Twitter Sentiment Analysis

Jon Tatum

John Travis Sanchez

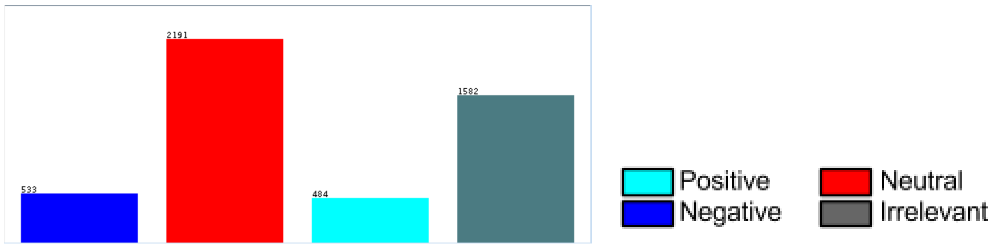
https://github.com/jnthntatum/cs229proj13

Abstract

As social media is maturing and growing, sentiment analysis of online communication has become a new way to gauge public opinions of events and actions in the world. Twitter has become a new social pulpit for people to quickly "tweet" or voice their ideas in a 140 characters or less. They can choose to "retweet" or share a tweet, to promote ideas that they find favorable and elect to follow others whose opinion that they value. Specifically, we studied sentiment toward tech companies in twitter. We tried several methods to classify tweets as positive, neutral, irrelevant, or negative. We were able to obtain high overall accuracy, with the caveat that the distribution of classes were skewed in our dataset.

Dataset

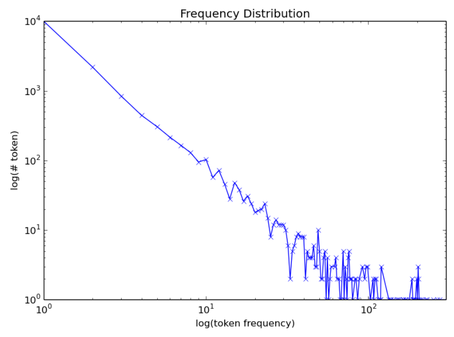
We used a hand-curated twitter sentiment dataset published by Sander's Lab. It contains tweets from 2007-2011 that mention one of four major Tech companies. Sander's Lab manually assigned labels for each tweet as either "Positive", "Negative", "Neutral", or "Irrelevant".[1] "Positive" and "Negative" indicated whether or not the tweet showed a generally favorable or unfavorable opinion toward the mentioned company. A "Neutral" labelling indicated that the tweet was either purely informative or the opinion of the tweet was otherwise ambiguous. An "Irrelevant" labelling indicated that the tweet could not be determined to fit into any of the other classes. This often indicated that the tweet was not written in english or that it was clearly spam.

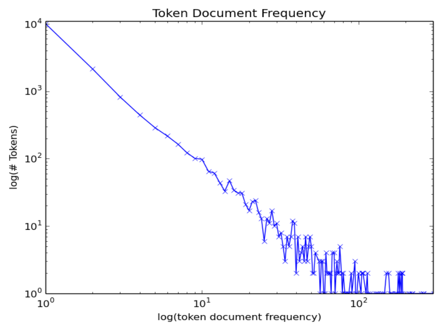


We hoped to see what techniques we could use in order to predict the labelling of an example tweet only using transformations of the tweet text. With this goal in mind, we spent some time analyzing the tweets themselves. These plots show, as expected from empirical evidence, the distribution of token frequencies roughly follows a power law distribution. Note that much of the right tail is cut off in each graph (a property of power law distributions is that the tails of the distribution are extremely heavy compared to most common probability distributions[2]). Given that a large proportion of tokens appear in only one or two examples, we can quickly reduce the complexity of our representation by filtering out those tokens.

[1] Sanders, Niek J. "Twitter Sentiment Corpus." Sanders Analytics. Sanders Analytics LLC., 2011. Web. 16 Nov. 2013.

