BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

NLP LAB - 2

[INFORMATION RETRIEVAL AND PROMPT ENGINEERING]

DATE: 02/12/2023 – 03/12/2023
TIME: 02 Hours

This Labsheet contains IR and Prompt Engineeering. The agenda (learning outcomes) for IR are preliminary for Neural IR, Intro to setup an IR experiment, Analysis of a dataset with BERT, (Re-)Ranking of Text, Dense Representations for Retrieval. We shall use OpenAI and Serper API keys for the prompt engineering section of the labsheet

Sources:

1. https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/
2. https://colab.research.google.com/github/tensorflow/recommenders/blob/main/docs/examples/basic_retrieval.ipynb

Information Retrieval (IR)

Conducting an IR experiment

Typical pipeline:

- Loading and processing a dataset
- Implement algorithms and baselines.
- Running algorithms (simulating real-world application).
- Evaluating the results (beyond loss).
- Analysing the results.
- Optional: optimize and put into production.

Setup for IR

Let us include all necessary libraries

```
!pip install -Uq sentence-transformers
!pip install datasets
```

```python
from datasets import load_dataset, DatasetDict
import sentence_transformers
import sentence_transformers.cross_encoder.evaluation
from sentence_transformers import SentenceTransformer, CrossEncoder,
InputExample  # High-level sentence encoders.
import sentence_transformers.models as models
import sentence_transformers.losses as losses
import torch
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm  # Enables progress bars
import pandas as pd

import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"

QUICK_RUN = False  # Config setting to switch between foreground
(subset) and background (full-dataset) running
```

Choosing the dataset

We will perform document retrieval and ranking. Hence, we choose a benchmark dataset in this domain on Scientific documents.

```python
# https://aclanthology.org/2020.acl-main.207/
# https://arxiv.org/abs/2104.08663
queries = load_dataset("BeIR/scidocs", "queries", split="queries")
docs = load_dataset("BeIR/scidocs", "corpus", split="corpus")
qrels = load_dataset("BeIR/scidocs-qrels", delimiter="\t",
split="test")
len(queries), len(docs), len(qrels), len(set(qrels["query-id"])),
len(set(qrels["corpus-id"]))
```

Structure

In IR, we have a query, a collection of documents and a assignment of relevancy between (some) queries and documents. This can be seen by the way the dataset is structured.

```python
queries, docs, qrels
```

```
(Dataset({ features: ['_id', 'title', 'text'], num_rows: 1000 }),
Dataset({ features: ['_id', 'title', 'text'], num_rows: 25657 }),
Dataset({ features: ['query-id', 'corpus-id', 'score'], num_rows: 29928
}))
```

Preprocessing. Sparsity. Cold-start problem.

Our dataset already has a natural representation. We could still peform filtering. We must ensure that no information is lost.

```
# For demonstration purposes only
if QUICK_RUN:
    queries = queries.select(range(100))
    docs = docs.select(range(2500))
    qrels = qrels.filter(lambda x: x["query-id"] in queries["_id"] and
x["corpus-id"] in docs["_id"])
```

Train, Val, Test sets

This step is very important.

- Train for algorithm parameter estimation.
- Validation (also called development) for finding a good model (e.g., hyperparameter optimization, cross-validation).
- Test (also called evaluation) only used for evaluation at the very end.

Sometimes datasets have predefined splits that can be used.

```
# 90% train, 10% test + validation
train_testvalid = qrels.train_test_split(..., seed=1) # TODO
# Split the 10% test + valid in half test, half valid
test_valid = train_testvalid['test'].train_test_split(... seed=1) #TODO
# gather everyone if you want to have a single DatasetDict
train_test_valid_dataset = DatasetDict({ # TODO
    'train': ...,
    'test': ...,
    'valid':...})
train_test_valid_dataset
```

```
DatasetDict({ train: Dataset({ features: ['query-id', 'corpus-id',
'score'], num_rows: 26935 }) test: Dataset({ features: ['query-id',
'corpus-id', 'score'], num_rows: 1497 }) valid: Dataset({ features:
['query-id', 'corpus-id', 'score'], num_rows: 1496 }) })
```

Splitting Methods

- depend on the task
- random sometimes not feasible -> data leakage

Common alternative splitting options:

- time-based
- session-based
- user-based

Analysis- BERTopic Showcase

First, we will analyze the data.

