Data Mining and Machine Learning Project

Krisztián Szuppinger
szuppinger.krisztian@hallgato.ppke.hu

Contents

I. Introduction

The dataset provided for the project consists of 60000 observations for 170 features. The data describes system failures for Scania trucks in everyday usage. The output class is a binary variable which is positive for trucks for which the system failure is associated with the air pressure system (APS) and negative for those where the failure comes from other components of the vehicle. The feature names are anonymized and the attributes consist of either single numerical values or bins of histogram. The task is to predict the value of the output class with maximal balanced accuracy (BA): $\text{BA} = 1/2 \cdot \frac{TP}{TP+FN} + \frac{TN}{TN+FP}$.

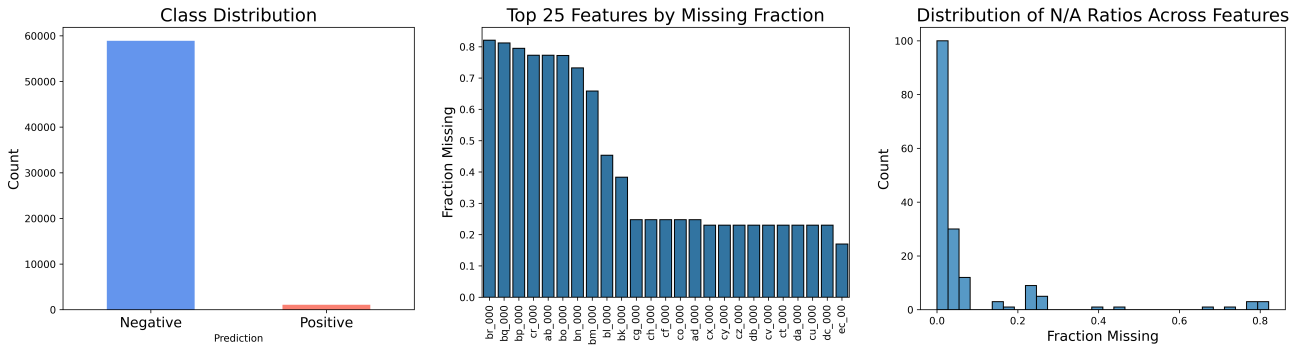
The following report documents my solution for the task above detailing the data exploration and preprocessing steps as well as the used algorithms, evaluation metrics, the trained models and the obtained results. In the end, a feature importance chart derived from the best-performing model is also presented.

II. Exploratory Data Analysis and Preprocessing

As the first step of the Exploratory Data Analysis (EDA), the size of the training and testing sets were determined to be (60000×170) and (16000×170), respectively.

Secondly, since the task involved predicting a binary class for failure of a specific system among many, severe class imbalance was suspected. After investigating this, we found that the negative class had 58914 samples, while the positive one consisted of only 1086 observations resulting in a positive to negative ration of p:n ≈ 54.25. This information will be used further in the workflow and is demonstrated visually in Figure 1a.

Finally, missing values (N/A) of the dataset were inspected, which revealed that there were several attributes for which no data was available for most of the training instances. The most severe feature with respect to data missingness was identified to be "br_000" with 49259 missing values out of 60000 observations. Figure 1b shows the 25 attributes with the highest N/A count, while Fig. 1c depicts the ratio of missing values across features. From the latter, we may conclude that most of the features contain only a few N/A-s, however a non-negligible number of attributes have N/A fractions around and even above 0.8.



(a) (b) (c)

Figure 1: Figure 1a depicts the class imbalance in the training dataset, while Figures 1b and 1c, illustrate the number of missing values in the dataset. Figure 1b shows the top 25 features with the most N/A values. The x-axis shows the attributes, while the y axis depicts the missing fraction value defined as $\frac{\#\text{N/A}}{\#\text{non-N/A}}$. It is clear that some features suffer from serious N/A counts. In Figure 1c the fraction of missing values are given on the x axis and the y axis gives the number of attributes in the dataset that possess the corresponding missing value fraction. The plot communicates that while most attributes have low missing fractions (i.e., are on the left side of the plot), a non-negligible number of attributes have missing fraction values around 0.8.

