XGBoost algorithm in classification problems

Małgorzata Rucka, Maja Szrubarczyk, Maciej Stroiński

1. Introduction

Gradient Boosted Decision Trees algorithms are frequently used in machine learning for both regression and classification problems. Multiple decision trees are combined in parallel to enhance the performance of a single decision tree. Using a wide range of hyperparameters can provide control over the model training procedure.

These algorithms, especially the Light Gradient Boosting version, are widely used in applications thanks to their effectiveness, simplicity of interpretation as well as the fast learning rate.

The main goal of this project is to test the performance of algorithms implemented in the XGBoost library in problems of classification. The Gradient Boosted Decision Trees algorithm will be applied to time series, numerical and textual data. Finally, the results of the algorithm will be compared to other methods for classification.

2. Datasets

In this section, the chosen datasets for classification will be presented.

2.1 Time series data

Predicting mortality of ICU Patients dataset.

The ‘Predicting mortality of ICU patients’ dataset consists of 8000 records in total. It contains information about the patients and the results of 37 different tests that were conducted on patients in the Intensive Care Unit within 48 hours after admission. The measurements were not taken regularly, which results in the data not being regularly spaced. Moreover, not all surveys were taken for every patient which causes a lot of missing values. In this classification problem, the patient can be classified as successfully treated or declared dead.

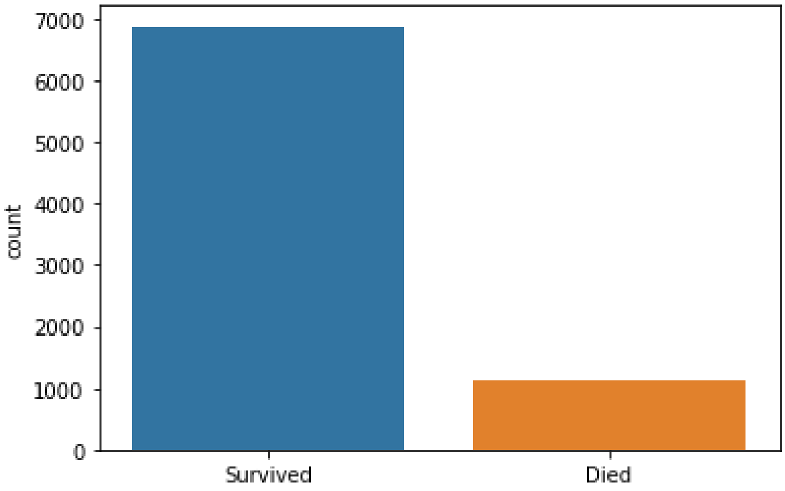


Image 1. Distribution of target variable with ‘Survived’ representing successful treatment and ‘Died’ representing patient’s death

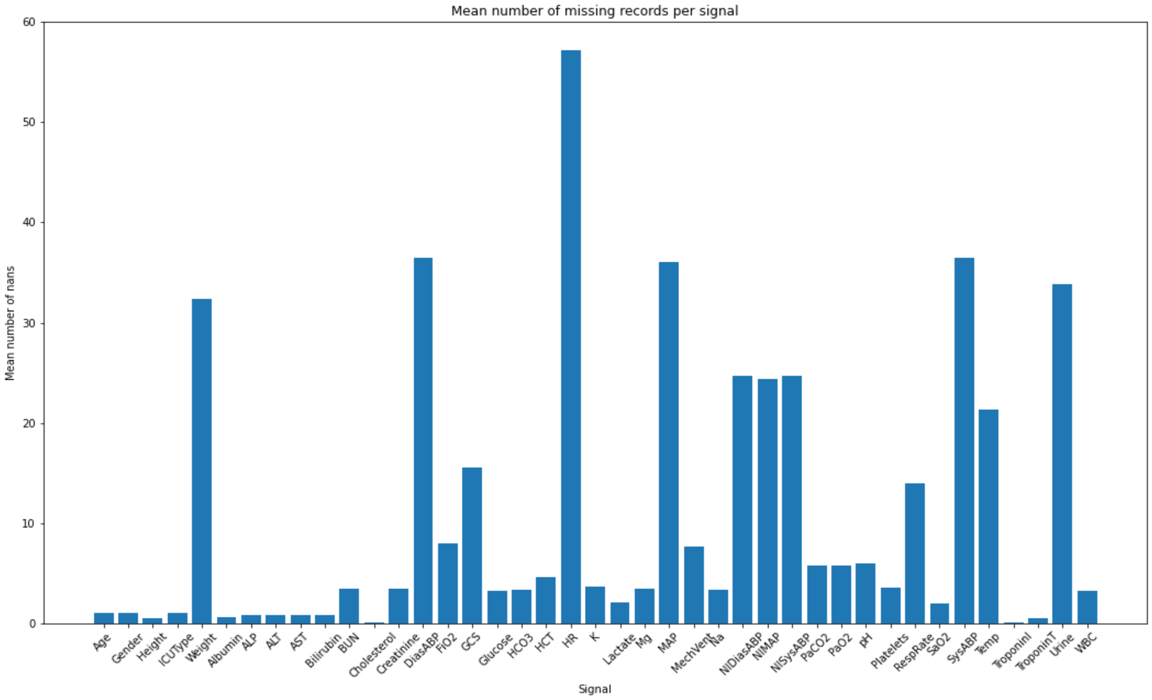


Image 2. Graph representing a mean number of missing records per signal calculated using the whole dataset

2.2 Numerical data

Polish companies bankruptcy dataset.

The ‘Polish Companies Bankruptcy’ dataset is described by 64 attributes and financial rates of several companies that indicate whether the company can be classified as bankrupt or not. This dataset is highly imbalanced as out of 10503 records only 495 were classified as bankrupt (10008:495).

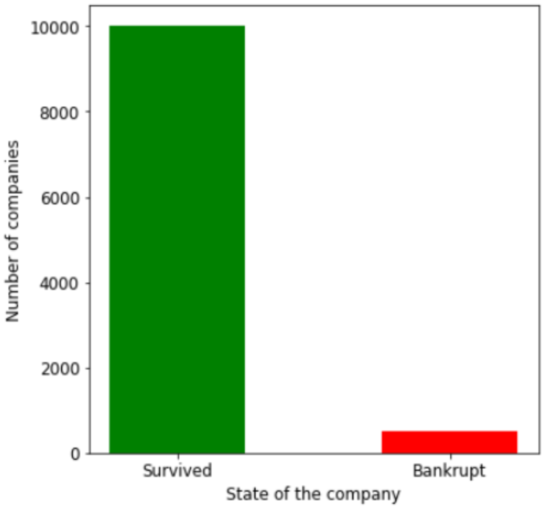


Image 3. Distribution of target variable with ‘Survived’ representing an active company and ‘Bankrupt’ representing a company that went bankrupt

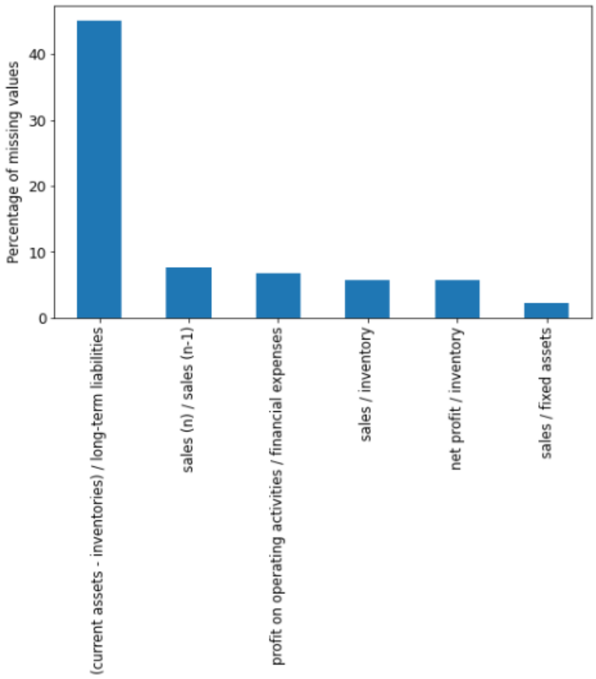


Image 4. Graph representing a percentage of missing records per attribute

2.3 Text data

Spooky Author Identification dataset.

In the ‘Spooky Author Identification’ dataset, there are 19589 records.

A single record consists of an exemplary text written by one of three authors: HP Lovecraft (5635), Edgar Allan Poe (7900), and Mary Wollstonecraft (6044). The average length of samples is 30 words (standard deviation = 20). This dataset contains no missing records and its class distribution is balanced.

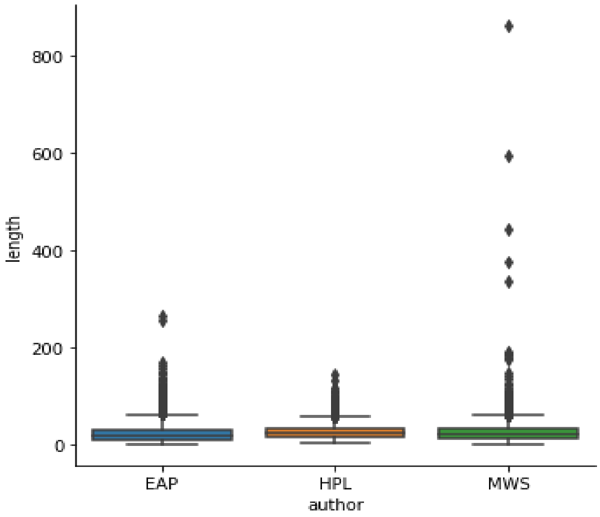


Image 5. Boxplots representing lengths of texts for each author from the dataset

3. Research methodology

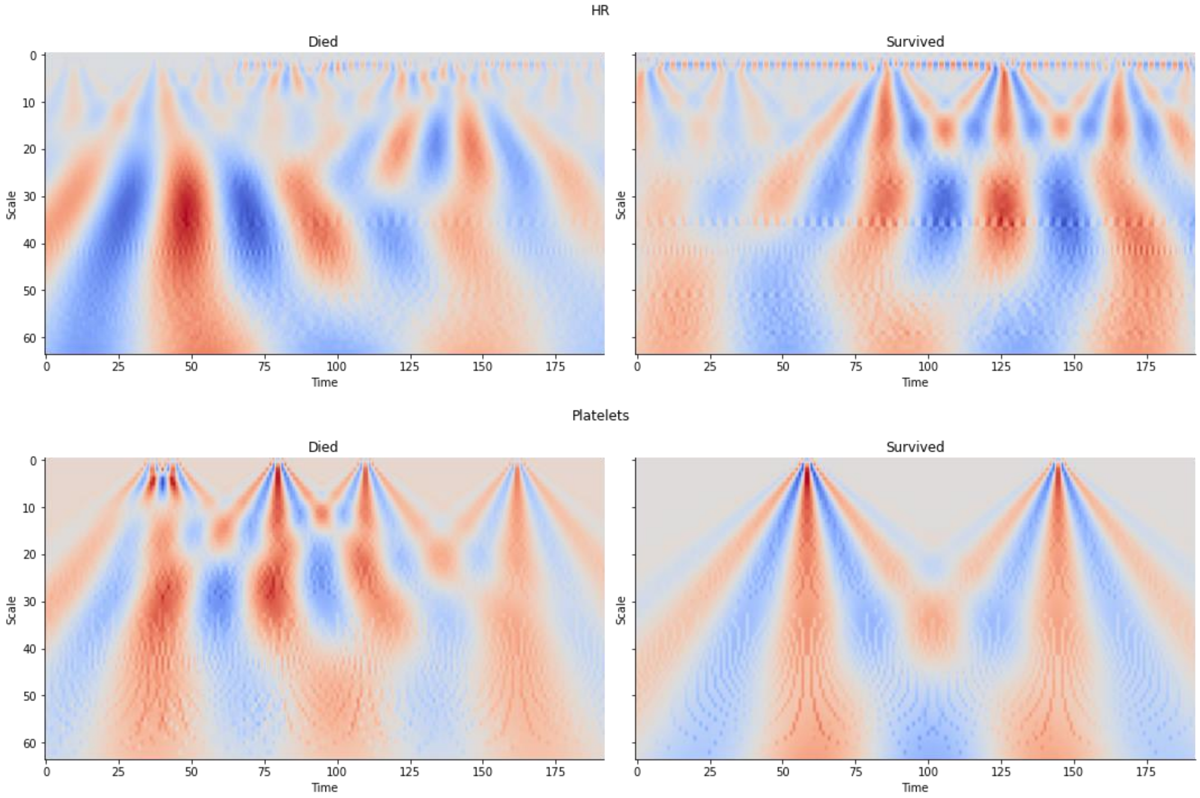
This section will explain the research methodologies that were used for each of the datasets.

3.1 Time series dataset

The first step performed after loading the dataset was to expand the data containing the examination times into a matrix representation. In this representation, each patient is represented by a matrix with 41 rows (4 patient data and 37 possible tests) and 2880 columns, each representing a minute of stay in the intensive care unit.

This way of representing data takes up a lot of space and is inconvenient for further analysis. Using the fact that the matrices for each patient are sparse, it is possible to perform 'bucketing' without losing information. Each bucket represented a 15-minute interval, which resulted in reducing the size of each patient's matrix to 41 rows and 192 columns.

The second step was to use a wavelet transform on the data. This will allow the data to be independent of time and find abrupt changes in the time series. Before it, missing values in a single signal were replaced by a mean value of it. Each signal for a given case is represented as a 64 (parameter scales) x 192 matrix created using wavelet transform. Those matrices can be visualized on a scalogram as shown in image 6.



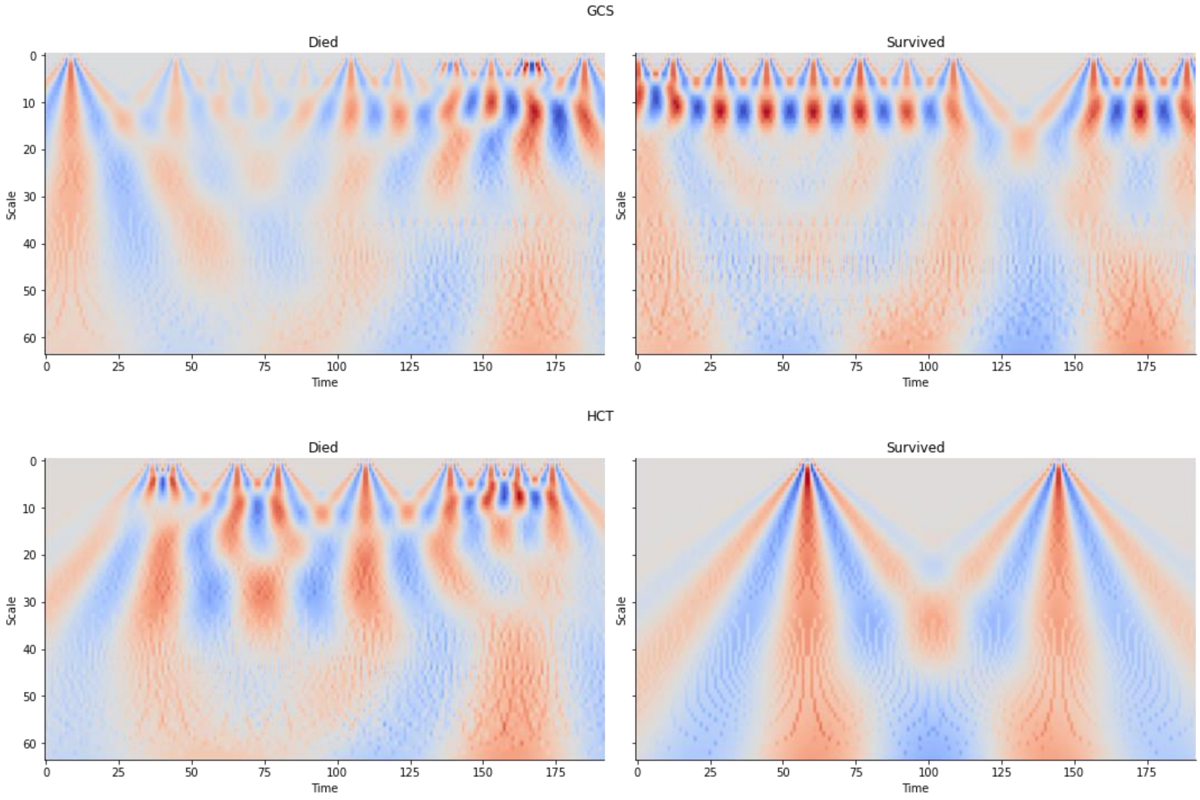


Image 6. Comparison of scalograms of chosen signals (HR, Platelets, GCS, HCT) after wavelet transform, one example for each case

The PCA algorithm then transforms those matrices to reduce the dimensionality and select the most important coefficients per scale to feed a classifier. The process is returning a vector of length 64 for each signal. The vectors are concatenated together, producing one vector of length 2624 for each patient.

The whole preprocessing and feature extraction pipeline is presented in image 7.

