Navya Bhandari (2115500100) – 3R

IBM Assignment

Problem Statement:

Earthquakes, floods, hurricanes – the fury of nature leaves a trail of devastation. But what if we could predict these disasters with greater accuracy and lead time, saving lives and minimizing damage? Can AI be the key to outrunning disaster?

**Your mission:** Design a neural network, the ultimate "Disaster Forecaster," to predict natural disasters:

* **Decode nature's whispers:** Analyze diverse data sources like weather observations, satellite imagery, seismic sensor readings, and historical records to identify patterns and precursors leading to disasters.
* **Think ahead, act faster:** Increase prediction accuracy and lead time, giving communities precious minutes or even hours to prepare and evacuate.
* **Tailor your predictions:** Account for regional variations and specific types of disasters (e.g., earthquakes vs. floods) to deliver targeted, actionable insights.

**Bonus**: Integrate your Disaster Forecaster with emergency response systems to automate alerts, trigger early evacuations, and optimize resource allocation.

**Ready to build a more resilient future where lives are protected and communities come together in the face of danger?**

Approach:

Introduction

Natural disasters pose significant threats to lives and infrastructure worldwide. However, advancements in artificial intelligence (AI) present an opportunity to predict these disasters with greater accuracy and lead time, ultimately saving lives and minimizing damage. The Disaster Forecaster is a neural network designed to decode nature's whispers, anticipate impending disasters, and provide actionable insights to communities and emergency responders. This documentation outlines the approach and step-by-step explanation for building and deploying the Disaster Forecaster.

Approach

1. **Data Collection**: Gather diverse data sources, including weather observations, satellite imagery, seismic sensor readings, and historical records, to capture the environmental conditions and precursors associated with different types of natural disasters.

2. **Data Preprocessing**: Clean, preprocess, and standardize the collected data to ensure consistency and compatibility across different sources. This may involve handling missing values, normalizing numerical features, and encoding categorical variables.

3. **Feature Engineering**: Extract relevant features from the raw data that capture patterns and precursors indicative of impending disasters. This could involve time-series analysis, spatial analysis, and domain-specific feature extraction techniques.

4. **Model Development**: Design and train a neural network architecture capable of learning complex relationships between the input features and the occurrence of natural disasters. The model should be optimized for prediction accuracy, lead time, and scalability.

5. **Validation and Evaluation**: Validate the performance of the Disaster Forecaster using historical data and evaluate its predictive accuracy, lead time, and robustness through cross-validation and performance metrics such as precision, recall, and F1 score.

6. **Regional Variation and Disaster Specificity**: Customize the Disaster Forecaster to account for regional variations in environmental conditions and specific types of disasters. This may involve training separate models for different regions or disaster types and fine-tuning model parameters accordingly.

7. **Integration with Emergency Response Systems**: Develop interfaces and APIs to seamlessly integrate the Disaster Forecaster with existing emergency response systems. This enables automated alerts, early evacuations, and optimized resource allocation based on real-time predictions and recommendations.

8. **Deployment and Monitoring**: Deploy the Disaster Forecaster in operational settings and continuously monitor its performance and effectiveness. This involves updating the model with new data, retraining periodically to adapt to changing environmental conditions, and incorporating feedback from users and stakeholders.

Step-by-Step Explanation

1. Data Collection: Collect data from various sources, such as weather stations, satellite imagery providers, seismic sensor networks, and disaster databases. Ensure data quality and coverage across different regions and disaster types.

2. Data Preprocessing: Clean the raw data by handling missing values, removing outliers, and standardizing features. Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.

3.Feature Engineering: Extract features from the raw data that capture relevant information about environmental conditions, geological factors, and historical trends. Features may include temperature trends, rainfall patterns, seismic activity, land use, and past disaster occurrences.

4. Model Development: Design a neural network architecture tailored to the prediction task, considering factors such as input data dimensionality, complexity of relationships, and computational resources. Experiment with different architectures, including deep learning models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or transformer models.

5. Validation and Evaluation: Split the data into training and testing sets for model validation. Use techniques like k-fold cross-validation to assess the model's performance across multiple splits. Evaluate the model's predictive accuracy, lead time, and robustness using appropriate metrics and validation techniques.