For the start let's just look at some example relevance assigenements and their associated content.

```
def get_triple_for_example(example):
    q = queries[queries["_id"].index(example["query-id"])]["text"]
    d = docs[docs["_id"].index(example["corpus-id"])]["title"]
    r = example["score"]
    return q, d, r

ex0 = get_triple_for_example(train_test_valid_dataset["test"][0])
ex1 = get_triple_for_example(train_test_valid_dataset["test"][1])
ex0, ex1
```

```
(('Provable data possession at untrusted stores', 'StreamOp: An
Innovative Middleware for Supporting Data Management and Query
Functionalities over Sensor Network Streams Efficiently', 0), ('Rumor
Detection and Classification for Twitter Data', 'Thumbs Up or Thumbs
Down? Semantic Orientation Applied to Unsupervised Classification of
Reviews', 1))
```

Label Distribution

```
from collections import Counter
from scipy import stats

# From Huggingface Evaluate
def label_dist(data):
    """Returns the fraction of each label present in the data"""
    c = Counter(data)
    label_distribution = {"labels": [k for k in c.keys()], "fractions":
[f / len(data) for f in c.values()]}
    if isinstance(data[0], str):
        label2id = {label: id for id, label in
enumerate(label_distribution["labels"])}
        data = [label2id[d] for d in data]
    skew = stats.skew(data)
    return {"label_distribution": label_distribution, "label_skew":
skew}

label_dist(data=train_test_valid_dataset["train"]["score"]),
label_dist(data=train_test_valid_dataset["valid"]["score"]),
label_dist(data=train_test_valid_dataset["test"]["score"])
```

```
({'label_distribution': {'labels': [1, 0], 'fractions':
[0.16461852608130684, 0.8353814739186931]}, 'label_skew':
1.8087864265977875}, {'label_distribution': {'labels': [0, 1],
'fractions': [0.8348930481283422, 0.16510695187165775]}, 'label_skew':
1.8040061996868444}, {'label_distribution': {'labels': [0, 1],
'fractions': [0.8350033400133601, 0.16499665998663995]}, 'label_skew':
1.8050841379113802})
```

Content

Next, we look at the content via topic modelling.

We use a recent model called [BERTopic](https://) (2022).

```
model_name = 'sentence-transformers/all-MiniLM-L6-v2'
```

```
docs.map(lambda x: {"title_text": x["title"] + ": " +
x["text"]})["title_text"][:2]
```

Map: 100%
25657/25657 [00:03<00:00, 7192.47 examples/s]

```
['A hybrid of genetic algorithm and particle swarm optimization for
recurrent network design: An evolutionary recurrent network which automates
the design of recurrent neural/fuzzy networks using a new evolutionary
learning algorithm is proposed in this paper. ..., demonstrating its
superiority.',
 'A Hybrid EP and SQP for Dynamic Economic Dispatch with Nonsmooth Fuel
Cost Function: Dynamic economic dispatch (DED) is one of the main functions
of power generation operation and control. ... from EP and SQP alone.']
```

```
!pip install -Uq bertopic
```

```
from bertopic import BERTopic
from bertopic.vectorizers import ClassTfidfTransformer
import plotly

docs_for_analysis = docs.map(lambda x: {"title_text": x["title"] + ": "
+ x["text"]})["title_text"]
topic_model = BERTopic(embedding_model=model_name,
ctfidf_model=ClassTfidfTransformer(reduce_frequent_words=True))
topic_model.fit(docs_for_analysis)
topic_model.get_topic_info().head()
```

Explain Embeddings (UMAP Plot)

```
topic_model.reduce_topics(docs_for_analysis, nr_topics=15)
fig = topic_model.visualize_documents(docs_for_analysis)
plotly.offline.plot(fig, filename='bertopic_doc_embeddings.html')
```

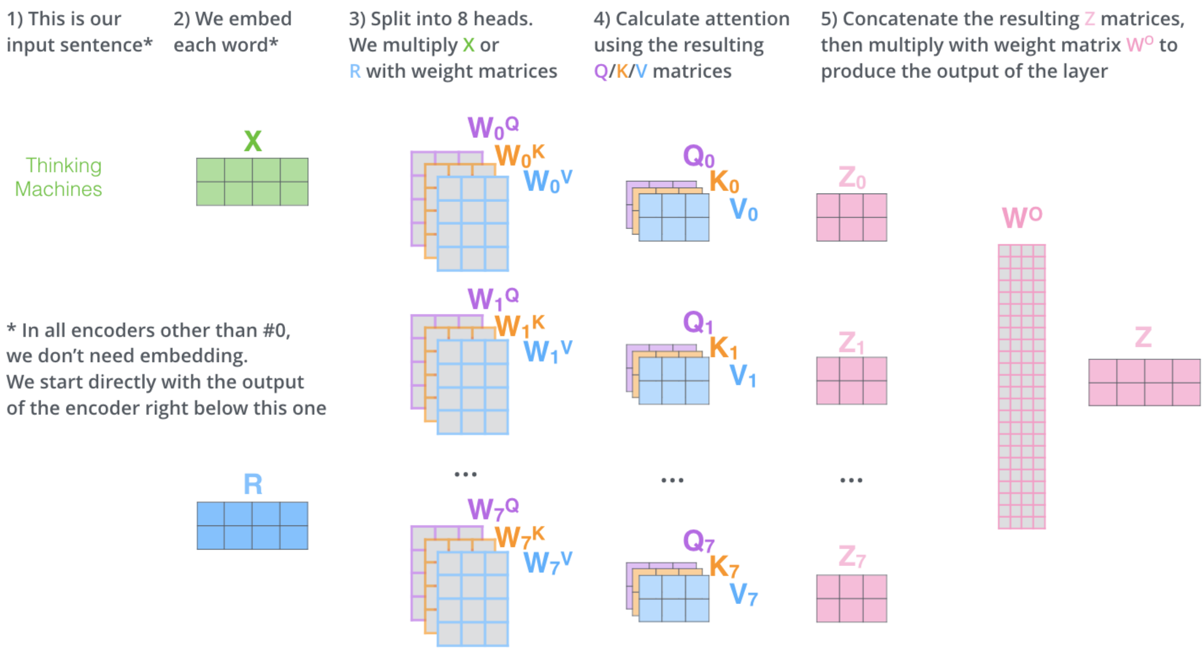
```
from IPython.display import IFrame
IFrame(src='bertopic_doc_embeddings.html', width=1200, height=800)
```

