

NAAN MUDHALVAN

PROJECT: Earthquake Prediction Model using Python

PHASE 5

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.

DESIGN APPROACH:

1. Choosing appropriate data source
2. Feature selection
3. Visualization
4. Data splitting
5. Model development
6. Model training
7. Model evaluation

8. Iterative refinement

9. Deployment

1. DATA SOURCE:

Choosing a suitable Kaggle dataset containing earthquake data with features like date, time, latitude, longitude, depth, and magnitude.

Dataset:

<https://www.kaggle.com/datasets/usgs/earthquake-database>

Context:

All earthquakes from 1973 until 2021, worldwide are available. Contains basic information such as location, time and magnitude. The model to be deployed is to be trained using this data, following instructions from here.

Acknowledgements: The United States Geological Survey (USGS)

2. KEY FEATURES REQUIRED:

- **Tectonic Plate Boundaries:** The majority of earthquakes occur along tectonic plate boundaries. Studying the interactions between these plates, such as convergent, divergent, and transform boundaries, is fundamental for understanding earthquake potential.
- **Geological History and Paleo seismology:** Studying the geological history of an area and identifying evidence of past seismic events through paleo seismology is essential for assessing the recurrence interval of earthquakes.
- Tabulating the places having a history of encountering earthquakes till available records and recording their geographic coordinates like latitude, longitude, altitude etc.

3.DATA MANIPULATION

By viewing the dataset, we find that the following parameters are provided for different earthquakes that have occurred in the past:

'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'

4. VISUALIZATION:

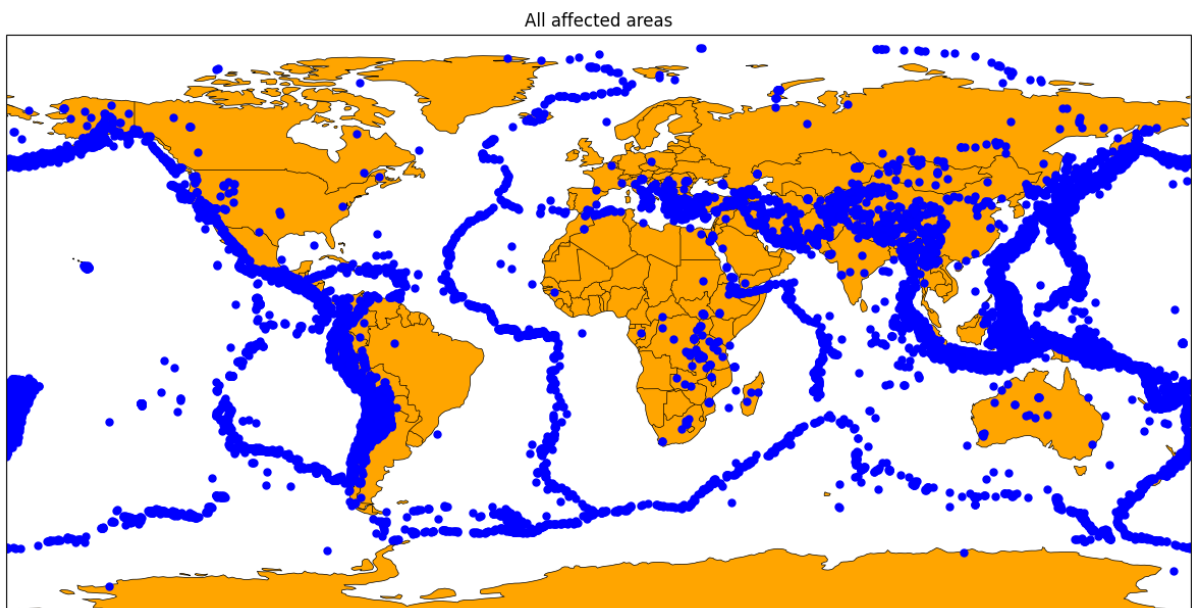
Create a world map visualization to display earthquake frequency distribution.

The following libraries are imported for this purpose:

- **cartopy.crs**
Cartopy.crs is a Python library that provides tools for defining and transforming coordinate reference systems (CRS) for cartographic visualization. It is part of the Cartopy package, which is designed to make drawing maps for data analysis and visualisation easy. Cartopy.crs allows users to create and manipulate various types of CRS objects, such as geodetic, geocentric, projected, and rotated CRS. It also enables users to transform points, lines, polygons, images, and vectors between different CRS using the Proj.4 library.
- **cartopy.feature**
Cartopy.feature is a Python library that provides tools for creating and manipulating geographic features for cartographic visualization. Cartopy.feature allows users to access various types of features, such as country borders, coastlines, lakes, rivers, and states, from Natural Earth or GSHHS shapefiles. It also enables users to customize the appearance and style of the features using Matplotlib.

- `cartopy.mpl.patch`
Cartopy.mpl.patch is a Python library that provides tools for converting shapely geometries into matplotlib paths for cartographic visualization.
- `matplotlib.pyplot`
Matplotlib.pyplot is a Python library that provides a state-based interface to Matplotlib, a comprehensive library for creating static, animated, and interactive visualizations in Python¹. Matplotlib.pyplot allows users to make plots using an implicit, MATLAB-like, way of plotting, and also opens figures on the screen and acts as the figure GUI manager
- `matplotlib.patches`
Matplotlib.patches is a Python library that provides tools for creating and manipulating 2D shapes for data visualization. It is part of the Matplotlib package, which is a comprehensive library for creating static, animated, and interactive visualizations in Python

Given below is a sample plot highlighting all the affected areas, which are represented by blue colour.



