Project Report of WQD7007 Big Data Management

SENTIMENT ANALYSIS WITH HDFS, HIVE, PIG, HBASE, SPARK & POWER BI

by

WQD180050 - Cheng Jiechao
WQD180024 - Teng Lung Yun
WQD180063 - Yong Wai Shin
WQD180055 - Nai Siu Hong
WQD180076 - Lim Kaomin
WQD180053 - Choy Siew Wearn
WQD180117 - Tan Chee Siang
WQD180054 - Wo Choy Yun

Abstract

This report provides a sentiment analysis project by using Big Data Tools with Twitter dataset describing how to use HDFS, Hive and Spark etc. to analyse the sentiment of the unstructured text dataset. It comprises brief introduction of sentiment analysis, data preparation, how to load data to HDFS, how to create Hive table, Pig table and HBase table, how to load data to these tables, how to analyse sentiment with Spark and BI, the results of sentiment analysis, as well as a final conclusion. The Logistic Regression model with CountVectorizer tokenization method gains the best result for sentiment analysis. Hive table is more efficient than Pig and HBase to upload external structured data.

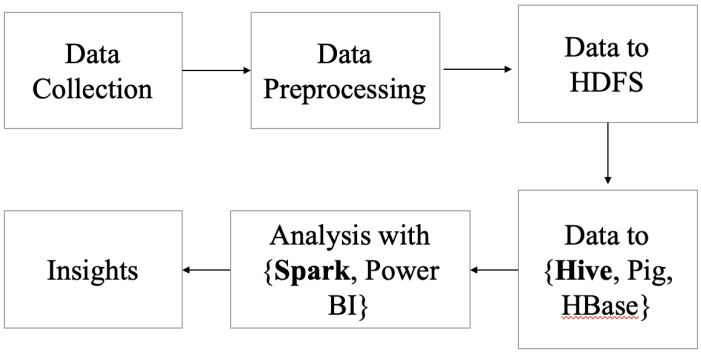
Contents

1 INTRODUCTION

1.1 Background of the Project

Sentiment analysis is the process of mining opinions from various data sources by using text mining technology. So many sentiment analysis cases use dataset coming from Internet social media platform, like Twitter, Facebook etc. Companies and organizations can know customers' opinions about products and services by using sentiment analysis.

In this project, we consider utilizing Big Data tools like Hadoop, Hive, Pig, HBase, Spark etc. to store and process the Twitter data for sentiment analysis. The procedures of our project are depicted as follows. The objective of this project is to make prediction on whether a tweet is negative or positive using text mining method. TF-IDF, 1-Gram TF-IDF, 2-Grams TF-IDF and a Count Vectorizer-IDF are being used to process to text data and then model will be trained and tested using Logistic Regression. Performance will be compared to determine which method is the best fit to process the tweet data set.



Procedures of the Project

1.2 Problem Statement

The dataset for sentiment analysis in our project is unstructured and coming from multiple sources with a big volume, thus we need some advanced tools can solve this problem. Furthermore, how to pre-process the unstructured text data to structured data is a challenge. Additionally, how to upload data from local directory to HDFS, then to Hive, Pig and HBase table are a set of difficult tasks filled with uncertainty. Finally, conducting sentiment analysis by using Big Data Tools with tweets text is a novel and interesting but challenging job.

Group Aggregation

Cheng Jiechao	Choy Siew Wearn	Wo Choy Yun	Tan Chee Siang	Yong Wai Shin	Teng Lung Yun	Nai Siu Hong	Lim Kaomin
WQD180050	WQD180053	WQD180054	WQD180117	WQD180063	WQD180024	WQD180055	WQD180076
Spark & Dataset	BI & R	BI & R	Hive	HBase/HDFS	HBase/HDFS	Spark	Pig

2 DATASET

2.1 Data Collection

The dataset used for this project is Sentiment140 Twitter data with 1.6 million tweets from Kaggle. Our Twitter dataset are a range of topics of short messages posted by tens of thousands of Twitter users. And the provided tweets have limitation of 140 characters and comes with emoticons, usernames and hashtags. There are totally six columns of our raw Twitter dataset: sentiment, id, date, query_string, user and text. The sentiment column has two types of values: 0 and 4 (0 means negative, 4 refers to positive). For the simplicity of our sentiment labels, we decide to map all '4' labels to '1'. Thus the records of label 0 and label 1 are both 800,000. The below figure is the sample of the raw dataset.

	sentiment	id	date	query_string	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

2.2 Data Preprocessing

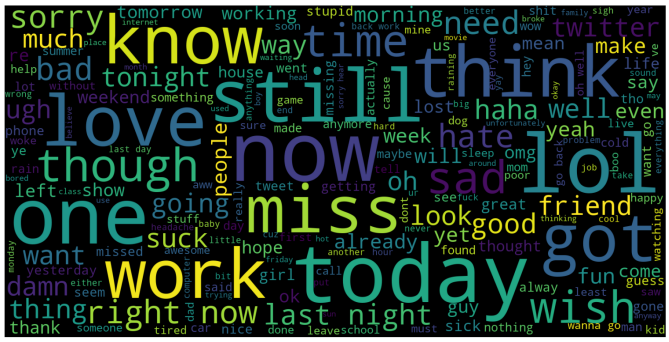
Before we use data for sentiment analysis, the data preprocessing needs to be completed. First we remove the HTML encoding such as '&amp', '&quot' etc. Then we remove the @mention and URL links. Next we transform UTF-8 BOM bytes to be unrecognizable special characters. We also remove numbers. Only 'sentiment' and 'text' columns are retained in our project for sentiment analysis. At last, all text is converted to lowercase. The sample of cleaned Twitter data can be viewed as below.

text	target
awww that bummer you shoulda got david carr of...	0
is upset that he can not update his facebook b...	0
dived many times for the ball managed to save ...	0
my whole body feels itchy and like its on fire	0
no it not behaving at all mad why am here beca...	0

2.3 Data Exploring

Textual analysis can provide a general idea of what kind of words are frequent in the corpus. In this part, we use a word cloud tool with python library wordcloud to identify the frequent words of positive and negative polarities for sentiment analysis. A word cloud refers to the word frequency in a corpus by rescaling every single word proportionally with its frequency.

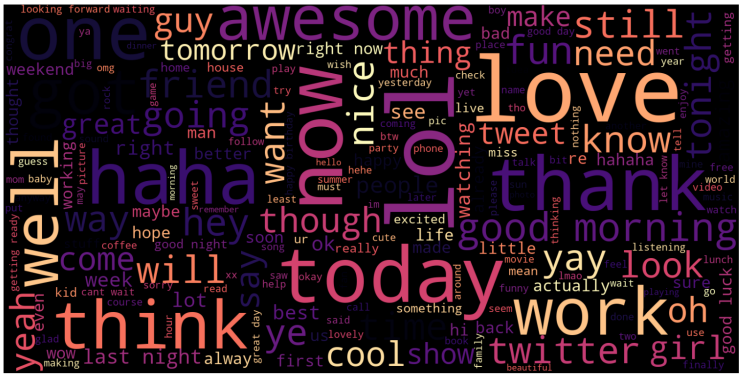
The right figure is a word cloud of the corpus with negative sentiment labels. Some big words such as “today” can be interpreted as neutral words. However, some words in small size have close relationships with negative corpus, such as “suck”, “miss”, “sorry”.



