Niyomukiza Thamar

Week 4 challenge

Prompt Engineering

Github link

Introduction

The world is going through a revolution in art (DALL-E, MidJourney, Imagine, etc.), science (AlphaFold), medicine, and other key areas, and this approach is playing a role in this revolution. However, those models are complex and require so much in terms of cost and skills to be trained and produce accurate results. Only Big companies are able to train them using billions of parameters because they can afford the resources needed.

Having shown good impact on society and the business in general, knowing how to use giant AI models for multiple cases in business and social problems is vital. It is predicted that more organizations are going to start centering their businesses on LLMs and similar products such as DALL-E 2, MidJourney, Bloom, etc. Therefore,specialized people in prompt engineering demand will grow fast. It is time to start getting familiar with these technologies. This article will introduce basics to get started with using LLMs by leveraging Cohere APIs. More about Cohere LLMs classification example here.

The content of this article is :

e. Recommendation and outcomes
f. Limitations of your analysis and future work
g. References.

Before starting it is worth understanding what word embedding is because it is the foundation of text analysis. As we all know, computers understand zeros and ones, but in this case, we want to work on text for us to perform machine learning tasks. Consequently, that pushes us to think and come up with ways to represent text or strings into numbers. Moreover, machine learning models accept input vectors, specifically arrays of numbers to process them into outputs. This can only be done by converting strings into tokens. Then convert tokens into arrays of numbers before feeding them into a model.

Word embeddings

A **word embedding** is a learned way of representing text into numerical vectors where words that have the same meaning have a similar representation. Individual words are represented as real-valued vectors in a predefined vector space. Word embeddings provide a way to use an efficient, dense representation in which similar words have a similar encoding. They are not computed manually, and weights are learned by the model during training, in the same way a model learns weights for a dense layer. When working on a natural language processing project, we have more than one type of word embedding techniques that we can use namely:

- Embedded layer
- Word2Vec
- Glove

In-context Learning

- In-context learning is an emerging behavior in large language models (LMs) where the LM performs a task just by conditioning on input-output examples, without optimizing any parameters.
- Vanilla methods for training large language models such as GPT-3 and BERT require the model to be pre-trained with unlabeled data and then fine-tuned for specific tasks with labeled data.
- But, using in-context learning, users are able to build models for a new use case in no time and they do not have to worry about fine-tuning and storing new parameters for each task. It only requires very few training cases to get the first model working.
- For in-context learning , a Language Model is given a prompt that contains a list of input and output pairs.

- Towards the end of the prompt, a test input is added to guide the LM the kind of prediction that it has to make. This is done by conditioning the prompt and predicting the next texts..

Fine-tuning

- Fine-tuning is one approach to transfer learning where you change the model output to fit the new task and train only the output model. In contrast, prompt-based learning models can autonomously tune themselves for different tasks by transferring domain knowledge introduced through prompts. .
- During the fine-tuning process, we add a task-specific layer to the PLMs(Pre Trained Language Models) and carry out the usual backpropagation method using a suitable loss function.
- Although task-specific fine-tuning is a relatively cheap task (few dollars) for models like BERT with a few hundred million parameters, it becomes quite expensive for large GPT-like models which have several billion parameters.

Few-shot, One-shot, and Zero-shot Learning.

Few-shot learning

- Few-shot learning (FSL), also referred to as low-shot learning (LSL) in a few sources, is a type of machine learning method where the training dataset contains limited information.
- Important because it helps companies reduce cost, time, computation, data management, and analysis.
- Few-shot learning is a test base where computers are expected to learn from a few examples like humans.
- By using few-shot learning, machines can learn rare cases. For example, when classifying images of animals, a machine learning model trained with few-shot learning techniques can classify an image of a rare species correctly after being exposed to a small amount of prior information.

One-shot learning

- One-shot learning performs classification tasks using past data.
- One-shot learning is the task of learning information about object categories from a single training example. Found mostly in computer vision.

- One-shot learning has been using the Siamese network approach.
- Eventually, Siamese networks were compared to comparative loss functions, after which the triplet loss function was proven to be better and the FaceNet system began using them.
- Contrastive loss and triplet loss functions are now used for high-quality face embeddings, which have become the foundation for modern facial recognition.

