DETECTING OPINION SPAMMERS

AND

CLASSIFYING AMAZON FOOD REVIEWS USING SENTIMENT ANALYSIS AND MACHINE LEARNING

* Figure above: Word cloud created for our dataset created using R

Apurva Alekar -aaa161530

Gajanan Golegaonkar – gsg160230

Akhilesh Joshi-adj160230

Table of contents

1. Introduction and problem description

Evaluative texts on the Web have become a valuable source of opinions on products, services, events, individuals, etc. Recently, many researchers have studied such opinion sources as product reviews, forum posts, and blogs. However, existing research has been focused on classification and summarization of opinions using natural language processing and data mining techniques. A principal issue that has been neglected so far is opinion spam or trustworthiness of online opinions

2. Related work

- J. McAuley and J. Leskovec.
  From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.
- Opinion Spam and Analysis by Nitin Jindal and Bing Liu
  https://pdfs.semanticscholar.org/16f2/deb863ef6d3d6f432de12a2e81149ab03e5a.pdf

3. Dataset Description

The dataset consists of reviews of fine foods from Amazon. Reviews are recorded up to October 2012. Reviews include product and user information, ratings, and a plaintext review. Given a review text, we plan to detect which users falsely give fake reviews either to degrade the rating of the product or upgrade it. e.g. A user is giving negative reviews for each of the products that he/she has reviewed, then may be that user is trying to spam.

Year:	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Counts:	6	32	13	73	133	560	1344	6686	22358	34144	55403	86092	163546	198064

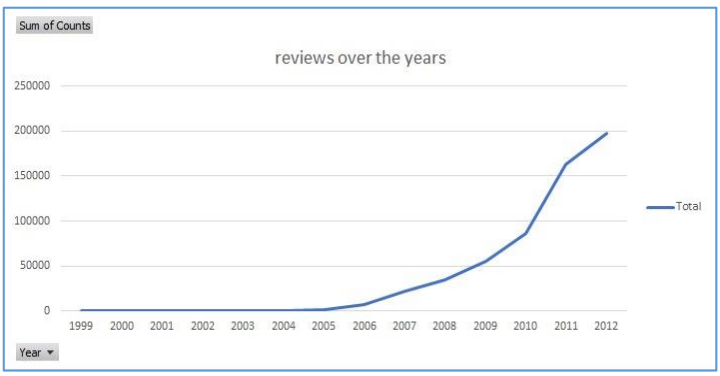
Dataset Details:

1. **No of features**: 9
2. **Features are**:
   i. **productId:** Product Id for which the review is given
   ii. **userId:** Unique Identifier for a user
   iii. **profileName:** a profile name for user.
   iv. **helpfulness:** Other users rating this review as helpful or not i.e. ratio of number of users who found the review helpful to number of users who indicated whether they found the review helpful
   v. **score:** A score/rating between 0.0 to 5.0
   vi. **time:** Unix Timestamp of the review.
   vii. **summary:** brief Summary of the review
   viii. **text:** text of review
3. **Number of Instances**: 568,455

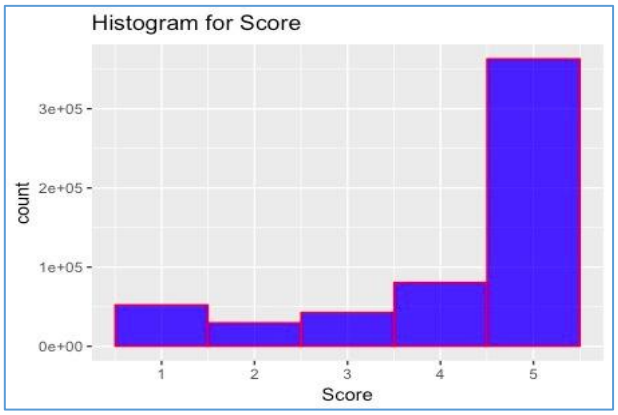
4. **Data Distribution**
   i. 0 N/A values found in data.
   ii. summary (based on Score)

Minimum	Maximum	Mean	Median
1.0	5.0	4.183	5.0

   iii. Number of Reviews we have based on Years:



Histogram for Score



4. Preprocessing techniques

Steps:

1. Remove all non-available data (missing values). This ensures all the data we have is complete and to the mark, resulting in more accurate results.

2. Using sentiment analysis, we classified each review as positive or negative about the product. Then we split the data into training and testing part. Created a model using random forest and logistic regression on training part. And tested them on testing data.

5. Experimental methodology

Sentiment Analysis is the process of 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. A common use case for this technology is to discover how people feel about a topic. We will be implementing the same in our project.

Using Sentiment Analysis, we are trying to predict the strong negative sense or strong positive sense in the user's review. If the same user is using this technique in all its reviews, then that user is spamming reviews. MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction. Spark includes a new package called spark.ml, which aims to provide a uniform set of high-level APIs that help users create and tune practical machine learning pipelines. We have used the pipelines to model our data with machine learning.

Pipelines are a simple and effective way to manage complex machine learning workflow. Its power stands out even more when we get to cross-validation for hyperparameter tuning. Overall, the Pipeline API is a major step in making machine learning scalable, easy, and enjoyable. In other words, it lets us focus more on solving a machine learning task, instead of wasting time spent on organizing code.

A Spark Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input Data Frame is transformed as it passes through each stage.

Why we used Pipelines?

Typically, during the exploratory stages of a machine learning problem, we find ourselves iterating through dozens, if not hundreds, of features and model combinations.

Trying to ensure that our training and test data go through the identical process is manageable, but also tends to be tedious and error prone. A better solution is to wrap each combination of steps with a Pipeline. This gives us a declarative interface where it's easy to see the entire data extraction, transformation, and model training workflow.

6. Coding language / techniques to be used:

- **Coding Language:**
    - Scala (spark)
    - Databricks cluster
    - Spark MLlib

Techniques:

Classification using Random Forest:

Classification is considered an instance of supervised learning. Classification is categorization of data based on some features. It can be used to predict which category the new data belongs given training set of data whose class (the category to which it belongs) is known.

Random forests are an ensemble learning method for classification ,regression or such other tasks. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forest Trees are used to avoid overfitting that is happened during normal decision tree training.

Text Analysis:

Text Analysis or sometimes referred as Text Mining is extracting useful information from plain text. Mostly Information is constructed from some Pattern or trends. It usually involves parsing of data, deriving some useful pattern, removal of obsolete stuff and coming to useful information which is output.

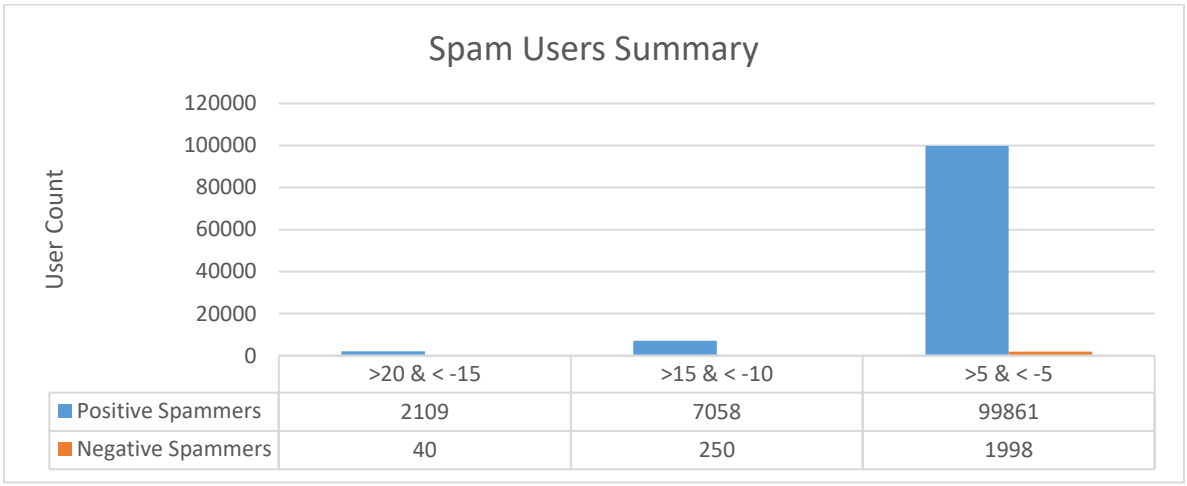
Sentiment Analysis:

Sentiment Analysis is the process of 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. A common use case for this technology is to discover how people feel about a topic. We will be implementing the same in our project.

7. Sentiment Analysis:

We used AFFIN dataset which provides us a comprehensive list of words and their sentiment score. With this information, we iterated through our reviews dataset to find a comprehensive sentiment score for each review. Then based on the average sentiment score of the entire dataset, we labelled the reviews as positive(good) or negative(bad). This process was done using map reduce paradigm and Spark RDDs for transformations.

Some of the interesting analysis we came up with are:



Sentiment Range	Positive Spammers	Negative Spammers
>20 & < -15	2109	40
>15 & < -10	7058	250
>5 & < -5	99861	1998

8. Sentiment Analysis code snippet:

Input: AFINN text file (word sentiment score file)
Output: sentiment score (positive or negative)

a) AFINN FILE is broadcasted first to calculate the sentiment scores

```
val scalaMap = AFINN.collectAsMap.toMap
val b = sc.broadcast(scalaMap)
```

▸ (1) Spark Jobs

```
scalaMap: scala.collection.immutable.Map[String,Int] = Map(youthful -> 2, forgotten -> -1, derided -> -2, terrible -> -3, rage -> -2, apologises -> -1, pollu
ter -> -2, disinclined -> -2, laughing -> 1, blurry -> -2, intrigues -> 1, scare -> -2, exploit -> -2, accident -> -2, badass -> -3, aggravates -> -2, rapist
-> -4, rainy -> -1, sweet -> 2, green washing -> -3, beautiful -> 3, funny -> 4, lonesome -> -2, devastating -> -2, warns -> -2, boycotted -> -2, drown -> -
2, killed -> -3, raptured -> 2, carefree -> 1, weird -> -2, misrepresentation -> -2, strengthened -> 2, unsure -> -1, euphoria -> 3, wow -> 4, racist -> -3,
 contemptuous -> -2, kills -> -3, scandal -> -3, appeased -> 2, deceives -> -3, futile -> 2, cruelty -> -3, worshiped -> 3, pardoning -> 2, enjoying -> 2, pl
ease -> 1, spirited -> 2, interesting -> 2, apocalyptic -> -2, trouble -> -2, greenwashers -> -3, stereotyped -> -2, screaming -> -2, disoriented -> -2, haun
t -> -1, sentences -> -2, rapture -> 2, victimized -> -3, picturesque -> 2, lunatic -> -3, incompetent -> -2, absolve -> 2, grieved -> -2, failure -> -2, pat
hetic -> -2, defer -> -1, winning -> 4, intimidation -> -2, intricate -> 2, impressive -> 3, reassure -> 1, exaggerate -> -2, intelligent -> 2, stunning ->
 4, prosperous -> 3, shoot -> -1, bully -> -2, losing -> -3, winner -> 4, funeral -> -1, contend -> -1, assfucking -> -4, support -> 2, banned -> -2, tout ->
-2, skeptics -> -2, choking -> -2, recommend -> 2, some kind -> 0, fascinated -> 3, chances -> 2, deceitful -> -3, superb -> 5, preventing -> -1, welcomed ->
2, scream -> -2, cancel -> -1, revengeful -> -2, enthral -> 3, angers -> -3, acrimonious -> -3, absorbed -> 1, naïve -> -2, horrible -> -3, unbelievable -> -
```

b) Code snippet for our custom transformer that calculates the sentiment score

```
val reviewsSenti = extracted_reviews.map(reviewText => {
 val reviewWordsSentiment = reviewText(9).toString.split(" ").map(word => {
val senti : Int = b.value.getOrElse(word.toLowerCase(),0)
senti;
});

val reviewSentiment = reviewWordsSentiment.sum
(reviewText(0).toString,reviewText(1).toString,reviewText(2).toString,reviewText(3).toString,reviewText(4).toString,
reviewText(5).toString,reviewText(6).toString,reviewText(7).toString,reviewText(8).toString,reviewText(9).toString,r
eviewSentiment) })
```

Sentiment Result sample(can be seen under column _11):

```
reviewsSenti.show()
```

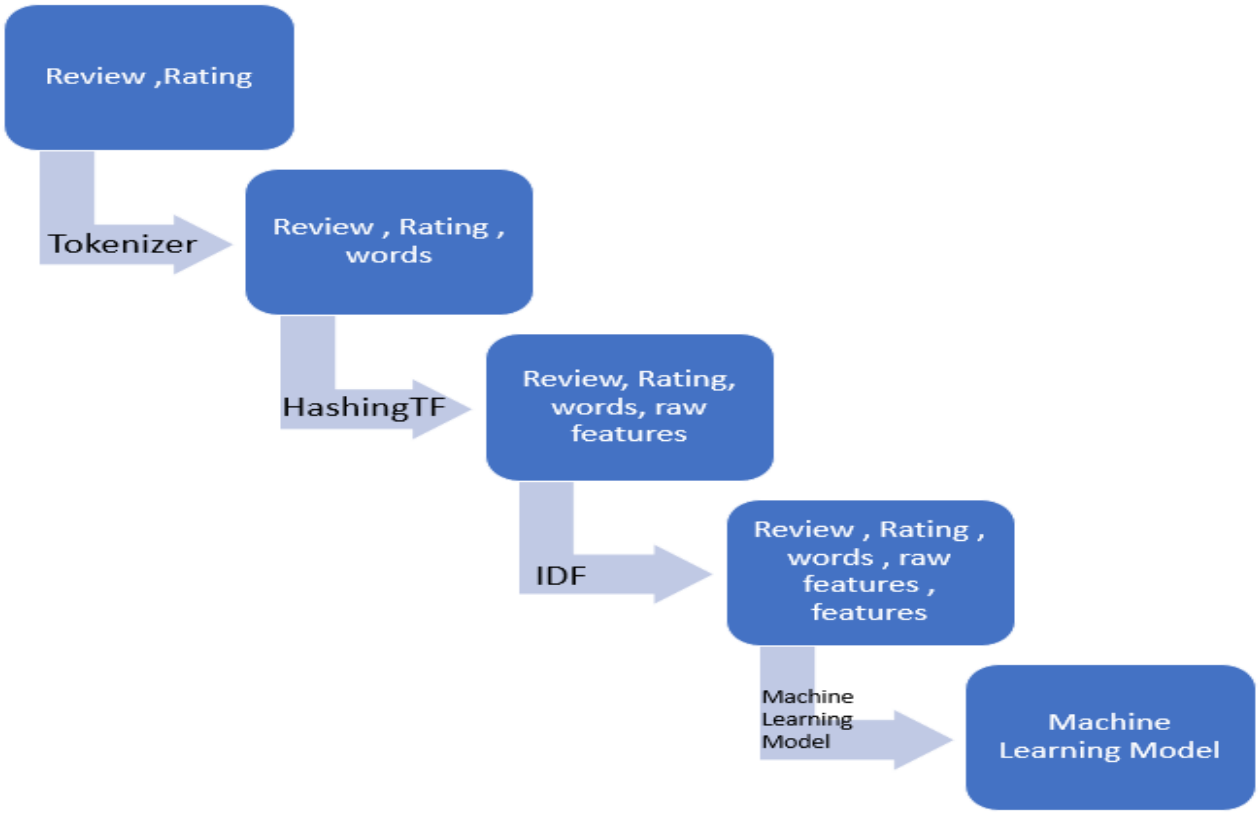
▸ (1) Spark Jobs

_1	_2	_3	_4	_5	_6	_7	_8	_9	_10	_11
1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog ...	I have bought sev...	12
2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	"Product arrived ...	-2
3	B000LQOCH0	ABXLMWJIXXAIN	"Natalia Corres "...	1	1	4	1219017600	"""Delight"" says...	"This is a confec...	1
4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are lookin...	0
5	B006K2ZZ7K	A1UQRSCLF8GW1T	"Michael D. Bigha...	0	0	5	1350777600	Great taffy	Great taffy at a ...	9
6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	4	1342051200	Nice Taffy	I got a wild hair...	2
7	B006K2ZZ7K	A1SP2KVKFXXRU1	David C. Sullivan	0	0	5	1340150400	Great! Just as g...	This saltwater ta...	6
8	B006K2ZZ7K	A3JRGQVEQN31IQ	Pamela G. Williams	0	0	5	1336003200	Wonderful, tasty ...	This taffy is so ...	2
9	B000E7L2R4	A1MZYO9TZK0BBI	R. James	1	1	5	1322006400	Yay Barley	Right now I'm mos...	3
10	B00171APVA	A21BT40VZCCYT4	Carol A. Reed	0	0	5	1351209600	Healthy Dog Food	This is a very he...	8

9. Classification and Model creation using Machine Learning

Steps Followed:

1. Split each document's text into words.
2. Convert each document's words into a numerical feature vector.
3. Learn a prediction model using the feature vectors and labels



Dataset for our predictive model

We have only considered 3 columns for our ML dataset. User Id, User Review and Sentiment score that is calculated from our sentiment analysis part.

Label generation from sentiment score

All the sentiment scores that are above average sentiments (5.22 in our case) are termed to be positive (assigned a value 1) otherwise negative (assigned a value 0)

Data Frame Schema for ML looks like as below:

root

|-- Id: integer (nullable = true)

|-- Text: string (nullable = true)

|-- label: double (nullable = false)

ML workflow

a. **Train / Test split (80 : 20)**

   Original Dataset Records = 568454

   Training Recods = 454639, 79.97815126641734%

   Test Records = 113815, 20.021848733582665%

b. **Pipeline summary**

   Tokenizer:

   inputCol: input column name (current: Text)

   outputCol: output column name (default: tok_536eb1576e51__output, current: words)

   ****

   Remover :

   caseSensitive: whether to do a case-sensitive comparison over the stop words (default: false, current: false)

   inputCol: input column name (current: words)

   outputCol: output column name (default: stopWords_d5b6651cffcb__output, current: filtered)

   stopWords: the words to be filtered out (default: [Ljava.lang.String;@fd36dcd)

   ****

   HashingTF:

   binary: If true, all non zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts (default: false)

   inputCol: input column name (current: filtered)

numFeatures: number of features (> 0) (default: 262144, current: 1000)

outputCol: output column name (default: hashingTF_cd05d42655c1__output, current: rawFeatures)

****

IDF:

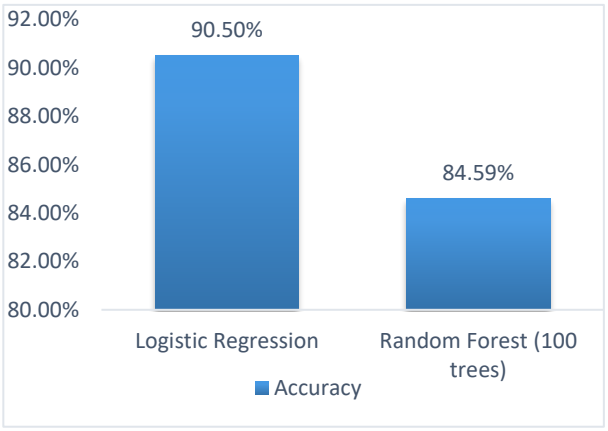
inputCol: input column name (current: rawFeatures)

minDocFreq: minimum number of documents in which a term should appear for filtering (>= 0) (default: 0, current: 0)

outputCol: output column name (default: idf_4c2cb4fe2820__output, current: features)

Accuracy of Machine learning models

In our case, logistic regression gave better performance than random forest on test data. The summary of accuracy is as show below:



Tuning the above machine learning model with cross validation

We ran cross validator to determine the best possible parameters for our model. It took around 15 mins to tune the model with 48 spark jobs and 906 sub jobs.

Improvements

Area under the ROC curve for non-tuned model = 0.9050443215592061

Area under the ROC curve for fitted model = 0.9050443215592016

Improvement = 0.00%

An improvement of 0 percent shows that we already had the model with best parameters. Hence, we didn't saw any difference for same.

10. Machine Learning code snippet:

Dataset

```
val mldatabase = sqlContext.sql("SELECT Id,Text,SentiScore as label FROM HELP")
mldatabase.printSchema()
```

```
root
 |-- Id: integer (nullable = true)
 |-- Text: string (nullable = true)
 |-- label: integer (nullable = false)

mldatabase: org.apache.spark.sql.DataFrame = [Id: int, Text: string ... 1 more field]
```

Command took 0.09 seconds -- by adj160230@utdallas.edu at 4/30/2017, 12:48:54 PM on 30april

Train/Test Split

```
//test - train split
val Array(training, test) = mldatabase.randomSplit(Array(0.8, 0.2), seed = 12345)
```

```
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Id: int, Text: string ... 1 more field]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Id: int, Text: string ... 1 more field]
```

Command took 0.13 seconds -- by adj160230@utdallas.edu at 4/30/2017, 12:48:58 PM on 30april

```
println("Original Dataset Records = " + mldatabase.count())
println("Training Recods = " + training.count() + ", " + training.count*100/(mldatabase.count()).toDouble + "%")
println("Test Records = " + test.count() + ", " + test.count*100/(mldatabase.count().toDouble) + "%")
```

▸ (7) Spark Jobs

```
Original Dataset Records = 568454
Training Recods = 454639, 79.97815126641734%
Test Records = 113815, 20.021848733582665%
```

Command took 31.55 seconds -- by adj160230@utdallas.edu at 4/30/2017, 12:49:01 PM on 30april

Pipeline creation :

```
val tokenizer = new Tokenizer().setInputCol("Text").setOutputCol("words")
val remover = new StopWordsRemover().setInputCol("words").setOutputCol("filtered").setCaseSensitive(false)
val hashingTF = new HashingTF().setNumFeatures(1000).setInputCol("filtered").setOutputCol("rawFeatures")
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features").setMinDocFreq(0)
val lr = new LogisticRegression().setRegParam(0.01).setThreshold(0.5)
val pipeline = new Pipeline().setStages(Array(tokenizer, remover, hashingTF, idf, lr))
```

```
tokenizer: org.apache.spark.ml.feature.Tokenizer = tok_d712692c8f49
remover: org.apache.spark.ml.feature.StopWordsRemover = stopWords_8e75ff56c958
hashingTF: org.apache.spark.ml.feature.HashingTF = hashingTF_1eb3466b851c
idf: org.apache.spark.ml.feature.IDF = idf_6b169f7635c3
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_8146c6458285
pipeline: org.apache.spark.ml.Pipeline = pipeline_d961f96aeed2
```

Command took 0.14 seconds -- by adj160230@utdallas.edu at 4/30/2017, 1:05:45 PM on 30april

Creating Machine Learning Model

```
val model = pipeline.fit(training)
```

▸ (42) Spark Jobs

```
model: org.apache.spark.ml.PipelineModel = pipeline_d961f96aeed2
```

Command took 46.56 seconds -- by adj160230@utdallas.edu at 4/30/2017, 1:05:50 PM on 30april

```
val predictions = model.transform(test)
```

```
predictions: org.apache.spark.sql.DataFrame = [Id: int, Text: string ... 8 more fields]
```

Command took 0.13 seconds -- by adj160230@utdallas.edu at 4/30/2017, 1:06:55 PM on 30april

Viewing the results

```
val evaluator = new BinaryClassificationEvaluator().setMetricName("areaUnderROC")
println("Area under the ROC curve = " + evaluator.evaluate(predictions))
```

▸ (4) Spark Jobs

```
Area under the ROC curve = 0.9050443215591997
evaluator: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = binEval_f813a10f3ae2
```

Command took 7.15 seconds -- by adj160230@utdallas.edu at 4/30/2017, 1:07:04 PM on 30april

11.Conclusion

We can conclude in two observations

1. By sentiment analysis we can see the values that are anomalous or very far from the mean or predicted sentiment scores are definitely opinion spammers whereas as the value approaches to the mean we can't say whether the opinion is spammed or not spammed.
2. The machine learning model that we built will serve the future purpose, where given a review we can say based on product and our ML model whether that review is positive or negative, also the user is spammer or not.

12. Contribution of team members

"The best teamwork comes from members who are working independently toward one goal in unison."
-James Cash Penney.

Every task was assigned, shared and peer reviewed by each of our team members. Every team member is well adept with all the things that are done into this project which served our common goal.

13. Problems Encountered

1. When we used case to read in data into RDD, we were not able to apply Actions on the RDDs such as collect, show etc.

2. When we tried using Stanford's exiting jar for sentiment analysis there was compatibility error with Data bricks Scala work book.

3. When we wrote our own transformer to collect sentiment scores we were unable to perform action on the created RDDs hence the solution that we broadcasted the values to perform the transform.

14. References:

https://spark.apache.org/docs/2.1.0/mllib-guide.html

https://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf

http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/6010/zip/imm6010.zip