The word cloud of positive tweets is shown as below. Some neutral words such as “today” and “now” are still much frequent. However, “love”, “lol” and “awesome” also have big size. The size of “work” delivers a message that some people might be positive to their works.

3 LOAD DATA TO HDFS

There is a clean tweet text data obtained from preprocessing step located in the local directory, we want to load local data to HDFS. So we first run the following commands on the terminal console to create a HDFS directory to store data.




```
student@student-VirtualBox:~$ hadoop fs -mkdir /home/
student@student-VirtualBox:~$ hadoop fs -mkdir /home/stephen
student@student-VirtualBox:~$ hadoop fs -mkdir /home/stephen/project
student@student-VirtualBox:~$ hadoop fs -mkdir /home/stephen/project/dataset
```

Next we put the local data file to HDFS using below command and check whether file is available in HDFS or not using below command:

```
student@student-VirtualBox:~/Desktop$ hdfs dfs -put /home/student/Desktop/clean_tweet.csv /home/stephen/project/dataset
student@student-VirtualBox:~/Desktop$ hdfs dfs -ls /home/stephen/project/dataset
Found 1 items
-rw-r--r--   1 student supergroup  112207794 2019-05-24 00:34 /home/stephen/project/dataset/clean_tweet.csv
```

The text from *clean_tweet.csv* is tokenized into words and word counts are computed using *wordcount* in *mapreduce.*

```
student@student-VirtualBox:~/Desktop$ hadoop jar /home/WQD7007/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examp
les-2.7.7.jar wordcount /home/stephen/project/dataset/clean_tweet.csv /user/labtest/mapreduce/out-wordcount
```

The top 300 words with highest frequency counts is saved and visualized in the word cloud below using R.

```
student@student-VirtualBox:~$ hadoop fs -cat /user/labtest/mapreduce/out-wordcount
/part-r-00000 | sort -n -k2 -r | head -n 300 > top300.txt
```

```
student@student-VirtualBox:~$ cat wordcloud.r
library(wordcloud)
library(RColorBrewer)

df <- read.csv("top300.txt", sep="\t", header=F)

set.seed(1234)
wordcloud(words = df$V1, freq = df$V2, min.freq = 1,
          max.words=300, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```

4 LOAD DATA TO HIVE

4.1 Creating Hive Table & Load Data

Now we load file in HDFS, we just need to create an external table on top of it. We can use following Hive scripts to create a Hive table clean_tweet in database project and load HDFS data to Hive with skipping first head line. We run the script in the right picture on Hive Command-Line Interface (CLI).

```
hive> select * from project.clean_tweet order by line_number desc limit 10;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a d
ifferent execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = stephen_20190410142425_bf9b7db0-82f1-4bdb-97bc-bc2b5eabe4b4
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1554834213562_0001, Tracking URL = http://ste:8088/proxy/application_1554834213562_0001/
Kill Command = /home/ste/hadoop/bin/hadoop job  -kill job_1554834213562_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-04-10 14:24:39,501 Stage-1 map = 0%,  reduce = 0%
2019-04-10 14:24:49,965 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.85 sec
2019-04-10 14:24:56,622 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.91 sec
MapReduce Total cumulative CPU time: 4 seconds 910 msec
Ended Job = job_1554834213562_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.91 sec   HDFS Read: 112215458 HDFS Write: 821 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 910 msec
OK
1599999 happy charitytuesday    1
1599998 happy th birthday to my boo of alll time tupac amaru shakur     1
1599997 are you ready for your mojo makeover ask me for details 1
1599996 thewdb com very cool to hear old walt interviews        1
1599995 just woke up having no school is the best feeling ever  1
1599994 yeah that does work better than just waiting for it in the end just wonder if have time to keep up good
 blog   1
1599993         1
1599992 recovering from the long weekend        1
1599991 mmm that sounds absolutely perfect but my schedule is full will not have time to lay in bed until sund
```

4.2 Data Checking in Hive

Once we create a Hive table and upload the data, we need check whether the data is showing in a table or not using the above command. Given that we want to verify the last few rows of the data, the Hive calls MapReduce to process massive data, which is more than a million instances.

4.3 Creating a Hive ORC Table

The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data.

ORC is the most used file format when it comes to minimizing the data storage cost. Thus after we created a Hive table, we decide to create a ORC-format Hive table. The right snapshot is the code of creation of Hive ORC table tweet_orc in a Hive database project.

```
hive> create table project.tweet_orc
    > (
    > line_number int,
    > text string,
    > label int
    > )
    > stored as orc;
OK
Time taken: 8.514 seconds
```

Then we insert data into ORC table by copying from the Hive table clean_tweet.

```
hive> insert into table project.tweet_orc select * from project.clean_tweet;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Con
sider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = stephen_20190410143546_bc5bc680-6330-4c32-a886-f5bcac44a9ed
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1554834213562_0002, Tracking URL = http://ste:8088/proxy/application_15548342
13562_0002/
Kill Command = /home/ste/hadoop/bin/hadoop job  -kill job_1554834213562_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2019-04-10 14:35:58,908 Stage-1 map = 0%,  reduce = 0%
2019-04-10 14:36:11,398 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.45 sec
MapReduce Total cumulative CPU time: 5 seconds 450 msec
Ended Job = job_1554834213562_0002
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/project.db/tweet_orc/.hive-st
aging_hive_2019-04-10_14-35-46_857_1157494686711524248-1/-ext-10000
Loading data to table project.tweet_orc
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 5.45 sec   HDFS Read: 112212069 HDFS Write: 44199331 SUC
CESS
Total MapReduce CPU Time Spent: 5 seconds 450 msec
OK
Time taken: 26.463 seconds
```

4.4 Checking Data in Hive ORC Table

In the Hive prompt we run the below code to verify the data in Hive ORC table, we just check and output the first six lines of table.

```
hive> select * from project.tweet_orc limit 6;
OK
NULL    text    NULL
0       awww that bummer you shoulda got david carr of third day to do it       0
1       is upset that he can not update his facebook by texting it and might cry as result schoo
l today also blah       0
2       dived many times for the ball managed to save the rest go out of bounds 0
3       my whole body feels itchy and like its on fire  0
4       no it not behaving at all mad why am here because can not see you all over there       0
Time taken: 0.342 seconds, Fetched: 6 row(s)
```

5 LOAD DATA INTO PIG

5.1 Start the Apache Pig

The Apache Pig can be started by running the command 'pig –x mapreduce'. What it does here is that it enters the grunt shell in MapReduce mode. MapReduce mode is where we load or process our dataset stored in the HDFS and stores back the results in HDFS.

