KTH Computer Science
and Communication

A sentiment-based chat bot

Automatic Twitter replies with Python

ALEXANDER BLOM
SOFIE THORSEN

Bachelor's essay at CSC
Supervisor: Anna Hjalmarsson
Examiner: Mårten Björkman
E-mail: aleblom@kth.se, sthorsen@kth.se

Abstract

Natural language processing is a field in computer science which involves making computers derive meaning from human language and input as a way of interacting with the real world. Broadly speaking, sentiment analysis is the act of determining the attitude of an author or speaker, with respect to a certain topic or the overall context and is an application of the natural language processing field.

This essay discusses the implementation of a Twitter chat bot that uses natural language processing and sentiment analysis to construct a believable reply. This is done in the Python programming language, using a statistical method called Naive Bayes classifying supplied by the NLTK Python package. The essay concludes that applying natural language processing and sentiment analysis in this isolated fashion was simple, but achieving more complex tasks greatly increases the difficulty.

Referat

Natural language processing är ett fält inom datavetenskap som innefattar att få datorer att förstå mänskligt språk och indata för att på så sätt kunna interagera med den riktiga världen. Sentiment analysis är, generellt sagt, akten av att försöka bestämma känslan hos en författare eller talare med avseende på ett specifikt ämne eller sammanhang och är en applicering av fältet natural language processing.

Den här rapporten diskuterar implementeringen av en Twitter-chatbot som använder just natural language processing och sentiment analysis för att kunna svara på tweets genom att använda känslan i tweetet. Detta görs i programmeringsspråket Python med en statistisk metod kallad Naive Bayes classifying med hjälp av Python-paketet NLTK. Rapporten gör slutsatsen att appliceringen av natural language processing och sentiment analysis är enkelt när det görs inom detta isolerade område, men att utföra mer komplexa uppgifter ökar svårighetsgraden markant.

Statement of collaboration

There was no clear distribution of the background reading workload so it will be considered an equal split. As for the reading part this table reflects who mainly wrote a specific part:

1 Introduction	Alexander
2 Background	N/A
2.1 Natural Language Processing	Alexander
2.2 Sentiment analysis	Sofie
2.3 Chat bots	Sofie
3 Problem statement	Sofie
4 Implementation	Sofie
4.1 Filtering	Alexander
4.2 Classification	Sofie
4.3 Running the bot	Alexander
5 Results	N/A
5.1 Cross-validation test	Alexander
5.2 Evaluation of replies	Sofie
6 Discussion	Alexander

Contents

1 Introduction

A chat bot, hereafter bot, is a computer program that is designed to simulate a conversation, either by being human-like or purely informational. For example a informational bot may reply with information about a train schedule or automatically text you with storm alerts.

Creating bots can be a fun and interesting way of applying computer science knowledge while also exploring topics such as natural language processing (NLP) and overall text processing. Twitter lends itself as the perfect platform for this given the huge amount of people and data available.

Neither bots nor NLP are new concepts as will be covered in later sections, but applying them to a modern platform as Twitter is an upcoming phenomenon with sometimes amusing results. Creating something more than a simple keyword-responder is more of a challenge and that is what this essay will discuss, it will describe the implementation of a sentiment-based Twitter bot which responds to tweets in different ways depending on the mood of each tweet.

1.1 Definitions, acronyms and abbreviations

API	Application programming interface
ASCII	American Standard Code for Information Interchange, a character-encoding scheme
Bot	Short for robot, a software application that run automated tasks
EULA	End-user license agreement
NLP	Natural language processing
REST	Representational state transfer
Tweet	A short message (max 140 characters) sent using Twitter
URL	Uniform resource locator, also known as web address

2 Background

2.1 Natural Language Processing

Natural language processing, denoted NLP, is a field in computer science which involves making computers derive meaning from human language and input as a way of interacting with the real world. Alan Turing proposed the "Turing test" in his now famous article[1] as a criterion on intelligence. It is based on the ability of a computer program to impersonate a human holding a conversation with a human judge, with the judge not being able to distinguish between a computer or a human.

Earlier attempts with NLP often tried to hand code a large set of rules to cover most cases. Today, modern NLP techniques are based on machine learning and especially statistical learning which uses a general learning algorithm combined with a large sample, a corpus, of data to learn the rules.

There are different types of machine learning algorithms which makes use of so called features, hereafter just called words for the sake of clarity, generated from input data, for example by using decision trees to generate hard if-then rules similar to the hand coded rules, or using a statistical model which produces probabilistic decisions based on attaching weights to each input word[2]. A statistical method will be explored further in this essay.

2.2 Sentiment analysis

Sentiment analysis is an application of NLP. Broadly speaking, sentiment analysis is the act of determining the attitude of an author or speaker, with respect to a certain topic or the overall context. What this attitude means differs since there are different aspects one could examine. This could for example be emotional state of the communicator, the intended emotional communication (the emotional effect the communicator wishes to have on the receiver) or the evaluation of an event[3].

In this essay, sentiment analysis will refer to the perhaps most basic form, which is if a text or sentence is positive or negative, also called the polarity.

2.3 Chat bots

SHRDLU was a program developed by Terry Winograd at M.I.T. in 1968-1970[4]. With the program a user was able to interact with the computer using basic English terms. The user could instruct SHRDLU to move around objects in a simple “block world” containing objects such as blocks, cones and pyramids. As SHRDLU included a basic memory, the program could figure out what the user meant by looking at previous entries. For example, a user could ask SHRDLU to “put the red cone on the blue block” and then by just referring to “the cone” SHRDLU would track that to be the red cone that the user previously mentioned.

Another early successful NLP program was ELIZA, written at M.I.T. by Joseph Weizenbaum between 1964 and 1966, that used simple pattern matching techniques to simulate an intelligent conversation[5]. The most famous script operated by ELIZA was DOCTOR which would simulate a Rogerian psychotherapist. Weizenbaum himself stated that ELIZA, with the DOCTOR script, provided a parody of "the responses of a nondirectional psychotherapist in an initial psychiatric interview."[5, p. 188]. The statement relates to the fact that it is somewhat acceptable to respond to questions with another question in the therapeutic situation. When a user exceeded the knowledge base, DOCTOR could therefore just respond with a generic question, for example, responding to “My stomach hurts” with “Why do you say your stomach hurts?”.

Twitter

There exists a large amount of Twitter bots[6], most of which are very simple and react to some part of the input and typically respond with a static answer. One such example is the @Betelgeuse_3 bot[7] which responds to a user saying “beetlejuice” three times with “IT’S SHOWTIME!”, a reference to the popular movie Beetlejuice.

Twitter has explicitly disallowed bots that simply respond to keyword matches in their EULA, however they seem to be permissive when it comes to well-behaved bots[8].

Sentiment on Twitter

There are several sites[9][10] and articles[11] which deal with sentiment analysis on Twitter, however, none that we can find deal with implementing a bot which determines the sentiment of the tweet it is responding to. They seem to deal mainly with determining sentiment for a large number of tweets, for example related to a company or product. This is not really surprising given the general application of sentiment analysis, opinion mining.

3 Problem statement

The purpose of this essay is to describe the implementation of a simple sentiment based Twitter bot. When implementing a bot that can stay active on Twitter several requirements has to be met. On the behalf of determining the sentiment there has to be some consideration on which approach to use. Tweets are short and may not always consist of proper sentences. Context can be hard to determine and badly constructed tweets can harm the learning if a statistical method is used.

Besides from determining the sentiment of a tweet, the bot must also respond to the tweet properly. What this means is that the bot must adhere to to the tone of the tweet and also be contextual, that is, respond in a way that a human would. Simply replying with a static predefined sentence could easily cause Twitter to shut the bot down, as well as the result being far less interesting than if the bot generated a custom response. This leads to the following questions:

- How can the sentiment of of a tweet be determined?
- Can the sentiment be used to construct a proper reply?

Even though it would have been interesting to respond to user by context, for example respond to "I hate my ugly car" with something about the car, it is out of scope for this essay.

4 Implementation

The implementation can be considered to have three parts. First, tweets has to be retrieved in order for the bot to learn from them. This is a small part of the system but could be tweaked in order to find a broad set of tweets to learn from, for example searching for appropriate tweets using keywords combined with other filtering methods.

