NLP-based Approach to Twitter User Classification

Matt Bush
Stanford University
Stanford, CA 94305
mattbush@stanford.edu

Ivan Lee
Stanford University
Stanford, CA 94305
ivan.lee@cs.stanford.edu

Tony Wu
Stanford University
Stanford, CA 94305
tonywu@cs.stanford.edu

Abstract

This paper addresses the issue of discovering new Twitter accounts to follow. Many users have difficulty reaching beyond their personal social networks and Twitter celebrities. The authors believe that by first training classifiers to categorize generic tweets, they can then begin classifying any user by breaking down and analyzing that user's past tweets. Tagging how often users tweet in a specific category can help other users identify promising users to follow. Another use is categorization of a user's feed. This allows users to read specific categories from the users they follow without having to browse through the entire list. Various approaches were used to tackle this issue, including Naïve Bayes, Language Models, Decision Trees, and the MaxEnt model.

1. Overview

The remaining sections of this paper will discuss how the authors used various models to maximize the accuracy on test sets. Section 2 will discuss how data was chosen and gathered. Section 3 describes the implementations for the different classifiers and parser as well as features and domain-specific algorithmic enhancements; Section 4 discusses our efforts to combine these classifiers to improve results. Section 5 gives the results for accuracy and precision, and an intermediate analysis on why we achieved these specific results. Section 6 ends with our conclusions and how we could potentially improve on these results in future works.

1.1 Twitter Overview

It is useful to begin with an initial introduction to our domain and the problems addressed in this project. Twitter is a microblogging service allowing users to post up to 140 characters at a time describing their thoughts. Users are able to "follow" other users, keeping up to date with information that user has to offer.

In recent years Twitter has surged in popularity, providing a huge source of untapped, unstructured data. Users have also used it for a wide variety of purposes and audiences. Twitter posts have been difficult for traditional NLP classifiers due to a less formal structure than other corpora. We hoped originally to classify Tweets based on popularity, but discovered that this ranking was not only too subjective, but that popularity would often depend on parameters outside of the realm of traditional NLP. Instead, we focus on real problems identified by users (Ramage, et al.) and turn to the challenge of categorization. It appears that some prior work has been performed on third party websites; but many appear to be hand tuned or, if automated, not always accurately.

First, though, we introduce some Twitter-specific terms and concepts referenced in the remainder of the paper:

Tweet - a microblogging post up to 140 characters in length.

@username - Tweets may be addressed to a specific user by placing their username followed by the "@" symbol somewhere in the tweet. These tweets remain public, but are used to direct tweets to another user, to thank another user, or to redirect traffic. Multiple users may be addressed in a single tweet. e.g. @chuckmd

#hashtag - Tweets can also be labeled as pertaining to a specific topic by using "hashtags": one-word labels following a "#" symbol. Multiple hashtags can be included in a single tweet. Twitter users can search based on hashtags to quickly find relevant content. e.g. #fun

RT - A retweet, labeled by the abbreviation "RT", is a Tweet that is a repeat of another user's tweet. The purpose of a retweet is to take a tweet and redistribute it on one's own feed in the hopes of broadcasting the message to a larger audience or demonstrating support of the tweet's contents.

1.2 Challenges to Resolve

i) One issue many users face is difficulty in *identifying interesting new users to follow*. Often, they are limited to their personal network and to well-known blogging celebrities; but finding new users with tweets relevant to their interests is scattershot or inefficient at best.

ii) Another problem for users is the *amount of clutter to sort through* in their Twitter feeds as they follow more and more users. Such clutter is cited as a primary reason for unfollowing users (Ramage, et al.)

The authors attempt to tackle the above challenges using three primary means:

i) Classifying and categorizing user's Twitter streams. Users can therefore be labeled and recommended to others. A user who is classified as having 70% sports-related posts and 20% news-related posts and 10% personal posts will be a better user to follow than someone who posts primarily about personal events.

ii) Expanding upon this concept, classifying all the users a single user follows can provide a clearer picture of her interests, and better users can be recommended for following.

iii) Finally, categorization of tweets can help organize a user's incoming Twitter feed. This will increase readability and filter out unwanted categories at times. We note that Twitter currently provides Lists as a feature, allowing someone to add other users they follow to predetermined lists and reading those lists. This organization is not as useful, though, when each of the followed users post in numerous categories and the user is currently interested in reading only news-related Tweets.

