**CSE487/587: DATA INTENSIVE COMPUTING**

Instructor: Bina Ramamurthy

**Data Aggregation, Big Data Analysis**

**&**

**Visualization**

****

**Submitted By:**

**Sourav Bihani**

**50290969**

**Goal:**

The goal of this project is to analyse and visualize the frequency of the words and their co-occurrences in the data (collected from 3 sources - Twitter, New York Times and Common Crawl) which is scrapped over the internet. The scrapped data is fed to the Word Count Map-Reduce and Word Co-occurrences algorithms.

**Data Collection**

**Topic Selected: *Sports***

We collected data for the following subtopics related to sports:

* Football
* Basketball
* Baseball
* Cricket
* Hockey

**Twitter:**

We collected the tweets by specifying an appropriate search query. Below are the steps which we followed:

* Convert search result tweets into data frame.
* Remove the retweets from the tweets.
* Get the text of the tweets.
* Save the text in a text file for further processing.

***Total number of tweets collected: 1,200,00***

**New York Times:**

We collected the New York Times articles using their search API. Below are the steps which we followed:

* Hit the API with a valid secret key, appropriate query string and time frame.
* Process the json received as the response from the API and save the web URL (attribute passed in the json response) of the articles.
* Get the web URL and scrape the data by the using the HTML parser.
* Retrieve the text from the paragraph tag.
* Save the text in a text file for further processing.
* Repeat the last 2 steps until all the web URL are scraped.

***Total number of URLs scrapped: 770***

**Common Crawl:**

Collected common crawl data which is saved in Amazon S3. Following are the steps taken.

* Hit the API <http://index.commoncrawl.org/CC-MAIN-2019-13-index?url=https://www.espn.com/&matchType=domain&output=json> with a valid index and domain name.
* Retrieve all the records.
* Iterate over the record list and hit the Amazon S2 bucket.
* Convert the response in the String I/O stream
* Extract the data by using gZip.
* Iterate over the anchor tags and get the URLs.
* Iterate over the URLs and extract the text from the paragraph tag.
* Save the text in a text file for further processing.

***Total number of links scrapped:1593***

**Data Cleaning**

The data scraped from various data sources should be clean before it can be fed to the word count map reduce algorithm. Below are the steps which we followed to clean the data:

* Download the required packages such as NLTK, Stemmer, etc.
* Retrieve the text from a file that needs to be cleaned.
* Remove all the stop words from the text.
* Remove all the punctuations, emoticons and other words that may influence the output of the algorithm.
* Save the cleaned data in a text file.
* Repeat the above steps for all the files for the data sources.

**Implementation**

Below are the steps which we followed in order to achieve the goal of this project:

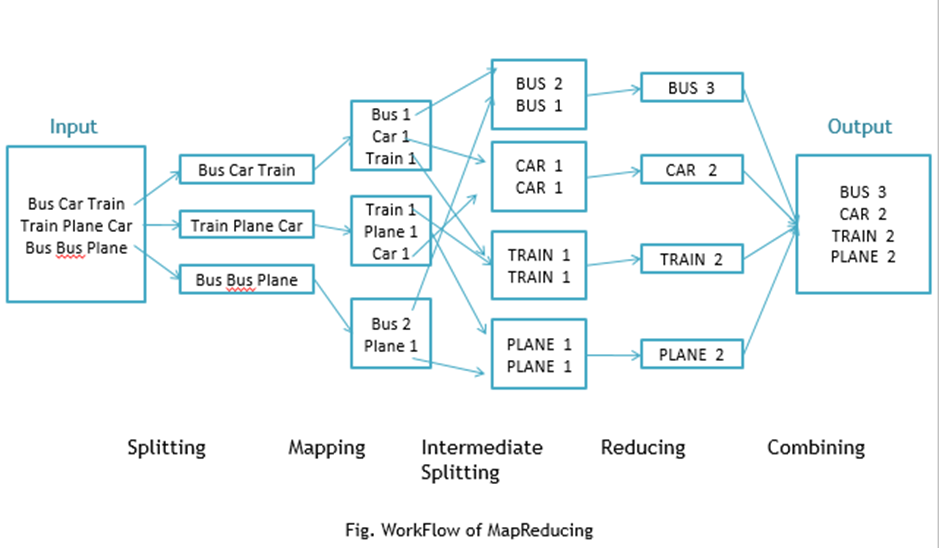
* The cleaned data for all the data sources is basically the prerequisite for the implementation. Once we had the cleaned data, we ran a map reduce algorithm on the Hadoop infrastructure.
* After getting the output file from the Hadoop, we retrieved the top 10 words by sorting the frequency in the descending order.
* After getting the top 10 words, we found their co-occurrences using word co-occurrence algorithm.
* We used Tableau software to analyse and visualize the outputs from the word count and word co-occurrence algorithms.

**Map Reduce Algorithm:**

MapReduce consists of 2 steps:

* Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).
* Reduce Function – It takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

**Workflow:**



Workflow of MapReduce consists of 5 steps:

1. Splitting – The splitting parameter can be anything.

e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).

1. Mapping – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).
2. Intermediate splitting – The entire process runs in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on the same cluster.
3. Reduce – It is nothing, but mostly group by phase.
4. Combining - The last phase where all the data (individual result set from each cluster) is combined together to form a result.

*Reference:* [*https://dzone.com/articles/word-count-hello-word-program-in-mapreduce*](https://dzone.com/articles/word-count-hello-word-program-in-mapreduce)

**Observations:**

The observations have been submitted as a part of Data visualization in Tableau workbook published publicly.

**Sample Code:**

1. **def** tokenize(text, regExForDeletingWords):
2. ps = PorterStemmer()
3. resultTokens = []
4. tokens = [t **for** t **in** text.split()]
5. **for** token **in** tokens:
6. # match = re.sub(r'([^\s\w]|\_)+', '', token)
7. # type(match)
8. # if match is None:
9. **if** token.isalpha():
10. ps.stem(token)
11. resultTokens.append(token)
12. **return** resultTokens

15. **def** SetupDataClean():
16. **import** nltk
17. # if there is a problem in downloading nltk data, then run the Install Certificates.command available in your Python folder and then execute the download command
18. nltk.download('stopwords')
19. #Reference =https://docs.python.org/3/library/re.html
20. deleteString = [ ]
21. deleteWordsRegEx = re.compile(r'(' + ',' +'.' + '"' + '|'.join(deleteString) + ')', re.VERBOSE | re.IGNORECASE)
22. **return** deleteWordsRegEx


26. **def** Clean(regExForDeletingWords,inputFileName, outputFileName):
28. textFile = open(outputFileName,"w")
29. **for** filelineno, line **in** enumerate(open(inputFileName)):
30. **if** len(line) == 0:
31. **continue**;
32. line = line.strip()
33. line = line.lower();
34. punctuationToRemove = list(string.punctuation)
35. stop = stopwords.words('english') + punctuationToRemove
36. new\_line = [term **for** term **in** tokenize(line,regExForDeletingWords) **if** **not** term **in** stop]
37. **for** word **in** new\_line:
38. textFile.write(word + " ")
39. textFile.close();

**Code Features:**

1. **Abstraction:** We have tried to achieve abstraction so that users do not have to deal with the entire code.
2. **Reusability**: We have divided the steps into different components and each component is coded as a function. Hence, users can input files dynamically and can create the chart.
3. **Comments**: We have added comments where ever required.