Twinfo

www.smileytweets.com

Curtis Yamanaka, Golf Sinteppadon, Jesse Shepherd, Jon McCord, Michael Amorozo, Sam Clark

Final Report
12/15/10

Project Goals:

The goal of Twinfo was to identify items the Twitter community finds newsworthy. Throughout the quarter the direction of our project changed drastically. We experimented with event identification, sentiment analysis, and eventually landed on trends. While the experiments varied considerably, one motivation connected all of them. We were fascinated by the sheer amount of information users broadcast about their opinions, future plans, and lives in general - and knew that there had to exist some way to synthesize this data into a representation of the pulse of Twitter. We were also motivated by a useful tool at our disposal - Named Entity Recognition (NER) on Twitter. Sam Clark and Alan Ritter PhD had been working on adapting NLP tools to Twitter, so the team was able to use a rough NER algorithm to find Named Entities (NEs). NEs proved to be a great feature level to track Twitter on . They allowed for large feature reduction, and activity surrounding an NE generally correlated with some real-world event.

We found newsworthy events on Twitter through the use of two key procedures. We tracked occurrences of NEs over 24-27 hour periods to see if they were showing up disproportionately, and we kept track of NEs that were co-occurring in Tweets together. We used these features to create events out of NEs that were simultaneously trending and co-occurring. This accomplished a number of important objectives. First, it gave us the ability to add context about the event. For example, an event just consisting of ‘Ryan Reynolds’ could have to do with his career or any other aspect of his life. However, an event ‘Ryan Reynolds’ and ‘Scarlett Johansson’ shows that something occurred between these two people. In this case the two mentioned NEs divorced. It also makes the event seem more intriguing, since someone is more likely to be curious about what occurred. Second, it prohibits multiple events that correspond to the same real-world event from appearing on our frontend. Lastly, it makes it easy to get Google search results about the event, since it limits the number of relevant pages to those involving all NEs. In fact, we are confident enough that this will lead to a story explaining the event (if one has been published fast enough) that we display Google results on our frontend.

We display information about these events to the user in the following way. On the left we have a list of all events ordered by how significant they are (how anomalous a given NE’s mentions are) at a given time period. Below this is a slider that when slid to the right forces events from past dates to be displayed (turns blue when it is loaded and can be slid). For the top event or any event that is clicked three tweets about it are displayed in the right column. The right column also includes a graph of occurrences and Google search results. Additionally, events are divided into roughly five categories (celebrity, entertainment, news, sports,web), and results

from these categories can be filtered out by de-selecting the corresponding check box at the top.

System Design and Algorithm Choices:

Backend

The goal for our backend was to create a robust and scalable system for retrieving tweets, running them through NER, storing information about co-occurring NEs, and detecting trends. This lead us to a very modularized pipeline in which there were a number of producer/consumer processes that used a DB as a buffer for temporary storage. The tasks were split up as follows:

Collecting tweets:
Twitter has a number of APIs through which one can access tweets. We used the Streaming API, which makes it possible to track tweets containing supplied keywords as soon as they are published. All users have access to a 1% sample of all public tweets by default, and the ability to track up to 400 words. While this was enough to get us off the ground, Machine Learning applications always benefit from an abundance of data. We contacted the Twitter API team - and after explaining our project - were granted the ability to get a 10% sample of all tweets, as well as track up to 10,000 words.

Since our initial plan was to track the change of sentiment of a given NE over time, we decided to track sentiment bearing tokens. We used 5000 tokens that University of Pittsburg researchers found to be sentiment bearing (http://www.cs.pitt.edu/mpqa/, under Subjectivity Lexicon).

To perform the tweet collection the pycurl library was used to maintain a persistent HTTP connection. The process managing this was also wrapped in another process, which monitored it and restarted the HTTP connection when it wasn't producing tweets.

We weren't able to process the tweets at the rate we were receiving them (100 – 150 /second), so we throttled it back to 90 tweets per second (after English classification, more below). We then stored these in a temporary database table.

Language classification:
As tweets were collected, they were classified as English or Other. This was done to avoid wasting computation on predominantly non-English tweets that made it through the tweet stream. Categorization was done using a very simple algorithm. A dictionary was generated by scraping tens of MB worth of data from the Wall Street Journal. However, Twitter is a space where language evolves rapidly, so human annotations of roughly 300 tweets were added to the dictionary as well. All words appearing only once were purged - to eliminate misspellings and creative spellings - and the rest written to disk. Tweets were classified as English if they contained a significant percentage of words which could also be found in the dictionary.

Running NER on the tweets:

Two Amazon EC2 machines were used for running NER on tweets, and another was used for tweet collection and metric tracking. The two NER machines obtained tweets by querying the table the tweet collection process stored them in.

NER was accomplished using a library supplied by Alan Ritter, a University of Washington grad student. He uses Mallets sequence tagger with a number of regular expression features, as well as a long list of dictionary features. The dictionary features include lists of common adjective, prepositions, verbs, first names, last names, company names, holidays, locations, etc. This seems to detect people with a high precision. Alan has been able to increase the performance of the NER using POS (Part Of Speech) tags, but at the cost of increased computational intensity. We opted to not use POS tags since we felt we could make up the NER's decrease in precision by processing more data. Financial restrictions made it difficult for us to test this (having already invested in two EC2 machines), but if the quality of the NEs in our final product is any indication, we made the right decision.

After the NEs were extracted from tweets, the tweet and NE combinations were placed into another temporary table in the DB. We ended up finding NEs in 15% of all tweets, resulting in a flow of 13-14 tweets per second being placed into the temporary table.

Metric tracking:
The goal in metric tracking is to find named entities whose # of mentions is unusually high. To accomplish this, we used a formula that gave a result similar to that of a z-score (the number of standard deviations from the mean). The trending score for a NE at time X is found using the following formula:

N1 = # of tweets in which the NE has occurred in between X and (X – 24) hours.
N2 = # of tweets in which the NE has occurred in between (X – 3) hours and (X – 27) hours.

N1' = Min(N1, 10)
N2' = Min(N2, 10)

Trending score = (N1' – N2') / Sqrt(N2')

All NEs that have been mentioned at least 10 times in the last three hours have their trending score tracked.

Events and Co-occurrence:
Events are created by finding co-occurring trending NEs. An event consists of a few components. First, there is the main NE - the NE in the event with the highest trending score. Next, there is the event score, which is the trending score of the main NE. Lastly, events consist of other NEs that have co-occurred with the main NE more than 5 times in the last 3 hours and have a trending score greater than 5. A NE can only be in one event at any given time, effectively collapsing multiple related trends into one. Events are created when a NE that is not currently in an event has a score that would put it in the top 100 event scores. An event exists

until 8 hours have passed since it was created or it is no longer in the top 100 events.

Every hour a snapshot of the top 100 events is stored in order to keep track of past events. Tweets containing NEs for each event are stored as well, along with mention counts for the top 1,000 NEs. This former allows us to provide the user with context, and the latter allows us to graph the trend for a given NE.

Categorization:
We divided NEs into five categories (Celebrity, Entertainment, News, Sports, Web, and a catch-all Other) based on the content of the tweets containing them. Categorization was handled with Naive Bayes. Initially, training data consisted of tweets from self-categorized twitter users, found on WeFollow (http://wefollow.com/). This data proved to be quite unreliable and noisy, producing inconsistent results for all but Sports. We then changed our training data set to relevant documents parsed from a wikipedia dump. This provided slightly more consistent results. Whenever an NE is stored to the database, it is categorized based on a subset of 30 of its tweets.

Frontend

With the backend in place, the goal of the frontend was to display the most current, relevant, and interesting data to the user in an intuitive and informative way. Most similar sites that we investigated had either poor data, or a cumbersome user interface that discouraged use. Twitter is a site that was built upon speedy, concise communication, and our website would need to emulate these virtues as well. It is currently hosted at www.smileytweets.com - a name which no longer reflects the direction the project has taken - and will be moved to a more suitable domain name shortly.

Show current trending:
The most prominent feature of our website is the “Popular events” table, where the top ten trending events are displayed. These events are live (updated as fast as our NLP algorithms will allow), and represent the ever-changing pulse of the twitter community. The primary NE associated with each event is shown most prominently, allowing the user to quickly find out what trends interest her. Other NEs associated with the event are also displayed, along with the trending score and the time that the trend peaked. Each event is color-coded according to its category, which will be described in more detail in its own section. Finally, to the right of the Popular events table, three tweets associated with the top event are displayed with all corresponding NEs in bold. This gives the user insight into the types of tweets driving the trend, and provides context for the event. If an event other than the top is clicked, tweets from that event will displayed instead, allowing for user interactivity.

Scroll back in time:
Although we want the focus of our site to be on live, news-breaking trends, many users will be interested in browsing past trends for interesting content. For these users, we have provided a scroll bar, allowing them to “scroll back in time”. Once the scroll bar indicator turns blue, users can scroll freely through our archived trends, currently spanning over two weeks. This trend

timeline has a granularity of 8 hours, allowing for fine-grained analysis of past trends. Once a time slice has been selected, the site behaves exactly as it does with live data.

Categorization:
Each event is assigned a category based on the content of the tweets of its leading NE. Categories are color-coded both for easy identification and to add interesting, moving parts to the website. By default, all categories are displayed as “popular events”. However, users can filter out categories that don’t interest them by clicking the corresponding check box at the top of the page. The check boxes themselves are labeled, and color-coded along with the trends. When a category is de-selected, the popular events are re-populated using the remaining categories, such that there are always ten items displayed to the user. If too few categories are selected, it may be the case that there are not enough trends remaining to re-populate the list to ten.

Google Results:
An important component of our site is the Google search results, displayed just below the tweets for any selected trend. This displays the first (most relevant) result from google for the collection of all NEs in an event, with NEs in bold. This provides a reliable and significant amount of context for events, and the user can follow the provided link for additional information. For past events, a date range is specified, to ensure consistency (new news won’t overwrite old news for a given event). We levarage Google’s impressive search API to handle the expensive task of finding context for trending events. As with the tweets, the Google search results are displayed for either the top event, or the one the user most recently clicked on.

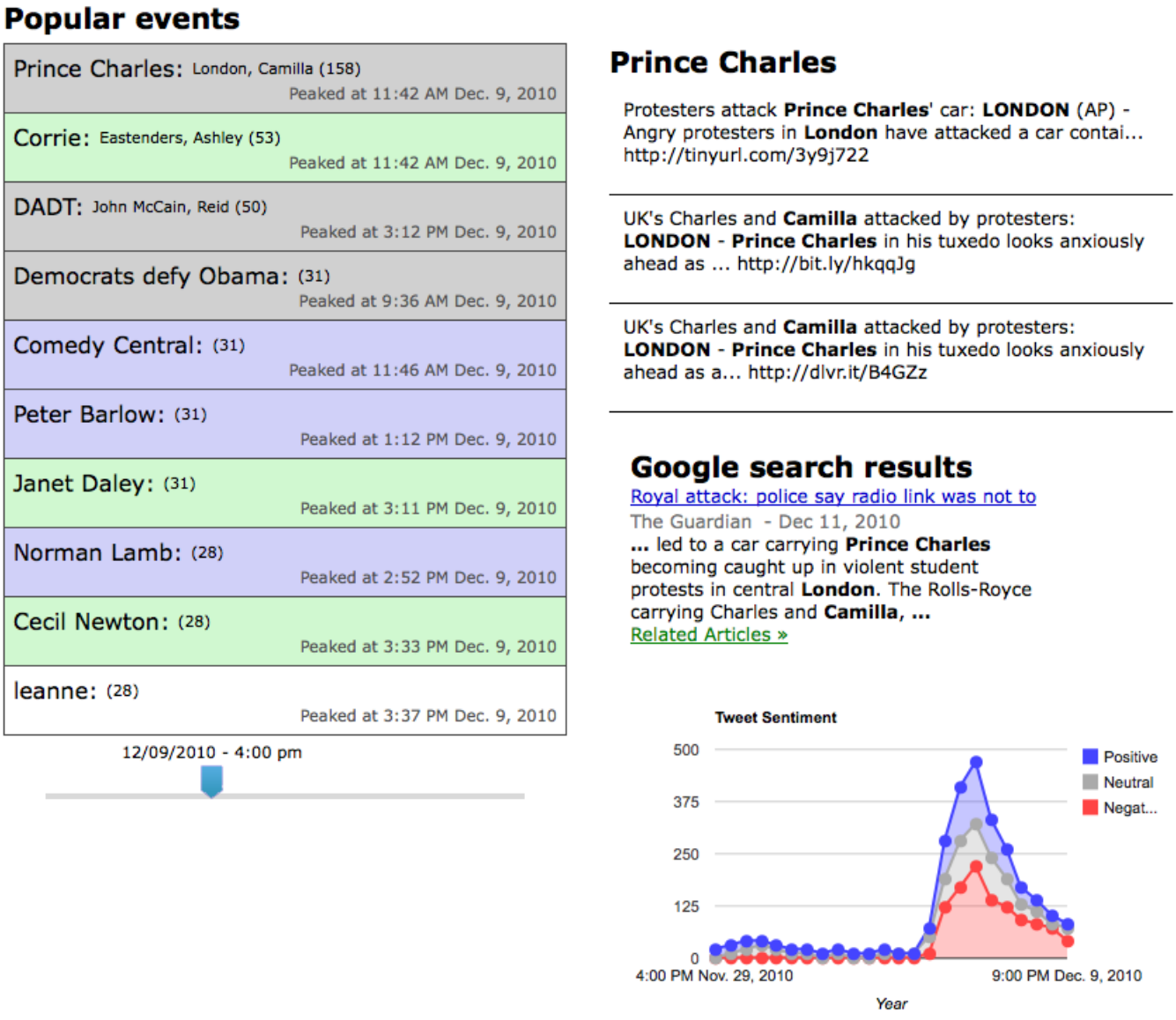
Graph magnitude of trends:
One piece of functionality that the backend was equipped to handle from the beginning was (rudimentary) sentiment trending. We make use of this data in an interesting and informative way through the use of the Google Chart API. Positive, Negative, and Neutral sentiment were recorded for each event, and are displayed over its entire history, to give the user a graphical representation of the magnitude and frequency of various trends. Sentiment was naively categorized based on the appearance of sentiment-bearing words. As with both tweets and Google search results, the graph is displayed for either the top event, or the event most recently selected. The “Year” label for the X-axis is an oversight, but the dates listed at either end of the graph are accurate for the most part. The Y-axis represents the total number of tweets of a given sentiment containing the NE (rounded down to the nearest 10). One bug currently occurring is a lack of present data points for events. If an event occurred earlier for the NE than that data is shown for that. Also, the X-axis labels can be off at times. The data is in the database to handle both of these problems, a few little changes just need to be made to the frontend.

Search:
At the top right of the page lies the Search bar. With this, users can search for the occurrence of various NEs within our database. When the user begins to type, an auto-complete box drops down to offer suggestions. If a valid option is entered, all events containing the given NE

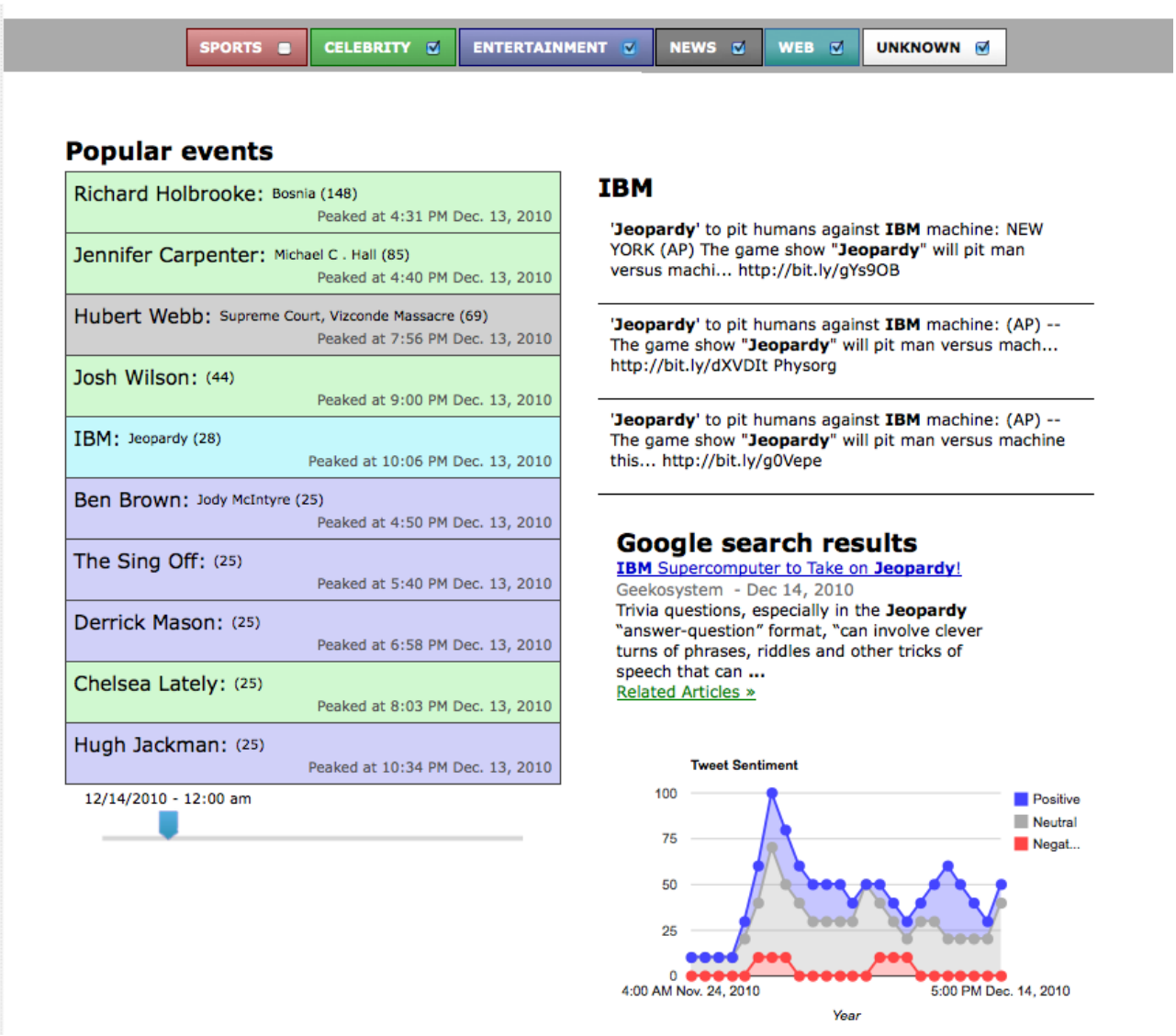
are displayed, along with the time that they began trending. The user can then scroll back to the relevant time slices on the main page, to display trending information for the desired NE. However, the front page only includes the top 10 events at any given moment and search looks at the top 100 events. This means there is a chance the event won't show up when scrolling back in time.

Sample Screen of Typical Usage:

Here is a standard view of the page (excluding categorization and search).



Here is a view after removing sports and clicking on the IBM/Jeopardy topic.



Experiments:

Language Filtering:
In order to test the accuracy of our language filter, 500 tweets from a 24-hour window were randomly selected, and annotated as English or Other. This was done to select a representative subset of all tweets, without placing bias on tweets from any part of the world. If a given tweet contained only language-agnostic content - such as URLs or numbers - then it was classified as Other.

Once annotation was complete, ten-fold cross validation was used to test the precision and recall. In each fold, nine-tenths of the data were used to train the algorithm, while the last tenth was used to test. The precision and recall reported below are an average of the precision and recall in each of the ten folds.

Precision: 0.913
Recall: 0.875

These scores seemed adequate, so we didn't try any other approaches.

Named Entity Categorization:
Originally, the training data for categorizing our NEs was obtained from actual tweets. Tweets from select users were categorized according to the user's self-chosen category on WeFollow (http://wefollow.com/). We began with fourteen categories, selected from WeFollow (as well as an Other category for any outlying NEs):

- Celebrity
- Music
- Socialmedia
- News
- Tech
- Tv
- Actor
- Comedy
- Politics
- Marketing
- Media
- Sports
- Entertainment
- Web

However, after experiencing difficulty differentiating between a few categories, we collapsed the list of categories into five, where each new category is a set of previous categories:

- Celebrity (Celebrity, Actor, Comedy)
- Entertainment (Entertainment, Music, Tv)
- News (News, Media, Politics)
- Sports (Sports)
- Web (Web, Socialmedia)

Both Marketing and Tech were removed, due to a lack of sufficient tweets in either category. With training tweets for each of these five categories, we were able to perform ten-fold cross validation on our categorizer. The average precision and recall scores are given below:

	Precision	Recall
Celebrity	0.408	0.820
Entertainment	0.293	0.262
News	0.451	0.284
Sports	0.773	0.519

- Web 0.130 0.174

However, this was only our first approach at categorization. While Sports and News had high precision, Celebrity greedily over-categorized, and took up the lion's share of NEs. We attempted to adjust priors to balance the categorization out, but ultimately looked into another method for categorization.

Our second approach involved parsing a dump of wikipedia. Wikipedia pages contain a category, although we had to retool them to better serve our purpose. This approach is more difficult to evaluate, since the training data is not something that can be tested on. Additionally, parsing the wikipedia dump took several hours, which was prohibitive for ten-fold cross validation. Instead, 100 random NEs were selected and human annotated. The precision and recall scores for each category are given below:

	Precision	Recall
Celebrity	0.500	0.550
Entertainment	0.367	0.579
News	0.272	0.231
Sports	1.000	0.667
Web	0.333	0.500

Our second approach does an excellent job of categorizing Sports, with 100% precision (in our small human sample size). "Greediness" is distributed evenly between Celebrity and Entertainment (as opposed to just Celebrity), with both categorizing the vast majority of NEs. News is lower in the second approach both in precision and recall, but comprises a small percentage of NEs. Overall, this approach was preferred, due to its high precision for Sports and its high recall for both Celebrity and Entertainment, which are the three most prominent categories on Twitter.

It should be noted that categorization, even for humans, is difficult. Many sports players could be considered celebrities, and entertainment has significant overlap with every category. Many NEs belong to several categories, or none. Similarly, the majority of tweets for a given NE might be of a very different nature than what the NE would imply, providing a source of error for the human annotator.

Named Entity Recognition:
Our project depends largely on NER, and a significant portion of time was spent testing this aspect. The correctness of a given NE can only be determined through human observation. A proper NE contains only the nouns being discussed (eg. 'President Obama' as opposed to 'Democrats defy Obama').

Due to the enormous amount of data being collected, it would be exceedingly difficult to write an NER algorithm with high precision. However, only the correct NEs will be repeatedly discovered. In fact, of the NEs that trend high enough to appear on the front page, 98% are actual NEs!

High precision is necessary to make our product interesting no matter how often a user visits the site.

NER was tested by investigating the events on the front page. Results from this examination are compiled below, under Appendix B: Observation Results.

Co-Occurrence:
Co-Occurrence measures the degree to which NEs were grouped together properly into events. Bad co-occurrence occurs if unrelated NEs are grouped into an event, or if we failed to collapse two events into one. Again, this can only be determined by human observation, which was done through an investigation of the events on the front page.

Our front page boasts 83% precision and 96% recall for co-occurrence. Nearly all trends are collapsed into a single event, although some precision is sacrificed in the process.

Results from this examination are compiled below, under Appendix B: Observation Results.

Google search result:
The google search results are meant to add context to a given event, but occasionally the searches yield information of no relevance to the user. The degree of relevance was determined by human observation, through investigation of the events on the front page.

A total of 68% of google results show relevant results for the given event. While this may seem low, there are very few ways that we could manipulate this score. We could provide more than one result, but this runs the risk of less relevant results regardless. Additionally, a trend may be so new that there are no relevant news stories yet. It's our goal to break the news, after all!

Results from this examination are compiled below, under Appendix B: Observation Results.

Event recognition:
Event recognition deals with how well we discover real events. Precision can be tested through human observation, checking if a given event corresponds to one in the real world (which roughly corresponds to our google search result test, above). Recall, however, is too subjective and intractable for humans to observe. To calculate recall, one would need to know exactly how many events were going on at a given time, that were not predicted by our algorithm. As such, we were unable to determine precision and recall scores for event recognition.

Surprises and things we learned:

Perhaps the biggest surprise was that we could reliably find events and context from the co-occurrence of NEs. This discovery caused us to switch the focus of our product, and steered us towards a more interesting idea. It seems that data can often lead you to decisions you wouldn't expect.

We also learned that using 3 Amazon EC2 machines non-stop for 2 months can be costly. By the end of the quarter, we had invested $500 in Amazon Web Credits into our project, with a grand total of 150 million (and counting!) tweets. We're currently investigating methods to keep it running long-term, as well as switching to a better domain name.

At first, we handled metric tracking within the DB. However, the queries necessary to calculate the trending score were prohibitively slow. One important lesson learned is that it's not enough to have your system function just in realtime, it has to be fast enough to re-process the data (metric scores and events) so that development of ideas can be iterative and the feedback loop short.

Conclusions and ideas for future work:

The Twitter community may be far from all-encompassing, but it is growing every day. It is an untamed frontier where users exchange ideas and opinions, discuss their lives, and share news. Trends on the Internet live and die in the blink of an eye, and most don't pause to consider that news coverage is lagging farther and farther behind events that matter to real people. We are confident in our ability to identify newsworthy events within the Twitter community as they occur. We offer a novel solution to finding trending events on Twitter, and with it have come compelling and contemporary results.

However, people tend to be particular about the news they choose to read, and the news we're collecting isn't (automatically) tailored towards any one group. The majority of the events we collect are either US or British and can range from pop-culture and sports to world events. This product might be improved by gearing it toward a specific group - eg US pop-culture or British sports fans. Events would make it into the top 10 (front page) quicker (less competition), which would result in the news being broken to the user much faster. It would also result in more focused results on the front page, which would add value to the user. Additionally, the look and feel could be changed in such a way that it would be more appealing to the intended audience (only Google can create a site that pop-culture, sports, and news enthusiast are all willing to use!).

Given the time, there are numerous ideas we would have liked to implement. All of our algorithms could be fine-tuned, to increase precision and recall of important data. Search could be augmented, to display a front page containing only events with NEs matching the search result. The display of the graph could be more interesting, or the graph could display trending information for all NEs on the front page. There are a number of features that could be added, but the core functionality of the site has been implemented to our satisfaction.

Appendix A: Division of Labor

Curtis Yamanaka: Frontend Coding

- Search bar for a basic search

- Autocomplete using scriptaculous
- Graph function using google API
- Streaming tweets: First iteration of site had scrolling tweets updated to the site. Unfortunately it was not implemented to the final site.
- Profile Pages: First iteration of site consisted of profile pages for name entities which contained tweets and graph, this was changed to the final version to be updated onto the main page instead.
- Search: Formats search results and displays events and co-occurring name entities.
- Live search: search database for a user given input for a namegraph data
- Graphdata: Given a name entity, grab metric for a given time period to use with graph

Golf Sinteppadon: Front End Design, Coding

- Designed look and feel of website over several design iterations
    - Implemented Slider functionality
    - Designed logo
    - Determined layout of the site
- Displayed queried information to user

Jesse Shepherd: Categorization

- Language Classification - annotated training data, tested, inserted into pipeline
- Category Classification - selected categories, acquired two sets of training data, tested, selected heuristics, inserted into pipeline
- Context Identification (w/ Sam) - find uncommon bigrams and trigrams for context
- Front End Support - support functions, bug testing, user study
- Final Report - contributed to, unified

Jon: Database

- Database schema design and optimization
- Frontend coding help
- Abstracting DB to interfaces for frontend and backend. Example methods:
    - Python
        - pytimeToDbtime(pytime) and dbtimeToPytime(dbtime)
        - putRows(table, colnames, rows, blocking=False)
        - pullRows(table, colnames, maxrows, orderby=”timestamp”)
        - exec(templateSql, params) // returns all rows in result set
    - PHP
        - getEvents(time=null) //uses batch queries and caching for performance
        - getTweets(eventid)

Michael: AWS Management and Data Collection

- Login scripts for EC2 machines and Database
- Creating training data for categories from Wikipedia (worked with Jesse)
- Front end coding help (Google search with Jon)
- Database querying

    - Collect tweets and organize by name entity
    - Collect dictionary of tweet strings and counts of each string
- Testing trending data through google to see if event happened or if it is a spam event

Sam Clark: Everything but frontend

- Setup NER and made robust
- Wired the backend up (robust enough to collect 4 weeks of data)
- Built system for monitoring NE trending scores and event creation
- Organized meetings
- Wrote part of the paper

Appendix B: Observation Results

C1 = Events
C2 = Is NE correct
C3 = Co-occurrence correct
C4 = Google results for event
C5 = Event Should be collapsed

C1	C2	C3	C4	C5
Bob Miley	1	0	0	1
Derrick Rose	1	1	1	0
George Karl	1	1	1	1
The Bulls	1	0	1	0
BILL SIMMONS	1	1	1	1
Indiana Jones	1	1	1	1
Shannon Brown	1	0	0	1
Chris Bosh	1	0	0	1
Golden State	1	0	1	1
Bolivia	1	0	1	1
Bernie Sanders	1	1	0	1
Elizabeth Smart	1	1	1	1
Watford	1	1	1	1
Billy Ray	1	1	1	1
Pamela	1	1	1	1
Sally	1	0	0	1
Janine	1	1	1	1
Barry Manilow	1	1	1	1
Rita	1	1	1	1
curitiba	0	1	0	0
Music Bank	1	1	0	1
David Cameron	1	1	1	1
District Lines	1	0	0	1

LIAM PAYNE	1	0	1	1
Manchester United	1	1	1	1
Glasgow	1	0	0	1
Frankie Boyle	1	1	1	1
Flats	1	0	0	1
Matt Cardle	1	1	0	1
Kinsey	1	0	0	1
Ray Allen	1	0	0	1
Betty White	1	1	1	1
Beyonce	1	1	1	1
Heather Mills	1	0	1	1
James Moody	1	0	1	1
Randy Moss	1	1	0	1
Barbra Walters	1	1	1	1
Reggie Wayne	1	1	0	1
Lee Corso	1	0	1	1
Selena	1	0	1	1
Prince Charles	1	1	1	1
Corrie	1	1	1	1
DADT	1	1	1	1
Democrats defy Obama	0	0	1	1
Comedy Central	1	0	1	1
Peter Barlow	1	0	1	1
Janet Daley	1	0	1	1
norman Lamb	1	0	1	1
Cecil Newton	1	0	1	1
leanne	1	0	0	1
Average	**0.98**	**0.83**	**0.68**	**0.96**

Appendix C: Externally Written Code

- Backend Libraries used:
    - Mallet
    - PyCurl
    - MySQLdB
    - twokenize
- Frontend libraries used:
    - Sriptaculuous
    - Google Charts API
    - Google Search API

Appendix D: README

- Code Locations
    - Code for gathering tweets is in tweet_scrape_main.py
    - Code for running NER on the tweets is in nlp_main.py
    - Code for finding and recording events is in scoring_fast_clean.py
    - Code for classifying language in language_classification/
    - Code for classifying category in category_classification/
    - All DB code is in db/ (tweetdb.py, Dbmethods.py)
- All frontend code is in the www/ directory