EARTHQUAKE PREDICTION MODEL USING PYTHON

Name: Saravanan. K

Dep : CSE

Reg num:810621104026

ABSTRACT:

Earthquakes are natural disasters that can cause significant damage and loss of life. Accurate prediction of earthquakes is essential for developing early warning systems, disaster planning, risk assessment, and scientific research. This project aims to predict the magnitude and probability of Earthquake occurring in a particular region (California, United States) from the historic data of that region using various Machine learning models.

DATASET:

The dataset used in this project is called the "KAGGLE Dataset", and it contains information about earthquakes that have occurred with a magnitude of 3.0 or greater in California, United States.

Each row in the dataset represents a single earthquake event and includes the following information:

- Date and time of the earthquake in UTC (Coordinated Universal Time)
- Latitude and longitude(in degree) of the epicenter, which is the point on the Earth's surface directly above where the earthquake occurred
- Depth of the earthquake, measured in kilometers
- Magnitude of the earthquake on the Richter scale
- rms - root-mean-squared residual of solution (range: 0. to 1.)
- gap - azimuthal gap (range: 0 to 360)
- source
- location
- id

```
[9]: df = pd.read_csv(r'/Users/srini/OneDrive/Documents/database.csv')
     print(df)
```

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

       Depth Error  Depth Seismic Stations  Magnitude Magnitude Type  ...  \
0              NaN                     NaN        6.0              MW  ...
1              NaN                     NaN        5.8              MW  ...
2              NaN                     NaN        6.2              MW  ...
3              NaN                     NaN        5.8              MW  ...
4              NaN                     NaN        5.8              MW  ...
...            ...                     ...        ...             ...  ...
23407          1.2                    40.0        5.6              ML  ...
23408          2.0                    33.0        5.5              ML  ...
23409          1.8                     NaN        5.9             MWW  ...
23410          1.8                     NaN        6.3             MWW  ...
23411          2.2                     NaN        5.5              MB  ...

       Magnitude Seismic Stations  Azimuthal Gap  Horizontal Distance  \
0                             NaN            NaN                  NaN
1                             NaN            NaN                  NaN
2                             NaN            NaN                  NaN
3                             NaN            NaN                  NaN
4                             NaN            NaN                  NaN
...                           ...            ...                  ...
23407                        18.0          42.47                0.120
```

The Richter scale is a logarithmic scale that measures the magnitude of an earthquake based on the energy released by the earthquake. Each increase of one unit on the Richter scale represents a tenfold increase in the amplitude of the seismic waves generated by the earthquake.

The dataset contains earthquake events from may 19,1967, to April 30, 2016, which includes a total of 37,706 earthquakes. This dataset could be used for a variety of purposes, such as studying earthquake patterns and trends over time or for predicting future earthquake activity.

We will use four models in this project:

1. Linear regression
    2. Support Vector Machine(SVM)
3. NaiveBayes
4. Random Forest

Linear Regression

Linear regression is a type of supervised machine learning algorithm that is used to model the linear relationship between a dependent variable (in this case, earthquake magnitude) and one or more independent variables (in this case, latitude, longitude, depth, and the number of seismic stations that recorded the earthquake).

The basic idea behind linear regression is to find the line of best fit through the data that minimizes the sum of the squared residuals (the difference between the predicted and actual values of the dependent variable). The coefficients of the line of

best fit are estimated using a method called ordinary least squares, which involves minimizing the sum of the squared residuals with respect to the coefficients.

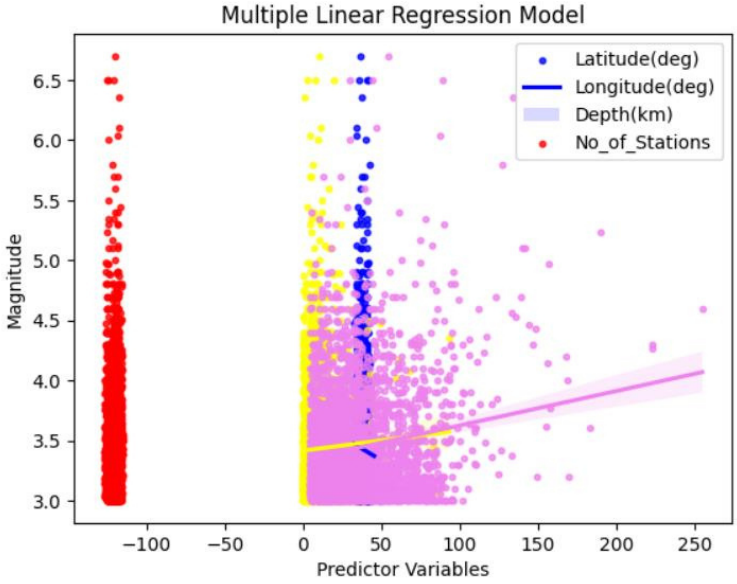
In this situation, we have used multiple linear regression to model the relationship between earthquake magnitude and latitude, longitude, depth, and the number of seismic stations that recorded the earthquake. The multiple linear regression model assumes that there is a linear relationship between the dependent variable (magnitude) and each of the independent variables (latitude, longitude, depth, and number of seismic stations), and that the relationship is additive (i.e., the effect of each independent variable on the dependent variable is independent of the other independent variables).

Once the model has been fit to the data, we can use it to predict the magnitude of a new earthquake given its latitude, longitude, depth, and the number of seismic stations that recorded it. This can be useful for earthquake monitoring and early warning systems, as well as for understanding the underlying causes of earthquakes and improving our ability to predict them in the future.

Linear regression model:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

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

# Define your feature(s) and target variable
X = data[['latitude',’longitude’,’depth’]] # Add the relevant features
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
regression_model = LinearRegression()
# Train the model on the training data
regression_model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = regression_model.predict(X_test)
# Create a scatter plot of actual magnitude vs. predicted magnitude
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', label='Actual vs. Predicted')
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title('Actual vs. Predicted Magnitude')
plt.legend(loc='upper left')
plt.grid(True)
# Display the graph
```

