Question 2 Report 96101902

Part 2-Preprocessing

After downloading and extracting the dump file, we had to parse the data into *JSON* format so our RDD could be analyzed simpler. After that, we need to use *flatmap* and *split* function (just like the third part in Question1 for extracting the actors' names). Then, we compute the count of each word in the document and by utilizing *takeOrdered* function we order them in a descending order. These steps are conducted through the codes below:

```
import json

articles_rdd = articles_rdd.map(lambda text: json.loads(text))  #parse the json string
words_rdd = articles_rdd.flatMap(lambda doc: doc['title'].split(' ') + doc['text'].split(' ')) #ext
words_count_rdd = words_rdd.map(lambda word: (word,1)).reduceByKey(lambda word1,word2:word1+word2)
top_100 = words_count_rdd.takeOrdered(100,key=lambda x:-x[1]) #find the 100 most common words
```

top_100 contains the 100 most common words which need to be removed from the documents. Therefore, we need to define a particular function capable of:

1. Extracting the title and the body of a text
2. Finding the common words(stop words list and text file)
3. Removing them

This function is written in the code snippet below:

```
#we define a function to do: 1. detect the top_100 words
# 2. remove them

def remove_stop_words(text , uselesswords):
    #extract the title and body of a text
    text_body = text['text'].split(' ')
    text_title = text['title'].split(' ')

    #iterate on every word existed in text_body and text_title
    for word in text_body:
        if word in uselesswords:
            text_body.remove(word)
    for word in text_title:
        if word in uselesswords:
            text_body.remove(word)

    #join the words by ' '
    text['text']= ' '.join(text_body)
    text['title'] = ' '.join(text_title)
    return text
```

After applying this function to *articles*_rdd, we get a set of documents in which stop words are excluded. Then we have to use *filter* function to remove the uncommon words, with a frequency rate below 20, from the *articles_withoud_stopwords_rdd*.

By doing so, we get the output below:

```
MIN_COUNT = 20
uncommon_words = words_count_rdd.filter(lambda word: word[1] < MIN_COUNT) #list of the words that have occured less
uncommon_words_list = uncommon_words.map(lambda x: (x,1) ).collectAsMap() #list of words with frequency<20
articles_cleaned_rdd = articles_without_stopwords_rdd.map(lambda doc: remove_stop_words(doc, uncommon_words_list))#
articles_cleaned_rdd.take(1) # This should output a dictionary with url,title and text keys. title and text should
```

Output

```
Out[9]: [{'id': '2618101',
  'revid': '100607',
  'url': 'https://fa.wikipedia.org/wiki?curid=2618101',
  'title': 'امیلی جین وایت',
  'text': 'امیلی جین وایت زادهٔ یک موسیقی\u200cدان ایالات متحده آمریکا وی سال ۲۰۰۶ میلادی تاکنون مشغول فعالیت بوده\u200cاست.'}]
```

Part 3-Exploration

*** Note: I used sample() method in PySpark to make the process faster. Thus, the achieved output differs from the whole dataset's output. ***

I used the command below:

articels_rdd = articles_rdd.sample(False, 0.01, 81)

1. How many unique words remain after the cleaning procedure?

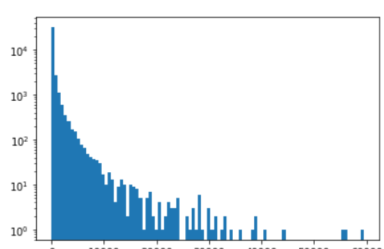
Code:

```
#How many unique words remain after the cleaning procedure?
#we have to use flatmap function to faltten the data
#and also distinct and count methods to compute to count of unique words
words_cleaned_rdd = articles_cleaned_rdd.flatMap(lambda doc: doc['title'].split()+doc['text'].split())
words_cleaned_rdd.distinct().count()
```

Output: 91915

2. Plot a distribution from document lengths using appropriate bin sizes with 100 bins

```
#Plot a distribution from document lengths using appropriate bin sizes with 100 bins
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
plt.figure(0)
plt.hist(articles_cleaned_rdd.map(lambda doc : len(doc['title'])+len(doc['text'])).collect(),bins=100)
plt.xlabel('Length')
plt.ylabel('Doc Counts')
plt.show()
```



3. **What is the *url* of the longest article?**

   We just have to extract the *url* of the RDD documents by using lambda function as well as the previous parts, then the longest *url* will be the first element of ordered url's in a descending order:

```
#What is the url of the longest article?
articles_cleaned_rdd.map(lambda document: (document['url'],len(document['title']) + len(document['text']))).takeOrdered(1, key=lambda x:-x[1])
```

