Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia Informática

Data Mining

Pratical Assignment (23/24)

João Paulo Machado Abreu

pg53928

João Pedro Dias Faria

pg53939

Ricardo Cardoso Sousa

pg54179

15th June 2024

Contents

Chapter 1

Introduction

The goal of this project is to develop a web application that uses a Fine tuned BERT model to detect whether a text was generated by a Large Language Model (LLM) and, if so, identifying which specific LLM authored the text. The targeted LLMs for this application are:

- GPT-4 (OpenAI)
- Meta-Llama-3-8B (Meta)
- Phi-3-mini-128k-instruct (Microsoft)
- Mixtral-8x7B-Instruct-v0.1 (Mistral AI)

While several existing websites can detect AI-generated text, there is currently no solution that can accurately differentiate which LLM authored a given text. Current tools, such as QuillBot, GPTZero, and Scribbr, focus on identifying AI-generated text but do not specify the LLM responsible. This project aims to fill this gap by developing a specialized tool that distinguishes between texts generated by different LLMs.

Chapter 2

Methodology

2.1 Data Collection

To complete this project, we needed to obtain text generated by each of the aforementioned LLMs. Acquiring a dataset with specific characteristics proved challenging, so we decided to initially focus on a widely used LLM, GPT-4, whose data acquisition is notably costly. On the Kaggle platform, we collected a dataset containing a large number of questions posed to GPT-4 along with its respective answers.

We extracted the first 30,000 questions from this dataset, along with all the GPT-4 answers to these questions, and stored them in JSON format. For the other LLMs used in this project, we submitted the same set of questions using the Hugging Face Serverless Inference API and stored their responses in JSON format as well.

2.2 Data Cleaning

After obtaining the responses from all the models, we needed to perform some data cleaning. Since the BERT model can only take 512 input tokens, we had to constrain the responses to that limit. Consequently, some responses became incomplete, so we truncated them at the end of the last complete sentence. We also removed code-related questions and responses with fewer than 80 characters, as such short texts would be difficult for the model to differentiate between the LLMs.

2.3 Fine-Tuning

After getting the data cleaned and ready, we decided to fine-tune the BERT model to the specific task of differentiating which LLM wrote the text. In order to do that, we used the Python programming language with the PyTorch library.

2.3.1 Merge of the datasets

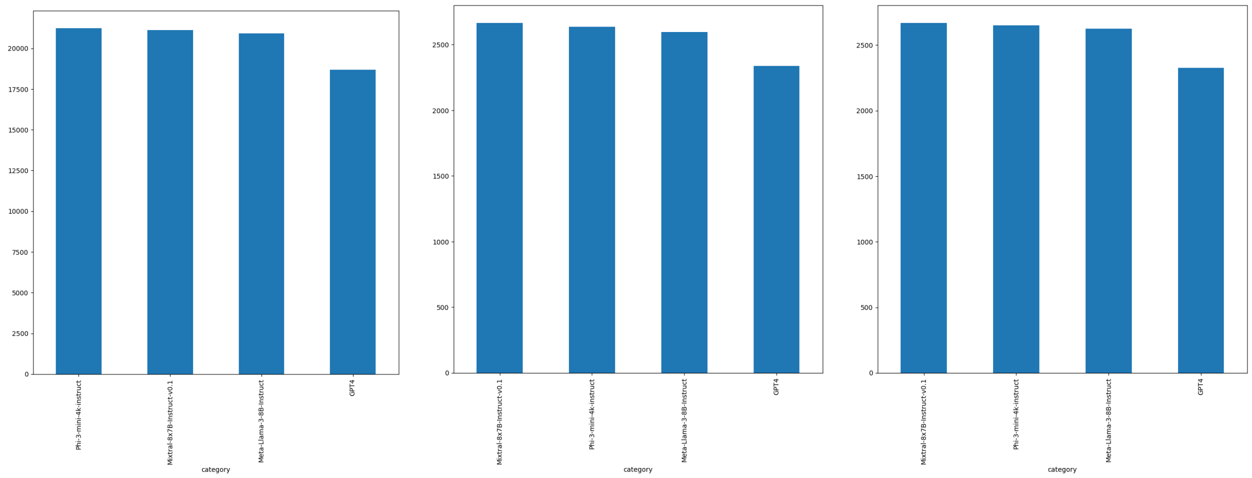
First we loaded the four datasets to a pandas DataFrame and assigned a numerical value to the labels, to make sure that BERT can classify the given problem. Label 0 was assigned to the Meta-Llama model, label 1 to the Phi-3, label 2 to the Mixtral and label 3 to GPT4. Then, concatenated them all together, which resulted in 102502 entries and 5 columns.

```
def load_and_process_data(file_path, label, category):
    with open(file_path) as f:
        data = json.load(f)
        for e in data:
            e['text'] = e.pop('Completion')
            e['labels'] = label
            e['category'] = category
    return pd.DataFrame(data)

df_llama = load_and_process_data('meta-llama/Meta-Llama-3-8B-Instruct_valid.json',
     0, 'Meta-Llama-3-8B-Instruct')
df_phi3 = load_and_process_data('microsoft/Phi-3-mini-4k-instruct_valid.json', 1,
    'Phi-3-mini-4k-instruct')
df_mixtral = load_and_process_data('mistralai/Mixtral-8x7B-Instruct-v0.1_valid.
    json', 2, 'Mixtral-8x7B-Instruct-v0.1')
df_gpt4 = load_and_process_data('openai/GPT4_valid.json', 3, 'GPT4')

df_combined = pd.concat([df_llama, df_phi3, df_mixtral, df_gpt4], ignore_index=
    True)
```

