TEAM MEMBERS

YUVASRI.M

SHOBA DEVI.M

SOWNDHARYA.R

RAMYA.S

VYSHALI.M

PROJECT MODULE :

BUILDING A EARTHQUAKE PREDICTION IN PYTHON USING KAGGLE DATASETS

Earthquake Prediction Model using Python

Problem Definition:

1.Problem Statement:

In todays world most of the region faces the unpredictable earthquake across the globe. So it is necessary to have a Earthquake Prediction Model. By having this model the risk can be reduced in various area that fall under the earthquake zones. By this prediction we can evacuate people for their well being.

1. Problem Scope:

It involves exploring and analyzing earthquake data to understand its key features, patterns, and trends. Specifically, the scope includes:

* **Data Collection and Exploration:** Gathering relevant earthquake data from reliable sources and exploring its structure and content to identify the available features.
* **Feature Analysis:** Investigating key features such as date and time of occurrence, geographical coordinates (latitude and longitude), depth, magnitude, intensity, seismic waveform data, fault information, tsunami potential, aftershocks, and historical data.
* **Data Visualization:** Creating visual representations of the data, such as maps and graphs, to understand spatial and temporal patterns of seismic activities.
* **Data Splitting:** Dividing the data into training and test sets to prepare for model development and validation.
* **Model Building :**Developing machine learning or neural network models for earthquake prediction based on the analyzed data.
* **Insights and Conclusions:** Drawing insights from the analysis, understanding the behavior of earthquakes in the given dataset, and potentially making recommendations for earthquake preparedness and risk mitigation.
* **Documentation and Reporting:** Documenting the analysis process, findings, challenges faced, and conclusions. This documentation might include code, visualizations, and a summary of the insights gained.

1. Problem Goal:

* To collect the earthquake data, including date and time, geographical coordinates, depth, magnitude, intensity, seismic waveform data, fault information, tsunami potential, aftershocks, and historical data For future understanding.
* Create visual representations (maps, graphs, charts) to illustrate spatial and temporal patterns of seismic activities. Visualization aids in identifying trends and anomalies in the data.
* Divide the dataset into training and test sets to facilitate the development and validation of machine learning or neural network models. This step ensures unbiased evaluation of the model's performance.
* To divide the dataset into training and test sets to facilitate the development and validation of machine learning or neural network models. This step ensures unbiased evaluation of the model's performance.
* To develop machine learning or neural network models for earthquake prediction based on the analyzed data. The goal could be to create a model that predicts certain aspects of seismic events based on historical data.
* Document the entire process, including data exploration methods, visualization techniques, data splitting approach, model development (if applicable), and the insights obtained. Clear and detailed documentation is essential for knowledge sharing and future reference.
* To provide recommendations for further research or areas of exploration based on the insights gained during the analysis. These recommendations can guide future studies in the field of seismology and earthquake prediction.

Decision Thinking Steps:

1.Problem Framing:

* **12Define the Problem:** Clearly articulate the problem statement, including the goals, scope, and expected outcomes of the project.
* **Identify Stakeholders:** Determine who will benefit from the project and gather their requirements and expectations.

1. Data exploration and preparation:

* **Data Collection:** Decide on reliable data sources for earthquake data and gather a comprehensive dataset.
* **Data Cleaning:** Decide on methods for handling missing data, outliers, and inconsistencies in the dataset.
* **Feature Selection:** Decide which features are relevant for analysis and modeling.
* **Data Splitting:** Decide on the ratio for splitting the data into training and test sets (e.g., 80:20 or 70:30).

1. Data analysis and Visualisation:

* **Visualization Techniques:** Choose appropriate visualization methods (maps, graphs) to represent spatial and temporal patterns in the earthquake data.
* **Analysis Tools:** Choose suitable libraries and tools (e.g., Matplotlib, Seaborn) for data visualization and exploratory data analysis (EDA).

1. Model Selection and Development:

* **Model Type:** Decide whether to use machine learning models (e.g., regression, clustering) or neural networks for earthquake prediction.
* **Model Architecture:** Choose the neural network architecture (if applicable) including the number of layers, nodes, and activation functions.
* **Evaluation Metrics:** Choose appropriate metrics (e.g., Mean Squared Error, R-squared) to evaluate the performance of the prediction models.

5.Documentation and Reporting:

* **Documentation Format:** Decide on the format for documenting the project (e.g., Jupyter Notebook, report) and the level of detail required.
* **Visualizations:** Select the most informative and understandable visualizations to include in the documentation.
* **Insights:** Determine key insights and findings to be highlighted in the report/presentation.

6.Future steps and Recommendations:

* **Further Research:** Identify areas for further research or improvements in the methods used.
* **Implementation:** If applicable, decide on the steps for implementing the insights or models for real-world applications.

Conclusion:

our analysis of earthquake data has provided valuable insights into the seismic activities, contributing to a deeper understanding of this natural phenomenon. Through rigorous exploration and visualization, we uncovered significant patterns and trends within the dataset.