The second part is training the bot on the retrieved set of tweets. This means that tweets has to be processed and associated with different sentiments. A probabilistic approach could be used, but with caution, since malformed tweets can teach the bot wrong behaviour.

The last part is the most challenging and depends on the learning from the collected tweets. This part contains the actual artificial intelligence, that is, the bot should be able to find a tweet, determine the sentiment and thereafter respond in a way that makes sense for the recipient.

Tools

The implementation is done using the programming language Python, it was chosen based on its popularity and access to the NLP Toolkit (NLTK)[12]. Together with Python the library Tweetstream is used to access the streaming API's of Twitter. MongoDB is used for persistence.

Based on the NLTK website, the library provides access to "...over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning". While we will only use a fraction of these it's good to have a complete set of other NLP algorithms.

Most code examples in this essay will be using NLTK but will be general enough to be understood without deeper knowledge of library specifics. However, some Python knowledge may be useful.

4.1 Filtering

Twitter provides a filter method in their streaming API[13]. It allows you to filter tweets based on several (up to 400) keywords. You can also filter by location and other criteria. The reason for them doing this is because of the huge amount of data in the raw stream which they do not want to expose to everyone.

The stream includes each tweet with metadata such as the language the message was written in. While this is mostly a guess from Twitter's side it could be used to exclude non-English tweets. If this is not sufficient the location (which is more specific) could be used to restrict the tweets. This though has the risk of being too specific while also not guaranteeing tweets to be in a specific language. A combination of all methods is probably the best option to reduce their individual risks.

When filtering tweets, the following keywords will be used. The keywords are based on several resources[14][15] and also some trial and error. Simple sentiments like "happy" and "sad" was removed since they triggered common and non-valuable tweets like "happy birthday" which does not contribute as training data. "Hate" was also removed, since the amount of tweets saying "I hate this/that" by far outnumbered more interesting tweets, that is, tweets with a more complex sentence structure.

Table 1. List of filtering keywords.

accepting	delighted	interested	satisfied
amazed	devastated	kind	sick
angry	disappointed	liberated	spirited
blessed	ecstatic	lucky	sympathetic
calm	energetic	overjoyed	thankful
comfortable	excited	peaceful	thrilled
confident	fortunate	playful	understanding
content	free	receptive	wonderful
courageous	great	reliable	

Tweets may also contain a lot of noise that is not relevant for processing, such as mentions/replies, hashtags, retweets and URLs. These can fortunately be stripped with a simple regular expression (regex). Regex is also used to strip all non-alphanumerical characters, including smileys. While for example smileys could be used for determining sentiment, which others have done[16], this is outside of the scope of this essay.

The regexes used for cleaning tweets are the following (in Python syntax):

```
username = r'@([A-Za-z0-9_]+)'
hashtag = r'\B#\w\w+'
retweet = r'^RT'
url = r'https?:\/\/t.co/[\w]*'
garbage = r'[^[:ascii:]]+'
```

The garbage regex simply removes all non-ASCII characters which has the effect of removing unicode smileys as well as foreign characters.

4.2 Classification

For the bot to be able to learn to distinguish between positive and negative tweets it will need to train on a set of manually annotated data. Retrieving tweets with filtering is not enough to start the learning since every tweet in the training set must have an associated sentiment. In this case we will restrict ourselves to using just positive or negative as sentiments even though it would be possible to be more precise (sad and angry are both negative sentiments for example). As the bot learns it will later be able to determine this for itself, but to do that a large set of annotated data is required.

To speed up the rate at which tweets could be annotated a simple command line interface was developed. It lists each tweet and allows the user to annotate it using shortcuts for:

- Discard - don't annotate this tweet, it is thrown away
- Skip - skip this tweet for now, it will be shown again next time the tool is run
- Positive - annotate as positive
- Negative - annotate as negative

The annotations are inserted into a separate database together with the tweet for later use when training the bot before running it.