6. Regional Variation and Disaster Specificity: Customize the model to account for regional variations and specific types of disasters. This may involve training separate models for different regions or disaster categories and fine-tuning model hyperparameters based on regional or disaster-specific data.

7. Integration with Emergency Response Systems: Develop APIs and interfaces to integrate the Disaster Forecaster with existing emergency response systems. Enable real-time communication and data exchange between the Forecaster and emergency management platforms to automate alerts, trigger evacuations, and allocate resources based on predicted disaster probabilities.

8. Deployment and Monitoring: Deploy the Disaster Forecaster in operational settings, ensuring scalability, reliability, and performance. Monitor the model's performance in real-time, collecting feedback from users and stakeholders to identify areas for improvement. Continuously update and retrain the model with new data to adapt to changing environmental conditions and improve predictive accuracy.

Conclusion

The Disaster Forecaster represents a powerful tool for predicting natural disasters with greater accuracy and lead time, enabling proactive measures to protect lives and minimize damage. By leveraging AI techniques and integrating with emergency response systems, the Forecaster can help communities and governments build resilience and respond effectively to environmental threats. With ongoing development and refinement, the Disaster Forecaster holds the potential to save lives and create a more resilient future for all.

CODE

import pandas as pd

from google.colab import data\_table

data\_table.enable\_dataframe\_formatter()

# Read CSV file with space delimiter

df = pd.read\_csv('/content/Earthquake\_Data.csv', delimiter=r'\s+')

# Print the first 5 rows of the data frame

display(df)

new\_column\_names = ["Date(YYYY/MM/DD)", "Time(UTC)", "Latitude(deg)", "Longitude(deg)", "Depth(km)", "Magnitude(ergs)",

"Magnitude\_type", "No\_of\_Stations", "Gap", "Close", "RMS", "SRC", "EventID"]

df.columns = new\_column\_names

ts = pd.to\_datetime(df["Date(YYYY/MM/DD)"] + " " + df["Time(UTC)"])

df = df.drop(["Date(YYYY/MM/DD)", "Time(UTC)"], axis=1)

df.index = ts

display(df)

df.info()

file\_name = 'Earthquake\_data\_processed.xlsx'

# saving the excel

df.to\_excel(file\_name)

print('DataFrame is written to Excel File successfully.')

import warnings

warnings.filterwarings(’ignore’)

from sklearn.model\_selection import train\_test\_split

# Select relevant columns

X = df[['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'No\_of\_Stations']]

y = df['Magnitude(ergs)']

# Split 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=0)

from sklearn.linear\_model import LinearRegression

# Train the linear regression model

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

from sklearn.metrics import r2\_score, mean\_squared\_error

scores= {"Model name": ["Linear regression", "SVM", "Random Forest"], "mse": [], "R^2": []}

# Predict on the testing set

y\_pred = regressor.predict(X\_test)

# Compute R^2 and MSE

r2 = r2\_score(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

scores['mse'].append(mse)

scores['R^2'].append(r2)

print("R^2: {:.2f}, MSE: {:.2f}".format(r2, mse))

# Predict on new data

new\_data = [[33.89, -118.40, 16.17, 11], [37.77, -122.42, 8.05, 14]]

new\_pred = regressor.predict(new\_data)

print("New predictions:", new\_pred)

import seaborn as sns

import matplotlib.pyplot as plt

# Plot the regression line

sns.regplot(x=X\_test['Latitude(deg)'], y=y\_test, color='blue', scatter\_kws={'s': 10})

sns.regplot(x=X\_test['Longitude(deg)'], y=y\_test, color='red', scatter\_kws={'s': 10})

sns.regplot(x=X\_test['Depth(km)'], y=y\_test, color='yellow', scatter\_kws={'s': 10})

sns.regplot(x=X\_test['No\_of\_Stations'], y=y\_test, color='violet', scatter\_kws={'s': 10})

plt.legend(labels=['Latitude(deg)', 'Longitude(deg)', 'Depth(km)', 'No\_of\_Stations'])

plt.xlabel('Predictor Variables')

plt.ylabel('Magnitude')

plt.title('Multiple Linear Regression Model')

plt.show()

from sklearn.svm import SVR

# Select a subset of the training data

subset\_size = 500

X\_train\_subset = X\_train[:subset\_size]

y\_train\_subset = y\_train[:subset\_size]

