Earthquake prediction model in Python

AI_Phases 4

K.ARTHI

711021104005

.

Get our environment set up

- The first thing we'll need to do is load in the libraries and dataset we'll be using. We'll be working with a dataset containing information on earthquakes that occured between 1965 and 2016.

- We have gathered this dataset from the publicly available domain Kaggle. We have used the Significant Earthquakes, 1965-2016 dataset from Kaggle in the CSV format. It includes a record of the date, time, location, depth, magnitude, and source of every earthquake with a reported magnitude 5.5 or higher since 1965.

- ```
  # modules we'll use import pandas as pd import numpy as np import seaborn as sns import datetime

  # read in our data earthquakes = pd.read_csv("../input/earthquake-database/database.csv")

  # set seed for reproducibility np.random.seed(0)
  ```

1) Check the data type of our date column

- We are working with the "Date" column from the earthquakes dataframe. We investigate this column now and see if it looks like it contains dates and what the dtype of the column is.

- ```
  # TODO: Your code here! earthquakes['Date'].head()
  ```

- ```
  0    01/02/1965 1    01/04/1965
  2    01/05/1965
  3    01/08/1965
  4    01/09/1965
  Name: Date, dtype: object
  ```

2) Convert our date columns to datetime

❖ Most of the entries in the "Date" column follow the same format: "month/day/four -digit year". However, the entry at index 3378 follows a completely different pattern. We run the code cell below to see this.earthquakes[3378:3383]

```
                Date                      Time  Latitude  Longitude  \
3378            1975-02-23T02:58:41.000Z  1975-02-23T02:58:41.000Z    8.017    124.075
3379            02/23/1975                03:53:36   -21.727    -71.356
3380            02/23/1975                07:34:11   -10.879    166.667
3381            02/25/1975                05:20:05    -7.388    149.798    3382            02/26/1975
                04:48:55    85.047    97.969
       Type  Depth  Depth Error  Depth Seismic Stations  Magnitude  \
```
```

```
3378  Earthquake  623.0        NaN                NaN       5.6
3379  Earthquake   33.0        NaN                NaN       5.6
3380  Earthquake   33.0        NaN                NaN       5.5
3381  Earthquake   33.0        NaN                NaN       5.5    3382  Earthquake   33.0        NaN
        NaN       5.6

     Magnitude Type  ...  Magnitude Seismic Stations  Azimuthal Gap  \
3378             MB  ...                         NaN            NaN
3379             MB  ...                         NaN            NaN    3380             MS  ...
                 NaN            NaN
3381             MB  ...                         NaN            NaN
3382             MS  ...                         NaN            NaN
      Horizontal Distance  Horizontal Error  Root Mean Square          ID  \
3378                  NaN               NaN               NaN  USP0000A09
3379                  NaN               NaN               NaN  USP0000A0A
3380                  NaN               NaN               NaN  USP0000A0C
3381                  NaN               NaN               NaN  USP0000A12
3382                  NaN               NaN               NaN  USP0000A1H

     Source Location Source Magnitude Source    Status
3378     US              US              US  Reviewed
3379     US              US              US  Reviewed
3380     US              US              US  Reviewed
3381     US              US              US  Reviewed
3382     US              US              US  Reviewed

[5 rows x 21 columns]
```

This does appear to be an issue with data entry: ideally, all entries in the column have the same format. We can get an idea of how widespread this issue is by checking the length of each entry in the "Date" column.

```
date_lengths = earthquakes.Date.str.len() date_lengths.value_counts()
```

```
10    23409
24        3
Name: Date, dtype: int64
```

Looks like there are two more rows that has a date in a different format. We Run the code cell below to obtain the indices corresponding to those rows and print the data.

```
indices = np.where([date_lengths == 24])[1]
print('Indices with corrupted data:', indices)
earthquakes.loc[indices]
```

```
Indices with corrupted data: [ 3378  7512 20650]

                          Date                      Time  Latitude  \
3378  1975-02-23T02:58:41.000Z  1975-02-23T02:58:41.000Z     8.017
```

```
7512   1985-04-28T02:53:41.530Z  1985-04-28T02:53:41.530Z   -32.998
20650  2011-03-13T02:23:34.520Z  2011-03-13T02:23:34.520Z    36.344

      Longitude       Type  Depth  Depth Error  Depth Seismic Stations  \
3378    124.075  Earthquake  623.0        NaN                   NaN
7512    -71.766  Earthquake   33.0        NaN                   NaN   20650
142.344  Earthquake   10.1        13.9                 289.0

      Magnitude Magnitude Type  ...  Magnitude Seismic Stations  \
3378        5.6             MB  ...                         NaN
7512        5.6             MW  ...                         NaN
20650       5.8            MWC  ...                         NaN

      Azimuthal Gap  Horizontal Distance  Horizontal Error  Root Mean Square  \
3378            NaN                  NaN               NaN               NaN
7512            NaN                  NaN               NaN              1.30
20650          32.3                  NaN               NaN              1.06
              ID Source Location Source Magnitude Source    Status
3378   USP0000A09     US              US               US  Reviewed
7512   USP0002E81     US              US              HRV  Reviewed
20650  USP000HWQP     US              US             GCMT  Reviewed

[3 rows x 21 columns]
```

Given all of this information, we create a new column "date_parsed" in the earthquakes dataset that has correctly parsed dates in it.

We have now converted all the date columns into datetime.

