Example of lab2

Python version: 3.6

Step 1

```
# re is a library for regular expression operations
import re
import os
from pyspark import SparkConf, SparkContext
import math
import pandas as pd

# Filelist
path = "./datafiles"
file_names = os.listdir(path)
file_names.sort(key=lambda x: int(x[:-4]))
file_paths = []
for file in file_names:
    file_path = os.path.join(path, file)
    file_paths.append(file_path)

# Load stopwords
# This part is easy to make a mistake. For example, "you'll" may be split as "you" and
"ll" if you use regular expressions to get the stopwords.
# A good habit is to check the contents and the number of lines of the file and your
processed result.
# We didn't give a penalty here because some of you may be unfamiliar with the method
of splitting words.
stopwords = pd.read_fwf('stopwords.txt', header = None).values.reshape(1,-1)[0]

# Step 1
# Read texts
conf = SparkConf()
sc = SparkContext(conf=conf)
line = sc.textFile(file_paths[0])

# Split and lower words, remove stopwords and NA
# We didn't give any penalty here.
word = line.flatMap(lambda l: [j for j in [item.lower() for item in [i for i in
re.split(r'[^\w]+', l) \
                                                                          if i]] if j not in
stopwords])
pairs = word.map(lambda w: ((0, w), 1))
```

```
# Compact into one rdd file
for file in file_paths[1:]:
    lines = sc.textFile(file)
    words = lines.flatMap(lambda l: [j for j in [item.lower() for item in [i for i in
re.split(r'[^\w]+', l) \
                                                                              if i]] if j
not in stopwords])
    pair = words.map(lambda w: ((file_paths.index(file), w), 1))
    pairs = pairs.union(pair)

TF = pairs.reduceByKey(lambda n1, n2: n1 + n2)
```

Step 2

```
# Step2
# Number of docs
N = len(file_names)

# Calculate idf
pairs = TF.map(lambda x: (x[0][1], 1))
DF = pairs.reduceByKey(lambda n1, n2: n1 + n2)
IDF = DF.map(lambda x: (x[0], math.log10(N / x[1])))

# Join tf and idf
TF = TF.map(lambda x: (x[0][1], (x[0][0], x[1])))
TI = TF.leftOuterJoin(IDF, numPartitions=None)

# Calculate df-idf
tfidf = TI.map(lambda x: ((x[1][0][0], x[0]), ((1 + math.log10(x[1][0][1])) * x[1]
[1])))
```

Step 3

```
# Step3
# Calculate sum of squares
s = tfidf.map(lambda x: (x[0][0], x[1] ** 2))
s = s.reduceByKey(lambda n1, n2: n1 + n2)

# Join tfidf and sum of squares
tfidf = tfidf.map(lambda x: (x[0][0], (x[0][1], x[1])))
ts = tfidf.leftOuterJoin(s, numPartitions=None)

# Calculate normalized tf-idf
n_tfidf = ts.map(lambda x: ((x[0], x[1][0][0]), x[1][0][1] / math.sqrt(x[1][1])))
```

Step 4

```
# Step4

# Read query
query = sc.textFile("query.txt")
query = query.flatMap(lambda l: re.split(r'[^\w]+', l)).collect()

# Norm of query vector sqrt(3),norm of docs vectors: 1
v = math.sqrt(len(query))

# Filter words in query
doc = n_tfidf.filter(lambda x: x[0][1] in query)
doc = doc.map(lambda x: (x[0][0], x[1]))

# Those do not contain words in query have 0 value
keys = doc.keys().collect()
doc_n = n_tfidf.filter(lambda x: x[0][0] not in keys)
doc_n = doc_n.map(lambda x : (x[0][0], 0))
doc = doc.union(doc_n)

# Compute relevance, relevance of that not in r is 0
r = doc.reduceByKey(lambda n1, n2: n1 + n2)
r = r.map(lambda x: (x[0], x[1] / v))
```

Step 5

```
# Step5
r_sort = r.sortBy(lambda x: x[1], ascending=False)
top10 = r_sort.take(10)
with open(r'step5.txt','w') as f:
    print(top10,file=f)
```

Step 6

```
# Step6
# Here, we split the sentences by • and .
# You can also add other ctuations such as ? and !. We didn't give any penalty here.
pattern = r'[•.]+'
top = []
for d in top10:
    # Read texts
    lines = sc.textFile(file_paths[d[0]])
    # Split sentences
    sentences = lines.flatMap(lambda l: [i for i in re.split(pattern, l) if i])
    # Split words
    wordlists = sentences.map(lambda l: (l, [j for j in [item.lower() for item in [i
for i in   re.split(r'[^\w]+', l) if i]] if j not in stopwords]))
    words = wordlists.flatMapValues(lambda w: [i for i in w])
    # Compute tf
    pairs = words.map(lambda w: ((w[0], w[1]), 1))
    TF = pairs.reduceByKey(lambda n1, n2: n1 + n2)
    # Number of sentences
    N = len(sentences.collect())
    # Compute idf
    p = TF.map(lambda x: (x[0][1], 1))
    DF = p.reduceByKey(lambda n1, n2: n1 + n2)
    IDF = DF.map(lambda x: (x[0], math.log10(N / x[1])))
    # Join tf and idf
    TF2 = TF.map(lambda x: (x[0][1], (x[0][0], x[1])))
    TI = TF2.leftOuterJoin(IDF, numPartitions=None)
    # Compute tfidf
    tfidf = TI.map(lambda x: ((x[1][0][0], x[0]), (1 + math.log10(x[1][0][1])) * x[1]
[1]))
    # Compute sum of squares
    s = tfidf.map(lambda x: (x[0][0], x[1] ** 2))
    s = s.reduceByKey(lambda n1, n2: n1 + n2)
    # Join tfidf and sum of squares
    tfidf2 = tfidf.map(lambda x: (x[0][0], (x[0][1], x[1])))
    ts = tfidf2.leftOuterJoin(s, numPartitions=None)
    # Compute normalized tfidf
    n_tfidf = ts.map(lambda x: ((x[0], x[1][0][0]), x[1][0][1] / math.sqrt(x[1][1])))

    # Filter query
    doc = n_tfidf.filter(lambda x: x[0][1] in query)
    doc = doc.map(lambda x: (x[0][0], x[1]))

    # Those do not contain words in query have 0 value
    keys = doc.keys().collect()
    doc_n = n_tfidf.filter(lambda x: x[0][0] not in keys)
    doc_n = doc_n.map(lambda x: (x[0][0], 0))
    doc = doc.union(doc_n)

    # Compute relevance
    r = doc.reduceByKey(lambda n1, n2: n1 + n2)
    r = r.map(lambda x: ((d[0], x[0]), x[1] / v))
    r_sort = r.sortBy(lambda x: x[1], ascending=False)
    r.coalesce(1).saveAsTextFile(f"step6/{d[0]}")
    top.append(r_sort.take(1))
```

Step 7

```python
# Step 7
with open(r'output.txt', 'w') as f:
    for i in range(10):
        print("<", top10[i][0], ">", "<", top10[i][1], ">", "<", top[i][0][0][1], ">",
"<", top[i][0][1], ">", "\n",file=f)
        print("<", top10[i][0], ">", "<", top10[i][1], ">", "<", top[i][0][0][1], ">",
"<", top[i][0][1], ">", "\n")
```

Result

```
< 18 > < 0.07397790324517098 > < appropriate mask management should be followed, which
includes how to use and dispose of masks > < 0.24877888957250752 >
< 142 > < 0.07347493232700521 > < mask, which type of mask should be used and when it
should be worn > < 0.2591142588078523 >
< 19 > < 0.049671258014073814 > < Identify, isolate and care for patients early,
including providing optimized care for infected patients;  > < 0.2331450273678335 >
< 52 > < 0.048304963506359835 > < how to use a mask), are being developed on demand,
> < 0.3948352741257387 >
< 59 > < 0.04487806643005116 > <   The first vaccine trial has begun just 60 days after
the genetic sequence of the  > < 0.23454739408797912 >
< 140 > < 0.044353287644493616 > < cover your cough and wear a mask if you attend a
protest > < 0.24086595720570408 >
< 83 > < 0.04286891247606776 > <   There is no evidence that the Bacille Calmette-
Guérin vaccine (BCG) protects  > < 0.22806391170709633 >
< 132 > < 0.04112408805099718 > <   This  working  group  provides  guidance  to
vaccine  > < 0.22363054683741482 >
< 146 > < 0.039427692231289384 > < research on a COVID-19 Vaccine > <
0.4290817597871959 >
< 96 > < 0.037892748873481534 > < and vaccine-preventable disease outbreaks during the
COVID-19 pandemic > < 0.2902321295024496 >
```

Note: There are many methods of preprocessing. For example, we could utilize library **nltk** to deal with the documents and sentences and we could split the sentences by different punctuations. Therefore, the result is not unique. We didn't give any penalty if your method is different but reasonable.

Problem

The following are the problems in your submissions.

Output format

- *The output should be sorted in descending order of the relevance of the documents to the query.*
- *Output the most relevant sentence for each of the top-10 documents.*

Some students forgot or misunderstood the requirements. Please read the requirements carefully before you start and check the results after you finish the task.

Documentation for the code

- *Zip your executable Spark program(s) with documentation in the code, the output files, and upload it to the Lab2 folder in Luminus.*

We gave a penalty to the students who didn't write the documentation for the code or whose documentation was bad. You should write detailed documentation for each step.

Out of memory Error or excessive runtime

It's hard for us to check the program if it needs lots of memory or the logic is bad which results in a long time to run. Please try to optimize your program and reduce the memory used by the program.

Careless mistakes

Some students are so careless that they utilize library `nltk` to extract the word stem but forget to do the same process to the query. Therefore, in the query, it is "isolate", "vaccine" but in the dataset, it is "isol", "vaccin". It results in the sentences containing only "mask". Please keep the preprocessing same between the dataset and the query.

Obvious errors in the results

Some students' results are obviously incorrect. For example, the relevance scores of about 8 documents are the same and some sentences in the result have nothing to do with the query. Please check the result before submission.