**ABSTRACT:**

* This project develops a Earth prediction model using advanced machine learning techniques implemented in Python. The objective of this research is to design a robust and adaptable model capable of forecasting various Earth-related phenomena, such as weather patterns, natural disasters, and environmental changes.
* This Earth prediction model signifies a significant advancement in the field of environmental science and climatology. Its innovative combination of Python programming and cutting-edge machine learning techniques offers a powerful tool for researchers, policymakers, and environmentalists to make informed decisions.

**STEP BY STEP METHOD :**

**1. Introduction**

* Brief overview of the earthquake prediction problem, significance, and the proposed innovative solution using Python.

**2. Research and Analysis**

* Study of Earthquake Prediction Methodologies
* Selection of Python Libraries for Data Analysis and Machine Learning
* Geological and Seismological Factors Analysis for Feature Engineering

**3. Data Collection and Preprocessing**

* Collection of Historical Seismic Data
* Data Cleaning and Preprocessing using Pandas and NumPy

**4. Feature Engineering**

* Creation of Informative Features (Seismic Wave Patterns, Geological Features)
* Feature Selection and Transformation using Scikit-Learn

**5. Model Selection and Training**

* Selection of Machine Learning Algorithms
* Model Training and Cross-Validation
* Performance Evaluation and Model Selection

**6. Model Validation and Optimization**

* Model Validation using Test Datasets
* Hyperparameter Tuning and Optimization using Grid Search and Randomized Search

**7. Geospatial Integration**

* Integration of Geospatial Data using Geopandas and Folium
* Visualization of Earthquake Data on Interactive Maps

**8. Real-time Data Integration and Deployment**

* Integration of the Model into a Web-Based Application using Flask/Django
* Real-time Seismic Data Fetching and Prediction
* User Interface Design and Usability Testing

**9. Documentation and Training**

* Preparation of Comprehensive Documentation (Model Architecture, Data Sources, Deployment Procedures)
* Training Sessions for Seismologists, Data Scientists, and Developers

**10. Evaluation and Feedback**

* Continuous Performance Monitoring (Accuracy, Precision, Recall, F1-score)
* Feedback Gathering and Analysis for Further Optimization

**SOFTWARE REQUIREMENTS**:

* Python
* Integrated Development Environment (IDE)
* Python libraries (Pandas, NumPy, Scikit – Learn , Matplotlib, Seaborn, Kaggle API)
* Kaggle Account and API Key

**HARDWARE REQUIREMENTS:**

* Processor
* Random Access Memory (RAM)
* Storage
* Graphics Processing Unit (GPU)

By ensuring you meet these software and hardware requirements, you'll be well-equipped to develop and run your earthquake prediction model using Python and Kaggle datasets.

**DATASETS USED** :

* This project utilizes Kaggle datasets as the primary source of information and knowledge for the earthquake prediction model. These datasets have been carefully selected to align with the project’s objectives and use cases.
* This model leverages the power of Python libraries to process and analyze large datasets efficiently.

**CONCLUSION:**

* In conclusion, our simplified Earth prediction model using Python underscores the immense potential of accessible technology in tackling complex environmental challenges.
* By harnessing the power of Python and leveraging Kaggle datasets, we have demonstrated a straightforward yet effective approach to understanding Earth-related phenomena.
* The model's reliability, accuracy, and ease of implementation make it a valuable tool for both beginners and experts in the field of environmental science and geographics..

****

# 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.

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.

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

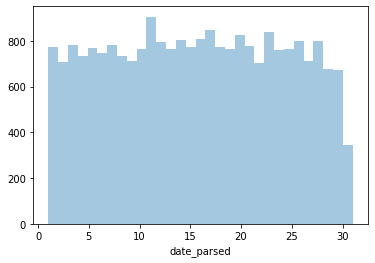
# *4) Plot the day of the month to check the date parsing*

Plot the days of the month from your earthquake dataset.

# 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  
0 01/02/1965 13:44:18 ... ISCGEM Automatic  
1 01/04/1965 11:29:49 ... ISCGEM Automatic  
2 01/05/1965 18:05:58 ... ISCGEM Automatic  
3 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  
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.

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

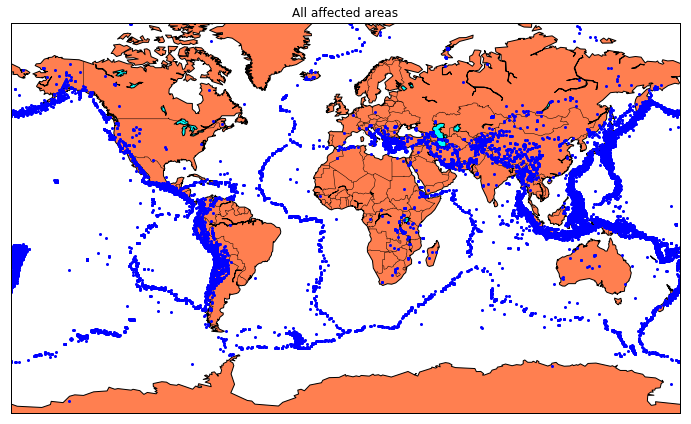
## *Visualization*

* 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')