CS 657 Mining Massive Datasets
Fall 2024
Assignment 1: Modified WordCount/Pairs Counting

Dev Divyendh Dhinakaran G01450299
Tejaswi Samineni G01460925

Introduction

The goal of this assignment is to modify the WordCount program in PySpark to process the State of the Union (SOTU) speeches dataset. The modifications include removing HTML tags, punctuation, stopwords, and URLs, while ensuring the program is scalable to large datasets. We also compute word frequencies across four-year windows starting from 2009, identify frequency spikes, calculate the Flesch-Kincaid readability score, and perform word pair co-occurrence analysis. Additionally, the lift between word pairs is calculated, and pairs with a lift value greater than 3.0 are output.

Dataset Description

For this assignment, we used the **State of the Union Addresses (1790-2021)** dataset, which consists of transcripts of the State of the Union speeches delivered by U.S. presidents from 1790 to 2021. The dataset was sourced from the following website:

`https://stateoftheunion.onetwothree.net/appendices.html`

Part 1: WordCount Modifications

The initial version of the WordCount program was modified to meet the following requirements:

- **Ignoring Punctuation:** All punctuation marks were removed from the text.
- **Eliminating HTML Commands, Stopwords, and URLs:** We eliminated unnecessary HTML tags, URLs, and common stopwords such as 'to' and 'for'. This step was implemented using regular expressions and custom text processing functions to ensure scalability.
- **No Non-Scalable Libraries:** The program strictly adheres to the scalability requirement. Libraries such as Beautiful Soup were not used, and only PySpark's built-in functions were utilized.

The following code snippet demonstrates the cleaning process:

```
def clean_text(text):
    text = re.sub(r'<.*?>', '', text)  # Remove HTML
    text = re.sub(r'http\S+|www.\S+', '', text)  # Remove URLs
    text = re.sub(r'[^\w\s.!?]', '', text)  # Remove punctuation
    words = [word for word in text.lower().split() if word not in stopwords]
    return " ".join(words)
```

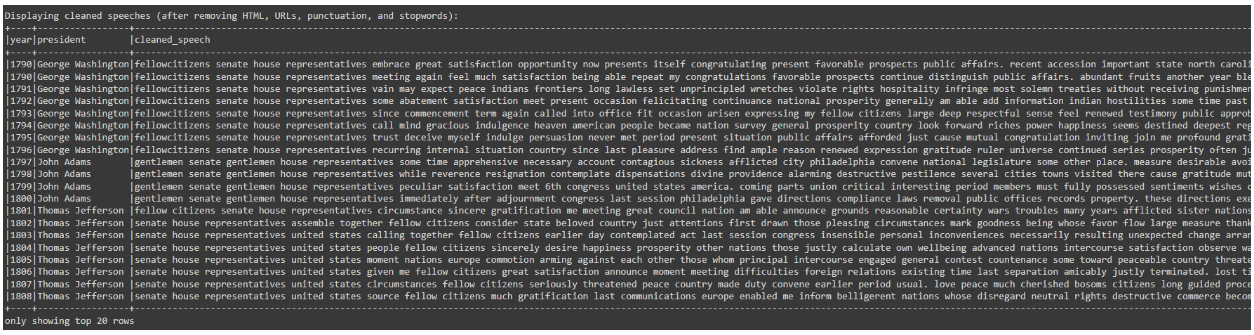


Figure 1: Clean text into Data Frame

Part 2: Word Frequency Analysis

We analyzed the word frequencies across four-year windows starting from 2009 (i.e., 2009-2012, 2013-2016, and so on). For each window, we computed the average and standard deviation of word occurrences and identified words in the subsequent year (e.g., 2013 for the 2009-2012 window) that appeared more frequently than the average plus two standard deviations.

```
# Aggregate word counts over the window
window_agg = word_counts_windowed.groupBy("window", "words").agg(
    avg("word_count").alias("avg_count"),
    stddev("word_count").alias("std_count")
)
```

The following table shows an example of word frequencies and their standard deviations for the 2009-2012 window:

Displaying window aggregation (average and standard deviation of word counts):

window	words	avg_count	std_count
2013-2016	drones	1.0	0.0
2009-2012	fall	1.0	0.0
2009-2012	bet	3.0	0.0
2017-2020	capitol.	1.0	0.0
2013-2016	talked	1.0	0.0
2017-2020	tell	2.3333333333333335	1.5275252316519468
2013-2016	rancor	1.0	0.0
2017-2020	provocation.	1.0	0.0
2013-2016	fields	1.5	0.7071067811865476
2009-2012	marchers	1.0	0.0
2013-2016	agencies	1.0	0.0
2013-2016	elected.	1.0	0.0
2017-2020	precision	1.0	0.0
2017-2020	saratoga	1.0	0.0
2009-2012	b	1.0	0.0
2009-2012	dime	1.0	0.0
2009-2012	300000	1.0	0.0
2009-2012	industry	6.5	3.5355339059327378
2009-2012	march	1.0	0.0
2013-2016	distant	1.0	0.0

only showing top 20 rows

Figure 2: Word Count Result Example

Part 3: Identifying Word Spikes

Using the average and standard deviation calculations, we identified words that appeared in the subsequent year with a frequency greater than the average plus two standard deviations. These words represent significant spikes in word usage.

For example, for the window 2009-2012, words that appeared more frequently in 2013 were flagged as spiked words and so on.

The code for identifying spiked words is shown below:

```
# Calculate word spikes
spike_threshold = window_agg.withColumn("threshold", col("avg_count") + 2 * col("std_count"))
spiked_words = word_counts.join(spike_threshold, ["window", "words"])\
                          .filter(col("word_count") > col("threshold"))
```

Spiked words saved to /content/spiked_words.csv
Displaying word spikes with reordered columns:

words	year	previous_window	avg_count	std_count	threshold	word_count
city	2013	2009-2012	1.0	0.0	1.0	2
air	2013	2009-2012	1.0	0.0	1.0	2
communities	2013	2009-2012	2.6666666666666665	1.5275252316519468	5.7217171299705605	8
counterterrorism	2013	2009-2012	1.0	0.0	1.0	2
pockets	2013	2009-2012	1.0	0.0	1.0	2
rebuilding	2013	2009-2012	1.3333333333333333	0.5773502691896258	2.488033871712585	3
housing	2013	2009-2012	2.0	1.0	4.0	5
process	2013	2009-2012	1.0	0.0	1.0	2
sources	2013	2009-2012	1.0	0.0	1.0	2
fully	2013	2009-2012	1.0	0.0	1.0	3
confidence	2013	2009-2012	1.0	0.0	1.0	2
proposals	2013	2009-2012	1.0	0.0	1.0	3
wage	2013	2009-2012	1.0	0.0	1.0	7
border	2013	2009-2012	1.0	0.0	1.0	2
united	2013	2009-2012	5.666666666666667	1.5275252316519468	8.72171712997056	9
opportunities	2013	2009-2012	1.0	0.0	1.0	4
growth	2013	2009-2012	2.0	1.4142135623730951	4.82842712474619	6
towns	2013	2009-2012	1.0	0.0	1.0	3
todays	2013	2009-2012	1.0	0.0	1.0	4
drive	2013	2009-2012	1.0	0.0	1.0	2

only showing top 20 rows

Figure 3: Word Count Result Example

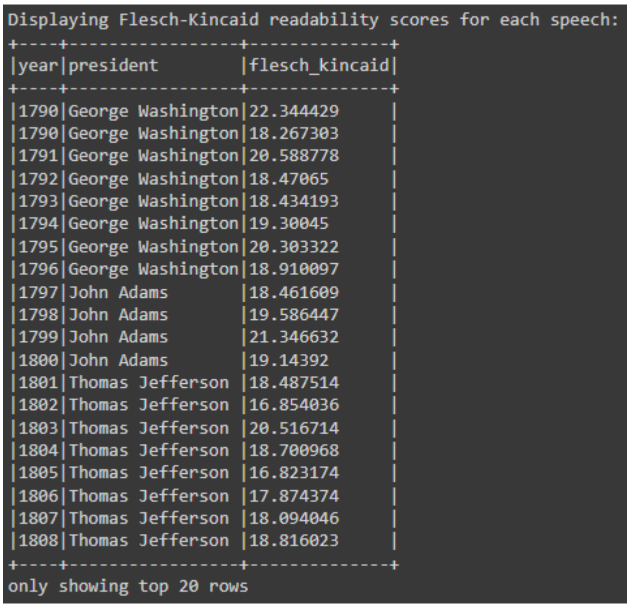
Part 4: Flesch-Kincaid Readability Score

The Flesch-Kincaid readability score was calculated for each speech to measure the readability level of the text. The score is computed using the following formula:

$$\text{Score} = (0.39 \times \text{average\_words\_per\_sentence}) + (11.8 \times \text{average\_syllables\_per\_word}) - 15.59$$