```
student@student-VirtualBox:~$ pig -x mapreduce
19/05/17 14:03:40 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
19/05/17 14:03:40 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
19/05/17 14:03:40 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2019-05-17 14:03:41,207 [main] INFO  org.apache.pig.Main - Apache Pig version 0.
16.0 (r1746530) compiled Jun 01 2016, 23:10:49
2019-05-17 14:03:41,207 [main] INFO  org.apache.pig.Main - Logging error message
s to: /home/student/pig_1558073021194.log
2019-05-17 14:03:41,442 [main] INFO  org.apache.pig.impl.util.Utils - Default bo
otup file /home/student/.pigbootup not found
2019-05-17 14:03:45,032 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2019-05-17 14:03:45,040 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-05-17 14:03:45,041 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.HExecutionEngine - Connecting to hadoop file system at: hdfs://localhost:9000
2019-05-17 14:03:48,024 [main] INFO  org.apache.pig.PigServer - Pig Script ID fo
r the session: PIG-default-af38dccc-1272-4137-8a03-1ab99c102f1a
2019-05-17 14:03:48,026 [main] WARN  org.apache.pig.PigServer - ATS is disabled
since yarn.timeline-service.enabled set to false
grunt>
```

5.2 Load the Dataset into Pig

Here we load the dataset from the HDFS path using 'LOAD' command. We used the 'PigStorage()' function to load and store the data as text file. We also define the attributes as 'line_number:int','text:chararray" and 'target:int'.

```
grunt> twitter = LOAD '/user/hdfs/twitter_sentiment'
>> USING PigStorage(',')
>> as (line_number:int,text:chararray,label:int);
2019-05-17 14:07:09,969 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
```

5.3 Show the Dataset

We can view the dataset using the 'Dump' command.

```
grunt> Dump twitter;
```

```
(,text,)
(1,is upset that he can not update his facebook by texting it and might cry as r
esult school today also blah,0)
(2,dived many times for the ball managed to save the rest go out of bounds,0)
(3,my whole body feels itchy and like its on fire,0)
(4,no it not behaving at all mad why am here because can not see you all over th
ere,0)
(5,not the whole crew,0)
(6,need hug,0)
(7,hey long time no see yes rains bit only bit lol fine thanks how you,0)
(8,nope they did not have it,0)
(9,que me muera,0)
(10,spring break in plain city it snowing,0)
(11,just re pierced my ears,0)
(12,could not bear to watch it and thought the ua loss was embarrassing,0)
(13,it it counts idk why did either you never talk to me anymore,0)
(14,would ve been the first but did not have gun not really though zac snyder ju
st douchecloun,0)
(15,wish got to watch it with you miss you and how was the premiere,0)
(16,hollis death scene will hurt me severely to watch on film wry is directors c
ut not out now,0)
(17,about to file taxes,0)
(18,ahh ive always wanted to see rent love the soundtrack,0)
```

6 LOAD DATA INTO HBASE

6.1 Load Data into HBase

1) Load clean_tweet.csv into HDFS use the below commands:

```
hdfs dfs -mkdir -p /user/hdfs/tweet
hdfs dfs -put /home/student/Downloads/clean_tweet.csv /user/hdfs/tweet
```

2) Check whether the file is in the folder:

```
student@student-VirtualBox:~$ hdfs dfs -ls /user/hdfs
Found 3 items
-rw-r--r--   1 student supergroup    6398990 2019-04-30 00:14 /user/hdfs/batting.csv
-rw-r--r--   1 student supergroup        235 2019-05-07 15:19 /user/hdfs/student_details.txt
drwxr-xr-x   - student supergroup          0 2019-05-24 14:46 /user/hdfs/tweet
student@student-VirtualBox:~$ hdfs dfs -ls /user/hdfs/tweet
Found 1 items
-rw-r--r--   1 student supergroup  112207794 2019-05-24 14:46 /user/hdfs/tweet/clean_tweet.csv
```

3) Load the file from HDFS into HBase:

```
student@student-VirtualBox:~$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator=',' -Dimporttsv.columns='HBASE_ROW_KEY,data:line_num,data:
text,data:label' tweet_data /user/hdfs/tweet/clean_tweet.csv
```

Below shows the process of loading the csv into HBase:

```
2019-05-24 15:55:00,019 INFO  [main] mapreduce.Job: Running job: job_1558683395075_0001
2019-05-24 15:55:11,353 INFO  [main] mapreduce.Job: Job job_1558683395075_0001 running in uber mode
 : false
2019-05-24 15:55:11,353 INFO  [main] mapreduce.Job:  map 0% reduce 0%
2019-05-24 15:55:23,475 INFO  [main] mapreduce.Job:  map 3% reduce 0%
2019-05-24 15:55:26,499 INFO  [main] mapreduce.Job:  map 7% reduce 0%
2019-05-24 15:55:29,538 INFO  [main] mapreduce.Job:  map 12% reduce 0%
2019-05-24 15:55:32,558 INFO  [main] mapreduce.Job:  map 17% reduce 0%
2019-05-24 15:55:35,583 INFO  [main] mapreduce.Job:  map 22% reduce 0%
2019-05-24 15:55:38,611 INFO  [main] mapreduce.Job:  map 28% reduce 0%
2019-05-24 15:55:41,643 INFO  [main] mapreduce.Job:  map 33% reduce 0%
2019-05-24 15:55:45,683 INFO  [main] mapreduce.Job:  map 39% reduce 0%
2019-05-24 15:55:48,716 INFO  [main] mapreduce.Job:  map 46% reduce 0%
2019-05-24 15:55:51,732 INFO  [main] mapreduce.Job:  map 52% reduce 0%
2019-05-24 15:55:54,753 INFO  [main] mapreduce.Job:  map 60% reduce 0%
2019-05-24 15:55:57,792 INFO  [main] mapreduce.Job:  map 68% reduce 0%
2019-05-24 15:56:00,815 INFO  [main] mapreduce.Job:  map 75% reduce 0%
2019-05-24 15:56:03,836 INFO  [main] mapreduce.Job:  map 77% reduce 0%
2019-05-24 15:56:06,855 INFO  [main] mapreduce.Job:  map 84% reduce 0%
2019-05-24 15:56:09,883 INFO  [main] mapreduce.Job:  map 89% reduce 0%
2019-05-24 15:56:12,925 INFO  [main] mapreduce.Job:  map 94% reduce 0%
2019-05-24 15:56:16,025 INFO  [main] mapreduce.Job:  map 97% reduce 0%
2019-05-24 15:56:19,120 INFO  [main] mapreduce.Job:  map 100% reduce 0%
```

4) Check the number of rows for the data:

```
hbase(main):003:0> count 'tweet_data'
```

```
Current count: 1586000, row: 987398
Current count: 1587000, row: 988298
Current count: 1588000, row: 989198
Current count: 1589000, row: 990097
Current count: 1590000, row: 990998
Current count: 1591000, row: 991898
Current count: 1592000, row: 992798
Current count: 1593000, row: 993698
Current count: 1594000, row: 994598
Current count: 1595000, row: 995498
Current count: 1596000, row: 996398
Current count: 1597000, row: 997298
Current count: 1598000, row: 998198
Current count: 1599000, row: 999098
Current count: 1600000, row: 999999
1600000 row(s) in 61.3690 seconds

