Thread and Post Classification Over Technical User Forum Data

Li Wang
Supervisor: Timothy Baldwin, Su Nam Kim
`li@liwang.info`

ABSTRACT *This project focuses on improving information access over troubleshooting-oriented technical user forums by 1) thread-level classification; 2) post-level classification; and 3) thread discourse structure analysis. The tasks have been done so far include: the design of thread-level and post-level class sets, the creation of experiment data set, and building baselines for thread classification task and initial thread classification. This report mainly focuses on describing these tasks and the experiment results analysis.*

Keywords Web Documents, Thread Classification, Post Classification, Thread Discourse Analysis

1 INTRODUCTION

The advent of Web 2.0 triggers the booming of web authorship from individuals of all descriptions. Web user forums, which often contain a large amount of valuable user generated content on different topics, are increasingly raising the interests in knowledge acquisition from forum threads. One branch of these interests is enhancing support sharing. Support sharing is the ability for users to look over the logs of past support interactions to determine whether there is a documented, immediately-applicable solution to their current problem.

This project addresses the task of enhanced support sharing, in the form of text mining over troubleshooting-oriented and computer-related technical user forum data. It focuses on automated thread-level analysis of the problem types (i.e. thread-level classification) and the thread structure, and the post-level classification. A example thread for our task is showed in Figure 1.

HTML Input Code - CNET Coding & scripting Forums

Poster A Post 1	**HTML Input Code** . . . Please can someone tell me how to create an input box that asks the user to enter their ID, and then allows them to press go.It will then redirect to the page. . . For example, . . . Thank You
Poster B Post 2	**Re: html input code** Part 1: create a form with a text field. See . . . Part 2: give it a Javascript action . . .
Poster C Post 3	**asp.net c# video** Ive prepared for you video.link click http://www.ah. . .
Poster A Post 4	**Thank You!** Thanks a lot for that. . . I have Microsoft Visual Studio 6, what program should I do this in? Lastly, how do I actually include this in my site?Thanks again. . .
Poster D Post 5	**A little more help** . . . You would simply do it this way: . . . You could also just . . . An example of this is:. . .

Figure 1: An example thread from the real-world forum

The tasks which have been done in the research so far include: the design of thread-level and post-level class sets, the creation of experiment data set, and building baselines for thread classification task and initial thread classification. This report will focus on describing these tasks.

The remainder of this report is structured as follows. Firstly, background information of relevant classification research, data set will be provided. Then, our thread-level and post-level class sets will be explained. The data source and the annotation process will also be described in this section.Thirdly, the classification models and weighting schemes used in out experiments will be explained. In the sixth section, the experiments setup will be presented. The experiment results will be listed and analyzed from different aspects in the following section. At last, future work and conclusion will be provided.

2 BACKGROUND

The main tasks in our project are: thread classification, post classification, and thread discourse structure analysis.

With respect to the thread classification task (i.e. the solution type and the problem source of the thread), there is little work that is specifically targeted at thread-level classification. The most related work is the ILIAD (Improved Linux Information Access by Data Mining) research which has been done by [1] and the PTC (Problem Type Classification) research which has been done by [20]. Actually, our thread class sets are created based on theirs. Their research mainly focuses on estimating the utility of a given troubleshooting thread. However, they stop at the thread-level classification stage.

Another research line that relates to the thread classification is discussion summarization. For example, in [21], technical online IRC (Internet Relay Chat) discussions are summarized and segmented. Then the message segments are clustered to find the most relevant information to the user by using machine learning schemes. Email summarization research also has been done by [12], [13], and [16]. They primarily concentrate on summarizing and organizing email archives by extracting overview sentences as discussion issues to help the user find the most useful email thread. In contrast, our research not only focuses on the overview of a troubleshooting thread, but also on the discourse structure of the thread and the overview of each post within the thread. These approaches can help the users firstly find the most relevant thread and then identify the most informative posts in this thread.

Regarding the post classification and the thread discourse structure analysis, there are some related research has been done in recent years. One of the direction focuses on facilitating the discourse structure analysis by detecting and extracting the question and answer pairs from email conversations (e.g. [14]) or online forum threads (e.g. [5]). Furthermore, [6] carry out research on extracting both contexts of the questions and answers for the questions (they assume that the questions are already identified) from forum threads, thus exploring the question-context-answer structure of the threads. Adjacency pair analysis in tutorial dialogue has also been inspected by [2], they try to identify all the dependent adjacency pairs in tutorial dialogues through χ^2 analysis. Our research, on the other hand, tries to analyze a more complex structure of the a thread by identifying the types of all the posts within the thread and the links between all the posts.

Another research direction on post-level analysis aims at automatically scoring or assessing the online discussion posts in online discussion forums, based on their contributions (e.g. [17], [19], and [18]). [8] also provides a probabilistic graphical model to estimate the joint probability of answer posts for answer ranking. While most of these research concerns the quality of the content of the answer posts, and try to give a score to the answer post, our research mainly focuses on identifying the types of the posts and the relations between them.

Additionally, the most relevant data set related to our task is the Reuters-21578[1] data set. Reuters-21578 is a standard document categorization data set which has been carefully edited for research usage. Besides the tags for each document (relevant to our thread classification) in the collection, the data set also has document-internal tags to delimit elements (relevant to our post classification) within each document. However, the types of the documents (i.e. Reuters newswire) and the class set (i.e. multiclass labels for each document) are not suitable for our task and the documents do not have many internal structures compared to thread data.

3 DATA AND ANNOTATION

3.1 Class Definition

3.1.1 Thread Classes

Thread classes are used to tag troubleshooting threads according to their characteristics and functionalities. Each thread class represents the basic nature of a problem in a given troubleshooting-oriented thread.

Our thread class sets are made up of a basic class set and a miscellaneous class set. The basic class set includes two groups containing 10 basic classes. The first group, which is named *Solution Type Classes*, includes 4 basic classes: *Documentation*, *Install*, *Search*, and *Support*. The second group, which is named *Problem Source Classes*, is made up of 6 basic classes: *Operating System*, *Hardware*, *Software*, *Media*, *Network*, and *Programming*. The miscellaneous class set, on the other hand, only contains two classes: *Other* and *Spam*.

With regard to the basic class set, because the *Solution Type Classes* group and the *Problem Source Classes* group describe two distinct features of a candidate thread (i.e. a troubleshooting thread that is not *Other* nor *Spam*) and a candidate thread always has both features, when building the actual class sets there are two choices: using only one group or using both groups. Respecting the miscellaneous class set, the two classes in it are used to tag threads that cannot be tagged with any classes from the basic class set. The detailed descriptions for the basic class set, the miscellaneous class set, and the actual class sets are presented in the following subsections.

Basic Class set (BASIC)

1. Solution Type Group (TYPE)
   This group is made up of 4 basic classes, and focuses on the actual problems and solution types of the

[1]http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

threads. A troubleshooting thread usually has at least one topic addressing a particular real-world problem. We only considered computer-related topics and designed 4 basic classes for all the potential solution types presented in the threads. We classified each thread according to the solution type presented in the thread, rather than the initial problem type given by the initiator of the thread. This is because, the initiator usually does not know the type of their problem (that's why they need troubleshooting), and the answerers may know. The detailed descriptions for these classes are listed below.

Documentation: This class targets the tasks of 'how to use certain functions of a computer related component', 'how to choose a certain computer related component within given candidates', or 'how to do something by using a computer related component'. These components include hardware-like parts associated with computers (e.g. sound card, router, and printer) and software-like parts related to computers (e.g. Window XP, JAVA programming language, and Microsoft Word). The tasks relating to 'installing a component' or 'fixing a problem regarding a component' are not included in *Documentation* class.

Install: This class aims at the tasks relating to 'installing a certain computer-related component'. Issues regarding the steps of installing a component and the problems with installing a component are included in this class.

Search: This class focuses on the tasks of 'where to find a certain computer-related component'. Issues in respect of searching for a component regarding a job or problem are included in this class.

Support: This class concentrates on the tasks which relate to 'fixing a problem regarding a certain computer-related component'. Issues with regard to solving problems of a component which arise after successful install and use are included in this class.

2. Problem Source Group (SOURCE)
This group includes 6 basic classes, and concentrates on the targets and the source of the problems presented in the threads. In a troubleshooting-oriented and computer-related thread, the problem described in the thread usually targets a particular computer-related component (i.e. the source of the problem). We designed 6 basic classes to describe all the potential computer-related components. The descriptions for these basic classes are presented below.

Hardware (HW): This class aims at the threads which target computer hardware related issues. In our definition, hardware refers to the core physical components of a personal computer. It includes all build-in components inside a computer (e.g. build-in wireless card, build-in Bluetooth, and hard disk) and standard external components outside a computer (e.g. different kinds of keyboards and mouses). Other external physical components (e.g. external hard disk and external wireless card) are not included in the *HW* class.

Operating System (OS): This class focuses on the threads which target operating system related issues. In our definition, operating system is an interface between hardware and users. It incudes the common operating systems (e.g. Microsoft Windows, Unix, and Mac OS X) used on a standard personal computer, the operating systems used on a mobile phone (e.g. Windows Mobile), the more basic firmware (e.g. BIOS) for a hardware and the system software (e.g. shutdown program, partitioning tools, and formatting tools)

Software (SW): This class concentrates on the threads which target software related issues. In our definition, software includes application software (e.g. word processor, media player, and web browser) and programming tools (e.g. Adobe Dreamweaver, MySQL, and Microsoft .NET Framework).

Media: This class aims at the threads which target media related issues. In our definition, media refers to computer related hardware which is either a non-standard external physical component of a personal computer (e.g. external hard disk and external wireless card) or a peripheral device (e.g. printer and USB flash drive).

Network: This class focuses on the threads which target network related issues. In our definition, network concerns with internet and intranet issues (e.g. connecting speed, network setting, and building physical network).

Programming: This class concentrates on the threads which target programming related issues. In our definition, programming only concerns with the actual coding, design, and learning problems. The problems of using certain programming software are not included in this class.

Miscellaneous Class set (MISC)

Other: When the topic of a thread is troubleshooting-related, but the target of the topic is not included in the SOURCE group, this thread will be labeled as *Other*. The problems described in the thread can be computer-related (e.g. problems with email account) or not strictly related to computers (i.e. searching for a service company).

Spam: Any thread which is not troubleshooting-oriented (e.g. discussion, announcement, and information) nor computer-related (e.g. searching for Internet Service Provider) will be labeled as *Spam*.

Class Sets Description

As is described in the previous section, *Solution Type* group and *Problem Source* group are two distinct basic class groups that can be used either separately or together. Therefore, there are three options to build the actual class sets. Combining these three options with the miscellaneous class set, three practical thread class sets are derived. The three actual thread class sets are presented in Table 1 and Table 2.

	TYPE Class Set (6 classes)	**SOURCE Class Set** (8 classes)
BASIC Classes	Documentation	HW
	Install	OS
	Search	SW
	Support	Media
		Network
		Programming
MISC Classes	Other	Other
	Spam	Spam

Table 1: Class sets which only concern one basic class group at a time

	Combined Class Set (26 classes)	
Combined BASIC Classes	Documentation-HW	Install-HW
	Search-HW	Support-HW
	Documentation-OS	Install-OS
	Search-OS	Support-OS
	Documentation-SW	Install-SW
	Search-SW	Support-SW
	Documentation-Media	Install-Media
	Search-Media	Support-Media
	Documentation-Network	Install-Network
	Search-Network	Support-Network
	Documentation-Programming	Install-Programming
	Search-Programming	Support-Programming
MISC Classes	Other	Spam

Table 2: Class set which concerns two basic class groups at a time

Take the thread in Figure 1 for example. This thread talks about how to do something by using a programming language (i.e. "how to create an input box…"), and the solution also follows this topic. Therefore, with respect to the TYPE group, it belongs to the *Documentation* class. Moreover, the problem source comes from a programming language (i.e. HTML programming). Therefore, regarding the SOURCE group, this thread belongs to the *Programming* class. The combined class label for this thread will be *Documentation-Programming*.

