ReneWind Failure Detection

Pipeline and Hyperparameter Tuning

3/22/2024

Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model performance summary for hyperparameter tuning.
- Model building with pipeline
- Appendix

Executive Summary

Actionable Insights:

- **Prioritize Recall:** The model's ability to catch as many actual failures (true positives) as possible is crucial to avoid expensive generator replacements. Therefore, continue to focus on recall without overly compromising precision.
- **Address Class Imbalance:** Given the imbalance in the target variable, continue using techniques such as SMOTE or adjust the class weight parameter within XGBoost to handle this.
- **Model Deployment Strategy:** Implement the model within a real-time monitoring system to provide predictive alerts on potential failures, enabling proactive maintenance actions.
- **Threshold Adjustment:** Depending on the costs associated with false positives and false negatives, consider adjusting the threshold for classifying a failure to fine-tune the balance between recall and precision.
- **Continuous Learning:** Set up a mechanism where the model can be retrained periodically with new data to adapt to changing conditions and prevent model drift.

Recommendations:

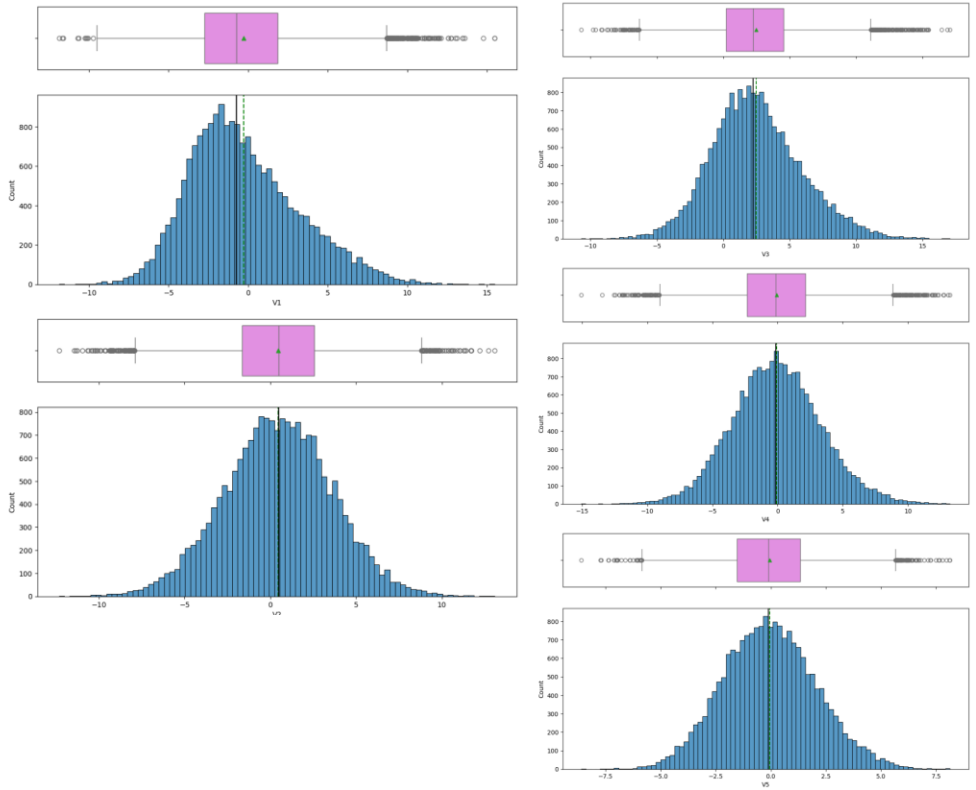
- **Integrate with Maintenance Schedules:** Use model predictions to schedule maintenance more effectively, thus minimizing downtime and extending the lifespan of the turbines.
- **Cost-Benefit Analysis:** Quantify the cost savings from reduced downtime and averted failures versus the costs of increased inspections due to false positives.
- **Risk Management:** Develop a risk assessment framework that uses model predictions to prioritize maintenance resources towards turbines with the highest risk of failure.
- **Feedback Loop:** Implement a feedback system where the maintenance team can confirm or refute model predictions, allowing for continuous model refinement.
- **Sensor Data Quality Assurance:** Ensure that the quality of sensor data is maintained, as the model's predictions are only as good as the data inputted.
- **Expand Scope of Predictive Analytics:** Explore the use of model predictions in optimizing energy production and in other operational aspects beyond maintenance.

