Assignment #2

B05902120 / Yu-Ting, TSENG

May 24, 2019

Basic Execution

- Platform: Unix (MacBook)
- Language: Python 3.6

Vector Space Model

The concept is same as what we have done in Assignment 1. We give each document a score for ranking their relevance. To begin, we parse the organized files and transform the information into what we need. The most difficult part is to calculate the length of the document. I compute it through os function, however, we do not have the real documents this time. Therefore, I compute it via `tf` and `len(word)`, which are both mentioned in inverted file instead.

```
### Format: file = {file: length, ...}
avdl = 0
file = {}
for word in Invt:
    for item in Invt[word]['docs']:
        for name, tf in item.items():
            temp = tf * len(word)

            if name not in file: file[name] = 0
            file[name] += temp
            avdl += temp

print("[Done] Parsing \"FileLength\"!")
return file, avdl / len(file)
```

Next, go through all the terms and find the documents that contains the terms, obtaining other parameters which are needed by the formula. The formula we use is "Okapi".

$$\text{score}= \sum[\ln \frac{N - df + 0.5}{df + 0.5} \frac{(k_1 + 1)tf}{k_1(1 - b + b\frac{dl}{avdl} + tf} \frac{(k_3 + 1)qf}{k_3 + qf}], \text{ where}$$

N is the number of documents in the total collection,
df is the number of documents that contains the term,
dl is the length of the document,
tf is the time of term frequency within document,
qf is the time of term frequency within query.
k_1, k_3 and b are three values that we should decided.
In common case, b is usually 0.75. Worth mention is something strange I have met – it seems the thing becomes worse when using normalized `df`, I choose the given data `idf`. Only the middle part of the formula is based on documents, therefore we could calculate the other parts at begin.

```
### Format: rank = {file: value, ...}
indx = 0
vect = [0] * len(term)
rank = {}
for word, time in qcnt.items():
    if word not in invt: continue

    ### Parameters
    k1 = ### undefined ###
    k3 = ### undefined ###
    qf = time
    # df = N / math.e**invt[word]['idf']
    # DF = math.log((N - df + 0.5) / (df + 0.5))
    DF = invt[word]['idf']
    QF  = ((k3 + 1) * qf) / (k3 + qf)

    if DF > 900: continue
    if DF < 1: continue

    for item in invt[word]['docs']:
```

```
        for name, tf in item.items():

            dl = file[name]
            TF = ((k1 + 1) * tf) / (k1 * (0.25 + 0.75 * dl
                / avdl) + tf)

            ### Add into Rank List
            if name not in rank: rank[name] = [0] * len(
               term)
            rank[name][indx] = float(DF * TF * QF)

    ### Add into Vect List
    vect[indx] = time
    indx += 1
```

Rocchio Relevance Feedback

What we do as feedback is same as the concept mentioned last time.

$$q_{new} = \alpha q_{original} + \frac{\beta}{|D_r|} \sum_{\forall q_{im} \in D_r} q_{im} - \frac{\gamma}{|D_n|} \sum_{\forall q_{im} \in D_n} q_{im}$$

However, the question we meet is that we don't actually know which is relevant and which is not. We could view those whose score is high originally as more relevant document, and irrelevant vice versa.

```
reln = int(len(Rank) * 0.15)

### Compute original
Original = np.array(Vect)

### Compute related
Related = np.array([0] * len(Vect))
for item in sort[:reln]:
    Related = Related + np.array(Rank[item[0]])
Related = np.array(Related)
```

```
### Compute irrelated
NotRelated = np.array([0] * len(Vect))
for item in sort[-reln:]:
    NotRelated = NotRelated + np.array(Rank[item[0]])
NotRelated = np.array(NotRelated)
```

Results and Discussion

What we might find special must be that the terms with high `idf` value have make the performance worse. While filtering them out of consideration, it seems to be good. It makes sense, since those terms whose `idf` values are large indicates that the terms is too specific and should not be considered that much.

Another worthy mentioning point is that while I compute Okapi with normalized `df`, thing becomes worse. The first situation I was trapped is that the recovered `df` is too large, and even overflow! Then I use log function while computing to solve this problem, however the performance become worse. Eventually, I choose to use the `idf` given already.