**Luke Walsh**

**May 11, 2016**

## PySpark Streaming and NLTK Text Processing:

## Naïve Bayes Classification for Twitter Sentiment Analysis

**Problem/Purpose:** Build a PySpark Streaming application using Python’s NLTK library that analyzes the sentiment of Tweets about a given topic/search term in near-real time.

**Big Data Set:**

Training data set:

-“Twitter Sentiment Analysis Dataset”

-~1.5 million Tweets already classified as positive or negative

-source: <http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/>

-format: csv

Classification data:

-Twitter live feed (streaming)

-resource URL: https://stream.twitter.com/1.1/statuses/filter.json

-returns specified number of Tweets for every specified time interval

-format: returns JSON

**Hardware:**

* CentOS 6.7 VM running on Windows 10 VMWare Workstation 11.0

**Software:**

|  |  |
| --- | --- |
| **Technology/tools** | **Description** |
| Hadoop 2.6.0-cdh5.7.0 | Distributed file system with 1 live data node (pseudo-distributed environment) |
| Spark version 1.6.0 | Cluster computing software; this app will use PySpark Streaming to analyze streaming data |
| Python version 3.3.3 | Language used to develop the app |
| Python NLTK library | Natural Language Toolkit Library – a Python package that has useful tools for natural language processing such as the Naïve Bayes classifier this app uses |
| D3.js | JavaScript library used for visualizing results |

**YouTube URLs here:**

* 2 min: <https://youtu.be/_DY7L50deqo>
* 15 min: <https://youtu.be/ARP5q4LiMSc>

**Overview of steps:**

1. Train a model (Naïve Bayes) that will classify Tweets as positive or negative

2. Set up Twitter stream and format incoming tweets to run through the model

3. Run each batch of Tweets through the classifier and capture results

4. Visualize

**Summary:**

This application aims to classify the sentiment of Tweets (positive/negative) in near-real time (i.e. batch processing) using PySpark and Python’s Natural Language ToolKit library. The application connects to Twitter’s live data feed and collects batches of Tweets in a time interval specified by the user. A Naïve Bayes classification model (from Python’s NLTK package) is used on these batches of Tweets in the PySpark environment to classify the Tweets as either positive or negative. The total number of positive and negative Tweets for each batch is counted, and D3.js is used to visualize these proportions for each batch over time.

Below are a few brief notes on the pros/cons of using this technology.

Pros:

* Spark has very fast batch processing compared to something like Hadoop MapReduce due to in memory computations
* Spark is well suited for streaming applications and is highly scalable
* Python NLTK library has many built-in tools to assist in NLP, and the Naïve Bayes classification model which is often used in text classification is built into the NLTK package
* Python’s ‘requests’ library makes it easy to set up a REST client to connect to streaming sources like Twitter’s data feed
* Overall, PySpark Streaming and NLTK are very useful when used together

Cons:

* No major concerns raised by this demo project
* Some online research suggests that, since Spark is built on Java/Scala, there are some operations that cannot be performed by Python and need to be done in Java or Scala. In this project I was able to accomplish everything in Python though.
* Scala and Java may potentially perform faster than Python. May depend on the application.

**Setup/Configuration**

Setup and configuration is fairly straightforward, assuming Hadoop and PySpark are already installed and running.

Here are the requirements (most of these tools should already be installed from previous class assignments):

* Running Hadoop cluster or pseudo-cluster
  + <https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html>
* PySpark
  + <https://spark.apache.org/docs/0.9.0/python-programming-guide.html>
* The following Python libraries. If you do not have these libraries you can use the ‘pip install {{packagename}}’ in order to download and install the library.
  + More info on using pip: <http://python-packaging-user-guide.readthedocs.io/en/latest/installing/>
  + After installation, the packages will need to be imported into the Python script using ‘import {{packagename}}’ (see code for examples)
  + List of packages needed (some come with Python; for these we just need an import statement in the script):
    - sys
    - ast
    - json
    - pyspark and pyspark.streaming
    - operator
    - csv
    - numpy
    - requests
    - string
    - time
    - nltk

**Discussion of Steps**

1. Train the classifier model

We will use a Naïve Bayes model to classify the tweets. This type of model is common in text classification and features are extracted based on the most common words in the text. The “Naïve” term comes from the fact that the model assumes independence between features (in this case words), which isn’t entirely valid especially for linguistics in which words have dependencies within sentences and paragraphs. However, a Naïve Bayes classifier is considered a common and useful tool for classifying documents.

First we need to use a training dataset to train the model. For this project we used a publicly available dataset of about 1.5 million tweets that are pre-classified as positive or negative. We’ll read in the text file and extract the tweets and sentiment (1 for positive; 0 for negative) from the dataset:

#read in the text file and create a dataset in memory

train\_file = sc.textFile("hdfs://localhost:8020/user/cloudera/final/Sentiment Analysis Dataset.csv")

train\_header = train\_file.take(1)[0]

train\_data\_raw = train\_file.filter(lambda line: line != train\_header)

#define a function that splits apart the rows for each line and

#creates a tuple and removes punctuation

def get\_row(line):

row = line.split(',')

sentiment = row[1]

tweet = row[3].strip()

translator = str.maketrans({key: None for key in string.punctuation})

tweet = tweet.translate(translator)

tweet = tweet.split(' ')

tweet\_lower = []

for word in tweet:

tweet\_lower.append(word.lower())

return (tweet\_lower, sentiment)

#call the function on each row in the dataset

train\_data = train\_data\_raw.map(lambda line: get\_row(line))

#create a SentimentAnalyzer object

sentim\_analyzer = SentimentAnalyzer()

Since we are using common words as features in the model, we want to remove stop words such as ‘is’, ‘the’, etc since they don’t really have much meaning. We will include stop words and their negated meaning since the model takes into account negation. So for example ‘is’ and ‘is\_NEG’ (\_NEG is the negation flag) will both be considered stop words and won’t be included when features are extracted for the model:

#get list of stopwords (with \_NEG) to use as a filter

stopwords\_all = []

for word in stopwords.words('english'):

stopwords\_all.append(word)

stopwords\_all.append(word + '\_NEG')

#take 10,000 Tweets from this training dataset for this example and get all the words

#that are not stop words

train\_data\_sample = train\_data.take(10000)

all\_words\_neg = sentim\_analyzer.all\_words([mark\_negation(doc) for doc in train\_data\_sample])

all\_words\_neg\_nostops = [x for x in all\_words\_neg if x not in stopwords\_all]

Next we’ll extract the 200 most common words from the set of text and use them as features in the model to predict sentiment. We need to create unigrams:

#create unigram features and extract features

unigram\_feats = sentim\_analyzer.unigram\_word\_feats(all\_words\_neg\_nostops, top\_n=200)

sentim\_analyzer.add\_feat\_extractor(extract\_unigram\_feats, unigrams=unigram\_feats)

training\_set = sentim\_analyzer.apply\_features(train\_data\_sample)

Finally we can train the model and run a few test sentences:

#train the model

trainer = NaiveBayesClassifier.train

classifier = sentim\_analyzer.train(trainer, training\_set)

#classify test sentences

test\_sentence1 = [(['this', 'program', 'is', 'bad'], '')]

test\_sentence2 = [(['tough', 'day', 'at', 'work', 'today'], '')]

test\_sentence3 = [(['good', 'wonderful', 'amazing', 'awesome'], '')]

test\_set = sentim\_analyzer.apply\_features(test\_sentence1)

test\_set2 = sentim\_analyzer.apply\_features(test\_sentence2)

test\_set3 = sentim\_analyzer.apply\_features(test\_sentence3)

classifier.classify(test\_set[0][0]) #output is ‘0’

classifier.classify(test\_set2[0][0]) #output is ‘0’

classifier.classify(test\_set3[0][0]) #output is ‘1’

Now we have a fully trained Naïve Bayes classification model and we can run text through it, in our case Tweets, and the model will classify it as positive or negative. Next we need to set up the Twitter stream.

First we have to create a Twitter developer account since we need to be authenticated in order to connect to Twitter’s stream. Authentication is fairly straightforward and can be done following the instructions at this link: <https://dev.twitter.com/oauth/overview> .

Once we have all the keys/tokens/secrets (there should be 4 total), we can create a Twitter authorization using Python’s requests library and define a function that will make a GET request on Twitter’s live data feed resource:

#set up Twitter authentication (these keys are specific to this application –normally

#these would NOT be shared)

key = "apaopGZ2zvfnQPUEu4Dm6OhSs"

secret = "sYTenLWQaUxAHlZshizX8ERbjmtvlMvCwUxM9Z1m1prTIrSGNl"

token = "709905344026320896-s4U8M6rCMDz4CqMRMV2CwBJu8KFKfZG"

token\_secret = "Jg8WCL0AZFszLXynsDXOSMcHlynKYThGh4UO8nSu1Kokh"

#specify the URL and a search term

search\_term='Trump'

sample\_url = 'https://stream.twitter.com/1.1/statuses/sample.json'

filter\_url = 'https://stream.twitter.com/1.1/statuses/filter.json?track='+search\_term

#’auth’ represents the authorization that will be passed to Twitter

auth = requests\_oauthlib.OAuth1(key, secret, token, token\_secret)

# Setup Stream

rdd = ssc.sparkContext.parallelize([0])

stream = ssc.queueStream([], default=rdd)

#define a function that makes a GET request to the Twitter resource and returns a #specified number of Tweets (blocksize)

def tfunc(t, rdd):

return rdd.flatMap(lambda x: stream\_twitter\_data())

def stream\_twitter\_data():

response = requests.get(filter\_url, auth=auth, stream=True)

print(filter\_url, response)

count = 0

for line in response.iter\_lines():

try:

if count > BLOCKSIZE:

break

post = json.loads(line.decode('utf-8'))

contents = [post['text']]

count += 1

yield str(contents)

except:

result = False

stream = stream.transform(tfunc)

coord\_stream = stream.map(lambda line: ast.literal\_eval(line))

Once the connection is set up, we can define a few functions that will run each batch of Tweets through the model and then save the output results:

#classify incoming tweets by applying the features of the model to each tweet

def classify\_tweet(tweet):

sentence = [(tweet, '')]

test\_set = sentim\_analyzer.apply\_features(sentence)

print(tweet, classifier.classify(test\_set[0][0]))

return(tweet, classifier.classify(test\_set[0][0]))

def get\_tweet\_text(rdd):

for line in rdd:

tweet = line.strip()

translator = str.maketrans({key: None for key in string.punctuation})

tweet = tweet.translate(translator)

tweet = tweet.split(' ')

tweet\_lower = []

for word in tweet:

tweet\_lower.append(word.lower())

return(classify\_tweet(tweet\_lower))

results = []

#save the results of the batch of Tweets along with a timestamp

def output\_rdd(rdd):

global results

pairs = rdd.map(lambda x: (get\_tweet\_text(x)[1],1))

counts = pairs.reduceByKey(add)

output = []

for count in counts.collect():

output.append(count)

result = [time.strftime("%I:%M:%S"), output]

results.append(result)

print(result)

#Call the above functions for each RDD in the stream’s batch

coord\_stream.foreachRDD(lambda t, rdd: output\_rdd(rdd))

Everything is now setup- now we just need to start the streaming context which will then start collecting Tweets in the specified batch time interval. When we want to see the results we can save the results to a csv file:

# Start streaming

ssc.start()

ssc.awaitTermination()

cont = True

while cont:

if len(results) >10:

cont = False

ssc.stop()

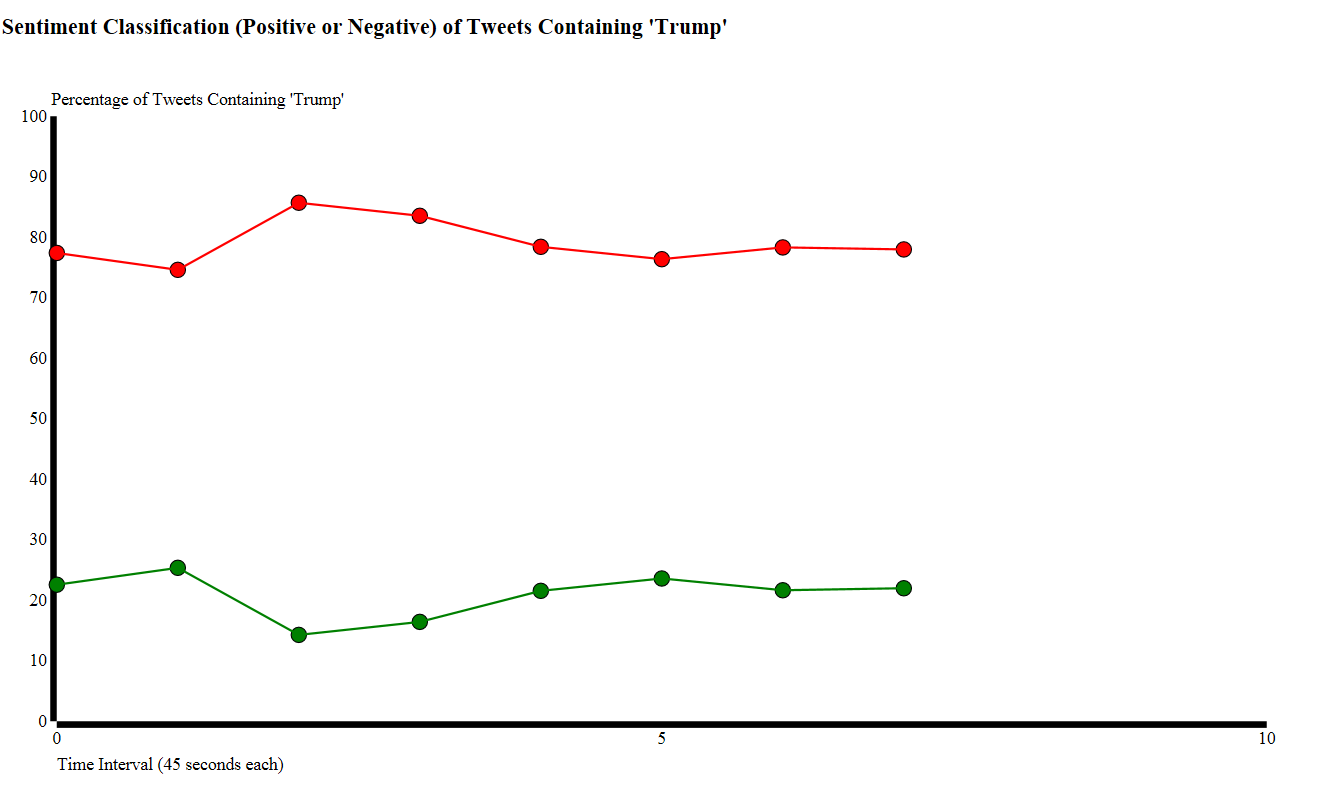
#save results

csvfile = '/user/cloudera/final/r'+time.strftime("%I%M%S")+search\_term

results\_rdd = sc.parallelize(results)

results\_rdd.saveAsTextFile(csvfile)

Now, we can visualize using D3.js. This project creates a simple graph showing the trend of positive vs negative Tweets over time using a similar approach to other D3.js visualizations from previous assignments in this course. Please see the commented code for the example. Here is the output:

****

There are other interesting ideas for visualizations that can be found online, such as using the Tweets’ location data to plot the Tweet origin on a map.

**References:**

The following websites were used for guidance on coding or information about the Naïve Bayes model:

* Twitter training dataset: <http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/>
* Documentation on Twitter’s streaming APIs: <https://dev.twitter.com/streaming/overview>
* Example code on how to connect to Twitter’s API’s: <http://will-farmer.com/twitter-civil-unrest-analysis-with-apache-spark.html>
* NLTK’s documentation on the Naïve Bayes classifier: <http://www.nltk.org/howto/sentiment.html>

**Thanks for reading!**