High-Level Overview of Transformer

Why transformers:

- Contextualized embeddings, such as ElMo (2018).
- Transfer learning, such as ULMFit (2018).
- Wide-variety of NLP tasks.

Transformers use multi-head Attention (Attention Is All You Need - 2017):



We end up with an embedding, that can then customized with a special head (e.g., classification or pooling).

Today focus on BERT (2018, bi-directional with masking).

```python
from transformers import AutoTokenizer, AutoModel,
AutoModelForSequenceClassification

# Tokenizer and model must match
ex_tokenizer = AutoTokenizer.from_pretrained(model_name)
ex_model = AutoModel.from_pretrained(model_name)
ex_model_with_head =
AutoModelForSequenceClassification.from_pretrained(model_name)  # Needs
fine-tuning, here for demonstration

test_sentences = ["This is the first sentence with complex tokens, such
as SentenceTransformers.", "We can batch multiple sentences."]

ex_tokenized = ex_tokenizer(test_sentences, return_tensors="pt",
padding=True, truncation=True)  # Collates data with padding
ex_res = ex_model(**ex_tokenized)
ex_res_with_head = ex_model_with_head(**ex_tokenized)

print("\nTokenized text:")  # Word Piece Tokenization
print(ex_tokenizer.tokenize(test_sentences))
print("\nToken IDs:")
print(ex_tokenized)
print("\nOutput Dictionary:")
print(ex_res.keys())
print("\nOutput Size:")
print(ex_res.last_hidden_state.size())
print("\nContextualized Token Embeddings (truncated):")
print(ex_res.last_hidden_state[:, :3, :7])  # First 3 tokens
print("\nPooled Embeddings (truncated):")
print(ex_res.pooler_output.shape, ex_res.pooler_output[:, :7])
print("\nPredicted Values (not fine-tuning)")
print(ex_res_with_head)
```

Tokenization

Can be learned. BERT uses WordPiece. Based on common sub-words (in comparison to word- or character-based). Can deal with unknown compound words.

Special Tokens

In BERT:
[CLS] sent1 [SEP] sent2 [SEP]
Other tokens:

- [MASK]
- [UNK]
- [PAD]

```python
# Uses Mean pooling
topic_model.embedding_model.embedding_model
```

Common Pooling Models

- Mean Pooling (average)
- Max Pooling
- Sequence-length dependant
- Special token ([CLS] in BERT)

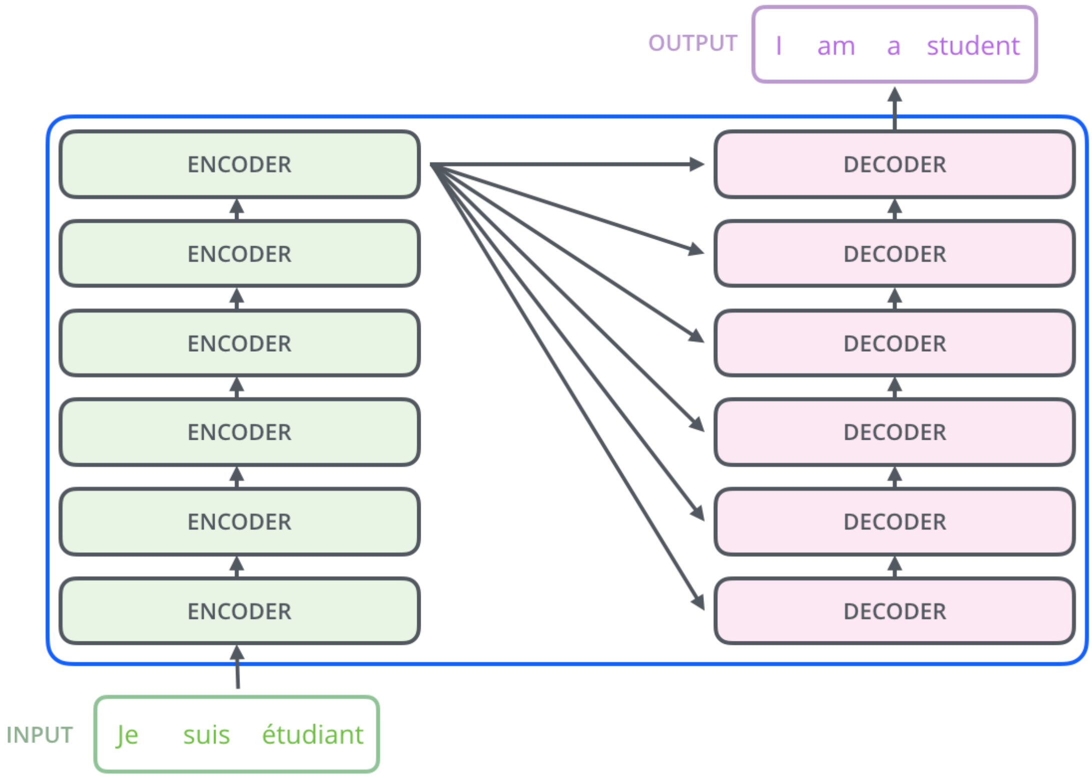
```python
# Starts with embeddings
topic_model.embedding_model.embedding_model[0]._modules["auto_model"]
```

Types of Transformers

BERT only uses Encoder Layers.

- Encoders (e.g., BERT, RoBERTa): auto-encoding models - NLU
- Decoders (e.g., GPT): auto-regressive models - NLG
- Seq2Seq (e.g., BART): encoder-decoder models - Translation

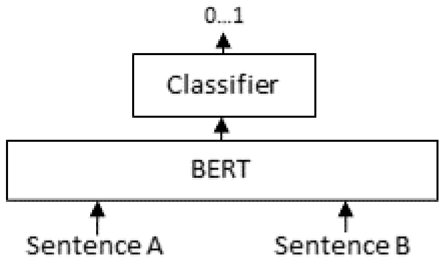
[Overview of Models]



BERT4Re-Ranking – monoBERT (cross-encoder)

Idea:

- Concat and predict.
- Can directly use pretrained architectures.
- Just fine-tuning required.



```python
from collections import defaultdict

class IRDataset(Dataset):
    def __init__(self, queries_ds, docs_ds, qrel_ds, mode="cross"):
        self.mode = mode

        qrels = defaultdict(set)

        def transform(x):
            q, d, r = x["query-id"], x["corpus-id"], x["score"]

            q_idx = queries_ds["_id"].index(q)
            x["query_text"] = queries_ds[q_idx]["..."] # TODO
            d_idx = docs_ds["_id"].index(d)
            x["doc_content"] = docs_ds[d_idx]["title"] + ": " +
docs_ds[d_idx]["text"]
            x["label"] = float(r)

            if r:
                qrels[q].add(d)

            return x

        qrel_ds = qrel_ds.map(transform)

        self.q_ids = qrel_ds["..."] #TODO
        self.d_ids = qrel_ds["..."] #TODO
        self.qrels = qrels

        self.queries = qrel_ds["query_text"]
        self.docs = qrel_ds["doc_content"]
        self.labels = qrel_ds["label"]
```

```
    def __getitem__(self, idx):
        qs = self.queries[idx]
        ds = self.docs[idx]
        if self.mode == "rep":
            if type(idx) is int:
                text_list = [{"query": qs}, {"doc": ds}]
            else:
                text_list = [[{"query": q} for q in qs], [{"doc": d}
for d in ds]]
            return InputExample(texts=text_list,
label=self.labels[idx])
        return InputExample(texts=[qs, ds], label=self.labels[idx])

    def set_mode(self, mode):
        self.mode = mode

    def __len__(self):
        return len(self.labels)
```

```
train_ds = IRDataset(queries, docs, train_test_valid_dataset["train"])
valid_ds = IRDataset(queries, docs, train_test_valid_dataset["valid"])
train_ds[0].__dict__
```

```
monoBERT = CrossEncoder(model_name, # We use cross-encoder as monoBERT
example
                     num_labels=1, # Perform binary classification
                     device=None,  # Will use CUDA if available
                    )
```

```
monoBERT.predict([ex0[:2], ex1[:2]])
```

```
print(train_ds[0])
```

```
train_dl = DataLoader(train_ds, batch_size=32)
# We need sentence pairs format for the library here.
# valid_dl = DataLoader(valid_ds, batch_size=32)
sentence_pairs = list(zip(valid_ds.queries, valid_ds.docs))
labels = valid_ds.labels
len(train_dl)
```

```
monoBERT.__dict__.keys()
```

```
class_evaluator =
sentence_transformers.cross_encoder.evaluation.CEBinaryClassificationEv
aluator(sentence_pairs, labels, show_progress_bar=True)
monoBERT.fit(train_dataloader=train_dl,
          loss_fct=None,  # uses nn.BCEWithLogitsLoss()
          evaluator=class_evaluator,
          epochs=10,
          optimizer_class=torch.optim.AdamW,
          show_progress_bar=True,
          save_best_model=True,
          output_path="./",
        )
```

```
# Tip: look at CUDA GPU.
!nvidia-smi
```

```
monoBERT.model
```

```
monoBERT.predict([ex0[:2], ex1[:2]])
```