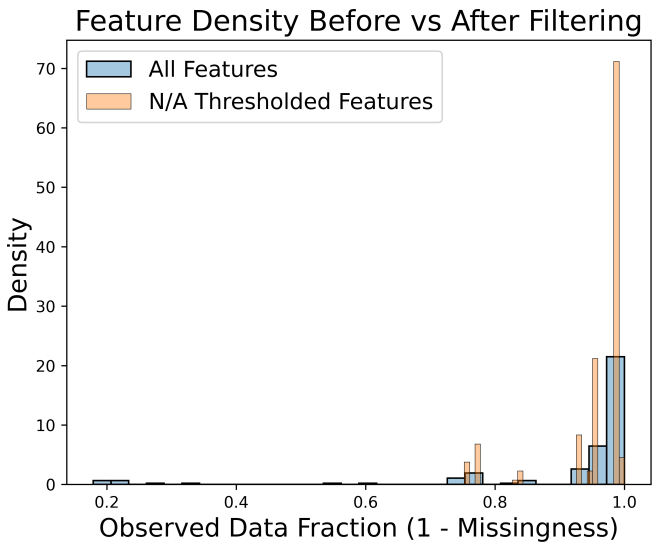


Figure 2: The figure depicts missing value densities for the two distinct preprocessing approaches. The x axis displays the fraction of actual data (i.e., not N/A) in a given column of the dataset, while the y axis shows the density of columns corresponding to the given observed data fraction. The orange histogram visualizes the filtered dataset, while the blue one shows the original one. The illustration shows that the density of information (i.e., non-N/A values) is largely increased in the filtered dataset as compared to the non-filtered one as a significantly higher number of features cluster close to an observed data fraction of one. The high missing fraction features of the original dataset are also visible at the left side of the plot.

III. Data Preprocessing

Based on EDA, four algorithms have been selected to train. Firstly, since the number of missing values is extreme in certain features and the number of attributes is also quite large, XGBoost (XGB) will be our main algorithm as this can handle both of these problems internally. We train two XGB models which differ in the preprocessing steps which are detailed below. In addition to the XGB models, we train a logit and a support vector machine (SVM) classifier, as well as a decision tree (scikit-learn $\approx$ C4.5) to assess how these simpler algorithms compare to the ensemble boosting method of XGB.

For two XGB model, the logit and SVM classifiers as well as the decision tree (Dec-Tree), the first preprocessing step excludes attributes that have a missing value fraction (i.e., $\frac{\#\text{N/A}}{\#\text{non-N/A}}$) greater than 30%. This reduces the number of attributes from 170 to 160. However, in order to ensure that we do not discard valuable information by neglecting these N/A-rich columns, we also train another XGB model (as this algorithm can handle N/A-s internally) that receives all 170 features. Figure 2 shows the the information desnity of features before and after filtering and highlights that after filtering (orange) most features have a near zero missing fraction values.

All following steps of the data preparation process (as well as the training and hyperparameter tuning

stages) are performed using scikit-learn's *pipeline* class to ensure no data-leakage and a clear and understandable code structure. The preprocessing steps were performed using methods of the scikit-learn library.

A. Preprocessing for Feature Selected Pipeline

For two XGB models, the SVM and logit classifiers and the decision tree (collectively referred to as the *feature-selected pipeline* from here on out), we handle N/A values using the SimpleImputer function, replacing all missing values with the median of the respective attribute. The median is chosen instead of the mean as outliers may distort the mean value and thus - after imputation - the processed data's distribution as well. This is followed by standardizing attribute values using the StandardScaler method. Finally, feature selection is performed by the SelectKBest method with the scoring function as the ANOVA F scores and a default k of $k = 40$ which will be tuned later as a hyperparameter (HP) of each model.

