# Data Mining and Machine Learning Project

Krisztián Szuppinger
szuppinger.krisztian@hallgato.ppke.hu

## Contents



## I. Introduction

The dataset provided for this project consists of 60,000 observations with 170 features. The data describe system failures in Scania trucks during everyday operation. The output variable is binary: it is positive for trucks whose failure is associated with the air pressure system (APS) and negative for those in which the failure originates from other vehicle components. Feature names are anonymized, and the attributes consist of either single numerical values or histogram bins. The objective of the task is to predict the output class while maximizing the balanced accuracy (BA), defined as $\text{BA} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$.

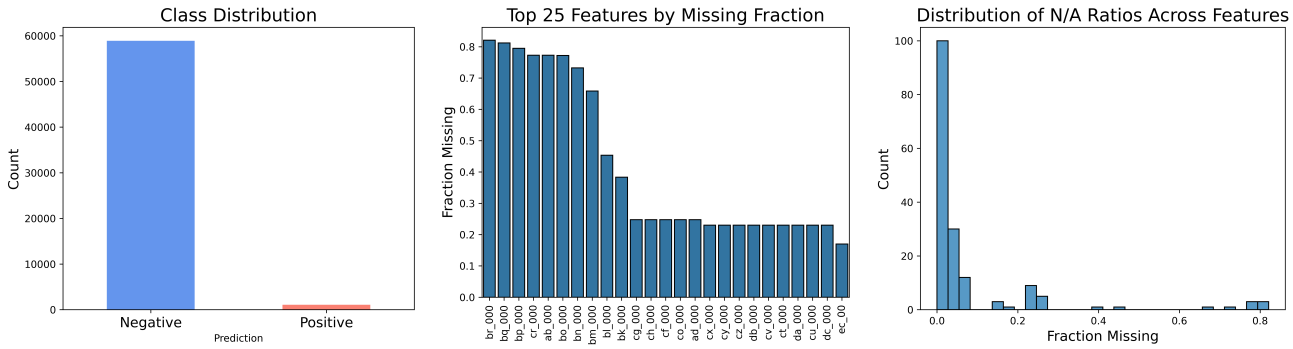
This report documents the solution approach, including exploratory data analysis, preprocessing steps, the selected algorithms, evaluation metrics, trained models, and the obtained results. Finally, feature importance estimates derived from the best-performing model are presented.

## II. Exploratory Data Analysis and Preprocessing

As a first step of the exploratory data analysis (EDA), the sizes of the training and test sets were determined to be $(60{,}000 \times 170)$ and $(16{,}000 \times 170)$, respectively.

Since the task involves predicting failures of a specific system among many possible failure sources, a strong class imbalance was expected. This was confirmed by inspection: the negative class contains 58,914 samples, whereas the positive class consists of only 1,086 observations, resulting in a negative-to-positive ratio of n:p $\approx 54.25$. This imbalance is visualized in Figure 1a and is taken into account throughout the subsequent modeling process.

Finally, the distribution of missing values (N/A) across features was examined. This analysis revealed several attributes for which data are missing in a large fraction of the training instances. The most extreme case is the feature `br_000`, which contains 49,259 missing values out of 60,000 observations. Figure 1b shows the 25 features with the highest fraction of missing values, while Figure 1c illustrates the overall distribution of missing-value fractions across all features. While most attributes contain only a small number of missing values, a non-negligible subset exhibits missing fractions close to or exceeding 0.8.




(a) (b) (c)

Figure 1: Exploratory data analysis results. (a) Class distribution in the training dataset, illustrating the strong imbalance between negative and positive samples. (b) The 25 features with the highest fraction of missing values, where the missing fraction is defined as $\frac{\#\text{N/A}}{\#\text{non-N/A}}$. (c) Distribution of missing-value fractions across all features. While most attributes have low missing fractions, a noticeable number exhibit missing fractions close to 0.8.

## III. Data Preprocessing

Based on the exploratory data analysis, four algorithms were selected for training. Since several features exhibit extreme levels of missingness and the overall dimensionality of the dataset is high, XGBoost (XGB) was chosen as the primary algorithm, as it can internally handle both missing values and high-dimensional feature spaces. Two XGB models were trained, differing in their preprocessing strategies, which are detailed below. In addition to the XGB models, a logistic regression (Logit), a support vector machine (SVM), and a decision tree classifier (Dec-Tree) were trained to assess how simpler models compare to the ensemble boosting approach of XGB.