2.3.2 Train, Test and Validation Split

Since we applied the same set of questions to all the models, we decided to split the data by question ID. This ensures that the model is trained with responses to a different set of questions than those used in the training and validation phases.

```
unique_ids = set(df_combined['id'])
unique_ids = list(unique_ids)

train_size = 0.8
train_ids, test_eval_ids = train_test_split(unique_ids, train_size=train_size,
    random_state=42)

mask = df_combined['id'].apply(lambda x: x in train_ids)
df_train = df_combined[mask]

test_size = 0.5
test_ids, eval_ids = train_test_split(test_eval_ids, test_size=test_size,
    random_state=42)

mask = df_combined['id'].apply(lambda x: x in test_ids)
df_test = df_combined[mask]

mask = df_combined['id'].apply(lambda x: x in eval_ids)
df_eval = df_combined[mask]
```



(a) Training Set (b) Test Set (c) Validation Set

Figure 2.1: Distribution of data across different models.

So, 80% of the questions were assigned to training, 10% to test and 10% for evaluation(validation).

The training set had 81991 lines, the test set had 10240 lines and the validation set had 10271 lines. Images 2.1a, 2.1b and 2.1c show the data distribution among all labels(LLMs) in the train, test, and validation sets, respectively. As we can see, all of the sets had the data well distributed.

2.3.3 Tokenizing the text

Now that our data is distributed across the three sets, we need to tokenize the input text to pass it to the BERT model. First, we dropped columns that were not necessary for the training phase from the DataFrames. Then, we converted the pandas DataFrames into datasets using the Datasets library, which PyTorch uses for model training.

```
df_train.drop(columns=['id', 'Prompt'], inplace=True)
df_test.drop(columns=['id', 'Prompt'], inplace=True)
df_eval.drop(columns=['id', 'Prompt'], inplace=True)

model_id = "bert-base-uncased"
tokenizer = BertTokenizerFast.from_pretrained(model_id)

train_dataset = Dataset.from_pandas(df_train)
eval_dataset = Dataset.from_pandas(df_eval)
test_dataset = Dataset.from_pandas(df_test)

def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True, max_length =
    512)

train_dataset = train_dataset.map(tokenize, batched=True, batch_size=len(
    train_dataset))
```

```
eval_dataset = eval_dataset.map(tokenize, batched=True, batch_size=len(
    eval_dataset))
test_dataset = test_dataset.map(tokenize, batched=True, batch_size=len(
    test_dataset))
```

After that, we imported the tokenizer from the bert-base-uncased model. For each response or text, we applied the tokenizer using padding and truncation techniques, limiting the max_length to 512, which is the maximum length supported by the BERT model.

