**DISTRIBUTED TWEETS CLASSIFIER**

**By**

**Abhinaya Dasarathy**

**ABSTRACT**

Microblogging today has become a very popular communication tool among Internet users. Millions of users share opinions on different aspects of life every day. Therefore, microblogging web-sites are rich sources of data for opinion mining and sentiment analysis. Because microblogging has appeared relatively recently, there are a few research works that were devoted to this topic.

Classification algorithms can be used to automatically classify documents, images, implement spam filters and in many other domains. The tweets can be classified using various algorithms like SGD, SVM, Neural Network, Random Forests. The main objective of this project is to categorize the tweets got from twitter by distributing the classification on the Hadoop cluster. Classification is done using Naïve Bayes Algorithm. It can make the classification faster if there is a huge number of tweets to classify.

**BACKGROUND AND MOTIVATION**

Millions of messages are appearing daily in popular web-sites that provide services for microblogging such as Twitter, Tumblr, Facebook. Authors of those messages write about their life, share opinions on variety of topics and discuss current issues. Because of a free format of messages and an easy accessibility of microblogging platforms, Internet users tend to shift from traditional communication tools (such as traditional blogs or mailing lists) to microblogging services. As more and more users post about products and services they use, or express their political and religious views, microblogging web-sites become valuable sources of people’s opinions and sentiments. Such data can be efficiently used for marketing or social studies. We use a dataset formed of collected messages from Twitter. Twitter contains a very large number of very short messages created by the users of this microblogging platform. The contents of the messages vary from personal thoughts to public statements. Table 1 shows examples of typical posts from Twitter. As the audience of microblogging platforms and services grows every day, data from these sources can be used in opinion mining and sentiment analysis tasks. For example, manufacturing companies may be interested in the following questions:

What do people think about our product (service, company etc.)?

How positive (or negative) are people about our product?

What would people prefer our product to be like?

Political parties may be interested to know if people support their program or not. Social organizations may ask people’s opinion on current debates. All this information can be obtained from microblogging services, as their users post everyday what they like/dislike, and their opinions on many aspects of their life. In our paper, we study how microblogging can be used for sentiment analysis purposes. We show how to use Twitter as a corpus for sentiment analysis and opinion mining. We use microblogging and more particularly Twitter for the following reasons:

• Microblogging platforms are used by different people to express their opinion about different topics, thus it is a valuable source of people’s opinions.

• Twitter contains an enormous number of text posts and it grows every day. The collected corpus can be arbitrarily large.

• Twitter’s audience varies from regular users to celebrities, company representatives, politicians4 , and even country presidents. Therefore, it is possible to collect text posts of users from different social and interests groups.

• Twitter’s audience is represented by users from many countries. Although users from U.S. are prevailing, it is possible to collect data in different languages.

funkeybrewster: @redeyechicago I think Obama’s visit might’ve sealed the victory

for Chicago. Hopefully the games mean good things for the city.

vcurve: I like how Google celebrates little things like this: Google.co.jp honors Confucius

Birthday — Japan Probe

mattfellows: Hai world. I hate faulty hardware on remote systems where politics

prevents you from moving software to less faulty systems.

brroooklyn: I love the sound my iPod makes when I shake to shuffle it. Boo bee boo

MeganWilloughby: Such a Disney buff. Just found out about the new Alice in Wonderland

movie. Official trailer: http://bit.ly/131Js0 I love the Cheshire Cat.

Table 1: Examples of Twitter posts with expressed users’ opinions

In this project, we will limit the tweets to deals by getting the tweets containing the hashtags #deal, #deals and #discount. We will classify them in the following categories:

apparel (clothes, shoes, watches, …)

art (Book, DVD, Music, …)

camera

event (travel, concert, …)

health (beauty, spa, …)

home (kitchen, furniture, garden, …)

tech (computer, laptop, tablet, …)

From this resulting set, we can find out the deals found in that particular category. Similarly, this can be extended to find out any tweet (say a celebrity’s tweet) and find what the tweet was about.

**PROJECT DESCRIPTION**

**OVERVIEW:**

The complete project is divided into two phases. In the first phase, we would be getting the required data set of the tweets using Twitter API and store them in HDFS, perform classification of tweets using Naïve Bayes Algorithm and Map Reduce paradigm. Later in the second phase, we distribute the classification to multiple nodes in Hadoop to enhance the performance when there are a large number of tweets to classify.

**NAÏVE BAYES CLASSIFICATION USING APACHE MAHOUT:**

Apache Mahout could be used to group documents into clusters of similar subject area. Mahout also includes a number of classification algorithms that can be used to assign category labels to text documents. One algorithm that Mahout provides is the Naive Bayes algorithm. This algorithm is used for a wide variety of classification problems and is as an excellent introduction into probabilistic classification. In order to perform class assignments, the algorithms that employ probabilistic classification techniques build a model based on the probability with which document features appear for a given class.

Naive Bayes is an algorithm that can be used to classify objects into usually binary categories. It is one of the most common learning algorithms in spam filters. Despite its simplicity and rather naive assumptions it has proven to work surprisingly well in practice.

Before applying the algorithm, the objects to be classified need to be represented by numerical features. In the case of e-mail spam each feature might indicate whether some specific word is present or absent in the mail to classify. The algorithm comes in two phases: Learning and application. During learning, a set of feature vectors is given to the algorithm, each vector labeled with the class the object it represents, belongs to. From that it is deduced which combination of features appears with high probability in spam messages. Given this information, during application one can easily compute the probability of a new message being either spam or not.

The algorithm does make several assumptions, that are not true for most datasets, but make computations easier. The worst probably being, that all features of an objects are considered independent. In practice, that means, given the phrase "Statue of Liberty" was already found in a text, does not influence the probability of seeing the phrase "New York" as well.

**IMPLEMENTATION:**

**Preparing the training set:**

To prepare a training set, we fetched the tweets with the following hashtags: #deals, #deal or #discount by using the script twitter\_fetcher.py. It is using the python-tweepy 2.1 library. You need to have consumer keys/secrets and access token key/secrets to use the api. If you don’t have them, simply login on the twitter website then go to: <https://dev.twitter.com/apps>. Then create a new application.  
When you are done, you should see in the section ‘OAuth settings’, the Consumer Key and secret, and in the section ‘Your access token’, the Access Token and the Access Token secret.

Pseudocode to fetch tweets:

import tweepy

import sys

CONSUMER\_KEY='REPLACE\_CONSUMER\_KEY'

CONSUMER\_SECRET='REPLACE\_CONSUMER\_SECRET'

ACCESS\_TOKEN\_KEY='REPLACE\_ACCESS\_TOKEN\_KEY'

ACCESS\_TOKEN\_SECRET='REPLACE\_ACCESS\_TOKEN\_SECRET'

Auth <- set the access token key

hashtags <- 'deal', 'deals', 'discount'// the tweets with these hashtags are got

for tag in hashtags:

        for i in the range of page count

                results <- search api

                print length of results

                for each result value:

                        print result value

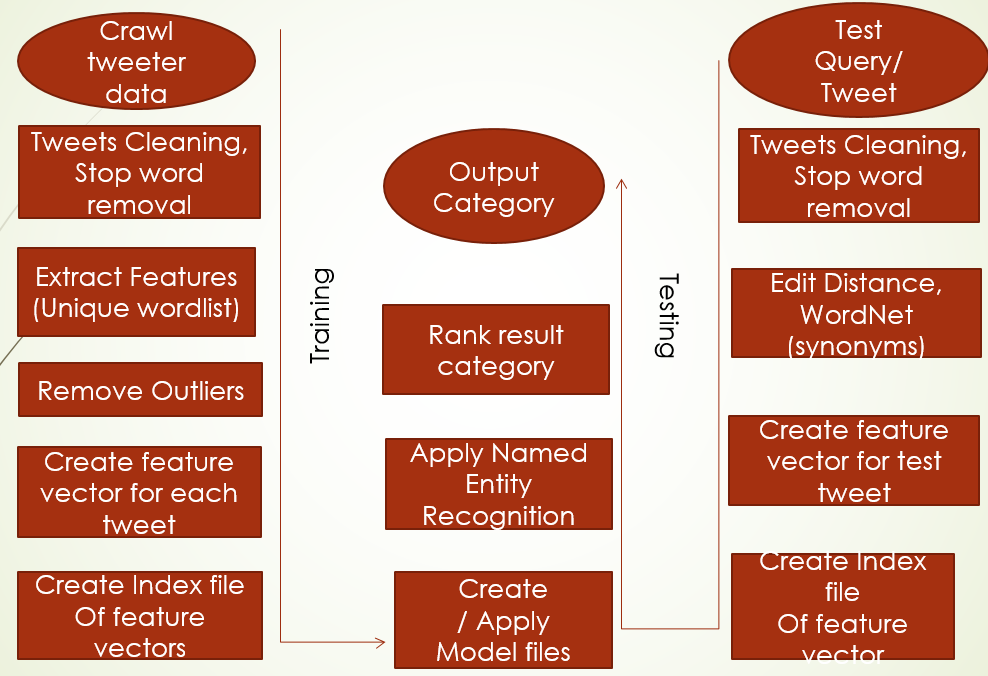
                        Only keep tweets pointing to a web page

