AUTOMATIC TAGGING OF COMMUNICATION DATA

Matthew Ray Hoyt

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2012

APPROVED:

Kathy Swigger, Major Professor
Robert Brazile, Minor Professor
DXiaohui Yuan, Committee Member
Barrett Bryant, Chair of the Department of Computer Science and Engineering
Costas Tsatsoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School
Globally distributed software teams are widespread throughout industry. But finding reliable methods that can properly assess a team's activities is a real challenge. Methods such as surveys and manual coding of activities are too time consuming and are often unreliable. Recent advances in information retrieval and linguistics, however, suggest that automated and/or semi-automated text classification algorithms could be an effective way of finding differences in the communication patterns among individuals and groups. Communication among group members is frequent and generates a significant amount of data. Thus having a web-based tool that can automatically analyze the communication patterns among global software teams could lead to a better understanding of group performance. The goal of this thesis, therefore, is to compare automatic and semi-automatic measures of communication and evaluate their effectiveness in classifying different types of group activities that occur within a global software development project. In order to achieve this goal, we developed a web-based component that can be used to help clean and classify communication activities. The component was then used to compare different automated text classification techniques on various group activities to determine their effectiveness in correctly classifying data from a global software development team project.
Copyright 2012
by
Matthew Ray Hoyt
## CONTENTS

### LIST OF TABLES

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
</tbody>
</table>

### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
</tr>
</tbody>
</table>

### CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>1.1. Related Work</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1. Automated Text Categorization</td>
<td>4</td>
</tr>
</tbody>
</table>

### CHAPTER 2. METHODOLOGY

<table>
<thead>
<tr>
<th>2.1. Data</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2. The Coding Scheme</td>
<td>7</td>
</tr>
<tr>
<td>2.3. Overview of the System</td>
<td>8</td>
</tr>
<tr>
<td>2.4. Architecture</td>
<td>9</td>
</tr>
<tr>
<td>2.5. Cleaning the Text</td>
<td>10</td>
</tr>
<tr>
<td>2.6. Splitting the Text into Sentences</td>
<td>11</td>
</tr>
<tr>
<td>2.7. Text Cleaning</td>
<td>12</td>
</tr>
<tr>
<td>2.8. Ngrams</td>
<td>12</td>
</tr>
<tr>
<td>2.9. Stop Words</td>
<td>12</td>
</tr>
<tr>
<td>2.10. Words and Ngrams weights</td>
<td>12</td>
</tr>
<tr>
<td>2.11. Building the Model and Classifying Project Activity</td>
<td>13</td>
</tr>
<tr>
<td>2.11.1. Adding and Updating a Classifier for Training</td>
<td>14</td>
</tr>
<tr>
<td>2.12. Classifier Storage</td>
<td>14</td>
</tr>
<tr>
<td>2.12.1. Review Classification</td>
<td>15</td>
</tr>
</tbody>
</table>

### CHAPTER 3. RESULTS

| 3.1. Whole Text | 17 |
| 3.2. Sentences | 19 |
CHAPTER 4. CONCLUSIONS 23

4.1. Findings 24
4.2. Conclusions 25
4.3. Future Work 26
4.4. Summary 26

BIBLIOGRAPHY 28
# LIST OF TABLES

2.1 Coding Scheme and Communication Behavior Categories [17] 8

3.1 Classification Whole Text 19

3.2 Classification Review Sentences 21
LIST OF FIGURES

2.1 Architecture 10
2.2 Cleaning Text 10
2.3 Breaking Text into Sentences 11
2.4 Classify Sentences 14
2.5 Classifications Review 15
3.1 Whole Text True Positive Rate 18
3.2 Whole Text False Positive Rate 18
3.3 Sentences True Positive Rate 20
3.4 Sentences False Positive Rate 21
3.5 Kappa Sentences and All Text 22
CHAPTER 1

INTRODUCTION

Because of the growing importance of teams in organizations, there has been a growing interest in analyzing group activities. Unfortunately, reliable and valid techniques for analyzing team activities has been difficult to achieve [32][39]. This has been particularly true of the team research that focuses on distributed software development teams. Individuals who work on distributed software projects are often separated by space and time, with asynchronous and distributed decisions becoming the norm, rather than the exception [13][44]. Further, group membership can be dynamic with the identity of participants and tasks changing over time. Currently there is no overall consensus on how to capture and analyze global software teams, partly because there are no consistent and automatic techniques for examining team activities[31]. Current approaches for analyzing group activities tend to rely on human observers who rate groups according to some prescribed set of categories [22]; a process that is both laborious and error-prone. Other researchers use questionnaires or surveys to determine the effects of specific factors such as orientation, cohesion, roles, and climate on teams [14][25][30][47][52][60]. Although such studies suggest areas in which certain techniques might be useful, they lack operational usefulness, since they require that the project be completed before the methods can be applied. These limitations need to be addressed in order to soundly analyze and ultimately impact team performance in real time.

As a result, researchers continue to look for more consistent ways to analyze distributed teams. Recent efforts have resulted in research that has studied more process-oriented behaviors such as the sequential patterns of communication and the flow of communication among team members. Communications among team members can be modeled and analyzed in different ways including content coding and code sequence analysis[36], hidden Markov models [61], classification along psychological or linguistic dimensions [46], and word counts [51]. Similar to the methods described above, these techniques are also labor intensive and applied after the completion of the actual task. Recent advances in information retrieval
and data mining, however, suggest that automated and/or semi-automated text classification algorithms are effective in finding differences in the communication patterns among individuals and groups [41][46][59]. Thus, it seems reasonable to conclude that these same linguistic and data mining techniques might also be used to find effective ways to analyze the activities of global software development teams in real time.

The goal of this thesis, then, is to investigate different automatic and semi-automatic measures of communication and evaluate their effectiveness in classifying global software team behaviors. In order to do this, the research led to the development of a web-based component that can analyze a team’s communication activities. The specific web-based component allows users to prepare text for the classification task; train and then apply a particular machine learning technique to uncoded text; and compare the results from several machine learning algorithms applied to the same text.

The context for this work was a computer-supported collaborative environment in which computer science students from Turkey, Panama, and the US collaborated on large programming projects. A current NSF grant allowed us to capture interaction data from the students who have participated in various team programming projects over the past two years. The contextually rich transcripts of team interactions were cleaned and processed for communication content. A coding system was then applied to the communication content. These coded texts were then used to train and test three computational linguistic techniques (i.e., Naive Bayes Multinomial, Logit Boost, Random Forest). The results from the three linguistic techniques were then compared to determine which method was most effective in predicting overall classification of the group content.

Being able to accurately and quickly tag data text generated from users’ activities allows researchers to use the tags in the ongoing monitoring of a project. The classifications of the tag data can be used to see if the current project is on the right track based on results from previous projects. The information can also be used to compare the ongoing projects with one another to see if there is a particular group that needs help or guidance.

Before describing this research, we provide some background on empirical findings

2
relevant to analysis of communication and team performance.

1.1. Related Work

Because of the critical role that communication plays in a team’s ability to achieve coordinated action, the measurement and analysis of communication behaviors has been an ongoing focus of team research. Team research studies are widespread, with numerous articles appearing in business [49][11], psychology [4][12], medicine [3], computer science [6], and education [62] journals. The most common method for analyzing communication data has focused on low-level quantitative measures, such as duration of communication or time of communication [7][2]. Quantitative measures can often capture a team’s communication flow by modeling the group’s interactions using methods such as frequencies, density [15][34][65], centrality [24][28], lag sequential and/or Markov chains, time series modeling, Fourier analysis [64] or other methods [56].

The second common communication analysis approach concentrates on a team’s content and usually involves creating a coding scheme that represents all the interesting categories of a particular type of communication, such as the rules being displayed in the conversation [20], the types of speech [10], or the actual meaning of the discussion [27]. The transcribed discourse is then divided into the smallest units of meaning, and those pieces of text that correspond to the categories of interest are tagged [20]. The tagged content can be analyzed either as frequency counts of the categories themselves or as a series of events (called “interaction analysis”) [33][50].

Both flow and content approaches have their own merit, and their own costs. Researchers using the content approach must make sure that they use multiple coders and that there is adequate agreement among all the coders. Transcribing and coding text is obviously very labor intensive, but the advantage of this approach is that group processing data are captured, including, in some cases, nonverbal communication [53][16]. Flow data are much easier to collect, but they do not capture the semantics of the communication activities. Both approaches have been widely used to analyze communication among groups in business [38], education [26], military [63], and computer science [21] domains, but few have applied these
techniques to the performance of global software teams.

1.1.1. Automated Text Categorization

Recent advances in the linguistic field have led researchers to consider automated methods for categorizing communication text. A typical text categorization problem begins with a training set of transcripts or documents that have been identified and hand coded into different categories (e.g., good versus bad; easy, medium, hard; red, white, blue, etc.). A machine learning algorithm is then used to find the best set of mappings (i.e., words, phrases, etc.) between the training text document and output categories[18]. The goal of the algorithm is to learn a general set of rules or patterns from training examples that can correctly classify a new set of examples. Support Vector Machines (SVM), K Nearest Neighbor (KNN), and Bayesian learning have all been used to classify documents[58][57].

