SPRINT 2 :

|  |  |
| --- | --- |
| Date | 15 November 2022 |
| Team ID | PNT2022TMID45948 |
| Project Name | Predicting the energy output of wind turbine based on weather condition |

In [ ]:

''' Model Building

Here we use 5 regression models as Linear Regression

Random Forest Regression

Support Vector Regressor Decision Tree Regressor XGBoost regressor

Check the metrics of the model save the model

'''

In [1]:

*# import libraries* **import** numpy **as** np **import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

df **=** pd**.**read\_csv("Turbine\_data.csv",low\_memory**=False**,parse\_dates**=**["Unnamed: 0 df**.**head()

Out[1]:

In [2]:

df**.**shape

00:00:00+00:00

00:10:00+00:00

|  |  |  |  |
| --- | --- | --- | --- |
| **Unnamed: 0 ActivePower** | **AmbientTemperatue** | **WindDirection** | **WindSpeed** |
| **0** 2018-01-01 -5.357727 | 23.148729 | 8.000000 | 2.279088 |
| **1** 2018-01-01 -5.822360 | 23.039754 | 300.428571 | 2.339343 |
| **2** 2018-01-01 -5.279409 | 22.948703 | 340.000000 | 2.455610 |
| **3** 2018-01-01 -4.648054 | 22.966851 | 345.000000 | 2.026754 |
| **4** 2018-01-01 -4.684632 | 22.936520 | 345.000000 | 1.831420 |

00:20:00+00:00

00:30:00+00:00

00:40:00+00:00



Out[2]: (118080, 5)

In [5]:

*# duplicate the date column to change it's name #parsing dates*

df['DateTime'] **=** df['Unnamed: 0']

df**.**drop('Unnamed: 0', axis**=**1, inplace**=True**)

In [6]:

*# Add datetime parameters*

df['DateTime'] **=** pd**.**to\_datetime(df['DateTime'], format **=** '%Y-%m-%dT%H:%M:%SZ',

errors **=** 'coerce')

df['year'] **=** df['DateTime']**.**dt**.**year

df['month'] **=** df['DateTime']**.**dt**.**month df['day'] **=** df['DateTime']**.**dt**.**day

df['hour'] **=** df['DateTime']**.**dt**.**hour

df['minute'] **=** df['DateTime']**.**dt**.**minute

In [4]:

*#check for null values*

df**.**isna()**.**sum()

|  |  |  |
| --- | --- | --- |
| Out[4]: | ActivePower | 23330 |
|  | AmbientTemperatue | 24263 |
|  | WindDirection | 45802 |
|  | WindSpeed | 23485 |
|  | DateTime | 0 |
|  | year | 0 |
|  | month | 0 |
|  | day | 0 |
|  | hour | 0 |
|  | minute  dtype: int64 | 0 |

In [7]:

*#handling null values*

df['AmbientTemperatue']**.**fillna(int(df['AmbientTemperatue']**.**mean()), inplace**=T** df['WindDirection']**.**fillna(int(df['WindDirection']**.**mean()), inplace**=True**)

df['WindSpeed']**.**fillna(int(df['WindSpeed']**.**mean()), inplace**=True**)

df['ActivePower']**.**fillna(int(df['ActivePower']**.**mean()), inplace**=True**)

In [6]:

*#splitting dependent and independent features*

independent\_features **=** df[['month','day','AmbientTemperatue','WindDirection', independent\_features**.**head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[6]: | **month** | **day** | **AmbientTemperatue** | **WindDirection** | **WindSpeed** |
|  | **0** 1 | 1 | 23.148729 | 8.000000 | 2.279088 |
|  | **1** 1 | 1 | 23.039754 | 300.428571 | 2.339343 |
|  | **2** 1 | 1 | 22.948703 | 340.000000 | 2.455610 |
|  | **3** 1 | 1 | 22.966851 | 345.000000 | 2.026754 |
|  | **4** 1 | 1 | 22.936520 | 345.000000 | 1.831420 |

In [7]:

target **=** df['ActivePower']

In [8]:

df\_new **=** independent\_features

X**=**np**.**asanyarray(df\_new)**.**astype('int') y**=**np**.**asanyarray(target)**.**astype('int') print(X**.**shape)

print(y**.**shape)

(118080, 5)

(118080,)

In [9]:

*# splitting the dataset into training and testing*

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, rand

# Linear Regression

In [10]:

**from** sklearn.linear\_model **import** LinearRegression LR **=** LinearRegression()

LR**.**fit(X\_train,y\_train)

Out[10]: LinearRegression() In [11]:

*# predicting*

y\_train\_predict**=**LR**.**predict(X\_train) y\_test\_predict**=**LR**.**predict(X\_test)

In [12]:

print("-------Test Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_test, y\_test\_predict)) print('MSE:', metrics**.**mean\_squared\_error(y\_test, y\_test\_predict))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_test\_predict)))

print("\n-------Train Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_train,y\_train\_predict)) print('MSE:', metrics**.**mean\_squared\_error(y\_train, y\_train\_predict))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_train, y\_train\_predict)))

print("\n-----Training Accuracy ")

print(round(LR**.**score(X\_train,y\_train),3)**\***100) print("-----Testing Accuracy ")

print(round(LR**.**score(X\_test,y\_test),3)**\***100)

