Group 16
Jay Shah
Rohit Makhija
Raja Gogineni

Sentiment Analysis of Yelp Reviews

Using Spark, Scala

The goal of the project is to perform sentiment analysis on the reviews for individual businesses of the yelp dataset. Based on the sentiment analysis, we provide our overall rating for each business between 1 to 5 and compare to its original rating.

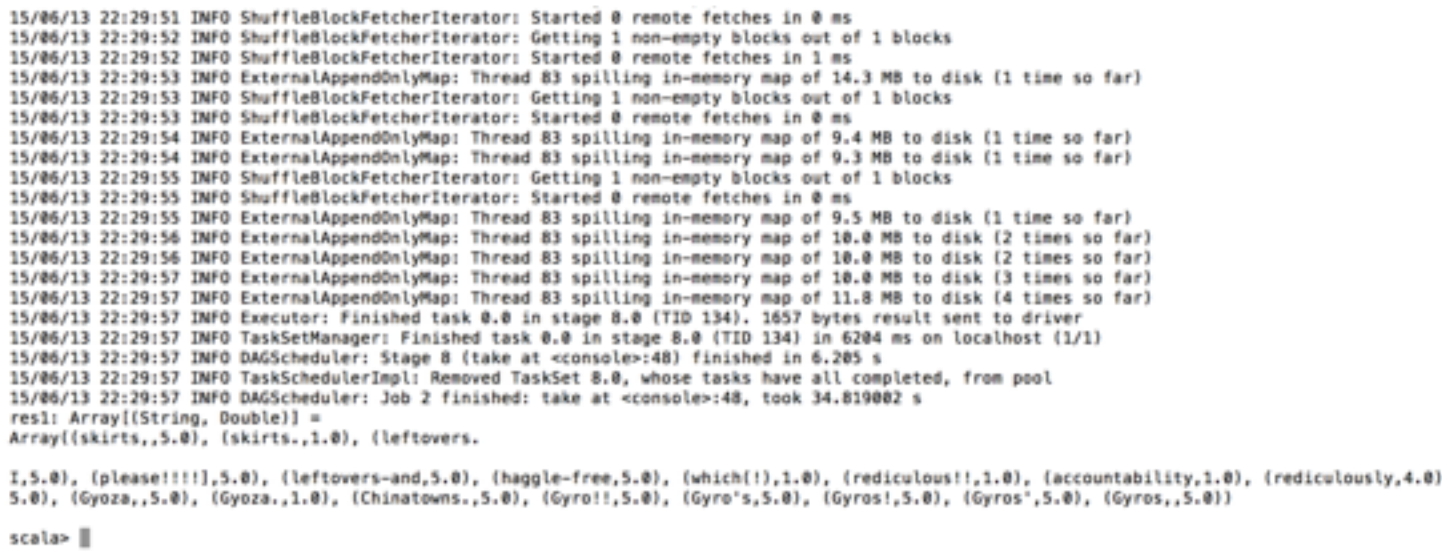
Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. A basic task in sentiment analysis is classifying the polarity of a given text at a sentence, or feature/aspect level — whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Variety of techniques and statistical methods are implemented on various sentiment measurement platforms. When the data is small humans can analyze sentiment but if the data is huge they rely solely on automated sentiment analysis. While some use a hybrid system. Every platform has to train the computers beforehand so that it can identify the sentiment accordingly.

A method for determining sentiment is the use of a scaling system whereby words commonly associated with having a negative, neutral or positive sentiment with them are given an associated number on a 1 to 5 scale and when a piece of unstructured text is analyzed using natural language processing, the subsequent concepts are analyzed for an understanding of these words and how they relate to the concept. Each concept is then given a score based on the way sentiment words relate to the concept, and their associated score. This allows movement to a more sophisticated understanding of sentiment.

We feel that the text analysis of each word based on what all users think about that word would give a precise and more accurate result compared to its original rating.

