EARTHQUAKE PREDICTION MODEL USING PYTHON

Phase-5 Documentation Submission

Team Member:

Balaji E

422521104006

Introduction:

- 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.
- A neural network model is designed and trained using TensorFlow or Keras to make predictions on earthquake magnitudes.
- 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.

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 worldwide earthquake events, including 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

Innovation

Reason for choosing this Model:

Diversity of Models: The ensemble combines multiple models that has different strengths and weaknesses, and by combining them, you can leverage their diversity to potentially capture a broader range of patterns in the data.

Complementary Strengths: Each of the base models in ensemble has its own unique approach and may excel in different aspects of the data. By combining these strengths, we can create a more well-rounded model.

Ensemble Tuning: We have the flexibility to fine-tune the weights of the models in our ensemble, which allows us to emphasize the models that perform best on your specific dataset.

Innovative Techniques :

Dynamic Model Selection: Instead of using fixed weights for the base models, implement dynamic model selection. During inference, you can use meta-learners or online learning techniques to adaptively assign weights to the base models based on their performance on the current data point or time window.

Feature Engineering: Explore feature engineering techniques to create new features that may be more informative for the ensemble. Feature engineering can help improve the base models' performance and, consequently, the ensemble's performance.

Ensemble Variance Reduction: Implement techniques to reduce the variance in the ensemble predictions. Techniques like Bagging or Bayesian model averaging can help reduce the uncertainty in predictions.

Hyperparameter Tuning:

Hyperparameter tuning is a crucial step to optimize the performance of the ensemble model. In this code, we can apply hyperparameter tuning using techniques such as grid search or random search to find the best hyperparameters for the individual base models and we can perform ensemble.

Data Preprocessing

- Data preprocessing is a crucial step in machine learning. It involves cleaning, transforming, and organizing raw data into a format that is suitable for the training model.

- The primary goals of data preprocessing are to improve data quality, reduce NaN, and make the data more amenable to training a model.

Steps to Preprocessing Earthquake Dataset:

1. Check the Longitude and Latitude degree
2. Select the relevant column from the dataset
3. Convert the date and time into the same format
4. Convert date and time into TimeStamp
5. Remove NaN row or data from a dataset
6. Check for magnitude max and min (2.5 to 9.1)

1. Checking for maximum and minimum of Latitude and Longitude:

- Always the Latitude and Longitude lies between constant range.
- Latitude should be between -90 to 90 degrees.
- Longitude should be lies between -180 to 180 degrees.

```
import pandas as pd

# Check latitude values between -90 to 90
df[df['Latitude'].between(-90, 90)]

# Check longitude values between -180 to 180
df[df['Longitude'].between(-180, 180)]

df.tail()
```

The .between(-90,90) method is used to check whether the value of latitude is falls between range or not. It returns a boolean result as ‘True’ or ‘False’.

It filters the dataset, retaining only the rows for the condition ‘True’. In other words, it keeps only the rows with valid latitude and longitude values

2. Select the relevant column from the dataset:

- Remove the non-relevant column from the dataset
- Because it does not help the model to learn but instead makes the learning as complex.

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

[31]:

	Date	Time	Longitude	Latitude	Depth	Magnitude
0	01/02/1965	13:44:18	145.6160	19.2460	131.60	6.0
1	01/04/1965	11:29:49	127.3520	1.8630	80.00	5.8
2	01/05/1965	18:05:58	-173.9720	-20.5790	20.00	6.2
3	01/08/1965	18:49:43	-23.5570	-59.0760	15.00	5.8
4	01/09/1965	13:32:50	126.4270	11.9380	15.00	5.8
...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	-118.8941	38.3917	12.30	5.6
23408	12/28/2016	09:13:47	-118.8957	38.3777	8.80	5.5
23409	12/28/2016	12:38:51	140.4262	36.9179	10.00	5.9
23410	12/29/2016	22:30:19	118.6639	-9.0283	79.00	6.3
23411	12/30/2016	20:08:28	141.4103	37.3973	11.94	5.5

23412 rows × 6 columns

3. Convert date and time into same format:

Dataset contains date different format like dd-mm-yyyy, dd/mm/yyyy.

So, convert date and time into same format using datetime library.

```
import pandas as pd

df = pd.read_csv('database.csv')

def formatconvert(date_str):
        return pd.to_datetime(date_str).strftime('%d/%m/%Y')

df['Date'] = df['Date'].apply(formatconvert)


df['Date']
```

```
0        02/01/1965
1        04/01/1965
2        05/01/1965
3        08/01/1965
4        09/01/1965
            ...
23406    28/12/2016
23407    28/12/2016
23408    28/12/2016
23409    29/12/2016
23410    30/12/2016
Name: Date, Length: 23411, dtype: object
```

4.Remove nan row or data from dataset:

- Check for Not a Number in the given dataset. If any nan present in dataset, remove the row from the dataset.
- To remove rows or data with NaN values from a dataset in python
- We use dropna() from pandas. it returns a Boolean result

[60]:
```
# Count the number of missing (NaN) values in each column
nan_counts = data.isna().sum()

# Print the results
print(nan_counts)
```

```
Latitude     0
Longitude    0
Depth        0
Magnitude    0
Timestamp    0
dtype: int64
```

5. Convert date and time into TimeStamp:

Timestamps are numerical values representing a specific point in time, usually in seconds or milliseconds. It provide a consistent and standardized way to represent time across different systems and programming language.

The timestamps represent the number of seconds since the Unix epoch from January 1, 1970, to till date so, drop the before it.

```
[39]: df=pd.DataFrame(df)

      df = df.drop(df.index[0:1457])
      df.reset_index(drop=True, inplace=True)
      df
```

[39]:

	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/04/1970	17:00:41	24.1850	102.5430	11.30	7.1
1	01/05/1970	11:49:10	23.9840	102.7320	15.00	5.9
2	01/06/1970	5:35:54	-9.5830	151.4930	15.00	6.3
3	01/07/1970	7:56:14	15.7850	-59.8080	36.70	6.0
4	01/08/1970	17:12:41	-34.8500	178.7820	199.40	6.8
...	...	...	...	...	...	...
21949	12/28/2016	8:22:12	38.3917	-118.8941	12.30	5.6
21950	12/28/2016	9:13:47	38.3777	-118.8957	8.80	5.5
21951	12/28/2016	12:38:51	36.9179	140.4262	10.00	5.9

- We can create a new column by combining the ‘Date’ and ‘Time’ Column into single timestamp using pd.to_datetime.strptime()
- Using mktime() we can convert date and time into timestamp.
- Finally remove the date and time column from the dataset.

[56]:
```
import datetime
import time

timestamp = []
for d, t in zip(df['Date'], df['Time']):
    ts = datetime.datetime.strptime(d+' '+t, '%d/%m/%Y %H:%M:%S')
    timestamp.append(time.mktime(ts.timetuple()))

timeStamp = pd.Series(timestamp)
```

[57]:
```
df['Timestamp'] = timeStamp.values
data = df.drop(['Date', 'Time'], axis=1)
data = data[data.Timestamp != 'ValueError']
data
```

[57]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	24.1850	102.5430	11.30	7.1	3.006410e+05
1	23.9840	102.7320	15.00	5.9	3.683500e+05
2	-9.5830	151.4930	15.00	6.3	4.323540e+05
3	15.7850	-59.8080	36.70	6.0	5.271740e+05
4	-34.8500	178.7820	199.40	6.8	6.469610e+05
...	...	...	...	...	...
21949	38.3917	-118.8941	12.30	5.6	1.482894e+09
21950	38.3777	-118.8957	8.80	5.5	1.482897e+09
21951	36.9179	140.4262	10.00	5.9	1.482909e+09

6.Check for magnitude max and min (2.5 to 9.1):

Find Maximun and Minimun value for magnitude because the magnitude value cannot be negative or more than 10.

Check for magnitude value within 2.5 to 9.1, if anything exceeds delete the row or data from the dataset.

```
[68]: data['Magnitude'].max()
```

```
[68]: 9.1
```

```
[67]: data['Magnitude'].min()
```

```
[67]: 5.5
```

```
[69]: data[data['Magnitude'].between(5.5, 9.1)]
```

[69]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	24.1850	102.5430	11.30	7.1	3.006410e+05
1	23.9840	102.7320	15.00	5.9	3.683500e+05
2	-9.5830	151.4930	15.00	6.3	4.323540e+05
3	15.7850	-59.8080	36.70	6.0	5.271740e+05
4	-34.8500	178.7820	199.40	6.8	6.469610e+05
...	...	...	...	...	...
21949	38.3917	-118.8941	12.30	5.6	1.482894e+09
21950	38.3777	-118.8957	8.80	5.5	1.482897e+09
21951	36.9179	140.4262	10.00	5.9	1.482909e+09
21952	-9.0283	118.6639	79.00	6.3	1.483031e+09
21953	37.3973	141.4103	11.94	5.5	1.483109e+09

21954 rows × 5 columns

7.Display the data after preprocessing:

Dataset after preprocessing is done. Save the file in csv format using to_csv(). The preprocessed dataset is use for train the model.

