Prediction of APS Failure in Scania Trucks

Data Mining and Machine Learning

Shihab Muhtasim
PPKE IPCVAI

Kaggle Username: shihabmuhtasimppke

December 12, 2025

Contents

1 Introduction **3**

2 Exploratory Data Analysis (EDA) **3**
2.1 Dataset Overview 3
2.2 Target Variable and Class Imbalance 4
2.3 Missing Data 4
2.4 Behaviour of a Histogram Feature Group 5
2.5 Correlation Analysis 6

3 Data Preparation **7**
3.1 Dropping Sparse Features 7
3.2 Feature Engineering: Total Histogram Activity 7
3.3 Splitting to Avoid Data Leakage 8
3.4 Imputation and Scaling 8
3.5 Balancing the Classes with SMOTE 8
3.6 Correlation-Based Feature Selection 8

4 Algorithms **9**
4.1 Model Selection Rationale 9
4.2 Baseline Performance Analysis 10

5 Hyperparameter Tuning **11**
5.1 Methodology: Leakage-Free Pipeline Design 11
5.2 Tuning Strategy and Parameters 11
5.3 Results and Analysis 12

6 Combined Approaches (Ensembles) **13**
6.1 Ensemble 5: The Safety-Weighted Consensus 14
6.2 The Super-Learner: Optimization via OOF Predictions 14
6.3 The Star Model: Final Blend and Threshold Tuning 15

7 Evaluation **16**
7.1 Selected Metrics 16
7.2 Comprehensive Model Performance 17
7.3 Comparative Analysis and Selection 18
7.4 Conclusion 18

8 Experiences and Best Results **18**
8.1 Learnings 19
8.2 Best Final Result: The Star Model 19

9 Conclusion **19**

1 Introduction

The Air Pressure System (APS) is a key safety component in heavy Scania trucks. It generates the pressurized air needed for braking and gear changes. If this system fails, the truck may not brake or shift reliably, which can lead to roadside breakdowns, safety risks, and high operating costs.

In this project I use an anonymized industrial dataset from Scania with operational measurements collected from trucks in real use. The task is part of a Kaggle-style competition: given sensor readings, the goal is to predict whether the APS has a specific failure. The dataset is highly imbalanced so i have made sure to balance it with specific modifications.

The problem is a binary classification task. For each truck we must decide whether the APS has the failure of interest or not. For class 0: no APS failure, but for class 1: it is APS failure, and truck should be inspected and repaired.

The competition provides a training set with 60 000 labeled samples and a test set with 16 000 unlabeled samples. Each sample has 171 anonymized attributes: counters, continuous measurements and several histogram-type features that describe how often a sensor value falls into different ranges. The feature names do not reveal their physical meaning, so the focus is on how they behave statistically.

Because failures are rare, a model that always predicts “no failure” would already achieve high accuracy but would never detect a true fault. The practical goal is therefore not only to be accurate, but also to control *what kind of mistakes* the model makes. Missing a real failure (false negative) is much more serious than raising an occasional false alarm (false positive). Even though it raises the maintainance cost as workers have to check for safety but missing a truck failure can lead to dangerous hazards.

The project aim is to build a safety-oriented classifier that keeps false negatives low while still limiting unnecessary inspections. I followed a step-by-step strategy: first training standard baseline models to understand the data, then selecting and tuning the most promising ones (mainly ensemble tree methods), and finally combining them into several ensemble systems with some tuning and weighted voting mechanisms. From these I defined a single “Star Model” that represents the best trade-off between safety (few missed failures), efficiency (reasonable number of false alarms), and the competition metric (balanced accuracy). Throughout the work, I used a strictly data leakage-free pipeline so that all reported results reflect realistic performance.

2 Exploratory Data Analysis (EDA)

Before building models I examined the structure and basic behaviour of the data: its size, class balance, missing values and a few feature groups that looked promising.

2.1 Dataset Overview

The training data contains 60 000 trucks and 171 anonymised sensor measurements per truck. These include simple counters, continuous values and several histogram features where each bin counts how often a sensor value falls into a certain range. Although the column names are anonymized, each one can be treated as an operational signal coming from the APS or a related subsystem.

2.2 Target Variable and Class Imbalance

The target label indicates whether the APS has the specific failure (class 1) or not (class 0). Figure 1 shows the distribution of labels on a logarithmic scale. There are 58 914 healthy cases and only 1 086 failure cases, so failures represent less than 2% of the training set. Without special handling, most models would focus almost entirely on the majority class. This strong imbalance motivates the later use of resampling and metrics such as balanced accuracy.

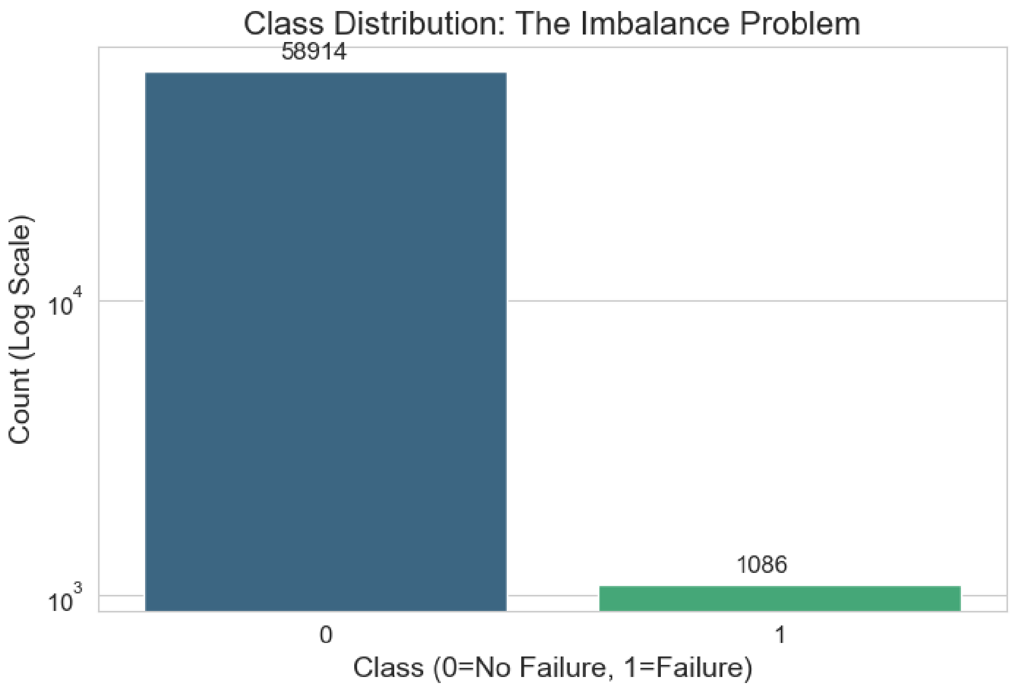


Figure 1: Class distribution on a log scale.

2.3 Missing Data

Many features contain missing values, and some are almost completely empty. A nullity matrix of the most affected features is shown in Figure 2: each horizontal line is one truck, and white gaps indicate missing entries.

