Project 3 - NLP

Introduction

In this project, you will learn about several use cases in NLP, such as text classification,text summarization and information retrieval. You will adjust an existing code, download datasets and learn about HuggingFace.

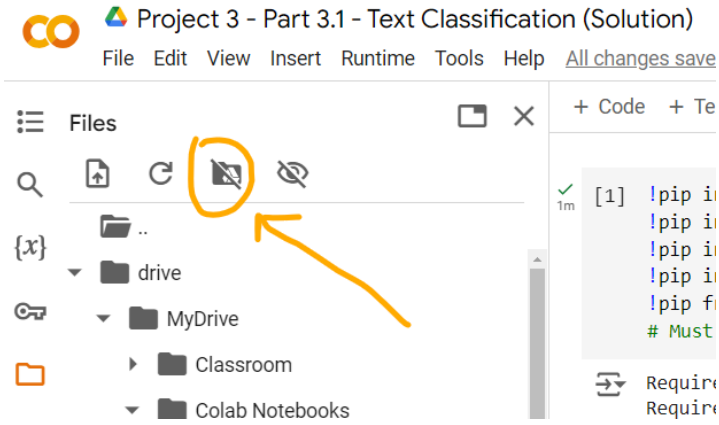
Installation

For part 1, you will use Google Collab to utilize the GPU.
For parts 2 and 3, create a new Anaconda environment with Python 3.8.19. Use pip to install all required libraries from requirements.txt. **Note**: Each part has its own requirements.

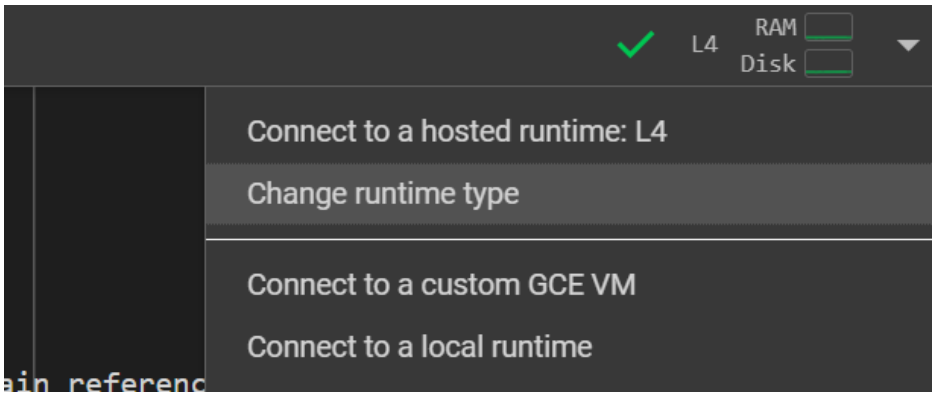
Part 1 - Text Classification

Instructions:

1. Download the emotion analysis dataset from Kaggle. Put it into your local Google drive in `"Colab Notebooks\emotion_sentiment_dataset.csv"`
2. Mount Google Drive to start working with the dataset csv file.



3. The dataset contains a column called ‘text’ which is the input, and a tagged emotion (ground truth) which is required to be predicted. Use the Google Collab code as a starting point for the project.
4. Make sure to change the runtime to GPU (Make sure to do this **before** running the first cell)



5. Uncomment the first cell, run it once to install all needed libraries. No need to run it again while being connected to the same environment.

1.1 Take a look at the emotions column. Draw a pie plot using matplotlib of all the possible emotions. Add the plot to your answer sheet.

1.2 Explain why precision and recall metrics are better than accuracy for measuring the success of a classification algorithm for this dataset.

We will now classify all emotions as neutral vs non-neutral. The given code uses a DistilBert model, which is a small transformer. It fine-tunes a Distilbert model on the given dataset.

1.3 Train the model and calculate the precision and recall, and find the confusion matrix. What can you say about the performance of the model?Based on the validation set results, would you recommend an early stopping?

