Assignment #1

B05902120 / Yu-Ting, TSENG

Apr 13, 2019

Basic Execution

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

Vector Space Model

The concept is to give each document a score for ranking their relevance. To begin, we separate the word in the query and calculate the frequency it appears. What's worthy noting is that we do not use bigram or unigram respectively, we instead combine those two – if there are just punctuation rather than a word after what we are considering, we would use unigram; otherwise, we would use bigram.

```
### Format: term{(word[0], word[1]): QuryTime}
for char in range(len(qury) - 1):
    ### Find the wordset
    temp = (qury[char], qury[char + 1])
    word = (vocb.index(temp[0]), vocb.index(temp[1]))

    ### Skip the punctuation
    if temp[0] in punctuation: continue
    if temp[1] in punctuation: word[1] = -1
    if word not in invt: continue

    ### Add into Term List
    if word not in term: term[word] = 0
    term[word] += 1
```

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. Only the middle part of the formula is based on documents, therefore we could calculate the other parts at begin.

```
### Format: rank = {file: [ScreTerm1, ScreTerm2, ...]}
for word, time in term.items():
    ### Parameters
    k1 = ### undecided ###
    k3 = ### undecided ###
    qf = time
    df = invt[word][0]
    DF = math.log((N-df+0.5) / (df+0.5))
    QF  = ((k3+1)*qf) / (k3+qf)

    for item in invt[word]:
        if type(item) != type((0, 0)): continue

        ### Based on document
        tf = item[1]
        dl = file[item[0]][1]
        TF = ((k1+1)*tf) / (k1*(0.25+0.75*dl/avdl)+tf)

        ### Add into Rank List
        if item[0] not in rank: rank[item[0]] = [0]*length
        rank[item[0]][indx] = float(DF * TF * QF)
    ### Add into Vect List
```

```
        vect.append(time)
        indx += 1
```

Rocchio Relevance Feedback

What we want to do is let the centroid of the documents close to the relevance documents as well as far from those irrelevance documents simultaneously. In order to achieve the goal, we set the weight and recompute the score, so that the centroid will move toward best result.

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

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

To sum up, the parameter under control would be k_1, k_3 of vector space model, as well as α, β and γ.

k_1	k_3	α	β	γ	**Precision**
0.0	0.2	0.9	0.3	0.2	0.74538
0.5	0.2	0.9	0.3	0.2	0.78516
1.0	0.2	0.9	0.3	0.2	0.79247
1.5	0.2	0.9	0.3	0.2	0.80189
2.0	0.2	0.9	0.3	0.2	0.79801
1.5	0.0	0.9	0.3	0.2	0.80092
1.5	0.1	0.9	0.3	0.2	0.80096
1.5	0.2	0.9	0.3	0.2	0.80189
1.5	0.3	0.9	0.3	0.2	0.79937
1.5	0.4	0.9	0.3	0.2	0.79458
1.5	0.5	0.9	0.3	0.2	0.79378

According to Greedy algorithm, when $k_1 = 1.5$ and $k_3 = 0.2$ has the best result. However, the precision seems not good enough; hence, we might want to give more tries. We could let α, β and γ be distinct values when dealing with different tag of queries. After several trial, we might find that,

```
if Labl == 1: a, b, c = 0.1, 0.0, 0.0    ### title
if Labl == 2: a, b, c = 0.7, 0.1, 0.0    ### question
if Labl == 3: a, b, c = 0.4, 0.1, 0.1    ### narrative
if Labl == 4: a, b, c = 0.9, 0.5, 0.0    ### concept
```

The precision would be 0.80189 in public score (above strong line). Another step, we can control is operating feedback or not. However, during my test, the results of them do not have significant difference. The table below displays the result of those with feedback and without feedback.

k_1	k_3	α	β	γ	Feedback	**Precision**
1.5	0.2	0.9	0.3	0.2	N	0.80189
1.5	0.2	0.9	0.3	0.2	Y	0.76856

Discussion

In my opinion, the concepts and structures of codes are not easy. The difficult we might meet is the parameters. Based on Greedy Algorithm, we might be able to find the optimized arrangement. Moreover, we might assume that the result with feedback would be better than those do not. In other words, we all know that the result of feedback would converge, however, we might observe that the precision do not truly increase. There are plenty of potential reason lead to the phenomenon, the most convincing possible reason – the relevant and irrelevant document we choose might not be accurate enough.