HCL internship

Domain: Machine Learning

Project Title:

Air Quality Prediction

Name : K. Shiva Shankar

Reg no : 39110453

Department : Computer Science and Engineering

AIR POLLUTION PREDICTION USING MACHINE LEARNING

Introduction:

AS per World Health Organization [WHO] air pollution is infectivity of the indoor or outdoor environment by any chemical and biological agent which changes characteristics of the environment. Household combustion devices, vehicles and forest fires are common origin of air pollution and noise

pollution.

Problem statement:

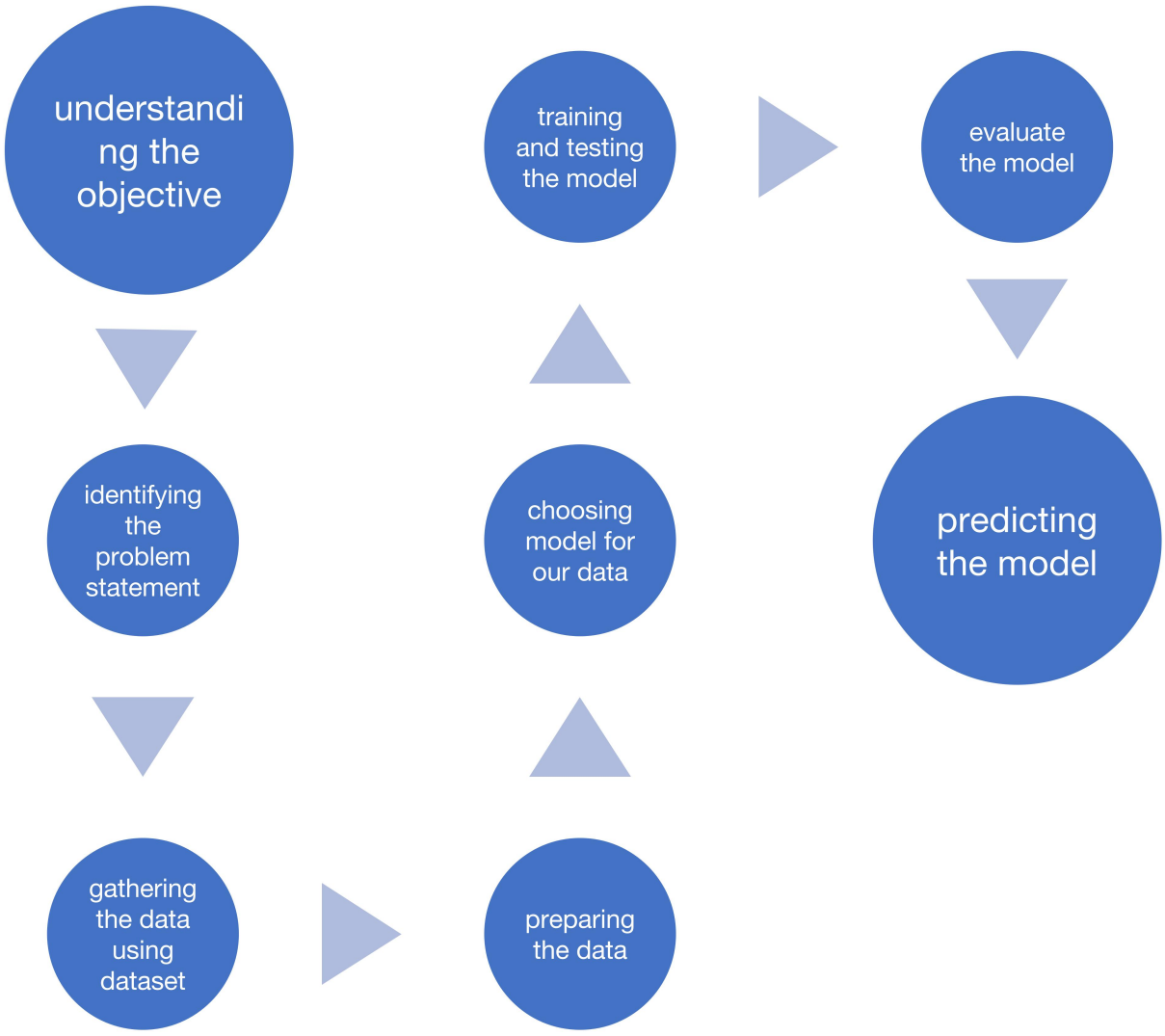
As we all know air pollution is one of the major problems in urban cities, where the particulate matter is the most dangerous part of air pollution which effects human than any other substances.

