Ming Hsieh Department
of Electrical Engineering

EE559
Mathematical Pattern Recognition
Course Project Assignment

Pattern Recognition Techniques for Credit Classification

Nikolaos Flemotomos
flemotom@usc.edu
USC I.D.: 7149389176

April 30, 2017

Abstract

In this project I implement and compare different classifiers for the `German Credit Risk` dataset, composed of 9 features and 1000 samples. This is a highly challenging dataset, due to a variety of problems that occur in practice in Machine Learning, such as unbalanced classes, missing data, a mixture of categorical and numerical features, etc. Three different classifiers are compared; namely a Support Vector Machine (SVM) with Gaussian kernel, a k-Nearest Neighbours (k-NN) -based classifier, and a Parzen Windows (PW) -based classifier. A simple classifier, based only on the majority class, is also given for reference. The SVM, the k-NN, and the PW resulted respectively to an F1-score of 0.7432, 0.7390 and 0.7427 for the majority class and 0.2692, 0.2667 and 0.1505 for the minority class. Although the differences are not significant and the results differ at each run, it seems that the statistical classifier give on average a bit better results than the SVM.

1 Introduction

This report describes the preprocessing of the `German Credit Risk` dataset (as provided in the file `Proj_dataset_1.csv`) and the implementation of three different classifiers of the 2-class problem that arises. The (already preprocessed) dataset is composed of 9 features and 1000 samples. The objective of the credit analysis is to find whether an individual applying for a loan is a good credit risk or not, something that can eventually lead to the minimization of risk from a bank's perspective.

To confront this problem, after suitably preprocessing the data, 3 main classifiers were built. One of them, a Support Vector Machine (SVM) lies in the are of distribution-free Pattern Recognition, while the other two, a k-Nearest Neighbors (k-NN) classifier and a Parzen Windows (PW) classifier are traditional examples of the so-called statistical Pattern Recognition.

In the following sections, I describe in detail the preprocessing of the dataset, the steps taken for the estimation of the optimal parameters for each algorithm, and the procedure followed for the evaluation of their performance. The implementation was done in MATLAB R2016b. The source code is provided in a separate file under the name `Flemotomos_Nikolaos_project_code.pdf`.

2 Preprocessing

The first thing done after loading the dataset is to split it into a training set and a test set, using an 80%-20% split. The test set will be used only for the final evaluation and is always treated with respect to the training set.

Since there are categorical variables, an appropriate encoding into numerical values is of great importance, since all the algorithms that will be used treat the samples as points in a feature space equipped by a specific distance metric (which is always the Euclidean distance for this project). In particular, the first column of the dataset is of course discarded since it just consists of the indices and is, thus, completely uninformative. The `Age`, `Credit amount` and `duration` are left unchanged, since they are valid ordered numerical variables. The `Job` also stays unchanged. It is in fact a qualitative variable, according to the documentation of the original dataset, already encoded for us. This encoding is completely valid, since there is a natural ordering between the unemployed, the unskilled employees, the skilled employees, etc. As for the `Housing`, even though there might be an ordering in terms of the credit risk, I choose not to take an arbitrary and possibly wrong decision, and thus I encode it using 2 dummy variables, according to

the one-to-one mapping

$$\begin{aligned} \text{own} &\leftrightarrow 1, 0 \\ \text{rent} &\leftrightarrow 0, 1 \\ \text{free} &\leftrightarrow 0, 0 \end{aligned}$$

Similarly, the `Purpose` is encoded with 7 dummy variables, according to the one-to-one mapping

$$\begin{aligned} \text{car} &\leftrightarrow 1, 0, 0, 0, 0, 0, 0 \\ \text{furniture/equipment} &\leftrightarrow 0, 1, 0, 0, 0, 0, 0 \\ \text{radio/TV} &\leftrightarrow 0, 0, 1, 0, 0, 0, 0 \\ \text{domestic appliances} &\leftrightarrow 0, 0, 0, 1, 0, 0, 0 \\ \text{repairs} &\leftrightarrow 0, 0, 0, 0, 1, 0, 0 \\ \text{education} &\leftrightarrow 0, 0, 0, 0, 0, 1, 0 \\ \text{business} &\leftrightarrow 0, 0, 0, 0, 0, 0, 1 \\ \text{vacation/others} &\leftrightarrow 0, 0, 0, 0, 0, 0, 0 \end{aligned}$$

