Q0:

```
[wu.tiany@compute-0-077 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)
[wu.tiany@compute-0-077 Assignment1]$
```

The above figure shows that what does the “python TextAnalyzer.py –help” print. It prints the usage of the python file and indicate the input and output.

This is printed because in the function it used argparse tool to set up the help function.

Q1.

a) The code I wrote is

```
rdd = sc.textFile(args.input)
rdd.flatMap(lambda line: line.split())\
   .map(lambda word : (toLowerCase(stripNonAlpha(word)), 1))\
   .reduceByKey(lambda x,y: x+y)\
   .filter(lambda (x,y): x != "")\
   .saveAsTextFile(args.output)
```

First, I put the input in to rdd. Then I use flatmap to spilt the article in to words then I use map to get rid of all the non-alphabetic characters and but all the words in lower key and make the word and 1 to be a tuple as (word, 1). Then I use reduceByKey to count the word appear times. Then I use filter to get rid of "". Finally I use saveAsTextFile to save the output.

b)

The contents of direction are 3 files part-00000, part-00001 and _success. Part-00000 and part-00001 contains the (key, value) pair show that how many times the key shows in the input file.

```
(u'wrong', 1)
(u'lighting', 2)
(u'seemingly', 1)
(u'curiosity', 1)
(u'water', 2)
```

The above figure shows the first 5 lines of file part-00000.

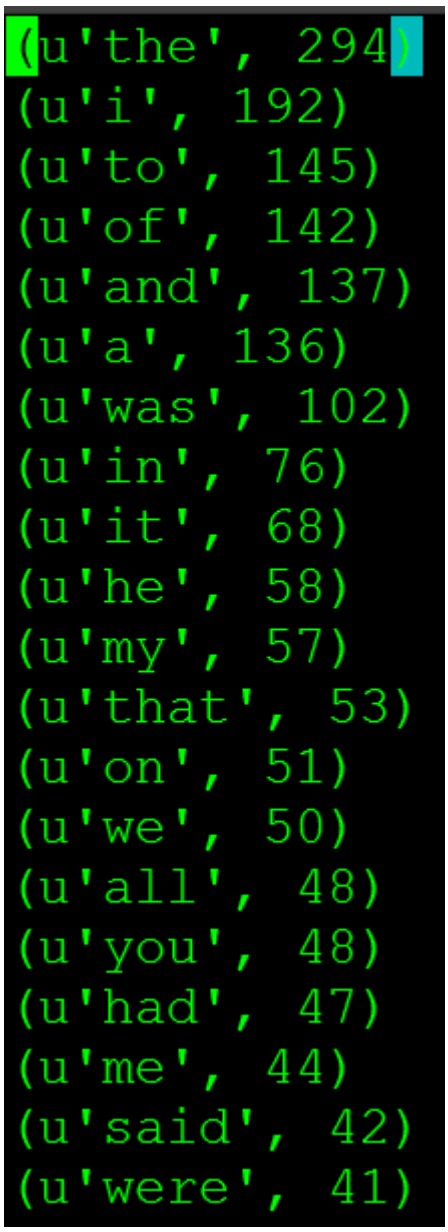
Q2

a)

```
rdd = sc.textFile(args.input)
_20 = rdd.map(lambda line: eval(line))\
    .takeOrdered(20, key=lambda (x,y): -y)
sc.parallelize(_20, 1)\
   .saveAsTextFile(args.output)
```

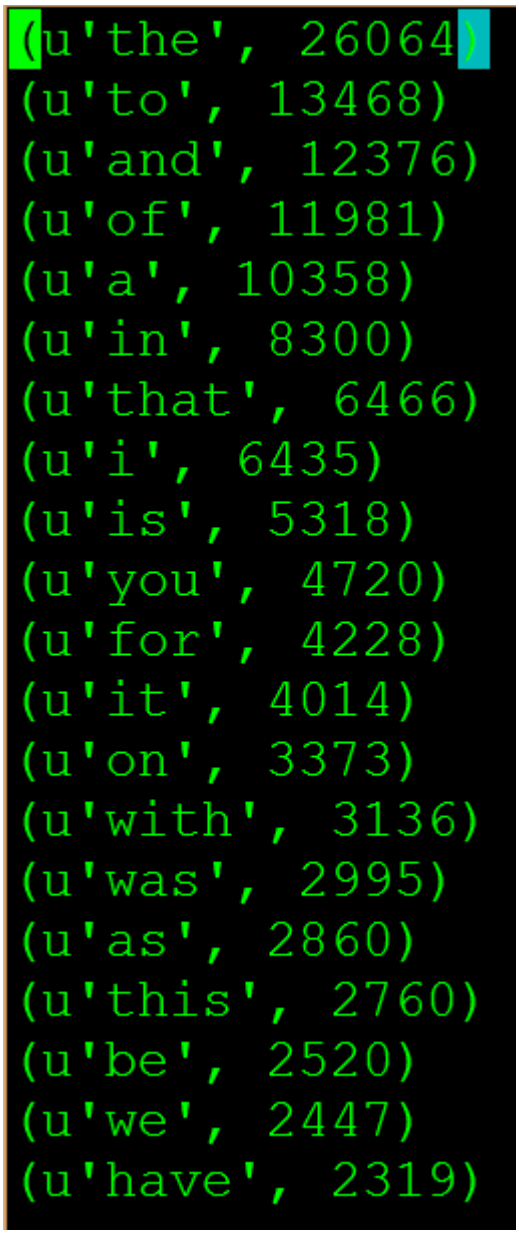
Above is the code I write. First, I put the input into rdd. Then I use eval to covert the string in to key value pairs. Then I use takeOrdered to put the key value pair in to descending order according to its value and take the top 20 pairs. Then I use parallelize to combine the 20 pair lists and then use saveAsTextFile to save the output