Another example thread is given in Figure 2. In this example, the topic is about whether a company deserves trust or not. This is a troubleshooting-oriented thread, however, the target (i.e. host company) does not belong to any of the classes in the SOURCE group. Therefore, it will be labeled with *Other* class.

3.1.2 Post Classes and link Information

Post classes are used to tag the posts within each thread. The class set is developed based on the analysis of the potential roles and different characteristics of the posts. Because a post may have multiple roles or topics in a thread, multiple classes are permitted for a post.

Can I Trust this host - CNET Web design & hosting Forums

Poster A Post 1	**Can I Trust this host** I have been looking into using http://host. . . they look like a new company but i dont know if I should take the chance.They have 14 day money back. help please
Poster B Post 2	**Feedback** Since they have a money back period you could just them out. They were founded in 2008. . . Here are some recent discussions on hostgator here on . . .
Poster C Post 3	**hi, i think hostgator has 30 day money back guarantee.** hi, i think hostgator has 30 day money back guarantee.
Poster B Post 4	**Money back guarantee** Yes they actually have a 45 day money back guarantee http://cas. . .

. . . .

Figure 2: An *Other* example thread from the real-world forum

When assigning a class to a post, this class is also combined with its link information. Except for the first post in a thread, the following posts often contain link information which refers back to one or more previous posts. These links reflect the discourse structure of a thread. We only assign one link information to one class label.

The post classes combined with link information can be used to analyze the discourse structure of a thread and furthermore contribute to the thread classification tasks.

12 classes are included in the post class set. These classes are furthermore categorized into 2 groups and 3 single classes. The detailed descriptions for this class set are presented in the following subsections.

Class Set Description

1. Answer Group
   This group consists of 5 classes, and targets answer-related posts.

   Answer-answer This class aims at the post that contains direct (e.g. giving explanation as an answer, making assumption and answer) or indirect (e.g. referring other website which has the answer) answers to the questions in a previous post. For example, in Figure 1's thread example, *Post 2* and *Post 3* are labeled with *Answer-answer*.

   Answer-add This class focuses on the post that includes additional direct or indirect answers which extend the answers in a previous post. For example, in Figure 1's thread example, *Post 5* is labeled with *Answer-add* because it adds additional information to the previous answers.

   Answer-correction This class aims at the post that contains direct or indirect information which fixes the faults of the answers in a previous post.

   Answer-confirmation This class targets the post which points out errors in the answers from a previous post without correcting them, or confirms the answers in a previous post to make it clear. In addition, the post that describes the unclearance of the previous answers (e.g. 'I don't understand the answer') will also be labeled with this class.

   Answer-objection This class aims at the post that is against the answers in a previous post. If the objection post only arises after trying the answers in the previous post (e.g. 'I tried it, but it didn't work'), this objection post will not be labeled as *Answer-objection*.

2. Question Group
   This group includes 4 classes, and targets question-related posts.

   Question-question This class aims at the post that brings a totally new question into the relevant thread. The first post of a troubleshooting-related thread is often labeled with this class.

   Question-add This class focuses on the post that contains additional information relating to the questions in a previous post, or brings additional questions that extend the questions in a previous post. For example, in Figure 1's thread example, *Post 4* will be labeled with *Question-add* because it adds a new question which relates to the initial question.

   Question-correction This class targets the post that corrects the errors of the questions in a previous post.

Question-confirmation This class aims at the post which points out errors in the questions from a previous post without correcting them, or confirms the questions in a previous post to make it clear. In addition, the post that describes the unclearance of the previous questions (e.g. 'I don't understand the question') will also be labeled with this class.

3. Resolution
   This is a single class which targets a certain post which is posted by the initiator (i.e. the person posts the first *Question-question*, which is often in the first post in a thread). If an initiator's post confirms an answer from a previous post is working, or objects an answer from a previous post after trying the answer, this post from the initiator will be labeled as *Resolution*. For example, in Figure 1's thread example, *Post 4* will be labeled with *Resolution* because it is posted by the initiator and the initiator confirms an answer from a previous post.

4. Reproduction
   This is a single class which aims at certain posts which are posted by the non-initiators. If a non-initiator's post repeats the same question (e.g. 'I've got the same problem, please help') as the initiator's original question (i.e. the initiator's *Question-question*), this post from the non-initiator will be labeled as *Reproduction*. Moreover, if a non-initiator's post confirms an answer from a previous post is working or not working after trying, this post from the non-initiator will also be labeled as *Reproduction*.

5. Other
   This is a single class which is used to label the posts that do not belong to any of the above post classes.

3.2 Data Description

There numerous forums, newsgroups and mailing lists available on the internet that cover computer related problems, from general discussions to specific troubleshootings. Three well-known, active forums were considered (i.e. CNET FORUMS[2], CHECKDNS.NET FORUM[3] and TECH SUPPORT FORUMS[4]). Because one of the tasks is post link analysis, and CNET forums can provide link information that can be used as a guideline, the CNET forums was used.

1000 threads were randomly crawled from CNET forums. We only collected threads that contain more than 1 post and less than 16 posts. This is because, on the one hand thread containing only one post has no answers and cannot provide solutions, on the other hand long thread tends to be more like discussion-oriented and/or contains too much information which may affect the processing accuracy. Then, the threads were preprocessed. Only the title and sub-forum information of each thread; the text, title, and author of each post were preserved. Then, all the threads were stored in a MySQL relational database. A lemmatized text version with POS (part of speech) tagging was also created and stored individually in the MySQL relational database. One point needs to be mentioned here is that there are certain errors within the post texts (e.g. typographical errors), and we do not use special methods to deal with these errors. This may affect the thread and post classification performance.

Originally, we planned to annotate 500 threads for our experiments. However, during the first pilot annotation process, we annotated 100 threads and found that 57 of them were labeled with *Spam*. This is because CNET forums contain many forums and sub-forums, and many of them are not computer-related (e.g. Car tech) nor troubleshooting-oriented (e.g. Forum feedback). Therefore, we filtered the threads to maximize the proportion of troubleshooting-related and computer-related threads, and only the threads from 4 particular forums (contain 34 sub-forums) of CNET forums were used. The details of these 4 forums and their sub-forums are listed in Table 3.

After filtering, our data collection contains 327 threads spanning 1371 posts. This data set was then put into MySQL relational database for annotation.

3.3 Annotation Process

Two annotators were involved in the task of annotation, and a dedicated annotation tool[5] was used. An improved κ calculation scheme was also proposed to compute the pairwise agreement statistics for our multi-class post annotation task. The detailed descriptions for the thread and post annotation processes and the κ calculation are presented in the following subsections.

[2] `http://forums.cnet.com`

[3] `http://www.checkdns.net/forum`

[4] `http://www.techsupportforum.com`

[5] `http://mandrake.csse.unimelb.edu.au/~rbp/cgi-bin/su/Classify.py`

Forum	Sub-forum		
Operating Systems	Windows 7	Windows Vista	Windows XP
	Windows 2000/NT	Windows ME	Windows 95/98
	Windows Mobile	Mac OS	Linux
Software	Audio & video	Browsers	CNET Download site
	E-mail, chat, & VoIP	Mac software	Office & productivity
	PC utilities	Photography & design	Spyware, viruses, & security
	Webware	Windows Live	
Hardware	Dell	Desktops	Laptops
	Mac hardware	Networking & wireless	PC hardware
	Peripherals	Storage	
Web Development	Coding & scripting	Web design & hosting	

Table 3: Data source forums and sub-forums

3.3.1 Thread Annotation

Thread Class Set

As is explained in Section 3.1.1, there are three thread class sets can be used. Because the combined class set (26 classes) is the superset of TYPE class set (6 classes) and SOURCE class set (8 classes), the combined class set was used to do the annotation.

Standard Cohen's Kappa for Thread Annotation

Standard Cohen's Kappa [4] was used to calculate the agreement statistics between two annotators:

$$\kappa = \frac{P(a) - P(e)}{1 - P(e)}$$

Where $P(a)$ refers to the relative observed agreement between two annotators, and $P(e)$ is the hypothetical probability of chance agreement between two annotators.

In order to illustrate the operations of calculating κ value, a set of hypothetical annotation statistics is made and presented in Table 4. In this hypothetical example, there are two annotators (i.e. 'S' and 'L'), three classes (i.e. 'A', 'B', and 'C'), and 5 threads. To calculate $P(a)$ and $P(e)$, a confusion matrix of the class set (i.e. [A, B, C] in this example) needs to be built with the two axes represent the two annotators' choices. The confusion matrix for this example is illustrated in Table 5. As we can see, for each thread, according to the choices made by the two annotators, we add one count to the relevant position in the confusion matrix. For example, in thread 4, the annotator 'S' labeled it with class 'A' while the annotator 'L' labeled it with class 'B'. Therefore, one count is added to the position (A, B) in the confusion matrix.

Annotator	Thread	Class label
S	1	A
	2	B
	3	C
	4	A
	5	B
L	1	A
	2	B
	3	C
	4	B
	5	A

Table 4: An hypothetical annotation example

		L		
		A	B	C
S	A	1	1	0
	B	1	1	0
	C	0	0	1

Table 5: Confusion matrix for the hypothetical example

After the confusion matrix is created, $P(a)$ and $P(e)$ are calculated by:

$$P(a) = \frac{\sum_i n_{i,i}}{N}, \qquad P(e) = \frac{\sum_i (\sum_j n_{i,j} \times \sum_j n_{j,i})}{N^2}$$

where $n_{i,j}$ depicts the value at the position (i, j) in the confusion matrix, and N refers to the total value summed across the whole confusion matrix.

Thread Annotation Process

First of all, a pilot annotation was performed before annotating the overall data set. The purpose of the pilot annotation is to ensure that the two annotators both comprehend the thread class set in the same way. 100 threads were randomly selected from the 1000 threads described in Section 3.2 and annotated by the two annotators independently. As is described in Section 3.2, after we annotated these 100 threads, we found that 57 of them are *Spam*. Because these *Spam* threads would be filtered later and not included in the actual thread classification process, it is not reasonable to include these 57 threads when calculating the κ value. Therefore, only the 43 non-*Spam* threads were used to compute the κ statistics. The κ value for the pilot annotation on these 43 threads only reached 0.41 and the $P(a)$ only attained 0.47.

Then, the 1000-thread data set was filtered as is explained in Section 3.2, and only 327 threads were left. The two annotators firstly checked some of the disagreed threads together to reach the agreement over these 43 threads. Furthermore, the filtered data set (i.e. 327 threads) was annotated by the two annotators independently. Although pilot annotation was used before annotating the whole data set, the κ value between two annotators was still not satisfactory: for the whole data set, the κ value only reached 0.43 and the $P(a)$ only attained 0.47.

Because the κ statistic was not satisfactory, re-annotation was carried out: firstly, 100 threads were randomly selected from the data set (i.e. 327 threads) and the disagreed threads were checked by the two annotators together to reach agreement. Then the rest of the data set (i.e. 227 threads) was re-annotated by the two annotators independently. The final κ value for the 227 threads reached 0.72 and the $P(a)$ attained 0.74. Then, the annotated data set was finalized: all the disagreed threads were checked by the two annotators together to reach agreement. One point needs to be noted is that there are two classes (i.e. *Search-Programming* and *Install-Programming*) were never used due to the lack of relevant threads.

Additionally, because the post class set cannot be applied on non-troubleshooting thread (which will be labeled as *Spam*) properly, in order to maintain the coherence between thread classification and post classification, all the *Spam* threads were removed from the data set. The final annotated data set contains 315 threads spanning 1332 posts.

3.3.2 Post Annotation

Multiple Post Classes and Post Links

In our posts, the situation in which one post raises multiple topics (e.g. the poster may answer part of the questions from a previous post, and confirm other parts of the questions to make them clearer) is not rare. This fact demands multiple classes for certain posts. Although introducing multiple classes into post classification task may increase the complexity, it is reasonable. This is because in a troubleshooting thread from a real world forum, it is normal for a poster to put several topics into one post.