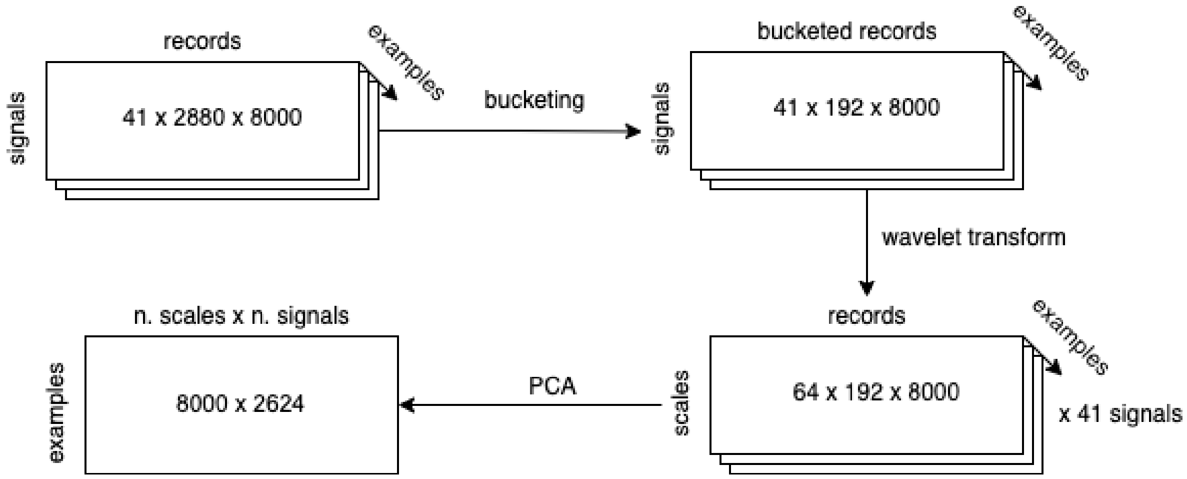


Image 7. Data preprocessing steps

After the initial experiments, the weight parameter was calculated. It puts more emphasis on the samples from the less frequent class, which helps to minimize mistakes caused by the unbalanced data. It was calculated as the ratio of the total of majority class examples (‘Survived’) to the total of minority class examples (‘Died’). The parameter was used during the model declaration during the next experiments.

The results achieved for XGBoost were compared with the SVM algorithm with stochastic gradient descent (SGD) learning and linear regression.

3.2 Numerical dataset

Due to the highly imbalanced class frequency in the ‘Polish Companies Bankruptcy’ dataset, the results of the models without any preprocessing are inaccurate as the minority class is misclassified.

Out of 10503 records, 5618 contained missing values denoted as “?”. Since XGBoost can perform well with missing values, they were kept in the dataset as NaN values.

After getting only numerical values from the data it was standardized with the StandardScaler from the scikit-learn library.

The XGBoost classification model was defined with the following parameters:

```
learning_rate=0.1, n_estimators=1000, max_depth=2,
objective='multi:softmax', nthread=4, num_class = 2,
scale_pos_weight=scale_pos_weight.
```

With this dataset, as well as with the previous one, the scaling weight, which describes the extent to which the algorithm is punished for the wrong prediction of the minority class, was calculated and used during the model declaration.

To compare the gradient boosting model to SVM and linear regression, more preprocessing steps were needed. As neither of these models is capable of handling missing values, they were imputed as the mean values of each column in which the missing values are located.

To handle the unbalanced class distribution it is possible to develop the cost-sensitive versions of both of the algorithms - similar to the XGBoost model. The SVC method for SVM classification provides the possibility to specify the class_weight parameter. Although it can be calculated with the same formula as in XGBoost (inverse of the class distribution), it was set directly with a “balanced” value. For the Logistic Regression model, the class_weights parameter had to be used. Its value was specified as the inverse of label distribution: 495:10008.

3.3 Textual dataset

The “Spooky authors” dataset required preprocessing, to convert it from the unstructured textual data into some numerical representation that can be used as an input to the model. Many popular algorithms, like TF-IDF, rely on statistical features of the data, like the frequency of word occurrence. This approach requires extensive preprocessing to reduce dimensionality and remove noise. Newer methods utilize transformer models, which can encode raw data while maintaining semantic and syntactic information and for that reason, this approach was chosen.

For this task, the only preprocessing steps were removing records that are longer than 510 words and tokenizing the data into word tokens with BertTokenizer. The length constraint is caused by the maximum number of input tokens for the BERT model, which was used to encode raw, textual data into vectors of 768 variables. Both BERT architecture and BertTokenizer were implemented with the HuggingFace library and pretrained with WikipediaCorpus and BookCorpus.
Two different sets of hidden states were used to generate word embeddings:

- the mean of the last 4 hidden layers
- the last hidden layer

Both sets of embeddings were used to train SVM, Linear Regression and XGBoost.

4. Results

In this section, the final results will be presented and compared with different algorithms.

4.1 Time series data

The calculated metrics for each algorithm are presented in Table 1. The highest accuracy was achieved with the XGBoost model trained on plain data, without preprocessing. It was expected, as the data was highly imbalanced - such a high score can be achieved by blindly predicting the class with more examples. Because of that, different metrics were also calculated - the F1 score for XGBoost with no preprocessing is very low and better depicts the model performance. After the preprocessing, the XGBoost model's F1 score is significantly improved, but accuracy drops. It shows, how important data preprocessing and feature engineering were for this case.