=> 1600000
```

5) View the first 5 rows:

```
hbase(main):003:0> scan 'tweet_data', {'LIMIT' => 5}
ROW                     COLUMN+CELL
 0                      column=data:label, timestamp=1558689173072, value=0
 0                      column=data:text, timestamp=1558689173072, value=awww that bummer you sho
                        ulda got david carr of third day to do it
 1                      column=data:label, timestamp=1558689173072, value=0
 1                      column=data:text, timestamp=1558689173072, value=is upset that he can not
                         update his facebook by texting it and might cry as result school today a
                        lso blah
 10                     column=data:label, timestamp=1558689173072, value=0
 10                     column=data:text, timestamp=1558689173072, value=spring break in plain ci
                        ty it snowing
 100                    column=data:label, timestamp=1558689173072, value=0
 100                    column=data:text, timestamp=1558689173072, value=body of missing northern
                         calif girl found police have found the remains of missing northern calif
                        ornia girl
 1000                   column=data:label, timestamp=1558689173072, value=0
 1000                   column=data:text, timestamp=1558689173072, value=um that would be hell no
                         to the fugly poker dog pants on the cruise hi jonathan sorry missed you
5 row(s) in 0.0260 seconds
```

6.2 Read Data from HBase by Spark

1) Create a hbase folder under /spark/jars/ folder

2) Move below file from /hbase/lib/ to /spark/jars/hbase/ folder

- All files with hbase as prefix
- guava-12.0.1.jar
- htrace-core-3.1.0-incubating.jar
- protobuf-java-2.5.0.jar

```
student@student-VirtualBox:/home/WQD7007/hbase/lib$ cd /home/WQD7007/spark/jars/
student@student-VirtualBox:/home/WQD7007/spark/jars$ mkdir hbase
student@student-VirtualBox:/home/WQD7007/spark/jars$ cd hbase
student@student-VirtualBox:/home/WQD7007/spark/jars/hbase$ cp /home/WQD7007/hbase/lib/hbase*.jar ./
student@student-VirtualBox:/home/WQD7007/spark/jars/hbase$ ls
hbase-annotations-1.2.12.jar        hbase-examples-1.2.12.jar           hbase-it-1.2.12-tests.jar        hbase-rest-1.2.12.jar
hbase-annotations-1.2.12-tests.jar  hbase-external-blockcache-1.2.12.jar  hbase-prefix-tree-1.2.12.jar     hbase-server-1.2.12.jar
hbase-client-1.2.12.jar             hbase-hadoop2-compat-1.2.12.jar     hbase-procedure-1.2.12.jar       hbase-server-1.2.12-tests.jar
hbase-common-1.2.12.jar             hbase-hadoop-compat-1.2.12.jar      hbase-protocol-1.2.12.jar        hbase-shell-1.2.12.jar
hbase-common-1.2.12-tests.jar       hbase-it-1.2.12.jar                 hbase-resource-bundle-1.2.12.jar hbase-thrift-1.2.12.jar
student@student-VirtualBox:/home/WQD7007/spark/jars/hbase$ cp /home/WQD7007/hbase/lib/guava-12.0.1.jar ./
student@student-VirtualBox:/home/WQD7007/spark/jars/hbase$ cp /home/WQD7007/hbase/lib/htrace-core-3.1.0-incubating.jar ./
student@student-VirtualBox:/home/WQD7007/spark/jars/hbase$ cp /home/WQD7007/hbase/lib/protobuf-java-2.5.0.jar ./
```

3) Download jar file from https://mvnrepository.com/artifact/org.apache.spark/spark-examples_2.11/1.6.0-typesafe-001.

4) Move the downloaded Jar file to /spark/jars/hbase/

```
student@student-VirtualBox:/home/WQD7007/spark/conf$ mv ~/Downloads/spark-examples* /home/WQD7007/spark/jars/hbase/
```

5) Change directory to /home/WQD7007/spark/conf

```
student@student-VirtualBox:/home/WQD7007/spark/conf$ gedit spark-env.sh
student@student-VirtualBox:/home/WQD7007/spark/conf$ ls
docker.properties.template  log4j.properties.template    slaves.template              spark-env.sh
fairscheduler.xml.template  metrics.properties.template  spark-defaults.conf.template  spark-env.sh.template
```

6) Create spark-env.sh and add the following lines into the text file.

```
Open ▾   spark-env.sh
         /home/WQD7007/spark/conf                                   Save
export SPARK_DIST_CLASSPATH=$(/home/WQD7007/hadoop/bin/hadoop classpath):$(/home/WQD7007/hbase/bin/
hbase classpath):/home/WQD7007/spark/jars/hbase/*
```

7) The configuration to connect Pyspark to HBase is done.

8) Next, open Pyspark.

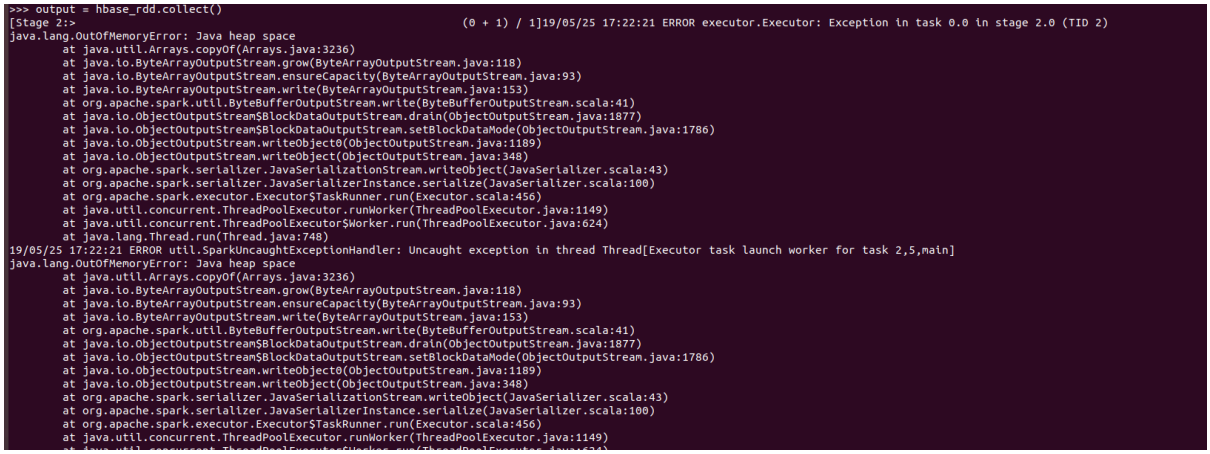
```
student@student-VirtualBox:/home/WQD7007/spark/conf$ pyspark
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/WQD7007/spark/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/WQD7007/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/WQD7007/hbase/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
19/05/25 17:13:58 WARN util.Utils: Your hostname, student-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
19/05/25 17:13:58 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
19/05/25 17:13:59 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.2
      /_/
```

9) Load the "tweet_data" from HBase using the below commands