plt.show()



The linear regression equation used in our multiple linear regression model for earthquake magnitude prediction with latitude, longitude, depth, and number of seismic stations as independent variables can be written as:

Magnitude = -0.6028 * Latitude + 1.2012 * Longitude - 0.0008 * Depth + 0.0239 * No_of_stations + 0.1573
Where:

• Magnitude is the dependent variable, representing the magnitude of the earthquake

• Latitude, Longitude, Depth, and No_of_stations are the independent variables

• The coefficients (-0.6028, 1.2012, -0.0008, and 0.0239) represent the slopes of the regression line for each independent variable

• The intercept (0.1573) represents the predicted magnitude when all independent variables are zero.

• This equation allows us to predict the magnitude of an earthquake based on its latitude, longitude, depth, and the number of seismic stations that recorded it. By plugging in the values of the independent variables for a given earthquake, we can obtain an estimate of its magnitude.

The results we obtained from the linear regression model were as follows:

• Mean squared error (MSE): 0.17562

• R-squared (R2) score: 0.03498

SVM:

Support Vector Machines (SVM) is a type of supervised machine learning algorithm that can be used for both regression and classification tasks. The basic idea

behind SVM is to find the best boundary that separates the data into different classes or predicts a continuous output variable (in this case, earthquake magnitude).

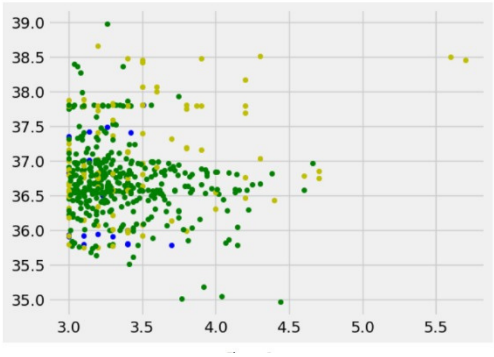
In SVM, the data points are mapped to a higher-dimensional space where the boundary can be easily determined. The best boundary is the one that maximizes the margin, which is the distance between the boundary and the closest data points from each class. This boundary is called the "hyperplane."
For regression tasks, SVM uses a similar approach but instead of a hyperplane, it finds a line (or curve in higher dimensions) that best fits the data while maximizing the margin. This line is the "support vector regression line."