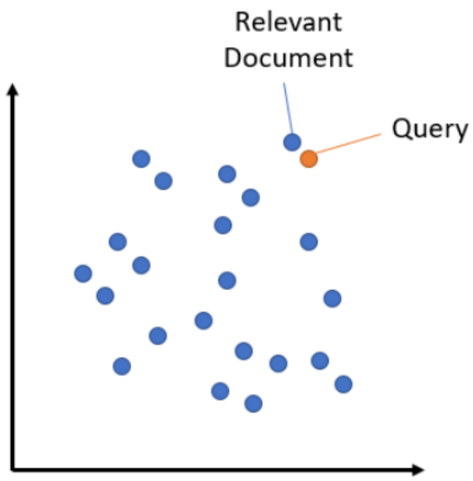
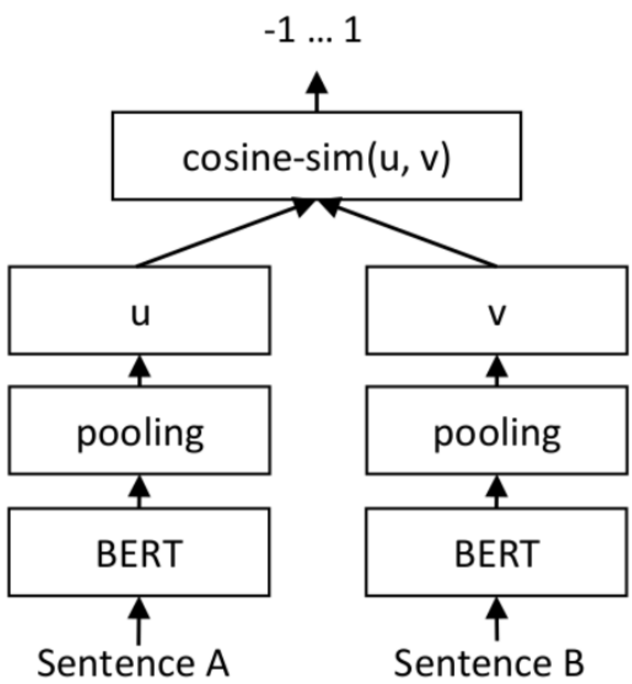
```
df = pd.read_csv("CEBinaryClassificationEvaluator_results.csv")
df.tail(n=10)
```

```
df.set_index("epoch").drop(columns=["steps"]).plot()
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
```

BERT4Retrieval – Representation based (biencoder)

Idea:

- Train embeddings for queries and documents.
- Asymmetric architecture vs Siamese network. (we use the latter for simplicity)
- Similarity function -> loss.

Embedding Based Retrieval	Architecture
 	

```
repBased = SentenceTransformer(model_name)
```

```
qs, ds = repBased.encode([{"query": ex0[0]}, {"query": ex1[0]}]),
repBased.encode([{"doc": ex0[1]}, {"doc": ex1[0]}])
sentence_transformers.util.cos_sim(qs, ds)
```

```
train_ds.set_mode("rep")
valid_ds.set_mode("rep")
train_dl_repBased = DataLoader(train_ds, batch_size=32,
collate_fn=repBased.smart_batching_collate)
valid_dl_repBased = DataLoader(valid_ds, batch_size=32,
collate_fn=repBased.smart_batching_collate)
assert next(iter(train_dl_repBased))
```

```
queries_dict = dict(zip(valid_ds.q_ids, valid_ds.queries))
docs_dict = dict(zip(valid_ds.d_ids, valid_ds.docs))
qrels_dict = valid_ds.qrels
```

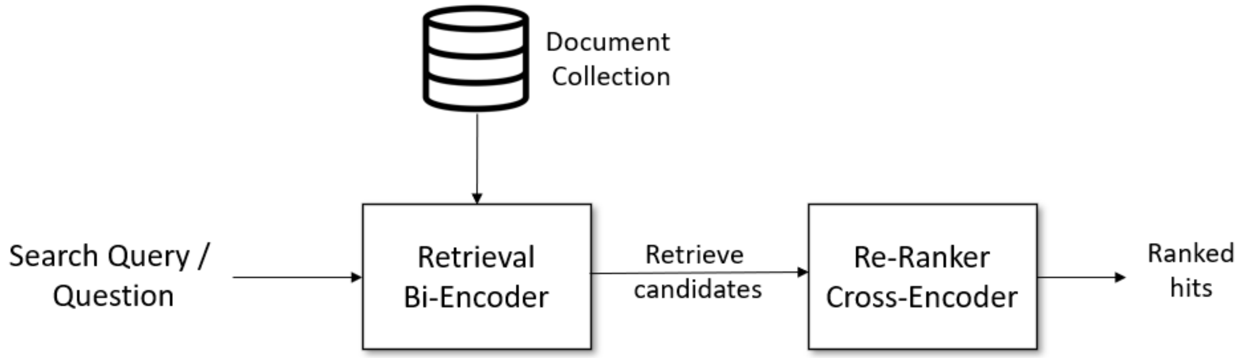
```
ir_evaluator =
sentence_transformers.evaluation.InformationRetrievalEvaluator(queries_
dict, docs_dict, qrels_dict, write_csv=True)
repBased.fit(train_objectives=[(train_dl_repBased,
losses.CosineSimilarityLoss(repBased))],
          evaluator=ir_evaluator,
          epochs=10,
          optimizer_class=torch.optim.AdamW,
          show_progress_bar=True,
          save_best_model=True,
          output_path="./",
        )
```

```
qs, ds = repBased.encode([{"query": ex0[0]}, {"query": ex1[0]}]),
repBased.encode([{"doc": ex0[1]}, {"doc": ex1[0]}])
sentence_transformers.util.cos_sim(qs, ds)
```

```
df = pd.read_csv("eval/Information-Retrieval_evaluation_results.csv")
df.tail(n=10)
```

```
df.set_index("epoch").drop(columns=["steps"]).plot(legend=False)
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5), ncol=3)
```

