Ben Roth
6/6/2010
Final project writeup – CS224N

Topic Extraction and Relation in Instant Messaging

Introduction

The task of topic extraction and relation attempts to first identify important topics in a corpus and then draw relationships between them. Some research has explored topic segmentation—simply dividing the corpus according to when the topic changes. Additional research has tried to extract positive and negative sentiment from product reviews and discussion comments. However, the idea of relating extracted topics in a corpus is also interesting. One could imagine a question and answer system that benefits from the ability to associate different keywords that may not necessarily be associated by a simple thesaurus. For example, in a conversation about Stanford, the words "school" and "California" may be strongly related to one another; if someone was talking about *Terminator*, then "movie" and "robot" might be related. This is semantic information that can only be decided by examining the language itself. Such knowledge could also conceivably be useful in spam filtering or other situations in which the computer needs a deeper understanding of important words and their relevance to one another.

The "school" and "movie" examples are both particular because they depend on the context of a conversation or some kind of discourse. For that reason, this project focuses on analyzing online conversations between two people (specifically, instant messenger logs). In such conversations, the topic can vary widely and change quickly, typos are widespread, and the participants make little effort to use grammatical English. These challenges make the problem different from previous related work (detailed in the next section).

This writeup details three main iterations of a system designed to discover important words in an instant messaging log and then decide which ones are related to one another. A baseline topic relation model is tested using a simple proximity heuristic. An improved model is presented that uses a maximum entropy classifier to perform NER in order to remove non-content words from the data. Finally, a third model (building on the second one) uses simple lexical chains and the *LCseg* algorithm [1] to segment the data into sections containing related topics.

Related Work

There is significant research on the problem of segmenting a document or a discourse based on topic changes. Hearst [2] presents the textTiling method for segmenting a text using lexical cohesion. Morris and Hirst [4] use a different measure of lexical cohesion, called lexical chains, to identify structure in a document, and Galley et al. [1] use a simpler version of lexical chains to segment meeting transcriptions called *LCseg* which is implemented in my project. They also examine some linguistic features such as pauses and overlaps which have likewise been examined in my project.

Additionally, several papers have been written about sentiment extraction from small pieces of text. A broad survey of sentiment analysis is presented by Pang and Lee [5]. Most systems written for sentiment analysis focus on things like reviews and comments, such as work in semantic orientation by Turney [6].

Less work has been done that attempts to extract data specifically from instant messaging conversations. Kucukyilmaz et al. [3] present one method of mining chat logs using term-based classification to determine the gender of the author. This work makes strong use of the presence of emoticons in the text.

Data and Infrastructure

My data set consists of several of my own instant messaging logs. Each of them are conversations between two people and consist of 250-2,700 timestamped lines of text. The specific data files are detailed in the table below. Additionally, about 3,200 words from one of the data files were hand-annotated with entity labels in order to train a maximum entropy classifier for NER. This data is available in the file medium_labeled.txt.

The system is written in Java and uses some of the CS224N utilities provided for the homework assignments. Also, most of the maximum entropy classifier code was ripped out of my Assignment 3 submission, although it has been significantly extended (details later in the writeup). The test harness also works in the same way as the ones from the assignments:

```
java cs224n.assignments.TopicExtractionTester –class someClass –dataFile
someFile
```

Where the default arguments are `cs224n.topic.BaselineTopicExtract` and `[project java directory]/data/small.txt`.

Data File	Size (lines of conversation)
small.txt	268
medium.txt	557
large01.txt	1040
large02.txt	1048
larger.txt	2715

Evaluating the Models

This problem addresses a very open-ended question: given some topic, which topics in a conversation are most strongly related to it? “Most strongly related” is a subjective phrase. For this reason, I decided to avoid creating a “gold” solution for the data against which to evaluate. Instead, I used the intuition that we really care about whether humans think the results of these models make sense. Therefore, the evaluation metric is simple: For a given conversation, print out the ten most common topics found by the model, and for each one, print out the five most closely related topics. This is a total of 50 “related topics.” Have a human read them and pick out the related topics that “don’t belong” to their parent topic. If the parent topic itself doesn’t make

sense, then all five related topics are considered not to belong. Penalize the score for every one of these. Using this method, the best score possible is zero, and the worst score possible is -50.