b)



Above is the highest 20 words and their frequency.

Q3



Above is the top 20 words with highest TF in all the files.

Q4

a)

```
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 to
    # lowercase, and have non alphabetic characters removed
    # (i.e., 'Ba,Na:Na.123' and 'banana' count as the same term). Empty strings ""
    # are removed
    rdd = sc.wholeTextFiles(args.input)
    count = rdd.count()
    wordnumber = rdd.flatMapValues(lambda x : x.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*count/y)))\
                    .saveAsTextFile(args.output)
```

Above is my code for idf. It takes the file as input and spilt it in to (path, article) pair and then split it in to (path, (word),(word)…) pair then use distinct to get all the unique words in a article and map them to (word,1) pair then use reduce by key to get the frequency of the same word in different article and use filter to get rid of ‘ ‘ the calculate the idf value.

b)

```
(u'aided', 4.867534450455582)
(u'unscientific', 5.966146739123692)
(u'revetts', 5.966146739123692)
(u'systematic', 5.272999558563747)
(u'skylit', 5.966146739123692)
```

Above is the first five line of part-00000 of the anc.idf file.

Q5 a)

```
if args.mode=='TFIDF':
    # Read  TF scores from file args.input the IDF scores from file args.idfvalues,
    # compute TFIDF score, 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 value.
    TF = sc.textFile(args.input)\
           .map(lambda line: eval(line))
    IDF = sc.textFile(args.idfvalues)\
            .map(lambda line: eval(line))
    TFIDF = TF.join(IDF)
    Score = TFIDF.map(lambda (x,y): (x,1.0 * y[0] * y[1]))\
                 .saveAsTextFile(args.output)
```

Above is my code for TFIDF. It takes tf file as input and idf file as idfvalues and then use join to get the words that both exist in the tf and idf file then times their score together to get the tfidf score.

b)

```
(u'adrienne', 202.848989130205 2)
(u'ship', 120.358403702194)
(u'zheng', 110.73299072983869)
(u'ray', 87.13417653379183)
(u'sarah', 83.48774539791273)
(u'kishori', 77.559907608608)
(u'tiffany', 63.27599470276496)
(u'she', 55.451774444795625)
(u'captain', 54.26703450864328)
(u'jefferson', 50.092647238747645)
(u'glass', 48.87144807032223)
(u'said', 46.14171612406061)
(u'her', 45.747713916956386)
(u'looked', 41.446531673892 83)
(u'he', 40.20253647247683)
(u'my', 39.50938929191688)
(u'sarahs', 35.79688043474215)
(u'maybe', 32.95836866004329)
(u'had', 32.57791748631743)
(u'didnt', 32.18875824868201)
~
```

The highest top 20 tfidf is showed above. The result is different from the tf result. I think tfidf is more representative because the tf only shows the most frequent word in this article but what if this word is highly frequent in all the other article. Then we have tfidf it uses tf times idf which is a weight coefficient which give the words who appears in many documents a low score and words that are unique in an article a high score. So that tfidf is more representative. It can show that the real unique high frequency words in the article.

Q6 a)

```
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 value.
    TFIDF1 = sc.textFile(args.input)\
               .map(lambda line: eval(line))
    SUM1 = TFIDF1.map(lambda x : x[1]**2)\
                 .reduce(lambda x,y: x + y)
    TFIDF2 = sc.textFile(args.other)\
               .map(lambda line :eval(line))
    SUM2 = TFIDF2.map(lambda x : x[1]**2)\
                 .reduce(lambda x,y: x + y)
    COMB = TFIDF1.join(TFIDF2)
    SUM3 = COMB.map(lambda (x,y):1.0 * y[0] * y[1])\
               .reduce(lambda x,y: x + y )
    COS = [(1.0 * SUM3)/math.sqrt(1.0 * SUM1 * SUM2)]
    sc.parallelize(COS, 1)\
      .saveAsTextFile(args.output)
```

Above is my code for SIM. It takes tfidf file as input and another tfidf file as other then compute the cos similarity between 2 tfidf file

b)

	Face-to-face	fiction	sapam
Face-to-face	1	0.285739757614	0.218984526266
fiction	0.285739757614	1	0.313996507076
sapam	0.218984526266	0.313996507076	1