1 Introduction

Nowadays, people tend to express their opinions in their own blog space. Usually, different people have different thoughts on the same event from different aspects. For example, student A might think UIUC is a great place to be a graduate student because of its excellent faculty and challenging research projects, while student B thinks that UIUC classes are generally more difficult on average and complains that he doesn’t have time to hang out with friends; student C thinks UIUC is awesome because there are few distractions so he can enjoy his studies, while student D thinks life at UIUC is very sad and boring as there are not many places to have fun.

This project aims to implement a Blog Miner, which could automatically extract opinions from blogs to summarize different thoughts on the same event from multiple aspects. Basically, the Blog Miner will primarily start as an opinion finder, based on the many public weblogs available online. Users will be able to enter in keywords and pick a subject area to find the popular opinions of topics of interest. For example, the keyword “Harry Potter” and the subject area of “Movie” would give users some notion of the popular opinion of the movie Harry Potter taken from blogs. This work involves two tasks: (1) modeling the multiple aspects of opinions for each subject (e.g., cars, movies); (2) extracting sentiments/opinions for each aspect.

The mining results will be represented in two dimensions: theme and sentiment. In the theme dimension, each blog article will be partitioned into multiple themes by classifying sentences. In the sentiment dimension, each sentence will be measured by how positive/negative its opinion is about the query topic. In this way, by searching for “Toyota Camry”, the user would expect to see what people say about the price, engine, maintainance etc., and what the positive or negative opinions are for each aspect.

2 Related Work

The explosive spread of weblogs (or blogs for short) has attracted increasing research work in automatically mining large numbers of blog pages for opinions and recommendations. Blogs have become a prevailing type of media on the Internet [1] and they are now providing rich information to benefit people in daily life, especially through internet search activities. Compared with traditional media such as online news sources (e.g., CNN online) and public websites maintained by companies or organizations (e.g., Yahoo!), the blog space has unique characteristics. Generally, the content of blogs are highly personal and they are usually associated with the personal information of their authors, such as personal experience and personal interests [2].

There are currently some research attempts on extracting personal opinions from blogs. However, they mainly consider extracting positive and negative opinions (i.e., sentiments) on the same event without taking into account that those opinions are usually for different aspects of the event. As personal opinions on the same event are typically addressed in different aspects, in this project, we want to extract the opinions from multiple aspects (or themes).

In text mining, some previous studies have presented several probabilistic models to model themes in documents (e.g., PLSI [3] and LDA [4]). Here, themes are defined as subtopics associated with a broad event (or topic). In [5], PLSI is extended to include a background component to explain the non-informative background words. A cross-collection mixture model was proposed in [5] to support comparative text mining. These existing techniques for theme extraction in text mining can be adapted to extract blog opinions over
different themes. Currently, there are two main kinds of similar blog services available: Google Blog Searcher\(^1\), which is designed to search the public weblogs available online for input queries; and Opinmind\(^2\), which is a newly developed search engine to mine the internet for sentiments.

This work differs from the existing work in two aspects: (1) we model the multiple aspects of opinions within each blog article; (2) we extract opinions for each theme. That is, we mine the blogs in a multi-dimension manner. The mining results will be represented as different opinions in multiple aspects.

### 3 System Overview

Figure 1 illustrates the system architecture. Basically, we will first build the theme models for each subject offline. When an online query comes, the crawler (i.e., Google Blog Searcher) will be invoked to crawl relevant blog articles. Each article goes through a sentence splitter to be broken to a set of sentences. Then each sentence will be assigned to a theme (or aspect) according to its relevance to the offline, pre-computed theme models. Meanwhile, each sentence is scored by its similarity to positive and negative sentiment models. In the end, sentences for each theme-sentiment pair are ranked according to a ranking strategy, and the top ranked sentences are displayed in the result page.

![System Architecture](image)

**Figure 1: System Architecture**

### 4 Implementation

There are six major components implemented:

1. **Crawler** A perl script was written to use Google Blog Search to crawl blog articles for a specific query topic.

2. **Model learner** The theme models and sentiment models are built using a comparative text mining technique based on the cross-collection mixture model proposed by [5]. Section 5 will go through more detail of this method.

3. **Sentence splitter** We used a perl script modified from the open source code “sentence-boundary.pl”\(^3\) to break each blog article into a set of sentences.

4. **Scoring** To classify each sentence into relevant aspects, we used the negative KL-divergence function to measure the similarity between the blog sentence and the theme model.

\[
-D(\theta_s||\theta_m) = \sum_w p(w|\theta_s) \log p(w|\theta_m) - \sum_w p(w|\theta_s) \log p(w|\theta_s),
\]

where \(\theta_s\), \(\theta_m\) represents the language model of the sentence and the theme respectively. Sentences are assigned to the highest scored theme. KL-divergence was also used to measure the sentiment given the positive/negative sentiment models. That is, the positive (negative) score is computed by the negative KL-divergence function of the sentence language model and the positive (negative) sentiment model.

---

\(^1\)http://blogsearch.google.com/

\(^2\)http://www.opinmind.com/

\(^3\)downloaded from http://www.algorithm.cs.sunysb.edu/lydia/documentation/sentence-boundary.html
5. **Ranking** For each theme-sentiment pair (e.g., positive opinions about the price of Toyota Camry), we need a ranking strategy to rank the sentences. One tricky issue is, the highly ranked sentences should be both relevant to the theme and with strong sentiment. For example, the sentence *Should I buy it mostly for reliability?* only indicates strong relevance to the theme of reliability, but barely contains any sentiment; sentence *The driver was a shaggy, scruffy, and unquestionably filthy chain smoker.* shows strong negative opinion, but has nothing to do with the theme of maintainance. Under this purpose, we chose a F1-like measure to balance the theme score and the sentiment score for the sentence ranking, i.e.,

\[
Score = \frac{S_{\text{theme}} \cdot S_{\text{sentiment}}}{S_{\text{theme}} + S_{\text{sentiment}}}
\]

Top ranked sentences are chosen to be displayed as the mining results.

6. **Interface** We implemented a very simple web interface to demonstrate our Blog Miner\(^4\). It takes 3 inputs: 1) number of blog articles to crawl; 2) query key words; 3) query subject.

5 **Comparative Text Mining**

We attempted two models to compute the theme models and sentiment models. Both approaches are variance of the cross-collection mixture model proposed in [5].

In the first method, we compute the theme models and sentiment models separately from collected training data. For computing the theme models, each collection is the set of training documents for a specific theme. For computing the sentiment models, each collection is the set of training documents for a sentiment (i.e., positive or negative). Hence there are \(m\) collections when computing the theme models (where \(m\) is the number of themes), and two collections when computing the two sentiment models. Basically, this method is a supervised learning of the language models since the theme labels and the sentiment labels are already given for the training data.

In the second method, we compute the theme models and sentiment models simultaneously from collected training data. In this model, only the sentiment labels are given, and each collection is the set of training documents for a sentiment. The theme models are computed in an unsupervised manner, given the number of themes specified.

The following discusses each method in detail, and gives corresponding EM updating formulas.

5.1 **Learning Theme Model and Sentiment Model Separately**

![Figure 2: 1D Mixture Model for Theme/Sentiment](http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get_query.pl)

Figure 2 illustrates the idea for explicitly distinguishing common background model that characterizes common information across all collections from special theme models that characterize collection-specific

\(^4\)http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get_query.pl
information. It involves a common background model as well as a collection-specific models for each collection. The basic idea of this method is to treat the words as observations from a mixture model where the component models are the collection-specific word distributions and a background word distribution across all collections. The collection-specific models (i.e., theme/sentiment models in our task) can be estimated using the Expectation Maximization (EM) algorithm to obtain the collection-specific word distributions.

Specifically, let \( \mathcal{C} = \{C_1, C_2, \ldots, C_m\} \) be \( m \) comparable collections of documents. Let \( \theta_1, \cdots, \theta_m \) be \( m \) collection unigram language models (i.e., word distributions) and \( \theta_B \) be a background model for the whole collection \( \mathcal{C} \). A document \( d \) is regarded as a sample of the following mixture model.

\[
p_d(w) = \lambda_B p(w|\theta_B) + (1 - \lambda_B)p(w|\theta_i),
\]

where \( w \) is a word, \( d \in C_i \), \( \lambda_B \) is the mixing weight of the background model \( \theta_B \). The log-likelihood of all the collections \( \mathcal{C} \) is

\[
\log p(\mathcal{C}) = \sum_{i=1}^{m} \sum_{d \in C_i} \sum_{w \in V} \{c(w, d) \log[\lambda_B p(w|\theta_B) + (1 - \lambda_B)p(w|\theta_i)]\}
\]

where \( c(w, d) \) is the count of word \( w \) in document \( d \).

According to the EM algorithm, we can use the following iterative updating formulas to estimate all the parameters. \( \{z_{d,w}\} \) is a hidden variable and \( p(z_{d,w} = i) \) indicates that the word \( w \) in document \( d \) is generated by theme \( i \). The algorithm will terminate when it achieves a local maximum of the log likelihood. In our experiments, we use multiple trials to improve the local maximum we obtain.

\[
p(z_{d,w} = B) = \frac{\lambda_B p^{(n)}(w|\theta_B)}{\lambda_B p^{(n)}(w|\theta_B) + (1 - \lambda_B)p^{(n)}(w|\theta_i)}
\]

\[
p(z_{d,w} = i) = \frac{(1 - \lambda_B)p^{(n)}(w|\theta_i)}{\lambda_B p^{(n)}(w|\theta_B) + (1 - \lambda_B)p^{(n)}(w|\theta_i)}
\]

\[
p^{(n+1)}(w|\theta_B) = \frac{\sum_{i=1}^{m} \sum_{d \in C_i} c(w, d)p(z_{d,w} = B)}{\sum_{w' \in V} \sum_{i=1}^{m} \sum_{d \in C_i} c(w', d)p(z_{d,w'} = B)}
\]

\[
p^{(n+1)}(w|\theta_i) = \frac{\sum_{d \in C_i} c(w, d)(1 - p(z_{d,w} = B))p(z_{d,w} = i)}{\sum_{w' \in V} \sum_{d \in C_i} c(w', d)(1 - p(z_{d,w'} = B))p(z_{d,w'} = i)}
\]

In our task, for computing the theme models, each collection is the set of training documents for a specific theme; for computing the sentiment models, each collection is the set of training documents for a sentiment, then \( m = 2 \).

### 5.2 Learning Theme Model and Sentiment Model Simultaneously

Figure 3 illustrates the idea for explicitly distinguishing common theme models that characterize common information across all collections from special sentiment models that characterize the sentiment collection-specific information. Thus we now consider \( k \) latent common themes as well as two sentiment models for two sentiment collections. The basic idea of this method is to treat the words as observations from a mixture model where the component models are the two sentiment (collection-specific) word distributions, \( k \) theme word distributions across all collections and a background word distribution. The sentiment models and the theme models can be estimated using the Expectation Maximization (EM) algorithm to obtain the collection-specific word distributions and the theme word distributions.

Specifically, let \( \mathcal{C} = \{C_1, C_2\} \) be the two sentiment collections of documents. Let \( \theta_1, \cdots, \theta_k \) be \( k \) theme unigram language models, \( \varphi_1, \varphi_2 \) be the 2 sentiment unigram language models and \( \theta_B \) be the background model for the whole collection \( \mathcal{C} \). Then a document \( d \) is regarded as a sample of the following mixture model.

\[
p_d(w) = \lambda_B p(w|\theta_B) + (1 - \lambda_B)\sum_{j=1}^{k} \pi_{d,j}[\lambda_C p(w|\theta_j) + (1 - \lambda_C)p(w|\theta_i)],
\]
positive  negative

![Diagram of a model with themes and mixing weights]

Figure 3: 2D Mixture Model for Theme & Sentiment

where \( w \) is a word, \( d \in C_i \), \( \pi_{d,j} \) is a document specific mixing weight for the \( j \)-th aspect theme, and \( \sum_{j=1}^{k} \pi_{d,j} = 1 \). \( \lambda_B \) is the mixing weight of the background model \( \theta_B \). The log-likelihood of all the collections \( C \) is

\[
\log p(C) = \sum_{i=1}^{2} \sum_{d \in C_i} \sum_{w \in V} \left\{ c(w, d) \log[\lambda_B p(w|\theta_B) + (1 - \lambda_B) \sum_{j=1}^{k} \pi_{d,j} [\lambda_C p(w|\theta_j) + (1 - \lambda_C) p(w|\theta_i)]] \right\}
\]

where \( c(w, d) \) is the count of word \( w \) in document \( d \).

Similarly, the following iterative updating formulas are for parameter estimations. \( \{z_{d,w}\} \) and \( \{y_{d,w}\} \) is hidden variables. \( p(z_{d,w} = j) \) indicates that the word \( w \) in document \( d \) is generated by theme model \( j \), and \( p(y_{d,w} = i) \) indicates that the word \( w \) in document \( d \) is generated by sentiment model \( i \).

\[
p(z_{d,w} = B) = \frac{\lambda_B p^{(n)}(w|\theta_B)}{\lambda_B p^{(n)}(w|\theta_B) + (1 - \lambda_B) \sum_{j=1}^{k} \pi_{d,j}^{(n)} [\lambda_C p^{(n)}(w|\theta_j) + (1 - \lambda_C) p^{(n)}(w|\theta_i)]}
\]

\[
p(y_{d,w} = i | z_{d,w} = j) = \frac{\lambda_C p^{(n)}(w|\theta_j)}{\lambda_C p^{(n)}(w|\theta_j) + (1 - \lambda_C) p^{(n)}(w|\theta_i)}
\]

\[
p(z_{d,w} = j | z_{d,w} \neq B) = \frac{\pi_{d,j}^{(n)} [\lambda_C p^{(n)}(w|\theta_j) + (1 - \lambda_C) p^{(n)}(w|\theta_i)]}{\sum_{j=1}^{k} \pi_{d,j}^{(n)} [\lambda_C p^{(n)}(w|\theta_j) + (1 - \lambda_C) p^{(n)}(w|\theta_i)]}
\]

\[
\pi_{d,j} = \frac{\sum_{w \in V} c(w, d)(1 - p^{(n)}(z_{d,w} = B))p(z_{d,w} = j | z_{d,w} \neq B)p(y_{d,w} = i | z_{d,w} = j)}{\sum_{w \in V} \sum_{i=1}^{2} \sum_{d \in C_i} c(w, d)(1 - p^{(n)}(z_{d,w} = B))p(z_{d,w} = j | z_{d,w} \neq B)p(y_{d,w} = i | z_{d,w} \neq B)}
\]

\[
p^{(n+1)}(w|\theta_j) = \frac{\sum_{w' \in V} \sum_{d \in C_i} c(w', d)(1 - p(z_{d,w} = B))p(z_{d,w} = j | z_{d,w} \neq B)p(y_{d,w} = i | z_{d,w} = j)}{\sum_{w' \in V} \sum_{d \in C_i} \sum_{j=1}^{k} c(w', d)(1 - p(z_{d,w} = B))p(z_{d,w} = j | z_{d,w} \neq B)(1 - p(y_{d,w} = i | z_{d,w} = j))}
\]

\[
p^{(n+1)}(w|\theta_i) = \frac{\sum_{w' \in V} \sum_{d \in C_i} \sum_{j=1}^{k} c(w', d)(1 - p(z_{d,w} = B))p(z_{d,w} = j | z_{d,w} \neq B)(1 - p(y_{d,w} \neq i | z_{d,w} = j))}{\sum_{w' \in V} \sum_{d \in C_i} \sum_{j=1}^{k} c(w', d)(1 - p(z_{d,w} = B))(1 - p(y_{d,w} \neq i | z_{d,w} = j))}
\]

6 Experiments

For each method discussed in Section 5, we experimented on one query subject.

In the first method (i.e., 1D mixture model), for training data of theme models, we downloaded the customer reviews for three aspects of the cars\(^5\): price, mpg, and maintainance. For each aspect, we collected

*from http://www.edmunds.com/insideline/do/ForumsLanding
around 100 reviews for five car models. In total, we have around 300 customer reviews for three collections, and each collection (corresponding to one theme) contains around 100 articles. Fit the mixture model in Section 5.1 to this data set, we estimated the three themes models. We also used Opinmind to collect positive and negative opinions about different cars, and computed the sentiment models by fitting the same mixture model with \( m = 2 \) to the data. Table 1 shows the 20 most frequent words for the computed language model. It indicated this mixture model works well to extract the specific word distribution in each collection.

<table>
<thead>
<tr>
<th>Theme 1 (finance)</th>
<th>Theme 2 (mpg)</th>
<th>Theme 3 (maintains)</th>
<th>Sentiment 1 (positive)</th>
<th>Sentiment 2 (negative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>otd</td>
<td>highway</td>
<td>fluid</td>
<td>awesome</td>
<td>hate</td>
</tr>
<tr>
<td>msrp</td>
<td>mph</td>
<td>oil</td>
<td>love</td>
<td>boring</td>
</tr>
<tr>
<td>rebates</td>
<td>traffic</td>
<td>maintenance</td>
<td>loved</td>
<td>sucked</td>
</tr>
<tr>
<td>invoice</td>
<td>32</td>
<td>viscosity</td>
<td>freddie</td>
<td>stupid</td>
</tr>
<tr>
<td>rebate</td>
<td>75</td>
<td>belt</td>
<td>prefer</td>
<td>horrible</td>
</tr>
<tr>
<td>edmunds</td>
<td>averaged</td>
<td>filter</td>
<td>loves</td>
<td>ugly</td>
</tr>
<tr>
<td>mats</td>
<td>cruise</td>
<td>replace</td>
<td>together</td>
<td>difficult</td>
</tr>
<tr>
<td>fees</td>
<td>tank</td>
<td>plastic</td>
<td>amazing</td>
<td>drivers</td>
</tr>
<tr>
<td>dealers</td>
<td>mile</td>
<td>timing</td>
<td>neat</td>
<td>suck</td>
</tr>
<tr>
<td>ttl</td>
<td>speeds</td>
<td>box</td>
<td>might</td>
<td>desperate</td>
</tr>
<tr>
<td>card</td>
<td>mpg</td>
<td>release</td>
<td>decent</td>
<td>information</td>
</tr>
<tr>
<td>discount</td>
<td>65</td>
<td>brake</td>
<td>building</td>
<td>owners</td>
</tr>
<tr>
<td>quote</td>
<td>gallons</td>
<td>yourself</td>
<td>asked</td>
<td>sucked</td>
</tr>
<tr>
<td>doc</td>
<td>hwy</td>
<td>lock</td>
<td>adore</td>
<td>start</td>
</tr>
<tr>
<td>civic</td>
<td>trips</td>
<td>key</td>
<td>hilarious</td>
<td>stupidest</td>
</tr>
<tr>
<td>leftover</td>
<td>filled</td>
<td>required</td>
<td>common</td>
<td>himself</td>
</tr>
<tr>
<td>jerk</td>
<td>steady</td>
<td>parts</td>
<td>willing</td>
<td>lets</td>
</tr>
<tr>
<td>invoice</td>
<td>85</td>
<td>inspected</td>
<td>currently</td>
<td>frustrated</td>
</tr>
<tr>
<td>college</td>
<td>limit</td>
<td>oils</td>
<td>bless</td>
<td>rental</td>
</tr>
<tr>
<td>website</td>
<td>overall</td>
<td>valve</td>
<td>opportunity</td>
<td>saudi</td>
</tr>
</tbody>
</table>

Table 1: Language Models for Cars

In the second method (i.e., 2D mixture model), we downloaded the review articles for movies from Lillian Lee’s group\(^6\). We fit the mixture model in Section 5.2 to the training data set, and estimated the five themes models and the two sentiment models simultaneously. Table 2 shows the 20 most frequent words for the computed language model. It indicated this mixture model works well to extract the specific word distribution in each collection. Comparing the themes models with that computed from the first method, we can see that unsupervised theme learning apparently does not work as well as that in the supervised manner. That is, the collection-specific word distributions are easier to be extracted than the latent theme word distributions. In Table 2, we somehow interpreted each theme model by looking at the frequent words. However, this interpretation is not accurate and only reflects the subjective understanding of the word distributions.

We developed an online demo to illustrate our Blog Miner at http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get\_query.pl. There are 4 query results are cached for the ease of experimentation. They are:

- “honda”: http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get\_query\_cached1.pl
- “honda accord”: http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get\_query\_cached2.pl
- “toyota camry”: http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get\_query\_cached3.pl
- “lord of the rings”: http://csil-linux49.cs.uiuc.edu:8888/cgi-bin/get\_query\_cached4.pl

The result for “toyota camry” looks best among the above four examples. However, the result for movie “lord of rings” does not make much sense. It is probably due to the worse themes models for movies computed from the 2D mixture model.

\(^6\)http://www.cs.cornell.edu/people/pabo/movie-review-data/
Currently, we are adapting the open source perl script “sentence-boundary.pl” for splitting the sentences of the original blog articles. It turns out this program does not work very well for blog articles mainly because these articles are written in a life-journal-like manner. When people wrote these articles, they really did not pay much attention to the grammar and the punctuation usages. Sometimes, you will see some sentences are not able to be split by the program. One of the future work could be using simple heuristics or rules to improve the sentence splitter.

We also noticed from the experiments that 1D mixture model works better than the 2D mixture models in extracting the theme models. That is mainly due to the gaps between supervised and unsupervised approaches. Given data with labeled themes, we expect to see a “smarter” and more accurate Blog Miner.

Further attempts to develop a better mixture model is also an interesting research direction for future work. The bad results seen in the experiment are somehow also caused by the ranking strategy. Currently, we used the KL-divergence to compute the relevance score for the themes and sentiments. These scores are combined in a heuristic way by the F1-like scoring function 5. There is plenty of room for further improving the ranking strategy. The research question to ask is: How do we combine the theme relevance score and the sentiment score so that the sentences are relevant to both the theme and the sentiment?

Also, we could think about what other functionalities could be developed for mining opinions from blogs.

### 7 Problems and Future Improvement

Table 2: Language Models for Movies

<table>
<thead>
<tr>
<th>Theme 1 (actor/actress)</th>
<th>Theme 2 (horror)</th>
<th>Theme 3 (action)</th>
<th>Theme 4 (love)</th>
<th>Theme 5 (sci-fi)</th>
<th>Sentiment 1 (positive)</th>
<th>Sentiment 2 (negative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>scream</td>
<td>batman</td>
<td>jackie</td>
<td>alien</td>
<td>awesome</td>
<td>stupid</td>
</tr>
<tr>
<td>trek</td>
<td>mulan</td>
<td>joe</td>
<td>trump</td>
<td>wars</td>
<td>amazing</td>
<td>depressing</td>
</tr>
<tr>
<td>wild</td>
<td>horror</td>
<td>flynt</td>
<td>chan</td>
<td>godzilla</td>
<td>love</td>
<td>terrible</td>
</tr>
<tr>
<td>ryan</td>
<td>mars</td>
<td>spawn</td>
<td>tarzan</td>
<td>star</td>
<td>laced</td>
<td>evil</td>
</tr>
<tr>
<td>carter</td>
<td>killer</td>
<td>king</td>
<td>titanic</td>
<td>aliens</td>
<td>volume</td>
<td>awful</td>
</tr>
<tr>
<td>jones</td>
<td>williamson</td>
<td>10</td>
<td>ship</td>
<td>jedi</td>
<td>copy</td>
<td>difficult</td>
</tr>
<tr>
<td>jay</td>
<td>mission</td>
<td>seagal</td>
<td>tarantino</td>
<td>menace</td>
<td>smiles</td>
<td>late</td>
</tr>
<tr>
<td>spice</td>
<td>melvin</td>
<td>love</td>
<td>carrey</td>
<td>effects</td>
<td>tonight</td>
<td>sucked</td>
</tr>
<tr>
<td>nbsp</td>
<td>urban</td>
<td>video</td>
<td>sandler</td>
<td>troopers</td>
<td>loves</td>
<td>lousy</td>
</tr>
<tr>
<td>54</td>
<td>species</td>
<td>schumacher</td>
<td>nr</td>
<td>phantom</td>
<td>role</td>
<td>head</td>
</tr>
<tr>
<td>bulworth</td>
<td>simon</td>
<td>harry</td>
<td>evil</td>
<td>will</td>
<td>enjoying</td>
<td>dislike</td>
</tr>
<tr>
<td>hanks</td>
<td>sidney</td>
<td>wrestling</td>
<td>hong</td>
<td>ripley</td>
<td>appreciate</td>
<td>knight</td>
</tr>
<tr>
<td>house</td>
<td>seth</td>
<td>babe</td>
<td>rose</td>
<td>shrek</td>
<td>onto</td>
<td>annoying</td>
</tr>
<tr>
<td>girls</td>
<td>2</td>
<td>pig</td>
<td>martial</td>
<td>lucas</td>
<td>scraping</td>
<td>useless</td>
</tr>
<tr>
<td>sam</td>
<td>run</td>
<td>robin</td>
<td>ordell</td>
<td>starship</td>
<td>nicey</td>
<td>beast</td>
</tr>
<tr>
<td>judd</td>
<td>damme</td>
<td>jolie</td>
<td>disney</td>
<td>patch</td>
<td>men</td>
<td>loathe</td>
</tr>
<tr>
<td>silent</td>
<td>last</td>
<td>anna</td>
<td>fight</td>
<td>jar</td>
<td>perfect</td>
<td>plane</td>
</tr>
<tr>
<td>west</td>
<td>music</td>
<td>cop</td>
<td>julia</td>
<td>special</td>
<td>several</td>
<td>fake</td>
</tr>
<tr>
<td>beatty</td>
<td>slasher</td>
<td>bats</td>
<td>brown</td>
<td>rising</td>
<td>saying</td>
<td>against</td>
</tr>
<tr>
<td>tom</td>
<td>arnold</td>
<td>pacino</td>
<td>austin</td>
<td>lebowski</td>
<td>tonite</td>
<td>country</td>
</tr>
</tbody>
</table>

### 8 Work Division

Xu ling and Matthew Ryan Wondra had weekly discussions about the project, and implemented the system collaboratively as the follows:

- Matthew implemented the Crawler for crawling training blog articles.
- Xu implemented CTM models for estimating theme and sentiment language models.
- Xu adapted the open source sentence splitter for splitting sentences of crawled blog articles.
- Xu implemented the sentence scoring and ranking strategies.
• Matthew implemented the online demo interface.

References