5. DATA SPLITTING:

In machine learning, datasets are often divided into two subsets: a training set and a testing (or validation) set. The training set is used to train a machine learning model, while the testing set is used to assess the model's performance and generalization to unseen data.

6. MODEL DEVELOPMENT:

Build a neural network model for earthquake magnitude prediction. Building a neural network involves designing its architecture, defining the number of layers, neurons, and activation functions. Data is pre-processed, normalized, and divided into training and testing sets.

7. MODEL TRAINING:

The model is trained using optimization algorithms to adjust weights and minimize the loss function. Regularization techniques are applied to prevent overfitting.

8. MODEL EVALUATION:

Post-training, the model's performance is assessed on the testing set.

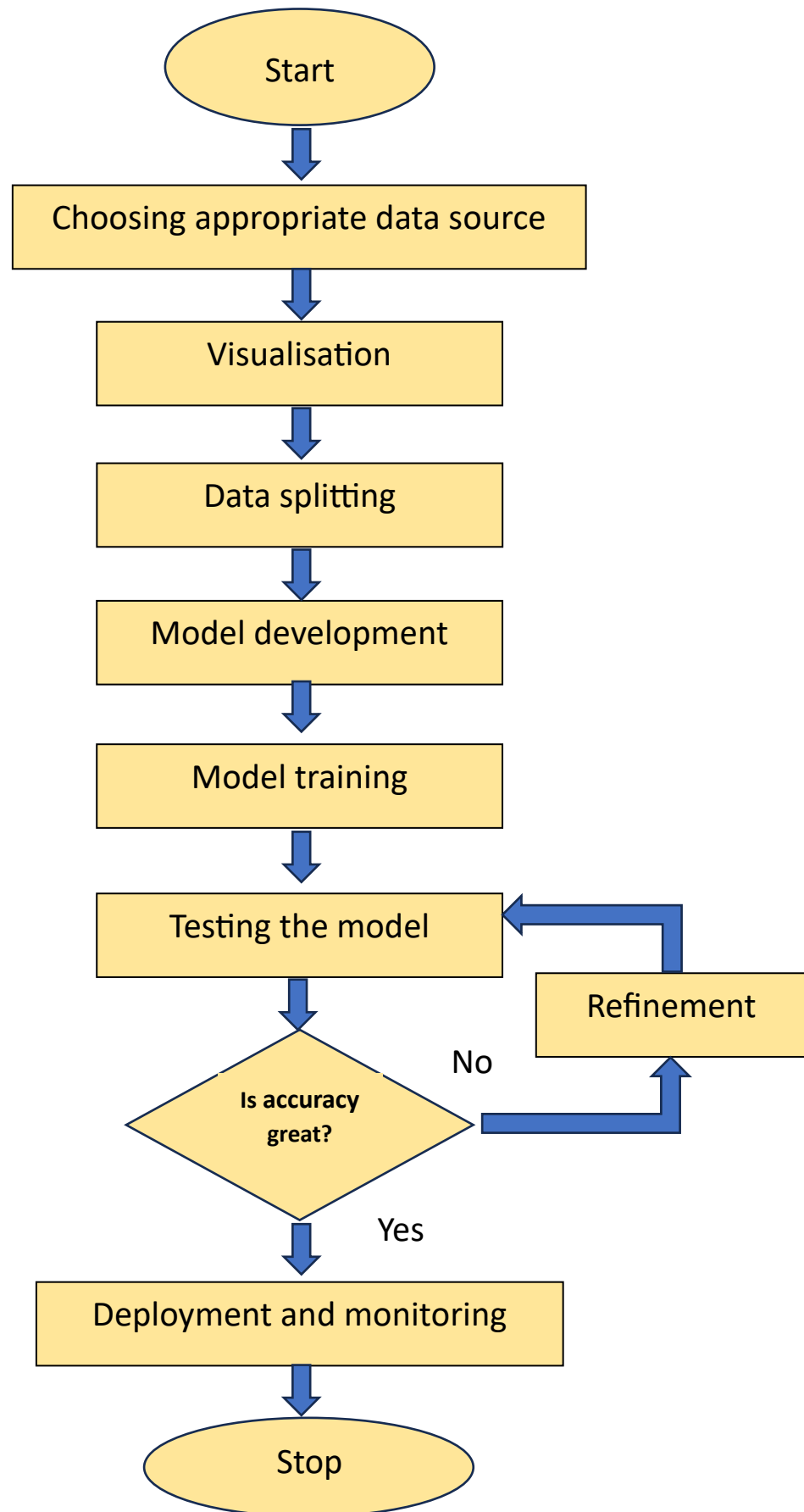
8. ITERATIVE REFINEMENT:

Iterate through the process, refining features and tuning hyperparameters to improve the model's accuracy and reliability. This iterative process underpins neural network applications in machine learning, image and speech recognition, and natural language processing, where models learn patterns and relationships in data inspired by the human brain's interconnected neurons.

9. DEPLOYMENT AND MONITORING:

Once we achieved a model with great accuracy, we will deploy it for real-time applications. The model is updated by monitoring its performance over time.

FLOWCHART:



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:

$$\text{softmax}(x)_i = e^{(x_i)} / \sum_j (e^{(x_j)}), \text{ for all } j$$

In this equation, x_i represents the input value for class i , and $\sum_j (e^{(x_j)})$ 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.