SI 650: Homework 2

Name: Junqi Chen

Uniqname: junqich

UMID: 03846505

NOTE: The name I use in *Kaggle* is Junqi Chen.

This file includes documentation on *indexing* and *customized ranker*.

Indexing

Before running the `main.py`, we have to deal with the data for the function `IndexReader` with following codes:

```
# transfer the csv to matching json
def csv_to_json(i = 1, filename = 'documents'):
    df = pd.read_csv("./data" + str(i) + "/" + filename + ".csv")
    df = df[["DocumentId", "Title", "Document Description"]] \
                .rename(columns={"DocumentId": "id", "Title": "title", "Document
Description": "contents"})
    doc_df = pd.DataFrame({"id": df["id"], "contents": [str(t) + " " + str(t) + " " +
str(c) for t, c in zip(df["title"], df["contents"])]})
    result = doc_df.to_json(orient="records")
    parsed = json.loads(result)
    with open("./data" + str(i) + "/" + filename + ".json", 'w') as outfile:
        json.dump(parsed, outfile, indent = 4)

csv_to_json(1, 'documents')
csv_to_json(2, 'documents_gaming')
csv_to_json(3, 'documents_android')
```

And run the following command lines:

```
! python -m pyserini.index -collection JsonCollection -generator
DefaultLuceneDocumentGenerator -threads 1 -input ./data1/ -index ./data1/index -
storePositions -storeDocvectors -storeContents
! python -m pyserini.index -collection JsonCollection -generator
DefaultLuceneDocumentGenerator -threads 1 -input ./data2/ -index ./data2/index -
storePositions -storeDocvectors -storeContents
! python -m pyserini.index -collection JsonCollection -generator
DefaultLuceneDocumentGenerator -threads 1 -input ./data3/ -index ./data3/index -
storePositions -storeDocvectors -storeContents
```

Then you will get the following file tree.

```
Assignment 2
|  main.py
|  rankers.py
├─data1
|  |  documents.csv
|  |  documents.json
|  |  query.csv
```

```
|   |
|   └─index.
├─data2
|   |   documents_gaming.csv
|   |   documents_gaming.json
|   |   gaming_query_sample_submission.csv
|   |   query_gaming.csv
|   |
|   └─index.
├─data3
|   |   android_query_sample_submission.csv
|   |   documents_android.csv
|   |   documents_android.json
|   |   query_android.csv
|   |
|   └─index.
```

Ranking

Run the `main.py` with following arguments:

```
python main.py [dataset_id] [index_dirname] [query_filename] [ranker]
```

where

- `dataset_id`: number of dataset you want to use (select among `1`, `2` and `3`)
- `index_dirname`: name of index directory (default = `index`)
- `query_filename`: name of file as query
- `ranker`: ranker function (select among `PLN`, `BM25` and `Custom`)

For example,

```
python main.py 1 index query.csv PLN
```

Then you will get the `output[i].csv` in a submission format.

Customized Ranker

In the customized ranker, I was enlighten by th `PLN` and `BM25` and design the following ranking function:

$$S(Q,D) = \sum_{t \in Q \cap D} \mathrm{idf} \cdot \mathrm{tf} \cdot \mathrm{qtf} \cdot \frac{\mathrm{tot\_term}}{\mathrm{cf}}$$

where idf, tf, qtf terms are variant from the original one with normalization. The $\mathrm{tot\_term}$ refers to number of terms in all collection (after analyzed) and cf is collection frequency of a term where

This function inherits from `BM25` but considers an additional collection frequency term. The adoption of the $\frac{\mathrm{tot\_term}}{\mathrm{cf}}$ term is used to put more weight on the term frequency in the collection. Consider a situation where we have only a few matching terms in whole collection, what we want is exactly documents have that term. However, if there are so many matching terms in the collection, the specific document with some terms may not be the relevant result since the term may be a frequent word. Simply speaking, this function put more weight on documents with "seldom"-used terms but reduce the influence of "frequently"-used terms so that it can return a better prediction result.

We then perform normalization on the term $\frac{\mathrm{tot\_term}}{\mathrm{cf}}$ so that: (I think a log function may be useful here)

$$\frac{\text{tot\_term}}{\text{cf}} = \frac{k_3}{(k_3+1)\cdot\frac{\text{tot\_term}}{c(t,C)}}$$

The overall equation shows like:

$$S(Q,D) = \sum_{t\in Q\cap D} \ln\left(\frac{N+1}{df(t)+1}\right)\cdot\frac{(k_1+1)\cdot c(t,D)}{k_1\left(1-b+b\frac{|D|}{avdl}\right)+c(t,D)}\cdot\frac{(k_2+1)\cdot c(t,Q)}{k_2+c(t,Q)}\cdot\frac{k_3}{(k_3+1)\cdot\frac{\text{tot\_term}}{c(t,C)}}$$

The default parameter of the customized ranker is:

- $k1 = 1.2$
- $k2 = 1.2$
- $k3 = \frac{1}{1000}$
- $b = 1$

This hyper-parameter set is enlighten from `BM25` default one. And $k3$ is chosen according to the relationship between the tot_term and avg_cf dataset. The tot_term is dataset 3 is around 8×10^7 while avg_cf is around 2×10^4 (around 1000 times smaller), so we choose a $k3 = 0.001$ since to make the influence of additional term visible.

After tuning between the range $\frac{1}{10}$ to $\frac{1}{4000}$ (with discrete values like $\frac{1}{100}$, $\frac{1}{250}$, $\frac{1}{500}$, $\frac{1}{1000}$ and so on), we choose the value with best outcome for $k3 = \frac{1}{100}$.