B. Preprocessing for N/A Safety XGB Pipeline

We also train an an XGB model that receives all 170 attributes of the dataset to begin with, however this number is tuned while the training occurs as a HP of the model. We use the SelectFromModel method and set the model as another XGB classifier with 200 estimators 4 maximum depth and a learning rate of 0.1. The threshold for selection is the median value. In this case, therefore, feature selection occurs according to an XGB model and this information is fed into the main XGB model via the defined pipeline (called as *n/a-safety xgb pipeline* from now on).

IV. Training

As mentioned previously four algorithms were used for training, resulting in a total of 9 models. This includes the model from the n/a-safety xgb pipeline as well as two versions of each model from the other pipeline. These latter models differ in their HP tuning setup: one model is trained with a maximum selectable feature count of 160 and another one with a maximum of 120.

In this section, we will first provide a general description of the training structure that is the same for all models and then discuss the individual models themselves.

A. General Training Structure

The general structure of the training architecture is the same across all nine models, with the random seed set as 42 for the whole notebook. Before training begins, the provided labeled dataset is split into training and validation sets with a $80:20$ training:validation ratio. This results in 48000 training and 12000 validation instances. Then the steps are as follows.

We first define a scikit-learn Pipeline object with the appropriate preprocessing steps (such as SimpleImputer or StandardScaler) and the classifier with its corresponding parameters. This is followed by the definition of an n-dimensional search space across different parameter values for the given classifier. Here, n is the number of parameters the values of which were specifically set in the pipeline step for the given classifier (i.e., parameters that were not simply left as their default values). The pipleine and search space objects are then passed to the BayesSearchCV method which will perform a bayesian search to find the hypeprparameter settings that maximize the balanced accuracy score among those that are sampled. This avoids an extensive grid search, thus shortening run times substantially. Finally, we retrieve the model with the best found hyperparameters.

B. Specific Models

XGB SelectKBest Models: The XGB models with the SelectKBest-type feature selections (XGB-SKB models) are trained first. Preprocessing in the pipeline occurs as described in Section III-A. The initial k-value is set to $k = 40$ for both the 120 and 160 XGB-SKB models and the found best feature number is always the maximal one. The evaluation metric of the XGBClassisfier object is set to logloss to prevent any discrete jumps that may be caused by the balanced accuracy score here. We also provided a scale_pos_weight argument to account for the severe imbalance between the number of the positive and the negative output classes. The search space intervals of BayesSearchCV (other than the k selection interval) are found in the notebook and are not detailed here for brevity. However, the HP values of the best-performing models are presented in Table I.

XGB SelectFromModel Setup: Except for the feature selection method, this model uses the same parameters as the XGB-SKB models. Feature selection is performed here an XGBClassifier object as described in Section III-B

Logit: Feature selection and preprocessing are as described in Section III-A. The penalty score of the classifier is set to 12 with a maximum iteration of 500 and balanced class weights. The search space for bayesian optimization consists of the k-value and the C parameter ranging between $[20, 120$ or $160]$ and $[10^{-3}, 100]$, respectively. A 5-fold cross-validation is performed with 40 sampled points.

SVM: Two SVM models are trained both with the SelectKBest method, however with different maximal selectable feature numbers. In the Bayesian cross validation search, only the k-value and the C parameters are tuned with the intervals of $k \in [20, 120$ or $160]$ and $C \in [10^{-3}, 1]$. Here, the number of sampled parameters (n_iter) is set to n_iter $= 15$ in order to reduce run time. The cv parameter is set to cv $= 4$ resulting in a 4-fold cross-validation.

Dec-Tree: The preprocessing here differs from what is described in Section III-A in that no scaling is applied. Bayesian search is performed for the best k-value maximal depth (md) and minimal leaf (msl) and sample split (mss) parameters with the following ranges $k \in [20, 120$ or $160]$, $md \in [2, 12]$, $msl \in [1, 50]$, and $mss \in [2, 50]$. The number of sampled parameter settings is 50 with a 5-fold cross validation.