Zero-shot learning

- Zero-shot learning involves little human intervention, and the models depend on previously trained concepts and additional existing data.
- Zero-shot learning is the challenge of learning modeling without using data labeling.
- Reduces the time and effort that data labeling takes.
- Instead of giving training examples, zero-shot learning gives a high-level description of new categories so that the machine can relate them to existing categories that the machine has learned about.
- Can be used in computer vision, natural language processing, and machine perception.

What is Prompt Engineering?

Traditional strategies for training large language models such as GPT-3 and BERT require the model to be pre-trained with unlabeled data and then fine-tuned for specific tasks with labeled data. In contrast, prompt-based learning models can autonomously tune themselves for different tasks by transferring domain knowledge introduced through prompts.

A **prompt** is a snippet of natural language text that is added to unlabel data during the pre-training phase. The art of writing useful prompts is called prompt engineering. During in-context learning, we give the Language Model(LM) a prompt that consists of a list of input-output pairs that demonstrate a task. At the end of the prompt, we append a test input and allow the LM to make a prediction just by conditioning on the prompt and predicting the next tokens. A good prompt directs the model step by step for it to generate good results.

Findings

Objective of the study

Large Language Model(LLMs) can be used for multiple use in business and social problems. The objective of this article is to introduce the concepts of LLMs and show how to generate prompts for LLMs to extract relevant entities from job descriptions and also to classify news artifacts given only a few examples of human scores.

Methodologies

Methodologies/Tools

MLFlow: is an open-source platform for managing the end-to-end machine learning lifecycle. It has the following primary components: Tracking: Allows you to track experiments to record and compare parameters and results.

- It is easy to set up a model tracking mechanism in MLflow.
- It offers very intuitive APIs for serving.
- It provides data collection, data preparation, model training, and taking the model to production.

Data Version Control(DVC): Open-source Version Control System for Machine Learning Projects. It is an open-source Version Control System for data science and machine learning projects.

- Along with data versioning, DVC also allows model and pipeline tracking.
- With DVC, you don't need to rebuild previous models or data modeling techniques to achieve the same past state of results.
- Along with data versioning, DVC also allows model and pipeline tracking.

Cohere: is A platform enabling developers to generate or analyze text to do things like write copy, moderate content, classify data, and extract information, all at a massive scale.

- Use the Cohere platform to build natural language understanding and generation into your product with a few lines of code.
- Cohere's large language models can solve a broad spectrum of natural language use cases, including classification, semantic search, paraphrasing, summarization, and content generation.
- Through fine tuning, users can create massive models customized to their use case and trained on their data.