[2]"Power Law" Wikipedia Web. 16 Nov, 2013





Modeling

The first challenge was finding a reasonable way to represent the tweet text. Language data is messy, considering each token as a distinct dimension in the feature space isn't viable. There are many words that are ubiquitous in text, and therefore aren't significant if discovered in a tweet. There are also words that only appear in one or two training examples across our corpus and don't provide a good estimate for how strongly associated they are with their labelling. Furthermore, there are many variations on words that don't change their meaning significantly, but require a different form for the sake of grammatical correctness.

In order to best represent the tweet text, we used several techniques shown to work well in document similarity problems. Specifically, we used a method loosely based on PubMed's related citation algorithm.[3] We stemmed tokens of the tweets using the Natural Language Tool Kit's[4] implementation of the porter stemmer, built a distribution over the frequency of stemmed words, and removed tokens that fell in the left tail of the distribution. We used the count of the filtered stemmed tokens in each input example as features in the vector representation of each tweet.

While it is reasonable to consider removing tokens from the right side of the distribution (ie tokens that appear extremely often), there is an *a priori* notion that some of the more common tokens are modifiers and are more significant in k-grams as opposed to single tokens. One example is "not" (or the stemmed, separated contraction form from our stemmer n't), which, seemingly, should invert the response of any predictor.

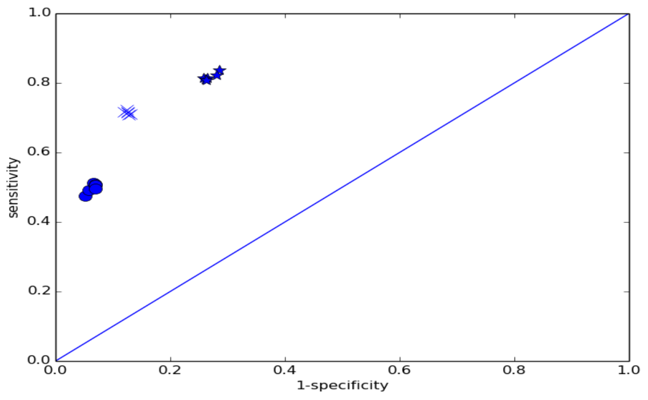
Feature Selection

To tune the amount of words that we removed from the left tail (infrequent tokens) so that our representation retained a reasonable amount of information from the tweets, we ran 4-fold cross validation on a multinomial bayesian classifier. We tried different different thresholds at which to retain tokens. We found that the data was skewed very heavily towards the neutral and irrelevant classes, biasing the predictions towards those classes.

[3] PubMed Help [Internet]. Bethesda (MD): National Center for Biotechnology Information (US); 2005-. PubMed Help. [Updated 2013 Nov 5]. Available from:/ http://www.ncbi.nlm.nih.gov/books/NBK3827

[4] Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

However, a very slight peak can be seen. Manually inspecting the performance, we found that ignoring terms that appear four times or fewer provided a good balance across the different metrics we considered (see table below). In considering where we drew the cutoff, we chose the threshold that gave us the highest sensitivity for positive and negative examples, without causing too much of a drop in specificity. Values for Positive Predictive Value (also called true positive rate and precision) are also shown. This reduced the number of features used from 13287 to 1646. In choosing this metric, we considered the use case of using a sentiment classifier as a first pass filter for analysis of the overall opinion of a population.