V. Results

The balanced accuracy scores of all nine trained models are above 90%, on the validation set, showing that the constructed pipelines work efficiently. The model that was used for the prediction of the test set is chosen to be the model that performs best on the validation set. This is the XGB-SKB$_{120}$ model, which is retrained on the whole training set (i.e., training + validation) to achieve maximum prediction strength. The achieved Kaggle score for this best model is 0.95787 (dropping from 0.9572).

In this section, we will first present the Bayesian search-tuned hyperparameters for all nine models and then present detailed statistics of the performance of all models. Additionally, we will visualize how the models' predictions compare via confusion matrices as well as by plotting precision-recall curves.

A. Optimized Hyperparameter Settings

The tuned hyperparameters for the XGB models are presented in Table I. As mentioned previously, for the XGB-SKB models, the k-value is always chosen as the maximal one. The learning rates are similar across the XGB-SKB$_{120}$, XGB-SKB$_{160}$, and XGB-SFM models with values of 0.070335, 0.067242, and 0.050833, respectively.

Table I: Tuned hyperparameters of the XGB models. The k-value denotes the number of features retained by the bayesian search and is not an intrinsic parameter of the XGBoost algorithm. The remaining hyperparameters control tree depth and complexity (max_depth, gamma, min_child_weight), learning dynamics (learning_rate), and stochastic regularization via row and feature subsampling (subsample, colsample_bytree). The maximal k-value is selected for both XGB-SKB models, however, as depicted in Table III, the performance of the two models differs, with XGB-SKB$_{120}$ achieving higher scores.

Model	k-value	colsample_bytree	gamma	learning_rate	max_depth	min_child_weight	subsample
XGB-SKB$_{120}$	120.0	0.5	5.0	0.070335	3	10	0.5
XGB-SKB$_{160}$	160.0	0.5	0	0.067242	3	10	1.0
XGB-SFM	-	0.5	3.93	0.050833	4	10	1.0

The hyperparameters of the best versions of the simple classifiers are provided in Tables IIa and IIb. In this case, the k-value varies more across models. For the Logit classifiers, it is again always the maximal value, however, for SVM$_{120}$ it is k-value $= 102$, while for SVM$_{160}$, it is only 65. This is probably because the Bayesian search algorithm does not find a global optimum, but rather chooses the best settings across those discovered. Therefore, since in this case, another C value is found as the optimal one, the corresponding k-value differs as well. For the decision tree, the k-values are even lower at k-value $= 46$ for Dec-Tree$_{120}$ and k-value $= 49$ for Dec-Tree$_{160}$. The C parameter - the denominator weight of the penalty term that the model optimizes during training - is found to be low for each model in Table IIa, showing that a higher regularization (and therefore lower overfitting) is favored.

Table II: Optimized hyperparameters of the simple classifiers. The k-value here again is not an intrinsic parameter of either algorithms, but rather a parameter of the whole pipeline and as opposed to the XGB models (Table I), its optimized value is not always the maximal one. The C regularization parameter is found to be low across all models of Table IIa resulting in lower risks of overtraining. For the decision tree models, md, msl, and mss denote the maximum tree depth, minimum samples per leaf, and minimum samples required to split an internal node, respectively.

(a)

HP	Logit_{120}	Logit_{160}	SVM_{120}	SVM_{160}
k-value	120	160	102	65
C	0.00506	0.00116	0.00103	0.00248

(b)

Model	k-value	md	msl	mss
Dec-Tree_{120}	46	5	50	50
Dec-Tree_{160}	49	5	34	50

B. Evaluation Metrics

The best model according to the evaluation metric of the balanced accuracy (BA) score is the XGB-SKB_{120} model with a validation score of 0.9510. The second and third models are the XGB-SKB_{160} and XGB-SFM models, respectively. However, and even though they lack behind, the simpler models (Logit, SVM, Dec-Tree) also achieve exceptional results, with the best being Dec-Tree_{160} achieving a BA score of 0.9396 using only 49 features out of 160 possible ones. This is an extraordinary reduction in complexity compared to the XGB models for a relatively low BA loss of only 0.0114. Additionally, the model outperforms both the SVM and Logit classifiers while also using less features. Further, detailed performance statistics of the results obtained from all nine trained models are provided in Table III.