```
Using Python version 2.7.15rc1 (default, Nov 12 2018 14:31:15)
SparkSession available as 'spark'.
>>> host = 'localhost'
>>> table = 'tweet_data'
>>> conf = {"hbase.zookeeper.quorum": host, "hbase.mapreduce.inputtable": table}
>>> keyConv = "org.apache.spark.examples.pythonconverters.ImmutableBytesWritableToStringConverter"
>>> valueConv = "org.apache.spark.examples.pythonconverters.HBaseResultToStringConverter"
>>> hbase_rdd = sc.newAPIHadoopRDD("org.apache.hadoop.hbase.mapreduce.TableInputFormat","org.apache.hadoop.hbase.io.ImmutableBytesWritable","org.apache.hadoop.hbase.client.Result",
ueConverter=valueConv,conf=conf)

>>>
>>> count = hbase_rdd.count()
>>> hbase_rdd.cache()
MapPartitionsRDD[2] at mapPartitions at SerDeUtil.scala:244
>>> output = hbase_rdd.collect()
```

10) Due to large matrix of the file and insufficient memory from VMWARE(utilizing 8GB ram), the output from HBase cannot be loaded into Spark.

"java.lang.OutofMemoryError" is an indication of memory leak. This error is thrown because there is insufficient space to allocate an object in the Java heap.



7 SENTIMENT ANALYSIS WITH SPARK

Sentiment analysis is a field of Natural Language Processing (NLP) that tries to mine the opinions from the text. However, the dataset of sentiment analysis is usually filled with unstructured information, which is not suitable for machine learning algorithm to analyze, therefore it is a necessary to transform the data to structured data. Sentiment analysis is capable of offering insights for companies to understand customers' opinions about their products and service.

7.1 Spark

When the volume of the data is very huge which cannot be fit into a single machine to deal with, the distributed computing system like Spark might solve the task efficiently. There are 3 different data structures in Spark system: RDD, Dataset and Dataframe.

Spark has three different data structures available through its APIs: RDD, Dataframe. Spark APIs can support Scala, Java, Python and R etc. programming languages. Here we focus on Pyspark.

7.2 PySpark

This section illustrates the use of PySpark to train a predictive model that predicts text sentiment with a set of labelled data stored in Hive. After setting up PySpark, a copy of hive-site.xml or a symbolic link has to be made to the spark conf folder in order to query hive table.

Once a Spark session is initiated, the spark executor and driver memory is set at 4g instead of the default 1g to avoid processing bottleneck. SparkContext is needed when the code runs in a cluster. It tells Spark where to access a cluster and helps to connect with Apache Cluster.

The snapshot below shows the configuration of the spark context.

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Python version 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018 02:44:43)
SparkSession available as 'spark'.
>>> conf = spark.sparkContext._conf.setAll([('spark.executor.memory', '4g'),('spark.driver.memory','4g')])
>>> spark = SparkSession.builder.config(conf=conf).getOrCreate()
>>> spark.sparkContext._conf.getAll()
[('spark.driver.memory', '4g'), ('spark.executor.memory', '4g'), ('spark.executor.id', 'driver'), ('spark.driver.host', '
user/hive/warehouse'), ('spark.sql.catalogImplementation', 'hive'), ('spark.rdd.compress', 'True'), ('spark.app.id', 'loc
ort', '64390'), ('spark.master', 'local[*]'), ('spark.submit.deployMode', 'client'), ('spark.ui.showConsoleProgress', 'tr
```

```
1  sc._conf.getAll()
```

```
[('spark.driver.port', '41895'),
 ('spark.sql.catalogImplementation', 'hive'),
 ('spark.app.id', 'local-1558708349624'),
 ('spark.rdd.compress', 'True'),
 ('spark.executor.memory', '4g'),
 ('spark.serializer.objectStreamReset', '100'),
 ('spark.executor.id', 'driver'),
 ('spark.submit.deployMode', 'client'),
 ('spark.driver.host', '10.0.2.15'),
 ('spark.ui.showConsoleProgress', 'true'),
 ('spark.master', 'local[1]'),
 ('spark.app.name', 'pyspark-shell')]
```

7.2.1 Data Preparation

The original dataset used in our project is from Stanford Project "Sentiment140", which is coming from a Stanford research project. However, in this part, we use pre-cleaned tweets and upload it to the Hive table.

Spark is connected to the existing Hive metastore hence it is able to access all the Hive tables and Databases. First of all, a Dataframe object is created by retrieving the tweets data from the Hive data warehouse by using the SQL select command. The snapshot below shows the top 20 rows of the table, the table has a total of 1,600,000 observations with 3 columns: line_number, text and label. Text refers to the text which is to be analyzed whereas label is the target binary variable where ‘0’ indicates negative sentiment and ‘1’ indicates positive.

The following figure shows the first 10 rows of the DataFrame and the observation counts of each of the respective labels.

```
+-----------+--------------------+-----+
|line_number|                text|label|
+-----------+--------------------+-----+
|          0|awww that bummer ...|    0|
|          1|is upset that he ...|    0|
|          2|dived many times ...|    0|
|          3|my whole body fee...|    0|
|          4|no it not behavin...|    0|
|          5|  not the whole crew|    0|
|          6|             need hug|    0|
|          7|hey long time no ...|    0|
|          8|nope they did not...|    0|
|          9|        que me muera|    0|
+-----------+--------------------+-----+
```

```
>>> df.groupby('label').count().show()
+-----+------+
|label| count|
+-----+------+
|    1|800000|
|    0|800000|
+-----+------+
```

Before sentiment analysis we need to split the dataset for training, validating and testing the model. The total dataset is split into three parts: training, validation, test. Since there are around 1.6 million Tweeter entries, the number of observations here are rather large. The data are split to - training:validation:test ratio of 75:15:10. The screenshot below shows the execution of both the mentioned split ratio. A random seed is specified to ensure reproducibility.

```
>>> (train_set, val_set, test_set) = df.randomSplit([0.75, 0.15, 0.10], seed = 2000)
```

7.2.2 Logistic Regression with TF-IDF

We use pyspark.ml library here to call machine learning models, because it consists of various common machine learning algorithms. Before the model building, the data used requires some transformation process. The original texts of the dataset are tokenized into individual words and term frequency are computed with hashing function with a target feature dimension of 2^{16}.