Advanced Section - Extra Points

1.4 Suppose that we wanted to detect only “surprise” emotion vs all the others. Calculate the percentage of “surprise” emotion. Propose a good strategy to train such a classifier, and measure its performance.

1.5 Let’s try to create a classification of 3 classes - neutral, positive, negative. For example “happiness” would be positive, but “sadness” would be negative. How can you easily adjust the ground truth? **Hint**: you can use Google Gemini during the writing phase of the code.

1.6 Adjust the code and train the classifier for the 3 class classification problem. What is the confusion matrix of the classifier you trained?

Part 2 - Text summarization

In this exercise, you will download the cnn dailymail dataset. It contains various articles from CNN that were manually summarized to a short sentence. Unlike classification, there is no simple objective metric for measuring success in text summarization. The dataset contains the `article` and `highlights` columns. The first one is the text that needs to be summarized and the second one is the summarization.

We will use only the first 1000 rows for faster runtime. We will use the `datasets` pip package, which is part of HuggingFace. It is an easy and convenient way to get a dataset without downloading it manually. The following line of code

```
dataset = load_dataset("cnn_dailymail", '3.0.0')
```

will download the dataset to the local disk, if it does not exist.

2.1 Create two new columns in the dataframe, called `article_len` and `highlights_len` one will include the length of the `article` column and one the length of the `highlights` column .

2.2 Draw a histogram of length of `article` column, and a histogram of length of `highlights` columns. Make sure to plot them one next to each other, and that it is easily seen that the `highlights` columns are much shorter on average.
Put the plots in the answer sheet.

2.3 Implement the Rouge-N metric in the code and calculate the score of the ground truth. Do not use an existing library, instead implement it by yourself. For the purpose of this exercise, it does not have to be very efficient.
Use **n=1** and **n=2**. Find the highest and the smallest scores in the dataset for both of them. Write them down in the answer sheet and analyze the example for **n=2** with the smallest score. Explain why it has such a low score.

2.4 Now use a summarization pipeline to summarize the text. Use the `"t5-small"` Model. Fill in the code that creates the pipeline. Calculate the rouge-2 score of the first 10 entries. Write down the scores in the answer sheet
Is there an instance where the score is lower than on the ground truth?

Advanced Section - Extra Points

2.5 Rouge-N metric is objective, yet it has several drawbacks. Explain in short, in your own words, how subjective metrics can be used to measure summarization. Propose a specific strategy if you have 100 people who can be used for giving subjective feedback. Explain how to quantize the results into a single number.

Part 3 - Information Retrieval

In this exercise, you will implement a system that retrieves information from a dataset using embeddings. Embeddings can often be used to find texts that have similar meanings or even answers to questions. We will use the DistilBERT model to generate embeddings for both the query text and articles from a dataset. The task is to find the most relevant article that matches the query based on cosine similarity, a common metric for comparing vectorized representations of text.

3.1 Fill in the code section of `find_most_relevant_article`. It should get as input the embedding of the query, the dataset and maximal number of articles. Assume that if `max_num_of_articles` is None, all rows are tested. Otherwise, only the first `max_num_of_articles` are being compared to.
Hints: Use `compute_embedding` and `cosine_similarity`

3.2 What is the most similar article to the following (From the first 1000 rows):

A. Leonardo DiCaprio
B. France
C. Python
D. Deep Learning

Change the code, find the answers and write them down.

Advanced Section - Extra Points

3.3 The code is currently working only on the first 1000 rows from the dataset, in order to run quickly. Let's make it more efficient and scalable, more similar to a real life system.
Design an architecture that

- Precomputes the embeddings on a large dataset, by using multiple threads/processes/machines with GPUs. How would you store the results?
- Finds the most similar embedding using the precomputed embeddings. Think how to use multiple threads/processes/machines.
- Allows addition and deletion of rows in the dataset.

Explain in words and diagrams your system design.

Submission:

Provide the results, including the plots and your explanations in a Google Doc. Use the template provided with the exercise. Also provide your final Python code.

Citations: