ReneWind

Project 6 – UT DSBA Program

08/10/2022

Contents

Executive Summary

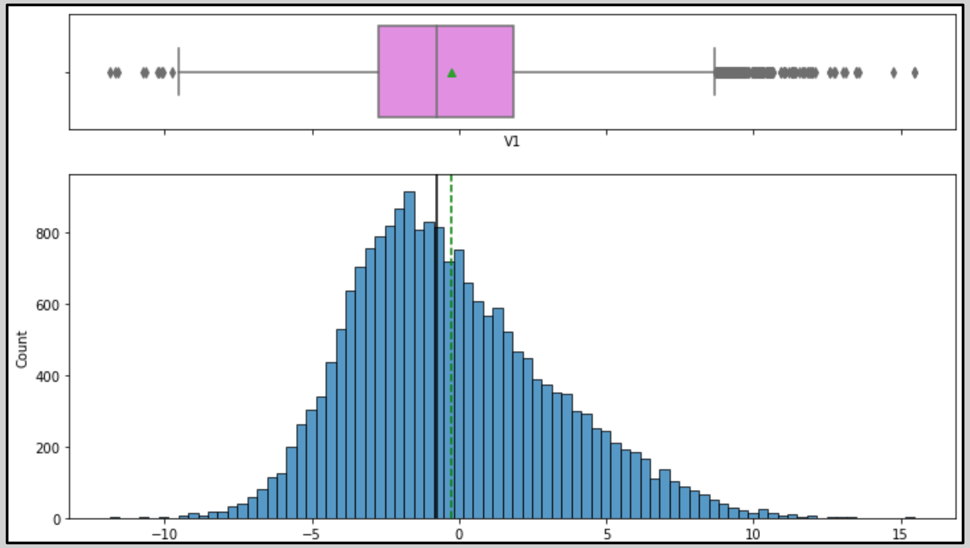
- After evaluating multiple different machine learning models using hypertuning and cross-validation techniques, a model using the XGBoost classifier was chosen to be the final model. A pipeline was created to productionize this model using the provided test data.
    - The performance metrics for this model can be found in the Model Performance Summary and evaluation logic for the rest of the models tested can be found in the appendix.
- V36 is the most important feature in predicting machine failure. V14, V26, V16, V18, and V1 follow as the next most important features.
- The least important features in predicting machine failure are V20, V25, V8, V40, and V31.
- Perhaps ReneWind can use the knowledge gained from analyzing the feature importance and place more sensors around the types of machine parts on the more important end of the spectrum.
- Conversely, less focus could be placed on the machine parts where the sensor data was deemed less important in the model. This would result in a more refined and accurate model in the future.

Business Problem Overview and Solution Approach

- ReneWind is working to improve their process for identifying machines that in need of preventative maintenance in order to cut down on overall repair costs and reduce machine failures.
- It has been determined that building a machine learning model to help predict machine failure is the best solution.
- The approach to creating a fully productionized machine learning model is as follows: a series of classification models will be built, tuned, and then compared to determine which is most accurate in successfully predicting machine failure.
- These three criteria will be used to measure success of the model:
    - True positives – results in repairing costs
    - False negatives – results in replacement costs (ideally low)
    - False positives – results in inspection costs (not as expensive as false negatives but still would like to minimize)
- Inspection costs are cheap while replacement costs are the most expensive. Repair costs are middle of the road.

Exploratory Data Analysis (Univariate)

