

Name : Haneen Amer / Hana Ali  
 Menna Mohamed

Instructor : Mohamed Agoor

Track : Generative Ai

**1. Project Overview**

The project aims to classify moonquake events based on seismic data collected during lunar exploration missions, specifically using a machine learning approach built on a Transformer model. This model processes velocity time series data, extracts features from time and frequency domains, and classifies different types of moonquakes. The dataset consists of CSV files containing velocity measurements over time, and a catalog that maps these files to their corresponding moonquake types.

The goal of this project is to develop a robust classifier that can accurately identify and classify moonquake events based on seismic data, potentially contributing to the study of seismic activity on the Moon.

**2. Data Processing Pipeline**

The data pipeline includes multiple steps for preprocessing, feature extraction, and model training.

**2.1 Data Loading**

The dataset consists of multiple CSV files containing two columns:

* **Velocity (m/s)**: Seismic velocity data at specific time intervals.
* **Time (sec)**: Time intervals corresponding to each velocity measurement.

The MoonquakeClassifier class is responsible for loading these files from a specified directory and matching them with entries from a catalog file. The catalog contains metadata for each file, including the type of moonquake.

**2.2 Feature Extraction**

For each seismic dataset (CSV file), several features are extracted:

* **Time-Domain Features**:
  + **Mean**: Average value of the velocity data.
  + **Standard Deviation**: Measure of the variability in the data.
  + **Maximum Absolute Value**: Maximum absolute velocity observed.
  + **Minimum**: Minimum value of the velocity data.
  + **Percentiles**: 75th and 25th percentiles.
  + **Mean of Absolute Differences**: Average of absolute differences between consecutive velocity measurements.
  + **Standard Deviation of Absolute Differences**: Measure of variability in the differences between consecutive velocity measurements.
* **Frequency-Domain Features**:
  + **FFT Magnitude**: The magnitude of the Fourier Transformed data. The Fast Fourier Transform (FFT) is used to convert the time-domain data into frequency-domain representation.
  + **Filtered FFT**: Frequencies outside a specified range (0.1 Hz to 10 Hz) are filtered out, and statistics (mean, max, sum) are computed on the remaining frequencies.
* **Spectral Features**:
  + **Spectrogram**: The spectrogram is computed using the scipy.signal.spectrogram function. It shows the frequency content over time, and features such as mean, standard deviation, and maximum power are extracted.

These extracted features are used as inputs to the machine learning model.

**2.3 Data Preprocessing**

* **Standardization**: The StandardScaler from sklearn is used to standardize the features. This ensures that the model trains efficiently, as input features are normalized to have a mean of 0 and a standard deviation of 1.
* **Label Encoding**: The target variable (moonquake type) is encoded using the LabelEncoder from sklearn. This converts categorical labels into numerical format for the classification model.

**2.4 Dataset Construction**

After extracting features from each CSV file, a dataset X (features) and y (labels) is constructed. The dataset is then split into training and testing subsets using an 80-20 split.

**3. Model Architecture**

The model is based on the **Transformer architecture**, a deep learning model primarily designed for sequence data such as text or time series. The Transformer model includes the following key components:

**3.1 TransformerModel Class**

* **Embedding Layer**: A linear layer that maps the input feature dimension to the model’s internal representation size (d\_model).
* **Positional Encoding**: A custom PositionalEncoding layer that adds positional information to the input features. This is essential for sequence-based models, as it allows the model to understand the temporal relationships between the data points.
* **Transformer Encoder Layer**: The core component of the Transformer. It uses self-attention mechanisms to capture dependencies between different time steps in the data. The encoder layer consists of multi-head attention and feed-forward neural networks.
* **Fully Connected (FC) Layer**: A final FC layer that maps the output of the Transformer encoder to the target space (moonquake classes).

**3.2 PositionalEncoding Class**

This custom class implements the sinusoidal positional encoding used in the Transformer model. It ensures that the model understands the position of each data point in the sequence, which is crucial for temporal data like time series.

**3.3 MoonquakeDataset Class**

The MoonquakeDataset class is a custom PyTorch dataset used to handle the features (X\_train and X\_test) and their corresponding labels (y\_train and y\_test). This class ensures that the data is properly formatted for training and testing.

**3.4 Training and Evaluation**

The model is trained using the **Adam optimizer** and **Cross-Entropy Loss**. The training loop consists of the following:

1. **Forward Pass**: For each batch, the input features are passed through the model, which produces a predicted classification for each sample.
2. **Loss Calculation**: The predicted output is compared to the true labels, and the loss is computed using Cross-Entropy Loss.
3. **Backward Pass**: The gradients are calculated using backpropagation, and the model parameters are updated via the optimizer.

**3.5 Model Evaluation**

After training, the model is evaluated on the test set. The **classification report** from sklearn is used to compute the following metrics:

* **Accuracy**: Proportion of correct predictions.
* **Precision**: Proportion of true positives among all positive predictions.
* **Recall**: Proportion of true positives among all actual positives.
* **F1-Score**: Harmonic mean of precision and recall, providing a balanced evaluation metric.

**4. Code Overview**

The main components of the code include:

**4.1 MoonquakeClassifier Class**

This class encapsulates the entire pipeline, from loading the data, extracting features, preparing the dataset, training the model, and evaluating its performance. The key methods in this class are:

* **extract\_features**: Extracts statistical, frequency, and spectral features from velocity time series data.
* **process\_single\_file**: Loads and processes a single CSV file, extracting the necessary features.
* **prepare\_dataset**: Loads all CSV files from the directory, extracts features, and matches them with labels from the catalog file.
* **train\_and\_evaluate**: Splits the dataset into training and testing sets, trains the model, and evaluates its performance.

**4.2 Main Function**

The main function initializes the MoonquakeClassifier class, prepares the dataset, trains the model, and prints the classification report after evaluation.

**5. Results and Performance Metrics**

The model's performance is evaluated using the **classification report**, which includes precision, recall, F1-score, and support for each class. These metrics give a detailed view of the model's ability to classify different moonquake types.

**5.1 Example Output (Classification Report):**

* **Precision**: How many of the predicted moonquake events for a particular class were correctly classified.
* **Recall**: How many of the actual moonquake events for a class were correctly identified.
* **F1-Score**: The harmonic mean of precision and recall, which balances the trade-off between the two.

**5.2 Insights:**

* The model performs well on the **Moonquake\_A** class with a high precision and recall.
* For **Moonquake\_B**, there is room for improvement in both precision and recall.
* **Moonquake\_C** has a balanced performance with good recall.

**6. Code Optimization and Future Work**

**6.1 Model Improvements**

* **Hyperparameter Tuning**: Experimenting with different values for d\_model, num\_layers, and nhead could lead to better model performance.
* **Data Augmentation**: Augmenting the seismic data (e.g., adding noise, shifting time series) could improve generalization.

**6.2 Performance Enhancements**

* **Parallelization**: Utilizing multiple GPUs or batch processing techniques could speed up training, especially with large datasets.
* **Model Pruning**: Reducing the number of parameters in the model without significantly impacting performance can make it more efficient for deployment.

**7. Conclusion**

This project successfully implemented a Transformer-based model to classify moonquake events using seismic data. The pipeline covers the entire process, from feature extraction to model training and evaluation. The model achieved reasonable accuracy, with some room for improvement in precision and recall for certain classes. The results show the potential of Transformer models in time series classification tasks, and the project sets the stage for further optimizations and extensions.