# NAANMUDHALVAN-IBM SKILL

## ARTIFICIAL INTELLIGENCE

## GROUP PROJECT

## Project Title: Earthquake Prediction model using python.

### Phase II . Submission


| S.No | Group Members Name | **Naan Mudhalvan** ID | Email ID |
|---|---|---|---|
| 1 | S Abirami | au820321106001 | veeravelabirami@gmail.com |
| 2 | M.Kalaimathi | au820321106019 | muhilvanangu@gmail.com |
| 3 | S.Amirtha | au820321106006 | amirthasaravanakumar04@gmail.com |
| 4 | S.Aishwarya | au820321106003 | aishwaryasivakumar12112003@gmail.com |
| 5 | P.Bavadharani | au820321106010 | dharanibava89@gmail.com |
| 6. | B.Gnanasri | au820321106014 | gnanasribaskar23@gmail.com |


## DESIGN

We will take you through how to create a model for the task of Earthquake Prediction using Machine Learning and the Python programming language. Predicting earthquakes is one of the great unsolved problems in the earth sciences.

With the increase in the use of technology, many seismic monitoring stations have increased, so we can use machine learning and other data-driven methods to predict earthquakes.

With the increase in the use of technology, many seismic monitoring stations have increased, so we can use machine learning and other data-driven methods to predict earthquakes

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:

```
import numpy as np
import pandas as pd
```

```
import
matplotlib.pyplot as
plt
```

Now let's load and read the dataset,

```
data =
pd.read_csv("database.csv")

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:

```
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 |

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:

```
e
    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 |

| | Latitude | Longitude | Depth | Magnitude | Timestamp |
|---|---|---|---|---|---|
| **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,urcrnrl
llcrnrlon=-180,urcrnrlon=180,lat_ts=20,resolution='


longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projecti

#resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=
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()
```

```
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  we 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'])
```

We are going to define the hyperparameters with two or more options to find the best fit:

```
from keras.wrappers.scikit_learn import KerasClassi


model = KerasClassifier(build_fn=create_model, verb
```

```
# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_si
'linear', 'exponential']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadel
'Adamax', 'Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']


param_grid = dict(neurons=neurons, batch_size=batch
epochs=epochs, activation=activation, optimizer=opt
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))
```

```
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))
```

view rawearthquake.py hosted with ♡ by GitHub

**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))
```

```
ics=['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))
```

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

## CONCLUSION

we can see in the above output that our neural network model for earthquake prediction performs well. I hope you liked this article on how to create an earthquake prediction model with machine learning and the Python programming language. Feel free to ask your valuable questions in the comments section below.