EÖTVÖS LORÁND TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR

EÖTVÖS LORÁND UNIVERSITY
FACULTY OF SCIENCE

Mathematics Expert in Data Analytics and Machine Learning

Project Work

Thermoplastic injection molding scrap detection from machine parameters

Zsolt Bugjó

Budapest, 2023

Contents

1. INTRODUCTION **3**
2. BACKGROUND INFORMATION **4**
2.1. PRODUCTION PROCESS 4
2.2. AVAILABLE DATA 5
2.3. TASK DEFINITION AND MOTIVATION 6
3. DATA PREPROCESSING **7**
3.1. DATA CLEANING 7
3.2. EXPLORATORY DATA ANALYSIS 8
3.3. PRINCIPAL COMPONENT ANALYSIS 9
4. MODELLING **12**
4.1. DATA MINING TECHNIQUES 12
4.2. NEURAL NETWORKS 16
4.2.1. Multi-layer perceptron (MLP) *16*
4.2.2. Autoencoder neural networks *18*
5. SUMMARY **22**
REFERENCES **23**
APPENDIX **24**
1. EXPLORATORY DATA ANALYSIS GRAPHS 24
2. GRAPHS OF MODELLING PHASE 27

1. Introduction

Machine learning and AI has improved dramatically in the past few years and thus it is increasingly getting integrated to our everyday lives, providing us more and more opportunities and functionalities. Also in the industrial segments it is getting more attention, since companies are seeking productivity to stay competitive in the market. The company where I work, is a supplier in the automotive industry and is no exception to this.

The core technology in the company is thermoplastic injection molding, and we put a lot of effort into process improvement, in order to reduce the manufacturing costs and minimize the losses that are present because of the scrap parts, and malfunctions of the machine, mold or other peripheries.

In my project work I was focusing on one specific machine with one specific mold producing an identical product, and my target was to develop a model which can detect the faulty cycles of this specific process from the available machine parameter data. During my work I experimented with the „classical" data mining techniques as well as with different types of neural networks in order to find out which approach is fitting my problem better.

In section 2 I describe all the necessary background information about the project including the brief overview of the production process, the available dataset and the corresponding challenges during the preprocessing, and the different motivations behind the task with the possible use-cases to value creation. After that I introduce the data preprocessing steps that I made including the cleaning and the exploratory analysis of the dataset. Then in section 4 I compare the different data mining algorithms and their performance in order to select the best fit ones to move forward to the hyperparameter optimization and test the tuned models on the unseen dataset. The outcomes of this analysis will be compared to the results of the other approach, which is the application of different neural networks. Based on the characteristics of the dataset I experimented with MLP (multi-layer perceptron) and autoencoder structures with different training methods and hyperparameters.

2. Background information

2.1. Production process

The subject of my project is an injection molding process of one specific product, which is produced with a 16 cavity mold, meaning that during one cycle 16 parts are made.

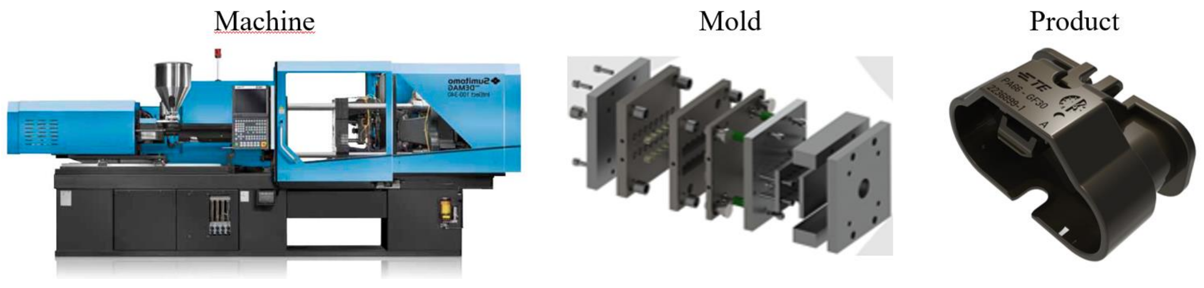


Figure 1: Illustration of machine, mold and product

The quality of the product is evaluated based on the measurements of multiple parameters, which can be divided into two groups. One is related to the machine and the other is referring to the mold. Every injection molding machine has the functionality to measure the critical parameters of the process, and there is a built-in control system which can detect if those parameters are going out of the predefined tolerance, and if that is the case the parts are automatically scrapped and put into a separated container by the machine. On the other hand, the general practice of the company is that we also install pressure sensors inside every cavity of the mold to get more information about every single part that is produced during the cycle. For every cavity there is a reference pressure curve which is related to the good part, and the tolerances are defined as windows at some of the noteworthy points of the curve. Then if any of the defined cavity pressure characteristics are out of the specification limits, the measurement system sends a signal to the machine to scrap the parts. Summarizingly we can say that if during the molding cycle, the mentioned machine parameters and the cavity pressure measurements are within the tolerance, the parts are most probably good quality parts. Why I used the expression most probably is because there are such types of failures that are occuring during the process, which cannot be detected by monitoring these values. As a consequence operators need to do visual inspections on the parts at predefined times. In our case every 2 hours one shot (i.e. 16 pieces) is checked visually.

2.2. Available data

For this pilot machine we established a data streaming for both measurements, meaning the machine parameters and the cavity pressure values. The latter could be considered as a consequence of the former, so from now on I am going to refer to them also as inputs and outputs. Because of the fact that the data is collected from separate sources and the measurements are taken at different times, we needed to find a way to link together the data which are related to the same cycle. It was solved by harmonizing the timestamps of the two sources.

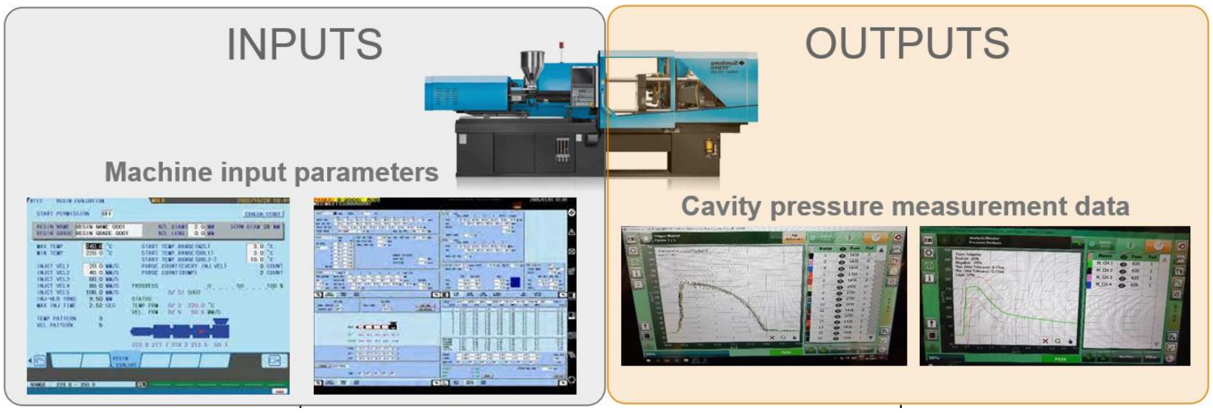


Figure 2: Different sources of data collection

This was working well for a while, but then something went wrong and the data was not collected properly and as a consequence the available dataset for those instances where the measurements are linked together is around 500K cycles. The automatic label for good/scrap parts could be derived from the machine parameters set, for which we have 1,5M cycles available. The typical scrap rate of this specific production process is around 1,5% and unfortunately, based on the currently available datasets we cannot distinguish between these scrap cycles in terms of which measurement resulted in a bad part. Relying on the experience of the subject matter experts, in most of the cases the cause of the scrap cycle is that the cavity pressure measurements are out of the specification limits. In the end this makes sense, because that measurement system is able to detect the effects of the process variation on part level, hence it is more sensitive. Actually this fact lead to one of the motivations that is initiated the actual project, which I will further describe in the next sub-section.

2.3. Task definition and motivation

The task itself was mentioned in the introduction, i.e. to develop a model that can detect scrap cycles of the molding process, using only the machine input parameters. The motivation behind this is to check if we can run the production with only one measurement system used for product quality evaluation, meaning that in case it works we can eliminate the cavity pressure measurement from our process. Assuming that we can provide the same quality, this could mean a significant amount of cost saving for the company, because the cavity pressure sensors:

- are relatively expensive
- need to be repaired/replaced frequently
- are in some cases difficult to install because of the mold characteristic

Despite the fact, that every mold is unique and the results of such analysis could vary, the scalability of this approach in our company is huge and thus could bring us enormous benefits in the future.

3. Data preprocessing

3.1. Data cleaning

For all the details of this section please refer to the „*data_cleaning.ipynb”* file, here I only describe the main steps that were made.

First of all I had a looked on the nan values of my dataset and based on the counts I dropped the affected columns or rows accordingly. Then I checked if any of the features have an identical value and dropped the column if so. After that I investigated those records that have 0 values, and since neither the number of rows and the ratio of scrap cycles was significant I decided to drop them as well (considering these instances as possible measurement or data streaming errors).

As a last step for data cleaning I made an outlier analysis for my predictor features. I used the z-scores ($z = \frac{x - \mu}{\sigma}$) to determine the outlier values, meaning that all samples that fall outside the $\mu \pm 3\sigma$ border are considered as outliers and then I compared the distribution of the output class in the outlier set to the whole dataset.

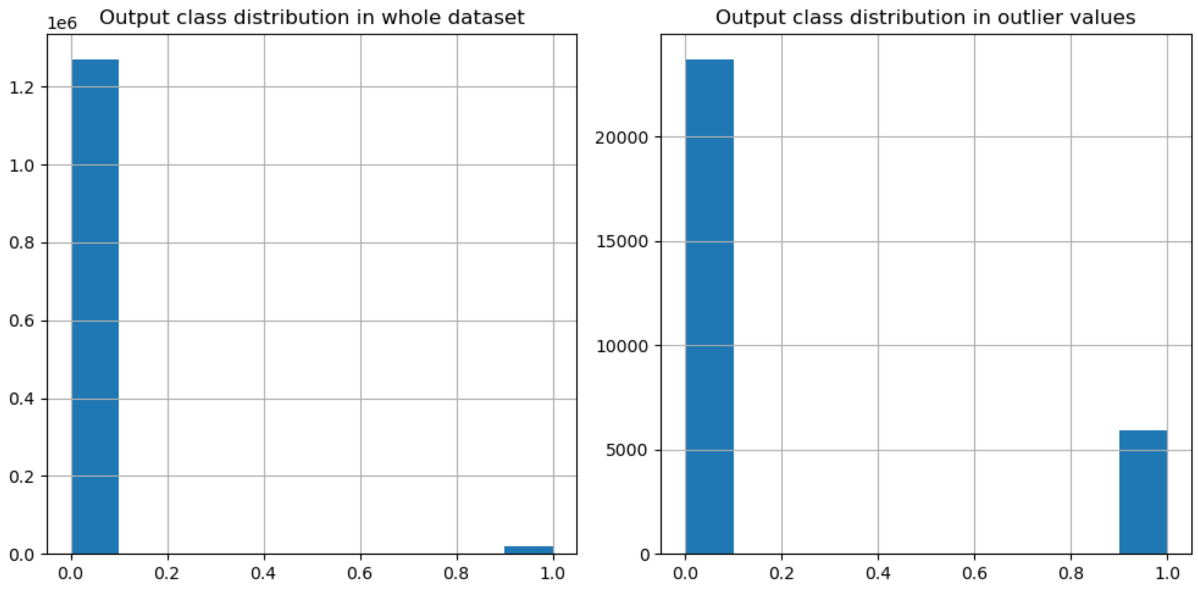


Figure 3: Output class distributions in the whole dataset and the outlier values

It is visible that the output class is highly imbalanced, we have a very small number of bad samples and because of the fact, that these scrap cycles have significantly higher ratio in the outlier records, I decided to drop only those outlier rows that are related to the 0 class (good cycle). After that I separated 10% of the dataset for testing.

3.2. Exploratory data analysis

We have all together 28 predictor features and all of them are continuous variables. I checked the distribution of all of them, but because of I left all the outliers related to the scrap class, these histograms are not so informative as you can see on *Graph 1* in the Appendix. Then I created a boxplot for every feature grouped by the output class. Generally we can derive from *Graph 2* (see in the Appendix) that at some of the features, the values are spread over a wider range in case of the scrap class (not considering the outliers), but no significant bias could be identified between the two classes.

In order to check the linear relationship between each predictor pairs and for the output class as well, I created a correlation matrix and visualized it as a heatmap.

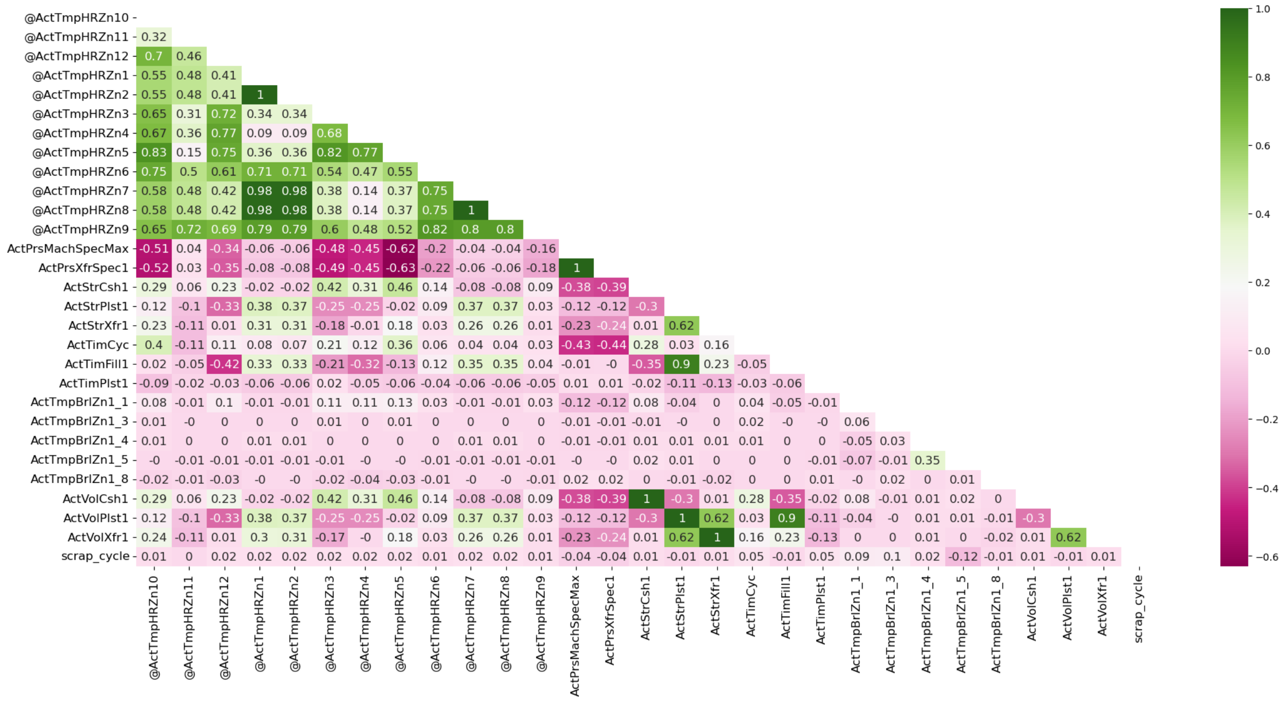


Figure 4: Heatmap of the correlation matrix

We can conclude from this graph that the input features are not strongly linearly correlated with the output class, but there are some interdependencies between them, and also some special cases where the correlation value is 1. After I further investigated these feature pairs I decided to drop one of them, except for the *„ActPrsMachSpecMax"* and *„ActPrsXfrSpec1"* variables. One is referring to the peak pressure value and the other is to the switch over pressure value, which are most of the cases are equal during injection molding processes, but not necessarily. I also made scatterplots for those pairs, which had correlation value over 0.8 (see *Graph 3* in Appendix).

3.3. Principal Component Analysis

As a last step of data preprocessing I made a Principal Component Analysis (PCA). The first thing I wanted to do is to have a 2D representation of my dataset in order to be able to visualize it.

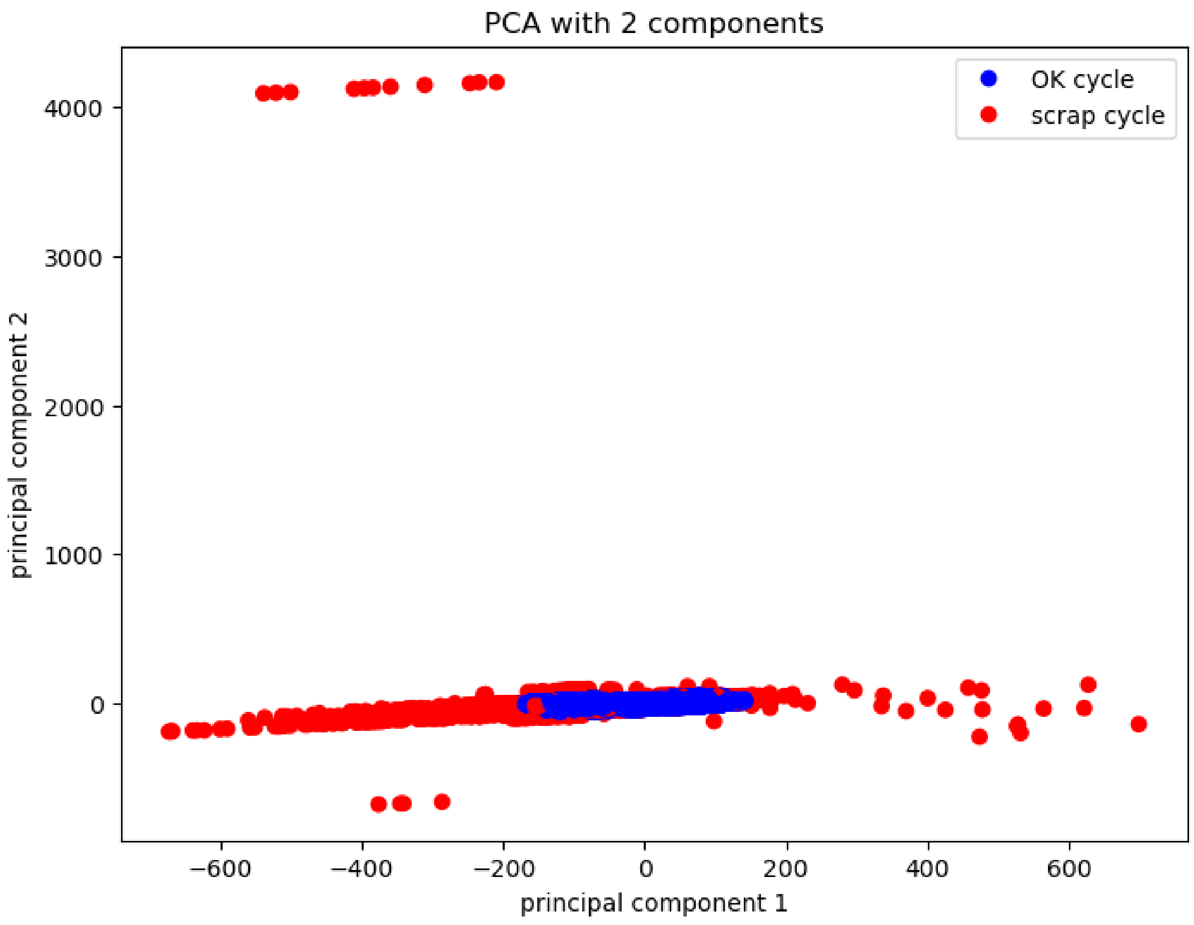


Figure 5: 2D representation of the dataset by using PCA with 2 components

Based on the scatterplot the visual intuition is that a subset of the scrap cycles could be easily separated, but there are samples that are very similar. I made grouped density plots for the principal components to get more insight.

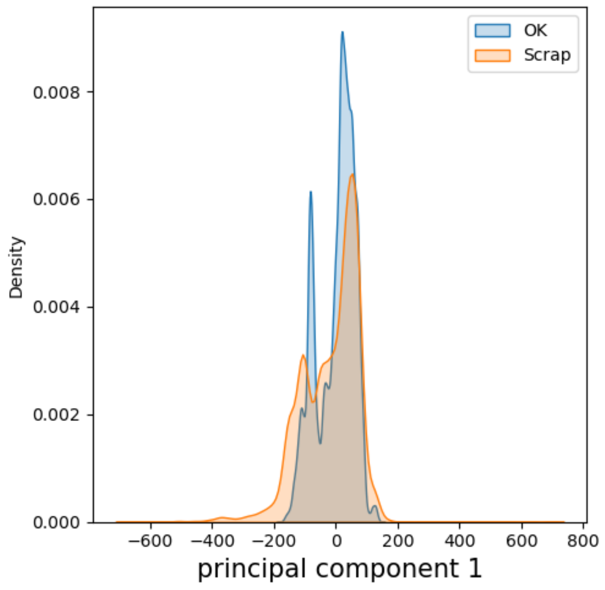


Figure 6: Grouped density plots of principal components

Not in the quantities I thought (more), but surely there are very similar samples of good and scrap cycles based on the principal components. Hoping that with an additional dimension the points in the middle could be further separated, I fitted the PCA class to the dataset with 3 principal components as well, and made a 3D scatterplot to check the results.

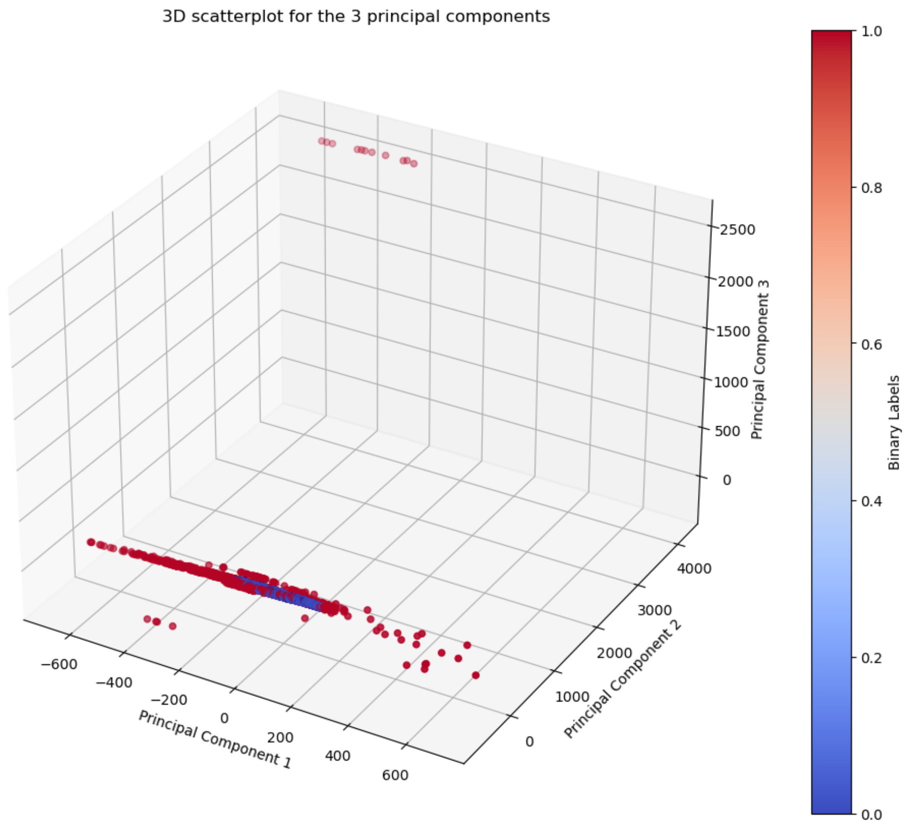


Figure 7: 3D scatterplot for the 3 principal components

Unfortunately, adding the third principal component and dimension did not help us to separate the points better in the middle. In order to find out what could be the optimal number of principal components, I made a so called scree plot to visualize how much of the variation is explained by the different number of factors.

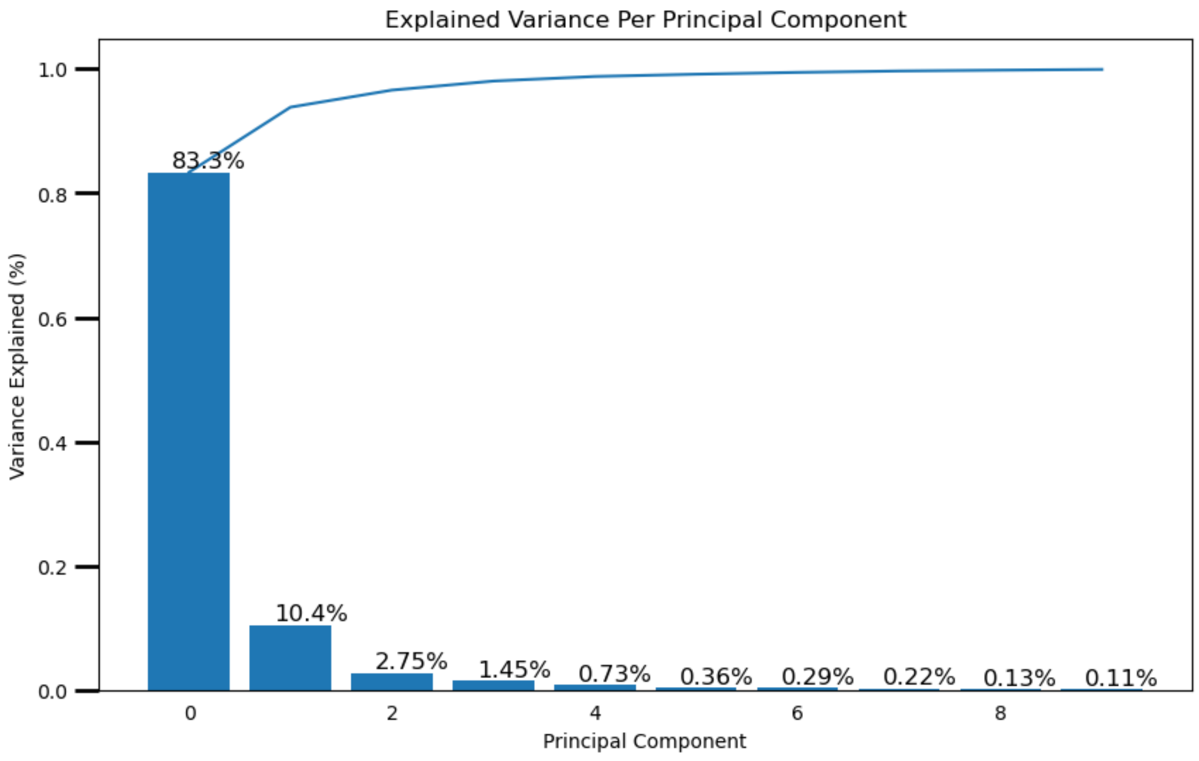


Figure 8: Scree plot

Based on these results I decided to keep those factors that are explaining at least 1% of the variance, so I fitted the algorithm with 4 components and saved the transformed dataset for later analysis.

4. Modelling

During the modelling phase I experimented with various machine learning techniques that could possibly fit my highly imbalanced binary classification problem. I started with conventional data mining techniques for supervised learning and then investigated some neural network architectures as well.

4.1. Data mining techniques

First of all I collected some algorithms to experiment with, which are potentially good candidates for this specific task:

- Random Forest
- XGBoost (Extreme Gradient Boosting)
- Light Gradient Boosting Machine (LGBM)
- Gaussian Naive Bayes
- Adaptive Boosting (AdaBoost)
- Logistic Regression

In order to get consequent results I fixed the random seed for all the instances and I left all the other hyperparameters on default values, because first I wanted to get an insight which architecture is worth further optimization. I created an evaluation framework for the model comparison, in which I used repeated stratified kfold for cross validation with 10 splits and 3 repeats, and during each fold I calculated a set of metrics and in the end I took the average of them:

- AUC – Area Under ROC Curve
- Accuracy
- Precision
- Recall
- Geometric mean
- F1-score

Because of the task characteristics, I was mainly focusing on recall, i.e. to catch as much of the scrap cycles as possible, but on the other hand it is necessary to monitor other metrics as well, like precision, to make sure not to have too many false positive predictions. On top of these metrics, I also added some visualizations to the comparison, namely the ROC curve and the

Precision-Recall curve. Latter can clearly show what is the trade-off between those two metrics when using different thresholds as a decision boundary for output class categorization. I applied the evaluation framework for the complete set of features and also for the previously prepared 4 principal components, and latter provided significantly worse results for all models.

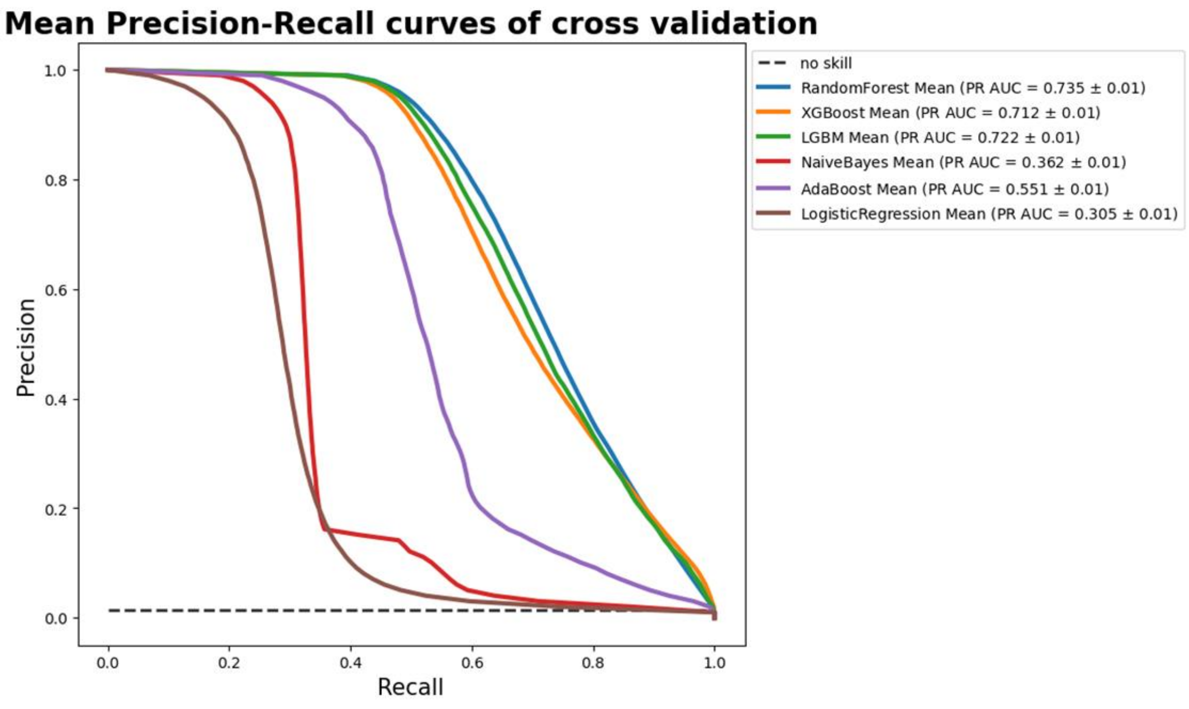
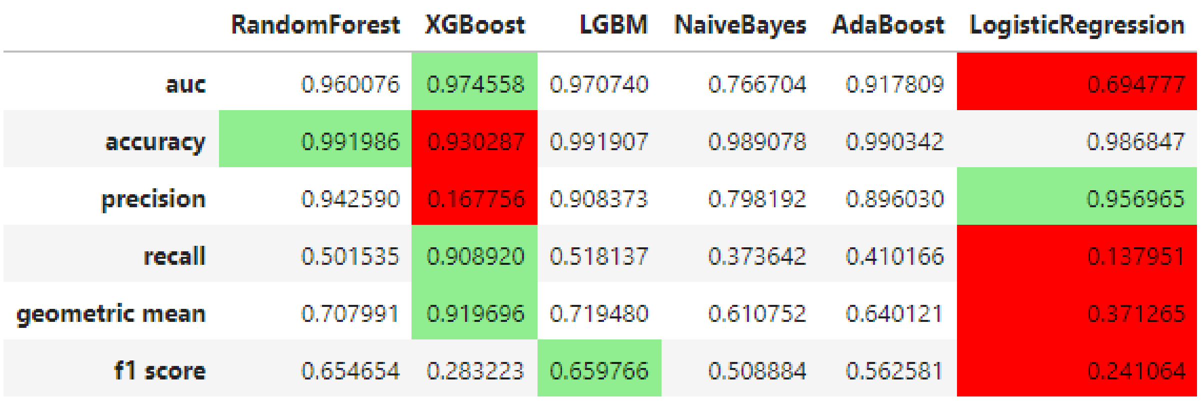


Figure 9: Precision-recall curves of cross-validations using all features

We can see that the decision tree based boosting methods were performing better, however it is also visible, that there is a significant trade-off between precision and recall. My expectation was something like this, because as we could see during the PCA, that even though most of the variance is explained by only two principal components, there are many good and bad samples that are pretty similar instances, and I believe this is the cause of such trade-off.

