NAAN MUDHALVAN

PROJECT: Earthquake Prediction Model using Python

PHASE 4

TEAM MEMBERS:

Srijeet Goswami - 2021504044

Akash R - 2021504501

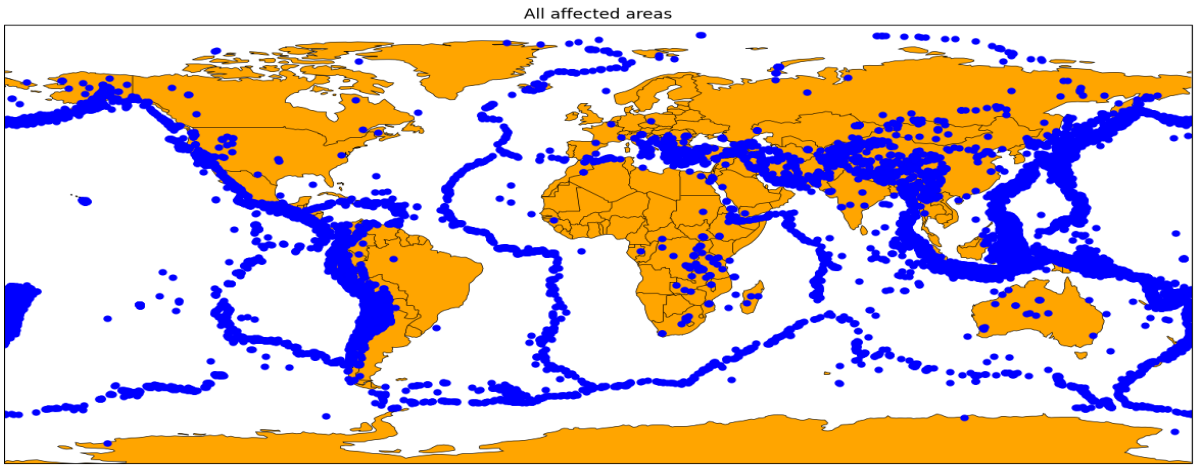
Gokul E – 2021504514

Balamurugan M – 2021504507

PROBLEM DEFINITION:

The problem is to develop an earthquake prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

In phase 3, we were successfully able to create a world map visualization to display earthquake frequency distribution as follows.



Machine learning algorithm used:

Multi-layer perceptron-

A feedforward neural network, also known as a multi-layer perceptron (MLP), is a fundamental type of artificial neural network. It consists of an input layer, one or more hidden layers, and an output layer. Here's how a feedforward neural network works:

1.Input Layer:

The framework used is Keras and specifically we have used the Sequential and Dense models. The input layer consists of neurons that represent the features or inputs of your data. Each neuron corresponds to one feature, and the values in this layer are the input data.

2.Weights and Biases:

Each connection between neurons in adjacent layers has an associated weight and a bias. These weights and biases are learned during training and determine the strength of the connections between neurons.

3.Hidden Layers:

Between the input and output layers, there can be one or more hidden layers. The hidden layers perform complex transformations on the input data through weighted sums and activation functions.

4.Weighted Sum:

For each neuron in a hidden layer, the weighted sum of its inputs is calculated. This is done by multiplying each input by its corresponding

weight and summing up these values. The weighted sum is then passed to the activation function.

5.Activation Function:

An activation function introduces non-linearity into the network. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). The activation function determines the output of a neuron based on the weighted sum.

The activation function we have used are-

- ReLU

  ReLU, or Rectified Linear Unit, is one of the most widely used activation functions in artificial neural networks, including deep learning models. It's a simple and effective non-linear activation function that introduces non-linearity into the model. The ReLU activation function is defined as:
  $f(x) = \max(0, x)$
  In other words, it outputs the input value if it's positive and zero if it's negative. The key properties of ReLU are:
  Simplicity: It's a very simple function computationally, making it efficient to compute.
  Non-linearity: ReLU introduces non-linearity, allowing neural networks to model complex relationships in the data.
  Addressing the vanishing gradient problem: ReLU can help mitigate the vanishing gradient problem that occurs with some other activation functions like sigmoid and tanh.
- Softmax

  The softmax function is a mathematical function that takes a vector of real numbers and normalizes it into a probability distribution. It's commonly used in the output layer of a neural network for multiclass classification problems, where it assigns probabilities to each class for a given input sample.

