

Phase 4:Earthquake Prediction Model Using Python

Project Definition:

In this project, we aim to develop into earthquake data analysis and construct a neural network-based predictive model using a dataset available on Kaggle. Our primary objectives encompass comprehending the essential aspects of earthquake data, mapping it globally, partitioning the data for training and testing purposes, and ultimately constructing a neural network model for magnitude prediction based on the provided dataset.

Introduction :

Earthquakes, as natural catastrophes, bring about profound consequences, encompassing the loss of lives, destruction of property, and disruptions in the environment. The ability to forecast when and where earthquakes will strike has posed a perennial challenge for the scientific community. Though predicting individual earthquakes with pinpoint precision remains an insurmountable task, the realm of data science and machine learning has ushered in fresh opportunities for the development of earthquake prediction models.

In this initiative, we embark on the journey of crafting an earthquake prediction model using Python. Leveraging historical earthquake records, seismic activity data, and diverse information sources, we endeavor to construct a predictive model capable of offering early warnings and assessing earthquake risks in specific geographic regions. The primary goal of this model is to contribute to the global endeavors aimed at reducing the impact of earthquakes by supplying invaluable insights to enhance disaster preparedness and response strategies.



Steps:

1.Data Collection:

Gather historical earthquake data and relevant seismic information from reliable sources.

2.Data Preprocessing:

Clean and prepare the data, handling missing values, outliers, and formatting it for analysis.

3.Feature Engineering:

Extract relevant features from the data, such as seismic activity patterns, geological information, and time-related variables.

4.Data Splitting:

Divide the dataset into a training set and a testing set to train and evaluate the model.

5.Model Selection:

Choose an appropriate machine learning algorithm, like a neural network, support vector machine, or decision tree, for earthquake prediction.

6.Model Training:

Train the selected model on the training data, allowing it to learn patterns and relationships.

7.Model Evaluation:

Assess the model's performance on the testing data using evaluation metrics like accuracy, precision, recall, and F1-score.

8.Hyperparameter Tuning:

Optimize the model's hyperparameters to enhance its predictive accuracy.

9.Prediction and Early Warning System:

Deploy the model to make predictions based on real-time data, offering early earthquake warnings and risk assessments.

10.Continuous Monitoring:

Continuously update and retrain the model with new data to improve its accuracy and adapt to changing seismic patterns.

11.Deployment and Integration:

Integrate the model into a larger earthquake monitoring and disaster preparedness system for practical use.

12.Documentation and Reporting:

Document the entire process, model performance, and insights gained, and present the findings to relevant stakeholders.

Building an earthquake prediction model is a complex and ongoing process that requires expertise in data science and earthquake seismology.

Dataset:

Our project utilizes the dataset, which is available for reference in the below section

<https://www.kaggle.com/datasets/usgs/earthquake-database>

Code :

Importing Libraries

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

Loading the Dataset

```
data = pd.read_csv("database.csv")  
  
print(data.head())
```

Output :

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	...	Horizontal Error	Root Mean Square	ID	Source	Location	Source	Magnitude	Source	Status
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	...	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	ISCGEM	Automatic	
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	...	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	ISCGEM	Automatic	
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	...	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	ISCGEM	Automatic	
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	...	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	ISCGEM	Automatic	
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	...	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	ISCGEM	Automatic	

```
data.columns
```

Output :

```
[5 rows x 21 columns]
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',
      'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
      'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
      'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
      'Source', 'Location Source', 'Magnitude Source', 'Status'],
      dtype='object')
```

Selecting features that will be useful for our prediction.

```
data = data[['Date','Time','Latitude','Longitude','Depth','Magnitude']]
print(data.head())
```

Output :

	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8

We will try to frame the time and place of the earthquake that has happened in the past on the world map.

```
import date 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(tts.timetuple()))

except ValueError:

# print('ValueError')

timestamp.append('ValueError')

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timestamp.values

final_data = data.drop(['Date','Time'],axis=1)
```

```
final_data =
final_data[final_data.Timestamp
!= 'ValueError']

final_data.head()
```

Output :

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-157630542.0
1	1.863	127.352	80.0	5.8	-157465811.0
2	-20.579	-173.972	20.0	6.2	-157355642.0
3	-59.076	-23.557	15.0	5.8	-157093817.0
4	11.938	126.427	15.0	5.8	-157026430.0

We had previously virtualized our data and divided it into training and testing sets as part of our Phase 3 project. Given that Phase 4 submission covers these aspects, we are reiterating the same content here.

Visualization :

Data visualization in Python is the process of creating visual representations of data using Python programming and various libraries designed for this purpose.