Moreover, in a troubleshooting thread, except for the topics which are labeled as *Question-question* or *Other*, each topic implicitly refers back to a topic from a previous post. This link information is essential for analyzing the discourse structure of a given thread. Therefore, in addition to annotate the classes for each post, the link information for each class label (representing a topic) also needs to be annotated.

Improved Cohen's Kappa for Multi-class Situation

The standard Cohen's Kappa cannot be used in multi-class annotation tasks. By the time the annotation process started, no standard methodology for calculating Cohen's Kappa for multi-class tasks was found. Therefore, an extended method for calculating κ value was proposed to address this problem. In order to explain this scheme, a set of hypothetical post annotation statistics is generated and listed in the Table 6. In this hypothetical example, there are two annotators (i.e. 'S' and 'L'), three classes (i.e. 'A', 'B', and 'C'), and 3 posts.

As is described in Section 3.3.1, the most important step for computing the κ value is calculating the accumulative counts for each position in the confusion matrix based on the two annotators' choices. In our proposed scheme, the link information for each class label is taken into account. Firstly, for a post, the choices (i.e. classes) made by the two annotators are partitioned according to their link labels: classes that have the same link will be put into one group. Then, within each group, the two annotator's choices are combined only when they are the same and two counts will be given to this combination. Otherwise, their choices will be combined with a hypothetical class 'NULL' separately and be given one count for each. At last, these counts for the pairs are added into the their relevant positions in the confusion matrix. This approach implicitly extends the size of the confusion matrix . In the example showed in Table 6, there are three possible classes (i.e. 'A', 'B', and 'C') and three possible links (i.e. '1', '2', and '3'). Hence, each axis of the confusion matrix has 10 dimensions (i.e. $3 \times 3 + 1$). The structure of the confusion matrix is showed in Table 7. Examples for calculating the counts distributions for *Post 2* and *Post 3* by using this method are given in Figure 3 and Figure 4 respectively. The final confusion matrix is presented in Table 7 (including counts from *Post 2*, *Post 3*, and *Post 4*).

After the confusion matrix is created, the κ value can be calculated using the method explained in Section 3.3.1.

Annotator	Post	Class label	Link label
S	2	A	1
		B	1
	3	B	1
		C	2
	4	A	1
		B	3
L	2	A	1
		B	1
	3	B	2
	4	A	2
		B	3

Table 6: An hypothetical post annotation example

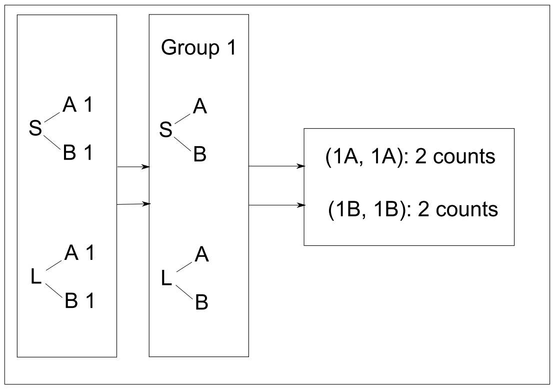

Figure 3: Calculating the counts distribution for *Post 2*

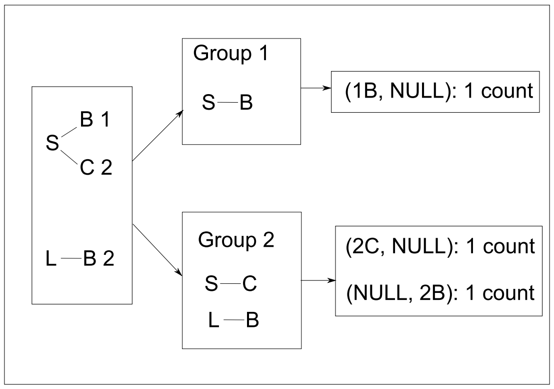

Figure 4: Calculating the counts distribution for *Post 3*

		L									
		1A	2A	3A	1B	2B	3B	1C	2C	3C	NULL
S	1A	2	0	0	0	0	0	0	0	0	1
	2A	0	0	0	0	0	0	0	0	0	0
	3A	0	0	0	0	0	0	0	0	0	0
	1B	0	0	0	2	0	0	0	0	0	1
	2B	0	0	0	0	0	0	0	0	0	0
	3B	0	0	0	0	0	2	0	0	0	0
	1C	0	0	0	0	0	0	0	0	0	0
	2C	0	0	0	0	0	0	0	0	0	1
	3C	0	0	0	0	0	0	0	0	0	0
	NULL	0	1	0	0	1	0	0	0	0	0

Table 7: Confusion matrix for the hypothetical post annotation example

This improved κ value calculation scheme still has some problems. Take the *Post3* in the Table 6 for example. The two annotators have certain amount of agreement because both of them thought about class 'B' for this post. However, under our calculation presented in Figure 4, there are no agreement counts generated. Therefore, this scheme still needs to be improved.

Post Annotation Process

First of all, 223 posts from the 43 threads described in Section 3.3.1 were used to do the pilot annotation. Our proposed scheme was used to calculate the κ value for the annotation agreement between two annotators. The κ value for the pilot annotation reached 0.56 and the $P(a)$ reached 0.67. Then, the two annotators checked some of the disagreed posts and links together.

In the second stage, the whole data set was annotated. 1299 posts from 315 threads (i.e. non-*Spam* threads) were annotated independently by the two annotators. The κ value for the annotation agreement reached 0.59 and the $P(a)$ reached 0.68 in this stage.

The next stage which will be done in the future is checking the disagreed posts and finalizing the annotation.

4 MODELS

Three different classification models were used in the experiments: 1-nearest-neighbor (1NN) model, Naïve Bayes (NB) model, and Support Vector Machine (SVM) model. Moreover, another two simple models were used as the baselines: majority class (ZeroR) model and random (Rand) model. We used a tool kit named Hydrat[6] to use these models in our experiments. The detailed descriptions for these models are outlined below. We assume that the size of the feature set F for each thread instance is n, the size of the class set C is m.

4.1 1-Nearest-Neighbor (1NN)

1NN model represents each thread as a data point in a n-dimensional space, where n is the number of features. Then, a test thread T is assigned a class based on the class of the closest training thread T_i (with class $c(T_i)$). The closest training thread of a test thread is determined by a distance or similarity metric. Regarding distance metric, this is based on:

$$\hat{c}(T) = c(\arg\min_{T_i} d(T, T_i))$$

With respect to the similarity metric sim, it is based on:

$$\hat{c}(T) = c(\arg\max_{T_i} sim(T, T_i))$$

In our experiments, a distance metric (i.e. skew divergence) and a similarity metric (i.e. cosine similarity) were used to build two different 1NN models (i.e. 1NNSkew and 1NNCos). In the following subsections, the applied distance and similarity metrics are described.

4.1.1 Skew Divergence (Skew)

Skew divergence [10] is a smoothed version of Kullback-Leibler divergence [9]. The basic idea is to treat each thread as a probability distribution with the value of each feature indicates the probability of a dimension. Then, the distance between two probability distributions (i.e. two threads) is calculated. In skew divergence, a smoothing factor α is used to linearly interpolate the two probability distributions ($P : \langle p_1, p_2, \ldots p_i, \ldots \rangle$ and $Q : \langle q_1, q_2, \ldots q_i, \ldots \rangle$):

$$s_\alpha(P, Q) = D_{KL}(P || \alpha Q + (1 - \alpha)P)$$

where:

$$D_{KL}(P||Q) = \sum_i p_i \log_2 \frac{p_i}{q_i}$$

In our experiments, α is set to 0.99 which follows the findings of [10] in the context of distributional similarity.

4.1.2 Cosine Similarity (Cos)

Cosine similarity measures the cosine of the angle between two thread vectors (with each feature as a dimension):

$$cos(\vec{P}, \vec{Q}) = \frac{\vec{P} \cdot \vec{Q}}{|\vec{P}||\vec{Q}|}$$

where:

$$\vec{P} \cdot \vec{Q} = \sum_i p_i q_i$$

and:

$$|\vec{P}| = \sqrt{\sum_i p_i^2}, \quad |\vec{Q}| = \sqrt{\sum_i q_i^2}$$

[6]`http://code.google.com/p/hydrat/source/checkout`

4.2 Naïve Bayes (NB)

Naïve Bayes model estimates the class-conditional probability for a thread by assuming that the features in the thread are conditionally independent. The class of a thread T is determined by:

$$\hat{c}(T) = \arg\max_{c_j \in C} P(c_j) \prod_i P(f_i|c_j)$$

where c_j is a class in the class set C, f_i is a feature in the feature set F. In multinominal Naïve Bayes, $P(f_i|c_j)$ is calculated by:

$$P(T|c_i) = \prod_{j=1}^{n} \frac{P(f_j|c_i)^{w_{T,f_j}}}{w_{T,f_j}!}, \quad P(f|c_i) = \frac{1 + \sum_{k=1}^{N} w_{k,f} P(c_i|T_k)}{n + \sum_{j=1}^{n} \sum_{k=1}^{N} w_{k,f_j} P(c_i|T_k)}$$

where w_{T,f_j} is the weight of the jth feature in thread T, n is the size of the feature set, and N is the size of the thread collection.

In our experiments, the rainbow [11] implementation of multinominal Naïve Bayes was used.

4.3 Support Vector Machine (SVM)

In the last few years, SVM has shown promising empirical results in many practical applications, especially in the field of text categorization [15]. It also works very well with high-dimensional data (i.e. large number of features) and has the ability to capture the interactions of the features. The basic idea of SVM is searching a maximum margin hyperplane: a decision boundary that can maximize the margin between training instances which belong to different classes.

The basic SVM only supports binary classification tasks and assumes the data is linearly separable. Further research has already extended the SVM to support multiclass classification and non-linearly separable data [3]. In our experiments, the bsvm[7] implementation of SVM was used. The default options of the bsvm was applied except the kernel option. According to our initial experiments, linear kernel[8] performed the best in our tasks. Therefore a multiclass bound-constrained support vector classification with a linear kernel was used.

4.4 Majority Class (ZeroR) and Random (Rand)

These two models were used as baselines of the experiments. Majority class, or ZeroR, model assigns the majority class of the training threads to each of the test threads. The random model randomly chooses a class from the class set presented in the training threads, and assign it to each of the test threads.

5 FEATURE WEIGHTING SCHEMES

Four different weighting schemes were used in our experiments: raw count (Raw), binary (Binary), Information Gain (IG), and Term Frequency-Inverse Document Frequency (TF-IDF). Raw count weighting uses the occurrences of the feature in a thread as its weight. Binary weighting assigns 1 to a feature if it appears in a thread, otherwise 0 will be assigned. The detailed explanations for IG and TF-IDF are presented in the following subsections.

5.1 Information Gain (IG)

Information Gain describes the discrimination ability of the features: the higher the Information Gain, the better discriminator the feature is. It is a supervised weighting scheme and is calculated for each feature based on the training thread set:

$$IG(f_i|R_T) = H(R_T) - \sum_{j=1}^{\mu} P(R_j)H(R_j)$$

where R_T represents the training thread set, $H(R_T)$ is the entropy of the class distribution within the training thread set, μ presents the number of different values of f_i (i.e. the ith feature), and R_j is the subgroup of R_T where $f_i = j$.

In our experiments, binary values of the features were used to create the subgroups of R_T. Therefore, for each feature, the thread set (i.e. R_T) was only divided into two subgroups (i.e. $\mu = 2$). One more characteristic of this scheme deserves notice is that the feature weight is calculated at threat set level rather than thread level. Therefore, in order to apply the weight to each individual thread, we multiplied the feature weight with the raw counts within each thread.

[7]http://www.csie.ntu.edu.tw/~cjlin/bsvm/

[8]Other kernel options (i.e. polynomial, radial basis function, and sigmoid) performed rather poorly in the initial experiments.

5.2 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is one of the most popular weighting schemes for text categorization, and has many variants. In general, it describes how important a feature is to a thread in the thread set. This importance increases proportionally to the number of times a feature appears in the thread but is offset by the frequency of the feature in the thread set. TF-IDF is a unsupervised scheme and was calculated separately for the training thread set and test thread set in out experiments. The formulas used in our experiments are:

$$TF_{i,j} = \frac{f_{i,j}}{l_j}, \quad IDF_i = \log_{10} \frac{N}{f_{i,N}}, \quad (TF\text{-}IDF)_{i,j} = TF_{i,j} \times IDF_i$$

where $f_{i,j}$ is the number of occurrences of the feature i in thread j, l_j is the length of the thread j, N is the size of the thread set, and $f_{i,N}$ is the number of threads in the thread set that have feature i appeared.

Because this weighting scheme is calculated at individual thread level for each feature, in our experiments, the weights were used directly on the features in each thread.

6 EXPERIMENTAL METHODOLOGY

6.1 Building Classifiers

6.1.1 Feature Sets

We assumed that the thread classification task is similar to text categorization task, and used the initial post or concatenated all posts in a thread to form the 'text' for the thread. Furthermore, in order to present the threads for the classifiers, we constructed different bag-of-words (BoW) feature sets by using the cross-product of the following options:

choosing the input text ($\times 4$):

- All the text contents of the posts in a thread are used to form the 'text' for the thread. (AllText)
- All the text contents and the titles of the posts in a thread are used to form the 'text' for the thread. (All)
- Only the text contents of the initial post in a thread are used to form the 'text' for the thread. (InitialText)
- Only the text contents and the title of the initial post in a thread are used to form the 'text' for the thread. (Initial)

pre-processing of the text ($\times 2$):

- The original contents of the text are used(Text).
- The original contents of the text are lemmatized before used (Lemma).

tokenization ($\times 2$):

- All the punctuation in the text is replaced by space character, and then the text is parsed into word tokens (PuncReplace).
- The text is parsed into word and non-word (i.e. punctuation) tokens (WordToken)

n-gram ($\times 2$):

- Unigram is used to process the tokens (Unigram).
- Bigram is unutilized to process the tokens (Bigram).

6.1.2 Feature Weighting

After the feature sets were derived, they were weighted by applying the cross-product of the following options:

weighting scheme ($\times 4$):

- The raw count of each feature in each thread is be used (Raw).
- Simple Binary scheme is used (Binary).
- The Information Gain is used to calculate the weight for each feature (IG)

- The common TF-IDF method is used to compute the feature weight (TF-IDF)

weighting threshold ($\times 3$):

- Every feature in each thread is weighted (CUT1).
- Only the features that appear more than once in a thread are weighted, otherwise the weight of 0 is given to the feature (CUT2).
- Only the features that appear more than twice in a thread are weighted, otherwise the weight of 0 is given to the feature (CUT3)

6.1.3 Model and Baseline

4 models were used to do the experiment: nearest-neighbor model using Cosine similarity (1NNCos), nearest-neighbor model using Skew divergence (1NNSKEW), Naïve Bayes model (NB), and Support Vector Machine model (SVM). As baselines, a majority class model (ZEROR) and a random model (RAND) were used.

6.2 Classification Tasks

As is described in Section 3.1.1, we have three different options to organize the thread class sets:

class set ($\times 3$):

- Only the *Solution Type* group and the 'Other' class will be used (TYPECLASS).
- Only the *Problem Source* group and the 'Other' class will be used (SOURCECLASS).
- The cross product of *Solution Type* group and the *Problem Source* group as well as the 'Other' class will be used (ALLCLASS).

Therefore there are three classification tasks.

6.3 Evaluation Metrics

The experiments were carried out based on 10-fold stratified cross-validation. The results were evaluated by using micro-averaged precision (P_μ), recall (R_μ) and F-score (F_μ), as well as macro-averaged precision (P_M), recall (R_M) and F-score (F_M). With respect to the micro-averaged statistics, they describe the average performance *per thread*. Because the prediction *per thread* is always unique, the micro-averaged precision, recall, and F-score should always be the same. Regarding the macro-averaged statistics, they describe the average performance *per class*. In both cases, the statistics were averaged across the 10-fold stratified cross-validation. This means that the averaged F_M is not necessarily the harmonic mean of the averaged P_M and R_M.

Because we are more interested in the classification accuracy at thread level, the micro-averaged F-score (F_μ) will be used as the primary evaluation metric.

7 RESULTS

7.1 Feature Sets Comparison and Analysis

In order to compare the performance of the different feature sets described in Section 6.1.1, firstly, the 'choosing the input text' option was set to ALLTEXT and 8 different feature sets were created by combining other options (i.e. 'pre-processing of the text', 'tokenization', and 'n-gram') listed in the Section 6.1.1. Secondly, 18 experiments (including the two baseline experiments) were set up by combining different models listed in Section 6.1.3 (i.e. 1NNCos, 1NNSKEW, NB, and SVM), different weighting scheme explained in Section 6.1.2 (i.e. RAW, BINARY, IG, and TFIDF), and the CUT1 threshold. Then, these 18 experiments were applied on the 8 different feature sets with three different class sets described in Section 6.2. The serial number and the name for each experiment is presented in Table 8. Additionally, all the 'choosing the input text' options were compared.

The main evaluation metric used in this stage is micro-averaged F-score (i.e. $F\mu$). Note that the macro-average statistics (i.e. P_M, R_M, and F_M) have the similar performance distribution on all the feature sets used in our experiments. Therefore, only the results for $F\mu$ will be showed in this part. Moreover, the experiment results for the three different class sets (i.e. TYPECLASS, SOURCECLASS, and ALLCLASS) also have the similar distribution and we are more interested in the ALLCLASS at the moment. Thus, only the experiment results for ALLCLASS class set will be presented.

No.	Experiment	No.	Experiment	No.	Experiment
1	ZeroR	2	Rand	3	SVM_Raw_Cut1
4	SVM_Binary_Cut1	5	SVM_IG_Cut1	6	SVM_TF-IDF_Cut1
7	1NNCos_Raw_Cut1	8	1NNCos_Binary_Cut1	9	1NNCos_IG_Cut1
10	1NNCos_TF-IDF_Cut1	11	NB_Raw_Cut1	12	NB_Binary_Cut1
13	NB_IG_Cut1	14	NB_TF-IDF_Cut1	15	1NNSkew_Raw_Cut1
16	1NNSkew_Binary_Cut1	17	1NNSkew_IG_Cut1	18	1NNSkew_TF-IDF_Cut1

Table 8: 18 experiments for feature sets comparison

7.1.1 N-gram Comparison and Analysis

The experiment results for AllClass are presented in Figure 5. Each of the four bar graphs shows the experiment results for a combination of input text (i.e. Text or Lemma) and tokenization method (i.e. PuncReplace or WordToken). Within each bar graph, black bars depicts the statistics of Unigram, while the white bars represents Bigram. The y-axis of the graph shows the micro-averaged F-score and the numbers on x-axis represent different experiments (18 experiments in total) listed in Table 8. The No. 20 on x-axis depicts the average performance results across the 18 experiments.

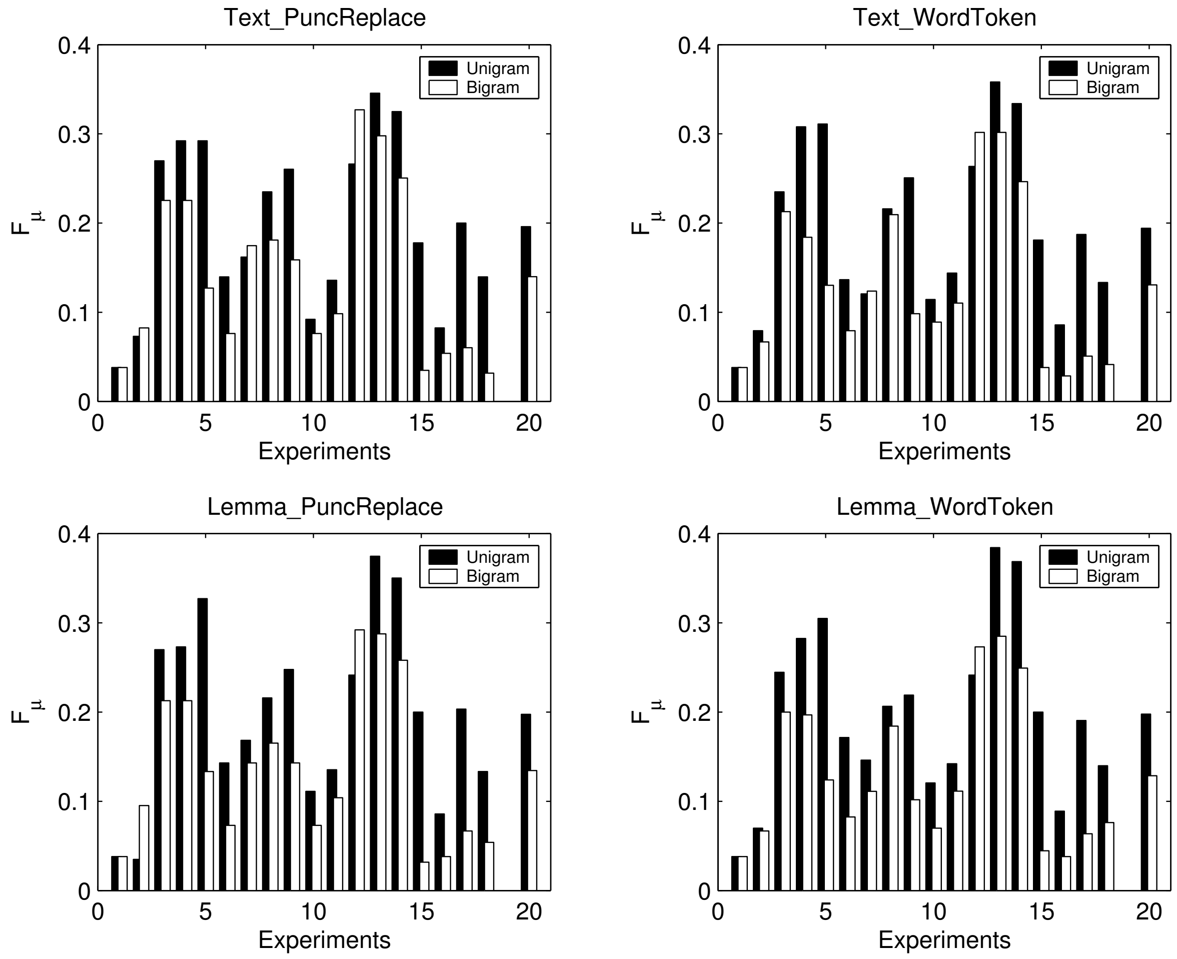

Figure 5: N-gram comparison

From the Figure 5, we can see that in most cases the Unigram based feature sets outperform the Bigram based feature sets, and the average performance of Unigram based feature sets is always much better than the average performance of Bigram based feature sets. Moreover, the Unigram based feature sets have the best performance statistics (always on NB_IG_Cut1). This is probably because the average length of each thread is relative short which makes the Bigram rather sparse. However, on classifier No. 12 (i.e. NB_Binary_Cut1) the Bigram based feature sets always outperform the Unigram based feature sets. This is probably because, on the one hand Binary weighting is not affected much by sparse data, on the other hand Bigram tends to produce a lot of isolated irrelevant features and NB is usually quite robust in this case.

In the rest experiments described in the report, only UNIGRAM based feature sets were considered.

7.1.2 Pre-processing and Tokenizer Comparison

Figure 6 illustrates the performance of all the UNIGRAM based feature sets over ALLCLASS class set. In this bar graph, the set up for x-axis and y-axis is the same as the set up described in Section 7.1.1.