The XGBoost model achieved the best metrics scores among chosen algorithms. Both SVM and Logistic Regression got lower scores for Accuracy, F1 Score, Precision, and Recall. It proves, that XGBoost was a good choice for this dataset.

There still is room for improvement - on confusion matrices shown in image 8. can be noticed, that the XGBoost model confuses the 'Survived' label with 'Died' more often than the other two algorithms. This may be corrected by trying different weights, which were set to handle an imbalanced dataset. SVM and Logistic Regression had more trouble with confusing the 'Died' label with 'Survived'.

Table 1. Comparison of results of XGBoost, SVM and Logistic Regression for Time series dataset

	XGBoost no preprocessing	**XGBoost**	**SVM**	**Logistic Regression**
Accuracy	0.8618	0.756	0.654	0.683
F1 Score	0.0089	0.4403	0.3024	0.3153
Precision Score	0.1666	0.3310	0.2142	0.2302
Recall Score	0.0046	0.6575	0.5136	0.5

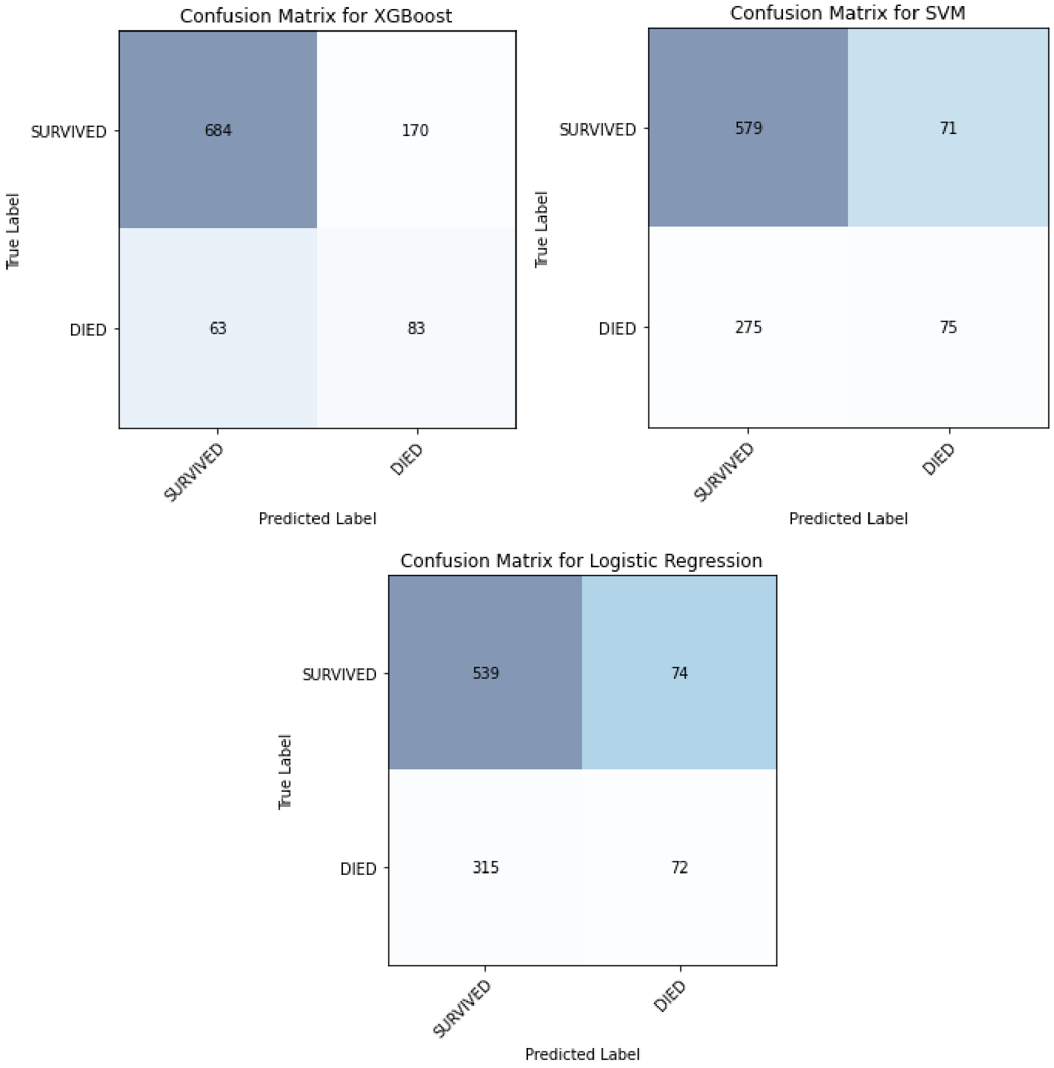


Image 8. Confusion matrices for each algorithm

4.2 Numerical dataset

In Table 2, the resulting metrics of the three algorithms are presented. As can be seen, the accuracy is the highest for the XGBoost achieving almost 97%. However, this is a common situation in the case of highly imbalanced datasets. Therefore, other metrics were calculated. When it comes to precision, the highest score was obtained by XGBoost and the lowest belonged to Logistic Regression. However, in the case of a recall, the score is not as good for XGBoost. Both SVM and