Visualizing the Performance of Computational Linguistics Algorithms

Stephen G. Eick*
SSS Research, Inc.

Justin Mauger†
SAIC Advanced Systems & Concepts

Alan Ratner‡
National Security Agency

ABSTRACT[1]

We have built a visualization system and analysis portal for evaluating the performance of computational linguistics algorithms. Our system focuses on algorithms that classify and cluster documents by assigning weights to words and scoring each document against high dimensional reference concept vectors. The visualization and algorithm analysis techniques include Confusion Matrices, ROC Curves, Document Visualizations showing word importance, and Interactive Reports. One of the unique aspects of our system is that the visualizations are thin-client web-based components built using SVG visualization components.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Screen Design; H.3.4 [Information Storage and Retrieval]: Systems and Software—Performance Evaluation; [H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—Linguistic Processing; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Clustering

Keywords: AJAX, thin-client, SVG, ROC curves, confusion matrices, document categorization

1 INTRODUCTION

One of the current challenges in computer science is developing new methods to represent, structure, analyze, and automatically process unstructured multilingual text to obtain semantic understanding. To aid with this task, we have built an analysis system and portal that helps researchers quantify the performance of their algorithms. Our target users are developing hardware-accelerated algorithms that are described in other papers, see for example [1], [2], [3], and [4].

The goals for our analysis system, called AFEWeb, are to provide unambiguous performance metrics for algorithm developers, develop visualizations that show key algorithm abstractions, and create new visualizations that show why documents are identified as belonging to specific concepts of interest. Our system compares the performance of various algorithms, helps our algorithm designers set thresholds, provides simple and intuitive access to misclassified documents, and displays key algorithmic concepts to users and developers in an understandable way. It cross-references all of the words seen, it is fully interactive, and displays in a standard web browser.

AFEWeb includes many standard document algorithm analysis tools such as Confusion Matrices, ROC curves, Precision and Recall plots, and Interactive Reports. It also includes new visualizations showing document clusters, and visualizations of the words in the documents themselves. Although, some of the analysis tools are known, what is new is to have them packaged together as an analytical system within a web-based document analysis portal. Furthermore, some of our visualizations are novel and particularly well-suited to document analysis. Our interactive portal is surprisingly useful. Using our portal we have identified several algorithm bugs and algorithm performance characteristics that have influenced the developers and our corpus collection techniques.

Our portal is written using AJAX[2] , SVG[3], and other Web 2.0 programming techniques. Using these techniques, it is possible to develop rich interactive user visualizations that are totally browser-based. Our portal combines the utility of rich desktop analysis systems with the flexibility and linking of web-based systems. It works remarkably well and demonstrates that is possible to build powerful visualization tools in a thin-client browser platform.

In the remainder of this paper we describe our system in more detail. First, however, we review key concepts for processing unstructured information using the constraints imposed by our hardware testbed as a concrete example. In subsequent sections we will describe AFEWeb's analysis tools and interactive portal.

2 COMPUTATIONAL LINGUISTIC BACKGROUND

This section briefly reviews some of the key ideas from computation linguistics and relates them back to our test environment. Although our test environment is somewhat constrained because of limitations in our hardware, our analysis and visualization techniques are broadly applicable.

2.1 Feature Vector Representations of Documents and Concepts

Our focus is on text analysis algorithms that use feature vectors to represent documents, as in [6]. We have mainly used centroid-based models (i.e. Rocchio, TFIDF [5]). In our particular implementation, the dimensions of the vector corresponding to a

*eick@sss-research.com
†maugerj@saic.com
‡asratne@nsa.gov

[1] This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract number MDA972-03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

[2] http://en.wikipedia.org/wiki/AJAX

[3] Scalable Vector Graphics, a W3C XML standard for vector graphics.

IEEE Symposium on Visual Analytics Science and Technology 2006
October 31 - November 2, Baltimore, MD, USA
1-4244-0592-0/06/$20.00 © 2006 IEEE

document are equivalence classes of words and the coefficients are the number of occurrences of words in each class. The hardware is designed to store 4000 4-bit counters per document vector. As an example, consider the document shown in Table 1 and word equivalence classes shown in Table 2. There are seven stop words that are assigned to dimension 0 (stop words), two football words assigned to dimension 1 (football concept), one compensation word assigned to dimension 2 (compensation concept), and one circus word assigned to dimension 4 (circus concept). The document vector representing the document in using the semantic dimensions in Table 2 is: $\vec{d} = (7, 2, 1, 0, 1, 0, ..., 0)$.