The `Saving accounts` and the `Checking account`, on the other side, do have an obvious natural ordering, in the sense little $<$ moderate $<$ quite rich $<$ rich. So, these values are encoded as

$$\begin{aligned} \text{little} &\leftrightarrow 1 \\ \text{moderate} &\leftrightarrow 2 \\ \text{quite rich} &\leftrightarrow 3 \\ \text{rich} &\leftrightarrow 4 \end{aligned}$$

Finally, the `Sex` is simply encoded as

$$\begin{aligned} \text{male} &\leftrightarrow 1 \\ \text{female} &\leftrightarrow 0 \end{aligned}$$

Following the above rules, the dimensionality of the data augments from 9 to 16. But before dealing with the dimensionality, special care has to be taken about the missing values.

By inspecting all the variables, there are only two variables with missing values. Since the percentage of the samples with missing values is quite big (379 out of the 1000 samples have one missing value and another 75 have two missing values), discarding those samples (just from the training set of course, since no samples can be discarded from the test set) would be probably a bad idea. Instead, the following heuristic algorithm is used:

- If a sample has a missing values for only one of the two types of accounts, use the other type as a predictor. For example, if an individual has a moderate savings account, assume his checkings account is also moderate.

- If a sample has two missing values, use the nearest neighbor as an estimator. So, the tuple (`Savings accounts`×`Checking account`) is the unknown, with $4^2 = 16$ possible outcomes and it is estimated by the corresponding value of its nearest neighbor, taking into consideration all the "non-account" features. For this procedure, the labels are not taken into consideration (since for example we assume we don't know the labels of the test samples). Additionally, both for the test and the training samples, we compare only with all the known training samples to find the nearest neighbor.

After the above processing, it is guaranteed that there are no missing samples in the dataset. Now, all the features are normalized so that they have mean zero and standard deviation 1, because the performance of many algorithms that depend on the distance between samples (as the ones which are going to be used) often deteriorates due to different or irrelevant scaling of the features. The normalization of the testing set if of course done with respect to the training set.

In order to deal with the possible problem of high dimensionality, Principal Component Analysis (PCA) is applied. MATLAB's built-in function `pca` is appropriately used for that reason. It is noted that the dimensionality is not particularly large for that problem (16 features with 800 samples, considering only the training set), but we should still consider the possibility that some of the features are redundant. Figure 1 shows the relative variance of each principal component.

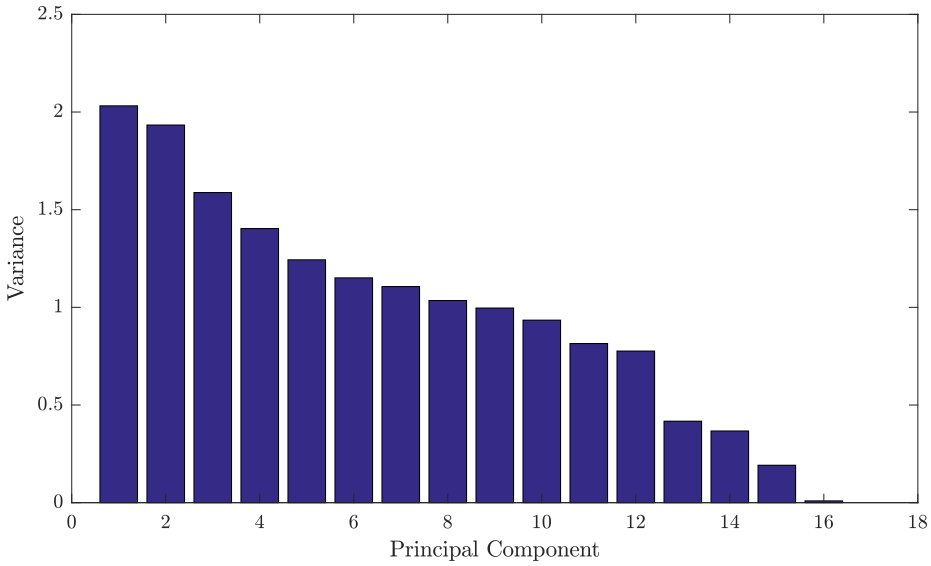

Figure 1: Variance of the principal components of the data.

Based on the particular result, 4 are the most promising options: (a) keep 12 principal components as our final feature set, (b) keep 14 principal components as our final feature set, (c) do not perform PCA and use the whole feature set, (d) perform PCA just to decorrelate the features but not for dimensionality reduction. All those 4 options are explored in Section 3, although not a rigorous cross-validation scheme is performed for that reason.

3 Implementation and Parametrization

After the basic preprocessing steps analyzed in the previous section, an SVM, a k-NN classifier and a PW classifier are implemented. Each one of those has specific parameters that have to be optimized. For that reason, 5-fold cross-validation is performed. Specifically, only the training set is being used and for each set of parameters - as they are pre-specified - a 5-fold cross-validation is performed. The set of parameters that yields the minimum weighted average F_1-score is considered to be "optimal" for our purposes. The weighted average F_1-score is defined as

$$\bar{F}_1 = \frac{\sum_{i=1}^{K} n_i F_1^{(i)}}{n} \tag{1}$$

where K the number of classes (here equal to 2), n_i the number of samples in the class i, and n the total number of samples. With that alternative definition, we take into consideration both classes, but we give more importance to the majority class.

We note that this cross-validation concerns only the parameters of the classifier and is not been done for the estimation of the final performance of the classifier. The missing data have already been estimated in the training set. Additionally, this procedure is repeated 4 times, one for each feature set as described in the previous Section.

3.1 Support Vector Machine

The first classifier implemented is an SVM with Gaussian kernel. The implementation in based on LIBSVM for MATLAB and especially the functions `svmtrain` and `svmpredict`. The two crucial parameters here are the slack variable C and the kernel's variance parameter γ. So, a grid search is performed for 20 values of C logarithmically spaced in the range $[10^2, 10^2]$ and 20 values of γ logarithmically spaced in the range $[10^{-2}, 10^1]$, in a 5-fold cross-validation, as already explained. Those ranges are chosen after a more coarse grid search in a wider range of values. The results, in terms of the weighted F_1-score and the accuracy, are graphically listed in Figure 2. As seen, the accuracy seems to be a much smoother metric. However, as it will be explained in more detail in section 4, it wouldn't be wise to choose it as the most important metric for evaluation of performance. At any case, there is a definite pattern for both metrics with respect to the parameters chosen.

Relying on the weighted F_1-score, the best set of parameters is chosen to be $C = 100.00$, $\gamma = 0.705$ after keeping the 12 first Principal Components. It should be underlined that (a) the results change slightly for each run of the program, (b) should we rely on a different metric for the optimal parameters, the final choice would be different. The choice of the weighted mean F_1-score is somehow a compromise between the accuracy (which favors the majority class) and the mean F_1-score, which treats the two classes exactly the same, no matter how large the imbalance is.

It is noted that the reason that some cells of the grid seem to have no value is because for the specific set of parameters the algorithm fail to assign any sample to the classe 2, so the sum of True Positives and False Positives for the specific class is 0; thus the F_1 score is not defined. The same argument goes for later graphs of the report, as well.

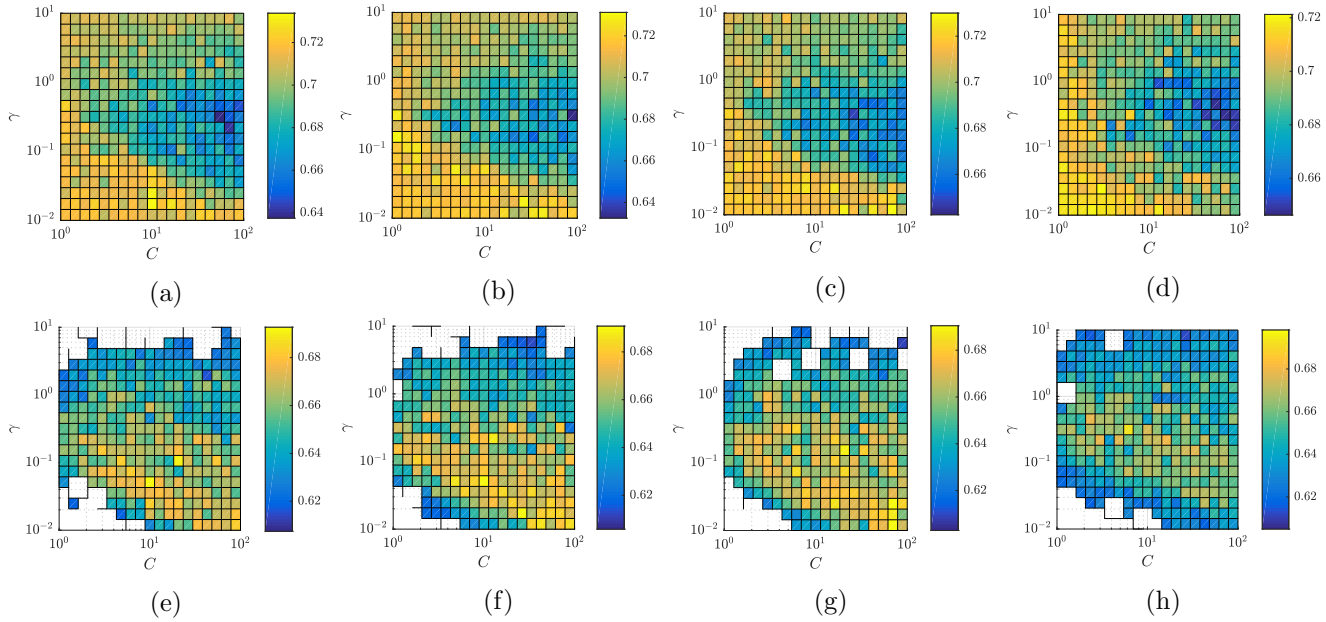

Figure 2: Accuracy (first row) and weighted mean F_1-scores (second row) for different parameterizations of an SVM with Gaussian kernel (C and γ) after a 5-fold cross-validation. 1st column: no PCA, 2nd column: 16 (all) PCs kept, 3rd column: 14 first PCs kept, 4th column: 12 first PCs kept.

3.2 k-Nearest Neighbors

The k-NN classifier is implemented with the built-in MATLAB function `fitcknn`, using the frequency of the appearance of each class as the prior probabilities. The crucial parameter for k-NN is the number of neighbors k. The results, in terms of the weighted F_1-score and the accuracy are graphically listed in Figure 2, when analyzing values of k equal to $\{1, 2, \cdots, 50\}$. Relying on the weighted F_1-score, the optimal value of k is chosen to be $k = 5$ after keeping the 12 first Principal Components. It is worth noting that the accuracy seems to approximate a stable point as the number of neighbors gets larger, while the weighted mean F_1 score has a downward trend. This gives a hint on why we shouldn't rely on accuracy a lot. It is obvious that as k gets larger and larger, each sample tends to be assigned to the majority class, which yields to an essentially useless model in the general case. This is why the accuracy doesn't change after some point, but the F_1 score, which takes under consideration both classes, keeps decreasing.

3.3 Parzen Windows

The PW classifier is implemented with the function `Parzen` from the `Classification Toolbox` for MATLAB. The crucial parameter for PW is the normalizing factor h. The results, in terms of the weighted F_1-score and the accuracy are graphically listed in Figure 2, when analyzing values of h equal to $\{1, 2, \cdots, 50\}$. Relying on the weighted F_1-score, the optimal value of h is chosen to be $h = 15$ without applying PCA. Similar notes made for the k-NN case for the comparison between the accuracy and the F_1 score apply here, too, even though it was maybe more clear with the k-NN classifier.

