Renewind Model Tuning Solution

Model Tuning
December 17, 2021
James Watt

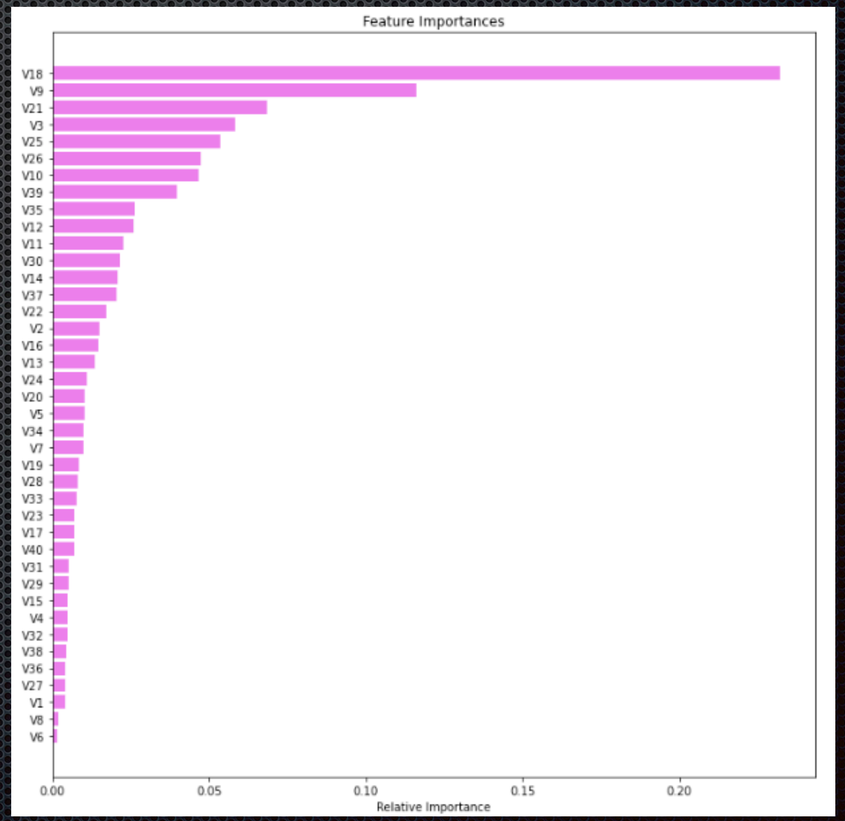
Contents

- Opportunity
- Data Overview
- Analysis
- Insight and Recommendations

Opportunity

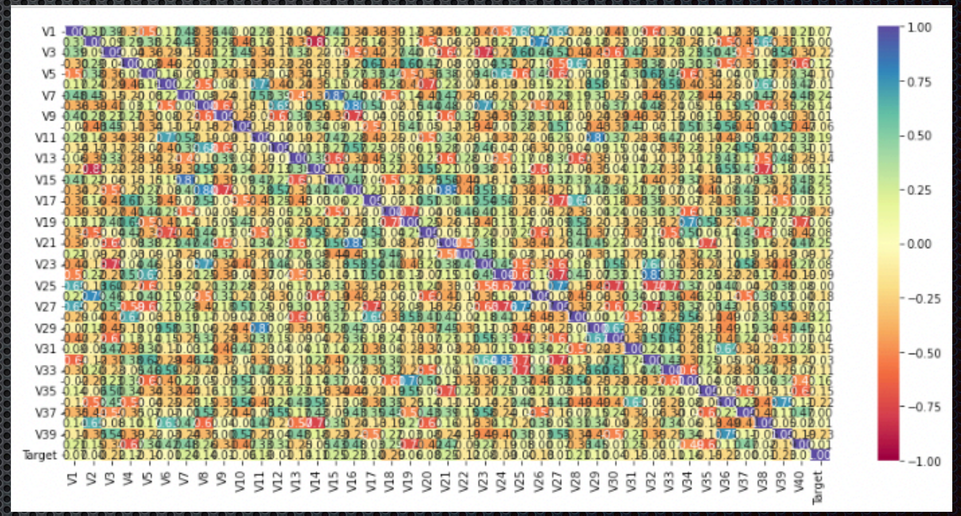
RENEWIND

- Renewind is a company which improves turbine production of wind energy.
- Build various classification models, tune them and find the best model to predict maintenance actions.
- Maintenance cost= Repair cost+Replacement Cost+ Inspection Cost
- Reduced maintenance cost is the goal.



