

# Effective Information Access Over Public Email archives

William Lee  
wwlee1@uiuc.edu

Hui Fang  
hfang@cs.uiuc.edu

Yifan Li  
yifanli@uiuc.edu

5/8/2005

## 1 Introduction

Email is one of the major means of communication in education, business and other areas. Email data can be classified into two types: personal email and public email. Public email archive is a collection of emails from the persons sharing some common interests. Both newsgroup and aggregate emails of a particular technique support group are good examples for such email archives. With the huge amount of public email data (e.g. mailing list, newsgroup etc.) available on-line, the email users are also suffering information overload as the web users. The key challenge is how to provide a better system so that the user can access such email data more effectively and more efficiently. Current systems usually provide two ways for user to access to the public email archive: browse and search. However, none of them can solve the information overload problem satisfactorily.

In order to solve this problem, we propose to use clustering method to group the similar conversations together and use summarization method to provide the gist of the conversation to the users. After clustering, the user only needs to go through the messages by topics and can identify some interesting topics for further exploration. After summarization, the users do not need to read through the whole thread to decide whether they are interested. Instead, the users only need to read the summary, which is usually much shorter than the original thread.

Both clustering and summarization have been extensively studied for traditional text collection. However, the following three differences indicate that the traditional clustering and text summarization techniques alone will not be enough. First, the assumption of document independence does not hold for email archives. One message in the email collection may be the reply to another message. The text in one message alone can not represent the content of that message very well. So the retrieval unit in email collection is not one email message anymore. Instead, thread is a much more appropriate choice. Each thread is treated as a single unit with special tree-like document structural properties. Second, unlike the traditional text document, the email messages have structural information (such as the Subject, From, To, CC, and Date header), which would provide some useful information as well. Finally, the email messages are usually short and informal.

In our project, we treat each thread as a semantic unit for clustering and summarization. We study email thread clustering by using the structural information. Furthermore, the clustering results have been visualized, which provides a new way for users to navigate the public email archive. We also propose to use topic shifting detection algorithm and subject expansion technique to summarize different types of threads. Experiment results show that our methods indeed perform better than some reasonable baselines. Our study can be used to help technical newsgroups to automatically or semi-automatically compile a list of FAQs. It can also be used by technical support staff to search and mine for the relevant answers in the archives for the questions that they received. In addition, our work can also be used to give new mailing list subscriber a comprehensive overview of the group based on frequently discussed topics.

## 2 Related Works

Unlike some researchers who are focusing on the personal email organization ([5], [2], [1], [10]), we puts an emphasis on supporting effective and efficient ways to access the public email archive. The existing systems only support either browse or search over the public email archive, which usually does not allow

the user to access to the data in more effective way. In order to solve this problem, clustering seems to be a natural choice, since clustering would allow users to browse the archive by different topics. Unfortunately, to our best knowledge, none of the previous work has addressed the problem of email thread clustering. The other solution to information overload is summarization. There are a few previous work studying summarization on email data. Muresan [7] has proposed to learn rules for noun phrase extraction in order to summarize individual email messages, as opposed to our goal to summarize a thread of emails as a whole. Nenkova [8] constructs the summary by extracting one sentence from the root message and each response with ad-hoc algorithms. Newman [9] performs clustering analysis on the emails to form different groups of topics, from which they extract summaries. Rambow [11] exploits the dialogic structure to do the summarization. Shrestha and McKeown [13] employ different methods for identifying questions and answers. Email conversation specific features (e.g., the number of intermediate messages between questions and answers) are explored to raise the recall ratio significantly, as shown by the experiments. However, there are two major differences between our work and previous work. First, none of the previous work really looks into the different types of the email threads as what we did in our project. For example, question-answer pair does not play an important role in announcement-driven threads. Therefore, [13] would fail to provide appropriate summary for announcement-driven threads. Second, only detecting question is not enough. In newsgroup, there are a lot of questions that do not really contain meaningful information, for example, "Anybody have any ideas?" In such cases, it would be necessary to identify the related meaningful context for the question, which is one of the goals in our project.

