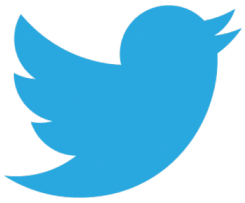
An Exploration of Narrow Twitter Sentiment Tagging Using Hidden Markov Models

James Stoyell* and Jesse Lupica*

E-mail: jms852@cornell.edu; jrl344@cornell.edu



Abstract

Our goal in this project was to examine methods for developing an agent that is able to analyze and parse the sentiment of a tweet at a level similar to that of a human. We hypothesized that training a Hidden Markov Model (HMM) on sentiment tags would be an effective method of understanding the overall sentiment of a tweet. We chose to focus on tagging many different sentiments down to the word level, an approach we had not seen before. It comes with its own unique challenges, as the largest obstacle in both generating and evaluating a model is ambiguity - even a human could tag a text in multiple different ways, and different humans would tag things differently. Using these narrow taggings also gave us sparsity issues, so we chose to incorporate word clusters generated via word vectors to mitigate them.

We focused on developing effective models to predict sentiment at the word and sentence levels. We were able to achieve relatively accurate predictions of overall

sentiment of the tweet be it positive, negative or neutral as well sentence and word level tagging of one of seven emotions: funny, happy, sad, angry, thoughtful, excited and neutral.

Challenges

Sparsity

Attempting to tag tweets at a token level as one of a variety of emotions is challenging in part because of the scope. In our training data we encountered an average of 14.7 words/tweet, meaning that the average $7^{14.7} = 2.65 * 10^{12}$ possible permutations of tags. This is an immense problem and no training set, regardless of size, could possibly account for these permutations. The problem of course becomes easier as most emotions fall within phrases or entire sentences, making the possible solution space much smaller, but still immense. Our training set of approximately 7852 words had 3868 unique word/tag pairs, meaning that our data is very sparse. This means that our model will encounter challenges with unseen words and unseen bigrams, making it very difficult to produce accurate lexical generation probabilities for calculating the probabilities of a Markov chain.

We approached this sparsity challenge in multiple different ways. First, we attempted to make our training set as large as possible, and we noted throughout the process that while there were diminishing marginal returns as the size of our training set increased, they were generally very small and each significant increase in the size of our training set yielded significant benefits when comparing against test data. We believe that a further increased test set would further increase the efficacy of our model, however, because of the diminishing marginal returns, these changes would get smaller and smaller. The lexicon of Twitter is also always changing, introducing new words and slang used in different ways, which further adds to the problem.

To help counteract this, we also introduced word clusters. We retrieved these clusters

from a publically available resource made available by researchers at Carnegie Melon University.[1] These is a collection of 1000 clusters trained on a corpus of tweets that allows us to reduce sparsity by combining known data about semantically similar words. Thus with our 3868 known word/tag pairs, we can account for dozens of times that many tokens and tags, as each cluster gives us more information that allows us to combine what we know about words.

Ambiguity

Human emotion in any context is inherently ambiguous. It is impossible to encompass the range of human emotion in any number of tags or any sequence of words. This is likely why many previous attempts at Twitter sentiment tagging[2][3][4] have focused on broader tagging - either tagging the emotions related to keywords across a wide number of tweets, or tagging on a scale of positivity. We did find references to narrow[5] tagging, but none at the token level. This ambiguity is one of the most challenging problems for machine learning algorithms, as it means often contradictory training data, especially when that training data comes from multiple sources.

Methodology

Data Generation

To gather Twitter data we leveraged the Twitter RESTful API via the python-twitter[6] package, which allowed us to gather hundreds of recent and random tweets. Twitter provides a useful API endpoint, `/statuses/sample`. The documentation for this endpoint[7] claims to return "a small random sample of all public statuses", but we quickly found that it returned entirely recent statuses - we saw many tweets about current events as they were happening.

After retrieving the tweets, we added a filter to collect only tweets that could be useful to us. The scope of our project is limited to English tweets, so we filtered out all tweets

in which 50% or more of the words did not match an English word in the Unix words file. Twitter also often sent back deleted tweets, which we also filtered out and ignored. The resulting sample of tweets that we collected were in no way skewed towards any one topic or sampled more highly from any geographic region other than unavoidable trends of sampling more highly in Western countries by filtering out non-English tweets.