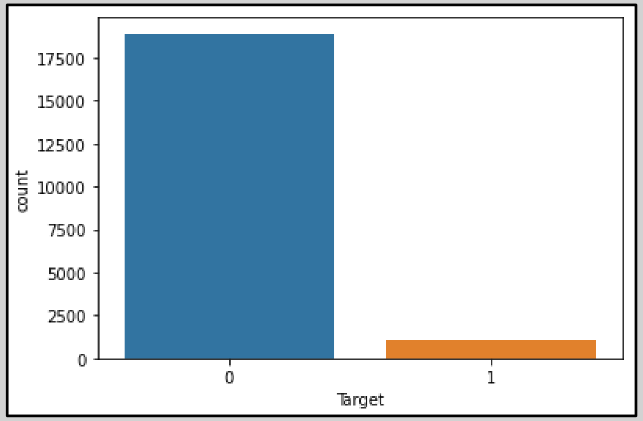
Distribution of V1 – Train

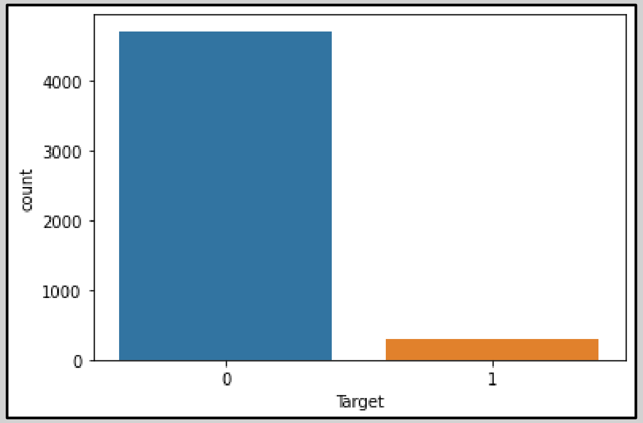


Observations

- Most of the columns in the data follow an approximately normal distribution.
- There are some with a slight skew, but the example above (column V1 from train data) is about as extreme as the skewness gets.

Distribution of Target – Train (left) vs Test (right)



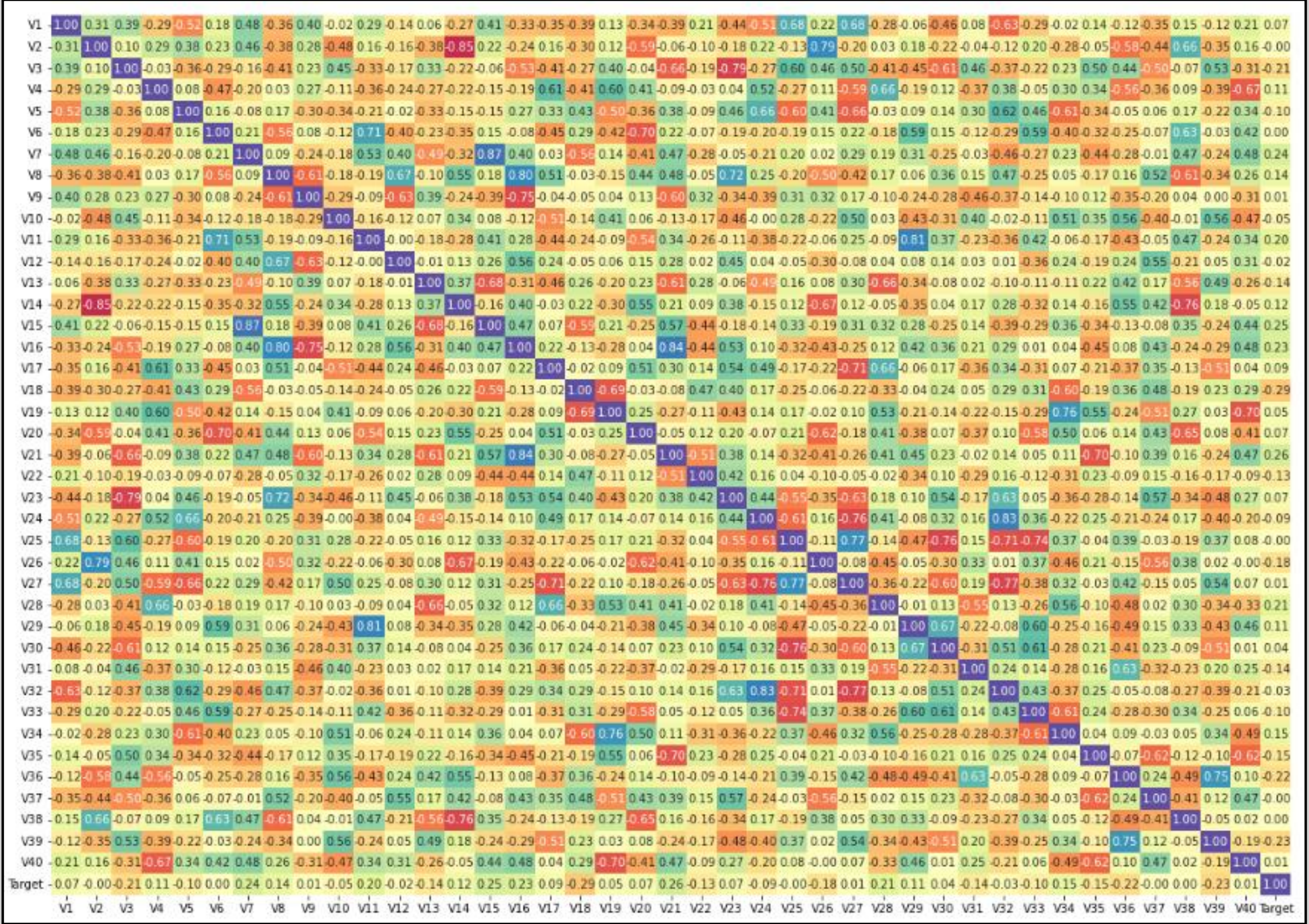


Observations

- The column Target does not follow a normal distribution. This is to be expected because although the variable is numerical, it is treated somewhat like a categorical variable – the data can only be either 0 or 1 and nothing in between.
- The distribution of the values representing a failure (value = 1) is roughly the same between the train and test datasets, indicating that the test dataset is likely a good sample of the train dataset.
  - Train = 5.55% (1110/20000)
  - Test = 5.64% (282/5000)

Exploratory Data Analysis (Multivariate)

Correlation Heatmap (Train)



Observations

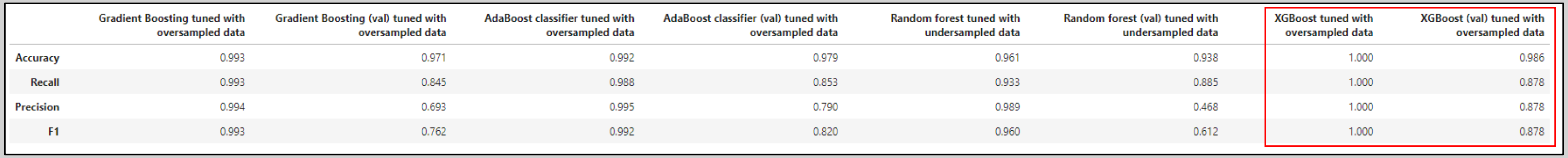
- While most features do not appear to have a strong positive or negative correlation, there are a handful that do.
- Some examples of strong positive correlation:
  - V2 & V21 with 0.79
  - V7 & V15 with 0.87
  - V8 & V16 with 0.80
  - V16 & V21 with 0.84
- Some examples of strong negative correlation:
  - V2 & V14 with -0.85
  - V3 & V23 with -0.79
  - V24 & V27 with -0.76
  - V27 & V32 with -0.77
- The structure of the data does not lend itself to many other graphing methods so we will stop the EDA here.
- The test dataset heatmap is very similar.

Data Preprocessing

- Neither the train nor test datasets had any duplicate values.
- There were some missing values in the train and test datasets.
    - The train dataset was missing 18 values in column V1 and 18 values in column V2.
    - The test dataset was missing 5 values in column V1 and 6 values in column V2.
    - These missing values were dealt with by imputing with the median value of the respective column.
- Each column (with exception of Target) has outliers; however, we did not do anything with these since they are real world values.
- The train dataset was split into two (train and validation) separate datasets.
- No additional feature engineering or preprocessing was required because all columns are numerical.

Model Performance Summary (Hypertuning)

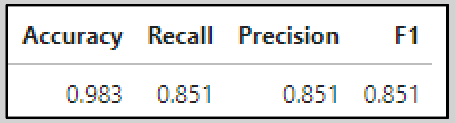
- The model building started with six models on the original data
  - Logistic Regression
  - Bagging
  - Decision Tree
  - AdaBoost
  - Gradient Boosting (later also used XGBoost)
  - Random Forest
- In order to reduce false negatives, we used recall as the scorer in our cross-validation and hyperparameter tuning.
- We ended up hypertuning these four models
  - Gradient Boosting with oversampled data
  - AdaBoost with oversampled data
  - Random Forest with undersampled data
  - XGBoost with oversampled data
- Of these models, the XGBoost produced the best results between the train and validation data.



	Gradient Boosting tuned with oversampled data	Gradient Boosting (val) tuned with oversampled data	AdaBoost classifier tuned with oversampled data	AdaBoost classifier (val) tuned with oversampled data	Random forest tuned with undersampled data	Random forest (val) tuned with undersampled data	XGBoost tuned with oversampled data	XGBoost (val) tuned with oversampled data
Accuracy	0.993	0.971	0.992	0.979	0.961	0.938	1.000	0.986
Recall	0.993	0.845	0.988	0.853	0.933	0.885	1.000	0.878
Precision	0.994	0.693	0.995	0.790	0.989	0.468	1.000	0.878
F1	0.993	0.762	0.992	0.820	0.960	0.612	1.000	0.878

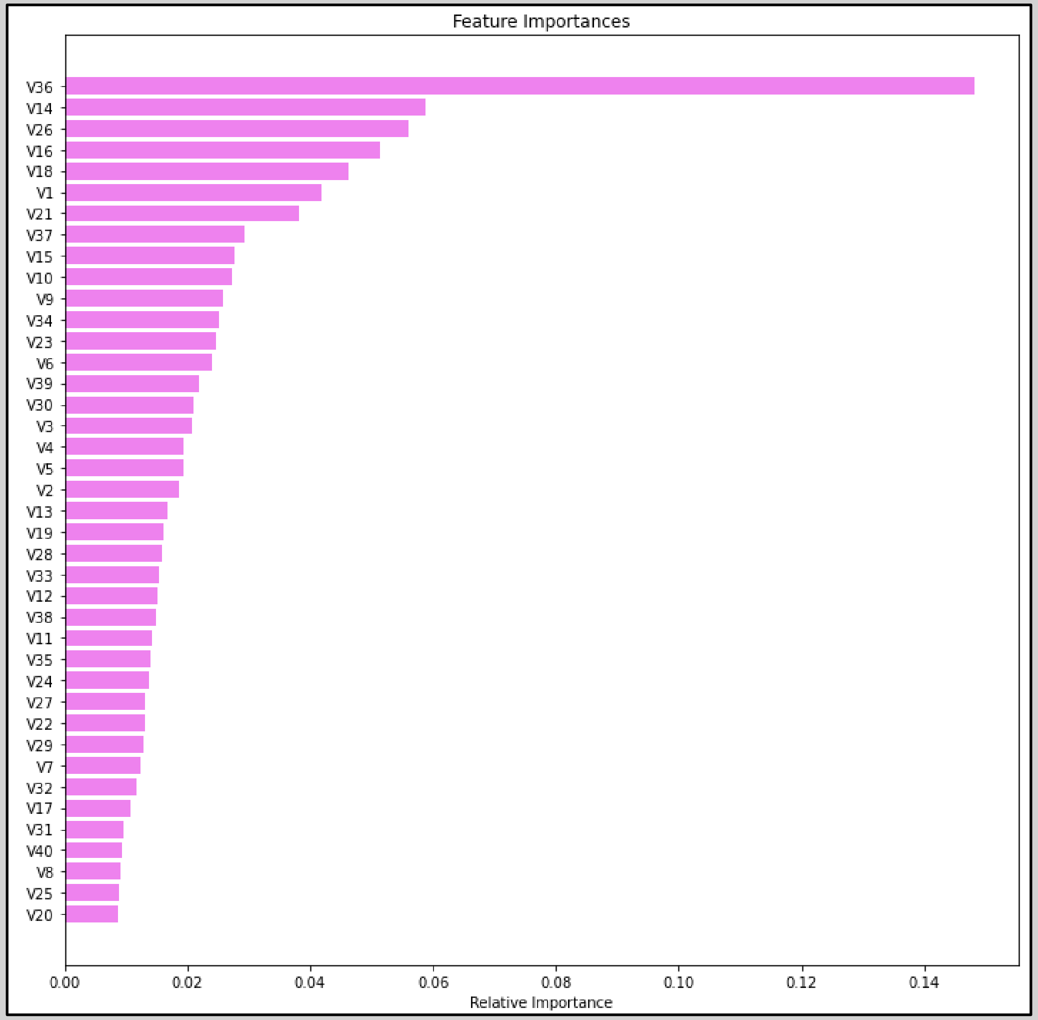
Model Performance Summary (Pipeline)