Types of data used

This project contains two data sets

1. News artifacts from web pages.data

The columns of this data are as follows

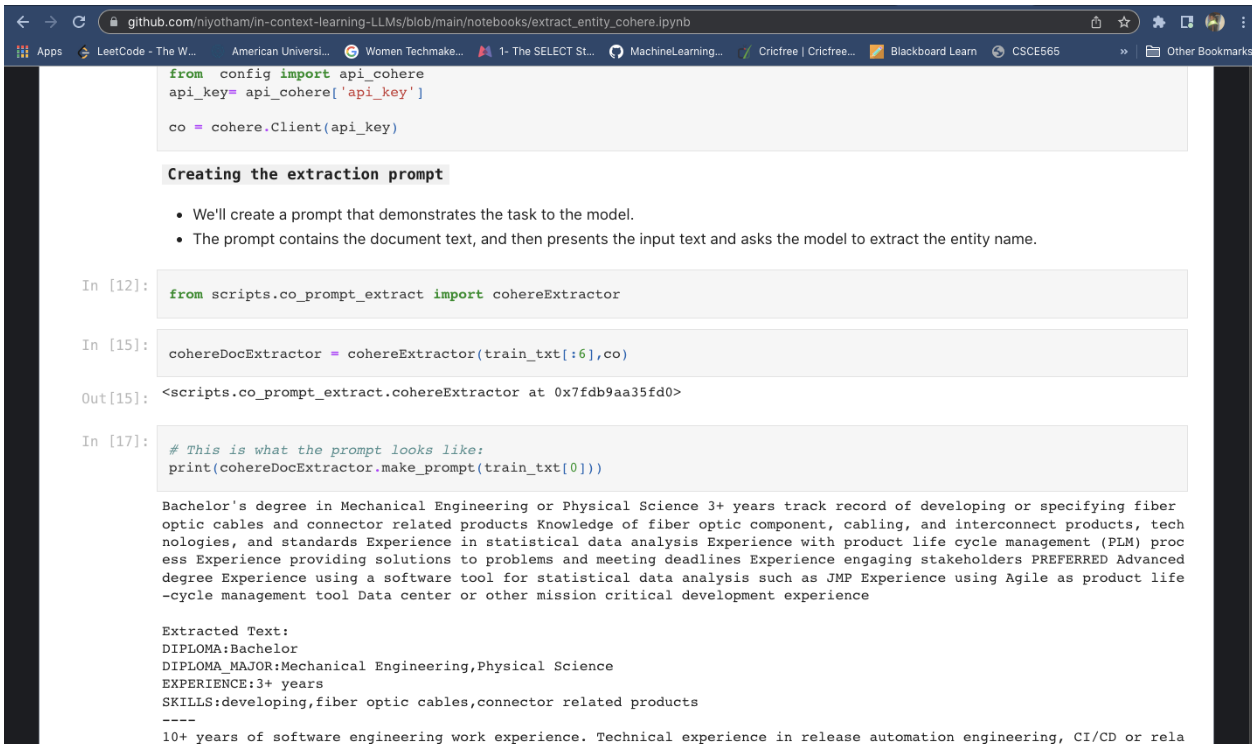
Domain - the base URL or a reference to the source these item comes from
Title - title of the item - the content of the item
Description - the content of the item
Body - the content of the item
Link - URL to the item source (it may not functional anymore sometime)
Timestamp - timestamp that this item was collected at
Analyst_Average_Score - target variable - the score to be estimated
Analyst_Rank - score as rank
Reference_Final_Score - Not relevant for now - it is a transformed quantity

2. Job descriptions (together named entities)

Results & discussions from the design and implementation of your MLOps pipeline

Prompt Engineering on Job Description Dataset

The full data preprocessing can be found at the following link [Preprocessing](). The below picture shows how to use the cohere API and create a prompt.



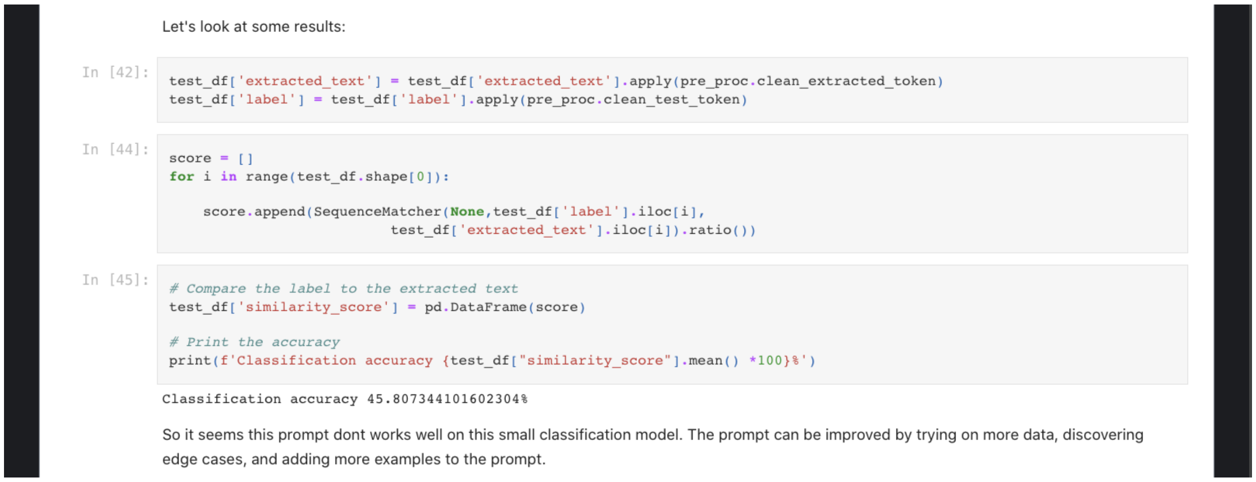
```
from  config import api_cohere
api_key= api_cohere['api_key']

co = cohere.Client(api_key)
```