Logistic Regression were better in detecting the actual positive cases. All in all, the combined score of precision and recall – F1 score – was the highest for XGBoost and lowest for Logistic Regression. Below are also presented the visualized results of predictions in the form of confusion matrices.

Table 2. Comparison of results of XGBoost, SVM and Logistic Regression for Numerical dataset

	XGBoost	SVM	Logistic Regression
Accuracy	0.9681	0.6639	0.6587
F1 Score	0.5442	0.1635	0.1534
Precision Score	0.7692	0.0921	0.0864
Recall Score	0.4210	0.7263	0.6842

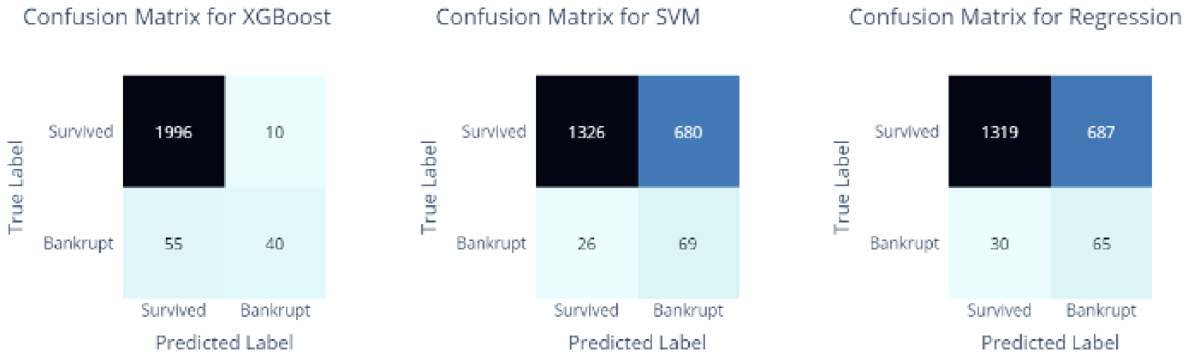


Image 9. Comparison of confusion matrices for XGBoost, SVM and Logistic Regression for Numerical dataset

As XGBoost seems to achieve the best results in a classification of the numerical dataset problem, the most important features of the classifier were distinguished. The results are presented in Image 10.:

- f26 - gross profit (in 3 years) / total assets
- f4 - current assets / short-term liabilities
- f45 - rotation receivables + inventory turnover in days
- f20 - (inventory * 365) / sales
- f28 - (net profit + depreciation) / total liabilities
- f57 - working capital
- f23 - net profit / sales
- f33 - (gross profit + interest) / sales
- f29 - profit on operating activities / financial expenses
- f32 - (total liabilities - cash) / sales

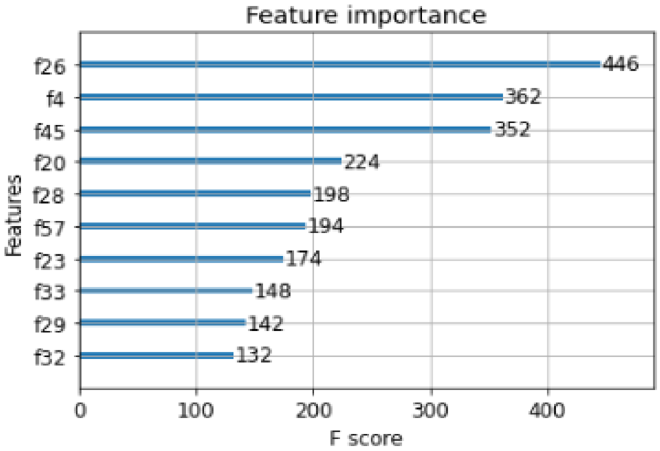


Image 10. The most important features in the XGBoost classifier

As a final remark, the time of calculations was the longest for the XGBoost model.