For two XGB models, as well as for the logistic regression, SVM, and decision tree classifiers, the first preprocessing step excludes attributes whose missing-value fraction (i.e., $\frac{\#\text{N/A}}{\#\text{non-N/A}}$) exceeds $30\%$. This reduces the number of attributes from 170 to 160. To avoid discarding potentially informative features with high missingness, however, an additional XGB model was trained without this filtering step. Since XGB can handle missing values internally, this model receives all 170 features. Figure 2 illustrates the feature information density before and after filtering and shows that, after filtering, most retained features exhibit near-zero missing-value fractions.

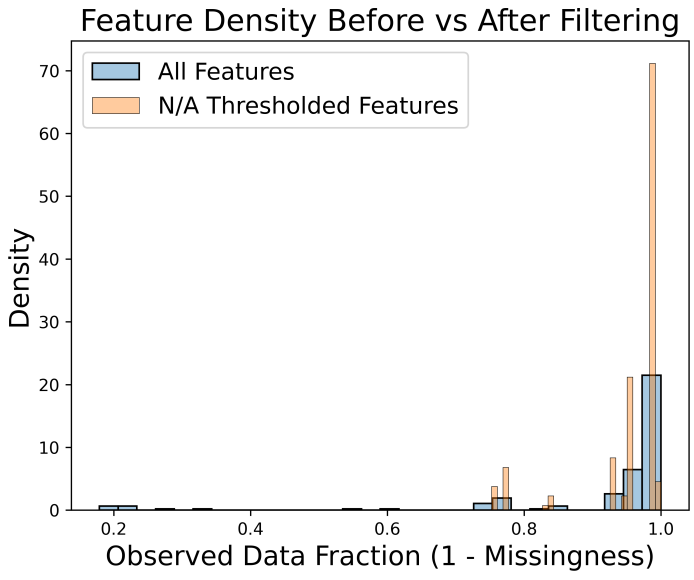



Figure 2: Missing-value density for the two preprocessing strategies. The x-axis shows the fraction of observed data (i.e., non-N/A values) in each feature, while the y-axis represents the density of features with the corresponding observed-data fraction. The blue histogram corresponds to the original feature set, whereas the orange histogram shows the dataset after filtering features with high missing-value fractions. After filtering, a substantially larger proportion of features cluster near an observed-data fraction of one, indicating a higher overall information density. Features with very high missing fractions in the original dataset (blue) remain visible on the left side of the plot.

All subsequent steps of the data preparation process, including model training and hyperparameter tuning, are implemented using scikit-learn's *Pipeline* class. This ensures that all preprocessing operations are applied

consistently within cross-validation folds, thereby preventing data leakage and maintaining a clear and reproducible code structure. Preprocessing steps are implemented using methods from the scikit-learn library.

### A. Preprocessing for Feature Selected Pipeline

For two XGB models, the SVM and logistic regression classifiers, and the decision tree (collectively referred to as the *feature-selected pipeline* from here on), missing values are handled using the SimpleImputer function, which replaces all missing entries with the median of the corresponding attribute. The median is chosen instead of the mean, as outliers may distort the mean value and, consequently, the distribution of the data after imputation. This step is followed by standardizing the attribute values using the StandardScaler method. Finally, feature selection is performed using the SelectKBest method with the ANOVA F-score as the scoring function and an initial value of $k = 40$, which is later tuned as a hyperparameter of each model.

### B. Preprocessing for N/A Safety XGB Pipeline

An additional XGB model is trained that initially receives all 170 attributes of the dataset. The effective number of retained features, however, is tuned during training as a hyperparameter of the model. Feature selection is performed using the SelectFromModel method, with the selector itself instantiated as an XGB classifier with 200 estimators, a maximum depth of 4, and a learning rate of 0.1. The selection threshold is set to the median importance value. In this setup, feature selection is therefore driven by an XGB model, and the selected feature set is passed to the final XGB classifier through the defined pipeline (referred to as the *N/A-safety XGB pipeline* from here on).

## IV. Training

As mentioned previously, four different algorithms were used for training, resulting in a total of nine trained models. This includes the model from the N/A-safety XGB pipeline, as well as two versions of each model from the feature-selected pipeline. These latter model variants differ in their hyperparameter tuning setup: one version is trained with a maximum selectable feature count of 160, while the other is restricted to a maximum of 120 features.

In this section, we first provide a general description of the training structure shared by all models and then discuss the individual model configurations.

