EARTHQUAKE PREDICTION MODEL USING PYTHON

Team Member

Balaji E

422521104006

Phase-2 Documentation Submission

Introduction:

- Earthquakes are natural disasters that can cause significant damage and loss of life, making their accurate prediction a matter of utmost importance for public safety and disaster preparedness.
- This project presents the development of an earthquake prediction model using Python.
- The primary objective is to leverage machine learning techniques to predict earthquake magnitudes based on a dataset obtained from Kaggle.

Content for project phase-2:

- Consider advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's performance.

Data Source:

- The data is provided by the United States Geological Survey (USGS), which monitors and reports on earthquake activity worldwide.
- The dataset contains information about earthquake events worldwide, including details such as location, time, magnitude, depth, and more. It is a comprehensive collection of earthquake-related data.

Dataset Link: https://www.kaggle.com/datasets/usgs/earthquake-database

```
[6]: df=pd.read_csv("database.csv")
     df.head()
```

[6]:

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

5 rows × 21 columns

Data Preprocessing:

- Check for data range in longitude and latitude: Check if the values fall within a reasonable range for the column they are in. For example, latitude values should be between -90 and 90, and longitude values should be between -180 and 180.
- The fit_transform method computes the necessary scaling parameters (minimum and maximum values) from the data and then scales the data accordingly.

- After applying this line of code, the 'Latitude' and 'Longitude' columns in the data DataFrame will be scaled between 0 and 1, based on the minimum and maximum values in those columns.
- Drop the columns with a large number of missing values (dmin, magError, and magNst), as they may not contribute significantly to your analysis.

Program:

scaler = MinMaxScaler()

data[['Latitude', 'Longitude','Depth']] = scaler.fit_transform(data[['Latitude', 'Longitude','Depth']])

X = data[['Latitude', 'Longitude','Depth']].values

y = data['Magnitude'].values

Innovative Techniques:

- we are using 4 type of deep learning model to predict the earthquake magnitude , at last using Ensemble method connect all model and produce better result.

- **Deep Learning Techniques:**

  - Deep learning is a subfield of machine learning that involves training artificial neural networks to perform specific tasks. Deep learning techniques are a set of methods and algorithms that utilize these deep neural networks, which are composed of multiple layers of interconnected nodes (or neurons).
  - The types of Deep Learning techniques we use in this project

    - Feed Forward Neural Network
    - Backpropagation
    - Recurrent Neural Network
    - Long Short Term Memory

Import Libraries:

- Start by importing the necessary libraries, such as TensorFlow or PyTorch for neural network building, and other relevant libraries for data manipulation and visualization.

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.neural_network import MLPRegressor
from tensorflow.keras.layers import SimpleRNN,
Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import VotingRegressor
```

Load Preprocessed Data:

- Load the preprocessed earthquake data, including features and labels, that you've prepared during the data preprocessing step.

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

Data Splitting:

- Split your data into training, validation, and test sets if you haven't already done so during preprocessing.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model-1:

Feed Forward Neural Networks:

- Feed forward neural networks (FNN) have been applied to various fields, including earthquake prediction, to leverage their capability to model complex relationships in data.
- However, it's important to note that alone are not typically used as the primary method for earthquake prediction, as earthquake prediction remains a challenging and complex problem.
- Instead, FNNs can be a part of a broader predictive modeling approach.

Model Creation:

- **Build the FFNN Model:** Define the architecture of your Feedforward Neural Network. You can adjust the number of layers, neurons per layer, activation functions, and other hyperparameters based on your specific problem and dataset**.**

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(2,)),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for magnitude prediction
(linear activation)
])
```

- **Compile the Model:** Specify the loss function, optimizer, and evaluation metrics for your model. The choice of these components depends on whether you're dealing with regression or classification.

```
model.compile(optimizer='adam',
loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

- **Result:** Train the model and predict the result using the test dataset to calculate the result we use Mean Square Error.