Table III: Performance comparison of all nine evaluated classifiers on the validation set. Among all approaches, the XGB-SKB_{120} model achieves the highest BA (0.9510), followed by XGB-SKB_{160} (0.9477) and XGB-SFM (0.9462), indicating that tree-based ensemble methods consistently provide the strongest overall performance. Nevertheless, the simpler white-box models remain highly competitive: in particular, Dec-Tree_{160} reaches a BA of 0.9396 while relying on only 49 selected features, representing a substantial reduction in model complexity compared to XGBoost with a marginal BA decrease of only 0.0114. Logistic regression and linear SVM models also perform above 0.9, achieving BA values above 0.91 while offering improved interpretability. The run times for each algorithm are also shown and, interestingly - and highlighting the curse of dimensionality - they are the longest for the two SVM models.

Model	Sensitivity	Specificity	BA	Precision	F1-score	Success Rate	Error Rate	Run Time [sec]
XGB-SKB_{120}	0.9355	0.9665	0.9510	0.3395	0.4982	0.9659	0.0341	114.6
XGB-SKB_{160}	0.9309	0.9644	0.9477	0.3253	0.4821	0.9638	0.0362	127.4
XGB-SFM	0.9217	0.9708	0.9462	0.3676	0.5256	0.9699	0.0301	277.3
Logit_{120}	0.8848	0.9751	0.9300	0.3959	0.5470	0.9735	0.0265	72.4
Logit_{160}	0.8710	0.9775	0.9242	0.4163	0.5633	0.9756	0.0244	85.5
SVM_{120}	0.8618	0.9775	0.9196	0.4137	0.5590	0.9754	0.0246	722.3
SVM_{160}	0.8525	0.9755	0.9140	0.3903	0.5355	0.9732	0.0268	1116.8
Dec-Tree_{120}	0.9401	0.9434	0.9368	0.2342	0.3750	0.9433	0.0567	106.1
Dec-Tree_{160}	0.9309	0.9482	0.9396	0.2488	0.3926	0.9479	0.0521	114.5

The precision scores in Table III remain low across all models which is probably due to the severe class imbalance. The most important score, for detecting potentially dangerous failures, the sensitivity values are high for all models and this, along with the objective score, BA, is highest for the XGB-SKB_{120} model as well. All models achieve high success rates and low error rates, however run times vary substantially. The SVM models, in particular, perform especially poorly in this regard, which is probably due to the curse of dimensionality that accompanies the fitting of a ≥ 65-dimensional maximum-margin hyperplane. The fastest model is the Logit_{120} with a run time of 72.4 seconds, however this is closely followed by Logit_{160} and is also not substantially lower than the run times of most models allowing us to conclude that this is not a crucial selecting feature, regardless the high number of features.

C. Visual Comparison of Models

To visualize the results of the individual classifiers we plot all nine confusion matrices in Fig. 3. The high number of negative samples is apparent in all confusion matrices and supports our previous claim about the

cause of the low precision values in Table III as the high number of true negatives (TN) inevitably results in a high number of misclassified negatives, i.e., false positives (FP).

