Jasen Jones, Miki Seltzer, Eric Whyne
W205 Fall 2015

W205 Final Project Report

Background

According to a 2013 Twitter study, Twitter users are more likely than the general population to shop on both Black Friday and Cyber Monday (though the difference is much smaller on Black Friday). Twitter released another report in 2014 aimed at helping companies understand the shopping patterns of Twitter users on Black Friday and Cyber Monday, noting that Twitter users were about twice as likely to pursue electronics deals. Analyzing tweets, therefore, has the potential to reveal important comparisons and consumer sentiments on two of the largest sales days of the year.

Due to the major marketing investment many companies make for Black Friday and Cyber Monday, it's unsurprising that research already exists into what companies and products are the most popular among consumers. Several studies, such as this study into Kohl's Twitter metrics, take an in-depth look at tweets about an individual retailer, tracking sentiment and interest over time. A more robust example from 2014 aggregates twitter data from Black Friday and Cyber Monday and assesses the frequency of hashtags and company names, while also performing basic sentiment analysis. This kind of study provides an example of how we envision our final deliverable, essentially monitoring Twitter traffic and providing aggregated results.

Previous public research, however, seems to focus on Twitter traffic on the actual date of Black Friday and Cyber Monday rather than the preceding days and weeks. We see an opportunity to provide some unique analysis by monitoring Twitter traffic between and in advance of these dates. This may be particularly relevant as an increasing number of companies have deals leaked weeks in advance, which may propagate in some Twitter channels.

We planned to monitor Twitter traffic in the weeks leading up to the holiday kickoff, through the weekend and possibly even a short while after Cyber Monday. Our focus was on consumer electronics brands, since it is the second most purchased category of products on Black Friday (behind clothing). In particular, we focused on large brands like Samsung, Apple, Sony, Lenovo, Dell, Canon, Nikon, Microsoft, Vizio, etc, in addition to consumer electronics categories like laptop, dslr, tablet, camera, etc.

In order to categorize tweets, we planned to use a list of keywords or hashtags, and collect only tweets that are of interest to us. Given the massive volume of tweets that will inevitably be available during the holiday kickoff weekend, we were judicious about how we choose to filter our data, although we did not want to restrict our data too much.

Architecture

Our initial plan was to process the data using Hadoop batch. However, since we had a nice streaming data set, we decided to also invest time in setting up stream processing. Rather than point our streaming pipeline directly at the Twitter API feed, we decided to use Apache Kafka as a queue. Kafka gives us the option of using Balanced Consumers to stream processing directly, or using other streaming software such as Apache Storm or Spark Streaming which can read directly from Kafka Topics.

After experimenting with Python code for consuming Kafka topics, we discovered that only one library supported the concept of Balanced Consumer and we had to increase the number of topic partitions to support this, since consumers simply coordinate their efforts by allocating partitions to each other. This means that in theory we should be able to scale to an arbitrary volume and velocity of data. In practice, we suspect that the coordination done through ZooKeeper between balanced consumers is going to cause problems. We didn't get to the velocity necessary to experiment with this theory during this project.

At several parts of our workflow it was necessary to conduct batch operations against the data. Our workflow for this varied from Spark and Hadoop streaming jobs to GNU Parallel. When embarrassingly parallel jobs were required, GNU Parallel ended up having the lowest development time and analogous performance to other parallel distributed methods for these data sets. As part of our code submission we've included an example that demonstrates it's use on a directory of files.

We utilized S3 to store our batch daily tweet files which were then run through our sentiment analysis code. The sentiment analysis was developed by our team using Python. Once our data was run through the sentiment analysis, the resulting data sets were loaded into Hive tables. Within Hive, we were able to do high level aggregation on our data.

As a serving layer, we used Hiveserver to access our data using Tableau for visualization and interaction.

Data Gathering

As we experimented with different processing options we collected data to file. Our scripts can be viewed in the data_gathering folder of our GitHub repository. The script is fairly straight forward, tracking terms using Twitter's Streaming API and dumping JSON records to a file whose name includes the current ISO-8601 formatted date. In order to ensure continued execution, Supervisord is used. Supervisord recovers the program in case of a timeout due to connectivity, reboot due to power loss, or program error.

Our initial set of keywords resulted in about 25GB of data being collected per day. After removing overly broad keywords, we saw about 11GB of data on week days and 9GB of data on weekends. The final list of keywords can also be viewed in our GitHub repository.

Data Minimization/Cleaning

Our initial set of keywords did not limit tweets to containing the term “black friday” or “cyber monday.” Thus, we ended up with a data set that was approximately 140 GB in size. In order to hone our analysis more closely, we decided to focus on Black Friday. Thus, we ran a data minimization process locally to filter out tweets that did not contain “black friday” or “blackfriday” (case insensitive). We also experimented with doing the filtering using GNU parallelization for improved performance.

We also had to do a small amount of data cleaning for our data to work well with Hive. The text field of a tweet is flexible, allowing new line characters, unicode characters, and emojis. In particular, Hive did not work well with tweets containing a newline character, so our data cleaning process involved replacing all occurrences of ‘\n’ with spaces. This allowed Hive to work harmoniously with the tweet data.

Sentiment Analysis

We implemented some basic sentiment analysis and integrated the results into our existing data store. Our overall concept is to rate tweets as positive, negative, or neutral depending on the text contained in the tweet. To do so we modified an algorithm from http://fjavieralba.com/basic-sentiment-analysis-with-python.html to tokenize each tweet and count the number of positive and negative words contained in the tweet based on sentiment dictionaries from academic literature. This method is limited in scope as it cannot account for things like sarcasm or historical trends, but should give us a basic understanding of aggregate sentiment values toward a product. It is also limited in that our sentiment analysis isn’t designed to be parallelized and takes too long to be inserted into the stream. Our workarounds for this are described in the Data Analysis/Results section below, and is one of the things we’d need to focus on optimizing before using in other applications.

To score each tweet we tokenize the text and assign a descriptive tag for each token. “Good” would be tagged with “positive” while “terrible” would be tagged with “negative” for example. We have also accounted for preceding words that increase, decrease, or invert the meaning of the original sentiment. Increasers like “completely”, “so”, and “totally” double the score of the following word, while decreasers like “kinda”, “barely”, and “sorta” halve the following word’s score. An inversion word like “not”, “won’t”, and “didn’t” plays a different role, reversing the value of a given token. This means that while “good” is normally positive, “not good” should be negative. We are also considering using existing sentiment dictionaries that include a gradient positive/negative scale rather than binary good/bad designators, but at this point we simply wanted to show proof of concept. We ideally would like to incorporate the tagging algorithm at the point of data collection so it can more easily be used during future queries.

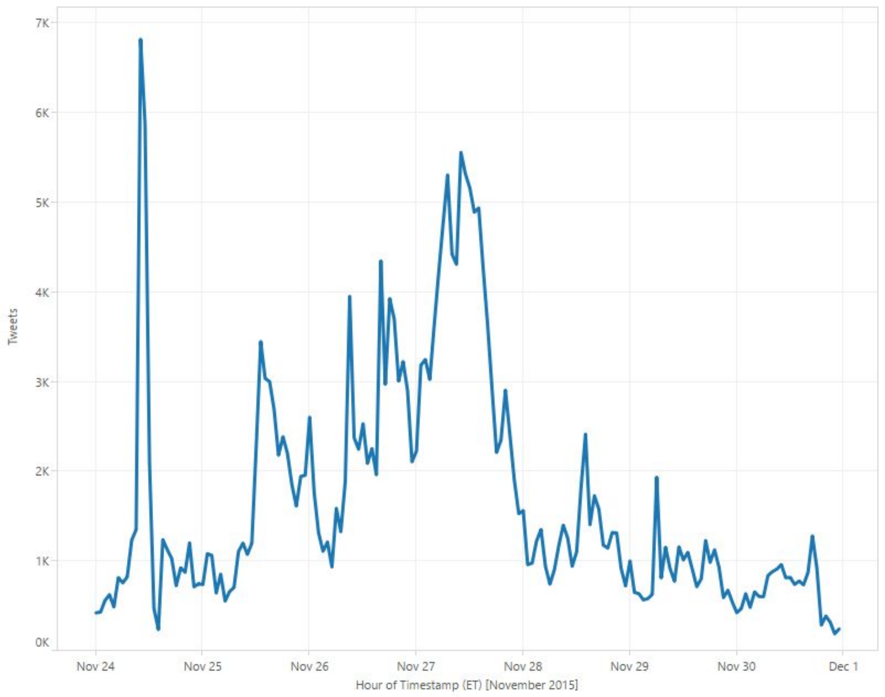
One caveat to mention is that since Twitter allows non-text characters to be added to tweets, we were not able to capitalize on doing any sentiment analysis on emojis. Being able to utilize emojis in sentiment analysis would further enhance the accuracy of the scoring system, since emojis can convey certain things that words cannot, such as sarcasm.