The data that are used as input to the text classifier algorithms are often represented as a bag-of-words, where the running words are separated from each other with space (that is the tokens, also called unigrams). When vector space models are used as learners (e.g., SVMs), each document is represented as a feature vector, where each feature is associated with a particular token in the document. A feature is usually weighted by counting the term frequency or by using some sort of weighting scheme (e.g., a term frequency-inverse document frequency (TF-IDF) value). In addition, the tokens can be stemmed, and/or analyzed on different linguistic levels: lexical, morphological, syntactic or semantic [57]. More complex feature representations (e.g., bigrams [8], part-of-speech [1], complex nominals [45], noun phrase chunks [31], and extracted keywords [35][40][43]) can also be used in combination with or as an alternative to the bag-of-words representation. In some studies, alternative input features are selected and prediction models trained on different representations are combined [55]. Recently, research on automatic domain adaptation has also been used[37].

One of the more relevant studies for this research is the Taghelper software application [54]. The system was developed to automate the coding of texts generated from groups of students doing math problems. The system initially used machine learning algorithms that were developed by programmers for the Minorthird text-learning toolkit[54]. Later versions
of Taghelp relied on machine learning programs from Weka, a collection of machine learning algorithms for data mining. The Taghelper interface allows users to train a model using pre-coded text from students’ group communication. Once the model is built, the user can then run Taghelper and select different machine learning algorithms to automatically tag new, uncoded text. Taghelper has also been used to find questions and answer pairs in forum data [9] using POS tags and searching for common patterns.
CHAPTER 2

METHODOLOGY

The ideas were synthesized from the significant theoretical constructs identified in the literature presented in the previous section. An analysis of this literature became the guiding tool for the software development and the experimental research that is described in this chapter.

2.1. Data

Data for this study was obtained from four global software development student projects comprised of students from the US, Turkey, and Panama. The students who participated in the global software engineering projects were generally junior or senior computer science or information technology students who had completed both an introductory and advanced programming course. Students enrolled in these courses were randomly assigned to groups and asked to design, code and test a large programming project. All communication among the groups was done through a project management system. If students used another system such as Skype they were asked to upload the data to the project management system. Students used primarily English to communicate with one another. The total time for each project tended to range between two and six weeks.

The actual projects that were assigned to students who participated in the study all related to the creation and querying of a database. For example, one project consisted of an assignment to design, create and query a database that could maintain a fleet of rental cars. Another project asked students to produce a database that could manage items contained in a bookstore. The students who participated in the projects were asked to act as members of a global software development team that was given the responsibility of developing and designing the different data management systems. Each student was provided a summary of the particular case and any background information about the project. All assignments required students to deliver both design and code as part of their final product.

After the projects were completed, the data was imported into a database in a generic
format to allow for easy processing. At that point, the data was cleaned and stored in another table to allow for easy comparison between the raw and cleaned data. Since much of the data was obtained from forums, which generally consist of multiple sentences from a single user, there was a need to divide longer text segments into individual sentences. Therefore, a separate table was created to store the individual sentence data, along with a key that related the sentence back to their original source.

2.2. The Coding Scheme

The specific text classification system that was used to train and subsequently compare our automated classifiers consisted of a five category system that looked at the collaborative behaviors of: planning, contributing, seeking input, reflection, and socializing. Since the particular communications of the global software development projects were aimed at trying to characterize the group dynamics that occurred within distributed software teams, we chose a coding scheme that tries to characterize student group’s collaborative behaviors [32]. Curtis and Lawson [32] identify nine different behaviors (described in Johnson & Johnson [23]) as being supportive of the collaborative process, and then developed a coding schema that could be used to categorize different utterances in on-line collaboration.

The final Curtis and Lawson coding scheme consists of 15 separate communication behaviors that are collapsed into 5 behavior categories. The behavior categories and the individual behaviors are given in Table 2.1.

The original Curtis/Lawson [32] coding scheme was based on an exploratory study that examined online discussions of students who were engaged in a collaborative task. Curtis and Lawson classified statements that related to organizing work, initiating activities, and group skills under a Planning category. Communications related to the utterances such as giving help, providing feedback, exchanging resources, sharing knowledge, challenging others or explaining one’s position were classified in the Contributing category. Other collaborative behaviors were also noted such as Seeking Input and Reflection. Conversations about social matters that were unrelated to the group task were placed in the Social Interaction category.

The training set for the automated classifying system was created by applying the
<table>
<thead>
<tr>
<th>Behavior Categories</th>
<th>Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>Group Skills, GS</td>
</tr>
<tr>
<td></td>
<td>Organizing work, OW</td>
</tr>
<tr>
<td></td>
<td>Initiating Activities, IA</td>
</tr>
<tr>
<td>Contributing</td>
<td>Help Giving, HeG</td>
</tr>
<tr>
<td></td>
<td>Feedback Giving, FBG</td>
</tr>
<tr>
<td></td>
<td>Exchanging Resources and Information, RI</td>
</tr>
<tr>
<td></td>
<td>Sharing Knowledge, SK</td>
</tr>
<tr>
<td></td>
<td>challenging others, Ch</td>
</tr>
<tr>
<td></td>
<td>Explaining or elaborating, Ex</td>
</tr>
<tr>
<td>Seeking Input</td>
<td>Help Seeking, HeS</td>
</tr>
<tr>
<td></td>
<td>Feedback Seeking, FBS</td>
</tr>
<tr>
<td></td>
<td>Advocating Effort, Ef</td>
</tr>
<tr>
<td>Reflection/Monitoring</td>
<td>Monitoring Group Effort, ME</td>
</tr>
<tr>
<td></td>
<td>Reflecting on medium, RM</td>
</tr>
<tr>
<td>Social Interaction</td>
<td>Social Interaction, SI</td>
</tr>
</tbody>
</table>

| **Table 2.1. Coding Scheme and Communication Behavior Categories** [17] |

Curtis/Lawson classification code to the content of the students’ email messages and forum discussions that were stored in the database using the Curtis/Lawson coding system. The system that was used to facilitate both the training and automated classification of the experimental text now follows.

2.3. Overview of the System

The automated text classification application resides within a web-based component that can access several existing specialized program libraries. The web interface allows the user to clean the data using various programs that remove things such as wiki markup and html tags. The interface relies on the browser’s capabilities for spell-checking the text fields. The web interface also provides access to several open-source programs such as the OpenNLP
library from apache http://opennlp.apache.org/, which is used to tokenize the text and the Lucene Snowball Stemmer from Apache Lucene http://lucene.apache.org/core/, which is used to stem the words. The Weka package, a collection of various machine learning programs http://www.cs.waikato.ac.nz/ml/weka/, is used to do the actual classification of the encoded text. This is accomplished by converting the data set into Weka Instance(s). The machine learning classifiers from Weka are called from within the web interface. Once the classifier is trained, it can be used to automatically tag text from a new project. Results from different classifiers can then be compared using the Classification Review option that is accessed from the web interface.

The following sections describe the web-based component and process that is used to automate the coding of text from the global software development projects.

2.4. Architecture

The web-based application is made up of small libraries that perform the auto tagging, cleaning, and data set abstraction (see Figure 2.1). The auto tagging is made up of two modules. One module is written in Clojure and is used to create the data set in a format that is easier to manipulate. The second library is also written in Clojure and is a wrapper for Weka and is designed to make the data sets easier to work with the Weka classifiers. The text cleaning library uses Apache OpenNLP for tokenizing the text, but it also contains a set of functions, written by the author, that help with the cleaning operations and are also used to configure the OpenNLP library. The data set library has a simple data format that uses the UJMP http://ujmp.org library for storing the data set in a matrix; the properties of this matrix are stored in a map. The auto tagger combines all of these libraries in order to classify text. The graphical front end calls the autotagger library for classifying text and building the classifiers. When building a classifier the auto tagger returns all the data that is used to classify a document. The auto tagger library is not dependent on the front end web application or how the text is stored. To build a classifier with a set of data, each instance must implement an Interface that has a function called classification, which returns the classification for the text.
2.5. Cleaning the Text

The classification process generally begins by cleaning the data. The web-based interface provides the user with the ability to select a specific project from the list of completed projects and clean the text data for that project. Once a project has been selected, the interface displays a list of all the communication activities for that project. The user then selects an activity, and the corresponding text appears in the adjacent column (figure 2.2). To edit the text, the user selects the Copy Raw Text menu item and makes any necessary changes such as removing a duplicated response from a forum post, fixing spelling/grammar errors, and removing any wiki or html markup (which can be done automatically by selecting either the Remove Wiki or HTML menu items.)
2.6. Splitting the Text into Sentences

After the text has been cleaned, the user then proceeds to the next step, which is separating the forum or chat postings into single sentences. Although it is possible to build a machine learning model using large pieces of text, it is better to use single sentences for greater accuracy.

Separating the text is done through a separate interface. The user begins by selecting the Create Sentences menu item from the left navigation panel, and then selecting one of the global software development projects from a drop down list at the top of the interface. Similar to the previous process, the interface displays a list of numbers indicating different students’ text inputs. The user then selects an individual communication activity, which is displayed in the adjacent window. At this point, the user has the option of allowing the system to generate sentences (Figure 2.3) or choosing to separate the text manually. If the user chooses to have the system separate the sentences, then the software calls the OpenNLP library sentence tokenizer. The sentences are further processed by detecting multiple punctuation marks, for example . . ., ???, and !!! so that the sentences can be displayed properly. Students from Panama tend to end sentences with three punctuation marks, which is a problem when
trying to detect where a sentence ends. The user can then use the tagsweb application to review results and fix any errors from the sentence tokenization process.

2.7. Text Cleaning

The OpenNLP word tokenizer is used to break up the sentences into words. This process removes all non alpha characters and forces the remaining text into lowercase. The snowball stemmer is then applied to all of the words. Any words that remain blank and all stop words (see Section below) are removed. Once this is done, the words are converted into numbers so they don't conflict with any of Weka’s class attributes.