Creating the extraction prompt

- We'll create a prompt that demonstrates the task to the model.
- The prompt contains the document text, and then presents the input text and asks the model to extract the entity name.

```
In [12]: from scripts.co_prompt_extract import cohereExtractor

In [15]: cohereDocExtractor = cohereExtractor(train_txt[:6],co)

Out[15]: <scripts.co_prompt_extract.cohereExtractor at 0x7fdb9aa35fd0>

In [17]: # This is what the prompt looks like:
         print(cohereDocExtractor.make_prompt(train_txt[0]))

Bachelor's degree in Mechanical Engineering or Physical Science 3+ years track record of developing or specifying fiber
optic cables and connector related products Knowledge of fiber optic component, cabling, and interconnect products, tech
nologies, and standards Experience in statistical data analysis Experience with product life cycle management (PLM) proc
ess Experience providing solutions to problems and meeting deadlines Experience engaging stakeholders PREFERRED Advanced
degree Experience using a software tool for statistical data analysis such as JMP Experience using Agile as product life
-cycle management tool Data center or other mission critical development experience

Extracted Text:
DIPLOMA:Bachelor
DIPLOMA_MAJOR:Mechanical Engineering,Physical Science
EXPERIENCE:3+ years
SKILLS:developing,fiber optic cables,connector related products
----
10+ years of software engineering work experience. Technical experience in release automation engineering, CI/CD or rela
```

Using **from** difflib **import** SequenceMatcher methode, we can check the similarities


Let's look at some results:

```
In [42]: test_df['extracted_text'] = test_df['extracted_text'].apply(pre_proc.clean_extracted_token)
         test_df['label'] = test_df['label'].apply(pre_proc.clean_test_token)
```

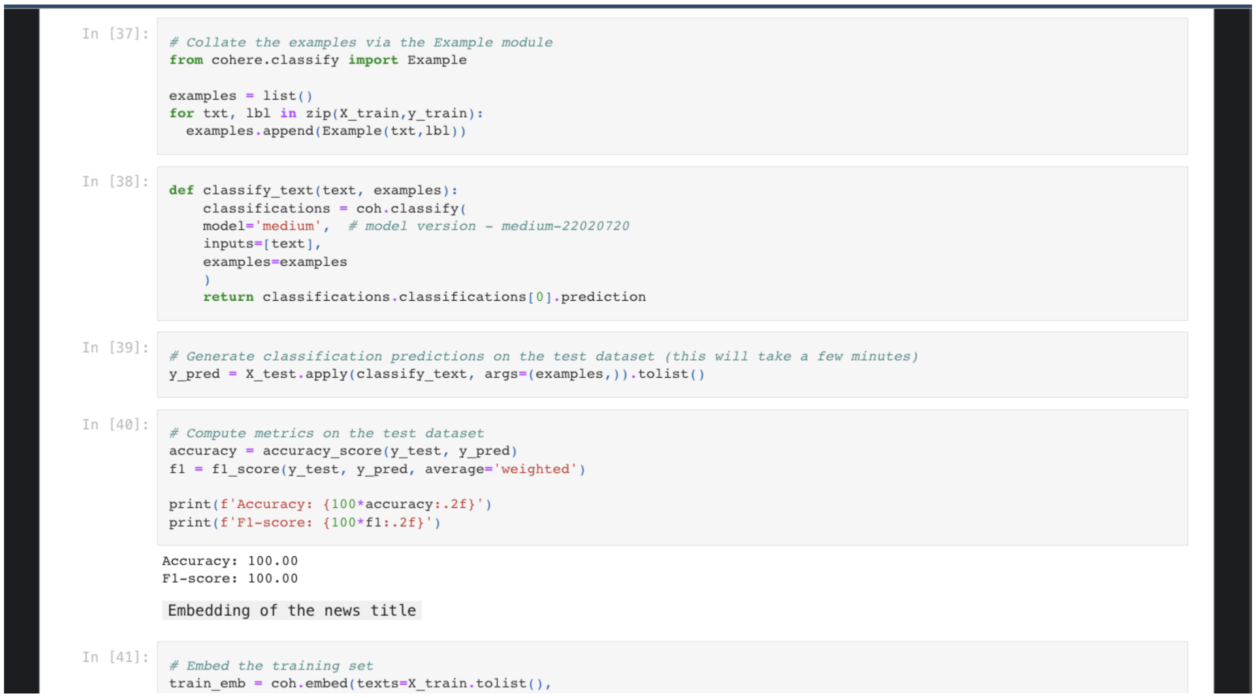
```
In [44]: score = []
         for i in range(test_df.shape[0]):

             score.append(SequenceMatcher(None,test_df['label'].iloc[i],
                                     test_df['extracted_text'].iloc[i]).ratio())
```

```
In [45]: # Compare the label to the extracted text
         test_df['similarity_score'] = pd.DataFrame(score)

         # Print the accuracy
         print(f'Classification accuracy {test_df["similarity_score"].mean() *100}%')
```

```
Classification accuracy 45.807344101602304%
```