From Figure 6, we can see that the performance of the 4 feature sets do not have distinct difference and the average results (i.e. No. 20) for these 4 feature sets are nearly the same. However, the UNIGRAM based LEMMA_WORDTOKEN feature set (i.e. ALLTEXT_LEMMA_WORDTOKEN_UNIGRAM) have the top two micro-averaged F-score (i.e. under NB_IG_CUT1 and NB_TF-IDF_CUT1).

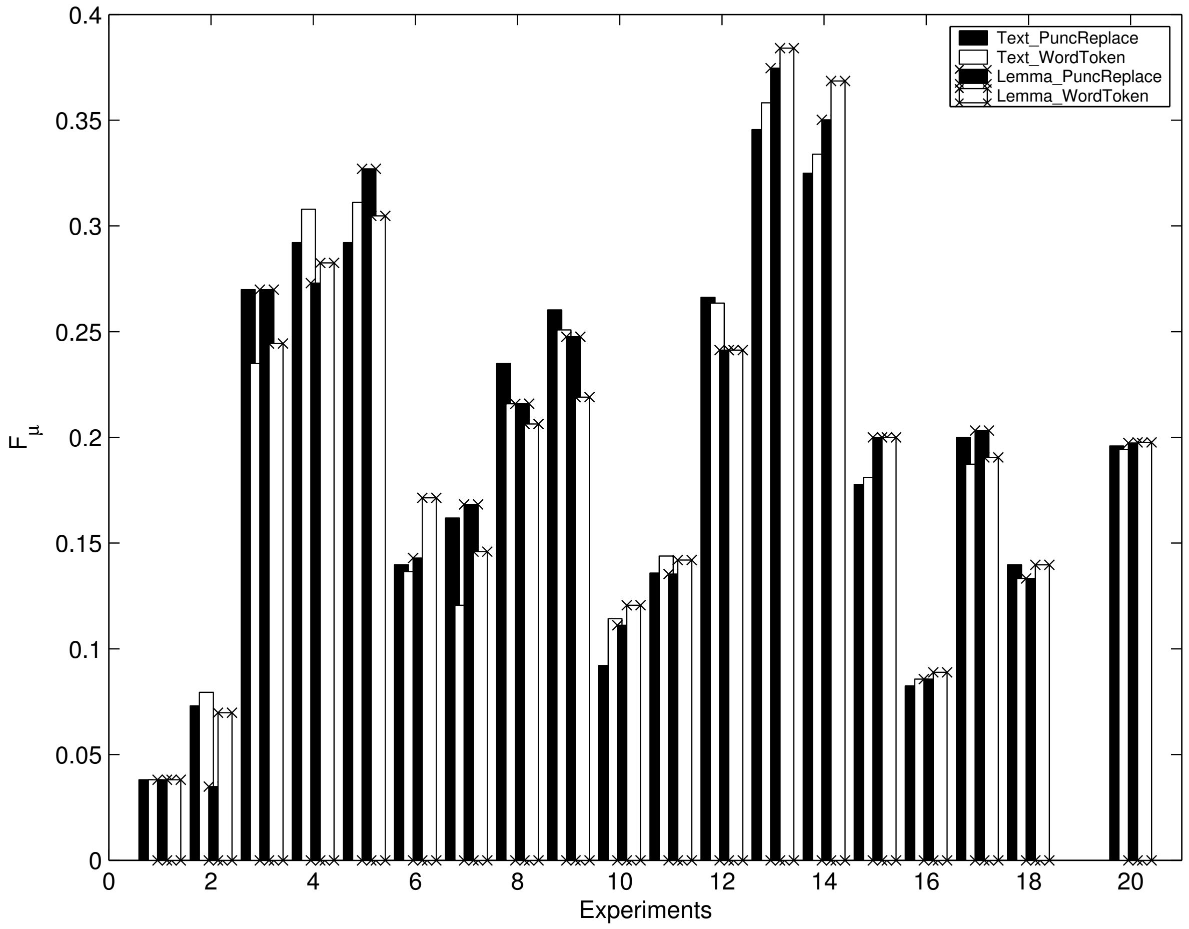

Figure 6: Pre-processing and Tokenizer Comparison

7.1.3 Input Text Comparison and Analysis

To compare the 4 different input text options (i.e. ALLTEXT, ALL, INITIALTEXT, and INITIAL) described in Section 6.1.1 , the combination of the rest feature sets options (i.e. 'pre-processing of the text', 'tokenization', and 'n-gram') was set to LEMMA_WORDTOKEN_UNIGRAM. Then, the same 18 experiments described at the beginning of Section 7.1 were applied on these 4 options. Moreover, for the similar reason explained at the beginning of Section 7.1, only $F\mu$ statistic and the results for ALLCLASS are presented. The experiment results are showed in Figure 7.

From the Figure 7, we can see, while the ALL option has the best average result (i.e. x-axis No. 20), the ALLTEXT option has the best result on No. 13 (i.e. NB_IG_CUT1). Moreover, both four options have the similar performance distributions across all the experiments. Therefore, we can only focus on one of them first (i.e. ALLTEXT).

7.2 Weighting Comparison and Analysis

To compare different thresholds (i.e. CUT1, CUT2, and CUT3) and different weighting schemes (i.e. RAW, BINARY, IG, and TF-IDF) described in Section 6.1.2, the ALLTEXT_LEMMA_WORDTOKEN_UNIGRAM feature set was used. Firstly, 14 experiments (including the 2 baseline experiments) were set up by combining different

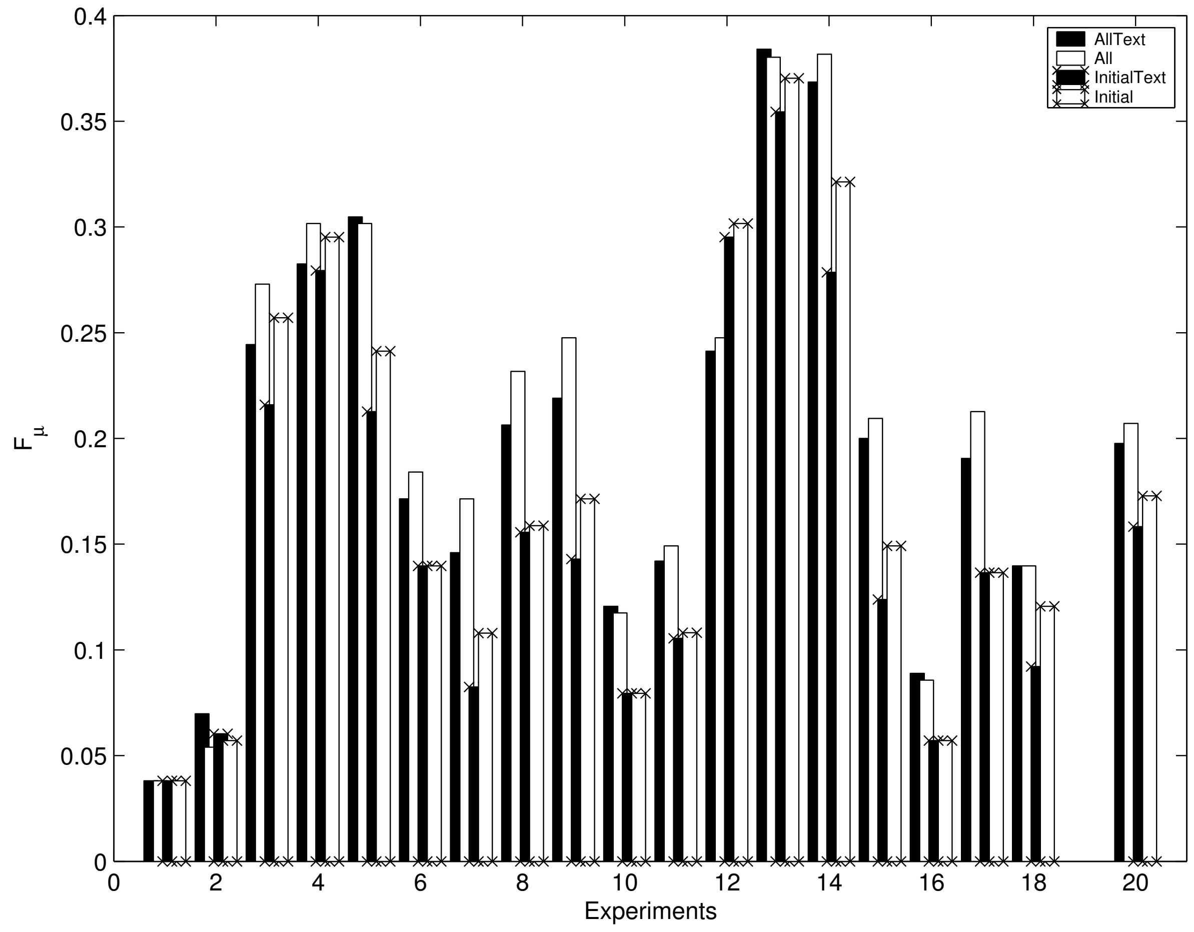

Figure 7: Input Text Comparision

models explained in Section 6.1.3 (i.e. 1NNCos, 1NNSkew, NB, and SVM) and different weighting schemes (i.e. BINARY, IG, and TF-IDF). These experiments were furthermore performed on 3 different threshold options (i.e. CUT1, CUT2, and CUT3) . The serial numbers and relative names of these 14 combinations are listed in Table 9. Secondly, 6 experiments (including the 2 baseline experiments) were set up by combing different models and the CUT1 threshold option. These experiments were furthermore performed on 4 different weighting schemes (i.e. RAW, BINARY, IG, and TF-IDF). The serial numbers and relative names of these 6 combinations are presented in Table 10.

No.	Experiment	No.	Experiment	No.	Experiment
1	ZeroR	2	Rand	3	SVM_Binary
4	SVM_IG	5	SVM_TF-IDF1	6	1NNCos_Binary
7	1NNCos_IG	8	1NNCos_TF-IDF	9	NB_Binary
10	NB_IG	11	NB_TF-IDF	12	1NNSkew_Binary
13	1NNSkew_IG	14	1NNSkew_TF-IDF		

Table 9: 14 experiments for threshold comparison

No.	Experiment	No.	Experiment	No.	Experiment
1	ZeroR	2	Rand	3	SVM
4	1NNCos	5	NB	6	1NNCos

Table 10: 6 experiments for weighting scheme comparison

Both micro-averaged F-score (i.e. F_μ) and macro-averaged statistics (i.e. P_M, R_M, and F_M) will be used in this case. Moreover, the experiment results for the three different class sets (i.e. TypeClass, SourceClass, and AllClass) have the similar distribution. Therefore, only the experiment results for AllClass class set will be presented.

7.2.1 Threshold Comparison and Analysis

The experiment results for comparing different weighting threshold options (i.e. CUT1, CUT2, and CUT3) are presented in Figure 8. Each of the four bar graphs shows the experiment results for a particular evaluation metric (i.e. F_μ, F_M, P_M, and R_M). Within each bar graph, three different bars depict three different threshold options. The numbers on the x-axis represent different combinations (from 1 to 14) listed in Table 9. The No. 16 on x-axis depicts the average performance results across the 14 experiments.

