Language Disambiguation for Disease Analysis

Dept. of CIS - Senior Design 2013-2014*

Nathan Fraenkel
fraenkel@seas.upenn.edu
Univ. of Pennsylvania

Varun Gupta
varung@seas.upenn.edu
Univ. of Pennsylvania

Jason Lucibello
jluc@seas.upenn.edu
Univ. of Pennsylvania

Boyang Zhang
zhangb@seas.upenn.edu
Univ. of Pennsylvania

ABSTRACT

Our research develops a machine learning model that will classify disease-related tweets based on whether or not someone has a given disease. More specifically, it will classify a tweet into one of three categories: (1) Self - *the tweeter has the given disease, (2)* Other - *the tweeter is talking about another person with the given disease, (3)* General - *the tweet is a general statement about the disease. We believe that this classifier is useful for medical and social science researchers who are looking to find potential participants who have a certain disease for clinical studies.*

We made our model by originally focusing solely on diabetes. We scraped Twitter for diabetes-related tweets and had them labeled for us through Amazon Mechanical Turk to build an annotated tweet corpus with which we trained and tested our model. Additionally, with various feature generation and feature selection methods, we were able to create an accurate tweet classifier. Finally, we were able to package our classifier into a tool for finding specific Twitter users with disease-related tools.

1. INTRODUCTION

People's health and well-being have always been a popular area of concern. Traditionally, expressing one's current state of health has been relegated only to doctor's offices, making prevention much more difficult than diagnosis. However, recent developments in social media have provided online users with a new means for expressing their level of well-being. The study of people's well-being via Twitter posts or Facebook status updates has become a common research topic in concordance with sentiment analysis and other fields related to human emotions and interactions.

We wanted to leverage natural language processing to understand the intersection of social media and healthcare. We believe that the availability of data from social networks such as Twitter will allow us to analyze public health and social wellbeing in novel ways. Natural language processing is one way for us to rapidly process and analyze this wealth of data. We feel that we could potentially provide the foundational steps for future research on the relationship between social media data and human sentiment to ultimately better understand and provide healthcare.

Researchers studying disease and the spread of diseases are currently looking into aggregating social media data to determine correlation between the sentiments expressed in specific subsets of social media posts from a common geolocated area and the rise of certain types of diseases. It is believed that a population's frequent posts on social media can be indicative of actual trends in the physical world. More concretely, there may exist a correlation between disease-related posts added by human users on social media and the prevalence and spread of those diseases in the physical location where those posts originated. Google has begun to explore this correlation with Google Flu Trends [4]: Google is able to identify a rising trend in Google queries relating to the flu, allowing the company to predict if a flu outbreak will occur in a specific area.

Besides Google Flu Trends, there are projects in the study of disease and well-being that focus on sentiment analysis through tweets and Facebook status updates and how they can be indicative of a population's well-being and even of the sentiment on a certain disease within a certain area. This is what the Word Well-Being Project (WWBP) [10], a collaboration between computer scientists, statisticians, and psychologists at the University of Pennsylvania, is currently working on. Two Penn researchers, Lyle Ungar and Andy Schwartz, work with the WWBP and attempt to find the correlations between tweets and people's well-being throughout the United States. Numerous projects exist today that are focused on trying to understand how social media can be indicative of a population's wellness and well-being, and this work is helping to further understand our population and help predict areas of weakness for those in need of help.

We believe that people's posts on social media can can also be used to find people who have (or know someone who has) a certain disease. The following machine learning model automates some of the manual labor required for locating people with a certain disease. Currently, determining suitable patients without access to medical records is incredibly difficult. This involves manually calling and obtaining permission from enough subjects, a process that is incredibly time consuming. A streamlined identification system would save time, money, and reduce potential bias.

The objective of this research is to analyze Twitter data related to a given disease and extract if someone actually has the disease from the language of the tweet. More specifically, given a tweet, we want to determine who, if anyone, has the given disease, and if that person is the tweeter themselves or another person entirely.

Disambiguating natural language is a skill that human

*Advisors: Lyle Ungar and Andrew Schwartz

beings do automatically and subconsciously using various context clues and pattern recognition from a individual sentence. However, this basic human skill is very difficult to implement in software due to major technical challenges. These challenges include word-sense disambiguation, understanding slang, jargon, and human idioms, and creating a useful model for processed data.

Word-sense disambiguation is understanding the different definitions of words based on their context. Given the word 'heart', we can have multiple meanings from completely different contexts such as:

1. My father is suffering from heart disease.
2. I heart you very much.
3. Lebron James puts his heart and soul into being the best basketball player in the world.

While human beings are able to glean the context almost instantaneously, it is very difficult for a computer to understand the meaning from the context of the word 'heart'. Other aspects of word-sense disambiguation is understanding internet jargon and English slang that does not appear in the Oxford English dictionary. Our goal is to bring together current knowledge of machine learning models and classifiers and apply it to Twitter content related to disease in order to streamline the process of identifying people who have, or know someone who has, a specific disease.

In the following sections, we will discuss related work on language disambiguation as well as some necessary background, our system implementation and design, and finally, our results and the applications for our model.

2. DEFINITIONS

The following is an explanation of the set of concepts that are required to understand the methodology and reasoning behind this research in language disambiguation.

2.1 Data Collections Libraries

2.1.1 TwitterSearch

TwitterSearch [5] is a set of Python libraries designed as a wrapper in python for the Twitter Search API. Twitter.com provides access to all publicly available tweets via a single set of tools, but these must be individually adapted for each programming language. TwitterSearch provides an accessible medium to access the Twitter's real-time data based on a wide array of features, including keyword, language, location, and username.

2.1.2 Amazon Mechanical Turk

Amazon Mechanical Turk [1] is a marketplace that provides an on-demand workforce for tasks that require human intelligence. Tasks include surveys, classification tasks, selectors, and anything else that may require human judgement or reasoning. MTurk Requesters pay anonymous members for completing these crowdsourced tasks, known as HITs. The service has been deemed a type of "*artificial* artificial intelligence".

2.2 Machine Learning Python Libraries

2.2.1 SKLearn

Scikit-learn [8] is a robust machine learning software suite developed as a project from Google's Summer of Code. It provides a set of machine learning and data analysis tools for python that supports model selection, classification, and 3D visualization. Scikit-learn importantly provides accessibility to a large number of model variables to create flexible, modular machine learning models.

2.2.2 NLTK

The NLTK [2], or the Natural Language Toolkit, is a community-driven platform of python programs that work specifically with human language data. It provides interfaces for such functions as tokenization, stemming, part-of-speech tagging, and data parsing. The NLTK manipulates language data into a more usable format for model training.

2.3 Machine Learning Models & Terminology

2.3.1 Feature Selection

A feature is the basic unit of classification in machine learning models. Features provide measurable heuristics that differentiate various classifications, and they are the unit of comparison for test data when the model is run. For text classification, these features take the form of unigrams, individual word frequencies, and bigrams, frequencies of pairs of words. Training data generates a set of features for each classification, and these sets are compared against the feature sets from each piece of test data, and classifications are given scores based on the statistical likelihood that the test set of features belongs to each class. Given a large corpus of data, the available features can be in the tens of thousands. For this model, the training corpus generated more than 27,000 unique features.

Feature selection is the process of reducing these features to the small percentage that make an appreciable difference in classification. Because of the enormous dictionary of possible bigrams, the frequency of most features is extremely low, which increases the likelihood that features create noise in the scoring data. Eliminating unhelpful features decreases misclassification and strengthens the importance of the features that correctly determine category. Our machine learning model used approximately 350 features. The process of selecting the best features is a principle deciding factor in model accuracy.

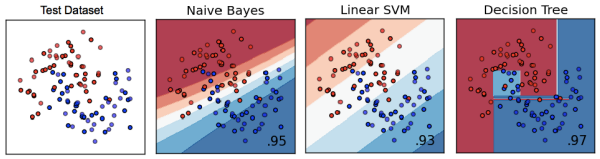


Figure 1: Machine Learning Classifiers [8]

2.3.2 Naive Bayes Classifier

A Naive Bayes classifier categorizes datasets by scoring individual features assigned with each class. It uses the naive assumption that all features are independent to create an efficient, if overly simplistic, classifier.

For example, a Naive Bayes Classifier for household objects might consider an object a basketball if it is orange, round, and approximately nine inches in diameter. It considers these factors independently in terms of their contribution to the probability that the object is a basketball. This can lead to misclassification for items with similar characteristics. For example, an orange might be classified as a basketball because it has many of the same features when considered independently.

2.3.3 Support Vector Machines (SVMs)

Support Vector Machines are perhaps the most widely used type of classifier in machine learning. SVMs treat each input datapoint as a n-dimensional vector, where each dimension is an individual feature. Using a collection of vectors from training data, the SVM creates an (n-1)-dimensional hyperplane that breaks the higher dimensional space into distinct categories. Depending on where our input test data points land in this hyperspace, they are classified as the respective category.

While nonlinear SVMs also exist in some machine learning models, linear support vector machines are best used for test data such as tweets that have a small number of features per datapoint. This project employs such a linearSVM in order to quickly create a hyperplane between classifications for data with relatively small feature sets.

2.3.4 Decision Tree Classifier

Aside from Naive Bayes and Support Vector Machines, Decision Tree Classifiers are a third popular alternative for machine learning classification models. Rather than score individual features, decision tree classifiers acts as a type of flow-chart. Each internal node of the constructed tree is a type of feature test on the test data, and each leaf node denotes a classification. A given piece of test data traverses the tree by performing these tests until it is eventually reaches a labeled leaf node. Because of its unique structure, the dividing lines between classes with decision tree classifiers can be highly nonlinear. A visual outcome of each of the three types of classifier is provided in **Figure 1**.

3. RELATED WORK

Word-sense disambiguation is a well-studied area in machine learning. The basic steps of building any type of natural language processing classifier is to first categorize, tag and tokenize words in a specific manner [3]. Tagging and tokenizing words is then followed by feeding these tagged words into a machine learning classifier such as Naive Bayes, linear SVM, and Max Entropy. This method is known as *supervised classification* and it allows us to take the words that we tagged with specific meanings and detect patterns in meanings that correlate with actual patterns of meaning [3]. However, training a machine learning classifier first involves developing a large corpus of tagged words to develop accurate predictions.

Work related to language disambiguation can be divided into two primary steps: Data Annotation (Sec. 3.1) and the Machine Learning Classifier (Sec. 3.2). Data annotation explores the process of quickly and accurately labeling large corpuses of data. Machine Learning Classifier describes the various models that we can train to read in text and predict the meaning of the text.

3.1 Data Annotation

In order to be accurate or effective, word sense disambiguation requires a huge corpus of knowledge from which to draw conclusions. Roberto Navigli 2009 survey entitled *Word Sense Disambiguation: A Survey* [7], attempted to solve the challenges of word ambiguity and nuance. Navigli explains that any technique for a word sense disambiguation system can be boiled down to a basic procedure: use a feature tagger such as n-grams that uses many sources of knowledge to tag words with the appropriate senses in context. He notes that creating this manual of knowledge is time-consuming and expensive, and must be repeated whenever the disambiguation scenario changes [7]. This is known as the *knowledge acquisition bottleneck*.

However, recent developments have noted that the use of crowdsourcing technologies can overcome this knowledge acquisition bottleneck. Through Amazon's Mechanical Turk, the corpus of knowledge that any disambiguation technique relies on can be built quickly, cheaply and efficiently. The Mechanical Turk is Amazon's crowdsourcing platform known as an "*artificial* artificial intelligence". We can leverage the first artificial because it can use human intelligence to solve computationally difficult computer science problems. According to Anna Rumshisky [9], Mechanical Turk has been utilized in other natural language processing tasks, such as testing machine translation systems. She notes that the Mechanical Turk platform has already had limited use in helping to create labeled data sets for word sense disambiguation and modify existing word sense databases. However, the databases were originally always created by experts and is therefore bottlenecked by the amount of experts that are able to create these word sense databases [9].

3.2 Machine Learning Classifier

In the aforementioned *Word Sense Disambiguation: A Survey*, Roberto Navigli surveys various supervised classifiers that have been used to generate accurate disambiguation of sense. We will discuss the classifiers that we utilized in our machine learning model.

The Naive Bayes classifier is a simple probabilistic classifier based on the application of Bayes' theorem [7]. It relies on calculating the conditional probability of each sense S_i of a word w given the features f_j in the context. Navigli states that although Naive Bayes is not the most optimal method, it compares well with most other machine learning classifiers and provides a good baseline to benchmark speed and accuracy. We can leverage this fact to benchmark our model on the Naive Bayes trainer as well as other supervised learning classifiers.

Besides Naive Bayes, Navigli discusses approaches with other classifiers such as a neural network. "A neural network represents 'artificial neurons' and uses input features to partition the training contexts into non-overlapping sets corresponding to the desired responses. As new pairs are provided, ink weights are progressively adjusted so that the output unit representing the desired response has a larger activation than any other output unit" [7]. However, we see that a neural network is not useful for the scope of our model. Although the neural network approach provides comparable accuracy with the other supervised learning classifiers, it is difficult to interpret the results of a neural network and it requires very fine tuning of parameters such as thresholds and decay. It also requires a much larger quantity of training

data compared to other supervised classifiers.

One of the best classifiers that Navigli recommends is the support vector machines method. This method, developed in 1992, is based on the idea of learning a linear hyperplane from the training sets that separate positive examples from negative examples [7]. This hyperplane is located in that point of the hyperspace that maximizes the distance to the closet positive and negative examples (aka support vectors) [7]. In sum, the support vector machines (SVMs) tend at the same time to minimize the empirical classification error and maximize the geometric margin between positive and negative examples. Navigli notes that SVMs have been applied to a number of NLP problems such as text categorization and have achieved some of the best accuracies for word-sense disambiguation compared to other supervised approaches.

4. SYSTEM MODEL

The main model of this research is a language disambiguator that will extract certain meaningful information about a particular disease from a large corpus of tweets. More specifically, we will take in a set of tweets as input and will output who or what the tweeter is referring to in reference to the disease. For example, given the tweet, "My dad hates taking his blood sugar measurements 5 times a day due to his diabetes", our disambiguator will be able to predict that the tweeter is an individual and that the tweet is referring to their father, who has diabetes. The machine learning process has three main components, data collection, data tagging, and a supervised machine learning classifier (See **Figure 2**). These elements allow disambiguation given a specific disease keyword.

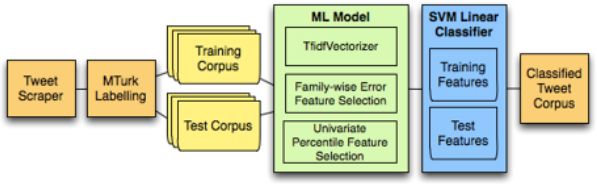


Figure 2: Block Diagram of the Full Model

4.1 Dataset Collection from Twitter

The first step is data collection and formatting. We must collect a large corpus of relevant tweets in order to train the machine learning classifier. For example, with diabetes as a target disease, tweets should be collected referencing the disease with English as a target language. While Twitter.com provides basic access to public tweets, a web scraper is required to harvest the usable and relevant data needed for training. The collected data must then be processed and formatted properly to prepare the data for annotation (training corpus) or prediction through the SVM classifier (test corpus).

4.2 Dataset Annotation with Mechanical Turk

After collecting a large corpus of relevant data it must be annotated with a correct classification to be useful to our training model. We must now label each tweet so that the classifier can categorize the elements of the tweet and associate it with a specific label. In order to expedite the process, we offset the labeling computation power to Amazon's Mechanical Turk crowdsourcing platform. Leveraging Mechanical Turk allows us to build a large, labelled training corpus at a low cost of time and money for the researchers.

4.3 Model Training with SKLearn

The final component involves the machine learning classifier. This component is broken down into two steps. The first involves feature creation and feature selection. The second is training the statistic model for accurate prediction. We generate tweet features using unigrams and bigrams. However, the feature creation generates noise, so we must do some careful feature selection to reduce the number of irrelevant features in our model. The second step involves formatting the data from text values to features that our SVM classifier can understand and utilize to categorize unknown tweets.

