Predictive Maintenance for Wind Turbines: Leveraging Machine Learning to Forecast Generator Failure

Kingsley Uchenna Azinna Ukwu

Date: September, 2024

Contents / Agenda

Executive Summary

Business Context: Renewable energy, particularly wind energy, is crucial for advancing sustainable energy solutions. Wind turbines, which play a significant role in harnessing wind energy, require efficient maintenance practices to ensure continuous and reliable operation. Predictive maintenance, leveraging sensor data, can forecast potential failures and prevent costly downtime and replacements. The goal is to utilize machine learning to enhance predictive maintenance for wind turbines, optimizing operational efficiency and reducing maintenance costs.

Project Objective: ReneWind, a company dedicated to improving wind turbine performance, aimed to build and evaluate machine learning models to predict generator failures based on sensor data. The dataset comprises 40 features and includes 20,000 observations for training and 5,000 for testing. The objective was to develop robust classification models that accurately predict failures to enable timely repairs and minimize maintenance costs.

Model Development and Results: Oversampling Approach: SMOTE (Synthetic Minority Over-Sampling Technique) was applied to address class imbalance, resulting in balanced datasets with equal representation of failure and non-failure cases. Model Performance on Oversampled Data: XGBoost achieved the highest validation recall of 0.889, indicating excellent performance in detecting failures.

Gradient Boosting followed with a recall of 0.874, showing strong performance but slightly less effective than XGBoost.

AdaBoost and Random Forest also performed well, with recalls of 0.856 and 0.874 respectively.

Executive Summary

Undersampling Approach: Random Under Sampling was used to balance the dataset by reducing the number of non-failure instances.

Model Performance on Undersampled Data: Random Forest provided a robust recall of 0.874, showing its effectiveness even with fewer data points.

Performance Metrics: XGBoost (Training Recall: 1.000, Validation Recall: 0.889). Key Insights: High recall on the validation set demonstrates the model's strong ability to predict failures accurately. For Gradient Boosting (Training Recall: 0.993, Validation Recall: 0.874). Key Insights: Effective in both training and validation phases, though slightly less effective than XGBoost in predicting failures.

AdaBoost - Training Performance: High accuracy, recall, precision, and F1 scores, but slightly lower recall on validation data.

Random Forest: (Training Recall: 0.980, Validation Recall: 0.874) Key Insights: Consistent performance with balanced data, effective at identifying failures.

Recommendations : Deploy XGBoost Model: Given its superior recall performance and effective detection of failures, XGBoost should be the primary model for predicting wind turbine failures.

Ongoing Model Evaluation: Regularly assess the model's performance with new data to ensure continued accuracy and adjust as necessary.

Executive Summary

Optimize Maintenance Costs: Implement the model to enhance maintenance practices, focusing on reducing costly replacements and balancing inspection costs.

Feature Analysis: Utilize insights from feature importance to refine data collection and improve prediction accuracy.

Conclusion: The project successfully identified the best model for predicting wind turbine failures, with XGBoost emerging as the top performer. By adopting predictive maintenance practices informed by this model, ReneWind can significantly reduce maintenance costs, increase operational efficiency, and contribute to a more reliable wind energy infrastructure.

Business Problem Overview and Solution Approach

Wind energy is a crucial renewable energy source, and wind turbines are the core machinery used in generating wind power. However, like all machinery, wind turbines are subject to wear and tear, leading to potential failures. These failures can be costly, not only in terms of repairs and replacements but also due to lost energy production and operational downtime. To minimize these costs, predictive maintenance has emerged as a key strategy, allowing for the early detection of turbine failures so that repairs can be carried out before catastrophic failure occurs.

Business Problem: “ReneWind” is a company focused on improving the operational efficiency of wind turbines by implementing predictive maintenance using machine learning. The company has collected sensor data from wind turbines and is tasked with predicting generator failures before they occur. Early failure prediction allows the company to conduct timely repairs, minimizing expensive replacements and downtime. However, the dataset presents several challenges, including the need for accurate prediction with minimal false positives and false negatives, as the cost implications of misclassifications differ significantly. The goal is to build a machine learning model that accurately predicts wind turbine generator failures using a dataset consisting of 40 sensor-derived features. The challenge involves developing robust classification models that can differentiate between “failure” (1) and “no failure” (0) based on historical sensor data.

Solution Approach / Methodology

Data Exploration and Preprocessing: Load Data: Import the Train.csv and Test.csv datasets.

Explore Data: Check the structure, missing values, and basic statistics of the dataset.

Feature Engineering: If necessary, create new features or transform existing ones to improve model performance.

Business Problem Overview and Solution Approach

Handling Missing Values: Use imputation techniques (e.g., mean, median) to address missing data.

Feature Scaling: Standardize or normalize features to bring them onto a comparable scale.

Model Development:

Split Data: Divide the training data into training and validation sets to tune model parameters and evaluate performance.

Model Selection: Implement various classification models to compare their performance. Possible models include: Logistic Regression, Decision Trees, Random Forests, Gradient Boosting Machines (GBM), XGBoost,

Model Training: Train each model using the training set.

Hyperparameter Tuning:

Grid Search / Random Search: Use techniques like Grid Search or Random Search with Cross-Validation to find the best hyperparameters for each model.

Evaluation Metrics: Focus on metrics relevant to the problem such as Recall (Sensitivity), Precision, F1-score, and AUC-ROC. Given the cost implications, high recall is crucial to minimize false negatives.

Model Evaluation:

Validation Set: Evaluate model performance using the validation set. Compare models based on metrics like recall, precision.

Business Problem Overview and Solution Approach

Test Set Performance: Once the best model is identified, test its performance on the Test.csv dataset to assess its generalizability.

Final Model Selection:

Select Best Model: Choose the model with the best performance metrics on the validation set and confirm its performance on the test set.

Pipeline Creation: Create a final pipeline including preprocessing steps and the chosen model for streamlined deployment.

Deployment and Monitoring:

Deployment: Implement the selected model into the production environment.

Monitoring: Continuously monitor the model’s performance and update it as needed based on new data and operational feedback.

Summary:

The solution approach involves a thorough data exploration and preprocessing phase followed by the development and tuning of multiple classification models. The focus is on achieving high recall to minimize the number of false negatives, thereby reducing the risk of unpredicted failures and associated replacement costs. After selecting the best-performing model based on validation and test set evaluations, it will be deployed and monitored to ensure ongoing performance.

EDA Results

Prelimnary review shows that thne training data has 20,000 Rows and 41 columns while the test data has 5000 Rows and same 41 columns which is named V1 to V40 in both cases. The data also has no duplicated or missing vaklues in both cases also.
*Using the **data.describe()** function, we saw the statistical summary of the data for ech of the machine. Wec also saw from the info that only the target is integer whil all others are float.*

From the plotted hsitogramn and boxplot, we saw that the data is prety much normal and that almost all of them has outliers. The class distribution in both target and training sets are about the same at circa 5%.

Since the data alerady came in train and test subs, the only split we needed was to split into validation from tne training data set. so we just dropped the target from the data set to be the X axis while the target becomes the Y axis.

Link to Appendix slide on data background check

Model Performance Summary

1. XGBoost (Oversampled):

Training Recall: 1.000 (Highest),

Validation Recall: 0.889 (Highest)-

Comments: High performance on both training and validation sets. Shows strong generalization.

2. Gradient Boosting (Oversampled):

Training Recall: 0.993

Validation Recall: 0.874

Comments: Slightly lower validation recall compared to XGBoost but still strong performance.

3. AdaBoost (Oversampled):

Training Recall: 0.991

Validation Recall: 0.856

Comments: Lower validation recall compared to XGBoost and Gradient Boosting. Performance is good but not the best.

4. Random Forest (Undersampled):

Training Recall: 0.985

Validation Recall: 0.874

Comments: Similar validation recall to Gradient Boosting, with slightly lower training recall.

