Kaggle Competition

Puerta de Toledo

Master in Big Data Analytics

Machine Learning

Ion Bueno Ulacia NIA 100364530
Daniel Martín Cruz NIA 100384121

Ion Bueno Ulacia NIA 100364530
Daniel Martín Cruz NIA 100384121

1. Intro

In this report we will briefly explain some of the ideas we have explored for the resolution of the problem proposed in this Kaggle competition. The report will be divided in two main blocks: the **preprocessing** part of the pipeline and the **classifier** used to predict. The best model was a hard voting between the predictions of 5 models.

2. Preprocessing

In this section we will explain the different steps we have followed for the preprocessing of the data to go from the raw document we downloaded to the optimal dataset used as input of our model.

2.1. Missing values imputation

As all the columns of the `X_train` provided had around 10% of missing values, it was important to fill that empty rows with proper values that allow us to produce models with a good performance. As a first approach we filled it with the median value of the column to start having some first scratch models and then we used PMM (Predictive Mean Matching) algorithm for the imputation of these values.

2.2. Feature selection

Due to the fact there were noisy variables, it was required to look if we could remove some features.

2.2.1. Correlated variables

The first approach, which it was the simplest, consisted of looking if there were highly correlated variables in order to remove them. In spite of having some features highly correlated with others, do not use them provides worse results in all the models tested until this moment. For this reason, we decided not to follow this approach, in addition to the fact that it is not the highest accurate processing.

2.2.2. Sequential Feature Selector

Second approach was looking which features were selected after applying the `SequentialFeatureSelector` from *sklearn*. After several attempts with different models in the selection as well as classifiers, the results were not satisfactory. Mention that the selected features were almost the same in all executions, but the performance was better without it.

2.2.3. Importance respect a random variable

Finally, we consider a simple technique but very effective in contexts with many variables. The idea is defining a random feature and using a classifier which allow us to examine the weights. Then we look which is the weight of the random predictor and remove the ones that are below (or below some threshold). Despite appearing to be a good approach, the results were again worse than using all variables.

2.3. Dimensions reduction

As none feature selection was successful, we trust on the idea of reducing the dimensions to avoid the effects of the noisy variables.

2.3.1. Principal Component Analysis (PCA)

It has been the best preprocessing step in most of the models, using all the variables. It has been tuned defining a range for the variance between 0.6 and 0.95, where in each model used in the voting has obtained a value inside the range. Mention the kernel option, Kernel PCA, has been also tested, but the results were worse than using the linear version.

2.3.2. Linear discriminant analysis (LDA)

The problem of this method is that we have just two classes, and the method of *sklearn* defines that the maximum dimension of the reduced space must be the minimum between the number of classes and the variables minus one. Then, the space was reduced to one dimension and the results were much worse than using PCA.

2.4. Scaler

Finally, we have employed the `StandardScaler` from *sklearn* in all the models, excluding the MLPs and in some Random Forest and Extra Trees.

3. Classifier

3.1. Machine Learning models

We have tried using KNN, Logistic Regresion, Random Forest, Extra Trees, Histogram Gradient Boosting, SVM and XGBoost. All of them with hyper-parameter tuning using `BayesSearchCV` from *sklearn*. The best results were obtained by the SVM with PCA and the Extra Trees, in both cases using PCA.

3.2. Neural Networks

We have employed MLPs with different number of layers and neurons, considering dropout and batch normalization. Mention the number of layers was three or lower and the number of units below 12, with the aim of avoiding overfitting. We also applied early stopping, in which the number of considered epochs was an important hyperparameter. We used 15 and 20. The same happens with the batch size, where the performance was very different between using 32, 64 and 128. The best results were obtained with 64.

3.3. Voting of model predictions

As mentioned, the best models consist in applying a hard voting of other models' predictions. The best submission was obtained by a voting of the following models:

- **SVM:** with scaler, PCA and hyperparameters tuning.
- **Random Forest:** without scaler and PCA, only hyperparameters tuning.
- **Extra Trees:** with scaler, PCA and hyperparameters tuning.
- **MLP:** with PCA and 3 layers with 4,6 and 4 units. No dropout nor batch normalization.
- **MLP:** with PCA and 3 layers with 8,10 and 10 units. No dropout nor batch normalization.

The public score with this model was **0.88546**, while the private one was **0.87798**. We have checked the rest of the submissions and see that other voting sets obtained a better performance. The best model in the private set has been a MLP with two layers with 8 and 10 neurons, obtaining a **0.88857** score.