Here is a example of how a run of the tool looks:

```
> Great something is definitely wrong with my car
> (d)iscard, (s)kip, (p)ositive, (n)egative: n

> So this is what it feels like to be stress free :)
> (d)iscard, (s)kip, (p)ositive, (n)egative: p
```

Having acquired an annotated training set of about a 100 tweets we want to send them to an extractor that determines which words that are relevant for the analysis. This is done after first removing the shortest words (a limit on at least three letters should be sufficient).

Applying the extractor to the tweets with sentiments will, with the use of NLTK functions, return a finished training set. Passing this training set to a classifier will cause the classifier to calculate, using prior probability and the frequency of words, the likelihood of a certain sentiment. When passing a new tweet to the classifier it will now determine the sentiment.

Naive Bayes Classifier

Most NLTK classifying functions are based on the Naive Bayes Classifier which is a simple probabilistic model based on Bayes theorem[17]:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Simplified, this calculates the probability of an outcome based on another, strongly assuming independence, hence why the classifier is called naive. As an example, mad and furious both describes an angry sentiment and may be related, but the Naive Bayes Classifier considers these independently contributing to the fact that the user is angry.

The advantage of using Naive Bayes is that the amount of training data required to estimate probability is small. Because of the independence assumption, only variances of variables (for each class) need to be determined instead of using an entire covariance matrix[18][19].

Example of training

As an example, say that the word "angry" appears in 1 of 5 tweets classified as negative and in none of the tweets classified as positive. This means that the likelihood of a "negative" sentiment, also called label, will be multiplied by 0.2 when "angry" is seen as part of the input.

As a continuation, the actual source code for some NLTK methods will be discussed.

```
def train(labeled_featuresets, estimator=ELEProbDist):
    ...
    # Create the P(label) distribution
    label_probdist = estimator(label_freqdist)
    ...
    # Create the P(fval|label, fname) distribution
    feature_probdist = {}
    ...
    return NaiveBayesClassifier(label_probdist, feature_probdist)
```

First, observe the `label_probdist`, which is the prior probability of each label, that is, the probability of a label with regards to the total amount of training data. For example, if the number of negative tweets were equal to the number of positive tweets, the probability of the two labels would be equal:

```
print label_probdist.prob('positive')
> 0.5
print label_probdist.prob('negative')
> 0.5
```

The other probability object is the `feature_probdist` which is an associative array of the word/value probability. This object links together expected likelihood with a word and label. It is therefore possible to examine the probability for a label given that the input contains a certain word. To clarify, take the following example.

The code snippet below (based on mock data) prints the `feature_probdist` object. The output lists for every label/word pair, the associated ELEProbDist object and how many samples the data is based on.

```
print feature_probdist
> {('negative', 'contains(view)'): <ELEProbDist based on 5 samples>,
> ('positive', 'contains(excited)'): <ELEProbDist based on 5 samples>,
> ('negative', 'contains(amazing)'): <ELEProbDist based on 5 samples>,
> ...}
```

So, say that we want to check the probability of the input to be negative when the input contains a certain word, we can do the following:

```
print feature_probdist[('negative', 'contains(amazing)')].prob(True)
> 0.076923076923076927
```

This states that the probability for the input to be negative is about 7.7% when the input contains the word ‘amazing’.

The `label_probdist` and the `feature_probdist` probability objects are used to create the actual classifier.

Example of classifying

When a classifier has been trained, it can be evaluated by first manually annotate a tweet and then compare the result from the classifier. For example, say that we have the tweet "Alex is my buddy" and that we state that the label is "positive". We can then let the classifier classify the tweet and look at the result. Say that the training set included a positive tweet that said "My buddy is the best". Then the classifier could detect the positive sentiment associated with "buddy":

```
tweet = 'Alex␣is␣my␣buddy'
print classifier.classify(extract_features(tweet.split()))
> positive
```

So in this mocked scenario, the classifier was correct. Looking at how this works internally, the word set of the tweet is passed to be analyzed. The dictionary can then say that the tweet in fact contains the word "buddy".

```
print extract_features(tweet.split())
```

```
> { ...
> 'contains(buddy)': True,
> ... }

def classify(self, featureset):
 # Discard any feature names that we've never seen before.
 # Find the log probability of each label, given the features.
 # Then add in the log probability of features given labels.
 # Generate a probability distribution dictionary using the
 # dict logprod.
 # Return the sample with the greatest probability from the
 # probability distribution dictionary
```

The associative array of the word set gets passed to the method classify. The first thing happening is that words not known by the classifier gets discarded, and the resulting associative array is obtained.

