COM6115 Text Processing
Assignment
Nov.2019
Xingchen Xiao UCard No: 001727624

Summary of Code Implementation:

Our goal in this project is to measure the cosine similarity score between a document and the query.

The cosine similarity function is given as:

$$\cos(\vec{q},\vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q^2}\sqrt{\sum_{i=1}^{n} d^2}}$$

When we compute scores to rank the candidate documents for a single query (so that the top N can be returned), the component $\sqrt{\sum_{i=1}^{n} q^2}$ (for the size of the query vector) is a constant. It can be dropped without affecting how candidate are ranked so we have following:

$$\cos(\vec{q},\vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} d^2}}$$

Before I calculated the cosine similarity score, there are some required values to compute:

1. the total number of documents in the collection (|D|):

```
def docNum (self, index):
      ******
      Return len(doc_list)
```

2. The document frequency dfw of each term w:

```
def docFreq (self, index):
      ******
    return df
```

3. The inverse document frequency log(|D|/dfw):

```
def inverseDocFreq (self, index):
      ******
        return IDF
```

4. The size of each document vector $\sqrt{\sum_{i=1}^{n} d^2}$:

```
for term, documents in self.tfidf_dict.items():
     for document, weight in documents.items():
          self.doc_length_list[document-1]  += weight**2
self.squareDocLen()
```

The size of each document vector $\sqrt{\sum_{i=1}^{n} d^2}$ (Which also can be written as |d|) is computed for all documents at the same time in a single pass.

There are three term Weighting methods in this assignment:

Binary Weighting: Looping through each term in the index. And then Looping every term in the outer dictionary and looping every document and weight in all documents of appearing term. For each term containing the term, increment the element in the document length list. And I can get the size of each document vector $\sqrt{\sum_{i=1}^{n} d^2}$. And for each term in the query, checking if the term in the index therefore selecting documents. Setting up a list for the appearing term in the documents when term matches for such document, increment 1 in product of Q and D. Now, I have $\sqrt{\sum_{i=1}^{n} q^2}$ computed.

Term frequency weighting: The process for computing $\sqrt{\sum_{i=1}^{n} d^2}$ is same as I did in Binary weighting. However, the difference is the computing method is not going to increment the match element in the document length list. It will take square of each term weight and add to each document length. And for $\sqrt{\sum_{i=1}^{n} q^2}$, the method is also similar but different in computing. In q_d_sum list, the product of query term weight and document term weight are added to the document.

TF.IFD weighting: I used a function which called tfidf_f1 and tfidf_f2 to get dictionary of TFIDF value which is a two-level dictionary. I used map function to increase the running speed because it took a long time if I use for loop. This is the process I cycle through all documents for a given term and multiply the term frequency by IDF. The process for computing $\sqrt{\sum_{i=1}^{n} d^2}$ is same as I did in term frequency weighting. For computing $\sqrt{\sum_{i=1}^{n} q^2}$. I need to get the TFIDF weight for appearing term in the query before setting up the list of documents that has the term. Then adding the product between query term frequency and document term frequency to the element within document in q_d_sum.

Test Result:

Term Weighting:	**Binary**		
Scores:	precision	recall	F-measure
Default (none)	0.08	0.07	0.07
with Stemming	0.10	0.08	0.09
with Stop list	0.14	0.11	0.13
Having both	0.17	0.14	0.15

Term Weighting:	**Term Frequency**		
Scores:	precision	recall	F-measure
Default (none)	0.08	0.06	0.07
with Stemming	0.12	0.09	0.10
with Stop list	0.16	0.13	0.14
Having both	0.20	0.16	0.17

Term Weighting:	**TF.IDF**		
Scores:	precision	recall	F-measure
Default (none)	0.17	0.14	0.16
with Stemming	0.24	0.19	0.21
with Stop list	0.18	0.15	0.16
Having both	0.25	0.20	0.22

Conclusion:

In Binary scheme: The measurement with stop-list is better than the one with stemming. I believe it is because the binary method produces higher score if a term appears multiple times. The measurement with stemming does not show significant improvement.

In Term Frequency scheme: The situation in term frequency test is quite like what happened in the binary method. The measurement with stop list gets the better score than the one with stemming.

In **TF.IDF** method: The measurement with stemming improved a lot than the one with stop list

Overall, the three schemes having both stemming and stop list get the best F-measure. TF.IDF is the best scheme compared to binary and term frequency schemes. And it has better improvement by taking advantage of using stemming and stop list.