CISC 839
Topics in Data Analytics
Competition 4

Big Data Analytics

Sahil Shethiya
20186685
19ss55@queensu.ca
School of Computing

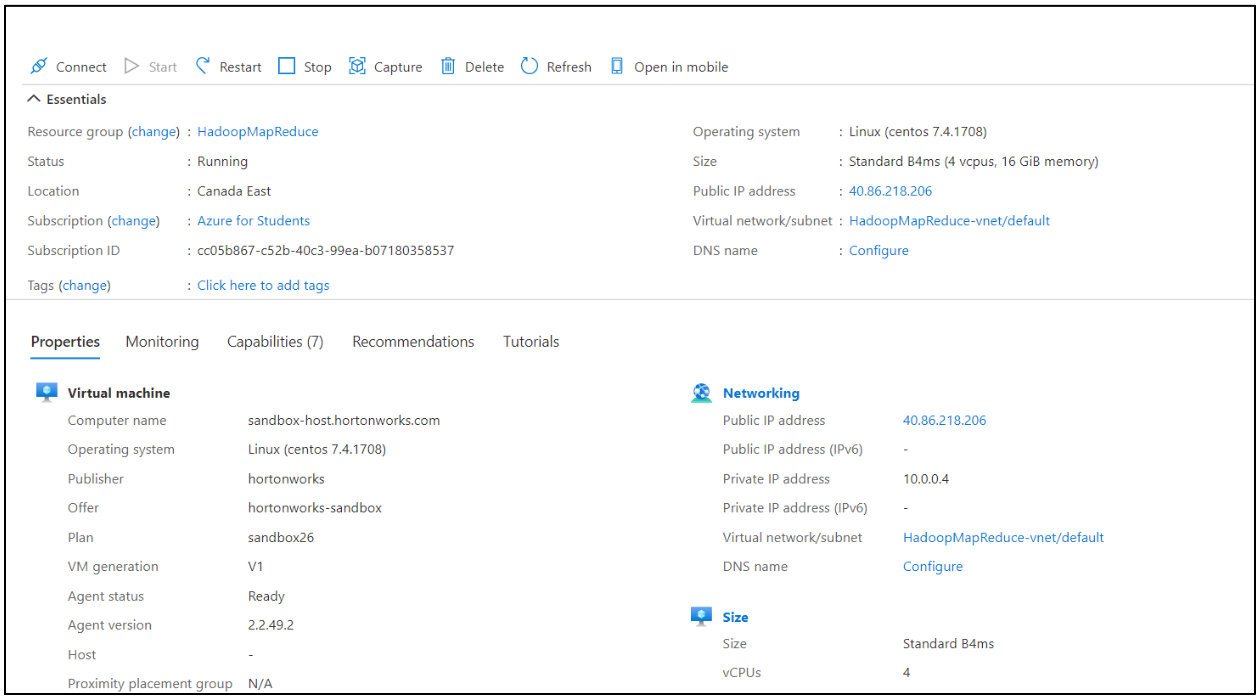
Kamal Andani
20188032
19kaa3@queenu.ca
Department of Electrical and
Computer Engineering

1. Tools and Software Used:

- Our entire solution for this competition was created and developed on Azure cloud HDP VM Instance.
- Programming Used : Java
- IDE : Eclipse (https://www.eclipse.org/downloads/)
- SSH Client : MobaXterm (https://mobaxterm.mobatek.net/)

2. Azure VM Configuration:

- We have created a VM instance on the Azure cloud with Horton data works sandbox platform with HDP 2.5 with the following configuration:



Essentials

Resource group (change)	: HadoopMapReduce	Operating system	: Linux (centos 7.4.1708)
Status	: Running	Size	: Standard B4ms (4 vcpus, 16 GiB memory)
Location	: Canada East	Public IP address	: 40.86.218.206
Subscription (change)	: Azure for Students	Virtual network/subnet	: HadoopMapReduce-vnet/default
Subscription ID	: cc05b867-c52b-40c3-99ea-b07180358537	DNS name	: Configure
Tags (change)	: Click here to add tags		