4.4 Model Classification with Support Vector Machines

Finally, the model uses its trained feature set to classify unknown input tweets. It similarly breaks each tweet down into its constituent features, and compares them against the weighted feature set for each category. Tweets are scored for each category by the features that they have in common and the weights that each of those features has to the category in terms of correlation. After these scores are calculated for each classification, the max score is selected as the classification for that tweet.

5. SYSTEM IMPLEMENTATION

In the System Model section, we went into a brief overview of the machine learning process needed for successful classification of tweets. Here, we will discuss the functionality of our model in more detail. This will be broken down into its four main components: the data collection and filtering process, the methodology for leveraging crowdsourcing to tag our corpus, the feature generation and selection process used to train our model, and finally the classification process on unseen tweets.

5.1 Dataset Collection, Annotation, and Filtering

The data collection and filtering proved crucial to developing an accurate model for our specific objective. We developed a *data collection funnel* to filter out unusable Twitter data and only kept the data that could help our model.

We had a specific methodology for collecting the correct type of data to train our supervised classifier. This is what we dubbed as our data collection funnel. This induced a specific type of bias into our model, due to the fact that the majority of the tweets that we scraped were ill-formatted and unusable. The reason we had this selection bias was because we needed our model to understand the specific type of tweet that we were looking for and recognize the rest as discardable.

The first step to our data collection was using a Twitter scraper that we built to obtain as many tweets about diabetes as possible. Using the TwitterSearch [5] python libraries, we were able to filter urlTwitter.com's real time firehose of information by keyword, language, and location. To create an effective model, we needed to look only at tweets relevant to diabetes, our chosen disease. Therefore, we only

searched for the terms diabetes, #diabetes, #t1d, and #t2d (for Type 1 Diabetes and Type 2 Diabetes, respectively). As we were doing language disambiguation, we also restricted tweets to English to avoid issues with translation, jargon, and language-based idioms.

Scraping from Twitter yielded an *unfiltered tweet corpus* of somewhere around 240,000 tweets. However, many of these tweets were duplicates, retweets, mislabeled as English by users, or tweets with unrecognizable symbols; they were all, therefore, unusable. We removed all retweets and removed tweets in a different language thanks to langid.py [6], a Python script that determines the language of input text.

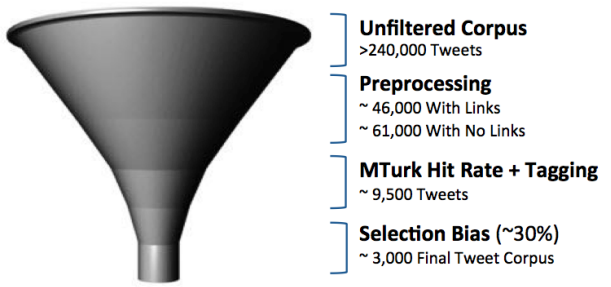


Figure 3: Data Collection Funnel

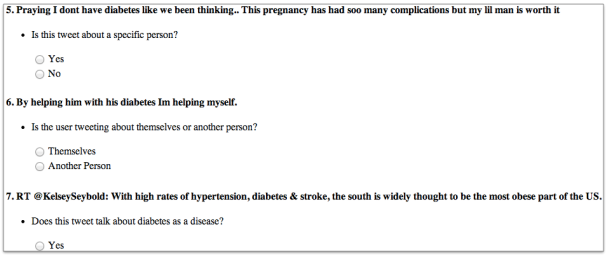
Next, we focused on removing all links from tweets as links are just noise for our machine learning model. After *preprocessing* the tweets, we fed the tweets into our Mechanical Turk labeling workflow. The process is described in more detail in the next section, but we essentially use Mechanical Turk to generate an accurately labelled corpus. As mentioned previously, we wanted each tweet in the corpus to be tagged with *self* if the individual is tweeting about him or herself with a disease, *other* if the person is tweeting about another person with a disease, and *general statement* if the tweet contains the disease keyword, but isn't in the previous two categories. This step further reduced the number of usable tweets due to potential error from the Turker's labels. Our corpus size from the *MTurk hit rate and tagging* was further reduced to 9,500 tweets.

Finally, we had to induce another *selection bias* on our annotated corpus. We wanted to train our model on an even distribution of tweets from each label. But, we had found in our research that roughly 80% of the accurately labelled corpus had the label of *general statement*. If we had used this entire corpus to train our model, we would have strongly overfit our machine model on tweets labelled with *general statement*. Thus, in order to obtain an even distribution across all labels, we had to use 10% of the data labelled *self*, another 10% labelled *other*, and a final 10% labelled general statement. This means we discarded 70% of the total labelled corpus. So, the final annotated data set that we split into test and training data was 3,000 tweets. As shown in **Figure 3**, only 1% of our original collected tweets were used in the final corpus. However, upon speaking with machine learning researchers, it is clear that such a low percentage of annotated tweets is not atypical for such a machine learning model.

5.2 Data Annotation with Mechanical Turk

Through Mechanical Turk, we leverage the power of anonymous people to label our tweets. We asked the Turkers to answer binary questions relating to the tweets and then programmatically filtered these contextualized tweets into three different buckets: *self* indicates that the tweeter themselves has the disease, *other* indicates that the tweeter is referring to someone else, such as a family member or friend, who has the disease, and *general statement* indicates that the tweet is either a general statement about the disease, a joke, or did not indicate anyone actually having the disease.

The Mechanical Turk process was far more challenging and time-consuming than first expected. We asked the Turkers a series of binary, unambiguous questions per tweet. Based on their answers to each of these questions for each tweet, our review process would automatically label a tweet with the appropriate label. This method of using numerous unambiguous questions in order to achieve one label per tweet (instead of simply asking one more ambiguous question per tweet), we were able to get high quality, correct hits that we were subsequently able to use. Our eventual hit acceptance rate was 55%, which was sufficient given how many tweets we sent to MTurk.


5. Praying I dont have diabetes like we been thinking.. This pregnancy has had soo many complications but my lil man is worth it

- Is this tweet about a specific person?
  - Yes
  - No

6. By helping him with his diabetes Im helping myself.

- Is the user tweeting about themselves or another person?
  - Themselves
  - Another Person

7. RT @KelseySeybold: With high rates of hypertension, diabetes & stroke, the south is widely thought to be the most obese part of the US.

- Does this tweet talk about diabetes as a disease?
  - Yes

Figure 4: Sample Mechanical Turk HIT

In order to verify the quality and integrity of each HIT, we embedded two control tweets in each HIT and verified that the Turker at the very least got those two correct. Each HIT had numerous questions for 20 tweets. Of these 20, two of them were control tweets that we randomly selected from a separate list of control tweets that we had made. These control tweets had clear and obvious labels to them. For example, "I have diabetes" was a control tweet that we used, which should obviously be labeled *self*. Turkers needed to correctly label both of the control tweets in their HIT in order for us to accept their HIT and for them to get paid.

After we had received our labelled tweets from Mechanical Turk, we had to then separate our data into equally represented sample sizes of data labeled self, other, or general statement. As we mentioned in the previous section, we had discovered that about 80% of tweets that had gone through our Mechanical Turk filter had the general statement label. Therefore, in order to train our machine learning classifier, we had to evenly distribute the number of tweets for each label. This meant that we could not include most of the general statement tweets in order to avoid overfitting our model. From this final pool of tweets, we created our training data set and our testing data set. Our final corpus was 3,000 tweets. We used 80% of this final corpus for training and 20% of our final corpus for testing.

We have created a deliverable tool, which we call our Data Collection Toolkit, that automatically filters and converts

tweets gathered from the Twitter scraper into a CSV file that is formatted correctly for input into Mechanical Turk. Also included in this deliverable is the Javascript used to create the Mechanical Turk HIT. This is a dynamic form that reads 20 tweets per HIT and provides binary questions for each tweet. There are four to five binary questions per tweet depending on previously selected answers. Once a question has been answered, it is dynamically replaced by the next question in the work flow. This method of providing a flow of binary questions was instrumental to the increased acceptance of HITs.

Furthermore, also included in the toolkit is the an automated review process once the HITs have been completed. This is in the form of a Python script that takes in as input the CSV with the worker's answers provided by MTurk and outputs a list of labelled tweets and a CSV, similar to the input CSV, detailing the approvals and rejections. This CSV is then uploaded back into the Mechanical Turk interface for the payment procedure. All of these Python scripts are packaged into an easy to use deliverable with a README outlining the complete data collection workflow as well as the inputs and outputs to the various programs.

5.3 Model Training with SKLearn

Now that we have discussed the data collection, filtering and labeling of our data, we will discuss the utilization of this data to train our supervised machine learning classifier. We split our labelled corpus into 80% training data and 20% testing data. The machine learning process was done in three steps: (1) feature generation, (2) feature selection, (3) train our classifier on the features that were selected.

In the first iterations of our machine learning process, we utilized the Natural Language Toolkit [2] APIs to do feature generation and supervised classification. We used unigram and bigram features to train a Naive Bayes classifier. However, this ultimately proved problematic for two reasons. First, there was no way to do any feature selection. Without any feature selection, many underrepresented features caused the model to inappropriately weigh them as being important, thus causing large amounts of noise that kept the accuracy of our classifier very low.

The second problem with the NTLK was related to using a Naive Bayes classifier. In a machine learning model, one must test the accuracy on both the training set and the test set. While using the NLTK and a Naive Bayes classifier, we had a 99% average training set accuracy, and only a 45% average test set accuracy. This indicates that we were seriously overfitting our model to our training data. Ideally, one would want the difference between the training set accuracy and the test set accuracy to be minimal. If that were the case, it would mean the classifier performs as well on previously seen data as it does on *unseen* data, indicating a very well-trained model (thanks to a good training data set) and a strong classifier.

As a result, we used another Python library to complement the NLTK called scikit-learn (sk-learn) [8]. We retained the usage of unigram and bigram feature generation from the NTLK, but with sk-learn we focused very heavily on feature selection.

With all of our unigram and bigram features that we had generated from our training set, we started off with a feature set of over 27,000 features. In order to get rid of useless features and only train our model with the most informative and useful features, we developed three feature selection processes to trim these unnecessary features from our model. First we utilize family-wise error selection to remove highly correlated tweets. This helped us reduce overfitting in our model, as well as the number of highly correlated features in our feature set. This, in turn, helped prune out redundancy within our features. Next, we used a percentile feature selection method to reduce the number of features that occurred infrequently. Finally, we utilized a technique called lemmatization. Lemmatization allowed us to group together the different inflected forms of a word so that they could be analyzed as a single word. For example, we reduced words such as 'talking', 'talked', and 'talks' to just 'talk'. Thanks to these feature selection methods, we were able to reduce our feature set from 27,000 to only 350.

After we created the refined feature set, we train our SVM Linear classifier to do word language disambiguation. We used an support vector machine classifier because it reduces overfitting and is a commonly used classifier for natural language processing problems, and is far more sophisticated and is more accurate than the Naive-Bayes model that we originally used.

Finally, we embedded this trained model into our prediction toolkit, which allowed for predictive labeling of tweets.

We utilize our Twitter scraper to feed tweets into our classifier. Our prediction toolkit scripts do so by formatting each tweet based on the username and tweet and putting them on separate lines in a file. Using the hyperplane created by the Support Vector Machine during the training process, the classifier labels each input tweet and outputs the tweet into separate files based on its label. The final result is a set of three files each holding the tweets associated with a specific label as well as their corresponding usernames. The user handles are what we believe can truly be useful for medical and social science researchers looking for potential participants for their clinical trials.

5.4 Sample Tweet Input

The following is a sample of the model classification process, given a sample tweet input.

1. We begin with a tweet using the keyword diabetes: *User @Cachuco: @driverRyanReed #DSDshirt is awesome! Im taking new steps to control my diabetes and joining the movement to Stop Diabetes!*

2. We then tokenize the tweet into unigram and bigram features:

   [('control': 1, 'and': 1, ('DSDshirt', 'is'): 1, 'is': 1, ('movement', 'to'): 1, ('Stop', 'Diabetes'): 1, ('and', 'joining'): 1, ('to', 'control'): 1, 'Stop': 1, ('taking', 'new'): 1, 'to': 1, ('to', 'Stop'): 1, ('driverRyanReed', 'DSDshirt'): 1, 'Im': 1, 'driverRyanReed': 1, 'awesome': 1, 'movement': 1, ('joining', 'the'): 1, ('Im', 'taking'): 1, ('awesome', 'Im'): 1, 'new': 1, ('steps', 'to'): 1, 'diabetes': 1, ('my', 'diabetes'): 1, 'DSDshirt': 1, ('diabetes', 'and'): 1, 'taking': 1, ('is', 'awesome'): 1, ('the', 'movement'): 1, 'steps': 1, ('new', 'steps'): 1, 'the': 1, 'Diabetes': 1, 'my': 1, ('control', 'my'): 1, 'joining': 1, 1)]

3. We utilize feature selection to trim away underrepresented features:

   (a) driverRyanReed

   (b) #DSDshirt is

   (c) awesome

   are examples of features that may not help indicate anything useful about a tweet related to diabetes.

4. Our trimmed feature set is converted into a sparse matrix that is readable by the model and compared to the features of our trained Support Vector Machine.

5. The tweets n-dimensional vector is updated, and based on its position in $(n-1)$-dimensional space, is given the classification of *Self*.

6. The tweet is added to our *Self* output file, signaling that researchers may want to contact the user @Cachuco about a potential diabetes trial.

6. EXPERIMENTAL RESULTS

Baseline Accuracy	Training Accuracy (Avg)	Test Accuracy (Avg)
33%	73%	69%

Figure 5: Table of Model Performance

6.1 Model Performance

Our final SVM classifier using Scikit-learn achieved vastly superior results to any of our original models while simultaneously minimizing overfitting. To test the model under a variety of circumstances, each time the model's accuracy was evaluated, we randomly selected which tweets would be included in the training set and which would be included in the test set. Additionally, we employed a form of cross validation, a procedure that repeats the random test/train split multiple times and outputs a composite score of multiple model runs, to get the most accurate estimate of test accuracy possible. On a final training set of 2,400 and test set of 600 tweets, we achieved an average accuracy of 73% for our training set, and an average accuracy of 69% for our test set. The source of the model accuracy increase is twofold: both from an increase in our annotate training corpus size and from the use of improved feature selection methods.

Much of the increased average accuracies can be attributed to our increased corpus size. As we can see in **Figure 6**, the accuracy of our model increases immensely as we boost the size of our tweet corpus. This is because our model was seeing more instances of certain features and could therefore select more relevant features on which to base the final model. However, **Figure 6** also shows that this increase in the average accuracy eventually plateaus, asymptotically approaching an accuracy value of approximately 0.62. Once this average accuracy was obtained, the corpus size alone had little bearing on the test accuracy of the model.

Instead, improved feature selection methods were the driving factor in increasing our model accuracy after hitting this corpus plateau. Using both unigram and bigram features and performing a Chi-Squared analysis on potential

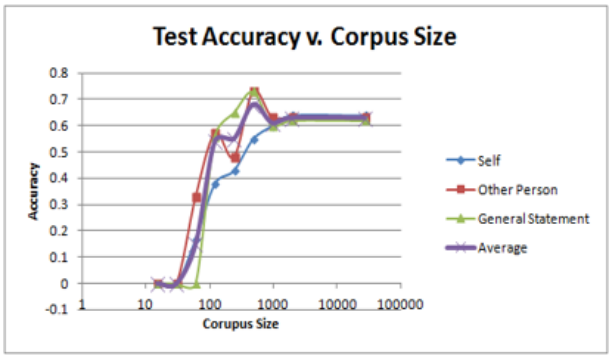


Figure 6: Test Accuracy in relation to Corpus Size (Logarithmic Scale)

features comparing expected and observed frequencies of features helped to improve our model accuracy.

However, selecting the most relevant features also plateaued at a certain point and on its own was not enough for a truly effective machine learning model. a A major challenge with this particular dataset was the lack of features that clearly signaled any of the three classifications. Instead, the model relied on an amalgamation of a large number of features each with low ability to determine classification. This can be seen most clearly using a Principle Component Analysis (PCA) of the test dataset (see **Figure 7**). Each input tweet in the test data is given a score for each model feature, creating a point in n-dimensional space. A Principle Component Analysis maps these n-dimensional points into two-dimensional space, taking into account only the two most influential features. As shown in **Figure 7**, the three categories are in no way linearly separable in two dimensional space, and therefore our top features have almost no bearing in determining classification on their own.

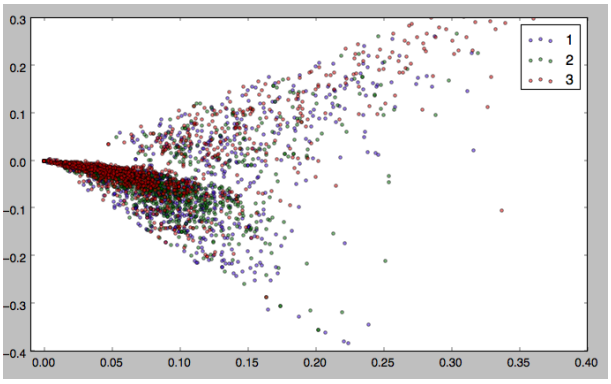


Figure 7: Principle Component Analysis: Data Separability through Linear Projection onto Two Dimensions

Increasing feature variance is therefore key to creating the best possible model. That is, combining like features together will increase their impact on one classification over another. To do this, we incorporated two methods: lemmatization and stemming. Lemmatization combines like features together by joining similar words. For example, a lemmatizer would combine the words 'add', 'adding', 'adds', and

'added' all as the word 'add'. This reduces the size of our potential dictionary of features while maintaining the general meaning behind each of the words in that dictionary. Similarly, stemming extracts the ending letters of longer words in the dictionary to reduce feature dictionary size significantly. As stemming can reduce word precision, it is used less aggressively and with greater discretion than lemmatization. Creating better features with the data available rather than selecting the original features provided to us was most effective in increasing our model accuracy rate.

6.2 Error Analysis

An important factor in improving the accuracy of the model was analyzing potential errors both in the machine learning model and in the annotated data used as the training set. Identifying these errors was a key component for better feature selection and model iteration.

6.2.1 Model Error

We evaluated the accuracies of our training and test sets using a standard L_0 norm. An L_0 norm is an error rate calculation that is based solely on whether or not our prediction was correct or not. The equation is as follows, assuming that $\hat{y}$ is our vector of predicted outcomes for a given classifier, and y is the correct set of outcomes:

$$||\hat{y}||_0 = \sum_{i=1}^{n} 1(y_i \neq \hat{y}_i)$$

Our summation simply gives us the number of predictions that we got wrong. We chose this instead of L_1 or L_2 norms because those both have a notion of being *more wrong* in certain cases. For example, for a certain classifier, predicting the label 1 when the correct label is actually 3 will give an error of 2 for an L_1 norm, and 4 for an L_2 norm. However, in our case, we believe that predicting the wrong label is all that matters, and that *which* incorrect label we predicted is not relevant. All that is relevant is the fact that we predicted the wrong output, which is why we are choosing to use an L_0 norm.

		Predicted Values		
		Self	Other	General
Actual	Self	162	23	39
	Other	24	139	35
	General	41	24	116

Figure 8: Model Confusion Matrix

To minimize model bias or overfitting, our model utilized confusion matrices during the initial training phase. Confusion matrices provide a visual representation for misclassifications by the trained model. We split our training set into two categories, 80% training set, and 20% evaluation set. For each tweet in the evaluation set, the confusion matrix details each misclassification so that we can evaluate the current features being used and adjust. Each row of the matrix represents the actual classification of the tweet, while each Column shows what are model predicted the classification to be. In **Figure 8**, we can see that our model is currently overconfident in predicting the *Self* category, leading to a high number of misclassifications of *General Statement* as *Self*. Looking closely at the features employed by the model, this led us to eliminate several features like 'I can', which referred both to *Self*, ("I can beat my diabetes"), and to *General Statement*, ("I can't believe diabetes kills so many people").

6.2.2 Training Data Error

Although Mechanical Turk is an excellent application for quickly and cheaply annotating data, the incentive system it creates for workers to rush through tasks raises issues regarding the quality of the resulting work. To minimize Mechanical Turk worker abuse, two control tweets were embedded in each HIT of twenty tweets provided to workers. These tweets were chosen due to their obvious labels, for example "I have diabetes", which must have a label of *Self*. Since the correct label for these tweets is known, the system can programmatically check whether the Turkers are providing genuine answers at a cost of only 10% of the potential annotated tweets. We randomized these from a list of about twenty tweets.

During the review process, software checked the two control tweets and rejected any HIT where either was answered incorrectly. However, even with these control tweets, there was a clear misclassification of many of the tweets we received. This could have been due to the highly ambiguous nature of many of the tweets or due to the incompetence of the Turkers. There is a tradeoff between cost and quality in this case. If we were to add more control tweets, the level of accuracy would increase, but the number of labeled tweets would decrease per HIT. We would therefore need more HITs to generate the same number of labeled tweets, thus increasing the cost.

To mitigate the ambiguous nature of tweet tagging, in subsequent sets of HITs on Mechanical Turk the procedure was revised to ask workers a series of binary questions rather than a direct classification. For example, a tweet that dealt with diabetes as a disease, spoke about a person with diabetes, and was a reflexive statement was labeled by the system as *Self*. The switch to a binary question system increased both the acceptance rate of HITs to 55% and the quality of accepted tweets, which were more than 80% accurate.

7. MODEL APPLICATIONS

There are a variety of applications and uses for our overall model. An example of this includes identifying individuals for clinical trial participation. Due to the robustness of our model, the scope is not just limited to health care. Since we are extrapolating specific information from tweets, namely identifying individuals, we can extend the uses of our model to further areas such as targeted advertising.

7.1 Identifying Trial Candidates

The use-case that we are focusing on is the identification of individuals for clinical trials. As mentioned, the current process of examining medical records and individually scanning tweets is a long and arduous process. The hope is to use machine learning techniques to quickly and effectively scan through tweets and output users who have a given disease. From this point, clinicians can contact these individuals for potential participation in various clinical trials. This method will vastly decrease the time and cost invested in

searching for individuals with particular diseases.

7.2 Targeted Advertising

Although the initial motivation was healthcare related, there are more far-reaching applications of machine learning model. Targeted advertising is highly used in today's social world. Companies like Facebook and Google are constantly attempting to provide the user with ads that relevant in their lives. Our machine learning classifier could be used to scan tweets and identify individuals that are tweeting about particular products. For example, if instead of a disease, a product such as Pepsi was an input to the model, as well as tweets related or mentioning Pepsi, our classifier would output those individuals who are referring to themselves in relation to the product. From here, advertisers would be able to target these users and provide them with relevant advertisements.

7.3 Deliverable Software Suite

We were successfully able to package three different deliverables: our Data Collection Toolkit, our SVM classifier, and our Packaged Prediction Tool. As of now, the deliverables that interface with users and user input are all based on the command line. The Data Collection Toolkit interfaces with a user via the command line, as does the Packaged Prediction Tool. The SVM classifier is simply included as part of the Packaged Prediction Tool.

8. FUTURE WORK

We have been and will continue to look for ways to improve the sophistication and accuracy levels of our SVM classifier. While our model is not yet perfect, we believe it predicts fairly accurately for our purposes. However, we will always be looking for ways to make a more accurate and sophisticated machine learning model. We believe that we can do so through additional features and with more data. We do a decent amount of feature selection to eliminate noisy, insignificant features in our model. We already do five different types of feature selection, which is why we believe that there is a limited amount of additional feature selection that we can do. So, we believe that the future of increasing our model accuracy and sophistication lie with having more features than simply unigrams and bigrams. We will experiment with trigrams and potentially more n-grams, as well as "non-word features" like having a feature for if a tweet is in the present or past tense, or if it contains a certain word phrase or clause, etc.

We are also looking further increase the modularity and flexibility of our Data Collection Toolkit by allowing users of our packages to specify the disease for which they seek Twitter data. We currently have only collected data related to diabetes. But, our model is already modular in that the disease of interest makes no difference when learning and predicting the classifications. We want our entire system to be modular - so we will be looking to provide a means for a user to specify the disease for which they want Twitter data.

9. ETHICS

There is some room for concern regarding the ethics associated with the data that our Data Collection Toolkit collects from Twitter users wand what it is used for. The Twitter platform is an inherently public forum, and as we are only looking at those users with public accounts, there is no ethical violation of data mining in our system. However, it has been brought to our attention that the idea of trying to determine something as personal as disease diagnosis via social media content may be considered invasive to many users who are not fully aware of Twitter's privacy policies.

However, what we are doing with Twitter data is not unheard of by any means. It is hardly different from targeted advertising, for example, and far better than the government using our tool over *private* social media data, for example. Additionally, we are simply scaling up a practice of determining clinical trial patients through social media that is already implemented in research facilities and clinics across the country. These businesses would continue their current practices manually. Our classifier itself is built as an ethical system in its current form because it deals primarily with public data.

10. CONCLUSION

While our model is not perfect, we believe it is a great first step towards providing medical and social science researchers with a truly useful and productive tool to help them find people who would be good candidates for clinical studies. With an average training accuracy of 75% and an average test accuracy of 70%, there is still room for improvement but we believe what we have is a strong foundation upon which improvements can be built. We hope that our system can benefit medical and social science researchers and eventually the people with these diseases in the long term.

11. REFERENCES

[1] Inc. Amazon.com. Amazon mechanical turk. 2005.

[2] Edward Loper Bird, Steven and Ewan Klein. Natural language processing with python. *O'Reilly Media Inc.*, 2009.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing With Python*. O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, first edition, 06 2009.

[4] Andrea Freyer Dugas, Mehdi Jalalpour, Yulia Gel, Scott Levin, Fred Torcaso, Takeru Igusa, and Richard E. Rothman. Influenza forecasting with google flu trends. *PLoS ONE*, 8(2):e56176, 02 2013.

[5] Christian Koepp. Twittersearch. 2013.

[6] Marco Lui. Stand-alone language identification system.

[7] Roberto Navigli. Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2):10:1–10:69, February 2009.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Anna Rumshisky. Crowdsourcing word sense definition. In *Proceedings of the 5th Linguistic Annotation Workshop*, LAW V '11, pages 74–81, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[10] H Andrew Schwartz, Johannes C Eichstaedt, Lukasz Dziurzynski, Margaret L Kern, Eduardo Blanco, Stephanie Ramones, Martin E P Seligman, and Lyle H Ungar. Choosing the right words: Characterizing and reducing error of the word count approach. In *Proceedings of *SEM-2013: Second Joint Conference on Lexical and Computational Semantics*, 2013.