**Earthquake Prediction Model Using Python:**

* It is well known that if a disaster occurs in one region, it is likely to happen again. Some regions have frequent earthquakes, but this is only a comparative amount compared to other regions.
* So, predicting the earthquake with date and time, latitude and longitude from previous data is not a trend that follows like other things, it happens naturally.
* I will start this task to create a model for earthquake prediction by importing the necessary python libraries:

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt

* Now let’s load and read the dataset. The dataset that I am using here can be easily downloaded here:

1. data = pd.read\_csv("database.csv")
2. data.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')

* Now let’s see the main characteristics of earthquake data and create an object of these characteristics, namely, date, time, latitude, longitude, depth, magnitude:

1. data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
2. 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

* Since the data is random, so we need to scale it based on the model inputs. In this, we convert the given date and time to Unix time which is in seconds and a number. This can be easily used as an entry for the network we have built:

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')

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()

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

**Data Visualization:**

* Now, before we create the earthquake prediction model, let’s visualize the data on a world map that shows a clear representation of where the earthquake frequency will be more:

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)

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

plt.title("All affected areas")

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

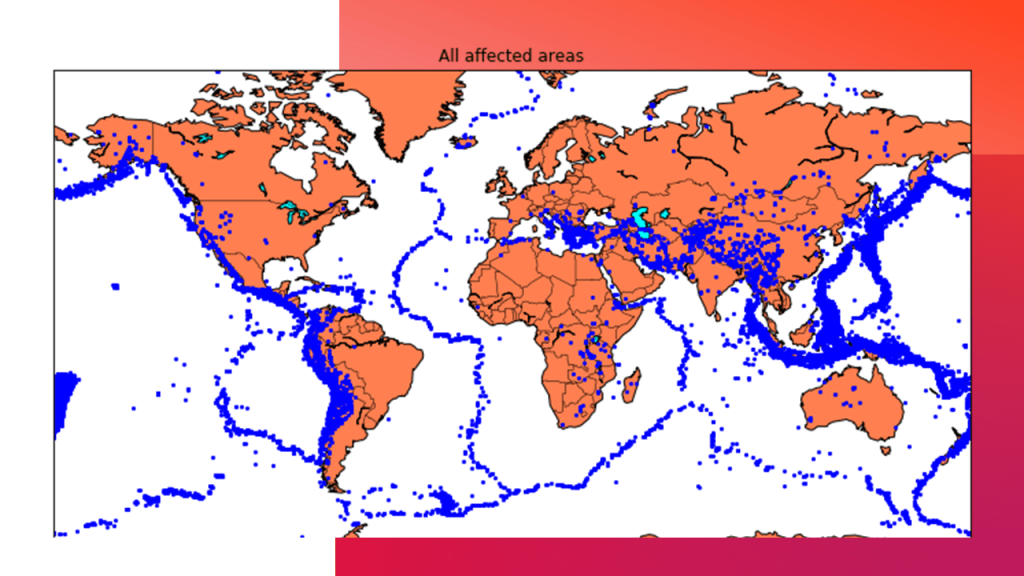
m.drawcoastlines()

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

m.drawmapboundary()

m.drawcountries()

plt.show()



**Splitting the Dataset:**

* Now, to create the earthquake prediction model, we need to divide the data into Xs and ys which respectively will be entered into the model as inputs to receive the output from the model.
* Here the inputs are TImestamp, Latitude and Longitude and the outputs are Magnitude and Depth. I’m going to split the xs and ys into train and test with validation. The training set contains 80% and the test set contains 20%:

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)

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

**Neural Network for Earthquake Prediction:**

* Now I will create a neural network to fit the data from the training set. Our neural network will consist of three dense layers each with 16, 16, 2 nodes and reread. Relu and softmax will be used as activation functions:

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

Now I’m going to define the hyperparameters with two or more options to find the best fit:

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)

Now we need to find the best fit of the above model and get the mean test score and standard deviation of the best fit model:

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: 0.957655 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'}

In the step below, the best-fit parameters are used for the same model to calculate the score with the training data and the test data:

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))

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

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