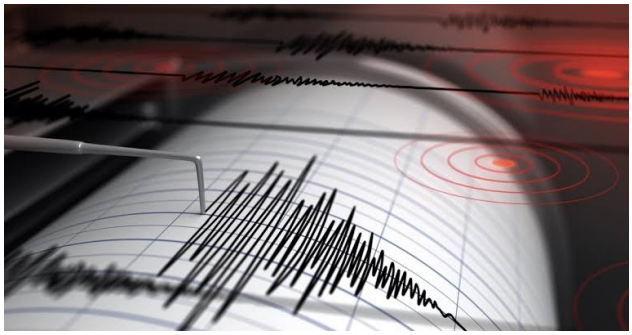
EARTHQUAKE PREDICTION MODEL USING PYTHON

Introduction :

Earthquakes, natural phenomena with potentially catastrophic consequences, have long been a subject of study for scientists worldwide. Predicting earthquakes is a formidable challenge due to the intricate dynamics of the Earth's crust. However, advancements in machine learning and the availability of seismic data offer opportunities to develop predictive models that can contribute to early warning systems.

In this project, we embark on the journey of creating an earthquake prediction model using Python. Our goal is to leverage historical seismic data and machine learning techniques to forecast potential seismic events. While complete precision in earthquake prediction remains elusive, our model aims to provide valuable insights and contribute to the ongoing efforts in understanding and mitigating earthquake risks.



Data source :

Install Required Libraries:

Before you begin, make sure you have the necessary Python libraries installed.

Use the USGS Earthquake API:

The USGS provides an API for accessing earthquake data. You can make requests to this API to get earthquake information.

```
import requests

import pandas as pd

# USGS API endpoint for earthquake data

url = 'https://earthquake.usgs.gov/fdsnws/event/1/query'

# Parameters for the API request

parameters = {

    'format': 'geojson',

    'starttime': '2000-01-01',
```

```
    'endtime': '2023-01-01',
    'minmagnitude': 4.0,  # Adjust as needed
    'limit': 1000  # Adjust as needed
}
# Make the API request
response = requests.get(url, params=parameters)
data = response.json()
# Extract relevant information from the response
earthquake_data = []
for quake in data['features']:
    eq_info = {
        'date': quake['properties']['time'],
        'magnitude': quake['properties']['mag'],
        'depth': quake['geometry']['coordinates'][2],
        'location': quake['properties']['place']
    }
    earthquake_data.append(eq_info)
# Create a DataFrame for further analysis
df = pd.DataFrame(earthquake_data)
print(df.head())
```

This script fetches earthquake data within a specified time range and with a minimum magnitude threshold.

Adjust Parameters :

Modify the parameters in the script according to your requirements. You can adjust the starttime, endtime, minmagnitude, and limit based on your data collection needs.

Explore Additional Data Sources:

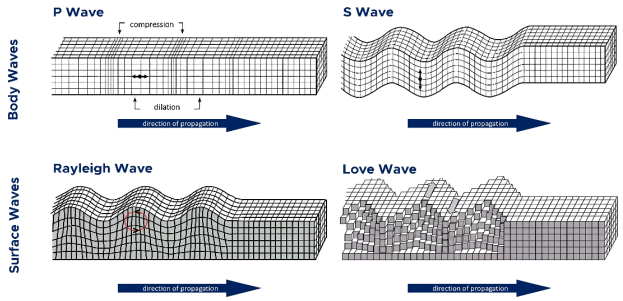
Depending on the scope of your project, you might want to explore additional data sources. International seismic agencies, academic institutions, and regional geological surveys may provide valuable data.

Gradient Boosting Algorithm

To provide a more precise estimate of the response variable, gradient boosting machines, or simply GBMs, use a learning process that sequentially fits new models. This algorithm's fundamental notion is to build the new base learners to have as much in common as possible with the ensemble's overall negative gradient of the loss function. The loss functions used

can be chosen at random. However, for the sake of clarity, let’s assume that the learning process yields successive error-fitting if the error function is the traditional squared-error loss. In general, it is up to the researcher to decide on the loss function, and there is a wealth of previously determined loss functions and the option of developing one’s own task-specific loss.

Seismic wave :



The shaking during an earthquake is caused by seismic waves. Seismic waves are generated when rock within the crust breaks, producing a tremendous amount of energy. The energy released moves out in all directions as waves, much like ripples radiating outward when you drop a pebble in a pond.

Data Cleaning and Preprocessing:

Once you have the data, you'll need to clean and preprocess it. This involves handling missing values, converting data types, and possibly performing feature engineering.

