Assignment 2: TF-IDF

Jaiteg S Mundi

September 29, 2018

1 Introduction

This program calculates the Term Frequency - Inverted Document Frequency TF-IDF of a collection of documents TF-IDF is used to determine how relevant a document is given specific keywords. The documents are defined with a document id and description in a csv file. Three major functions are implemented in order to achieve the goal of this program.

TF-IDF helps to reduce weight of documents so it can be used increase efficency of scraping and searching tools.

1.1 tf(d, term)

The TF function accepts two inputs d (document id) and term (term to be searched in the document). In this function n(d): number of terms in a document and n(d,t): number of times term t occurs in document d are calculated. The following formula is implemented in this function:

$$TF(d,t) = log(1 + \frac{n(d,t)}{n(d)})$$

The results of the term frequency TF(d,t) are returned in a dictionary with term as key and TF as value. Sample outputs listed show below.

```
def tf(d, t):
....
        #Find the frequency of given term t in the document d
        for word in word:
                termCount = 0
                tFreq = 0
                termCount = termCount + counts[word]
                tFreq = (math.log(1+(termCount/docLength)))
                tfList[word] = tFreq
....
        return tfList
```

Sample Output from tf(d,t)

```
tf(2, blackberry)
returns: {'blackberry': 0.019418085857101516}
```

1.2 relevance(d, query)

The relevance function accepts two arguments d (document id) and query. The number of documents containing the query is calculated using nDocs function. The TF is obtained using the TF function and result of TF is divided by n(t) which is the number of documents containing query which gives us relevance of the query in document d. If the TF is returned 0 the relevance is returns 0 as well. The following formula is used.

$$relevance(d, Q) = \sum_{t \epsilon Q} \frac{TF(d, t)}{n(t)}$$

```
def relevance(d, query):

        #Calculate total number of times query appears in the corpus
        nDocsQ = nDocs(d, query)
        relv = 0

        #Get the tf
        TF = tf(d, query)

        #Test if terms found or not
        if(TF == 0):
                return 0

        #Calculate relevance
        sTF = sum(TF.values())
        relv = sTF/nDocsQ

        return relv
```

Sample Output from relevance(d,query)

```
relevance(2, honors)
returns: 0.010309643601367805
```

1.3 tf_idf(query, k)

The TF-IDF function accepts two inputs query (string with space deliminator) and k (top k results). If one of the query terms does not appear in the corpus, the function returns an empty list. No results with score of 0 are returned by this function. The method returns a list of k ids ranked in descending order. This function call relevance function to get the score. Then it zips the score and id into a list of python tuples. The return format is a list of python tuples containing (id, score).

```
def tf_idf(query, k):

.....

        #Calculate the score for the list of documents
        for x in range(len(docsList)):
                #Add the relevance to rel list
                rel.append(relevance(docsList[x], query))
                #Add the document ID to d list
                d.append(docsList[x])
                #Create a list of tuples using rel and d
                score = (list(zip(d, rel)))

        #Sort the score
        score = sorted(score, key=itemgetter(1), reverse=True)

        #Return k results in list of tuples
        return(score[:k])
```

Sample Output from tf_idf(d,query)

```
tf_idf(taste, 3)
returns:
[('3291', 0.0007998387568759352),
('3286', 0.0005557631422242855),
('999', 0.0005204704641734471)]
```

2 Sample of Database

The data parsed by this program is provided in a csv format with all lower case strings. The csv file contains two main columns id and description. This data is parsed into a dictionary data structure where keys are ids and values are description. Snippet of data is shown below.

id	description
243	round and velvet smooth it has blackberry fruits that are full in the mouth...
244	crisp and deliciously fruity this is lively and bright acidity points up......
245	full bodied and resplendent in ripe apple and brioche this well crafted wine..

Table 1: Sample Data from wine.csv

3 Tests

```
tf-idf(fruits, 3)
returns:
[('4627', 0.00010797989115947138),
('1026', 9.871919488764874e-05),
('1369', 9.338034761705664e-05)]

tf_idf(bubble gum, 3)
returns:
[('247', 0.019901975433120753),
('1651', 0.01346847154501298),
('3875', 0.012012337001785474)]

tf_idf(orange, 3)
returns:
[('1166', 0.0002281317353176194),
('722', 0.00019182436319306926),
('753', 0.00017583416202771341)]

tf_idf(chardonnay, 1)
returns:
[('1017', 0.00039689073723958747)]

tf_idf(vanilla, 2)
returns:
[('3065', 0.00014391549344312454), ('1402', 0.00013809708842012493)]

tf_idf(foo, 3)
returns:
[]
```