تمرین سوم

موتور جست و جوی اخبار

شایان محمدی زاده سماکوش ۹۸۱۰۲۲۷۳

نونا قاضی زاده ۹۸۱۷۱۰۰۷

مقدمه

در این تمرین هدف ما پیاده‌سازی یک موتور جستجو است بدین صورت که ابتدا دیتا مورد نیاز که موضوع و متن اخبارها می‌باشد را کراول می‌کنیم و سپس پیش پردازش‌های لازم را روی آن انجام می‌دهیم، انتخاب اینکه چه پیش پردازش‌هایی رو متن با صورت بگیرد را با روش آزمون و خطا انجام داده‌ایم و در نهایت با استفاده از چهار روش boolean, tf-idf, transformers, fasttext بهره می‌گیریم و در نهایت با استفاده از نتایجی که به دست می‌آوریم و معیار MRR نتیجه گیری می‌کنیم کدام روش برای جست و جو بهتر است.

پیاده‌سازی

توجه: پیش از گرفتن خروجی از پروژه باید فایل‌های crawler را برای داشتن دیتای عنوان و متن اخبار ران کنیم.

پیش پردازش

برای این بخش ابتدا متن اخبار را نرمالایز می‌کنیم. سپس با توکنایز کردن متن اخبار آن را به کلمات می‌شکانیم، سپس از stopword هایی که در کلاس داده شده بود استفاده می‌کنیم، اما از آنجا که این stopword ها کامل نیستند و تمام کلمات اضافه و علائم نگارشی را ندارند بنابراین یک فایل دیگر ایجاد می‌کنیم و در این فایل سایر کلمات اضافه و علائم نگارشی که نیاز داریم را می‌افزاییم و بعد از مرحله توکنایزیشن این کلمات را حذف می‌کنیم و سپس با استفاده از stemming یا lemmatization کلمات را به ریشه‌شان می‌بریم. همچنین با روش آزمون و خطا متوجه می‌شویم که در صورت حذف کلمات با فرکانس بالا و پایین هر کدام از روش ها دچار مشکلاتی می‌شوند به طور مثال اگر کلمات با فرکانس کم یا زیاد را حذف کنیم به علت روش بولین که دقیقا با جستجوی خود کلمه است صورت می‌گیرد باعث ایجاد خطا می‌شوند بنابراین این کار را برای پیش پردازش انجام نمی‌دهیم

بازیابی boolean

در بازیابی بولین بدین صورت عمل می‌کنیم که در صورتی که کلمات کوئری دقیقا در متن اخبار بیاید آن عنوان خبر را باز می‌گردانیم مدلمان بدین صورت است که یک ماتریس در نظر می‌گیریم که سطر های ماتریس بیانگر اخبار و ستون های آن بیانگر کلمات کوئری می‌باشد سپس در صورتی که کلمه در خبر وجود داشته باشد آن درایه را برابر با یک می‌کنیم و در نهایت سطر که تمام درایه‌هایش برابر یک باشد همان خبر مدنظر ماست.

```
[ ]  query_words = query.split(" ")
     boolean_retrival = [[0 for _ in range(len(query_words))] for __ in range(len(news_df))]
     for i in range(len(news_df)):
       keywords = news_df.iloc[i].clean_keyword
       for j,q_word in enumerate(query_words):
           if q_word in keywords.split(","):
               boolean_retrival[i][j] = 1
     boolean_df = pd.DataFrame(boolean_retrival, columns = query_words)
     boolean_df
```

```
[ ]  converted_df = boolean_df.all(axis='columns')
     converted_df = converted_df.to_frame('res')
     boolean_related_docs_index = converted_df.index[converted_df['res'] == True].tolist()
     boolean_related_titles = []
     for idx in boolean_related_docs_index:
       boolean_related_titles.append(news_df.iloc[idx,0])
```

```
[ ]  for idx,title in enumerate(boolean_related_titles[:DOC_RELATED_NUM]):
       print(f"{idx+1}    {title}")
```

بازیابی tf-idf

در بازیابی tf-idf بدین صورت عمل می‌کنیم که با استفاده از کتابخانه زیر مدلمان را fit می‌کنیم و سپس بین این مقدار و کوئری فاصله کسینوسی را محاسبه می‌کنیم و ده اخباری که نزدیک‌تر به کوئری مدنظرمان باشد را به کاربر می‌دهیم

