Siddheswar Chandrasekhar
siddhesw@buffalo.edu

Pratik Appaso Vagyani
pratikap@buffalo.edu

Lab 2

OVERVIEW

In this lab, we expanded our skills in data exploration developed in Lab1 and enhanced them by adding big data analytics and visualization skills. This document describes Lab2: Data Aggregation, Big Data Analysis and Visualization, involves (i) data aggregation from more than one source using the APIs (Application programming interface) exposed by data sources, (ii) Applying classical big data analytic method of MapReduce to the unstructured data collected, (iii) store the data collected on WORM infrastructure Hadoop and (iii) building a visualization data product.

We used HortonWork's **HDP** as hadoop platform. HDP comes with namenode, datanode as well as hadoop ecosystem like ambari, yarn, hdfs. We wrote shell script to run hadoop command to execute mapreduce job on input data and stored data in HDFS using hdfs commands.

GOALS

1. **Source Data** from Twitter, NYT, Commoncrawl.
2. Use **MapReduce** for Word Count, Word Co-occurrence.
3. Data visualization using **Tableau**

Data Collection

We collect data for our topic of Artificial Intelligence and its related sub-topics such as Machine Learning, Self Driving Cars, Deep Learning etc. from Twitter, New York Times and Common Crawler.

Twitter

We use Python's Twython (Twitter REST API) package to gather tweets related to our topic. We gather approximately 22,000 tweets collected over the past few weeks.

NYT

For collecting articles related to our topic, we use the *'nytimesarticle'* Python package. We collect gather approximately 600 articles collected over the past few months.

Common Crawl [2]

Common Crawl provides an API that enables us to query a particular index for a particular domain and get back results that point to the location where the actual HTML content for that snapshot lives. After getting the results from the API query, we try to reach into the compressed archive files stored on Amazon S3 and pull out the actual content.

We write an automated Python script to perform all the tasks mentioned above and gather data from approximately 1000 URLs (location where the HTML content lives) collected over the past few months.

Data Cleaning:

For all the three sources, we perform data cleaning to get rid of unwanted / garbage characters and stop words (from NLTK library) before feeding it to our Mapper. We also removed tags and special characters from html and only keep the ASCII characters in lowercase.

Each map function reads a line as input. For Commoncrawl and NYT, we keep all text from a paragraph into one single line so that we can perform map function on every paragraph.

MapReduce

Word Count

Goal : List all keys in text and their count in text. To achieve this, we are using mapreduce paradigm.

Mapper reads text from input file, divided text into words and emit word as key with count 1 as value. We are ignoring stop words and words with length less than 2.

Reducer receives words key and list of their counts as value. For each key, Reducer sums all count and emit word as key and sum as total count.

Mapper:

```
#!/usr/bin/env python
import sys
```

```
stopWords = set(['each', "hasn't", "weren't", 'had', 'off', 'at',
'ourselves', 'be', 'not', 'does', 'mightn', 'once', 'his', 'yourselves',
'until', 'into', 'hasn', 'against', "won't", 'did', 'before', 'theirs',
'both', 'am', 'there', 'only', "should've", 'we', 'it', 'of', 'by', 'more',
'aren', "you'd", 'and', 'whom', 'further', 't', 'what', 'being', 'herself',
"couldn't", "mustn't", 'all', 'shan', 'with', 'any', 'yourself', 're',
'me', 'from', 'needn', 'to', 'm', 'because', 'yours', "needn't", 'over',
'as', 'have', 'wouldn', 'when', 'myself', "shan't", 'don', 'won',
"wouldn't", 'can', 'their', 'on', 'below', "it's", 'wasn', 'will', 'most',
'should', "hadn't", 'isn', 'who', 'them', 'which', 'having', "she's",
'doing', 'how', 'nor', 'those', 'or', 'been', 'where', 's', 'he', 'him',
'so', 'here', 'its', 'you', 'themselves', "shouldn't", 'mustn', "you're",
'ma', 'some', 'her', 'hers', "mightn't", 'y', 'my', 'while', 'itself',
'this', 'do', 'doesn', 'during', 'above', 'through', 'out', 'ours', 'up',
'i', 'than', 'that', 'are', "you'll", 'under', 'has', "wasn't", 'your',
'such', "don't", 'they', 'shouldn', 'was', 'll', 'o', 'between', 'same',
'the', 'why', 'other', 'ain', 'these', 'then', 'in', 'if', 'again', 'just',
'couldn', "haven't", 'a', 'but', 'd', 'very', 'no', "you've", 'about',
'hadn', 'our', 'she', 'too', 'few', 'now', 'for', 'haven', "isn't", 'is',
'weren', 'own', 'an', 'were', 'down', "aren't", "didn't", 'after',
'himself', "that'll", 've', "doesn't", 'didn'])

#--- get all lines from stdin ---

for line in sys.stdin:
    #--- remove leading and trailing whitespace---
    line = line.strip()
    words = line.split()

    #--- output tuples [word, 1] in tab-delimited format---
    for word in words:
        # ignore words wih length 0,1 or stop words.
        if len(word)>1 and word not in stopWords:
            print '%s\t%s' % (word, "1")
```

Reducer

```
#!/usr/bin/env python
import sys
```

```
# maps words to their counts
word2count = {}

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()

    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue
    try:
        word2count[word] = word2count[word]+count
    except Exception as e:
        word2count[word] = count

for word in word2count.keys():
    print '%s\t%s'% ( word, word2count[word] )
```

Top N

Goal : List get N words with highest count. To achieve this, we are using output of **WordCount**.

Mappers are emitting words as key & count as value from result of word count.

Reducer creates dictionary with words as key and count as value. Dictionary gets sorted based on value. Reducer emits top N from sorted dictionary keys & values.

Mapper:

```
#!/usr/bin/env python
import sys

# --- get all lines from stdin ---
for line in sys.stdin:
    line = line.strip()
    # print records from word count
    print line
```

Reducer:

```
#!/usr/bin/env python
import sys
import operator
# maps words to their counts
word2count = {}

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue
    try:
        word2count[word] = count
    except:
        pass

# sort dictionary based on values
sorted_d     =     sorted(word2count.items(),     key=operator.itemgetter(0),
reverse=True)
#print top 10 set of words only
for key in sorted_d[:10]:
    print '%s\t%s' % (key[0], key[1])
```

Word Co-occurrence

Goal : List pair of words and how many times they present in input.

Mapper receives list of top N words as arguments & each map function receives tweet or paragraph as input. Mapper creates dictionary of words and mark all keys present in given text. Mapper then creates pair of present dictionary keys & emit them with 1 as count.

Reducer receives word pair as key and list of count as value. Reducer sum all count for each word pair and then emit word pair as key & sum as value.

Mapper:

```
#!/usr/bin/env python
import sys
tw = sys.argv[1].split(",")
top_words = {word:0 for word in tw}


#--- get all lines from stdin ---
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    t_words = dict(top_words)
    li = []
    for word in words:
        if word in t_words:
            t_words[word]=1
    for word in tw:
      if t_words[word]==1:
            li.append(word)

    # make pair of words and emit words
    for i in xrange(len(li)):
        for j in xrange(i+1,len(li)):
            print '%s\t%s' % (str(li[i])+","+str(li[j]), "1")
```

Reducer:

```
#!/usr/bin/env python
import sys

# maps words to their counts
word2count = {}

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
```

```
    except ValueError:
        continue

    try:
        word2count[word] = word2count[word]+count
    except:
        word2count[word] = count

for word in word2count.keys():
    print '%s\t%s'% ( word, word2count[word] )
```

Top N Word Co-occurrence

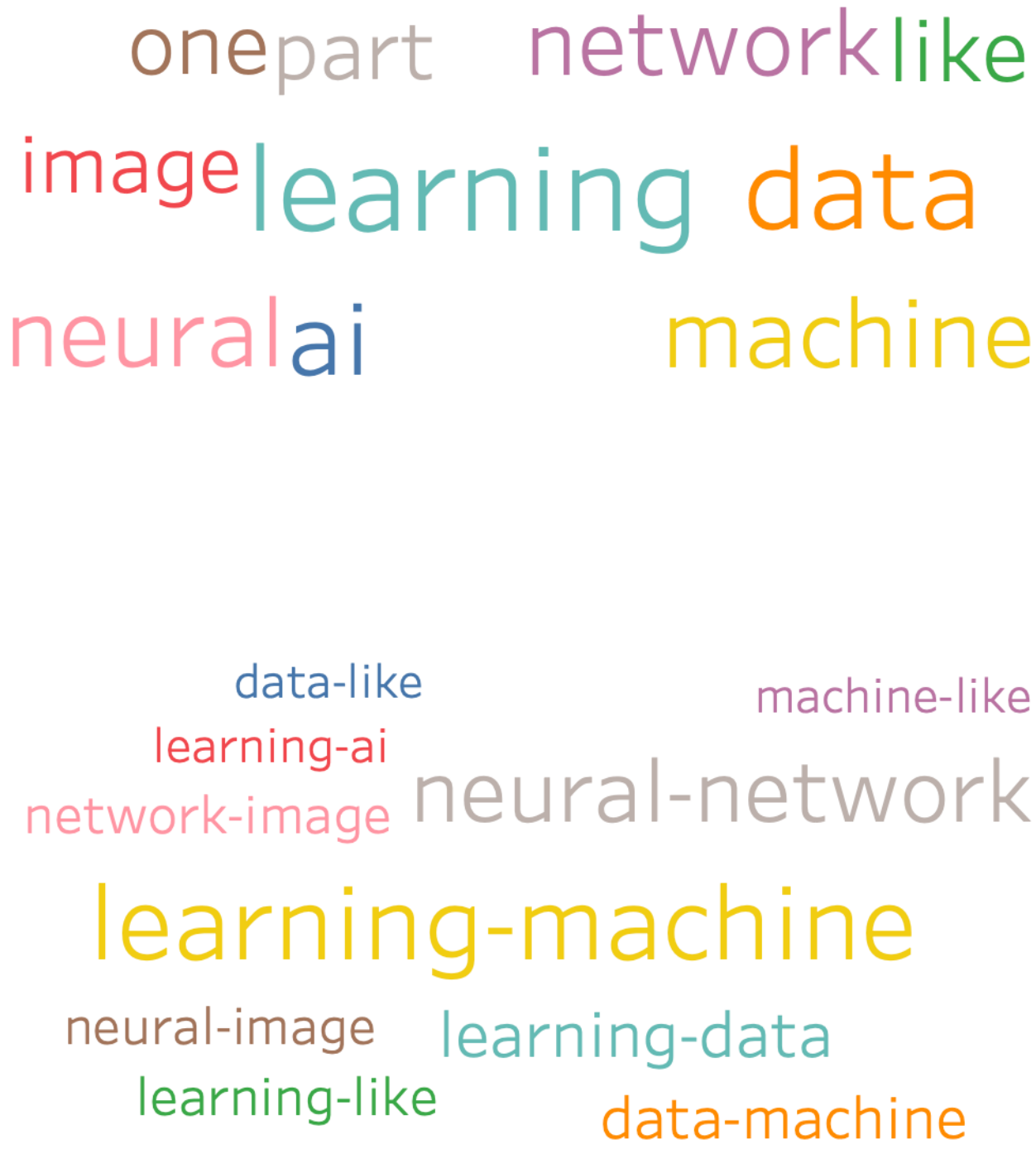
Goal : List top N word pairs

Same logic as TopN is used on output of **Word Co-occurrence.**

Data visualization

We use Tableau, a powerful and fast data visualization tool used in Bussiness Intelligence. We visualize our data for the TopN words and TopN word co-occurrences for twitter, NYT & Common Crawl using ‘wordclouds’.

We import the files generated by our Reducers as CSV files and create wordcloud for each individual data. Additionally, we also create a Dashboard to showcase all of our wordclouds do demonstrate data gathered from different sources for our analysis.


onepart
network
like
image
learning
data
neural
ai
machine
data-like
machine-like
learning-ai
neural-network
network-image
learning-machine
neural-image
learning-data
learning-like
data-machine

Reference:

1. https://hortonworks.com/products/data-platforms/hdp/
2. https://www.bellingcat.com/resources/2015/08/13/using-python-to-mine-common-crawl/
3. https://onlinehelp.tableau.com/current/pro/desktop/en-us/dashboards_sheet_selector.htm