《深度学习》实验报告

唐小卉　10215501437　数据科学与大数据技术 数据科学与工程学院

一、实验环境

本次导入的库有：

- numpy：用于数值计算，特别是数组和矩阵运算。
- scikit-learn：
  - TfidfVectorizer：用于提取文本的 TF-IDF 特征向量。
  - cosine_similarity：用于计算文本向量之间的余弦相似度。
  - normalize：用于对向量进行归一化处理。
- rank_bm25：
  - BM25Okapi：用于基于 BM25 算法的文档评分。
- tqdm：用于显示循环进度条。

二、实验过程

数据预处理：

1. 读取数据

   使用 read_json 函数读取三个 JSON 文件：document.json、query_testset.json 和 query_trainset.json。

```
# 读取数据集
documents = read_json('input/document.json')
queries = read_json('input/query_testset.json')
train_data = read_json('input/query_trainset.json')
```

2. 提取文本数据

   (1) 从文档和查询中提取 fact_input_list 和 query_input_list。

```
# 提取文档和查询的文本数据
document_texts = [' '.join(map(str, doc["fact_input_list"])) for doc in documents]
query_texts = [' '.join(map(str, query["query_input_list"])) for query in queries]
```

   (2) 提取训练集中的文档和查询文本数据。

```
# 提取训练集中的文档和查询文本数据
train_document_texts = []
for train in train_data:
    train_document_texts.extend([' '.join(map(str, evidence["fact_input_list"])) for evidence in train["evidence_list"]])
train_query_texts = [' '.join(map(str, train["query_input_list"])) for train in train_data]
```

3. 合并文本数据

合并训练集和测试集的文档文本数据。

```
# 合并训练集和测试集文档
all_document_texts = document_texts + train_document_texts
```

模型构建：

1. TF-IDF

使用 TfidfVectorizer 提取文本的 TF-IDF 特征向量，并调整其参数。参数我后期进行了多次尝试和调整，目前来看代码中的是最佳参数。

```
vectorizer = TfidfVectorizer(max_df=0.85, min_df=2, ngram_range=(1, 2))
all_document_tfidf = vectorizer.fit_transform(all_document_texts)
query_tfidf = vectorizer.transform(query_texts)
```

之后分离出原始文档的 TF-IDF 特征向量。

```
# 分离出原始文档的TF-IDF
document_tfidf = all_document_tfidf[:len(document_texts)]
```

TF-IDF 的原理是：首先计算一个词在文档中的出现频率（TF），然后计算逆文档频率（IDF），即衡量一个词在整个语料库中的重要性，然后结合 TF 与 IDF，得到每个词在文档中的权重。

tf-idf = tf(词频) × idf (逆向文档频率) 利用逆向文档频率来控制约束词频

tf = n / N

n: 词语在某篇文本中出现的频率 ；

n / N 的目的是实现归一化， N ：该文件中所有词汇的数目

idf = log(D / d)

D: 总的文档数

d: 词语所在的文档数

2. 计算余弦相似度

计算查询和文档之间的余弦相似度。

```
# 计算查询和文档之间的余弦相似度
similarities = cosine_similarity(query_tfidf, document_tfidf)
```

3. 嵌入向量处理

(1) 提取训练集中的查询和文档嵌入向量并进行归一化处理

```
# 提取训练集中的嵌入向量
train_query_embeddings = [train["query_embedding"] for train in train_data]
train_fact_embeddings = [evidence["fact_embedding"] for train in train_data for evidence

# 将训练集中的嵌入向量进行归一化处理
normalized_train_query_embeddings = normalize(np.array(train_query_embeddings))
normalized_train_fact_embeddings = normalize(np.array(train_fact_embeddings))
```

(2) 转换测试集查询文本的嵌入向量

```
# 转换测试集查询文本的嵌入向量
query_embeddings = [query["query_embedding"] for query in queries]
normalized_query_embeddings = normalize(np.array(query_embeddings))
```

(3) 计算查询和训练集文档嵌入向量之间的余弦相似度

```
# 计算查询和训练集文档嵌入向量之间的余弦相似度
embedding_similarities = np.dot(normalized_query_embeddings, normalized_train_fact_embeddings.T)
```

4. 综合排序

(1) 使用 TF-IDF 获取最相似的前 200 个文档索引

```
# 对每个查询进行处理
for i, query in tqdm.tqdm(enumerate(queries), total=len(queries)):
    # 使用TF-IDF获取最相似的前200个文档索引
    top_indices = similarities[i].argsort()[-200:][::-1]
    candidate_texts = [document_texts[idx] for idx in top_indices]
```

(2) 对选出的 200 个文档应用 BM25 算法

```
# 对选出的200个文档应用BM25
bm25 = BM25Okapi([doc.split() for doc in candidate_texts], k1=0.75, b=0.75)
bm25_scores = bm25.get_scores(query_texts[i].split())
```

(3) 计算训练集嵌入相似度

```
# 计算训练集嵌入相似度
embedding_sim_scores = embedding_similarities[i].argsort()[-200:][::-1]
```

(4) 综合 TF-IDF、BM25 和嵌入相似度得分

```
# 组合TF-IDF、BM25和嵌入相似度得分
# combined_scores = [(alpha * bm25_scores[j] + (1 - alpha) * similarities[i][top_indices[j]] + embedding_similarities[i
combined_scores = [(0.4 * bm25_scores[j] + 0.3 * similarities[i][top_indices[j]] + 0.3 * embedding_similarities[i][
    embedding_sim_scores[j]]) for j in range(30)]
```

(5) 进行 ReRank

```
# 重新排序后获取最相关的前3个文档
top_indices_combined = np.argsort(combined_scores)[-3:][::-1]

# 映射回原始文档索引
final_top_indices = [top_indices[idx] for idx in top_indices_combined]
evidence_list = [{"fact_input_list": documents[idx]["fact_input_list"]} for idx in final_top_indices]
```

检索后 rerank 是为了更好得到更加相近的文档，在本次实验中是使用计算查询与文档的 TF-IDF 相似度、BM25 得分和嵌入向量相似度，并结合上述三个特征，计算综合得分。

三、实验结果

Recall@3:0.2699

MRR@3:0.2109

其实在本次实验的过程中我真的尝试了多种多样的模型，从最简单的余弦相似度再到 BM_25,TF-IDF,BERT，神经网络，注意力机制等等，几乎所有课上学过的内容我都有去尝试，但还是没能得到很好的结果，后续我也标注了数据的起始符号和终止符号，也进行了各种各样的处理，但随着越来越优化，模型越来越强大，反而造成结果变得更差，甚至结果一度达到了 0。日后还是继续加油吧。