CS 657 Massive Mining Datasets
Assignment 1

Maitreyi Pitale - G01326362
Sai Sahana Bhargavi - G01330358

Introduction

The assignment consists of the data of the text transcriptions of the State of the Union Addresses given by the Presidents to Congress from 1790 to the present year.

Obtaining the datasetAll data in this assignment has been obtained from the State of the Union Website. (http://stateoftheunion.onetwothree.net/)

The data for the preprocessing was obtained from

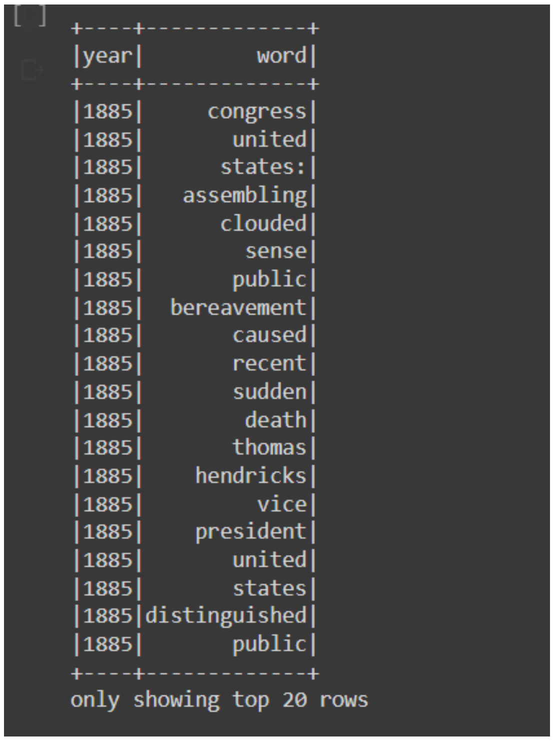
```
url = "http://stateoftheunion.onetwothree.net/texts/index.html"

headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36'}
```

Using the zip file of texts, data can then be accessed through the pyspark code which contains the preprocessing of texts into words, year, count, and standard deviation. The initial iteration of the assignment included using this compressed file, which included a single text file with all addresses. Splitting this file into individual addresses within the SparkContext was quite challenging, however.

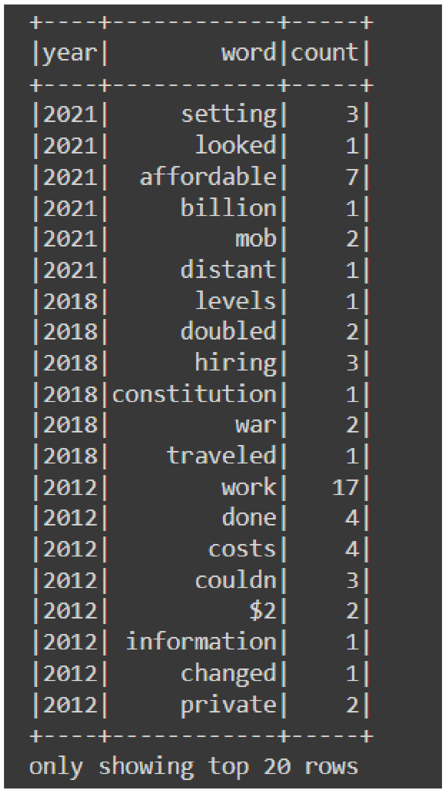
Preprocessing Data

After that, the address data files were imported into the spark context. I used the Link.split() function to break the column into year+word+count based on the identifier to return the ArrayTypeTo accomplish this, I used the wholeTextFiles function, which places each file into an RDD as a tuple (filename+filename, contents). It is very convenient to carry out operations with this format.

