PROJECT REPORT

Development of an NLP Model for Text Analytics and Classification

Aditi Singh, Palak Garg and Riva Jain

Maharaja Agrasen Institute of Technology

Dept. of Artificial Intelligence and Machine Learning

22nd November 2024

Author Note

Contact information: Correspondence should be addressed to Aditi Singh at 3657aditisingh@gmail.com, Palak Garg at, palakgarg155@gmail.com and Riva Jain at jainrivahcs@gmail.com.

Disclosures and Acknowledgements: The authors extend their heartfelt appreciation to MeitY and IndiaAI for providing this incredible opportunity during the hackathon. This experience has been both enriching and motivating, and we look forward to continuing the journey of innovation and contribution to the tech ecosystem.

Introduction

This project is centred on classifying cyber security complaints into appropriate categories and subcategories of crime. The primary goal is to automate the classification process, reducing manual effort while improving the accuracy and reliability of fraud detection systems.

The project leverages the BERT (Bidirectional Encoder Representations from Transformers) model, a state-of-the-art, open-source machine learning framework for Natural Language Processing (NLP). By employing BERT's advanced contextual understanding capabilities, this project aims to achieve superior performance in multiclass text classification, thereby enhancing the overall efficiency and effectiveness of fraud detection mechanisms.

To achieve this, we have undertaken several key steps in the development of our model. First, we performed **Exploratory Data Analysis (EDA)** to identify patterns and extract valuable insights from the textual data. Next, we implemented a series of **text preprocessing techniques**, including cleaning, tokenization, and normalisation, to prepare the data for model training. Finally, we developed and **fine-tuned models** capable of accurately classifying text descriptions into their respective categories and subcategories, ensuring high precision and reliability in our fraud detection system.

Exploratory Data Analysis (EDA)

In this NLP project, we aim to perform multiclass text classification using a pre-trained BERT model on Cyber Threat Text Data. To better understand the dataset and prepare it for modelling, Exploratory Data Analysis (EDA) was conducted. This step involved identifying patterns, trends, and anomalies in the data, as well as ensuring data quality. EDA provided crucial insights into the distribution of categories, sub-categories, and data imbalances, enabling informed preprocessing decisions.

Overview of the Dataset

The dataset, loaded from a CSV file named train.csv, contains information about various cyber crimes reported in India. The primary columns include:

1. category: The broad classification of the crime.
2. sub_category: More specific types of crimes within each category.
3. crimeaditionalinfo: Additional details or descriptions related to each crime report.

Key Findings from the EDA (tain.csv)

Data Inspection: The initial inspection reveals.