[72]:
```
data.to_csv("resultdata.csv",index='False')
data
```

[72]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	24.1850	102.5430	11.30	7.1	3.006410e+05
1	23.9840	102.7320	15.00	5.9	3.683500e+05
2	-9.5830	151.4930	15.00	6.3	4.323540e+05
3	15.7850	-59.8080	36.70	6.0	5.271740e+05
4	-34.8500	178.7820	199.40	6.8	6.469610e+05
...	...	...	...	...	...
21949	38.3917	-118.8941	12.30	5.6	1.482894e+09
21950	38.3777	-118.8957	8.80	5.5	1.482897e+09
21951	36.9179	140.4262	10.00	5.9	1.482909e+09
21952	-9.0283	118.6639	79.00	6.3	1.483031e+09
21953	37.3973	141.4103	11.94	5.5	1.483109e+09

21954 rows × 5 columns

Result of Preprocessing:

- The preprocessing steps are foundational in preparing the earthquake dataset for further analysis and model development.
- By cleaning, organizing, and standardizing the data, we create a solid foundation for accurate and meaningful predictions regarding earthquake occurrences and magnitudes.
- This dataset can now be used for exploratory data analysis, feature engineering, and the development of machine learning models.

As we progress further, we will dive into more advanced aspects of machine learning and develop a model that can potentially contribute to earthquake forecasting and risk reduction.

Model Creation

Ensemble Learning Method

Ensemble learning is a machine learning method that combines the predictions of multiple models (classifiers or regressors) to improve overall predictive performance.

The basic idea behind ensemble learning is that by combining the output of several base models, the ensemble model can often achieve better results than any individual model.

Choose Base Models:

Select a set of diverse base models. Diversity is key to achieving better results. Common base models include decision trees, random forests, support vector machines, gradient boosting models e.g. XGBoost, LightGBM, neural networks, and linear regression.

Train Base Models:

Train each base model on the training data. Each one of the models learns to make predictions based on the input data.

Combine Predictions:

Combine the predictions of the base models to create an ensemble prediction. The method of combination depends on whether you're dealing with classification or regression tasks:

For classification, you can use majority voting (for "hard" voting) or weighted averaging of class probabilities (for "soft" voting).

For regression, you can use simple averaging or weighted averaging of the numeric predictions.

Performance Evaluation:

Evaluate the overall performance of the ensemble on a separate testing dataset to assess its ability to generalize to unseen data.

Model Testing:

After combining the base models the performance is analysed, the model learns to map the input features to earthquake magnitudes. Validation data is used to monitor the model's performance during training.

Model Building

Import Modules:

Import statements provide the necessary tools to build, train, and evaluate your deep learning model for earthquake magnitude prediction. The model architecture includes RandomForestRegressor, GradientBoostingRegressor, VotingRegressor, AdaBoostRegressor, SVR, XGBRegressor, and Lasso are regression algorithms from scikit-learn used to build machine learning models.

Data Loading and Preprocessing:

The code loads our earthquake data from a CSV file named 'resultdata.csv' using pd.read_csv.

Feature Extraction:

It extracts features from the dataset, which include 'Longitude', 'Latitude', 'Depth', and 'Timestamp'. These features are stored in the features array.

The target variable, 'Magnitude', is stored in the labels array.

Normalization:

The longitude and latitude features are normalized to a range between 0 and 1 using the MinMaxScaler from scikit-learn. This scaling helps neural networks perform better.

Train-Test Split:

The dataset is split into training and testing sets using train_test_split from scikit-learn. The split ratio is 80% for training and 20% for testing.

Base Models:

- **RandomForestRegressor:** Ensemble of 100 decision trees for earthquake magnitude prediction. random_state=42 ensures consistent results.
- **GradientBoostingRegressor:** Utilizes 100 decision trees to refine magnitude predictions iteratively. random_state=42 for reproducible results.
- **Support Vector Regressor (SVR):** Predicts magnitudes using support vector machines with an RBF kernel. C=1.0, epsilon=0.2 control error and tube width.
- **XGBoostRegressor:** Efficiently combines 100 trees for magnitude prediction. random_state=42 ensures consistent outcomes.
- **AdaBoostRegressor:** Combines 100 weak learners for more accurate predictions. random_state=42 for result reproducibility.
- **LightGBMRegressor:** Speedy gradient boosting with 100 trees for magnitude estimation. random_state=42 for reproducibility.
- **Lasso Regression:** L1-regularized linear regression for magnitude prediction. alpha=0.1 controls model complexity and feature selection.

```
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

gb_regressor = GradientBoostingRegressor(n_estimators=100,
random_state=42)

svr_regressor = SVR(kernel='rbf', C=1.0, epsilon=0.2)

xgb_regressor = XGBRegressor(n_estimators=100, random_state=42)

ada_regressor = AdaBoostRegressor(n_estimators=100, random_state=42)

lgbm_regressor = lgbm.LGBMRegressor(n_estimators=100, random_state=42)

lasso_regressor = Lasso(alpha=0.1)
```

Ensemble Model:

- An ensemble model is created using the VotingRegressor from scikit-learn. This ensemble model combines the predictions of the previously defined individual regressors.
- Each individual regressor is provided a name and is included in the ensemble.
- The VotingRegressor combines the predictions through a weighted average.

```
ensemble_regressor = VotingRegressor(estimators=[

('random_forest', rf_regressor), ('gradient_boosting', gb_regressor),('svr', svr_regressor),

('xgboost', xgb_regressor), ('adaboost', ada_regressor), ('lightgbm', lgbm_regressor),

 ('lasso', lasso_regressor)

])
```

Model Training:

The ensemble model is trained on the training data using the fit method.

Model Predictions:

The trained ensemble model is used to make predictions on the test data using the predict method.

Model Evaluation:

The Mean Squared Error (MSE) is calculated to evaluate the performance of the ensemble model. MSE is a common metric used to measure the accuracy of regression models.

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor, AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from xgboost import XGBRegressor
import lightgbm as lgbm  # Import LightGBM
from sklearn.linear_model import Lasso

data = pd.read_csv('resultdata.csv')

features = data[['Latitude', 'Longitude','Depth','Timestamp']].values
labels = data['Magnitude'].values

scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)

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

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
gb_regressor = GradientBoostingRegressor(n_estimators=100, random_state=42)
svr_regressor = SVR(kernel='rbf', C=1.0, epsilon=0.2)
xgb_regressor = XGBRegressor(n_estimators=100, random_state=42)
ada_regressor = AdaBoostRegressor(n_estimators=100, random_state=42)
lgbm_regressor = lgbm.LGBMRegressor(n_estimators=100, random_state=42)  # Use LightGBM
lasso_regressor = Lasso(alpha=0.1)

ensemble_regressor = VotingRegressor(estimators=[
    ('random_forest', rf_regressor),
    ('gradient_boosting', gb_regressor),
    ('svr', svr_regressor),
    ('xgboost', xgb_regressor),
    ('adaboost', ada_regressor),
    ('lightgbm', lgbm_regressor),
    ('lasso', lasso_regressor)
])

ensemble_regressor.fit(X_train, y_train)

# Make predictions using the ensemble model
y_pred = ensemble_regressor.predict(X_test)

# Calculate Mean Squared Error as a metric
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000405 seconds.
```

Result:

- The result of the program is a quantitative evaluation of the earthquake magnitude pre diction model. In the code, the test loss is reported as 0.1704475. This value represent s the mean squared error (MSE) of the model's predictions on the test dataset.
- Test Loss: 0.1704475585642093

- A test loss of 0.1704475585642093 indicates the average squared difference between the model's predicted earthquake magnitudes and the actual earthquake magnitudes in the test dataset. A lower test loss is generally preferred, as it implies that the model's p redictions are closer to the actual values.
- Keep in mind that while the test loss is an essential quantitative metric, it's often a good practice to complement it with visualizations and other evaluation metrics to gain a more comprehensive understanding of the model's performance.
- The interpretation of the test loss value can also depend on the specific context and requirements of your earthquake magnitude prediction project.

Model Evaluation:

- The Mean Squared Error (MSE) is calculated to evaluate the performance of the ensemble model.
- MSE is a common metric used to measure the accuracy of regression models.

```
In [10]: # Calculate Mean Squared Error as a metric
         mse = mean_squared_error(y_test, y_pred)
         print(f"Mean Squared Error: {mse}")

         Mean Squared Error: 0.1704475585642093
```

Conclusion:

- They empower individuals, organizations, and communities to better understand seismic risks and make informed decisions to mitigate the impact of earthquakes.
- By combining data analysis, machine learning, and geospatial visualization, we enhance our ability to comprehend, predict, and respond to seismic events.
- In conclusion, this project is a starting point for earthquake magnitude prediction using deep learning. It demonstrates the potential of Ensemble models in capturing temporal patterns in earthquake data. Further refinements and research are required to improve the accuracy and reliability of earthquake predictions, contributing to the safety and preparedness of regions prone to seismic activity.