So it seems this prompt dont works well on this small classification model. The prompt can be improved by trying on more data, discovering edge cases, and adding more examples to the prompt.

For the news scoring dataset the code for preprocessing and training the LLM using cohere can be found here. The process is not quite different from the entity though this is a classification model. The code snipped on how to apply classification using cohere is bellow



```
In [37]: # Collate the examples via the Example module
         from cohere.classify import Example

         examples = list()
         for txt, lbl in zip(X_train,y_train):
           examples.append(Example(txt,lbl))
```

```
In [38]: def classify_text(text, examples):
             classifications = coh.classify(
             model='medium',  # model version - medium-22020720
             inputs=[text],
             examples=examples
             )
             return classifications.classifications[0].prediction
```

```
In [39]: # Generate classification predictions on the test dataset (this will take a few minutes)
         y_pred = X_test.apply(classify_text, args=(examples,)).tolist()
```

```
In [40]: # Compute metrics on the test dataset
         accuracy = accuracy_score(y_test, y_pred)
         f1 = f1_score(y_test, y_pred, average='weighted')

         print(f'Accuracy: {100*accuracy:.2f}')
         print(f'F1-score: {100*f1:.2f}')
```

```
Accuracy: 100.00
F1-score: 100.00
```

`Embedding of the news title`

```
In [41]: # Embed the training set
         train_emb = coh.embed(texts=X_train.tolist(),
```

Recommendation and outcomes

Classification accuracy 45.807344101602304% for entity extraction is very low.

So it seems this prompt doesn't work well on this small classification model. The prompt can be improved by trying on more data, discovering edge cases, and adding more examples to the prompt.

However, for the classification method for news scoring. The model seems to be doing well. We can recommend it to be used.

Limitations of your analysis and future work

The project is huge and needs time to accomplish all tasks such as writing a flask backend. This is the work for the future as well as fine tuning the model.

References.

1. https://www.tensorflow.org/text/guide/word_embeddings
2. https://www.quora.com/Is-word-embedding-word2vec-a-type-of-vector-space-model
3. *https://www.quora.com/Is-word-embedding-word2vec-a-type-of-vector-space-model*
4. https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model
5. https://os.cohere.ai/
6. https://docs.cohere.ai/api-reference/
7. https://docs.cohere.ai/finetuning-wiki/
8. https://arxiv.org/pdf/1909.08593.pdf%5D
9. https://www.analyticsvidhya.com/blog/2020/06/nlp-project-information-extraction/
10. https://towardsdatascience.com/how-i-used-natural-language-processing-to-extract-context-from-news-headlines-df2cf5181ca6
11. https://fourweekmba.com/prompt-engineering/
12. https://www.xcubelabs.com/blog/faqs-few-shot-learning-everything-you-need-to-know/
13. https://machinelearningmastery.com/what-are-word-embeddings/
14. https://analyticsindiamag.com/how-to-train-your-llm-efficiently/