   Output: https://fa.wikipedia.org/wiki?curid=10678

4. **How many articles contain your first name?!**

   We just have to search through doc['title'] and doc['text'] and see if there is any sign of my firstname:

```
#How many articles contain your first name?!
articles_cleaned_rdd.filter(lambda document: (if u'محمدرضا' in document['title'].split()) or (if u'محمدرضا' in document['text'].split())).count()
```

   Output: 108

Part 4-TF-IDF+Search

In the first part, we ought to compute the frequency of words in each document. We somehow did this before by using *flatMap* and *reduceByKey*. In the next part we have to implement a function capable of returning the td-idf metric for each word. Then we have to update each of the objects in RDD with a key value of td-idf vector as shown below:

```
#calculate document frequncy for each word
word_df_rdd = articles_cleaned_rdd.flatMap(lambda doc: [(word,1) for word in set(doc['title'].split()+doc['text'].split())]).reduceByKey(lambda x,y:x+y).collectAsMap()

def tf_idf(document):
    tf_idf={}
    for word in document['text'].split()+document['title'].split():
        tf_idf_result[word]=float(document['text'].split()+document['title'].split().count(word))/len(document['text'].split()+document['title'].split())*np.log(1+
(articles_cleaned_rdd.count()/(word_df_rdd[word]+1)))
    return tf_idf

def updateDoc(document,updatedObj):
    document.update(updatedObj)
    return document

#add `vector` key to articles_cleaned_rdd dictionary with the tf_idf dictionary
articles_tf_idf_vectors = articles_cleaned_rdd.map(lambda doc: updateDoc(doc,{'vector':tf_idf(doc)}))
articles_tf_idf_vectors.take(1)
```

```
Out[16]: [{'id': '5310821', 'url': 'https://fa.wikipedia.org/wiki?curid=5310821', 'title': 'آنا گودایل', 'text': 'آنا گودایل آنا گودایل زادهٔ ۱۸ مارس ۱۹۸۳) قایقران روئینگ ایالات متحده آمریکا وی مسابقات کشوری بین المللی مجموع برندهٔ ۱ مدال طلا\u200c', 'vector': {'آنا': 0.7912117875281975, 'گودایل': 1.2299386093321338, 'زادهٔ': 0.1104340468722631, '۱۸': 0.17367013403299186, 'مارس': 0.16506330652539775, '۱۹۸۳)': 0.3015618858699991, 'قایقران': 0.24708726395450434, 'روئینگ': 0.2669633123944671, 'ایالات': 0.09657457185284947, 'متحده': 0.09694310597367341, 'آمریکا': 0.09228458472153891, 'وی': 0.10241174486680323, 'مسابقات': 0.15503561240236344, 'کشوری':
```

vectors, then:

$$Cos(\theta) = \frac{d_2.q}{||d_2||\,||q||}$$

Therefore we have to calculate to cosine of two td-idf vectors to determine which documents resemble each other the most. The code and the output can be shown below:

```
from scipy.spatial.distance import cosine

query = 'هخامنشیان ساسانیان هگمتانه'

def get_tf_idf(query):
    tf_idf =[]
    BoW_len =
    for word in query.split():
        tf=query.split().count(word)
        df =word_df_rdd[word]+1
        tf_idf.append((float(tf)/len(query.split()))*np.log(1+(articles_cleaned_rdd.count()/(df+1))))
    return tf_idf
```

```
query_tf_idf = get_tf_idf(query)

def docScore(doc):
    common_doc =[]
    for word in query.split():
    if word in doc['vector']:
        common_doc.append(doc['vector'][word])
    else:
        common_doc.append(0)
    return np.nansum([0,cosine(query_tf_idf,common_doc)])
```

```
#find similar wikipedia articles based on cosine similarity of tf-idf vectors
top_10_similar = articles_tf_idf_vectors.map(lambda document: (document['url'],docScore(document))).filter(lambda x:x[1]>0).takeOrdered(10,key=lambda x:x[1])
top_10_similar
```

```
[('https://fa.wikipedia.org/wiki?curid=4727464',    0.1532075111168003),
('https://fa.wikipedia.org/wiki?curid=1598063',    0.2907297300625329),
('https://fa.wikipedia.org/wiki?curid=1492123',    0.290729730062533),
('https://fa.wikipedia.org/wiki?curid=385330',    0.290729730062533),
('https://fa.wikipedia.org/wiki?curid=6289',    0.29097595350852545),
('https://fa.wikipedia.org/wiki?curid=793139',    0.29506367532718913),
('https://fa.wikipedia.org/wiki?curid=4796138',    0.29506367532718925),
('https://fa.wikipedia.org/wiki?curid=240875',    0.29506367532718925),
('https://fa.wikipedia.org/wiki?curid=93185',    0.29506367532718925),
('https://fa.wikipedia.org/wiki?curid=656764',    0.29506367532718925)]
```