I realize this method of evaluation may be controversial, but I believe it addresses the most important "metric" of any of the models, which is whether their results make sense to people. It may be less consistent than defining a gold standard over a data set, but I think it is a truer evaluation.

Here is a summary of the final results:

Model	**Average Score**
Baseline	-40.2
NER	-20.4
Segmented	-23.6

More detailed results are given in the descriptions of each particular model.

The Models

A topic relation model trains on a chat log consisting of a set of timestamped lines of text. For a given topic word from the chat log, it can return a distribution over other words in the log that the input word is related to or associated with.

1. Baseline Model

1.1 Description

The baseline model uses a proximity heuristic: the simple intuition that conversation topics are related to one another if they tend to pop up near each other in the conversation. It slides a window ("neighborhood") across the data and computes the affiliation between pairs of words simply based on how close they appear to one another. Specifically, for some word w1 on the fringe of the neighborhood and some other word w2 contained earlier in the neighborhood at index y in [0, neighborhoodSize), the association between w1 and w2 is increased by y/neighborhoodSize. Intuitively, if w2 is on one end of the neighborhood and w1 on the other, their association is increased by 1/neighborhoodSize; if they are right next to one another, their association is increased by exactly 1. After passing over the entire data set, mappings from a word to its associate words are normalized such that they become probability distributions.

The baseline model also makes a very simple observation that lines of text occurring far apart in time are unlikely to be related. Indeed, Galley et al. [1] argue that such features as long temporal gaps are usually good cues for topic change. Therefore, if the model observes a line that appeared several hours after the previous one, it clears its neighborhood. Other linguistic cues explored by Galley et al. were also considered, but either they did not apply (e.g. speaker change in a two-person conversation) or were not relevant to the medium (e.g. speaker overlap).

Finally, in order to avoid words that clearly contribute nothing to the topic of a conversation, which I will refer to as non-content words, the baseline model ignores words with length less

than 5 or greater than 12, and it has a small hard-coded list of very common words (such as "really," "could," and "would") that it ignores. This blacklist is kept rather short in order to avoid conforming to the particular writing style of my data.

1.2 Results

The results of the baseline model are mixed. The most obvious observation is that it is overrun with non-content words, even despite its primitive filtering and small blacklist. These problems become increasingly bad as the conversation data files get longer, since some uninteresting words just occur very commonly in chat conversations (such as "stuff", "weird", "though", "hahaha"). Blacklisting is an ineffective way of filtering these because they are dependent on the writing style of the data. Also, certain function words can take on different spellings—for example, it is common to omit apostrophes in contractions. As a result of all this, the model repeatedly finds the same common words that should not have been considered topics in the first place, and relates them all very closely to one another.

However, the results aren't as abysmal as these non-content words might have it. Clearly, in the limit, an infinitely long conversation would stupefy this model completely; however, with reasonably short conversations it does pick out a few good relationships. For example, it associated "music" with "listening" and "playing" in small.txt, "money" with "investment" in medium.txt, and "spain" with "madrid" in large01.txt. Here are the detailed results for the baseline model:

Data File	**Score**
small.txt	-41
medium.txt	-38
large01.txt	-40
large02.txt	-35
larger.txt	-47
Average	*-40.2*

2. NER Model

2.1 Description

The main motivation after seeing the performance of the baseline model was to find a better way of ignoring non-content words and picking interesting ones. A few strategies were considered initially. First, a part-of-speech tagger, or even a sentence parser, seems intuitive because most of the words we care about are nouns and adjectives, or possibly noun phrases or other higher-level groups of words. However, the problem is broader: we really care about several parts of speech, and we don't care to know exactly which part of speech they are. We only care whether they are a topic or not. Furthermore, because the manner of speech used in instant messaging conversations is so informal and irregular, "smarter" semantic algorithms don't perform very reliably on the data. Therefore, I framed it instead as a NER problem, where entities in the data

are interesting words that we shall analyze as topics, and non-entities are all other non-content words.