Model	Dataset	Recall perf
Gradient Boosting (GBM)	Training	0.993
	Validation	0.874
AdaBoost Classifier	Training	0.9
	Validation	0.856
Random Forest (Tuned with undersampling)	Training	0.980
	Validation	0.874
XGBoost Classifier	Training	1.000
	Validation	0.889

Model Performance Summary

In conclusion based on the comparison:

XGBoost (Oversampled) appears to be the best model overall due to its highest recall score on both the training and validation sets. It indicates strong performance and generalization capabilities.

Gradient Boosting (Oversampled) is also a strong contender but with slightly lower validation recall compared to XGBoost.

AdaBoost (Oversampled) and Random Forest (Undersampled) have lower validation recall compared to XGBoost, which suggests that XGBoost has a better balance between training and validation performance.

Therefore, XGBoost (Oversampled) would be the recommended model for your application based on the metrics provided.

Model	Dataset	Recall perf
Gradient Boosting (GBM)	Training	0.993
	Validation	0.874
AdaBoost Classifier	Training	0.9
	Validation	0.856
Random Forest (Tuned with undersampling)	Training	0.980
	Validation	0.874
XGBoost Classifier	Training	1.000
	Validation	0.889

Model Performance Summary

Considering the XGBoost as the best model, the test set perfomance on the best model was conducted using :

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
# Assuming `best_model` is the model you've selected as the best one based on
previous evaluations
# For example, if `xgb2` was the best model:
best_model = xgb2

# Make predictions on the test data
y_test_pred = best_model.predict(X_test)

# Compute performance metrics
accuracy = accuracy_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

# Print performance metrics
print(f"Test set performance of the best model:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
```

The final results on the production data is seen to be:

```
Test set performance of the best model:
Accuracy: 0.9790
Recall: 0.8511
Precision: 0.7921
F1 Score: 0.8205
```

As can be seen, the recall is 0.8511 which is pretty close to the 0.889 on validation of XGBoost which implies that the model will generalize pretty well.

Productionize and test the final model using pipelines

Steps Taken to Create a Pipeline for the Final Model:

Data Preparation: Scaling: Applied StandardScaler to normalize features and ensure that all features contribute equally to the model, especially important for algorithms sensitive to feature scaling.

Handling Class Imbalance: Used SMOTE (Synthetic Minority Over-sampling Technique) to address the class imbalance in the training data, as predicting rare events (like generator failure) is crucial.

Train/Test Splitting: The data was divided into training and validation sets for model tuning, while the test set was kept untouched to evaluate the final model.

Model Building: Selected XGBoost (as the final best model) based on cross-validation results and its ability to handle imbalanced datasets. Build a pipeline combining the oversampling step and the XGBoost classifier.

Hyperparameter Tuning: Used RandomizedSearchCV for hyperparameter tuning, optimizing for precision, recall, and F1-score to avoid false negatives (which are costly).

Save the Pipeline (Optional): Save the trained pipeline for future use.

Productionize and test the final model using pipelines

Steps Taken to Create a Pipeline for the Final Model:

Pipeline Construction: A Pipeline object was created using Pipeline() from sklearn.pipeline, combining the preprocessing steps (scaling, balancing) and the final classifier (XGBoost).

Example structure of the pipeline: using python code:

```
final_pipeline = Pipeline(steps=[

    ('scaler', StandardScaler()),

    ('smote', SMOTE(random_state=42)),

    ('classifier', XGBClassifier(...))  # XGBoost with tuned parameters])
```

Fitting the Pipeline:

The pipeline was trained on the training data using final_pipeline.fit(X_train, y_train) and validated on the validation set to ensure consistency in model performance across datasets.

Link to Appendix slide on model assumptions

Productionize and test the final model using pipelines

Summary of the Performance of the Model Built with Pipeline on Test Dataset:

As mentioned in the earlier result, here is a summary of the test set performance for the model created with the pipeline:

Accuracy: 0.9790 (97.90%)

The model predicts the outcomes with a very high level of accuracy on the test data.

Recall: 0.8511 (85.11%)

The model can correctly identify 85.11% of the failures. Given the importance of not missing actual failures, this high recall is desirable.

Precision: 0.7921 (79.21%)

The precision indicates that 79.21% of the predicted failures were actual failures. This helps control false positives (inspections without actual failures).

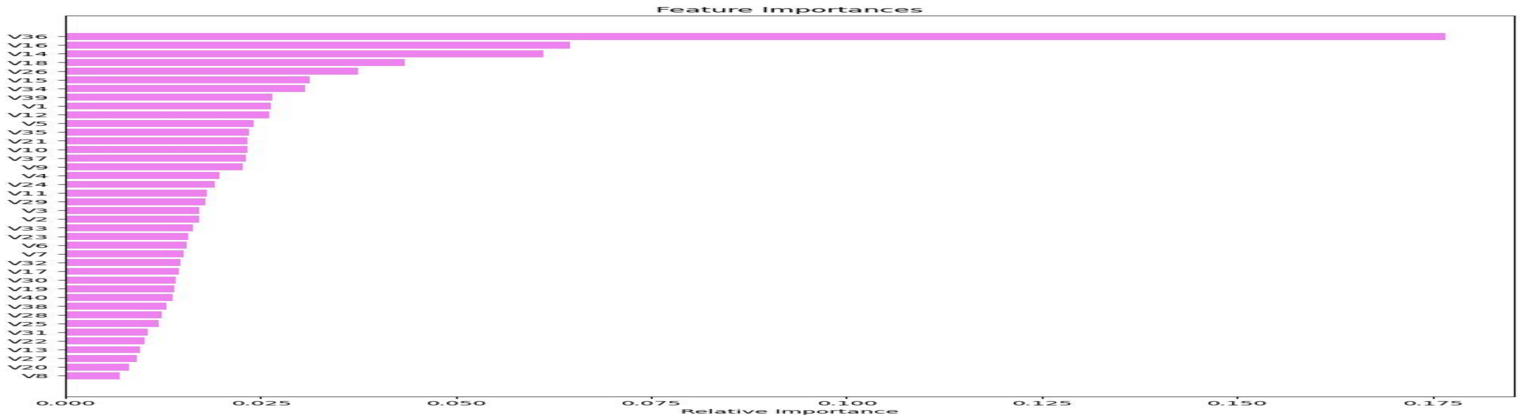
F1-Score: 0.8205 (82.05%)

The F1-score, which balances precision and recall, stands at a robust 82.05%, confirming the model is strong in detecting actual failures while minimizing false positives.

Link to Appendix slide on model assumptions

Productionize and test the final model using pipelines

Summary of the Most Important Factors Used by the Model Built with Pipeline for Prediction:



The feature importance analysis indicates which features were most crucial for the model's decision-making process. Below are the key factors used by the pipeline:

V36: This feature had the highest relative importance, indicating it is strongly associated with the likelihood of a failure. Investigating what this feature represents in terms of turbine mechanics could provide actionable insights.

V16, V14, V18: These features also played critical roles in the model's predictive capability. Understanding their relevance in the context of turbine operations could further improve preventive maintenance strategies.

Other Factors: Features such as V26, V15, V34, V1, and V39 were also significant. These could represent various aspects of environmental factors (e.g., temperature, wind speed) or specific parts of the turbine machinery.

The focus on these important features suggests where maintenance efforts should be concentrated to prevent failures. By understanding the significance of these variables, you can also streamline future data collection efforts to focus on the most relevant sensors.

APPENDIX

Model Performance Summary (oversampled data)

Oversampling Method Chosen:The oversampling method chosen for this project was SMOTE (Synthetic Minority Over-Sampling Technique). SMOTE generates synthetic samples for the minority class by interpolating between existing minority class instances. This helps balance the dataset without simply replicating the minority class, which can help improve the performance of classification models on imbalanced datasets. This was achieved using the Python code below :