```
# TODO: Your code here earthquakes.loc[3378, "Date"] = "02/23/1975" earthquakes.loc[7512, "Date"] = "04/28/1985" earthquakes.loc[20650, "Date"] = "03/13/2011" earthquakes['date_parsed'] = pd.to_datetime(earthquakes['Date'], format="%m/%d/%Y")
```

3)Select the day of the month

Create a Pandas Series day_of_month_earthquakes containing the day of the month from the "date_parsed" column.

```
# try to get the day of the month from the date column day_of_month_earthquakes
= earthquakes['date_parsed'].dt.day
```

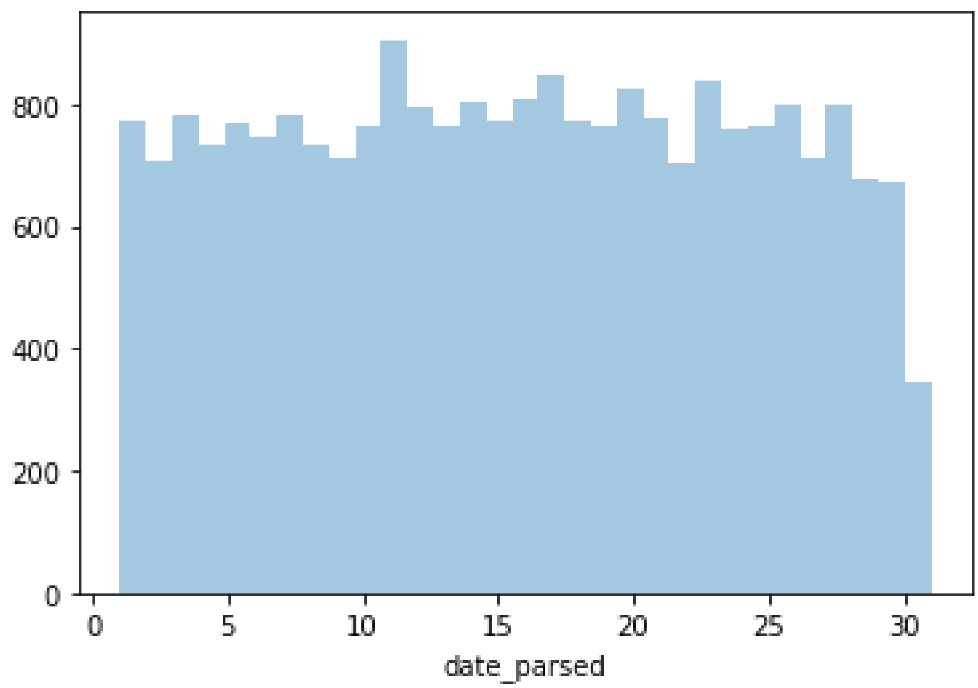
4)Plot the day of the month to check the date parsing

Plot the days of the month from your earthquake dataset.

```
# TODO: Your code here!
# remove na's day_of_month_earthquakes =
day_of_month_earthquakes.dropna()

# plot the day of the month sns.distplot(day_of_month_earthquakes,
kde=False, bins=31)
```

/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms). warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='date_parsed'>



Now we have visualized a graph that shows the days of the month. This data parsing is just for visualizing the data. When training, we import and use the dataset as it is.

Import Libraries and Dataset

Here we import the other neccessary libraries for further data visualization and import the dataset as well

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

```
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.

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

```
        Date      Time    ...     Magnitude Source     Status
1  01/02/1965  13:44:18   ...           ISCGEM  Automatic
2  01/04/1965  11:29:49   ...           ISCGEM  Automatic
3  01/05/1965  18:05:58   ...           ISCGEM  Automatic
4  01/08/1965  18:49:43   ...           ISCGEM  Automatic 4  01/09/1965  13:32:50   ...
   ISCGEM  Automatic

[5 rows x 21 columns] 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')
```

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

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

```
        Date      Time  Latitude  Longitude  Depth  Magnitude
1  01/02/1965  13:44:18    19.246    145.616  131.6       6.0
2  01/04/1965  11:29:49     1.863    127.352   80.0       5.8
3  01/05/1965  18:05:58   -20.579   -173.972   20.0       6.2
4  01/08/1965  18:49:43   -59.076    -23.557   15.0       5.8
5  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.