### A. General Training Structure

The overall training architecture is identical across all nine models, with the random seed fixed to 42 throughout the notebook. Prior to training, the labeled dataset is split into training and validation sets using an 80:20 ratio, resulting in 48,000 training and 12,000 validation instances. Training proceeds as follows.

First, a scikit-learn Pipeline object is defined, containing the appropriate preprocessing steps (e.g., *SimpleImputer* or *StandardScaler*) and the classifier with its corresponding parameters. Next, an $n$-dimensional hyperparameter search space is specified for the given model, where $n$ denotes the number of parameters explicitly tuned in the pipeline (i.e., parameters not left at their default values). The pipeline and search space are then passed to the BayesSearchCV method, which performs Bayesian optimization to identify the hyperparameter configuration that maximizes the balanced accuracy score among the sampled candidates. This approach avoids an exhaustive grid search and therefore substantially reduces training time. Finally, the model corresponding to the best-performing hyperparameter configuration is selected. Performance here is based on the BA score.

### B. Specific Models

*XGB SelectKBest Models:* The XGB models using SelectKBest-based feature selection (XGB-SKB models) are trained first. Preprocessing within the pipeline is performed as described in Section III-A. The initial feature count is set to $k = 40$ for both the 120- and 160-feature XGB-SKB models, and the optimal number of selected features is consistently found to be the maximum allowed value. The evaluation metric of the *XGBClassifier* object is set to *logloss* in order to avoid discrete jumps that may arise when optimizing balanced accuracy directly. To account for the severe class imbalance between positive and negative samples, the *scale_pos_weight* parameter is also specified. The remaining hyperparameter search ranges used by *BayesSearchCV*, aside from the feature-count interval, are documented in the accompanying notebook and are omitted here for brevity. The optimized hyperparameter values of the best-performing models are summarized in Table I.

*XGB SelectFromModel Setup:* Apart from the feature selection strategy, this model uses the same parameter settings as the XGB-SKB models. Feature selection is performed using an *XGBClassifier* within the pipeline, as described in Section III-B.

*Logit:* Feature selection and preprocessing are performed as described in Section III-A. The penalty value of the classifier is set to 12 with a maximum iteration of 500 and balanced class weights. The hyperparameter search space for Bayesian optimization consists of the number of selected features $k$ and the regularization parameter $C$, where $C$ controls the inverse strength of the penalty term optimized during training. The search ranges are $k \in [20, 120 \text{ or } 160]$ and $C \in [10^{-3}, 100]$. A 5-fold cross-validation is performed with 40 sampled hyperparameter configurations.

*SVM:* Two SVM models are trained using the SelectKBest feature selection method, differing only in the maximum number of selectable features. During Bayesian hyperparameter optimization, only the number of selected features $k$ and the regularization parameter $C$ are tuned, with search ranges of $k \in [20, 120 \text{ or } 160]$ and $C \in [10^{-3}, 10]$. To limit computational cost, the number of sampled configurations is set to n_iter $= 15$. The cross-validation parameter is set to cv $= 4$, resulting in a 4-fold cross-validation.

*Dec-Tree:* For the decision tree models, preprocessing differs from the pipeline described in Section III-A in that no feature scaling is applied. Bayesian optimization is used to tune the number of selected features $k$, the maximum tree depth ($md$), the minimum number of samples per leaf ($msl$), and the minimum number of samples required to split an internal node ($mss$). The corresponding search ranges are $k \in [20, 120 \text{ or } 160]$, $md \in [2, 12]$, $msl \in [1, 50]$, and $mss \in [2, 50]$. A total of 50 hyperparameter configurations are evaluated using 5-fold cross-validation.

## V. Results

All nine trained models achieve balanced accuracy scores above 90% on the validation set, indicating that the constructed pipelines perform effectively. The model selected for test set prediction is the one with the best validation performance, namely the XGB-SKB$_{120}$ model. This model is retrained on the full labeled dataset (i.e., training and validation sets combined) to maximize predictive performance. The resulting Kaggle score obtained with this model is 0.95787 (dropping from 0.97572).

In this section, we first present the hyperparameters obtained through Bayesian optimization for all nine models, followed by a detailed comparison of their performance statistics. In addition, the predictive behavior of the models is compared visually using confusion matrices and precision–recall curves. Finally, feature importance is examined for the best-performing model, namely the XGB-SKB$_{120}$ model.