2.8. Ngrams

Prior to creating a classification model or classifying new text, the text undergoes further pre-processing. For example, two, three, and four ngrams are created by tokenizing the text, applying the Lucene stemmer, and filtering out all non alpha characters. If the ngrams appear more than three times in the text, then they are saved. This procedure was followed because after several trials, we discovered that the removal of ngrams that appear less than three times significantly reduces the number of overall ngrams and provides a higher accuracy then without the ngrams. The ngrams are then converted into numbers to prevent any conflicts with weka.

2.9. Stop Words

A separate program is then used to remove stop words. The program is fairly simple in that it removes words such as for, that, to, too, a, and the. Initially, the words from and when were also removed, but the removal of these two words led to a significant drop in classification performance (e.g., the Naive Bayes Multinomial classifier had a 3% drop).

2.10. Words and Ngrams weights

Log entropy term weighting is applied to the words and ngrams[19]. Log entropy term weighting reduces the weight of words that appear frequently and increases the words that appear infrequently. To use the formula the documents and words must be stored in a word by document matrix. The exact formula that is used to assign the weights is:
\[ W_{ij} = L_{ij} G_i \]

\[ L_{ij} = \log_2 (t f_{ij} + 1) \]

\[ G_i = 1 - E_i \]

\[ E_i = \sum_{j=1}^{m} \frac{p_{ij} \log_2 p_{ij}}{\log_2 m} \]

\[ p_{ij} = \frac{t f_{ij}}{\sum_{j=1}^{m} t f_{ij}} \]

Where \( j \) is the document and \( i \) is the term. \( W_{ij} \) is the weight we use for the word. \( p_{ij} \) is the probability of the word only appearing in document \( j \). \( E_i \) is the entropy of the word appearing in all of the documents. \( L_{ij} \) is the calculation for the word in the document. \( G_i \) is the global weight for the word. To be able to calculate word weights for other documents, the \( G_i \) for each word is stored in a hash map. This particular program was implemented so that the user could easily swap out or substitute a different term weighting schemes.

2.11. Building the Model and Classifying Project Activity

Once all the text for a project has been separated into sentences, the user can then create a training set or learning model by manually tagging a subset of the project’s text. The user does this by selecting the Classify Tags and then the Classify Sentences options from the right-hand navigation window. Similar to the previous procedure, the user must select a project, and then select an activity. The user manually tags the data by selecting one of the categories from the drop down menu at the top of the right hand column (see Figure 2.4). Once the manual tagging has been completed, the model is loaded and the automated classifier is ready to classify a project.

To actually run a classifier on new text, the user selects the Auto Tagger option from the right-hand navigation panel. Once the classifier is finished running the classifications for the project, it displays all the results in an Open for you to Review panel.
If the user selects a classifier for a project that has been previously classified, then the old results are automatically displayed without actually re-running the classifier.

2.11.1. Adding and Updating a Classifier for Training

All the classifiers are stored in list. The list contains the name of the classifier which is displayed to the user, a name for lookup in a hash map, and the options for the classifier. The options for the classifier can be either true/false or a value that can be assigned by the user. To tell the application to generate a textbox or checkbox, a user can set a flag to either true for textbox or false for a checkbox. New classifiers can be added to the system by adding them to the classifier configuration map.

2.12. Classifier Storage

Once a classifier has tagged a new project, the results are stored in the database by serializing the classifier using the Java Serializable interface. The MySQL max packet size had to be increased since the classifier size is bigger than the default MySQL max packet.
size. The words and ngrams are stored along with the counts so that the word weights can be recalculated and the classifier can re-assign the tags for different text activities.

2.12.1. Review Classification

The web interface also allows the user to examine individual sentences and compare the results from different classifiers and human taggers. The user first selects the **Review Tags** from the left-hand Navigation window and then the **Classification Sentence** option. The user must again select a specific project from the project list. The system automatically defaults to a display of the results from all the machine learning classifiers as well as human taggers(Figure 2.5). The user can then filter out any user tagged classifications or a specific results from one of the machine learning classifiers as well as select all the sentences in which the automated classifier differed from user tags (Figure 2.5). In the display, each sentence is highlighted in blue. The following line contains the name of the tagged category and all the user(s) and automated classifiers that placed the sentence in the same category. The name of the user who agreed with the classification begins with `usr_` and the names of the automated classifiers start with `cl_`. The third line contains information about users or

**Figure 2.5.** Classifications Review
automated classifiers that disagreed with the classification. Similar to the previous line, it contains the name of the tagged category and the user(s) and classifications that placed the sentence in one of the opposing categories. The panel on the right presents percentages that provide information about the overall level of agreement among the different classifiers and user tagged sentences.
As previously stated, the data for testing the automated classifier system consisted of text obtained from four different global software engineering projects (see Section 2.1). The system was tested using three prominent classifiers: Naive Bayes Multinomial, Logit Boost and Random Forest. Naive Bayes Multinomial classifier is a modification of Naive Bayes designed specifically for document classification[42]. Random forest is a combination of different decision tree algorithms. Each tree is built using a random sample of attributes[5]. LogitBoost uses a weak classifier[29] and applies a boosting method to increase its accuracy. Unlike AdaBoost, LogitBoost can be used with multiple classifications. Each of these classifiers was run by accessing the code that is stored in the Weka machine learning package. All classifiers were run using 5 cross validation folds to estimate the predictive performance of the models.