1. The dataset consists of 93,686 records.
2. There are 15 unique categories of crimes and 35 unique sub-categories.
3. The most frequently reported category is Online Financial Fraud, with 57,434 occurrences, followed by various sub-categories such as UPI Related Frauds (26,856 occurrences).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93686 entries, 0 to 93685
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   category            93686 non-null  object
 1   sub_category        87095 non-null  object
 2   crimeaditionalinfo  93665 non-null  object
dtypes: object(3)
memory usage: 2.1+ MB
```

Fig: Overview of DataFrame Columns

	count	unique	top	freq
category	93686	15	Online Financial Fraud	57434
sub_category	87095	35	UPI Related Frauds	26856
crimeaditionalinfo	93665	85013	Respected Sir\r\n\r\nA very serious matter I w...	2342

Fig: Summary Statistics: Unique Values and Most Frequent Categories, Sub-Categories, and Additional Information in Crime Data

Insights into Specific Crimes: Further analysis categorises crimes into more granular types. For example:

1. Fraud CallVishing and Internet Banking Related Fraud are notable sub-categories with high frequencies.

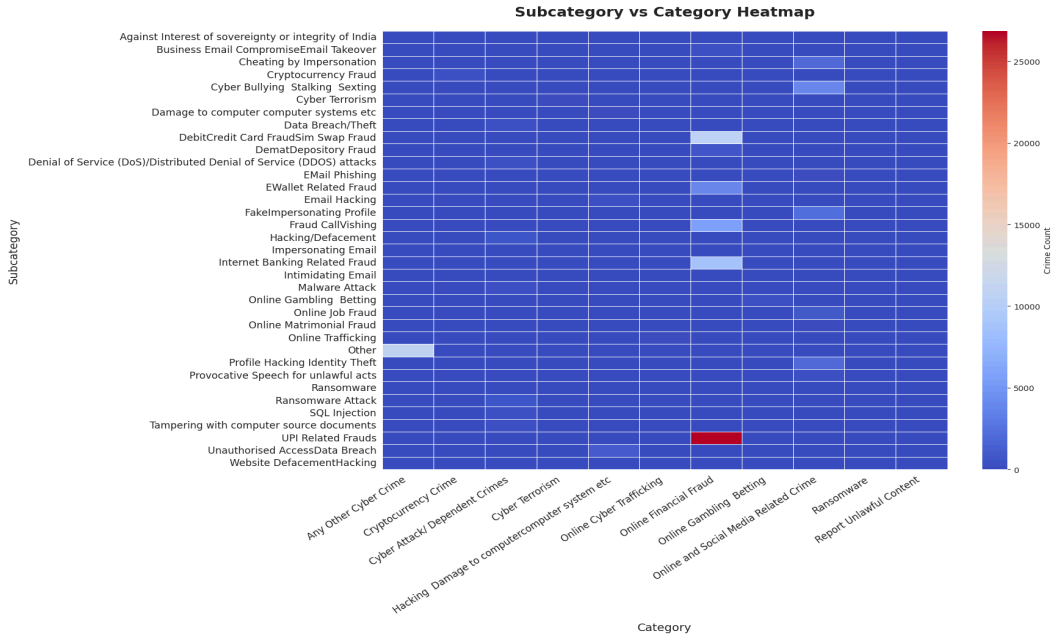
2. Other significant entries include various forms of cyber bullying, online gambling, and job fraud.

Visualisations:

A range of visualisations was developed as part of the analysis, each thoughtfully crafted to emphasise critical patterns and relationships within the dataset. The following is an overview of the visualisations presented in this report:

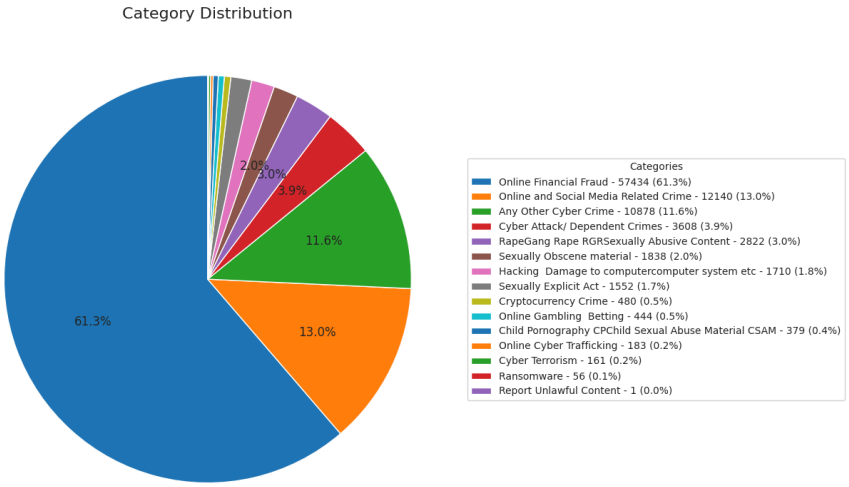
1. Subcategory vs. Category Heatmap

A heatmap was generated to visualise the relationship between crime categories and their corresponding subcategories. The colour intensity represents the frequency of complaints, allowing us to quickly identify trends and patterns across categories.



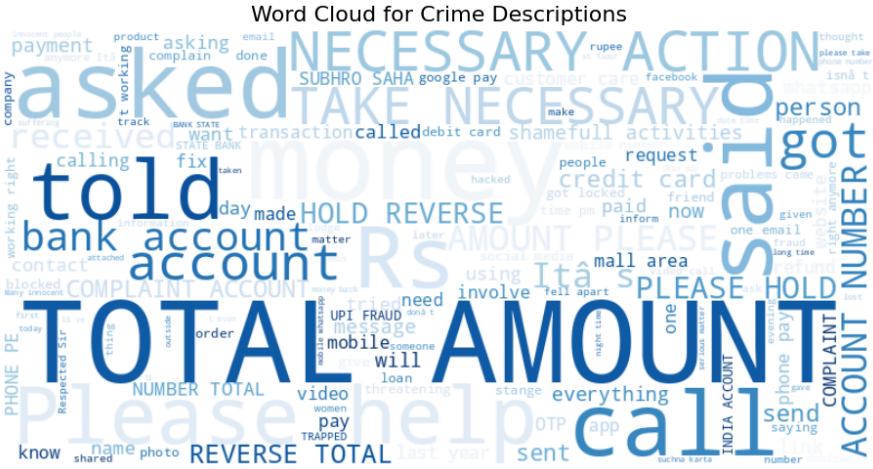
2. Pie Chart for Sub-Category Distribution

A pie chart was created to represent the proportional distribution of complaints across various subcategories. This visualisation provides a clear and intuitive understanding of how complaints are divided among the subcategories, highlighting the dominant ones at a glance.



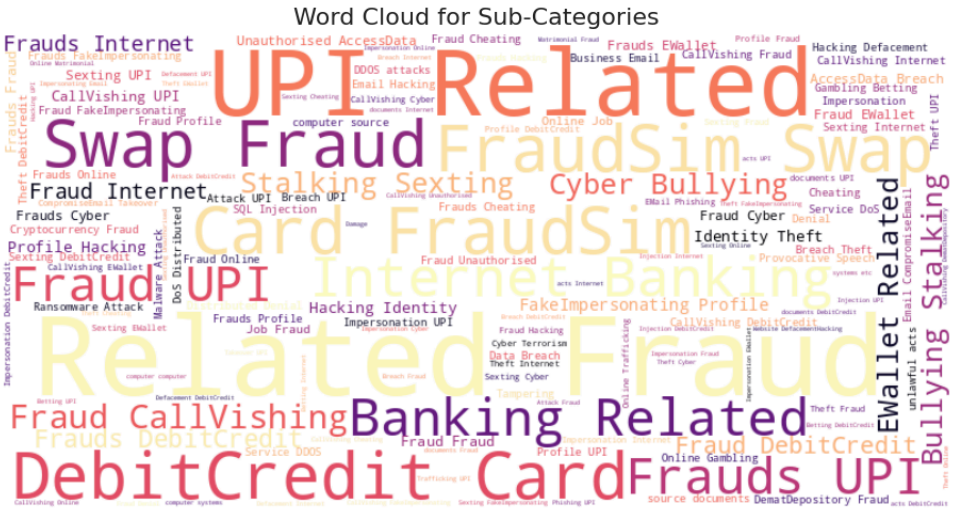
3. Word Cloud for Crime Descriptions

A word cloud was generated to visualise the most frequently occurring words in the crime descriptions. Larger words indicate higher frequencies, providing an immediate and intuitive representation of the most common terms and themes present in the dataset.



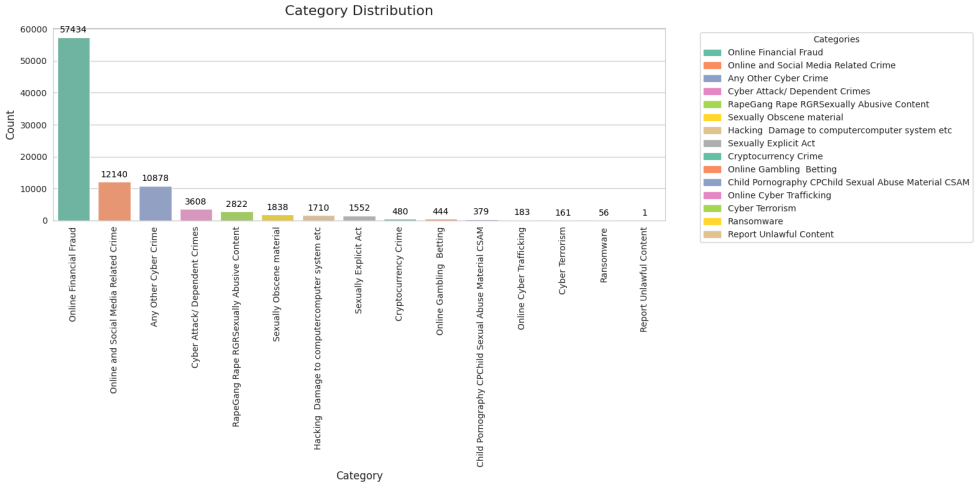
4. Word Cloud for Sub-Category Distribution:

A word cloud was created to showcase the most frequently occurring subcategories within the dataset. The size of each word reflects its frequency, offering a quick and visually engaging way to identify the dominant subcategories.



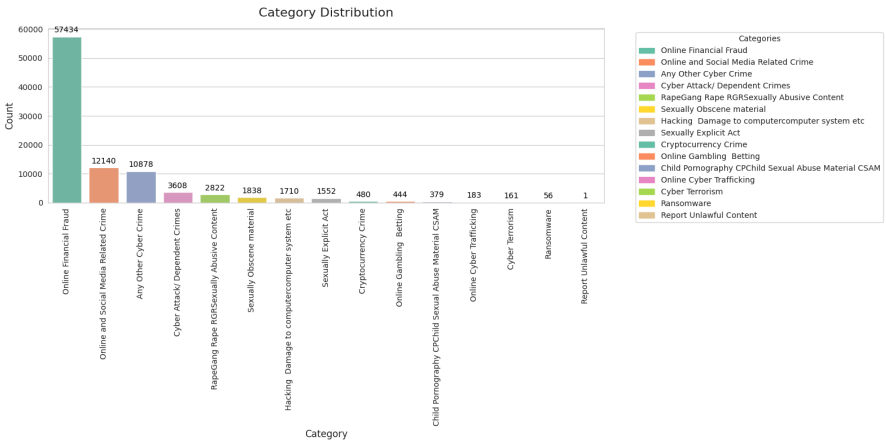
5. Bar Plot for Sub-Category Distribution:

A bar plot was generated to illustrate the frequency of complaints across different subcategories. This visualisation provides a clear and detailed comparison, highlighting the subcategories with the highest and lowest occurrences within the dataset.



6. Bar Plot for Category Distribution:

A bar plot was created to display the frequency of complaints across various categories. This visualisation offers a clear comparison, making it easy to identify the categories with the highest and lowest number of complaints in the dataset.



Text Preprocessing

In this project, we followed a comprehensive text preprocessing pipeline to ensure the quality and consistency of the data before feeding it into the model. The preprocessing steps were crucial in enhancing the model's ability to understand and classify the text accurately.

1. **Data Imputation**: Initially, any missing or incomplete data were handled by imputing values to maintain the integrity of the dataset.

2. **Lowercasing**: To ensure uniformity, all text data were converted to lowercase, eliminating any inconsistencies caused by case sensitivity.

3. **Stop Words Removal**: We removed common stop words—such as "the," "and," "is,"—that do not contribute significant meaning to the classification task.

4. **Punctuation Removal**: Punctuation marks were stripped from the text, as they do not carry essential information for the classification model.

5. **Lemmatization**: Each word was lemmatized to its base form (e.g., "running" to "run") to reduce the number of variations of the same word and improve model efficiency.

6. **Tokenization:** The text was tokenized, breaking it down into individual words or tokens to facilitate further analysis.

7. **POS Tagging**: Part-of-speech (POS) tagging was applied to identify the grammatical structure of the sentences and the role of each word (e.g., noun, verb, adjective), which helped in understanding the context of the complaints.

8. **Named Entity Recognition (NER)**: NER was used to detect and classify proper nouns, such as the names of individuals, organisations, or locations, providing important context for categorising complaints.

By performing these preprocessing steps, we ensured that the data was well-prepared for modelling, with cleaner text that was easier for the BERT model to process and classify. These techniques helped to capture the essential information from the complaints, improving the accuracy of the crime classification task.

Model Development

The goal is to leverage the power of the BERT (Bidirectional Encoder Representations) model, an open-source ML framework for Natural Language Processing, to achieve state-of-the-art results in multiclass text classification over cyber security.

BERT

BERT (Bidirectional Encoder Representations from Transformers) leverages a transformer-based neural network to understand and generate human-like language. BERT employs an encoder-only architecture. In the original Transformer architecture, there are both encoder and decoder modules. The decision to use an encoder-only architecture in BERT suggests a primary emphasis on understanding input sequences rather than generating output sequences.

Bidirectional Approach of BERT

