* **EARTHQUAKE PREDICTION MODEL USING PYTHON.**

**PHASE4: DEVELOPMENT PART 2.**

**Submitted by:**

**Ramanaboina Venkata Sai Hema Yadav**

**Au723921104042**

[**rhemashrie156@gmail.com**](mailto:rhemashrie156@gmail.com)

**problem :**

# Visualizing the data on a world map

# Splitting it into training and testing sets

**Content:**

Earthquake Prediction

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.

**Earthquake prediction model:**

import numpy asnp

import pandas aspd

import matplotlib.pyplot asplt

importos

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

['database.csv']

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

data.head()

importdatetime

importtime

timestamp = []

ford, 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

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 | -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.

frommpl\_toolkits.basemapimportBasemap

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()

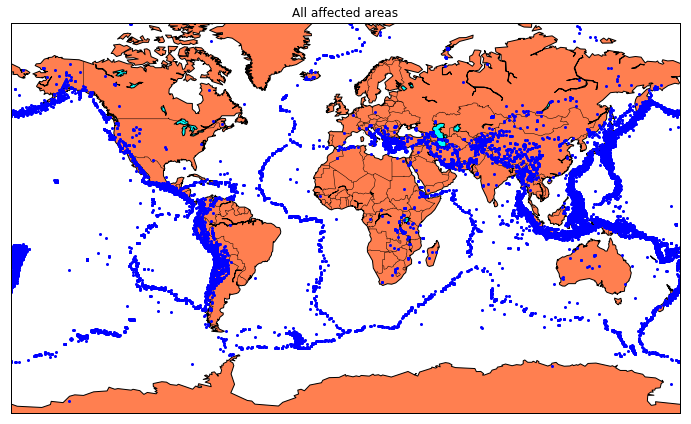
plt.show()

/opt/conda/lib/python3.6/site-packages/mpl\_toolkits/basemap/\_\_init\_\_.py:1704: MatplotlibDeprecationWarning: The axesPatch function was deprecated in version 2.1. Use Axes.patch instead.

limb = ax.axesPatch

/opt/conda/lib/python3.6/site-packages/mpl\_toolkits/basemap/\_\_init\_\_.py:1707: MatplotlibDeprecationWarning: The axesPatch function was deprecated in version 2.1. Use Axes.patch instead.

if limb is not ax.axesPatch:



### **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%.

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

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

from sklearn.cross\_validation importtrain\_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)

from sklearn.ensemble importRandomForestRegressor

reg = RandomForestRegressor(random\_state=42)

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

reg.predict(X\_test)

/opt/conda/lib/python3.6/site-packages/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

output:

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]])

from sklearn.model\_selection importGridSearchCV

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)

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 ]])

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

output:

0.8749008584467053

### **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. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

fromkeras.modelsimportSequential

fromkeras.layersimportDense

defcreate\_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'])

returnmodel

Using TensorFlow backend.

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

from keras.wrappers.scikit\_learn importKerasClassifier

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']*

Using TensorFlow backend.

In this, we 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']*

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.

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))

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

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 10/20

18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 11/20

18727/18727 [==============================] - 4s 221us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 12/20

18727/18727 [==============================] - 4s 231us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 13/20

18727/18727 [==============================] - 5s 248us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 14/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 15/20

18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 16/20

18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 17/20

18727/18727 [==============================] - 4s 225us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 18/20

18727/18727 [==============================] - 4s 219us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 19/20

18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

Epoch 20/20

18727/18727 [==============================] - 5s 258us/step - loss: 0.5038 - acc: 0.9182 - val\_loss: 0.5038 - val\_acc: 0.9242

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

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

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

Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995

Output:

model.save('earthquake.h5')