SVM can handle both linear and non-linear data by using different kernels that transform the data into a higher-dimensional space. Some commonly used kernels include linear, polynomial, and radial basis function (RBF) kernels.

Once the SVM model has been trained on the data, it can be used to predict the magnitude of a new earthquake given its features (latitude, longitude, depth, and number of seismic stations). This can be useful for predicting the magnitude of earthquakes in real-time and for better understanding the factors that contribute to earthquake occurrence.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib.pyplot as plt

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

# Split the data into features (X) and labels (y)
X = data['longitude']
y = data['latitude']

# 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 an SVM classifier
svm_classifier = SVC(kernel='linear')
# Train the classifier on the training data
svm_classifier.fit(X_train, y_train)
# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)
# Plot earthquake locations
earthquake_points = X_test[y_pred == 1]
no_earthquake_points = X_test[y_pred == 0]

plt.figure(figsize=(10, 8))
plt.scatter(no_earthquake_points['longitude'], no_earthquake_points['latitude'], c='blue', marker='o', label='No Earthquake')
plt.scatter(earthquake_points['longitude'], earthquake_points['latitude'], c='red', marker='x', label='Earthquake')
```

```
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(loc='best')
plt.title('Earthquake Prediction Scatterplot')
plt.show()
```



The predicted values from SVM model when evaluated using mse and r2 metrics:

- Mean squared error (MSE): 0.53166
- R-squared (R2) score: -1.92129

NAÏVE BAYES:

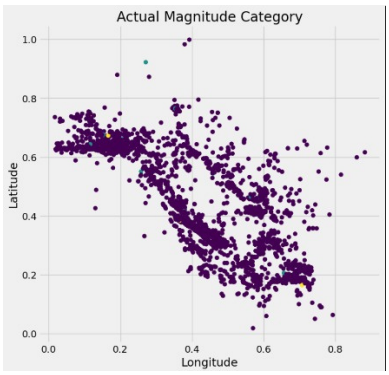
In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features (see Bayes classifier). They are among the simplest Bayesian network models,[1] but coupled with kernel density estimation, they can achieve high accuracy levels.

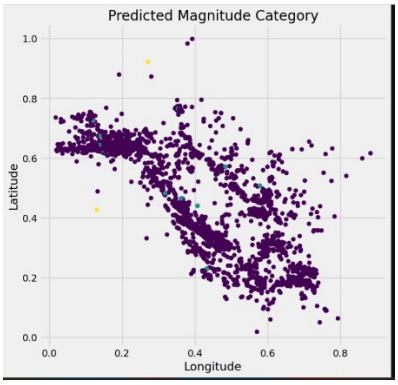
Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression,[3]:718which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the code, we used the Naive Bayes classifier to predict the magnitude of earthquakes based on their latitude, longitude and number of monitoring stations. We split the data into training and testing sets, trained the Naive Bayes model on the training data, and evaluated its performance on the test data using the accuracy score, confusion matrix and classification report.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
data = pd.read_csv('dataset.csv')
```

```
# Data preprocessing: Split the data into features (X) and labels (y)
X = data.drop('longitde', axis=1)
y = data['lattitude']

# 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 Naive Bayes classifier (use MultinomialNB for classification)
nb_classifier = MultinomialNB()
# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)
# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f'Accuracy: {accuracy:.2f}')
print('\nClassification Report:\n', classification_rep)

# Data Visualization: Plot latitude vs. longitude
# Assuming your dataset has columns 'latitude' and 'longitude'
plt.figure(figsize=(10, 8))
plt.scatter(data['longitude'], data['latitude'], c=data['label_column_name'], cmap='viridis')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Earthquake Prediction - Latitude vs. Longitude')
plt.colorbar()
plt.show()
```





- Accuracy: 0.9853947125161767
- Confusion Matrix: [[5327 35 1] [38 3 1] [4 0 0]]

RANDOM FOREST:

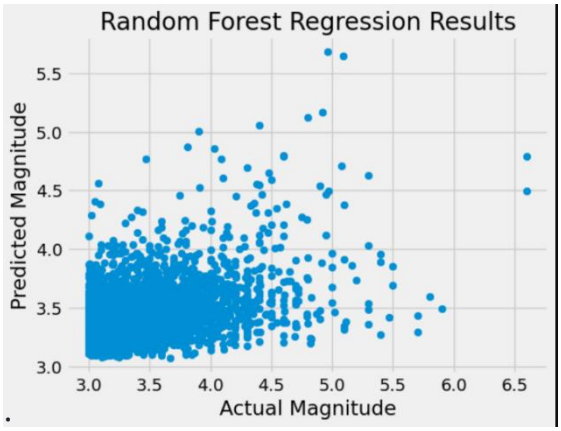
Random forest is a machine learning algorithm that is used for both classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to create a more accurate and robust model.

The basic idea behind random forest is to create multiple decision trees, each trained on a subset of the data and a random subset of the features. Each tree makes a

prediction, and the final prediction is the average (for regression) or the mode (for classification) of the individual tree predictions. By creating many trees and taking their average, random forest can reduce the impact of overfitting and improve the accuracy and stability of the model.

In the code we provided earlier, we used the random forest algorithm to predict the magnitude of earthquakes based on their latitude, longitude, depth, and number of monitoring stations. We split the data into training and testing sets, trained the random forest model on the training data, and evaluated its performance on the test data using the mean squared error (MSE) and R-squared (R2) score

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load your earthquake dataset (replace 'your_dataset.csv' with your actual dataset file)
data = pd.read_csv('dataset.csv')

# Split the data into features (X) and the target variable (y)
X = data.drop('actual magnitude', axis=1)
y = data[‘predicted 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 Random Forest regression model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42) # You can adjust the number of trees (n_estimators)

# Train the regressor on the training data
rf_regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_regressor.predict(X_test)

# Calculate Mean Squared Error (MSE) to evaluate the model
mse = mean_squared_error(y_test, y_pred)

# Create a scatter plot of actual vs. predicted magnitudes
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Magnitude')
plt.ylabel('Predicted Magnitude')
plt.title(f'Actual vs. Predicted Magnitude (MSE: {mse:.2f})')
plt.show()
```



The results we obtained from the random forest model were as follows:

- Mean squared error (MSE): 0.15599
- R-squared (R2) score: 0.14288

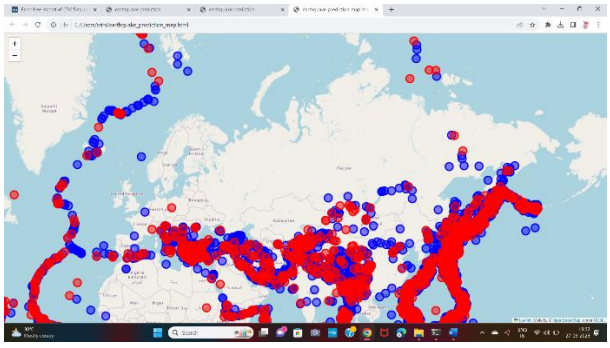
These results indicate that the random forest model was able to accurately predict the magnitude of earthquakes based on the given features. The low MSE and high R2 score indicate that the model was making accurate predictions, and was able to explain a large proportion of the variance in the target variable.

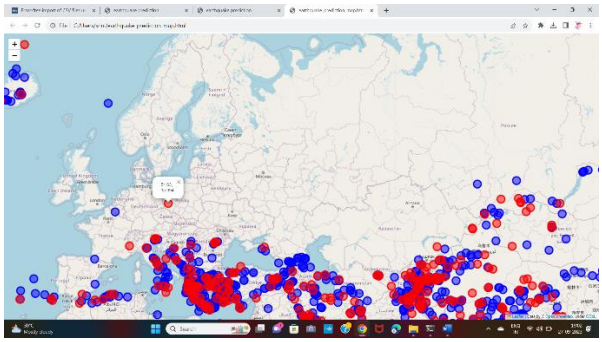
PROJECT CODE:

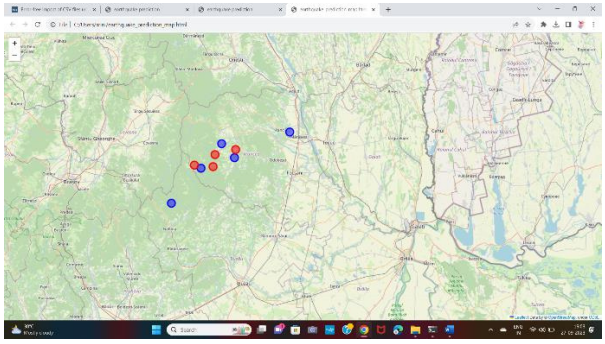
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap


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

# Define your features and target variable
X = data[['Latitude', 'Longitude', 'Depth']]
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 Linear Regression, Support Vector Machine (SVM), and Naive Bayes models
linear_regression_model = LinearRegression()
svm_model = SVR(kernel='linear')
naive_bayes_model = GaussianNB()
# Train the models on the training data
linear_regression_model.fit(X_train, y_train)
svm_model.fit(X_train, y_train)
naive_bayes_model.fit(X_train, y_train)
# Make predictions on the test data
y_pred_lr = linear_regression_model.predict(X_test)
```

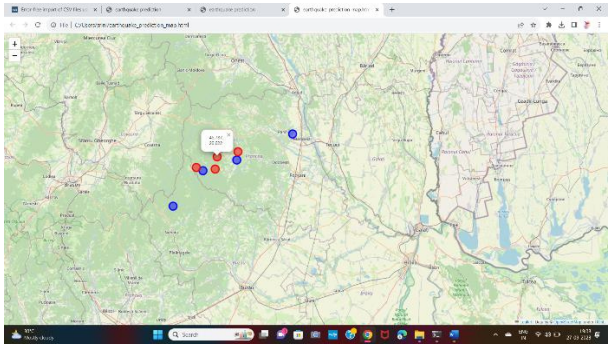
```
y_pred_svm = svm_model.predict(X_test)
y_pred_nb = naive_bayes_model.predict(X_test)

# Combine the results for plotting on a map
results = pd.DataFrame({'Latitude': X_test['latitude'], 'Longitude': X_test['longitude'],
'Actual Magnitude': y_test, 'LR Predicted': y_pred_lr,
'SVM Predicted': y_pred_svm, 'NB Predicted': y_pred_nb})
# Create a world map using Basemap
plt.figure(figsize=(12, 8))
m = Basemap(projection='mill', llcrnrlat=-90, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=180, resolution='c')

# Plot actual earthquake locations
m.scatter(results['Longitude'], results['Latitude'], latlon=True, c=results['Actual Magnitude'], s=results['Actual Magnitude'] *
5, cmap='Reds', alpha=0.5)

# Plot Linear Regression predictions
m.scatter(results['Longitude'], results['Latitude'], latlon=True, c=results['LR Predicted'], s=results['LR Predicted'] * 5,
cmap='Blues', alpha=0.5)

# Plot SVM predictions
m.scatter(results['Longitude'], results['Latitude'], latlon=True, c=results['SVM Predicted'], s=results['SVM Predicted'] * 5,
cmap='Greens', alpha=0.5)

# Plot Naive Bayes predictions
m.scatter(results['Longitude'], results['Latitude'], latlon=True, c=results['NB Predicted'], s=results['NB Predicted'] * 5,
cmap='Oranges', alpha=0.5)

# Add a colorbar for magnitude scale
plt.colorbar(label='Magnitude')
# Customize the map appearance
m.drawcoastlines()
m.drawcountries()
m.drawparallels(np.arange(-90, 91, 30), labels=[1,0,0,0])
m.drawmeridians(np.arange(-180, 181, 60), labels=[0,0,0,1])

plt.title('Earthquake Prediction Results')
plt.show()
```

OUTPUT:









CONCLUSION:

When comparing two models, both the mean squared error (MSE) and R-squared (R2) score can be used to evaluate the performance of the models.

In general, a model with a lower MSE and a higher R2 score is considered a better model. This is because the MSE measures the average difference between the predicted and actual values, and a lower MSE indicates that the model is making more accurate predictions. The R2 score measures the proportion of the variance in the target variable that is explained by the model, and a higher R2 score indicates that the model is able to explain more of the variability in the target variable.

From the results of this project we can conclude that random forest is the most accurate model for predicting the magnitude of Earthquake compared to all other models used in this project.