davidjeg 50290785
msundara 50290572

DIC Project 2

CSE 587
April 22, 2019

Analysis of SPORTS for word count and co-occurrence

Introduction

The topic of choice is **SPORTS** and there are 5 different subtopics selected for this lab, namely [Cricket, Football, Basketball, Hockey and Tennis] This topic has always been mentioned in the US all the time and would generate huge volumes of data under each category.

Flow

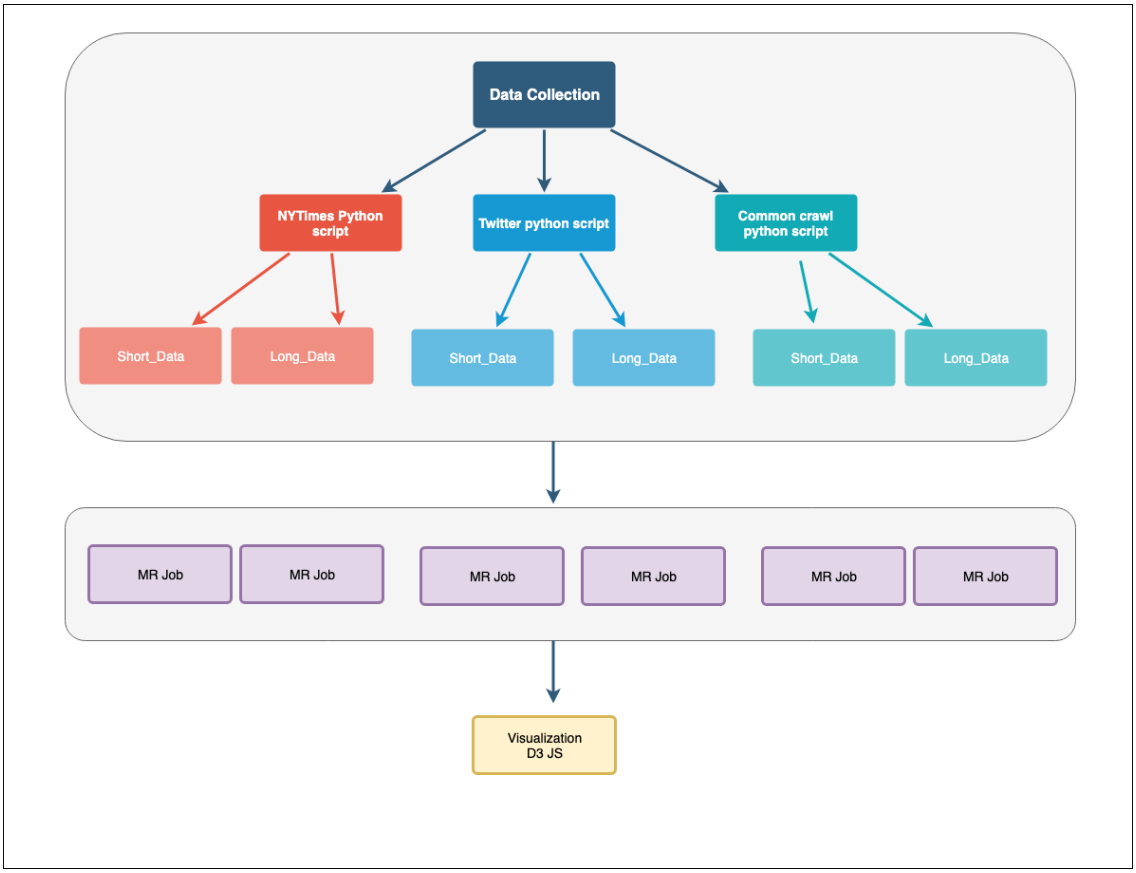


Figure 1: Flow chart

We generate articles from NYTimes, Twitter and Common crawl respectively from Jan 2019 for two categories, long_duration and short_duration. We then use MR and NTLK packages to clean the articles and find the most occurring word and co-occurring pairs. Later D3js is used to visualize the results.

DIC Project 2

1. NYTimes Data Extraction

Algorithm 1: NY Search - API access

```
from nytimesarticle import articleAPI
for i in range(30):
 api = articleAPI(apikey)
 response = api.search(q = query, begin_date = int(today), page = i)
 for j in range(len(response['response']['docs'])):
   urls.append(response['response']['docs'][j]['web_url'])
```

From nytimesarticle we provide API key to extract news article for query and current date. We then parse json and add urls to list. The we use beautiful soup to extrcat the text and save it into a file locally.

Algorithm 2: HTML Parser

```
for i in urls:
  response = requests.get(i)
  soup = BeautifulSoup(response.content, 'html.parser')
  soup.prettify()
  for sentences in soup.find_all('p'):
    subset_sentences = subset_sentences + (sentences.string)
  subset_sentences_list.append(subset_sentences)
```

2. Twitter Data Extraction

Algorithm 3: Twitter - API access

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
tweet_api = tweepy.API(auth, wait_on_rate_limit=True)

for t in tweepy.Cursor(api.search,q=query+'-filter:retweets',lang='en').items(10):
        if(t.retweeted == False):
            tweets.append(t.text.encode('utf-8'))
```

From tweepy we provide API key to extract tweets for query. We then parse tweepy object and add tweets to list. Then we extract the tweets and save it into a file locally.

3. Common crawl Data Extraction

Algorithm 4: Common crawl - API access

```
index = '2019-13'
cc_url = "http://index.commoncrawl.org/CC-MAIN-%s-index?" % index
cc_url += "url=%s&matchType=domain&output=json" % domain
response = requests.get(cc_url)
if response.status_code == 200:
  records = response.content.splitlines()
  for record in records:
    record_list.append(json.loads(record))
```

We use the March month common crawler index for extracting query relevant terms. We use REST API call to request from the url. Then parsing the json we access the S3 common crawl for url using offset and index. The file is unzipped and responses are appended in the file locally. This script is heavily inspired from the website https://www.bellingcat.com/resources/2015/08/13/using-python-to-mine-common-crawl/ and uses Python 2.7 environment.

Algorithm 5: Common crawl - Offset and length download

```
offset, length = str(record['offset']), str(record['length'])
offset_end = int(offset) + int(length) - 1
prefix = 'https://commoncrawl.s3.amazonaws.com/'
resp = requests.get(prefix + str(record['filename']),
        headers={'Range': 'bytes={}-{}'.format(offset, offset_end)})
raw_data = StringIO.StringIO(resp.content)
f = gzip.GzipFile(fileobj=raw_data)
data = f.read()
warc, header, response = data.strip().split('\r\n\r\n', 2)
#return response
```

```
515  ls
516  virtualenv -p /usr/bin/python2.7 venv
517  source venv/bin/activate
518  python
519  pip install requests bs4
520  vi commoncrawler.py
521  ls
522  python commoncrawler.py -d bellingcat.com
```

Figure 2: Command in cmd

4. **Search indexes**

Index	Key words
sports	[sports,games,league match, Olympic, athlete]
cricket	[cricket, icc, cricket world cup, IPL, dhoni, virat kohli,]
football	[football, soccer, FIFA manchester united, liverpoolfc, fcb, real madrid]
basketball	[basketball, lebronjames, NBA, NCAA]
hockey	[hockey, NHL, icehockey]
tennis	[tennis, miamiopen, usopen, atpworldtour, federer, nadal]

5. **Mapreduce**

(a) **Word count Mapper** Mapper outputs the key,value pair for each word in the articles provided. The value is 1 and the key is the individual words.

(b) **Word count Reducer** Reducer processes the output of the mapper aggregating the key,value pair of each word. The reducer finally emits the sum as output.

(c) **Word cooccurance Mapper** Similar to word count, we select the pairs of words in the neighbourhood of 1 and determine the key,value. Here value remains 1 and key is the input word. We have done both pairs and stripes approach. Pairs approach runs quicker because there is no object serialization/deserialization for long data.

(d) **Word cooccurance Reducer** Similar to word count, we aggregate the wordpair, value for emitting as results.

The environment setup

- Cloudera Quickstart VM is used for running the Map reduce jobs
- It comes pre installed with Hadoop,Spark,Oozie and so on and requires a 8GB RAM machine for running in a VirtualBox
- In order to run the Cloudera Manager 10GB RAM allocation must be required

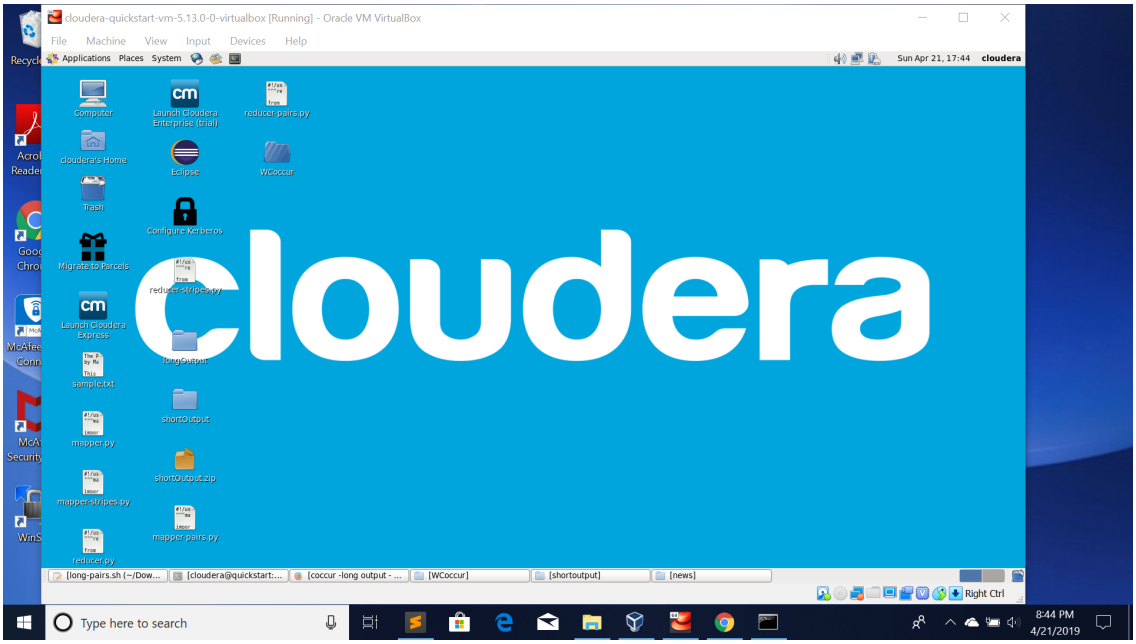


Figure 3: Cloudera VM screenshot

Data upload to Cluster

- The data collected from sources like Twitter,New York Times,Common Crawl is first loaded into the VM
- Data is separated as longData and shortData with subfolders for the three data sources inside
- Both the above mentioned folders is moved to HDFS through the command,it could be either hadoop fs or hdfs dfs
  hdfs dfs -copyFromLocal <localpath> <hdfspath/destionation>

Running MR Job for Word Count

The below is the instructions to run the MR job

Algorithm 6: Shell script for running Map Reduce Jobs in Cloudera VM

```
#!/bin/bash
#Filename mr.sh
echo Filename :"$1"
echo inputDirectory :"$2"
echo outputDirectory: "$3"
for i in {0..5}
do
   echo "Running $i times"
   #HDFS Outpath
   outputpath="$3file$i"
   #Name of the Input File to read
   filename="$1_$i.txt"
   #checks to see if the output directory exists
   hdfs dfs -test -d "$outputpath"
   if [ $? == 0 ]; then
     hdfs dfs -rm -r "$outputpath"
   else
     echo "Output file doesn't exist and will be created when MR job
        ↪ runs"
   fi

