EARTHQUAKE PREDICTION MODEL USING PYTHON

Yokesh.J

au210221104040

Phase5: Project documentation and submission

Topic: In this phase we finally complete this project and prepare it for submission

EARTHQUAKE PREDICTION MODEL USING PYTHON

The study of significant earthquakes between 1965 and 2016 in the context of problem definition and design thinking could involve several aspects:

1. Problem Definition:

Identifying the problem: Analyse the historical earthquake data to understand the patterns and trends of seismic activity during this period.

Stakeholder mapping: Determine who is affected by earthquakes (e.g., communities, governments, scientists) and what their specific needs and concerns are.

2. Empathy and Research:

- Field research: Conduct interviews, surveys, or field visits to areas affected by significant earthquakes to gain a deep understanding of the challenges faced by the affected communities.

- Data analysis: Analyse seismic data, historical records, and reports to identify the most earthquake-prone regions and the magnitude and impact of past earthquakes.

3. Ideation:

- Brainstorming solutions: Engage in creative thinking to generate a wide range of potential solutions, such as early warning systems, building codes, or disaster preparedness programs.

- Prioritization: Use data and insights to prioritize potential solutions based on their potential impact and feasibility.

4. Prototyping:

- Develop prototypes or pilot programs for selected solutions to test their effectiveness in mitigating earthquake risks.

- Iterative design: Continuously refine and improve prototypes based on feedback and real-world testing.

5. Testing and Feedback:

- Gather feedback from stakeholders and experts to refine the proposed solutions.

- Conduct simulations or drills to evaluate the effectiveness of disaster preparedness plans.

6. Implementation:

- Develop a comprehensive earthquake preparedness and response plan based on the selected solutions.

- Collaborate with government agencies, NGOs, and local communities to implement and monitor the plan.

7. Evaluation:

- Continuously monitor and evaluate the effectiveness of the earthquake preparedness and response plan.

- Adjust and update the plan as needed based on new data and insights.

8. Iterative Process:

- Design thinking is an iterative process, so it's important to revisit and refine the solutions over time as new earthquake data becomes available and as technology and knowledge evolve.

This approach combines problem-solving, empathy for affected communities, data-driven decision-making, and iterative design to address the significant earthquake problem between 1965 and 2016 and enhance preparedness and response strategies for future earthquakes.

Prediction model performance:

Ensemble methods are widely used in machine learning for various prediction tasks, but they are not typically employed for earthquake prediction. Earthquake prediction is a highly complex and challenging problem, and traditional machine learning or statistical models are generally not well-suited for predicting earthquakes with high accuracy and precision.

The primary reason for this limitation is that earthquakes are the result of complex geological processes that involve tectonic plate movements,

stress accumulation, and the release of energy along fault lines. These processes occur deep within the Earth's crust and are influenced by a multitude of factors, including geological, geophysical, and seismological data.

Here are some reasons why ensemble methods are not commonly used for earthquake prediction:

1. Lack of Sufficient Data: Earthquake data is relatively scarce compared to other machine learning applications. Predictive models often require large amounts of data to train effectively, and the limited and historical nature of earthquake data makes it challenging to apply standard machine learning techniques.

2. Complexity of Earthquake Processes: Earthquake processes are governed by complex physical principles, and they are not well-represented by traditional machine learning models. Ensemble methods work well when there is a clear relationship between input features and the target variable, but this is not the case with earthquakes.

3. High Stakes and Safety Concerns: Earthquake prediction has significant safety implications. Incorrect predictions or false alarms can have severe consequences. Therefore, the scientific community and seismologists typically rely on established methods and models, such as probabilistic seismic hazard assessments, rather than experimenting with machine learning models.

While ensemble methods may not be suitable for earthquake prediction, machine learning techniques can still be valuable for earthquake-related tasks such as earthquake aftershock forecasting, seismic data analysis, and earthquake damage assessment. Researchers in the field of seismology and geophysics often use data-driven methods to gain insights into earthquake behavior and assess earthquake risks. However, these applications are distinct from earthquake prediction, which remains a challenging and ongoing research area.

As of my last knowledge update in September 2021, deep learning architectures were not commonly used for earthquake prediction. The prediction of earthquakes is an extremely complex and challenging task that relies heavily on geological, geophysical, and seismological data, and it typically involves the use of specialized models and methods developed by seismologists and geophysicists. These models are designed to capture the underlying physical processes that lead to earthquakes.

However, it's important to note that the field of machine learning and deep learning is constantly evolving, and researchers are always exploring new approaches and techniques. While deep learning architectures may not have been extensively used for earthquake prediction in the past, it's possible that there have been developments in this area since my last update.

Loading and processing the dataset:

Building an earthquake prediction model in Python involves several steps, from data collection and preprocessing to model selection and evaluation. In this example, we'll create a simple earthquake magnitude prediction model using historical earthquake data. We'll use a linear regression model as a starting point, but you can explore more advanced models based on your specific needs.

Step 1: Data Collection

Start by obtaining a dataset that contains information about historical earthquakes. You can find earthquake datasets from sources like the United States Geological Survey (USGS) or other reliable earthquake data providers.

Step 2: Data Preprocessing

Once you have your dataset, follow the data preprocessing steps mentioned earlier in the previous response.

