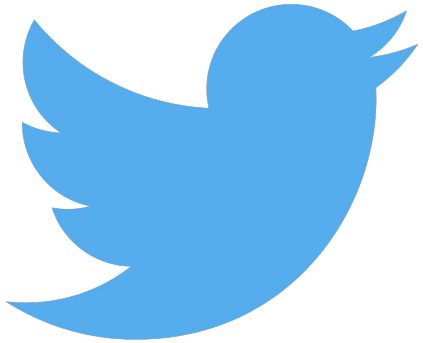


CS106S: Week 3

Sentiment Analysis on Refugee-Related Tweets

Download the starter code:
bit.ly/cs106s-refugees

Follow the slides:
bit.ly/cs106s-refugees-slides

Announcements

- CS106S Email List has been created!
  - If you did not receive an email from us last week, let us know so we can add you :-)
- CS106S Outside-Of-Class Help
  - Questions about anything we do in class?
    Feel free to contact us after!

Today: Sentiment Analysis on Refugee-Related Tweets

1) What is Sentiment Analysis?

2) The Problem:
   Refugee Sentiment on Twitter

3) Our Solution:
   Building a Refugee Sentiment Tweet Classifier

Sentiment Analysis: What is it?

Dictionary

sentiment analysis

sen·ti·ment a·nal·y·sis

noun

the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.
"companies have key lessons to learn about harnessing the power of social media and sentiment analysis"

Translations, word origin, and more definitions

Idea behind Sentiment Analysis

- ➔ Look at a piece of text
- ➔ Based on features in the text, classify the piece as positive or negative
- ➔ Common Features include:
  - ◆ How indicative certain **words** are with positivity/negativity?
  - ◆ How indicative certain **phrases** are with positivity/negativity?

- ➔ Powerful Uses

Positive or negative movie review?

- unbelievably disappointing

- Full of zany characters and richly applied satire, and some great plot twists

- this is the greatest screwball comedy ever filmed
- It was pathetic. The worst part about it was the boxing scenes.

Google Product Search

HP Officejet 6500A Plus e-All-in-One Color Ink-jet - Fax / copier / printer / scanner
$89 online, **$100** nearby ★★★★☆ 377 reviews
September 2010 - Printer - HP - Inkjet - Office - Copier - Color - Scanner - Fax - 250 sh

Reviews

Summary - Based on 377 reviews

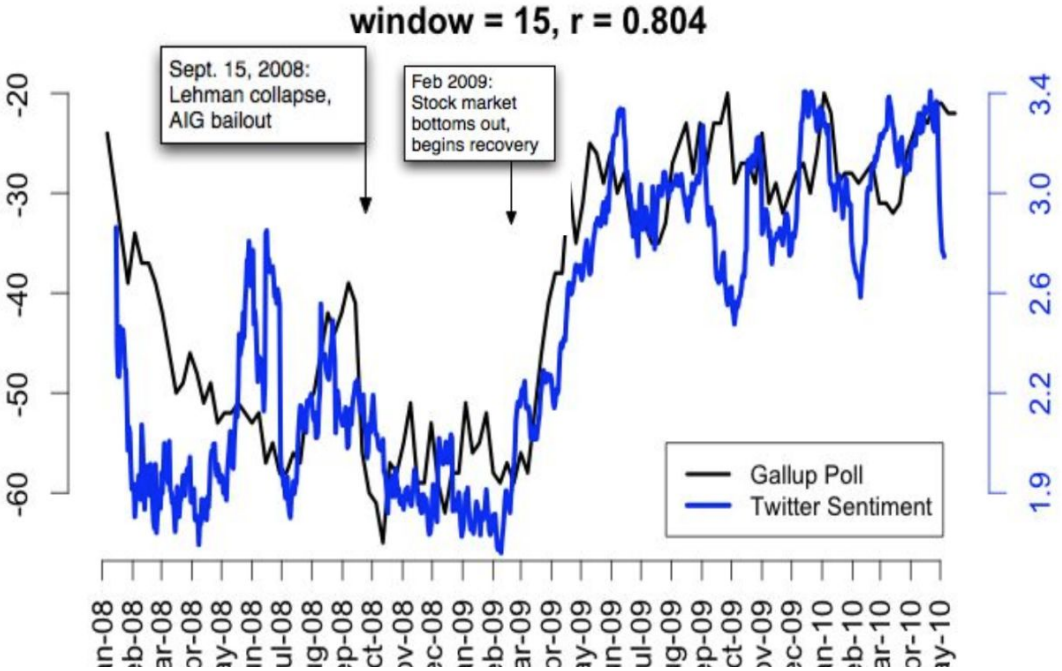
1 star	2	3	4 stars	5 stars

What people are saying

ease of use	"This was very easy to setup to four computers."
value	"Appreciate good quality at a fair price."
setup	"Overall pretty easy setup."
customer service	"I DO like honest tech support people."
size	"Pretty Paper weight."
mode	"Photos were fair on the high quality mode."

Twitter sentiment versus Gallup Poll of Consumer Confidence

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In ICWSM-2010

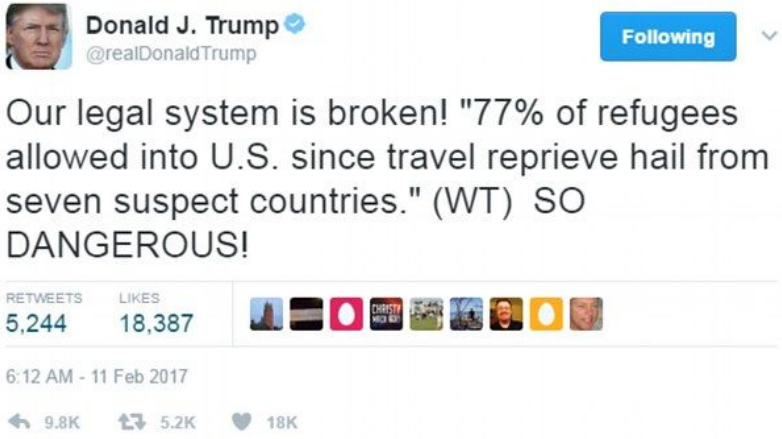


Uses of Sentiment Analysis

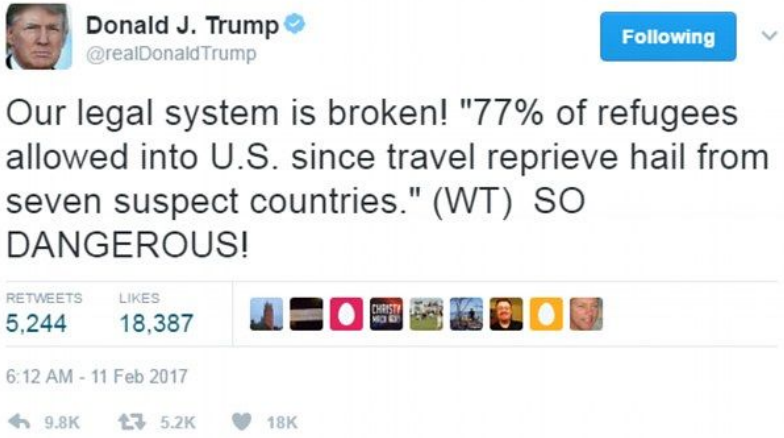
- ➔ Movie Reviews: What does the public think of this new movie?
- ➔ Products: What do people think of the new iPhone?
- ➔ Public Sentiment: Reaction to news? Current events?
- ➔ Politics: What do people think of X issue?
- ➔ Prediction: What will the outcome of an election be based on social media data?

Uses of Sentiment Analysis

- ➔ Movie Reviews: What does the public think of this new movie?
- ➔ Products: What do people think of the new iPhone?
- ➔ Public Sentiment: Reaction to news? Current events?
- ➔ **Politics: What do people think of X issue? (today)**
- ➔ Prediction: What will the outcome of an election be based on social media data?



The Problem: Understanding Refugee Sentiment on Twitter



Given a bunch of tweets, can we classify anti-refugee sentiment from pro-refugee sentiment?

Why does this matter?

Classifying Sentiment allows us to build:

- ➔ Bots that can automatically detect hateful tweets and respond to them with educational resources
  - ◆ Swaying public opinion
- ➔ Metrics on demographics of anti-refugee actors to inform resistance and education efforts
- ➔ Understanding general opinion in America
  - ◆ Spring ‘17 CS50 Project with the United Nations!

Our Solution:
Building a Refugee Sentiment Tweet Classifier

Download the files: bit.ly/cs106s-refugees

Again, slides can be found here:
bit.ly/cs106s-refugees-slides

```
var wordMap = {
    “illegal”: -3,
    “welcome”: 2,
    “fear”: -1,
    “innocent”: 4,
    ...
}
```

Key Approach

For each word in a tweet, assign a value to that word corresponding to how indicative of pro/anti-refugee sentiment it is.