DESIGN:

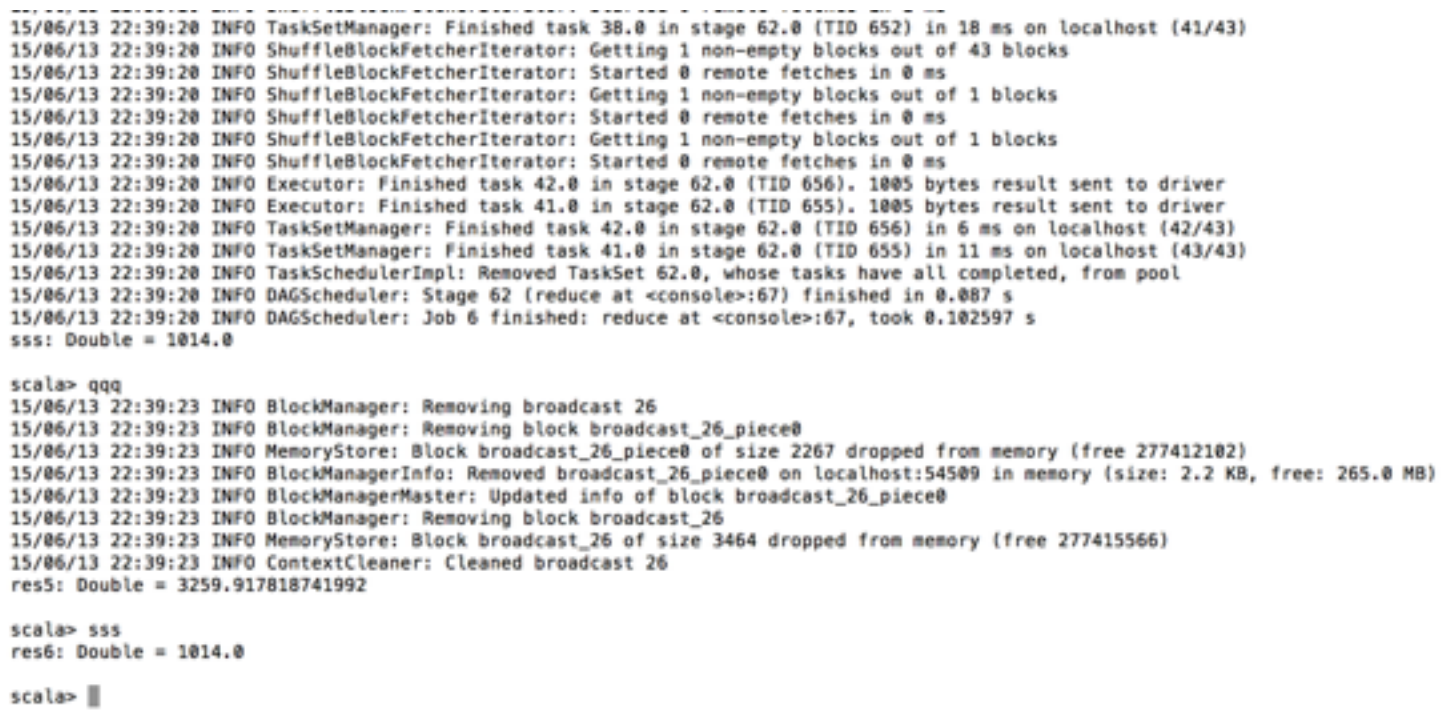
We used Spark to do Sentiment Analysis. There are no in-built machine learning tools in Spark like there in MapReduce, which has Mahout to perform machine learning computations. So we have to implement our own custom algorithm. Although lexicographic approach is the most reliable way to do sentiment analysis, it has drawbacks, it is very language specific and training dataset dependent. And we don’t have a training dataset. So we used a approach that is close to Naïve Bayes Algorithm which works on the probabilities to identify the impact of a word rather than the actual meaning of the word.We wanted to create a rich training dataset. For this we included all the reviews on yelp which had a rating o 5 or 1. After that, we gave a value of 5 and 1 to each word. We did this through simple SQL queries. We then convert each RDD to a Mapped RDD. We do a word count with its values to create an optimum dataset. Initially we had around 1800000 lacs rows of a Key-Value pair having each word and its corresponding value. We then reduced it to 80000 rows and created an optimum dataset.
Sample values of the training dataset.



```
15/06/13 22:29:51 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:29:52 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:29:52 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
15/06/13 22:29:53 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 14.3 MB to disk (1 time so far)
15/06/13 22:29:53 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:29:53 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:29:54 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 9.4 MB to disk (1 time so far)
15/06/13 22:29:54 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 9.3 MB to disk (1 time so far)
15/06/13 22:29:55 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:29:55 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:29:55 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 9.5 MB to disk (1 time so far)
15/06/13 22:29:56 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 10.0 MB to disk (2 times so far)
15/06/13 22:29:56 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 10.0 MB to disk (2 times so far)
15/06/13 22:29:57 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 10.0 MB to disk (3 times so far)
15/06/13 22:29:57 INFO ExternalAppendOnlyMap: Thread 83 spilling in-memory map of 11.8 MB to disk (4 times so far)
15/06/13 22:29:57 INFO Executor: Finished task 0.0 in stage 8.0 (TID 134). 1657 bytes result sent to driver
15/06/13 22:29:57 INFO TaskSetManager: Finished task 0.0 in stage 8.0 (TID 134) in 6204 ms on localhost (1/1)
15/06/13 22:29:57 INFO DAGScheduler: Stage 8 (take at <console>:48) finished in 6.205 s
15/06/13 22:29:57 INFO TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all completed, from pool
15/06/13 22:29:57 INFO DAGScheduler: Job 2 finished: take at <console>:48, took 34.819002 s
res1: Array[(String, Double)] =
Array((skirts,,5.0), (skirts.,1.0), (leftovers.

I,5.0), (please!!!!],5.0), (leftovers-and,5.0), (haggle-free,5.0), (which(!),1.0), (rediculous!!,1.0), (accountability,1.0), (rediculously,4.0)
5.0), (Gyoza,,5.0), (Gyoza.,1.0), (Chinatowns.,5.0), (Gyro!!,5.0), (Gyro's,5.0), (Gyros!,5.0), (Gyros',5.0), (Gyros,,5.0))

scala> ▌
```