Data Overview

- Data consists of 50000 total inputs, which are split into training and production data.
- Training is further split into training, validation and test data.
- There are 40 predictors.



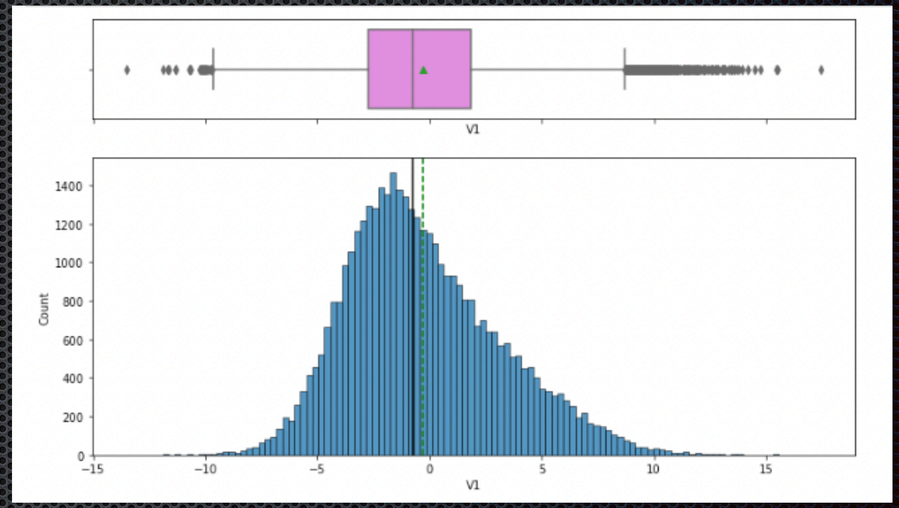
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
0	-4.465	-4.679	3.102	0.506	-0.221	-2.033	-2.911	0.051	-1.522	3.762	-5.715	0.736	0.981	1.418	-3.376	-3.047	0.306	2.914	2.270	4.395	-2.388
1	-2.910	-2.569	4.109	1.317	-1.621	-3.827	-1.617	0.669	0.387	0.854	-6.353	4.272	3.162	0.258	-3.547	-4.285	2.897	1.508	3.668	7.124	-4.096
2	4.284	5.105	6.092	2.640	-1.041	1.308	-1.876	-9.582	3.470	0.763	-2.573	-3.350	-0.595	-5.247	-4.310	-16.232	-1.000	2.318	5.942	-3.858	-11.599
3	3.366	3.653	0.910	-1.368	0.332	2.359	0.733	-4.332	0.566	-0.101	1.914	-0.951	-1.255	-2.707	0.193	-4.769	-2.205	0.908	0.757	-5.834	-3.069
4	-3.832	-5.824	0.634	-2.419	-1.774	1.017	-2.099	-3.173	-2.082	5.393	-0.771	1.107	1.144	0.943	-3.164	-4.248	-4.039	3.689	3.311	1.059	-2.143