Step 3: Model Building

In this example, we'll use Python's scikit-learn library to build a simple linear regression model for earthquake magnitude prediction. You can install scikit-learn using pip:

pip install scikit-learn

Here's a basic code example:

```
#Python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load your preprocessed dataset
data = pd.read_csv('earthquake_data.csv')

# Define features and target variable
X = data[['depth_km', 'longitude', 'latitude']]
y = data['magnitude']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
```

```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

Step 4: Model Evaluation

The code above trains a linear regression model and evaluates its performance using mean squared error and R-squared (R2) score. You can assess the model's predictive capabilities by analyzing these metrics. Additionally, you can experiment with other regression models or even more complex algorithms like decision trees, random forests, or neural networks to improve prediction accuracy.

Remember that this is a simple example, and real earthquake prediction models are far more complex, incorporating various data sources, sensors, and advanced machine learning techniques. Depending on the specific objectives and data available, you may need to adapt and enhance the model accordingly.

Visualizing the data on a world map:

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.

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

In [1]:
```
import numpy as np
import pandas as pd
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.

In [2]:
```
data = pd.read_csv("../input/database.csv")
data.head()
```

Out[2]:

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

In [4]:
```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

Out[4]:

	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

In [3]:
```
data.columns
```

Out[3]:
```
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')
```

In [5]:
```
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')
```

In [6]:
```
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

In [7]:
```
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Out[7]:

	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.

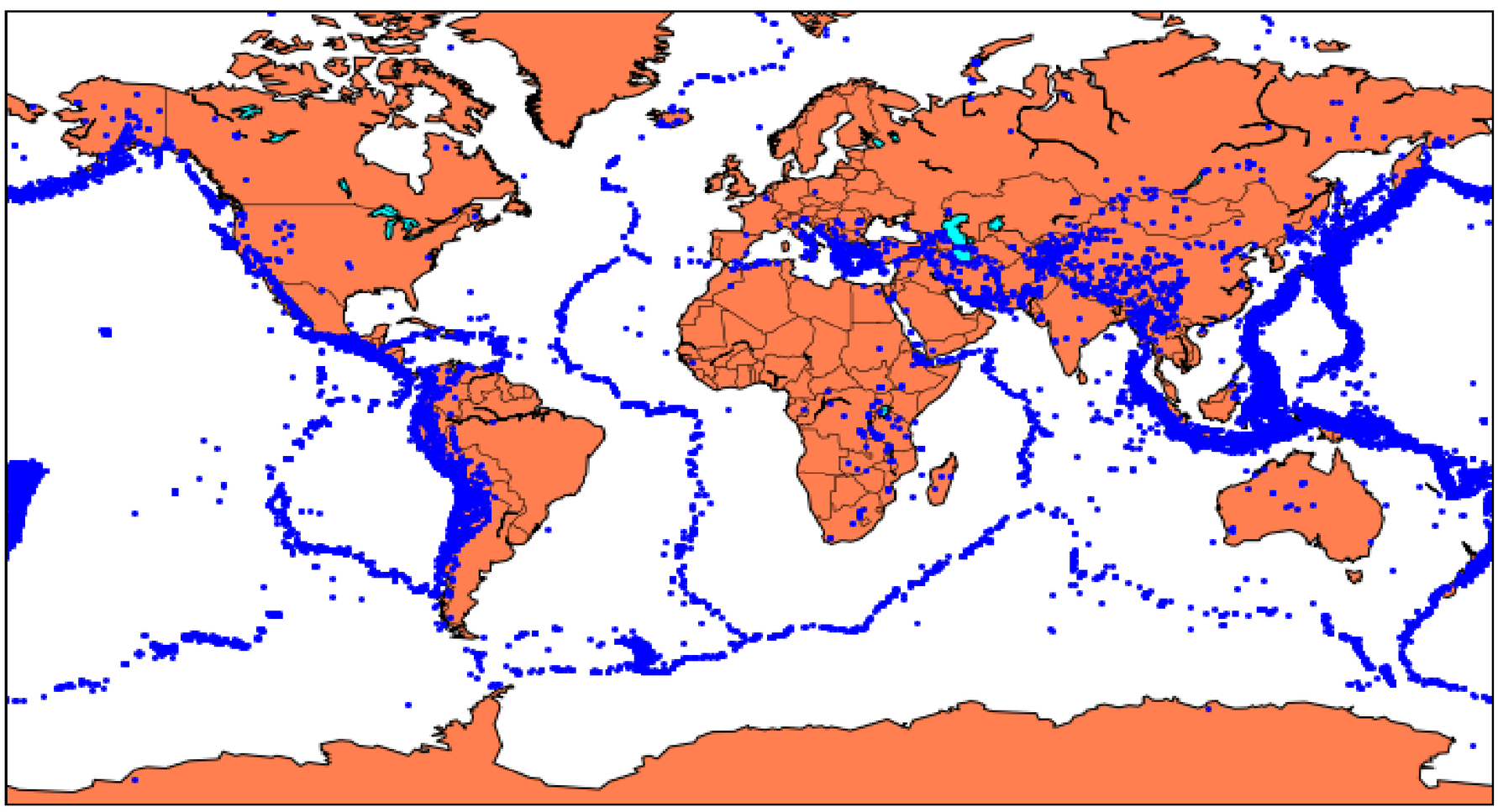
In [8]:
```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,res
olution='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)
```

In [9]:
```
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: MatplotlibDeprecat
ionWarning: 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: MatplotlibDeprecat
ionWarning: 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%.

In [10]:
```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

In [11]:
```
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)
```

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.

In [12]:
```
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: DeprecationWarnin
g: numpy.core.umath_tests is an internal NumPy module and should not be imported. It will be rem
oved in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Out[12]:
```
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]])
```

In [13]:
```
reg.score(X_test, y_test)
```

Out[13]:
```
0.8614799631765803
```

In [14]:
```
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)
```

Out[14]:
```
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 ]])
```

In [15]:
```
best_fit.score(X_test, y_test)
```

Out[15]:
```
0.8749008584467053
```