Question 0:

(a)

The argparse module generates help and usage messages, indicates input/output files as well as figure out how to parse those out of sys.argv.

(b)

```
[zhang.xianl@discovery2 Assignment1]$ python TextAnalyzer.py --help
usage: TextAnalyzer.py [-h] [--master MASTER] [--idfvalues IDFVALUES]
                       [--other OTHER]
                       {TF,IDF,TFIDF,SIM,TOP} input output

Text Analysis through TFIDF computation

positional arguments:
  {TF,IDF,TFIDF,SIM,TOP}
                        Mode of operation
  input                 Input file or list of files.
  output                File in which output is stored

optional arguments:
  -h, --help            show this help message and exit
  --master MASTER       Spark Master (default: local[20])
  --idfvalues IDFVALUES
                        File/directory containing IDF values. Used in TFIDF
                        mode to compute TFIDF (default: idf)
  --other OTHER         Score to which input score is to be compared. Used in
                        SIM mode (default: None)
```

Figure 1

Figure 1 shows what the "python TextAnalyzer.py --help" print, the following lines cause this 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 containi
ng IDF values. Used in TFIDF mode to compute TFIDF')
    parser.add_argument('--other',type=str,help = 'Score to which input score is to be comp
ared. Used in SIM mode')
    args = parser.parse_args()
```

Question 1:

(a)

The modified code is as following:

```
    if args.mode=='TF':
        # Read text file at args.input, compute TF of each term,
        # and store result in file args.output. All terms are first converted to
        # lowercase, and have non alphabetic characters removed
        # (i.e., 'Ba,Na:Na.123' and 'banana' count as the same term). Empty strings, i.
e., ""
        # are also removed
        myrdd = sc.textFile(args.input)
        myrdd.flatMap(lambda s : s.split())\
        .map(lambda word: (toLowerCase(stripNonAlpha(word)), 1))\
        .reduceByKey(lambda x, y : x + y)\
        .filter(lambda (x, y) : x != "").saveAsTextFile(args.output)
```

First we use sc.textFile() to make the article as rdd, and then the flatMap() makes the article as a tuple of all words. Map() is for giving counter to each word, in which we also call toLowerCase() and stripNonAlpha() to standardize words as instruction said. By using reduceBykey(), we reduce all same words' counters into one structure. And at last, we use filter() to remove the key empty string ("") and save results in output file.

(b)

This directory contains three files, part-00000, part-00001 and _SUCCESS. The first 5 lines of file part-00000 is as following:

```
[zhang.xianl@discovery2 hotel-california.tf]$ head -n 5 part-00000
(u'all', 48)
(u'savior', 1)
(u'dance', 4)
(u'mattered', 1)
(u'ephemeral', 1)
```

Question 2:

(a)

The following is the code I wrote:

```
    if args.mode=='TOP':
        # Read file at args.input, comprizing strings representing pairs of the
form (TERM,VAL),
        # where TERM is a string and VAL is a numeric value. Find the pairs with
 the top 20 values,
        # and store result in args.output
        myrdd = sc.textFile(args.input)
        top_20 = myrdd.map(lambda s : eval(s))\
        .takeOrdered(20, key = lambda (x, y) : -y)
        sc.parallelize(top_20).saveAsTextFile(args.output)
```

myrdd creates rrd from args.input, and top_20 makes data from string to tuple through eval() and sorts data by takeOrdered(). At last use sc.parallelize() to make it as rdd, and save it in args.out file.

(b)

The most 20 frequent words are as following:

```
[zhang.xianl@discovery2 q2.tf]$ cat *
(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:

```
[zhang.xianl@discovery2 q3.tf]$ cat *
(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:

(a)

The code I modified is as following:

```
    if args.mode=='IDF':
        # Read list of files from args.input, compute IDF of each term,
        # and store result in file args.output.  All terms are first converted t
o
        # lowercase, and have non alphabetic characters removed
        # (i.e., 'Ba,Na:Na.123' and 'banana' count as the same term). Empty stri
ngs ""
        # are removed
        myrdd = sc.wholeTextFiles(args.input)
        doc_num = myrdd.count()
        myrdd.flatMapValues(lambda s : s.split())\
        .map(lambda (x, y) : (x, toLowerCase(stripNonAlpha(y))))\
        .distinct().map(lambda (x, y) : (y, 1))\
        .reduceByKey(lambda x, y : x + y)\
        .filter(lambda (x, y) : x != "")\
        .map(lambda (x, y) : (x, math.log(1.0 * doc_num / y)))\
        .saveAsTextFile(args.output)
```

(b)

```
[zhang.xianl@discovery2 anc.idf]$ head -n 5 part-00000
(u'aided', 4.867534450455582)
(u'unscientific', 5.966146739123692)
(u'revetts', 5.966146739123692)
(u'systematic', 5.272999558563747)
(u'skylit', 5.966146739123692)
```

Question 5:

(a)

The modified code is as following:

```
    if args.mode=='TFIDF':
        # Read  TF scores from file args.input the IDF scores from file args.idf
values,
        # compute TFIDF score, and store it in file args.output. Both input file
s contain
        # strings representing pairs of the form (TERM,VAL),
        # where TERM is a lowercase letter-only string and VAL is a numeric valu
e.
        TF = sc.textFile(args.input).map(lambda s : eval(s))
        IDF = sc.textFile(args.idfvalues).map(lambda s: eval(s))
        TFIDF = TF.join(IDF)
        score = TFIDF.map(lambda (x, y) : (x, y[0] * y[1]))
        #score.saveAsTextFile(args.output)
        score_20 = score.takeOrdered(20, key = lambda (x, y) : -y)
        sc.parallelize(score_20).saveAsTextFile(args.output)
```

(b)

The top 20 TFIDF files are as following, they are different from Q2(b). Apparently, TFIDF is more representative than TF, since it considers the correlation between documents and terms. TF only takes the term frequency into account, but some words may be just frequent in one or two articles, which may affect the results we want.

```
[zhang.xianl@discovery2 q5.tf]$ cat *
(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)
```

Question 6:

(a)

First, use command “find masc_500k_texts -name filename -type d” to find path of these directory, and calculate parameters we need. The modified code is as following:

```
    if args.mode=='SIM':
        # Read  scores from file args.input the scores from file args.other,
        # compute the cosine similarity between them, and store it in file args.
output. Both input files contain
        # strings representing pairs of the form (TERM,VAL),
        # where TERM is a lowercase, letter-only string and VAL is a numeric val
ue.
        F_1 = sc.textFile(args.input).map(lambda s : eval(s))
        F_2 = sc.textFile(args.other).map(lambda s : eval(s))
        intersect = F_1.join(F_2)
        sum_1 = intersect.map(lambda (x, y) : y[0] * y[1])\
        .reduce(lambda x, y : x + y)
        sum_2 = F_1.map(lambda (x, y) : y ** 2)\
        .reduce(lambda x, y : x + y)
        sum_3 = F_2.map(lambda (x, y) : y ** 2)\
        .reduce(lambda x, y : x + y)
        # Make it as list, or error
        cosine = [(1.0 * sum_1) / math.sqrt(1.0 * sum_2 * sum_3)]
        sc.parallelize(cosine, 1).saveAsTextFile(args.output)
```

(b)

	face-to-face	fiction	spam
face-to-face	1	0.285739757614	0.218984526266
fiction	0.285739757614	1	0.313996507076
spam	0.218984526266	0.313996507076	1