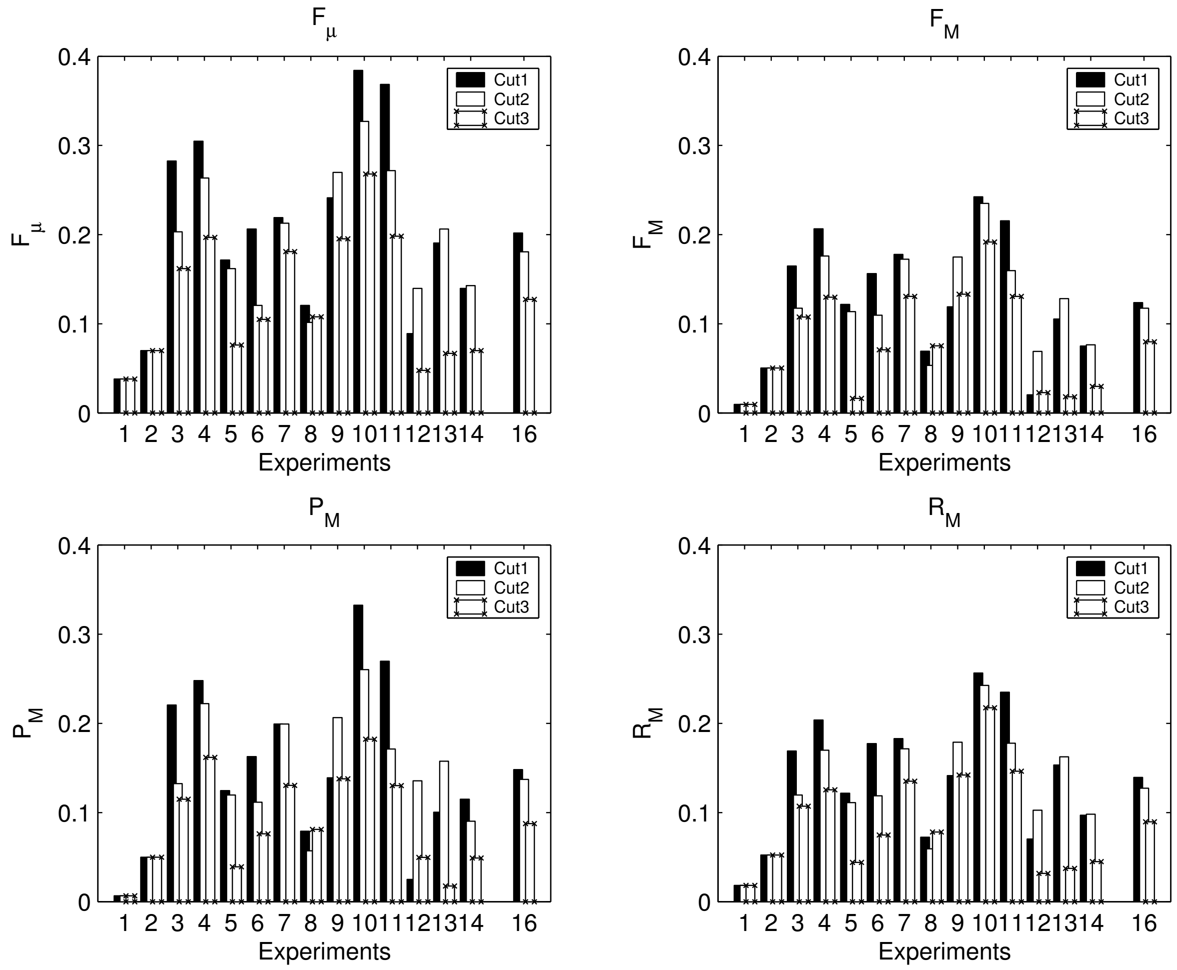

Figure 8: Threshold comparison

From Figure 8, we can see that the top two performance statistics (i.e. NB_IG and NB_TF-IDF) in all four bar graphs always come from the CUT1 threshold option. Moreover, the average performance results for CUT1 never become worse than other options across 4 different evaluation metrics.

7.2.2 Weighting Scheme Comparison and Analysis

The experiment results for comparing different weighting schemes (i.e. RAW, BINARY, IG, and TF-IDF) are illustrated in Figure 9. Each of the four bar graphs shows the experiment results for a particular evaluation metric (i.e. F_μ, F_M, P_M, and R_M). Within each bar graph, four different bars depict four different weighting schemes. The numbers on the x-axis represent different combinations (from 1 to 6) listed in Table 10. The number 8 on x-axis describes the average performance results across the 6 experiments.

From Figure 9, we can see that when measured by F_μ, F_M, and P_M, IG weighting scheme outperforms other schemes most of the time and have the best average performance. However, the RAW weighting scheme tends to be aggressive and results in high R_M with low P_M (thus derives relatively low F_M), especially when combined with NB model.

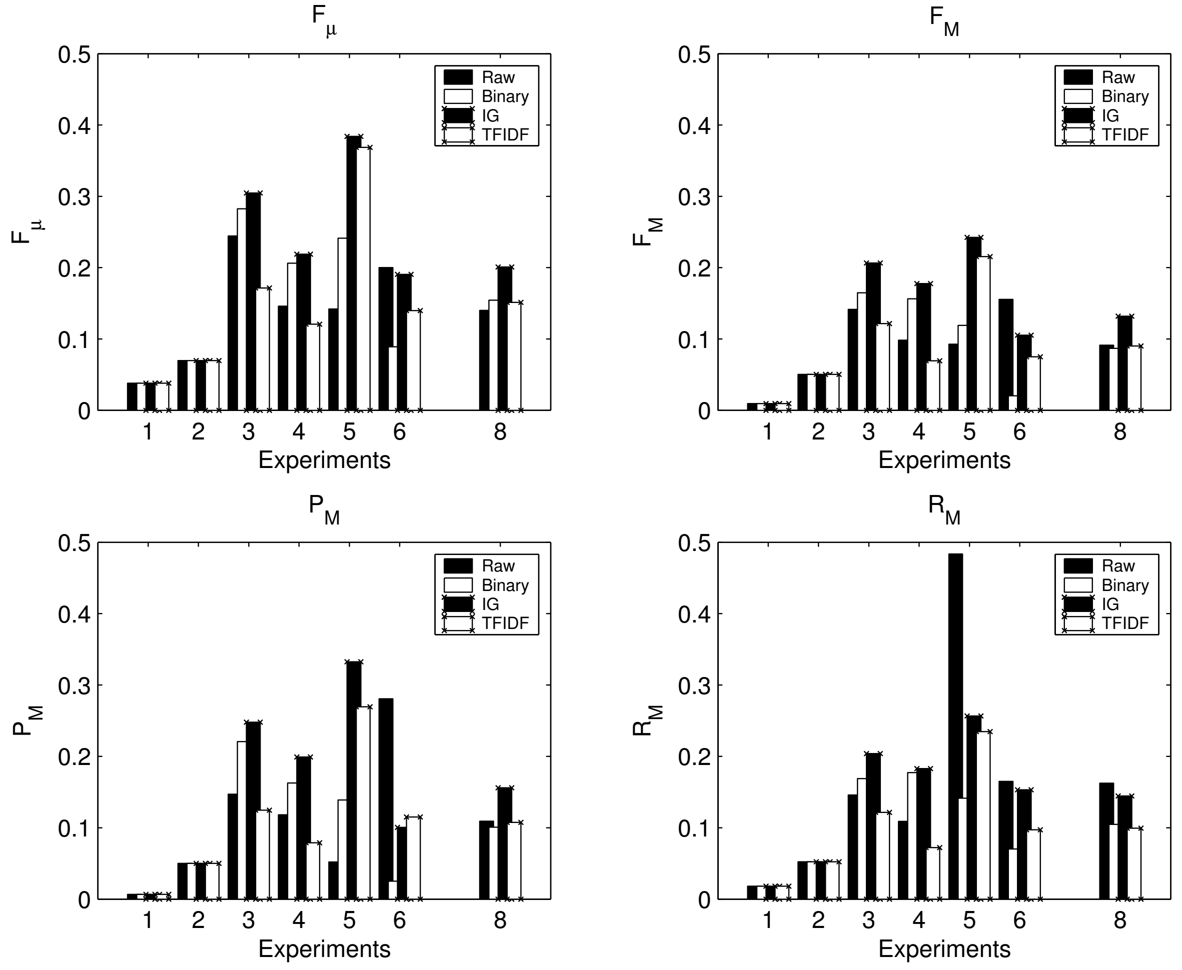

Figure 9: Weighting scheme comparison

7.3 Model Comparison and Analysis

7.3.1 Using ALLTEXT Based Feature Set[9]

In order to compare the performance of different models (i.e. 1NNCos, 1NNSkew, NB, and SVM) described in Section 6.1.3, the ALLTEXT_LEMMA_WORDTOKEN_UNIGRAM feature set and the CUT1 threshold were used. We combined different models with different weighting schemes (i.e. RAW, BINARY, IG, and TF-IDF), and set up 18 experiments (including the two baseline experiments) for the three class sets (i.e. TYPECLASS, SOURCECLASS, and ALLCLASS). The detailed experiment results are listed in Table 11, Table 13, and Table 14. In each case, the best result for each column is marked in **boldface**.

As a further extension, the class predictions for TYPECLASS and SOURCECLASS were combined in a *1-to-1* (i.e. the same classifier[10] is used for each side) manner to produce the prediction for ALLCLASS class set. When doing the combination, in order to make sure that the combined class set (COMBINECLASS) is the same as the ALLCLASS, the combinations of *Other-*,*[11] and **-Other* were treated as *Other* class. The detailed results for the COMBINECLASS are presented in Table 12. Again, the best result for each column is marked in **boldface**.

With respect to the Table 11, Table 13, and Table 14, the first point to notice is that NB model outperforms other models most of the time. This is probably because our bag-of-words feature set may contain many isolated irrelevant features and NB can robustly handle them. Secondly, RAW weighting scheme always results in the highest R_M, however other statistics for RAW weighting scheme are relatively low. Thirdly, SVM model with IG weighting has the highest P_M (i.e. 0.765) in the case of SOURCECLASS, however, other statistics for it are relatively low. Moreover, while in ALLCLASS class set (with 25 classes), IG weighting scheme with NB model leads to the best statistics in all field except R_M, in TYPECLASS class set (with 5 classes), TF-IDF weighting scheme with NB model performs best in all field except R_M. However, in the SOURCECLASS case, while IG with SVM performs the best at P_M, IG with NB derives the highest F_μ and F_M. This performance distribution

[9]In our context, the ALLTEXT based feature set refers to the ALLTEXT_LEMMA_WORDTOKEN_UNIGRAM feature set

[10]In this context, classifier refers to the combination of feature set, weighting method, and model.

[11]* refers to any possible basic class label

Model	Weighting	P_M	R_M	F_M	F_μ
ZeroR	-	.006	.018	.009	.038
Rand	-	.050	.052	.050	.069
SVM	Raw	.147	.145	.141	.244
SVM	Binary	.220	.169	.164	.282
SVM	IG	.247	.203	.206	.304
SVM	TF-IDF	.124	.121	.121	.171
1NNCos	Raw	.118	.109	.098	.146
1NNCos	Binary	.162	.177	.156	.206
1NNCos	IG	.199	.182	.177	.219
1NNCos	TF-IDF	.079	.072	.069	.120
NB	Raw	.052	**.483**	.092	.142
NB	Binary	.139	.141	.119	.241
NB	IG	**.332**	.256	**.242**	**.384**
NB	TF-IDF	.269	.234	.215	.368
1NNSkew	Raw	.280	.165	.155	.200
1NNSkew	Binary	.025	.070	.020	.088
1NNSkew	IG	.100	.153	.105	.190
1NNSkew	TF-IDF	.115	.097	.075	.139

Table 11: AllText based results for AllClass

Model	Weighting	P_M	R_M	F_M	F_μ
ZeroR	-	.006	.036	.011	.076
Rand	-	.032	.035	.033	.050
SVM	Raw	.222	.200	.199	.307
SVM	Binary	.246	.211	.203	.320
SVM	IG	.218	.187	.176	.295
SVM	TF-IDF	.164	.143	.137	.187
1NNCos	Raw	.118	.109	.098	.146
1NNCos	Binary	.162	.177	.156	.206
1NNCos	IG	.214	.203	.202	.273
1NNCos	TF-IDF	.079	.072	.069	.120
NB	Raw	.172	.093	.080	.155
NB	Binary	.154	.147	.130	.254
NB	IG	.284	**.268**	.254	**.387**
NB	TF-IDF	**.365**	.258	**.270**	.377
1NNSkew	Raw	.280	.165	.155	.200
1NNSkew	Binary	.025	.070	.020	.088
1NNSkew	IG	.089	.161	.102	.200
1NNSkew	TF-IDF	.115	.097	.075	.139