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[ ]  vocabulary = set()
     for doc in news_df.clean_keyword:
         vocabulary.update(doc.split(','))
     vocabulary = list(vocabulary)
```

```
[ ]  tfidf = TfidfVectorizer(vocabulary=vocabulary, use_idf = True, dtype=np.float32)
     tfidf_tran=tfidf.fit_transform([' '.join(doc) for doc in removed_tokenized_words])
```

```
[ ]  query_vec = tfidf.transform([query])
     tfidf_results = cosine_similarity(tfidf_tran,query_vec).reshape((-1,))\
```

```
[ ]
     for idx,title in enumerate(tfidf_results.argsort()[-10:][::-1]):
         print(f"{idx+1}    {news_df.iloc[title,0]}")
```

بازیابی به کمک مدلی بر پایه transformer ها

برای بازیابی با روش ترنسفرمرها از کتابخانه زیر bigbird که در زبان فارسی نیز توسعه یافته است استفاده می‌کنیم و ابتدا پاراگراف ما را توکنایز می‌کند و سپس برای هر کلمه یک embedding ایجاد می‌کند سپس برای به دست آوردن embedding

برای کل خبر میان تمام embedding های خبر میانگین می‌گیریم و در نهایت همچنین این مدل را روی کوئری‌مان محاسبه می‌کنیم و در نهایت بین embedding های کوئری نیز میانگین می‌گیریم و فاصله کسینوسی بین کوئری و متن خبرهای متفاوت را محاسبه می‌کنیم و ده نزدیک ترین به کوئری مان را به کاربر می‌دهیم.

```
from transformers import BigBirdModel,AutoTokenizer
```

```
[ ] from transformers import BigBirdModel, AutoTokenizer

    MODEL_NAME = "SajjadAyoubi/distil-bigbird-fa-zwnj"
    model = BigBirdModel.from_pretrained(MODEL_NAME, block_size=32)
    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
```

```
[ ] docs_embedding_tran = {}
    for idx, doc in enumerate(news_df.clean_keyword):
      text = ' '.join(doc.split(','))
      tokens = tokenizer(text, return_tensors='pt')
      try:
        output = model(**tokens)
      except Exception as e:
        text = ' '.join(doc.split(',')[:1000])
        tokens = tokenizer(text, return_tensors='pt')
        output = model(**tokens)
      docs_embedding_tran[idx] = output[0][0].detach().numpy()
```

```
[ ] docs_embedding_tran_avg = {}
    for k,v in docs_embedding_tran.items():
      sum = np.zeros(768)
      for idx,word_embedding in enumerate(v):
        sum = np.sum([sum, docs_embedding_tran[k][idx]], axis=0)
      docs_embedding_tran_avg[k] = sum /len(docs_embedding_tran[k])
```

```
[ ] qtokens_tran = tokenizer(query, return_tensors='pt')
    qoutput_tran = model(**qtokens_tran)
    query_embedding_tran = qoutput_tran[0][0].detach().numpy()
    qsum_tran = np.zeros(768)
    for qidx,qword in enumerate(query_embedding_tran):
      qsum_tran = np.sum([qsum_tran, query_embedding_tran[qidx]], axis=0)
    q_avg_tran = qsum_tran/len(query_embedding_tran)
```

```
[ ] docs_cosine_similarity_tran = {}
    for idx,doc_embedding in docs_embedding_tran_avg.items():
      docs_cosine_similarity_tran[idx] = np.dot(doc_embedding, q_avg_tran) / (np.linalg.norm(doc_embedding)*np.linalg.norm(q_avg_tran))

    docs_cosine_similarity_tran = reversed(sorted(docs_cosine_similarity_tran.items(), key=lambda x:x[1]))
    sort_docs_cosine_similarity_tran = dict(docs_cosine_similarity_tran)
```

```
[ ] for idx, doc_id in enumerate((list(sort_docs_cosine_similarity_tran.items()))[:10]):
      print(f"{idx+1}   {news_df.iloc[doc_id[0]].title}")
```

بازیابی با استفاده از میانگین وزن‌دار بردار های تعبیه (fasttext)

برای بازیابی با روش میانگین وزن‌دار بردارهای تعبیه از کتابخانه fasttext استفاده می‌کنیم بدین صورت که این مدل برای هر کلمه یک embedding ایجاد می‌کند سپس برای به دست آوردن embedding برای کل خبر میان تمام embedding های خبر میانگین می‌گیریم و در نهایت همچنین این مدل را روی کوئری‌مان محاسبه می‌کنیم و در نهایت بین embedding های کوئری نیز میانگین می‌گیریم و فاصله کسینوسی بین کوئری و متن خبرهای متفاوت را محاسبه می‌کنیم و ده نزدیک ترین به کوئری مان را به کاربر می‌دهیم.