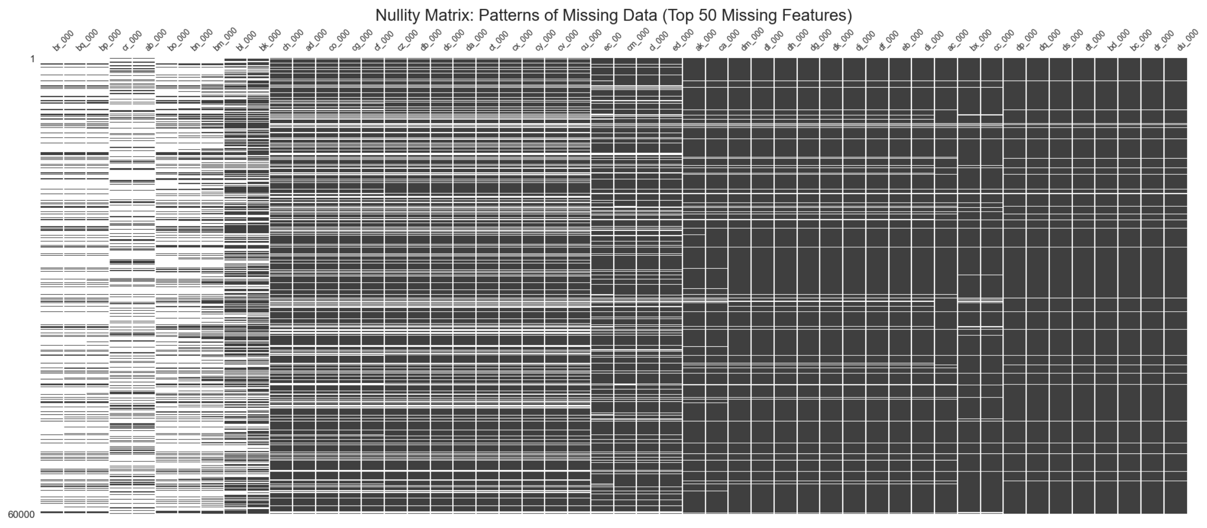


Figure 2: Nullity matrix for features with the highest amount of missing data. White lines indicate missing values.

Several columns have more than 70% missing values and contribute virtually no information. To avoid injecting almost entirely imputed columns into the models, these heavily missing features, these visually obvious ones, were later removed and the remaining ones were imputed more carefully.

2.4 Behaviour of a Histogram Feature Group

A histogram feature group related to one particular sensor (denoted by the prefix "ag") stood out during the analysis. Each bin in this group counts how often the sensor value fell into a given range. Figure 3 compares the average value of each bin for healthy and failed trucks (log scale).

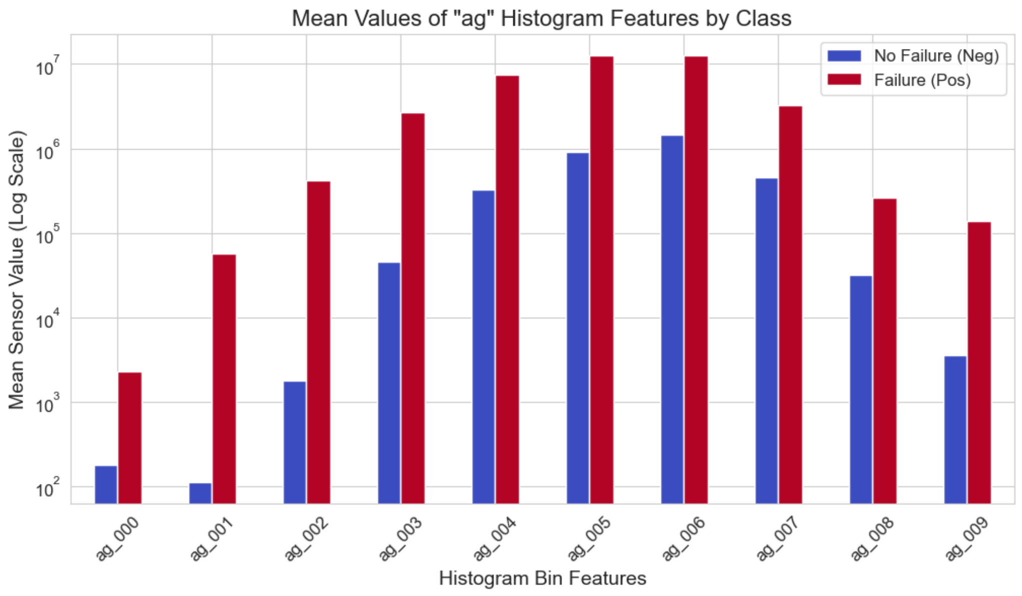
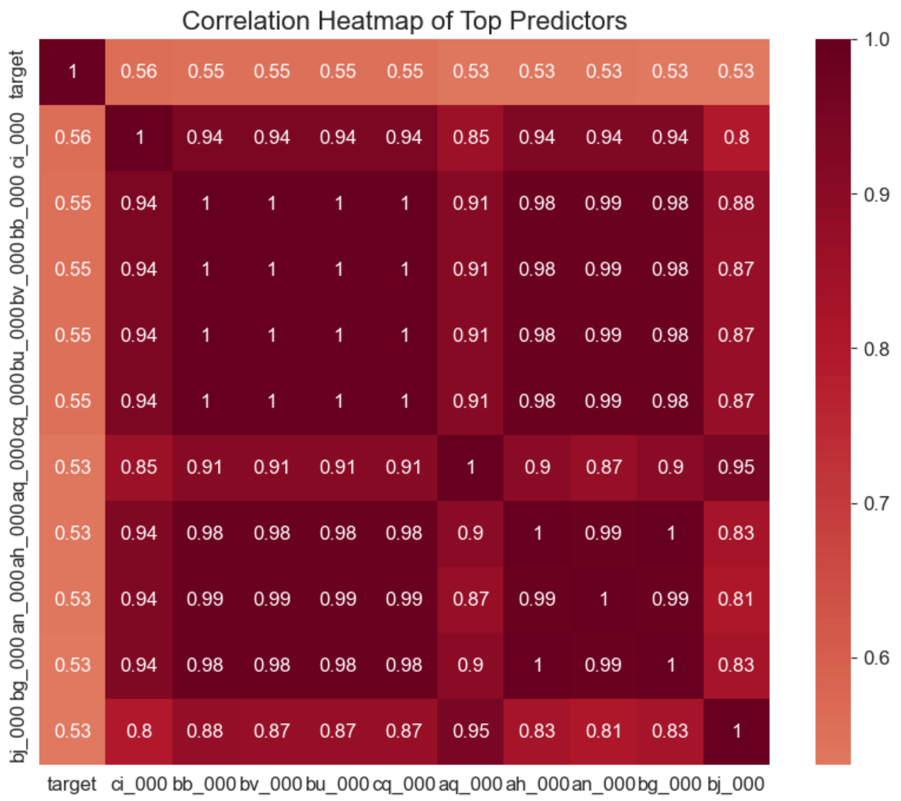


Figure 3: Mean values of a histogram feature group for the two classes (log scale).

Failure cases show much higher counts across almost all bins, especially in the middle bins that likely correspond to "medium to high" sensor values. This suggests that, for this sensor, trucks with APS problems experience more frequent or more intense activity. Since we are more concerned about the safety, these observations motivated later feature engineering on this group.