1.3 Hypothesis

Given the informal structure, abbreviated vocabulary set, and Twitter-specific features associated with this project, it is difficult to predict the success of various classifiers. Instead, we approached the problem with a number of classifiers, hoping that different classifiers will do well on different categories.

2. Gathering Twitter Data

2.1 Downloading Data

We had two main sources of data for our final count of 2497 hand-labeled tweets. The first was a live stream of tweets directly from Twitter's Search feature, at search.twitter.com. We searched for common words, especially those common to PERSONAL tweets, in order for the Search feature to return tweets. The Search feature formats these tweets as XML.

The second data source was the website www.chirrps.com, which is a Twitter aggregator for English-language tweets. Chirrps aggregates tweets that have been recently posted and re-tweeted heavily. Chirrps already sorts tweets by category, so we used it to get tweets that were exemplary of some of the categories we have chosen, and to make up for a lack of these categories in our training data. We copied tweets from this site as plain text.

One final category of data we acquired was spam tweets. Since these were not readily available from most Tweet aggregators, we located some heavily spamming users and branched out those users' networks to find other spammers. Again, tweets were copied directly from their twitter pages as plain text. As a side note, we quickly discovered in this stage that it was difficult to distinguish even by the human eye the difference between business promotions and spam tweets, as both had very similar advertising structures. However, a significant feature to be used in future spam classifiers is that spam accounts often submit via "API" for efficiency's sake, and post much more frequently; on the other hand, businesses will usually use the Twitter website or a third party application and post promotions at most once or twice a day.

2.2 Labeling Data

Unlabeled data was very easy to download in bulk, but we soon discovered that, given Twitter's esoteric language structure and vocabulary, labeled data would be much more useful for this project. We decided on the labels PERSONAL, NEWS, INSPIRATIONAL, SPORTS, and SPAM. These labels are all easily distinguishable by hand, fairly readily available, and they each seemingly have enough unique word patterns and language features that we predicted they could be distinguished by NLP methods as well. We considered other labels such as Celebrities and Tech, but as categories got more specific, they had fewer distinguishing features among their peers.

Sometimes this labeling was done by hand for individual tweets, in the case of the data extracted from the Search feature, in which case we added a tag to the tweet in the XML data file. At other times we had a large sample of tweets that were all in the same category, such as tweets we got from www.chirrps.com, in which case labeled data was already provided.

2.3 Parsing Data

Tweets passed to the classifier are formatted as a Tweet Java class (tweetdata.Tweet), with text, optional label, and metadata such as posting time and Twitter author. For XML data, we wrote a parser in Java, based on the SAX Parser java classes. For text data, we parsed the data line-by-line and added the category based on the file that we were parsing.

2.4 N-Fold Validation

We used N-Fold Validation in order to make the most of our data. We commonly used 6-folds, as we empirically found that it was just the right number which would allow for enough training data to train comprehensive models, as well as allow enough test data to verify accurately and effectively. We found that if we increased the number of folds and made it any larger, our folds had high variation in terms of reported accuracy, since the test set was too small to give an accurate assessment. If we decreased the number of folds, we would over-fit our classifiers due to not having enough training samples. Therefore, 6-folds was most appropriate for this project. The code for the N-Fold validation is in tweetclassifier.TweetValidator.java.

2.5 Measuring Performance

Throughout our project, we measure the performance of our classifiers by calculating how many tweets in our test set are correctly labeled. Specifically, we calculate the proportion of correctly-labeled tweets out of all the tweets in the test set. This performance metric suffices for comparing our classifiers, since their performance varies widely. It might be more accurate to calculate an F1 score by looking at precision and recall, but because we are placing tweets into one of five categories, rather than asking a binary question about tweets, this kind of score would be more complex to calculate and not much more useful to us.