The NER model builds upon the baseline model. It uses the same proximity heuristic; the major difference is that it also uses a maximum entropy classifier to filter out non-content words. The classifier is trained on the 3,200 hand-labeled words in medium_labeled.txt. The words are separated into four categories of entity—PERSON, SETTING, ACTIVITY, and the catch-all INTEREST label, as well as O which indicates a non-content word. I separated the entities into different categories in order to provide richer features to the classifier, but the topic relation model does not distinguish between anything except whether the word is an entity or not.

Because chat logs consist of timestamped lines of text, the classifier is also extended to use timestamped data. The extractFeatures() method is now passed a TimestampedSentence datum, the previous word label, and the previous sentence timestamp. This means that a number of interesting features involving the timestamps were explored inside the classifier. In particular, the intuition from the baseline classifier that long gaps indicate a change in topic is now included as a TIME_PASSED feature inside the classifier. Additionally, I tried features such as IS_MORNING and IS_EVENING based on the time of day on the timestamp. This is based on the idea that, for example, someone might say "good night" at 2 in the morning, in which case we don't care about the topic "night," but they might say "San Francisco has good night life" at 6pm, in which case we might care about the word "night." Broadly, the time of day might change whether particular words become topics we care about. None of my data spanned enough days for this to make a difference in the accuracy of the classifier, but I am convinced that it would be worth exploring on a larger corpus. Similar features such as day of the week or even month of the year could be helpful.

I also included more standard features in the classifier, such as word, previous tag, previous/next/next-next words, ends with "ing," contains "thing," and contains apostrophe. Two particularly valuable features were those for short words and long words. Some work [6] has shown that emoticons can be a good feature for classification of chat logs, but the only one of my data files to significantly feature these is larger.txt; the rest have almost zero examples of them. In order to test the features, I split the labeled data file into 75%-training 25%-testing pieces. When the classifier is incorporated into the larger topic relation system, I use the entire labeled file for training.

It should be noted that the FB1 score of my maximum entropy classifier (on the 25% of the labeled data file) was not very high. After adding all these features I was able to increase it from 36 to 45. This is due in part to the fact that I did not implement very many features particular to the labeled training file, as writing styles differ greatly between different conversations and I wanted the classifier to remain robust. In general, chat logs are much more volatile than, for example, the repetitive biology texts we used in Assignment 3. However, the low FB1 score is not much of a problem. Since the classifier is essentially being used as a filter, we would rather err on the side of filtering more non-content out, as long as the words it picks as entities are reasonably good.

2.2 Results

This model shows a dramatic improvement over the baseline model, having scores in the negative teens and twenties instead of forties and fifties. The biggest difference is that the classifier achieves a remarkable amount of success weeding out irrelevant words, especially those like “things” and “stuff.” Some large penalties accumulated by the baseline classifier were the result of picking nonsensical topics to begin with, like “there’s,” and this problem is mostly gone in the NER model.

There are still some problems. A word like “think” ends up being very popular because it happens that these conversations contain a lot of phrases like “I think.” Unfortunately, that means think gets associated with many other unrelated topics. It could be justified as a legitimate topic because it is an interesting verb, but in many of these cases it is incorrect. Similarly, phrases like “sounds hilarious” or “sounds awkward” were common enough in large02.txt that “sounds” became a much more frequent topic than it needed to, even though it seems perfectly fine for the classifier to treat “sounds” as an entity. Also, there are some occasions where the NER filter is too strong, and we lose a few rare but valuable words that the classifier mistook for non-content words.

Some examples of particularly successful associations produced by this model: “halloween” with “costume”, “party”, “photos” and “friend”; “microsoft” with “google”, “search”, “technology” and “internet”; “playing” with “listening” and “guitar.”

Some examples of bad or ambiguous associations produced by this model: choosing “always” as a topic in small.txt; relating “living” to “chameleon” and “spanish”; relating “working” to “different” and “relationship”.

Data File	**Score**
small.txt	-20
medium.txt	-15
large01.txt	-26
large02.txt	-17
larger.txt	-24
Average	*-20.4*

3. Segmented Model

3.1 Description

The final model attempts to address the issue that the baseline’s proximity heuristic, while not bad, could be improved. In particular, although related words definitely appear near one another in a conversation, their relation is not just a function of *how near* they are within the conversation. Relevant words might appear relatively far from one another—although within the same discussion—and we don’t necessarily want to discount them solely because the speakers digressed for a couple sentences in between.

The solution is to attempt to segment the conversation—not just temporally, as the other models do, but by topic, using existing research on discourse segmentation. In particular, the segmented model uses simple lexical chains and an implementation of the *LCseg* algorithm [1], described below, to find likely topic boundaries in the data. Within a particular segment of the conversation, it relates all content words to each other simply with a score of 1. If a pair of words appears together in many segments, their relation can be strengthened many times, but the actual distance apart within the segment is no longer taken into account at all. The results are normalized to create probability distributions. This model still uses the NER classifier to eliminate non-content words.

Within the *LCseg* algorithm, a lexical chain is a simple record of all repetitions of a particular word in a text, as well as information (in this case, line numbers) about where in the text this word is repeated. A lexical chain is scored according to its structural importance to the text, which is determined by the authors to depend on how frequently the word is repeated and how densely the word occurs (or, put another way, the compactness of the chain with respect to the size of the whole text). Since this score is an indication of the strength of a lexical chain, I experimented with using these scores to associate words; specifically, a pair of words within the same segment would get an association score of the product of their two respective lexical chain scores within that segment. However, since the segments were already computed using these scores, this strategy only ended up promoting the same strong words repeatedly and discarding some of the less frequent (but still correct) words. For example, in small.txt, this caused the word "music" to be strongly associated with almost all other words. I abandoned this scoring scheme in favor of the simpler strategy detailed before.

The *LCseg* algorithm works as follows:

1. Identify the lexical chains in the conversation. If a word is not seen for *h* sentences, it is considered to have taken a "hiatus" and we split the lexical chain in order to avoid generating weakly linked chains.
2. Score each lexical chain R_i according to its frequency and compactness:

   $$score(R_i) = freq(t_i) \cdot log(\frac{L}{L_i})$$

   Where t_i is the word this lexical chain represents, L is the length (in lines) of the whole conversation, and L_i is the span (in lines) of this particular chain.
3. Slide two adjacent windows A and B across the conversation, and at each position, compute the similarity between them. This is given by

   $$cosine(A, B) = \frac{\sum_i w_{i,A} \cdot w_{i,B}}{\sqrt{\sum_i w_{i,A}^2 \sum_i w_{i,B}^2}}$$