3.1. Whole Text

The first set of experiments examined only text activities that were classified in a single category, as opposed to several activities. As reported above, many of the text segments were taken from forum postings, which often contain multiple sentences. As a consequence, the human taggers who rated these text activities often used multiple categories to characterize the various text items. Since such a condition could pose problems for the tagger, the data set for this particular experiment consisted of only those activities that were placed into a single category. After removing all multi-category activities, the final data set for this experiment contained 1596 text activities.

The results obtained from the three automated classifiers are shown below in Table 3.1 and Figures 3.1 and 3.2.

Both the Table 3.1 and Figures 3.1 and 3.2 show clearly that the Naive Bayes Multinomial classifier is the best performing classifier for texts that have been characterized by a single category. It also has the lowest false positive rate. Although the Logit Boost and
Figure 3.1. Whole Text True Positive Rate

Figure 3.2. Whole Text False Positive Rate
<table>
<thead>
<tr>
<th>Classification</th>
<th>Number</th>
<th>Naive Bayes Multinomial</th>
<th>Logit Boost</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP %</td>
<td>FP %</td>
<td>TP %</td>
<td>FP %</td>
</tr>
<tr>
<td>Contributing</td>
<td>715</td>
<td>76.5</td>
<td>26.8</td>
<td>88.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>54.9</td>
<td>81.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Planning</td>
<td>206</td>
<td>39.8</td>
<td>0.3</td>
<td>23.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.5</td>
<td>17.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>Social Interaction</td>
<td>331</td>
<td>68.3</td>
<td>5.1</td>
<td>45.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.1</td>
<td>65.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.0</td>
<td></td>
</tr>
<tr>
<td>Seeking Input</td>
<td>314</td>
<td>55.4</td>
<td>12.6</td>
<td>35.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7.3</td>
<td>35.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8.6</td>
<td></td>
</tr>
<tr>
<td>Reflection Monitoring</td>
<td>30.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1596</td>
<td>64.5</td>
<td>16.5</td>
<td>59.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>27.0</td>
<td>59.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22.5</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.1. Classification Whole Text**

Random Forest classifiers have high True Positive (TP) rates for the Contributing category, they both have very high False Positive (FP) for this same category, indicating that they incorrectly classified some text items as being Contributing behaviors when they actually belonged in another category. False Positive rates for the other categories were also fairly high when compared to Naive Bayes Multinomial. All three classifiers had difficulty determining Reflection and Monitoring activities, but that was probably due to the low number of activities in this particular category.

In order to determine if the performance of the classifiers could be improved with the addition of more data, we decided to separate the multi-tagged text segments into single sentences and then re-tag them, which would obviously produce additional training samples of single-category text activities. By separating the text into single sentences, we were able to get an additional 519 training samples.

3.2. Sentences

The sentence data was then used to compare results from the three classifiers: Naive Bayes Multinomial, Random Forest, and Logit Boost. This experiment ran the classifiers over all of the tags with 5 cross validation folds. Results from this experiment are displayed in Tables 3.1 and Figures 3.3 and 3.4.
Once again, Naive Bayes Multinomial has the highest overall true positive (TP) rates and the lowest false positive (FP) rates as compared to both the Logit Boost and Random Forest classifiers. Again, both Logit Boost and Random Forest have higher true positives (TP) for the contributing activities, but also have high false positives for this same category. It appears that the high FP for the contributing category may be caused by both classifiers having overfitted the data for this category. This overfitting may have come at the expense of correctly categorizing Planning behaviors. For example, both Logit Boost and the Random Forest classifiers have low TPs in the Planning categories, as compared to the Naive Bayes classifier.

Reliability between human and automated taggers was evaluated using a pairwise Kappa measure. Mean Kappas for the three automated taggers on all-text data were .38 for Logit Boost, .32 for Random Forest, and .47 for Naive Bayes. In contrast, mean Kappas for the sentence data were .43 for Logit Boost, .44 for Random Forest, and .56 for Naive Bayes (Figure 3.5). Thus, by separating the texts into single sentences, we were able to achieve higher agreement between human and automated taggers.