Data Tagging

Because we were unable to find any publicly available repositories of curated with tweets with word-by-word sentiment tagging, we found it necessary to develop a method for manually tagging tweets in the manner necessary to train our model. To do this we developed TweetTagger (TT). TT is a command line interface that pulls tweets from local data stores and lets users efficiently tag each word as one of the seven emotion tags:

- Funny
- Happy
- Excited
- Angry
- Sad
- Thoughtful
- Neutral

Sentence level emotion tags were computed as the word tag that appears the most frequently within a given tweet. Sentence level sentiment tags were computed as the sentiment tag associated with the calculated emotion tag.

Figure 1: Sentiment tags and their associated emotion tags

Sentiment Type	**Emotions**
Positive	Funny, Happy, Excited
Neutral	Neutral, Thoughtful
Negative	Sad, Angry

Hidden Markov Model

We implemented a Hidden Markov Model from scratch using Python. We chose Hidden Markov models for this task because they excel at taking in features and adding tags to that data based on existing tags. Our "tags" in this case are one of the seven emotions on each word. We also include a distinct "TWEET_START" tag, to identify the start of a tweet and give the model a place to begin. Because we gave every word in our data set a unique tag, Hidden Markov Models are an ideal machine learning algorithm to tag each token in the context of the tweet. The model relies on two different probabilities: a bigram tag probability, the probability of a tag given the previous tag, and a lexical generation probability, the probability of a word given a tag for that word.

The training process of the algorithm is simple. We count the occurrences of all tags in the corpus of tweets that we manually tagged. We then count the occurrences of all bigrams of tags - that is, the number of times each tag appeared adjacent to each other tag. Finally, we obtain a count of each word/tag pair in the corpus.

For instance, in our training data we have the tweet:

my—THOUGHTFUL

life—THOUGHTFUL

is—THOUGHTFUL

messier—FUNNY

than—FUNNY

my—FUNNY

room—FUNNY

This would result in a tag count of 3 THOUGHTFUL tags and 4 FUNNY tags, a bigram tag count of 2 (THOUGHTFUL, THOUGHTFUL) tags, 1 (THOUGHTFUL, FUNNY) tag, and 3 (FUNNY, FUNNY) tags, and a word/tag count of (word,tag) for each word in the tweet. This process would then continue for the entire corpus. These will be used for

probability generation.

Tagging tweets is where the meat of the algorithm lies. We use the Viterbi Algorithm to find the most likely sequence of tags for any given tweet given our trainng data.

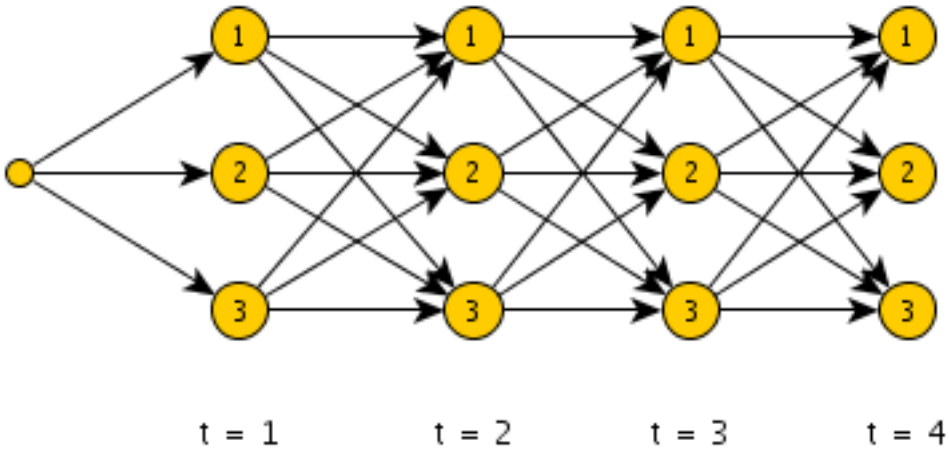


Figure 2: Viterbi Algorithm[8]

As seen above, the Viterbi algorithm works by starting from a source, in our case the TWEET_START tag. In then calculates the probability of moving to each next tag, in other words, the probability of tagging the next word as each available tag. The example above shows 3 tags; our implementation includes 7 separate tags. From there, it computes the probability of tagging the next word as each available tag given each prior tag, given that probability of each tag for the prior word is optimal. At the end, the optimal tag sequence can be found by backtracking through the tags that provided the optimal probability for the final word. The recurrence relation for this is as follows:

$$P(W_{i,t}) = max(P(W_{i-1,x}) * P(t|x) * P(W_i|x) \forall x \in T))$$

Essentially, we choose the tag and previous tag that maximizes the probability of the chosen tag given the previous tag, the word given the tag, and the probability of the previous tag/word calculated by Viterbi. By memoizing previous values, this algorithm is done in

$O(nt^2)$, where t is the number of tags and n is the length of the tweet.