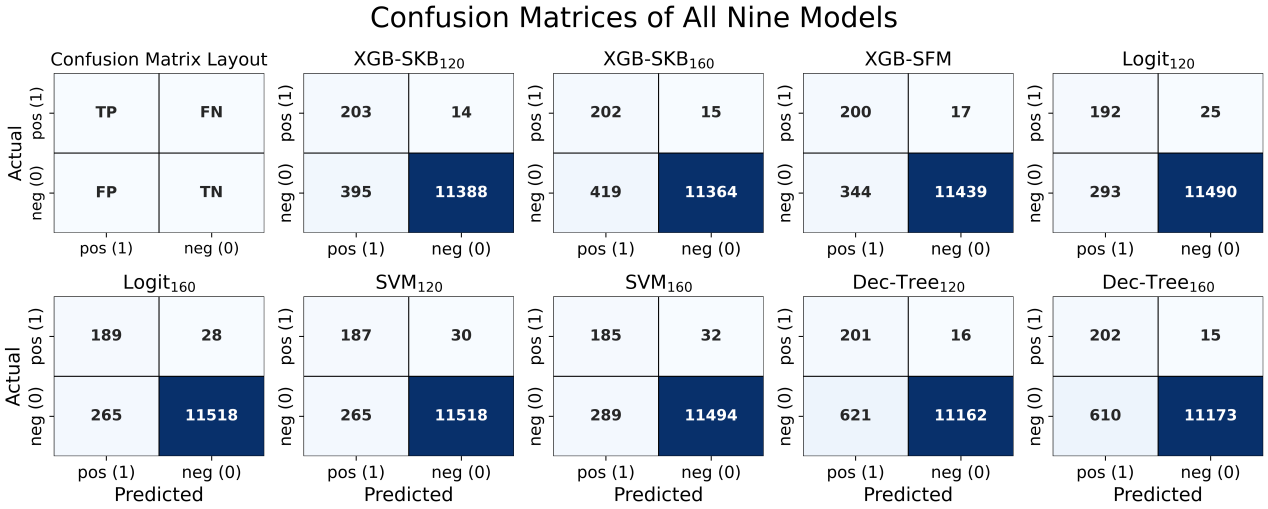
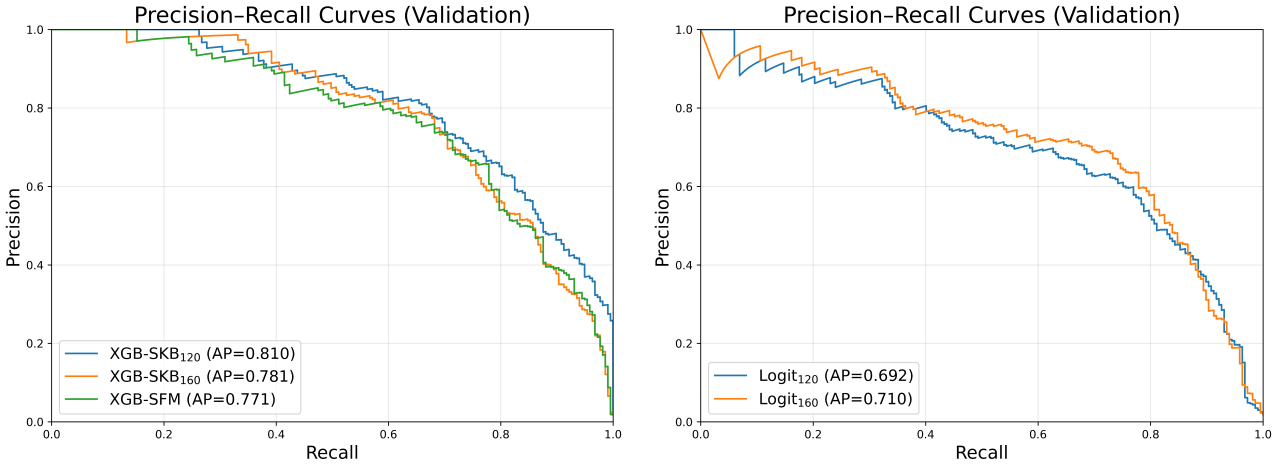


Figure 3: Confusion matrices of all nine trained classifiers evaluated on the validation set. The matrices are arranged in a 2×5 grid, with the top-left panel illustrating the confusion-matrix layout and cell interpretation. The remaining nine panels display the confusion matrices of the individual models as indicated by the subplot titles. All matrices use the orientation defined by the top left matrix, with true positives (TP) shown in the upper-left cell, false negatives (FN) in the upper-right cell, false positives (FP) in the lower-left cell, and true negatives (TN) in the lower-right cell. The horizontal axis denotes the predicted class (positive or negative), while the vertical axis denotes the true class. Each cell contains the absolute number of samples belonging to the corresponding category with a maximum instance number of 12000.



(a) (b)

Figure 4: Precision–recall (PR) curves evaluated on the validation set. The left panel shows the precision–recall curves of the three XGB models (XGB-SKB$_{120}$, XGB-SKB$_{160}$, and XGB-SFM), while the right panel shows the PR curves for the two logistic regression models (Logit$_{120}$ and Logit$_{160}$). In each plot, precision is shown on the vertical axis and recall on the horizontal one. Each curve is generated by varying the classification threshold applied to the continuous model output scores. The legend of each subplot reports the average precision (AP), which summarizes the area under the precision–recall curve for the respective model. This metric also classifies the XGB-SKB$_{120}$ model as the best. Additionally, this model is also the one that reaches a precision of one on the validation set with the highest recall among all other visualized models.

From the perspective of the actual task, the most important metric is not the balanced accuracy, but rather the false negative count. This is because this number characterizes the frequency of trucks that are classified as not faulty, even though they are, which means that a faulty truck is not checked and thus this increases the probability of possible accidents. The best model according to this metric, however, is also the XGB-SKB$_{120}$ model. It is also worth mentioning that with regards to the FN metric, the Logit and SVM classifiers perform substantially worse than the XGB and Dec-Tree algorithms, possibly indicating that the proble is well-conditioned for tree-based algorithms.

To assess comparative performance of ensemble and simpler algorithms, we computed the precision recall curve of the XGB models and the Logit models as these produce easy-to-threshold probabilities as their results. This visualization was chosen instead of the ROC curve in order to prevent the high TN numbers from distorting the plots. The results are shown in Figures 4a and 4b, respectively.

Figure 4a, depicts the precision-recall curves for the three different XGB models that were trained. While all models produce similar curves here, the XGB-SKB$_{120}$ model performs the best again, as quantified by the average precision (AP) metric. The AP value also shows in Fig. 4b that, while the Logit classifiers produce BA scores above 0.9, they are noticeably worse in the AP metric with values $AP^{lg}_{120} = 0.692$, $AP^{lg}_{160} = 0.710$ against values of $AP^{SKB}_{120} = 0.810$, $AP^{SKB}_{160} = 0.781$, and $AP^{SFM} = 0.771$. The figures also highlight that the XGB classifiers reach a precision of one - crucial metric for detecting truck subsystem failures - with a higher still recall value than in the case of the Logit classifiers.

VI. Conclusions

The goal of this project was to predict air pressure system failures in Scania trucks from a high-dimensional, severely imbalanced dataset while maximizing balanced accuracy and minimizing false negatives. To achieve this, multiple machine learning pipelines were constructed using systematic preprocessing, feature selection, and Bayesian hyperparameter optimization, including XGBoost, logistic regression, SVM, and decision tree models. All nine trained models achieved balanced accuracy scores above 0.9 on the validation set, demonstrating that the proposed pipelines handle class imbalance and missing data effectively. The best-performing model was XGB-SKB$_{120}$, achieving a balanced accuracy of 0.9510 and the highest average precision. Crucially, this model also produced the lowest false negative count, making it the most suitable choice for the intended safety-critical application. This low false negative rate additionally indicates that the model can reliably identify faulty trucks in real-world deployment, thereby reducing the risk of undetected system failures.

VII. Additional Information

Code Availability: All codes and files used for this project are available at this GitHub repository. The trained models and related statistics may also be found at this page.

Disclaimer: The GitHub repository mentioned in Section VII also contains two anaconda environment files. Note that for training the model that was uploaded to Kaggle, the environment specified "kaggle_level_env.yml" was used. The other environment named "report_level_env" was used later while writing this report. The latter environment has a newer version of numpy which - for reasons unknown to me - results in different best parameters for the models as determined by the bayesian optimization search. Note, therefore, that if the instructor will use a different environment, the models might produce slightly different results even if the random seed is fixed.