- The pipeline for the final model was created using these steps:
    - SimpleImputer(strategy="median")
    - XGBClassifier
        - random_state=1
        - eval_metric="logloss"
        - subsample=0.8
        - scale_pos_weight=10
        - n_estimators=250
        - learning_rate=0.2
        - gamma=0
    - The model was then fit over X_train_over and y_train_over (the oversampled x and y datasets).
    - Finally, we ran the Synthetic Minority Over Sampling Technique to before fitting the model to the test dataset using these parameters:
        - sampling_strategy=1
        - k_neighbors=5
        - random_state=1
- After fitting the pipeline model to the test data, the performance metrics are similar to what we saw with the validation data on the hypertuning step.



Accuracy	Recall	Precision	F1
0.983	0.851	0.851	0.851

Model Performance Summary (Feature Importances)



<u>Observations</u>

- V36 by far the most important feature on the train dataset of the final model with a relative importance value of just over 0.14; this is over double the relative importance value of the next most important feature.
- V14 and V26 are the next two most important features at around 0.06 relative importance value.
- The rest of the features fall between just under 0.01 and 0.06 relative importance.

Appendix

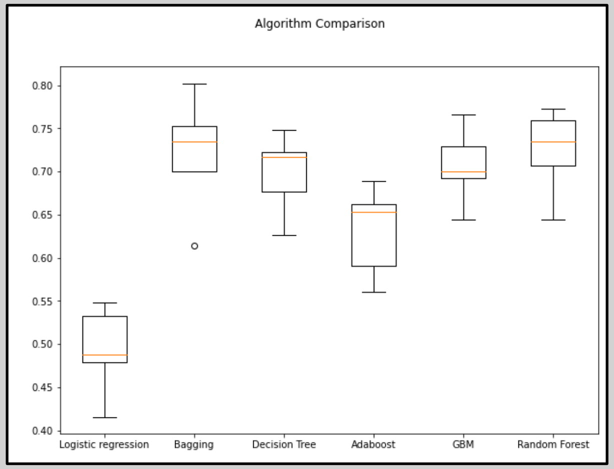
Data Dictionary and Shape

Column	Data Type	Description
V1 thru V40	float64	predictor variable transformed from the original data collected via sensor
Target	int64	the target variable; either 0 (no failure) or 1 (failure)

- Shape of the train data is 20,000 rows and 41 columns
  - This will be broken out into two separate datasets – one for train and one for validation
- Shape of the test data is 5,000 rows and 41 columns

Model Performance Summary (original data)