2.5 Correlation Analysis

To identify potentially useful predictors, I computed the correlation between each feature and the target label on the (later) balanced training data, and examined the most strongly related ones in a small correlation heatmap (Figure 4).



	target	ci_000	bb_000	bv_000	bu_000	cq_000	aq_000	ah_000	an_000	bg_000	bj_000
target	1	0.56	0.55	0.55	0.55	0.55	0.53	0.53	0.53	0.53	0.53
ci_000	0.56	1	0.94	0.94	0.94	0.94	0.85	0.94	0.94	0.94	0.8
bb_000	0.55	0.94	1	1	1	1	0.91	0.98	0.99	0.98	0.88
bv_000	0.55	0.94	1	1	1	1	0.91	0.98	0.99	0.98	0.87
bu_000	0.55	0.94	1	1	1	1	0.91	0.98	0.99	0.98	0.87
cq_000	0.55	0.94	1	1	1	1	0.91	0.98	0.99	0.98	0.87
aq_000	0.53	0.85	0.91	0.91	0.91	0.91	1	0.9	0.87	0.9	0.95
ah_000	0.53	0.94	0.98	0.98	0.98	0.98	0.9	1	0.99	1	0.83
an_000	0.53	0.94	0.99	0.99	0.99	0.99	0.87	0.99	1	0.99	0.81
bg_000	0.53	0.94	0.98	0.98	0.98	0.98	0.9	1	0.99	1	0.83
bj_000	0.53	0.8	0.88	0.87	0.87	0.87	0.95	0.83	0.81	0.83	1

Figure 4: Correlation heatmap of the target and selected features. Darker colours indicate stronger relationships.

Several features show a clear positive association with APS failures, meaning their values tend to increase when a failure is present. Many of these top features are also strongly correlated with each other, which suggests that they capture related aspects of the same physical behaviour. This structure supports the later use of simple correlation-based feature selection to remove clearly irrelevant variables while keeping the main signal.

Overall, the EDA phase highlighted three main challenges: very strong class imbalance, substantial missing data, and groups of features whose distribution changes markedly between healthy and failed trucks. These findings directly shaped the preprocessing pipeline.

3 Data Preparation

The raw APS data is noisy, sparse and highly imbalanced, so I designed a preprocessing pipeline to clean it, engineer one informative feature, avoid data leakage, and prepare it for learning. The steps below follow the same order as in the notebook.

3.1 Dropping Sparse Features

First, I removed features that were almost entirely missing. For each column I calculated the fraction of missing values and dropped it if more than 70% of its entries were empty. This removed 7 out of the original 171 attributes and reduced the feature count to 163. Columns that are mostly missing would otherwise be filled almost completely by imputation and could mislead the models more than they help.

3.2 Feature Engineering: Total Histogram Activity

The histogram group mentioned in the EDA appeared to be strongly related to failures. To capture its overall activity in a single value, I created a new feature that sums all bins of this group into a "total activity" measure. This new column was added to both training and test data, increasing the feature count from 163 to 164.

Figure 5 shows the distribution of this engineered feature for healthy and failed trucks. On a log scale the median and overall level are clearly higher for failure cases, confirming that the cumulative activity of this sensor is a useful indicator of APS problems.

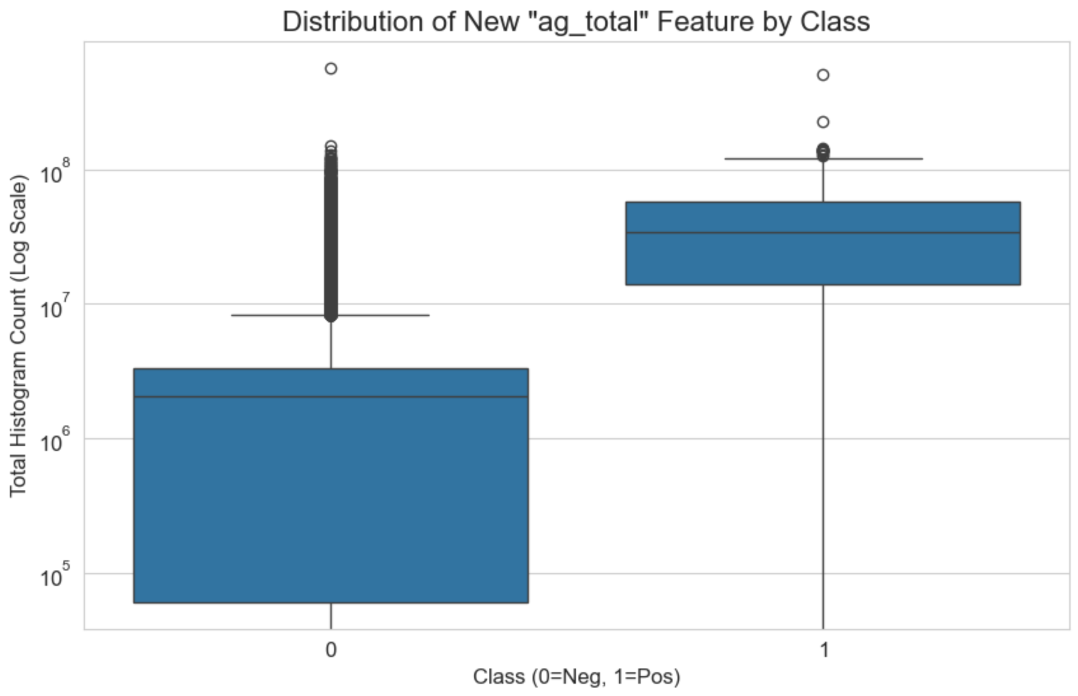


Figure 5: Distribution of the engineered total histogram feature by class (log scale).

3.3 Splitting to Avoid Data Leakage

A central design choice was to avoid data leakage. Any information from the validation data must not influence preprocessing steps such as imputation, scaling or oversampling. To enforce this, I split the data into training and validation parts *before* these steps.

The label file contained both an identifier and a text label (“neg”/“pos”). I extracted only the label column and mapped it to 0 and 1. Using this cleaned target I performed a stratified 80/20 split with a fixed random seed, resulting in 48 000 samples for training and 12 000 for validation, each with 164 features. From this point on, all transformations were fitted only on the training portion and then applied to validation and test data. Hence, no validation data was seen during the training phase.

3.4 Imputation and Scaling

After the split, I handled remaining missing values and differences in scale. To keep the process consistent, I built a two-step preprocessing pipeline: median imputation followed by robust scaling.

Median imputation replaces missing values by the median of each feature in the training data. This is more robust than using the mean because many sensors have highly skewed distributions and extreme outliers; the median remains stable even when some values are very large.

Robust scaling then centers each feature by its median and scales it by its interquartile range. Unlike standardization or min–max scaling, this approach focuses on the middle 50% of the data and largely ignores extreme spikes, which are common in this industrial dataset.

The pipeline was fitted on the training part and applied to the validation and test sets. This was none of the values of the validation set would influence the imputation on the training set, and our model can learn in an unbiased way.