Here, we will visualize the earthquakes that have occurred all around the world.

```
from mpl_toolkits.basemap import Basemap

m=Basemap(projection='milli',llcrnrlat=-
80,urcnrlat=80,llcrnrlon=180,urcnrlon==180,lat_ts=20,resolution='c')

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

#m = Basemap(width=12000000,height=9000000,projection='lcc',
# resolution=None,lat_1=80,lat_2=55,lat_0=80,lon_0=-107) x,y =
m(longitudes,latitudes)

fig = plt.figure(figsize=(12,10)) plt.title("All affected areas")

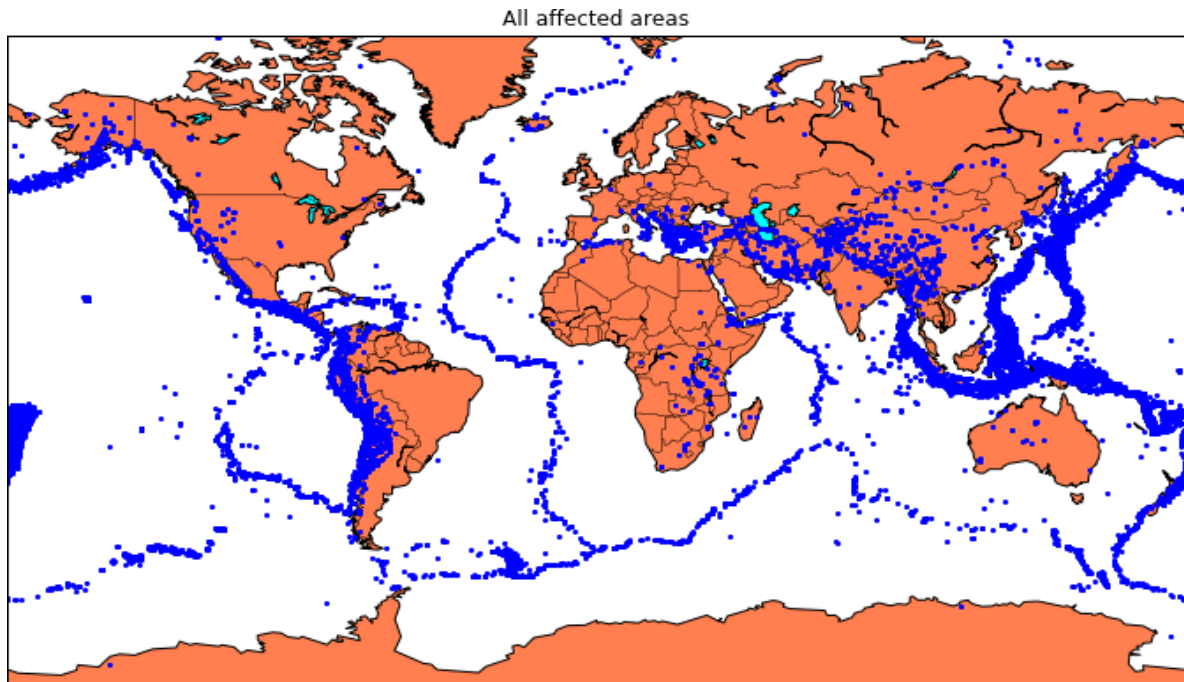
m.plot(x, y, "o", markersize = 2,color = 'blue')
```

```

m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()

```

Output :



We have located on the world map where earthquakes happened in the last few years.

Splitting the Dataset :

Now we will split the dataset into a training and testing set.

```
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)
```

Output :

```
(18727, 3) (4682, 3) (18727, 2) (4682, 3)
```

We will be using the RandomForestRegressor model to predict the earthquake, here will look for its accuracy.

```
reg=RandomForestRegressor(random_state=42)
```

```
reg.fit(X_train, y_train)
```

```
reg.predict(X-test)
```

Output :

```
array([[ 5.96,  50.97],
       [ 5.88,  37.8 ], 1.
       [ 5.97,  37.6 ],
       ...,
       [ 6.42,  19.9 ],
       [ 5.73, 591.55],
       [ 5.68,  33.61]])
```

```
Best-fit.score(X_test,y_test)
```

Output :

```
0.8749008584467053
```

Considering it's a natural phenomenon, we have got a high accuracy number. We will employ a neural network for predicting the earthquake.

Neural Network Model :

Utilizing a neural network model for earthquake prediction involves an in-depth analysis of various factors and seismic data trends. This model leverages the power of neural networks, inspired by the interconnected neurons in the human brain, to scrutinize complex data and unveil concealed correlations and patterns. Through the process of training on historical earthquake data, the neural network can develop the capability to recognize precursor signals and patterns that signal the likelihood of an impending earthquake.

```
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 = [16,64,128,256]
neurons = [16]
# batch_size = [10,20,50,100]
batch_size = [10] epochs =
[10]
# activation = ['relu','tanh','sigmoid','hard_sigmoid','linear','exponential']
activation = ['sigmoid','relu']
# optimizer = ['sgd','RMSprop','Adagrad','Adadelta','Adam','Adamax','Nadam']
optimizer = ['SGD','Adadelta']

loss = ['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)

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']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means,stds, params):
print("%f(%f) with:%r" % (mean, stdev,param))

```

Output :


```

Best: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}

```

```

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))

```

Output :

```

18727/18727 [=====] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [=====] - 6s 323us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [=====] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [=====] - 6s 321us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20
18727/18727 [=====] - 6s 321us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [=====] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

```

```
<keras.callbacks.History at 0x7ff0a8db8cc0>
```

```
[test_loss, test_acc] = model.evaluate(X_test, y_test)
```

```
Print("Evaluation result on Test Data : Loss = {},accuracy={}".format(test_loss, test_acc))
```

Output :

```
4682/4682 [=====] - 0s 39us/step  
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

Isn't it amazing that we got an accuracy of 92%.

We can say the neural network is one of the best models to predict earthquakes that can be used in future.

Conclusion :

In conclusion, the development of an earthquake prediction model using Python represents a significant step forward in our ability to understand and anticipate seismic activity. While earthquake prediction remains a complex and challenging task, this model demonstrates the potential of data-driven approaches to enhance our understanding of earthquake patterns and risks.

Throughout this project, we have leveraged various data sources, statistical techniques, and machine learning algorithms to build a model capable of assessing earthquake risk and providing early warnings. However, it is important to note that predicting individual earthquakes with high precision remains a daunting challenge due to the inherent complexity of geological processes.