Table 12: AllText based results for CombineClass

Model	Weighting	P_M	R_M	F_M	F_μ
ZeroR	-	.122	.168	.140	.304
Rand	-	.203	.203	.202	.311
SVM	Raw	.385	.378	.377	.558
SVM	Binary	.529	.359	.363	.552
SVM	IG	.448	.334	.310	.555
SVM	TF-IDF	.370	.353	.346	.498
1NNCos	Raw	.281	.254	.243	.374
1NNCos	Binary	.318	.294	.302	.412
1NNCos	IG	.452	.406	.419	.485
1NNCos	TF-IDF	.259	.264	.258	.384
NB	Raw	.214	**.728**	.325	.416
NB	Binary	.357	.275	.252	.476
NB	IG	.503	.448	.455	.555
NB	TF-IDF	**.596**	.477	**.506**	**.610**
1NNSkew	Raw	.395	.334	.287	.441
1NNSkew	Binary	.106	.307	.156	.346
1NNSkew	IG	.294	.338	.204	.371
1NNSkew	TF-IDF	.274	.260	.223	.412

Table 13: AllText based results for TypeClass

Model	Weighting	P_M	R_M	F_M	F_μ
ZeroR	-	.038	.142	.060	.266
Rand	-	.114	.116	.115	.158
SVM	Raw	.474	.413	.422	.542
SVM	Binary	.578	.436	.451	.568
SVM	IG	**.765**	.433	.471	.555
SVM	TF-IDF	.278	.275	.270	.358
1NNCos	Raw	.258	.256	.250	.307
1NNCos	Binary	.380	.371	.353	.400
1NNCos	IG	.480	.472	.474	.536
1NNCos	TF-IDF	.202	.204	.199	.273
NB	Raw	.167	**.777**	.274	.343
NB	Binary	.401	.373	.365	.530
NB	IG	.637	.597	**.609**	**.683**
NB	TF-IDF	.654	.510	.541	.606
1NNSkew	Raw	.417	.362	.338	.361
1NNSkew	Binary	.386	.243	.117	.222
1NNSkew	IG	.437	.508	.435	.476
1NNSkew	TF-IDF	.319	.250	.234	.304

Table 14: AllText based results for SourceClass

probably results from the probability distribution of the classes in our data set. Because larger class set tends to have skewer distributions in which IG often works better, while smaller class set tends to have a more uniform distribution in which IG's performance decreases. The TF-IDF weighting scheme, on the other hand, is not affected too much by the class distribution.

Regarding the Table 11 and Table 12, by comparing the highest statistics between these two tables, it is interesting to see that the CombineClass outperforms AllClass slightly at both F_μ (0.387 vs. 0.384) and F_M (0.270 vs. 0.242). Moreover, one point should be noted is that the results of ZeroR from the two tables are different, this is because we have two majority classes in our data set (i.e. *Documentation-SW* and *Support-SW*) with the same occurrences (i.e. 29). In these two ZeroR presented in the tables, while *Documentation-SW* was selected as the majority class in the AllClass situation, the *Support-SW* was chosen as the majority class in the CombineClass situation. Additionally, according to the discussion presented in the previous paragraph, different classifiers perform differently on different basic class sets (i.e. TypeClass and SourceClass). Therefore, by combining different classifiers (rather than *1-to-1* combination) for each basic class set to produce the CombineClass may introduce further improvements.

7.3.2 Using ALL Based Feature Set[12]

As is illustrated in the Section 7.1.3, the ALL based feature set (i.e. ALL_LEMMA_WORDTOKEN_UNIGRAM) gives the best average performance. Therefore, it is necessary to look into this feature set when doing the classifier combination and the detail statistic analysis. This is because, when doing the classifier combination, we are more interested in the basic class sets (i.e. TYPECLASS and SOURCECLASS). We used the same experiments set up (i.e. 18 experiments) and classifier combination technical (i.e. *1-to-1* combination) described in Section 7.3.1 to do the experiments and combination on the ALL_LEMMA_WORDTOKEN_UNIGRAM feature set. The experiment results are presented in Table 15-18. In each case, the best result for each column is marked in **boldface**.

Model	Weighting	P_M	R_M	F_M	F_μ
ZEROR	-	.006	.018	.009	.038
RAND	-	.038	.040	.038	.054
SVM	RAW	.176	.170	.167	.273
SVM	BINARY	.245	.180	.177	.301
SVM	IG	.229	.202	.204	.301
SVM	TF-IDF	.130	.137	.131	.184
1NNCOS	RAW	.155	.144	.123	.171
1NNCOS	BINARY	.194	.207	.187	.231
1NNCOS	IG	.202	.209	.199	.247
1NNCOS	TF-IDF	.063	.071	.062	.117
NB	RAW	.055	**.480**	.097	.149
NB	BINARY	.122	.143	.116	.247
NB	IG	**.293**	.253	**.236**	.380
NB	TF-IDF	.290	.244	.228	**.381**
1NNSKEW	RAW	.249	.172	.156	.209
1NNSKEW	BINARY	.031	.064	.018	.085
1NNSKEW	IG	.155	.172	.121	.212
1NNSKEW	TF-IDF	.115	.092	.069	.139

Table 15: ALL based results for ALLCLASS

Model	Weighting	P_M	R_M	F_M	F_μ
ZEROR	-	.006	.036	.011	.076
RAND	-	.041	.037	.039	.057
SVM	RAW	.269	.244	.241	.346
SVM	BINARY	.272	.216	.215	.323
SVM	IG	.221	.197	.184	.311
SVM	TF-IDF	.134	.129	.118	.184
1NNCOS	RAW	.155	.144	.123	.171
1NNCOS	BINARY	.194	.207	.187	.231
1NNCOS	IG	.237	.217	.220	.288
1NNCOS	TF-IDF	.063	.071	.062	.117
NB	RAW	.171	.108	.100	.181
NB	BINARY	.152	.151	.132	.260
NB	IG	.335	**.287**	**.280**	**.409**
NB	TF-IDF	**.374**	.260	.269	.384
1NNSKEW	RAW	.249	.172	.156	.209
1NNSKEW	BINARY	.031	.064	.018	.085
1NNSKEW	IG	.116	.151	.094	.190
1NNSKEW	TF-IDF	.115	.092	.069	.139

Table 16: ALL based results for COMBINECLASS

Model	Weighting	P_M	R_M	F_M	F_μ
ZEROR	-	.122	.168	.140	.304
RAND	-	.149	.145	.147	.228
SVM	RAW	.413	.406	.407	.581
SVM	BINARY	.565	.378	.389	.571
SVM	IG	.407	.330	.306	.549
SVM	TF-IDF	.404	.376	.367	.511
1NNCOS	RAW	.328	.310	.295	.412
1NNCOS	BINARY	.414	.367	.383	.444
1NNCOS	IG	.412	.395	.398	.485
1NNCOS	TF-IDF	.234	.242	.234	.355
NB	RAW	.221	**.728**	.334	.428
NB	BINARY	.397	.285	.261	.495
NB	IG	.562	.465	.484	.585
NB	TF-IDF	**.563**	.472	**.495**	**.603**
1NNSKEW	RAW	.379	.350	.279	.428
1NNSKEW	BINARY	.102	.287	.150	.352
1NNSKEW	IG	.267	.325	.186	.355
1NNSKEW	TF-IDF	.266	.259	.216	.409

Table 17: ALL based results for TYPECLASS

Model	Weighting	P_M	R_M	F_M	F_μ
ZEROR	-	.038	.142	.060	.266
RAND	-	.134	.136	.134	.190
SVM	RAW	.470	.417	.424	.546
SVM	BINARY	.576	.435	.452	.561
SVM	IG	**.786**	.462	.499	.590
SVM	TF-IDF	.264	.269	.262	.352
1NNCOS	RAW	.278	.274	.268	.330
1NNCOS	BINARY	.439	.429	.413	.431
1NNCOS	IG	.479	.468	.472	.552
1NNCOS	TF-IDF	.191	.205	.194	.260
NB	RAW	.172	**.757**	.279	.355
NB	BINARY	.404	.375	.365	.533
NB	IG	.664	.619	**.634**	**.694**
NB	TF-IDF	.680	.526	.545	.625
1NNSKEW	RAW	.416	.392	.348	.381
1NNSKEW	BINARY	.325	.227	.113	.219
1NNSKEW	IG	.430	.495	.417	.469
1NNSKEW	TF-IDF	.295	.229	.215	.295

Table 18: ALL based results for SOURCECLASS

The results distributions in the Table 15-18 (i.e. ALL based) are relatively similar compared to the results distributions in the Table 11-14 (i.e. ALLTEXT based). However, there are several interesting aspects need to be noted.

First of all, while the best F_M and F_μ for ALLCLASS appear in the ALLTEXT based feature set (i.e. in Table 11, NB model with IG weighting), the best F_M and F_μ for COMBINECLASS appear in the ALL based feature set (i.e.

[12]In our context, the ALL based feature set refers to the ALL_LEMMA_WORDTOKEN_UNIGRAM feature set

in Table 16, NB model with IG weighting). This is because the classification performance in COMBINECLASS situation is not directly affected by the results from the ALLCLASS. It is directly decided by the results from TYPECLASS and SOURCECLASS. In this case, the F_M and F_μ of NB model with IG weighting in the Table 17 (i.e. ALL based feature set for TYPECLASS) are higher than the corresponding values in Table 13 (i.e. ALLTEXT based feature set for TYPECLASS); at the same time the F_M and F_μ of NB model with IG weighting in the Table 18 are also higher than the corresponding values in the Table 14. This probably makes the higher F_M and F_μ for COMBINECLASS appear in Table 16 (i.e. ALL based feature set with NB model and IG weighting) rather than Table 12 (i.e. ALLTEXT based feature set with NB model and IG weighting). In short, this observation may indicate that by using better classifier for each basic class set (i.e. TYPECLASS and SOURCECLASS), better results for the COMBINECLASS may derive.

Furthermore, another interesting observation is that the best F_M and F_μ for TYPECLASS appear in the ALLTEXT based feature set (i.e. in Table 13, NB model with TF-IDF weighting), while the best F_M and F_μ for SOURCECLASS appear in the ALL based feature set (i.e. in Table 18, NB model with IG weighting). This observation implies that we may need different classifiers for the two basic class set (i.e. TYPECLASS and SOURCECLASS) to achieve the best performance for each class set. Therefore, the *N-to-N* (i.e. different classifiers are used for each side) combination scheme becomes more appealing.

7.4 N-to-N Combination Analysis

As is explained in Section 7.3, if the classification results for TYPECLASS and SOURCECLASS were combined in a *N-to-N* (i.e. different classifiers are used for each side) manner to produce the predictions for ALLCLASS class set, further improvements may be achieved. To test this assumption, 4 classifiers[13] were built by combining NB model with different feature sets (i.e. ALLTEXT based feature set and ALL based feature set) and different weighting schemes (i.e. IG, and TF-IDF). Then, these 4 classifiers were applied on the two basic class sets (i.e. TYPECLASS and SOURCECLASS) individually. Furthermore, the results from the two basic class sets were combined in a *N-to-N* manner. The results are listed in the Table 19. The best result for each evaluation metric is marked in **boldface**.

