Advanced Machine Learning – Project 2

Project report

Piotr Bielecki
Tomasz Krupiński
Norbert Niziołek

June 3, 2024

Contents

1 Introduction

The project task was binary classification on an artificially generated dataset containing 500 features and 5 000 samples. The goal of the project was to find 1 000 samples which are the most likely to belong to class 1, while using as few features as possible.

The solution was implemented in Python using method implementations from the `scikit-learn` library and XGBoost implementation from the `xgboost` library.

2 Metrics

Before starting the experiments, it was important to establish metrics that would be used for evaluation of the trained models. The simplest method used for this purpose was accuracy – the fraction of the samples which were correctly classified. This metric, however, does not apply well to our task.

For this reason, another metric was devised, called *top 20% accuracy* (Acc_{20}). It's goal is to be the equivalent to the scoring system of the task. Its algorithm is:

1. Calculate the class 1 probabilities for the samples from the test set.
2. Sort the probabilities by value, descending.
3. Take the top 20% of this list – this is equivalent to taking 1 000 out of 5 000 samples.
4. Calculate how many of those samples actually belong to class 1.

Based on the top 20% accuracy, a score can be calculated by multiplying the value by 10 000 (which is the maximum amount of points that can be achieved in the task) and then subtracting the number of used features multiplied by 200. This score was used to compare the trained models and select the best one that was eventually used for creating the deliverable files.

$$Score = 10\,000 \cdot Acc_{20} - 200 \cdot \#[used\ features]$$

When the experiments were repeated over multiple train/test splits, the scores were averaged. When models were tested with various combinations of hyper-parameters, the scores for the best-performing combinations were selected, and they were used to generate all the plots in the report.

3 Experiments

3.1 Preliminary testing

The first step to select the best methods was preliminary testing, for which the goal was to give us a general idea of the performance of various models for feature selection and classification. This was split into two separate experiments.

For both of the experiments, every combination of models was tested once using a single 80/20 train/test split and 1, 4, 7, 10, 13, 16 or 19 features.

The first experiment was focused on comparing classifiers. The feature selection was conducted using Mutual information with `SelectKBest` model and the following models were tested:

- `GradientBoostingClassifier`
- `RandomForestClassifier`
- `SVC` (SVM classifier) with `linear` kernel
- `SVC` (SVM classifier) with `rbf` kernel
- `LinearDiscriminantAnalysis` (LDA)
- `QuadraticDiscriminantAnalysis` (QDA)

The second experiment was focused on comparing feature selectors. Multiple feature selection models were tested with the `RandomForestClassifier` classifier. The models tested were:

- `RFE` (Recursive Feature Elimination)
- `SelectFromModel`
- `SelectKBest`
- `SequentialFeatureSelector` (only for 1, 4 and 7 features because of execution time)

The results of those two experiments are shown in figures 1 and 2. The conclusions from this experiment were that LDA, SVM with a linear kernel, `SelectKBest` and `Sequential FeatureSelector` were significantly worse than other methods and it is not worth investigating them further. It was also found that the execution time for `SelectFromModel` was two orders of magnitude faster than RFE or SFS, while providing the best results, so it was used for the next experiments.

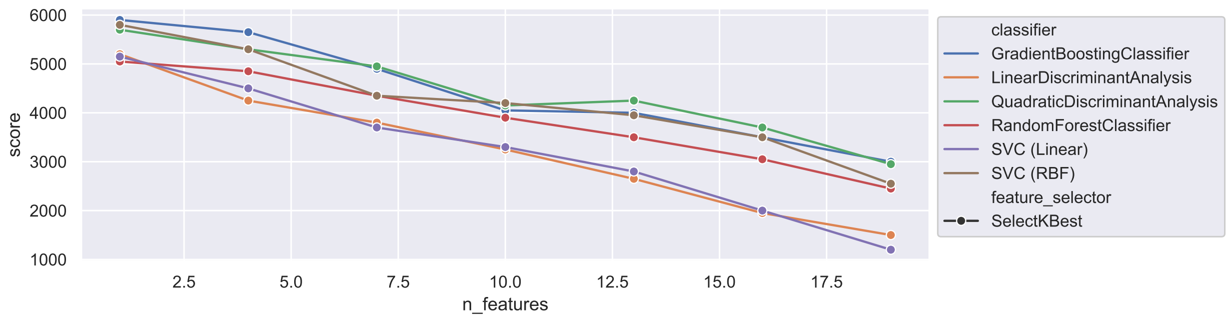

Figure 1: Results of preliminary testing of classifiers.

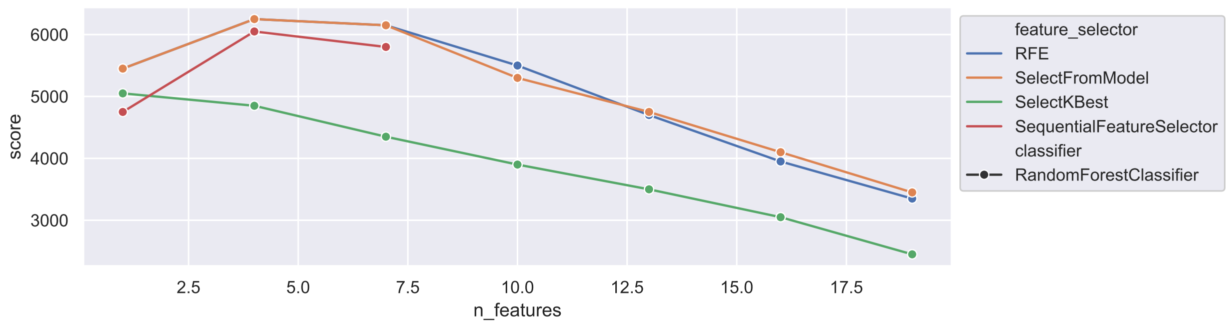

Figure 2: Results or preliminary testing of feature selectors.

3.2 Classification model comparison

After the initial tests, the next step was to test some of the methods in more detail – using all possible numbers of features, adjusting their hyper-paremeters. In these experiments, each model was trained five times on different train/test splits in order to get more statistically meaningful results. For each model, for each number of features, the optimal hyper-parameters were selected using the grid search method – by evaluating all combinations of their values.

This stage was split in two parts. The first one was a comparison of methods which support the `SelectFromModel` feature selector. This feature extraction method works, by utilising the coefficients of a classification model, so it cannot be used with models which do not provide coefficients. The following classification methods were selected:

- `XGBClassifier` (from `xgboost` library) – chosen as a more powerful, more efficient and more configurable alternative to `GradientBoostingClassifier` from `scikit-learn`
- `RandomForestClassifier` – chosen because of its high configurability

In the second stage, models which could not be used with `SelectFromModel` were considered. In order to test them, a feature ranking was first created using Recursive feature elimination(`RFE`) model with `RandomForestClassifier` estimator. Then, the classification models were trained on subsets of n best features from this ranking. The models used here were:

- `SVC` - Support Vector Classifier
- `MLPClassifier` – Multi-layer Perceptron classifier, chosen for its higher complexity

The results of this comparison are presented in figure 3. The `XGBClassifier` and `SVC` models provided the best results, and for both of them, the score values were the highest at 3 and 4 features.

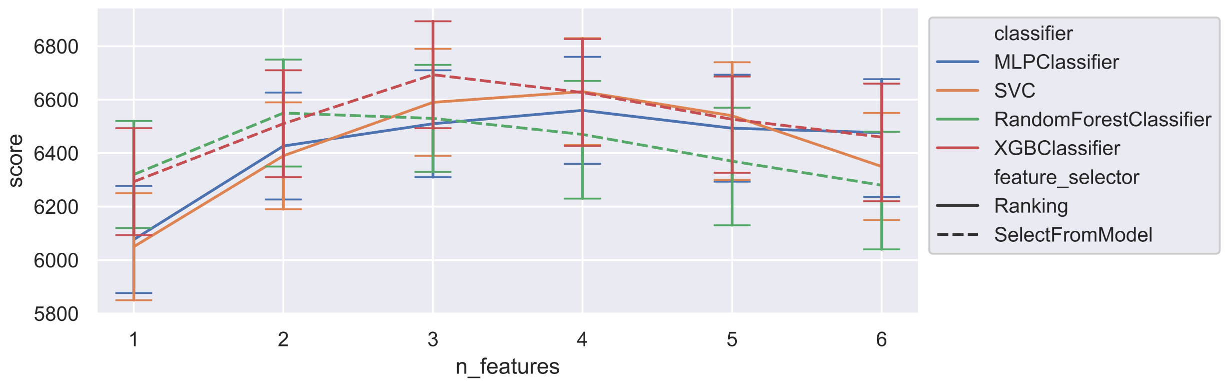

Figure 3: Results of classification model comparison.

3.3 Ensembles

Having conducted the tests mentioned previously in the chapter, we decided to try to ensemble several models with soft-vote strategy, using the features ranked by RFE selector:

1. `SVC`
2. `MLPClassifier`
3. `XGBoost`

In order to do that, we first conducted a hyperparameter space grid search for all three of the classifiers above. The best parameter combinations were then used in ensembling evaluation. The mean results from 10-fold crossvalidation for each possible combination of the classifiers are presented in figure 4.

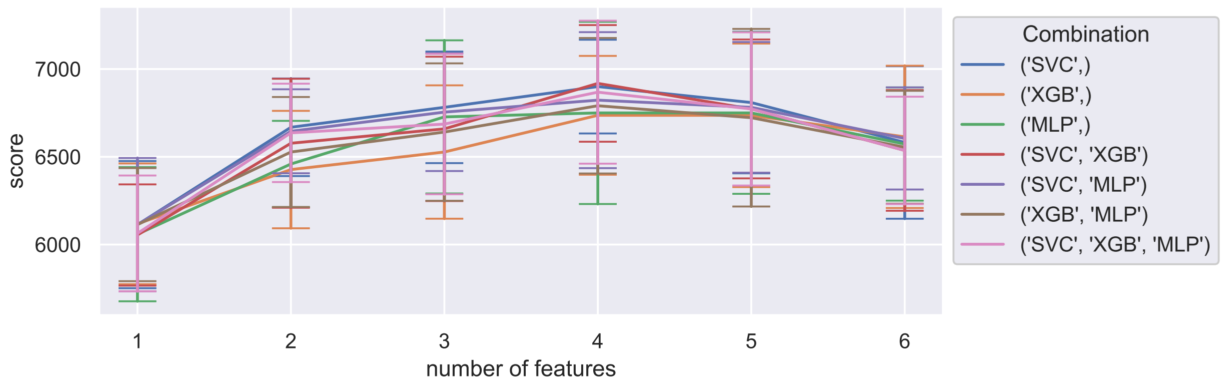

Figure 4: Results of ensemble models

The highest scoring combination turned out to be an ensemble of SVM & XGBoost models for 4 features, scoring just above the 6900 mark.

4 Final model

We figured, that the best hyperparameters for a single model may not be optimal for an ensemble, therefore we performed another, more localised hyperparameter search (around the best hyperparameter set established previously), for this specific model pair, for 4 features, on multiple train-test splits. Our final model achieved a crossvalidated score, still, just above 6900.

Model	**Hyperparameter**	**Value**
SVM	`kernel`	`poly`
	`degree`	`4`
	`gamma`	`scale`
	`coef0`	`0.01`
XGB	`n_estimators`	`75`
	`learning_rate`	`0.01`
	`min_child_weight`	`0`
	`subsample`	`0.5`
	`lambda`	`0.1`
	`max_depth`	`2`
	`tree_method`	`approx`

Table 1: final ensemble model hyper-parameters

Our final model is an ensemble of SVM and XGBoost with soft-vote, with hyper-parameters described in table 1.