Word Cluster Extension

For word clusters, we downloaded a pre-made set of clusters[1] generated using machine learning techniques. This data set contains 216,856 unique words spread across 1000 clusters, generated using 56,345,753 containing 847,372,038 individual tokens. The clusters are arranged as follows:

`01010000110 stay remain stayy saty stayyy -stay staay 2stay stayyyy keepit`

Generated using machine learning techniques, each cluster contains words whose word vectors are close to each other in distance, which in vector spaces relates to a similar meaning, found by finding words that are used in similar contexts. As seen in the above example, the words "stay", "remain", "stayy", etc are used in similar contexts and have very similar meaning. We leverage these word clusters to reduce sparsity in our data by treating all words in the same cluster as a single unique word. In doing so, we gain information about the hundreds of thousands of words found in these clusters though we only see thousands of words in our corpus. This is an enormous increase in the "knowledge" that our training set can leverage and does a great deal to reduce sparsity issues.

In our HMM, every time we see a word in our training set, we convert it to a binary ID representing its cluster ID. For instance, if we saw the word "stay", we would interpret that as "01010000110" while training our model, just as we would interpret any word in that cluster the same. If a word was not in any cluster, we used the Python hash function, then converted that to binary. In the end, we had 5572 words in our training data that existed in our clusters and 2279 words that did not exist in our clusters.

Results

Basic Hidden Markov Model

We approached analysis by having two people tag the same sample of 113 tweets. We used this as a test set, and compared the results produced by the HMM for those tweets to these manual tags. We did this because of the inherent ambiguity in Twitter tagging, as discussed in the analysis section.

For each result we also give values for accurate given either human tagged data. The rationale for this value is that if a human tagged the data as two different things, those two tags are approximately equally likely to be correct, and thus this gives us a better approximation of how "correct" these tags are relative to a human tagger. These "all" numbers unfortunately are not viable to compare to the human accuracy, but provide a general picture of how well the tags compare to a human.

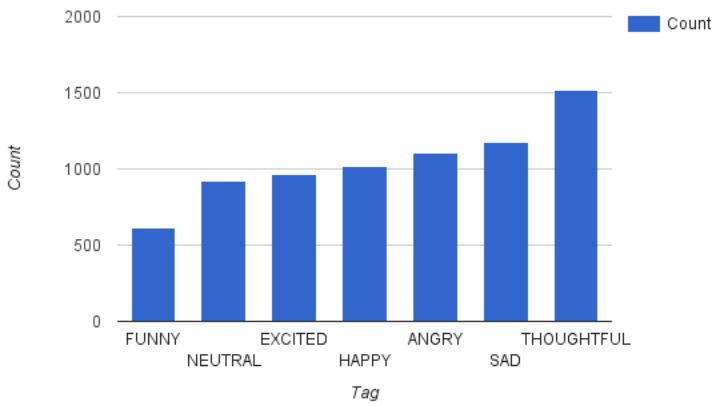


Figure 3: Distribution of Tags in Training Data

As can be seen in the figure above, the tags are moderately evenly distributed, so allowing a naive random baseline model to choose between two tags approximately doubles its probability of choosing a correct tag. However, as discussed below, because the human token data agrees approximately 50% of the time, the model only has an "extra" option about 50%

of the time, meaning we would only see at 150% in a hypothetical naive baseline.

We ran a variety of different tests to see in what ways our model was performing well. Figure 3 depicts how well our tagging agent performed compared to the tags from one member of our group. In this test we did not utilize word clusters in the manner described in the previous section and instead made no association among tokens. Our token match rate of 31.30% represented an average of the percent match rate within each tweet. For example, if tweet 1 had 3 out of 5 correct word tags and tweet 2 had 6 out of 7 correct word tags then their token match rate would be 72.85%. Thus, the average percent match rate of all 113 tweets between this person's tags and the model was 31.30%. This statistic was helpful in determine on a very fine granularity how well the the model did compared to a human. Because there are even disagreements among humans as to what the correct tag for a tweet is, we developed another metric. Despite its imperfections, this metric is a good baseline for our understanding of the performance of the model.

