**Naan Mudhalvan-Artificial Intelligence -Group 1**

**Earthquake Prediction Model Using Python**

**Abstract:**

It is well known that some areas are more vulnerable to natural disasters than others, such as earthquakes. It's crucial to remember that while certain areas may have a history of frequent earthquakes, accurately predicting earthquakes based only on historical information, such as date and time, latitude, and longitude, is not the same as anticipating trends in more predictable phenomena. Earthquakes happen naturally, and their patterns are by nature complicated and frequently surprising.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

print(os.listdir("../input"))

data = pd.read\_csv("../input/database.csv")

data.head()

data.columns

data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]

data.head()

import datetime

import time

timestamp = []

for d, t **in** zip(data['Date'], data['Time']):

try:

ts = datetime.datetime.strptime(d+' '+t, '%m/**%d**/%Y %H:%M:%S')

timestamp.append(time.mktime(ts.timetuple()))

timestamp.append('ValueError')

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timeStamp.values

**1.Data Preparation:**

* Using Pandas, you read seismic data from a CSV file.
* You handle datetime parsing issues and extract specified columns (such as "Timestamp," "Latitude," "Longitude," "Magnitude," and "Depth").
* Rows with "ValueError" in the "Timestamp" column are removed

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError']

final\_data.head()

from mpl\_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat\_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()

latitudes = data["Latitude"].tolist()

x,y = m(longitudes,latitudes)

fig = plt.figure(figsize=(12,10))

plt.title("All affected areas")

m.plot(x, y, "o", markersize = 2, color = 'blue')

m.drawcoastlines()

m.fillcontinents(color='coral',lake\_color='aqua')

m.drawmapboundary()

m.drawcountries()

plt.show()

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timeStamp.value

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError']

final\_data.head()

**2. Data Visualization:**

* To visualize earthquake locations on a map, construct a Basemap.

from mpl\_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat\_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()

latitudes = data["Latitude"].tolist()

x,y = m(longitudes,latitudes)

fig = plt.figure(figsize=(12,10))

plt.title("All affected areas")

m.plot(x, y, "o", markersize = 2, color = 'blue')

m.drawcoastlines()

m.fillcontinents(color='coral',lake\_color='aqua')

m.drawmapboundary()

m.drawcountries()

plt.show()

**3. Data Splitting:**

* Using train\_test\_split, divide the data into training and testing sets.
* You train a Random Forest Regressor using the training set of data.
* The score() method is used to assess the model

X = final\_data[['Timestamp', 'Latitude', 'Longitude']]

y = final\_data[['Magnitude', 'Depth']]

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print(X\_train.shape, X\_test.shape, y\_train.shape, X\_test.shape)

from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random\_state=42)

reg.fit(X\_train, y\_train)

reg.predict(X\_test)

reg.score(X\_test, y\_test)

**4.Hyperparameter Tuning (Grid Search):**

* GridSearchCV is used to adjust the hyperparameters of the Random Forest Regressor.

from sklearn.model\_selection import GridSearchCV

parameters = {'n\_estimators':[10, 20, 50, 100, 200, 500]}

grid\_obj = GridSearchCV(reg, parameters)

grid\_fit = grid\_obj.fit(X\_train, y\_train)

best\_fit = grid\_fit.best\_estimator\_

best\_fit.predict(X\_test)

best\_fit.score(X\_test, y\_test)

**5.Neural Network (Keras):**

* You build an earthquake prediction Keras neural network model.
* To tune hyperparameters, you define a parameter grid.
* GridSearchCV is used to adjust the neural network's hyperparameters.

from keras.models import Sequential

from keras.layers import Dense

def create\_model(neurons, activation, optimizer, loss):

model = Sequential()

model.add(Dense(neurons, activation=activation, input\_shape=(3,)))

model.add(Dense(neurons, activation=activation))

model.add(Dense(2, activation='softmax'))

model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

return model

from keras.wrappers.scikit\_learn import KerasClassifier

model = KerasClassifier(build\_fn=create\_model, verbose=0)

neurons = [16]

batch\_size = [10]

epochs = [10]

activation = ['sigmoid', 'relu']

optimizer = ['SGD', 'Adadelta']

loss = ['squared\_hinge']

param\_grid = dict(neurons=neurons, batch\_size=batch\_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)

**6.Model Evaluation**:

* You use the training data to create the final neural network model.
* Using the test data, you evaluate the model.
* The trained neural network model is saved to a file called "earthquake.h5" in step eight.
* It appears that you are now asking for changes to the code and a suitable CSV file. Here are a few ideas:

**Data Quality:**

* + Ensure that your seismic data are of high quality. The performance of the model depends on it.

**Feature Engineering:**

* + Investigate and develop fresh characteristics that could raise prediction precision.

grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1)

grid\_result = grid.fit(X\_train, y\_train)

print("Best: **%f** using **%s**" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

means = grid\_result.cv\_results\_['mean\_test\_score']

stds = grid\_result.cv\_results\_['std\_test\_score']

params = grid\_result.cv\_results\_['params']

for mean, stdev, param **in** zip(means, stds, params):

print("**%f** (**%f**) with: **%r**" % (mean, stdev, param))

**7. Hyperparameter Tuning:**

* + To identify the ideal model configuration, experiment with various hyperparameters for both the neural network and the Random Forest Regressor.

**Data Scaling:**

* + Scaling or normalizing your input features is something to think about, especially if you're utilizing neural networks.

**Cross-Validation:**

* Use cross-validation to evaluate the stability of the model and decrease overfitting.

**8.Model Evaluation Metrics:**

* + To evaluate the performance of a model, use appropriate evaluation metrics for regression tasks, such as Mean Absolute Error (MAE) or Mean Squared Error (MSE).

model = Sequential()

model.add(Dense(16, activation='relu', input\_shape=(3,)))

model.add(Dense(16, activation='relu'))

model.add(Dense(2, activation='softmax'))

model.compile(optimizer='SGD', loss='squared\_hinge', metrics=['accuracy'])

model.fit(X\_train, y\_train, batch\_size=10, epochs=20, verbose=1, validation\_data=(X\_test, y\_test))

**9.Data Source:**

* Check that the earthquake dataset you have access to is reputable and current. On websites like the USGS Earthquake Hazards Program, you can find earthquake datasets.

[test\_loss, test\_acc] = model.evaluate(X\_test, y\_test)

print("Evaluation result on Test Data : Loss = **{}**, accuracy = **{}**".format(test\_loss, test\_acc))

**10.File directories:**

* + Ensure that the file directories you use to load data and save models are precise and easy to retrieve.

**Conclusion:**

In order to properly build a CSV file for earthquake data, make sure your dataset has necessary columns like "Date," "Time," "Latitude," "Longitude," "Magnitude," and "Depth." You can get reliable sources for earthquake data, such as the USGS Earthquake Hazards Program or others, from these repositories. When you obtain the data, verify that it is in the right format and make any necessary changes to your code to properly process and use this important information.