-------Test Data--------

MAE: 149.04421616824322

MSE: 43810.98108666043

RMSE: 209.31072855126283

-------Train Data--------

MAE: 149.11934775839532

MSE: 42671.04510091187

RMSE: 206.56971002766082

-----Training Accuracy-------

85.8

-----Testing Accuracy--------

85.39999999999999

# SVM Regressor

In [ ]:

*#SVM regressor*

**from** sklearn **import** preprocessing

**from** sklearn **import** svm

svm\_regr **=** svm**.**SVC(kernel**=**'rbf')

svm\_regr**.**fit(X\_train, y\_train)

C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expec ted. Please change the shape of y to (n\_samples, ), for example using ravel

().

y = column\_or\_1d(y, warn=True)

In [ ]:

y\_test\_predict **=** svm\_regr**.**predict(X\_test)

y\_train\_predict **=** svm\_regr**.**predict(X\_train)

In [ ]:

**from** sklearn **import** metrics

print("-------Test Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_test, y\_test\_predict)) print('MSE:', metrics**.**mean\_squared\_error(y\_test, y\_test\_predict))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_test\_predict)))

print("\n-------Train Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_train,y\_train\_predict)) print('MSE:', metrics**.**mean\_squared\_error(y\_train, y\_train\_predict))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_train, y\_train\_predict)))

print("\n-----Training Accuracy ")

print(round(svm\_regr**.**score(X\_train,y\_train),3)**\***100) print("-----Testing Accuracy ")

print(round(svm\_regr**.**score(X\_test,y\_test),3)**\***100)

# Decision Tree Regressor

In [10]:

**from** sklearn.tree **import** DecisionTreeRegressor

In [11]:

dec\_model **=** DecisionTreeRegressor(random\_state **=**1)

In [12]:

dec\_model**.**fit(X\_train,y\_train)

Out[12]: DecisionTreeRegressor(random\_state=1) In [13]:

*#13.Test the model*

y\_test\_pred **=**dec\_model**.**predict(X\_test) y\_test

Out[13]: array([ -5, -8, 972, ..., 284, 66, -5])

In [14]:

y\_preds **=** dec\_model**.**predict(X\_train)

In [15]:

y\_test\_pred **=** dec\_model**.**predict(X\_test)

In [16]:

**import** math

**from** sklearn.metrics **import** mean\_absolute\_error,r2\_score print(math**.**sqrt(mean\_absolute\_error(y\_train,y\_preds)))

3.7195365334034864

In [17]:

print(math**.**sqrt(mean\_absolute\_error(y\_test,y\_test\_pred)))

7.900532258260076

In [40]:

**from** sklearn **import** metrics

print("-------Test Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_test, y\_test\_pred)) print('MSE:', metrics**.**mean\_squared\_error(y\_test, y\_test\_pred))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_test\_pred)))

print("\n-------Train Data ")

print('MAE:', metrics**.**mean\_absolute\_error(y\_train,y\_preds)) print('MSE:', metrics**.**mean\_squared\_error(y\_train, y\_preds))

print('RMSE:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_train, y\_preds)))

print("\n-----Training Accuracy ")

print(round(dec\_model**.**score(X\_train,y\_train),3)**\***100) print("-----Testing Accuracy ")

print(round(dec\_model**.**score(X\_test,y\_test),3)**\***100)

-------Test Data--------

MAE: 62.41840996380805

MSE: 14889.978097767444

RMSE: 122.02449794105873

-------Train Data--------

MAE: 13.834952023323225

MSE: 2258.2346855734895

RMSE: 47.520886834880194

-----Training Accuracy-------

99.2

-----Testing Accuracy--------

95.0

In [17]:

print(math**.**sqrt(mean\_absolute\_error(y\_test,y\_test\_pred)))

7.900532258260076

In [29]:

print(r2\_score(y\_train,y\_preds))

0.9924647507707248

In [18]:

print(r2\_score(y\_test,y\_test\_pred))

0.9502981638437535

In [18]:

*#save the model*

**import** joblib

joblib**.**dump(dec\_model,'dec\_model.sav')

Out[18]:

In [ ]:

['dec\_model.sav']

2018**-**01**-**01 15:40:00**+**00:00 216.0396777 27.39363139 258 4.479

In [31]:

y\_preds **=** model**.**predict([[1,1,27.39363139,258,4.479508]]) y\_preds

Out[31]:

In [13]:

**from** sklearn.ensemble **import** RandomForestRegressor

random\_forest\_model **=** RandomForestRegressor(max\_depth**=**100, max\_features**=**'sqrt min\_samples\_split**=**10, n\_estimators**=**800)

random\_forest\_model**.**fit(X\_train, y\_train)

array([212.])