The figure below shows the first five rows of the transformed train data. It is now ready for model building.

```
>>> (train_set, val_set, test_set) = df.randomSplit([0.75, 0.15, 0.10], seed = 2000)
>>> tokenizer = Tokenizer(inputCol="text", outputCol="words")
>>> hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
>>> idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
>>> label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
>>> pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])
```

```
>>> train_df = pipelineFit.transform(train_set)
>>> val_df = pipelineFit.transform(val_set)
>>> train_df.show(5)
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+
|line_number|                text|label|               words|                  tf|            features|class|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+
|          0|awww that bummer ...|    0|[awww, that, bumm...|(65536,[8436,8847...|(65536,[8436,8847...|  0.0|
|          1|is upset that he ...|    0|[is, upset, that,...|(65536,[1444,2071...|(65536,[1444,2071...|  0.0|
|          2|dived many times ...|    0|[dived, many, tim...|(65536,[2548,2888...|(65536,[2548,2888...|  0.0|
|          3|my whole body fee...|    0|[my, whole, body,...|(65536,[158,11650...|(65536,[158,11650...|  0.0|
|          4|no it not behavin...|    0|[no, it, not, beh...|(65536,[1968,4488...|(65536,[1968,4488...|  0.0|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+
only showing top 5 rows
```

The predictive model is built using Logistic Regression and tested on both the validation and test set. The figure below shows the model accuracy of the 75:15:10 ratio, which is 78.52% and 78.71% respectively.

```
>>> predictions = lrModel.transform(val_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
>>> print(accuracy)
0.7852053951314055
>>>
>>> predictions = lrModel.transform(test_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(test_set.count())
>>> print(accuracy)
0.7871474918891939
```

7.2.3 Logistic Regression with CountVectorizer-IDF

Another way to obtain term frequency with Inverse Document Frequency (IDF) is by using CountVectorizer. Instead of reducing dimension with relative impacts, CountVectorizer drops the infrequent term tokens. A Count Vectorizer converts a collection of text documents into a matrix of token counts where each token refers to a word which can be matched to its token ID. The original texts of the dataset are also tokenized into single words and a target feature dimension of 2^{16} is used in the hashing function to calculate term frequency. The figure below is the first five rows of the transformed data.

```
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+--------
------------+--------------------+----------+
|line_number|                text|label|               words|                  cv|            features|class|       r
awPrediction|         probability|prediction|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+--------
------------+--------------------+----------+
|          0|awww that bummer ...|    0|[awww, that, bumm...|(16384,[0,3,5,10,...|(16384,[0,3,5,10,...|  0.0|[3.68135
290712708...|[0.97543001663847...|       0.0|
|          0|awww that bummer ...|    0|[awww, that, bumm...|(16384,[0,3,5,10,...|(16384,[0,3,5,10,...|  0.0|[3.68135
290712708...|[0.97543001663847...|       0.0|
|          1|is upset that he ...|    0|[is, upset, that,...|(16384,[3,4,6,7,1...|(16384,[3,4,6,7,1...|  0.0|[4.06736
991567062...|[0.98316587735562...|       0.0|
|          1|is upset that he ...|    0|[is, upset, that,...|(16384,[3,4,6,7,1...|(16384,[3,4,6,7,1...|  0.0|[4.06736
991567062...|[0.98316587735562...|       0.0|
|          2|dived many times ...|    0|[dived, many, tim...|(16384,[0,1,9,10,...|(16384,[0,1,9,10,...|  0.0|[-0.4221
955820047...|[0.39599148619943...|       1.0|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+-----+--------
------------+--------------------+----------+
```

Figure below shows the accuracy of the model with 75:15:10 ratio, it has an accuracy on validation set of 79.12% and 79.28% on the test set.

```
>>> predictions_val = pipelineFit.transform(val_set)
>>> accuracy = predictions_val.filter(predictions_val.label == predictions_val.prediction).count() / float(val_set.count())
>>> print("Validation Accuracy Score: {0:.4f}".format(accuracy))
Validation Accuracy Score: 0.7912
>>> predictions_t = pipelineFit.transform(test_set)
>>> accuracy_test = predictions_t.filter(predictions_t.label == predictions_t.prediction).count() /float(test_set.count())
>>> print("Test Accuracy Score: {0:.4f}".format(accuracy_test))
Test Accuracy Score: 0.7928
```

7.2.4 Logistic Regression with N-gram

In Spark, the implementation of N-gram is a little complicated. N-gram function can be called from the pyspark.ml library, and we use pipeline to combine features of N-gram. A target feature dimension of 2^{16} is still used for term frequency.

First we use unigram model to extract features. As expected, the result of using a one-gram tokenization is identical to the one where words are tokenized using the default Tokenizer function as the words are processed in the same manner by both method – each word is considered as a token. With the 75:15:10 ratio, the accuracy on validation set and test set are 78.52% and 78.71% respectively.

```
from pyspark.ml.feature import NGram
```

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
ngram = NGram(n=1, inputCol="words", outputCol="n_gram")
hashtf = HashingTF(numFeatures=2**16,inputCol="n_gram", outputCol="tf")
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer, ngram, hashtf, idf, label_stringIdx, lr])
pipelineFit = pipeline.fit(train_set)
```

```
>>> predictions = lrModel.transform(val_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
>>> print(accuracy)
0.7852053951314055
>>>
>>> predictions = lrModel.transform(test_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(test_set.count())
>>> print(accuracy)
0.7871474918891939
>>>
```

Then the bigram tokenization approach is attempted to test if it helps to enhance the model accuracy. The difference between a 2-gram and 1-gram tokenization approach is that the 2-gram approach takes 2 contiguous sequence as a token where 1-gram takes each word as a token. The snapshot below shows the step to construct the predictive model. Aside from tokenization method, all the other parameters are remained unchanged.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression(maxIter=100)
>>> lrModel = lr.fit(train_df)
[Stage 1771:================================================>      (7 + 1) / 8]19/05/15 18:32:38 WARN MemoryStore: Not enough space to
19/05/15 18:32:38 WARN BlockManager: Persisting block rdd_3857_0 to disk instead.
>>> predictions = lrModel.transform(val_df)
```

```
>>> ngrams = NGram(n=2,inputCol='words', outputCol='ngrams')
>>> hashtf = HashingTF(numFeatures=2**16, inputCol="ngrams", outputCol='tf')
>>> idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
>>> label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
>>> pipeline = Pipeline(stages=[tokenizer, ngrams,hashtf, idf, label_stringIdx])
>>> pipelineFit = pipeline.fit(train_set)
>>> train_df = pipelineFit.transform(train_set)
>>> val_df = pipelineFit.transform(val_set)
>>> train_df.show(5)
+-----------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+-----+
|line_number|                text|label|               words|              ngrams|                  tf|            features|class|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+-----+
|          0|awww that bummer ...|    0|[awww, that, bumm...|[awww that, that ...|(65536,[532,4855,...|(65536,[532,4855,...|  0.0|
|          1|is upset that he ...|    0|[is, upset, that,...|[is upset, upset ...|(65536,[4536,4824...|(65536,[4536,4824...|  0.0|
|          2|dived many times ...|    0|[dived, many, tim...|[dived many, many...|(65536,[1199,3107...|(65536,[1199,3107...|  0.0|
|          3|my whole body fee...|    0|[my, whole, body,...|[my whole, whole ...|(65536,[18047,222...|(65536,[18047,222...|  0.0|
|          4|no it not behavin...|    0|[no, it, not, beh...|[no it, it not, n...|(65536,[3094,6363...|(65536,[3094,6363...|  0.0|
+-----------+--------------------+-----+--------------------+--------------------+--------------------+--------------------+-----+
only showing top 5 rows
```

Figure below shows the performance of the model on validation and test set from the 75:15:10 ratio, which are 72.98% and 73.33% respectively.

```
>>> lr = LogisticRegression(maxIter=100)
>>> lrModel = lr.fit(train_df)
[Stage 482:================================================>       (7 + 1) / 8]19/05/28 21:13:03 WARN MemoryStor
19/05/28 21:13:03 WARN BlockManager: Persisting block rdd_1076_0 to disk instead.
>>> predictions = lrModel.transform(val_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
>>> print(accuracy)
0.72987327347061
>>>
>>> predictions = lrModel.transform(test_df)
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(test_set.count())
>>> print(accuracy)
0.7332855003743449
```

The table below summarizes the performance of each of the method attempted with the two specified ratio.

