HA4

March 20, 2023

1 HA4 by Adil Akhmetov

[29]:
```
from ir_measures import *
import ir_measures
from elasticsearch.helpers import parallel_bulk
import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import time
from elasticsearch import Elasticsearch
from sentence_transformers import SentenceTransformer, util
%config IPCompleter.greedy=True
from time import time
import csv
import torch
import pandas as pd
```

-

1.1 Re-rank top20 documents returned for WikiIR test queries by Elasticsearch (or another retriever used in HA2)

We already have top20 documents returned for WikiIR test queries by Elasticsearch from HA2. I only modified the results functions to add the content and query itself. Now, the `corpuses.csv` file contains the following columns:

```
- queryId: the query id
- docId: the document id
- score: the score returned by Elasticsearch
- query: the query itself
- content: the content of the document
```

The modified code to generate `corpuses.csv` file:

```
def run_test_queries():
    with open('../wikIR1k/test/queries.csv', 'r') as csvfile:
        with open("corpuses.csv", 'w') as file:
            writer = csv.writer(file)
            writer.writerow(['queryID', 'docID', 'score', 'query', 'content'])
            reader = csv.reader(csvfile)
            next(reader, None)
            for row in reader:
                query_id = int(row[0])
                query = {
                    'query': {
                        'bool': {
                            'should': {
                                'match_phrase': {
                                    'content.stemmed': row[1]
                                }
                            }
                        }
                    }
                }
                res = es.search(index='wikir', body=query, size=20)['hits']
                for hit in res['hits']:
                    data = [query_id, hit['_id'],
                            hit['_score'], row[1], hit['_source']['content']]
                    writer.writerow(data)
```

•

1.2 Use cosine similarity between query and document embeddings to rank documents

[11]:
```
model = SentenceTransformer('msmarco-distilbert-dot-v5')
```

[8]:
```
def rerank(query, queryId, corpus, writer, top = 20):
        query_embedding = model.encode(query, convert_to_tensor=True)
        corpus_embeddings = model.encode(
                [content for _, content in corpus], convert_to_tensor=True)

        top_k = min(len(corpus), top)

        cos_scores = util.cos_sim(query_embedding, corpus_embeddings)[0]
        top_results = torch.topk(cos_scores, k=top_k)

        c = 0
        for score, idx in zip(top_results[0], top_results[1]):
                writer.writerow([queryId, 0, corpus[idx][0], c, score.item(),␣
 ↪'CosSim'])
                c += 1
```

[20]:
```
with open('../seminar_elastic/corpuses.csv', 'r') as corpuses:
    with open('corpuses.run', 'w') as runs:
        reader = csv.reader(corpuses)
        writer = csv.writer(runs, delimiter=' ')
        next(reader)
        corpus = []
        query = None
        queryId = None

        while (row := next(reader, None)) is not None:
            if queryId is None:
                        queryId = row[0]
                        query = row[3]

            if row[0] != queryId:
                rerank(query, queryId, corpus, writer)

                queryId = row[0]
                query = row[3]
                corpus = []
            corpus.append((row[1], row[4]))

        rerank(query, queryId, corpus, writer)
```

•

1.3 Evaluate new rankings using P@10, p@20, MAP@20

[30]:
```
qrels = ir_measures.read_trec_qrels('../wikIR1k/test/qrels')
run = ir_measures.read_trec_run('./corpuses.run')
ir_measures.calc_aggregate([P@10, P@20, MAP], qrels, run)
```

[30]:
```
{P@10: 0.174, AP: 0.11989138021776306, P@20: 0.10949999999999999}
```

1.4 Task*

Re-rank documents based on a combination of BM25 scores and cosine similarity of query/document embeddings

•

1.4.1 Sample 100-500 queries from train subset

•

1.4.2 Get top50 documents for each query using Elasticsearch (BM25)

[2]:
```
es = Elasticsearch(
    "https://localhost:9200",
    verify_certs=False,
    ssl_show_warn=False,
    basic_auth=("elastic", "gdJplaadFKhdJuBysJDX")
)
```

```
[19]: runs = []
      SCORES = []
```

```
[20]: def run_training_queries():
          with open('../wikIR1k/training/queries.csv', 'r') as csvfile:
              reader = csv.reader(csvfile)
              next(reader, None)
              line = 0
              for row in reader:
                  if line > 100:
                      break
                  line += 1

                  query_id = int(row[0])
                  query = {
                      'query': {
                          'bool': {
                              'should': {
                                  'match_phrase': {
                                      'content.stemmed': row[1]
                                  }
                              }
                          }
                      }
                  }
                  res = es.search(index='wikir', body=query, size=50)['hits']

                  scores = [hit['_score'] for hit in res['hits']]
                  SCORES.extend(scores)

                  for hit in res['hits']:
                      data = [query_id, hit['_id'],
                              hit['_score'], row[1], hit['_source']['content']]
                      runs.append(data)
```

```
[21]: start = time()
      run_training_queries()
      print(f"Time spent {time() - start:0.2f} seconds")
```

```
/var/folders/k6/3zkmms5n2j3_c0wt5q78hm040000gn/T/ipykernel_54418/2159404077.py:2
3: DeprecationWarning: The 'body' parameter is deprecated and will be removed in
a future version. Instead use individual parameters.
  res = es.search(index='wikir', body=query, size=50)['hits']
```

```
Time spent 2.32 seconds
```

-

1.4.3 Min-max normalize BM25 scores, so they are in the range [0,1]

[22]:
```
for run in runs:
    min_max = (run[2] - min(SCORES)) / (max(SCORES) - min(SCORES))
    run.append(min_max)
```