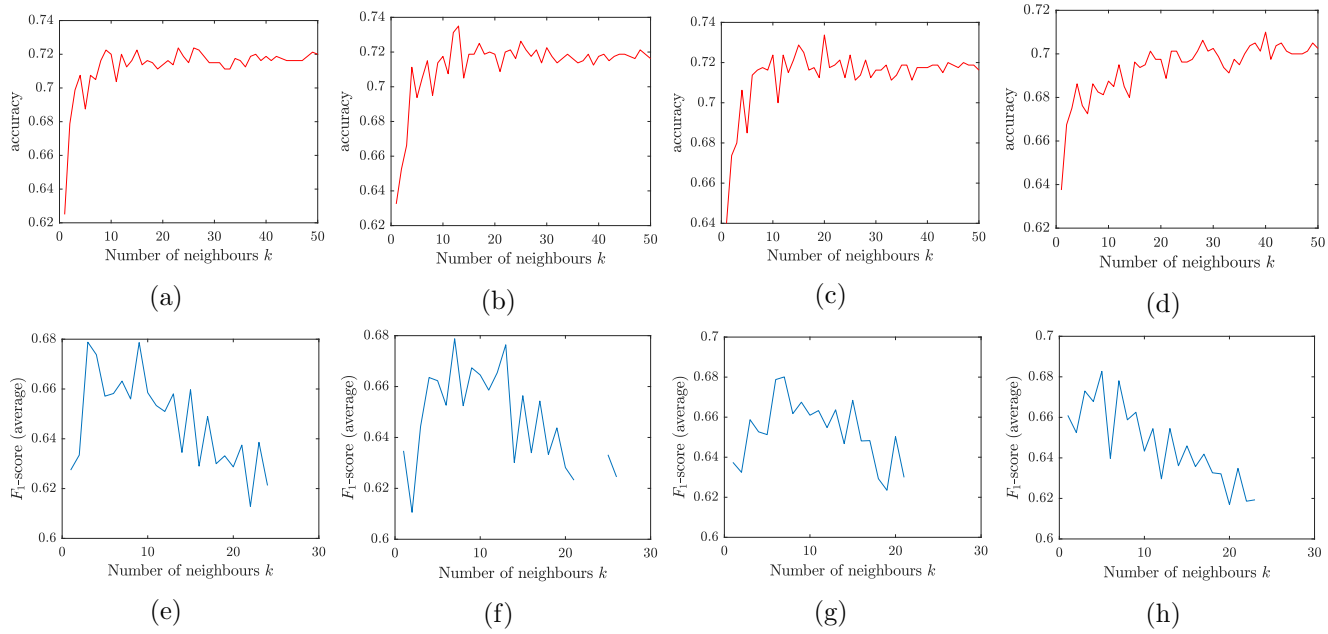

Figure 3: Accuracy (first row) and weighted mean F_1-scores (second row) for different parameterizations (number of neighbors k) of a k-NN classifier after a 5-fold cross-validation. 1st column: no PCA, 2nd column: 16 (all) PCs kept, 3rd column: 14 PCs kept, 4th column: 12 PCs kept.

4 Results

After having found the optimal feature set and the optimal set of parameters for each algorithm, an "overall" 5-fold cross-validation is applied in order to estimate the performance of the algorithms. For that reason, only the training set is used. For each fold, the missing values of the validation set are filled with respect to the training set (which is different for each fold), as it has been explained in section 2 for the case of the test set. The normalization and the PCA (if applicable) are also done with respect to the training set. The results are given in Table 1. The results of a baseline system who classifies only considering the prior probabilities (and thus always assigns a new sample to the majority class) are also given. Instead of the weighted mean F_1 score, here the F_1 scores of each class are given (together with the unweighted mean).

algorithm	accuracy (%)	$F_1^{(1)}$	$F_1^{(2)}$	mean F_1
Baseline	71.75	0.8352	–	–
SVM	66.50	07792	0.2964	0.5442
k-NN	70.25	0.8092	0.3233	0.5661
PW	70.12	0.8108	0.2846	0.5477

Table 1: Performance metrics for the different classifiers after a 5-fold cross- validation.

Now, the classifiers are ready to be given unknown samples as input. The unknown samples are exactly the samples in the test set. By having set aside the test set from the beginning, we can estimate the