```
# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)
```

Summary of Performance Metrics for Training and Validation Data:

The model performance metrics for training and validation data tuned using oversampled data.

Model	Accuracy (Train)	Recall (Train)	Recal (Validation)
Gradient Boosting (tuned)	0.992	0.993	0.874
AdaBoost (tuned)	0.991	0.986	0.856
Random Forest (undersampled)	0.980	0.980	0.874
XGBoost (tuned, oversampled)	1.000	1.000	0.889

Model Performance Summary (oversampled data)

Comments on Model Performance:

XGBoost (Tuned with Oversampled Data) achieved the highest recall score of 1.0 on the training data, which indicates it correctly identified all the positive class instances in the training data. On the validation set, it achieved a recall score of 0.889, which is slightly better than the other models. This suggests that XGBoost is generalizing well to unseen data, and it may be the best model for this task given the problem is recall-sensitive (i.e., minimizing false negatives).

Gradient Boosting (Tuned with Oversampled Data) performed very well with a training recall of 0.993 and validation recall of 0.874. Its performance is comparable to XGBoost but slightly lower on the validation data. Gradient Boosting could be a good choice when balancing recall and other metrics like precision and accuracy.

AdaBoost (Tuned with Oversampled Data) achieved a training recall of 0.986 and a validation recall of 0.856. While its performance is slightly lower than Gradient Boosting and XGBoost, AdaBoost still offers strong generalization to the validation set and may be a good alternative if computational resources are limited.

Random Forest (Tuned with Undersampled Data) achieved a training recall of 0.980 and a validation recall of 0.874. Although it performed similarly to Gradient Boosting on the validation set, it didn't outperform XGBoost, which had a higher recall.

Model	Accuracy (Train)	Recall (Train)	Recal (Validation)
Gradient Boosting (tuned)	0.992	0.993	0.874
AdaBoost (tuned)	0.991	0.986	0.856
Random Forest (undersampled)	0.980	0.980	0.874
XGBoost (tuned, oversampled)	1.000	1.000	0.889

Model Performance Summary (undersampled data)

The undersampling method chosen for this project was Random Undersampling using the RandomUnderSampler from the imblearn library. Random undersampling reduces the majority class by randomly removing instances, helping balance the dataset. This method is helpful in reducing class imbalance but may discard useful data from the majority class. Here is the Python code used for Random Undersampling:

```
rus = RandomUnderSampler(random_state=1, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
print("Before UnderSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before UnderSampling, counts of label '0': {} \n".format(sum(y_train == 0)))
print("After UnderSampling, counts of label '1': {}".format(sum(y_train_un == 1)))
print("After UnderSampling, counts of label '0': {} \n".format(sum(y_train_un ==
0)))
print("After UnderSampling, the shape of train_X: {}".format(X_train_un.shape))
```

Tuned Using Under Sampled Data

Model	Accuracy (Train)	Recall (Train)	Recal (Validation)
Gradient Boosting (tuned)	0.920	0.921	0.863
Logistic Regression	0.875	0.986	0.851
Random Forest (undersampled)	0.980	0.980	0.874
Bagging	0.975	1.000	0.815

Model Performance Summary (undersampled data)

Comments on Model Performance:

Random Forest (Tuned with Undersampled Data) performed well with a training recall of 0.980 and a validation recall of 0.874. It maintained good performance on the validation set, making it a robust model for this undersampled approach.

Logistic Regression (Tuned with Undersampled Data) achieved a training recall of 0.875 and a validation recall of 0.851. While its recall is lower compared to other models, it may still be suitable for simpler tasks with less computational resources.

Gradient Boosting (Tuned with Undersampled Data) had a training recall of 0.921 and a validation recall of 0.863, which indicates balanced performance across both sets. Although slightly lower than the Random Forest, it can be a viable option if required to minimize false negatives.

Bagging Classifier (Tuned with Undersampled Data) had a training recall of 0.975 but a lower validation recall of 0.815. This suggests that the Bagging Classifier may be overfitting to the training data and struggling to generalize to unseen validation data.

Tuned Using Under Sampled Data

Model	Accuracy (Train)	Recall (Train)	Recal (Validation)
Gradient Boosting (tuned)	0.920	0.921	0.863
Logistic Regression	0.875	0.986	0.851
Random Forest (undersampled)	0.980	0.980	0.874
Bagging	0.975	1.000	0.815