[23]:
```
runs[0]
```

[23]:
```
[123839,
 '806300',
 16.740032,
 'yanni',
 'it is a compilation of tracks from his five previous studio albums released
between 1980 and 1989 plus three new compositions yanni was encouraged to
release the album by his then partner actress linda evans reflections of passion
became yanni s fastest selling and most successful album of his career upon
release it reached no 1 on the billboard top new age albums chart and no 29 on
the billboard 200 yanni supported the album with a nationwide concert tour in
1990 that featured his band and an orchestra in 1995 it was certified double
platinum for selling 2 million copies in the us in august 1989 yanni released
his fifth studio album niki nana the album marked his stylistic development from
solo keyboard music towards rock with the addition of additional vocalists
musicians and choir around the same time of its release yanni s newfound
relationship with american actress linda evans who had become a fan of his music
received press attention not long into their relationship evans pitched an idea
she had for an album from yanni which was long in duration reflected a single
mood throughout and something that she could play at dinner parties she
mentioned',
 0.37955932350949895]
```

-

1.4.4 Get cosines for query/document embeddings

-

1.4.5 Find an alpha that maximizes MAP@20 on train data `alpha*BM25 + (1-alpha)*q_d_cosine_similarity`

[24]:
```
def rerank_v2(query, queryId, corpus, writer, alpha = 0.1):
        query_embedding = model.encode(query, convert_to_tensor=True)
        corpus_embeddings = model.encode(
                [content for _, content, _ in corpus], convert_to_tensor=True)

        top_k = min(len(corpus), 50)

        cos_scores = util.cos_sim(query_embedding, corpus_embeddings)[0]
        top_results = torch.topk(cos_scores, k=top_k)
        c = 0
        for score, idx in zip(top_results[0], top_results[1]):
                score = alpha * corpus[idx][2] + (1 - alpha) * score.item()
                writer.writerow([queryId, 0, corpus[idx][0], c, score,␣
↪'CosSimBM25'])
                c += 1
```

[61]:
```
def getMap20(alpha):
        with open(f'{alpha}_runs.csv', 'w') as csvfile:
                writer = csv.writer(csvfile, delimiter=' ')

                corpus = []
                query = None
                queryId = None

                for run in runs:
                        if queryId is None:
                                queryId = run[0]
                                query = run[3]

                        if run[0] != queryId:
                                rerank_v2(query, queryId, corpus, writer, alpha)

                                queryId = run[0]
                                query = run[3]
                                corpus = []
                        corpus.append((run[1], run[4], run[5]))

                rerank_v2(query, queryId, corpus, writer, alpha)

        qrels = ir_measures.read_trec_qrels('../wikIR1k/training/qrels')
        return ir_measures.calc_aggregate([MAP@20], qrels, ir_measures.
↪read_trec_run(f'{alpha}_runs.csv'))
```

[62]:
```
map01 = getMap20(0.1)
```

[64]:
```
map03 = getMap20(0.3)
```

[66]:
```
map05 = getMap20(0.5)
```

[67]:
```
map08 = getMap20(0.8)
```

[68]:
```
print(map01, 'map01')
print(map03, 'map03')
print(map05, 'map05')
print(map08, 'map08')
```

```
{AP@20: 0.012652878728836033} map01
{AP@20: 0.012514469132369439} map03
{AP@20: 0.012203960977199892} map05
{AP@20: 0.011710759272232175} map08
```

-

1.4.6 Apply the formula to the test data (again, to top50)

[69]:
```
def run_test_queries():
    with open('../wikIR1k/test/queries.csv', 'r') as csvfile:
        reader = csv.reader(csvfile)
        next(reader, None)
        line = 0
        for row in reader:
            if line > 100:
                break
            line += 1

            query_id = int(row[0])
            query = {
                'query': {
                    'bool': {
                        'should': {
                            'match_phrase': {
                                'content.stemmed': row[1]
                            }
                        }
                    }
                }
            }
            res = es.search(index='wikir', body=query, size=50)['hits']

            scores = [hit['_score'] for hit in res['hits']]
            SCORES.extend(scores)

            for hit in res['hits']:
                data = [query_id, hit['_id'],
                        hit['_score'], row[1], hit['_source']['content']]
                runs.append(data)
```

[72]:
```
runs = []
SCORES = []
```

[73]:
```
start = time()
run_test_queries()
print(f"Time spent {time() - start:0.2f} seconds")
```

```
/var/folders/k6/3zkmms5n2j3_c0wt5q78hm040000gn/T/ipykernel_54418/816007642.py:23
: DeprecationWarning: The 'body' parameter is deprecated and will be removed in
a future version. Instead use individual parameters.
  res = es.search(index='wikir', body=query, size=50)['hits']
```

```
Time spent 2.09 seconds
```

[74]:
```
for run in runs:
    min_max = (run[2] - min(SCORES)) / (max(SCORES) - min(SCORES))
    run.append(min_max)
```

[75]:
```
with open(f'test_runs.csv', 'w') as csvfile:
                writer = csv.writer(csvfile, delimiter=' ')

                corpus = []
                query = None
                queryId = None

                for run in runs:
                        if queryId is None:
                                queryId = run[0]
                                query = run[3]

                        if run[0] != queryId:
                                rerank_v2(query, queryId, corpus, writer, 0.1)

                                queryId = run[0]
                                query = run[3]
                                corpus = []
                        corpus.append((run[1], run[4], run[5]))

                rerank_v2(query, queryId, corpus, writer, 0.1)
```

[77]:
```
qrels = ir_measures.read_trec_qrels('../wikIR1k/test/qrels')
run = ir_measures.read_trec_run('./test_runs.csv')
ir_measures.calc_aggregate([P@10, P@20, MAP], qrels, run)
```

[77]:
```
{P@10: 0.176, AP: 0.13685233493777194, P@20: 0.12200000000000003}
```