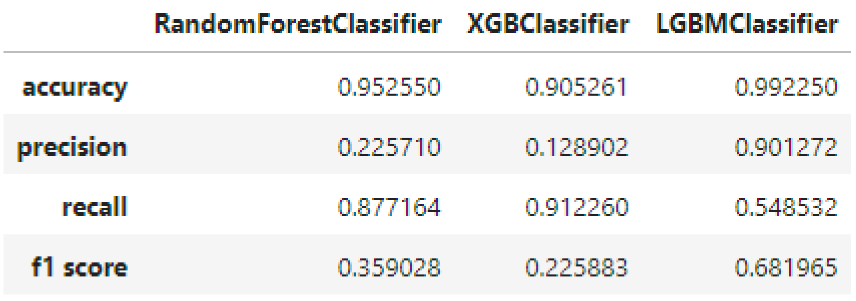


	RandomForest	XGBoost	LGBM	NaiveBayes	AdaBoost	LogisticRegression
auc	0.960076	0.974558	0.970740	0.766704	0.917809	0.694777
accuracy	0.991986	0.930287	0.991907	0.989078	0.990342	0.986847
precision	0.942590	0.167756	0.908373	0.798192	0.896030	0.956965
recall	0.501535	0.908920	0.518137	0.373642	0.410166	0.137951
geometric mean	0.707991	0.919696	0.719480	0.610752	0.640121	0.371265
f1 score	0.654654	0.283223	0.659766	0.508884	0.562581	0.241064

Figure 10: Evaluation metrics for the different algorithms

Based on the evaluation metrics I decided to continue working with Random Forest, XGBoost and LightGBM algorithms. In order to fine tune the model hyperparameters I used randomized search with cross-validation, because I could experiment with a broader range of parameters without increasing the computation time too much. For all three models I included the default values as well in the parameter lists, and I ran ten iterations with default 5 fold cross-validation. I monitored precision, recall and F1-score as scoring metrics and set the refit parameter to recall. After the first round of randomized search, I was able to identify some hyperparameters that could be fixed, and in case of others I needed to broaden the parameter space, and do a second round of search. Based on the results, I was able to narrow down most of the parameters, and the best estimators of Random Forest and XGBoost had promising recall scores above 80%. For all the details of the randomized search steps please refer to the *„data_mining_techniques.ipynb”* notebook.

For the best estimators after two rounds of randomized search I made model evaluations for each case individually. I used nearly the same approach as in the beginning, however I used repeated stratified kfold now with 5 splits and 2 repeats and furthermore I supplemented the framework with optimal decision threshold analysis, i.e. in each fold based on the precision-recall curve values I looked for the threshold resulting in highest F1-score.



	RandomForestClassifier	XGBClassifier	LGBMClassifier
accuracy	0.952550	0.905261	0.992250
precision	0.225710	0.128902	0.901272
recall	0.877164	0.912260	0.548532
f1 score	0.359028	0.225883	0.681965

Figure 11: Evaluation metrics of fine-tuned models

From the results we can see that XGBoost had the highest recall value over 90%, but on the other hand the precision was only around 13%, which was the lowest. As an opposite, the precision of the LightGBM model was above 90%, however its recall score was just above 50%. Finally we can say that the Random Forest is an intermediate model with a fair balance of precision over 20% and recall around almost 90%. It is also visible on the precision-recall curves (please see *Graph 4-5-6* in Appendix), that by moving the decision boundary, Random Forest and LightGBM models have similar potential to achieve higher, but still limited precision score given a recall score over 80%. Generally there is a clear and significant trade-off between these two metrics, and if we would like to detect most of the scrap cycles we need to consider that there will be a bunch of false positive predictions. Additionally, since recall is more important in our case, I made an extra round of threshold search using F-beta score. This metric was developed to be able to set priorities between precision and recall by adding the beta parameter into the equation: $F_\beta = \frac{(1+\beta^2)(precision*recall)}{(\beta^2*precision+recall)}$. The F1-score is actually a special case of F-beta score where beta equals to 1. The behaviour of this metric could be described as follows:

- When beta = 0, the F-beta score equation reduces to precision
- When beta = 1, the F-beta score equation balances precision and recall equally.
- When beta > 1, the F-beta score equation places more emphasis on recall.

Based on this I set my beta parameter to 4, meaning that recall is four times more important than precision.

In the end I evaluated the fine-tuned models on the test dataset, with the default 0.5 decision boundary and the other two thresholds obtained for F1 and F-beta score, and compared the results of every case.

model	**Random Forest**			**XGBoost**			**LightGBM**		
threshold	**default**	**F1**	**F4**	**default**	**F1**	**F4**	**default**	**F1**	**F4**
precision - 0	0.99821	0.99309	0.99778	0.99860	0.99260	0.99716	0.99302	0.99362	0.99786
precision - 1	0.22518	0.67758	0.25970	0.12744	0.75667	0.23105	0.91171	0.84084	0.26776
recall - 0	0.95293	0.99597	0.96222	0.90327	0.99744	0.95786	0.99919	0.99830	0.96356
recall - 1	0.88900	0.54976	0.86138	0.91818	0.51693	0.82282	0.54351	0.58363	0.86607
f1-score - 0	0.97504	0.99453	0.97968	0.94855	0.99501	0.97712	0.99609	0.99595	0.98041
f1-score - 1	0.35934	0.60702	0.39908	0.22381	0.61424	0.36079	0.68103	0.68901	0.40905

Table 1: Classification report for the different decision thresholds on the test set

I also added the confusion matrices of each test into the Appendix (*Graph 7-8-9*). The final conclusions will be summarized after the comparison with neural networks.

4.2. Neural Networks

In this section I describe two approches and the related results towards my task. Firstly I used conventional fully connected multi-layer perceprton architecture with supervised learning on the whole training dataset, and after that I experimented with autoencoder structures, but that time applied supervised learning only on the good samples of the training data and then used the reconstruction error values to differentiate between the two classes using the test data.

4.2.1. Multi-layer perceptron (MLP)

I created a class which builds a fully connected neural network for binary classification with three hidden layers using batch normalization and customizable activation function between the layers, but sigmoid after the final layer. My plan was to experiment with different combinations of activation functions, optimizers and learning rates. Since there is already a sigmoid activation function after the final layer used the Binary Cross-Entropy With Logits Loss function of torch, which is appropriate for the binary classification task, especially that it has a parameter where I can assign weights to the target classes in order to handle the class imbalance. I experimented with the following activation functions:

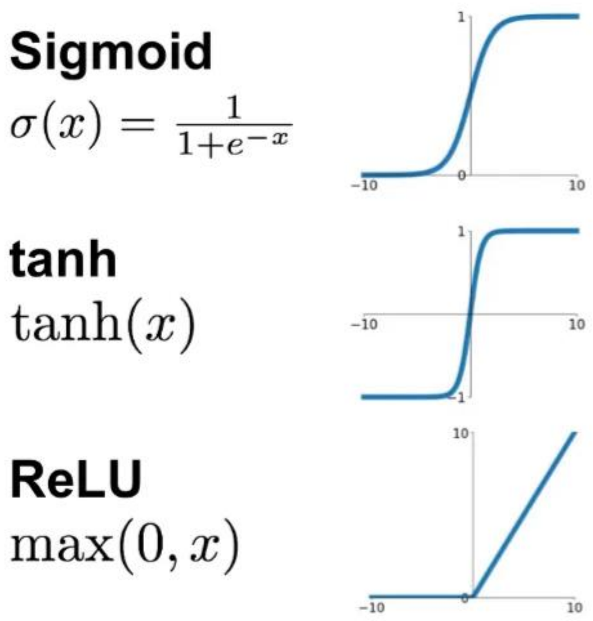


Figure 12: Visualization of the used activation functions

Furthermore I am going to use Adam optimizer with a learning rate of 0.001 and Stochastic Gradient Descent with a learning rate of 0.01 and momentum of 0.9. I also added a learning rate scheduler to the training loop, namely the Cosine Annealing LR, which gradually decreases the learning rate from an initial value to its minimum following a cosine curve pattern. During the training I was monitoring the loss, accuracy, precison, recall and F1-score for each epoch, and I save the model if the train recall and F1-score is higher than the former one. With each combination of the mentioned parameters I trained the networks for 10 epochs, and then evaluated their performance on a validation set.

optimizer	**Adam**			**SGD**		
activation	**tanh**	**relu**	**sigmoid**	**tanh**	**relu**	**sigmoid**
accuracy	0.985963	0.939200	0.974667	0.015270	0.015147	0.778458
precision	0.607477	0.151357	0.272620	0.015124	0.015147	0.039574
recall	0.207066	0.654214	0.403128	0.998262	1.000000	0.585578
f1-score	0.308855	0.245837	0.325272	0.029796	0.029843	0.074138

Table 2: Evaluation metrics for the different parameter combinations after 10 epochs

We can see that with this parameter settings of SGD, it was not performing well. On the other hand Adam seemed to be a good choice, so I decided to continue with that optimizer. As a next step I have initiated the same trainings but this time for 50 epochs, and made the same evaluation.

activation	**tanh**	**relu**	**sigmoid**
accuracy	0.940319	0.064937	0.963239
precision	0.180111	0.015310	0.184531
recall	0.827686	0.959166	0.417318
f1-score	0.295844	0.030139	0.255905

Table 3: Evaluation metrics after 50 epochs

Depending on the evaluation results, the neural network with tanh activation functions between the hidden layers was able to obtain an acceptable balance between precision and recall, however the values could be better.

Finally, I evaluated the models of each activation function on the test set, using the final state and the best model state regarding recall and F1-score obtained during the 50 epochs.

state	**after 50 epochs**			**best recall and F1-score during 50 epochs**		
model	**tanh**	**relu**	**sigmoid**	**tanh**	**relu**	**sigmoid**
precision - 0	0.997709	0.997501	0.992881	0.997145	0.988545	0.990945
precision - 1	0.026042	0.115174	0.443777	0.183505	0.015349	0.189663
recall - 0	0.464466	0.899107	0.989609	0.943564	0.050511	0.972219
recall - 1	0.930693	0.853570	0.538822	0.824388	0.961959	0.422616
f1-score - 0	0.633853	0.945751	0.991242	0.969615	0.096111	0.981493
f1-score - 1	0.050667	0.202961	0.486703	0.300190	0.030215	0.261824

Table 4: Classification report for the neural networks on the test set

4.2.2. Autoencoder neural networks

The other type of neural network architecture I investigated is the autoencoder, which consist of two main parts, an encoder and a decoder layer. The encoder maps the input data into a lower-dimensional representation by capturing the most important features of it, and then the decoder takes this latent representation and tries to reconstruct the original data. The performance can be measured by the differences between the original and the reconstructed values, most commonly by Mean Squared Error (MSE) or Mean Absolute Error (MAE), which can be used as loss functions during the training.

I experimented with two types of autoencoder architectures, one with fully connected layers with tanh activations and another one with 1D Convolution layers using kernel_size=3, stride=1, padding=1 and relu activation functions. My plan was to train the networks on a dataset that contains only good samples and then check the performance on a test set which contains all the scrap cycles and also good samples in the same amount. The idea was to use a meaningful threshold value based on the losses obtained during the training, and use it as a decision boundary for the test dataset, i.e. if the loss obtained on a test sample exceeds this threshold – meaning that the instance differs significantly from the normal process the model have learnt – that instance is classified as scrap cycle.

For both architectures I created training loops and trained them for 20 epochs minimizing the L1Loss function of torch.nn (which is actually the Mean Absolute Error MAE), and using Adam optimizer and Cosine Annealing LR scheduler. During training I saved the parameters of the models that had the lowest training loss and plotted them.