Business Problem Overview and Solution Approach

- **Problem Definition**
  - Predict wind turbine failures using sensor data.
  - Dataset: 40 predictors, 20,000 training, and 5,000 test observations.
  - Minimize costly downtime by accurately predicting failures.
- **Solution Approach**
  - Preprocess Data: Clean and prepare data.
  - Model Building: Use classification models (Logistic Regression, Decision Trees, Random Forest, Gradient Boosting).
  - Class Imbalance: Apply SMOTE for oversampling or undersampling.
  - Evaluate Models: Prioritize recall to reduce false negatives.
  - Hyperparameter Tuning: Optimize model parameters.
  - Final Evaluation: Test model on unseen data focusing on recall.
  - Deployment: Implement and monitor the predictive model.
  - Goal: Enhance operational efficiency by preventing failures, thus saving on repair/replacement costs.

EDA Results

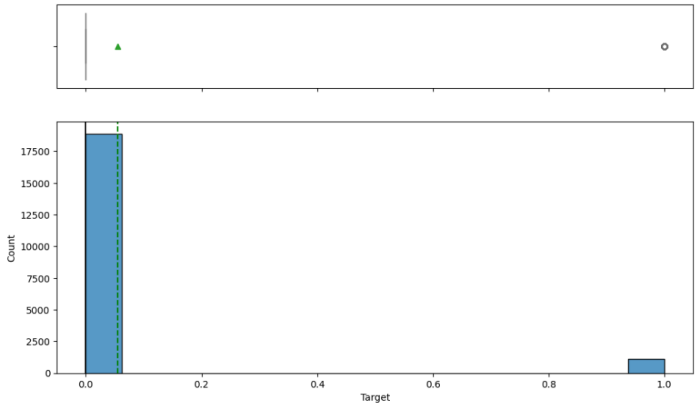
- **Symmetrical Distributions:** Many histograms exhibit a symmetrical shape around the central value, suggesting a normal-like distribution.
- **Consistency Across Variables:** The shapes of the histograms are fairly consistent across different variables, which could imply similar scale or type of data, possibly standardized features.
- **Limited Outliers:** There don't appear to be significant outliers in most variables, as there are no long tails extending from the bulk of the distributions.
- **Peakedness:** Some histograms show a peaked distribution (leptokurtic), which means there is a high frequency of values around the mean, and thin tails.



Link to Appendix slide on data background check

EDA Results

- **Imbalanced Classes:** There is a significant class imbalance with a much larger number of "0" (No failure) cases compared to "1" (Failure) cases.
- **Data Distribution:** The majority of the observations indicate no failure (0), suggesting that actual failures are relatively rare events in the dataset.
- **Training vs. Testing:** The class imbalance is consistent between both the training and test datasets.
- **Potential Model Training Implication:** The imbalance may bias predictive models towards predicting "No failure" and may require techniques such as oversampling, undersampling, or adjusting class weights to properly learn the minority class.
- **Evaluation Metric Consideration:** Given the imbalance, accuracy may not be a reliable performance metric, and it would be more appropriate to consider other metrics such as recall, precision, F1-score.



Train Data

```
Target
0    18890
1     1110
Name: count, dtype: int64
```

Test Data

```
Target
0    4718
1     282
```

Link to Appendix slide on data background check

Data Preprocessing

- Duplicate value check
  - 0 duplicate values
- Missing value treatment
  - V1 has 5 missing values
  - Imputer used for missing values
- Feature engineering
  - Oversampling and undersampling
- Data preparation for modeling
  - Split train data into training and validation (75:25 ratio)
  - Split test data into X_test and y_test

Model Performance Summary

Training performance comparison:

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data	XGBoost tuned with oversampled data
Accuracy	0.993	0.991	0.990	0.999
Recall	0.993	0.986	0.980	1.000
Precision	0.993	0.996	1.000	0.999
F1	0.993	0.991	0.990	0.999

Validation performance comparison:

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data	XGBoost tuned with oversampled data
Accuracy	0.965	0.982	0.942	0.983
Recall	0.830	0.856	0.874	0.885
Precision	0.638	0.816	0.478	0.816
F1	0.721	0.835	0.618	0.849

- All models perform well on train data
- Highest accuracy: AdaBoost Tuned
- Highest Recall: XGBoost Tuned
- Highest Precision: AdaBoost Tuned, XGBoost Tuned
- Highest F1-Score: 0.849

- **Final Model:** XGBoost Tuned
  - Highest recall and balanced precision and accuracy scores
  - Given the nature of the objective, a high recall score is the most beneficial

Link to Appendix slide on model assumptions

Productionize and test the final model using pipelines

Pipeline Process

1. Create a pipeline for XGBoost Tuned model
2. Separate target variable from other variables in train data and test data
3. Imputer for missing value treatment
4. Use SMOTE for oversampling
5. Fit the pipeline model to the oversampled data

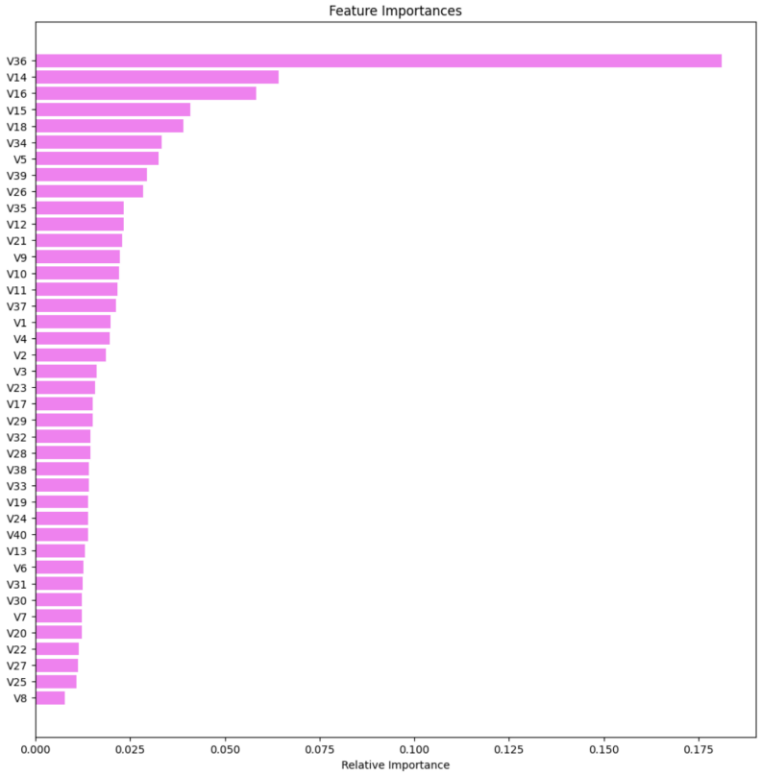
- Pipeline model performance on test set:
  - Again, high recall and balanced F1-Score

	Accuracy	Recall	Precision	F1
0	0.977	0.855	0.768	0.809

Link to Appendix slide on model assumptions

Productionize and test the final model using pipelines

- Summary of most important features

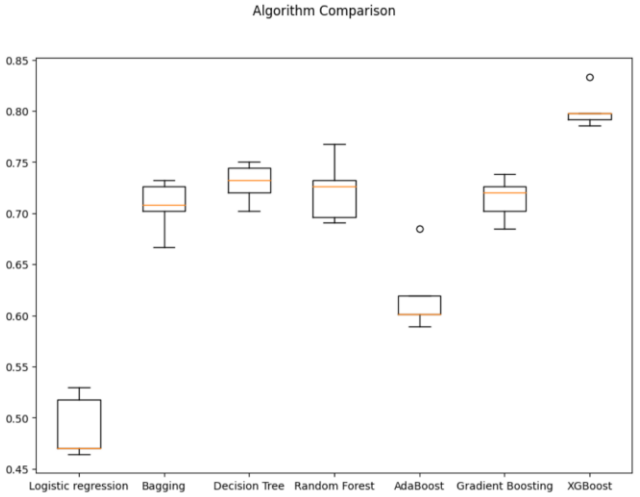


Link to Appendix slide on model assumptions

APPENDIX

Model Performance Summary (original data)

- **Top Performers:** XGBoost & Gradient Boosting lead in recall scores.
- **Consistency:** XGBoost shows high recall consistently across training and validation.
- **Moderate Performers:** Bagging & Random Forest have similar recall; Random Forest shows variability.
- **Underperformers:** Logistic Regression has the lowest recall; Decision Tree struggles with generalization.
- **Overfitting Indicator:** AdaBoost shows a performance drop from training to validation.
- **General Insight:** High variability in model performance suggests further tuning needed for stability.