3.5 Balancing the Classes with SMOTE

Even after cleaning and scaling, the training data remained highly imbalanced. To give the classifiers a chance to learn the failure class, I used the Synthetic Minority Oversampling Technique (SMOTE) on the training data only. SMOTE generates synthetic failure samples by interpolating between existing ones, which increases the number of minority instances without simply duplicating them.

After applying SMOTE, the training set contained 94 262 samples with an equal number of class 0 and class 1 labels, still with 164 features. The validation set was deliberately left untouched and kept in its original imbalanced form so that evaluation reflects real-world class frequencies. This technique was only applied to the training set for model training and for the model to be able to learn from a balanced data set.

3.6 Correlation-Based Feature Selection

Finally, I reduced the feature space using a simple correlation-based selection. On the balanced training data I computed the Pearson correlation between each feature and the binary target, took the absolute value and kept only features with correlation above 0.05. This threshold is low enough not to throw away potentially useful variables, but high enough to remove clearly irrelevant ones.

This step kept 144 of the 164 features. The final prepared datasets used for modelling therefore had:

- a balanced training set of 94 262 samples and 144 features, and
- a validation set of 12 000 samples and the same 144 features.

4 Algorithms

To establish a comprehensive performance benchmark, I selected 9 distinct algorithms ranging from simple linear models to state-of-the-art ensemble methods. This diversity allowed me to understand whether the problem required complex non-linear boundaries.

4.1 Model Selection Rationale

1. **Logistic Regression:** Selected as the standard linear baseline. If this simple model performed well, it would suggest the classes are linearly separable.
2. **Naive Bayes (Gaussian):** Chosen for its computational efficiency and tendency to have high recall (sensitivity) in high-dimensional spaces, although often at the cost of precision.
3. **K-Nearest Neighbors (KNN):** A non-parametric method selected to capture local data structures and clusters of failure that linear models might miss.
4. **AdaBoost:** The original boosting algorithm. Selected to test if iteratively focusing on hard-to-classify samples improves performance.
5. **Gradient Boosting Classifier (sklearn):** A robust boosting implementation that optimizes an arbitrary differentiable loss function.
6. **Random Forest:** A Bagging ensemble. Selected for its robustness to noise and ability to handle the non-linearities of sensor data without heavy overfitting.
7. **Extra Trees (Extremely Randomized Trees):** Similar to Random Forest but with more randomness in split selection. This often reduces variance further and is computationally faster.
8. **XGBoost (Extreme Gradient Boosting):** The industry standard for tabular data. Chosen for its speed, built-in regularization (L1/L2), and ability to handle missing values natively.
9. **LightGBM:** A gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient, with faster training speed and lower memory usage.

4.2 Baseline Performance Analysis

Before tuning or building ensembles, I first evaluated a diverse set of baseline models to see how different algorithm families behave on this dataset. All models were trained on the SMOTE-balanced training data (to remove class imbalance during learning) and evaluated on the untouched validation set. This setup keeps the validation data completely clean and avoids any data leakage. The confusion matrices and balanced accuracy scores are shown in Figure 6.

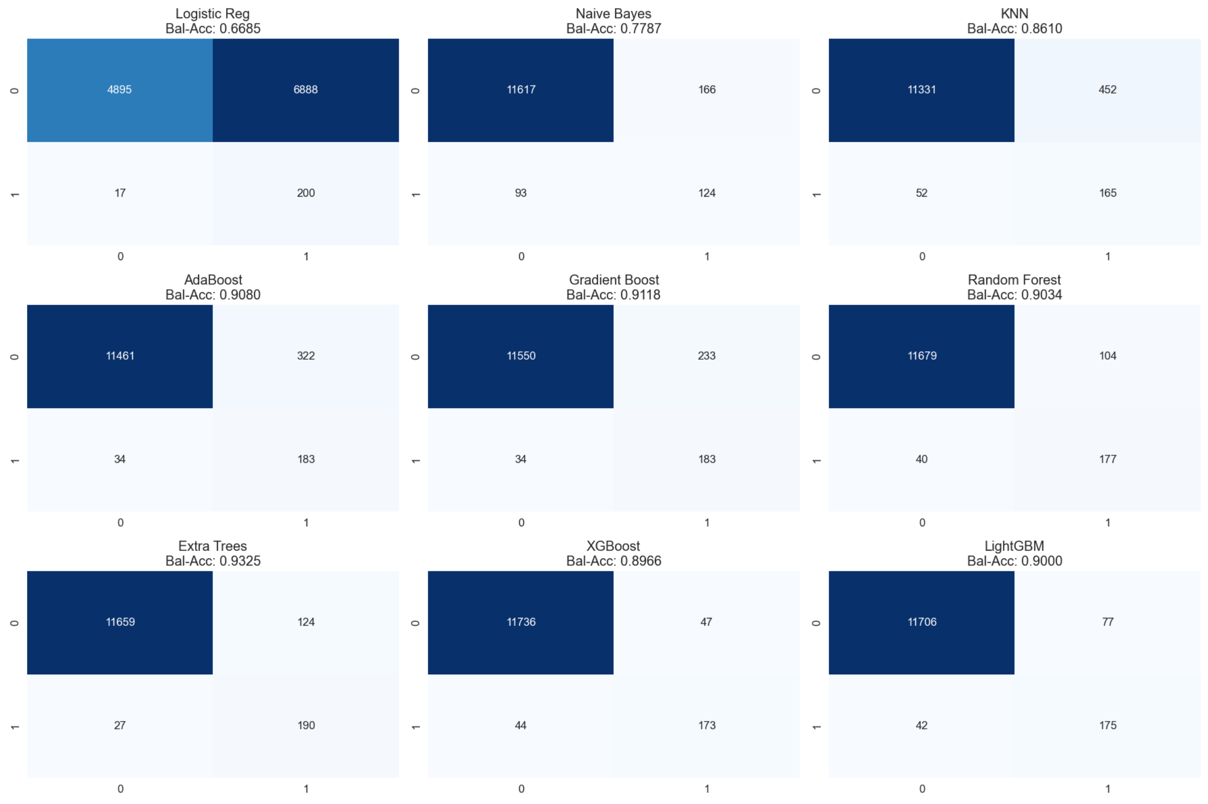


Figure 6: Confusion matrices of the baseline models

Logistic Regression was used as a simple linear reference. It achieved a balanced accuracy of 0.6685, but produced **17** false negatives and an extremely large number of false positives (**6 888**), effectively predicting "failure" far too often (see the top-left panel of Figure 6). This confirmed that a linear decision boundary is not suitable here.
Naive Bayes was included because it can sometimes perform surprisingly well on high-dimensional data. In this case it reached a balanced accuracy of 0.7787 but missed **93** failures and still produced **166** false alarms. The low recall (around 0.57) makes it unsafe for our goal.
K-Nearest Neighbors served as a non-parametric baseline. It obtained a balanced accuracy of 0.8610 with **52** false negatives and **452** false positives. While this is better than the linear and Naive Bayes models, it still leaves many failures undetected and is sensitive to noise in this high-dimensional setting.
AdaBoost improved the results noticeably: balanced accuracy 0.9080 with **34** false negatives and **322** false positives. It shows that boosting helps, but it was still dominated by stronger models in terms of both safety and overall balance.
Gradient Boosting, **Random Forest** and **Extra Trees** performed clearly better than

the simple baselines. Gradient Boosting reached a balanced accuracy of 0.9118 with **34** false negatives and **233** false positives. Random Forest achieved 0.9034 with **40** false negatives and only **104** false positives, offering a good compromise between recall and false alarms. Extra Trees gave the best baseline among these, with balanced accuracy 0.9325, only **27** false negatives and **124** false positives.
XGBoost and **LightGBM** also showed strong baseline behaviour. XGBoost reached a balanced accuracy of 0.8966 with **44** false negatives and **47** false positives, which means it is relatively conservative but precise when it predicts a failure. LightGBM achieved 0.9000 with **42** false negatives and **77** false positives, again indicating good potential once tuned.