```
Epoch 1/10
586/586 [==============================] - 2s 3ms/step - loss: 2.6945
Epoch 2/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1794
Epoch 3/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1794
Epoch 4/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1796
Epoch 5/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1799
Epoch 6/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1798
Epoch 7/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1813
Epoch 8/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1795
Epoch 9/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1800
Epoch 10/10
586/586 [==============================] - 1s 2ms/step - loss: 0.1813
147/147 [==============================] - 0s 2ms/step
Mean Squared Error: 0.18492417428152197
```

Result of Feed Forward Neural Network : 0.18492417428152197

Model-2:

Backpropagation:

- Backpropagation (short for "backward propagation of errors") is a fundamental training algorithm for artificial neural networks (ANNs), a type of machine learning model.
- It is used to adjust the model's weights and biases during the training process, so that the network can learn to make more accurate predictions.
- Backpropagation is a supervised learning technique, which means it requires a labeled dataset for training.

Model Creation:

- **Hidden Layers:** We've designed our model with two hidden layers, containing 64 and 32 neurons, respectively. This architecture enables the network to capture complex patterns in the data.

- **Activation Function:** We employ the Rectified Linear Unit (ReLU) activation function. ReLU introduces non-linearity, allowing the network to learn and model intricate relationships within the data.

- **Maximum Iterations:** To ensure convergence, we've set a maximum of 1000 iterations for training, preventing the model from training indefinitely.

```
model = MLPRegressor(hidden_layer_sizes=(64, 32),
activation='relu', max_iter=1000, random_state=42)
model.fit(X_train, y_train)
```

- **Result:** Train the model and predict the result using the test dataset to calculate the result we use Mean Square Error.

Result of Backpropagation : 0.18479814758971097

Model-3:

Recurrent Neural Network :

- A Recurrent Neural Network (RNN) is a type of neural network designed for processing sequences of data.
- Unlike traditional feedforward neural networks, RNNs have connections that loop back on themselves, allowing them to maintain a "memory" of previous inputs.
- This makes them well-suited for tasks involving sequential data, where the order of elements matters, such as time series data, text, speech, and more.

Model Creation:

- **Neural Network Architecture:** We construct a neural network using Recurrent Neural Networks (RNNs) for their ability to handle sequential data, which is vital for time-series prediction like earthquake magnitudes.

- **Simple RNN Layer:** The model includes a SimpleRNN layer with 50 units, employing the ReLU activation function to capture temporal dependencies and patterns in seismic data.

- **Dense Output Layer:** We incorporate a Dense output layer with one unit for magnitude prediction, designed for regression tasks.

```
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu',
input_shape=(sequence_length, 2)))
model.add(Dense(units=1))
```

- **Optimization:** The model is optimized using the Adam optimizer, a popular choice for training neural networks.

```
model.compile(optimizer='adam',
loss='mean_squared_error')
```

- **Result:** Train the model and predict the result using the test dataset to calculate the result we use Mean Square Error.

```
Epoch 1/10
586/586 [==============================] - 3s 3ms/step - loss: 1.9280
Epoch 2/10
586/586 [==============================] - 2s 3ms/step - loss: 0.2071
Epoch 3/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1972
Epoch 4/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1945
Epoch 5/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1985
Epoch 6/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1954
Epoch 7/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1964
Epoch 8/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1937
Epoch 9/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1985
Epoch 10/10
586/586 [==============================] - 2s 3ms/step - loss: 0.1927
147/147 [==============================] - 0s 2ms/step
Mean Squared Error: 0.2638302490778608
```

Result of Recurrent Neural Network : 0.2638302490778608

Model-4:

Long Short-Term Memory Neural Networks :

- LSTM networks are a type of recurrent neural network (RNN) capable of modeling temporal dependencies in sequential data.
- LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period of time.
- The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate.

Model Creation:

- **Input shape :** represents no of time step in input sequence. I mention a time steps to the model .

- **Dropout Layer:** we are adding a dropout layer to the model with a dropout rate of 0.2. at approximately 20% of neuron in the previous layer will be randomly "dropped out " or deactivated.

- **Fully-Connected Layer:** The output layer has a single neuron. in regression tasks where you want to predict a single continuous value, such as earthquake magnitude.

- **Activation :** 'Linear' function means that the output of the neuron is directly proportional to the weighted sum of its inputs.