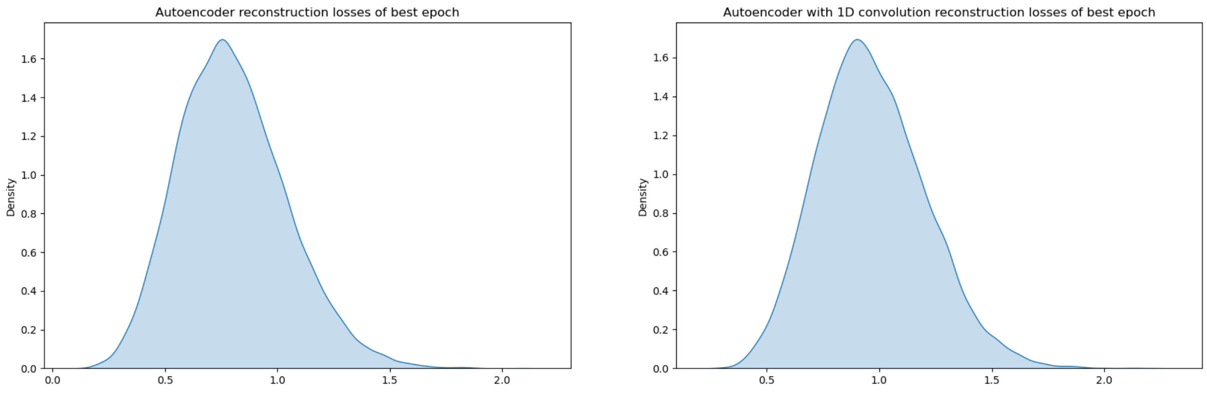
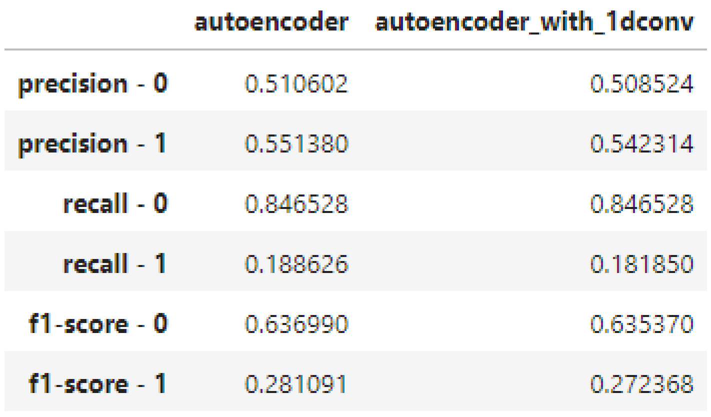


Figure 13: Reconstruction losses of the two autoencoder models

Both looks like normally distributed, so I decided to set the threshold for classification boundary to three standard deviation distance from the mean $\mu + 3\sigma$, and evaluated model performances.



	autoencoder	autoencoder_with_1dconv
precision - 0	0.510602	0.508524
precision - 1	0.551380	0.542314
recall - 0	0.846528	0.846528
recall - 1	0.188626	0.181850
f1-score - 0	0.636990	0.635370
f1-score - 1	0.281091	0.272368

Figure 14: Autoencoder classification reports

The performance of the models are pretty similar and the results are not so promising. In order to get more insight, I made some visualizations of the loss values obtained during the evaluation from the test set. Because of the outliers, the plots of the losses by classes was not so informative, but at least it was visible that the majority of the losses fall below the value of 10, so I could reduce the interval and create the following plot:

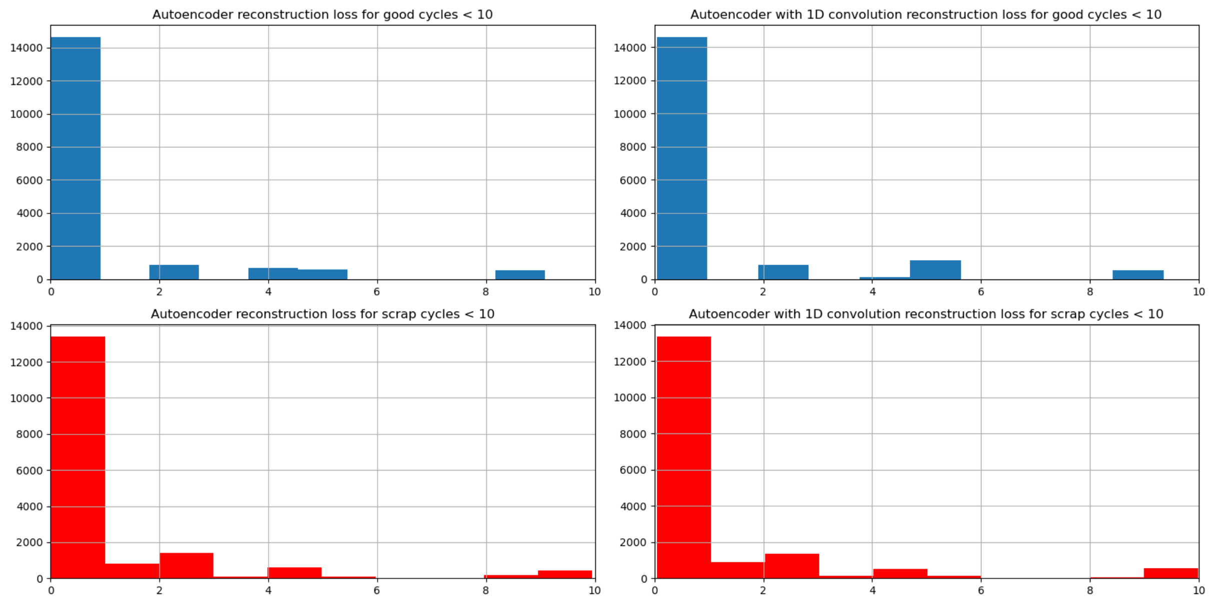


Figure 14: Reconstruction error distribution under 10 (blue = good cycles, red = scrap cycles)

We can see that the good and the scrap cycles have very similar values of reconstruction error inside this interval, and the majority is lower than 1, so finally I reduced the interval further to observe the distribution of the losses between 0 and 1.

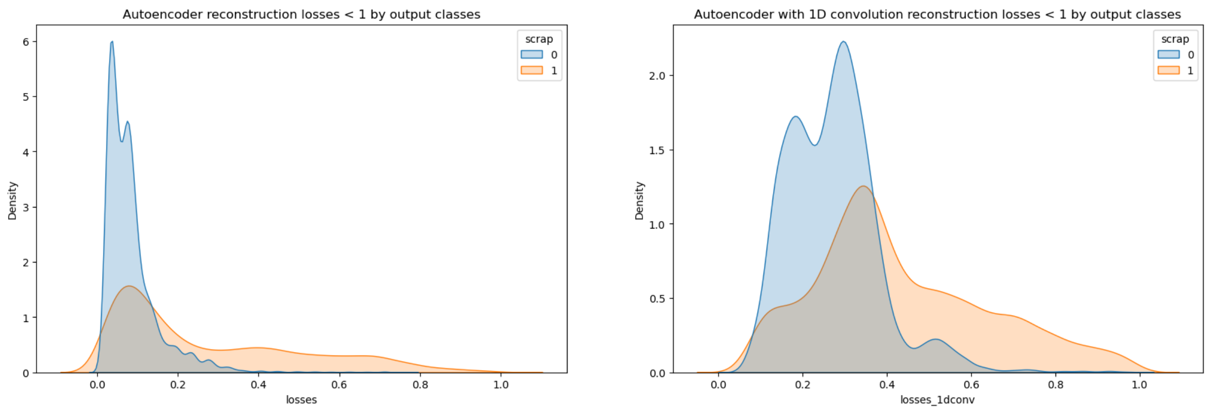


Figure 15: Reconstruction error distribution under 1 (blue = good cycles, orange = scrap cycles)

Looking at this small interval, we can slightly differentiate between the good and bad cycles, but still there are many samples which are pretty similar, and furthermore comparing these values to the lowest training losses, these are even below the mean value in both cases.

5. Summary

Considering the three approaches I tried, the last one using the autoencoder neural network architectures brought the worst results with limited ability to detect the scrap cycles of this injection molding process from machine parameters. The conventional fully connected networks provided better performance especially the one with tanh activation functions between the hidden layers, but even this was outperformed by the decision tree based methods, which I tried first.

Generally we can state, that in case of this dataset, many of the samples from the different output classes are pretty similar (on machine parameter level), so after a certain level of desired scrap detection ratio, there is a significant trade-off between the precision and recall metrics, meaning that there will be more pseudo-scrap cycles with increasing recall. Based on the actual results of the fine-tuned model which can be seen in *Table 1* the three instances that I marked with green are able to detect more than 85% of the scrap cycles, but in return every 4-5th detected cycles are actually bad

I have not yet prepared the business case of this application – i.e. if we use one of these decision tree based models for scrap detection, what are the profits on one hand by eliminating the costs of cavity pressure sensors installation and maintenance, and the losses on the other hand by the increased pseudo-scrap ratio and the non-detected defective parts – but in my opinion for this specific case it is not a viable scenario. However I believe that this kind of analysis and approaches could be tried at a lot of machine-tool-product combinations, hopefully with more promising results, and on top of this the obtained feature importances of the best fit models (see *Graph 10-11-12* in Appendix) surely helps us to at least put more focus on controlling the most critical input parameters of the process.

References

[1] A Gentle Introduction to Threshold-Moving for Imbalanced Classification –

https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/

[2] Step-By-Step Framework for Imbalanced Classification Projects –

https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/

[3] Tour of Evaluation Metrics for Imbalanced Classification –

https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

[4] Receiver Operating Characteristic (ROC) with cross validation –

https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html

[5] ROC vs Precision-Recall Curves with N-Folds Cross-Validation –

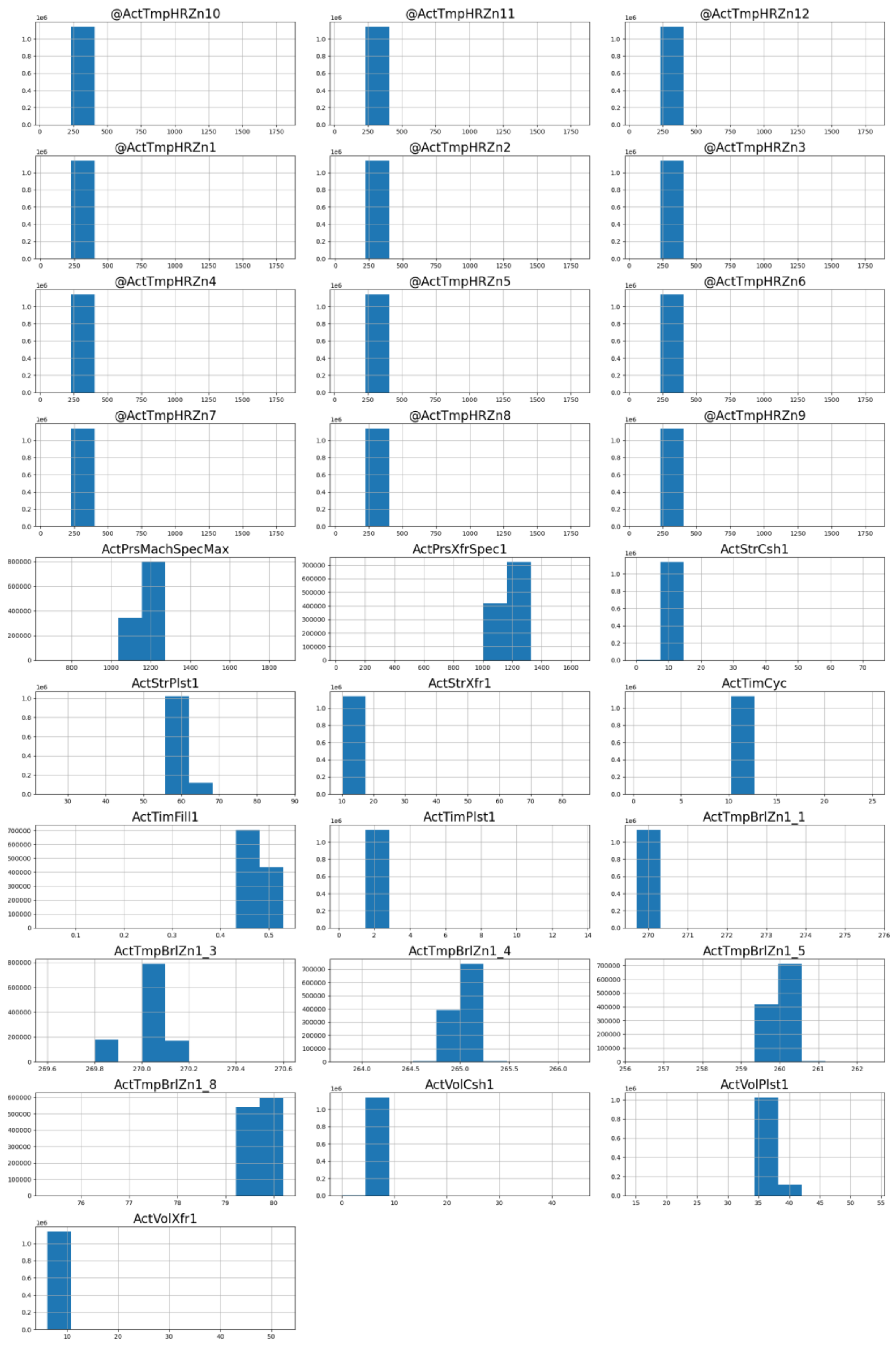
https://amirhessam88.github.io/roc-vs-pr/