The file tweets-train.tsv contains a list of tweets in a tab separated value format. The first number is the tweet id followed by the tweet message in the following format.

Training set can be prepared by adding the category of the tweet at the beginning of the line followed by a tab character.

Make sure to use tab between the category and the tweet id and between the tweet id and the tweet message.

For the classifier to work properly, this set must have at least 50 tweets messages in each category.



**TRAINING THE MODEL WITH MAHOUT:**

First, we need to convert the training set to the hadoop sequence file format:

$ java -cp target/twitter-naive-bayes-example-1.0-jar-with-dependencies.jar TwitterClassifier.TweetTSVToSeq data/tweets-train.tsv tweets-seq

The sequence file has as key: /[category]/ and as value.

Pseudocode to convert tweet tsv to sequence file:

        inputFileName <- arg[0] given as input

        outputDirName <- args[1] given aa input

        Initialise file system

Read line by line from the tsv file

Tokens[] <- Split the line based on space

  if (tokens.length != 3)

                skip line

        String category <- tokens[0];

        String id <- tokens[1];

          String message <- tokens[2];

            Key <- "/" + category + "/" + id;

            Value<- message;

            Write (key, value)

Then we upload this file to HDFS-

**$ hadoop fs -put tweets-seq tweets-seq**

We can run mahout to transform the training sets into vectors using [tfidf weights](http://en.wikipedia.org/wiki/Tf%E2%80%93idf)(term frequency x document frequency):

**$ mahout seq2sparse -i tweets-seq -o tweets-vectors**

It will generate the following files in HDFS in the directory tweets-vectors:

df-count: sequence file with association word id => number of document containing this word

dictionary.file-0: sequence file with association word => word id

frequency.file-0: sequence file with association word id => word count

tf-vectors: sequence file with the term frequency for each document

tfidf-vectors: sequence file with association document id => tfidf weight for each word in the document

tokenized-documents: sequence file with association document id => list of words

wordcount: sequence file with association word => word count

In order to do the training and check that the classification works fine, Mahout splits the set into two sets: a training set and a testing set:

**$ mahout split -i tweets-vectors/tfidf-vectors --trainingOutput train-vectors --testOutput test-vectors --randomSelectionPct 40 --overwrite --sequenceFiles -xm sequential**

We use the training set to train the classifier:

**$ mahout trainnb -i train-vectors -el -li labelindex -o model -ow -c**

If the percentage of correctly classified instance is too low, you might need to improve your training set by adding more tweets or by changing your categories to not have too many similar categories or by removing categories that are used very rarely. After you are done with your changes, you would need to restart the training process.

To use the classifier to classify new documents, we would need to copy several files from HDFS:

model (matrix word id x label id)

labelindex (mapping between a label and its id)

dictionary.file-0 (mapping between a word and its id)

df-count (document frequency: number of documents each word is appearing in)

**$ hadoop fs -get labelindex labelindex**

**$ hadoop fs -get model model**

**$ hadoop fs -get tweets-vectors/dictionary.file-0 dictionary.file-0**

**$ hadoop fs -getmerge tweets-vectors/df-count df-count**

To get some new tweets to classify, you can run the twitter fetcher again

**$ python scripts/twitter\_fetcher.py 1 > tweets-to-classify.tsv**

Now we can run the classifier on this file.

Pseudo Code to classify the tweets using the model and the dictionary file:

        Define the NaiveBayesModel model using the library

        Define the StandardNaiveBayesClassifier  library

        Define the analyzer used to extract word from tweet

        int labelCount = labels.size(); % labels of categories

        int documentCount <- document Frequency %number of documents in training set

Read the tweets path

        Read each line in the tweets file

        String[] tokens <- split the line

        String tweetId <- tokens[0];

        String tweet <- tokens[1];

Multiset words = ConcurrentHashMultiset.create();

Extract words from tweet

        if the word is not in the dictionary, skip it

Else

        create vector wordId => weight using tfidf

        With the classifier, we get one score for each label

        The label with the highest score is the one the tweet is more likely to be associated to

Most of the tweets are classified properly but some are not. For example, the tweet “J and R – Roku 3 Streaming Player 4200R $89.99” is incorrectly classified as camera. To fix that, we can add this tweet to the training set and classify it as tech. You can do the same for the other tweets which are incorrectly classified. When you are done, you can repeat the training process and check the results again.

**Analytical Example:**

**View words which are the most representative of each category:**

The code is in topCategoryWords.java file:

Output:

$ java -cp target/twitter-naive-bayes-example-1.0-jar-with-dependencies.jar TopCategoryWords model labelindex dictionary.file-0 df-count

Top 10 words for label camera  
– digital: 70.05728101730347  
– camera: 63.875202655792236  
– canon: 53.79892921447754  
– mp: 49.64586567878723  
– nikon: 47.830992698669434  
– slr: 45.931694984436035  
– sony: 44.55785942077637  
– lt: 37.998433113098145  
– http: 29.718397855758667  
– t.co: 29.65730857849121

Top 10 words for label event  
– http: 33.16791915893555  
– t.co: 33.09973907470703  
– deals: 26.246684789657593  
– days: 25.533835887908936  
– hotel: 22.658542156219482  
– discount: 19.89004611968994  
– amp: 19.645113945007324  
– spend: 18.805208206176758  
– suite: 17.21832275390625  
– deal: 16.84959626197815

**PHASE 2: Distributed Classification**

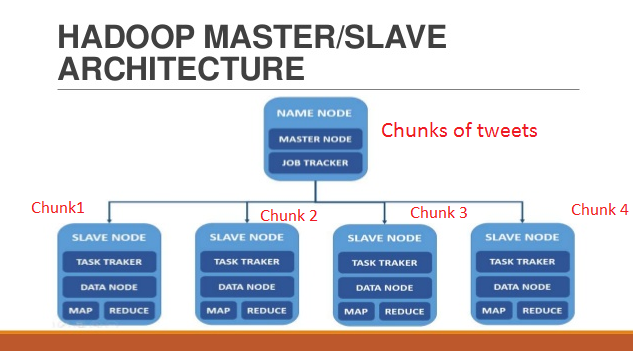
In phase 2, we have distributing the classification over multiple Hadoop nodes to improve the performance when considering large number of tweets. We are trying to minimize the number of times we load the naive bayes model in memory.

When the input file is split into chunk. It is distributed on all the nodes of the hadoop cluster. Each node will handle some chunks. For each of those chunk, Hadoop spawns a new jvm(process) instead of reusing an existing one.

To distribute the classification on the hadoop nodes, we are going to define a mapreduce job:

* the csv containing the tweets to classify is split into several chunks
* each chunk is sent to the hadoop node that will process it by running the map class
* the map class loads the naive Bayes model and some document/word frequency into memory
* for each tweet of the chunk, it computes the best matching category. The result is written in the output file. We are not using a reducer class as we don’t need to do aggregations.

**IMPLEMENTATION:**



**Architecture**

* Input Available:

- Tweets file having tweet id and tweet message

- Document frequency file (from the training using Mahout)

- Dictionary (from the training using Mahout)

* Mapper class:

- Splits each line in the Tweets csv file as Tweet id and Tweet message (tokens)

- Sends these tweet messages to the Classifier (classifier.java) to classify these messages into categories

- Finds the best matched category for the tweet.

- Writes in the output file

(key, value) – (tweet id, category)

* No reducer class

In the jvm, we can put the data model somewhere in the memory and the subsequent tasks that uses the same JVM can use the model without having to reload it again. It is done by using a static attribute (see method MapReduceClassifier.initClassifier).

If you have run the commands in the previous post, you should have the following files in HDFS:

**tweets-vectors/dictionary.file-0**

**tweets-vectors/df-count/part-r-00000**

We would need to copy the file data/tweets-to-classify.tsv to HDFS so it can be read by the hadoop job:

**$ hadoop fs -put data/tweets-to-classify.tsv tweets-to-classify.tsv**

To run the mapreduce job:

**$ hadoop jar target/mahout-naive-bayes-example2-1.0-jar-with-dependencies.jar model tweets-vectors/dictionary.file-0 tweets-vectors/df-count/part-r-00000 tweets-to-classify.tsv tweet-category**

After it is done, we can copy the result from HDFS to the local filesystem:

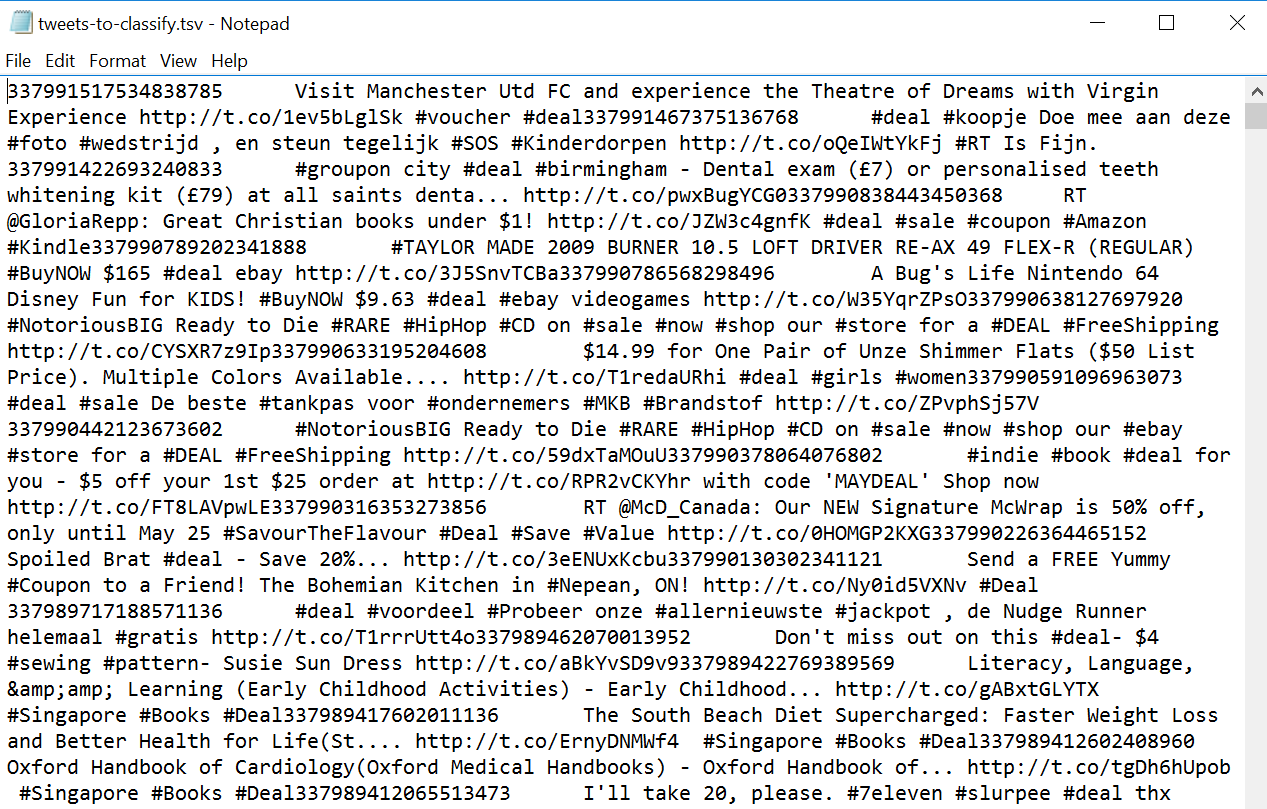
**$ hadoop fs -getmerge tweet-category tweet-category.tsv**

Now we can see the results by using the ResultReader class.