    #Command to run the Map Reduce python program
   /usr/lib/hadoop/bin/hadoop jar /usr/lib/hadoop-0.20-mapreduce/
      ↪ contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.13.0.jar -
      ↪ file /home/cloudera/Desktop/mapper.py -mapper "python mapper.
```

```
      ↪ py" -file /home/cloudera/Desktop/reducer.py -reducer "python
      ↪ reducer.py" -input /user/cloudera/"$2"/"$filename" -output /
      ↪ user/cloudera/"$outputpath"
done
#After the successfull run of all the files for a particular topic,the
   ↪ folders are copied from HDFS
 hdfs dfs -copyToLocal "$3" /home/cloudera/Desktop/WordCount
```

- The above shell script can be used to run map reduce task a source which has six subtopics or files
- The script takes three input arguments filename,hdfs input path and hdfs output path
- Then results for the six files are copied from HDFS directory to the local VM path
- The below commands can be used to run for longData for the three different sources

Algorithm 7: Shell script for running Map Reduce Jobs for three different sources

```
chmod +x mr.sh
./mr.sh tw_news longData/twitterData longOutput/tw/
./mr.sh ny_times_news longData/newsData longOutput/news/
./mr.sh cc_news longData/crawlData longOutput/crawl/
```

The below program shows the screenshot of execution of the above command for Twitter Data

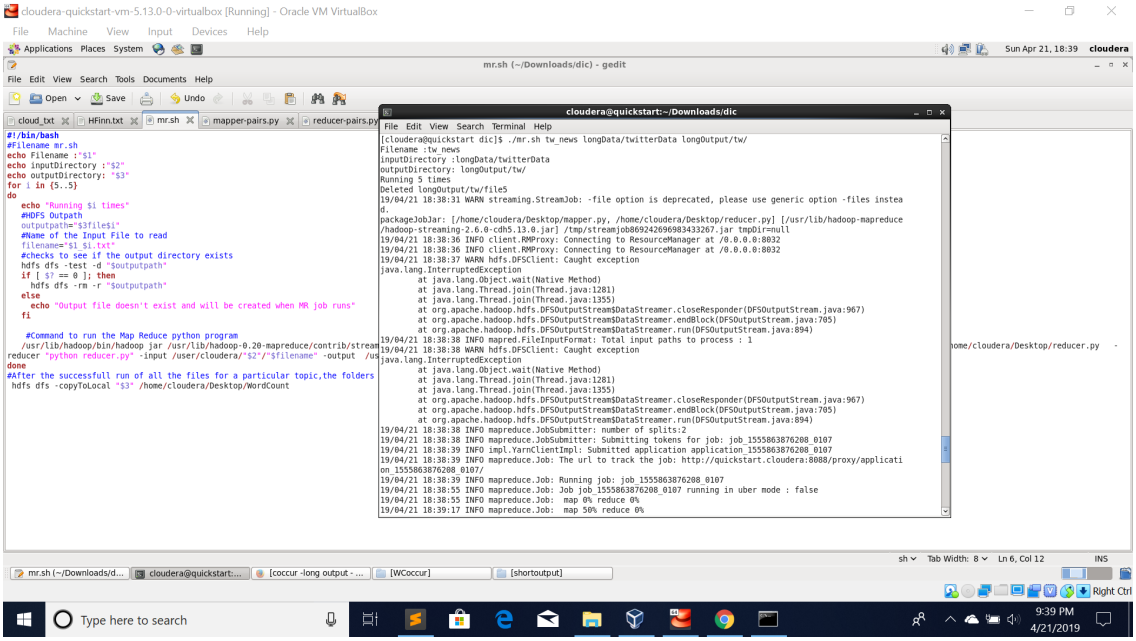
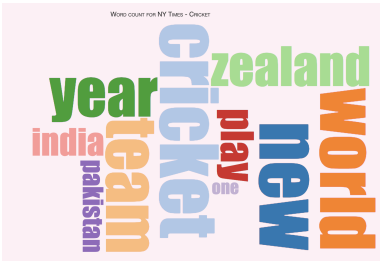


Figure 4: Screenshot of the execution of MR Job

6. Visualization

The left is the word count for small data, center is the word count for large data and the right is for word cooccurance.

Figure 5: NYTimes Sport



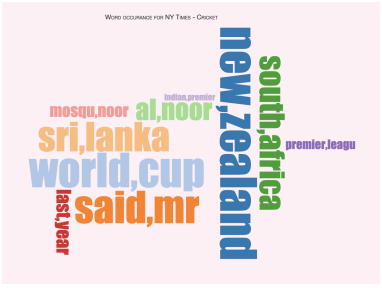
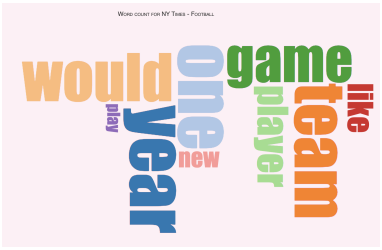


Figure 6: NYTimes Cricket



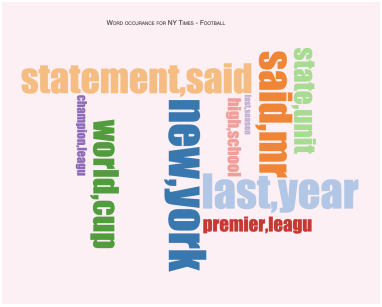


Figure 7: NYTimes Football

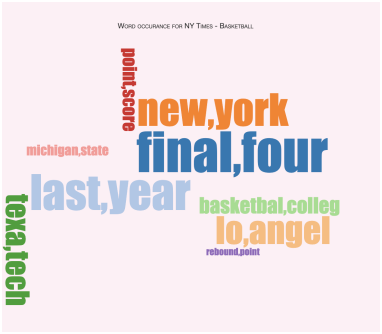
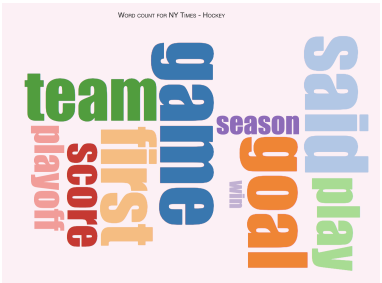
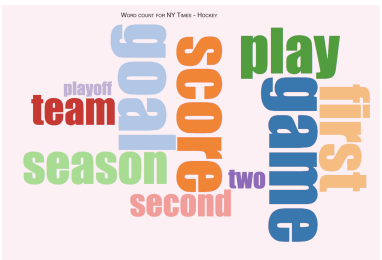


Figure 8: NYTimes Basketball





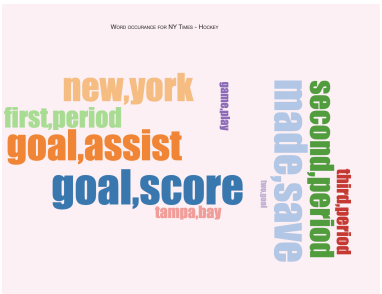


Figure 9: NYTimes Hockey

Figure 10: NYTimes Tennis

Figure 11: Twitter Sport

Figure 12: Twitter Cricket

Figure 13: Twitter Football

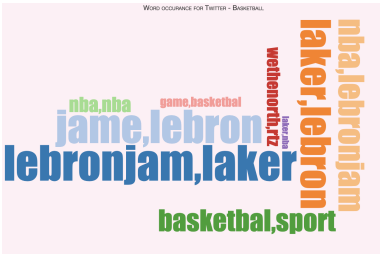
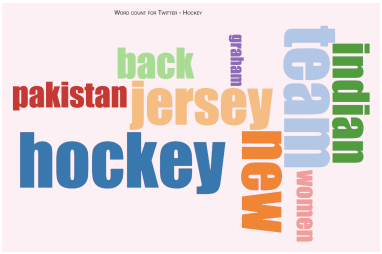


Figure 14: Twitter Basketball



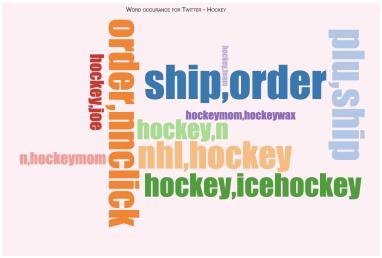
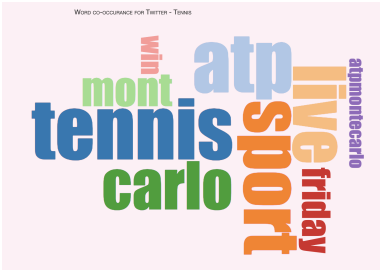
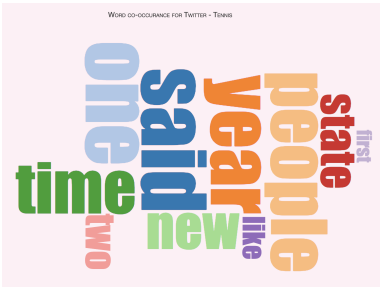


Figure 15: Twitter Hockey





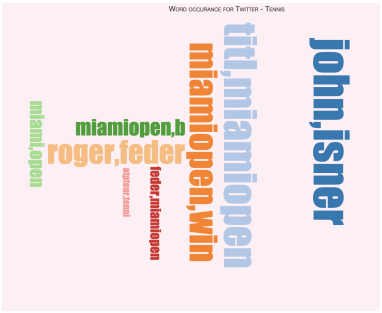
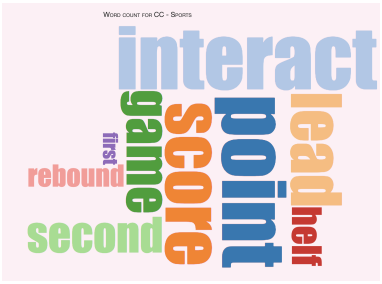


Figure 16: Twitter Tennis



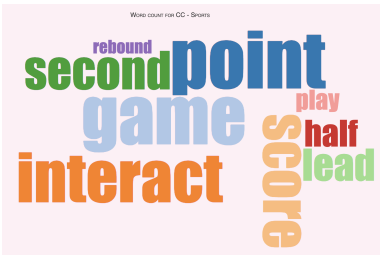


Figure 17: CommonCrawl Sport

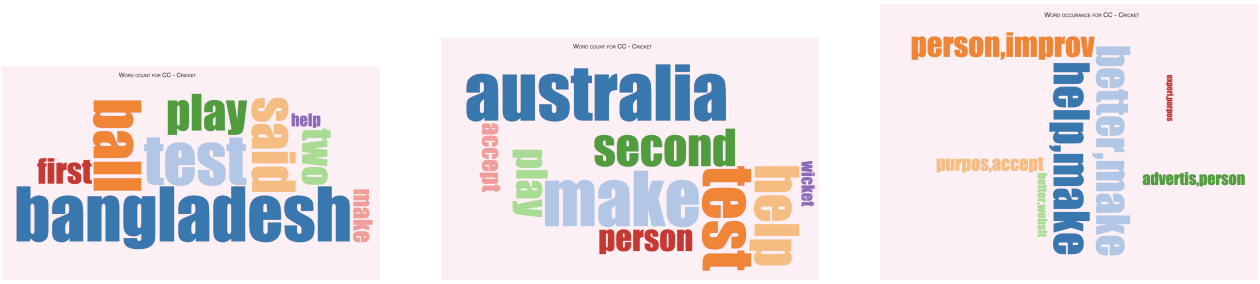


Figure 18: CommonCrawl Cricket

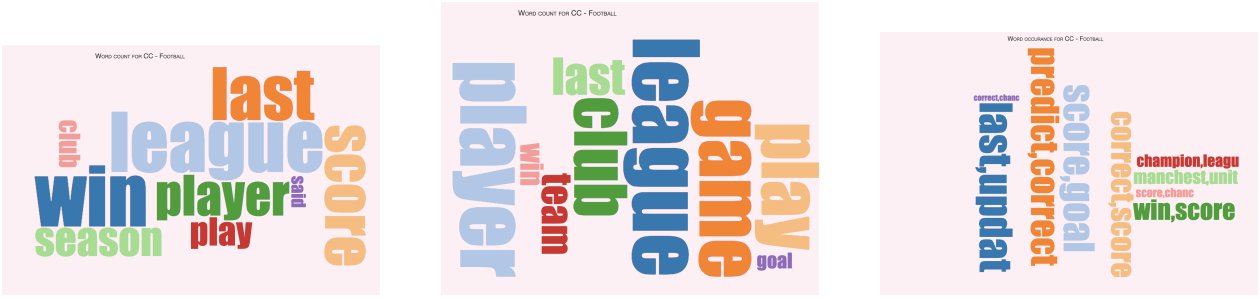


Figure 19: CommonCrawl Football

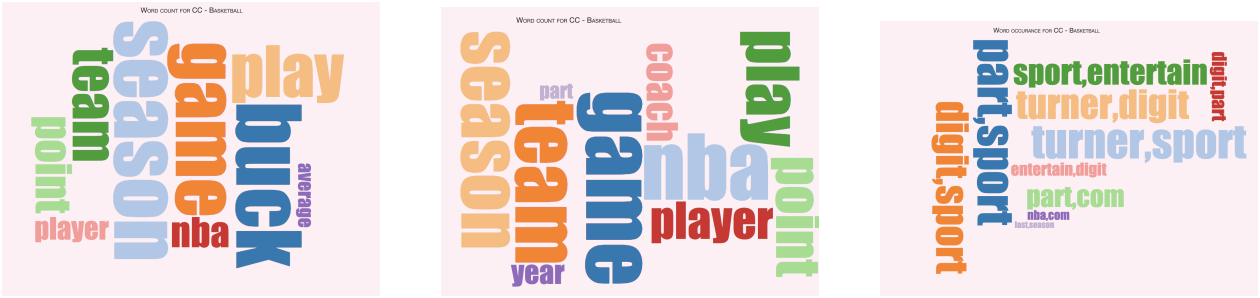


Figure 20: CommonCrawl Basketball

Figure 21: CommonCrawl Hockey

...

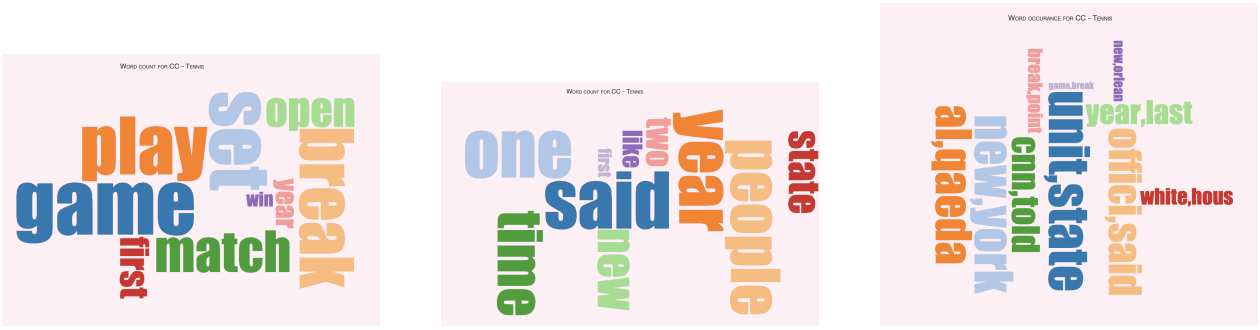


Figure 22: CommonCrawl Tennis

Results

We observe that Twitter in all topics and subsections, gave relatively poor output compared with NYTimes and Common crawl. Topics had the most accurate results, because of the amount of data, and diversification of data on the whole. As subsections are more specific we observe that the word clouds are specific to the subtopic. Also, common crawl requires extensive cleaning as many results were irrevelant.

Appendix

We have generated the Optional LDA and have tabluated the results for each of the topic in large and small data.

```
--------------------------------------------
ny_topic
[(0, '0.017*"s" + 0.015*"sport" + 0.012*"said" + 0.007*"year"'), (1, '0.023*"s" + 0.014*"said" + 0.008*"sport" + 0.00
7*"coach"')]
--------------------------------------------
ny_subtopic1
[(0, '0.022*"s" + 0.007*"cricket" + 0.007*"said" + 0.006*"world"'), (1, '0.019*"said" + 0.017*"s" + 0.012*"new" + 0.0
10*"zealand"')]
--------------------------------------------
ny_subtopic2
[(0, '0.018*"s" + 0.012*"said" + 0.011*"footbal" + 0.007*"year"'), (1, '0.018*"s" + 0.011*"said" + 0.006*"mr" + 0.006
*"footbal"')]
--------------------------------------------
ny_subtopic3
[(0, '0.022*"s" + 0.010*"said" + 0.009*"game" + 0.008*"point"'), (1, '0.020*"s" + 0.012*"said" + 0.008*"basketbal" +
0.007*"team"')]
--------------------------------------------
ny_subtopic4
[(0, '0.021*"game" + 0.017*"goal" + 0.016*"s" + 0.012*"score"'), (1, '0.023*"s" + 0.012*"said" + 0.009*"hockey" + 0.0
08*"team"')]
--------------------------------------------
ny_subtopic5
[(0, '0.021*"s" + 0.012*"6" + 0.012*"said" + 0.009*"match"'), (1, '0.023*"s" + 0.010*"said" + 0.009*"coach" + 0.007
*"colleg"')]
--------------------------------------------
```

Figure 23: LDA data

Directory structure based on this image.

- ***yourUBITLab2.zip***
  - **Report.pdf**
  - **Video.mp4**
  - **part1** (folder)
    - **Code** (folder) -> *all scripts and codes for collecting data*
    - **Data** (folder)
      - Twitter (folder) -> *all tweets that contain your topics*
      - NYT (folder) -> *all NYT articles that contain your topics*
      - Commoncrawl (folder) -> *all CC articles that contain your topics*
  - **part2 (folder)**
    - *mapper.py reducer.py .etc (project directory and dependencies required for running part2)*
    - *other files (screenshots/results/etc)*
  - **part3 (folder)**
    - **Twitter** (folder)
      - Code (folder) -> *all scripts and code files required for processing twitter (mapper.py/reducer.py/etc)*
      - Images (folder) -> *all visualizations .jpg/.png AND .js /.twbx files to create Twitter images*
    - **NYT** (folder)
      - Code (folder) -> *all scripts and code files required for processing NYT (mapper.py/reducer.py/etc)*
      - Images (folder) -> *all visualizations .jpg/.png AND .js /.twbx files to create NYT images*
    - **Commoncrawl** (folder)
      - Code (folder) -> *all scripts and code files required for processing CC (mapper.py/reducer.py/etc)*
      - Images (folder) -> *all visualizations .jpg/.png AND .js /.twbx files to create CC images*
  - **Webpage (folder)**
    - *(For d3.js users) interactive webpage structure showing your results based on topics and data sources*

Figure 24: Directorystrucutre