DeZyre Hadoop Capstone

Sentiment Analysis in Twitter

Introduction

I was inspired by this summary of a study that found sentiment analysis in Twitter to be a powerful predictor of deaths caused by heart disease. My goal is to create a Hadoop-based analysis that compares the sentiments of tweets. This project serves as the capstone to DeZyre's Certificate in Big Data and Hadoop. I used Cloudera's Quickstart VMs to create my Hadoop cluster (specifically, CDH 5.3.x for Virtual Box). I first used Apache Flume to capture tweets from the Twitter API. Next, I used Hive to transform the data from JSON. I then performed sentiment analysis via a MapReduce job written in Java. My final analysis was done with Hive.

Data Acquisition

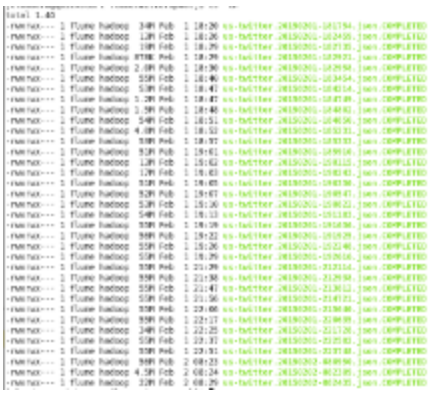
Failed Attempt: Flume's Twitter Firehose Source

I followed this tutorial to ingest the Twitter Streaming API via Apache Flume. The Flume agent is configured to use the 'twitter' source, a memory channel, and an HDFS sink. The tutorial is 51 weeks old, and reported ingesting 1.3 GB in a day. Using the same method, I ingested 16.6 GB in 2 hours. Unfortunately, the data did not come across in a usable format – there were a lot of binary characters and the JSON was not well-formed. I abandoned this ingestion approach due to these data issues.

Successful Attempt: Python's Tweepy Library and Flume's SpoolDir

I followed these examples to access the Twitter API via the Tweepy Python library. I used the same files, slistener.py and streaming.py. I added some more error handling to the former. For the latter, I removed the filter on tracking terms, and instead filtered on streams from the United States. When running this script, I still had unexpected errors that caused crashes. To handle these errors, I wrapped the run command in run.sh, which would restart the program in these cases. I let the program run for a few hours, until I collected 1 million lines, which took up 1.4 GB.

I configured a Flume agent to use SpoolDir as a source, a memory channel, and an HDFS sink (see configuration file flume-twitter.properties). On local disk, I had 35 files ranging from less than 1MB to 56MB in size. Since Hadoop is inefficient with many small files, I configured my Flume agent to roll the files up into files of 250MB in size. In practice (i.e., not a sandbox project), I would roll these up into larger files.



```
[cloudera@quickstart flumeTwitterSpool]$ hadoop fs -ls -h /twitter/raw
Found 6 items
-rw-r--r--   1 flume supergroup    238.6 M 2015-02-02 14:25 /twitter/raw/us-twitter.1422915861709.json
-rw-r--r--   1 flume supergroup    238.6 M 2015-02-02 14:26 /twitter/raw/us-twitter.1422915861710.json
-rw-r--r--   1 flume supergroup    238.6 M 2015-02-02 14:26 /twitter/raw/us-twitter.1422915861711.json
-rw-r--r--   1 flume supergroup    238.6 M 2015-02-02 14:27 /twitter/raw/us-twitter.1422915861712.json
-rw-r--r--   1 flume supergroup    238.6 M 2015-02-02 14:28 /twitter/raw/us-twitter.1422915861713.json
-rw-r--r--   1 flume supergroup    142.3 M 2015-02-02 15:27 /twitter/raw/us-twitter.1422915861714.json
[cloudera@quickstart flumeTwitterSpool]$
```

Data Cleaning

Looking at my files in HDFS, I saw that there were blank lines between every tweet. I used Hadoop Streaming and sed to strip out these lines (see commands.txt). This means that I have about half as many tweets as I originally thought, so approximately 500,000.

Data Inspection

I put a Hive table on the data so that I could see the value ranges for fields and look at the data in aggregate. These command are given in commands.txt. The external Hive table used a JSON serde. Following are some counts of interest:

- 494,683 tweets
- 139,244 reply tweets
- 53.7 characters per tweet, on average
- 452,881 tweets with geo-coordinates
- 474,706 tweets with US country code

I then created another hive table containing only the tweets with US country code and with geo-coordinates. 428,588 tweets met this criteria. This table's underlying HDFS file will be my input to my sentiment analysis. Seeing that I had a sizeable number of reply tweets, I decided to investigate whether reply tweets are significantly happier than non-reply tweets.

Sentiment Analysis in MapReduce

Obtaining a Happiness Lexicon

I used a sentiment lexicon from (Mohammad, Kiritchenko, Zhu, 2013). Specifically, I used the unigram lexicon called Sentiment140-Lexicon (v0.1). Sentiment140 Lexicon v0.1. I modified this lexicon by removing twitter handles, urls, and terms with non-ascii characters. I then loaded this into HDFS via the put command. My modified Lexicon has in 43,968 entries, of which 24,507 are positive and 19,461 are negative. The positive unigrams have a mean score of 0.92 while the negatives have a mean score of -0.98. These statistics were generated using Hive.

Running a MapReduce Job

I wrote a single MapReduce Java program to perform sentiment analysis. The unigram lexicon mentioned above is loaded into memory in a configuration step. The map step reads the file mentioned in the section "Data Inspection." Some standardization is performed on the tweet (e.g., switching to lower case, removing twitter handles) to better match the lexicon. I accumulate the sentiment score for each word in the tweet, and append this to the record. There is no Reduce component to this job. See SentimentScore.java for implementation details.

I decided to write a job that appends the sentiment score (instead of doing my full analysis in this single MapReduce job) so that I would have an intermediate file that servers as a starting point for other analyses outside of this capstone.

Sentiment Scores for Reply Tweets

I created a Hive table with a location pointing to the output of my MapReduce job. My results are given in the following table:

Tweet Type	Number of Tweets	Mean Sentiment Score	Std Deviation Sentiment Score
Reply	114,217	-0.0383	1.66
Non-Reply	314,371	-0.406	1.82

It seems possible that reply tweets have a higher sentiment score. Conclusions should not be drawn without more formal statistical analysis.