2.3.4 Training

In the training phase, first we loaded the model to the GPU device.

```
device = 'cuda' if cuda.is_available() else 'cpu'
model = BertForSequenceClassification.from_pretrained(model_id, num_labels=
    NUM_LABELS, id2label=id2label, label2id=label2id)
model.to(device)
```

Then, defined the training arguments and the metrics for our model to compute.

```
output_dir = 'logs/bert-base-uncased-llm-classificator'

training_args = TrainingArguments(
    output_dir= output_dir,
    do_train=True,
    do_eval=True,
    num_train_epochs=3,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    warmup_steps=100,
    weight_decay=0.01,
    logging_strategy='steps',
    logging_dir=f"{output_dir}/logs",
    logging_steps=50,
    evaluation_strategy="steps",
    eval_steps=50,
    save_strategy="steps",
    load_best_model_at_end=True
)

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)

    macro_precision, macro_recall, macro_f1, _ = precision_recall_fscore_support(
    labels, preds, average='macro')
    class_precision, class_recall, class_f1, _ = precision_recall_fscore_support(
    labels, preds, average=None)

    acc = accuracy_score(labels, preds)

    metrics = {
        'Accuracy': acc,
        'Macro_F1': macro_f1,
        'Macro_Precision': macro_precision,
        'Macro_Recall': macro_recall,
        'Class_Precision': class_precision.tolist(),
        'Class_Recall': class_recall.tolist(),
        'Class_F1': class_f1.tolist()
    }

    return metrics

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    compute_metrics= compute_metrics
)

trainer.train()
```

To note that we did training and evaluation simultaneously. We chose to do 3 epochs because we had a lot of data and in the end, 3 epochs proved to be just enough to train the model for our use case.

For evaluating our model, we selected several metrics including accuracy, macro F1 score, macro precision, and macro recall. Additionally, we computed precision, recall, and F1 score individually for each label to assess the model's performance across all labels. This approach allows us to evaluate:

- **Accuracy:** Overall correctness of predictions.
- **Macro Recall:** Computes the average recall across all classes.
- **Macro Precision:** Computes the average precision across all classes.
- **Macro F1 Score:** Harmonic mean of precision and recall across all labels, giving equal weight to each label.

Calculating precision, recall, and F1 score individually for each label ensures that we can evaluate how well the model distinguishes and classifies each specific label. This approach provides a detailed understanding of the model's classification performance across all categories.

2.4 Results

As we can see in Table 2.1, our model obtained an accuracy of 73.73%, with consistent macro metrics across precision, recall, and F1-score, indicating balanced performance across all classes.

Accuracy	Macro Recall	Macro Precision	Macro F1
73.73%	73.52%	74.47%	73.48%

Table 2.1: Overall Test Results

Table 2.2 shows specific results for the class Meta-Llama. It achieved a high recall of 88.14%, indicating that the model correctly identified 88.14% of all instances belonging to Meta-Llama. However, precision is slightly lower at 71.55%, suggesting some instances not belonging to Meta-Llama were incorrectly classified as such.

Recall	Precision	F1
88.14%	71.55%	78.98%

Table 2.2: Test Results for Meta-Llama

Table 2.3 presents results for the class Phi-3. It shows balanced precision and recall scores around 77%, resulting in a harmonious F1-score of 76.89%.

Recall	Precision	F1
77.09%	76.68%	76.89%

Table 2.3: Test Results for Phi-3

Table 2.4 displays results for the class Mixtral. While the recall is 65.09%, indicating moderate success in identifying Mixtral instances, precision at 67.39% implies a notable number of false positives.

Recall	Precision	F1
65.09%	67.39%	66.22%

Table 2.4: Test Results for Mixtral

Finally, Table 2.5 outlines results for the class GPT4. It exhibits a high precision of 82.24%, indicating few instances not belonging to GPT4 were incorrectly classified as such. However, the lower recall at 63.78% suggests that some instances of GPT4 were missed.

Recall	Precision	F1
63.78%	82.24%	71.85%

Table 2.5: Test Results for GPT4

Observing Figure 2.2, we can conclude that GPT4 poses the greatest challenge for our Fine-Tuned BERT model in classification. It is frequently confused with Mixtral and Phi-3, accounting for 28% of misclassifications. Mixtral also exhibits sub-optimal classification, often being misidentified as Meta-Llama 19% of the time. In contrast, both Phi-3 and Meta-Llama are consistently well-classified.

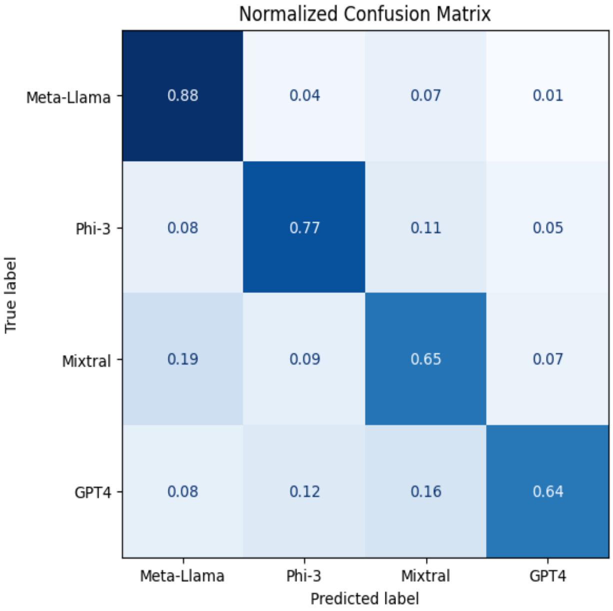


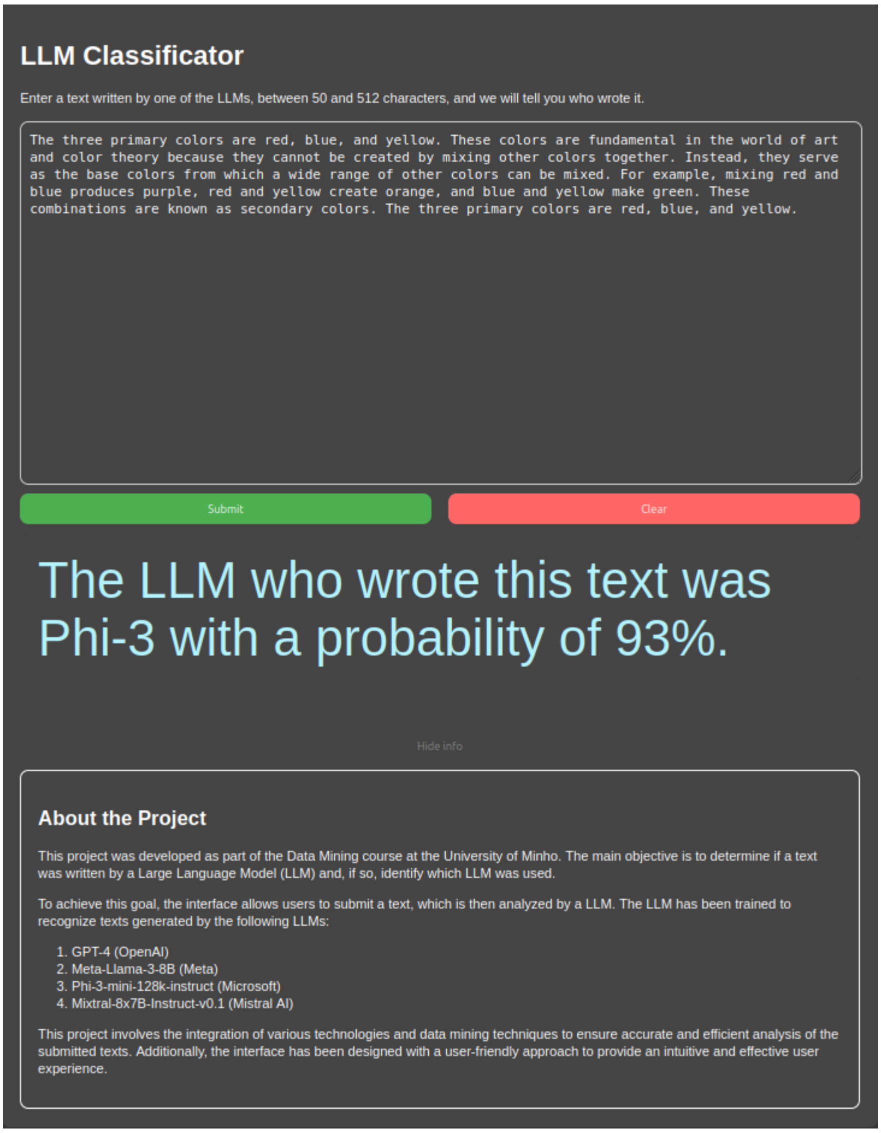
Figure 2.2: Confusion Matrix

2.5 Publication

After training and evaluating, we published our model on the Hugging Face Hub [model] and it can now be accessed via the Hugging Face Serverless Inference API or downloaded for local use.