The text was also converted into ngrams, and was again used to test the accuracy
Table 3.2. Classification Review Sentences

<table>
<thead>
<tr>
<th>Classification</th>
<th>Number</th>
<th>Naive Bayes Multinomial</th>
<th>Logit Boost</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>TP %</td>
<td>FP %</td>
<td>TP %</td>
</tr>
<tr>
<td>Contributing</td>
<td>972</td>
<td>79.3</td>
<td>27.5</td>
<td>89.3</td>
</tr>
<tr>
<td>Planning</td>
<td>202</td>
<td>52.5</td>
<td>5.2</td>
<td>24.3</td>
</tr>
<tr>
<td>Social Interaction</td>
<td>348</td>
<td>74.7</td>
<td>4.0</td>
<td>49.7</td>
</tr>
<tr>
<td>Seeking Input</td>
<td>427</td>
<td>62.1</td>
<td>8.3</td>
<td>49.7</td>
</tr>
<tr>
<td>Reflection Monitoring</td>
<td>66</td>
<td>25.8</td>
<td>FP 0.8</td>
<td>10.6</td>
</tr>
<tr>
<td>Total</td>
<td>2015</td>
<td>70.4</td>
<td>16.3</td>
<td>65.4</td>
</tr>
</tbody>
</table>

of the classifiers. When using all of the generated ngrams, Naive Bayes Multinomial was able to obtain a 58% accuracy rate. Unfortunately, the ngram conversion also led to an increase in the classifier’s processing time (the program took over 16 hours to complete its run). Although the addition of the ngrams causes an increase in the performance in the
tagging of both seeking input and reflection/monitoring activities, it also results in decreased performance for the tagging activities in the planning category. One explanation for this decrease in performance may be due to the overfitting of the model for the Contributing category because of the high number of text activities that tend to be assigned to this particular category.

The ngrams were not tested on either the Logit Boost or Random Forest classifiers because it was determined that the processing time would be excessive. Also, since the addition of ngrams did not result in any significant increase in accuracy for the Naive Bayes classifier, it was decided to forgo any ngram comparisons with Logit Boost or the Random Forest classifiers. However, the Bayes Multinomial with the reduced ngrams takes only a few minutes to run and has a 12% higher accuracy.
CHAPTER 4

CONCLUSIONS

The major goal of this research was to develop a tool that could assist with the automated tagging of text from groups engaged in a global software development project. The motivation for such a goal was to determine whether automated tagging tools could effectively categorize student group communication. The reason for wanting to categorize student communication is that it leads to a better understanding of how groups work in teams. Other researchers have attempted to solve the automated tagging problem, but these systems are generally stand alone and are not easily accessible over the Internet[48]. This and other work provided insights for this thesis by revealing what type of automated tagger is more useful for the classification problem and how to better structure a web-based interface to accommodate user needs. As a result, this project was begun in order to develop a web-based interface that could be used for automating the classification of global software student team communication. The software allows users to select text from various student team projects, classify that text, and then use the classify text for training the classifiers and automating the tagging process. In addition, the interface allows users to view different classifications and compare the results of human and automated taggers. The interface also contains programs and functions for cleaning text and separating longer text segments into single sentences. It was initially thought that the automated system could achieve an 80% tagging accuracy, given a substantial training sample of student messages. Therefore, the idea for this study was that automated text categorization would be both more efficient and more effective than human tagging. In order to test these ideas, data was selected from four different global software development student projects and processed by the system. Data from all three projects were aggregated and categorized by both whole-text segments as well as individual text statements. Three different machine learning algorithms were then tested using this data to determine the accuracy of the different learning algorithms.
4.1. Findings

Although the overall accuracy of the three machine learning algorithms was relatively similar for both the all-text and single sentence condition, the Naive Bayes Multinomial outperformed both the Random Forest and Logit Boost algorithms. The Naive Bayes Multinomial classifier had higher Truth Positive and lower False Positive rates than either the Random Forest or Logit Boost methods, except for text classified as Contributing. Even for this category, Naive Bayes Multinomial had lower False Positive rates than the other two classifiers. Moreover, Naive Bayes Multinomial’s performance was better on both the all-text and single sentence condition.

All machine learning classifiers performed better on the single sentence condition as compared to the all-text condition. This increased performance was partly due to the increase in the number of training examples that occurred as a result of separating the text. Another reason for the improved performance was that the single sentence condition allowed both the human and automated classifiers to be more precise about the exact nature of the text and how that text should be characterized. In other words, there was less ambiguity in the meaning of the text.

