CSE 587 DIC Lab 2 Report

Akshay Chaudhary
Rohan Hemanshu Sheth

Description

- In this project, we will collect data from three different sources as tweets from Twitter, New York Times articles and articles from multiple random web sources using Common Crawl about a handful of sub-topics, each related to a single topic. Twitter is an opinion-based source while New York Times is research-based. We do this data aggregation from multiple sources using the APIs provided by these sources.
- The unstructured data collected will be processed and then stored on WORM infrastructure Hadoop. The processing and analysis of the data will be done using Python as the processing language.
- We will then apply classical big data analytical method of MapReduce for word count and word co-occurrence to the stored data on the instance of Hadoop infrastructure on AWS.
- Next, data visualization will be done using the visualization software Tableau.

New York Times:

Methodology

1. **Data Collection**
    - For the purpose of data collection, we have selected the topic as sports and sub-topics as soccer, golf, baseball, basketball, tennis and hockey.
    - We then install the python package nytimesarticle which is a python wrapper for the New York Times Article Search API to fetch NYT articles of the current year.. This package allows querying the API through python.
    - The response returned by the NYT API is then parsed in a list of dictionaries of relevant fields. The search function returns a dictionary of the first 10 results. To get the next 10, we have to use the page parameter in a for loop.

```
articles = api.search(q = query,
        begin_date = 20190101,
        page = i)
```

    - We are using the python package requests to fetch the articles from their URLs as follows:

```
page = requests.get(soccer_articles[0][0]['url'])
```

    - We then use the python library BeauitfulSoup which is a HTML parser used for pulling data out of HTML and XML files to save all the text in HTML <p> tags of the articles into a text file, paragraph wise.

```
soup = BeautifulSoup(page.content, 'html.parser')
```

2. Data Processing

- **Stopwords removal**:

  After the article is saved in a text file, cleaning is done by removing stopwords. A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that takes up unnecessary space and processing time. They can safely be ignored without sacrificing the meaning of the sentence. We download stopwords to our python environment by using package NLTK (Natural Language Toolkit). We add additional stopwords if necessary.

- **Stemming**:

  Stemming is a process of removing and replacing word suffixes to arrive at a common root form of the word. We use Porter Stemmer from NLTK package in python to stem words while sentences are tokenized into words to stem them.

  ```
  porter = PorterStemmer()
  stemmed_word = porter.stem(word)
  ```

- **Special characters and numbers removal:**

  Since the special characters and numbers are irrelevant for MapReduce methods for word count and co-occurrence in this project, we remove them from the text by using re module in python that provides regular expression matching operations. Both patterns and strings can be Unicode strings as well as 8-bit strings. We also convert all the characters to lowercase using string operation.

  ```
  str=str.lower()
  new_str = re.sub('[^a-z]', ' ', str)
  ```

We repeat this process of data collection and processing for every article fetched in a for loop and append the processed text in a single file for one sub-topic. In this way, we create 6 text files corresponding to each sub-topic containing data of 100 articles each.

3. MapReduce on Hadoop instance of AWS:

These text files with total data of 600 articles are given as input to the instance of Hadoop infrastructure on AWS along with codes for mapper and reducer to run MapReduce for finding word count and word co-occurrence. Reducer for word count sums up the 1’s for every word and returns a file with a count for every word in our data. The top 10 words with the most count are selected and used in the Mapper for word co-occurrence where for every pair of word in the list of top words and the word following it in a paragraph, Mapper prints 1 and this continues for all paragraphs. Reducer again sums up the 1’s for every pair and returns a file with a count for every pair.

4. **Visualization using Tableau:**

The most relevant words and pairs with the most count can be visualized in a software Tableau. For word count, our input is a text file with word counts returned by the Reducer and for word co-occurrence, our input is a csv file with co-occurrence counts.

Twitter:

1. **Data Collection**

- Tweets are collected using the same procedure as in Lab 1.
- Twitter's searchtwitter API is used to collect tweets in a dataframe in R Studio.

uefa.tweets = searchTwitter("cricket",n=5000, lang="en", since='2019-04-15', until='2019-04-21')

- This dataframe is converted into a .csv file in R Studio which is then converted to a .txt file in python which is used for further processing explained below.

2. **Data Processing**

- Processing includes converting text to lowercase, removing special characters, symbols, emojis, numbers, mentions, links.
- Next steps are removal of stopwords and stemming using Porter Stemmer.

```
str = str.lower()
new_str = re.sub(r"http\S+", "", str)
new_str = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",new_str).split())
tweet1 = new_str.encode('ascii', 'ignore').decode('ascii')

tweet2 = re.sub('[^a-zA-Z]', ' ', tweet1)
tweet3 = re.sub(r'\s+[a-zA-Z]\s+', " ", tweet2)
```

MapReduce and Visualization steps are the same as in New York Times.

Common Crawl:

3. Data Collection

We download .wet file from commoncrawl.org. Then we read each record in the file in python 2.7 environment as follows:

```
block = record.payload.read().decode('utf-8')
```

This reads the .wet file in blocks and then the language of the block is detected. If the detected language is English, then the block is checked for the presence of any of the words in the sub-topic query list. If any of the sub-topic word is present in the block, the block is written to a text file, otherwise, next block is checked.

Another approach would be fetching all the urls in the .wet file and collecting them in a list.

```
url = record.header.get('warc-target-uri', None)
list_url.append(url)
```

Then the urls can be checked for the presence of the words in the sub-topic list and then append those urls in a filtered url list. Then the process of data collection is same as in New York Times. Articles can be fetched using requests package using the urls in the filtered urls list and BeautifulSoup can be used as a HTML parser. Then the article text can be saved in a text file, paragraph wise using <p> tag. Instead of filtering URLs, requests package and BeautifulSoup can be used to fetch all articles and they can be checked if they are relevant to the sub-topics and appropriately saved or discarded.

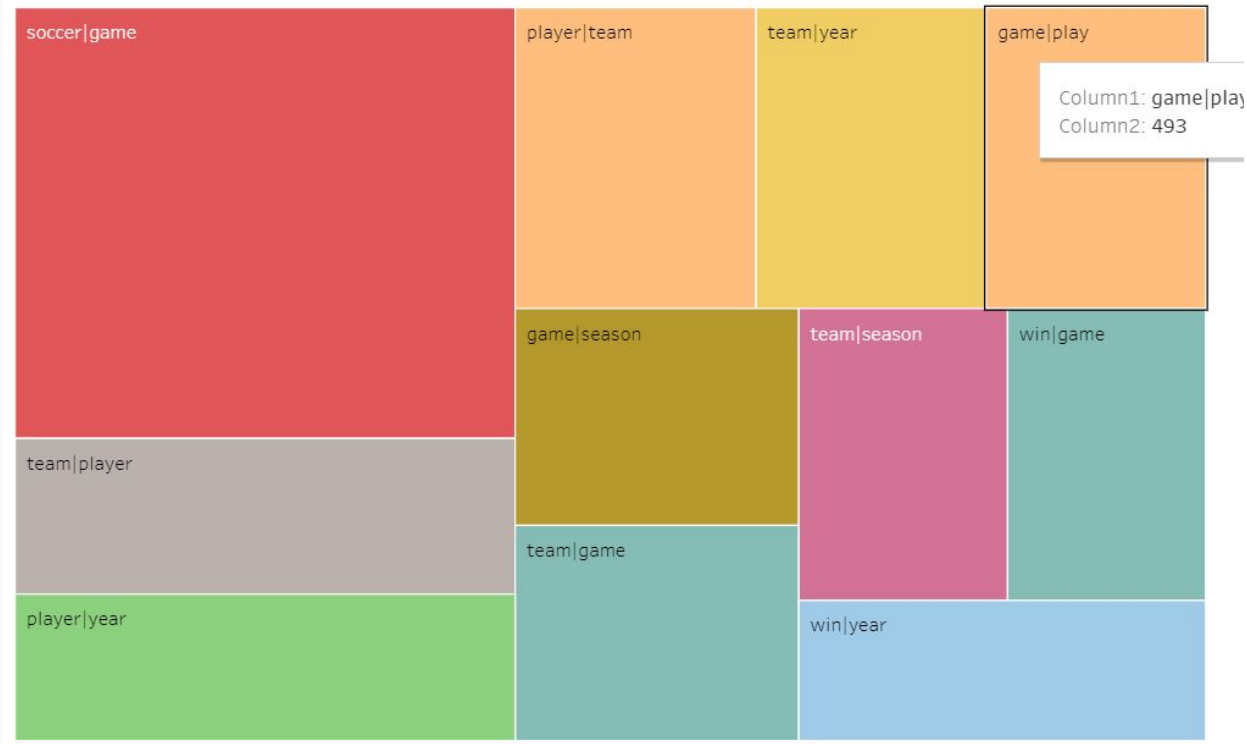
4. Data Processing

Processing of the final text file is the same as described in New York Times section.

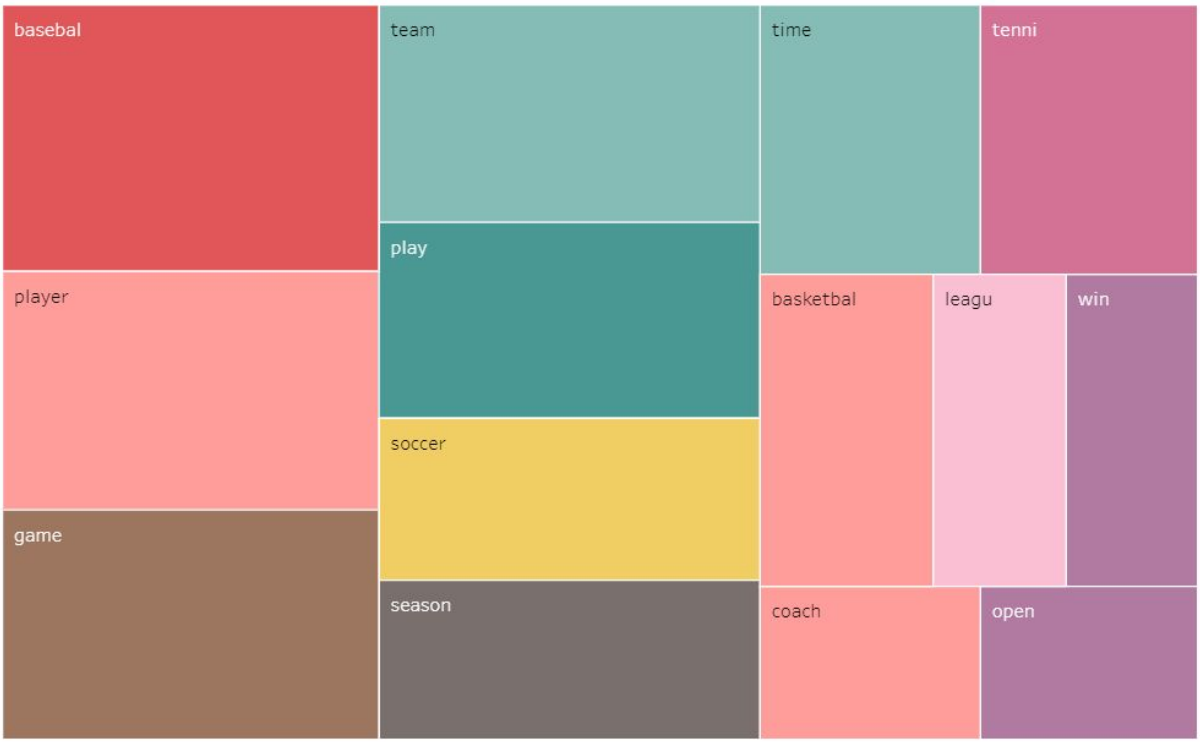
MapReduce and Visualization steps are the same as in New York Times.

Results/Screen shots/Visualization:

<Word_Coocurance_NYTimes>



<Word_Count_NYTimes>

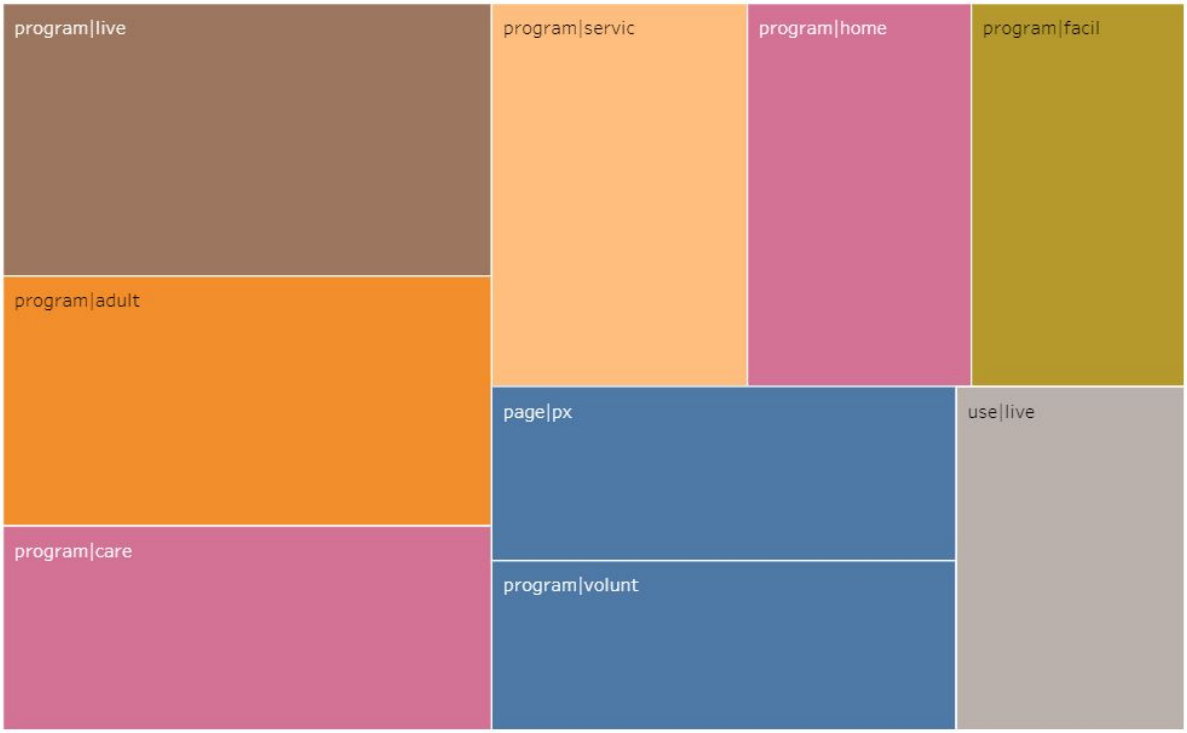


+ableau

<Word_Cooccurance_Twitter>

<Word_Count_Twitter>

<Word_Cooccurance_CC>



<Word_Count_CC>