### A. Optimized Hyperparameter Settings

The optimized hyperparameters for the XGB models are summarized in Table I. As noted previously, the $k$-value selected for the XGB-SKB models is consistently the maximal value permitted by the corresponding search space. The optimized 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). Although the maximal $k$-value is selected for both XGB-SKB models, their predictive performance differs, as shown in Table III, 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 optimized hyperparameters of the best-performing simple classifiers are reported in Tables IIa and IIb. In contrast to the XGB models, the selected $k$-values vary more substantially across these classifiers. For the logistic regression models, the optimized $k$-value is consistently the maximal one. For the SVM models, however, the optimal feature count differs, with $k = 102$ for SVM$_{120}$ and $k = 65$ for SVM$_{160}$. This behavior is likely a consequence of the Bayesian optimization procedure, which explores the hyperparameter space

stochastically and identifies locally optimal configurations rather than guaranteeing a global optimum. As a result, different optimal values of the regularization parameter $C$ can lead to different corresponding feature counts. For the decision tree models, the selected $k$-values are even lower, with $k = 46$ for Dec-Tree$_{120}$ and $k = 49$ for Dec-Tree$_{160}$. Additionally, the optimized $C$ values for the logistic regression and SVM models are consistently low, as shown in Table IIa, indicating a preference for stronger regularization and reduced risk of overfitting.

Table II: Optimized hyperparameters of the simple classifiers. The $k$-value is not an intrinsic parameter of the classifiers themselves, but rather a parameter of the full preprocessing pipeline. In contrast to the XGB models (Table I), the optimized $k$-value is not always maximal for the simpler classifiers. The regularization parameter $C$ is consistently small across the logistic regression and SVM models, indicating a preference for stronger regularization. For the decision tree models, $md$, $msl$, and $mss$ denote the maximum tree depth, the minimum number of samples per leaf, and the minimum number of 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-performing model according to the balanced accuracy (BA) metric is the XGB-SKB$_{120}$ model, which achieves a validation score of $0.9510$. The second- and third-best models are XGB-SKB$_{160}$ and XGB-SFM, respectively. Although their performance is slightly lower, the simpler models (Logit, SVM, and Dec-Tree) also achieve strong results. In particular, Dec-Tree$_{160}$ reaches a BA score of $0.9396$ while using only $49$ features out of $160$, representing a substantial reduction in model complexity for a relatively small BA decrease of only $0.0114$ compared to the best XGB model. This decision tree model also outperforms both the Logit and SVM classifiers while relying on fewer features. Detailed performance metrics for all nine trained models are reported 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 – 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, exhibit considerably longer run times, which can be attributed to the curse of dimensionality associated with fitting a maximum-margin hyperplane in a feature space of dimension $\geq 65$. The fastest model is $\text{Logit}_{120}$ with a run time of 72.4 seconds, although this is closely followed by $\text{Logit}_{160}$ and is not substantially lower than the run times of most other models, indicating that computational efficiency is not a decisive factor in model selection in this case.

### C. Visual Comparison of Models

To visualize the performance of the individual classifiers, the confusion matrices of all nine models are shown in Fig. 3. The dominance of the negative class is apparent across all matrices and supports the earlier explanation for the low precision values reported in Table III, as a large number of true negatives (TN) inevitably leads to an increased number of misclassified negative samples, i.e., false positives (FP).