The first step was to clean up the fetched data by removing punctuations, HTML tags, and stopwords. It was also necessary to lowercase the text. For this purpose, three different functions have been written. This function was used in the map function that allowed each row in the rdd to be manipulated. The result for an rdd with cleaned text content with the same tuple format. I have used regex_replace for removing html tags, stopwords and spaces.



```
[ ]
    +----+------------+
    |year|        word|
    +----+------------+
    |1885|    congress|
    |1885|      united|
    |1885|     states:|
    |1885|  assembling|
    |1885|     clouded|
    |1885|       sense|
    |1885|      public|
    |1885| bereavement|
    |1885|      caused|
    |1885|      recent|
    |1885|      sudden|
    |1885|       death|
    |1885|      thomas|
    |1885|   hendricks|
    |1885|        vice|
    |1885|   president|
    |1885|      united|
    |1885|      states|
    |1885|distinguished|
    |1885|      public|
    +----+------------+
    only showing top 20 rows
```



```
+----+------------+-----+
|year|        word|count|
+----+------------+-----+
|2021|     setting|    3|
|2021|      looked|    1|
|2021|  affordable|    7|
|2021|     billion|    1|
|2021|         mob|    2|
|2021|     distant|    1|
|2018|      levels|    1|
|2018|     doubled|    2|
|2018|      hiring|    3|
|2018|constitution|    1|
|2018|         war|    2|
|2018|    traveled|    1|
|2012|        work|   17|
|2012|        done|    4|
|2012|       costs|    4|
|2012|      couldn|    3|
|2012|          $2|    2|
|2012| information|    1|
|2012|     changed|    1|
|2012|     private|    2|
+----+------------+-----+
only showing top 20 rows
```

Fig(1) - Year+Word Data

Fig(2) - Year +Word + Count

Part 1

Task 1 - Compute the average use of every word over all the years

Each row in the udf has to be separated into an array of words for the word count. The filepath+filename were removed, and the resulting udf was a massive array with each item containing an array of text. I used the GroupBy Function from the pool of aggregate functions to get the column name and its count as the argument. This was then flat-mapped into a single array containing all words. The words were then converted into tuples in the format (word, 1) and aggregated using the key (the word). The total word counts of each work were the outcome. The numbers were divided by the number of years from 1970 to 2021, yielding the annual average usage of each term.

Task 2 - Compute the maximum and minimum times a word appears in all the addresses and consider only pairs of words that appear together more than 10 times.

For this procedure, a similar strategy was used. The original rdd containing preprocessed data was reused. The filepath was removed from the initial half of each row in the tuple, leaving only the filename or the date of the address as a string. The sums of each word on each day were calculated using the above-mentioned reduce aggregation approach. The date was then removed, resulting in a (word, sum) tuple rdd with many duplicates that were then aggregated. The output was a (word, [sum array]) tuple, which was then sent through the min() and max() methods to display the least and maximum number of times each word appeared in all addresses.

Task 3 -Starting in 2009, compute the average and standard deviation of times a word appears in a window of four years

Similarly to the frequency count technique, each year was substituted by the "range of years" or the presidential term the year came between, rather than a (year, word) tuple. This operation resulted in a tuple (word, [(start, end), freq)). This tuple was then turned into (word, [(start, finish), [frequencies in the 4 years]) using aggregation and grouping. The array was exposed to the mean() and stddev() methods, which reported the average and standard deviation of each word that appeared in each four-year frame.

Task 4 -Use the previous result to output the words that appear in the year following the window with a frequency that exceeds the average plus two standard deviations

For the final section of the question, the earlier estimated frequency rdd ((word, year), freq) was added to the rdd determined from the preceding question (('word', end + 1), avg+2std deviations).

Task 5 -Count the frequency of repetitive words in the SOTU text within the same sentence.

Data frames were employed in this part of the work. As in Part 1, each sentence was parsed and cleaned from scratch, however the "." punctuation was not cleaned because it was necessary to separate each sentence. After splitting each phrase, the words were retrieved and stored in a dataframe. Each word was also assigned a number that indicated which phrase it belonged to. Following this, each word was turned into pairs. This was accomplished by performing a self join on the data frame, resulting in a data frame with two rows containing unique word pairs. It was made sure to avoid the same word being on both parts of the pair, as well as ensuring that no pair had its inverse (x, y) -> (y, x) in the data frame. Their frequency of co-occurrence was then calculated by grouping by both the words of the pair columns and then calling count().

Task 5 - Consider only pairs of words that appear together more than 10 times.

Output 20 frequent pairs of words.

Using the previous dataframe, a lift_filter had to be applied on the data frame where the count of each pair exceeded 10.