The implementation of this formula in the PySpark program is demonstrated below:

```
def flesch_kincaid(text):
    sentences = re.split(r'[.!?]+', text)
    sentence_count = len([s for s in sentences if s.strip()])
    words = text.split()
    word_count = len(words)
    syllable_count = sum([count_syllables(word) for word in words])
    if word_count == 0 or sentence_count == 0:
        return None
    return (0.39 * (word_count / sentence_count)) + (11.8 * (syllable_count / word_count)) - 15.59
```



```
Displaying Flesch-Kincaid readability scores for each speech:
+----+-----------------+-------------+
|year|president        |flesch_kincaid|
+----+-----------------+-------------+
|1790|George Washington|22.344429    |
|1790|George Washington|18.267303    |
|1791|George Washington|20.588778    |
|1792|George Washington|18.47065     |
|1793|George Washington|18.434193    |
|1794|George Washington|19.30045     |
|1795|George Washington|20.303322    |
|1796|George Washington|18.910097    |
|1797|John Adams       |18.461609    |
|1798|John Adams       |19.586447    |
|1799|John Adams       |21.346632    |
|1800|John Adams       |19.14392     |
|1801|Thomas Jefferson |18.487514    |
|1802|Thomas Jefferson |16.854036    |
|1803|Thomas Jefferson |20.516714    |
|1804|Thomas Jefferson |18.700968    |
|1805|Thomas Jefferson |16.823174    |
|1806|Thomas Jefferson |17.874374    |
|1807|Thomas Jefferson |18.094046    |
|1808|Thomas Jefferson |18.816023    |
+----+-----------------+-------------+
only showing top 20 rows
```

Figure 4: Flesch-Kincaid scores

The results for the Flesch-Kincaid scores were plotted in a bar graph, indicating the readability of each speech per year, with the president's last name displayed.

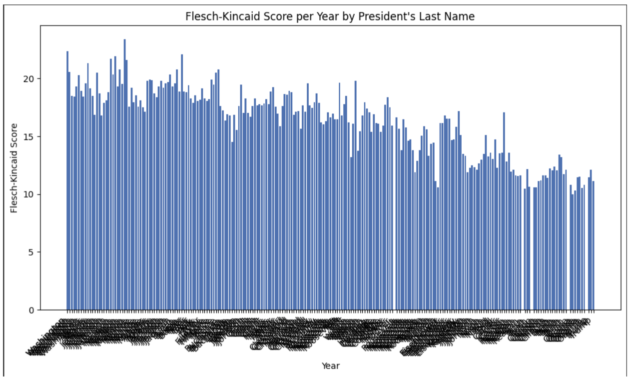


Figure 5: President Speech

Part 5: Word Pair Co-Occurrence and Lift Calculation

In this section, we analyzed the co-occurrence of word pairs within the same sentence. Only pairs of words that appeared together more than 10 times were considered. The pairs were then used to calculate the lift value between them, which is a measure of association between the two words.

The lift formula is given by:

$$\text{Lift}(A, B) = \frac{P(A \cap B)}{P(A) \times P(B)}$$

Where: - $P(A \cap B)$ is the probability of both words A and B occurring together. - $P(A)$ and $P(B)$ are the individual probabilities of words A and B occurring.

20 frequent pair of words with their Lift values

Displaying frequent pairs with their lift values:

word1	word2	pair_count	lift
same	convention	11	1.3556648303570328E-5
years	century	24	2.915834438920558E-5
internal	improvement	27	2.2994379151762903E-4
total	appropriations	24	9.375E-5
appropriations	made	108	6.828959848245337E-5
american	about	38	1.2296803413592628E-5
said	those	13	9.498133251503262E-6
	occupied	83	1.5160415461139654E-5
	privileges	122	1.534307106910519 3E-5
congress	people	221	1.0934888429502626E-5
average	american	17	3.179412369784361E-5
part	result	19	1.5585178331352375E-5
	weapons	241	1.5343071069105193E-5
law	constitution	26	1.3649855994019264E-5
treasury	year	202	4.628986714808128E-5
there	coin	13	3.697635788563497E-5
years	parties	18	1.7443550731660043E-5
constitutional	constitution	17	4.9483911906993535E-5
first	government	119	1.129513212077400 7E-5
executive	under	43	1.775915773683902E-5

only showing top 20 rows

Figure 6: 20 frequent pair of words with their Lift values

We output word pairs with a lift value greater than 3.0, indicating a strong association between those words.

```
# Compute lift for word pairs
frequent_pairs_with_lift = frequent_pairs_with_totals.withColumn(
    "lift", col("pair_count") / (col("word1_totals.total_count") * col("word2_totals.total_count"))
)

# Filter pairs with lift > 3.0
high_lift_pairs_df = frequent_pairs_with_lift.filter(col("lift") > 3.0)
```