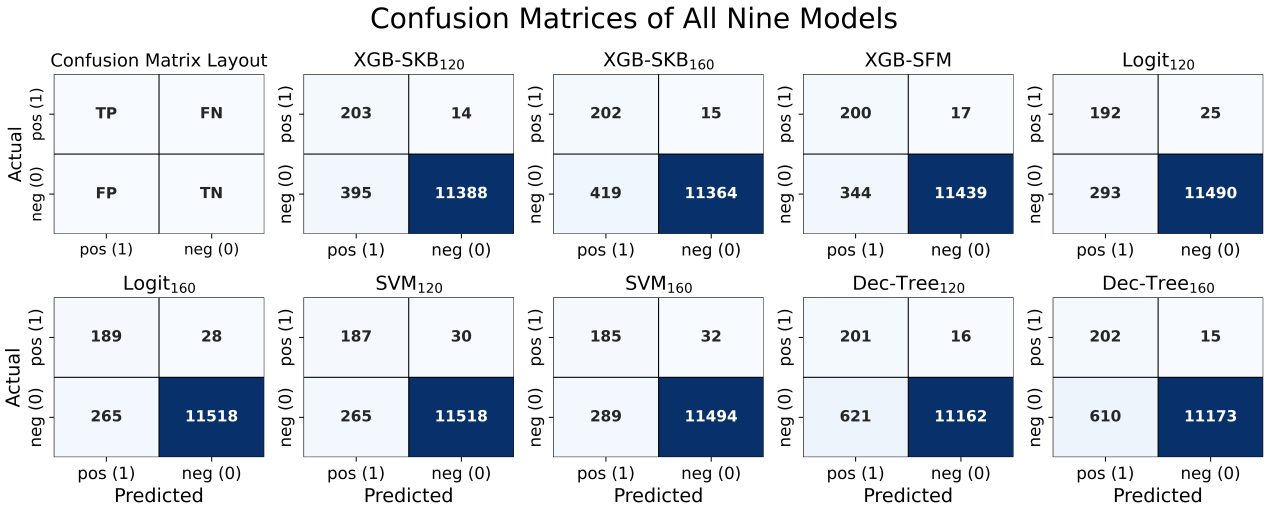
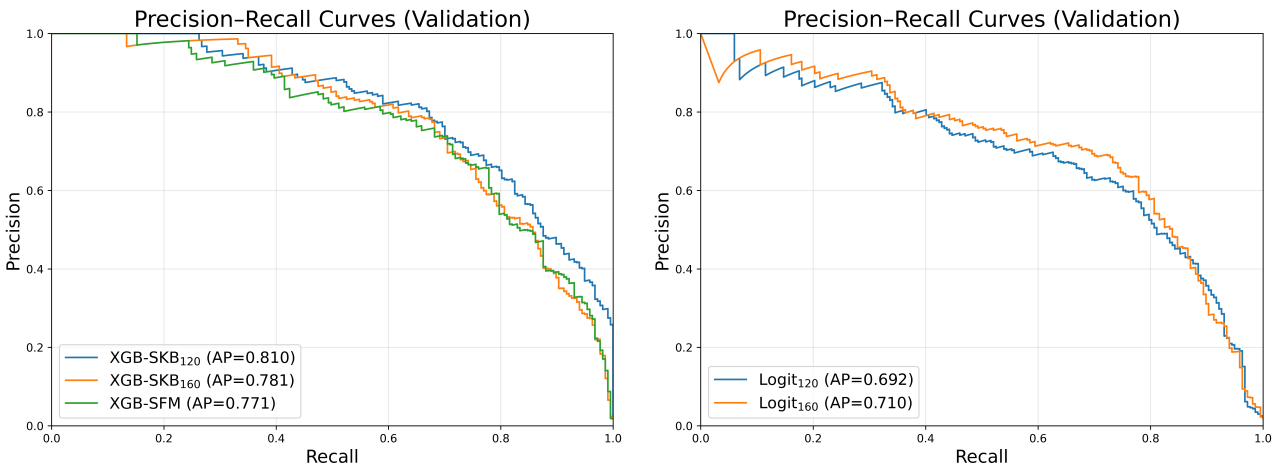



Figure 3: Confusion matrices of all nine trained classifiers evaluated on the validation set. The matrices are arranged in a $2 \times 5$ grid, with the top-left panel illustrating the confusion-matrix layout and cell interpretation. The remaining nine panels display the confusion matrices of the individual 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 12,000.

From the perspective of the actual task, the most important metric is not the balanced accuracy but rather the false negative count. This metric reflects the frequency with which faulty trucks are incorrectly classified as non-faulty, resulting in missed inspections and an increased risk of accidents. The best-performing model with respect to the FN count is again $\text{XGB-SKB}_{120}$. It is also notable that, in terms of false negatives, the logistic regression and SVM classifiers perform substantially worse than the XGB and decision tree models, suggesting that the problem is particularly well suited to tree-based approaches.

To compare the performance of ensemble and simpler algorithms, precision-recall curves were computed for the XGB and logistic regression models, as these produce easy-to-threshold probabilities as their results. This visualization was chosen over ROC curves to avoid distortion caused by the large number of true negatives. The resulting curves are shown in Figures 4a and 4b, respectively.

