Using Twitter to understand the emotional state of people living in Ireland.

Written by James Eggers.

School: St. Michael's College
Stand Number: 3202
Age: 16

Introduction *(1)*

Twitter is fast becoming an incredibly powerful tool for everybody from journalists looking for the public opinion on something to marketing executives looking for new ways to market products and ideas to the public.

Every day 60+ million tweets are posted to the site from all around the world. Twitter is often referred to as a *'micro-blogging'* system. *'Tweets'*, as they are referred to, are very short, and are limited to 140 characters. Therefore, most of these tweets will be information rich and collectively unbiased. Indeed, some tweets will undoubtably be biased in some direction or another, but others will no doubt be biased in the opposite direction on the same topic. Therefore, creating a fair and level playing field.

Another advantage to using Twitter to understand public mood is that you can easily collect tweets from a whole country, or the whole planet, not just a small, specific area as with a handout survey.

My project is a method of understanding how the mood Ireland (or theoretically any country) changes / fluctuates throughout a predefined period, for example days, months or even years. I achieved very interesting results (which are described later) in doing this project.

Some Problems *(1.2)*

As with everything, there are some problems associated with doing this type of experiment.

Every programmer has different ideas about how to accomplish a task. Algorithms and their logic are always going to be different from programmer to programmer.

Building an algorithm to tag tweets as to being Positive, Negative and Neutral is tricky. Often, algorithms will be too lenient on scoring a tweet and give the tweet a positive score when it should be negative or simply just give the tweet a neutral score when it should be the latter.

Natural Language Processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human languages. [1]

Semantics is the study of meaning of signifiers such as words, phrases, signs and symbols, and what they stand for. It typically focuses on the relation between *signifiers*, such as words, phrases, signs and symbols, and what they stand for. [2]

Using NLP and Semantics, I created an algorithm that can (most of the time) discern between *'happy'*, *'negative*' and *'neutral'* tweets.

Getting the Tweets is also another problem in itself. The Twitter search API has a usage limit of about 1500 queries a day. That's a little over a query a minute. Which was fine for me because there are so little Tweets originating from Ireland per minute. However, If I was doing this experiment again, I would use the streaming API instead, as you can collect more tweets per minute than you could by using the search API.

[1] = Wikipedia Reference, en.wikipedia.org/wiki/Natural_language_processing.
[2] = Wikipedia Reference, en.wikipedia.org/wiki/Semantics.

When Twitter responds with the Tweets my data miner requested, the location that the tweet originated from is included with the tweet but only if the author of the tweet has previously consented for this to be possible.

The API (Application Programming Interface) gives a JSON response with the tweet's origins, which are either the exact latitude and longitude or the '*Hometown'* listed on the Tweet's author's Twitter Profile. Either is fine. The tricky part here though, is sorting the tweets into their respective constituents, say putting all the tweets that come from Dublin together with all the other Dublin Tweets. Many people don't specifically say '*Dublin, Ireland'*, but instead they might say *'Dún Laoghaire, Ireland'.* Or, a a person may add punctuation, *'Dún Laoghaire-Ireland!!!'*.

Building an algorithm to parse this isn't too hard, it's something I needed to think about though. How will the algorithm know that Dún laoghaire and other places in various counties is in their respective counties? The Google Maps API was the first thing that came to my mind. Unfortunately the Geocoding (and Reverse Geocoding) APIs have a query limit of 2500 per day. Not very helpful when trying to parse millions of Tweets. Even the Premier Edition is capped with a 100000 per day query limit. I'll look into this in more detail later, but I used the Bing Maps API, which doesn't seem to have a usage cap.

There are yet more problems I encountered. In my opinion, MongoDB is the most suitable database for use in this experiment. You could also argue that Cassandra could be used, and I'm sure it would work fine, but it's very complex and I've no experience with it. I went ahead with MongoDB, choosing it for it's simplicity, ease of use and because it was recommended to me by friends.

I used Python as my main language, again for it's simplicity and ease of use. I did however build the data miner with PHP, which works well.

