Project Report: Predictive Maintenance using Sensor Data

By Promise Ekpo

1. Problem Statement & Business Context

Problem: Predict generator failures in wind-turbine equipment using sensor data so that components can be repaired before catastrophic failure, lowering maintenance and replacement costs.

Business context: ReneWind (a wind-energy company) provided a ciphered sensor data set. Predictive maintenance reduces expensive replacements by forecasting failures; repairing is much cheaper than replacing, and inspections are cheaper than repairs. The cost structure motivates maximizing detection of true failures (minimizing false negatives).

Source: Dataset as well as project Git hub repo is available from the below link: promwizy911/Predictive-Maintenance-using-Sensor-Data

2. Dataset Description

Source and structure: The data is a transformed/ciphered sensor data set (Train.csv and Test.csv). Each set contains 40 predictor variables (V1..V40) and a binary target (Target: 1=failure, 0=no failure). Train set: 20,000 rows; Test set: 5,000 rows. □

Key statistics and imbalance: The target distribution in the training set is 18,890 of '0' and 1,110 of '1' (≈5.6% failures), so the data set is highly imbalanced. Summary statistics (means, std, ranges) and uni-variate plots were used during EDA. □

Missing data & challenges: Two features, V1 and V2, contained 18 missing values each; other columns were complete. Features are numerical (float64) except Target (int). Correlations between features are generally weak with a few moderate relationships identified. □□

Limitations / biases: Data set is ciphered (feature names obfuscated) so domain-meaning interpretation of individual predictors is not possible from the file alone. The imbalance (few failure cases) requires careful sampling or algorithmic treatment to avoid biased models.

3. Data Preprocessing & Feature Engineering

Splits used in development: An initial stratified split created training and validation sets (X_train: 14,000; X_val: 6,000). For the final pipeline an 80/20 split was also illustrated (X_train: 4,000; X_test: 1,000) when building the production pipeline. □filecite□turn1file17□turn1file2□

Missing-value handling: KNNImputer (k=5) was used to impute missing values in V1 and V2 on training and validation sets to avoid leakage. After imputation there were no missing values left.

Sampling / imbalance handling: The team experimented with (a) training on original data, (b) oversampling minority class using SMOTE, and (c) under sampling using RandomUnderSampler. Under-sampling produced models with better recall and less over fitting for some algorithms; oversampling improved recall but increased over fitting in others.

Feature engineering: The data set was already transformed/ciphered; EDA identified feature relationships (e.g., strong correlations among V15–V7 and negative correlation between V7 and V18) used to guide modelling. The final pipeline used ColumnTransformer + KNN imputer to wrap preprocessing. No explicit domain-derived features are added in the notebook.

4. Model Architecture & Approach

Model family: The project focuses on classical supervised classifiers (tree-based ensembles and linear models). Models trained and compared include DecisionTree, Bagging, RandomForest, GradientBoosting (GBM), AdaBoost, XGBoost, RidgeClassifier, LogisticRegression (and Lasso).

Why GBM was chosen as final approach: Gradient Boosting (GBM) showed strong recall performance across sampling strategies and, when tuned and combined with under-sampling, produced the best generalization (high recall with relatively low over-fitting). The choice is empirical — GBM provided the best trade-off for maximizing recall (the priority metric). ▯

Conceptual workflow (preprocessing → sampling → model):

1. Read Train.csv / Test.csv → 2. EDA & missing-value detection → 3. Impute V1,V2 with KNNImputer → 4. Optionally resample data (SMOTE or undersample) → 5. Train candidate models with stratified K-Fold CV optimizing Recall → 6. Hyperparameter tuning (Randomized / Grid search) → 7. Final pipeline assembly with preprocessing + undersampling + tuned GBM → 8. Evaluate on held-out test set. ▯

5. Training Strategy

Train / validation approach: A stratified K-Fold (k=5) cross-validation was used to assess models on training data. An explicit train/validation split was also used (70% train, 30% validation) for rapid evaluation. For final pipeline testing an independent hold-out test set was used.

Scoring metric and loss: Recall (sensitivity) was selected as the primary scorer because minimizing false negatives (missed failures) has the largest business cost. Scorer used in cross-validation and hyperparameter search = recall_score.