```
[ ] tokens_list = news_df.clean_keyword.tolist()
    all_news_content = ''
    for doc in tokens_list:
        all_news_content += ' '.join(doc.split(",")) + '\n'

[ ] with open('/content/data.txt', 'w') as text_file:
        text_file.write(all_news_content)

[ ] fasttext_model = fasttext.train_unsupervised('/content/data.txt', model='skipgram', minCount=4)

[ ] docs_embedding_avg = {}
    for idx,doc in enumerate(news_df.clean_keyword):
      sum = np.zeros(100)
      for word in doc.split(','):
        sum = np.sum([sum,fasttext_model[word]],axis=0)
      docs_embedding_avg[idx] = sum / len(doc.split(','))

[ ] query_sum = np.zeros(100)
    for qword in query_words:
      query_sum = np.sum([query_sum, fasttext_model[qword]], axis = 0)
    query_embedding_avg = query_sum / len(query_words)

[ ] docs_cosine_similarity = {}
    for idx,doc_embedding in docs_embedding_avg.items():
      docs_cosine_similarity[idx] = np.dot(doc_embedding, query_embedding_avg) / (np.linalg.norm(doc_embedding)*np.linalg.norm(query_embedding_avg))

    docs_cosine_similarity = reversed(sorted(docs_cosine_similarity.items(), key=lambda x:x[1]))
    sort_docs_cosine_similarity = dict(docs_cosine_similarity)

[ ] for idx, doc_id in enumerate((list(sort_docs_cosine_similarity.items()))[:10]):
      print(f"{idx+1}   {news_df.iloc[doc_id[0]].title}")
```

نمونه خروجی

نمونه‌های خروجی در گوگل داک با توجه به نحوه‌ای که گفته شده بود قرار داده شده است و نظر هر یک از اعضای تیم نیز در آن آورده شده است همچنین در فولدر results هر روش csv مخصوص به خود را دارد که در آن آمده است و در نهایت معیار MRR را برای هر کدام به صورت زیر محاسبه کردیم.

Boolean MRR:

Q1: (1+1/9)/2

Q2: (1/5+1/5)/2

Q3: (1/2+1)/2

Q4: (1+1)/2

Q5: (1/3+1/3)/2 => 4.746/10 = 0.4746

Q6: (1/4+1/5)/2

Q7: (1+1/3)/2

Q8: (1+1/2)/2

Q9: (1/9+1/9)/2

Q10: (1/2+1/7)/2

TF-IDF MRR:

Q1: (1/3+1/3)/2

Q2: (1+1)/2

Q3: (1+1)/2

Q4: (1+1)/2

Q5: (1+1/3)/2 => 6.755/10 = 0.6775

Q6: (1/2+1/2)/2

Q7: (1/3+1/3)/2

Q8: (1/5+1/7)/2

Q9: (1+1)/2

Q10: (1+1/2)/2

Transformers MRR:

Q1: (1+1)/2

Q2: (1+1/2)/2

Q3: (1+1)/2

Q4: (1+1/2)/2

Q5: (1+1)/2 => 8.667/10 = 0.8667

Q6: (1+1)/2

Q7: (1+1)/2

Q8: (1/3+1)/2

Q9: (1+1)/2

Q10: (1/2+1/2)/2

Fasttext MRR:

Q1: (1+1)/2

Q2: (1/2+1/2)/2

Q3: (1+1)/2

Q4: (1+1)/2

Q5: (1+1)/2 => 7.4167/10 = 0.74167

Q6: (1+1/2)/2

Q7: (1+1)/2

Q8: (1/4+1/3)/2

Q9: (1/4+1/6)/2

Q10: (1/3+1)/2

با توجه به معیار MRR می توان نتیجه گرفت بهترین نتیجه جستجو مربوط به روش transformer ها و سپس fasttext و بعدی td-idf و در نهایت بولین است. ترکیب دو روش transformer ها و بولین که elasticsearch بر مبنای آن است بهترین روش برای جست و جو است.