Table 1. A sample document

The NFL of the modern era is a high flying circus act of elite athletes who make big salaries and big hits.

Table 2. Semantic dimensions for a corpus

Dimension	Equivalence Classes of Words
0	a, are, and, if, is, of, was, …
1	athletes, NFL, football, sports, pass, downfield, running, passing, points, …
2	bonus, compensation, options, salary, …
3	Gemini, Taurus, Virgo, Libra, Aquarius, Pisces, Capricorn, Sagittarius, Scorpio, ...
4	circus, flying, tiger, trapeze, ...
5	backwards, forward, sideways, left, right, behind, ahead, …
…	...
3999	dog, cat, rabbit, fish, turtle,

A concept vector $\vec{c} = (c_0, \cdots, c_{3999})$ is a weighting of the semantic dimensions that represents the content in a particular class of documents. The counters for the concept vectors are 8 bits wide, and thus have a maximum value of 255. So, for example, a concept vector representing the concept of *football* might be $\vec{c}_{football} = (0, 255, 175, 0, 0, 0, \cdots, 0)$

$\vec{c}_{football}$ assigns no weight to stop words, maximum weight to the words in dimension 1, medium weight to financial words in dimension 2, no weight to the horoscope words in dimension 3 and no weight to the other dimensions.

2.2 Document Classification

The *raw score* $ns_{football}$ of $\vec{d}$ with respect to $\vec{c}_{football}$ is the dot product of these two vectors:

$$s_{football} = \vec{d} \circ \vec{c}_{footbal} = 685$$

Since both the size of the concept vectors and size of the document may vary widely, we use the *normalized score* $ns_{football}$ (cosine distance) as the measure of how close document $\vec{d}$ is to concept $\vec{c}_{football}$:

$$ns_{football} = \frac{\vec{d} \circ \vec{c}_{footbal}}{\left|\vec{d}\right| \cdot \left|\vec{c}_{football}\right|} = \frac{685}{\sqrt{55}\sqrt{9565}} = .2986$$

A document is scored against all the concept vectors to determine how closely it matches each concept and is tentatively assigned to the concept that it best matches. It is then "rejected" as noise if its score statistic is less than a concept threshold t_i: $\vec{d}$ is assigned to concept i if $ns_i = \max_{1 \le j \le m}\{ns_j\}$ and $ns_i \ge t_i$, where t_i is the *threshold* for concept *i*, and *m* is the number of concept vectors. If $ns_i < t_i$, then $\vec{d}$ is rejected as none of the known concepts. One of the challenges for our visualization software is to set the thresholds $\{t_i\}$.

3 Corpus Organization

To evaluate our algorithms we organize documents in our corpus into three categories labeled *signal, interference,* and *noise*. *Signal* documents are message documents that belong to the concepts that our algorithms are trained on. *Interference* documents belong to concepts that are not exposed to our algorithms during training. *Noise* documents are not message documents but documents that include executables, images, audio and video. The challenge for our algorithms is to classify the *signal* documents into the correct concepts, discover and group the *interference* documents, and reject the *noise* documents.

To illustrate our approach, the table below shows one of our corpora derived from Google newsgroups. It consists of seven *signal* groups, four *interference* groups, and a 1,000 file *noise* group consisting of GIF and JPEG files with sizes ranging uniformly from 250 to 5000 bytes.

Table 3. Corpus organized into *training* and *testing* documents

Signal Concepts	Training Documents	Testing Documents
baseball	21	43
equestrian	27	54
frugal	10	22
neural_net	15	32
programming	31	62
wagner	13	40
writing	24	49

Interference Concepts	Training Documents	Testing Documents
archeology	0	138
libraries	0	32
logic	0	60
marital_arts	0	56

Noise	Training Documents	Testing Documents
Noise	0	1000

4 Visualizing Algorithm Performance

This section describes our analysis methods in detail.

4.1 Measuring Overall Algorithm Performance – Aggregate Statistics

For each experiment, the system calculates a variety of global statistics including Classification Accuracy, Rejection Accuracy, Total Accuracy, Total Precision, Total Recall, and F1-Measure. These measures are well known in the information retrieval community.

