Fifth Machine Learning Assignment

Gilles Peiffer
École Polytechnique de Louvain
Université catholique de Louvain
Louvain-la-Neuve, Belgium
gilles.peiffer@student.uclouvain.be

Abstract

This paper contains a short report detailing the various implementation choices used in the design of the model we use for classification. This includes a note on the software that was used for classification, the various models which were explored and how they were evaluated, as well as some notes on specific topics which were deemed useful.

1 Software

Various existing, public software packages were used (in Python): Scikit-learn, Pandas, NumPy, XGBoost and CatBoost. The first of these is a general library for machine learning-related functionality; the second and third are data manipulation and numerical computation libraries. Finally, XGBoost and CatBoost are two libraries for gradient-boosted decision trees.

2 Data preprocessing

A first concern with the data at hand is about filling missing values. Various strategies were explored, and after careful benchmarking, we decided to fill in missing values in a given column by the most frequent value in that column. Other possibilities included using the mean, the median, or even an imputer based on the nearest neighbors of a data point.

Next, we had to decide on how to deal with categorical features, since not all algorithms are able to work with this type of data. The dataset has, fundamentally, two types of categorical features: so-called *ordinal* features, where some order is present in the possible values of the feature (e.g. "normal" < "high"), and *nominal* features, where this is not the case. In the first case, we use an encoding which maps the n possible values of the features to $\{0, \dots, n-1\}$, thus preserving the order between the various feature values. In the second case, we use a so-called "one-hot" encoding, which creates new separate features for each possible value, putting a "1" in the column for the right feature value and a "0" in the other columns.

In order to deal with feature values which are not present in the training set, we need only worry in the case where the feature is ordinal: in that case, it is important to specify what the ordering is for that feature, with the new value. In the numeric or nominal case, the treatment is done automatically, as explained above.

Other preprocessing steps include feature extraction (using a principal component analysis), feature selection (using a χ^2 statistical test to select the most significant features, as well as significant feature selection using the gradient-boosted decision trees), and scaling of the features to prepare them for variance-based classifiers. Multiple sklearn pipelines were used to make this process efficient and easily verifiable.

3 Model selection

Various models were explored: (boosted/bagged) decision trees, random forests, (deep) neural networks, SVMs, LDA, QDA, and KNN.

To gauge the quality of each model, stratified k-fold cross-validation using the BCR was used, to deal with the class imbalance present in the data. Several high-scoring models were then kept in order to further tune their hyper-parameters to obtain our final model.

More precisely, initial testing revealed that three types of models led to good performances: decision trees (with gradient-boosted DTs scoring the best out of them), MLP models, and linear SVMs. After playing around with hyper-parameters and preprocessing steps, it

quickly became clear that the MLP models were very prone to overfitting, whereas the DTs could be used as a preprocessing step rather than the final classifier. On top of that, SVM models are very theoretically sound, especially in "big-p-little -n" problems where the dimensionality exceeds the sample size, because they are fairly resistant to overfitting when tuned correctly.

To determine the optimal hyper-parameters for each model, we used sklearn's `GridSearchCV` method on an exponentially distributed grid of hyper-parameters, which we then further refined depending on the results of previous tests.

4 Final classifier

Our final classifier uses the preprocessing steps in Section 2 to select 240 features from the original set (60 from each of PCA, XGBoost, CatBoost, and k-best using a χ^2 test). These features are then scaled to have zero mean and unit variance, before going through another (whitened) PCA with 30 features, and are then fed to a random forest of bagged linear SVMs, which determine the final label prediction.

Despite the PCA being an unsupervised technique, we chose not to include the test set when computing it, as this would introduce an uncertainty in the computation of the expected BCR of Section 5 without guaranteeing improved results.

5 Expected results

In order to accurately predict our true balanced classification rate on the test set, we use stratified k-fold cross-validation. A plot of the learning curve during cross-validation is shown in Figure 1. As one can see on Figure 1, the classification rate seems to be close to 72 %. We can also see that, surprisingly, the standard deviation increases when training on larger datasets, despite the expected increase in

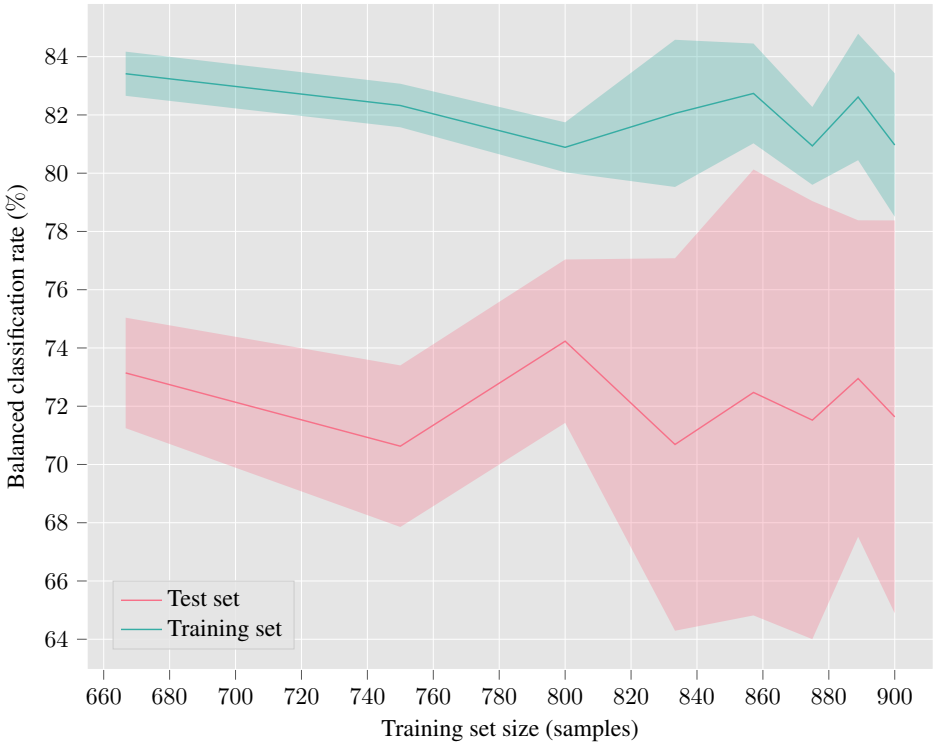

Figure 1: Learning curve of the final classifier (the bars indicate the mean of the cross-validated BCR, whereas the shaded region covers one standard deviation)

accuracy due to the larger training set and the higher number of folds. However, one possible explanation for this is that because the test set gets increasingly small, due to the class imbalance, several wrong classifications in the infrequent class can have a big impact on the balanced classification rate. Taking all of this into account, we expect our model to perform in the 72 % range on the final test set.

This learning curve only contains the relevant part, once the BCR seems to stabilize, as smaller training set sizes are not representative for our final BCR.