2.6 Web Application

To create the front-end of our web application, we used the Flask Python library. The back-end was straightforward to implement as well. We simply needed to load the model and tokenizer from HuggingFace and create a function that applies the user-inputted text to the model, returning its response. Additionally, we implemented some validation for the input text. We do not accept texts with fewer than 50 characters, as they are too short for the model to make accurate predictions, and we do not accept texts with more than 512 characters, as this exceeds the model's input capacity. Furthermore, if the model's response has a confidence score below 30%, we display a message to the user indicating that the model could not determine the author of the text. Below we present an image of the application.



LLM Classificator

Enter a text written by one of the LLMs, between 50 and 512 characters, and we will tell you who wrote it.

```
The three primary colors are red, blue, and yellow. These colors are fundamental in the world of art
and color theory because they cannot be created by mixing other colors together. Instead, they serve
as the base colors from which a wide range of other colors can be mixed. For example, mixing red and
blue produces purple, red and yellow create orange, and blue and yellow make green. These
combinations are known as secondary colors. The three primary colors are red, blue, and yellow.
```

The LLM who wrote this text was Phi-3 with a probability of 93%.

About the Project

This project was developed as part of the Data Mining course at the University of Minho. The main objective is to determine if a text was written by a Large Language Model (LLM) and, if so, identify which LLM was used.

To achieve this goal, the interface allows users to submit a text, which is then analyzed by a LLM. The LLM has been trained to recognize texts generated by the following LLMs:

1. GPT-4 (OpenAI)
2. Meta-Llama-3-8B (Meta)
3. Phi-3-mini-128k-instruct (Microsoft)
4. Mixtral-8x7B-Instruct-v0.1 (Mistral AI)

This project involves the integration of various technologies and data mining techniques to ensure accurate and efficient analysis of the submitted texts. Additionally, the interface has been designed with a user-friendly approach to provide an intuitive and effective user experience.

Figure 2.3: Example Image of the web application

Chapter 3

Conclusion

In this project, we successfully built and fine-tuned a BERT model to differentiate between texts generated by various large language models (LLMs), including Meta-Llama, Phi-3, Mixtral, and GPT-4. We meticulously collected, cleaned, and preprocessed data, ensuring the model was trained on high-quality inputs. The data collection phase involved obtaining responses from multiple LLMs for a set of standardized questions, which was crucial for ensuring consistency across our dataset.

Data cleaning was a critical step, where we truncated responses to meet BERT's input token limit and removed uninformative texts. The fine-tuning phase involved carefully splitting the data into training, test, and validation sets, followed by tokenizing the text and training the BERT model with balanced evaluation metrics to assess its performance comprehensively.

Our results demonstrated that the fine-tuned BERT model achieved a commendable accuracy of 73.73%, with balanced macro precision, recall, and F1 scores. Detailed analysis of the results showed that the model performed consistently well across most classes, although it faced challenges in correctly classifying GPT-4 outputs, often confusing them with Mixtral and Phi-3. The confusion matrix highlighted areas for potential improvement, particularly in distinguishing between certain models.

To make our model accessible, we published it on the Hugging Face Hub, allowing others to use it via the Hugging Face Serverless Inference API or download it for local use. Additionally, we developed a web application using Flask to provide a user-friendly interface for model interaction. The application includes input validation to ensure text length falls within acceptable ranges and checks the model's confidence level before displaying results.

In conclusion, this project showcases the feasibility of using fine-tuned transformer models like BERT to differentiate between texts generated by various LLMs. The comprehensive methodology—from data collection to model deployment—demonstrates a robust approach to developing and deploying machine learning models in practical applications. Future work could focus on enhancing the model's accuracy, particularly for challenging cases, and expanding the dataset to include more LLMs and diverse text types.