4.2 Interactive Confusion Matrices – Showing Classification Mistakes

A confusion matrix organizes documents into cells with the document labels (true concepts) along the rows and the assigned concepts as columns. The number in each cell is the number of documents either correctly classified (diagonal entries) or incorrectly classified (off diagonals). In our implementation, as shown in Figure 1, we display *Precision, Recall* and *F1-measure* for each signal concept. We also display *Rejection Accuracy* for each interference concept and noise. The first seven rows are the *signal* categories, the next four are the *interference* categories, and the last row represents the *noise* documents.

In this example our algorithm had trouble with the *frugal* concept and confused *frugal* documents with *equestrian* documents. Our *frugal* threshold setting may be too high as 10 of the 22 *frugal* documents were rejected as noise. (Our algorithm also had trouble rejecting the *libraries* interference category.)

Although confusion matrices are well known, there are three interesting aspects to ours. First, our matrix is interactive and runs in a thin-client web browser. Second, as described above, we exploit the browser's ability to hyperlink to tie various pieces of an analysis together. For example, clicking on the highlighted cell corresponding to the five misclassified *frugal* documents launches an interactive report. This report lists the documents that make up the counts in that cell and provides an interface to read these documents to see why they are misclassified. Third, the rejection rates in the confusion matrix depend on the current value of each concept's rejection threshold. Users may manipulate this threshold using our ROC visualizations, which we describe next.

4.3 Setting Concept Thresholds using ROC Curves

A standard approach for evaluating the performance of a classification algorithm with a single test statistic is the use of ROC curves. For each concept, a ROC curve shows the true positive and (log) false positive rates corresponding to each possible threshold value. It shows the tradeoffs among correctly classified documents, incorrectly classified documents, and rejected documents.

Figure 2 shows our implementation of ROC curves that are tied to our confusion matrix. One of the interesting usability aspects of the web page shown in Figure 2 is that it is fully interactive. As a user manipulates the thresholds by moving the vertical line in each ROC display the ROC curves, confusion matrix, and global statistics update continuously. According to human factors guidelines, achieving smooth updating requires that the display update at least every 100 milliseconds. The round trip access time for server calculations is typically around a couple seconds and thus it is not possible to access any server resources and achieve 100 millisecond responses. All of the calculations are done within the browser to provide a real-time response to user manipulations.

Although it would not be difficult to recompute the confusion matrix on a server, it is not possible to recalculate it on a desktop personal computer. The problem is that a generic personal computer is not powerful enough to meet our 100-millisecond response goal. To get around this problem and still provide real-time response, we use instead a lookup table. We pre-calculate all of the possible change points for each cell in the confusion matrix and download the relevant tables asynchronously into the user browser. When the user updates a threshold, rather than recompute the cell the browser performs a binary search and table lookup. Using this technique it is possible to provide real-time browser-based display updating even on inexpensive PCs. One potential problem with this approach is the startup delay in downloading of the lookup tables. It would be distracting if the computer froze for a few seconds while the display initialized itself. To overcome this problem we use a new style of web programming called AJAX that employs asynchronous downloads.

4.4 Visualizing Words within Documents

A question of interest to algorithm developers is determining why particular documents are assigned to concepts. The reason, of course, is that certain words in the document are relevant to the semantic dimensions that are weighted by the concept. To present this information in a useful way we have created a document visualization that shows the importance of each word to the relevant concept. Our technique uses both the font size of each word and places a superscript next to each word to show the respective weight that each word receives for the selected concept vector. So, for example in Figure 3, we show the document with the highest *frugal* score. The *frugal* concept assigns a high weight, 255, to *rent, rental,* and *ownership* and less weight, 56, to *advantages* (clipped in the figure).

There are three pleasing features of this visual representation. The first, and most obvious, is that important words for a concept are large and stand out visually. A user's attention is drawn to the important words. The second feature involves the user's ability to recalibrate the display by selecting a different concept. This makes it easy to see how well a document might match another concept. The third feature involves new words. Any word not observed in the training documents is colored green. For example, the name of the individual making this posting, *Shawn Hearn,* is colored green since it was not seen in the training documents. Although it does not occur here, in some documents there are green words with superscripts. This would indicate that a new word not observed in the training corpus is contributing to a document's assignment to a concept. The reason for this is a hash collision in the hardware lookup tables.

4.5 Visualizing Related Groups of Documents

One of the goals of our system is to group related documents. For example, in the corpus shown in Table 3, an ideal clustering would identify 12 unique clusters. Seven of the clusters would contain each of the *signal* concepts, four would contain each of the *interference* concepts, and one would contain the *noise* documents. Thus a perfect clustering would uniquely identify and group all of the documents in the *signal*, *interference,* and *noise* concepts, respectively. This is rarely possible.

Figure 4 shows a visualization for flat clustering algorithms. It resembles a confusion matrix, with a couple of significant differences. The clusters are shown as rows and columns contain the number of document in the cluster for each of the concepts. Each cluster is labelled according to its most popular concept. That is a cluster is assigned the label of the concept that has the most documents in that cluster. A perfect clustering would have a diagonal structure with all of the documents in a cluster belonging to a single concept. The clustering shown in Figure 4 is adequate since it is somewhat close to diagonal, although there are many

interference clusters which should be combined[4]. The bars on the right display the size of each cluster. As before, each cell and cluster is hyperlinked to reports listing the relevant documents.

5 AFEWeb in Use

Although we have not conducted any formal user studies, our experience using AFEWeb to analyze the performance of various algorithms has been very positive. The value has come in several ways. First, the system calculates authoritative statistics for comparing algorithms that prevents researchers from (unintentionally) misreporting their results by using slightly different calculations.

Second, AFEWeb's confusion matrix has been very helpful for cleaning our corpus and identifying mislabeled documents. Clicking on any off diagonal cell in the confusion matrix brings up an interactive report that lists misclassified documents. It is frequently the case, or at least at the beginning of our study, that many of the misclassified documents were really mislabelled in our corpus.

Third, using AFEWeb's document visualization we have discovered how specific words contribute to a concept. For example the highest scoring document for the "baseball" concept had a set of unusual high scoring words including *shooty, igloo, dcfg, dfb,* and *a's.* In this case *shooty, dcfb,* and *dfb* were the user id's for frequent posters to the baseball newsgroup. In another example from a different experiment, the word *misarie* appeared in one test document and one training document. Both documents discussed an inscription in Old English that allegedly appeared on a tombstone. This case leads us to question the advisability of assigning high weights to words that only appear in a single training document.

6 System Implementation

There are two interesting parts to this system: the front end and back end. Our portal, the front end, is implemented as a series of interactive PHP[5] pages that use Scalable Vector Graphics[6] (SVG) to draw the visualizations. PHP is a popular scripting language for building dynamic websites and SVG is an emerging standard for browser-based 2D graphics. Our backend, described later in this section, is a series of Python[7] programs that analyze the score files and cross-reference the documents. Python is one of the newer scripting languages for building systems. The interface between the front and backend is a series of XML files. First, we describe our portal.

6.1 Front-end Processing – Thin-client SVG Visualization Components

Each of the visualizations is implemented as an interactive SVG component that runs in a web page. The components may be moved around freely on the web page as if they were windows. They are linked to each other so that interactive operations propagate. The remarkable aspect is they run completely within the browser and require no client-side software install. Data is downloaded to the components using an innovative new programming paradigm that is often called Web 2.0 or AJAX in the popular press.

[4] This clustering came from another experiment and is not related to the previous experiment.

[5] http://www.php.net/

[6] http://www.w3.org/Graphics/SVG/

[7] http://www.python.org/

6.2 Backend Processing – Python Scripts for Document Processing and Indexing

We have developed approximately eight python programs that analyze algorithm performance, calculate statistics, generate Confusion Matrices, ROC, Precision and Recall curves (not shown in this paper), and index the words in the documents. Intermediate data is written out and stored in XML files. Although it is beyond the scope of this paper, many of the calculations are interesting in the way that they pre-compute many relevant statistics to enable interactive browser response.

7 Discussion and Summary

A critical problem for computational linguistic researchers is to understand the performance of their algorithms, set rejection thresholds, determine why documents are assigned to concepts, and investigate the internal workings of their algorithms. To help with this task we have built an environment for analyzing the performance of text analysis algorithms. Although our system is tuned for a particular platform, it is broadly useful and applicable to any of the linguistic algorithms based on document feature vectors.

Our system consists of backend processing programs that analyze the performance of experiments and a portal that shows the results. There is a clean separation between the backend and our portal. The results of backend processing are a series of XML files that capture key analytical results. The portal presents the results to users.

There are several interesting aspects to our portal. The first involves an interactive page that combines ROC curves and Confusion Matrices. Using this page, algorithm designers manipulate concept rejection thresholds on the ROC curves and observe the effects on the confusion matrix. One of the unique aspects of this page is its ability to rapidly display new results without performing large recalculations. This effect is achieved by pre-computing partial results, storing arrays of change points, and asynchronously downloading the information into the browser. This occurs asynchronously while allowing the user to continue working. The usability effect is dramatic and the resulting user interface for manipulating thresholds is even better than what can be achieved with a rich desktop visualization application.

The second interesting aspect is our document visualization. By using font size and superscripts to encode dimension weights, the display clearly shows which words cause documents to be assigned to concepts. This visual abstraction provides a clear and succinct representation of how our system works and why documents are assigned to the respective concepts.

The third interesting aspect is the hyperlinks between analysis pages. Our engineering goal is to hyperlink every item so that a click brings up the relevant information to answer the next important question. Each of the cells in the confusion matrix is hyperlinked to interactive lists of documents. The documents in these lists are hyperlinked to document visualizations. Each of the words in the document visualization is hyperlinked to other documents which contain that word. The word superscripts showing the dimension weights are linked to the other words in that equivalence class. Each of these words is in turn linked to the documents that contain it.

The fourth interesting aspect involves its implementation. Our portal runs as a thin client and is completely browser based. Each of the visualizations is done with SVG web components that provide the look and feel of independent windows except that they are browser-based. The rendering is accomplished using SVG, while the interactivity is achieved by manipulating the web page's document object model. To achieve interactive

performance while downloading large information the gadgets use a new style of programming called AJAX. Taken together, the effect is stunning. In many ways it is superior to a desktop visualization without the installation and maintenance hassles.

REFERENCES

[1] Mike Attig, Stephen G. Eick, Chip Kastner, Andrew Levine, John W. Lockwood, Ron Loui, James Moscola, and Doyle J. Weishar, "Transformation Algorithms for Data Streams", IEEE Aerospace Conference, Big Sky, Montana, March 2005.

[2] John Byrnes, Stephen G. Eick, Andrew Levine, John Lockwood, Ron Loui, , Justin Mauger, Alan Ratner, and Doyle Weishar. "Hardware Accelerated Algorithms for Semantic Processing of Document Streams", IEEE Aerospace Conference, 2006.

[3] John Byrnes and Richard Rohwer, "Text Modeling for Real-Time Document Categorization", IEEE Aerospace Conference, 2005.

[4] G. Adam Covington, Charles M. Kastner, Andrew A. Levine, John W. Lockwood, "HAIL: A Hardware-Accelerated Algorithm for Language Identification", 15th Annual Conference on Field Programmable Logic and Applications (FPL), Tampere, Finland, 2005.

[5] J. Rocchio. "Relevance Feedback in Information Retrieval", in The SMART Retrieval System: Experiments in Automatic Document Processing, Chapter 14, pp. 313-323, Prentice-Hall Inc., 1971.

[6] G. Salton, "Dynamic Information and Library Processing", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

FIGURES

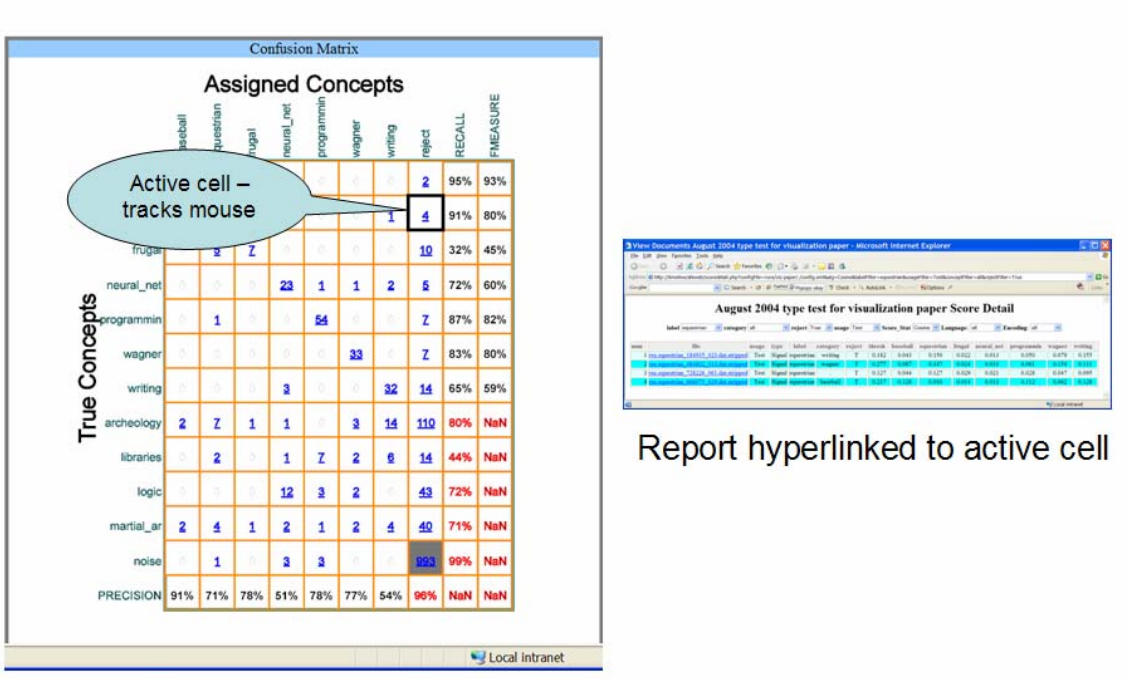

Figure 1. Interactive Confusion Matrix.