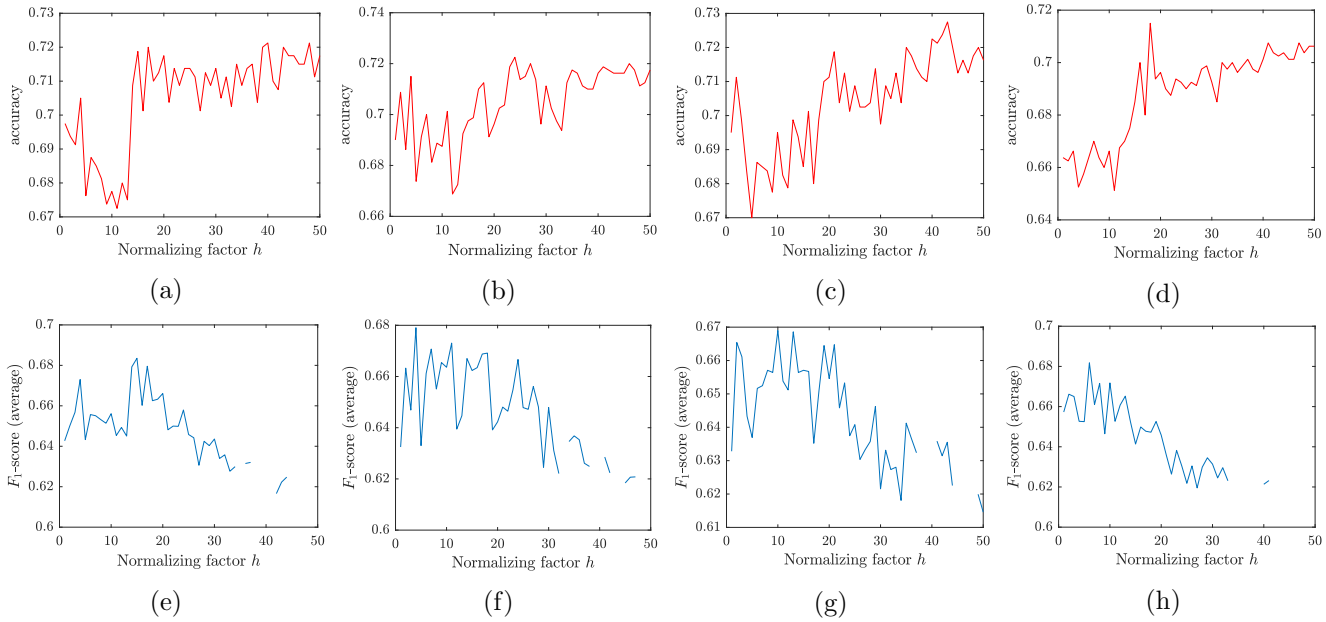

Figure 4: Accuracy (first row) and weighted mean F_1-scores (second row) for different parameterizations (normalizing factor h) of a PW classifier after a 5-fold cross-validation. 1st column: no PCA, 2nd column: 16 (all) PCs kept, 3rd column: 14 PCs kept, 4th column: 12 PCs kept.

performance of the classifiers to real new samples. The results are given in Table 2.

algorithm	accuracy (%)	$F_1^{(1)}$	$F_1^{(2)}$	mean F_1
Baseline	63.00	0.7730	–	–
SVM	62.00	07432	0.2692	0.5062
k-NN	61.50	0.7390	0.2667	0.5029
PW	60.50	0.7427	0.1505	0.4466

Table 2: Performance metrics for the different classifiers after a 5-fold cross- validation.

As seen from both Tables, the classifiers seem to fail to bear the performance of the baseline naive classifier. However, in reality this is not the case. All the classifiers are optimized with respect to the (weighted) mean F_1 score. If they were optimized with respect to accuracy, this metric would be higher compared to the baseline. In fact, when I optimized k-NN with respect to the accuracy, the accuracy achieved on the test set would be as high as $\sim$74% when the corresponding baseline accuracy was $\sim$71% (which is of course still not a huge difference). However, it is important to be able to classify correctly both classes and not only one of them, which the baseline classifier obviously fails to do. In practice, a system that classifies everybody as a good or a bad credit risk would be absolutely useless for a bank. That would mean that the bank would approve or would deny a loan to everyone, which would be of course detrimental.

Another point that should be mentioned is the fact that the results are not very stable between different runs of the program. For example, it can be seen from the big differences between the cross-validation and

the test set performance metrics (which were not always the case) that the initial split into training and test sets was of big importance. This, in combination with the generally low performance of the classifiers, can be attributed to (a) small number of samples, (b) low predictive power of the features, (c) possibly not optimal choice of the feature set of the way the missing values should be treated. However, it should be noted that the relative difference in different classifier's performance is stable, which is an important characteristic.

5 Conclusion

Three different classifiers were implemented and analyzed to attack the problem of credit classification; namely an SVM with Gaussian kernel, a k-NN classifier and a PW classifier. Various techniques related to Pattern Recognition were applied. Extensive research of the optimal parametrization of the classifier and of their performance was made. Even though the classifiers did not give very good classification results for the specific problem at a first glance, they did give useful insight. It was made clear that the accuracy in not always the best metric, especially for problems unbalanced classes. Finally, it was made clear that techniques which are widely used, sometimes without meticulous inspection of their effects, such as PCA, are not always useful. In our case, PCA either led to very small improvements or sometimes even deteriorated the performance.

References

[1] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley & Sons, 2012.

[2] Chang, Chih-Chung, and Lin CJ LIBSVM. "A Library for Support Vector Machines, 2001." Software available at http://www. csie. ntu. edu. tw/cjlin/libsvm (2012).