```
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
1    19.246    145.616  131.6       6.0 -1.57631e+08
2    1.863    127.352   80.0       5.8 -1.57466e+08
3    -20.579   -173.972   20.0       6.2 -1.57356e+08
4    -59.076    -23.557   15.0       5.8 -1.57094e+08
5    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.

```
from mpl_toolkits.basemap import Basemap

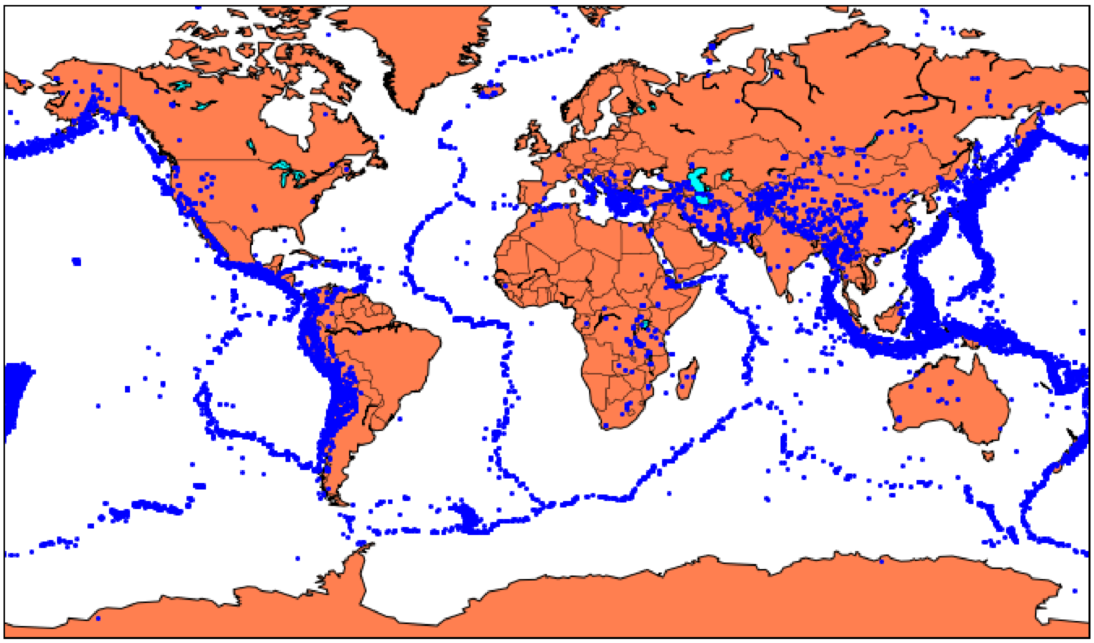
m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-
180,urcrnrlon=180,lat_ts=20,resolution='c')

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()

/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 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)

/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Training using Random Forest

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.

```
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/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
```

```
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]])
```

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

```
0.8614799631765803
```

```
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)
```

```
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)
```

```
0.8749008584467053
```

Building the 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.

```
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.

```
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.

```
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: 1.000000 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.936562 (0.000858) 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'}
0.646286 (0.411324) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
1.000000 (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 [==============================] - 3s 134us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 2/20
18727/18727 [==============================] - 2s 122us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 3/20
18727/18727 [==============================] - 2s 118us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 4/20
18727/18727 [==============================] - 2s 120us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 5/20
18727/18727 [==============================] - 2s 121us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 6/20
18727/18727 [==============================] - 3s 135us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 7/20
18727/18727 [==============================] - 2s 124us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 8/20
18727/18727 [==============================] - 2s 119us/step - loss: 0.5000 - acc:
0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 9/20
```

18727/18727 [==============================] - 2s 118us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 10/20
18727/18727 [==============================] - 2s 120us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 11/20
18727/18727 [==============================] - 2s 123us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 12/20
18727/18727 [==============================] - 2s 118us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 13/20
18727/18727 [==============================] - 2s 121us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 14/20
18727/18727 [==============================] - 2s 119us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 15/20
18727/18727 [==============================] - 2s 125us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 16/20
18727/18727 [==============================] - 2s 121us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 17/20
18727/18727 [==============================] - 2s 124us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 18/20
18727/18727 [==============================] - 2s 120us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 19/20
18727/18727 [==============================] - 2s 120us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186
Epoch 20/20
18727/18727 [==============================] - 3s 135us/step - loss: 0.5000 - acc: 0.0189 - val_loss: 0.5000 - val_acc: 0.0186

<keras.callbacks.History at 0x7838b345a358>

[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 22us/step
Evaluation result on Test Data : Loss = 0.5, accuracy = 0.018581802648440837
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 that could be done with a user interface. model.save('earthquake.h5')