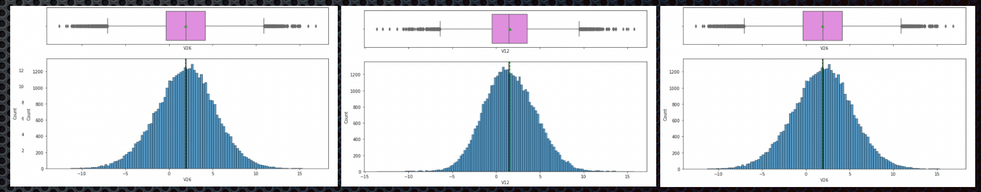
```
df.tail()
```

V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
2.133	-4.502	2.777	5.728	1.620	-1.700	-0.042	-2.923	-2.760	-2.254	2.552	0.982	7.112	1.476	-3.954	1.856	5.029	2.083	-6.409	1.477	-0.874	0
3.524	-7.015	-0.132	-3.446	-4.801	-0.876	-3.812	5.422	-3.732	0.609	5.256	1.915	0.403	3.164	3.752	8.530	8.451	0.204	-7.130	4.249	-6.112	0
1.458	-6.429	1.818	0.806	7.786	0.331	5.257	-4.867	-0.819	-5.667	-2.861	4.674	6.621	-1.989	-1.349	3.952	5.450	-0.455	-2.202	1.678	-1.974	0
2.031	-8.849	0.566	-6.040	-0.043	1.656	4.250	1.727	-1.686	-3.963	-2.642	1.939	-1.257	-1.136	1.434	5.905	3.752	-1.867	-1.918	2.573	-5.019	0
1.958	-4.454	0.464	-4.952	-1.624	2.965	2.009	5.712	-2.910	-2.287	-3.676	5.678	-4.310	-0.709	-1.359	1.639	7.766	-0.245	-1.124	2.872	1.902	0

Exploratory Data Analysis

- Sensor data provided in the training set was examined.
- It contained two fields with 85 missing values. V1, V2.
- The other 38 sensors report a normal distribution of data.
- Some outliers exist. These are naturally occurring phenomena and were included as they indicate stress placed on the system as captured by the sensor.





Decision Tree Training and Validation

RENEWIND

- Algorithm comparison of models shows a strong advantage for XGBoost.
- Decision Tree model will also be developed to show a competitive comparison.

Training performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	1.000	1.000	1.000	1.000	1.000

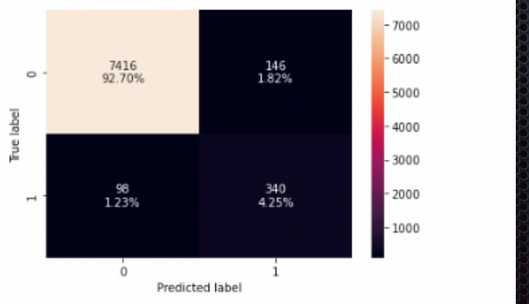
```
# That result looks overfitted and unrealistic.
```

```
# Calculating different metrics on validation set
dtree_grid_val = model_performance_classification_skl
print("Validation performance:")
dtree_grid_val
```

Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.970	0.776	0.700	0.736	0.674

```
# creating confusion matrix
confusion_matrix_sklearn(dtree_tuned1, X_val, y_val)
```



Training performance:

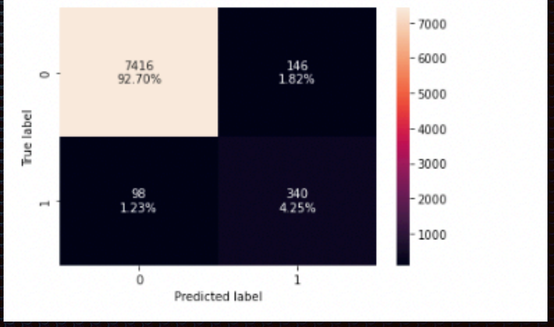
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	1.000	1.000	1.000	1.000	1.000

```
# Calculating different metrics on validation set
dtree_random_val = model_performance_classification_s
print("Validation performance:")
dtree_random_val
```

Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.973	0.779	0.743	0.760	0.685

```
# creating confusion matrix
confusion_matrix_sklearn(dtree_tuned1, X_val, y_val)
```



XGBoost Training and Validation

- Algorithm comparison of models shows a strong advantage for XGBoost.
- Decision Tree model will also be developed to show a competitive comparison.