3. Designing and Building Classifiers

3.1 The TweetClassifier interface

The base interface for all of our classifiers is the following methods:

- training based on a provided collection of Tweets that include their category
- testing based on a list of Tweets, returning a list of the best guess of category for each

3.2 Naive Bayes Classifier

The Naive Bayes classifier provides a baseline for comparison in this project. By considering individual tokens and their word counts within categories, it runs almost instantly to provide a solid vocabulary-based classifier. Our implementation removes common words such as "a", "and", and "the". On a limited data set it provided good results, achieving 71.0787% when including all labels. Better yet, on a full data set the accuracy went up to 76.0205%. On limited numbers of labels, the classifier actually did worse. For example, PERSONAL vs. INSPIRATIONAL scored only a 47.6776%. This can be explained by a greater intersection of vocabulary sets, and the intersection decreases with increasing numbers of labels.

Sample of miscategorized tweets and error analysis:

Predicted: SPAM
Actual: PERSONAL
Tweet: *Just completed a 2.31 km run with @runkeeper.*
Check it out! http://rnkpr.com/a67k7i #RunKeeper

Predicted: NEWS
Actual: INSPIRATIONAL
Tweet: *RT @dahara: An avalanche begins with a snowflake. Joseph Compton #quote http://ow.ly/17AfVI*

Similar to the LanguageModel classifier, the Bayes classifier makes mistakes when tweets have words characteristic of other categories, or have relatively uncommon words. In the first tweet, the words "Check it out" may be closely associated with SPAM, so the tweet was tagged as SPAM. In the second tweet, most of the key content words, like avalanche, snowflake, and Compton, are probably unrecognized or rare, and the best that the classifier can do is guess.

3.3 N-Gram Language Model Classifier

The N-Gram LanguageModel Classifier was one of our earliest things to try out. The Language Model we use is the Katz Backoff model from Assignment 1 in CS224N. We considered using Language Models in order to classify tweets because it made intuitive sense that certain types of tweets would have different distributions of unigrams, bigrams, and trigrams. For example, NEWS tweets would have "breaking news" or "Barack

Obama" as a bigram. Because of this, our approach was to take each category of tweets in the training set and build a separate LanguageModel (KatzBackoff). Then we treat each tweet as a "sentence" and get the probability of each sentence. We then normalize over all the categories, to get a properly formed probability for each category, for a given tweet.

The N-Gram language model performed okay. It classified an average of **51.69%** accurately according to six-fold validation.

Predicted: SPAM Actual: NEWS Tweet: *RT @cnsnews_com: Check out what THIS Dem has to say about 1,200 troops on the southern border... http://ow.ly/1SJ9l*

Predicted: PERSONAL Actual: INSPIRATIONAL Tweet: *Life is very nice, but it lacks form. It's the aim of #art to give it some. JEAN ANOUILH #writing*

Words can be misleading. In the first, to the parser words such as "check out THIS" are more evocative of SPAM than "Dem" and "border" are of NEWS. Since we're not doing any sentence parsing or POS tagging, we can't differentiate between more and less important words. In the second, only common words are used, there are few words that are specific to INSPIRATIONAL in the tweet.

In addition, Twitter users will speak in less formalized ways then standard English, which affects our language model's training. For example, many users will just say "spill", instead of "oil spill" to talk about British Petroleum's recent events in the news. For this reason, the LanguageModel Classifier had certain pitfalls and didn't perform stellar, but was nevertheless a good starting point for us.

3.4 Maximum-Entropy Markov Model Feature Classifier

Twitter users differ in their style of writing just as much as their content, and we thought to take advantage of these differences by writing a feature classifier, based on the Maximum-Entropy Markov Model Feature Classifier from a previous project. For example, news tweets tend to use more capitalized words and nearly always include a link; inspirational tweets are more likely than other types of tweets to use retweeting, attribution, and quotation marks; personal tweets over the other tweets are more likely to use first-person pronouns, question marks; and so forth.