```
model = Sequential()
model.add(LSTM(64, input_shape=(X_train.shape[1], 1),
return_sequences=True))
model.add(LSTM(64))
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.2))
```

model.add(Dense(1, activation='linear')) # Output layer for predicting magnitude

- **Optimizer:** Adam is chosen for efficient weight updates during training.

- **Loss Function:** Mean Squared Error, minimize the squared difference between predicted and actual values.

model.compile(optimizer='adam', loss='mean_squared_error')

- **Result:** Train the model and predict the result using the test dataset to calculate the result we use Mean Square Error.

```
Epoch 1/10
586/586 [==============================] - 7s 6ms/step - loss: 3.7721 - val_loss: 0.1969
Epoch 2/10
586/586 [==============================] - 3s 5ms/step - loss: 0.4436 - val_loss: 0.1855
Epoch 3/10
586/586 [==============================] - 3s 5ms/step - loss: 0.4373 - val_loss: 0.1858
Epoch 4/10
586/586 [==============================] - 3s 5ms/step - loss: 0.4317 - val_loss: 0.1846
Epoch 5/10
586/586 [==============================] - 3s 6ms/step - loss: 0.4279 - val_loss: 0.1877
Epoch 6/10
586/586 [==============================] - 3s 5ms/step - loss: 0.4178 - val_loss: 0.1912
Epoch 7/10
586/586 [==============================] - 3s 5ms/step - loss: 0.4089 - val_loss: 0.1849
Epoch 8/10
586/586 [==============================] - 3s 6ms/step - loss: 0.4022 - val_loss: 0.1864
Epoch 9/10
586/586 [==============================] - 3s 6ms/step - loss: 0.3895 - val_loss: 0.1901
Epoch 10/10
586/586 [==============================] - 3s 5ms/step - loss: 0.3762 - val_loss: 0.1891
147/147 [==============================] - 0s 3ms/step - loss: 0.1891
Test Loss: 0.18909773230552673
147/147 [==============================] - 1s 3ms/step
```

Result of Long Short-Term Memory Neural Networks: 0.18909773230552673

- **Ensemble Model:**
  - Ensemble Model combines the predictions of multiple machine learning models, often of different types, to improve predictive accuracy. The diversity in models helps capture different aspects of the data, reducing overfitting and increasing generalization.

Model Creation :

- We create an ensemble using the VotingRegressor from scikit-learn. This ensemble combines the predictions of the four individual models using a voting mechanism
- The ensemble combines the predictions from different neural network architectures (FNN, RNN, Backpropagation and LSTM) using a majority vote, which can lead to improved accuracy and robustness in earthquake magnitude prediction. You can further explore hyperparameter tuning, different voting strategies, and combining a wider variety of models to enhance ensemble performance.

```
ensemble = VotingRegressor(estimators=[
    ('FNN', FNN_model),
    ('RNN', RNN_model),
    ('LSTM', LSTM_model),
    ('BackPropagation',backpropagation_model)
])
```

Improving Result:

- **Hyperparameter Tuning:**
  - Optimize hyperparameters such as the number of model layers, units per layer, learning rate, batch size, and dropout rate.
  - Consider using techniques like grid search or random search to find the best combination of hyperparameters for all type of model.
- **Feature Engineering:**
  - Continue to explore and engineer features that may capture important patterns or relationships in the data.
  - Consult with domain experts to identify potential features that could enhance prediction accuracy.
- **Data Augmentation:**
  - Generate additional training data through techniques like data augmentation to increase the model's ability to generalize.
- **Advanced Architectures:**
  - Experiment with more complex neural network architectures, like CNN or GRU, which may capture longer-term dependencies in seismic data.

Conclusion:

- In conclusion, ensemble methods that combine different neural network architectures like Feedforward Neural Networks (FNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) can significantly enhance the accuracy and robustness of earthquake magnitude prediction. This approach leverages the strengths of diverse models to provide more reliable forecasts.

- In summary, the ensemble of FNN, RNN, and LSTM models for earthquake magnitude prediction offers a promising approach to address the complexities of seismic data. It exemplifies the idea that a combination of models is often more robust and reliable than any single model. By continually refining and optimizing ensemble techniques, we can improve our ability to predict earthquake magnitudes, ultimately enhancing public safety and disaster preparedness.