SENTIMENT ANALYSIS OF TWITTER DATA: A *REPORT* ON BIG DATA PROJECT

INTRODUCTION TO SENTIMENT ANALYSIS:

Opinion mining (sometimes known as sentiment analysis or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.

Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. It's also known as **opinion mining**, deriving the opinion or attitude of a speaker.

Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

BROAD OVERVIEW OF THE PROJECT:

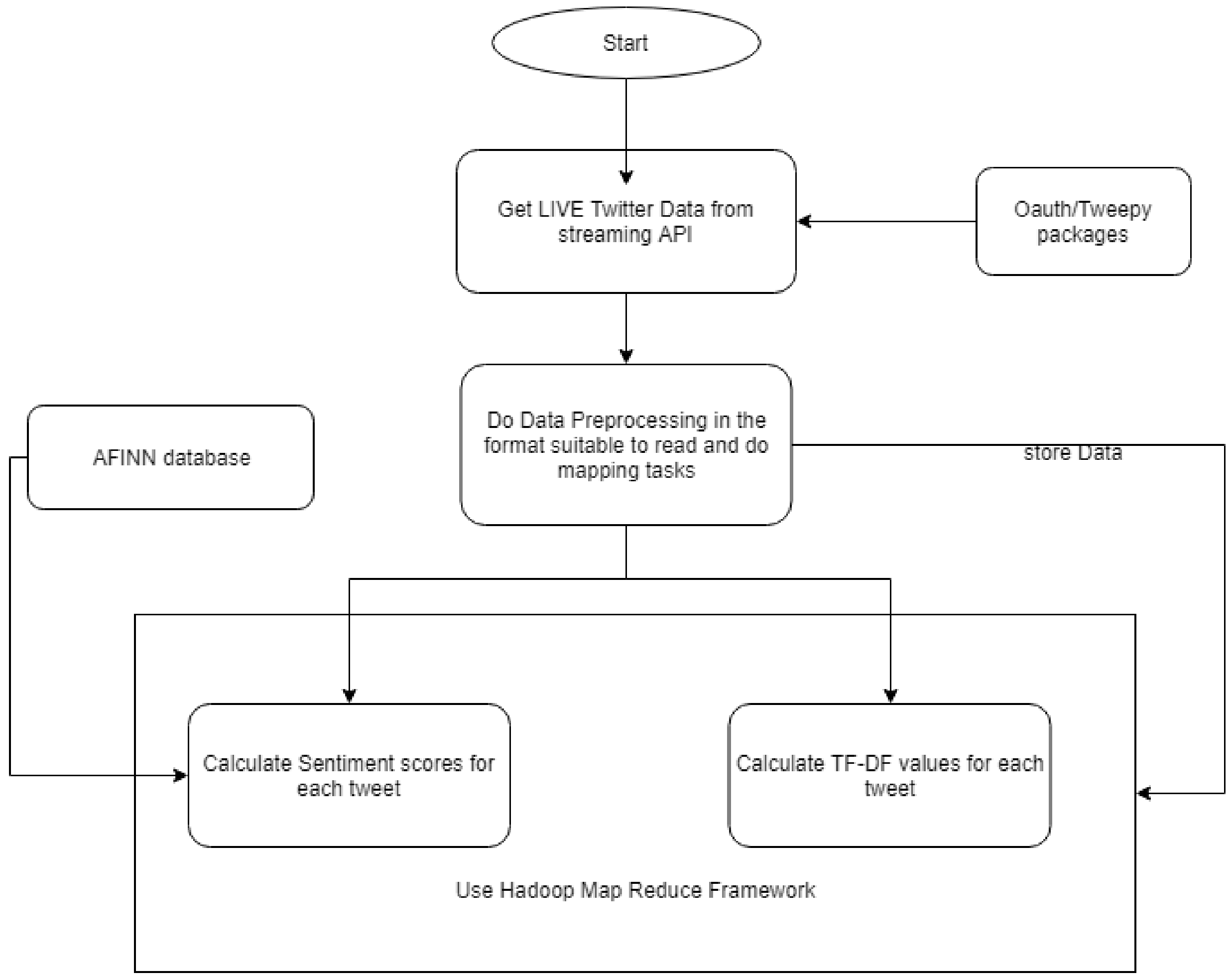
1. Access the Twitter Application Programming Interface(API) using python.
2. Estimate the public's perception (the sentiment) of the tweet's.
3. Calculate the TF-DF(term frequency-document frequency) of terms in a tweet

The Twitter Application Programming Interface:
Twitter provides a very rich REST API for querying the system, accessing data, and control your account. You can read more about the Twitter API here (https://dev.twitter.com/docs).

Getting Twitter Data:

To access the live stream of twitter data

1. To access the live stream, the oauth2 library is used so we can properly authenticate. We can install it ourself in the Python environment.
   a. `$ pip install oauth2`
   b. `easy-install oauth2`


Start
Get LIVE Twitter Data from streaming API
Oauth/Tweepy packages
AFINN database
Do Data Preprocessing in the format suitable to read and do mapping tasks
store Data
Calculate Sentiment scores for each tweet
Calculate TF-DF values for each tweet
Use Hadoop Map Reduce Framework

2. Create an Access Token. You can Read more about Oauth authorization. The parts of an access token are:

1. "API Key(Consumer Key)“
2. "API secret(Consumer Secret)“
3. "Access token”
4. "Access token secret”.

```
import oauth2 as oauth
import urllib.request as urllib

response = twitterreq(url, "GET", parameters)
      for line in response:
            print( line.strip() )
```

The output is then stored using;

```
with open('twitDB.json', 'a') as fp:
            json.dump(datajson, fp)
```

Or `python twitterstream.py > twitDB.txt`

Unicode strings:

Unicode strings Strings in the twitter data prefixed with the letter "u" are unicode strings. For example: `u"This is a string"`

Unicode is a standard for representing a much larger variety of characters beyond the roman alphabet (greek, russian, mathematical symbols, logograms from non-phonetic writing systems such as kanji, etc.)

In most circumstances, you will be able to use a unicode object just like a string. If you encounter an error involving printing unicode, you can use the `encode` method to properly print the international characters, like this:

```
unicode_string = u"aaaÃ Ã§Ã§Ã§Ã±Ã±Ã±"
encoded_string = unicode_string.encode('utf-8')
print encoded_string
```

Python MapReduce Framework: (My Work)

A python library called MapReduce.py that implements the MapReduce programming model. The framework faithfully implements the MapReduce programming model, but it executes entirely on a single machine -- it does not involve parallel computation.

```
import json

class MapReduce:
    def __init__(self):
        # initialize dictionary for intermediate values from Map task
        self.intermediate = {}
        # initialize list for results of Reduce task
        self.result = []
```

```
def execute(self, data, mapper, reducer):
        # read each line from input file; call Map function on each record
        for line in data:
            record = json.loads(line)
            mapper(record)

        # for each key:valuelist in intermediate dictionary, call Reduce
task
        for key in self.intermediate:
            reducer(key, self.intermediate[key])

        jenc = json.JSONEncoder()
        self.result.sort()
        # print each result in result list
        for item in self.result:
            print jenc.encode(item)
```

Steps in any mapReduce program:

- In Part 1, we create a MapReduce object that is used to pass data between the map function and the reduce function; you won't need to use this object directly.
- In Part 2, the mapper function tokenizes each document and emits a set of key-value pairs. The key is a word formatted as a string and the value is the integer 1 to indicate an occurrence of word.
- In Part 3, the reducer function sums up the list of occurrence counts and emits a count for word. Since the mapper function emits the integer 1
- In Part 4, the code loads the JSON file and executes the MapReduce program which prints the result to stdout.

Problem 1: Derive Tweet Sentiment Score (My Work + 1 teammate)

About AFINN:

AFINN is a list of English words rated for valence with an integer between minus five (negative) and plus five (positive). { -5,+5 }
The words have been manually labeled by Finn Årup Nielsen in 2009-2011. The file is

tab-separated.

AFINN-111: is the Newest version with 2477 words and phrases.

To use the data in the AFINN-111.txt file, it is useful to build a dictionary.
Note that the AFINN-111.txt file format is tab delimited, meaning that the term and the score are separated by a tab character.

```
import sys
import string
import re
import MapReduce

mr = MapReduce.MapReduce()
scores = {}
count=1

if __name__ == '__main__':
    afinnfile = open("AFINN-111.txt")
    for line in afinnfile:
        term, score = line.split("\t")  # The file is tab-delimited. #\t
means the tab character.
        scores[term] = int(score)  # Convert the score to an integer.
    #tweet_data = open(sys.argv[2])
    tweet_data = open("output.txt")
    mr.execute(tweet_data, mapper, reducer)
```

My Task:

1. Write the `mapper()` and the `reducer()` functions which calculates the Sentiment score of each tweet present in twitDB.txt.
2. Each line of twitDB.txt is a streaming message and is represented as a JSON object, which stands for JavaScript Object Notation. It is a simple format for representing nested structures of data. It is straightforward to convert this JSON object into a Python data structure using a library called `json`.
3. As we can see below, there is a call `mr.execute(tweet_data, mapper, reducer)` which makes use of the mapreduce framework.
4. Looking at the MapReduce.py, we will see that it imports the json library and calls json.loads() on each tweet line. It then calls the mapper() and reducer() function for each tweet.

5. We have to implement the logic for this mapper() and reducer() function in your tweets_sentiment.py file.

Mapper Function:

- The input is a python dictionary containing the tweets data: each_tweet. Extract the 'text' from this tweet which is now stored in a dictionary. For further reading, see the Twitter documentation (https://dev.twitter.com/docs/platform-objects/tweets) to understand what information each tweet contains and how to access it, but it's not too difficult to deduce the structure by direct inspection.

- Now that the 'text' for each tweet is available, use that to calculate the sentiment score of each tweet. For that you have to lookup each term of the tweet in the AFINN-111.txt dictionary and add up their respective sentiment scores.

- Keep in mind that all the terms in AFINN-111.txt are lower case. You will have to use the string.lower() function to convert your tweet to all lower case for this problem.

- To encode the tweet string in utf-8 and remove ASCII punctuation, import the string library and use the following code:
  `newstring = mystring.encode('utf-8').translate(None, string.punctuation)`
  Some tweets contain URL's (eg: https:\/\/t.co\/8wl9wrT), hashtags(starting with #,@) and prefix like "RT @ManUtd:", which should be ignored while calculating the sentiment score.

- We made use of python regular expression library (`import re`) for the same. Also the extra terms found in the tweets but not in `AFINN-111.txt` and should be given a sentiment score of 0.

```
def mapper(each_tweet):
    global count
    for key in each_tweet:
        if key=="text":
            for word in each_tweet[key].split(" "):
                if "#" in word:
                    word=word.split("#")[0]
                url =
re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F
][0-9a-fA-F]))')
                retw=re.compile("(?<!RT\s)@\S+")

                if url.match(word) or word.startswith('#') or
retw.match(word) or word=="RT" or word.startswith('@'):
                    continue
                word=word.lower()

word=word.encode('UTF-8').translate(None,string.punctuation)

                if word in scores:
                    mr.emit_intermediate(count,scores[word])
                else:
                    mr.emit_intermediate(count,0)
    count=count+1
```

Reducer Function:

The script prints to stdout the sentiment of each tweet in the file, one numeric sentiment score per line.
The first score corresponds to the first tweet, the second score corresponds to the second tweet, and so on.

```
def reducer(key, list_of_values):
    tot=0
    for value in list_of_values:
        tot=tot+value
    mr.emit((key,float(tot)))
```

This outputs a .txt file as follows:

```
[1, 0.0]  #<tweet number, sentiment score>
[2, 0.0]
[3, 0.0]
[4, 0.0]
[5, 1.0]
[6, 0.0]
```

```
[7, -3.0]
[8, 0.0]
[9, -1.0]
[10, 0.0]
[11, 0.0]
[12, 2.0]
[13, 0.0]
[14, 2.0]
[15, 0.0]
[16, -3.0]
[17, 0.0]
[18, 0.0]
[19, 0.0]
[20, 0.0]
...
```

These are the sentiment scores for the first 20 tweets in the database

The scores range from -5 to +5 to depict the sentiment from most negative to most positive

Here, we can analyse that many tweets are neutral because the computer cannot understand sarcasm unless we explicitly define rules for that.

Problem 2: TF-DF of each term(term frequency – document frequency)

1. ***Background***: Given a document, tf (term frequency) of a term in the document is the number of occurrences of the term in the document.
2. For example, consider the following short document D:

In a swanky hotel ballroom, he told the crowd of alumni and donors that Texas had become the largest feeder of students to USC after California, and that students from their state scored significantly higher on the SAT than the average of all applicants.

3. The term "students" occurs twice in the documents, so its tf is 2 for this document.
4. Moreover, given a collection of documents, df (document frequency)of a term refers to the number of documents in the collection where the term occurs. For example,

suppose that there are 10 documents in the collection and that “students” occur in three of them, e.g., one of which may be the document D above. Then the document frequency of “students” is 3.

5. tf and df of a term indicate the importance of the term in the document and across the documents. They are important statistics for effective information retrieval. Note that df can be used to compute inverse document frequency or IDF

The input is a python dictionary containing the tweets data: `each_tweet`.

The output should be a list of `<term, df, a list of <tweet_no,tf>>` for each term in one line. Note that the framework turns your output tuples into JSON arrays

Conclusion:

- Accessed the Twitter Application Programming Interface(API) using python.
- Estimated the public's perception (the sentiment) of the tweet’s.
- Calculated the TF-DF(term frequency-document frequency) of terms in a tweet.