# Random Forest Regressor

Out[13]:

In [14]:

y\_train\_predict**=**random\_forest\_model**.**predict(X\_train) y\_test\_predict**=**random\_forest\_model**.**predict(X\_test)

RandomForestRegressor(max\_depth=100, max\_features='sqrt', min\_samples\_leaf=4, min\_samples\_split=10, n\_estimators=800)

In [15]:

print("-----------Training Accuracy ")

print(round(random\_forest\_model**.**score(X\_train,y\_train),3)**\***100) print("-----------Testing Accuracy ")

print(round(random\_forest\_model**.**score(X\_test,y\_test),3)**\***100)

-----------Training Accuracy------------

97.6

-----------Testing Accuracy------------

96.7

In [16]:

**from** sklearn.metrics **import** mean\_squared\_error,r2\_score

r2\_score(y\_train,y\_train\_predict)

Out[16]: 0.9761993646024032 In [ ]:

In [17]:

r2\_score(y\_test,y\_test\_predict)

Out[17]: 0.9669643887056775

In [21]:

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** datetime **import** datetime

**from** matplotlib.pyplot **import** figure

**from** sklearn.preprocessing **import** MinMaxScaler

**import** sklearn

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.decomposition **import** PCA

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.ensemble **import** RandomForestClassifier

**import** xgboost **as** xg

**import** numpy **as** np

**from** sklearn **import** svm

**from** sklearn.linear\_model **import** LinearRegression

# XGBoost Regressor

In [22]:

xg\_model **=** xg**.**XGBRegressor()

In [23]:

xg\_model**.**fit(X\_train,y\_train)

Out[23]:

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1, early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise', importance\_type=None, interaction\_constraints='',

learning\_rate=0.300000012, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1, missing=nan, monotone\_constraints='()', n\_estimators=100, n\_jobs

=0,

=0,

num\_parallel\_tree=1, predictor='auto', random\_state=0, reg\_alpha reg\_lambda=1, ...)

In [23]:

y\_train\_predict**=**xg\_model**.**predict(X\_train) y\_test\_predict**=**xg\_model**.**predict(X\_test)

In [40]:

r2\_score(y\_train,y\_train\_predict)

Out[40]: 0.9695960085906646

In [31]:

*#x\_std = (x-x.min(axis =0))/(x.max(axis=0)- x.min(axis =0))*

**from** sklearn.preprocessing **import** MinMaxScaler scale1 **=** MinMaxScaler()

scale2 **=** MinMaxScaler()

xscaled **=** scale1**.**fit\_transform(X\_train) y\_train **=** y\_train**.**reshape(**-**1,1)

yscaled **=** scale2**.**fit\_transform(y\_train)

x\_test\_scaled **=** scale1**.**fit\_transform(X\_test) y\_test **=** y\_test**.**reshape(**-**1,1)

y\_test\_scaled **=** scale2**.**fit\_transform(y\_test)

In [25]:

xg\_model**.**fit(xscaled,yscaled)

Out[25]:

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1, early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise', importance\_type=None, interaction\_constraints='',

learning\_rate=0.300000012, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1, missing=nan, monotone\_constraints='()', n\_estimators=100, n\_jobs

=0,

=0,

num\_parallel\_tree=1, predictor='auto', random\_state=0, reg\_alpha reg\_lambda=1, ...)

In [41]:

y\_train\_scaled\_predict **=** xg\_model**.**predict(xscaled)

In [42]:

y\_test\_scaled\_pred **=** xg\_model**.**predict(x\_test\_scaled)

In [33]:

print(r2\_score(y\_test\_scaled,y\_test\_scaled\_pred))

0.9620078964068732

In [27]:

r2\_score(yscaled,y\_train\_scaled\_predict)

Out[27]: 0.9699157582990736

In [29]:

*#save the model*

**import** joblib

joblib**.**dump(xg\_model,'xg\_RFR\_forecast\_model.sav')

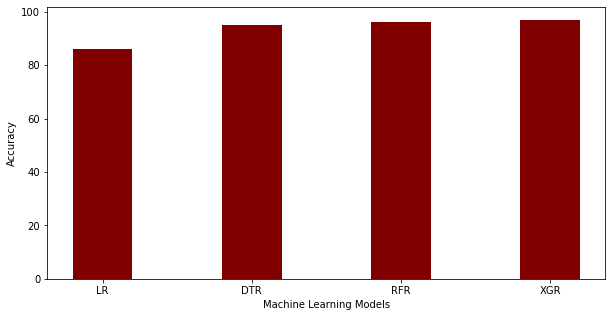
Out[29]: ['xg\_RFR\_forecast\_model.sav'] In [ ]:

# ploting accuracy graph to choose best model for prediction

In [1]:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt



In [3]:

data **=**{"LR":85.9,"DTR":95.1,"RFR":96.2,"XGR":96.9}

models **=** list(data**.**keys()) acc **=** list(data**.**values())

fig **=** plt**.**figure(figsize **=**(10,5))

*#bar plot*

plt**.**bar(models,acc,color **=**'maroon',width **=**0.4) plt**.**xlabel("Machine Learning Models")

plt**.**ylabel("Accuracy") plt**.**show()