Unfortunately, the agreement among human and automated classifiers was only around .5 for the three different classifiers. Although the Naive Bayes Multinomial had the highest rate of agreement between the automated and human classifications, it was able to agree on only half (.57) of the test data. Although similar low Kappa rates can be found in other studies in the area of classifying group communication (rose), the use of multiple classifiers had led this researcher to expect more agreement between human and automated taggers. One of the reasons for the low Kappa rate may have been the over abundance of training examples for the Contributing category. Since the training set for the Contributing category was so large, as compared to the other categories, it may have been difficult for the classifiers to find unique sets of words/ephrases to distinguish text found in the other categories from that found in the Contributing category.
4.2. Conclusions

In previous research, it was found that machine learning technology was able to help with the labor intensive process of classifying the content of text generated in a computer supported collaborative learning setting\[54]\[53]. These findings prompted this researcher to develop a web-based interface that can automate the classification process of the text that is generated by groups participating in a global software development student learning project. Although human coders are often used to categorize different content generated from group discussions, this process is often tedious and can often lead to miss-classifications among different human coders. Thus, this system demonstrates that it is possible to develop a system that can classify text similar to human coders and do so much more efficiently than human coders. While this finding is clearly not an argument for the main goal of the study, it does tend to suggest that an automated classification system can classify the group content, and that this classification process is more efficient than human coders. We are able to achieve about 70% accuracy with the Naive Bayes Multinomial using ngrams, and it is done in less time than it would take a human coder.

The study, however, failed to provide any major evidence that the classification process was superior to human coding. While the Bayes Multinomial was able to achieve a 57% agreement rate, it was still less than what was anticipated. Part of the issue with the low agreement rates may be related to the fact that there was an unequal representation of the Contributing category, which may have cause the classifiers to overfit the classifications in this area. However, this researcher believes this thesis makes a contribution by showing how simple automated tools can lead to a better understanding of group communication and activities. The contribution of this particular study has implications for software development teams in the work force, and for project teams in general.

Although we were unable to fully automate the tagging of communication activities (i.e., the cleaning of the data), we were able to show that an automated tagging system can achieve about 70% accuracy with the Naive Bayes Multinomial using ngrams.
4.3. Future Work

Although this thesis was able to show that it was possible to create an interface that could automatically classify text from student projects, it was not able to show that the automated classifiers achieve high performance. Although potential areas of research were explored during the development of the tools, they were deemed outside the scope of this particular project. These additional areas of research are now outlined below.

Future studies should look at how to increase the accuracy of our classifiers. Most classification studies indicate that an 80% accuracy rate is a gold standard that people should try to achieve when classifying text. These higher performance rates might be achieved by using an algorithm similar to a context aware spell checker that would look for frequent patterns that appear in the categories. While the current set of data was too small for the use of this type of algorithm, future work may be able to generate more sample data sets.

Another method that might be used to increase the accuracy rates is a question detection method, as described in “Finding Question-Answer Pairs from Online Forums”[9]. Since the system doesn’t know which words to keep from a phrase after a Part of Speech (POS) tagger has completed its process, a genetic algorithm might be used to determine what should be kept. By using this method, the system should be able to increase the number of ngram matches, which could also lead to a decrease in the false positive rates for the Contributing category and an increase in the accuracy for the remaining categories. Compared to the context aware spell checker algorithm, this method requires a much smaller data set so it could be implemented very quickly using the current sample data.

Finally, future work should investigate the possibility of using a combination of classifiers followed using a voting scheme to select an appropriate classification for a given text. However, since the results from the different classifiers were relatively close, it is unclear if such a method would lead to higher accuracy rates.

4.4. Summary

This research was intended to test the feasibility of automating the classification process of text generated from global software student project. In order to accomplish this
goal, this researcher developed a web-based application that allows users to select different projects and create training sets for classification tasks. Moreover, the interface supports the data cleaning and sentence separation tasks. Once the data has been prepared, the user can then run various classifiers to automatically classify new data. Finally, the interface displays results from the automated classifiers as well as a sentence by sentence comparison between human and automated classifiers.

Because we were also interested in the performance of the classifiers, we compared three different classifiers on a classification task. Results from this experiment indicate that an automated tagging system can achieve about 70% accuracy with the Naive Bayes Multinomial using ngrams. Future work will examine different methods that can lead to higher accuracy rates and better performance.


[10] Noshir S Contractor and Susan J Grant, *The Emergence of Shared Interpretations in*


[27] Blaz Fortuna, Eduarda Mendes Rodrigues, and Natasa Milic-Frayling, Improving the classification of newsgroup messages through social network analysis, Training (2007), 877.


[31] J Furnkranz, T Mitchell, and E Riloff, A Case Study in Using Linguistic Phrases for


[38] Sara Kiesler and Lee Sproull, Group decision making and communication technology, Organizational Behavior and Human Decision Processes 52 (1992), no. 1, 96–123.

[39] Paul Layzell, O P Brereton, and Andrew French, Supporting collaboration in distributed software engineering teams, Software Engineering Conference 2000 APSEC 2000 Pro-


[49] Linda Plotnick, Rosalie Ocker, Starr Roxanne Hiltz, and Mary Beth Rosson, *Leadership*