[6] ROC Curves and Precision-Recall Curves for Imbalanced Classification –

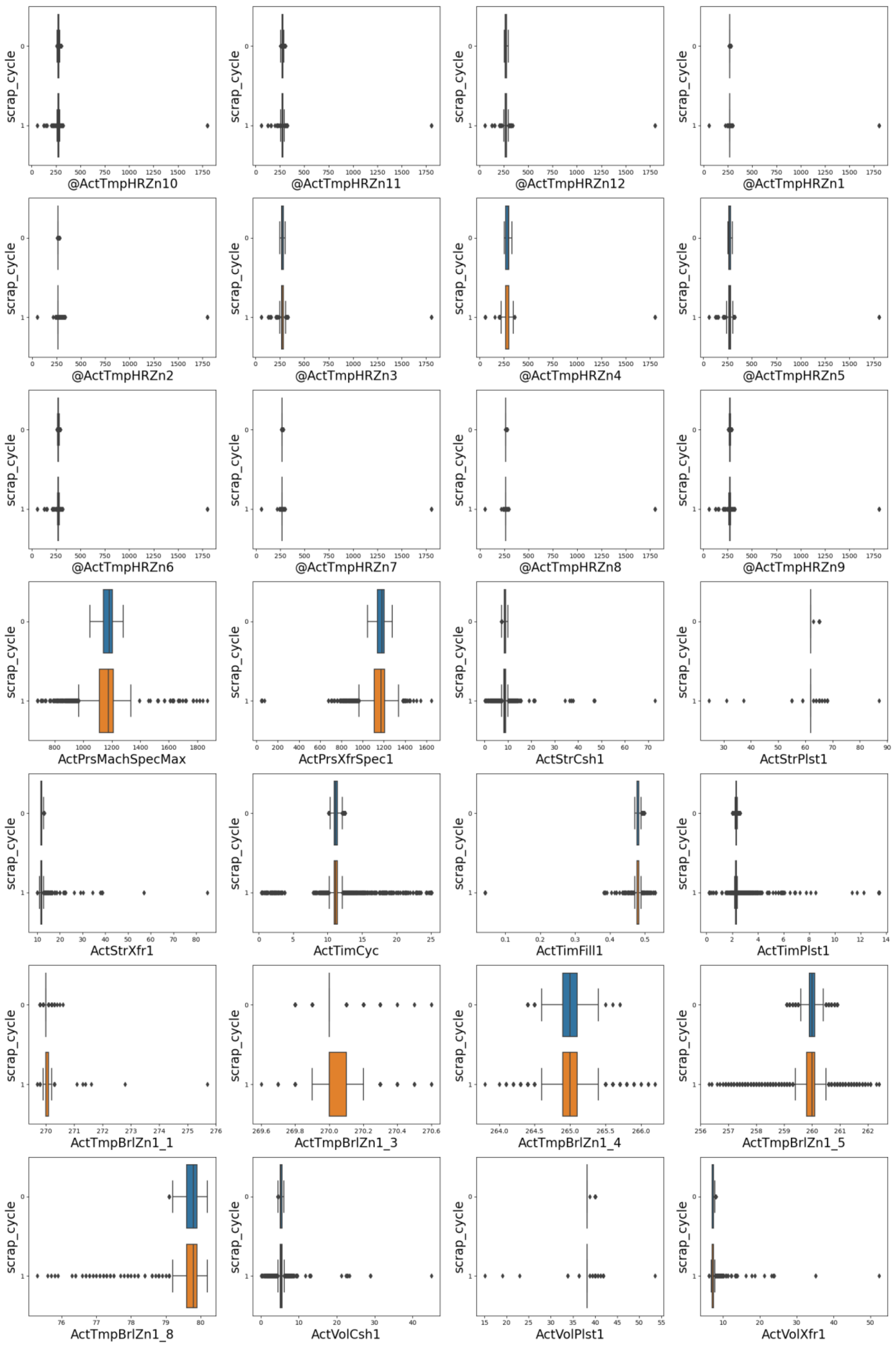
https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/

Appendix

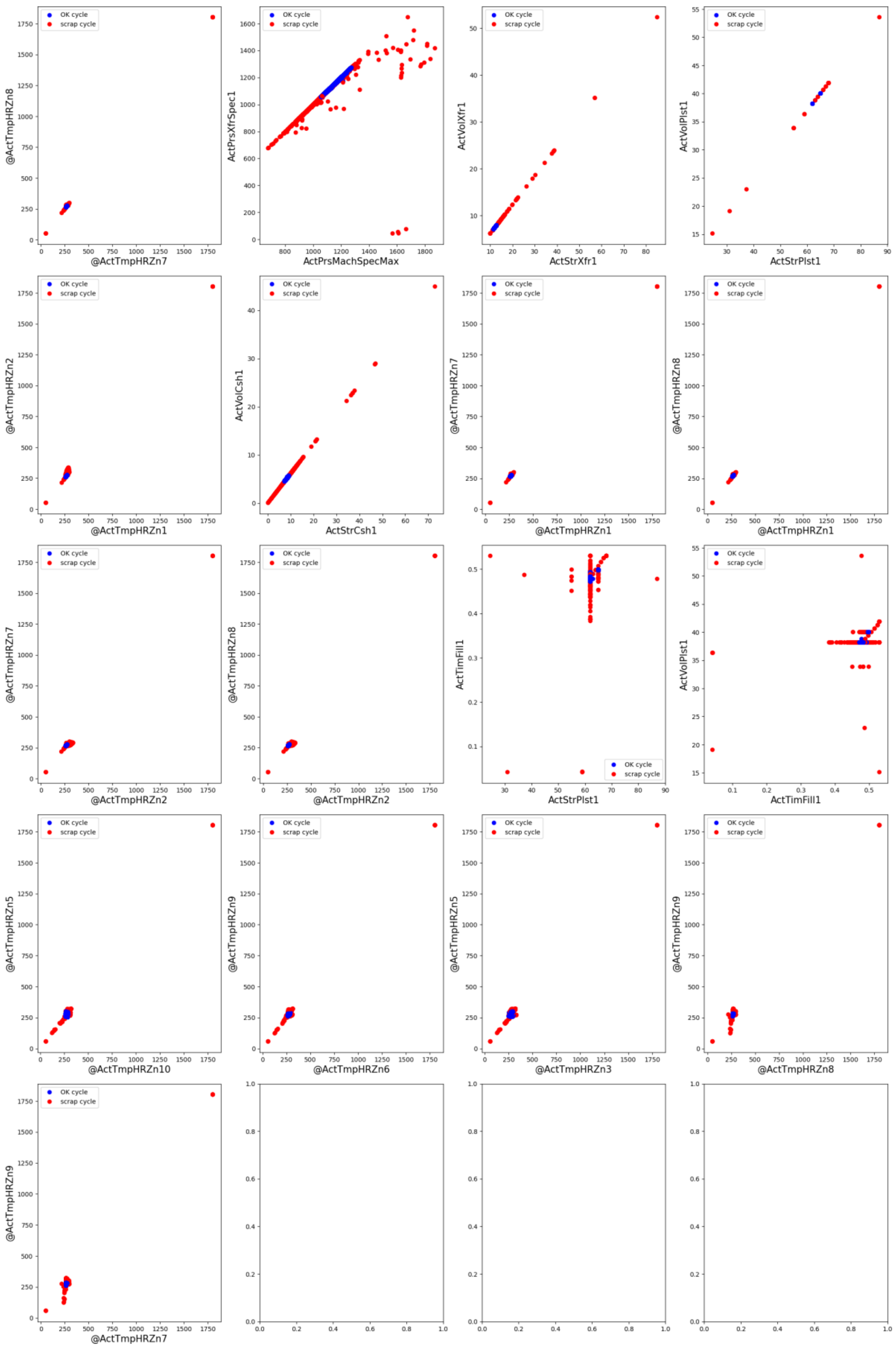
1. Exploratory data analysis graphs



Graph 1: Distribution of predictor features

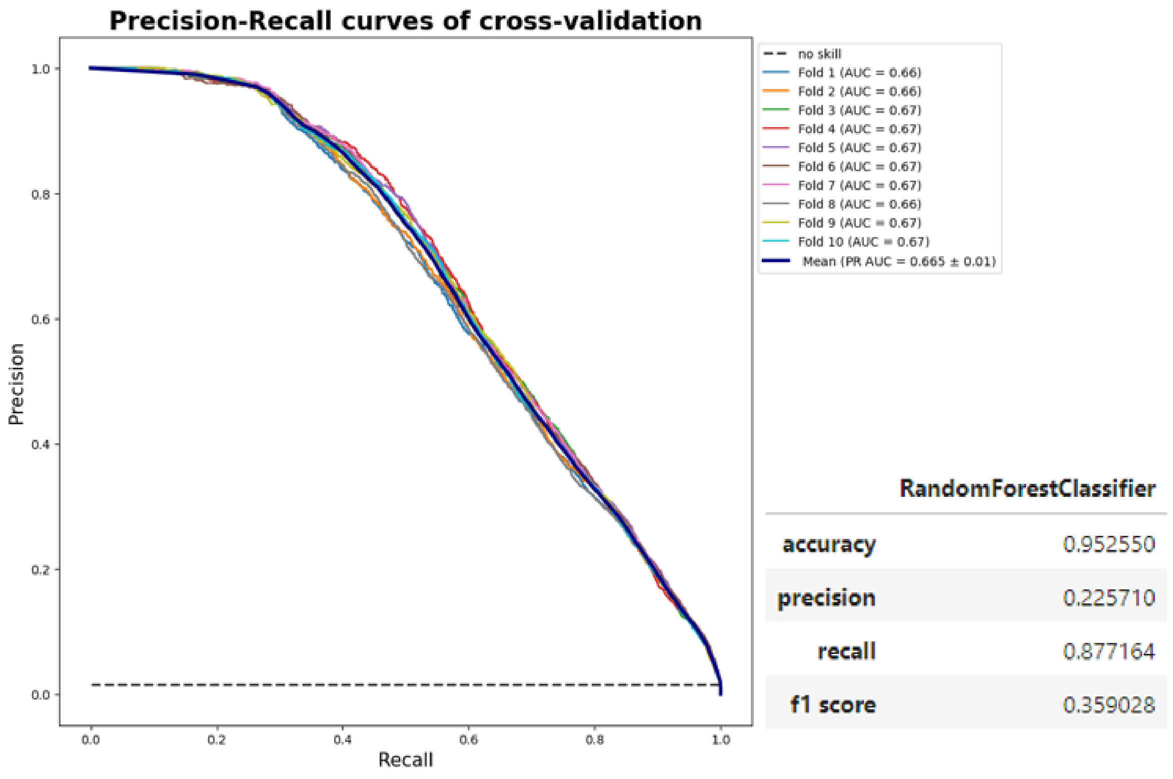


Graph 2: Grouped boxplots of predictor features by output class



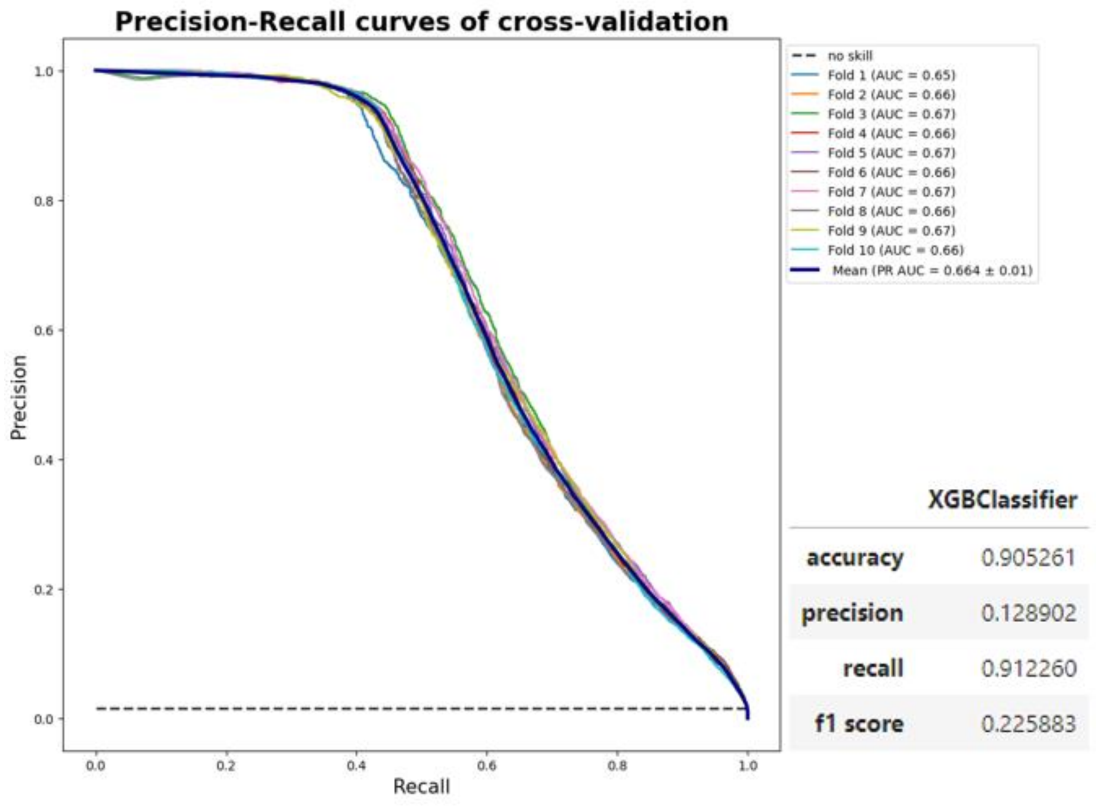
Graph 3: Scatterplots for feature pairs with correlation value over 0.8

2. Graphs of modelling phase



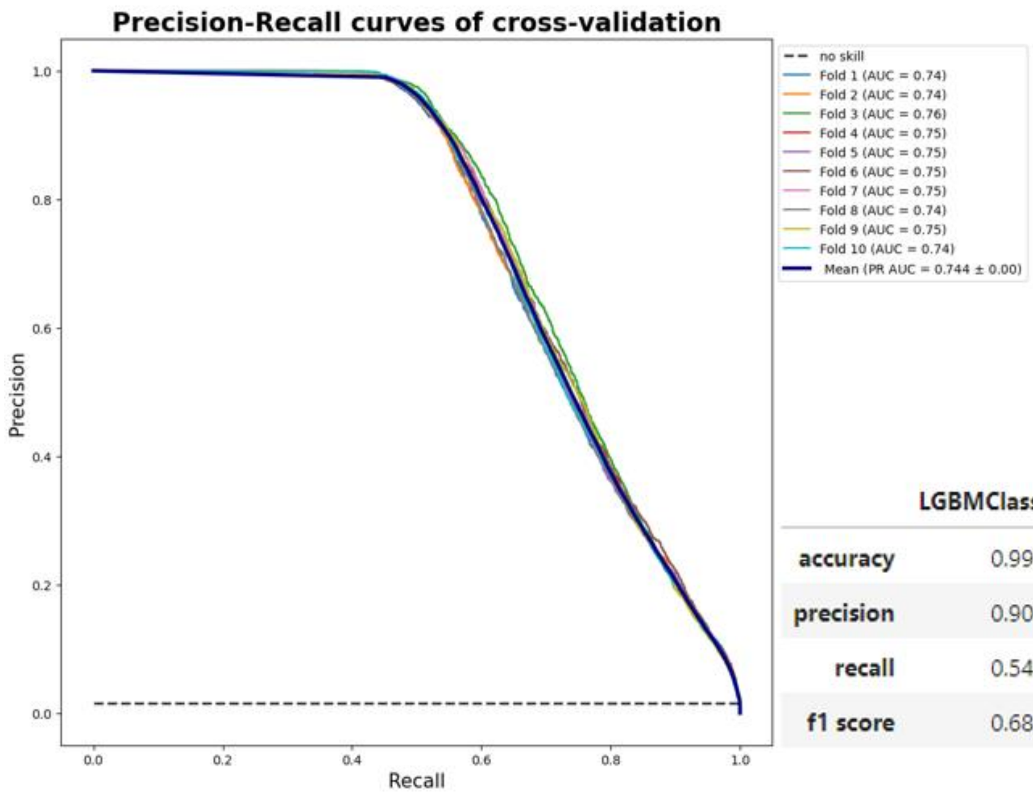
	RandomForestClassifier
accuracy	0.952550
precision	0.225710
recall	0.877164
f1 score	0.359028

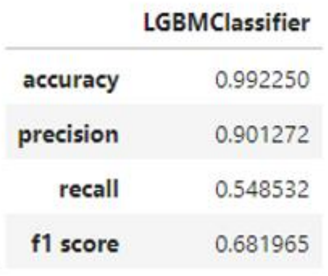
Graph 4: Random Forest cross-validation results



	XGBClassifier
accuracy	0.905261
precision	0.128902
recall	0.912260
f1 score	0.225883

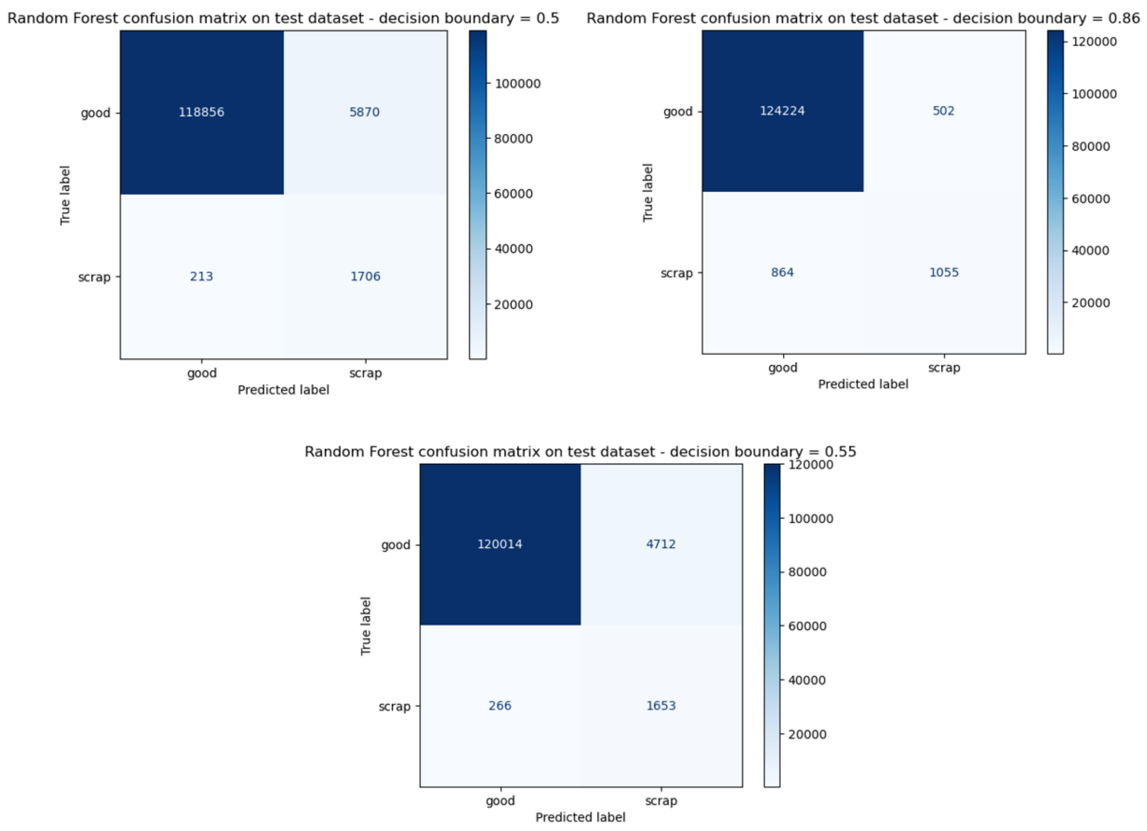
Graph 5: XGBoost cross-validation results



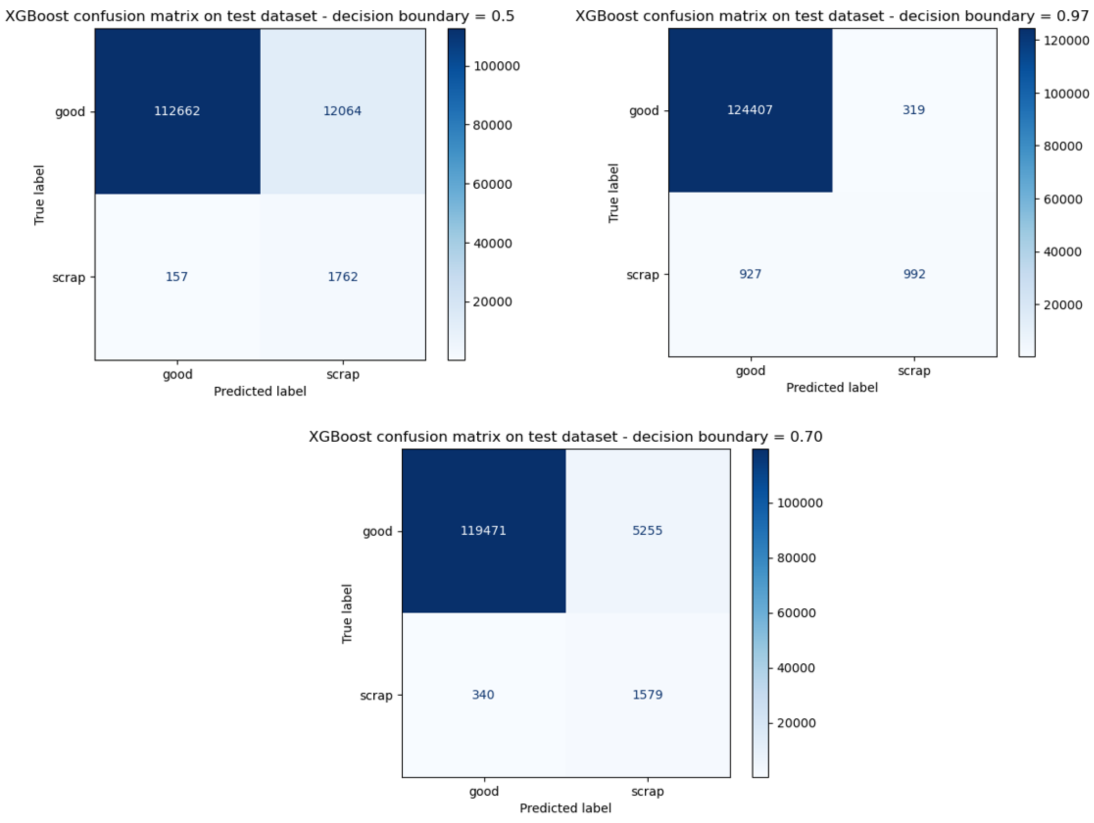


	LGBMClassifier
accuracy	0.992250
precision	0.901272
recall	0.548532
f1 score	0.681965

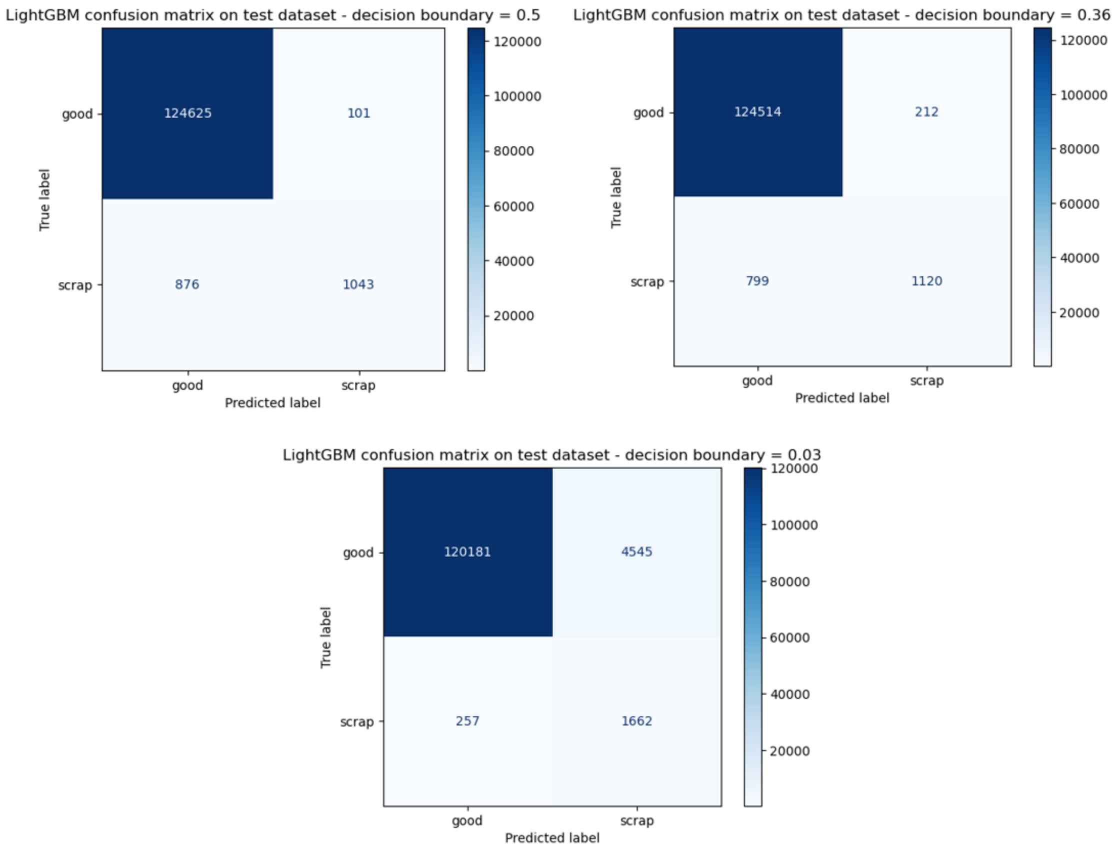
Graph 6: LightGBM cross-validation results



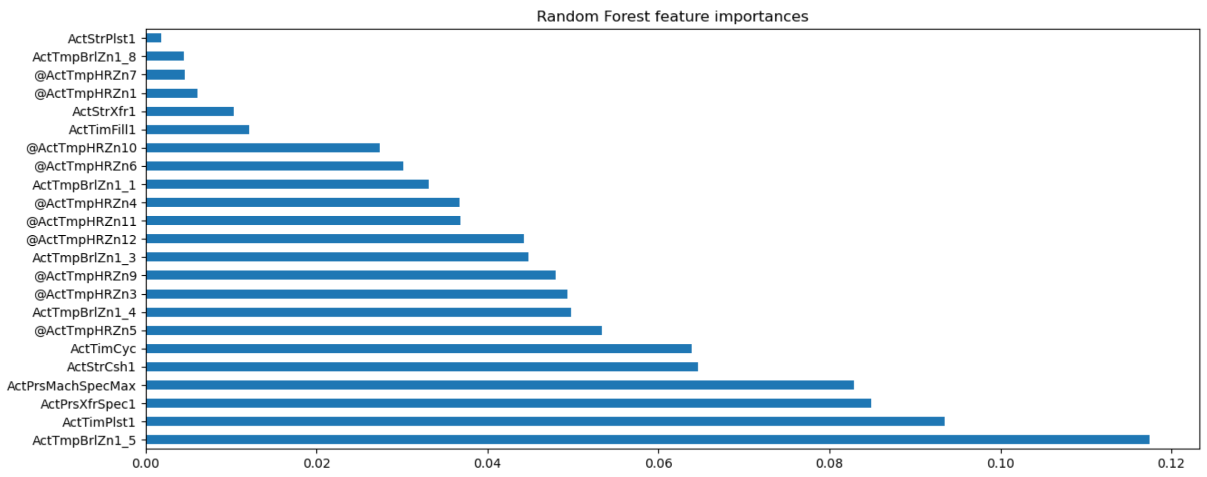
Graph 7: Random Forest confusion matrices on test data



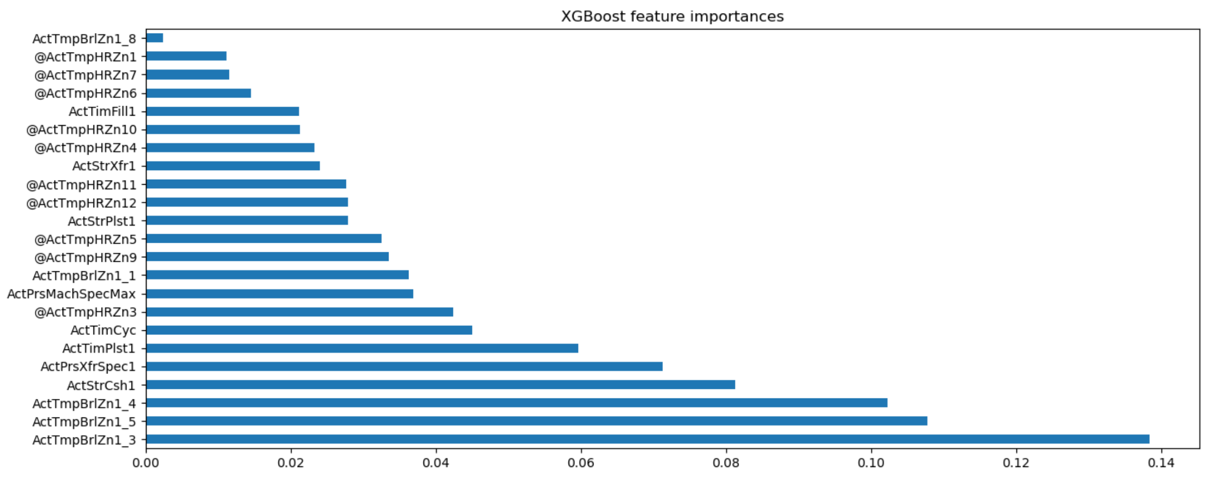
Graph 8: XGBoost confusion matrices on test data



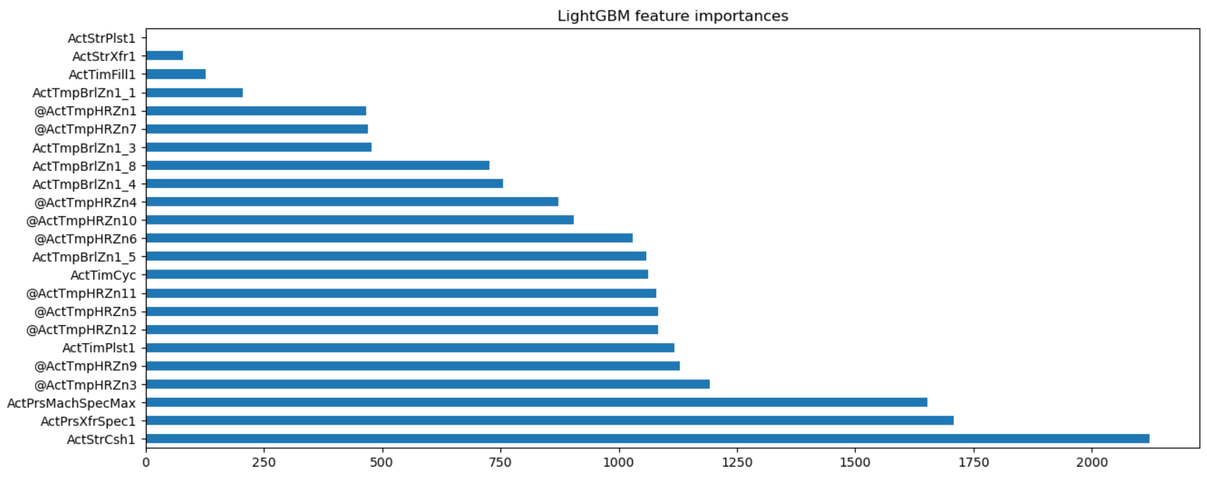
Graph 9: LightGBM confusion matrices on test data



Graph 10: Fine-tuned Random Forest feature importances



Graph 11: Fine-tuned XGBoost feature importances



Graph 12: Fine-tuned LightGBM feature importances