We separated this classifier into two parts: the classifier itself, and sets of closely-related features, that we could easily activate, deactivate, and edit individually. We ended up with six feature sets total:

1. Capitalization Features - generates features for capitalized words and acronyms.
2. Punctuation Features - generates features for special punctuation, especially those that end sentences--question marks and exclamation marks.
3. Symbol Features - generates features for non-words such as numbers and dollar amounts.
4. Word Features - generates a feature for word types and qualities, such as the number of first- and second-person pronouns, and average word length.
5. Twitter Tool Features - generates a feature for each Twitter tool used and its approximate location within the tweet, such as addressing users, hash tags, and retweeting.
6. Twitter Link Features - generates a feature for each unique domain name that appears in hyperlinks within the tweet.

Unlike our earlier use of the MaxEnt classifier, Tweets were handled as a single piece of data rather than a collection of words. Because we are trying to tag Tweets with labels, not words with labels, we did not need a fine-grained classifier.

Sample of miscategorized tweets and error analysis:

Predicted: SPORTS
Actual: PERSONAL
Tweet: *Going to attempt to straighten my hair at 2 in the morning. Fun.*

Predicted: NEWS
Actual: SPAM
Tweet: Web Profit Plans - *Free Affiliate Marketing Video Maps. http://bit.ly/aInnAa #makemoney*

MaxEnt depends on looking at key features of categories. For example, PERSONAL tweets are more likely to have @user tags at the beginning, and NEWS and Sports are most likely to use proper capitalization. But when Tweets lack these features, it is difficult to categorize them. The first tweet is miscategorized as Sports because the only features it has are proper capitalization and it mentions a number, which are characteristic of Sports. The second has features that align more closely with news than spam, and is missing some of SPAM's occasional characteristic features like dollar signs, all-caps words, and exclamation marks.

3.5 Hyperlink Domain Classifier

We hypothesized that since many of the tweets contained a URL, link, we would use the domain of that URL as a feature. Tweets linking to news articles would often use the same URL or URL shortener. Likewise for spam, tweets that constantly spammed would advertise the same website, or usually point to the same domains. For this reason, we believed that using the hyperlink domain embedded in select tweets would help us classify them effectively.

We trained on this feature by recording the percentage of tweets in a particular domain that belong to a particular category. That effectively gives us P(category | domain). For tweets that do not contain URLs, the classifier simply picks randomly. Needless to say, this classifier is more interesting and effective when used on tweets contain *at least one URL* in the tweet.

Note that this classifier is best meant to be used in conjunction with other classifiers (see ***Section 4.2 and 4.3*** on *Combined Classifiers*). Simply running this classifier by itself on the data yields scores that are low. Here are the six-fold validation results:

All the data *(2375)*:

[0.3026, 0.3385, 0.3308, 0.3513, 0.2436, 0.2941] Average: **0.3101 (31.01%)**

However, if we restrict the set of tweets to only be ones that contain a URL or link in the tweet, we get much better performance.

Only tweets ***with links*** *(951):*

[0.3766, 0.4740, 0.5714, 0.5649, 0.3831, 0.4254] Average: **0.4659 (46.59%)**

If we further restrict the set of tweets to only be ones that are PERSONAL, NEWS, and SPAM, or even just NEWS vs. SPAM we get much higher results:

Only tweets ***with links*** *that are* ***PERSONAL, NEWS, or SPAM*** *(673):*

[0.6455, 0.6909, 0.7818, 0.8273, 0.5727, 0.5854] Average: **0.6839 (68.39%)**

Only tweets ***with links*** *that are* ***NEWS, or SPAM*** *(627):*

[0.6311, 0.6990, 0.8058, 0.8252, 0.6893, 0.6071] Average: **0.7096 (70.96%)**

One of the shortfalls for why domain link classifier doesn't work well for all tweets, is because only some of the tweets tend to have links in them (951 out of the 2375 in our training data). In addition, the advent of url shorteners (bit.ly, tinyurl) masks the underlying domain of the links ultimate destination.