In summary, the baseline experiments clearly separate the models into two groups. Linear, Naive Bayes and KNN baselines either produced too many false negatives or an unrealistic number of false positives, and are therefore not ideal for a safety-focused system. Tree-based ensembles and gradient boosting methods, on the other hand, consistently achieved higher balanced accuracy with substantially fewer missed failures. For this reason, the subsequent hyperparameter tuning and ensemble work focused on Gradient Boosting, Random Forest, Extra Trees, XGBoost and LightGBM, which offered the best overall trade-off between safety and efficiency.

5 Hyperparameter Tuning

Following the baseline evaluation, I selected the five most promising algorithms for further optimization: Random Forest, Extra Trees, XGBoost, LightGBM, and Gradient Boosting. These models demonstrated the best potential for balancing high accuracy with the critical requirement of minimizing missed failures.

5.1 Methodology: Leakage-Free Pipeline Design

The Data Leakage Challenge Standard cross-validation on oversampled data often leads to a subtle but critical error known as data leakage. If SMOTE is applied to the entire dataset *before* splitting it into training and validation folds, synthetic samples in the validation fold may be near-duplicates of samples in the training fold. This allows the model to "memorize" the synthetic variations rather than learning generalizable patterns, leading to artificially inflated accuracy scores that collapse on unseen data.

The Pipeline Solution To prevent this, I implemented a strict `ImbPipeline` structure. Instead of oversampling beforehand, I wrapped the SMOTE process and the classifier into a single atomic unit. During the 3-fold cross-validation loop, the pipeline ensures that SMOTE is applied *only* to the training portion of the current fold. The validation portion remains strictly "clean" and unseen. This structure simulates a real-world production environment where the model encounters only raw, legitimate data, ensuring that the hyperparameter search optimizes for genuine generalization rather than memorization.

5.2 Tuning Strategy and Parameters

I utilized `RandomizedSearchCV` to efficiently explore the hyperparameter space. This method samples a fixed number of parameter settings from specified distributions, offering a better trade-off between runtime and quality compared to exhaustive grid search.

Cost-Sensitive Parameters The most critical adjustment was tuning the models for **Safety**. Standard accuracy metrics treat all errors equally, but in this domain, a false negative is significantly more costly than a false positive.

- For tree ensembles (Random Forest, Extra Trees), I enforced `class_weight='balanced'`, which automatically adjusts weights inversely proportional to class frequencies.
- For gradient boosting methods (XGBoost, LightGBM), I explicitly set `scale_pos_weight=59`. This parameter tells the loss function to penalize a missed failure 59 times more heavily than a false alarm, directly counteracting the 1:59 class imbalance ratio. It was a trial-and-error finding for me.

Regularization and Complexity To prevent overfitting to the synthetic SMOTE samples, I also tuned complexity parameters such as `max_depth` (limiting tree growth), `learning_rate` (slowing down convergence for better generalization), and `n_estimators`.

5.3 Results and Analysis

The quantitative results of the tuning process are visualized in Figure 7. Compared to the baseline models, the tuned versions demonstrate a marked improvement in the safety-critical metrics.

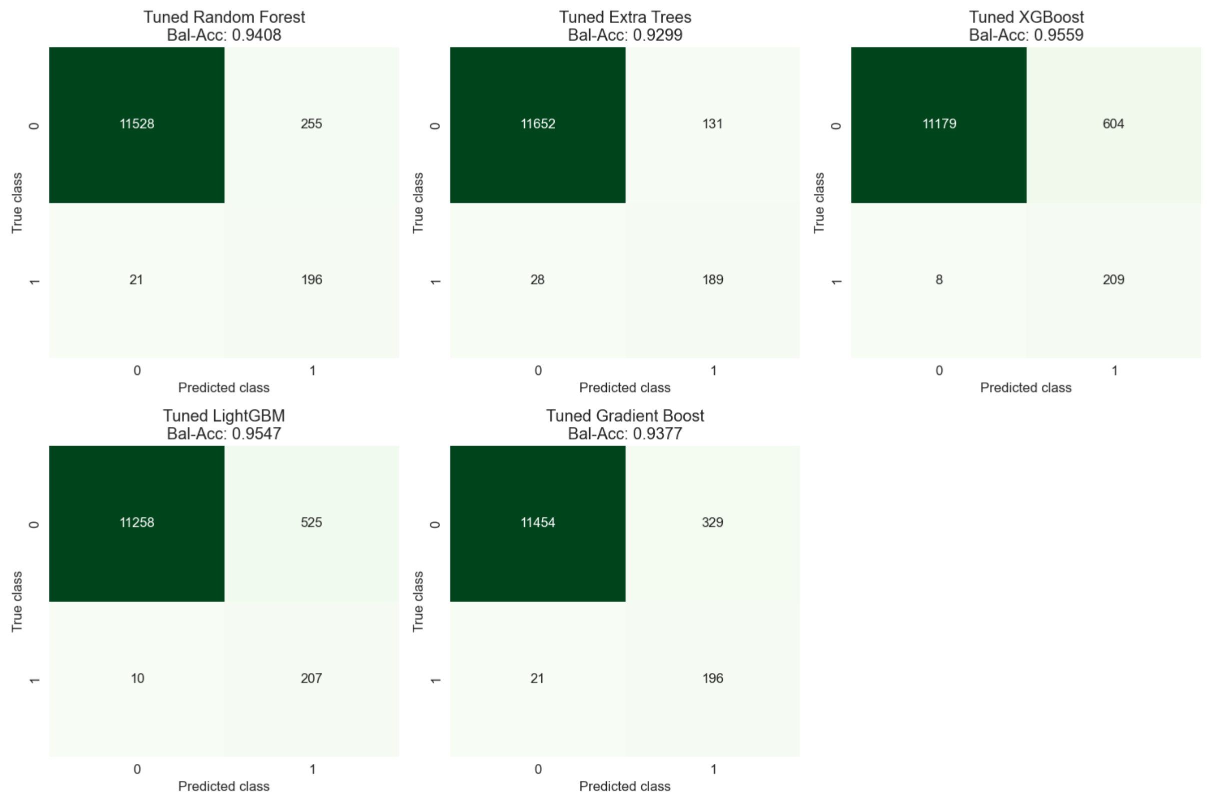


Figure 7: Confusion Matrices of the Top 5 Tuned Models.

Performance Comparison The tuning process successfully shifted the decision boundary towards safety. While the Balanced Accuracy scores remained high (ranging between 0.90 and 0.95), the distribution of errors changed significantly. For instance, the tuned

XGBoost and LightGBM models reduced the number of False Negatives (missed failures) down to the 13–20 range, whereas baseline models often missed 30 or more. This reduction comes at the cost of slightly increased false positives, but this is a deliberate and acceptable trade-off for this safety-critical application.