Traditional language models process text sequentially, either from left to right or right to left. This method limits the model's awareness to the immediate context preceding the target word. BERT uses a bi-directional approach considering both the left and right context of words in a sentence, instead of analysing the text sequentially, BERT looks at all the words in a sentence simultaneously.

Pre-Training on Large Data

1. BERT is pre-trained on large amounts of unlabeled text data. The model learns contextual embeddings, which are the representations of words that take into account their surrounding context in a sentence.
2. BERT engages in various unsupervised pre-training tasks. For instance, it might learn to predict missing words in a sentence (Masked Language Model or MLM task), understand the relationship between two sentences, or predict the next sentence in a pair.

Fine-Tuning on Labelled Data

1. After the pre-training phase, the BERT model, armed with its contextual embeddings, is then fine-tuned for specific natural language processing (NLP) tasks. This step tailors the model to more targeted applications by adapting its general language understanding to the nuances of the particular task.
2. BERT is fine-tuned using labelled data specific to the downstream tasks of interest. These tasks could include sentiment analysis, question-answering, named entity recognition, or

any other NLP application. The model's parameters are adjusted to optimise its performance for the particular requirements of the task at hand.

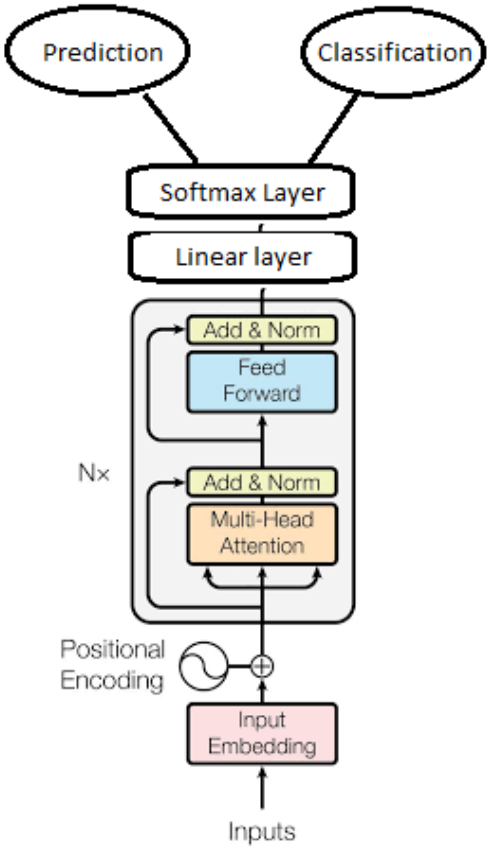


Fig.: Block Diagram representing the BERT model Workflow

BERT is designed to generate a language model so only the encoder mechanism is used. Sequence of tokens are fed to the Transformer encoder. These tokens are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors, each corresponding to an input token, providing contextualised representations.
When training language models, defining a prediction goal is a challenge. Many models predict the next word in a sequence, which is a directional approach and may limit context learning. BERT addresses this challenge with two innovative training strategies:

1. <u>Masked Language Model (MLM)</u>

In BERT's pre-training process, a portion of words in each input sequence is masked and the model is trained to predict the original values of these masked words based on the context provided by the surrounding words.

In simple terms,

- **Masking words:** Before BERT learns from sentences, it hides some words (about 15%) and replaces them with a special symbol, like [MASK].
- **Guessing Hidden Words:** BERT's job is to figure out what these hidden words are by looking at the words around them. It's like a game of guessing where some words are missing, and BERT tries to fill in the blanks.
    a. **How BERT learns:**
        - BERT adds a special layer on top of its learning system to make these guesses. It then checks how close its guesses are to the actual hidden words.
        - It does this by converting its guesses into probabilities, saying, "I think this word is X, and I'm this much sure about it."
    b. **Special Attention to Hidden Words**
        - BERT's main focus during training is on getting these hidden words right. It cares less about predicting the words that are not hidden.
        - This is because the real challenge is figuring out the missing parts, and this strategy helps BERT become really good at understanding the meaning and context of words.

From a Technical Perspective:

I. BERT adds a classification layer on top of the output from the encoder. This layer is crucial for predicting the masked words.
II. The output vectors from the classification layer are multiplied by the embedding matrix, transforming them into the vocabulary dimension. This step helps align the predicted representations with the vocabulary space.
III. The probability of each word in the vocabulary is calculated using the SoftMax activation function. This step generates a probability distribution over the entire vocabulary for each masked position.
IV. The loss function used during training considers only the prediction of the masked values. The model is penalised for the deviation between its predictions and the actual values of the masked words.

V. The model converges slower than directional models. This is because, during training, BERT is only concerned with predicting the masked values, ignoring the prediction of the non-masked words. The increased context awareness achieved through this strategy compensates for the slower convergence.

2. Next Sentence Prediction (NSP)

BERT predicts if the second sentence is connected to the first. This is done by transforming the output of the [CLS] token into a 2×1 shaped vector using a classification layer, and then calculating the probability of whether the second sentence follows the first using SoftMax.

In the training process, BERT learns to understand the relationship between pairs of sentences, predicting if the second sentence follows the first in the original document.

- 50% of the input pairs have the second sentence as the subsequent sentence in the original document, and the other 50% have a randomly chosen sentence.
- To help the model distinguish between connected and disconnected sentence pairs. The input is processed before entering the model:
    a. A [CLS] token is inserted at the beginning of the first sentence, and a [SEP] token is added at the end of each sentence.
    b. A sentence embedding indicating Sentence A or Sentence B is added to each token.
    c. A positional embedding indicates the position of each token in the sequence.
- BERT predicts if the second sentence is connected to the first. This is done by transforming the output of the [CLS] token into a 2×1 shaped vector using a classification layer, and then calculating the probability of whether the second sentence follows the first using SoftMax.

Parameters and Training Arguments for BERT

1. **Output and Checkpoints**
    - **output_dir**: Specifies where the model checkpoints and outputs will be saved.
    - **overwrite_output_dir**: If set to True, any existing files in the output directory will be overwritten.
2. **Training Duration**
    - **num_train_epochs**: Defines the number of complete passes through the training dataset. For example, if this is set to 3, the model will see the entire dataset three times.
    - **save_steps**: Saves the model's checkpoint every specified number of steps. This ensures you can resume training if needed.
3. **Batch Sizes**
    - **per_device_train_batch_size**: The number of training samples processed on each GPU or CPU in a single forward and backward pass.
    - **per_device_eval_batch_size**: Similar to the training batch size but used for evaluation.
4. **Learning and Optimization**
    - **learning_rate**: Sets the initial learning rate for the optimizer. Smaller values generally make the model learn more slowly but more stably.
    - **weight_decay**: Applies L2 regularisation to reduce overfitting.
    - **adam_beta1**, **adam_beta2**, **adam_epsilon**: These control the behaviour of the Adam optimizer, which is commonly used for training deep learning models.
5. **Warmup and Scheduling**
    - **warmup_steps**: Defines the number of steps during which the learning rate linearly increases from 0 to the set learning rate.
    - **lr_scheduler_type**: Specifies the learning rate decay strategy (e.g., linear decay after the warmup phase).

6. **Logging and Evaluation**
    - **logging_dir**: Directory where logs (e.g., TensorBoard logs) will be saved.
    - **logging_steps**: Determines how often training metrics are logged (e.g., loss, accuracy).
    - **evaluation_strategy**: Specifies when to evaluate the model. Options include:
        a. "steps": Evaluate after a fixed number of steps.
        b. "epoch": Evaluate at the end of each epoch.
        c. "no": Disable evaluation.
    - **eval_steps**: If evaluation is done in steps, this defines the interval for evaluation.
7. **Checkpointing**
    - **save_total_limit**: Sets a limit on the number of saved checkpoints to prevent excessive disk usage.
    - **load_best_model_at_end**: Ensures that the best-performing checkpoint (based on the evaluation metric) is loaded at the end of training.
8. **Performance and Precision**
    - **fp16**: Enables mixed-precision training, which can speed up training on compatible GPUs and reduce memory usage.
    - **gradient_accumulation_steps**: Allows the accumulation of gradients over multiple steps to effectively simulate a larger batch size without increasing memory usage.
9. **Reproducibility**
    - **seed**: Ensures the training process is reproducible by setting a random seed.
10. **Best Model Selection**
    - **metric_for_best_model**: Specifies the evaluation metric (e.g., accuracy or F1 score) to determine which checkpoint is the best.
    - **greater_is_better**: Indicates whether higher values of the chosen metric are better.

Tech Stack

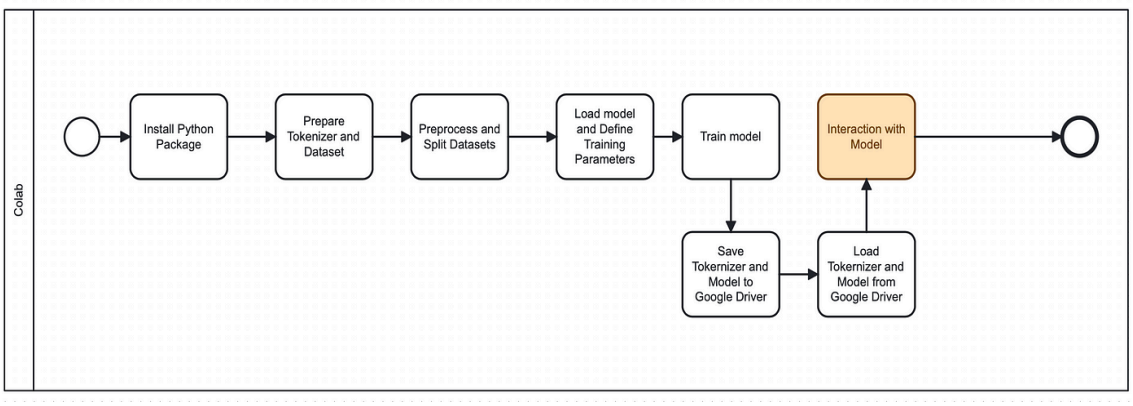
Language: Python
Libraries: pandas, torch, nltk, numpy, pickle, re, tqdm, sklearn, transformers

Prerequisite

Install the torch framework
Understanding of Multiclass Text Classification using Naive Bayes
Familiarity with Skip Gram Model for Word Embeddings
Knowledge of building Multi-Class Text Classification Models with RNN and LSTM
Understanding Text Classification Model with Attention Mechanism in NLP

Approach

- Data Processing
- Read CSV, handle null values, encode labels, and preprocess text.
- Exploratory Data Analysis
- Model Building
- Create BERT model, define dataset, train and test functions.
- Training
- Load data, split, create datasets and loaders.
- Train BERT model on GPU.
- Predictions
- Make predictions on new text data.



Model finetuning and testing approach

Performance Metrics used for Evaluation

1. Accuracy

- **Definition**: Accuracy measures the proportion of correctly predicted instances out of the total instances.
  Accuracy=Total Number of Predictions/Number of Correct Predictions
- **Pros**:
  - Easy to interpret.
  - Useful when the dataset is balanced (i.e., classes have similar representation).
- **Cons**:
  - Can be misleading in imbalanced datasets. For example, in a dataset where 95% of samples belong to one class, a model predicting only the majority class achieves 95% accuracy but fails on the minority class.

2. Precision

- **Definition**: Precision measures the proportion of true positive predictions out of all positive predictions made by the model.
  Precision=True Positives (TP)/True Positives (TP)+False Positives (FP)
- **Interpretation**:
  - Precision focuses on the correctness of positive predictions.
  - A high precision means the model avoids false positives, making it crucial for tasks like spam detection, where predicting a non-spam email as spam (false positive) is undesirable.

3. Recall (Sensitivity or True Positive Rate)

- **Definition**: Recall measures the proportion of true positive predictions out of all actual positive instances.
  Recall=True Positives (TP)/True Positives (TP)+False Negatives (FN)
- **Interpretation**:
  - Recall emphasises capturing all positive instances.
  - High recall is important for scenarios like medical diagnosis, where missing a true positive (e.g., detecting a disease) can have severe consequences.

4. F1 Score

- **Definition**: The F1 Score is the harmonic mean of Precision and Recall, providing a single metric that balances the two.
- **Interpretation**:

    - A good choice when you need a balance between Precision and Recall.
    - Especially useful when the class distribution is imbalanced.

5. Classification Report

The classification report is a summary of key metrics, typically provided for each class in a classification task. It includes:

- **Precision**: For each class.
- **Recall**: For each class.
- **F1 Score**: For each class.

Key Metrics:

1. **Macro Average**:
    - Averages Precision, Recall, and F1 Score equally across all classes, regardless of their support.
    - Useful when all classes are equally important.
2. **Weighted Average**:
    - Averages the metrics by considering the support of each class.
    - Better reflects the performance on imbalanced datasets.

Project Structure

Input: complaints.csv
Output: bert_pre_trained.pth, label_encoder.pkl, labels.pkl, tokens.pkl
Source: model.py, data.py, utils.py
Files: Engine.py, bert.ipynb, processing.py, predict.py, README.md, requirements.txt

Evaluation and Results

The performance metrics used for evaluation are accuracy, recall, precision as well as an overall classification report.

Hence our accuracy comes out to be 100%, with 67% precision and 71% recall.