As you can see, under select circumstances, the domain link classifier performs much better, namely when we only look at tweets that have a link in them, as well as only if we are deciding between two particular categories. Although having to remove this generality for the classifier to perform well means that it can't serve as a standalone classifier for tweets, it can be effectively used in combination with other classifiers, as you will see in Section 4.2 and 4.3.

3.6 Decision Tree Feature Classifier

We also reasoned that a Decision Tree Classifier might prove effective, and that it would be a natural thing to try next. Decision Trees are known to work well, even when there is little training data, so we felt that a Decision Tree would generalize well. Furthermore our data, even the numerical/continuous data fit very well into logical buckets (for example, 0 hash tags, 1 hash tag, 2, hash tags, 3+).

We also knew that a Decision Tree Classifier has the ability to separate the feature space into classes that are not necessarily contiguous, which might be advantageous. The drawback, however is that a decision tree requires your decision boundaries be perpendicular/orthogonal to the axis of your feature space, so it limits the flexibility somewhat in the classification.

Since a lot of our features were discretized/bucketized features, the implementation was natural. We made use of the package jaDTi, decision tree builder, but we wrote the classifier ourselves.

Sample of miscategorized tweets and error analysis:

Predicted: INSPIRATIONAL
Actual: SPAM
Tweet: *ACAI Berry Blast, Lose weight and feel great with the worlds #1 superfood promo -http://tinyurl.com/yh7t39p*

Predicted: INSPIRATIONAL
Actual: NEWS
Tweet: *As Hurricane Season Kicks Off, Gulf Oil Worries Grow http://bit.ly/bKWHJO #bp #oilspill*

The total results via six-fold validation (**all categories, all data**) are:

Scores: [0.3927, 0.4512, 0.5244, 0.4878, 0.4659, 0.4273] **Average: 0.4582 (45.82%)**.

The decision tree performed reasonable, but guessed INSPIRATIONAL on far too many tweets, including the ones that other parsers passed correctly. Most likely what is happening is that it had overfit INSPIRATIONAL due to a relatively small training corpus and a high presence of INSPIRATIONAL tweets in that corpus. We describe how we deal with this below (in Section 4.2, and 4.3), by combining the Decision Tree classifier with other classifiers, and limit the effect of the decision tree's tendency to over-select INSPIRATIONAL by consulting other classifiers in these situations like these.

4. Combining Classifiers

4.1 Extending the TweetClassifier interface

To use our classifiers above in conjunction, we modify the Classifier interface as follows:

- testing now returns a list of probabilities for each category that a tweet could belong to, rather than just the top category
- another class weighs and combines these scores to come up with the final guess

4.2 Smart Combined Classification

After we built the first set of classifiers for the project we took it further and built a classifier on top of that, which would automatically learn which classifiers to obey under what circumstances.

Consider the following example:

Tweet real label: INSPIRATIONAL

Entire tweet is: “The best way to predict the future is to invent it” ~@ParisHilton #quote

domainLinkDistribution:

PERSONAL:	0.25
NEWS:	0.25
SPAM:	0.25
INSPIRATIONAL:	0.25

decisionTreeDistribution:

PERSONAL:	0.10967117988394594
NEWS:	0.3650158403258049
SPAM:	0.11781909095547541
INSPIRATIONAL:	0.4074938888347734

featureDistribution:

PERSONAL:	0.11894653971314237
NEWS:	0.2516638767429611
SPAM:	0.10219075580945536
INSPIRATIONAL:	0.5271988277344414

langModelDistribution:

PERSONAL:	0.01641615160049283
NEWS:	0.979684985148234
SPAM:	0.0038988632512732056
INSPIRATIONAL:	0.0

In this example, the real label is INSPIRATIONAL, and even though the Language Model (langModel) gives it a 0% probably of occurring, and has labeled it as NEWS, the Decision Tree and TweetFeature Models have the correct category, *INSPIRATIONAL*, as the highest score. Looking at this, we decided we need rules that would govern, given the existing distributions and each classifier's assigned probabilities to each category, how we select the actual best category. Essentially, in this situation, we would like to have a rule on top of the existing categories that states something like, "if DecisionTreeClassifier and TweetFeatureClassifier both have INSPIRATIONAL as the highest, then the category to pick is INSPIRATIONAL, regardless of whether LanguageModelClassifier agrees". This is the intuition behind the combined classifier. We manually set these logical rules, and found decision boundaries for which to cut off the data and decide labels.

Some of the rules we hand-set were:

1. If DomainLinkClassifier's probability is > 0.99 for SPAM then label is SPAM.
2. If DomainLinkClassifier's probability is > 0.60 for PERSONAL, then label is PERSONAL.
3. If a majority (3) of the classifiers pick a particular category (give it the highest probability), use that probability.

4. If there is no clear majority, then go with the Bayes classifier (which had the highest individual accuracy overall).

The results were overall very good:

On six-fold validation using all categories **except for SPORTS**, the test folds' accuracies were:

[0.6824, 0.7770, 0.8209, 0.7872, 0.8041, 0.7492] for an **average of 0.7701 (77.01%)**

On six-Fold validation using all categories **including SPORTS**, the test folds' accuracies were:

[0.6744, 0.7436, 0.8051, 0.7923, 0.7949, 0.7553] for an **average of 0.7609 (76.09%)**

Obviously these results are very good and the best results we've come up with. It must of course be prefaced by saying there is the human component of us manually setting which rules to use in our logic. We made the rules as general as possible without looking at each test sample and picked rules that made logical sense and that we thought would generalize well. Though it performed better than the automated rule generation (see below Section 4.3), we believe that this is due more to the fact that the decision tree building algorithm chose poor splits than as a result of our specific rule-picking being tailored to the data. We discuss it in more detail below.

4.3 Automatic Combined Classification

Instead of hand-setting the features, we then decided to learn them using another high-level Decision Tree. Each feature in this overarching decision tree would be the category probability/score that each classifier assigned, for a total of NUMCATEGORIES * NUMCLASSIFIERS features. Therefore there would be a large decision tree on top of the existing classifiers, that would combine the results of each classifier below it (including another DTree itself) and determine the best course of action. This way, the DTree would learn the rules itself and determine at what probabilities for which classifiers would we label certain things the label X.

The file associated with Automatic Combined Classification is in tweetclassifier.SmartCombinedTweetClassifier.java.

Overall it did fairly well:

On six-fold cross validation with all categories except for SPORTS, t**he test folds' accuracies were:**

[0.689, 0.7466, 0.7905, 0.7838, 0.7905, 0.7554] for an average of 0.7593 (75.93%).

On six-fold cross validation with all categories including SPORTS, the test folds' accuracies were:

[0.6667, 0.6897, 0.7692, 0.7769, 0.7949, 0.7694] for an average of 0.7445 (74.45%).

These results were obviously really good, however slightly lower than that of the Smart Combined Classifier. Because the rules are learned, we didn't have as much control over setting them to fit our observations perfectly. In addition, though the Decision Tree building algorithm splits on features in order of maximum entropy, the algorithm didn't choose discretized buckets to the continuous probabilities very well. For example, in one of the folds, DomainLinkClassifierSPAM <> 0.6 was one of the cutoff splits in the DTree. However, upon further analysis of the data, splitting at a higher position, say 0.9, would have been better because the DomainLink Classifier. Since DomainLinkClassifier was very sensitive to spam (because of any repeatedly seen links being tweeted many times), and so it usually returned very high probability of spam if the label was truly spam. Therefore the information gain at splitting at 0.9 was much greater than that of 0.6.

4.4 User Classification

In line with our original purpose, we ran two of our best performing classifiers – Naïve Bayes and Smart Combined classifiers – on various popular Twitter streams. We trained on all of our collected labeled data, and tested on the users' Twitter streams. We printed out the percentage of tweets belonging to each category for the users, thus allowing categorization of the user. The classifiers did quite well, scoring consistently above 70%. The results can be found in Section 5.2.