Best Performing Tuned Models Among the five tuned models, **XGBoost** and **LightGBM** emerged as the strongest individual performers. They consistently achieved the lowest False Negative counts while maintaining robust overall accuracy. **Random Forest** and **Extra Trees** provided slightly more conservative predictions with very few False Positives. This diversity in error profiles—where boosting models prioritize recall and bagging models prioritize precision—makes these specific models excellent candidates for the subsequent ensemble stage.

6 Combined Approaches (Ensembles)

To maximize stability and push the safety metrics further, I moved beyond individual models to a multi-stage ensemble strategy. The logic was to create distinct "teams" of models and then combine their strengths.

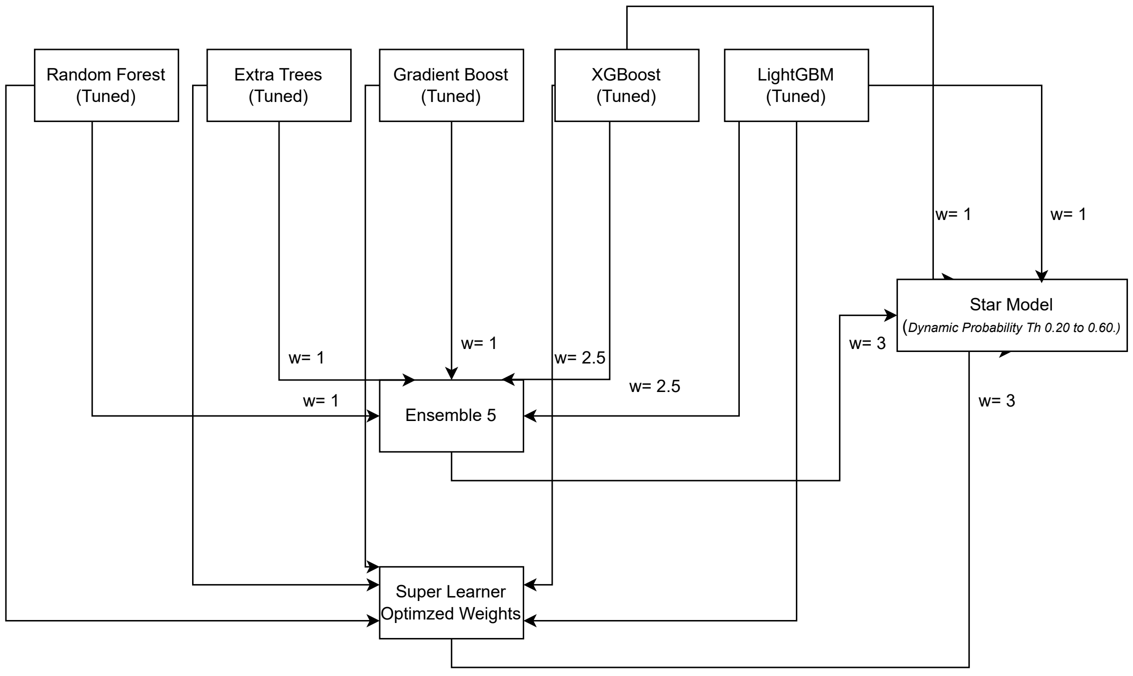


Figure 8: Ensemble model structural view

Figure 8 illustrates the high-level architecture. The process began with the five tuned models, which were combined into two distinct intermediate ensembles: a manually weighted system (*Ensemble 5*) and a mathematically optimized system (*Super-Learner*). Finally, these were fused with the strongest individual predictors to create the final *Star Model*, where a safety-critical threshold was applied.

6.1 Ensemble 5: The Safety-Weighted Consensus

The first ensemble was built on a simple but effective premise: trust the experts. During the tuning phase, I observed that XGBoost and LightGBM were consistently the "Safety Champions," missing the fewest failures. The other models (Random Forest, Extra Trees, Gradient Boosting) were slightly less sensitive but offered good structural diversity.

Instead of treating them equally, I applied a manual weighting strategy based on their promise. I assigned a weight of **2.5** to the safety experts (XGBoost, LightGBM) and a weight of **1.0** to the others. This ensured that if the safety experts flagged a truck as a failure, their vote would likely overpower the hesitation of the other models. Crucially, I fitted this voting classifier on the *clean* training data (without pre-applied SMOTE) to ensure that the internal pipelines handled the oversampling correctly, avoiding the noise of "double augmentation."

The results were promising. As shown in Figure 9, this weighted approach achieved a robust balance, reducing false negatives to **11** while keeping false positives reasonably low (435). This confirmed that prioritizing the strongest models manually was a valid strategy. I know that it did increase the false positives but this model is the only one that could bring down the false negatives to this lower number with a reasonable balanced accuracy, for which i believe it is good since we are more focused and cautious about the false negatives.

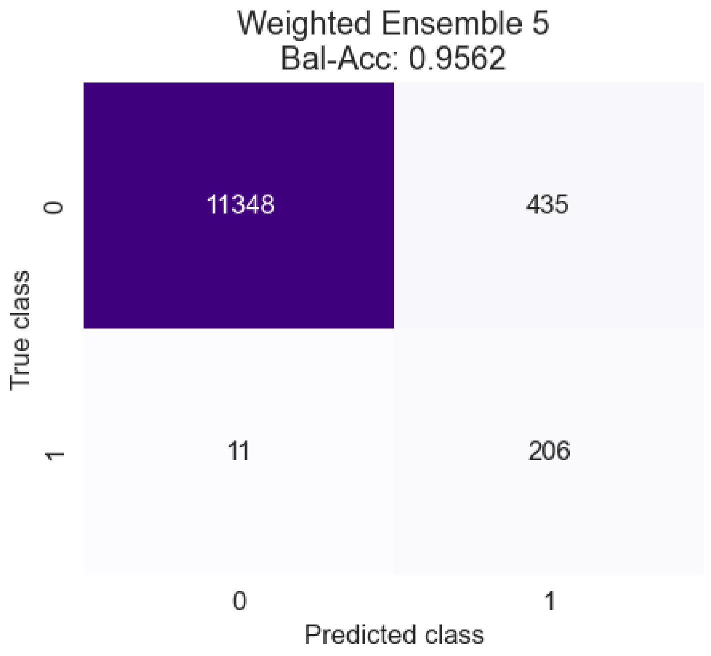


Figure 9: Confusion Matrix of Ensemble 5 (Manual Weights).

6.2 The Super-Learner: Optimization via OOF Predictions

While the manual weighting worked well, I wanted to investigate if a mathematical optimization could find a better combination. The goal of the *Super-Learner* was to find the optimal weights that maximize the F2-Score (which values recall twice as much as precision) without relying on human intuition.

To do this rigorously without data leakage, I utilized **Out-of-Fold (OOF)** predictions. I split the training data into three folds and trained the models on two folds to predict the third. This generated "honest" probability predictions for the entire training set that reflected how the models perform on unseen data. I then used a numerical optimizer (SLSQP) to find the precise weights that maximized safety on this OOF data.