Training performance:

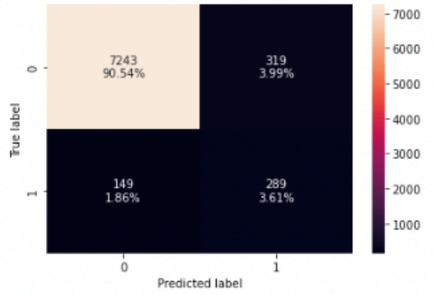
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.940	0.655	0.465	0.544	0.548

```
# Calculating different metrics on validation set; Tr
xgboost_grid_val = model_performance_classification_s
print("Validation performance:")
xgboost_grid_val
```

Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.942	0.660	0.475	0.553	0.553

```
# creating confusion matrix
confusion_matrix_sklearn(xgb_tuned1, X_val, y_val)
```



Training performance:

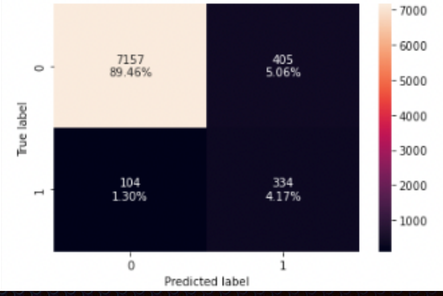
	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.936	0.751	0.447	0.560	0.580

```
# Calculating different metrics on validation set
xgboost_random_val = model_performance_classification
print("Validation performance:")
xgboost_random_val
```

Validation performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.936	0.763	0.452	0.568	0.587

```
# creating confusion matrix
confusion_matrix_sklearn(xgb_tuned2, X_val, y_val)
```



Model Comparison

- Decision Tree initially overfitting.
- XGBoost initially appears to deliver better result.
- On validation and testing dat, Decision Tree model rendered better accuracy, recall, and most importantly a lower maintenance cost.


```
# Calculating different metrics on the test set
dtree_grid_test = model_performance_classification_
print("Test performance:")
dtree_grid_test
```

Test performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.967	0.725	0.685	0.704	0.637

```
# Calculating different metrics on the test set
xgboost_grid_test = model_performance_classificati
print("Test performance:")
xgboost_grid_test
```

Test performance:

	Accuracy	Recall	Precision	F1	Minimum_Vs_Model_cost
0	0.939	0.632	0.455	0.529	0.536

Training performance comparison:

	DecisionTree Tuned with Grid search	DecisionTree Tuned with Random search	Xgboost Tuned with Grid search	Xgboost Tuned with Random Search
Accuracy	1.000	1.000	0.940	0.936
Recall	1.000	1.000	0.655	0.751
Precision	1.000	1.000	0.465	0.447
F1	1.000	1.000	0.544	0.560
Minimum_Vs_Model_cost	1.000	1.000	0.548	0.580

```
# Validation performance comparison; Decision Tree is overfitting.

models_val_comp_df = pd.concat(
    [dtree_grid_val.T, dtree_random_val.T, xgboost_grid_val.T, xgboost_random_val.T,],
    axis=1,
)
models_val_comp_df.columns = [
    "DecisionTree Tuned with Grid search",
    "DecisionTree Tuned with Random search",
    "Xgboost Tuned with Grid search",
    "Xgboost Tuned with Random Search",
]
print("Validation performance comparison:")
models_val_comp_df
```