   where

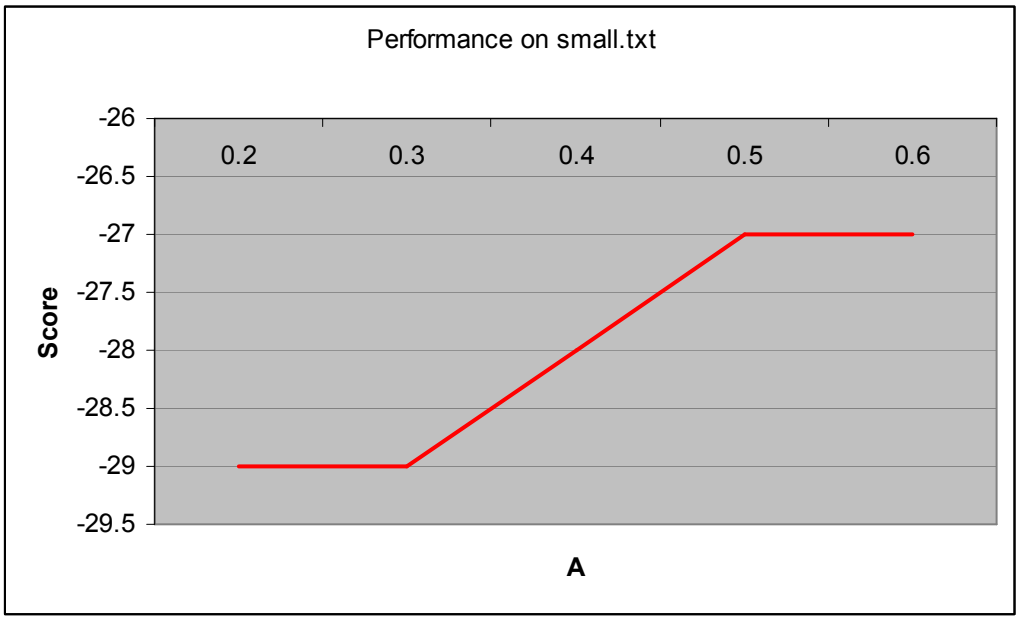
   $$w_{i,\Gamma} = \begin{cases} score(R_i) & \text{if } R_i \text{ overlaps } \Gamma \in \{A, B\} \\ 0 & \text{otherwise} \end{cases}$$
4. Smooth these similarity scores using a moving average filter to obtain the lexical cohesion function over the whole conversation.

5. Each local minimum in the cohesion function is a candidate for a segment boundary. For each local minimum m, identify the adjacent local maxima l and r. The hypothesized segmentation probability at this boundary is

$$P(m_i) = \frac{1}{2}[cohesion(l) + cohesion(r) - 2 \cdot cohesion(m)]$$

   This probability represents the sharpness of the minimum. A sharper minimum will have a higher probability of being a segment boundary.
6. Discard all candidate boundaries with probabilities lower than a constant p_{limit}. With the remaining candidates, compute their mean μ and standard deviation σ in order to obtain the threshold $(\mu - A\sigma)$.
7. Return the candidate boundaries with probability above this threshold: they are the chosen segment boundaries of the conversation.

The *LCseg* algorithm includes four parameters: p_{limit}, A, *h*, and the width of the sliding windows. All of them were initially set to the values given in the original paper—0.1 for p_{limit}, 0.5 for A, 11 for *h*, and 2 for the window width—and then varied to see if results improved. The sliding window width, A, and p_{limit} proved to have little effect when varied slightly and hurt results when varied greatly. Indeed, the following graph shows the experimental results that support the original value of 0.5 for A:



The authors of *LCseg* suggest a value of 11 for *h*—that is, if a word does not appear for 11 consecutive sentences in the discourse, we should separate its lexical chains at this gap. This is the hiatus parameter. Interestingly, increasing this parameter improved the segmented topic relation model. For example, increasing *h* to 13 caused the performance on small.txt to improve from -24 to -21; a wider range of topics made it into the set of suggestions, and more of these were correct than the older, more redundant set. This can be attributed to the fact that much of the chat dialogue is not substantive; there are lots of lines like “haha” and “yup,” whereas

sentences in a meeting transcript could be expected to contain more content. Rarely does an instant messaging line contain multiple sentences, but frequently it contains a piece of a sentence. Therefore it makes sense to allow a larger gap between recurrences of the same word in the same lexical chain.

