Phase II – Testing and Comparison

Submitted by:

Sandesh Pabitwar

Modern Education Society's College of Engineering Pune

Project Title: INTP22-ML-2 Remaining Usable Life Estimation (NASA Turbine dataset)

Objective of Phase -III**:**

- To train the model on training set.
- Improve accuracy of trained model.
- Test the trained model on test set.
- Compare the accuracy of trained model.
- Choose the best model for deployment.

Training a model.

Model training is process in which we fit the training data into a selected model to get the desired output from that model.

Linear regression model.

Linear regression is a statistical method for modeling relationships between a dependent variable with a given set of independent variables.

It is a simplest algorithm in machine learning. Multiple linear regression is one type of linear regression. Multiple linear regression attempts to model the relationship between two or more features and a response by fitting a linear equation to the observed data.

In the given dataset sensors values are acting as a features and RUL is acting as a targeted value (response). Using a linear regression, we can get a linear equation which will be used to predict the RUL.

```
lm = LinearRegression()
lm.fit(X_train, y_train)

# predict and evaluate
y_hat_train = lm.predict(X_train)
evaluate(y_train, y_hat_train, 'train')

y_hat_test = lm.predict(X_test)
evaluate(y_test, y_hat_test,'test')
```

```
train set RMSE:44.66819159545432, R2:0.5794486527796756
test set RMSE:31.952633027739118, R2:0.40877368076584075
```

Using linear regression model RMSE on train set is 44.66 and on test set is 31.95.

This model will act like a base model. This model is not able to give desired accuracy so we will move on to another model.

Random Forest algorithm

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

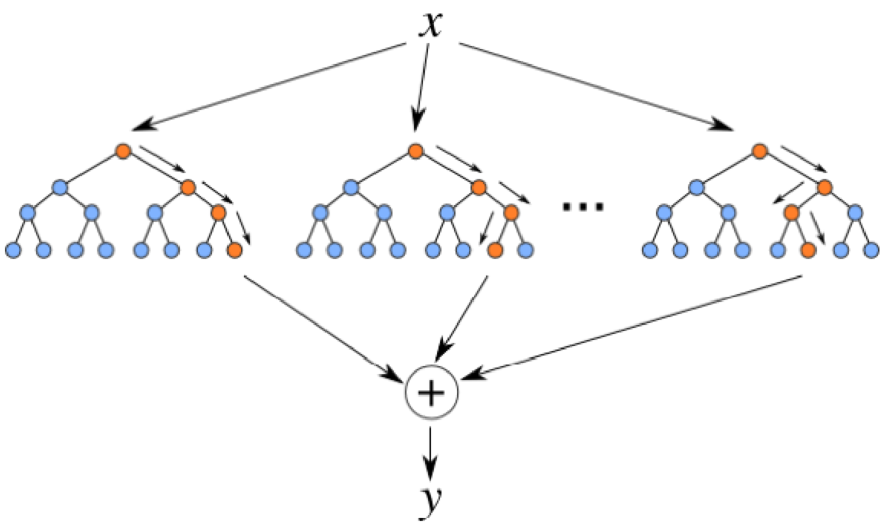


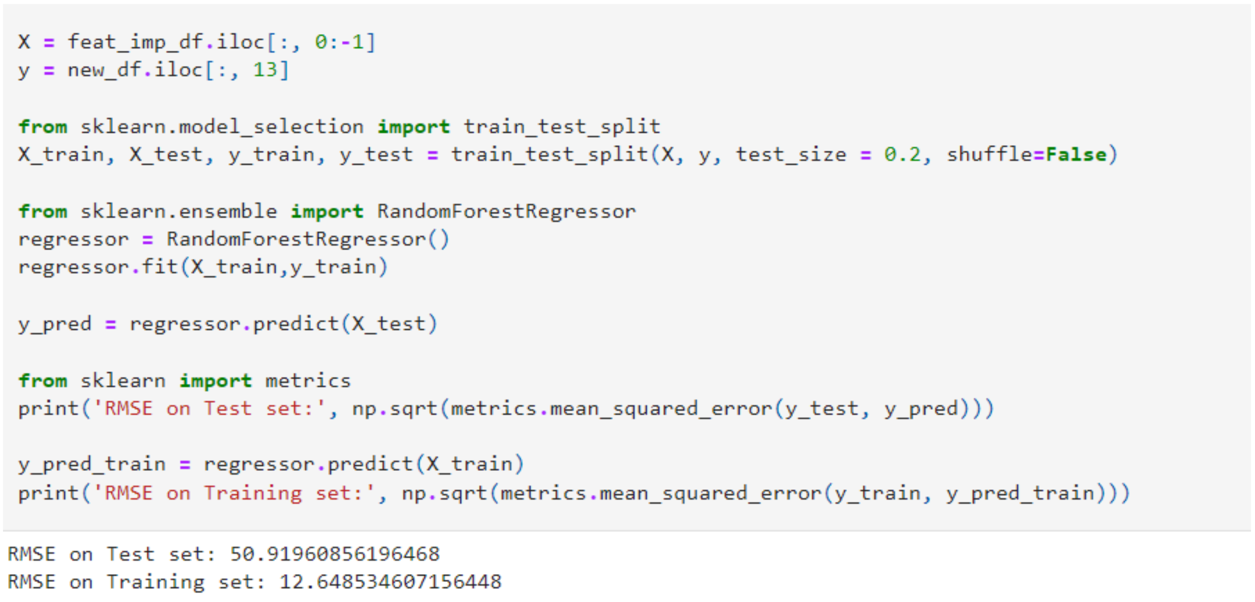
Figure 1 Random Forest algorithm

As figure 1 shows to predict the output y model uses multiple decision trees and combines all the decision trees.

Before training the Random Forest model I have done some pre-processing on the data. Trained the Random Forest model on 13 features. These features are selected after EDA and feature engineering.

Trained the model on 13 features. After the training the model it is giving the RMSE 50 on test set and RMSE 12 on test set.

From RMSE values it is clear that the model is overfitting the data.



```
X = feat_imp_df.iloc[:, 0:-1]
y = new_df.iloc[:, 13]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle=False)

from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor()
regressor.fit(X_train,y_train)

y_pred = regressor.predict(X_test)

from sklearn import metrics
print('RMSE on Test set:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

y_pred_train = regressor.predict(X_train)
print('RMSE on Training set:', np.sqrt(metrics.mean_squared_error(y_train, y_pred_train)))
```

```
RMSE on Test set: 50.91960856196468
RMSE on Training set: 12.648534607156448
```

Figure 2 Random Forest model implementation

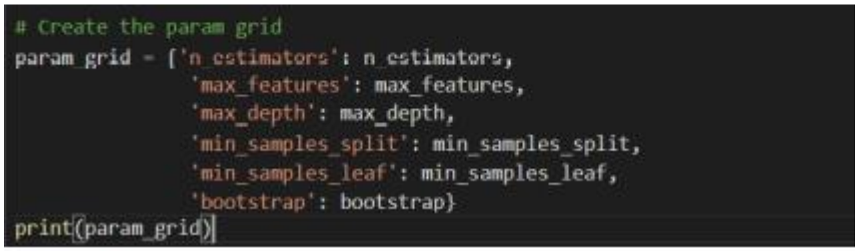
This model is not giving the good accuracy and it is overfitting the data so to improve accuracy and overfitting hyperparameter tunning is done.

hyperparameter tunning on random forest.

Using the get_params() function I got all the 17 parameters available for Random forest.

- Parameters available for Random forest are { 'bootstrap', 'ccp_alpha', 'criterion', 'max_depth','max_features' , 'max_leaf_nodes', 'max_samples' , 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split' , 'min_weight_fraction_leaf' , 'n_estimators' , 'n_jobs' , 'oob_score', 'random_state', 'verbose' , 'warm_start' }

- Out of those parameters I selected 6 parameters for tuning.



```
# Create the param grid
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(param_grid)
```

- To get the best parameters I have used Randomize search and Grid search.

