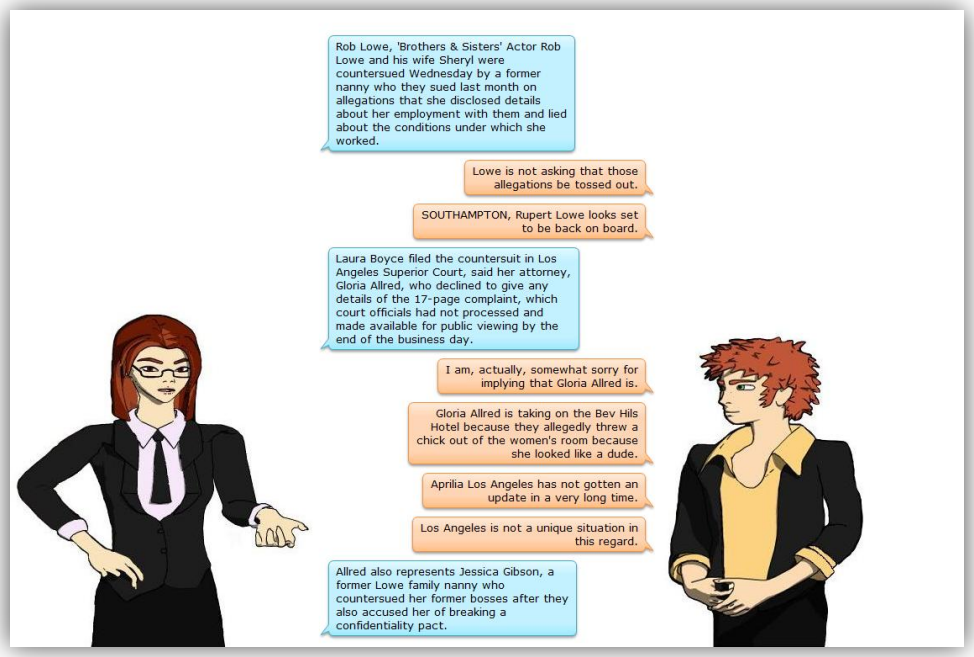
EECS 394: SOFTWARE ENGINEERING

Zap2It Project Report

Prepared for Nate Nichols

Rudy Galfi, James Gross, and Tim Zwiebel

11-Jun-08



Problem

The aim of our project is the creation of a back-end system for serving interesting, subjective comments in response to celebrity- and entertainment-oriented news stories. Our client for this project is Nate Nichols, who is working on a much broader project called "News at Seven," which is an online service that features an animated news anchor reading news stories relevant to users' interests. In a similar way, the intention of our project is to have an original news story that can be read by one anchor while another anchor interjects comments as the story is being read. Most of the celebrity- and entertainment-themed news stories will be provided by an online news service called Zap2It (http://zap2it.com), which provides the name of the overall project.

Thus, the overall goal of the project is to take a given news story and generate a script that can be read by two anchors consisting of the original story and interjected comments pertaining to subjects in the story. While our work is mostly concerned with a back-end system without an interface, we decided that we would provide a web interface in order to display the script and visually see the original story inserted with comments. An additional constraint is that the code for the back-end be written in Python so that our engine can be easily integrated with the whole "News at Seven" engine.

Approach

Our approach to the project consists of two central aspects:

(1) A general strategy that would be employed so that the project could be divided into three separate iterative phases in which each phase built upon work done in the previous phase. In this way, the project could end with a presentable product by the conclusion of the third iteration.
(2) A set of team roles so that each person on the team can specialize in some aspects of the project to allow work to be divided and done efficiently while also ensuring that each team member has adequate knowledge of aspects of the entire project.

General Strategy

The general strategy for the project entailed finding technologies and algorithms to accomplish several aspects of the project:

- A CherryPy web server was used to run a web server that provided a front-end interface for our engine. The web server would simply host a page with a form that accepted a news story in a text box and would allow submission of that form. Then, the story would be processed to serve another page featuring the two-person script that could be read.
- An entity detector based on Wikipedia's categorization of its numerous articles was used to identify the subjects of a story. The entity detector could be primed by associating certain Wikipedia categories with relevant classifications. (For example, the category "People born in 1964" is associated with the classification "person." Thus, all articles in such a category are effectively tagged to be a person.)

- Google's Blog Search API was used to submit search queries to Google and retrieve several results. The search queries used were constructed from the subject and some wrapper text to become a phrase (e.g. "____ makes me"); the intention is that such phrases will elicit interesting comments. From these results, we extracted the title and a preview of the content of the blog entry for use as comments. The use of the content preview was sufficient because the preview includes the searched phrase and, thus, a potential comment of interest.
- Several sentiment analysis tools were used to attempt to score the amount of emotional content in a comment. Once scored, only the most interesting comment would be chosen for use in the final script.

Team Roles

As mentioned previously, it was important to strike a balance where team members could specialize to complete a certain aspect of the project while also ensuring that each team member was always apprised of the progress of each component of the project and had a basic understanding of the approaches being used. Specializations developed somewhat naturally along the following lines:

- James Gross worked with the entity detector and algorithms employing TFIDF, such as the relevance scorer, which attempts to identify how relevant a comment is to a story.
- Rudy Galfi worked with the Google API to retrieve comments, which included the construction of search query phrases and the subsequent extraction of the relevant comment sentence and clean-up of such sentences with regard to punctuation and capitalization.
- Tim Zwiebel worked with various sentiment-analysis tools in order to evaluate each one's effectiveness in scoring the emotional content of comments. He was also tasked with maintaining the web interface.

Communication occurred daily regarding who would be handling each part of the project and at what times. We also set aside meeting times each Wednesday to do more extensive planning and work together to develop code and prepare for iteration demonstrations.

Iterations and Changes

Iteration 1

The first iteration was concerned with establishing a front-end web interface and being able to take a news story and identify the important subjects of that story.

During this first iteration, at the recommendation of our client, we set up a CherryPy web server to establish a front-end web interface to accept news stories through a form and display output on a second page.

We also deliberated methods of finding important words and phrases in the story: TFIDF analysis and entity detection. We ultimately settled on using entity detection after observing how accurate it

was and how it fit our needs very well. We developed a set of categories to submit to the entity detection engine in order to limit the list of results to only entities we cared to use (people, places, etc.). The list of entities was ultimately displayed as the output of our web form.

Iteration 2

The second iteration dealt with taking a list of subjects in a story and finding a large set of comments about those subjects from blogs.

We researched and used Google's Blog Search API after finding that it would return up to eight results from any search query that we fed it. In order to make comment finding easier, we brainstormed a list of common phrases that would appear in an opinionated comment. We then inserted the entity results into these phrases to compose the list of queries that would be used to search Google. We received approximately 100 results from these searches on a moderately popular entity. Additionally, since Google provides snippets of the content of its results that are centered on the query, we found it sufficient to extract only these snippets from the results rather than following the blog URL and scraping the page.

With a wide selection of comments, we set about trying to create functions that would scrub the snippets that were returned because they tended to have very poor punctuation and it was difficult to decipher where sentences (and, thus, comments) began and ended.

With the scrubbing functionality, we were able to output entities and all of the comments related to an entity on the output page of our website.

Iteration 3

The third iteration consisted of evaluating the comments returned on the bases of relevance to the source story, similarity with other comments, and emotional content.

With all of the resources in hand to accomplish the task of inserting comments into a news story, we set about trying to decide which comments were the most worthy of inclusion. We had three criteria for evaluation:

(1) Relevance to the source story
(2) Similarity with other comments
(3) Emotional content

It was desirable to have comments that were about an entity from the story but that had little relevance to the content of the story itself. For this reason, we implemented a function that would evaluate the relative relevance of a comment compared to the story. Comments that were deemed too relevant would be thrown out. This relevance detection was accomplished using a simple string comparison that would evaluate how many of the words in the comment also appeared in the story. We also attempted to solve this problem using TFIDF but found that TFIDF algorithms that ranked the important of words were too inclined to choose words that were unique to rank high instead of words that were important to the story.

Another concern was that of duplicate comments. Blogs tend to have the same phrase in them because that phrase comes from quoted material. Thus, we implemented an algorithm to detect such duplicates and keep only one. The detection of a duplicate requires that less than 5% of the characters in the one comment is different that the characters in a previously seen comment; in this case, the latter comment is discarded. We found this algorithm to work well in getting rid of exact (or near-exact) replications of a story, but it did not catch alternate versions of a news story.

Finally, we needed to judge the emotional content of a comment in order to select one to appear in the final script. For this, we evaluated several alternatives that are discussed in the Appendix, Evaluation and Discussion of Sentiment Analysis Tools. Ultimately, we settled on using functionality from the Natural Language Toolkit (NLTK) to build a corpus of positive and negative words. With this, we are able to analyze a comment by examining each word and determining the probability of it being positive or negative. We found this method to work rather well in, say, 90% of circumstances with regard to the polarity of the comment. To a lesser extent, this approach will correctly sort comments based on their emotional content, which means that the comment supplied will often be of the correct nature, but it will not necessarily be the best choice. We had also considered using the Emotional Classifier for these purposes but decided that its results were far less reliable and more time-consuming to obtain.

Final Result

With the aforementioned analyses complete, our engine was functional and capable of processing a story, finding its entities, obtaining comments about those entities, and choosing the "best" comment to insert into the final script. As a final touch, we made further modifications to the web interface to make the script more graphical in nature for the purposes of display and demonstration. Now, the output page features the dialogue of two of the News at Seven anchors, one who is shown to be reading the story as submitted through the form and the other who is commenting on the story using the retrieved comments.

Conclusion and Future Work

We believe that our final product addresses the requirements of the original problem very well. We believe the front-end interface is very helpful in visualizing the desired system, which is one that accepts a news story and generates a script for two people, one of whom is reading the story and the other who is interjecting comments about people, places, and things in the story.

Moreover, our back-end system accomplishes many of the goals of the project such as entity detection, comment-finding, and comment-ranking. Still, there is more work that can be done.

- More processing can be done with the entity results to make sure that the entities actually appear in the story.
- More trigger phrases could be added to increase the diversity of comments and bring in more potential results.
- Comments could be evaluated based on the correctness of their grammar in order to decide which comments are nonsensical and should be discarded.

- More scrubbing of comments strings would be desirable to reduce the amount of errant punctuation that is common in blogs. There should also be a more accurate way of detecting the beginning and end of sentences even when those sentences do not begin and end with the typical indicative punctuation.
- The detection of duplicates and the evaluation of relevance to the story could be made more sophisticated.
- Other methods of sentiment analysis could be used to judge the polarity and emotional content of a comment.

While numerous areas of expansion remain, we feel that our final product has already taken a great step forward in all of these areas and presents a fun, new spin on getting celebrity news.

Appendix: **Evaluation and Discussion of Sentiment Analysis Tools**

Overview

Sentiment analysis refers to attempting to determine the emotion of a string of text. We are concerned with sentiment analysis because it can help us pick the most interesting comments out of a list of comments. Once we have a list of sentences about a subject, we need to decide which one to pick. Sentiment analysis can help us determine the subjectivity as well as the polarity of a sentence. We want our sentences to be subjective and either very positive or very negative to ensure that they are interesting comments.

Possibilities

There are many existing tools that offer sentiment analysis. We looked into several of them, before choosing one to implement in our demo. Below are the results of our research.

LingPipe

LingPipe offers sentiment analysis through a suite of Java libraries. They offer a range of licenses that range from free to $40,000 per server.

http://alias-i.com/lingpipe/index.html

Lexalytics

Lexalytics offers a Sentiment Toolkit, which is based on their Salience Engine. They have APIs in C, Java, Python, PHP, and C#.

http://www.lexalytics.com/index-4.html

RapidMiner (formerly YALE)

RapidMiner is another sentiment analysis tool. It appears to be based on WEKA (data mining software in Java from the University of Waikato). They offer both open source and closed source options.

http://rapid-i.com/content/blogcategory/10/68/lang,en/

OpinionFinder

OpinionFinder is a simple sentiment analysis tool offered by the University of Pittsburgh. They offer a free license for research and teaching purposes.

http://www.cs.pitt.edu/mpqa/opinionfinderrelease/

NLTK

The Natural Language Toolkit is a suite of open source Python modules for natural language processing.

http://nltk.org/index.php/Main_Page

Our Choice

We chose to use NLTK in our demo. We chose it because it is in Python, and it is open source and has good documentation. Specifically, we used the `NaiveBayesClassifier` class which is a simple probabilistic classifier.

NLTK Classifier Demo

During our research, we came across a demo that guessed the gender of names. The demo used the NaiveBayesClassifier from NLTK. First, it trains the classifier with a corpus of male and female names. When a name is analyzed, it is assigned a set of features. In the demo, the set of features included what letters were in the name, how many times a letter was used, the first letter of the name, and the last letter of the name. Once the classifier has been trained, it assigns probabilities to certain features. For example, after the classifier is trained, it might determine that names ending in the letter 'a' have a high probability of being female.

Our Implementation

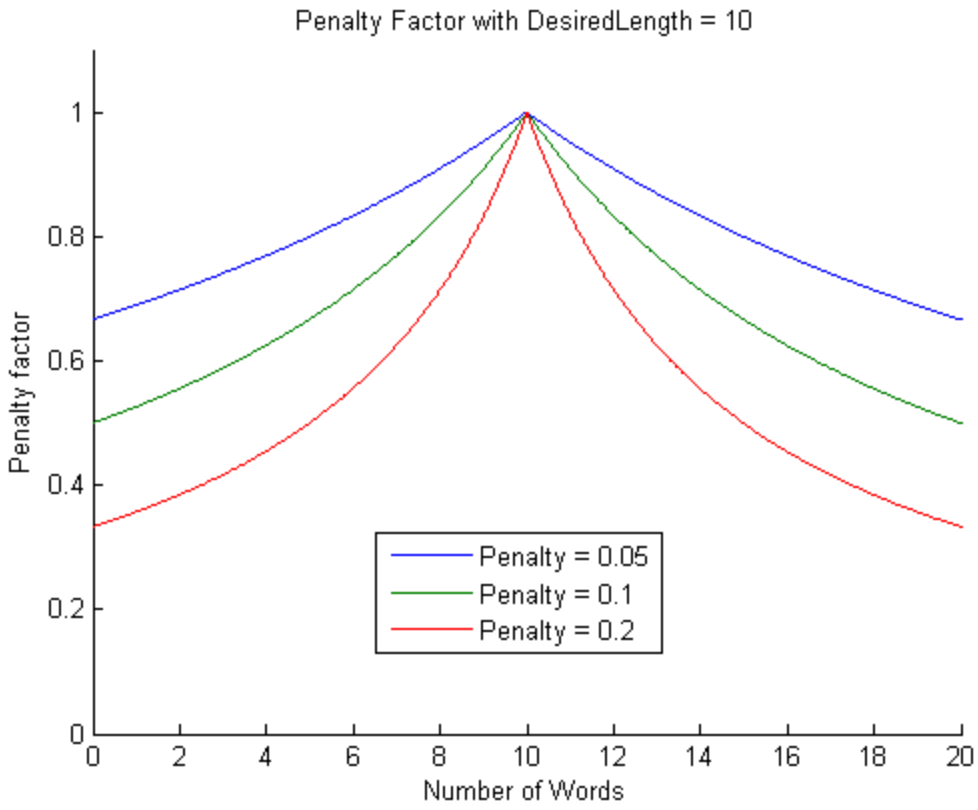
Our implementation used a strategy similar to the one in the NLTK classifier demo. We start by giving the classifier a list of positive words and a list of negative words. When a sentence is analyzed, it is assigned features. Currently our set of features only has what words are in the sentence. Based on the words in the sentence and the words in the corpus, the classifier gives a probability of a sentence being positive or negative. Then, we manipulate this emotional score so that a neutral sentence (a sentence with no words) has a value of 0.0 and the scores range from -2.0 to +2.0. Finally, we use this emotional score as well as the number of words in the sentence to generate a final score for the comment. The formula is given below:

$$OverallScore = \frac{EmotionalScore}{Penalty * |NumberOfWords - DesiredLength| + 1}$$

DesiredLength = the number of words in an ideal comment

Penalty = the penalty factor for a comment being too long or too short

The penalty factor is the number multiplied by *EmotionalScore* to get *OverallScore*. Below is a graph of the penalty factor.



As *Penalty* increases, the curve becomes steeper and *OverallScore* becomes a smaller fraction of *EmotionalScore*. Taking the comments with the highest absolute value of the overall score should theoretically give us the comments that are the most emotional and that are closest to the desired length.

Extension

Our method is simple and obtains decent results, but it can be improved greatly. First of all, a larger corpus with full sentences would benefit the classifier. Also, expanding the set of features used to train the classifier would likely return better results. We tried using a corpus with positive and negative movie reviews in addition to the emotional words, but the emotional words alone yielded better results, most likely due to the simplistic nature of the set of features we are using. If the set of features were made more complex by incorporating grammar, it could greatly improve the classifier. For example, the word 'like' has a positive connotation, as in the sentence “I like Chicago”, but the sentence “John Doe acts like an idiot” is not a positive sentence. In that case, like is used in a simile and is neutral. If the set of features included grammatical features, the classifier might be able to better distinguish between the two.