# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

## Abstract

For my final project, I focused on figuring out what would make email better. The main part of my project focuses on performing sentiment analysis on my email feed. My program works by using Node and Express to create a simple web server. I then connect to the Gmail API to with an authentication token to receive all my emails. When the server starts, the call to the Gmail API is made and I perform sentiment analysis using mainly the AFINN database. I also made optimizations along the way which I will mention in the paper.

## 1. Problem Statement

E-mail has been stale for a very long time. The reason I am interested in improving email is because I am fascinated by communication. Recently, the way we communicate has begun to change rapidly. With the introduction and growth of instant messaging, email is seeming to become a different. Many professional teams are communicating using instant messaging tools such as Slack or HipChat. Both of the previous two companies I interned at used an instant messaging/group messaging service to communicate. This leaves us with the questions: what is the current state of email? What is the future of email? This is what I wanted to explore while thinking about natural language processing and its applications. In total, there were three problems I looked into solving.

### 1.1 Sentiment Analysis

What would be possible if we knew the sentiment of an email? Or, what would be possible if we knew the sentiment of many emails? I thought this would be useful for both personal and enterprise use.

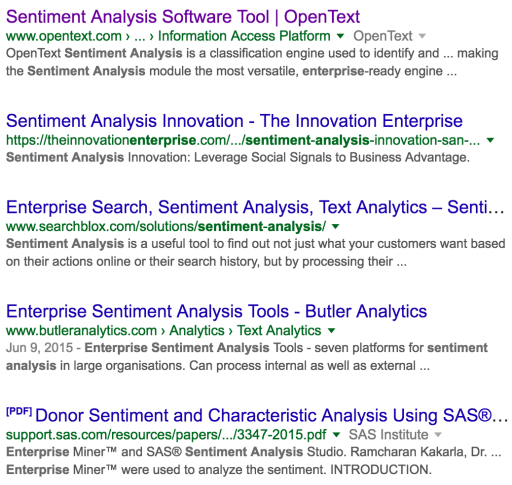
Personally, I would enjoy it if I knew an email was positive or negative. This way I could open positive emails or negative emails depending on my mood. When I check email now, I don't know what to expect. Each email I open is a sentiment I don't have control over. By knowing the sentiment of an email, I

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

control the mood I am in when reading emails.

On an enterprise level, sentiment analysis can be very helpful. For instance, if I owned a service and provided an email for support, I would want to respond to emails with negative sentiment before responding to emails with positive sentiment. This way, I could do damage control with my service. By replying sooner to negative emails, I would have the opportunity to change the mind of the user due to quick support. I noticed when I searched for this on Google, that solutions like this somewhat exist.




The problem with the solutions that came up on Google was that these solutions didn't seem very accessible to the average person. My goal with my project was to create a simple web app that anyone could use. My implementation also is very lightweight. Although my implementation is lightweight, that means it is not too complicated, the result is that it loses out a bit in accuracy. I will talk more about this later in the paper.

## 1.2 Question Analysis

Something else I thought about was figuring out if an email needed a reply or not. Even though I was not able to implement this, it was interesting to think about how I would. One way I could do this would be by using a prior probabilities table to figure out which kinds of sentences had questions. I would do this by first having a training set that would say whether a sentence had a question or not. After training my table on this set, I would be able to take common patterns in my prior probabilities table and map them to whether those patterns produce questions or not. Another more easy way to do this would be by checking for more concrete and common

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

phrases like a question mark, or the word snippets like “can you” or “will he”. Each email in the GUI would be labeled with a tag that would say if a reply was needed or not.

### 1.3 Email Generation

The last thing I thought about that would improve email was email generation. I don't like writing emails and I think a very useful tool would be something that wrote emails for me based on topics that I gave. For example, if I was a recruiter at a company and had the need to email potential candidates about open positions, it would be amazing if I could automate that process. I’m sure recruiters already do this by using generic emails, but this way would be able to use a unique email every time. The way this would work is by using training sets for different categories for emails. For instance there would be a training set for emails about making a job offer, or there would be a training set about emails that give information about the company. The training sets would be made into likelihood tables and prior probability tables and we would use both to generate words. This idea seems like the least feasible of the 3 that I have mentioned because this would require a lot of training data to create sensible emails.

## 2. Method of Sentiment Analysis

My idea behind how I wanted to do sentiment analysis was simple. I wanted to make something that would eventually be lightweight enough such that it wouldn't have to be run on a server, but rather on a users machine itself. Because the program would be run locally on someones machine, the program would have to perform fast and not very resource intensive. This is why I went with a very naïve way of doing sentiment analysis.

### 2.1 My Process of Doing Sentiment Analysis

My sentiment analysis is entirely dependent on the AFINN database. AFINN is a list of words tagged positive or negative with numbers between -5 and 5. [1] I take this list and import the JSON object and use it as a hashed dictionary of words. Since the words are hashed, the access times are O(1).

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

With the AFINN defined as a global hashed/ object, I'm able to loop through every word in the email and see if the word exists in the AFINN database. This runs in O(n) time, as I only have to run through the email once to calculate the total sentiment value. On a more structural level, my app works by using Node and Express to create a simple web server. The web server connects to the Gmail API and authenticates with a token that I pre-register. After the server is authenticated with Gmail, all of my emails are retrieved and I use the AFINN database to analyze the emails. Lastly, this information propagates to the front-end file. Positive emails are marked green, negative emails are marked red and neutral emails are marked gray.

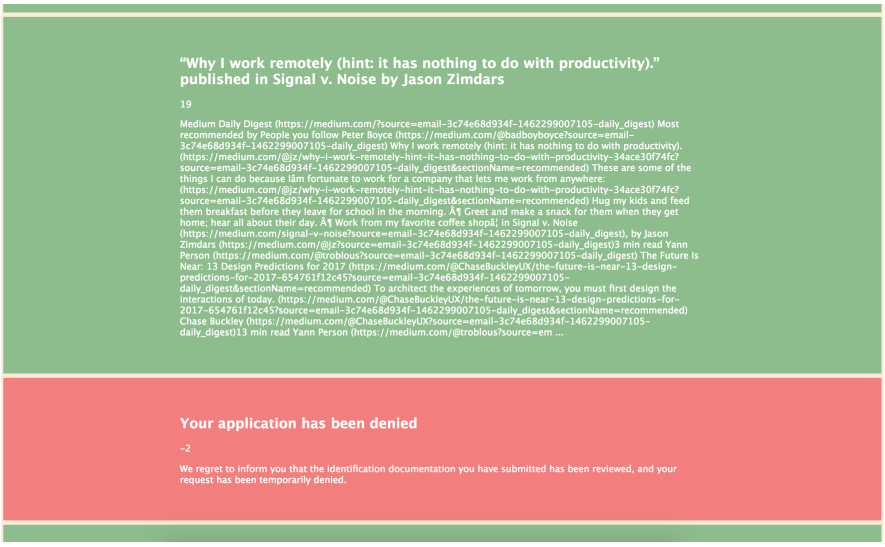
## 3. Results

The results from my first iteration using only the AFINN database were alright. After scrolling through my emails and reading them, I manually labeled the sentiment of the email, and then compared it from the sentiment produced out by my algorithm. I realized I could make many some improvements.

### 3.1 Stemming

My first step was checking which words matched in the AFINN database and which words didn't. I noticed that there were many out of vocabulary words. For instance, the word "accomplishing" was in an email but the word didn't match in the AFINN database. Although "accomplishing" was not in the data base, the word "accomplish" was. To my surprise, I checked for more stems and matched many more words.

In order to actually compute the stems, I used a library called "Snowball". Snowball is a



# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

library that came from Martin Porter. Martin Porter created stemming libraries in the 1980s and then later open sourced them. The way the stemmer algorithms can both be complex and simple. One of the more simple methods is by using the "Production Technique". This technique simply prepends suffixes such as "ing" or "ed". The word "head" would become "heading" or "headed". Another technique is called "Suffix stripping" This works by removing common suffixes from the end of words in order to find the stem. I learned about these techniques both from Wikipedia and and Martin Porter's paper about stemming. [2].

### 3.2 Results of Stemming

My use of stemming, the results were very promising. Here are some of the results

**Stemmed words found in AFINN**

| | | | |
|---|---|---|---|
| word is commitments | word is protection | word is extended | stem is fit |
| stem is commit | stem is protect | stem is extend | word is helped |
| word is striking | word is extending | word is sharing | stem is help |
| stem is strike | stem is extend | stem is share | word is wants |
| word is dangerous | word is critical | word is complaining | stem is want |
| stem is danger | stem is critic | stem is complain | word is joined |
| word is wrecked | word is adopters | word is grande | stem is join |
| stem is wreck | stem is adopt | stem is grand | word is wanted |
| word is fits | word is adopters | word is joining | stem is want |
| stem is fit | stem is adopt | stem is join | word is highlighted |
| word is gracing | word is wanted | word is accomplishing | stem is highlight |
| stem is grace | stem is want | stem is accomplish | word is diamond |
| word is accomplishments | word is helped | word is paying | stem is diamond |
| stem is accomplish | stem is help | stem is pay | word is diamonds |
| word is successfully | word is adoption | word is joined | stem is diamond |
| stem is success | stem is adopt | stem is join | word is wanted |
| word is enjoyed | word is 'threats | word is wanted | stem is want |
| stem is enjoy | stem is threat | stem is want | word is critical |
| | word is expanding | word is fits | stem is critic |
| | stem is expand | | |

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

Below is the simple code from the Node SnowBall API.