Data Analysis/Results

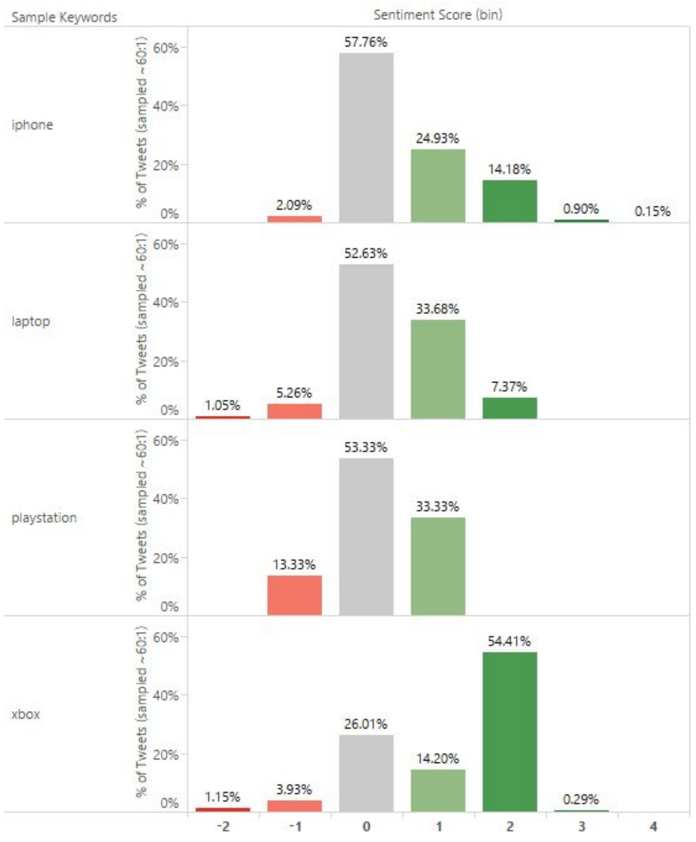
Once our data was tagged with a sentiment score, we loaded the data into Hive for high level aggregation. We chose to use Hive because it provided easy access for Tableau for our analysis. Although it is possible to connect to Hive and work with the data in real time, the version of Tableau we were running (Tableau Desktop) does not perform very well with large data sets using a live connection. Thus, once Hive server was running and Tableau was connected to our Hive tables, we performed a data extract so we could perform our analysis more efficiently.

Because of the amount of time it took to add a sentiment score to each tweet, we sampled the tweets that we processed with sentiment analysis. Our sampling algorithm consisted of only using tweets that were created at the top of the minute (seconds = 00). We confirmed that this was a reasonable sampling mechanism by looking at the distribution of tweets for every second. The distribution was approximately uniform, which allowed us to determine that there was no underlying bias in the second that a tweet was created. Thus, our tweets are sampled at approximately a 60:1 ratio.

Once our data was extracted and prepared for Tableau, we took a look at the time progression of consumer electronics tweets containing “black friday”. When aggregating to an hourly level, we see the expected increase in tweets leading up to Black Friday, then tailing off after Black Friday.



