An Interactive Tool for Supporting Error Analysis for Text Mining

Elijah Mayfield
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15216, USA
elijah@cmu.edu

Carolyn Penstein-Rosé
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15216, USA
cprose@cs.cmu.edu

Abstract

This demo abstract presents an interactive tool for supporting error analysis for text mining, which is situated within the Summarization Integrated Development Environment (SIDE). This freely downloadable tool was designed based on repeated experience teaching text mining over a number of years, and has been successfully tested in that context as a tool for students to use in conjunction with machine learning projects.

1 Introduction

In the past decade, more and more work in the language technologies community has shifted from work on formal, rule-based methods to work involving some form of text categorization or text mining technology. At the same time, use of this technology has expanded; where it was once accessible only to those within studying core language technologies, it is now almost ubiquitous. Papers involving text mining can currently be found even in core social science and humanities conferences.

The authors of this demonstration are involved in regular teaching of an applied machine learning course, which attracts students from virtually every field, including a variety of computer science related fields, the humanities and social sciences, and the arts. In five years of teaching this course, what has emerged is the finding that the hardest skill to impart to students is the ability to do a good error analysis. In response to this issue, the interactive error analysis tool presented here was designed, developed, and successfully tested with students.

In the remainder of this demo abstract, we offer an overview of the development environment that provides the context for this work. We then describe on a conceptual level the error analysis process that the tool seeks to support. Next, we step through the process of conducting an error analysis with the interface. We conclude with some directions for our continued work, based on observation of students' use of this interface.

2 Overview of SIDE

The interactive error analysis interface is situated within an integrated development environment for building summarization systems. Note that the SIDE (Kang et al., 2008) software and comprehensive user's manual are freely available for download[1]. We will first discuss the design of SIDE from a theoretical standpoint, and then explore the details of practical implementation.

2.1 Design Goals

SIDE was designed with the idea that documents, whether they are logs of chat discussions, sets of posts to a discussion board, or notes taken in a course, can be considered relatively unstructured. Nevertheless, when one thinks about their interpretation of a document, or how they would use the information found within a document, then a structure emerges. For example, an argument written in a paper often begins with a thesis statement, followed by supporting points, and finally a conclusion. A reader

[1]SIDE and its documentation are downloadable from `http://www.cs.cmu.edu/~cprose/SIDE.html`

Proceedings of the NAACL HLT 2010: Demonstration Session, pages 25–28,
Los Angeles, California, June 2010. ©2010 Association for Computational Linguistics

can identify with this structure even if there is nothing in the layout of the text that indicates that certain sentences within the argument have a different status from the others. Subtle cues in the language can be used to identify those distinct roles that sentences might play.

Conceptually, then, the use of SIDE proceeds in two main parts. The first part is to construct filters that can impose that structure on the texts to be summarized, to identify the role a sentence is playing in a document; and the second part is constructing specifications of summaries that refer to that structure, such as subsets of extracted text or data visualizations. This demo is primarily concerned with supporting error analysis for text mining. Thus, the first of these two stages will be the primary focus.

This approach to summarization was inspired by the process described in (Teufel and Moens, 2002). That work focused on the summarization of scientific articles to describe a new work in a way which rhetorically situates that work's contribution within the context of related prior work. This is done by first overlaying structure onto the documents to be summarized, categorizing the sentences they contain into one of a number of rhetorical functions. Once this structure is imposed, using the information it provides was shown to increase the quality of generated summaries.

2.2 Building Text Mining Models with SIDE

This demo assumes the user has already interacted with the SIDE text mining interface for model building, including feature extraction and machine learning, to set up a model. Defining this in SIDE terms, to train the system and create a model, the user first has to define a filter. Filters are trained using machine learning technology. Two customization options are available to analysts in this process.

The first and possibly most important is the set of customization options that affect the design of the attribute space. The standard attribute space is set up with one attribute per unique feature - the value corresponds to the number of times that feature occurs in a text. Options include unigrams, bigrams, part-of-speech bigrams, stemming, and stopword removal.

The next step is the selection of the machine learning algorithm that will be used. Dozens of options are made available through the Weka toolkit (Witten and Frank, 2005), although some are more commonly used than others. The three options that are most recommended to analysts beginning work with machine learning are Naïve Bayes (a probabilistic model), SMO (Weka's implementation of Support Vector Machines), and J48, which is one of many Weka implementations of a Decision Tree learner. SMO is considered state-of-the-art for text classification, so we expect that analysts will frequently find that to be the best choice.

As this error analysis tool is built within SIDE, we focus on applications to text mining. However, this tool can also be used on non-text data sets, so long as they are first preprocessed through SIDE. The details of our error analysis approach are not specific to any individual task or machine learning algorithm.

3 High Level View of Error Analysis

In an insightful usage of applied machine learning, a practitioner will design an approach that takes into account what is known about the structure of the data that is being modeled. However, typically, that knowledge is incomplete, and there is thus a good chance that the decisions that are made along the way are suboptimal. When the approach is evaluated, it is possible to determine based on the proportion and types of errors whether the performance is acceptable for the application or not. If it is not, then the practitioner should engage in an error analysis process to determine what is malfunctioning and what could be done to better model the structure in the data.

In well-known machine learning toolkits such as Weka, some information is available about what errors are being made. Predictions can be printed out, to allow a researcher to identify how a document is being classified. One common format for summarizing these predictions is a confusion matrix, usually printed in a format like:

```
  a  b   <-- classified as
 67 19 |  a = PT
 42 70 |  b = DR
```

This lists, for example, that 19 text segments were classified as type DR but were actually type PT. While this gives a rough view of what errors are

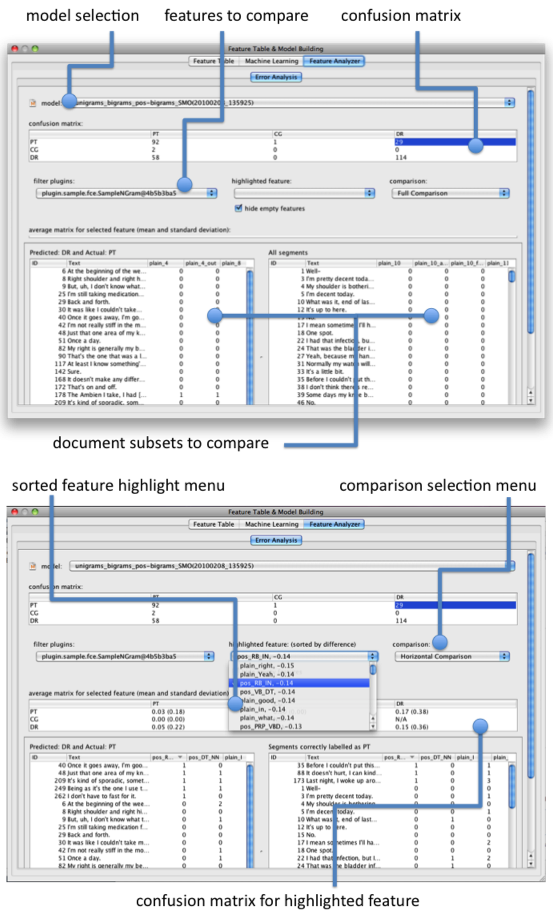
model selection
features to compare
confusion matrix
document subsets to compare
sorted feature highlight menu
comparison selection menu
confusion matrix for highlighted feature

Figure 1: The error analysis interface with key functionality locations highlighted.

appearing, it gives no indication of why the errors are being made. This is where a more extensive error analysis is necessary. Two common ways to approach this question are top down, which starts with a learned model, and bottom up, which starts with the confusion matrix from that model's performance estimate. In the first case, the model is examined to find the attributes that are treated as most important. These are the attributes that have the greatest influence on the predictions made by the learned model, and thus these attributes provide a good starting point. In the second case, the bottom-up case, one first examines the confusion matrix to identify large off-diagonal cells, which represent common confusions. The error analysis for any error cell is then the process of determining relations between three sets of text segments[2] related to that cell.

Within the "classified as DR but actually PT" cell, for instance, error analysis would require finding what makes these examples most different from examples correctly classified as PT, and what makes these examples most similar to those correctly classified as DR. This can be done by identifying attributes that mostly strongly differentiate the first two sets, and attributes most similar between the first and third sets. An ideal approach would combine these two directions.

4 Error Analysis Process

Visitors to this demo will have the opportunity to experiment with the error analysis interface. It will be set up with multiple data sets and previously trained text mining models. These models can first be examined from the model building window, which contains information such as:

- Global feature collection, listing all features that were included in the trained model.
- Cross-validation statistics, including variance and kappa statistics, the confusion matrix and other general information.
- Weights or other appropriate information for the text mining model that was trained.

By moving to the error analysis interface, the user can explore a model more deeply. The first step is to select a model to examine. By default, all text segments that were evaluated in cross-validation display in a scrolling list in the bottom right corner of the window. Each row contains the text within a segment, and the associated feature vector. Users will first be asked to examine this data to understand the magnitude of the error analysis challenge.

Clicking on a cell in the confusion matrix (at the top of the screen) will fill the scrolling list at the bottom left corner of the screen with the classified segments that fall in that cell. A comparison chooser dropdown menu gives three analysis options - full, horizontal, and vertical. By default, full comparison

[2]Our interface assumes that the input text has been segmented already; depending on the task involved, these segments may correspond to a sentence, a paragraph, or even an entire document.

is selected, and shows all text segments used in training. The two additional modes of comparison allow some insight into what features are most representative of the subset of segments in that cell, compared to the correct predictions aligned with that cell (either vertically or horizontally within the confusion matrix). By switching to horizontal comparison, the scrolling list on the right changes to display only text segments that fall in the cell which is along the confusion matrix diagonal and horizontal to the selected cell. Switching to vertical comparison changes this list to display segments categorized in the cell which is along the diagonal and vertically aligned with the selected error cell.

Once a comparison method is selected, there is a feature highlighting dropdown menu which is of use. The contents in this menu are sorted by degree of difference between the segments in the two lists at the bottom of the screen. This means, for a horizontal comparison, that features at the top of this list are the most different between the two cells (this difference is displayed in the menu). We compute this difference by the difference in expected (average) value for that feature between the two sets. In a vertical comparison, features are ranked by similarity, instead of difference. Once a feature is selected from this menu, two significant changes are made. The first is that a second confusion matrix appears, giving the confusion matrix values (mean and standard deviation) for the highlighted feature. The second is that the two segment lists are sorted according to the feature being highlighted.

User interface design elements were important in this design process. One option available to users is the ability to "hide empty features," which removes features which did not occur at all in one or both of the sets being studied. This allows the user to focus on features which are most likely to be causing a significant change in a classifier's performance. It is also clear that the number of different subsets of classified segments can become very confusing, especially when comparing various types of error in one session. To combat this, the labels on the lists and menus will change to reflect some of this information. For instance, the left-hand panel gives the predicted and actual labels of the segments you have highlighted, while the right-hand panel is labelled with the name of the category of correct prediction you are comparing against. The feature highlighting dropdown menu also changes to reflect similar information about the type of comparison being made.

5 Future Directions

This error analysis tool has been used in the text mining unit for an Applied Machine Learning course with approximately 30 students. In contrast to previous semesters where the tool was not available to support error analysis, the instructor noticed that many more students were able to begin surpassing shallow observations, instead forming hypotheses about where the weaknesses in a model are, and what might be done to improve performance.

Based on our observations, however, the error analysis support could still be improved by directing users towards features that not only point to differences and similarities between different subsets of instances, but also to more information about how features are being used in the trained model. This can be implemented either in algorithm-specific ways (such as displaying the weight of features in an SVM model) or in more generalizable formats, for instance, through information gain. Investigating how to score these general aspects, and presenting this information in an intuitive way, are directions for our continued development of this tool.

Acknowledgements

This research was supported by NSF Grant DRL-0835426.

References

Moonyoung Kang, Sourish Chaudhuri, Mahesh Joshi, and Carolyn Penstein-Rosé 2008. *SIDE: The Summarization Integrated Development Environment*. Proceedings of the Association for Computational Linguistics, Demo Abstracts.

Simone Teufel and Marc Moens 2002. *Summarizing Scientific Articles: Experiments with Relevance and Rhetorical Status.* Computational Linguistics, Vol. 28, No. 1.

Ian Witten and Eibe Frank 2005. *Data Mining: Practical Machine Learning Tools and Techniques*, second edition. Elsevier: San Fransisco.