DOCUMENT LOCATOR

Using Spark 2.2

Khaled Hyder
Mdkhaled.hyder@ryerson.ca

Contents

Objective

This project aims to develop an application to find document location using Apache Spark. The idea is similar to a search engine, where a user query with a keyword will return the document location. TF-IDF computation used for indexing documents according to scores. TF-IDF is the short form of Term Frequency-Inverse Document Frequency. TF-IDF is one of the most frequently used techniques in text mining where some exciting use cases are, identifying document category, document search, text analysis, auto tag generation, etc. In response to users key-word based query, the application shall display the directories where documents are located.

Description:

To compute TF-IDF scores we need to identify the TF and IDF separately and then to multiply both values we can derive the TF-IDF scores. Here in below, we describe how we can find the desired outcome.

$$\text{TF} = \frac{No\ of\ times\ terms\ t\ found\ in\ a\ document}{Total\ number\ of\ terms\ in\ document}$$

$$\text{IDF(t)} = \log e \frac{Total\ document\ count}{No\ of\ document\ count\ with\ term\ t}$$

Value = TF * IDF

Technology and Platform:

Spark 2.2.0

Jupyter Notebook

Code:

```
# Reason why we have the getOrCreate code
# http://stackoverflow.com/questions/28999332/how-to-access-sparkcontext-in-pyspark-script
sc = SparkContext.getOrCreate()
#Creating RDD of text2 which load all files from football directory
#Using wholeTextFiles will create RDD which keeps file name as key and file contents as value.
#To parallelize the code we placed 6 on the method()
text2= sc.wholeTextFiles("C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/football/*.txt", 6)
```

In [2]:

```
# Display text2 RDD as key = file name and value = contents
```

```
text2.take(1)
```

Out[2]:

```
[(u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/001.txt',
  u'Man Utd stroll to Cup win . . . . . Referee: R Styles (Hampshire)
\n')]
```

In [3]:

```
#Counting the number of documents, here count function applied on key of text
2 RDD.
count_docs = text2.count()
```

In [4]:

```
#Display the number of documents.
count_docs
```

Out[4]:

```
265
```

In [5]:

```
#Import regular expression operation to split the line into words
import re
#Define a method to tokenize text into word
def tokenize(s):  #Create tokenize method
    return re.split('\W+', s.lower()) # convert all uppercase letters to lowe
rcase.
```

In [6]:

```
#Create tokenize_text2 RDD, which contain transformed text2 RDD after applyin
g map function on the value; tokenize()
tokenized_text2 = text2.map(lambda (title,text): (title, tokenize(text)))
```

In [7]:

```
#Apply flatMap function to each key, value pair of tokenized_text2 RDD withou
t changing the key and count the term/word
t_f = tokenized_text2.flatMapValues(lambda x: x).countByValue()
```

In [8]:

```
#Here the items() method returns a list of a  (key, value) tuple pair of docu
ment id and word. Here wer are showing 20 items [doc id, term, count]
t_f.items()[:10] #Display items before index 10
```

Out[8]:

```
[((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/087.txt',
   u'chairman'),
  1),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/159.txt',
   u'what'),
```

```
  2),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/209.txt',
   u'has'),
  5),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/030.txt',
   u'would'),
  2),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/124.txt',
   u'they'),
  16),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/083.txt',
   u'pleaded'),
  1),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/018.txt',
   u'felt'),
  1),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/032.txt',
   u'going'),
  1),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/023.txt',
   u'who'),
  2),
 ((u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/foot
ball/188.txt',
   u'lose'),
  1)]
```

In [9]:

```
# Created doc_freq rdd which contains the term/words and count of documents t
hey are present.
doc_freq = tokenized_text2.flatMapValues(lambda x: x).distinct()\
                        .map(lambda (title,word): (word,title)).countByKey()
```

In [10]:

```
doc_freq.items()[:10] #Display items before index 10
```

Out[10]:

```
[(u'', 265),
 (u'galactico', 1),
```

```
 (u'foul', 6),
 (u'four', 61),
 (u'francesco', 1),
 (u'plaudits', 1),
 (u'cyprus', 3),
 (u'looking', 34),
 (u'electricity', 1),
 (u'crossbar', 1)]
```

In [11]:

```
###### Compute TF-IDF scores: #######

import numpy as np

#Define a method, which produce result containing document id, TF-IDF score a
nd Term
def tf_idfcount(count_docs, t_f, doc_freq):
    #create a null result variable
    result = []
    for key, value in t_f.items():
        doc = key[0]
        term = key[1]
        df = doc_freq[term]
        if (df>0):
            #Calculate TF-IDF
          tf_idfcount = float(value)*np.log(count_docs/df)

        result.append({"doc":doc, "score":tf_idfcount, "term":term})
    return result
#Calling TF-IDF function and Display 1
tf_idf_out = tf_idfcount(count_docs, t_f, doc_freq)
```

In [12]:

```
tf_idf_out[:5] #Display items before index 5
```

Out[12]:

```
[{'doc': u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcspor
t/football/087.txt',
  'score': 2.1972245773362196,
  'term': u'chairman'},
 {'doc': u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcspor
t/football/159.txt',
  'score': 2.1972245773362196,
  'term': u'what'},
 {'doc': u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcspor
t/football/209.txt',
```

```
  'score': 0.0,
  'term': u'has'},
 {'doc': u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcspor
t/football/030.txt',
  'score': 1.3862943611198906,
  'term': u'would'},
 {'doc': u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcspor
t/football/124.txt',
  'score': 0.0,
  'term': u'they'}]
```

In [13]:

```
# The search Funtion takes the query and the number of Top related documents
to return

tfidf_RDD = sc.parallelize(tf_idf_out).map(lambda x: (x['term'],(x['doc'],x['
score']) )) # the corpus with tfidf scores
```

In [14]:

```
def search(query, topN):
  token = sc.parallelize(tokenize(query)).map(lambda x: (x,1) ).collectAsMap(
)
  brd_Tokens = sc.broadcast(token)

  #Create new rdd connected to tfidf_RDD with terms in the Query. to Limit th
e computation space
  joined_tfidf = tfidf_RDD.map(lambda (k,v): (k,brd_Tokens.value.get(k,'-'),v
) ).filter(lambda (a,b,c): b != '-' )

  #compute the score using aggregateByKey
  scount = joined_tfidf.map(lambda a: a[2]).aggregateByKey((0,0),
  (lambda acc, value: (acc[0] +value,acc[1]+1)),
  (lambda acc1,acc2: (acc1[0]+acc2[0],acc1[1]+acc2[1])) )

  scores = scount.map(lambda (k,v): ( v[0]*v[1]/len(token), k) ).top(topN)


  return scores
```

In [15]:

```
search("Manchester United striker", 1)
```

Out[15]:

```
[(13.183347464017316,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/001.txt')]
```

In [16]:

```
search("Manchester United striker", 3)
```

Out[16]:

```
[(13.183347464017316,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/001.txt'),
 (10.986122886681096,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/090.txt'),
 (10.253714694235692,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/007.txt')]
```

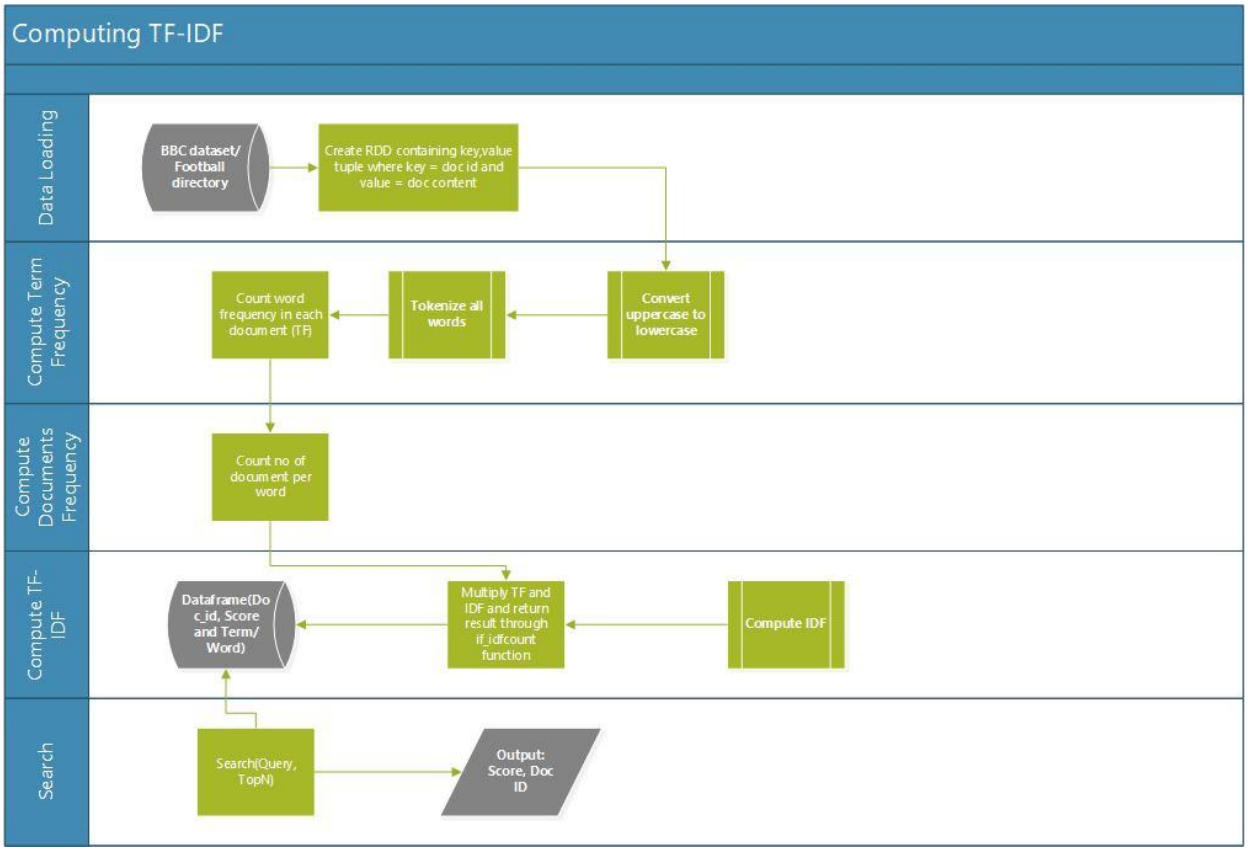
In [17]:

```
search("Manchester United striker", 5)
```

Out[17]:

```
[(13.183347464017316,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/001.txt'),
 (10.986122886681096,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/090.txt'),
 (10.253714694235692,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/007.txt'),
 (9.8875105980129891,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/262.txt'),
 (9.8875105980129874,
  u'file:/C:/Study/From HP/Mgt of big Data/Final Exam/Data Set/bbcsport/footb
all/141.txt')]
```

Summary:



Index Building:

Step1: Load the data (Ref: ln-2)

We have used BBC sports dataset to analyze TF-IDF scores, where we have selected football folder and its contents. To compute TF-IDF, I didn't use any existing library such as HashingTF() or IDF() instead used few lines of spark command to calculate TF-IDF.

Step2: Compute TF-IDF

- Find a document N, which replicates total no. of the document in the corpus.(Ref ln:3 & 4).
- Find TF(Term Frequency): Count the no of times the word occurs in the document. We have used regular expression function of python to split the text into word (Ref ln5) and then created an RDD called `tokenized_text2` tokenize the words. (Rer: ln 6). At last the RDD t_f stored the TF values (Ref: ln 7)
- Compute Document Frequency: Created document frequency RDD which contains the number of the document in the corpus includes the word. (Ref: ln 9 and 10)
- Compute TF-IDF: Here we will define a function which will take no of document N, TF and Document frequency to compute the TF-IDF score and build the index. (Ref: ln 11 and 12)

Query:

Score(Query, Doc) $=\frac{|q \cap Doc|}{|q|} \sum_{q \in Q} \text{TFIDF}(q, Doc)$ (Ref: ln 13 and 14)

Where:

$|q \cap Doc|$ = What terms the query and document have in common

$TFIDF(q, Doc)$ = TF-IDF scores of the terms in common

Characteristics of Search Function:

- Search (Query, TopN): TopN_documents
- Tokenize(Query)
- Perform a broadcast to filter out 'interesting' documents efficiently
- Used aggeregateByKey to compute similarity score.
- Return the TopN document.

Search timing:

The output is not real-time, I have tried to implement search function with spark SQL with Jupyter notebook but couldn't run because of the following error.

```
IllegalArgumentException: u"Error while instantiating 'org.apache.spark.sql.hive.HiveSessionStateBuilder':"
```