4.3 Textual dataset

Table 3 contains classification results with word embeddings calculated as the mean of the last 4 layers, and table 4 with word embeddings calculated as the last hidden layer. Usage of BERT for encoding data resulted in a 20% boost in accuracy for XGBoost, compared to Bag-of-word feature extraction.

Table 3. Comparison of results of XGBoost, SVM, and Logistic Regression for Textual dataset and word embeddings calculated as a mean of last 4 layers of BERT.

	XGBoost	**SVM**	**Logistic Regression**
Accuracy	0.792	0.809	0.806
F1 Score	0.792	0.809	0.806
Precision Score	0.793	0.811	0.806
Recall Score	0.792	0.809	0.806

Table 4. Comparison of results of XGBoost, SVM, and Logistic Regression for Textual dataset and word embeddings calculated as the last layer of BERT.

	XGBoost	**SVM**	**Logistic Regression**
Accuracy	0.788	0.800	0.799
F1 Score	0.788	0.800	0.799
Precision Score	0.789	0.800	0.800
Recall Score	0.788	0.800	0.799

This dataset proved to be a difficult task for the XGBoost algorithm. No matter the embedding used, the results were always lower than ones from SVM and LR. Poor performance of XGBoost might be contributed to:

- the lack of categorical variables, which are very useful for any tree-like algorithm
- low variance of a single feature encoded with transformer architecture, which makes it hard to use them in decision nodes

Furthermore, the size of the training data (768 features and 15661 texts) results in a very long training time of XGBoost compared to the other algorithms.

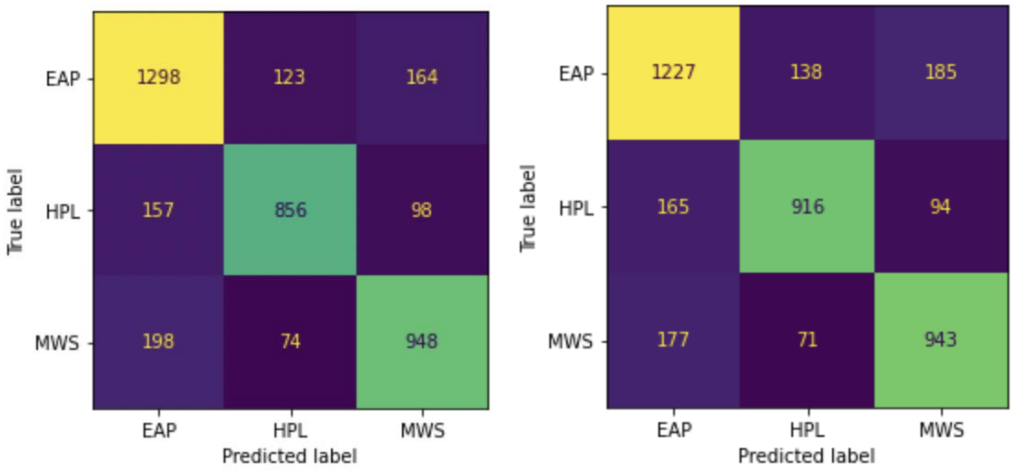


Image 11. Confusion matrices for XGBoost for 4 layer embedding (left) and 1 layer embedding (right)

Comparing confusion matrices shows that there are no classes that are getting mixed up with one another frequently.

5. Conclusion

In this project gradient boosting algorithms were compared with support vector machines and logistic regression on different classification tasks. On two of them, usage of XGBoost resulted in higher scores in every metric, and for one type of data, the results were marginally lower.
The best results were achieved for the Polish bankruptcy dataset, which contains categorical as well as numerical variables. This combination of features makes it a perfect scenario for any tree-like algorithm.
Both textual and time-series datasets were transformed into high-dimensional spaces, with each feature having a small variance. This way of representing data makes it harder to construct efficient trees, which results in worse classification.

For every task, different preprocessing and feature extraction techniques were used:

- bucketing, wavelet transform and PCA for time series data
- balancing datasets and imputation of missing values for numerical data
- word tokenization and data encoding with BERT architecture for textual data
- In every case additional effort with feature extraction resulted in better results.

6. References

https://machinelearningmastery.com/xgboost-for-imbalanced-classification/

https://machinelearningmastery.com/cost-sensitive-svm-for-imbalanced-classification/

https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

https://towardsdatascience.com/weighted-logistic-regression-for-imbalanced-dataset-9a5cd88e68b

https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/

https://towardsdatascience.com/multiple-time-series-classification-by-using-continuous-wavelet-transformation-d29df97c0442