Virtual machine

Computer name	sandbox-host.hortonworks.com
Operating system	Linux (centos 7.4.1708)
Publisher	hortonworks
Offer	hortonworks-sandbox
Plan	sandbox26
VM generation	V1
Agent status	Ready
Agent version	2.2.49.2
Host	-
Proximity placement group	N/A

Networking

Public IP address	40.86.218.206
Public IP address (IPv6)	-
Private IP address	10.0.0.4
Private IP address (IPv6)	-
Virtual network/subnet	HadoopMapReduce-vnet/default
DNS name	Configure

Size

Size	Standard B4ms
vCPUs	4

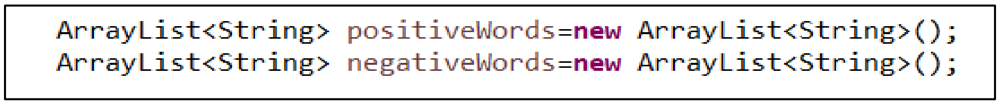
3. Data Exploration:

- To perform sentiment analysis for this solution we have used the following text files:
  i. **Positive.txt** – The file which contains all the words which reflect positive sentiment.
  ii. **Negative.txt** - The file which contains all the words which reflect negative sentiment.
  iii. **SampleReviewData.txt** – The sample review file which is used in the early stage for code verification which contains reviews in the following form defined below: <member id> \t <product id> \t <date> \t <number of helpful feedbacks> \t <number of feedbacks> \t<rating> \t <title> \t <body>

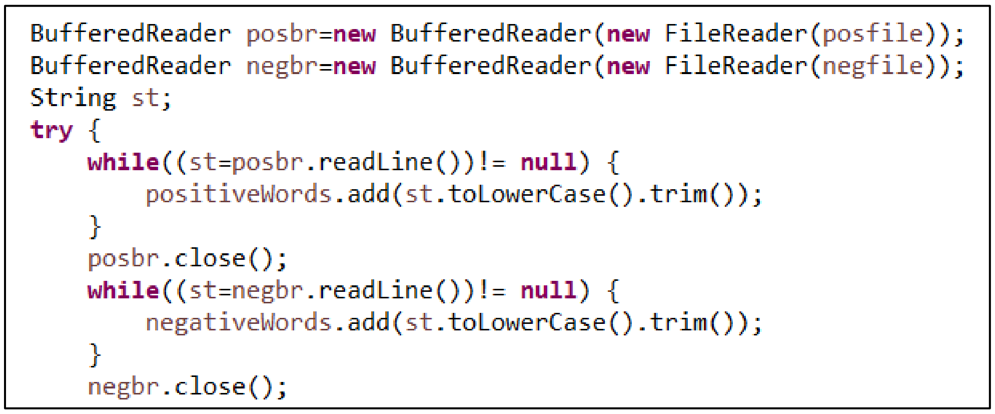
iv. **Reviews.txt** – The main file on which we use our code to do the sentiment analysis and produce results from it. The reviews in this file are in the same format as SampleReviewsData.txt file.

4. Methodology:

- To start with, we are using Hadoop map-reduce to implement our solution.
- Firstly, we are reading both the files i.e. positive.txt and negative.txt, and creating two ArrayLists for each respectively, and storing the words from those files inside the ArrayList respectively.

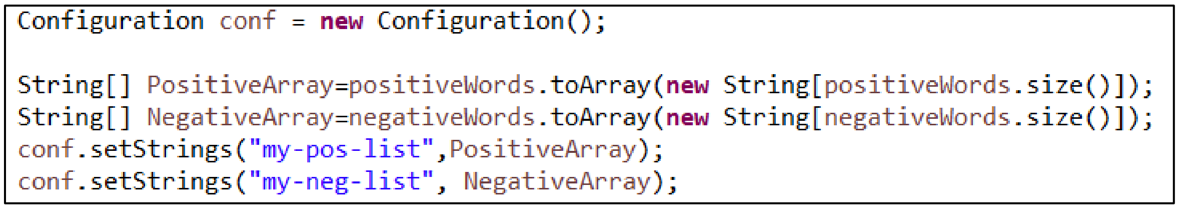


```
ArrayList<String> positiveWords=new ArrayList<String>();
ArrayList<String> negativeWords=new ArrayList<String>();
```



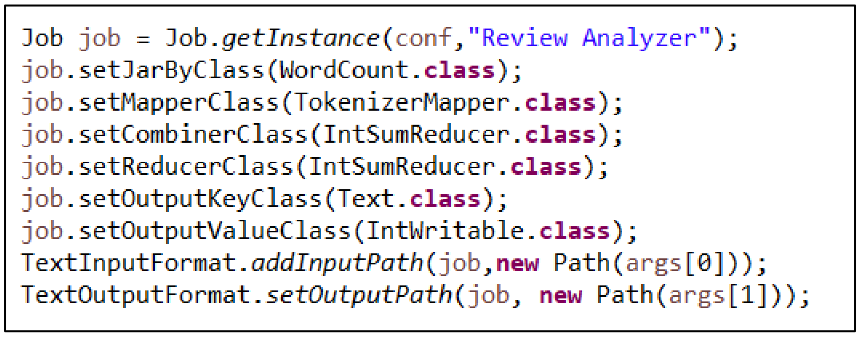
```
BufferedReader posbr=new BufferedReader(new FileReader(posfile));
BufferedReader negbr=new BufferedReader(new FileReader(negfile));
String st;
try {
    while((st=posbr.readLine())!= null) {
        positiveWords.add(st.toLowerCase().trim());
    }
    posbr.close();
    while((st=negbr.readLine())!= null) {
        negativeWords.add(st.toLowerCase().trim());
    }
    negbr.close();
```

- While storing the words inside ArrayList, we convert each word into lowercase and remove access blank space using the trim() command.
- Secondly, we are creating a configuration variable and creating two arrays of strings (i.e. **PositiveArray[]**, **NegativeArray[]**) for each of the ArrayList defined above and store it inside them respectively. The reason behind using Arrays of strings and not using ArrayList directly is because there is no method to store the ArrayList inside the Configuration directly.



```
Configuration conf = new Configuration();

String[] PositiveArray=positiveWords.toArray(new String[positiveWords.size()]);
String[] NegativeArray=negativeWords.toArray(new String[negativeWords.size()]);
conf.setStrings("my-pos-list",PositiveArray);
conf.setStrings("my-neg-list", NegativeArray);
```

- Then, we are creating a Job Instance and set appropriate properties for it.

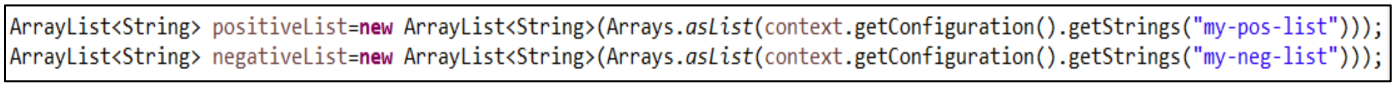


```
Job job = Job.getInstance(conf,"Review Analyzer");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
TextInputFormat.addInputPath(job,new Path(args[0]));
TextOutputFormat.setOutputPath(job, new Path(args[1]));
```

- We define TextInputFormat as our Input format for Hadoop, which will split the file by lines and those lines will be fed into each mapper separately.

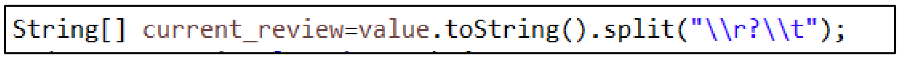
Mapper:

- Firstly, we are converting back the Arrays of Strings from configuration into ArrayList.