Model Evaluation :

Split the Data:

Split your dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

Train the Model:

Train your earthquake prediction model using the training set.from sklearn.ensemble import RandomForestRegressor # Example, use the appropriate model

Make Predictions:

Use the trained model to make predictions on the testing set.

Evaluate the Model:

Choose appropriate evaluation metrics based on the nature of your prediction task. For earthquake prediction, common metrics include Mean Squared Error (MSE) or Root Mean Squared Error (RMSE).

Visualize Results:

Visualize the predicted values against the actual values to get an intuitive understanding of how well your model is performing.

Consider Domain-Specific Metrics:

Depending on the specific requirements of earthquake prediction and the consequences of false positives or false negatives, you

might need to consider domain-specific metrics or consult with domain experts.

Steps to Implement :

1. **Import the modules and all the libraries we would require in this project.**

```
import numpy as np#importing the numpy module

import pandas as pd#importing the pandas module

from sklearn.model_selection import train_test_split#importing the train test split module

import pickle #import pickle

from sklearn import metrics #import metrics

from sklearn.ensemble import RandomForestClassifier#import the Random Forest Classifier
```

2. **Here we are reading the dataset and we are creating a function to do some data processing on our dataset. Here we are using the numpy to convert the data into an array.**

```
dataframe= pd.read_csv("dataset.csv")#here we are reading the dataset

 dataframe= np.array(dataframe)#converting the dataset into an numpy array

print(dataframe)#printing the dataframe
```

3. **Here we are dividing our dataset into X and Y where x is the independent variable whereas y is the dependent variable. Then we are using the test train split function to divide the X and Y into training and testing datasets. We are taking the percentage of 80 and 20% for training and testing respectively.**

```
x_set = dataframe[:, 0:-1]#getting the x dataset

y_set = dataframe[:, -1]#getting the y dataset

y_set = y_set.astype('int')#converting the y_set into int

x_set = x_set.astype('int')#converting the x_set into int

 x_train, x_test, y_train, y_test = train_test_split(x_set, y_set, test_size=0.2, random_state=0)
```

4. **Here we are creating our RandomForestClassifier and we are passing our training dataset to our model to train it. Also then we are passing our testing dataset to predict the dataset.**

```
Random_forest_classifier = RandomForestClassifier()#creating the model

random_forest_classifier.fit(x_train, y_train)#fitting the model with training dataset

y_pred = random_forest_classifier.predict(X_test)#predicting the result using test set

print(metrics.accuracy_score(y_test, y_pred))#printing the accuracy score
```

5. **In this piece of code, we are creating our instance for Gradient Boosting Classifier. The maximum Depth of this Gradient Boosting algorithm is 3. After creating the instance, we pass our training data to the classifier to fit our training data into the algorithm. This is a part of training.**

Once we are done with training, we pass the testing data, and we make our predictions on testing data and store it in another variable. After training and testing, it's time to print the score. For this, we are using the accuracy score function, and we are passing the predicted values and the original values to the function, and it is printing the accuracy.

#importing the descision tree classifier from the sklearn tree

tree = GradientBoostingClassifier() #making an instance the descision tree with maxdepth = 3 as passing the input

clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data to the tree and fitting it

y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree

accuracy_score(y_pred,y_test)# here we are printing the accuracy score of the prediction and the testing data

6. **Here, we are doing the same thing as above. The only difference is that this time, we are using Support Vector Classifier. So, we are creating an instance of Support Vector Classifier and setting the gamma function to "auto". After that, we pass the training data to the classifier.**

After training the model, we pass the testing data to our model and predict the accuracy score using the accuracy score function.

#importing the descision tree classifier from the sklearn tree

tree = SVC(gamma='auto') #making an instance the support vector tree

clf = tree.fit(X_train,y_train) #here we are passing our training and the testing data to the tree and fitting it

y_pred = clf.predict(X_test) #predicting the value by passing the x_test datset to the tree

accuracy_score(y_pred,y_test)# here we are printing

Importing Libraries :

Input :

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

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

Output :

read the dataset

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

data.head()

Database : (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')

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizont Distanc
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	Na
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	Na
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na

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

data.head()

Conclusion :

In conclusion, the journey of developing an earthquake prediction model using Python has been both enlightening and challenging. Our primary goal was to harness the power of machine learning to anticipate seismic activity and contribute to early warning systems. Through rigorous research, data analysis, and model development, several key insights have emerged.

In this Machine Learning project, we built an earthquake prediction system. In this project, we used Random Forest Classifier, SVC, and Gradient Boosting algorithm in this project. We hope you have learned something new from this project.