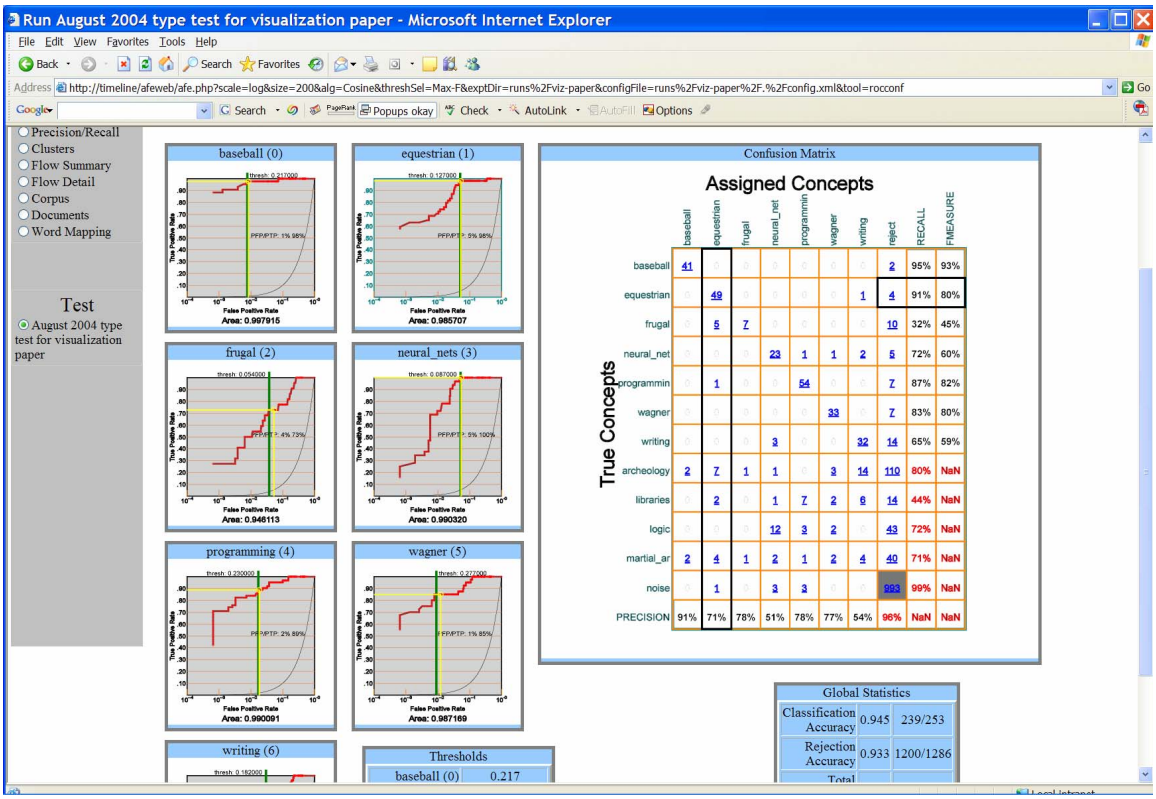

Figure 2. Linked (log-transformed) ROC and Confusion Matrices.

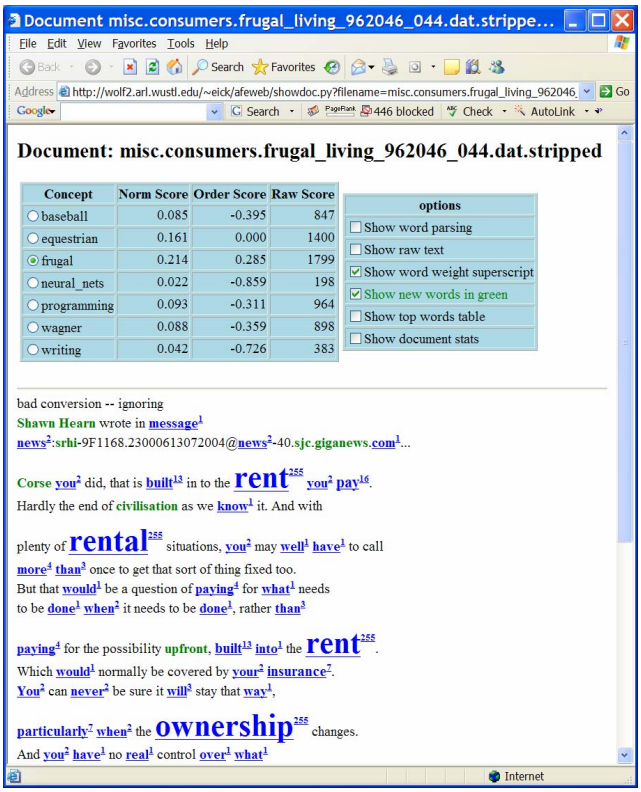

Figure 3. Document visualization showing the importance of words to the frugal concept. Word font size and superscript are tied to the weight assigned to the word by the selected concept. Green words are new words not observed in the training documents.

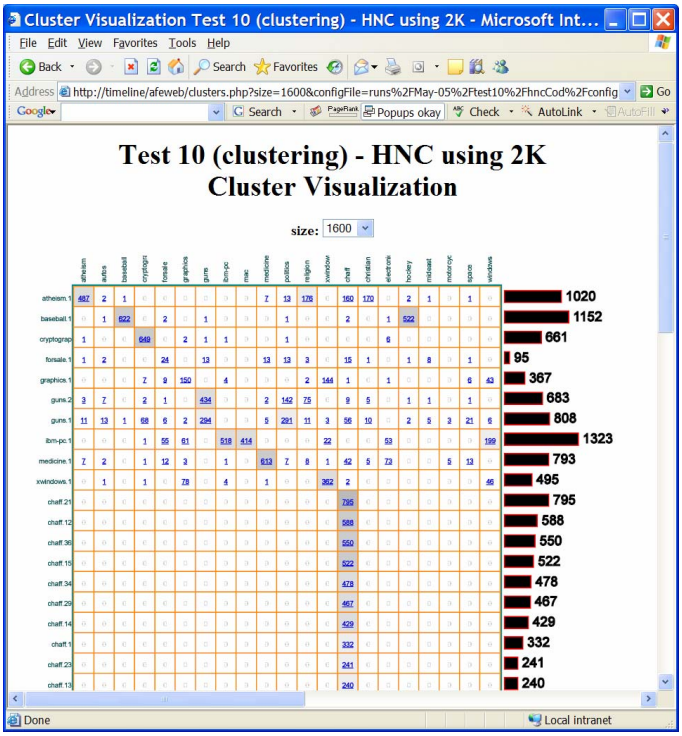

Figure 4. Flat Cluster Visuallization