The Data Miner *(1.3)*

The data miner is relatively easy to create. I am using the Search API for this experiment, because it suits this experiment better.

I went with the Search API. I read over the (very good) twitter API docs and started building a program that would gather tweets.

My data miner is written in PHP and is dead simple. It executes every minute, gathering new tweets from Twitter, at a rate of about a 100 new tweets per minute.

A simplified version of the code I used for my data miner is below.

```
function getSearchData() {
    $url = "http://search.twitter.com/search.json"
    $ch = curl_init()
    curl_setopt($ch, CURLOPT_URL. $url);

    $result = curl_exec($ch);
    curl_close($ch)

    return $result;

}
```

It initiates a simple curl request to get the JSON response from twitter.

As you can see it's fairly straightforward, just fetch the JSON data with a cURL request. Then just parse the JSON response and dump everything twitter gives you into a collection in MongoDB

I haven't included the full URL or the sake of simplicity, but the extra arguments (extra bits of information) are the latitude and longitude of the area I want to collect Tweets from (Ireland in my case), and the radius around the geographic point I supplied from which I want to collect Tweets. That, in essence, is the data miner.

Geocoding the Tweets *(1.4)*

Now that we have the tweets, it's necessary to split them up into different constituencies. I need to group all tweets from from each county together. If I was being more specific, I could group all tweets from each individual town together, for example, group all tweets from Dún Laoghaire together. However, that's more difficult.

I built four algorithms to do this. The first, scans over all the tweets looking for tweets from a particular place, I have a tuple (list of values) containing 32 counties. I simply loop through this tuple, searching the tweets for any tweets from the county in question, I then store these tweets, with county information. This technique was able to sort roughly half of all the tweets I had.

The second algorithm, did almost the exact same, except it didn't search for counties. I compiled a list of the top 30 highest populated cities in Ireland, and searched for them, and then stored each tweet with it's new county information.

The third algorithm is more complex. It loops through all the tweets, and counts each time a tweet is sent from a patricular location. So, we then have a list of places where at least two of our tweets originated from. We can then loop through the shortlist of place names and query the Bing Maps API with each place name, and get the county from the results returned from the API, and then store the tweet with the county information.

I then queried the Bing Maps API with the origins of the rest of the tweets an tagged them with their respective counties.

Some Twitter users list their hometown as something nonsensical like 'a castle' or 'an island'. These tweets are much less useful.

Processing the Tweets *(1.5)*

For my experiment, I used the Subjectivity Lexicon (from the University of Pittsburg, http://www.cs.pitt.edu/mpqa/). The paper provides normative emotional ratings for a large number of words. This Paper was extremely helpful for me in this experiment.

I'm going to explain a few key-terms I'll be using before I continue:

- A **Token**. Before any processing can be done on the Tweet, it needs to be segmented into linguistic units such as words, numbers, punctuation or alphanumerics. Each of these units is known as a 'Token'.
- A **Sentence** is an ordered string of tokens.
- A **Corpus** is a body of text, that is usually quite large.
- A **POS Tag** (Part-of-speech tag) is tag that classifies a word as a noun (NN), verb (VB), adjective (JJ) definite article (the). One of the oldest and most commonly used tag sets is the *Brown Corpus*.

That's it for now. A major flaw in a lot of these types of algorithms is that the context that a word is used in is not taken into account. For example, a bad algorithm would say that this is a positive tweet: "*I am not happy".* For the simple reason that the algorithm sees the word 'happy' and automatically assumes it means that the whole sentence is positive/happy.

This algorithm completely ignored the semantic meaning of the word "not" and how it completely redefines the meaning of the sentence. A good algorithm would of course recognise that this is a negative tweet, not positive.

Natural Language Processing is very important for this to work to it's full potential. There's no point just being lazy and not bothering to address this problem.

All the following code examples will be in Python. Let's imagine this is our tweet: "I love The Simpsons". Let's Tokenize it:

In this simplified version of the code, each token will now be correctly formatted without any punctuation, making it far easier to analyze the string. (I also stored the original string with the punctuation so I could analyse the placing of the punctuation). Also, I've only included three punctuation markers in the example below, because putting them all in will take up too much space, the rest need to be added in however to make this work properly.

```
string =  """I love the Simpsons!"""
punctuation = (('"', ' '), ("'", ' '), ('(', ' '))

def removePunctuation(str, punctuation):
        for find, replace in punctuation:
                str = str.replace(find, replace)
        return str

tokens = tuple(removePunctuation(string,
punctuation).split(' '))
```

Now that the string has been Tokenized, we can move on.

Once you tokenize the string, I started analysing the words properly.

I used the Subjectivity Lexicon in my algorithms to find out what the polarity (mood) of each word is. Other than the words in a tweet however, there are other markers and traits which will give you're algorithms clues to the mood of the tweet. Symbol Analysis, is very important. Symbols like :) or :(are often found in tweets, and instantly describe the mood of a tweet.

However, some tweets will not contain any symbols and I needed to build an algorithm which will figure out the overall sentiment of the Tweet correctly. My algorithms had about a 75% accuracy rate, which is quite good.

A huge issue in this area is Sarcasm. It is extremely hard for a computer to understand and detect when a Tweet is sarcastic.

Another issue is that one word could have many meanings, for example, the word "plane", could be referring to an airplane, it could also be referring to a geometric plane, or the verb, to plane, as in to glide through the air.

There are some identifiers that can be used. For example, if a word is preceded by 'a' it is most likely a noun. If a word ends in 'ing', it is most likely a verb. The list is endless, and once you start thinking about it, it's quite easy, you can add loads of grammatical rules. Also, if a word is preceded by 'not' then the word in front of it will most likely be the opposite of what the Subjectivity Lexicon says it is. For Example, a sentence like "I am not sad" would be happy. My algorithm takes words and sentences apart, and takes all the words (that it recognises) in a sentence into account, and analyses them to try to figure out the meaning of the whole sentence, which it does very successfully.

A POS (Part-Of-Speech) tagger is very useful. The Subjectivity Lexicon has a POS tag for each word in the Lexicon, which was very handy, as it enabled my algorithm to instantly check what type of word a word is, so it can better understand the meaning, which, again worked very well.

Using a culmination of the above, I built a very accurate and efficient algorithm to score each Tweet relative to its sentiment.

Analysing the Data (1.6)

After the tweets have been tagged as being positive or negative, I started to analyse the data.

Once I found the overall sentiment of each county, by writing a program which mathematically figured out the sentiment, I was able to easily start going into more depth with it.

The first thing I did was create a graph, detailing the average mood over a week, and I found that there was a definite dip in the mood during the middle of the week, with the trough occurring on Thursday morning. This graph is a proof of concept, it shows that the system works, because we can see that, as expected, the mood rises on the weekend and falls as the week progresses. Then, I created a series of other graphs, such as a graph detailing the daily fluctuations in the collective mood. I found people were happiest at about 18:00 and least happy at 04:00. I then created a graph which compares the average mood of the East coast to the average mood of the West coast of Ireland, and found the on average, the west coast of Ireland is nearly always happier than the East coast of Ireland.

However, I still needed more proof to backup my project. I then analysed tweets from the day that the Budget was read in Ireland, and plotted the results on a graph, against the average mood in Ireland over a day, and I found that about half and hour after the budget had started to be read, the mood started to decline rapidly, the general mood was also well below average that day. This shows that people weren't happy with the Budget, this acts as further proof of concept, in conjunction with the weekly mood graph.

Afterwards, I started to think about other ways I could display the data in a meaningful way. I decided to use a Map of Ireland with changing colors depending on the mood to display my data. I used an SVG base map that I got from the Wikimedia Commons. I modified the code which built the map so as my algorithms could change the colours of the counties automatically. This worked out very well and I created a minute-by-minute time-lapse video of the fluctuations of the mood in Ireland on a county by county basis. It's available to watch on http://thevibesofireland.com/ or on YouTube.

Real Time (1.7)

I decided to take the project a little bit futher than I originally planned. I decided to create a real-time, auto-updating mood-map of Ireland.

This required a lot of work. Instead of me issuing commands to the computer for it to create the different graphs and maps, I had to create a complex program that would automatically take care of itself and create graphs, maps and deal with errors all by itself.

I succeeded in doing this, and it works very well. Every minute, the computer gathers new tweets from Twitter and puts the data into a file on the server. The computer then executes another sub-routine in the program which takes the data and transforms it into a map, whilst simultaneously creating a graph detailing the mood for the last twenty four hours for each county in Ireland, that's a huge amount of work but only takes about ten seconds.

The end result is a real-time, auto-updating mood-map of Ireland, with embedded graphs and demographics of each county - a perfect marketing tool.

Conclusion & Results (1.8)

A Breakdown of my results:

Weekly Analyses
From the weekly analyses, I found that people are happiest on Friday evening, and least happy on Thursday morning. This is consistent with my expectations and supports the theory that in general people are least happy during the week and happiest on the weekends. I found that there was a definite dip in the mood during the middle of the week (when compared to the weekend), which is prominent on Figure 1 on the next page.

Daily Analyses
From the Daily Analyses, I found that people are generally happiest at about 18:00, whilst the trough (least happy mood) occurs in the early hours of the morning, at about 04:00. This again supports the theory that people are unhappy waking up early in the morning, but are happiest during the evening when the day is over. This is shown in Figure 2 on the next page.

East Coast vs. West Coast
When I created a graph which compares the collective mood of the East Coast of Ireland to the West Coast of Ireland, I found that the West Coast was nearly always happier than the East Coast by quite a large amount. This is shown on Figure 3 on the next page.

The Budget
Upon analysing data from the day of the Budget, I found that at almost the exact time the Budget was being read, the mood plummeted down, and stayed consistently down for about four or five hours, showing that the drop in mood was not an anomaly. This graph can be see in Figure 4.

Testing (1.8)

To Test the algorithms, I wrote a program to randomly choose 500 Tweets from the database. I also wrote a program that would allow a human to easily mark these Tweets as being positive, negative or neutral. I got people to mark these tweets as to their sentiment, and found that my algorithms have a 70% - 80% accuracy rate.

Figure 1:

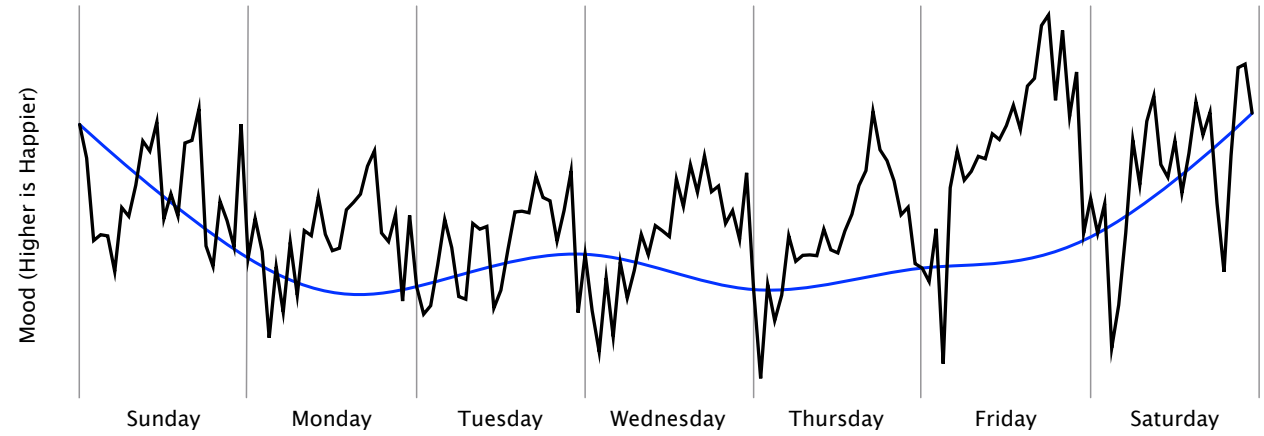

Figure 2:

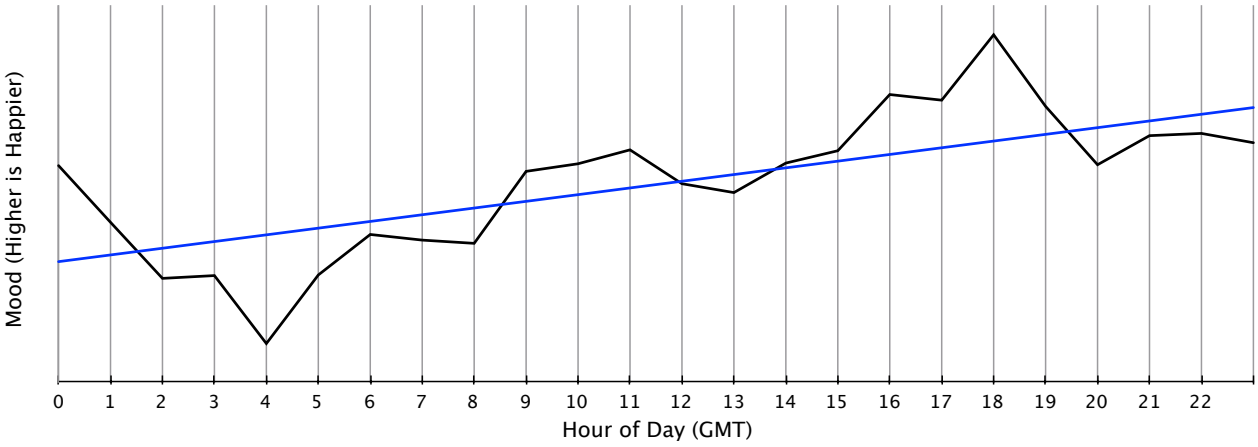

Figure 3:

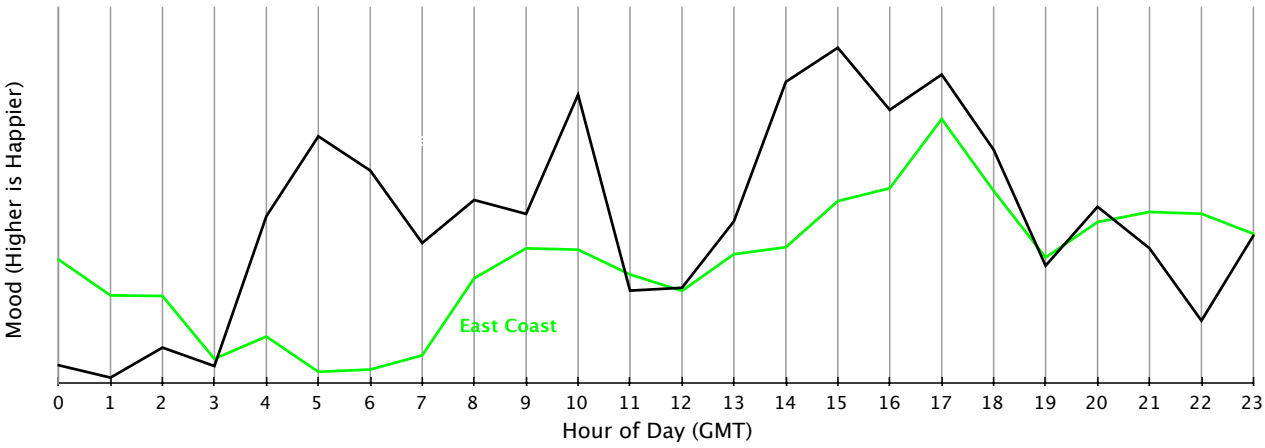

Figure 4: