ARTIFICIAL INTELLIGENCE

EARTHQUAKE PREDICTION

Phase 2 Assesment

Creating a notebook:

- Open a new notebook in google colab
- Name the notebook as “Earthquake prediction”
- Connect the machine to run the cells
- Upload the dataset in the notebook

Notebook link:
https://colab.research.google.com/drive/1gxaGQVUDJSyn8ZgIz9xzbVH870R9ft1s?usp=sharing

Importing the dataset

Using the pandas package we can import the dataset witha built-in function ‘read_csv()’ and passing the name of the dataset as an argument.

#Importing necessary packages

import pandas as pd
import matplotlib.pyplot as plt
import time
import datetime

#Reading the dataset using pandas

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

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap
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

5 rows × 21 columns

Data analysis

- Before usage of the data in the dataset we need to analyse the data.
- Firstly get to know the features that are important for the model.
- Secondly we need to know which features belongs to input and which features belongs to output

```
[ ] 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')
```

```
[ ] data.isnull

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

          Timestamp
0      -157630542.0
1      -157465811.0
2      -157355642.0
3      -157093817.0
4      -157026430.0
```

```
data.rank
```

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

```
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

Feature Engineering and Data cleaning

Once the analysis and feature selection is done we can create or manipulate the features according to the need of our problem statement and output needed. Here we create a new feature called ‘*Timestamp’ using* two features from our dataset ie. ‘*Date’* and ‘*Time’.* The features date and time are string objects which are needed to be converted to object datatype for our convenience.

Code snippet

```
#Feature engineering

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:
        timestamp.append('ValueError')
```

#converting array into Series using pandas

timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
pdata = data [data['Timestamp']!='ValueError']

#Selection of important features

pdata = pdata[['Latitude', 'Longitude', 'Depth', 'Magnitude','Timestamp']]
pdata.shape

```
(23409, 5)
```

Post analysis

```
[ ] pdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23409 entries, 0 to 23411
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Latitude   23409 non-null  float64
 1   Longitude  23409 non-null  float64
 2   Depth      23409 non-null  float64
 3   Magnitude  23409 non-null  float64
 4   Timestamp  23409 non-null  object
dtypes: float64(4), object(1)
memory usage: 1.1+ MB
```

```
[ ] pdata.describe()
```

	Latitude	Longitude	Depth	Magnitude
count	23409.000000	23409.000000	23409.000000	23409.000000
mean	1.678763	39.636726	70.748526	5.882558
std	30.113379	125.514881	122.605748	0.423084
min	-77.080000	-179.997000	-1.100000	5.500000
25%	-18.652000	-76.352000	14.530000	5.600000
50%	-3.569000	103.981000	33.000000	5.700000
75%	26.188000	145.027000	54.000000	6.000000
max	86.005000	179.998000	700.000000	9.100000

```
pdata.isnull
```

```
<bound method DataFrame.isnull of        Latitude  Longitude   Depth  Magnitude     Timestamp
0       19.2460   145.6160  131.60        6.0  -157630542.0
1        1.8630   127.3520   80.00        5.8  -157465811.0
2      -20.5790  -173.9720   20.00        6.2  -157355642.0
3      -59.0760   -23.5570   15.00        5.8  -157093817.0
4       11.9380   126.4270   15.00        5.8  -157026430.0
...         ...        ...     ...        ...           ...
23407   38.3917  -118.8941   12.30        5.6  1482913332.0
23408   38.3777  -118.8957    8.80        5.5  1482916427.0
23409   36.9179   140.4262   10.00        5.9  1482928731.0
23410   -9.0283   118.6639   79.00        6.3  1483050619.0
23411   37.3973   141.4103   11.94        5.5  1483128508.0

[23409 rows x 5 columns]>
```

Once the data cleaning is done, post analysis is made to know about the data after the pre-processing of the data. The analysis reaveals the reduced dataset with important features in the columns and there are no known null values in the cleansed data.

Hyperparameter tuning

- Hyperparameter tuning is used to improve the model created to increase the accuracy.
- By understanding the analysis made on the dataset we can use different parameter for training the model.
- Finding the right set of hyperparameters can lead to better accuracy, faster training, and more robust models.
- The parameters like the layers, activation functions, learning rate can be adjusted to improve the performance.

```
params = { ‘neurons’ : [16, 32, 64, 128],
           ‘batch_size’ : [10, 20, 30, 40],
           ‘epochs’ : [25, 50, 75, 100],
           ‘activation’ : [‘ ‘sigmoid’, ‘relu’ ]
           ‘optimizer’ : ['SGD', 'Adadelta'],
           ‘loss’ : [‘mse’, ‘mae’]}
```

Ensembling techniques

Ensembling techniques in machine learning involve combining multiple individual models (often referred to as base models or weak learners) to create a more robust and accurate composite model. Ensembling can lead to improved predictive performance, reduced overfitting, and increased model stability.

- ***Bagging***: Reduces variance by training multiple base models independently on different subsets of the training data. Commonly used with Random Forests.
- ***Boosting***: Reduces bias by training a sequence of base models, where each model corrects the errors of the previous one. Includes algorithms like AdaBoost and Gradient Boosting.
- ***Stacking***: Combines the predictions of multiple base models using a meta-learner to make the final prediction. Effective for leveraging diverse models.
- ***Voting***: Combines predictions through majority voting or averaging, often used in ensemble classifiers and regressors.