Now, for each label, the logarithmic probability gets obtained. If the number of positive tweets were equal to the number of negative tweets the probability of the two labels would be 0.5 each as stated earlier. The logarithmic probability would therefore be -1 since the log base 2 of 0.5 is just that:

```
{'positive': -1.0, 'negative': -1.0}
```

Then for each label, the items in the word set is looped through and the logarithmic probability of each item is added to `logprod[label]`. As an example, take the tweet "My car is not pretty" with the premises that the word "pretty" would be part of fewer positive tweets than the word "not" would be part of negative tweets. This would result in the output "negative" since the negative word "not" weighs more than the positive word "pretty". It has therefore nothing to do with the arrangement of words, a tweet like "This song is not bad" would in this case be classified as "negative" even though it is positive.

4.3 Running the bot

When the bot is running it will monitor new tweets from the stream and carefully choose ones that it can create a believable response to. The choice depends on the same rules that are used for cleaning a tweet in the filtering phase. The relative percentage of the sentiment will also need to be high for the tweet to be chosen. This way we can minimize the risk of a wrong reply while also making sure than we do not respond too early.

Note that the bot can be very picky when choosing which tweets to respond to based on the very large volume of tweets posted in even a second of time[20]. This means it can also restrict itself to only responding to longer tweets, also to maximize the chance of a proper sentiment.

When it has chosen a specific tweet it will choose one of several prewritten answers based on the sentiment of a tweet and the keywords it contains. By using the keywords we can be more specific as it gets possible to separate on a more narrow level than negative and positive. If the bot for example knows that a tweet contains "angry" it could use that information to be more custom in its response. With that said, the response sentences was still constructed to be general to fit as many tweets as possible.

The replies are below, divided into two tables for positive and negative respectively:

Table 2. List of positive replies.

Keyword	Positive reply
amazed	Fantastic!
blessed	Bless your tweet :)
excited	I can see why you're excited, sounds awesome! Sounds great :D
free	Love that feeling! Free like a bird?! Tweet tweet! Best feeling ever :)
great	Nice:) My day got even better hearing that:D
kind	True words.
lucky	I agree
sick	Sick! :D

Table 3. List of negative replies.

Keyword	Negative reply
amazed	That doesn't sound to good...
angry	:(You're scaring me with your anger :(I used to be that angry. Then my head exploded
devastated	That's not fun at all, wish I could help :<
disappointed	Yeah, that's disappointing right there.
excited	Why not?
free	Ouch :/
great	Hehe, sorry but your irony is funny :> That sucks :(
kind	It's that kind of day huh.
lucky	That's scary :<
sick	Are you sick of being sick? I feel you, struggled with my cold for 2 weeks :(

The response will be sent to the user using the Twitter REST API which makes the response come from a specific account. While this may seem obvious it is important to provide the bot with a real-looking user account, perhaps complete with a history separate from the automated replies.

The bot will run continuously but only respond to a small number of tweets per day in an effort to remain as human as possible while also not be perceived as "spammy".

5 Results

5.1 Cross-validation test

To evaluate the the trained classifier a cross-validation test was performed where the annotated data was partitioned up into two complementary subsets, performing training on the training set and then validating the classifier against the validation set. Validation was done by first classifying a text and then comparing it to the manually annotated sentiment. Different partitions are performed and the result is then averaged.

For our purposes a simple 5x2-fold cross-validation is sufficient[21]. It can be implemented by first randomly shuffling the annotated data and then splitting it into two equally sized subsets d0 and d1. We then train on d0 and test on d1 followed by doing the opposite. This process is then repeated five times averaging the measurements to a single value.