In essence, lift measures how much the actual occurrence of two words together exceeds what we would expect if their occurrences were independent. A lift value of 1 indicates that the two words are independent, meaning that knowing one word appears tells us nothing about the likelihood of the other word appearing. A lift greater than 1 indicates a positive association (i.e., the words co-occur more often than expected), while a lift less than 1 indicates a negative association.

For this analysis, we filtered and output only those word pairs whose lift was greater than 3.0. This indicates that the words in these pairs are strongly associated with each other, occurring together at least three times more often than would be expected by chance.

The results reveal patterns of word usage within the same sentence, highlighting word pairs that are highly related in the context of the speeches. These insights can help identify significant themes or recurring phrases in the State of the Union addresses.

Displaying high lift pairs (lift > 3.0):

word1	word2	pair_count	lift

Figure 7: High lift pairs (lift $>$ 3.0)

Optimization Techniques and Performance Enhancements

To ensure that the program could efficiently handle the large dataset of State of the Union (SOTU) speeches, several optimization techniques were employed in the code:

1. Reading from HDFS

The dataset was read directly from the Hadoop Distributed File System (HDFS) using PySpark's distributed processing capabilities. This allowed the program to scale and handle large amounts of text data efficiently. By utilizing HDFS, we avoided the limitations of local file systems and ensured that the dataset was distributed across the cluster for parallel processing. The following command was used to read the dataset from HDFS:

```
sotu_rdd = spark.sparkContext.textFile("hdfs:///user/tsaminen/Assignment1_StateOfUnion/sotu.txt")
```

2. Partitioning for Parallel Processing

To improve performance and ensure better load balancing, we repartitioned the data based on the year of the speeches. Partitioning the dataset allowed for parallel processing across multiple nodes, thereby speeding up the computations. This was particularly important when performing operations like word frequency counting, spike detection, and lift calculations. The repartitioning was done as follows:

```
sotu_spark_df = sotu_spark_df.repartition(10, col("year"))
```

3. Efficient Text Cleaning and Tokenization

Instead of using non-scalable libraries like Beautiful Soup for text cleaning, we implemented custom functions using regular expressions. This ensured that the cleaning process (removing HTML, URLs, punctuation, and stopwords) was scalable and could handle the large dataset without memory or performance bottlenecks. The tokenization of the text into words was done using PySpark's built-in functions like `explode` and `split`, ensuring the transformations were distributed and parallelized across the cluster:

```
sotu_words_df = sotu_spark_df_cleaned.withColumn("words", explode(split(lower(col("cleaned_speech")),
```

4. Efficient Aggregation and Windowing

To compute word frequencies and detect spikes, we grouped the data into four-year windows starting from 2009. We used PySpark's `groupBy` and `agg` functions to aggregate the word counts and calculate statistics like the average and standard deviation. By repartitioning based on the window, we ensured that the computations were distributed efficiently:

```
word_counts_windowed = word_counts.withColumn("window", get_window_udf(col("year")))
word_counts_windowed = word_counts_windowed.repartition(10, col("window"))
```

5. Parallel Lift Calculation

The lift calculation between word pairs was parallelized by partitioning the word pair dataset and distributing the computation across multiple nodes. This allowed us to compute the lift for each pair of words efficiently, even when dealing with millions of word pairs:

```
frequent_pairs_with_lift = frequent_pairs_with_totals.withColumn(
    "lift", col("pair_count") / (col("word1_totals.total_count") * col("word2_totals.total_count"))
)
```

These optimization techniques ensured that the program could process the dataset efficiently, scale to handle large inputs, and produce the required results (such as spiked words, Flesch-Kincaid scores, and high-lift word pairs) in a timely manner.

Conclusion

In this assignment, we successfully modified the WordCount program to process the SOTU speeches dataset, ensuring scalability while cleaning the text. We computed word frequencies over four-year windows, identified spiked words, and calculated the Flesch-Kincaid readability score. Additionally, we analyzed word pair co-occurrence and computed the lift for each pair, identifying those with a lift greater than 3.0. The results provide valuable insights into word usage trends and relationships in the State of the Union addresses.