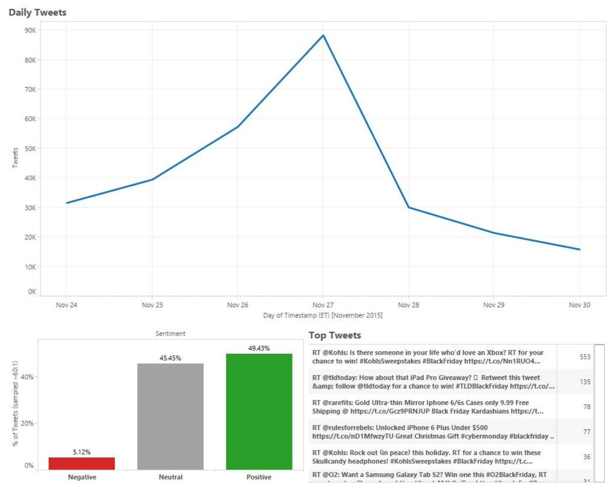
We noticed a massive spike in Black Friday tweets on November 24th, and discovered that Kohl's had tweeted a sweepstakes to win an Xbox. The sweepstakes required Twitter users to retweet to be entered into the contest. Thus, we see a massive spike in tweets. It is interesting to note, however, that the sweepstakes did not create a long-lasting increase in tweets. The spike lasted only a few hours.



When we turn to analyzing the sentiment scores, we sliced our data in several ways. We looked at categories of products, and compared the distribution of sentiment scores. For example, we looked at the terms iphone, laptop, playstation and xbox and found that most terms had a majority of neutral tweets. The term iphone had only 2% negative scores, while playstation had 13% negative scores. The term xbox had a majority of the tweets score positively in sentiment, as opposed to other terms who were mostly neutral in sentiment. Playstation had the narrowest distribution of sentiment, with most tweets either neutral or mildly positive/negative.

We also did a breakdown on brands mentioned within tweets, particularly Apple, Samsung, Sony and Microsoft. We found that Sony's tweets mirrored the Playstation terms, with tweets being either neutral or mildly positive/negative. Surprisingly, Samsung and Microsoft both did not have any negative sentiment tweets, and only had neutral and positive sentiment. Again, Apple's sentiment was about 2% negative, similar to what we found in the iPhone keyword. This is likely because people use the keywords Apple and iPhone together in tweets.

To pull all of these visualizations together, we created a very simple dashboard. The dashboard contain 3 visualizations: daily volume of tweets, histogram of negative/neutral/positive tweets, and sample tweets. Each visualization is interactive, and if any element of any visualization is clicked, the rest of the visualizations will be filtered to include only the information from the item that was clicked. For example, if the user clicked on a certain day in the daily tweets visualization, the sentiment visualization and sample tweets would update to only contain tweets from that day. The same would occur if a user clicked on a particular sentiment, or a particular tweet.



RT @Kohls: Is there someone in your life who'd love an Xbox? RT for your chance to win! #KohlsSweepstakes #BlackFriday https://t.co/Nn1RUO4...	553
RT @tldtoday: How about that iPad Pro Giveaway? ▯ Retweet this tweet &amp; follow @tldtoday for a chance to win! #TLDBlackFriday https://t.co/...	135
RT @rarefits: Gold Ultra-thin Mirror Iphone 6/6s Cases only 9.99 Free Shipping @ https://t.co/Gcz9PRNJUP Black Friday Kardashians https://t...	78
RT @rulesforrebels: Unlocked iPhone 6 Plus Under $500 https://t.co/nD1MfwzyTU Great Christmas Gift #cybermonday #blackfriday ..	77
RT @Kohls: Rock out (in peace) this holiday. RT for a chance to win these Skullcandy headphones! #KohlsSweepstakes #BlackFriday https://t.c...	36
RT @O2: Want a Samsung Galaxy Tab S2? Win one this #O2BlackFriday, RT	31

Other Considerations

Batch updates

Since our data is collected into daily tweet files, doing a batch update is a simple procedure, which we have implemented in our GitHub repo. The data retrieval script takes in a parameter for the date that the data should update with, and that day's data is appended to the cleaned tweet data. We would need to make small modifications to the script that ingests the data into a Hive table, as it currently creates a new table each time the script is run. This was done to create the initial Hive table, however, for batch updating purposes, we would be inserting into an existing table rather than creating a new table.

The queries could easily be run against the updated Hive table, and the data would flow into Tableau once the data extracts are refreshed. Thus, we could have a daily updating dashboard with a few minor tweaks to our data aggregation scripts.

Scaling the application

After operating this architecture for the duration of this project, we have ideas about how to better scale it. Spark streaming directly implements twitter4j and several python options also exist. By porting the sentiment analysis code to a spark streaming job and having it operate directly on the stream in this manner we might have been able to keep up with the twitter streams. The Sentiment analysis ended up being our most time consuming processing stage.