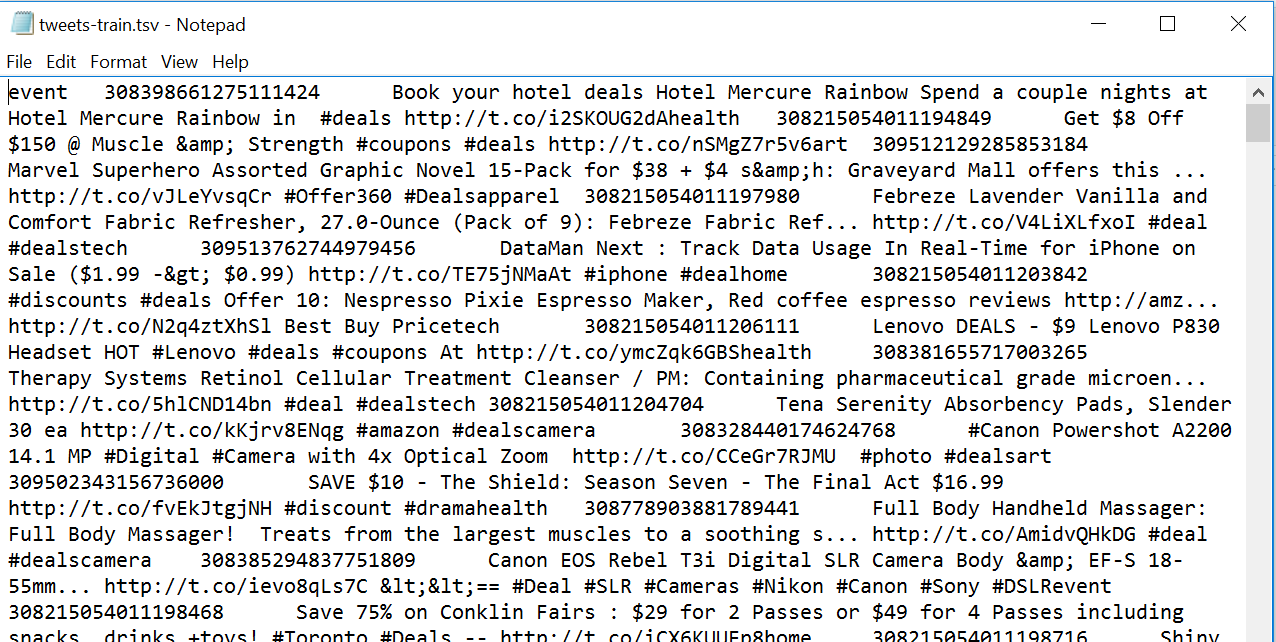
**RESULTS:**

Distributed classification:

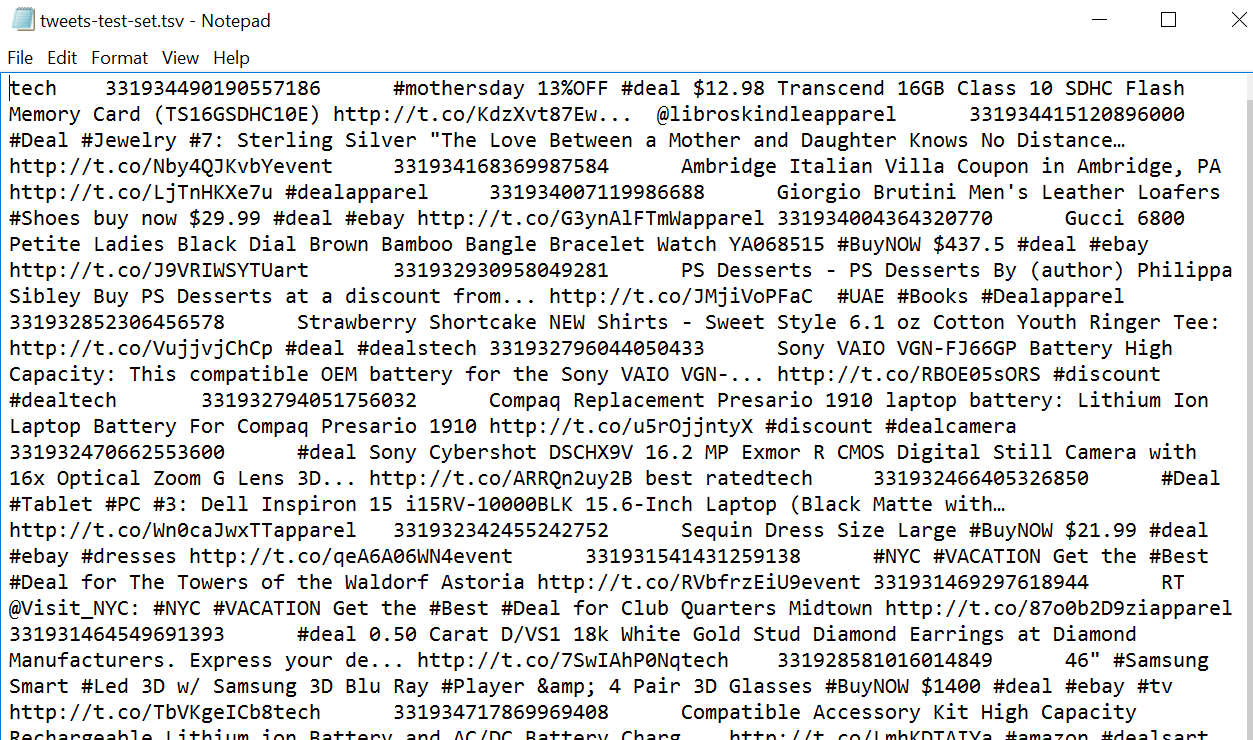
Tweets\_to classify.tsv:



Tweets-train.tsv:

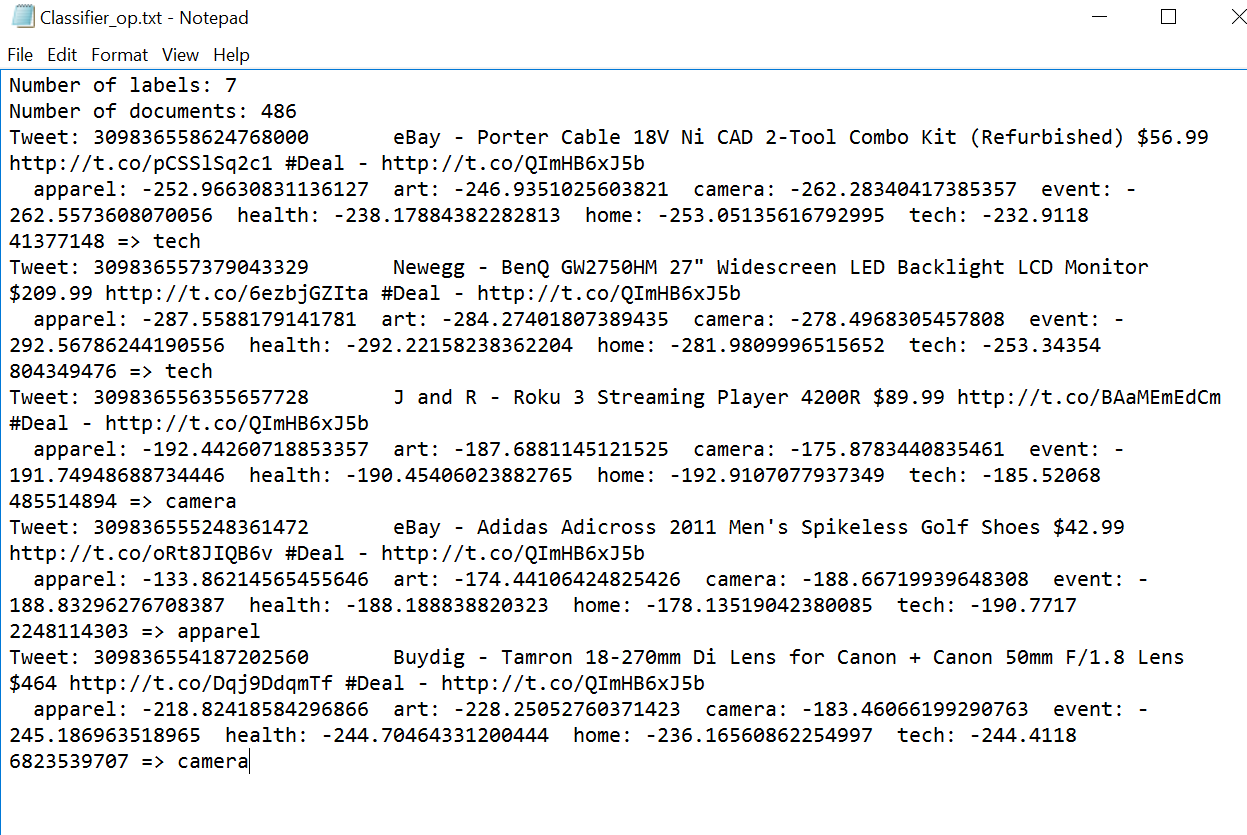


Tweets-test-set.tsv:

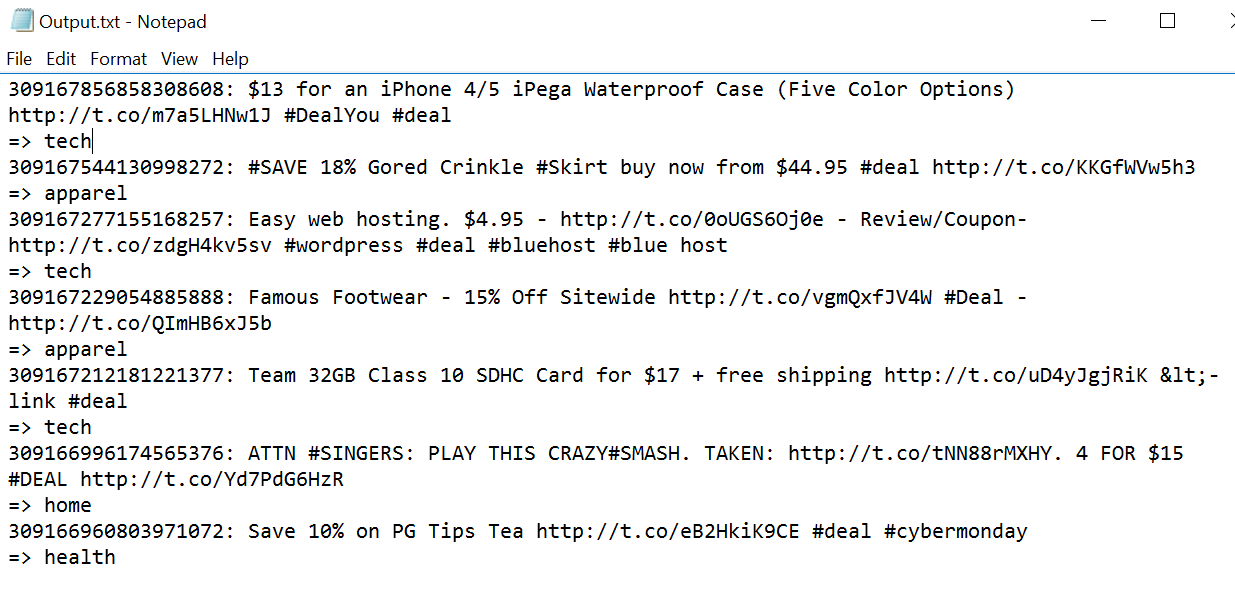


Output for classifier:

Single Node:



Distributed Classifier Output:



**SCOPE FOR FUTURE WORK:**

This project was aimed at just getting the categories in which any deals are tweeted. However, twitter is just one example. This distributed classification can be applied to images and videos which make this area more challenging. Image classification still exists in distributed lezrning. However, categorizing videos will pose an interesting challenge to first figure out what the video is all about and what category can it fall on.

**CONCLUSION:**

In this project, distributed classification of tweets using naïve Bayes classifier and Hadoop is performed. We also have described how to speed up the execution of the job by minimizing the number of time the model is loaded into memory by using the Hadoop property to reuse the same jvm and by storing the data in memory using a static variable. It can make the classification faster if there is a huge number of tweets to classify.

REFERENCES

[1] <https://mahout.apache.org/users/classification/bayesian.html>

[2] <https://pdfs.semanticscholar.org/ad8a/7f620a57478ff70045f97abc7aec9687ccbd.pdf>

[3] <http://www.tmrfindia.org/ijcsa/v9i15.pdf>

[4] <http://mahout.apache.org/>

[5]<https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/CommandsManual.html>

[6] Libraries referenced: Apache mahout libraries jar file - <https://github.com/apache/mahout>

[7] Twitter API - <https://dev.twitter.com/rest/public>