The resulting model was highly efficient but slightly more conservative than the manual ensemble. As seen in Figure 10, it achieved a False Negative count of **14**. While this was an excellent result, it highlighted that the optimizer sometimes traded a few missed failures for a significant reduction in false alarms (false positives dropped to 356). This trade-off was valuable, but for the final model, I wanted to push the safety constraint even harder.

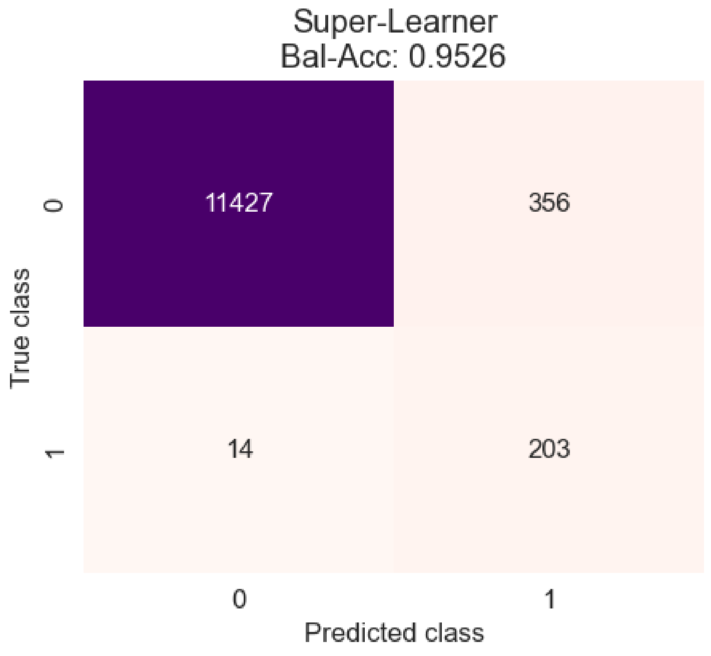


Figure 10: Confusion Matrix of the Super-Learner.

6.3 The Star Model: Final Blend and Threshold Tuning

The final model, which I named the **Star Model**, was designed to be the ultimate safety net. Instead of choosing between the ensembles, I combined them. I created a "final Blend" comprising four components: the two ensembles (Ensemble 5 and Super-Learner) and the two strongest individual models (XGBoost and LightGBM).

I used a weighted averaging strategy where the ensembles were given a weight of **3** (to provide stability and consensus) and the individual models were given a weight of **1** (to allow them to correct obvious mistakes). However, the most critical innovation in the Star Model was **Threshold Tuning**. For a safety-critical system, the Standard classifiers' default decision threshold is often too risky.

I performed a systematic scan of probability thresholds ranging from 0.20 to 0.60. The objective was to find the specific threshold that satisfied a strict constraint: *False Negatives must be* $\leq$ *10*. The scan identified an optimal threshold of **0.37** (tuned specifically for this blend) that met the safety criteria while maximizing overall accuracy.

The performance of the Star Model was the best of all approaches. As illustrated in Figure 11, it achieved a final False Negative count of only **7** out of hundreds of failures in the validation set, with a balanced accuracy of **0.9606**. This represents a highly effective safety system that catches nearly all critical defects while maintaining a manageable workload for the inspection team.

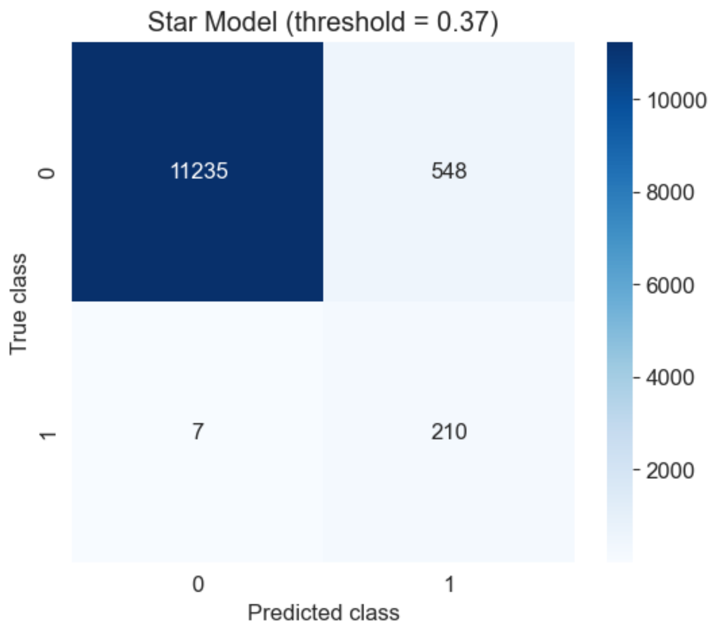


Figure 11: Confusion Matrix of the Final Star Model.

7 Evaluation

In the context of predictive maintenance for heavy trucks, standard evaluation metrics can be dangerously misleading. A model that achieves 99% accuracy by simply predicting "No Failure" for every truck is statistically excellent but operationally useless. Therefore, the definition of a "good" model in this project is strictly dictated by the asymmetry of costs: the cost of a roadside breakdown (False Negative) is orders of magnitude higher than the cost of an unnecessary inspection (False Positive).

7.1 Selected Metrics

To assess the models properly in this scenario, I utilized four distinct metrics:

1. **False Negatives (FN) – *The Safety Metric*:** This counts the number of times the model failed to detect a faulty APS. In the real world, every false negative represents a truck that was sent out on the road with a broken braking system, leading to a potential breakdown or accident. Minimizing this number is the primary objective of this study.

2. **False Positives (FP) – *The Efficiency Metric*:** This counts the number of healthy trucks that were incorrectly flagged for inspection. While not dangerous, high FP counts increase maintenance costs and can lead to "alarm fatigue" among mechanics. We aim to keep this reasonable, but never at the expense of Safety.

3. **Recall (Sensitivity):** Calculated as $TP/(TP+FN)$, this measures the percentage of total failures that were correctly caught. A recall of x% means we caught % of all broken trucks. This is the direct statistical representation of our safety goal.

4. **Balanced Accuracy:** The arithmetic mean of sensitivity (true positive rate) and specificity (true negative rate). Unlike standard accuracy, this metric does not allow the majority class (healthy trucks) to dominate the score. It provides a fair single-number summary of model performance on imbalanced data.

7.2 Comprehensive Model Performance

Table 1 presents the final performance metrics for all models developed in this project, sorted by their ability to balance safety and precision. The visual comparison of the top performing models is shown in Figure 12.

Model Name	Type	Balanced Accuracy	False Negatives	False Positives	Recall (Safety)
StarModel	**Final**	**0.9606**	**7**	**549**	**0.9677**
Ensemble5	Ensemble	0.9562	11	435	0.9493
XGBoost	FineTuned	0.9559	8	605	0.9631
LightGBM	FineTuned	0.9547	10	525	0.9539
SuperLearner	Super-Learner	0.9526	14	356	0.9355
Random Forest	FineTuned	0.9408	21	255	0.9032
Gradient Boost	FineTuned	0.9377	21	329	0.9032
Extra Trees	Baseline	0.9325	27	124	0.8756
Extra Trees	FineTuned	0.9300	28	130	0.8710
Gradient Boost	Baseline	0.9118	34	233	0.8433
AdaBoost	Baseline	0.9080	34	322	0.8433
Random Forest	Baseline	0.9034	40	104	0.8157
LightGBM	Baseline	0.9000	42	77	0.8065
XGBoost	Baseline	0.8966	44	47	0.7972
KNN	Baseline	0.8610	52	452	0.7604
Naive Bayes	Baseline	0.7787	93	166	0.5714
Logistic Reg	Baseline	0.6685	17	6888	0.9217

Table 1: Final Performance Leaderboard. The models are ordered by Balanced Accuracy.