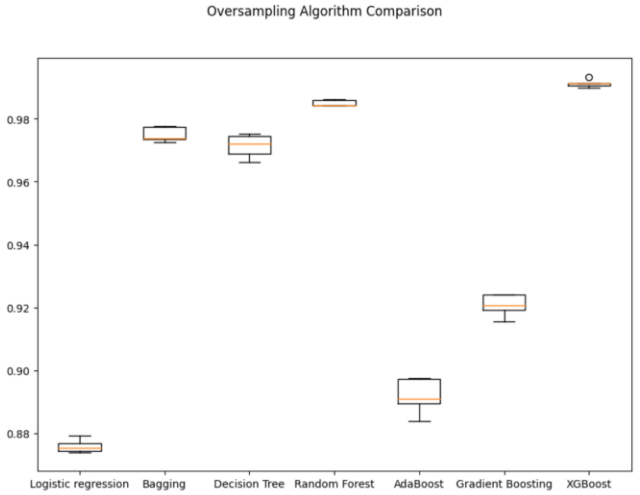



```
Cross-Validation performance on training dataset:

Logistic regression: 0.4904761904761905
Bagging: 0.7071428571428572
Decision Tree: 0.7297619047619047
Random Forest: 0.7226190476190476
AdaBoost: 0.6190476190476192
Gradient Boosting: 0.7142857142857142
XGBoost: 0.8011904761904761

Validation Performance:

Logistic regression: 0.48148148148148145
Bagging: 0.7222222222222222
Decision Tree: 0.7111111111111111
Random Forest: 0.6962962962962963
AdaBoost: 0.5777777777777777
Gradient Boosting: 0.6888888888888889
XGBoost: 0.8
```

Link to Appendix slide on model assumptions

Model Performance Summary (oversampled data)

- **Best Overall:** XGBoost outperforms with the highest recall.
- **High Performers:** Bagging and Gradient Boosting show strong results with limited variability.
- **Consistent Output:** Decision Tree and Random Forest have consistent but moderate recall scores.
- **Lowest Recall:** Logistic Regression falls behind significantly in recall.
- **Training vs. Validation:** All models except Logistic Regression show a drop in recall from training to validation.
- **Need for Balance:** High recall in training yet lower in validation suggests potential overfitting; models may require further tuning for better generalization.

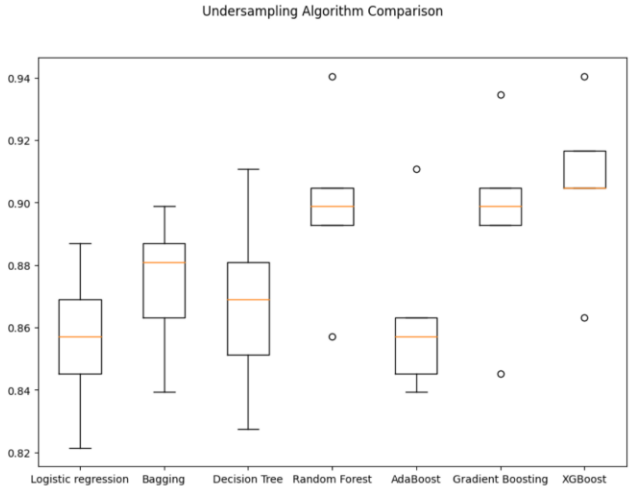



```
Cross-Validation performance on training dataset:

Logistic regression: 0.8759180790960451
Bagging: 0.975
Decision Tree: 0.9713983050847457
Random Forest: 0.9848870056497174
AdaBoost: 0.8918079096045199
Gradient Boosting: 0.9206920903954803
XGBoost: 0.9911723163841808

Validation Performance:

Logistic regression: 0.8518518518518519
Bagging: 0.8148148148148148
Decision Tree: 0.7851851851851852
Random Forest: 0.8407407407407408
AdaBoost: 0.8555555555555555
Gradient Boosting: 0.8629629629629629
XGBoost: 0.8592592592592593
```

Link to Appendix slide on model assumptions

Model Performance Summary (undersampled data)

- XGBoost excels again with high recall, despite some outliers.
- Random Forest and Gradient Boosting show competitive recall scores.
- Bagging and Decision Tree display variability in performance.
- AdaBoost and Logistic Regression are less effective, with lower recall.
- **Overall:** XGBoost proves robust across sampling methods, while Logistic Regression consistently lags.




```
Cross-Validation performance on training dataset:

Logistic regression: 0.855952380952381
Bagging: 0.8738095238095237
Decision Tree: 0.8678571428571427
Random Forest: 0.8988095238095237
AdaBoost: 0.8630952380952381
Gradient Boosting: 0.8952380952380953
XGBoost: 0.9059523809523808

Validation Performance:

Logistic regression: 0.8555555555555555
Bagging: 0.8481481481481481
Decision Tree: 0.837037037037037
Random Forest: 0.877777777777778
AdaBoost: 0.8555555555555555
Gradient Boosting: 0.8888888888888888
XGBoost: 0.8888888888888888
```

Link to Appendix slide on model assumptions

Great Learning

Happy Learning !