### 3 Email Thread Clustering

#### 3.1 Representation of Threads

Instead of treating each individual email as a clustering unit, we want to group the threads of discussion together. Therefore, we first need a formal model to represent threads in a newsgroup. We define a newsgroup  $G$  as a set of message threads  $T_1, T_2, \dots, T_n$ . Each thread  $T_i$  represents a set of messages  $m_{i1}, m_{i2}, \dots, m_{i|T_i|}$ . We assume that one message can only belong to one thread. In other words,  $G = \bigcup_i T_i$  and  $T_i \cap T_j = \emptyset$  for any  $i$  and  $j$ . It should be noted that the messages  $m_{ij}$  are grouped together in a reply tree within  $T_i$ . We can simplify the retrieval model within a thread by treating each message within  $T_i$  as one single message. Thus, in the traditional bag-of-word model, we let  $T_i$  and  $m_{ij}$  be bag of words. We then set  $T_i = m_{i1} \cup \dots \cup m_{iq}$ .

#### 3.2 Basic Similarity Functions

The similarity between two discussions is measured by  $sim_x(T_i, T_j)$ , where  $x$  is considered a particular "perspective" when comparing the threads. Larger  $sim_x(T_i, T_j)$  indicates higher similarity. In particular, for any pair of threads  $T_i$  and  $T_j$ , we considers the following similarity functions:

1.  $sim_m(T_i, T_j)$  measures the similarity between the main contents of two threads. This includes the subject of the thread, quoted text, and unquoted text of all the messages in the thread. It does not include any header field other than the "Subject:" line.
2.  $sim_s(T_i, T_j)$  measures the similarity between the subject properties. The subject property, as described before, is just the subject of the first message of the thread.
3.  $sim_{mnq}(T_i, T_j)$  is similar to  $sim_m(T_i, T_j)$ , except that  $sim_{mnq}(T_i, T_j)$  excludes quoted text from the content. Quoted text is defined as lines that start with ">". This is done so that we can safely ignore much of the repeated content in the replies. The quoted paragraphs have usually appeared before in the thread, so logically they are not necessary.
4.  $sim_a(T_i, T_j)$  indicates the authorship similarity. As mentioned before, this computes the similarity between the Author properties. This can potentially be used to indicate similarity if the same authors are engaged in a similar conversation across different threads.
5.  $sim_f(T_i, T_j)$  is similar to  $sim_m(T_i, T_j)$ , except we use only the first message for comparison.

6.  $sim_r(T_i, T_j)$  is similar to  $sim_m(T_i, T_j)$ , except we use all messages other than the first one for comparison.
7.  $sim_{rnq}(T_i, T_j)$  is similar to  $sim_r(T_i, T_j)$ , except that  $sim_{rnq}(T_i, T_j)$  excludes quoted text.
8.  $sim_{date}(T_i, T_j)$  measures the similarity based on the Date property.
9.  $sim_{uniform}(T_i, T_j)$  is the baseline similarity function, it always returns 0.5. This is served as the weakest baseline in our experiment.

Each similarity function is then normalized to a value between 0 and 1, based on the maximum and minimum value of the distance function.