Objective:

The main objective of the study:

1. To predict real-time air pollution levels in urban cities.
2. To provide air quality information for the public.
3. To identify and provide different categories and levels of pollution in air
4. To develop more accurate and cost-effective air quality prediction system by using ml algorithm.

Architecture diagram:



Source Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.metrics import accuracy_score,confusion_matrix
data=pd.read_csv("/Users/apple/Desktop/HCL PROJECT/data.csv")
df.head()
df.shape
df.info()
df.isnull().sum()
df.describe()
df.nunique()
df.columns
sns.pairplot(data=df)
df['state'].value_counts()
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.state.hist()
```

```
plt.xlabel('state')
plt.ylabel('Frequencies')
plt.plot()
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.state.hist()
plt.xlabel('state')
plt.ylabel('Frequencies')
plt.plot()
df['type'].value_counts()
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.type.hist()
plt.xlabel('Type')
plt.ylabel('Frequencies')
plt.plot()
df['agency'].value_counts()
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.agency.hist()
plt.xlabel('Agency')
plt.ylabel('Frequencies')
plt.plot()
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='so2',data=df);
plt.rcParams['figure.figsize']=(30,10)
df[['so2','state']].groupby(["state"]).mean().sort_values(by='so2').plot.bar(color='purple')
plt.show()
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
```

```
sns.barplot(x='state',y='no2',data=df);
df[['no2','state']].groupby(["state"]).mean().sort_values(by='no2').plot.bar(color='purple')
plt.show()
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='rspm',data=df);
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='spm',data=df);
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='pm2_5',data=df);
nullvalues = df.isnull().sum().sort_values(ascending=False)
nullvalues
missing_data_with_percentage= pd.concat([nullvalues, null_values_percentage], axis=1, keys=['Total', 'Percent'])
missing_data_with_percentage
df.drop(['agency'],axis=1,inplace=True)
df.drop(['stn_code'],axis=1,inplace=True)
df.drop(['date'],axis=1,inplace=True)
df.drop(['sampling_date'],axis=1,inplace=True)
df.drop(['location_monitoring_station'],axis=1,inplace=True)
df.isnull().sum()
df
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])
df.fillna(0, inplace=True)
df.isnull().sum()
df
def cal_SOi(so2):
    si=0
    if (so2<=40):
```

```
    si= so2*(50/40)
   elif (so2>40 and so2<=80):
    si= 50+(so2-40)*(50/40)
   elif (so2>80 and so2<=380):
    si= 100+(so2-80)*(100/300)
   elif (so2>380 and so2<=800):
    si= 200+(so2-380)*(100/420)
   elif (so2>800 and so2<=1600):
    si= 300+(so2-800)*(100/800)
   elif (so2>1600):
    si= 400+(so2-1600)*(100/800)
   return si
df['SOi']=df['so2'].apply(cal_SOi)
data= df[['so2','SOi']]
data.head()
def cal_Noi(no2):
   ni=0
   if(no2<=40):
    ni= no2*50/40
   elif(no2>40 and no2<=80):
    ni= 50+(no2-40)*(50/40)
   elif(no2>80 and no2<=180):
    ni= 100+(no2-80)*(100/100)
   elif(no2>180 and no2<=280):
    ni= 200+(no2-180)*(100/100)
   elif(no2>280 and no2<=400):
    ni= 300+(no2-280)*(100/120)
   else:
    ni= 400+(no2-400)*(100/120)
   return ni
df['Noi']=df['no2'].apply(cal_Noi)
```