3.2 Results

This model solves some problems of the previous models, but also introduces a new (rather significant) problem. This problem is that when the NER component selects a non-content word by mistake as an entity, this mistake is amplified when that word is turned into a full-blown lexical chain, whereas in the older model such a mistake may be submerged amid later correct decisions. For example, the results on larger.txt suffered greatly when words like "gotcha" and "perhaps" played too large a role in the associations.

At the same time, since the proximity heuristic was eliminated in favor of topic boundaries, certain words emerged, having not been included in results from older models, that were mentioned more rarely but had important roles in the topic relations. One example is "iphone" with "store", "program" and "apple" in medium.txt (where the NER classifier associated it more with "medical" and "marijuana"). Another example is the word "bainbridge" in large01.txt. Bainbridge Island is the name of the suburb where I went to high school, and the subject of the conversation at one point. The NER model associates "bainbridge" correctly with "school" and "working," but the reverse is weaker, as it associates "school" most strongly with "chameleon." The segmented model associates "bainbridge" with "school", "working", and "island", and it associates "school" with "bainbridge", "friends" and "apartment".

The final scores of the segmented model reflect these changes. On average it is slightly worse than the NER model, although it does better on large01.txt. Since the method of evaluation is subjective, the performance difference between the Segmented and NER models is probably not statistically significant.

Data File	**Score**
small.txt	-21
medium.txt	-19
large01.txt	-21
large02.txt	-28
larger.txt	-29
Average	*-23.6*

Conclusions and Future Work

I have presented a method for extracting useful topics from instant messaging data and relating these topics to one another. The second and third models I implemented both achieve average scores around -20; this means that for topics that the models extract most often, they suggest good related topics about 60% of the time. While probably not usable at this point, this is a reasonable start and there are clear directions for improvement.

The biggest obstacle in solving the problem well is the extraction step, and this could be improved greatly by having a better-trained NER component. Specifically, the Segmented Model's main shortcoming was that mistakes by the maximum entropy classifier did a disproportionate amount of damage to the results, and so lowering the likelihood of such mistakes would improve this model. Even with the current problem, it performs just about as well as the simple NER model. The maximum entropy classifier currently trains on only 3,200 words; having something more like 32,000 words here would do a lot of good.

Also, this model uses the simplest form of lexical chains, looking only at word repetition. Using a thesaurus to chain words with similar meaning, as explained by Morris and Hirst [4], might provide deeper insight into the topical structure of the data.

Finally, given a much larger corpus spanning a greater time interval, it would be interesting to investigate in more detail the time-related features I touched on while implementing the maximum entropy classifier. Since timestamps are unique to this kind of data, it would be worthwhile to see in more detail how useful they are.

References

[1] Galley, M., McKeown, K., Fosler-Lussier, E., and Jing, H. 2003. Discourse segmentation of multi-party conversation. In *Proceedings of the 41st Annual Meeting on Association For Computational Linguistics - Volume 1* (Sapporo, Japan, July 07 - 12, 2003). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 562-569.

[2] Hearst, Marti A. 1993. TextTiling: A quantitative approach to discourse segmentation. Technical Report Sequoia 93/24, Computer Science Department, University of California, Berkeley.

[3] Kucukyilmaz et al., 2006 Kucukyilmaz, T., Cambazoglu, B. B., Aykanat, C., & Can, F. (2006). Chat mining for gender prediction. In Proceedings of the fourth Biennial conference on advances in information sciences (pp. 274–284). Izmir, Turkey.

[4] Morris, J. and Hirst, G. 1991. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Comput. Linguist.* 17, 1 (Mar. 1991), 21-48.

[5] Pang, B., Lee, L., and Vaithyanathan, S. 2002. Thumbs up?: sentiment classification using machine learning techniques. In Proceedings of the Acl-02 Conference on Empirical Methods in Natural Language Processing - Volume 10 Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 79-86.

[6] Turney, P. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), 2002.