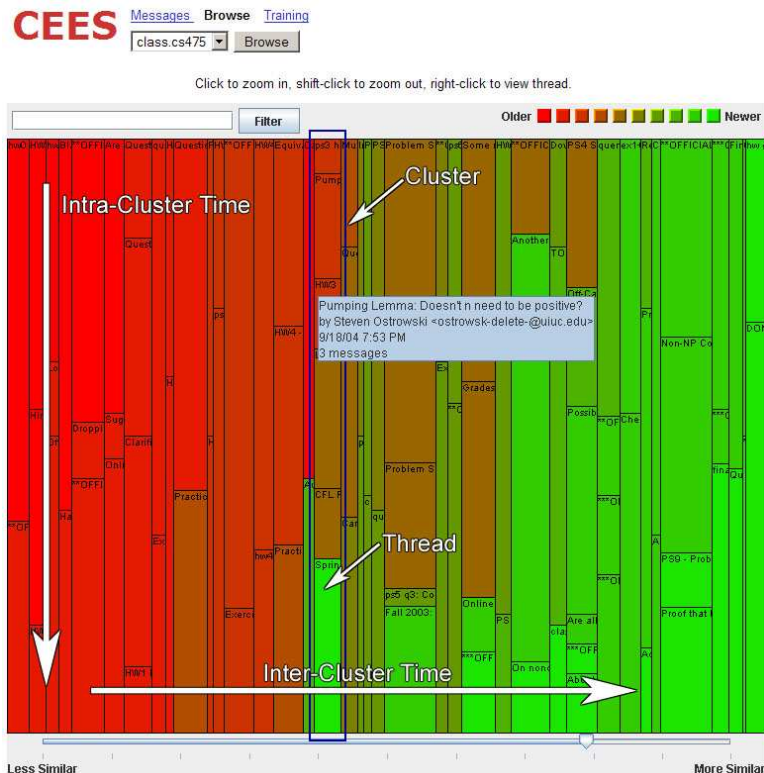


Figure 1: Conversation Map

### 3.3 Computing Similarity Functions

For the similarity functions except for  $sim_{date}(T_i, T_j)$  and  $sim_{uniform}(T_i, T_j)$ , we use a vector space model to compute the similarity between two fields. Let  $t'_i = \{v'_{i1}, \dots, v'_{in}\}$  where  $v'_{ij}$  represents the number of occurrence of a word in  $T_i$ . We want to transform this vector into a weighted term frequency vector, where each term is weighted according to its frequency and document frequency. We use a variation of the Okapi scoring formula [14, 12] to weight each term. Suppose our final Okapi-weighted vector is  $t_i = \{v_{i1}, \dots, v_{in}\}$  and we treat each  $v'_{ij}$  as term frequency (tf), we can transform  $t'_i$  into  $t_i$  by setting:

$$v_{ij} = \frac{k_1 \cdot v'_{ij}}{v'_{ij} + k_1((1-b) + b(\frac{dl}{avdl}))} \ln \frac{N - df + 0.5}{df + 0.5}$$

where  $k_1 = 1.2$ ,  $b = 0.7$ ;  $N$  is the total number of documents in the collection;  $df$  is the document frequency;  $dl$  is the document length;  $avdl$  is the average document length.

In order to compute the similarity between two threads, we calculate a simple dot product for the two vectors:

$$sim_x(T_i, T_j) = \sum_{k=1}^n v_{ik} \cdot v_{jk} \quad (1)$$

where  $x$  represents a particular perspective for comparison. For  $sim_{date}(T_i, T_j)$ , we would like to model the exponential time decay based on the age of the message. Assuming that we have the date for the thread as  $date(T_i)$ , the similarity function is  $sim_{date}(T_i, T_j) = e^{-|date(T_i) - date(T_j)|}$ . It should be noted that the date is measured by day, so two threads posted on the same day would have similarity of 1.

### 3.4 Learning to Combine Similarity Functions

Essentially, given two email threads  $T_1$  and  $T_2$ , we would like to find the function  $sim(T_1, T_2)$  that can best satisfy our clustering goal. It is worth noting that  $sim(T_1, T_2)$  may vary based on user preferences, since no two human beings share the exact same perspectives. In general, we want to maximize the intra-cluster similarity and minimize the inter-cluster similarity. Therefore, if two threads are in the same cluster in the training set, then we should set the ideal similarity to 1 (most similar). Conversely, if two threads are not in the same cluster, we should set the ideal similarity to 0 (not similar). For any pair of threads in the training clusters, we would create a positive example if the two threads are in the same cluster and construct a negative example if the two threads are from different clusters. Given those training examples, we can apply linear and logistic regression and construct a composite similarity function.

### 3.5 Clustering and Visualizing Clusters

We use the agglomerative clustering algorithm and the complete link cluster distance function in order to cluster the threads into a hierarchical tree. This tree is then display through a mechanism named Conversation Map (CM). This visualization mechanism is described in more details in [6]. In essence, it is a Treemap-base visualization techniques that involves sorting the time that the conversation took place in two dimensions. Figure 1 shows an example of a CM.

## 4 Email Thread Summarization

Another solution to solve information overload is to provide a summary of the email conversation (i.e. threads). Such summarization gives users a rough idea about what are being discussed, which allows users to exploit the information more easily and efficiently. It is especially important when large amount of email threads are present.

After studying the data, we observe that all the threads can be classified into two categories based on how the conversation is started. The first category is referred as **question-driven**. All the conversation (i.e.threads) in such category starts from a problem/question. The purpose of issuing such conversation is looking for a answer to the question or asking for help on a problem. The rest of the conversation (if any) usually provides the answer or help. The question usually plays an important role in such conversation. The second category is referred as **announcement-driven**. The purpose of issuing such conversation is to make an announcement. The rest of the conversation is usually about the clarification of the announcement. Unlike the messages of the first category, the subject usually plays an important role. Due to the different characteristic of these two categories, we propose to use different summarization methods for these two different types of threads.

### 4.1 Summarization Method for Question-Driven Threads

In the question-driven category, the conversation (i.e. thread) always starts from a question. The rest of conversation usually contains either the answer to the question or the clarification of the question. It is obvious that the question plays an important role in the conversation, therefore, the question naturally can be treated as a summarization of the conversation. Now the question boils down to how we can find the

- |   |  |
|---|--|
| <ol style="list-style-type: none"> <li>1. Subject: Compiler Error</li> <li>2. I've been debugging things using DDD with dbx, but I'm running into a weird problem.</li> <li>3. My PatriciaTree class is basically a wrapper around a root pointer, so I observe that pointer, and dereference it to give me my first PatriciaNode, and then look at that's inside that, and one of those things is a pointer to "data" within the Array object that's embedded in my PatriciaNode.</li> <li>4. An address shows up fine, but I'd like to dereference it to take a look at the actual array, so I can continue to examine the structure of my tree.</li> <li>5. But when I dereference it in DDD, it just shows up as "(nil)". I know for a fact that there's valid data in that array somewhere, because I can access it in my program, I just can't look at it in the debugger.</li> <li>6. Anybody have any ideas?</li> <li>7. It'd be really nice to look at that info for debugging purposes.</li> </ol> <p style="text-align: center;"><b>(a) Question -driven</b></p> | <ol style="list-style-type: none"> <li>1. SUBJECT: This week in discussion section</li> <li>2. Hello all</li> <li>3. Time: 7:00pm - 8:00pm</li> <li>4. Location: DCL 1111</li> <li>5. Topic: This week in the discussion section, I plan to cover as much of the following topics as time allows: - Generalizing Karatsuba's algorithm - Job scheduling with minimum total weighted waiting time - Clustering - Maximum bottleneck tree - Greedy coloring of graphs.</li> <li>6. These are gleaned from HW3 and HW4.</li> <li>7. Feel free to come with questions as well -- perhaps from old exams, for instance.</li> <li>8. With the midterm exam just under two weeks away, hopefully the discussion section can be helpful!</li> </ol> <p style="text-align: center;"><b>(b) Announcement -driven</b></p> |
|---|--|

Figure 2: Examples of Summarization

questions in the conversation. Unlike the traditional text collection, the sentences in the email conversation are rather informal. The question itself rarely contains meaningful information. One typical example is that the problem is first described in the narrative format, and then followed by a sentence with question mark — "Anybody have any ideas?" Therefore, the key challenge of this method is how to identify the question and the meaningful context for the question. In this project, we propose to use *topic shifting detection* algorithm to identify the question and the related meaningful context.

Let us first use a real example (as shown in Figure 2(a)) to illustrate the main idea of the topic shifting detection algorithm. (To save the space, only the first message of the thread is presented.) In this example, sentence 5 and 6 are selected as summary of the thread. Sentence 6 is a question, however sentence 5 contains more meaningful information. By further analyzing the message, we observe that there are several topic shifting in the message. The sender started with a general opening (sentence 2). Then he explained the background of the problem (sentence 3 and 4). After that, he stated the problem specifically (sentence 5) and raised his question (sentence 6). Finally, he explained why he is interested in this question (sentence 7). Note that subject of the thread (sentence 1) can not provide any useful information for the summarization.

Next, we describe our summarization method for question-driven threads.

1. For each message in the thread, we segment the message by using topic shifting algorithm
2. Identify the segments containing question
3. Remove the segments with redundant question
4. Return the remaining segments as the summary for the thread

In Step 1, our code is built based on TextTiling algorithm [4]. TextTiling is a method for partitioning text documents into coherent segments that correspond to a sequence of subtopics. The method is based on the assumption that a set of words is used for a given subtopic and the change of subtopics causes a significant proportion of the vocabulary changes. For email conversation, such assumption is also valid for each message. However, compared with the traditional text documents, the email messages are usually much shorter. So we make a few modifications of algorithm to make it also work well on the shorter documents. The modifications are mainly related to the stopping criterion, threshold setting and smoothing. Note that we use modified TextTiling to segment each message of the thread instead of segmenting the thread. In this way, we can make sure the sentences from different senders would not be falsely put into one segment. In Step 2, we currently only deal with the explicit questions (i.e. the sentences with the question mark). Actually, it is

not hard to find the implicit questions (e.g. the sentence starting with “I am wondering”) by using machine learning algorithm. However, due to the limited time, no training set was build for such purpose. We plan to explore it in the future. Step 3 is also a very interesting research question. In our project, we use a rather simple method — threadshold cutoff (i.e. The segments are sorted based on time. For each segment, we compute its similarity with the newer segments. If the similarity is larger than a learned threshold, the newer segment would be removed.) A more sophisticated method would be to first cluster the segments and return the centroid for each cluster. Such method would capture some global information, and the performance is expected to be better than our current method. Again, we plan to explore it in the future work.

## 4.2 Summarization Method for Announcement-Driven Threads

In contrast to the question-driven email threads, we notice that another class of threads is basically announcements, where little questions are proposed or important (e.g., some questions are simply for clarifications). The main objective of such email thread is to broadcast information about some events, where subject usually provide an important clue about the summary. An example is shown in Figure 2(b). In the example, sentence 3, 4, and 5 are selected as the summary of the thread. We can see that the subject plays an important role in selecting the relevant sentences. One straightforward way is to find the most similar sentences with respect to the subject line as the summary. However, this usually is not sufficient (e.g., sentence 3 and 4 would probably fail to be chosen). Our observation here is that there is usually a set of *important words* associated with a given topic of threads, other than the words in the subject lines. For instance, a topic like “discussion section” may find words like “time”, “location”, “topic”, ... important to decide selecting sentences as the summary. We call such words *pivot words*. Thus, if those pivot words are available, we could extend the subject title by adding the pivot words before comparing it with the sentences to obtain the summary. For example, with stop-words excluded, the bag of words of the extended subject would be { **this**, **week**, **discussion**, **section**, **time**, **location**, **topic** }.

Motivated by the observation, we propose to use *subject expansion* approach. Before we present our summarization algorithm, we first describe the notations that will be referred to later. For an email thread  $T$ , we denote its subject by  $T_{subject}$ , which is a bag of words. The summary of the thread  $T_{summary}$  is a set of sentences. A cluster  $C$  of threads is a set of threads, and the subject of a cluster  $C_{subject}$  is defined to be  $\bigcup_{T_i \in C} T_{subject}$ . In similar, the summary of a cluster  $C_{summary}$  is  $\bigcup_{T_i \in C} T_{summary}$ . We define a boolean value  $occur(w, T) = (w \in T_{summary})?1 : 0$ , which indicate whether word  $w$  appears in  $T$ 's summary. The underlying rationale here is that we care much more about whether a word shows up in the summary or not, than how many times it shows up, since we think the former can reflect the importance of the word more accurately. Furthermore, we represent the frequency of a word  $w$  in a cluster  $C$  by  $freq(w, C) = \sum_{T_i \in C} occur(w, T_i)$ .

**Definition 1** Given a cluster  $C$  of threads, its summary  $C_{summary}$ , and  $0 < \theta < 1$ , we define the pivot words associated with  $C$  as  $\{w | w \in C_{summary} \wedge \frac{freq(w, C)}{|C_{summary}|} > \theta\}$ , denoted by  $C_{pw}$ .

In fact, given a certain topic, the pivot words are the ones frequently occurring in the summary.

In the training phase, we perform the followings two steps:

1. Clustering threads by subjects.
2. For each cluster  $C$ , find the  $C_{pw}$ .

In this way, we are able to learn the pivot words for different topics (clusters) from the training dataset (with summaries identified).

In the testing phase, given a thread  $T$ , we will

1. Find the cluster  $C'$  s.t.  $\forall C, sim(T_{subject}, C_{subject}) \leq sim(T_{subject}, C'_{subject})$  and  $sim(T_{subject}, C'_{subject}) > \tau$ , where  $\tau$  is some predefined threshold.
2. If such  $C'$  exists, extend the subject of  $T$ :  $T_{subject} = T_{subject} \cup C'_{pw}$ .
3. Return the set of sentences  $\{s | s \in T \wedge sim(s, T_{subject}) > \eta\}$  as the summary, where  $\eta$  is some predefined threshold.

In step 1, we find the most similar cluster in terms of the subjects. In step 2, we take advantage of the learned pivot words of the cluster to extend the subject line of the thread of similar topic. In Step 3, we choose the most similar sentences as the summary of the thread. The similarity between sentences is computed by using the *Jaccard similarity measure*. Cosine measure has also been tested with TF weighting, which have not shown much difference. We believe that this is because the sentences have relatively much smaller sizes compared to documents.

## 5 Evaluation

### 5.1 Experiment Setup

#### 5.1.1 Data

We use three UIUC CS class newsgroups (CS225, CS473 and CS475) as our data collection. More specifically, the CS225 is a lower-division data structure course. CS473 and CS475 are two upper-division/graduate-level algorithm classes. Some topics, such as NP-complete problems, are taught in both CS473 and CS475. There should be no obvious overlapping topics between CS225 and the upper-division algorithm classes. In order to evaluate the performance of our method, we need the judgement files for both clustering task and summarization task. In our project, all the judgement files are manually created by ourselves.

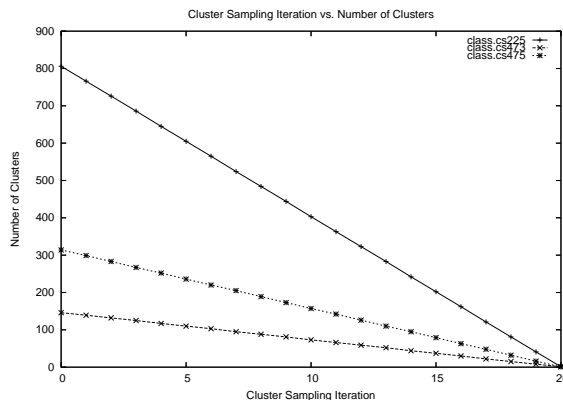


Figure 3: Cluster Sampling Iteration vs. Number of Clusters

#### 5.1.2 Evaluation Measures

In order to evaluate our clustering performance, we use overall entropy [3], which is a linear combination of two quantitative measures: Cluster Entropy and Class Entropy. Cluster Entropy measures the entropy (or the degree of mixture) of the resulting clusters from the clustering algorithm. Class Entropy, on the other hand, compares the actual clusters from a golden set and measures how many of the elements in the golden set cluster are assigned to different resulting clusters. But how to set the weights for such linear combination is still challenging. In our project, we decide to set the weights such that the two extreme clustering cases have the same penalty. In other words, if we have  $n$  samples in the set that we want to cluster, and our clustering results also have  $n$  results (all-in-all), we want the overall entropy measure to be the same as if we have 1 cluster for the  $n$  samples (all-in-one).

For our summarization tasks, both the returned results and the actual summaries are sets of sentences. So we use precision and recall to measure the performance.

Similarity Function	Combined Entropy			Combined Entropy Average		
	225	473	475	225 + 473	225 + 475	473 + 475
class.cs225 LinearReg	<i>1.589</i>	1.159	1.427	<i>1.374</i>	<i>1.508</i>	1.293
class.cs225 Logistic	<i>1.568</i>	1.163	<b>1.397</b>	<i>1.366</i>	<i>1.483</i>	<b>1.280</b>
class.cs473 LinearReg	1.634	<i>1.136</i>	1.477	<i>1.385</i>	1.556	<i>1.307</i>
class.cs473 Logistic	1.657	<i>1.154</i>	1.481	<i>1.406</i>	1.569	<i>1.318</i>
class.cs475 LinearReg	1.613	1.179	<i>1.415</i>	1.396	<i>1.514</i>	<i>1.297</i>
class.cs475 Logistic	<b>1.592</b>	1.195	<i>1.403</i>	<b>1.394</b>	<i>1.498</i>	<i>1.299</i>
authors	1.944	1.204	1.738	1.574	1.841	1.471
contents	1.662	1.141	1.486	1.402	1.574	1.314
contents no quote	1.666	<b>1.131</b>	1.462	1.399	1.564	1.297
date	1.855	1.400	1.677	1.628	1.766	1.539
first mesg	1.629	1.177	1.431	1.403	<b>1.530</b>	1.304
rest mesg	1.694	1.137	1.517	1.416	1.606	1.327
rest mesg no quote	1.785	1.188	1.533	1.487	1.659	1.361
subject	1.672	1.155	1.483	1.414	1.578	1.319
uniform	1.961	1.491	1.765	1.726	1.863	1.628

Table 1: Clustering Performance

## 5.2 Experiment Results

### 5.2.1 Clustering Results

Since setting different number of clusters yields different entropy scores, we want to compute the clustering performance based on the average of the entropy scores across different number of clusters. Since we use agglomerative clustering, we can evaluate based on any number of clusters from 1 to  $n$ , where  $n$  equals the number of threads in the group. For each group that we would like to cluster, we take 21 snapshots (0 to 20) during the agglomerative clustering process at even intervals. We then average the Cluster Entropy, Class Entropy, and Combined Entropy based on the 21 snapshots. Figure 3 shows how the Cluster Sampling Iteration affects the number of resulting clusters for all three groups. Essentially, the higher the Cluster Sampling Iteration, the lower the number of clusters. The number of cluster decreases linearly in proportion to the maximum number of the threads in the group.

Table 1 presents the clustering results for each of the similarity function specified in Section 3.2. In particular, “[newsgroup] LinearReg” or “[newsgroup] Logistic” represents the a combined similarity functions using all the other similarity functions except for “uniform” (authors, contents, content no quote, date, first mesg, rest mesg, rest mesg no quote, and subject). The [newsgroup] prefix implies that the similarity function has been learned from the [newsgroup]’s training set. For example, “class.cs225 Logistic” is a similarity function that uses Logistic Regression to learn from the actual class.cs225 clusters. The columns in the table describe the Combined Entropy and the Average Combined Entropy. The Average Combined Entropy is the average value for the two groups indicated in the column headings. This is similar to performing a 3-way cross validation for the CS225, CS473, and CS475 newsgroups.

One can see from Table 1 that using either Linear or Logistic Regression would exceed or match the best performance of any single similarity function. We can also see that Logistic and Linear Regression have very similar performance. The numbers in italic are generated when the group is testing and training on itself, either entirely or partially. The bold numbers are the best run excluding the numbers in italic. Unsurprisingly, due to overfitting, training and testing on the same group yields the best performance. There is no single similarity function that performs the best across all the groups.

Interestingly, even with three different newsgroups and three different taggers, Table 1 shows that training the similarity function on one group can be used to learn the parameters of the combined similarity function for another group. This shows that our learning algorithm can be effective in learning the parameters of the general similarity function. Furthermore, since the tagging behavior for a single group is consistent, one would imagine that the learning algorithm can perform even better when training on a subset of the

Method	Average Precision %	Average Recall %	Average F1 %
Topic Shifting Detection	64.88	81.82	72.33
Baseline (whole paragraph)	51.14	100	67.67

Table 2: Performance comparison of summarization method for question-driven threads

Method	Average Precision %	Average Recall %	Average F1 %
Subject Expansion with pivot words	73.2	62.1	67.8
Baseline (Subject Only)	60.0	55.3	57.6

Table 3: Performance comparison of summarization method for announcement-driven threads

newsgroup and apply the parameters on the similarity function to cluster the other messages in the same group.

### 5.2.2 Summarization Results

#### Question-Driven

In order to demonstrate the strength of our approach (i.e. topic shifting detection), we need to compare it with a baseline. One straightforward baseline is to return the whole paragraph containing the question. The performance comparison is shown in Table 2. As expected, by returning the whole paragraph, the recall is 1. However, it hurts the precision. Our approach would increase the precision with some sacrifices on the recall. Based on F1 value, our approach (i.e. topic shifting detection) performs better than the baseline.

#### Announcement-Driven

We compare our methods (i.e. subject expansion) with a baseline approach which is similar to ours except that no subject line expansion is performed. That is, we simply select the similar sentences with respect to the subject as the summary. We set the  $\theta = 0.03$ ,  $\tau = 0.1$ , and  $\eta = 0.1$ . The results can be summarized in Table 3. We observe that the baseline method can lead to somewhat reasonable result, since the subject of the thread gives a good clue for the summary. On the other hand, our approach with subject extension can significantly improve the performance by extracting the pivot words.

## 6 Conclusion and Future work

In the project, we study two new ways to help users to access the public email archive more effectively and efficiently. First, we use clustering method to group the similar conversations together. In this way, some redundant topics can be detected so that the users do not need to read the details of some threads if they already read the similar ones. Furthermore, thread clustering provides a new way for users to navigate the public email archive. The user only needs to go through the messages by clusters and can identify some interesting clusters for further exploration, which would make the exploration more focused and more efficient. We use the agglomerative clustering algorithm and the complete link cluster distance function. The similarity function between two threads is learned by linear/logistic regression. Second, we use summarization method to provide the gist of the conversation to the users. Such summarization gives users a rough idea about what are being discussed, which allows users to exploit the information more easily and efficiently. By studying the data, we observe that all the threads can be classified into two categories based on how the conversation is started. We propose to use different summarization method for different types of threads: (1)topic shifting detection algorithm is used for question-driven thread summarization; (2)and subject expansion technique is used for announcement-driven thread summarization. Experiment results show that both methods indeed achieve better performance than the baseline.

There are several possible directions for the future work. First, it would be interesting to study how to use language modeling approach for summarization. Currently we use quite different approaches for two types of threads. In fact, by using LM approach it would be possible to use a more unified method for both types. Second, as we discussed in the previous section, clustering question and the related context would be

another interesting research topic. It allows us to see the global picture of the questions and provide another way to detect the redundant questions. Third, after finding the question, it would be interesting to detect the corresponding answer as well. Finally, in our project, we treat each thread as a unit and only explore the syntactic relation (i.e. based on the reply to information) of the messages within the thread. However, building and exploring semantic relation of the messages within the thread would be more interesting and useful.

## References

- [1] William Cohen. Learning rules that classify e-mail. In *Advances in Inductive Logic Programming*. 1996.
- [2] Yanlei Diao, Hongjun Lu, , and Dekai Wu. A comparative study of classification based personal e-mail filtering. 2000.
- [3] Ji He, Ah-Hwee Tan, Chew-Lim Tan, and Sam-Yuan Sung. On quantitative evaluation of clustering systems. In Weili Wu and Hui Xiong, editors, *Information Retrieval and Clustering*. Kluwer Academic Publishers, 2002. In press.
- [4] Marti Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. 1197.
- [5] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. 2004.
- [6] William Lee. Intelligent access to public email conversations. Master’s thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 2005.
- [7] Smaranda Muresan, Evelyne Tzoukermann, , and Judith Klavans. Combining linguistic and machine learning techniques for email summarization. *Computational Natural Language Learning*, 2001.
- [8] Ani Nenkova and Amit Bagga. Facilitating email thread access by extractive summary generation. *Recent Advances in Natural Language Processing*, 2003.
- [9] Paula S. Newman and John C. Blitzer. Summarizing archived discussions: a beginning. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 273–276. ACM Press, 2003.
- [10] Terry Payne. *Learning Email Filtering Rules with Magi, A Mail Agent Interface*. PhD thesis, Dept. of Computer Science, University of Aberdeen, 1994.
- [11] Owen Rambow, Lokesh Shrestha, John Chen, , and Christy Lauridsen. Summarizaing email threads. 2004.
- [12] S E Robertson and S Walker. Okapi/keenbow at trec-8.
- [13] Lokesh Shrestha and Kathleen McKeown. Detection of question-answer pairs in email conversations. 2004.
- [14] Amit Singhal. Modern information retrieval: A brief overview. 2001.