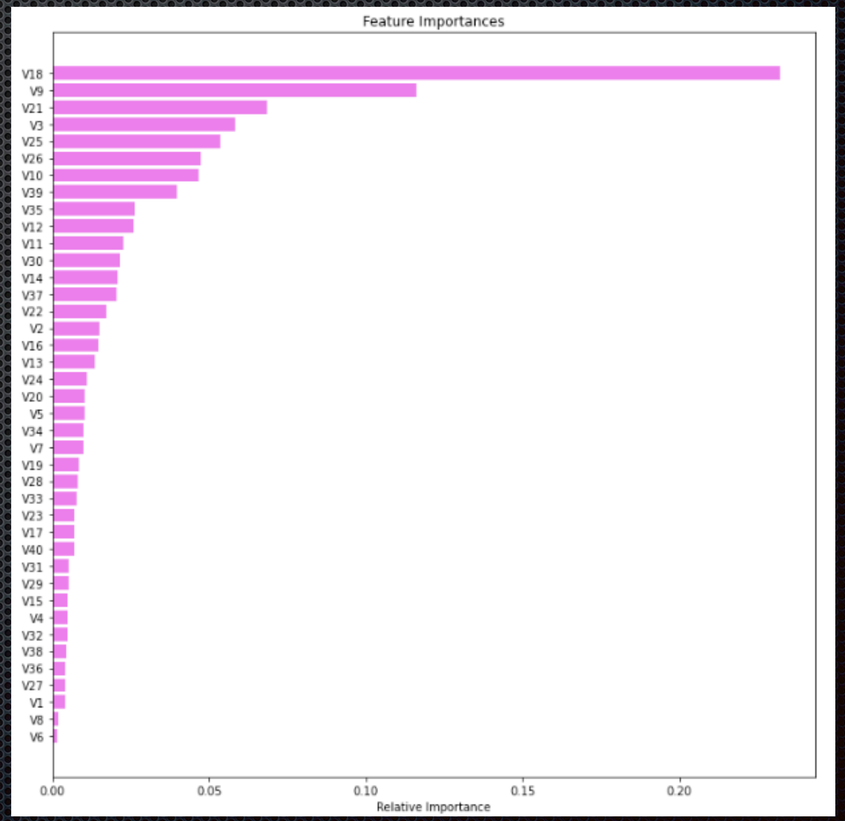
Validation performance comparison:

	DecisionTree Tuned with Grid search	DecisionTree Tuned with Random search	Xgboost Tuned with Grid search	Xgboost Tuned with Random Search
Accuracy	0.970	0.973	0.942	0.936
Recall	0.776	0.779	0.660	0.763
Precision	0.700	0.743	0.475	0.452
F1	0.736	0.760	0.553	0.568
Minimum_Vs_Model_cost	0.674	0.685	0.553	0.587

Feature Importance

RENEWIND

- Strong feature importance indicated for one feature. (V18)
- All variables except target variable were utilized in final model due to their noted importance.



Limited Data Processing of Production Data

- Production data was never examined.
- Target variable was dropped.
- Data was split to train and test.
- X,y Split: (6987, 40) (2995, 40)
- Data was pre-processed and then fitted on Decision Tree model.

```
X1 = data.drop(["Target"], axis=1)
y1 = data["Target"]
```

```
# Splitting the data into train and test sets
X1_train, X1_test, y1_train, y1_test = train_test_split(
    X1, y1, test_size=0.30, random_state=1, stratify=y1
)
print(X1_train.shape, X1_test.shape)
```

```
(6987, 40) (2995, 40)
```

```
# Creating new pipeline with best parameters
model = Pipeline(
    steps=[
        ("pre", preprocessor),
        (
            "dtree",
            DecisionTreeClassifier(
                random_state=1, criterion="gini", max_depth=None, min_samples_split=2
            ),
        ),
    ]
)
# Fit the model on training data
model.fit(X1_train, y1_train)
```

Recommendations

- Unfortunately, all of the predictions failed to achieve the theoretical minimum maintenance cost for operation.
- This is not a particularly good result.
- While theoretically a vlaid result in the real world, this result, when combined with procedural challenges during training data manipulation, points towards a need for further data pre-processing and refinement.
- Based on another round of model tuning, it may be possible to achieve the theoretical lowest maintenance cost.
- Recommendation is to continue trial and further refine model in order to tune and achieve the minimum cost for maintenance of wind powered turbines.