Hyperparameter tuning: RandomizedSearchCV and GridSearchCV were used for GBM, RandomForest and XGBoost across original, over-sampled and under-sampled data sets. Final tuned hyperparameters for GBM (under-sampled best): learning_rate=0.15, max_features=0.67, n_estimators=87, subsample=0.2. Random and grid searches reported cross-validated recall scores for candidate parameter sets.

Notes on optimizers / loss: The models used (tree ensembles and XGBoost) rely on their internal boosting objectives (log-loss / deviance for GBM), and XGBoost used its standard objectives (eval_metric='logloss' shown). The notebook did not apply neural-network optimizers. □

6. Model Evaluation

Primary metrics reported: Accuracy, Recall, Precision, and F1. Recall was prioritized. A confusion-matrix helper and a model_performance function were used to compute and tabulate these metrics. □□

Highlights of results (representative numbers from the notebook):

- Best baseline XGBoost on original data: validation recall ≈ 0.825. □□

- GBM tuned on original data (GridSearch/Randomized): validation recall ≈ 0.79–0.80 but with overfitting observed. □

- GBM trained on oversampled data achieved CV recall ≈ 0.97 (training/oversampled CV) but showed signs of overfitting; validation recall lower (~0.81–0.88 depending on tuning).

- GBM trained and tuned on undersampled data produced strong recall with reduced overfitting; example tuned undersampled GBM produced CV recall ≈ 0.88–0.90 and validation recall ≈ 0.88–0.89. This was judged the best generalizing model. □

Final pipeline evaluation on test split: The assembled pipeline (ColumnTransformer KNN imputer → undersampling transformer → GBM with tuned parameters) evaluated on the held-out test set reported: Accuracy=0.964, Recall=0.625, Precision=0.700, F1=0.660. (These are the notebook's reported test metrics for the final pipeline run.) □filecite□turn1file2□

Comparison with baselines: The project compared multiple algorithms (DecisionTree, Bagging, RandomForest, GBM, XGBoost, logistic/ridge/lasso). GBM (with appropriate sampling/tuning) and XGBoost were consistently top performers for recall. Undersampling + tuned GBM offered better generalization versus over-sampled variants that tended to over-fit.

7. Deployment / MLOps (If Applicable)

Reproducible pipeline: The notebook assembles an sklearn Pipeline that encapsulates preprocessing (KNN imputation via ColumnTransformer), an undersampling FunctionTransformer, and the final tuned GradientBoostingClassifier. This pipeline structure supports serialization (e.g., via joblib/pickle) and is deployment-friendly.

CI/CD & monitoring: The project demonstrated pipeline creation and evaluation but did not include explicit CI/CD scripts, model registry steps, or runtime monitoring dashboards. Those would be recommended next steps but were not done in the project.

Scalability considerations: Tree ensembles are CPU-bound at training time but fast at inference and can be served in a standard model-serving environment (REST API, batch jobs). The use of a single pipeline simplifies reproducible preprocessing during serving. The project itself does not include auto-scaling or production monitoring code.

8. Key Insights & Business Impact

Top predictors: The most important features in the final model are V18 (top), V36, V15, V3, and V26 — these features correlate with failure risk and are actionable triggers for maintenance. fileciteturn1file2

Actionable rule-of-thumb (from EDA & model interpretation): Conditions such as V15 > -5 combined with V7 > 0, or V18 dropping below 0 with V36 below thresholds, significantly raise failure probability. The project suggested immediate maintenance when V15 increases above −5. These rules provide tangible operational triggers for inspection/repair.

Expected business impact: Using the model to flag likely failures allows pre-emptive repairs (cheaper than replacements), reducing downtime and lowering maintenance costs. The focus on Recall aligns model objectives with the business cost structure described.

9. Final Conclusion

Learning and challenges: The project shows that:

(a) careful treatment of class imbalance is critical.

(b) ensemble models (GBM/XGBoost) perform well for failure detection.

(c) oversampling can increase recall but risks over-fitting. Missing values were minimal and handled with KNN imputation. Feature names were ciphered, limiting domain interpretability.

Future scope and improvements:

- Collect richer labeled failure examples to reduce imbalance and increase confidence in rare-event prediction.

- Add domain-meaningful feature engineering if original sensor labels become available (e.g., aggregations over time, rolling statistics).

- Implement model versioning, automated CI/CD deployment, and production monitoring (drift detection, alerting).

- Consider cost-sensitive learning or custom cost function optimization to directly minimize expected maintenance + replacement costs. (Not implemented in the provided notebook.)