We implemented our own algorithm similar to Naive Baye's algorithm. We decided to take all the reviews for individual business id and store all the words from that review file in a RDD as key value var and having the value 0. After that we compare each word with its value from the training dataset and make a final RDD having all our words as a key and its sentiment value as the value from the training dataset. But we had several repetitions for many words, so we then created a separate RDD for the word count of all our words.

<u>RDD values for our review text(after comparison) and its repetition values.</u>

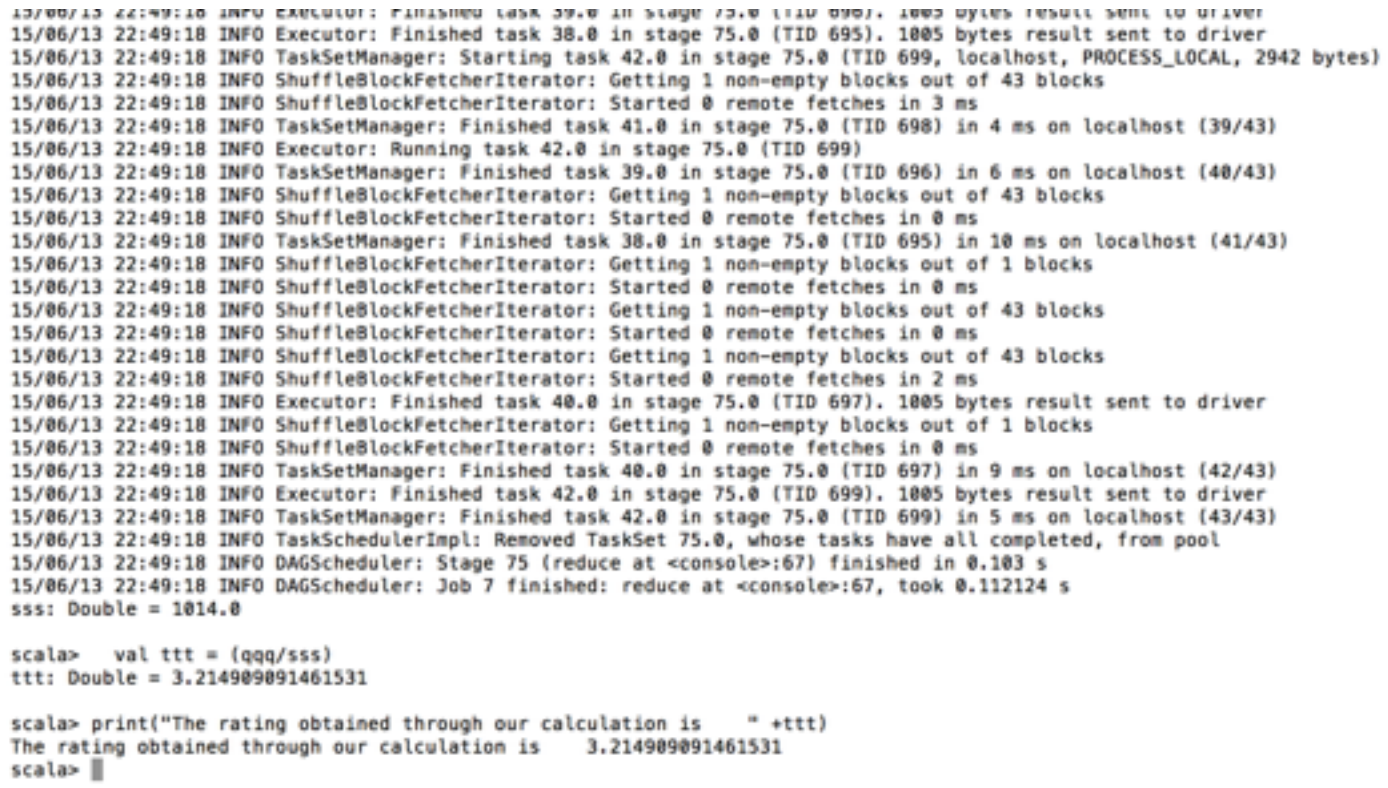


```
15/06/13 22:39:20 INFO TaskSetManager: Finished task 38.0 in stage 62.0 (TID 652) in 18 ms on localhost (41/43)
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 43 blocks
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:39:20 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:39:20 INFO Executor: Finished task 42.0 in stage 62.0 (TID 656). 1005 bytes result sent to driver
15/06/13 22:39:20 INFO Executor: Finished task 41.0 in stage 62.0 (TID 655). 1005 bytes result sent to driver
15/06/13 22:39:20 INFO TaskSetManager: Finished task 42.0 in stage 62.0 (TID 656) in 6 ms on localhost (42/43)
15/06/13 22:39:20 INFO TaskSetManager: Finished task 41.0 in stage 62.0 (TID 655) in 11 ms on localhost (43/43)
15/06/13 22:39:20 INFO TaskSchedulerImpl: Removed TaskSet 62.0, whose tasks have all completed, from pool
15/06/13 22:39:20 INFO DAGScheduler: Stage 62 (reduce at <console>:67) finished in 0.087 s
15/06/13 22:39:20 INFO DAGScheduler: Job 6 finished: reduce at <console>:67, took 0.102597 s
sss: Double = 1014.0

scala> qqq
15/06/13 22:39:23 INFO BlockManager: Removing broadcast 26
15/06/13 22:39:23 INFO BlockManager: Removing block broadcast_26_piece0
15/06/13 22:39:23 INFO MemoryStore: Block broadcast_26_piece0 of size 2267 dropped from memory (free 277412102)
15/06/13 22:39:23 INFO BlockManagerInfo: Removed broadcast_26_piece0 on localhost:54509 in memory (size: 2.2 KB, free: 265.0 MB)
15/06/13 22:39:23 INFO BlockManagerMaster: Updated info of block broadcast_26_piece0
15/06/13 22:39:23 INFO BlockManager: Removing block broadcast_26
15/06/13 22:39:23 INFO MemoryStore: Block broadcast_26 of size 3464 dropped from memory (free 277415566)
15/06/13 22:39:23 INFO ContextCleaner: Cleaned broadcast 26
res5: Double = 3259.917818741992

scala> sss
res6: Double = 1014.0

scala> 
```

After that we clubbed both the RDDs in one single RDD and then for each key we multiplied the value with its no. of occurrences.
e.g. If a word good has a value of 4 in the training dataset and have come 20 times in the review text then its value considered is 4*20=80. We did that for all the words. The last step was a division computation like this,

sum (optimum values of all the words)/ count of all the words in the review text.

This is the result through sentiment analysis.



```
15/06/13 22:49:18 INFO Executor: Finished task 39.0 in stage 75.0 (TID 696). 1005 bytes result sent to driver
15/06/13 22:49:18 INFO Executor: Finished task 38.0 in stage 75.0 (TID 695). 1005 bytes result sent to driver
15/06/13 22:49:18 INFO TaskSetManager: Starting task 42.0 in stage 75.0 (TID 699, localhost, PROCESS_LOCAL, 2942 bytes)
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 43 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 3 ms
15/06/13 22:49:18 INFO TaskSetManager: Finished task 41.0 in stage 75.0 (TID 698) in 4 ms on localhost (39/43)
15/06/13 22:49:18 INFO Executor: Running task 42.0 in stage 75.0 (TID 699)
15/06/13 22:49:18 INFO TaskSetManager: Finished task 39.0 in stage 75.0 (TID 696) in 6 ms on localhost (40/43)
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 43 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:49:18 INFO TaskSetManager: Finished task 38.0 in stage 75.0 (TID 695) in 10 ms on localhost (41/43)
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 43 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 43 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 2 ms
15/06/13 22:49:18 INFO Executor: Finished task 40.0 in stage 75.0 (TID 697). 1005 bytes result sent to driver
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
15/06/13 22:49:18 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
15/06/13 22:49:18 INFO TaskSetManager: Finished task 40.0 in stage 75.0 (TID 697) in 9 ms on localhost (42/43)
15/06/13 22:49:18 INFO Executor: Finished task 42.0 in stage 75.0 (TID 699). 1005 bytes result sent to driver
15/06/13 22:49:18 INFO TaskSetManager: Finished task 42.0 in stage 75.0 (TID 699) in 5 ms on localhost (43/43)
15/06/13 22:49:18 INFO TaskSchedulerImpl: Removed TaskSet 75.0, whose tasks have all completed, from pool
15/06/13 22:49:18 INFO DAGScheduler: Stage 75 (reduce at <console>:67) finished in 0.103 s
15/06/13 22:49:18 INFO DAGScheduler: Job 7 finished: reduce at <console>:67, took 0.112124 s
sss: Double = 1014.0

scala>   val ttt = (qqq/sss)
ttt: Double = 3.214909091461531

scala> print("The rating obtained through our calculation is    " +ttt)
The rating obtained through our calculation is    3.214909091461531
scala> 
```

And the original average rating of the users is 3.5

IMPLEMENTATION:

Dataset:

Overall Cluster

- Environment: Cloudera CDH-5.2 - YARN (MapReduce v2) and Spark
- Worker Nodes: 24
- Cores: 96
- Threads: 192
- RAM: 768GB

- HDFS Storage (Raw): 261TB
- HDFS Storage (Usable): 80TB (After factoring replication overhead)

Specific Nodes

- NameNode
  - Hostname: jshah.hadoop.dc.engr.scu.edu
  - Cores: 4
  - RAM: 24GB
- SecondaryNameNode
  - Hostname: jshah.hadoop.dc.engr.scu.edu
  - Cores: 4
  - RAM: 24GB
- 24x Worker Nodes (DataNode/NodeManager)
  - Hostname: worker-M-NN.hadoop.dc.engr.scu.edu (Where M is 1-2 and NN is 01-12)
  - Cores: 4
  - Threads: 8
  - RAM: 32GB
  - Storage: 12TB
  - Bandwidth: 1Gbit

RELATED WORK:

The code is explained with the comments.

```
val path = "yelp_academic_dataset_review.json"
val mainFile = sqlContext.jsonFile(path)

// create a temporary table call ‘reviews’
mainFile.registerTempTable(“reviews”)

//get all the reviews with stars=1
```

```
val oneRatings = sqlContext.sql("SELECT text FROM reviews WHERE stars=1.0 limit 15000”)

//get all the reviews with stars=5
val fiveRatings = sqlContext.sql("SELECT text FROM reviews WHERE stars=5.0 limit 30000”)

// converting to string
val oneRatings1 = oneRatings.map(row => row.toString())
val fiveRatings1 = fiveRatings.map(row => row.toString())

//word-count on the result of queries
val counts = oneRatings1.flatMap(line => line.split(" ")).map(word => (word, 1.0)).reduceByKey(_ + _)
val counts1 = fiveRatings1.flatMap(line => line.split(" ")).map(word => (word, 5.0)).reduceByKey(_ + _)

//combining results of these two RDD’s into a single RDD
val train_set = counts ++ counts1

//summation of word-count from each query
val summ = train_set.reduceByKey( (x, y) => x + y)

//word count
val cc = oneRatings1.flatMap(line => line.split(" ")).map(word => (word, 1.0)).reduceByKey(_ + _)
val ccc = fiveRatings1.flatMap(line => line.split(" ")).map(word => (word, 1.0)).reduceByKey(_ + _)

//combining the two RDD’s
val train_setc = cc ++ ccc


val abc = train_setc.reduceByKey(_ + _)
val finals = abc ++ summ

//taking the average of the the associated number for each word
val finalTrainingSet = finals.reduceByKey( (a, b) => (if (a>b) a/b else b/a))

//starting hdfs, Spark and sql context.
setup cdh-5.2
spark-shell
val sqlContext = new org.apache.spark.sql.SQLContext(sc)


//function which takes a business_id to calculate overall rating for the given
business
def ans(busId: String)
{
   val allReviews = sqlContext.sql("SELECT text FROM people WHERE
business_id='" + busId + "'")
  val allReviewsString = allReviews.map(row => row.toString())
  val ourReviewWords= allReviewsString.flatMap(line => line.split(" ")).map(word
=> (word, 0))
  val uselessWords = finalTrainingSet.subtractByKey(ourReviewWords)
  val properWords = finalTrainingSet.subtractByKey(uselessWords)
  val hhh = allReviews.subtractByKey(properWords)
//joining and subtracting for the words in our text which have a mapping in the
training data set
  val iii = eee.subtractByKey(hhh)
//mapping all its words to a value of 1
  val jjj = iii.mapValues(x => x+1.0)
// word count of those words
  val kkk = jjj.reduceByKey((a, b) => a + b)
//addition of the adds we just created
  val lll = ggg ++ kkk
multiplying for each key to get a correct values for its no. of occurances
  val multi = lll.reduceByKey( (a, b) => (a*b))
// taking just the values
  val ppp = multi.map(t => t._2)
//reducing for total count of its values
  val qqq = ppp.reduce((a, b) => a + b)
  val rrr = kkk.map(t => t._2)
//same for the other words
  val sss = rrr.reduce((a, b) => a + b)
// finally, diving by its total sum with the value of its occurrences.
val ttt = (qqq/sss)
  print("The rating obtained through our calculation is    " +ttt)
  println()
}

// a sample business_id
var are1="vcNAWiLM4dR7D2nwwJ7nCA"

//function call to the and function passing business id as the argument
ans(are1)

//  for loop to calculate the ratings of all the businesses(business_id's)
val z= sqlContext.sql("SELECT distinct business_ids from reviews")
val abc=z.collect()
for(y<-1 to 2){
  var are=abc(y)
  var are1=are.mkString
  ans(are1)
}

//FOR CALCULATING THE ORIGINAL RATING from another file.
val path1="yelp/yelp_academic_dataset_business.json"
val people1=sqlContext.jsonFile(path1)
people1.registerTempTable("people1")
val busID1="vcNAWiLM4dR7D2nwwJ7nCA"
val originalRating = sqlContext.sql("SELECT stars FROM people1 WHERE business_id='" + busId1 + "'")
print("Original Rating given by the user is")
originalRating.take(1)
println()
    //you will get this value as ARRAY[3.5]
```

REFERENCES:

1. HTTPS://SPARK.APACHE.ORG
2. HTTPS://WWW.CS.BERKELEY.EDU/~MATEI/PAPERS/2012/NSDI_SPARK.PDF
3. HTTP://WWW.CS.BERKELEY.EDU/~MATEI/PAPERS/2010/HOTCLOUD_SPARK.PDF
4. HTTPS://WWW.SAFARIBOOKSONLINE.COM/LIBRARY/VIEW/LEARNING-SPARK/9781449359034/
5. HTTPS://SPARK.APACHE.ORG/DOCS/LATEST/PROGRAMMING-GUIDE.HTML