VAL / TEST	VAL	TEST
TF-IDF	78.52%	78.71%
CountVectorizer-IDF	79.12%	79.28%
UniGram TF-IDF	78.52%	78.71%
BiGram TF-IDF	72.98%	73.33%

The above summary shows that the performance of TF-IDF and UniGram TF-IDF are same and equally good. However, CounterVectorizer-IDF method perform better than these two methods, and all three of these models significantly outperformed the bigram + TF-IDF method.

7.3 Code of Sentiment Analysis

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext

SparkContext.setSystemProperty("spark.executor.memory", "4g")
sc = SparkContext('local[1]')
hc = HiveContext(sc)

hc.sql('use project')
df = hc.sql('select * from tweet_orc where line_number is not null')

(train_set, val_set, test_set) = df.randomSplit([0.75, 0.15, 0.1], seed = 2000)

# Logistic Regression with TFIDF

from pyspark.ml import Pipeline
from pyspark.ml.feature import HashingTF, IDF, Tokenizer, CountVectorizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
#minDocFreq: remove sparse terms
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
train_df = pipelineFit.transform(train_set)
val_df = pipelineFit.transform(val_set)
test_df = pipelineFit.transform(test_set)

lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)

predictions = lrModel.transform(val_df)
# evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
# evaluator.evaluate(predictions)
accuracy = predictions.filter(predictions.label == predictions.prediction).count() /
float(val_set.count())
print("valication accuracy: ", accuracy)
predictions_test = lrModel.transform(test_df)
accuracy_test = predictions_test.filter(predictions_test.label ==
predictions_test.prediction).count() / float(test_set.count())
print("test accuracy: ", accuracy_test)

# Logistic Regression with CountVectorizer and IDF
```

```
from pyspark.ml.feature import CountVectorizer

tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
#minDocFreq: remove sparse terms
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, lr])
pipelineFit = pipeline.fit(train_set)

predictions_val = pipelineFit.transform(val_set)
accuracy = predictions_val.filter(predictions_val.label == predictions_val.prediction).count() /
float(val_set.count())
print("Validation Accuracy Score: {0:.4f}".format(accuracy))
predictions_t = pipelineFit.transform(test_set)
accuracy_test = predictions_t.filter(predictions_t.label == predictions_t.prediction).count() /
float(test_set.count())
print("Test Accuracy Score: {0:.4f}".format(accuracy_test))

# Logisitic Regression with N-gram

from pyspark.ml.feature import NGram
tokenizer = Tokenizer(inputCol="text", outputCol="words")
ngram = NGram(n=1, inputCol="words", outputCol="n_gram")
hashtf = HashingTF(numFeatures=2**16,inputCol="n_gram", outputCol="tf")
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "label", outputCol = "class")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer, ngram, hashtf, idf, label_stringIdx, lr])
pipelineFit = pipeline.fit(train_set)

predictions_val = pipelineFit.transform(val_set)
accuracy = predictions_val.filter(predictions_val.label == predictions_val.prediction).count() /
float(val_set.count())
print("Validation Accuracy Score: {0:.4f}".format(accuracy))

predictions_t = pipelineFit.transform(test_set)
accuracy_test = predictions_t.filter(predictions_t.label == predictions_t.prediction).count() /
float(test_set.count())
print("Test Accuracy Score: {0:.4f}".format(accuracy_test))
```

8 SENTIMENT ANALYSIS WITH R AND POWER BI

8.1 R code

Below is the R programming code that is using to perform sentiment analysis then feed the data into Power BI for visualization purpose.

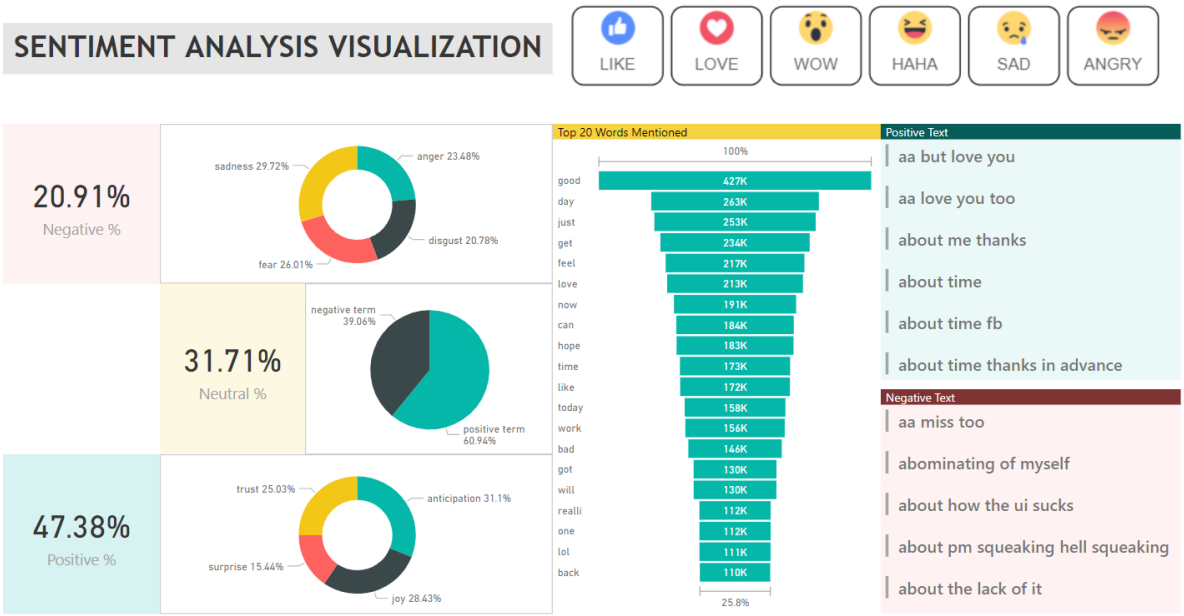
```
library(SnowballC)
library(twitteR)
library(RCurl)
library(httr)
library(tm)
library(wordcloud)
library(syuzhet)
library(sentimentr)
library(SentimentAnalysis)
library(NLP)
library(qdapTools)

frequent_terms <- freq_terms(testing$Description, 10)
plot(frequent_terms)
ft<-data.frame(frequent_terms)
write.csv(ft,"ft2.csv")

emotions <- get_nrc_sentiment(tweets.df$text)
emo_bar = colSums(emotions)
emo_sum = data.frame(count=emo_bar, emotion=names(emo_bar))
emo_sum$emotion = factor(emo_sum$emotion, levels=emo_sum$emotion[order(emo_sum$count, decreasing = TRUE)])
```

In here we will obtain the text status, whether is Negative, Positive or Neutral. We also can count the frequent terms and extract the words that represents emotion such as Sadness, Anger, Fear, Disgust, Trust, Surprise, Anticipation and Joy.

8.2 Power BI

Once the sentiment analysis in R had been done, it is feed into Power BI, some data cleaning can be done either in R or Power BI.

```
# create corpus
corpus = Corpus(VectorSource(wordcloud_tweet2))

# remove punctuation, convert every word in lower case and remove stop words
corpus = tm_map(corpus, tolower)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, c(stopwords("english")))
corpus = tm_map(corpus, stemDocument)

tdm = TermDocumentMatrix(corpus)

# convert as matrix
tdm = as.matrix(tdm)
tdmnew <- tdm[nchar(rownames(tdm)) < 11,]

corpus <- c("Positive text", "Neutral but uncertain text", "Negative text")

sentiment<-analyzeSentiment(tweets.df$text)
Convertvalue<-data.frame(convertToDirection(sentiment$SentimentQDAP))
binaryresponse<-data.frame(table(convertToBinaryResponse(sentiment$SentimentLM)))
counting<-countWords(tweets.df$text, removeStopwords=FALSE)
```



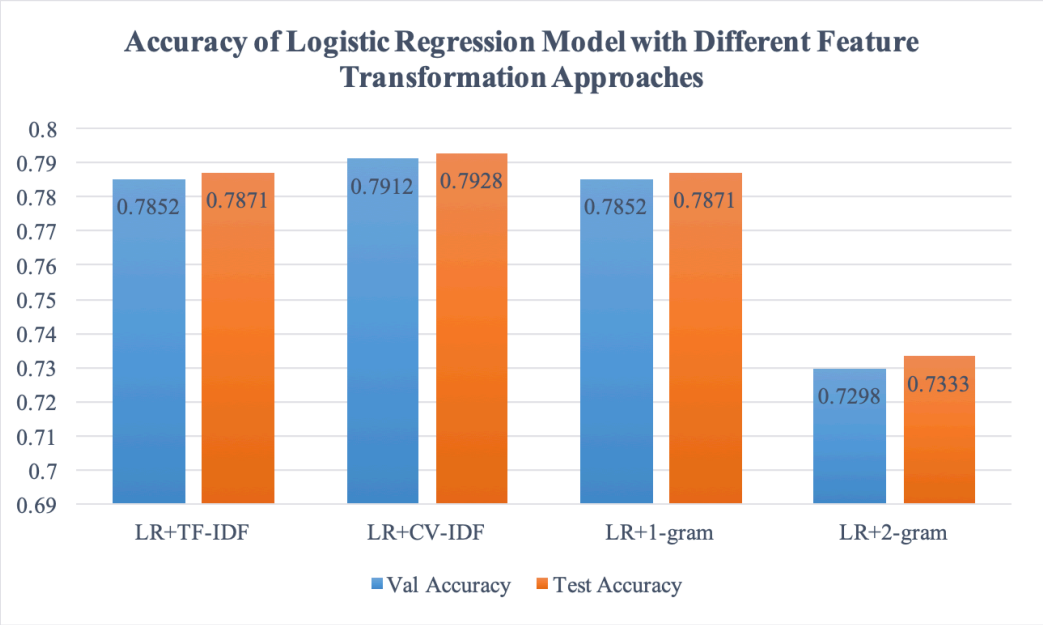
At here, it is obvious that most of the text are positive, then comes to neutral only negative. Beside each the percentage, there are donut and pie chart that shows the percentage of word emotion that is categorised as Sadness, Anger, Fear, Disgust, Trust, Surprise, Anticipation, Joy, Positive term and Negative term.

The top 20 of most frequent words had been visualized as well, most frequent used word is “good” seconded by “day”. At the right hand side, there are two boxes of text lists that is rated as positive and negative respectively.

9 RESULT & DISCUSSION

In this project, we use various feature transformation methods to convert the original text data to numerical vectors. The feature transformation approaches we utilized here include TF-IDF, CounterVectorizer with IDF, N-gram. When Logistic Regression (LR) algorithm combines with different feature transformation fashions, the prediction result of sentiment analysis would be a little different. As shown in the following figure, the accuracy of LR+TF-IDF is identical to the accuracy of LR+1-gram. The LR+2-gram method has worst performance, the accuracy of validation data and test data is 72.98% and 73.33% respectively. The LR+CV-IDF model obtains the best result, the accuracy of LR-CV-IDF is 79.12% and 79.28% for validation dataset and test dataset respectively.

We also compare the efficiency of uploading external structured data to different Data Warehouse tables. Among Hive, Pig and HBase tables, the fast execution of uploading 1.6 million instances of external data is Hive, which takes less than 1 second, the pig takes around 15 seconds, and the Hbase takes more than 1 minute. Thus the descending order of data uploading efficiency of three tables are: Hive, Pig, HBase.



10 CONCLUSION

The text dataset used in the project is collected from public website, and the total instances are more than 1.5 million. After we preprocess the original data, we upload the cleaned dataset to HDFS directory, then we create Hive table, Pig table, HBase table and feed data from HDFS directory to these tables. We found Hive table is the most efficient for uploading external structured data. Finally, we can read data from tables by using PySpark and conduct sentiment analysis.

We can conclude CountVectorizer tokenization is more appropriate for Twitter sentiment classification in this project, because it can gain best prediction performance (79.28%) than other feature transformation approaches such as TF-IDF, unigram, and bigram. The bigram tokenization is the worst model, its accuracy of validation data and test data is only 72.98% and 73.33% respectively. Furthermore, the LR+CV-IDF also gains the best validation accuracy (79.12%) for sentiment analysis. In the future work, we can explore more advanced

machine learning algorithms with Big Data Tools to identify better model for sentiment analysis.

References

[1] http://help.sentiment140.com/for-students/

[2] https://www.kaggle.com/youben/twitter-sentiment-analysis

[3] https://monkeylearn.com/sentiment-analysis/

[4] https://en.wikipedia.org/wiki/Sentiment_analysis

[5] http://bigdataprogrammers.com/load-csv-file-into-hive-orc-table/

[6] http://bigdataprogrammers.com/load-csv-file-in-pig/

[7] http://dwgeek.com/apache-hbase-bulk-load-csv-examples.html/

[8] http://dblab.xmu.edu.cn/blog/1715-2/