Figure 4a shows the precision-recall curves of the three trained XGB models. Although the curves are similar overall, the $\text{XGB-SKB}_{120}$ model again performs best, as quantified by the average precision (AP) metric. Figure 4b further illustrates that, while the logistic regression models achieve balanced accuracy scores above 0.9, their AP values are noticeably lower, with $AP^{lg}_{120} = 0.692$ and $AP^{lg}_{160} = 0.710$, compared to $AP^{SKB}_{120} = 0.810$, $AP^{SKB}_{160} = 0.781$, and $AP^{SFM} = 0.771$ for the XGB-based models. Figure 4a also highlights that the $\text{XGB-SKB}_{120}$ classifier reaches a recall of one, meaning that all true positives are detected, while maintaining substantially higher precision than any of the other models shown in Figure 4.

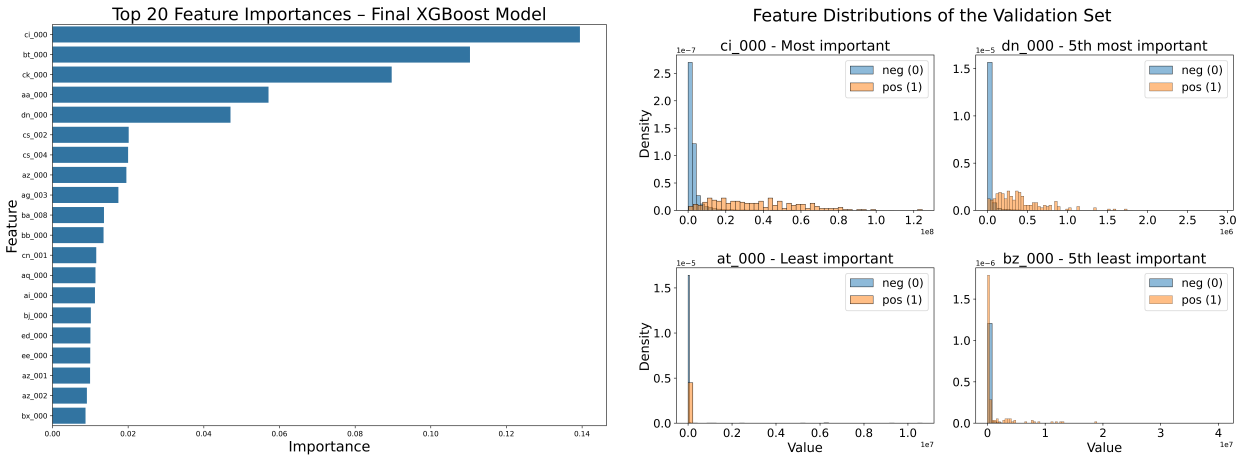



(a) (b)

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.

### *D. Visual Analysis of the Best Model*

After identifying XGB-SKB$_{120}$ as the best-performing model, the feature importances of the corresponding XGB classifier were examined to identify the attributes that contribute most strongly to high balanced accuracy. The twenty most important features are shown in Fig. 5a. Feature importance scores are computed as defined by the XGBoost algorithm.




(a) (b)

Figure 5: Results of a visual analysis of the best-performing final model, XGB-SKB$_{120}$. Panel (a) shows the twenty most important features as quantified by the XGBoost feature importance measure, with noticeable drops in importance both within the top five features and between the top five and the remaining ones. Panel (b) compares the distributions of selected highly important and least important features for the positive and negative classes. For the least important features, the class-conditional distributions exhibit low variability and strong overlap. In contrast, the most important features show higher variability for the positive class and substantially reduced overlap between the two classes.

Additionally, the differences between the most and least important features are visualized to better understand what distinguishes them. As shown in Figure 5b, most instances of the highly important features exhibit limited overlap with the negative class, whereas for the least important features, almost all instances overlap with the 0-class. It is also apparent that the variability of values is higher for the most important features than for the least important ones.

## VI. Conclusions

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

## VII. Additional Information

*Code Availability:* All codes and files used in this project are available in the following GitHub repository. The trained models and the corresponding evaluation statistics can also be found at this location. For clarity, the best trained model, namely XGB-SKB$_{120}$, is saved here as $best\_of\_nine\_model.pkl$.

*Disclaimer:* The GitHub repository referenced in Section VII contains two Conda environment files. The model submitted to the Kaggle competition was trained using the environment specified in `kaggle_level_env.yml`. Although all experiments were conducted with fixed random seeds, Bayesian hyperparameter optimization involves stochastic exploration and floating-point computations, which may lead to slightly different optimal hyperparameter configurations when using different software versions (e.g., newer releases of NumPy or XGBoost). When the provided environment is used, however, the reported results should be reproducible.