EECE 5698 Assignment 1

Zifeng Wang

Question 0:

(1) This prints out a help documentation of the TextAnalyzer.py file which specifies the arguments and their meanings.

(2) Lines pasted below causes this to be printed.

```
    parser = argparse.ArgumentParser(description = 'Text Analysis
through TFIDF
computation',formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('mode', help='Mode of
operation',choices=['TF','IDF','TFIDF','SIM','TOP'])
    parser.add_argument('input', help='Input file or list of files.')
    parser.add_argument('output', help='File in which output is
stored')
    parser.add_argument('--master',default="local[20]",help="Spark
Master")
    parser.add_argument('--idfvalues',type=str,default="idf",
help='File/directory containing IDF values. Used in TFIDF mode to
compute TFIDF')
    parser.add_argument('--other',type=str,help = 'Score to which
input score is to be compared. Used in SIM mode')
    args = parser.parse_args()
```

(3) The module is argparse. The documentation is as below.

https://docs.python.org/3/library/argparse.html

(4) Using that documentation, we find that first line creates an ArgumentParser object which holds all the information necessary to parse the command line into Python data types. The consecutive lines tell the ArgumentParser how to take the strings on the command line and turn them into objects, or in other words, add command line parameters to the program. Finally, the last line parses arguments through the parse_args() method.

Question 1:

1(a) Code snippet is below: (comments are explanations)

```
input_rdd = sc.textFile(args.input)  # Read txt file as rdd
input_rdd.flatMap(lambda x:x.split()) \ # split lines as words
       .map(lambda x:stripNonAlpha(toLowerCase(x))) \ # remove non-
```

```
alphabetic characters and make the remaining letters lowercase
        .filter(lambda x:x != '') \ # remove empty strings
        .map(lambda x:(x, 1)) \  # map each word to value 1
        .reduceByKey(lambda x, y: x+y) \ #reduce to calculate frequency
of each words
        .saveAsTextFile(args.output) # save the final results
```

1(b) Here are 3 files _SUCCESS, part-00000, part-00001
_SUCCESS shows the program runs successfully.
part-0000* are parts of the final output result.
First 5 lines of part-00000:

```
(u'all', 48)
(u'savior', 1)
(u'dance', 4)
(u'mattered', 1)
(u'ephemeral', 1)
```

Question 2:

2(a) Code snippet is below: (comments are explanations)

```
input_rdd = sc.textFile(args.input) # read input file as rdd
top20_tuples = input_rdd.map(eval) \ # map each string to tuple
                        .sortBy(lambda (x,y):y, ascending=False) \
                        # sort the tuples by key as descending order
                        .take(20) # take the first 20 tuples with
                      highest frequency
with open(args.output, 'w') as f: # save the results in a txt file
    for top20_tuple in top20_tuples:
        f.write(str(top20_tuple))
        f.write('\n')
```

2(b) Here are the 20 most frequent words in `hotel-california.txt` and their TF

```
(u'the', 294)
(u'i', 192)
(u'to', 145)
(u'of', 142)
(u'and', 137)
(u'a', 136)
(u'was', 102)
(u'in', 76)
(u'it', 68)
(u'he', 58)
(u'my', 57)
```

```
(u'that', 53)
(u'on', 51)
(u'we', 50)
(u'all', 48)
(u'you', 48)
(u'had', 47)
(u'me', 44)
(u'said', 42)
(u'were', 41)
```

Question 3:

20 most frequent words in the entire corpus:

```
(u'the', 26064)
(u'to', 13468)
(u'and', 12376)
(u'of', 11981)
(u'a', 10358)
(u'in', 8300)
(u'that', 6466)
(u'i', 6435)
(u'is', 5318)
(u'you', 4720)
(u'for', 4228)
(u'it', 4014)
(u'on', 3373)
(u'with', 3136)
(u'was', 2995)
(u'as', 2860)
(u'this', 2760)
(u'be', 2520)
(u'we', 2447)
(u'have', 2319)
```

Question 4:

4(a)Code snippet is below (no comments or explicit explanations as the instruction does not mention that)

```
allFiles = sc.wholeTextFiles(args.input)
num_of_files = allFiles.count()
allFiles.flatMapValues(lambda x:x.split()) \
      .mapValues(lambda x:stripNonAlpha(toLowerCase(x))) \
      .filter(lambda (key,val):val!='') \
      .distinct() \
```

```
    .map(lambda (key, val): (val, 1)) \
    .reduceByKey(lambda x, y: x+y) \
    .mapValues(lambda x: math.log(float(num_of_files) / x)) \
    .saveAsTextFile(args.output)
```

4(b) The first 5 lines of file part-00000

```
(u'unimaginative', 5.966146739123692)
(u'moskowitz', 5.966146739123692)
(u'revetts', 5.966146739123692)
(u'hallwood', 5.966146739123692)
(u'skylit', 5.966146739123692)
```

Question 5:

5(a) Code snippet is below (no comments or explicit explanations as the instruction does not mention that)

```
tf_scores = sc.textFile(args.input).map(eval)
idf_scores = sc.textFile(args.idfvalues).map(eval)
tf_scores.join(idf_scores) \
         .mapValues(lambda (v1, v2):v1 * v2) \
         .sortBy(lambda (x,y):y, ascending=False) \
         .saveAsTextFile(args.output)
```

5(b) Here are the words with top 20 TFIDF scores

```
(u'adrienne', 202.84898913020552)
(u'ship', 120.60550917719912)
(u'zheng', 110.73299072983869)
(u'i', 96.44446734683174)
(u'ray', 87.13417653379183)
(u'sarah', 83.48774539791273)
(u'kishori', 77.559907608608)
(u'was', 65.59994951849896)
(u'tiffany', 63.27599470276496)
(u'she', 62.86790337805013)
(u'my', 59.631451357847325)
(u'he', 59.42125035783449)
(u'captain', 54.26703450864328)
(u'her', 50.275350926901396)
(u'jefferson', 50.092647238747645)
(u'glass', 48.87144807032223)
(u'said', 47.788986076498425)
(u'had', 44.59674440851208)
(u'looked', 42.88730041201648)
```

```
(u'me', 40.599751376995606)
```
This words are rarer compared to the terms in Question 2(b), which are frequently appeared words in almost every English written passages. So I think TFIDF is more representative of a document because TF is a representation of words inside a particular document, but just TF is not good because common words will always prevail. However, IDF is the nature of every words among all different documents, the rarer a word, the higher IDF it will get. So TF * IDF is a good representation of a document among all documents because it kind of combined local and global feature, so that some rare words (representative words of this particular document) will stand out and represent this document.

Question 6:

6(a) Code snippet is below (no comments or explicit explanations as the instruction does not mention that)

```
tfidf_f1 = sc.textFile(args.input).map(eval)
tfidf_f2 = sc.textFile(args.other).map(eval)
numerator = tfidf_f1.join(tfidf_f2) \
               .mapValues(lambda (v1, v2):v1 * v2) \
               .map((lambda (key, value): value)) \
               .reduce(lambda x, y: x+y)
denom1 = tfidf_f1.map(lambda (key,val): val*val) \
              .reduce(lambda x, y: x+y)
denom2 = tfidf_f2.map(lambda (key,val): val*val) \
              .reduce(lambda x, y: x+y)
denominator = math.sqrt(denom1*denom2)
sim = numerator/denominator
with open(args.output, 'w') as f:
    f.write(str(sim))
```

6(b) The result table is below

	face-to-face	fiction	spam
face-to-face	1	0.2857397576137937	0.218984526265908
fiction	0.2857397576137937	1	0.31399650707568616
fpam	0.218984526265908	0.31399650707568616	1