- Best parameters returned using Randomize search are
  - 'bootstrap': True,
  - 'max_depth': 4,
  - 'max_features': 'sqrt',
  - 'min_samples_leaf': 1,
  - 'min_samples_split': 2,
  - 'n_estimators': 72

- Best parameters returned using Grid search are
  - 'criterion': 'gini',
  - 'max_depth': 55,
  - 'max_features': 'log2',
  - 'min_samples_leaf': 0.001,
  - 'min_samples_split': 0.001,
  - 'n_estimators': 190

- Using Randomize search accuracy was
  - Train Accuracy -: 0.714 RMSE 35
  - Test Accuracy -: 0.558 RMSE 52

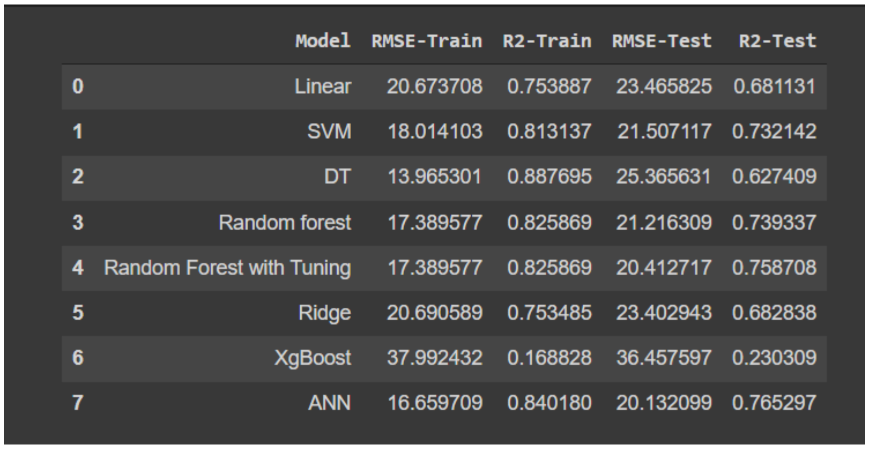
- Using Grid search for tuning accuracy was
  - Train Accuracy -: 0.6764420746485701
  - Test Accuracy -: 0.6966319360310153

Training deep learning models:

Through the regression models the accuracy was not quite good so I implemented some deep learning models on the given dataset.

While implementing deep learning models using clip function clipped the RUL greater than 125.

With the clipped values the models that I have trained different models, results are shown in table 1.

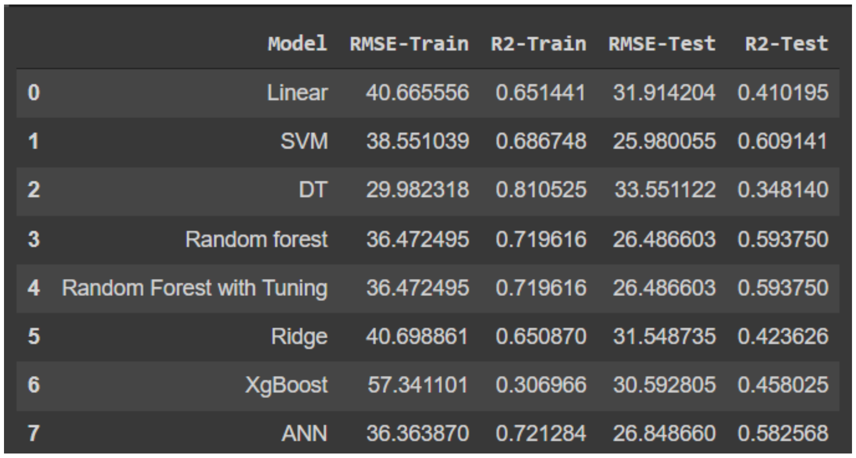


	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	Linear	20.673708	0.753887	23.465825	0.681131
1	SVM	18.014103	0.813137	21.507117	0.732142
2	DT	13.965301	0.887695	25.365631	0.627409
3	Random forest	17.389577	0.825869	21.216309	0.739337
4	Random Forest with Tuning	17.389577	0.825869	20.412717	0.758708
5	Ridge	20.690589	0.753485	23.402943	0.682838
6	XgBoost	37.992432	0.168828	36.457597	0.230309
7	ANN	16.659709	0.840180	20.132099	0.765297

Table 1 deep learning models

RMSE of ANN model is the least RMSE among all the trained models.

After doing this trained the same models without clipping the RUL then the results were different. Table 2 shows the results without clipping.



	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	Linear	40.665556	0.651441	31.914204	0.410195
1	SVM	38.551039	0.686748	25.980055	0.609141
2	DT	29.982318	0.810525	33.551122	0.348140
3	Random forest	36.472495	0.719616	26.486603	0.593750
4	Random Forest with Tuning	36.472495	0.719616	26.486603	0.593750
5	Ridge	40.698861	0.650870	31.548735	0.423626
6	XgBoost	57.341101	0.306966	30.592805	0.458025
7	ANN	36.363870	0.721284	26.848660	0.582568

Table 2 result of deep learning models without clipping

In test set so many engines were having RUL greater than 125 for those engines model trained with clipped RUL were not able to predict the RUL correctly so these models are not fit to predict the RUL on test data.

Models trained without the clipped RUL those were giving high RMSE so these are also not fit for predicting RUL on test set.

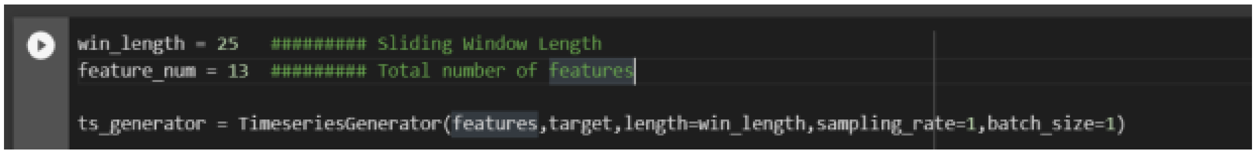
Training CNN model

In one research paper I found that using CNN model on Nasa Turbofan dataset we can achive RMSE 11 so implanted CNN model.

CNN models are mostly used for image processing to use CNN on time series data some processing is required.

To train model firstly I have used minmax scalar function to convert all data into the range of 0 to 1.

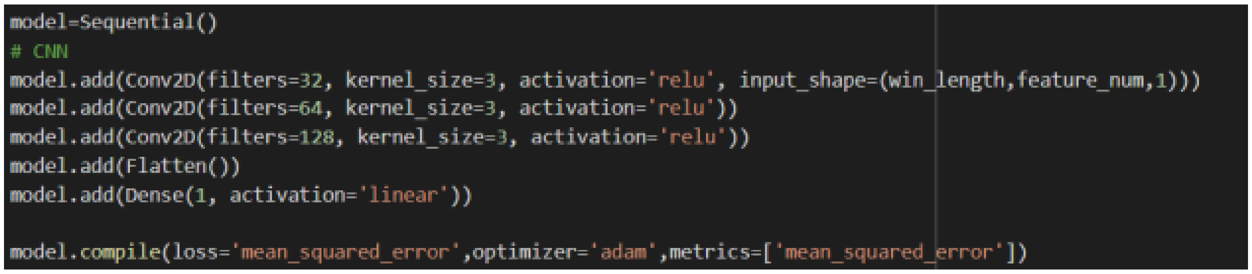
To feed the data into CNN model some processing is required. To do that I have used time series generator



```
win_length = 25   ######### Sliding Window Length
feature_num = 13  ######### Total number of features

ts_generator = TimeseriesGenerator(features,target,length=win_length,sampling_rate=1,batch_size=1)
```

CNN model’s layers are shown below.



```
model=Sequential()
# CNN
model.add(Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=(win_length,feature_num,1)))
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(Conv2D(filters=128, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(1, activation='linear'))

model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mean_squared_error'])
```

In the CNN model I have added 5 layers. 1st is input layer, 2nd is 2D convolution layer, 3rd is same as 2nd layer, 4th is flatten layer and 5th is dense and last is compiling layer.

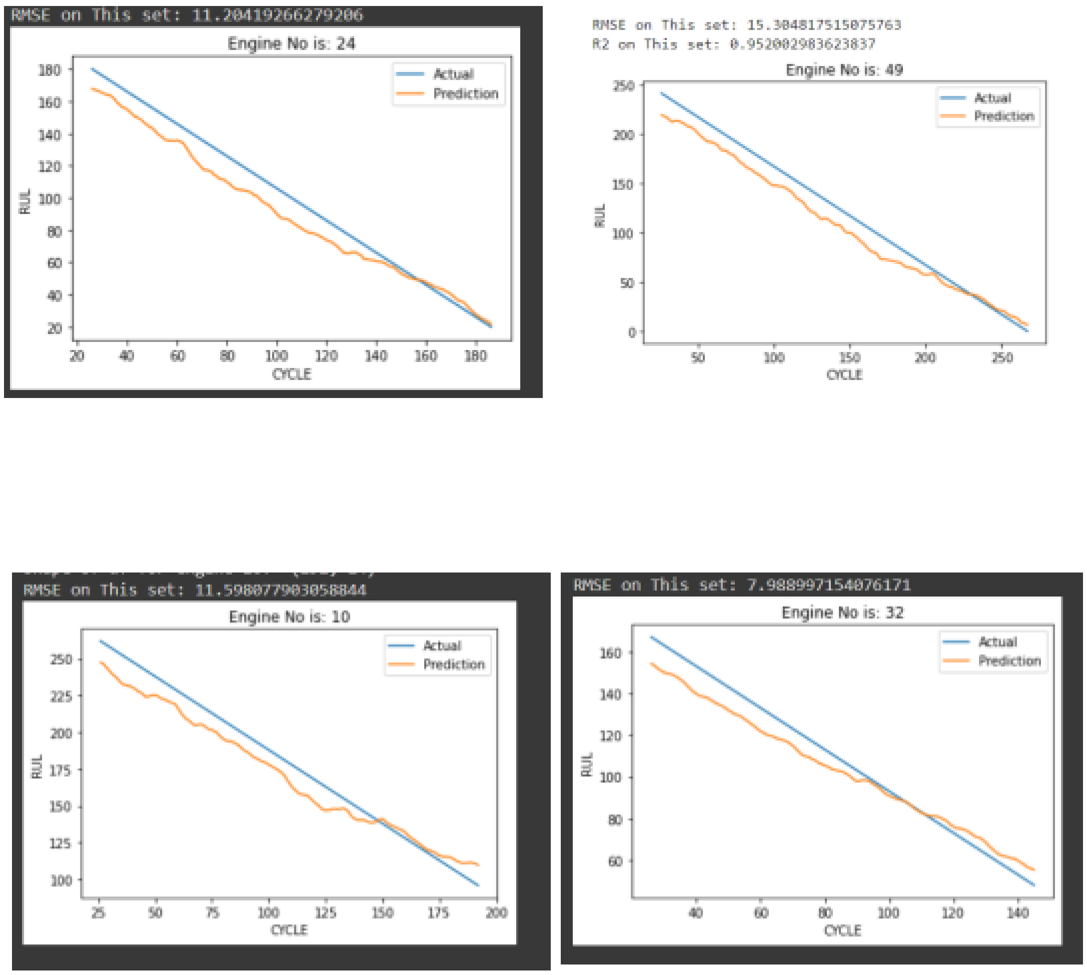
3 convolution layers convolves over the spatial and time dimensions of input data.

Flatten layer is used to convert 2D data into a single 1D array.

Dense layer will receive input from each neuron of previous layer. At last compile is used to compile all the layers with Adam optimizer.

Using this I got model RMSE 11 on test set 1 and RMSE 18 on test set 3.

Results of CNN model.



```
RMSE on This set: 11.20419266279206
```

```
RMSE on This set: 15.304817515075763
R2 on This set: 0.952002983623837
```

```
RMSE on This set: 11.598077903058844
```

```
RMSE on This set: 7.988997154076171
```

Figure 2 Result of CNN model

Result of some engines is shown in figure 3.