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. For each word in the tweet:
    a. Stem the word*
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. For each word in the tweet:
    a. Stem the word*
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Example: "We love refugees #refugees" – Positive Tweet

Creating the wordMap

1. **Iterate through each training tweet in trainTweets**
2. Get the tweet's classification
3. For each word in the tweet:
    a. Stem the word*
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Example: "We love refugees #refugees" – Positive Tweet

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. **Get the tweet's classification from label**
3. For each word in the tweet:
    a. Stem the word*
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Example: "We love refugees #refugees" – **Positive** Tweet

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. **For each word in the tweet:**
    a. Stem the word*
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Example: "**We love refugees #refugees**" – Positive Tweet

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. **For each word in the tweet:**
   a. **Stem the word*** // var stemmedWord = stemmer(word);
   b. If tweet is anti-refugee, –1 from the word's score in wordMap
   c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

Example: "**We love refugees #refugees**" – Positive Tweet

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. **For each word in the tweet:**
   a. Stem the word* // var stemmedWord = stemmer(word);
   b. If tweet is anti-refugee, –1 from the word's score in wordMap
   c. Else, +1 to the word's score in wordMap
4. Once we go through all words in a tweet, move to the next tweet.

+1 +1 +1 +1

Example: "**We love refugees #refugees**" – Positive Tweet

```
var wordMap = {
    “we”: 1,
    “love”: 1,
    “refugees”: 1,
    “#refugees”: 1,
    …
}
```

WordMap after one tweet

Example: “**We love refugees #refugees**” – Positive Tweet

Creating the wordMap

1. Iterate through each training tweet in trainTweets
2. Get the tweet's classification
3. For each word in the tweet:
    a. Stem the word* // var stemmedWord = stemmer(word);
    b. If tweet is anti-refugee, –1 from the word's score in wordMap
    c. Else, +1 to the word's score in wordMap
4. **Once we go through all words in a tweet, move to the next tweet.**

```
For (var i = 0; i < trainTweet.length; i++) {
    var tweet = trainTweet[i];
    var tweetText = tweet.tweet;
    var tweetTextLower = tweet.tweet.toLowerCase();
    var tweetLabel = getScore(tweet.classification)
}
```

Tip #1: Tweet Structure

For each trainingTweet, you have access to its **text** and **classification**.

```
For (var i = 0; i < trainTweet.length; i++) {
    var tweet = trainTweet[i];
    var tweetText = tweet.tweet;
    var tweetTextLower = tweet.tweet.toLowerCase();
    var tweetLabel = getScore(tweet.classification)
}
```

Tip #2: getScore() is helpful!

Returns 1 if pro-refugee, -1 if anti-refugee.

```
For (var i = 0; i < trainTweet.length; i++) {
    var tweet = trainTweet[i];
    var tweetText = tweet.tweet;
    var tweetTextLower = tweet.tweet.toLowerCase();
    var tweetLabel = getScore(tweet.classification)
}
```

Tip #3: Make the tweet lower case!

Who knows why?

Classifying Test Tweets

1. Iterate through each test tweet in testTweets
2. Initialize a variable to store our “score” of the tweet.
3. For each word in the tweet:
    a. Stem the word*
    b. If the word is in our wordMap, add that word’s score to our tweet score variable
4. Once we go through all words in a tweet, classify!
    a. If our score is > 0, classify as pro-refugee (false)
    b. If our score is < 0, classify as anti-refugee (true)

Classifying Test Tweets

-5 +4 +3 +2

“**Don’t hate. Love all refugees #refugees**” – Positive

Classifying Test Tweets

-5 +4 +3 +2

“**Don’t hate. Love all refugees #refugees**” – Positive

Set myGuesses[testTweet.tweetID] = false

// if this were a anti-refugee tweet, we set to true

And that’s all you need to know!

Let’s get started

Improving Our Performance: TF-IDF

TF-IDF computes a weight which represents the importance of a term inside a document. It does this by comparing the frequency of usage inside an individual document as opposed to the entire data set (a collection of documents).

The importance increases proportionally to the number of times a word appears in the individual document itself--this is called Term Frequency. However, if multiple documents contain the same word many times then you run into a problem. That's why TF-IDF also offsets this value by the frequency of the term in the entire document set, a value called Inverse Document Frequency.

```
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).
Value = TF * IDF
```

Let's say you have a 100 word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

```
TF = 5/100 = 0.05
```

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

```
IDF = log(10,000/100) = 2
```

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

```
TF-IDF = 0.05 * 2 = 0.1
```

Try that!

You'll get way better results now!

Final Discussion

Let’s analyze a bit what we just did

Check-off Link!

bit.ly/cs106s-refugees-checkoff