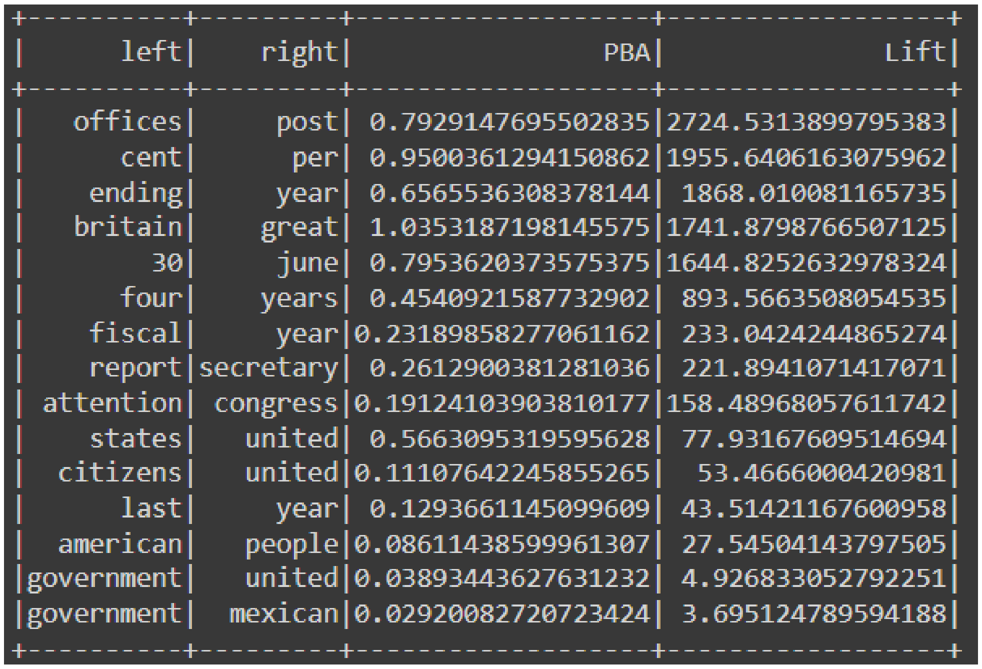
Task 5 -Modify your program to compute the conditional Probability P(B/A) where A and B are words.

Cases when P(A/B) is more than 0.8 P(A intersection B) / P(B|A) is the probability of P(B|A) (A). The frequencies of each pair were divided by the total number of pairings in P(A intersection B), and the frequencies of each word were divided by the total number of words in P(A). This formula was computed as a new column and shown using Spark SQL queries. P(A|B) is the same as P(A) == P(A) (B).

Task 6 - Compute the lift between two words and output those whose lift is bigger than 3.0.?

The lift is calculated as P(A intersection B) / P(A) * P(A) (B). This was done as a new column. Confidence / Expected Confidence = Lift The greater the lift, the less likely it is that the association between the two elements is merely coincidental. Examining the results for word pairings with lift 3.0 demonstrates this.

Task 7 - PRESENT THE RESULTS IN A LEGIBLE MANNER. FOR EXAMPLE: A LIST OF THE 10 MOST SALIENT WORDS IN EACH CASE



```
+----------+---------+-------------------+------------------+
|      left|    right|                PBA|              Lift|
+----------+---------+-------------------+------------------+
|   offices|     post| 0.7929147695502835|2724.5313899795383|
|      cent|      per| 0.9500361294150862|1955.6406163075962|
|    ending|     year| 0.6565536308378144| 1868.010081165735|
|   britain|    great| 1.0353187198145575|1741.8798766507125|
|        30|     june| 0.7953620373575375|1644.8252632978324|
|      four|    years| 0.4540921587732902| 893.5663508054535|
|    fiscal|     year|0.23189858277061162| 233.0424244865274|
|    report|secretary| 0.2612900381281036| 221.8941071417071|
| attention| congress|0.19124103903810177|158.48968057611742|
|    states|   united| 0.5663095319595628| 77.93167609514694|
|  citizens|   united|0.11107642245855265|  53.4666000420981|
|      last|     year| 0.1293661145099609| 43.51421167600958|
|  american|   people|0.08611438599961307| 27.54504143797505|
|government|   united|0.03893443627631232| 4.926833052792251|
|government|  mexican|0.02920082720723424| 3.695124789594188|
+----------+---------+-------------------+------------------+
```

Conclusion

The State of the Union dataset was analyzed using the distributed Spark platform, which would be inefficient if basic operations were performed on a single system. Part one identified the most commonly used terms, as well as new words used in introductory speeches as contrasted to those used in the previous presidential term. This allowed for the reduction of a large number of words that appeared to be random data into meaningful combinations of nouns and verbs. Aside from the findings, I learnt about the spark platform, generating and manipulating rdd, web scraping using df, and optimizing actions on these objects within the Spark Context.The preprocessing part of the text data, the left, right column division for the comparison between the last year's repeated words was the interesting part and has to go through a lot of pyspark and python documentation to get the correct lift value >3.0 where I got to learn that a lift value less (larger) than 1 indicates a negative (positive) dependence or substitution (complementary) effect.