Putting things together

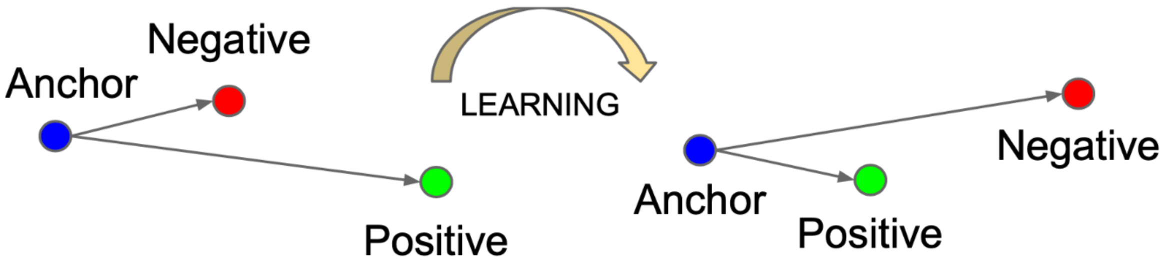


Multiple re-rankers (e.g., DuoBERT).

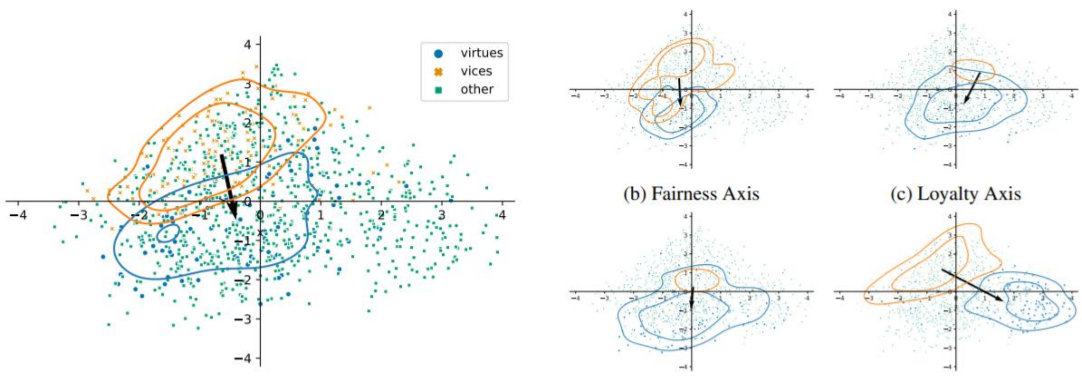
Evaluate (offline on test data) or online in production system.

Future Directions (Self-Exploration)

- Contrastive Learning (pretraining or auxiliary tasks)



- Distillation (Student and Teacher, e.g., distilBERT from HF)
- Prompt engineering (zero-shot) vs few-shot fine-tuning
- Recently, usage in Vision (ViT)
- Multi-modality (CLIP embeddings in Stable Diffusion)
- Content Bias like framing of messages (e.g., https://github.com/socialcomplab/icwsm21-framing).



(a) Care Axis (b) Fairness Axis (c) Loyalty Axis (d) Authority Axis (e) Sanctity Axis

- Question Answering is another common IR-related task tackled with transformers.

Prompt Engineering

This notebook contains examples and exercises to learn about prompt engineering.
We will be using the OpenAI APIs for all examples. Here, we are using the default settings temperature=0.7 and top-p=1

Basics

Objectives

- Load the libraries

- Review the format
- Cover basic prompts
- Review common use cases

Below we are loading the necessary libraries, utilities, and configurations.

```
%%capture
# update or install the necessary libraries
!pip install --upgrade openai
!pip install --upgrade langchain
!pip install --upgrade python-dotenv
```