# Create an SVM model

svm = SVR(kernel='rbf', C=1e3, gamma=0.1)

# Train the SVM model on the subset of data

svm.fit(X\_train\_subset, y\_train\_subset)

# Evaluate the model on the test set

score = svm.score(X\_test, y\_test)

print("Test score:", score)

# Predict on the testing set

y\_pred\_svm = svm.predict(X\_test)

# Compute R^2 and MSE

r2\_svm = r2\_score(y\_test, y\_pred\_svm)

mse\_svm = mean\_squared\_error(y\_test, y\_pred\_svm)

scores['mse'].append(mse\_svm)

scores['R^2'].append(r2\_svm)

print("SVM R^2: {:.2f}, MSE: {:.2f}".format(r2\_svm, mse\_svm))

# Predict on new data

new\_pred\_svm = svm.predict(new\_data)

print("New SVM predictions:", new\_pred\_svm)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib import style

from sklearn.svm import SVC

style.use('fivethirtyeight')

# create mesh grids

def make\_meshgrid(x, y, h =.02):

x\_min, x\_max = x.min() - 1, x.max() + 1

y\_min, y\_max = y.min() - 1, y.max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

return xx, yy

# plot the contours

def plot\_contours(ax, clf, xx, yy, \*\*params):

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

out = ax.contourf(xx, yy, Z, \*\*params)

return out

# color = ['y', 'b', 'g', 'k']

subset\_size = 500

# modify the column names based on the dataset

features = df[['Magnitude(ergs)','Latitude(deg)']][:subset\_size].values

classes = df['Magnitude\_type'][:subset\_size].values

# create 3 svm with rbf kernels

svm1 = SVC(kernel ='rbf')

svm2 = SVC(kernel ='rbf')

svm3 = SVC(kernel ='rbf')

svm4 = SVC(kernel ='rbf')

# fit each svm's

svm1.fit(features, (classes=='ML').astype(int))

svm2.fit(features, (classes=='Mx').astype(int))

svm3.fit(features, (classes=='Md').astype(int))

fig, ax = plt.subplots()

X0, X1 = features[:, 0], features[:, 1]

xx, yy = make\_meshgrid(X0, X1)

# plot the contours

'''

plot\_contours(ax, svm1, xx, yy, cmap = plt.get\_cmap('hot'), alpha = 0.8)

plot\_contours(ax, svm2, xx, yy, cmap = plt.get\_cmap('hot'), alpha = 0.3)

plot\_contours(ax, svm3, xx, yy, cmap = plt.get\_cmap('hot'), alpha = 0.5)

'''

color = ['y', 'b', 'g', 'k', 'm']

for i in range(subset\_size):

if classes[i] == 'ML':

plt.scatter(features[i][0], features[i][1], s = 20, c = color[0])

elif classes[i] == 'Mx':

plt.scatter(features[i][0], features[i][1], s = 20, c = color[1])

elif classes[i] == 'Md':

plt.scatter(features[i][0], features[i][1], s = 20, c = color[2])

else:

plt.scatter(features[i][0], features[i][1], s = 20, c = color[4])

plt.show()

print(df.columns)

df['Magnitude\_type'].unique()

import pandas as pd

import numpy as np

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Read CSV file with space delimiter

df = pd.read\_csv('/content/Earthquake\_Data.csv', delimiter=r'\s+')

new\_column\_names = ["Date(YYYY/MM/DD)", "Time(UTC)", "Latitude(deg)", "Longitude(deg)", "Depth(km)", "Magnitude",

"Magnitude\_Category", "No\_of\_Stations", "Gap", "Close", "RMS", "SRC", "EventID"]

df.columns = new\_column\_names

# Convert magnitude column to categorical data

df['Magnitude\_Category'] = pd.cut(df['Magnitude'], bins=[0, 5, 6, 7, np.inf], labels=['Minor', 'Moderate', 'Strong', 'Major'])

# Encode Magnitude Category

le = LabelEncoder()

df['Magnitude\_Category\_Encoded'] = le.fit\_transform(df['Magnitude\_Category'])

# Normalize latitude and longitude values

scaler = MinMaxScaler()

df[['Latitude(deg)', 'Longitude(deg)']] = scaler.fit\_transform(df[['Latitude(deg)', 'Longitude(deg)']])

# Select features

X = df[['Latitude(deg)', 'Longitude(deg)', 'No\_of\_Stations']]

y = df['Magnitude\_Category\_Encoded']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the Gaussian Naive Bayes model on the training data

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

# Use the trained model to make predictions on the testing data

y\_pred = gnb.predict(X\_test)

# Calculate the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)

# Calculate and print the confusion matrix and classification report

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion Matrix:\n', cm)

cr = classification\_report(y\_test, y\_pred, labels=[0, 1, 2, 3], target\_names=['Minor', 'Moderate', 'Strong', 'Major'])

print('Classification Report:\n', cr)

# Create a scatter plot of actual vs predicted values

plt.figure(figsize=(8, 8))

plt.scatter(X\_test['Longitude(deg)'], X\_test['Latitude(deg)'], c=y\_test, cmap='viridis')

plt.title('Actual Magnitude Category')

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.show()

print(" ")

plt.figure(figsize=(8, 8))

plt.scatter(X\_test['Longitude(deg)'], X\_test['Latitude(deg)'], c=y\_pred, cmap='viridis')

plt.title('Predicted Magnitude Category')

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.show()

print(" ")

# Create a heatmap of the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')

plt.xlabel('Predicted Magnitude Category')

plt.ylabel('Actual Magnitude Category')

plt.show()

print(" ")

cr = classification\_report(y\_test, y\_pred, labels=[0, 1, 2, 3], target\_names=['Minor', 'Moderate', 'Strong', 'Major'], output\_dict=True)

# Convert classification report dictionary to DataFrame

cr\_df = pd.DataFrame(cr).transpose()

# Create bar plot of classification report scores

plt.figure(figsize=(8, 6))

sns.barplot(x=cr\_df.index, y=cr\_df['f1-score'])

plt.xlabel('Magnitude Category')

plt.ylabel('F1 Score')

plt.title('F1 Score by Magnitude Category')

plt.show()

print(" ")

from sklearn.ensemble import RandomForestRegressor

# Initialize a random forest regressor with 100 trees

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

# Fit the regressor to the training data

rf.fit(X\_train, y\_train)

# Predict the target variable on the test data

y\_pred = rf.predict(X\_test)

# Evaluate the performance of the model using mean squared error and R^2 score

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

scores['mse'].append(mse)

scores['R^2'].append(r2)

print('Mean Squared Error: ', mse)

print('R^2 Score: ', r2)

# Plot the predicted and actual values

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual Magnitude')

plt.ylabel('Predicted Magnitude')

plt.title('Random Forest Regression Results')

plt.show()

importances = rf.feature\_importances\_

features = ['Latitude', 'Longitude', 'Depth', 'No. of Stations']

plt.bar(features, importances)

plt.xlabel('Feature')

plt.ylabel('Importance')

plt.title('Feature Importance Plot')

plt.show()

import seaborn as sns

sns.residplot(x= y\_test, y =y\_pred, color='orange')

plt.xlabel('Predicted Magnitude')

plt.ylabel('Residual')

plt.title('Residual Plot')

plt.show()

plt.plot(y\_test.index[:20], y\_test[:20], color='blue', label='Actual Magnitude')

plt.plot(y\_test.index[:20], y\_pred[:20], color='orange', label='Predicted Magnitude')

plt.xlabel('Index')

plt.ylabel('Magnitude')

plt.title('Actual vs. Predicted Line Plot')

plt.legend()

plt.show()

scores\_df = pd.DataFrame(scores)

display(scores\_df)

scores\_df[scores\_df["mse"] == scores\_df["mse"].min()]

scores\_df[scores\_df["R^2"] == scores\_df["R^2"].max()]