Stemming

```
stemmer.setCurrent(word);
stemmer.stem();
var stemmedWord = stemmer.getCurrent();
if(afinn.hasOwnProperty(stemmedWord)){
  count += afinn[stemmedWord];
}
```

### 3.3 Greetings and Signatures

Another improvement I made was that I took out the positivity values if they came from the beginning or end of the email. This is because emails usually have a positive greeting and a positive ending. For instance, I always begin my emails by writing "Good morning" or "Good afternoon" and I always end with "Best regards". To counter this, if the email started with positive words or ended with negative words, I didn't add them to the sentiment count. The sentiment count was the count I used to check if an email was positive. I will talk more about this count in a bit. Here is the bit of code that checks for positive greetings and signatures. I did this by keeping an index which corresponded to the place I was in the email. There is a corresponding photo at the bottom of the page. After implementing this feature, I noticed that short emails without much sentiment at all would be correctly labeled as neutral, instead of having the greeting and signature affect the emails score.

Handling Greetings and Signatures

```
var emailBodyArray = emailBodyText.toLowerCase().split(" ");
var count = 0;
var lengthOfArray = emailBodyArray.length
var index = 0
emailBodyArray.forEach(function(word){
  if(afinn.hasOwnProperty(word)){
    if(index > 3 && index < (lengthOfArray - 3)){
      count += afinn[word];
    }
  }else{
    stemmer.setCurrent(word);
    stemmer.stem();
    var stemmedWord = stemmer.getCurrent();
    if(afinn.hasOwnProperty(stemmedWord)){
      if(index > 3 && index < (lengthOfArray -3)){
        count += afinn[stemmedWord];
      }
    }
  }
  index += 1;
});
```

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

### 3.4 Changing Threshold for What is Positive or Negative

Another problem I ran into was that if emails were very short, small changes in sentiment count would change the final outcome of the sentiment. The sentiment count is what I used to calculate sentiment value. The way I get the total sentiment count is by adding up the matches I get in the AFINN database. Previously, I would label an email as neutral if the count added up to 0. I realized this was a very difficult outcome to achieve. Therefore I changed the thresholds such that an email would be neutral if the sentiment count was within a certain threshold. This threshold would be based on the length of the email. If an email was greater than a certain length, the threshold would increase. You can view this code on the right.

Threshold code

```
var threshHoldLow = false;

if(emailBodyArray.length < 30){
  threshHoldLow = true;
}

if(threshHoldLow){
  if(count > 1){
    var sentiment = "positive";
  }else if(count < -1){
    var sentiment = "negative";
  }else{
    var sentiment = "neutral";
  }
}else{
  if(count > 6){
    var sentiment = "positive";
  }else if(count < -6){
    var sentiment = "negative";
  }else{
    var sentiment = "neutral";
  }
}
```

# Improving Email with Natural Language Processing

**Jhishan Khan**
May 8th 2016
New York University
jhishan@nyu.edu

## 4 Retrospective

I knew from the beginning that my method of sentiment analysis would not be too accurate. I mentioned earlier why I did it anyway, it was because the method I choose was very fast. The pitfall here was that my program didn't take into account of context when calculating sentiment. There are tricky areas where even when positive words are used, it is in a very negative context. For situations like these, my program fails. The way that a traditional sentiment analysis would be implemented is by first manually parsing testable text, creating a feature vector with a stop word list for parsing, a classifier text, and then a training corpus that would use the aforementioned elements in conjunction with a machine learning algorithm to generate sentiment. These algorithms range from Naive Bayes to Support Vector Machines, and are subdivided by “supervised” and “unsupervised” classification algorithms. Once the training is completed, I can use real time emails as a prediction corpus and generate the general sentiment index. [3]

**Resources used**

[1] http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

[2] wikipedia
https://en.wikipedia.org/wiki/Stemming
http://snowball.tartarus.org/texts/introduction.html
Martin Porter
http://snowball.tartarus.org/texts/introduction.html

[3] Context conscious sentiment analysis
https://homes.cs.washington.edu/~jfogarty/publications/uist2010.pdf

http://www.cs.cmu.edu/~mwen/papers/edm2014-camera-ready.pdf