TYPECLASS Classifier		SOURCECLASS Classifier		COMBINECLASS Results			
Feature Set	Weighting	Feature Set	Weighting	P_M	R_M	F_M	F_μ
ALLTEXT based	IG	ALLTEXT based	IG	.284	.268	.254	.387
ALLTEXT based	IG	ALLTEXT based	TF-IDF	.236	.233	.222	.339
ALLTEXT based	TF-IDF	ALLTEXT based	IG	.346	.309	.310	.428
ALLTEXT based	TF-IDF	ALLTEXT based	TF-IDF	.365	.258	.270	.377
ALL based	IG	ALL based	IG	.335	.287	.280	.409
ALL based	IG	ALL based	TF-IDF	.309	.254	.247	.377
ALL based	TF-IDF	ALL based	IG	.366	**.319**	**.321**	.425
ALL based	TF-IDF	ALL based	TF-IDF	.374	.260	.269	.384
ALLTEXT based	IG	ALL based	IG	.297	.277	.264	.393
ALLTEXT based	IG	ALL based	TF-IDF	.261	.240	.228	.355
ALLTEXT based	TF-IDF	ALL based	IG	.352	.312	.315	**.431**
ALLTEXT based	TF-IDF	ALL based	TF-IDF	.370	.269	.278	.396
ALL based	IG	ALLTEXT based	IG	.322	.278	.270	.403
ALL based	IG	ALLTEXT based	TF-IDF	.313	.248	.243	.361
ALL based	TF-IDF	ALLTEXT based	IG	.357	.316	.316	.422
ALL based	TF-IDF	ALLTEXT based	TF-IDF	**.376**	.253	.265	.371

Table 19: N-to-N Combination Analysis

The results from the Table 19 are appealing: the best F_μ (i.e. 0.431) and F_M (i.e. 0.321) are the highest so far. Compared to the best results we have got for ALLCLASS (i.e. 0.384 and 0.242 respectively), the improvements are reasonably significant. Moreover, it is interesting to take a look at the combination that derives the best F_μ, as F_μ is the most important statistic in our case. The classifier on the TYPECLASS side is made of ALLTEXT based feature set, TF-IDF weighting, and NB model; and the classifier on the SOURCECLASS side is composed of ALL based feature set, IG weighting, and NB model. As is discussed in the Section 7.3.2, these two classifiers performs the best on the relevant basic class set separately. Therefore, it is possible to get good results from the COMBINECLASS by combining better classifiers from both basic class sets.

[13]Choosing of the candidate components for building the classifiers is based on the analysis that have been done in the previous subsections

7.5 Class Distribution and Performance Analysis

At last, in order to improve the thread-level classification performance furthermore in the future, it is necessary to look into the actually class distribution of our class sets and the details of the classification performance for each class. The analysis can be done from two different aspects: the ALLCLASS aspect and the COMBINECLASS aspect.

7.5.1 Class Analysis on ALLCLASS

From the previous experiments, we know that the combination of ALLTEXT based feature set, NB model with IG weighting and CUT1 threshold can produce the best F_μ on the ALLCLASS class set. Therefore, the results of this classifier will be analyzed. The name, number of occurrence, and the Precision, Recall, F-score statistics for each class are listed in Table 20.

Class Name	Occurrence	Precision	Recall	F-score
Search-OS	1	.000	.000	.000
Support-Programming	1	.000	.000	.000
Install-SW	3	.000	.000	.000
Search-Network	3	.000	.000	.000
Install-HW	5	.000	.000	.000
Search-HW	5	.000	.000	.000
Documentation-Programming	7	.500	.143	.222
Install-Media	8	.000	.000	.000
Other	8	.667	.500	**.571**
Documentation-Network	9	.000	.000	.000
Install-Network	9	.167	.222	.190
Install-OS	9	.500	.111	.182
Search-Media	13	.000	.385	.556
Documentation-Media	14	.000	.071	**.133**
Support-Media	15	.400	.533	.457
Support-Network	18	.464	.722	.565
Search-SW	23	.750	.130	**.222**
Support-HW	23	.340	.783	.474
Documentation-OS	27	.375	.222	**.279**
Documentation-HW	28	.545	.643	.590
Support-OS	28	.386	.607	.472
Documentation-SW	29	.196	.310	**.240**
Support-SW	29	.357	.517	.423
overall	315	.384	.384	.384

Table 20: Detailed statistics for ALLCLASS class set

The first point to notice is that our classifier works rather poorly on classes that have small number of threads (e.g. less than 10). We could call these classes minority classes. The majority classes (i.e. contain more than 10 threads), on the other hand , have relatively good results most of the time. Moreover, there are some inconsistencies within both minority class group and majority class group. For example, the *Other* class has relatively high statistics (with a F-score of 0.571) in the minority class group, while the *Documentation-Media* class has relatively low statistics (with a F-score of 0.133) in the majority class group. In the future research, we could focus more on the inconsistencies in the majority class group. For instance, finding features to help classifiers identify *Documentation-Media*, *Search-SW*, *Documentation-OS*, and *Documentation-SW* classes more accurately.

7.5.2 Class Analysis on COMBINECLASS

Doing class-level analysis for COMBINECLASS is a little different. Because the results for COMBINECLASS are derived from the combination of the results of TYPECLASS and SOURCECLASS. Therefore, the analysis should be carried out on these two basic class sets. Again, the results from the best classifier for each basic class set will be used. In our case, the best classifier on the TYPECLASS side is made of ALLTEXT based feature set, TF-IDF weighting, and NB model, and the best classifier on the SOURCECLASS side is composed of ALL based feature set, IG weighting, and NB model. The detailed results for each basic class set are presented in Table 21 and Table 22.

Class Name	Occurrence	Precision	Recall	F-score
Other	8	.750	.375	.500
Install	34	.346	.265	.300
Search	45	.632	.267	.375
Documentation	114	.600	.684	.639
Support	114	.655	.798	.719
overall	315	.607	.613	.610

Table 21: Detailed statistics for TYPECLASS class set

Class Name	Occurrence	Precision	Recall	F-score
Other	8	.667	.500	.571
Programming	8	.400	.250	.308
Network	39	.810	.872	.840
Media	50	.778	.560	.651
HW	61	.691	.770	.729
OS	65	.636	.646	.641
SW	84	.667	.738	.701
overall	315	.693	.695	.694

Table 22: Detailed statistics for SOURCECLASS class set

The results from Table 21 and Table 22 illustrate the similar performance distributions: majority classes (e.g. more than 50 in the case of TYPECLASS and more than 10 in the case of SOURCECLASS) tend to have higher F-score than minority classes. Therefore, further research may focus more on these minority classes.

8 FUTURE WORK

Only half of the project was accomplished by the time this report was finished. With respect to the finished parts, there are still a number of areas need to be enhanced.

First of all, the κ value calculation we used for multiple class tasks in our research is still problematic. New schemes need to be designed to address our posts classification tasks.

Secondly, a new feature weighting scheme named BNS (Bi-Normal Separation) which shows promising experiment results in [7]'s research can be considered. Because the original BNS weighting method can only be used in binary classification tasks, further extension for BNS also needs to be designed.

Furthermore, more work needs to be performed on the feature engineering of thread feature sets including feature selection and exploring new features for the threads. The work can be done from two aspects: ALLCLASS oriented and COMBINECLASS oriented.

Regarding the rest part of this project, firstly, more sophisticated thread classification experiments will be performed. Then, the post annotation will be finalized and initial post-level classification experiments will be carried out. Furthermore, more advanced experiments and analysis will be conducted on posts classes and links. Additionally, the thread discourse structure will be analyzed. The posts classification results and the thread discourse structure information will then be used to enhance the thread deacidification tasks.

9 CONCLUSION

This first part of the project mainly focused on the thread and post class sets design, data collection and preprocessing, thread and post annotation, and thread classification.

Most of the efforts were used on the annotation of the thread data and the thread classification. With respect to the thread annotation, two passes of annotation were performed before finalizing the data. Within each pass, a pilot annotation was performed before the whole data set was annotated independently by the two annotators.

Regarding the thread classification, various input texts, pre-processing methods, tokenizers, n-grams, feature weighting methods, and classification models were combined to form different classifiers. These classifiers were furthermore applied on three thread class sets (i.e. ALLCLASS, TYPECLASS, and SOURCECLASS). The results from TYPECLASS and SOURCECLASS were combined in a *1-to-1* manner to produce the COMBINECLASS.

Our experiment results showed that, firstly, the NB model performs better than other models in our thread classification tasks. Moreover, the IG and TF-IDF weighting have their strengths and weaknesses over different class sets (i.e. ALLCLASS, TYPECLASS, and SOURCECLASS). Furthermore, the COMBINECLASS showed tempting results. These observations inspired us to combine the TYPECLASS and SOURCECLASS in a *N-to-N* manner which derived appealing results.

With respect to the finished parts of the project, there are some aspects can be enhanced. Moreover, the rest part of this project also needs to be done in the future.

Acknowledgements Great thanks to Timothy Baldwin and Su Nam Kim who provided many valuable comments for the report. Special thanks to Marco Lui who gave me technical supports for using the Hydrat tool kit.

References

[1] T. Baldwin, D. Martinez and R.B. Penman. Automatic Thread Classification for Linux User Forum Information Access. In *Proceedings of the Twelfth Australasian Document Computing Symposium (ADCS 2007)*, pages 72–9, 2007.

[2] K. E. Boyer, E. Y. Ha, M. Wallis, R. Phillips, M. Vouk and J. Lester. Discovering Tutorial Dialogue Strategies with Hidden Markov Models. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence in Education*, pages 1–4. Brighton, U.K., 2009.

[3] H. Chih-Wei, C. Chih-Chung and L. Chih-Jen. A practical guide to support vector classification. *Technical report, Department of Computer Science, National Taiwan University*, pages 1–15, 2009.

[4] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, Volume 20, Number 1, pages 37, 1960.

[5] G. Cong, L. Wang, C.Y. Lin, Y.I. Song and Y. Sun. Finding question-answer pairs from online forums. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 467–474. ACM New York, NY, USA, 2008.

[6] S. Ding, G. Cong, C.Y. Lin and X. Zhu. Using conditional random fields to extract contexts and answers of questions from online forums. In *In Proceedings of ACL-08: HLT*. ACL, 2008.

[7] G. Forman. Bns feature scaling: an improved representation over tf-idf for svm text classification. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge mining*, pages 263–270, New York, NY, USA, 2008. ACM.

[8] J. Ko, E. Nyberg and L. Si. A probabilistic graphical model for joint answer ranking in question answering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 343–350. ACM New York, NY, USA, 2007.

[9] S. Kullback and RA Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

[10] L. Lee. On the effectiveness of the skew divergence for statistical language analysis. In *Artificial Intelligence and Statistics*, pages 65–72. Citeseer, 2001.

[11] A.K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996.

[12] A. Nenkova and A. Bagga. Facilitating email thread access by extractive summary generation. *Recent advances in natural language processing III: selected papers from RANLP 2003*, pages 287, 2004.

[13] O. Rambow, L. Shrestha, J. Chen and C. Lauridsen. Summarizing email threads. In *HLT/NAACL*, pages 2–7, 2004.

[14] L. Shrestha and K. McKeown. Detection of question-answer pairs in email conversations. In *Proc. of COLING*, pages 889–895, 2004.

[15] P.N. Tan, M. Steinbach and V. Kumar. *Introduction to data mining*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2005.

[16] S. Wan and K. McKeown. Generating overview summaries of ongoing email thread discussions. In *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics Morristown, NJ, USA, 2004.

[17] N. Wanas, M. El-Saban, H. Ashour and W. Ammar. Automatic scoring of online discussion posts. In *WICOW '08: Proceeding of the 2nd ACM workshop on Information credibility on the web*, pages 19–26, New York, NY, USA, 2008. ACM.

[18] M. Weimer and I. Gurevych. Predicting the perceived quality of web forum posts. In *Proceedings of the 2007 Conference on Recent Advances in Natural Language Processing, RANLP*. Citeseer, 2007.

[19] M. Weimer, I. Gurevych and M. Mühlhäuser. Automatically assessing the post quality in online discussions on software. In *ACL*. The Association for Computer Linguistics, 2007.

[20] P. Zhao. Minor research project: Experiments on problem-type document classification over linux forum threads. Technical report, CSSE, University of Melbourne, VIC 3010 Australia, November 2008.

[21] L. Zhou and E. Hovy. Digesting virtual geek culture: The summarization of technical internet relay chats. *Ann Arbor*, Volume 100, 2005.