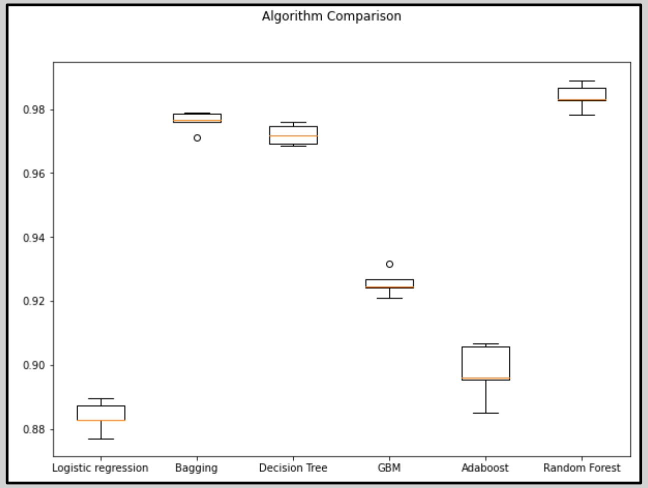
- Six model classifiers were built off the original data:
  - Logistic regression
  - Bagging
  - Decision Tree
  - AdaBoost
  - Gradient Boost
  - Random Forest
- These models were all cross-validated using StratifiedKFold with these parameters:
  - n_splits=5
  - shuffle=True
  - random_state=1
  - scorer=metrics.make_scorer(metrics.recall_score)
- The model performance can be seen below. None of the models are particularly strong, where Logistic regression and AdaBoost are the weakest.



	Logistic Regression	Bagging	Decision Tree	AdaBoost	Gradient Boost	Random Forest
Training Performance	0.49	0.71	0.70	0.63	0.71	0.72
Validation Performance	0.48	0.73	0.71	0.68	0.72	0.73

Model Performance Summary (oversampled data)

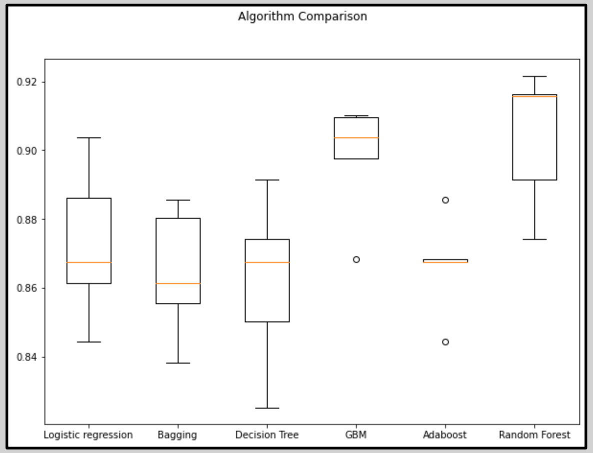
- The data was oversampled using SMOTE with the following parameters
  - sampling_strategy=1
  - k_neighbors=5
  - random_state=1
- The same parameters as before were used for the cross-validation of the train and validation data, this time fit to new dataframes created for the oversampled data.
- The performance of the oversampled models is much better than the original data (for all models), but there still might be room for improvement so we will try undersampling the data as well.



	Logistic Regression	Bagging	Decision Tree	AdaBoost	Gradient Boost	Random Forest
Training Performance	0.88	0.98	0.97	0.93	0.90	0.98
Validation Performance	0.85	0.83	0.78	0.88	0.86	0.85

Model Performance Summary (undersampled data)

- The data was undersampled using RandomUnderSampler with the following parameters
  - sampling_strategy=1
  - random_state=1
- The same parameters as before were used for the cross-validation of the train and validation data, this time fit to new dataframes created for the undersampled data.
- The performance of the undersampled data as a whole is not better than the oversampled data; however, the random forest performance is tighter between training and validation than the oversampled model. We will take this into account as we will hypertune random forest using undersampled data; AdaBoost, Gradient Boost, and XGBoost will be hypertuned using the oversampled data.



	Logistic Regression	Bagging	Decision Tree	AdaBoost	Gradient Boost	Random Forest
Training Performance	0.87	0.86	0.86	0.90	0.87	0.90
Validation Performance	0.85	0.87	0.84	0.89	0.85	0.89