1 Own implementation of an AutoML pipeline

1.1 Pipeline description

Our pipeline is made for binary classification tasks. As parameters, it takes training set, the variable to predict and set on which we want to make predictions.

It performs a simple preprocessing, which starts with removal of duplicates. After this, missing values are being imputed. To do this, it uses KNNImputer with number of neighbours set to 1. Thanks to this, our pipeline will fill the gaps with values, that are already present in the set. It encodes categorical variables using OneHotEncoder. The pipeline drops columns with variance under 0.2. At the end of preprocessing it scales variables to be in range [0,1].

After preprocessing our pipeline trains three classic ML models: SVC, XGBoost and RandomForestClassifier. To find the best hyperparameters it launches Bayesian optimization on predefined parameters space. Then it trains two models, which use deep learning. Those models where taken from Keras package. Next, it takes all of the five models and ensembles them using Soft Voting to make the final model.

At the end, the pipeline fits the final model on training set and predicted variable, and counts the probabilities for the test set. The pipeline returns the final model and probabilities.

1.2 Results

To find out, how good our pipeline is, we run it on a set, which was given to us on lecture and compared the results with Auto-PyTorch results.

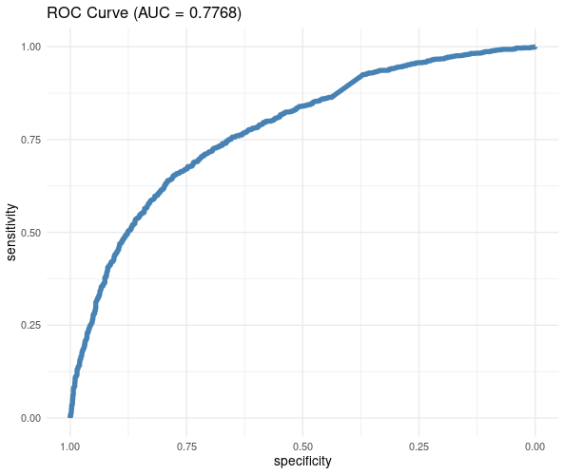

Figure 1: ROC curve - our pipeline

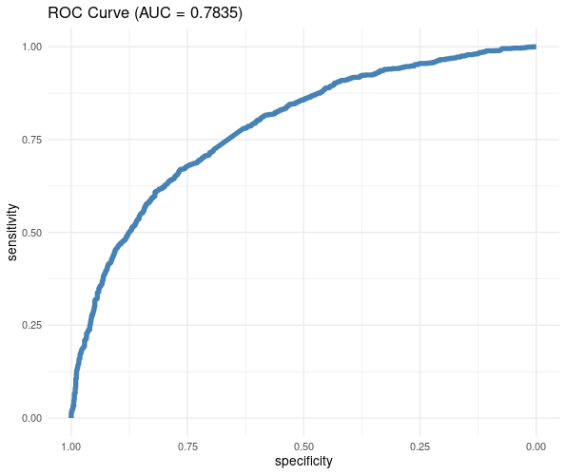

Figure 2: ROC curve - Auto-PyTorch

As we can see on Figure 1 and Figure 2, AUC of Auto-PyTorch prediction was higher than AUC of our pipeline prediction, but the difference was not big. Nevertheless, our pipeline needed one hour to evaluate, when Auto-PyTorch needed only ten minutes.