Assignment 1: Modified WordCount/Pairs counting
G01276965

Introduction:

For this Homework, we are using are the text transcriptions of the State of the Union Addresses given by Presidents to Congress since 1790 to the present year as the dataset. There are two parts to this Assignment. In the first part of the assignment, I have done wordcount of the data , cleaning of the data and computing the average and standard deviation of the words. In the second part of the assignment, I have converted the data into pair of words and calculated the frequency of co-occurrence of words and I have also calculated the lift of two words bigger than 3.0 .

Pre-Processing Data:

In the index page of the State of the Union Addresses, there are many individual speeches delivered by the president each year. First, I used the urlopen() method from the urllib.request.Request module to read the contents of the website and save them . With the contents, I further took out the links present in the index page using the h_ref . The links are saved in a file alllink.txt . After that I opened each link, I wrote their data onto separate text files such as “contentfile-17900108.html.txt” for the year 1790 for the sub-link http://stateoftheunion.onetwothree.net/texts/19620111.html. Like this, data of all years are separately stored in text files and then accessed using a “allnames” list which has the names of all stored txt files. Using this, I access all the data present in the speeches. Below you can see the code to load the data.

```
#loading the website and its contents
url = "http://stateoftheunion.onetwothree.net/texts/index.html"
requ = urllib.request.Request(url,headers=headers)
req_url = urllib.request.urlopen(requ)
contents = req_url.read()
contents = str(contents)

#writing the contents into alldata file.
f = open('alldata.txt', 'w')
f.writelines(contents)
f.close()
f = open('alldata.txt','r')
contents = f.read()
h_r = r'href=[\'"]?([^\'" >]+)'
pagenames = re.findall(h_r, contents)
allnames = list()

#creating and saving the contents of each link into thier specific txt file such as contentfile-17901208.html.txt
for index,pagename in enumerate(pagenames):
    if pagename[0:2] == '17' or pagename[0:2] == '18' or pagename[0:2] == '19' or pagename[0:2] == '20':
        allnames.append(pagename)
        file = open('contentfile-%s.txt' % pagename, 'w')
        url = "http://stateoftheunion.onetwothree.net/texts/"+pagename
        req = urllib.request.Request(url,headers=headers)
        request_url = urllib.request.urlopen(req)
        contents = request_url.read()
        file.write(str(contents))
```

After loading the data, the data had to be cleaned. I separated each word by space and then I removed punctuations , HTML commands , Tags , \t and \n, stop words and digits .I even removed empty strings and again separated the data by space. To remove the punctions and other extra characters , I created functions for each and used ".map" to call the functions and remove them. Below, In the image you can see how I cleaned the data.

```
def RemovePunc(a):
    punc='!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~-'
    lowcase = a.lower()
    for ch in punc:
        lowcase = lowcase.replace(ch, " ")
    return lowcase
```

```
my_rdd = my_rdd.map(RemovePunc)
my_rdd = my_rdd.map(RemoveHtml)
```

Calculating the average and standard deviation for four year window:

Once the pre-processing of the data is done then we take the rdd and count each word . For counting of same key word, I used "reduceByKey(lambda x,y:(x+y)).sortByKey()" which calculates the number of times a word occurs and stores them as word and count value in rdd. The "sortByKey" stores the keys in alphabetic order for convinience. At the end of each page, the word count is stored in the list and after 4 times we do sc.union to accumulate all the words in a 4-year interval. "reduceByKey" is again used to reduce repeated words. For calculating the average and standard deviation , I have used ".mapValues(lambda x" to calculate both. The image below shows the calculation of average and standard deviation.

```
rdd2 = rdd.reduceByKey(lambda x,y:(x+y)).sortByKey()
#for average we have divided by 4 as there are 4 year groups
rdd2 = rdd2.mapValues(lambda x: x / 4)
print("The average of each word is as follows")
print(rdd2.collect())


y = rdd.groupByKey()
y = y.sortByKey()
# this creates a calculation of standard deviation with result as false for those whose len is less than 1
y1 = y.mapValues(lambda x: len(x)>1 and stdev(x))
print("The standard deviation of each word is as follows")
print(y1.collect())
```