Load environment variables. You can use anything you like, or ~~hard-code the secret~~ (which is never ever recommended, as putting a secret key on a publically accessible platform can cause unauthorized use). Just create a .env file with your OPENAI_API_KEY then load it.

```
import openai
import os
import IPython
from langchain.llms import OpenAI
from dotenv import load_dotenv
```

```
load_dotenv()

# API configuration
client = OpenAI(
  api_key=os.environ['OPENAI_API_KEY'],
)

# for LangChain
os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")
os.environ["SERPER_API_KEY"] = os.getenv("SERPER_API_KEY")
```

```python
def set_open_params(
    model="text-davinci-003",
    temperature=0.7,
    max_tokens=256,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0,
):
    """ set openai parameters"""

    openai_params = {}

    openai_params['model'] = model
    openai_params['temperature'] = temperature
    openai_params['max_tokens'] = max_tokens
    openai_params['top_p'] = top_p
    openai_params['frequency_penalty'] = frequency_penalty
    openai_params['presence_penalty'] = presence_penalty
    return openai_params
```

```python
def get_completion(params, prompt):
    """ GET completion from openai api"""

    response = openai.Completion.create(
        engine = params['model'],
        prompt = prompt,
        temperature = params['temperature'],
        max_tokens = params['max_tokens'],
        top_p = params['top_p'],
        frequency_penalty = params['frequency_penalty'],
        presence_penalty = params['presence_penalty'],
    )
    return response
```

Basic prompt example

```python
# basic example
params = set_open_params()

prompt = "The sky is"

response = get_completion(params, prompt)
```

```
response.choices[0].text
```

```
IPython.display.Markdown(response.choices[0].text)
```

```
params = set_open_params(temperature=0)
prompt = "The sky is"
response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.1 Text Summarization

```
params = set_open_params(temperature=0.7)
prompt = """Antibiotics are a type of medication used to treat
bacterial infections. They work by either killing the bacteria or
preventing them from reproducing, allowing the body's immune system to
fight off the infection. Antibiotics are usually taken orally in the
form of pills, capsules, or liquid solutions, or sometimes administered
intravenously. They are not effective against viral infections, and
using them inappropriately can lead to antibiotic resistance.

Explain the above in one sentence:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.2 Question Answering

```
prompt = """Answer the question based on the context below. Keep the
answer short and concise. Respond "Unsure about answer" if not sure
about the answer.

Context: Teplizumab traces its roots to a New Jersey drug company
called Ortho Pharmaceutical. There, scientists generated an early
version of the antibody, dubbed OKT3. Originally sourced from mice, the
molecule was able to bind to the surface of T cells and limit their
cell-killing potential. In 1986, it was approved to help prevent organ
rejection after kidney transplants, making it the first therapeutic
antibody allowed for human use.

Question: What was OKT3 originally sourced from?

Answer:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.3 Text Classfication

```
prompt = """Classify the text into neutral, negative or positive.

Text: I think the food was okay.

Sentiment:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.4 Role Playing

```
prompt = """The following is a conversation with an AI research
assistant. The assistant tone is technical and scientific.

Human: Hello, who are you?
AI: Greeting! I am an AI research assistant. How can I help you today?
Human: Can you tell me about the creation of blackholes?
AI:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.5 Code Generation

```
prompt = "\"\"\"\nTable departments, columns = [DepartmentId,
DepartmentName]\nTable students, columns = [DepartmentId, StudentId,
StudentName]\nCreate a MySQL query for all students in the Computer
Science Department\n\"\"\""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

1.6 Reasoning

```
prompt = """The odd numbers in this group add up to an even number: 15,
32, 5, 13, 82, 7, 1.

Solve by breaking the problem into steps. First, identify the odd
numbers, add them, and indicate whether the result is odd or even."""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

Advanced Prompting Techniques

2.1 Few Shots Prompting

```
prompt = """The odd numbers in this group add up to an even number: 4,
8, 9, 15, 12, 2, 1.
A: The answer is False.

The odd numbers in this group add up to an even number: 17,  10, 19, 4,
8, 12, 24.
A: The answer is True.

The odd numbers in this group add up to an even number: 16,  11, 14, 4,
8, 13, 24.
A: The answer is True.

The odd numbers in this group add up to an even number: 17,  9, 10, 12,
13, 4, 2.
A: The answer is False.

The odd numbers in this group add up to an even number: 15, 32, 5, 13,
82, 7, 1.
A:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

2.2 Chain-of-Thought (CoT) Prompting

```
prompt = """The odd numbers in this group add up to an even number: 4,
8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.

The odd numbers in this group add up to an even number: 15, 32, 5, 13,
82, 7, 1.
A:"""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

2.3 Zero-shot CoT

```
prompt = """I went to the market and bought 10 apples. I gave 2 apples
to the neighbor and 2 to the repairman. I then went and bought 5 more
apples and ate 1. How many apples did I remain with?