Here is the implementation, `validate()` is a function which takes a pivot and compares the two parts:

```
def cross_validate(tweets):
    """Perform a 2-fold cross-validation"""
    random.shuffle(tweets)
    sample1 = validate(tweets, 0.5)
    tweets.reverse()
    sample2 = validate(tweets, 0.5)

    return (sample1 + sample2) / 2

tweets = ...

times = 5
print 'Correct:␣{}'.format(sum(map(cross_validate,
    repeat(tweets, times))) / times)
> Correct: 0.664705882353
```

This is not a great correctness percentage, it would give a large amount of false positives and vice versa. The classifier can however also give a probability distribution of each label which serves as a great tool for being picky. Requiring a sentiment classification to have a probability of 99% gives the following output:

```
> Correct: 0.972549019608
```

A correctness of 97% is obviously much better but it needs to be noted that this is partly because we only use the tweets the classifier is certain about. This is not a problem for the bot though.

5.2 Evaluation of replies

To get some feedback on the generated answers, we decided to conduct a survey in order to try to measure how well the answers of the bot fit in to the context of the tweets. The questions was simple, the respondent was faced with a tweet and a response and was asked to give his/hers view on how good the answer was on a scale from 1 to 10, where 1 refers to really bad and out of context and 10 to be human-like and accurate. The survey was anonymous, filled in online, and sent out to responders via social media.

As we used our own Facebook accounts to send out the survey, we can assume that the age of the responder is somewhere between 20-30. This should not have any real significance other than the assumption that those responding actually knows what Twitter/tweets are (if the age span was higher, some responders may not have known what the survey was about).

Selection of questions

As the stream of tweets, even though it is filtered, contains a lot of noise we decided to have a limit of correctness to determine if we respond to a tweet or not. From the discussion earlier, this limit was set to 99%. By using this limit we hoped that the selected tweets would not just have a higher quality language wise, but also that the bot responded correctly.

With this premise we let the bot respond to the stream of tweets to get our set of responses. Responses that were "duplicates" in the meaning that they had used the same keyword to generate a response were removed. The reason of that was that we wanted to showcase the responses from as many keywords as possible. Besides from filtering duplicates we just took the first 10 responses the bot had generated.

Survey results

The survey got 35 responses, the survey questions and their average rating are summarized below:

Table 4. List of survey results.

Question	Rating
Tweet: Thanks Mother Nature for the free two day car wash Reply: Best feeling ever :)	6.1
Tweet: I don't know if I'm sick or if I have allergies Reply: I feel you, struggled with my cold for 2 weeks :(	8.2
Tweet: Last week I took him out like this & he got sick Reply: Are you sick of being sick?	3.8
Tweet: the buckwild crew is so annoying on twitter right now i'd prob watch it still but they should be lucky they got on mtv in the first place Reply: That's scary :<	3.8
Tweet: A good day of singing and recording with the lovely @KattW_. Excited to see how today's efforts turn out! Reply: Sounds great :D	7.5
Tweet: excited just one more day excited Reply: I can see why you're excited, sounds awesome!	5.9
Tweet: I get a burrito tonight, but it's not the kind I'm craving. Reply: It's that kind of day huh.	6.8
Tweet: K. Well I just lost all the people I talk to in 5th hour over something that wasn't about me in the first place... Great. Reply: Hehe, sorry but your irony is funny :>	5.2
Tweet: Good morning everyone, lets make it a great day and keep smiling!! Reply: My day got even better hearing that:D	8.3
Tweet: i am still angry about the time i wasted watching gummo and i basically sit around all day wasting time so you know it was bad Reply: You're scaring me with your anger :(	4.9

It is clear that some replies where not good enough (those below a 5 rating) but the rest can be considered believable. Some of the answers may not be totally contextual, this is expected as we ignore context, instead they can be seen as a human making a funny reply.

6 Discussion

NLP is a very broad and interesting topic with a lot of applications. While it is simple to apply it in an isolated fashion (as in this essay) it can be harder to create something more complex due to the multiple steps required to do even the smallest things.

Using the NLP Toolkit however was easy and a basic implementation was completed quickly. The developed command line tool also helped when classifying a large amount of tweets. Choosing Python made it easy to implement validation as well as using 5x2 cross validation.