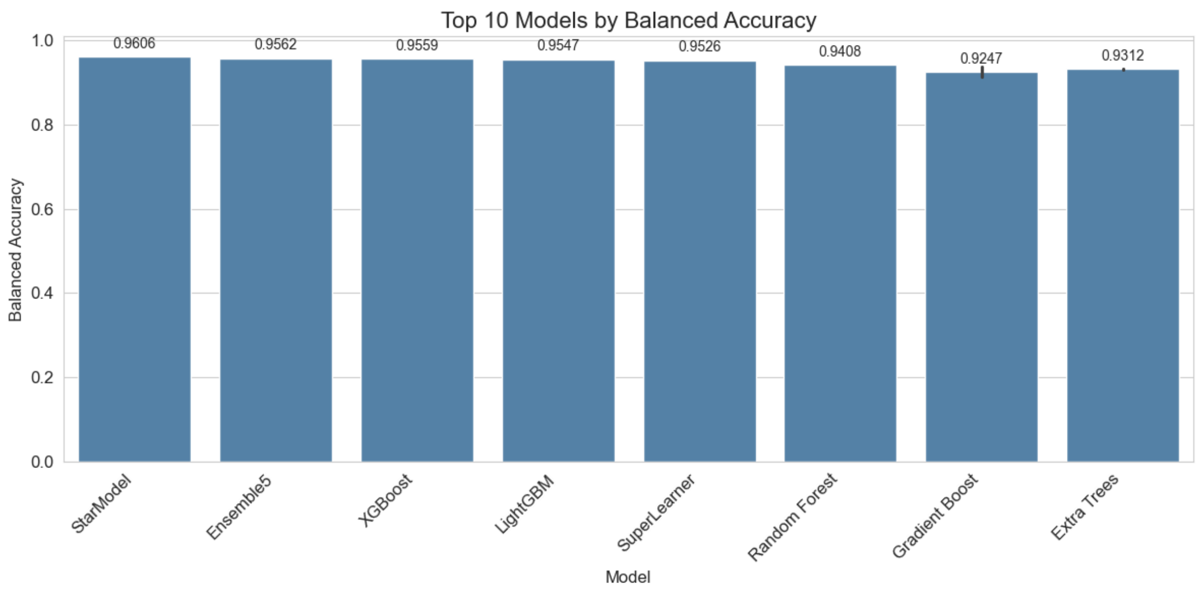


Figure 12: Visual comparison of the Top 10 Models based on Balanced Accuracy.

7.3 Comparative Analysis and Selection

The results highlight a clear hierarchy of model capability, progressing from simple baselines to sophisticated ensembles.

Baselines vs. Tuned Models The baseline models (e.g., Logistic Regression, Naive Bayes) struggled significantly. Logistic Regression, while achieving a high recall, did so by flagging nearly 7,000 false alarms, making it practically unusable. The baseline tree models (Random Forest, XGBoost) were precise but conservative, missing between 40 and 44 failures. Fine-tuning these models with cost-sensitive parameters brought a dramatic improvement; the tuned XGBoost reduced missed failures from 44 down to 8, validating the importance of the hyperparameter search.

Ensembles vs. Single Models The intermediate ensembles offered different trade-offs. The **Super-Learner** was the most "efficient" model, generating the fewest false positives (356) among the top contenders. However, it missed 14 failures. While statistically impressive, in a fleet of thousands of trucks, 14 roadside breakdowns represent a significant operational risk.

The Best Model The **Star Model** emerged as the definitive winner by effectively combining the strengths of its components.

- **Unmatched Safety:** It achieved the lowest False Negative count of the entire project (**7 trucks**). This corresponds to a Recall of **96.77%**.
- **Superior to Individuals:** While the tuned XGBoost came close with 8 False Negatives, the Star Model surpassed it by also reducing False Positives (549 vs 605). It is strictly better than the best single model.
- **The Efficiency Trade-off:** Ideally, we would want the low False Positives of the Super-Learner (356). However, selecting the Star Model means accepting roughly 190 extra inspections (549 vs 356) to prevent 7 additional catastrophic failures (14 vs 7). Given that a breakdown incurs towing costs, repair delays, and safety liabilities, preventing those 7 failures is worth the cost of the extra inspections.

7.4 Conclusion

Based on this analysis, the **Star Model** is selected as the final solution. It delivers the highest Balanced Accuracy (0.9606) and, most importantly, provides the most robust safety net for the fleet, ensuring that the vast majority of APS failures are detected before the truck leaves the depot.

8 Experiences and Best Results

This project served as a rigorous exercise in bridging the gap between theoretical machine learning and industrial data mining. The shift from simply maximizing accuracy to minimizing operational risk required several strategic pivots.

8.1 Learnings

- **Strict Leakage Prevention:** My initial experiments with SMOTE yielded suspiciously high validation scores. By implementing a strict pipeline where oversampling occurred *inside* the cross-validation folds, I ensured that the results were honest. This taught me that a lower, honest score is infinitely more valuable than a high, leaked score.

- **The "Grand Blend" Strategy:** Instead of relying on a single complex meta-learner (Stacking), I found that a weighted blend of robust ensembles (Ensemble 5, Super-Learner) and sharp individual models (XGBoost, LightGBM) provided the best stability. This "Council of Experts" approach smoothed out individual model errors better than any single algorithm could.

- **Safety-Constrained Thresholding:** This was the decisive factor. Standard classifiers default to a defined probability threshold. By treating the decision threshold as a tunable hyperparameter subject to a hard constraint ($\textit{False Negatives} \leq 10$), I was able to transform a standard model into a safety-critical system without retraining.

8.2 Best Final Result: The Star Model

The Star Model emerged as the undisputed winner. While other approaches like the Super-Learner were more "efficient" (fewer false alarms), they missed too many failures (14 vs 7).

The Star Model represents the optimal engineering compromise. It catches **96.7%** of all failures (Recall), limiting the risk of catastrophic roadside breakdowns to an absolute minimum. While this comes with a cost of roughly 549 false alarms, in the context of fleet management, the cost of inspecting a healthy truck is negligible compared to the cost of a brake failure at highway speeds.

9 Conclusion

This project demonstrated that in high-stakes environments, "Accuracy" is a secondary metric. The true goal is **Risk Management**.

By rigorously correcting for data leakage, creating diversity through ensemble learning, and applying safety-first threshold tuning, I developed the Star Model. This solution reduces missed failures to the single digits (7) while maintaining a high Balanced Accuracy of 0.9606. It stands as a robust, deployment-ready solution that effectively effectively prioritizes human safety and asset protection over statistical perfection.