Let's think step by step."""

response = get_completion(params, prompt)
IPython.display.Markdown(response.choices[0].text)
```

2.4 PAL – Code as Reasoning

```
# lm instance
llm = OpenAI(model_name='text-davinci-003', temperature=0)
```

```
question = "Which is the oldest penguin?"
```

```
PENGUIN_PROMPT = '''
"""
Q: Here is a table where the first line is a header and each subsequent
line is a penguin:
name, age, height (cm), weight (kg)
Louis, 7, 50, 11
Bernard, 5, 80, 13
Vincent, 9, 60, 11
Gwen, 8, 70, 15
For example: the age of Louis is 7, the weight of Gwen is 15 kg, the
height of Bernard is 80 cm.
We now add a penguin to the table:
James, 12, 90, 12
How many penguins are less than 8 years old?
"""
# Put the penguins into a list.
penguins = []
penguins.append(('Louis', 7, 50, 11))
penguins.append(('Bernard', 5, 80, 13))
penguins.append(('Vincent', 9, 60, 11))
penguins.append(('Gwen', 8, 70, 15))
# Add penguin James.
penguins.append(('James', 12, 90, 12))
# Find penguins under 8 years old.
penguins_under_8_years_old = [penguin for penguin in penguins if
penguin[1] < 8]
# Count number of penguins under 8.
num_penguin_under_8 = len(penguins_under_8_years_old)
answer = num_penguin_under_8
"""
Q: Here is a table where the first line is a header and each subsequent
line is a penguin:
name, age, height (cm), weight (kg)
Louis, 7, 50, 11
Bernard, 5, 80, 13
Vincent, 9, 60, 11
Gwen, 8, 70, 15
For example: the age of Louis is 7, the weight of Gwen is 15 kg, the
height of Bernard is 80 cm.
Which is the youngest penguin?
"""
# Put the penguins into a list.
penguins = []
penguins.append(('Louis', 7, 50, 11))
penguins.append(('Bernard', 5, 80, 13))
penguins.append(('Vincent', 9, 60, 11))
penguins.append(('Gwen', 8, 70, 15))
```

```
# Sort the penguins by age.
penguins = sorted(penguins, key=lambda x: x[1])
# Get the youngest penguin's name.
youngest_penguin_name = penguins[0][0]
answer = youngest_penguin_name
"""
Q: Here is a table where the first line is a header and each subsequent
line is a penguin:
name, age, height (cm), weight (kg)
Louis, 7, 50, 11
Bernard, 5, 80, 13
Vincent, 9, 60, 11
Gwen, 8, 70, 15
For example: the age of Louis is 7, the weight of Gwen is 15 kg, the
height of Bernard is 80 cm.
What is the name of the second penguin sorted by alphabetic order?
"""
# Put the penguins into a list.
penguins = []
penguins.append(('Louis', 7, 50, 11))
penguins.append(('Bernard', 5, 80, 13))
penguins.append(('Vincent', 9, 60, 11))
penguins.append(('Gwen', 8, 70, 15))
# Sort penguins by alphabetic order.
penguins_alphabetic = sorted(penguins, key=lambda x: x[0])
# Get the second penguin sorted by alphabetic order.
second_penguin_name = penguins_alphabetic[1][0]
answer = second_penguin_name
"""
{question}
"""
'''.strip() + '\n'
```

```
llm_out = llm(PENGUIN_PROMPT.format(question=question))
print(llm_out)
```

```
exec(llm_out)
print(answer)
```

Tools and Applications

3.1 LLMs and Applications

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
```

```
llm = OpenAI(temperature=0)

tools = load_tools(["google-serper", "llm-math"], llm=llm)
agent = initialize_agent(tools, llm, agent="zero-shot-react-
description", verbose=True)
```

```
# run the agent
agent.run("Who is Olivia Wilde's boyfriend? What is his current age
raised to the 0.23 power?")
```

3.2 Data-Augmented Generation

```
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.embeddings.cohere import CohereEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores.elastic_vector_search import
ElasticVectorSearch
from langchain.vectorstores import Chroma
from langchain.docstore.document import Document
from langchain.prompts import PromptTemplate
```

```
with open('./state_of_the_union.txt') as f:
    state_of_the_union = f.read()
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
texts = text_splitter.split_text(state_of_the_union)

embeddings = OpenAIEmbeddings()
```

```
docsearch = Chroma.from_texts(texts, embeddings, metadatas=[{"source":
str(i)} for i in range(len(texts))])
```

```
query = "What did the president say about Justice Breyer"
docs = docsearch.similarity_search(query)
```

```
from langchain.chains.qa_with_sources import load_qa_with_sources_chain
from langchain.llms import OpenAI
```

```
chain = load_qa_with_sources_chain(OpenAI(temperature=0),
chain_type="stuff")
query = "What did the president say about Justice Breyer"
chain({"input_documents": docs, "question": query},
return_only_outputs=True)
```

```
template = """Given the following extracted parts of a long document
and a question, create a final answer with references ("SOURCES").
If you don't know the answer, just say that you don't know. Don't try
to make up an answer.
ALWAYS return a "SOURCES" part in your answer.
Respond in Spanish.

QUESTION: {question}
=========
{summaries}
=========
FINAL ANSWER IN SPANISH:"""

# create a prompt template
PROMPT = PromptTemplate(template=template,
input_variables=["summaries", "question"])

# query
chain = load_qa_with_sources_chain(OpenAI(temperature=0),
chain_type="stuff", prompt=PROMPT)
query = "What did the president say about Justice Breyer?"
chain({"input_documents": docs, "question": query},
return_only_outputs=True)
```

Exercise

1. Complete the TODOs in IR part
2. Create OpenAI and Serper API keys
3. Migrate the script and execute code for prompt engineering