The average tweet is often badly formed, both in regards grammatical correctness as well as to its content. For the purpose of replying to a tweet the content doesn't really matter as much as the language correctness, especially considering abbreviations, misspellings and more informal language. These things can completely change the interpretation of a tweet to a computer. The content hardships is a bit different, for example, "kind" can be used to describe a personal trait, as well as being used as an informal way, such as "that kind of way". This shows that sentiments can be used to construct a human-like reply, but not in all cases. To expand the implemented chat bot it could be interesting to try to correct misspellings, understand abbreviations and informal language.

The tweets the bot chooses to reply to are mostly well formed because of the requirement of a 99% probability but this could be increased further by using more metrics such as requiring the tweet author to have a minimum number of followers etc.

The generated responses mostly fits the tweet well but due to the low number of prewritten responses the replies will repeat themselves quickly. The easy solution is to write more responses or perhaps use a more advanced way of composing them.

In conclusion, this essay describes how to implement a sentiment based Twitter bot which uses the sentiment to construct a reply. Determining the sentiment was easily done, and did not require a huge amount of data. Constructing a proper reply from the acquired sentiment also worked fairly well, but was not without problems.

Bibliography

[1] Alan M. Turing, *Computing Machinery and Intelligence.* Mind LIX (236): 433–460, 1950.

[2] Joakim Nivre, *On Statistical Methods in Natural Language Processing.* `http://stp.lingfil.uu.se/~nivre/docs/statnlp.pdf` retrived 2013-04-06, 2002.

[3] Adam Westerski, *Sentiment Analysis: Introduction and the State of the Art overview.* 2009.

[4] Terry Winograd, *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language.* MIT AI Technical Report 235, February 1971.

[5] Joseph Weizenbaum, *Computer power and human reason: from judgment to calculation.* W. H. Freeman and Company, 1976.

[6] Kashmir Hill, *The Invasion of the Twitter Bots.* `http://www.forbes.com/sites/kashmirhill/2012/08/09/the-invasion-of-the-twitter-bots/` retrieved 2013-02-16, 2012.

[7] Colin Mitchell, *Twitter user Betelgeuse_3.* `https://twitter.com/Betelgeuse_3` retrieved 2013-02-16, 2013.

[8] Colin Mitchell, *The Current State of Bots on Twitter.* `http://muffinlabs.com/2012/06/04/the-current-state-of-bots-on-twitter/` retrieved 2013-02-16, 2012.

[9] *Sentiment140.* `http://www.sentiment140.com/` retrived 2013-04-06, 2013.

[10] *TweetFeel Twitter Sentiment.* `http://www.tweetfeel.com/` retrived 2013-04-06, 2013.

[11] Alec Go, Richa Bhayani Lei Huang, *Twitter Sentiment Classification using Distant Supervision.* `http://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf` retrived 2013-04-06, 2009.

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

[13] Twitter, *POST statuses/filter.* `https://dev.twitter.com/docs/api/1.1/post/statuses/filter` retrieved 2013-02-16, 2013.

[14] Richard Niolon, *List of Feeling Words* `http://www.psychpage.com/learning/library/assess/feelings.html` retrieved 2013-04-06, (n.d.).

[15] Finn Årup Nielsen, *A new ANEW: Evaluation of a word list for sentiment analysis in microblogs*, 2011.

[16] Owoputi et al. *Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters.* Technical Report, Machine Learning Department, CMU-ML-12-107, September 2012.

[17] Weisstein, Eric W, *Bayes' Theorem.* From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/BayesTheorem.html` retrieved 2013-04-04, 2013.

[18] Harry Zhang, *The Optimality of Naive Bayes.* FLAIRS Conference 2004: 562–567, 2004.

[19] Irina Rish, *An empirical study of the naive Bayes classifier.* IBM Research Report, November 2001.

[20] Daniel Terdiman, *Report: Twitter hits half a billion tweets a day.* `http://news.cnet.com/8301-1023_3-57541566-93/report-twitter-hits-half-a-billion-tweets-a-day/` retrieved 2013-04-06, October 2012

[21] Thomas G. Dietterich, *Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms.* Neural Computation, vol 10: 1895–1923 1998.