**Midterm Project:**

**"Building a Deep Learning Classifier for Identifying LLM Usage in Input Pairs"**

To implement this project, the goal is to build a deep learning classifier that can predict which Large Language Model (LLM) was used to complete a given text. Below is a step-by-step approach:

**1. Data Preparation and Collection**

We need pairs of input texts (x\_i) and their completions (x\_j) generated by different LLMs.

For example:

X\_i = "Yesterday I went"

X\_j\_1 = "to Costco." (LLM1)

X\_j\_2 = "to Walmart and bought some apple." (LLM2)

Then, for each pair (x\_i, x\_j), we label it with the corresponding LLM (e.g., LLM1, LLM2, etc.). I used multiple LLMs to generate completions for each truncated text (x\_i). The following is the lists of used LLMs:

|  |  |
| --- | --- |
| **Open-Source Platform** | **Model** |
| 1. Google | "gemini-1.5-flash" |
| 1. Google | "text-bison-001" |
| 1. Google | "bert-base-uncased" |
| 1. OpenAI API | "gpt-3.5-turbo" |
| 1. Meta | "llama3" |
| 1. Microsoft | "microsoft/phi-3" |
| 1. Amazon | "titan-3b" |
| 1. Apple | "apple/OpenELM-3B-Instruct" |
| 1. Bloom | "bigscience/bloom-1b3" |
| 1. Falcon | "tiiuae/falcon-40b" |
| 1. Cohere | "command-r-plus-08-2024" |
| 1. Mistral AI | "mistral-large-latest" |
| 1. AI21 Labs | "jamba-instruct-preview" |
| 1. CMU&Princeton | "state-spaces/mamba-2.8b" |
| 1. Alibaba | "qwen1.5-110b-chat" |
| 1. Shanghai AI Laboratory | " internlm/internlm2-chat-7b" |
| 1. DataBricks | " mosaicml/mpt-7b-instruct" |
| 1. BigCode | "bigcode/starcoder" |
| 1. Stability AI | "stabilityai/stablelm-2-12b-chat" |
| 1. Zhipu AI | "glm-4" |

Various LLMs services type such as:

1. Open-Source
2. APIs/AI Model
3. Developer Communities and Forums:GitHub, Stack Overflow, Kaldi Users Forum

We tried to use opensource APIs. LLM\_functions.py contains all implementation and API calls from the mentioned LLM models and APIs.

**2. Feature Extraction**

For each pair (x\_i, x\_j), we consider the following:

* Semantic Similarity: Compute the similarity between the truncated text (x\_i) and the completion (x\_j). TF-IDF (Term Frequency-Inverse Document Frequency) converts text into a vector representation, where each word gets a weight based on its frequency and importance in the document. Cosine similarity measures the angle between the vectors of xi and xj, which can indicate how similar they are semantically. We can also use word embeddings to capture the meaning of the sentences, using models like BERT. For instance, BERT provides a contextual embedding for each word, which can be averaged to form a vector representing the entire sentence.
* Style Features: Extract syntactic features like sentence structure, tone, word choice, and complex\_ity to differentiate between LLMs. The stylistic features capture the writing patterns of different LLMs. We can calculate the number of words or characters in the sentence as a feature. We can calculate the average word length or count the number of complex words (longer than 6 characters, for example). Different LLMs might favor different parts of speech (e.g., more adjectives, fewer adverbs). We can use **spaCy** to tag the text. N-gram looks at common n-grams (sequences of n words) used by different LLMs. We can extract and compare n-grams to identify patterns in LLM-generated text. We can extract unigrams (single words), bigrams (pairs of words), and trigrams (triples of words) from each text using the nltk library.
* N-gram analysis Analyze common phrases or words each LLM might use more frequently.

**3. Model Selection**

As mentioned, each input will be a tuple (x\_i, x\_j) representing the truncated text and its completion.

We used Transformer Models BERT to encode the text pairs.

Inputs are text pairs (xi, xj), representing the truncated text and the completion generated by an LLM. Our objective is to classify which LLM generated the completion. We need to feed these text pairs into a model, which can be done by tokenizing them and passing them as inputs for deep learning models.classifier.py contains all the implementation of our deep classifier.

* Input Processing: we tokenize both x\_i and x\_j and pass them as input sequences to the model.
* Embedding Layer: Convert tokens to embeddings using a pre-trained embedding layer (BERT).
* Fully Connected Layer: A dense layer to predict which LLM generated the completion.

**4. Training the Classifier**

We Define a supervised learning task where the input is the pair (x\_i, x\_j) and the label is the LLM that generated the completion.

For loss function, “cross-entropy loss” is used since this is a multi-class classification problem.

Regarding evaluation metrics,accuracy, precision, recall, and F1 score is used to evaluate the performance of the classifier.

**5. Evaluate and Analysis**

After training, we evaluate the model on a test set. Analyzing the features is important to see what kinds of completions each LLM tends to produce. To evaluate the performance of our model, we need to split our dataset into training and testing sets. The test set will be used to assess how well our model generalizes to unseen data.

Training Set: The data used to train our model.

Test Set: The data used to evaluate model performance after training.

We can use an 80/20 or 70/30 split, depending on our dataset size. We use train\_test\_split from scikit-learn.

Once the model is trained, evaluate it on the test set using common classification metrics.

Metrics to Use:

* **Accuracy:** The percentage of correct predictions.
* **Precision, Recall, F1-Score:** These metrics give a better picture for multi-class classification, especially if your classes (LLMs) are imbalanced.
* **Confusion Matrix:** Shows where your model is confusing one LLM with another. A confusion matrix helps you see which LLMs are often confused with one another. It’s a table that compares actual labels vs. predicted labels.

Now that we have a trained model and evaluation results, it’s important to understand what features the model relies on when making predictions. We can visualize which tokens or sequences in the text pairs are contributing most to the model’s decision in Transformer Models (BERT). Shapley Additive explanations (SHAP) values explain the output of machine learning models by showing how much each feature contributes to a prediction. Local Interpretable Model-agnostic Explanations (LIME) explains predictions by generating local approximations of the model’s behavior for a specific input. For transformer models like BERT, we can visualize the attention scores that BERT assigns to each token. We can also use libraries like transformers-interpr or bertviz to visualize which parts of xi and xj the model is paying attention to.

**Analyzing LLM Behavior**

Once we have feature importance results, we can start analyzing the behavior of different LLMs:

* Common Patterns: Does one LLM generate longer completions? More formal language? Unique phrases?
* Semantic Differences: Do certain LLMs tend to produce more semantically rich or diverse completions?
* Style Features: Which stylistic features (sentence structure, tone, etc.) seem to be distinctive for specific LLMs?

**6. Hyperparameter Tuning**

Different hyperparameters like learning rate, batch size, model architecture, and embedding dimensions to optimize performance were used.