Figure 4: Tag Sample 1 without Word Clusters

Match Type	**Percent Matching**
Token	31.30%
Sentiment	46.02%
Mood	26.55%

Sentiment match rate represented the number of tweets that the model tagged that had the same overall sentiment as the manually tagged tweets, meaning that the most common emotion tag in the tweet among the two tags were in the same sentiment group as described in Figure 1. This was a very helpful metric in determining how effective our model was because while there can often be dispute as to whether a single tweet is mostly happy or excited, it is much less common for two humans to disagree that a certain tweet is overall positive or negative. Thus, we found this to be the most significant statistic when focusing on maximizing and optimizing our model.

A good compromise between the two previously mentioned statistics was the mood match rate, which was the rate at which the model's most common emotion tag for a tweet was the same as the that of its manually tagged counterpart.

Figure 5: Tag Sample 2 Without Word Clusters

Match Type	**Percent Matching**
Token	41.47%
Sentiment	55.57%
Mood	36.28%

We found that our model tagged data significantly more consistently with person #2. This could be due to a bias towards person #2's tagging pattern because of how the task of tagging the training data was split up where person #2 might have contributed more. Additionally, person #2 could be a more consistent tagger while person #1 could be more erratic and have less predictable tags that works less effectively with the model. While the individual comparisons of the model against a single person were lower than the human accuracy, shown in Figure 14, the all tags test, which counts a match on the token, mood, and sentiment level if it matches either manual tag, was close to the human accuracy, which shows that much of what the model is tagging is considered logical and legitimate.

Figure 6: All Tags Without Word Clusters

Match Type	**Percent Matching**
Token	47.91%
Sentiment	63.72%
Mood	43.36%

Word Clusters

To analyze the word clusters, we use the same accuracy methods utilized for the standard Hidden Markov Model and compare the results to those produced by the same tests using just Hidden Markov Models.

Figure 7: Tag Sample 1 With Word Clusters

Match Type	**Percent Matching**
Token	25.84%
Sentiment	40.71%
Mood	20.35%

Figure 8: Tag Sample 2 With Word Clusters

Match Type	**Percent Matching**
Token	31.09%
Sentiment	45.13%
Mood	23.89%

Figure 9: All Tags With Word Clusters

Match Type	**Percent Matching**
Token	39.90%
Sentiment	57.52%
Mood	33.63%

As can be seen from the above data, the word clusters still perform moderately well on the data. Against tag sample 2, it managed to match the tokens 31.09% of the time, which is approximately double what we would expect from a baseline that assigned each word a

random tag. On the "All Tags" comparison, we improve to matching on the token level approximately 40% of the time, which is adequate.

However, when we compare to the results of the HMMs alone, we see a clear trend.

Figure 10: **Comparison of success between model with and without word clusters**

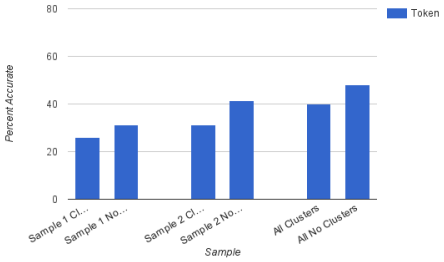


Figure 11: Accuracy of Individual Token Tags

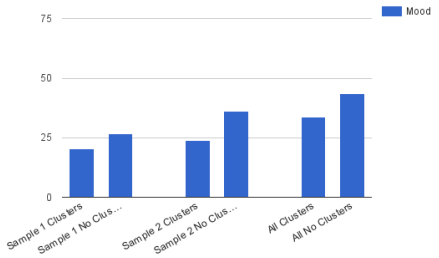


Figure 12: Accuracy of Sentence Level Mood Tags

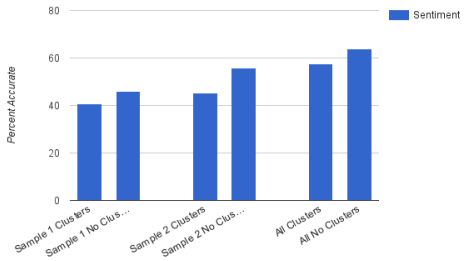


Figure 13: Accuracy of Sentence Level Sentiment Tags

On every single metric, the HMMs alone perform significantly better than the word clusters. These changes vary from a 5% to 15% decrease.

Given the obvious data sparsity issues that exist in handling sentiment tagging, it is unexpected that the addition of word clusters significantly degraded the performance of the system. We performed multiple checks and tests to confirm that clusters were working as intended on small datasets. These results and hypothesis for why are discussed further in the section below.

Analysis

Our overall conclusion was that we were able to successfully determine the mood and sentiment of a statically significant number of tweets. While it does not tag as well as a human, we believe that more training data and a few minor optimizations could make this a highly effective agent for determining the mood of a tweet.

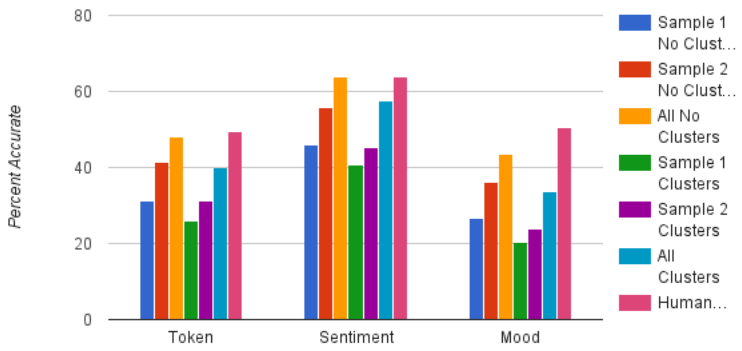


Figure 14: Comparison of all model variations on each metric

Among the results collected, we determined that word clusters were detrimental to the effectiveness of the model. We hypothesize that this is due to an over-generalization of the clusters. While many word clusters contained helpful connections between related words such as "trees" and "forest," many word clusters contained words that were antonyms that happen to appear in similar contexts. For example, "angry" and "ecstatic" are in the same word cluster despite their distinct differences in sentiment. We hypothesize that this causes less clear data as the data for the cluster becomes more muddled and less distinct with woulds with very obvious sentiment to a human such as "angry".

Figure 15: Agreement Percentages Among Manual Taggers

Match Type	**Percent Matching**
Token	49.48%
Sentiment	63.72%
Mood	50.44%

In our training of the model, we found a significant improvement (~10-15% increase on all three metrics) in the quality of the model's tags when we tripled the training data. Thus, we believe that to make this better and more human-like simply requires more training data.

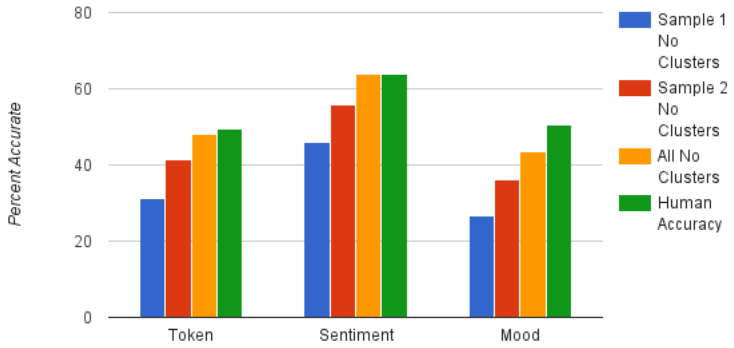


Figure 16: Agreement Percentages Among Manual Taggers

Conclusion

Narrow Twitter sentiment tagging is a challenging process that is ambiguous and extremely sparse, with an ever changing lexicon that make creating a model extremely difficult. However, we have shown that a Hidden Markov Model with a limited training set can accomplish upwards of 40% accuracy on the token level, which approaches the accuracy of a human tagger. This shows that HMMs are powerful tools with incredibly broad applications into many areas and can make connections and assign values in a larger context.

Given more time and more training data, we are confident that our model could even further improve to further approach human accuracy. This model could be used for determining the sentiment of a particular person over time, sentiment about a particular subject, or any number of applications. Through the project, we applied Markov chains and knowledge of machine learning systems to accomplish our natural language processing task.

References

(1) http://www.cs.cmu.edu/~ark/TweetNLP/.

(2) http://help.sentiment140.com/for-students/.

(3) https://www.csc2.ncsu.edu/faculty/healey/tweet_viz/tweet_app/.

(4) http://www.sananalytics.com/lab/twitter-sentiment/.

(5) https://www.aclweb.org/anthology/N/N16/N16-2011.pdf.

(6) https://github.com/bear/python-twitter.

(7) https://dev.twitter.com/streaming/reference/get/statuses/sample.

(8) http://www.athoughtabroad.com.