The softmax function takes an input vector (often referred to as logits or scores) and transforms it into a probability distribution over multiple classes. It is defined as follows:

softmax(x)_i = e^(x_i) / Σ(e^(x_j)), for all j

In this equation, x_i represents the input value for class i, and Σ(e^(x_j)) represents the sum of exponential values of all classes. The softmax function has the following properties:

a) Outputs a probability distribution: The result of the softmax function is a set of values between 0 and 1, and they sum up to 1. These values represent the estimated probabilities of the input belonging to each class.
b) Amplifies the largest value: The softmax function amplifies the largest input value, making it more prominent in the output distribution. This means that the class with the highest score gets the highest probability.
c) Differentiability: The softmax function is differentiable, which is important for gradient-based optimization algorithms used in training neural networks.

The softmax function is widely used for multiclass classification tasks, such as image recognition, text classification, and language modelling, where it assigns class probabilities based on the network's raw output scores.

6.Forward Propagation:

The network's computations progress from the input layer to the hidden layers and finally to the output layer. Each layer's output becomes the input for the next layer. This process is known as forward propagation.

7.Output Layer:

The output layer produces the final predictions or results of the network. The number of neurons in the output layer depends on the

problem type. For binary classification, there may be one neuron with a sigmoid activation function. For multi-class classification, there can be multiple neurons with a softmax activation function. For regression, there is typically a single neuron.

8.Loss Calculation:

The output of the network is compared to the target or ground truth values, and a loss function is used to measure the error. Common loss functions include mean squared error (MSE) for regression and categorical cross-entropy for classification.

9.Backpropagation:

To improve the network's performance, the gradients of the loss with respect to the network's weights and biases are computed. These gradients are used to update the weights and biases through optimization algorithms like gradient descent. This process is known as backpropagation.

Training and Evaluation:

The network is trained by iteratively feeding the training data through it, computing the loss, and updating the weights and biases using backpropagation. This process continues until the model converges to a state where the loss is minimized.

Once trained, the feedforward neural network can be used for making predictions on new, unseen data by simply performing forward propagation. The features we are going to provide are latitude, longitude and depth which are going to be used for predicting the magnitude of earthquake if at all it occurs there.

Feedforward neural networks are versatile and can be applied to a wide range of tasks, including classification, regression, pattern

recognition, and function approximation. The architecture and complexity of the network can vary depending on the specific problem.

Following is the code used for training and generating the predicted earthquake.

Code:

```
def train_model(neurons, activation, optimizer, loss, X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))
    model.add(Dense(neurons, activation=activation))
    model.add(Dense(2, activation='softmax'))


    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(X_test, y_test))


    [test_loss, test_acc] = model.evaluate(X_test, y_test)
    return model, test_loss, test_acc


# Use the GridSearchCV to find the best hyperparameters
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
import numpy as np


neurons = [16, 64, 128, 256]
batch_size = [10, 20, 50, 100]
epochs = [10]
activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponential']
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
loss = ['squared_hinge']
```

```
    param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)

    model = KerasClassifier(build_fn=create_model, verbose=0)
    grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
    grid_result = grid.fit(X_train, y_train)

    best_params = grid_result.best_params_
    best_neurons = best_params['neurons']
    best_activation = best_params['activation']
    best_optimizer = best_params['optimizer']
    best_loss = best_params['loss']

    # Train the best model
    best_model, test_loss, test_acc = train_model(best_neurons, best_activation, best_optimizer, best_loss, X_train, y_train, X_test, y_test)

    # Now, you can use the trained best_model to make predictions on new data
    new_data = np.array([[feature1, feature2, feature3]])  # Replace with your input features
    predicted_magnitude = best_model.predict(new_data)
    print("Predicted Magnitude: ", predicted_magnitude)
```