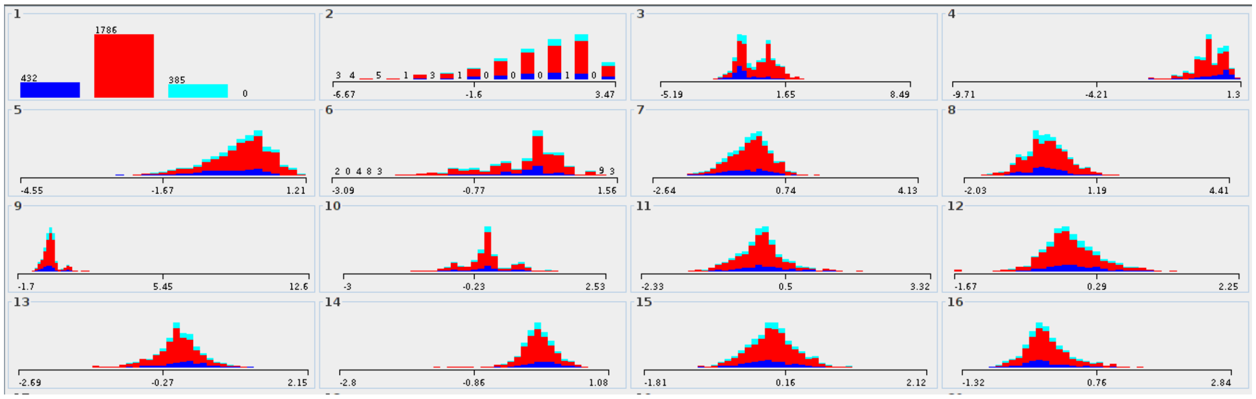


Min Freq	Vector Len	Pos PPV	Neg PPV	Neu PPV	Pos Sens	Neg Sens	Neu Sens	Pos Spec	Neg Spec	Neu Spec
2	4643	0.5091	0.4448	0.8989	0.4029	0.6811	0.8556	0.9563	0.8936	0.6431
3	2732	0.5077	0.4292	0.9159	0.4752	0.7167	0.8367	0.9482	0.8807	0.7148
4	2039	0.4897	0.4118	0.9158	0.4917	0.7092	0.8219	0.9424	0.8732	0.7200
5	1647	0.4644	0.4208	0.9195	0.512	0.7223	0.814	0.9336	0.8755	0.7355
6	1382	0.4539	0.4164	0.9211	0.508	0.7148	0.8136	0.9313	0.8746	0.7414
7	1212	0.4469	0.4152	0.9200	0.4959	0.7167	0.8113	0.9308	0.8736	0.7384

PCA

We investigated the use of Principal Component Analysis to reduce the feature size. We implemented PCA in numpy using the linear algebra module. We projected the unigram token counts using a threshold of two document occurrences to a 200 dimensional feature space. Using an off-the shelf machine learning workbench, Weka, we used the lower dimensional data set to train a Neural Network classifier. For this test, we used the unigram token count vectors generated using a minimum occurrence threshold of 2. We were limited by memory requirements of the library we used, this was the lowest threshold that we could run on our machines.

Visualizing the data projected onto the lower dimensional space showed promise. Viewing the first 15 principal components as stacked histograms indicated that the distribution of each of the classes for some of the features was noticeably different. This gave an indication that a general purpose classifier would perform well. Since we didn't have any strong notions of the underlying relations between the projected features and the classes, we went with a neural network, a model that performs well with many features.


(Figure: Weka Visualization of training data projected onto first principal components)

Classification

Having settled on some details of our model, we tested it with several different algorithms, using off the shelf libraries rather than implementing our own. We wanted to get an Idea for what methods worked well and which didn't. We generated training and test sets from our corpus by randomly permuting the examples and using 80% of examples for training set and 20% for test set.

We found that the results between the different models didn't vary greatly. All were able to achieve on the order of 70% accuracy and a precision of around .65. The confusion matrices for the Neural Network and the Multinomial Naive Bayes classifiers show some of the strengths and drawbacks of each. Naive Bayes performs better on correctly classifying negative and neutral tweets but doesn't perform very well on classifying positive tweets. On the other hand, the neural network has more even results across all classes.

Neural Network
(Weka using PCA reduced design Matrix)
Accuracy: 67.4%
ROC Area: 0.751
Precision: 0.678

Negative	Neutral	Positive	<- Prediction
58	48	11	Negative
57	340	37	Neutral
19	40	41	Positive

Multinomial Bayes
(Weka using Bigram Counts threshold at 5)
Accuracy: 68.9%
ROC Area: 0.74
Precision: 0.666

Negative	Neutral	Positive	<- Prediction
52	57	3	Negative
49	379	27	Neutral
12	54	16	Positive

The SVM classifier proved to have high accuracy for negative and positive tweets, while having a lower accuracy for neutral tweets. It also showed low sensitivity for the positive and

SVM

(SciKit in Python)

Accuracy: 0.754

Precision: 0.643

K-fold Validation Output:

Threshold	Label	Accuracy	Sensitivity	Specificity	Precision
4.00	Negative	0.90	0.58	0.94	0.56
4.00	Neutral	0.78	0.78	0.80	0.77
4.00	Positive	0.92	0.45	0.96	0.53
5.00	Negative	0.90	0.57	0.94	0.55
5.00	Neutral	0.79	0.77	0.80	0.77
5.00	Positive	0.92	0.44	0.95	0.52
6.00	Negative	0.89	0.56	0.94	0.53
6.00	Neutral	0.79	0.77	0.80	0.77
6.00	Positive	0.92	0.44	0.95	0.52
7.00	Negative	0.89	0.58	0.94	0.55
7.00	Neutral	0.78	0.78	0.79	0.76
7.00	Positive	0.92	0.45	0.96	0.53
8.00	Negative	0.89	0.55	0.94	0.52
8.00	Neutral	0.78	0.78	0.79	0.76
8.00	Positive	0.93	0.45	0.96	0.56
9.00	Negative	0.89	0.53	0.94	I0.51
9.00	Neutral	0.78	0.78	0.79	0.76
9.00	Positive	0.92	0.45	0.96	0.55

negative tweets which meant that the SVM correctly identified a low number of the positive and negative tweets. On the other hand, the SVM maintained a high specificity for the negative and positive tweets which meant that the classifier did often correctly classified negatives (alternatively, had maintained a low false positive rate). This means that our SVM model overall provides an useful way for finding some of the positive and negative tweets, and providing a high accuracy rate, but may often fail to identify many of the positive and negative tweets. While it would likely not provide a holistic view of public sentiment on a topic, it could work reasonably well filtering many of the uninformative tweets.

Conclusion

Results from this method aren't particularly compelling. We thought that the limited space permitted in tweets would allow for purely statistically based techniques, however that assumption seems to not hold. Across three models that are known to work well for this class of problem, we found poor performance. This indicates that our text representation isn't particularly good. Given that our feature space is still quite large, a more intelligent method for feature selection could lead to better results. his is somewhat corroborated by the most informative bigrams from the multinomial Bayes Classifier. Using the metric $log\frac{p(bigram|class)}{p(bigram)}$,

we found that many of the top terms are entity names which, whil likely strongly related to the classes, probably don't convey muc information on the underlying opinion of the tweets. While our proje hasn't shown a particularly great model for sentiment classification, has provided some suggestions on future approaches that may work

1. '7', 'devic',
2. 'microsoftstor', 'http',
3. 'offer', 'up',
4. 'microsoft', 'microsoftstor',
5. 'store', 'offer',
6. 'devic', 'microsoft',
7. 'up', 'free',
8. "n't", 'wait',
9. '...', 'i',
10. 'phone', '!',
11. 'replac', 'my',
12. 'thank', 'for',
13. 'love', 'the',
14. 'i', 'love',
15. 'googl', 'nex'

As requested in our meeting with Sameep on Travis's role in the project, here is a rough division of labor for the Project at this point.

Jon:
Procuring Dataset
Scraping Tweets
Text Processing / conversion to count vector
Feature Selection / code infrastructure for K-Fold cross validation
PCA
Classifier tests in Weka
Exporting data for use with Python Libraries
Majority of Writeups for Milestone, Presentation, and Final report

Travis:
Research on SVM libraries for python
Running SVM Classifier
40% Help with Presentation writeup
20% Help with Final Report

Travis plans to extend the project as he completes the course material over winter break and January.