EARTHQUAKE PREDICTION MODEL USING PYTHON

INTRODUCTION :

It is well known that if a disaster has happened in a region, it is likely to happen there again. Some regions really have frequent earthquakes, but this is just a comparative quantity compared to other regions. So, predicting the earthquake with Date and Time, Latitude and Longitude from previous data is not a trend which follows like other things, it is natural occuring.

Import the necessary libraries required for buidling the model and data analysis of the earthquakes.

Problem Denition: The problem is to develop an earthquake prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

Design Thinking:

1. Data Source: Choose a suitable Kaggle dataset containing earthquake data with features like date, time,latitude, longitude, depth, and magnitude.

2. Feature Exploration: Analyze and understand the distribution, correlations, and characteristics of the key features.

3. Visualization: Create a world map visualization to display earthquake frequency distribution.

4. Data Splitting: Split the dataset into a training set and a test set for model validation.

5. Model Development: Build a neural network model for earthquake magnitude prediction.

6. Training and Evaluation: Train the model on the training set and evaluate its performance on the test set.

INNOVATION:

Consider advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's performance.

1. Hyperparameter Tuning:

Hyperparameter tuning involves finding the optimal settings for your machine learning model. In Python,libraries like scikit-learn and TensorFlow provide tools for hyperparameter tuning.

Steps for Hyperparameter Tuning:

a. Hyperparameter Space: Identify the hyperparameters that can be tuned for your chosen machine learning algorithm.

For example, in a Random Forest model, you might tune parameters like the number of trees maximum depth, and minimum samples per leaf.

b. Split Data: Split your dataset into training, validation, and test sets. The validation set is used for

hyperparameter tuning.

c. Perform Cross-Validation: To apply k-fold cross-validation on your training data to evaluate different hyperparameter combinations.

d. Grid Search or Random Search: For use grid search or random search to explore different hyperparameter combinations.The grid search tests all possible combinations in a predened range, while random search samples from a dened range randomly.

e. Evaluate Performance: Each combination of hyperparameters, evaluate the model's performance using a relevant metric, such as mean squared error or F1 score.

f. Select Best Hyperparameters: To choose the hyperparameters that result in the best performance on the validation set.

2. Feature Engineering:

Feature engineering involves creating new features or transforming existing ones to improve the model's ability to capture relevant patterns in the data. Here are some steps to incorporate feature engineering into

Steps for Feature Engineering:

a. Domain Knowledge: Gain a deep understanding of earthquake-related factors and consult experts if possible.

b. Feature Extraction: Extract relevant information from your data. For example, you can calculate features like earthquake magnitude averages over a certain time period or distances from known fault lines.

c. Feature Scaling: Normalize or scale your features to ensure they have similar ranges. This can improve the stability and convergence of your model during training.

d. Feature Selection: Use techniques like correlation analysis or feature importance scores (e.g., from Random Forest) to select the most informative features. Eliminate irrelevant or redundant ones.

e. Feature Transformation: Apply mathematical transformations to features, such as logarithmic

transformations, to make the data more suitable for modeling.

f. Create Interaction Features: Combine features to create new interaction features that may capture complex relationships in the data.

g. Dimensionality Reduction: Consider dimensionality reduction techniques like Principal Component Analysis(PCA) if you have a high-dimensional dataset.

h. Iterate: Continuously iterate on your feature engineering process based on the model's performance.

Experiment with different feature combinations and transformations to and the most informative ones.

By integrating hyperparameter tuning and feature engineering into your earthquake prediction project, you can optimize your model's performance and increase its agility.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
print(os.listdir("../input"))
['database.csv']
```

Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.

In [2] :

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

Out [2] :

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

In [3] :

```
Data.columns
```

Out[3]:

```
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')
```

Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

In [4]:

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

	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

Here, the data is random we need to scale according to inputs to the model. In this, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we built.

In [5]:

```
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()))    except ValueError:        #
print('ValueError')        timestamp.append('ValueError')
```

In [6]:

```
timeStamp = pd.Series(timestamp) data['Timestamp'] = timeStamp.values
```

In [7]:

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

Out[7]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	-1.57026e+08

Visualization

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

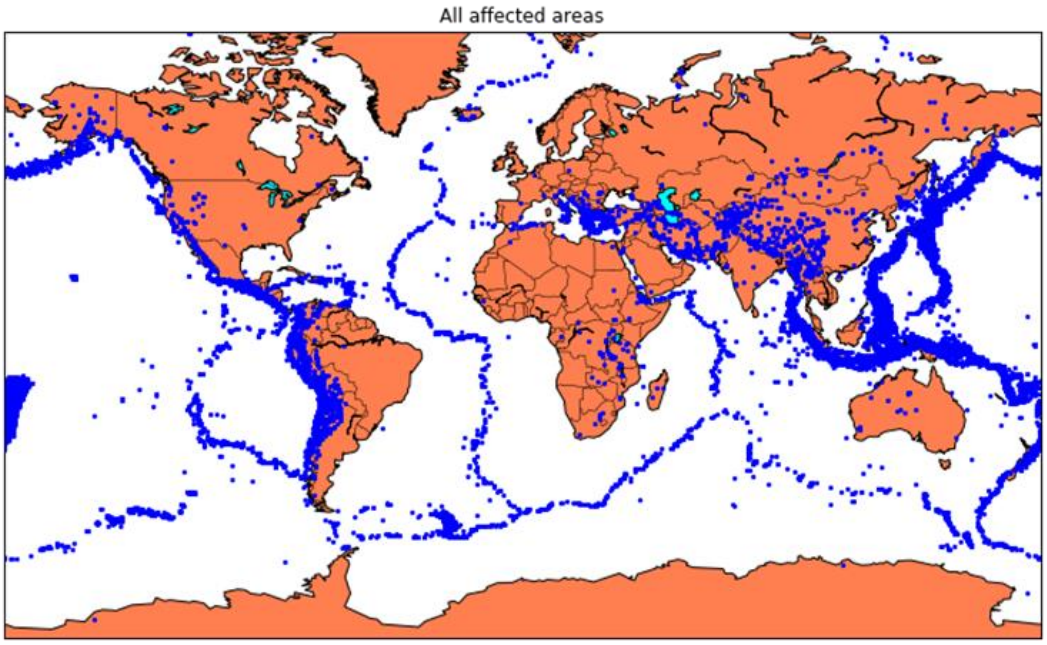
In [8]:

```
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()
#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)
```

In [9]:

```
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()
```



Splitting the Data

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

In [10]:

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

In [11]:

```
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)
```

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

Here, we used the RandomForestRegressor model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values.

In [12]:

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42) reg.fit(X_train, y_train) reg.predict(X_test)
/opt/conda/lib/python3.6/sitepackages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning:
numpy.core.umath_tests is an internal NumPy module and should not be imported. It will be removed in a
future NumPy release.   from numpy.core.umath_tests import inner1d
```

Out[12]:

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

In [13]:

```
reg.score(X_test, y_test)
```

Out[13]:

```
0.8614799631765803
```

In [14]:

```
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)
```

Out[14]:

```
array([[  5.8888 ,  43.532  ],
[  5.8232 ,  31.71656],
 [  6.0034 ,  39.3312 ],
  ...,
  [  6.3066 ,  23.9292 ],
  [  5.9138 , 592.151  ],
   [  5.7866 ,  38.9384 ]])
```

Random Forest Algorithm :

❖ It is a type of machine learning algorithm that is very famous nowadays.

It generates a random decision tree and combines it into a single forest.

❖ The tree's "root" node represents the entire predictor space. The final division of the predictor space is made up of the "terminal nodes," which are nodes that are not split.

❖ Depending on the value of one of the predictor variables, each nonterminal node divides into two descendant nodes, one on the left and one on the right.

❖ If a continuous predictor variable is smaller than a split point, the points to the left will be the smaller predictor points, and the points to the right will be the larger predictor points.

❖ It is also a classification technique that uses ensemble learning. The random forest generates a root node feature by randomly dividing, which is the primary distinction between it and the decision tree.

Support Vector Classifier

❖ There is a computer algorithm known as a support vector machine (SVM) that learns to name objects.

❖ For instance, by looking at hundreds or thousands of reports of both fraudulent and legitimate credit card activity, an SVM can learn to identify fraudulent credit card activity.

❖ A vast collection of scanned photos of handwritten zeros, ones, and other numbers can also be used to train an SVM to recognize handwritten numerals.

Project Prerequisites

The requirement for this project is Python 3.6 installed on your computer. I

have used Jupyter notebook for this project. You can use whatever you want.