For average, the number has simply been divided by 4 as the data is of 4 years. While for standard deviation, I have used stdev() function to calculate the standard deviation. The data input is in different format. As stdev() function uses list of data to calculate the standard deviation, so I have used groupByKey() to make the values of word in list format. So, if the list has more than one number then the standard deviation will be calculated. The image below shows the average of each word:-

```
The average of each word is as follows
[('new', 25.0), ('american', 23.5), ('america', 23.25), ('jobs', 22.25), ('us', 20.25), ('one', 18.75), ('years', 18.25), ('ame
ricans', 17.25), ('know', 17.0), ('year', 16.0)]
```

The image below shows the standard deviation of each word:-

```
The standard deviation of each word is as follows
[('right', 10.279429296739517), ('us', 8.421203397773187), ('get', 8.220908303425682), ('american', 7.852812659593164), ('ameri
cans', 7.5), ('would', 7.118052168020874), ('year', 7.0710678118654755), ('families', 6.652067347825035), ('world', 6.582805886
043833), ('tax', 6.454972243679028)]
```

We also must compare the word count of year following the 4-year window(such as 2013 for 2009-2012)to average plus twice the standard deviation of the same word. For this we load the contents of the following year into a separate rdd. I also calculated the average plus twice the standard deviation and stored in into a separate rdd . I then used the .lookup() function to search for word count higher than the average plus twice the standard deviation of the 4-year window. The image below shows the output of the words in the following year that meet the criteria(average plus twice standard deviation):-

```
the frequency word exceeding the average plus two standard deviations is :- core
the frequency word exceeding the average plus two standard deviations is :- countries
the frequency word exceeding the average plus two standard deviations is :- courage
the frequency word exceeding the average plus two standard deviations is :- creation
the frequency word exceeding the average plus two standard deviations is :- cyber
the frequency word exceeding the average plus two standard deviations is :- decisions
the frequency word exceeding the average plus two standard deviations is :- deeper
the frequency word exceeding the average plus two standard deviations is :- democracy
the frequency word exceeding the average plus two standard deviations is :- deserve
the frequency word exceeding the average plus two standard deviations is :- developing
the frequency word exceeding the average plus two standard deviations is :- diplomatic
the frequency word exceeding the average plus two standard deviations is :- drive
the frequency word exceeding the average plus two standard deviations is :- earlier
the frequency word exceeding the average plus two standard deviations is :- earned
the frequency word exceeding the average plus two standard deviations is :- east
the frequency word exceeding the average plus two standard deviations is :- economists
the frequency word exceeding the average plus two standard deviations is :- efforts
the frequency word exceeding the average plus two standard deviations is :- elected
the frequency word exceeding the average plus two standard deviations is :- enforcement
the frequency word exceeding the average plus two standard deviations is :- engine
the frequency word exceeding the average plus two standard deviations is :- engineering
the frequency word exceeding the average plus two standard deviations is :- ensures
the frequency word exceeding the average plus two standard deviations is :- entire
the frequency word exceeding the average plus two standard deviations is :- europe
the frequency word exceeding the average plus two standard deviations is :- example
the frequency word exceeding the average plus two standard deviations is :- expand
```

Calculating frequency of co-occurence and Lift Between two words:

This is the part 2 of the assignment in which we must calculate the frequency of co-occurrence of words in the SOTU text within the same sentence . In the pre-processing, I have split the contents by "." Instead of " "(space) as the co-occurrence of the same sentence is asked. To make it a word-pair of 2, I have used itertools.combinations(x,2). I have then calculated the number of same word pairs with ".reduce(lambda x, y: x + y)" command. Now, we must only print and take word pairs whose occurrence is more than 10 so I have used ".filter(lambda KeyValue: KeyValue[1]>10)" to filter out only occurrence more than 10.

```
printing all the frequent pairs above value 10
[(('work', 'together'), 20), (('net', 'onetwothree'), 235), (('two', 'countries'), 12), (('united', 'america'), 17), (('god',
'america'), 19), (('subject', 'attention'), 12), (('every', 'us'), 11), (('right', 'thing'), 15), (('government', 'governmen
t'), 17), (('search', 'search'), 235), (('war', 'war'), 12), (('one', 'year'), 11), (('must', 'america'), 11), (('may', 'nece
ssary'), 11), (('us', 'america'), 11), (('united', 'nations'), 11), (('bless', 'america'), 18), (('must', 'war'), 11), (('sub
ject', 'congress'), 13), (('attention', 'congress'), 12), (('last', 'session'), 11), (('one', 'another'), 13), (('let', 'mak
e'), 15), (('last', 'year'), 26), (('google', 'analytics'), 235), (('recommend', 'congress'), 13), (('america', 'world'), 2
0), (('states', 'states'), 12), (('minimum', 'wage'), 11), (('government', 'states'), 13), (('new', 'states'), 12), (('woul
d', 'us'), 12), (('let', 'us'), 46), (('let', 'give'), 13), (('world', 'war'), 13), (('one', 'nation'), 11), (('years', 'ag
o'), 17), (('com', 'urchin'), 235), (('god', 'bless'), 34), (('five', 'years'), 12), (('two', 'years'), 17), (('consideratio
n', 'congress'), 16), (('economic', 'growth'), 11), (('submitted', 'senate'), 11), (('federal', 'government'), 13), (('worl
d', 'world'), 11), (('good', 'good'), 15), (('every', 'one'), 15), (('four', 'years'), 11), (('would', 'would'), 22), (('firs
t', 'time'), 14), (('every', 'american'), 26), (('one', 'us'), 17), (('free', 'world'), 13), (('new', 'new'), 16), (('men',
'women'), 23), (('one', 'one'), 17), (('get', 'done'), 12), (('must', 'make'), 17), (('nation', 'nation'), 11), (('governmen
t', 'would'), 12), (('state', 'union'), 16), (('one', 'states'), 12), (('time', 'time'), 12), (('states', 'america'), 18),
(('social', 'security'), 24), (('united', 'states'), 157)]
```

I have also taken 20 frequent word-pairs using .take(20).The image below shows 20 frequent word-pairs.

```
printing 20 frequent word-pairs
[(('work', 'together'), 20), (('net', 'onetwothree'), 235), (('two', 'countries'), 12), (('united', 'america'), 17), (('god',
'america'), 19), (('subject', 'attention'), 12), (('every', 'us'), 11), (('right', 'thing'), 15), (('government', 'governmen
t'), 17), (('search', 'search'), 235), (('war', 'war'), 12), (('one', 'year'), 11), (('must', 'america'), 11), (('may', 'nece
ssary'), 11), (('us', 'america'), 11), (('united', 'nations'), 11), (('bless', 'america'), 18), (('must', 'war'), 11), (('sub
ject', 'congress'), 13), (('attention', 'congress'), 12)]
```

Now we must calculate the lift between two words. lift is a measure of performance and used specifically in association rule. Lift controls for the support (frequency) of consequent while calculating the conditional probability of occurrence of {B} given {A}. A value of lift greater than 1 vouches for high association between {A} and {B}. More the value of lift, greater are the chances of preference to get the word {B} if the word {A} has already occurred .The formula for lift(A,B)= P(A ∩B)/P(A) * P(B). P(A ∩B) can now be calculated using the word-pairs word count and P(A) and P(B) can we be calculated individually. Also adjusted to print only lift higher than 3.This means that words with lift as 3 or more will occur more frequently than words with lower lift with respect to the comparing another word. The image below shows the lift between words:-

```
the lift is
('without', 'received') = 0.00040096230954290296
('meet', 'men') = 0.0002293052052281586 7
('whether', 'left') = 0.0009009009009009009
('defensive', 'land') = 0.004761904761904762
('land', 'legislature') = 0.0026455026455026454
('character', 'call') = 0.0015384615384615385
('general', 'american') = 0.0002977963073257892
('states', 'humane') = 0.0021008403361344537
('objects', 'merit') = 0.08333333333333333
('interests', 'union') = 0.0009861932938856016
('calculations', 'seem') = 0.02564102564102564
('may', 'object') = 0.0008064516129032258
('commercial', 'mutually') = 0.018518518518518517
('relations', 'mutually') = 0.016666666666666666
('eleven', 'states') = 0.0021008403361344537
('deserving', 'retained') = 0.25
('requisites', 'combined') = 0.5
('mexican', 'negotiated') = 0.025
```

Conclusion:

In conclusion, I have learned how to Input access data from links and load it in spark rdd. I have learned how to pre-process data in certain ways such as removing punctuation marks, HTML commands, stop words and URL addresses. I have learned and used varies spark rdd commands such as map , filter , mapValues, reduceByKey , flatMap and more . Through this process I was able to do word count of words , take average of words and even calculate standard deviation of the word. I have also learned how to pair data using itertools combinations which helped in calculation of the frequency co-occurrence of word pairs. This co pairing of words helps in the calculation of lift of two words which helps in knowing the probable occurrence of one word with another.