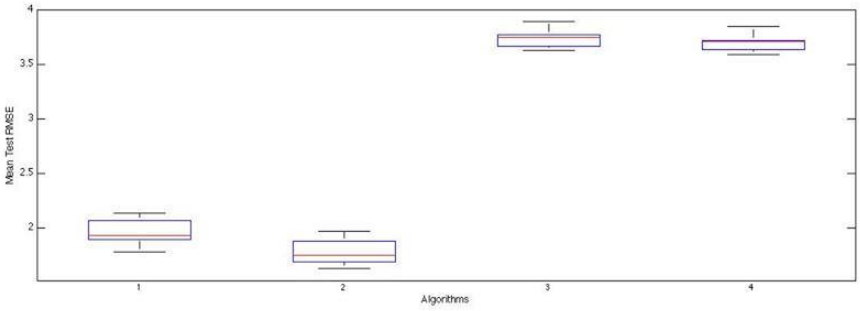


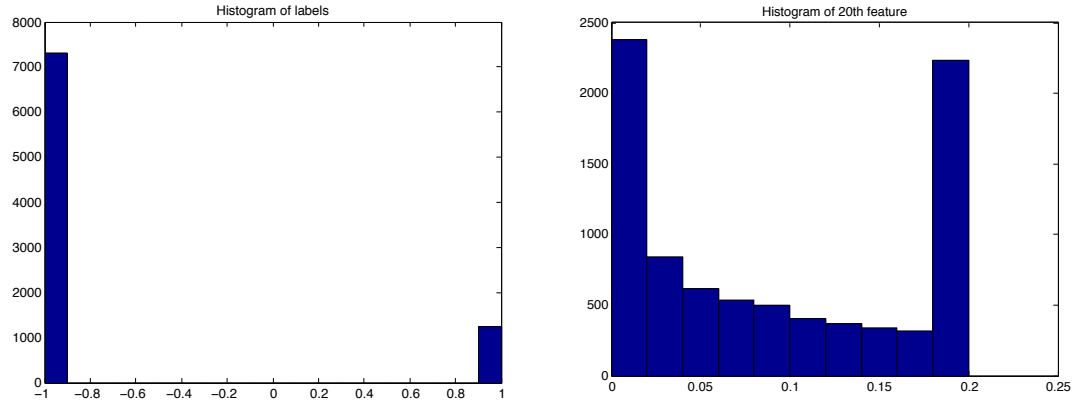
Figure 5: Box plot of the error estimation. This plot was generated by running a 10-fold cross validation.

proach without involving the friendship information. This model will be used to produce the final counts predictions for the given test data set.

2 Person Detection

2.1 Data Description

This problem consists of classifying an image in a positive one thus contains a person or as a negative image in the contrary case. Our dataset consists of 8545 images and 9360 corresponding observation for each. The images are well centered and form the same size. The related features corresponds to the histogram of oriented gradients (HOG) of the image. After plotting some samples of HOG images, we can guess that the features try to find the edges on a photo to determine wether there is a person's shape or not. We plotted the histogram of the labels(outputs) (figure 6a) and we can see that there is much more negative labels than positive ones. This can lead to a good prediction of positive images and a bad prediction of negative ones which can be already observed with the random predictor in the figure. This is one reason why the average of true positive rate (TPR) will be considered as a good method to evaluate the solution mode for this binary classification. When plotting features' histograms (figure 6b), we see that they have two peaks at the max value and the zero one.



(a) Histogram of images' labels

(b) Histogram of 20^{th} feature

Figure 6: Data visualisation

2.2 Neural Network training

Having a high dimensional problem, we try as a first approach to train a $neuralnetwork^{[1]}$(NN) on our data starting with baseline parameters, a 2 layers NN with linear classification, *tanh* activation function and 10 hidden variables. We apply to all our NN algorithms a 4-fold Cross Validation to avoid overfitting. The baseline NN give us a average TPR of 0.73 . To improve the results and base on the fact that we have a large set of inputs, we tried a larger number of hidden in variables from 100 and 200. We get better results with an average TPR between 0.739 and 0.799, the maximum is reached for 170 hidden variables. In comparison withe the hyperbolic tangent, the sigmoid activation function changes the output for a small change in the weights [2] what helps to determine if the change is good for the prediction. Thus, we try a 2-layers NN with a sigmoid activation function and a range between 100 and 200 hidden variables given us a an average TPR of 0.862 for 150 hidden vaiables. A 3-layers NN gives as best result an average TPR of 0.845 for 130 hidden variables. As expected, 2 layers are sufficient to have a good model, the 3^{rd} layer only reduces by $2x10^{-5}$ the variance of the average TPR among the 4-fold CV.

2.3 Random Forest training

As second thought, to deal with the high dimensionality curse and knowing that the dimensions represents gradients possibly very correlated, random forest (RF) algorithm could help thanks to its randomness for choosing features and the bootstrapping sampling that allows replacement in the features samples. We use the *random forest* of pitor's toolbox with "*datasample*" to get sampling

with replacement. A baseline version with 1000 features per tree and a forest of 200 gives an average TPR of 0.28. Training different models with different numbers of features in a equally distanced range from 100 to 1500 and a number of trees varying in an equally distanced range from 10 to 100 give us a best average TPR of 0.4167 with 1200 features and 40 trees. Having very poor results, we tried to train the forest on a less unbalanced dataset. We reduced the number of negative images to the half and to the 3/4 th and trained again the RF [3]. The set being much less larger the results are very biased and yields very low averageTPR. Another solution, would be to ameliorate the sampling to ensure that the sample of features is representative of the data set (otherwise it's very likely that we have no positive images sample for a tree). We processed to a subsampling with replacement within the sampling of the tree but the results does't improve.

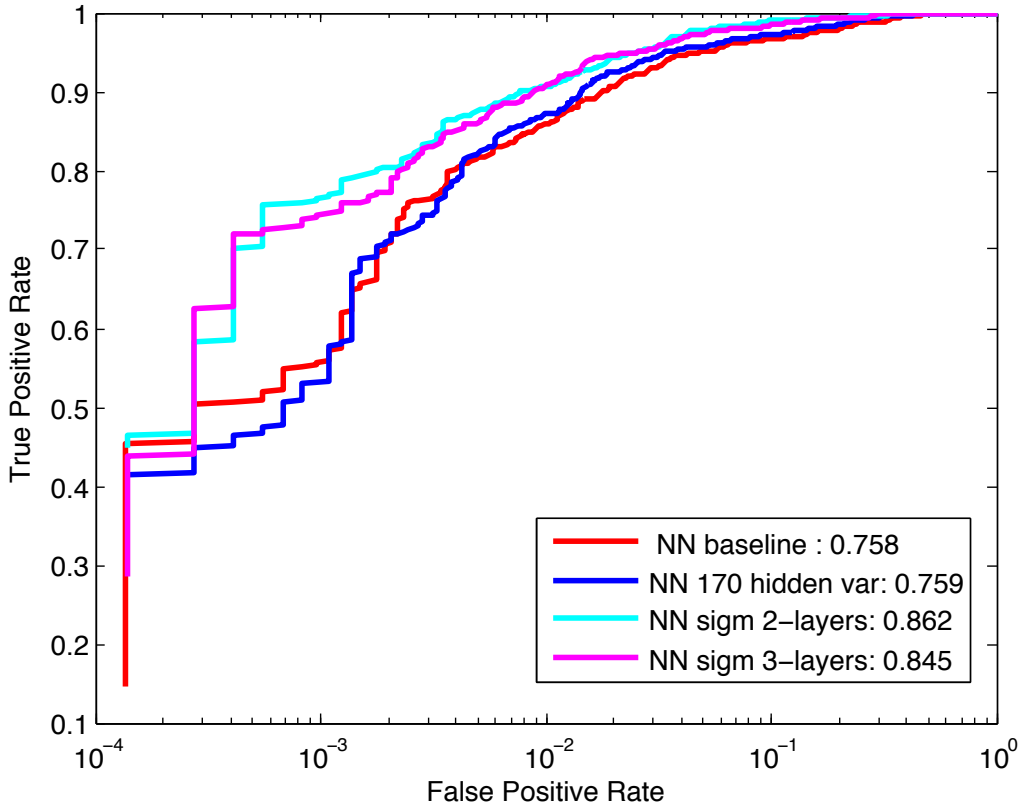


Figure 7: Neural Network Optimization

2.4 SVM training

Using the libsvm toolbox [4], to see if a right kernel function we could overcome course of dimensionality and data unbalance probabilities although the svm isn't expected to perform well with the latter problem . The baseline algorithm with a linear kernel give us very bad performance as expected. We tried an rbf kernel with a range of gammas and costs and we have a best average TPR of 0.585 for a gamma equals 0.4 and a cost of 8.

2.5 Conclusion

With our dataset, neural network behaves the best. We will use it for our predictions with 2 layers and 150 hidden variables.

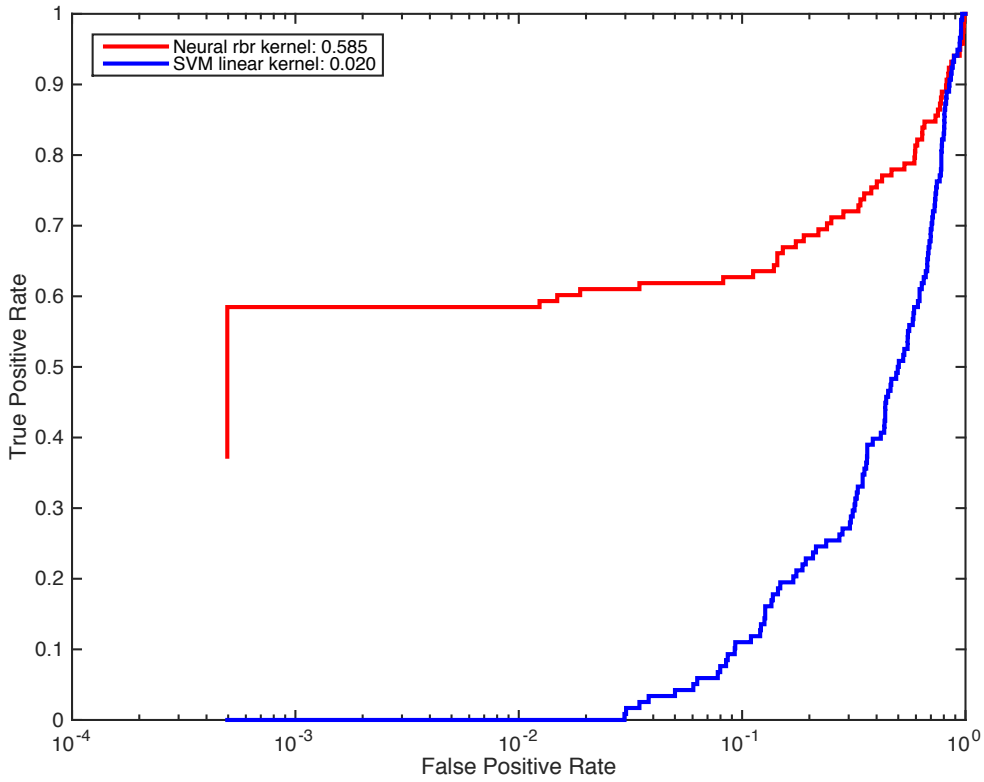


Figure 8: SVM Optimization

References

[1]Prediction as a candidate for learning deep hierarchical models of data](http://www2.imm.dtu.dk/pubdb/views/publication _details.php?id=6284)

[2] http://www.faqs.org/faqs/ai-faq/neural-nets/

[3] http://stackoverflow.com/questions/8704681/random-forest-with-classes-that-are-very-unbalanced

[4] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[5] Using Random Forest to Learn Imbalanced Data http://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf

[6] Alternative Least Squares with Weighted-λ-Regularization (http://www.hpl.hp.com/personal/Robert$_Schreiber/papers/2008\%20AAIM\%20Netflix/netflix_aim08(submitted).pdf$