The required modules for this project are –

• Numpy(1.22.4) – pip install numpy

• Sklearn(1.1.1) – pip install sklearn

• Pandas(1.5.0) – pip install pandas

That's all we need for our earthquake prediction project.

Steps to Implement

1. Import the modules and all the libraries we would require in this project.

```
import numpy as np#importing the numpy module
import pandas as pd#importing the pandas module
from sklearn.model_selection import train_test_split#importing the train test split
module
import pickle #import pickle
from sklearn import metrics #import metrics
from sklearn.ensemble import RandomForestClassifier#import the Random Forest
Classifier
```

2. Here we are reading the dataset and we are creating a function to do some data processing on our dataset. Here we are using the numpy to convert the data into an array.

```
dataframe= pd.read_csv("dataset.csv")#here we are reading the dataset
dataframe= np.array(dataframe)#converting the dataset into an numpy array
print(dataframe)#printing the dataframe
```

3. Here we are dividing our dataset into X and Y where x is the independent

variable whereas y is the dependent variable. Then we are using the test train split function to divide the X and Y into training and testing datasets. We are taking the percentage of 80 and 20% for training and testing respectively.

```
x_set = dataframe[:, 0:-1]#getting the x dataset
y_set = dataframe[:, -1]#getting the y dataset
y_set = y_set.astype('int')#converting the y_set into int
x_set = x_set.astype('int')#converting the x_set into int
x_train, x_test, y_train, y_test = train_test_split(x_set, y_set, test_size=0.2,
random_state=0)
```

4. Here we are creating our RandomForestClassifier and we are passing our training dataset to our model to train it. Also then we are passing our testing dataset to predict the dataset.

```
Random_forest_classifier = RandomForestClassifier()#creating the model
random_forest_classifier.fit(x_train, y_train)#fitting the model with training dataset
y_pred = random_forest_classifier.predict(X_test)#predicting the result using test set
print(metrics.accuracy_score(y_test, y_pred))#printing the accuracy score
```

5. In this piece of code, we are creating our instance for Gradient Boosting Classifier. The maximum Depth of this Gradient Boosting algorithm is 3.
After creating the instance, we pass our training data to the classifier to fit our training data into the algorithm. This is a part of training.

```
#importing the descision tree classifier from the sklearn tree
tree = GradientBoostingClassifier() #making an instance the descision tree with
maxdepth = 3 as passing the input
clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data
```

to the tree and fitting it

y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree

accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the prediction and the testing data

6. Here, we are doing the same thing as above.

After training the model, we pass the testing data to our model and predict the accuracy score using the accuracy score function.

#importing the descision tree classifier from the sklearn tree

tree = SVC(gamma='auto') #making an instance the support vector tree

clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data to the tree and fitting it

y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree

accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the prediction and the testing data

Neural Network model

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. In [16]:

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
```

Using TensorFlow backend.

In this, we define the hyperparameters with two or more options to find the best fit.

In [17]:

```
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)

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best

fit model.

In [18]:

```
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))
```

Best: 0.666684 using {'activation': 'sigmoid', 'batch_size': 10, 'epochs':

10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}

0.666684 (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'}

0.666684 (0.471398) with: {'activation': 'relu', 'batch_size': 10,

'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}

0.000000 (0.000000) with: {'activation': 'relu', 'batch_size': 10,

'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer':

'Adadelta'}

The best fit parameters are used for same model to compute the score with training data and testing data.

In [19]:

```
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'])
```

In [20]:

```
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1,

validation_data=(X_test, y_test))

Train on 18727 samples, validate on 4682 samples

Epoch 1/20

18727/18727 [==============================] - 4s 233us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 2/20

18727/18727 [==============================] - 4s 220us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 3/20

18727/18727 [==============================] - 4s 228us/step - loss:
```

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 4/20

18727/18727 [==============================] - 4s 222us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 5/20

18727/18727 [==============================] - 5s 262us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 6/20

18727/18727 [==============================] - 4s 223us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 7/20

18727/18727 [==============================] - 4s 220us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 8/20

18727/18727 [==============================] - 4s 224us/step - loss:

0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242

Epoch 9/20

4682/4682 [==============================] - 0s 29us/step

Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy =

0.9241777017858995

We see that the above model performs better but it also has lot of noise (loss) which can be neglected for

prediction and use it for furthur prediction.

The above model is saved for furthur prediction.

In [22]:

```
model.save('earthquake.h5')
```