```
data= df[['no2','Noi']]
data.head()
def cal_RSPMI(rspm):
    rpi=0
    if(rpi<=30):
     rpi=rpi*50/30
    elif(rpi>30 and rpi<=60):
     rpi=50+(rpi-30)*50/30
    elif(rpi>60 and rpi<=90):
     rpi=100+(rpi-60)*100/30
    elif(rpi>90 and rpi<=120):
     rpi=200+(rpi-90)*100/30
    elif(rpi>120 and rpi<=250):
     rpi=300+(rpi-120)*(100/130)
    else:
     rpi=400+(rpi-250)*(100/130)
    return rpi
df['Rpi']=df['rspm'].apply(cal_RSPMI)
data= df[['rspm','Rpi']]
data.head()
def cal_SPMi(spm):
    spi=0
    if(spm<=50):
     spi=spm*50/50
    elif(spm>50 and spm<=100):
     spi=50+(spm-50)*(50/50)
    elif(spm>100 and spm<=250):
     spi= 100+(spm-100)*(100/150)
    elif(spm>250 and spm<=350):
     spi=200+(spm-250)*(100/100)
    elif(spm>350 and spm<=430):
```

```
    spi=300+(spm-350)*(100/80)
  else:
    spi=400+(spm-430)*(100/430)
  return spi

df['SPMi']=df['spm'].apply(cal_SPMi)
data= df[['spm','SPMi']]
data.head()
def cal_aqi(si,ni,rspmi,spmi):
  aqi=0
  if(si>ni and si>rspmi and si>spmi):
    aqi=si
  if(ni>si and ni>rspmi and ni>spmi):
    aqi=ni
  if(rspmi>si and rspmi>ni and rspmi>spmi):
    aqi=rspmi
  if(spmi>si and spmi>ni and spmi>rspmi):
    aqi=spmi
  return aqi

df['AQI']=df.apply(lambda x:cal_aqi(x['SOi'],x['Noi'],x['Rpi'],x['SPMi']),axis=1)
data= df[['state','SOi','Noi','Rpi','SPMi','AQI']]
data.head()
def AQI_Range(x):
  if x<=50:
    return "Good"
  elif x>50 and x<=100:
    return "Moderate"
  elif x>100 and x<=200:
    return "Poor"
  elif x>200 and x<=300:
```

```
        return "Unhealthy"
    elif x>300 and x<=400:
        return "Very unhealthy"
    elif x>400:
        return "Hazardous"

df['AQI_Range'] = df['AQI'] .apply(AQI_Range)
df.head()
df['AQI_Range'].value_counts()
X=df[['SOi','Noi','Rpi','SPMi']]
Y=df['AQI']
X.head()
Y.head()
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=70)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
DT=DecisionTreeRegressor()
DT.fit(X_train,Y_train)
#predicting train
train_preds=DT.predict(X_train)
#predicting on test
test_preds=DT.predict(X_test)
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',DT.score(X_train, Y_train))
print('RSquared value on test:',DT.score(X_test, Y_test))
RF=RandomForestRegressor().fit(X_train,Y_train)
#predicting train
train_preds1=RF.predict(X_train)
```

```
#predicting on test
test_preds1=RF.predict(X_test)
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds1)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds1)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',RF.score(X_train, Y_train))
print('RSquared value on test:',RF.score(X_test, Y_test))
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
X2 = df[['SOi','Noi','Rpi','SPMi']]
Y2 = df['AQI_Range']
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X2, Y2, test_size=0.33, random_state=70)
#fit the model on train data
DT2 = DecisionTreeClassifier().fit(X_train2,Y_train2)

#predict on train
train_preds3 = DT2.predict(X_train2)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train2, train_preds3))

#predict on test
test_preds3 = DT2.predict(X_test2)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test2, test_preds3))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test2,test_preds3))
#fit the model on train data
```

```
RF=RandomForestClassifier().fit(X_train2,Y_train2)
#predict on train
train_preds4 = RF.predict(X_train2)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train2, train_preds4))

#predict on test
test_preds4 = RF.predict(X_test2)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test2, test_preds4))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test2,test_preds4))
import pickle
pickle.dump(RF,open('airpollution.pkl','wb'))
```