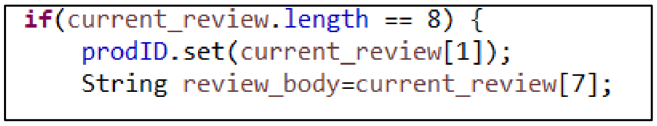
```
ArrayList<String> positiveList=new ArrayList<String>(Arrays.asList(context.getConfiguration().getStrings("my-pos-list")));
ArrayList<String> negativeList=new ArrayList<String>(Arrays.asList(context.getConfiguration().getStrings("my-neg-list")));
```

- Secondly, the mapper takes (key, value) pair as input where value is the content of each line from the input file. Now, we convert each value into a string and split it by tab.



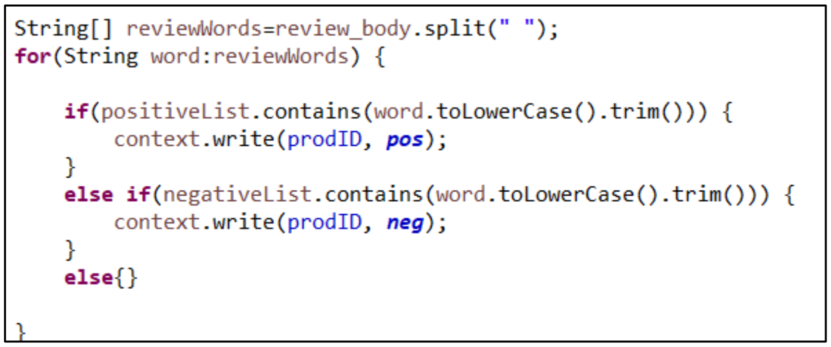
```
String[] current_review=value.toString().split("\\r?\\t");
```

- Now, since we have each review separated into different lines, each review contains 8 different parts from which we are storing the second part as **prodID** and eighth part as the **review_body**.



```
if(current_review.length == 8) {
    prodID.set(current_review[1]);
    String review_body=current_review[7];
```

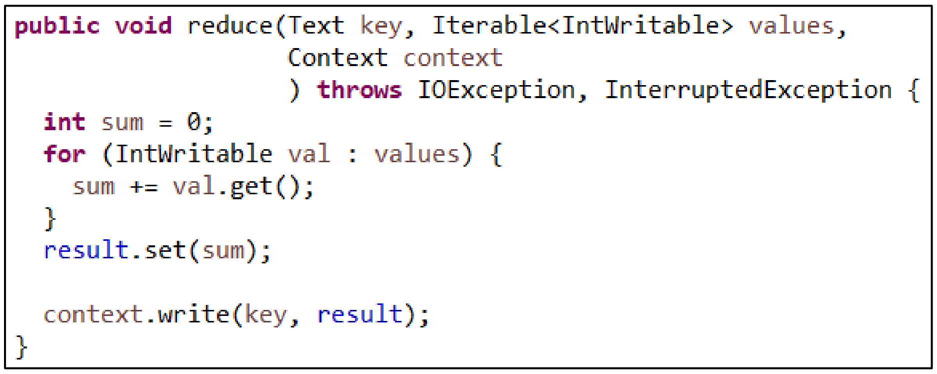
- Furthermore, now we are iterating through each word of **review_body**, and check if the word is positive or negative by checking whether the word is present in **positiveList** or **negativeList**. The code is shown below for that. After that, we are writing mapper output as (key, value) pair where the key is **prodID** and value is **+1** if the word is positive or **-1** if the word is negative.



```
String[] reviewWords=review_body.split(" ");
for(String word:reviewWords) {

    if(positiveList.contains(word.toLowerCase().trim())) {
        context.write(prodID, pos);
    }
    else if(negativeList.contains(word.toLowerCase().trim())) {
        context.write(prodID, neg);
    }
    else{}

}
```

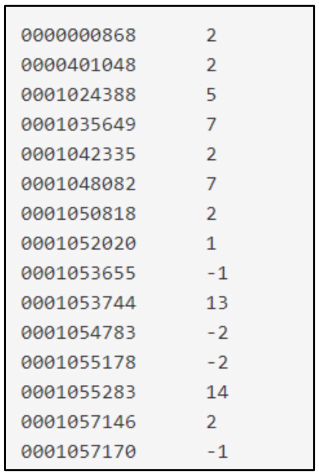
Reducer:

- The reducer takes the output from all the mappers as input and combines them based on the prodID.
- Here it will iterate through all of the values for same **prodID** and do addition for those values. It will give output as (key, Value) pair where key is **prodID** and value is the addition which it calculated. Here if the value of addition is more than zero, then we can say that product has positive review, as number of positive words are more compared to negative words and if value is less than zero, it means that number of negative words are more than positive words, thus that product has negative review.



```
public void reduce(Text key, Iterable<IntWritable> values,
                   Context context
                   ) throws IOException, InterruptedException {
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);

  context.write(key, result);
}
```

This output file will look like this:

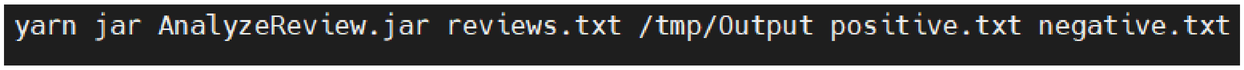


```
0000000868      2
0000401048      2
0001024388      5
0001035649      7
0001042335      2
0001048082      7
0001050818      2
0001052020      1
0001053655      -1
0001053744      13
0001054783      -2
0001055178      -2
0001055283      14
0001057146      2
0001057170      -1
```

Where first column indicates **prodID** and second column indicates review score for that product. (i.e. NoOfPositiveWords – NoOfNegativeWords)

The above steps are performed in **AnalyzeReview.jar**
We used **MobaXterm** to connect VM which we created on Azure.

To run it using Terminal:



```
yarn jar AnalyzeReview.jar reviews.txt /tmp/Output positive.txt negative.txt
```

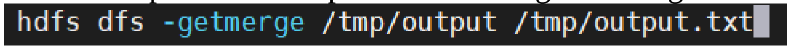
Here we take four arguments where the arguments specify respectively:

1. Path to Input File
2. Path for storing the Output
3. Path to file positive.txt
4. Path to file negative.txt

Formatting Output File:

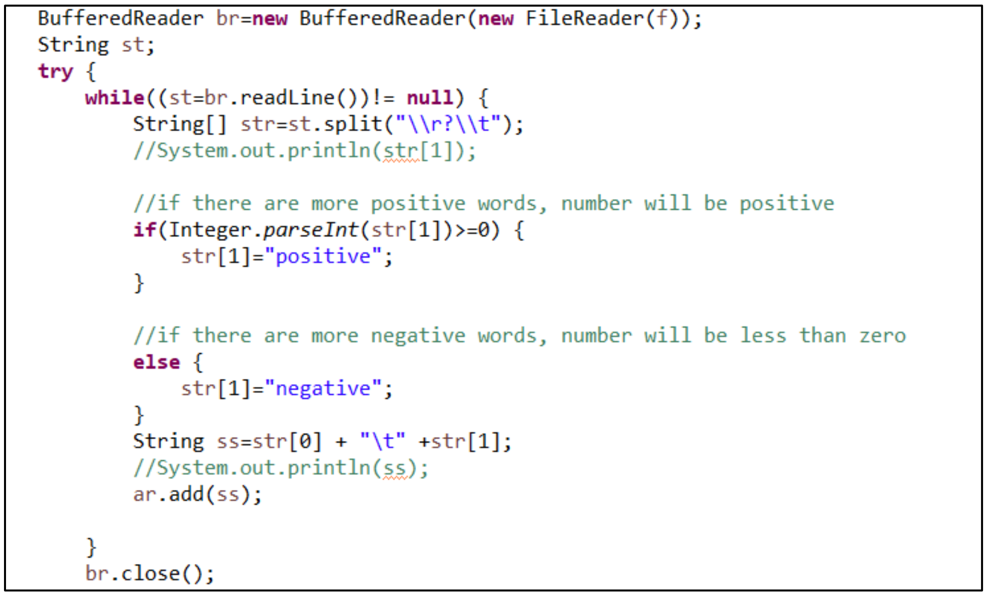
As the output got from above is not as per requirement, we change it by providing appropriate labels (i.e. “positive” or “negative’) to each of the records with the help of **AnalyzeReviewCreateResult** program developed by us.

- Firstly we convert the output file to simple text file using following command:



```
hdfs dfs -getmerge /tmp/output /tmp/output.txt
```

- Next, we read the file line by line and split it by tab. The first part of split string is **prodID** and second part is review score. We then give label ‘positive’ is score >= 0 and ‘negative’, otherwise.

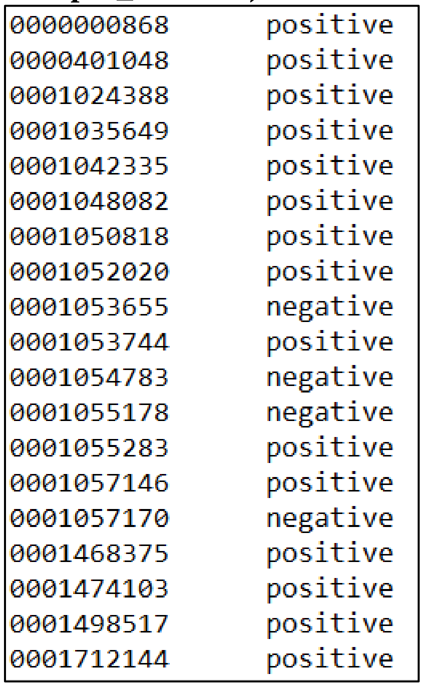


```
BufferedReader br=new BufferedReader(new FileReader(f));
String st;
try {
    while((st=br.readLine())!= null) {
        String[] str=st.split("\\r?\\t");
        //System.out.println(str[1]);

        //if there are more positive words, number will be positive
        if(Integer.parseInt(str[1])>=0) {
            str[1]="positive";
        }

        //if there are more negative words, number will be less than zero
        else {
            str[1]="negative";
        }
        String ss=str[0] + "\t" +str[1];
        //System.out.println(ss);
        ar.add(ss);

    }
    br.close();
```

- Finally, we store the final output to a text file **reviewAnalyzerOutput_Final.txt**

The **AnalyzeReviewCreateResult** program takes two argument:
1. Path to input file
2. Path to output file

Result:
The final result (**reviewAnalyzerOutput_Final.txt)** is shown in the screenshot below:



```
0000000868	positive
0000401048	positive
0001024388	positive
0001035649	positive
0001042335	positive
0001048082	positive
0001050818	positive
0001052020	positive
0001053655	negative
0001053744	positive
0001054783	negative
0001055178	negative
0001055283	positive
0001057146	positive
0001057170	negative
0001468375	positive
0001474103	positive
0001498517	positive
0001712144	positive
```

5. Experience with Hadoop and Challenges Faced:

- To start with Challenges, it was our first time working with Hadoop, At First, we didn’t know how the input file is split inside the mapper and another problem we faced was how to split each review on the line by line basis, which was solved using TextInputFormat.
- Secondly, we didn’t know how to pass a variable with value from main program to mapper in Hadoop. We solved this issue by storing the variable inside the configuration variable and then accessing inside the mapper.
- In the end, we can say that as we had never worked on distributed systems and big data, we learnt a lot of new things from this Competition. Overall, it was a great experience.