Computer Science Department
CPSC 097

Class of 2005 Senior Conference on Natural Language Processing

Proceedings of the Conference

Spring 2005
Swarthmore College
Swarthmore, Pennsylvania, USA

Order copies of this proceedings from:

Computer Science Department
Swarthmore College
500 College Avenue
Swarthmore, PA 19081
USA
Tel: +1-610-328-8272
Fax: +1-610-328-8606
richardw@cs.swarthmore.edu

Introduction

About CPSC 097: Senior Conference

This course provides honors and course majors an opportunity to delve more deeply into a particular topic in computer science, synthesizing material from previous courses. Topics have included advanced algorithms, networking, evolutionary computation, complexity, encryption and compression, and parallel processing. CPSC 097 is the usual method used to satisfy the comprehensive requirement for a computer science major.

During the 2004-2005 academic year, the Senior Conference was led by Richard Wicentowski in the area of Natural Language Processing.

Computer Science Department

Charles Kelemen, Edward Hicks Magill Professor and Chair
Lisa Meeden, Associate Professor
Tia Newhall, Associate Professor
Richard Wicentowski, Assistant Professor
Benjamin Kuperman, Visiting Assistant Professor

Program Committee Members

Joshua Berney
Nicholas Guerette
Frederick Heckel
America Holloway
Andrew Lacey
Benjamin Mitchell
Jason Perini
Zachary Pezzementi
Zac Rider
Jiwon Shin
Nicolas Ward
Richard Wicentowski

Conference Website

`http://www.cs.swarthmore.edu/~richardw/cs97-s05/`

Conference Program

Tuesday, April 21

1:15-1:40 *A Simple Probabilistic Approach to Ranking Documents by Sentiment*
Andrew Lacey

1:40-2:05 *Table Recognition and Evaluation*
Jiwon Shin and Nick Guerette

2:05-2:30 *Grammar Checking using POS Tagging and Rules Matching*
Zac Rider

Thursday, April 23

1:15-1:40 *Political Blog Analysis Using Bootstrapping Techniques*
Fritz Heckel and Nick Ward

1:40-2:05 *Developing a Morphological Segmenter for Russian*
America Holloway

2:05-2:30 *Report on Political Leaning Classification*
Ben Mitchell and Zach Pezzementi

Tuesday, April 28

1:15-1:40 *Word Alignment of Parallel Texts*
Joshua Berney and Jason Perini

Table of Contents

A Simple Probabilistic Approach to Ranking Documents by Sentiment

Andrew Lacey
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
lacey@cs.swarthmore.edu

Abstract

The problem of determining the sentiment of documents has often been approached via highly human-structured or highly complex methods. These approaches have generally been constrained to dividing an input corpus into two categories, positive and negative. I present a simple and straightforward probabilistic algorithm that attempts to rank an entire corpus in increasingly-positive order of sentiment, which is a more useful output. The output can be easily reinterpreted to solve a two-category sentiment classification task, in which case the proposed algorithm performs nearly as well as existing approaches to that problem.

1 Introduction

Humans generally have little difficulty in determining the sentiment – that is, overall "positiveness" or "negativeness" – of documents. However, this has proved to be a rather difficult task for machines. Furthermore, much of the research in this direction has involved classifying the documents in a corpus into two categories, as if the task were essentially the same as topic classification, which it is not. In the case of topic classification, a document is generally, for example, about baseball or not about baseball. The output of a program ranking documents in order of the extent to which they are about baseball would probably seem strange to a human reader – a document that uses the term "home run" to describe the sales of the Chrysler 300, which might rank near the middle of such a scale, is really not about baseball at all. But this is not the case with sentiment analysis. Sentiment of documents is essentially a continuous spectrum. One imagines that the concept of "half stars" in movie reviews was introduced because human reviewers found the choice between a mere five ratings to be constraining. Thus, it makes sense to approach the sentiment analysis task in a way that naturally lends itself to ranking an entire corpus in order of sentiment, rather than simply (or, in fact, not so simply at all) making a decision between two categories, for documents that are clearly in one of those categories. The algorithm presented here takes such an approach, and is also easily modified to provide output that can be compared to that of two-category sentiment classifiers.

2 Background and Explanation

The problem of determining sentiment of documents really consists of two subproblems. The first is generating a list or dictionary of some kind containing some sort of sentiment-term data. This could take the form of a list of sentiment terms, a list of sentiment phrases, and/or a list of grammatical structures that assign the sentiment of a term in one position to a noun in another, to name a few approaches. The second subproblem is that of actually using this sentiment-term data to assign sentiment to documents in a test set.

It is possible to essentially skip the first subproblem, as far as machine NLP is concerned. That is, one can manually generate a list of sentiment data by

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

referring to a dictionary, a thesaurus, and common grammatical knowledge. This might be an acceptable approach if it could be done definitively once. However, sentiment data for varying domains can be quite different. In automotive reviews, a reference to "spongy" brake pedal feel seems to be a case of negative sentiment. But the term "spongy" is probably irrelevant to sentiment in the context of movie reviews. There are many similar examples. Thus, a human would be required to manually develop a list of sentiment data for each new domain of documents. This is a significant drawback.

It is also possible to design an algorithm – often, in existing research, a rather complex one – to develop a body of sentiment data from a training set of documents. The drawback of such methods is that they can be relatively complicated, difficult to implement, and may suffer from long running times. If they work well, that is a price worth paying. But they do not work exceedingly well, as explained in the following section. I show that similar results can be achieved using a more straightforward approach than those attempted in previous research.

3 Previous Work

A key piece of research on sentiment classification involves several methods of classifying movie reviews into positive and negative categories (Pang et al., 2002). This paper limits the domain to documents that humans have classified as clearly positive or negative. It does not attempt to rank documents on a spectrum. The methods include two probabilistic approaches, both more involved than that presented here, and a support vector approach that creates vectors describing training documents and finds a hyperplane that best separates them. The best accuracy reported by these authors is 82.9% correctly classified.

(Turney, 2002), working on a similar task, tries an interesting method: using a Web search engine to find associations between various words and the words "poor" and "excellent," classifying words that co-occur frequently with "poor" and infrequently with "excellent" to be negative sentiment terms, and vice versa. Although he achieves impressive 84.0% accuracy on automotive reviews, his attempt at classifying movie reviews logged a lackluster 65.8% accuracy. Turney mentions that "descriptions of unpleasant scenes" could be hampering the movie-review results. This is not surprising, because his sentiment data is gleaned from a Web search of general documents, where words might be used very differently than in movie reviews – not to mention the dubious choice of the word "poor" as the flag for negative sentiment, when the word is frequently used in the economic sense.

(Yi et al., 2003) reports an interesting variation on sentiment analysis. They developed a method for mining the sentiment about particular attributes of an item from a document, rather than classifying the sentiment of the entire document. While this is not directly related to the work presented here, it is interesting because it goes beyond the common task of binary document-level classification. This work is an example of a highly structured approach to sentiment analysis – the researchers used predefined dictionaries of terms and sentiment-phrase structures. They found that accuracy was quite good – 85.6 % – on product reviews, but deteriorated rapidly when the corpus contained general Web documents where sentiment phrases were sparse.

4 Algorithm

The proposed approach to the first subproblem – extracting a list of sentiment terms from a training set – functions entirely on a unigram level, with one exception, to be discussed later. A training set of documents, each of which is associated with a numerical score (the range of possible scores is specified as a parameter), is used to construct a list of words. Each word is associated with two numerical values: the number of documents in which it was seen, and the sum of the scores of all documents in which it was seen. A given word is counted only once per document. When all the documents have been processed and the table has been filled, the table data is used to compute an average score for each word. This word-scoring formula appears in equation (1), where d represents a document, w represents a word, D represents a set of documents, and S(x) represents the score of item x. The words are then ranked in order of their final scores, and the list of words and scores constitutes the output.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

$$S(w) = \frac{\sum[S(d) \mid w \in d]}{\mid (D : d \mid w \in d) \mid} \quad (1)$$

I added one heuristic that uses a bigram model. The program contains a predefined list of negation words, such as "no" and "didn't". These words are never inserted into the word table. Instead, they indicate that the next word should be flagged as negated. This is done by prepending a '!' to the lexeme. For example, the phrase "didn't satisfy" would result in the word token !SATISFY being inserted in the word table (or its count being incremented, if it were already there).

It is worth noting that this approach produces a list containing a large number of terms that are not sentiment terms by any reasonable standard. One could argue that using such a list would be a case of overfitting the data. However, in a certain sense, all intelligent approaches to sentiment-data gathering involve overfitting the data. As alluded to previously, a good set of sentiment data for movie reviews would probably perform poorly when used to classify automotive reviews. A generic set of sentiment data would probably perform poorly on most domains, as (Turney, 2002) discovered. Either it would be so small as to miss most of the sentiment terms in each document, or it would be so large that it would assign sentiment to terms that were not sentiment-oriented within the domain. Thus, a good set of sentiment data for a given domain is defined functionally – as a set of data that performs well at judging sentiment for that domain. Whether the words in the list "look like" sentiment words is not particularly relevant. Incidentally, this implementation does ignore words containing capital letters, on the grounds that assigning sentiment to proper names might indeed be a case of excessive overfitting. Reviews of 1990 Hyundais are much more negative than reviews of 2004 Hyundais, so a corpus with a large number of early Hyundai reviews might result in negative sentiment being assigned to the word "hyundai", which is clearly undesirable even within the domain. But, in general, I do not view the inclusion of non-sentiment terms in the word table as a serious problem.

There are two key parameters that can be adjusted to change the results of the sentiment-term-discovery process. The parameter *f* is the number of times that a word must be seen in order to be included in the final word list. For example, a word that was only seen in one review might be rare, spurious or irrelevant. The parameter *n* is the number of words on each end of the sentiment spectrum that will be included in the final word list. One approach would be to essentially set this parameter to infinity, leaving the entire word list intact. Since words near the center of the spectrum are probably not sentiment words and may be irrelevant, I hypothesized that removing a portion from the middle of the word list might give better results when scoring test reviews. A sample of resulting words and scores from running this algorithm on a training set of movie reviews appears in Table 1. Note that scores are automatically normalized to a zero-to-one scale for consistency across corpora.

Word	*Score*
!FUNNY	0.415323
SLOG	0.426471
DISMAL	0.431818
REDEEMING	0.4375
UNFUNNY	0.446429
...	...
ENCHANTING	0.897727
TRANSCENDS	0.902778
RESIGNED	0.902778
BLESSED	0.908333
HEARTBREAKING	0.910256

Table 1: Sample output of sentiment-term extraction

Once the list of scored words has been generated, the second subproblem – ranking a test set of documents – can be attacked, essentially in the inverse way. Each test document receives the score equal to the average of the scores of the words appearing in the document. Words in the document that are not in the list of sentiment words are ignored. The output is a list of pairs of numbers – the score assigned to the document by the program, and the document's actual score. The document-scoring formula appears in equation (2), where *L* is the list of words resulting from the term-extraction process and *W* is a set of words.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

$$S(d) = \frac{\sum[S(w) \mid (w \in d) \wedge (w \in L)]}{\mid [W : (w \in d) \wedge (w \in L)] \mid} \quad (2)$$

It should be noted that the assigned scores output by this algorithm tend to be packed very close to the middle of the score range, and thus are not directly usable as meaningful scores. They could obviously be normalized to the scale if meaningful scores for individual reviews were desired. The goal of this work was to output a ranking of documents, which could be compared to the true human-determined ranking for a clear picture of the overall performance of the algorithm. For a large body of documents, this seems to be closest to the sort of task we would want to perform in a real-world situation. It also allows for some telling visual representations of the results. Much previous work on sentiment classification has involved classifying documents merely as positive or negative. Aside from the fact that this seems not terribly useful in practical applications, a few statistics showing the percent of documents correctly classified in a two-bucket model does not exactly provide a nuanced look at the performance of the algorithm.

5 Results

I ran trials on two corpora: a very large set of Roger Ebert's movie reviews (approximately 4,000,000 words) and a smaller set of Consumer Reports Magazine's automotive reviews (approximately 86,000 words). I divided each corpus into a training set and a test set of approximately equal size. For each corpus, I tried several settings of the two parameters described previously. Results are presented in graphical form. The horizontal axis represents the ranking this algorithm assigned to a document. The vertical axis represents the human-assigned score of a document. Thus, completely correct output would be a nondecreasing y-value as x increases. In the case of the movie reviews, where the number of documents is large and the number of possible scores is small, this would look like a step function. Lines connecting the points have been omitted from these graphs for clarity.

While the graphs clearly deviate significantly from an ideal result, they do provide some promising evidence. The Ebert graphs in Figures 1 and 2 show an obvious trend of increasing y-values as the x-values increase, which is a desired result. The algorithm works rather well with reviews that received scores at an extreme, particularly those that received very low scores (0 and 0.5). The middle of the spectrum is somewhat muddled, though an upward trend is still visible. It should be noted that the graph of an ideal result would not show steps of an equal width, because there are fewer 0-score reviews than 2.5-score reviews, for example. Thus, the fact that the 2.5-score reviews form a wide cluster is not entirely unexpected. The higher setting of both parameters seems to give better performance at the lower and upper ends of the scale, as is evident in the graphs. I tried a number of different parameter settings not shown here. There was not a great deal of variation in the results from different parameter settings.

In addition to generating the graphs showing the ranking of all reviews, I also tried ignoring all documents with human-assigned scores other than 0, 0.5, 3.5 and 4, supposing that the goal was to correctly classify strongly negative and positive documents as such. I counted a document that appeared in the lower half of the overall ranking as being ranked negative, and a document appearing in the upper half as being ranked positive. On this scale, 81% of strongly negative and strongly positive documents were correctly classified in the best trial. The best accuracy achieved by (Pang et al., 2002) on a similar task using much more complex approaches was 82.9%, which was achieved through the use of support vector machines. Thus, this algorithm appears to be within striking distance of a known benchmark on the binary classification task.

The results from the Consumer Reports corpus, in Figures 3 and 4, were similar in character to the Ebert results, though appear less informative due to the small size of the corpus. While there is a correct upward trend in the graphs of results, there is quite a bit of fluctuation. However, it is notable that, if the task is recast as a binary decision task, where reviews with a human-assigned score of 80 points or higher are considered positive, those with a human-assigned score of 40 points or lower are considered negative (a larger section of low scores was used because almost no reviews in the corpus have human-assigned scores below 20 points), and all other reviews are ignored, the classification ac-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

curacy is 87%, which is quite good. In fact, this is better than the accuracy numbers reported by any of the similar binary classification experiments noted in Section 3. This is, no doubt, at least partially due to the nature of the Consumer Reports documents, which are written in a very straightforward and unembellished style that does not wander off on tangents not directly related to the product being reviewed, as movie reviews sometimes tend to do. None of the other sentiment-classification research I have examined used Consumer Reports documents, so the accuracy number presented here is not directly comparable. If a larger Consumer Reports corpus were compiled, and some additional heuristics possibly added to the algorithm, I would not be surprised to see accuracy approaching 90%. Further experimentation in this direction is warranted.

6 Future Work

Assembling a larger corpus of Consumer Reports documents for further experimentation is an obvious next step. I would expect better results for a larger training set. Some further heuristics, perhaps including additional bigram analysis or part-of-speech sensitivity, may be worth trying. However, I would be careful to add such features gradually and conservatively. The results presented here show that other researchers who began with fairly complex approaches may be over-thinking the problem.

7 Conclusions

Automatic sentiment classification is a relatively difficult problem. However, when a large corpus is available, a simple probabilistic approach yields results that are nearly comparable to those derived from substantially more complicated algorithms. Furthermore, this simple approach lends itself naturally to creating an ordered ranking of documents by sentiment, rather than merely classifying documents into two buckets. This may be a much more useful approach in real-world applications. A human who is trying to read reviews of products will likely want to read the best few reviews, not the best half, particularly when thousands of reviews are available. The approach presented here is inherently suited for generating such results.

References

Bo Pang, Lillian Lee and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. In *Proc. of the Conference on Empirical Methods in NLP, July 2002, pp. 79-86.*

Peter D. Turney. 2002. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *Proc. of the ACL*

Jeonghee Yi, Tetsuya Nasukawa, Razvan Bunescu, Wayne Niblack. 2003. Sentiment analyzer: extracting sentiment about a given topic using natural language processing techniques. In *Third IEEE International Conference on Data Mining, Nov 2003.*

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

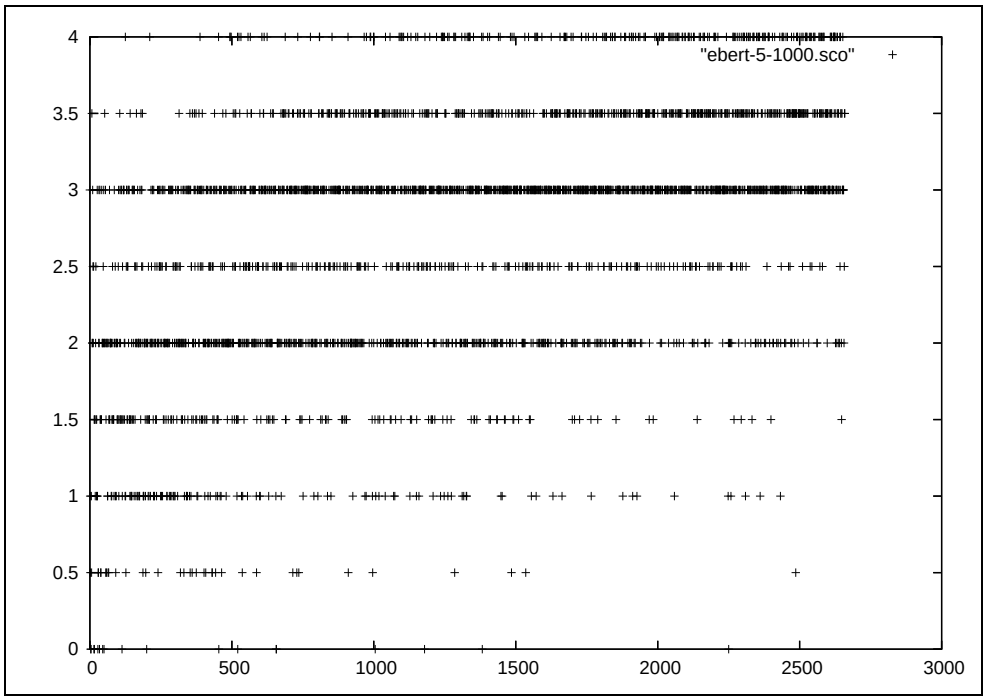


Figure 1: Results for Ebert corpus with *f*=5 and *n*=1000

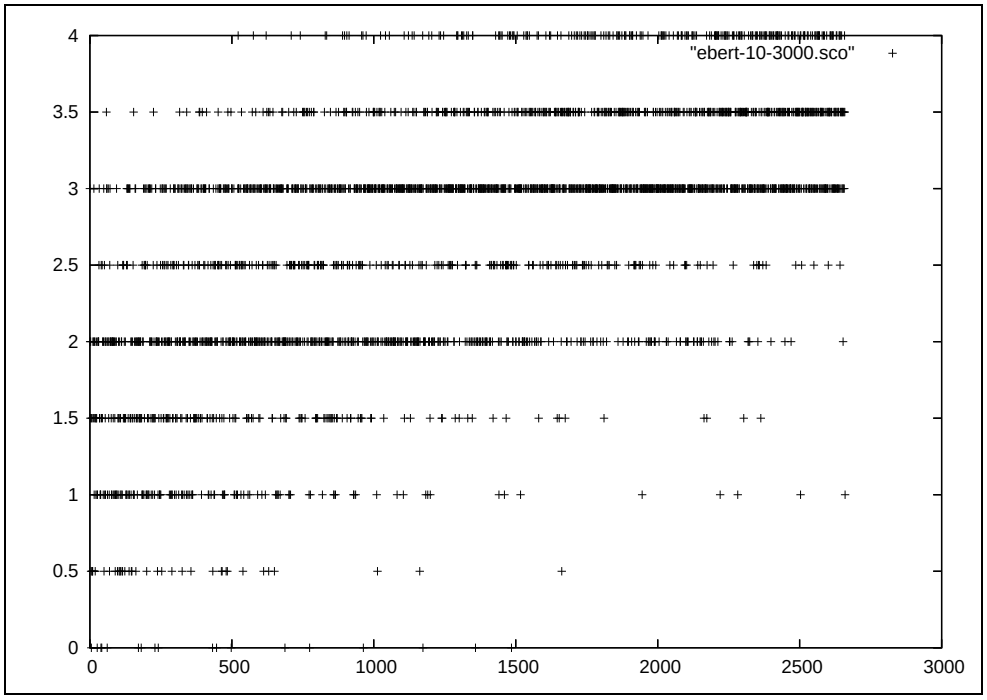


Figure 2: Results for Ebert corpus with *f*=10 and *n*=3000

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

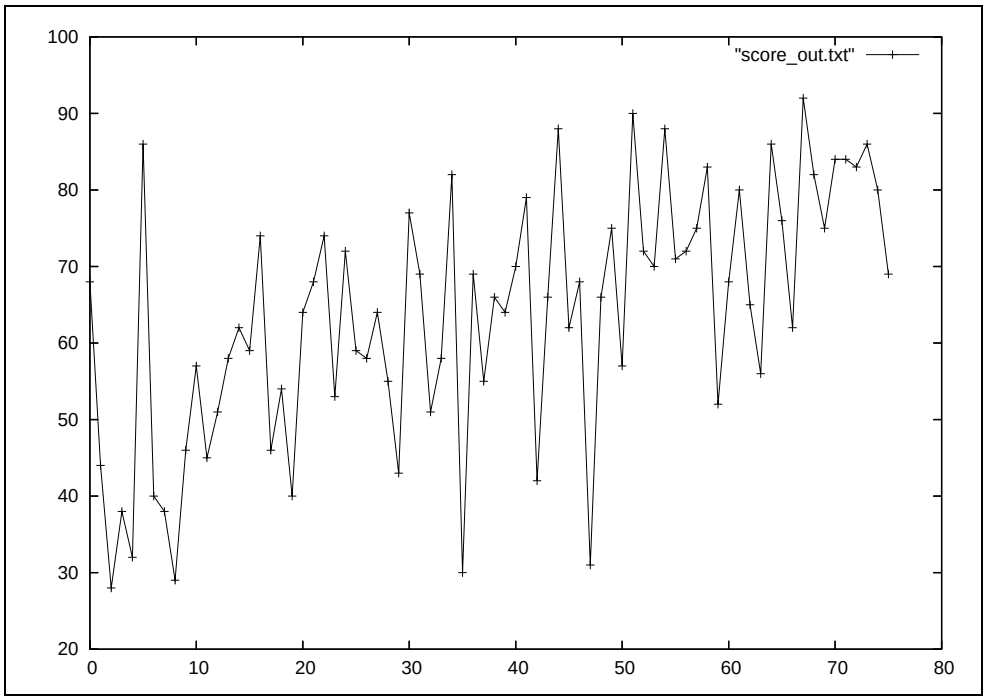


Figure 3: Results for Consumer Reports corpus with f=2 and n=500

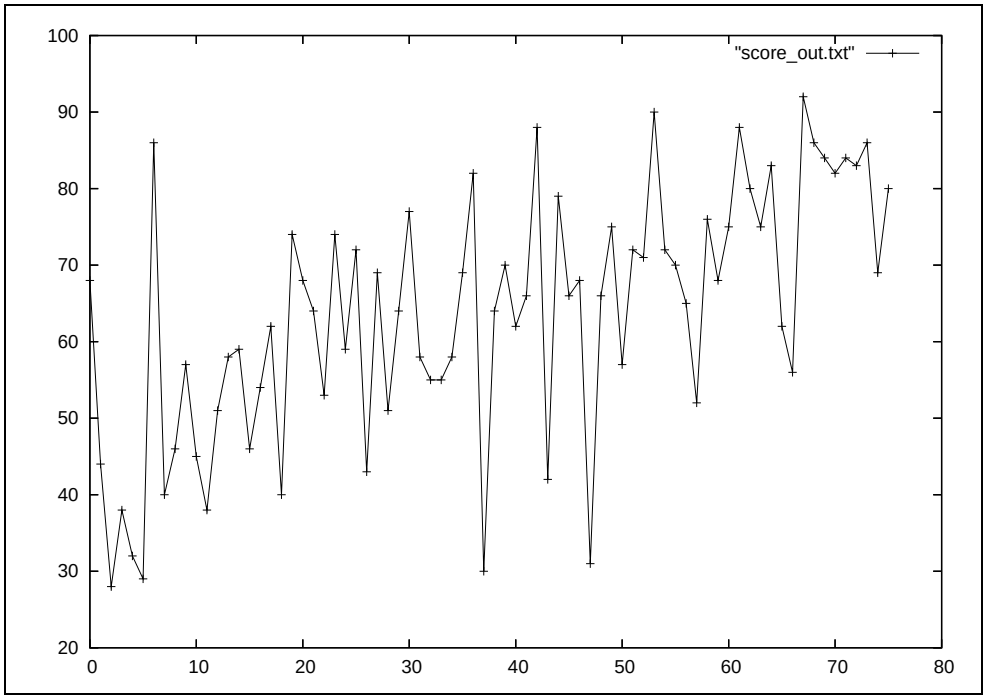


Figure 4: Results for Consumer Reports corpus with f=2 and n=∞

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

Table Recognition and Evaluation

Jiwon Shin
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
jiwon@cs.swarthmore.edu

Nick Guerette
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
ngueret1@cs.swarthmore.edu

Abstract

We present an algorithm that recognizes tables in document images and extracts their structural information. We use region growing to locate bounding boxes around text, and cluster them into columns by examining spatial relationships between bounding boxes and their vertical neighbors. Once initial clustering is complete, a series of post-processing steps are applied to the clusters to find columns that line up horizontally and may form tables.

1 Introduction

In performing optical character recognition on document images containing tabulated text, it is necessary to extract the text of each table cell, and desirable to obtain information about the relationships of table cells to each other.

Our goal is to create a system that recognizes tables in document images and extracts the portions of the image that correspond to each of the table cells, keeping track of the spatial relationships between table cells.

2 Previous Work

A number of researchers have suggested algorithms for table extraction and table structure recognition. A survey of the field is provided by (Zanibbi, Blostein, and Cordy , 2003).

(Watanabe et al., 1991) created a hierarchical table recognition and analysis system that first locates line segments separating table cells, uses the spatial relationships among table cells to deduce the logical relationships among them, and passes the extracted cells to higher-level processing functions. They propose allowing higher-level data processing functions to return information about contradictions encountered to lower-level functions, so that the lower-level functions can attempt a different analysis.

(Chandran and Kasturi, 1993) modified that method to require only a line at the top and bottom of a table to allow it to be recognized, but not lines separating all cells of the table. They instead use vertical and horizontal projections of binary images of extracted tables to identify boundaries between rows and between columns.

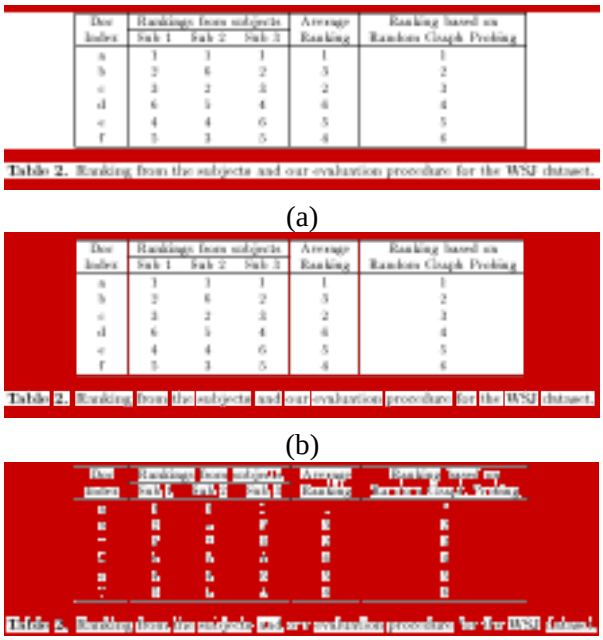
Our table recognition algorithm is based on (Kieninger, 1998). The author proposed a method that identifies tabular structures in a document by grouping word bounding boxes together and searching for vertically-aligned groups of words that could potentially be columns.

3 Algorithm

3.1 Overview

Our program takes a document image as input, and places bounding boxes around blocks of text by region growing. It then executes a set of post-processing steps to determine if any of the bounding boxes should be merged. Finally, spatial relationships between bounding boxes are examined in order to locate possible tables and identify their structure.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College



(a) (b) (c)

Figure 1: Segmenting by divider line identification (a) locating the divider rows (b) locating the divider columns between two divider rows (the result of the first iteration) (c) after multiple iterations; the gray regions represent areas that do not belong to a bounding box

3.2 Finding Bounding Boxes

3.2.1 "Divider" Line Method

We initially attempted to find bounding boxes by downsampling the input image and recursively locating "divider" lines. The program takes a document as input, and downsamples it to have a width in the range $[256, 512)$. It then scans across each row of the downsampled image, counting the number of times that the intensity of a pixel differs from the intensity of a neighboring pixel in that row. If the number of changes of intensity in a row is below a threshold, which is proportional to the length of the row, that row is marked as a "divider" row. A "divider" row is assumed to not go through any text (Figure 1(a)). Once all the divider rows are identified, the system executes the same procedure to scan down columns within clusters of non-dividing rows (Figure 1(b)). This procedure produces a list of bounding boxes, where each bounding box contains the minimum and the maximum row and column values. The list is added to the global bounding box queue, which is initialized to be empty. For each bonding box in the queue, we repeat the "divider" line procedure described above on the portion of image defined by the bounding box, unless the box is smaller than a threshold, and add the list of bounding boxes this procedure outputs to the queue. If the procedure returns an empty list, the bounding box is removed from the queue and is stored in a separate list of final bounding boxes. If, on the other hand, the procedure returns a list of one or more smaller bounding boxes within the original bounding box, then the original one is discarded. Bounding boxes that are smaller than a threshold are removed from the queue and added to the list of final bounding boxes. This iterative process continues until the queue is emptied.

This method of finding bounding boxes did not work well (Figure 1(c)). While the results were acceptable when there were no lines separating table cells, the algorithm's performance on tables with lines was poor. There were several problems with the algorithm. First, the downsampling process sometimes causes lines to have inconsistent intensity, resulting a failure in recognition. Second, when a table does not have many columns, the opposite can occur. Some of the rows in the table may be marked as "divider" rows because the intensity changes only few times, even though these rows are not "divider" rows. Third, when the downsampling step was skipped to attempt to correct the above problems, the algorithm generated an excessive number of bounding boxes, most of which were a character wide. This can be minimized by adding a dilation step, but because pixels that are turned "on" tend to be the same intensity, dilated columns often passed the "divider" test, failing to fundamentally fix the problem.

3.2.2 Region Growing Method

After noticing the weaknesses of the "divider" line method, we decided to implement a region growing algorithm to search for bounding boxes around words. After reading the greyscale input image, the system determines an intensity threshold using an adaptation of the ISODATA clustering algorithm (see Appendix A), and uses the threshold to binarize the image (one intensity for background, another for text). Based on the assumption that even

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

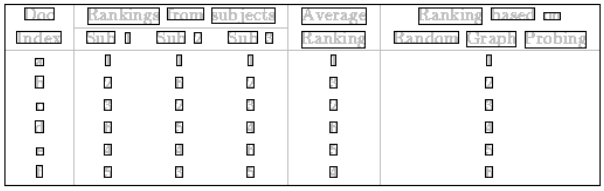


Figure 2: A set of bounding boxes that are inside a big bounding box

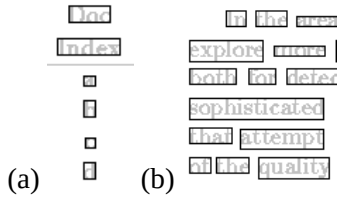


Figure 3: Two different cluster types (a) type 1 (b) type 2

a light tone of grey belongs to a letter and not the white background, we modified the ISODATA algorithm to be biased toward identifying pixels as belonging to text. Letters in the binary image are then dilated horizontally so that all characters in most words are connected. The amount by which to dilate is determined as follows. First, a histogram of lengths of horizontal runs of background pixels between text pixels is created, and the most common length is found. Then, the histogram counts of increasingly longer lengths beyond that most common length are examined, and the first whitespace length to have a count less than half that of the most common whitespace length is chosen as the amount by which to dilate. The dilation is executed by marking that many pixels to the right of each text pixel in the original image also as a text pixel.

Once the preprocessing is complete, we apply a region growing algorithm to place bounding boxes around each region of connected text pixels (with the rightmost edge of the bounding box being adjusted leftward by the amount by which text pixels were dilated earlier, so that the bounding boxes are around the original text and not the dilated text). When the image contains a table with an outline, this results in a bounding box that surrounds the table in addition to a bounding box for each word in the table. To detect this outer box, we mark all the bounding boxes whose height is greater than the average height of all the bounding boxes, and test them to determine whether or not they contain any smaller bounding boxes. We keep track of all of these "big" bounding boxes and the smaller bounding boxes they contain, as they are likely to form a table (Figure 2). This information ended up not being used in the final algorithm, however, except to ignore the "big" bounding boxes.

This method located bounding boxes more successfully and reliably, and hence, we used this method to find bounding boxes.

3.3 Table Identification

Our table identification algorithm, described below, is a modified version of the algorithm presented in (Kieninger, 1998). We had initially implemented the algorithm as it is described in the paper, which produced a set of incorrectly-grouped clusters, as expected. The problem we faced was that some of the incorrectly-grouped clusters required post-processing procedures that were too complicated, detracting from the benefits that the clustering step offered. We hence decided to modify the algorithm to do more sophisticated clustering, which simplified the post-processing step.

The table identification algorithm takes as input a list of bounding boxes that are not big bounding boxes. The first box in the list is picked as a "seed" and moved up and down by its height to test if any box in the list overlaps the region defined by the the seed box (If the height of the seed box is less than the average height of bounding boxes, we increase its height to the average height of bounding boxes for the purposes of this step). If we locate such a box, we perform the *one-to-one relation* test, which tests if the seed box has at maximum one overlapping bounding box above and one below it, and that the boxes above and below are not horizontally offset. If the found box preserves the seed box's *one-to-one relation*, we put it into a cluster with the seed box and grow it vertically to find other boxes that need to be linked. This process continues until the *one-to-one relation* is no longer preserved or the program finds no more bounding boxes to add to the cluster. This process is then repeated on bounding boxes as yet unexamined until all bounding boxes have been clustered or found not to belong to a clus-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

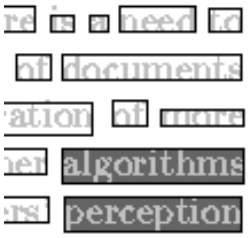


Figure 4: A false identification of a table column. The bounding boxes in gray are clustered together and are marked as *type 1* because they happen to line up vertically.

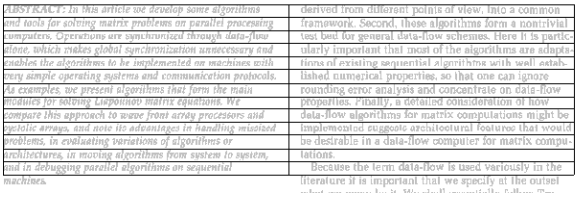


Figure 5: An incorrect table identification

Figure 6: An incorrect structural analysis of a table

ter. The clusters generated in this process are called *type 1* clusters (Figure 3(a)), and are candidates for being columns of a table. All the bounding boxes that did not have any *one-to-one relations* with the vertical neighbors were put into a separate cluster named *type 2* (Figure 3(b)).

After we obtain clusters, we apply a series of post-processing steps to avoid errors that are created by the initial clustering process.

For each bounding box in the *type 2* cluster, we calculate the distance between it and its horizontal neighbors. A threshold for horizontal distance between neighboring bounding boxes is calculated using the same algorithm used earlier to calculate the amount by which to dilate text pixels, except examining the distance between bounding boxes rather than horizontal runs of background pixels. If the distance is less than the threshold, the seed box and the relevant neighboring box are joined into one box, and the distance between the new, larger box and its horizontal neighbors is calculated to determine whether or not more boxes should be joined to it, and this is repeated until all bounding boxes have been examined and joined if necessary. Although boxes in *type 1* clusters are not used as seed boxes in this step, to allow for large tables with closely-spaced columns, boxes in *type 1* clusters that are neighbors to a box in the *type 2* cluster being used as a seed may be joined with that seed.

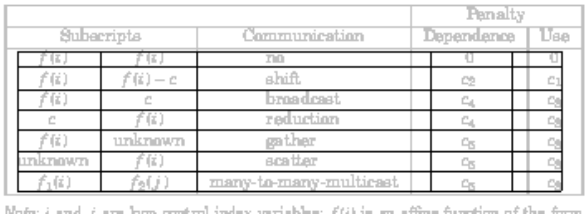
After horizontal joining is completed, the same algorithm used to find *type 1* clusters is applied again on the new set of bounding boxes. The number of false identifications in blocks of text is now reduced (Figure 4). With these final *type 1* clusters, they are now all compared to each other, and any clusters that contain any bounding boxes that vertically align are marked as belonging to a table.

4 Results

We tested our algorithm on twenty different document images ranging from images that only contain texts to those with pictures, figures, tables, as well as text. Our algorithm had 28.2% precision and 90.0% recall counting tables (even if rows and columns weren't correctly identified or extra rows and columns were identified outside the real table) and 40.8% precision/87.2% recall counting individual table cells identified. The most common error was an incorrect identification of a text block as a table (Figure 5). This false identification occurred especially frequently among documents that had two columns. Another error that frequently occurred was an incorrect analysis of the structure of a table. As shown in figure 6, some of the table cells are subdivided into smaller cells, causing an overproduction of table columns. The problem that occurred in figure 6 cannot be fixed easily because all the cells are in a *type 1* cluster. None of the cells will ever be tested for horizontal grouping, and the cells stay in separate columns.

Our program was able to locate tables with and without borders, tables with cells that span multiple columns (Figure 7), as well as tables of pictures (Figure 8). We had the most trouble extracting

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College



(a)

Doc Index	Rankings from subjects Sub 1	Sub 2	Sub 3	Average Ranking	Ranking based on Random Graph Probing
a	1	1	1	1	1
b	2	6	2	3	2
c	3	2	3	2	3
d	6	5	4	6	4
e	4	4	6	5	5
f	5	3	5	4	6

(b)

Figure 7: An identification of a table with cells that span multiple columns (a) cells that span multiple columns are ignored (b) cells that span multiple columns are divided into smaller cells

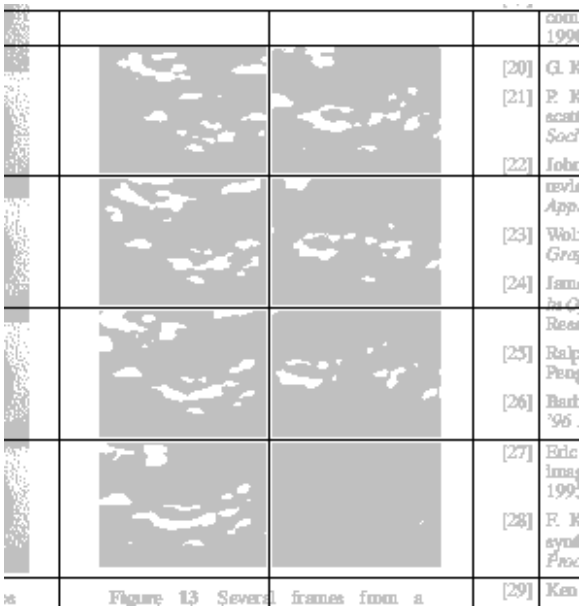


Figure 8: A correct identification of a table of pictures

the structural information of tables with cells that span multiple columns. In some cases, the cells that span multiple columns were ignored (if there were no cells in that row not spanning multiple columns) while in other cases, those cells were divided into smaller cells (if there were some cells in the same row not spanning multiple columns, so they would have been part of one of the clusters making up the table). If the latter happened, the list of table cells returned the correct information, i.e. the cell spans three columns, even though the corresponding output image did not reflect it.

5 Conclusions

We presented an algorithm that identifies tables in a document and extracts their structural information. Our algorithm finds bounding box for each unit (a word, a picture, etc.) in the document image, and clusters the bounding boxes together. The clusters go through a post-processing step, after which table cells are grouped together and tables are identified. We have shown that this algorithm works reasonably well regardless of the content of the table cells. It was capable of identifying about 90.0% of all the tables in scientific documents, whether the document was a simple document with a table on top and text in the bottom or a complex one with pictures, graphs, source codes, tables, and texts in two columns.

6 Future Work

There are several ways this algorithm can be improved. As mentioned earlier, the algorithm often returns a block of text as a table when the input document has two columns. This can be minimized by locating the column divider that divides the document into two columns, and disallowing a bounding box from one side of the divider to be tested against one from the other side.

Throughout the project, we assumed that the input image is correctly aligned. However, for documents where this algorithm is useful, such is not necessarily the case. Many documents are hand-scanned, and thus do not line up perfectly. To handle such documents, we need to add a preprocessing step to the algorithm, aligning the document before bounding boxes are located. The preprocessing step would

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

constitute determining the major and minor axes for the document and rotating the image by the discrepancy.

After the horizontal joining step, table cells spanning multiple columns are often contained in a single bounding box, and this is information that could be used, once a table is identified, to correctly assess the structure of the table by examining alignment between these bounding boxes and cells known to be in the table.

Once the algorithm extracts tables with high precision and recall, it can be integrated into an OCR software to correctly extract information in a table.

References

S. Chandran and R. Kasturi 1993. Structural Recognition of Tabulated Data *Proceedings of the Second ICDAR*, 516-519. Tsukuba Science City, Japan.

T. V. Kieninger. 1998. Table Structure Recognition Based on Robust Block Segmentation *Proceedings of Document Recognition*, volume V. 22-32. San Jose, CA.

N. B. Venkateswarlu and P. S. V. S. K. Raju 1992. Fast ISODATA Clustering Algorithms *Pattern Recognition*, volume 25(3). 335-342.

T. Watanabe, H. Naruse, Q. Luo, and N. Sugie 1991. Structure Analysis of Table-Form Documents on the Basis of the Recognition of Vertical and Horizontal Line Segments *Proceedings of the First ICDAR*, 638-646. Saint-Malo, France.

R. Zanibbi, D. Blostein, and J.R. Cordy 2003. A Survey of Table Recognition: Models, Observations, Transformations, and Inferences *http://citeseer.ist.psu.edu/zanibbi03survey.html* .

Appendix A

The following is the pseudo-code for our adaptation of the ISODATA algorithm used by the system. This version is designed to allow biasing toward either extreme. In our implementation, we used a white bias of 1 (the default value), and a black bias of 3. More information on ISODATA can be found in (Venkateswarlu and Raju, 1992).

```
thresh = range/2;

while(1) {

   black_mean = white_mean = 0;
   black_count = white_count = 0;

   for all buckets i
         in the histogram below thresh
      black_mean += i * bucket_count[i];
      black_count += bucket_count[i];

   for all buckets i
         in the histogram not below thresh
      white_mean += i * bucket_count[i];
      white_count += bucket_count[i];

   black_mean = black_mean / black_count;
   white_mean = white_mean / white_count;
   weighted_mean = ((blackmean * whitebias +
                     whitemean * blackbias)/
                    (whitebias + blackbias));

   if weighted_mean == thresh then
      break;
   else
      thresh = weighted_mean;
}

return thresh;
```


Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 8–13
Computer Science Department, Swarthmore College

Grammar Checking using POS Tagging and Rules Matching

Zac Rider
Computer Science Department
Swarthmore College
Swarthmore, PA 19081
rider@cs.swarthmore.edu

Abstract

This paper is an examination of various techniques that could be used for grammar checking and the description of the results that were generated using a simple rules matching system. To generate the rules for this system, two techniques were considered: hand construction and an algorithm that randomly generates large numbers of rules and uses comparison against large corpora to find valid rules. While individual construction of rules proved to be effective for addressing specific errors, the random algorithm proved to be effective for a larger number of grammatical errors.

1 Introduction

There's something wrong with the sentence: *Microsoft company should big improve Word grammar check*, but Word 2004 thinks that the only problem is that *company* should be capitalized. Grammar checking is one of the more complicated tasks for word processing, and the more irregular and exception-filled the language, the more difficult the problem becomes. Problems such as a noun-verb mismatch: *one of the mistakes are bad*, or adjectives incorrectly used as adverbs: *I can't read so good*, are much easier to find than a somewhat ambiguous mistake such as: *The badger was acted upon* (passive voice).

The simplest method of fixing grammatical errors, which was used for the experiments for this project, is the process of rules matching, that is, constructing a rule that applies to a given grammar and then checking that the given input follows, or does not follow, that rule. Using lexigraphically aided finite state machines is another, more complicated method, that combines a bootstrapped learning algorithm with parsing and POS tagging (Sofkova Hashemi et al., 2003). Other methods include syntactic analysis and parse tree analysis (Bender et al., 2004).

One thing that differs in the methods of grammar checking systems is whether or not the system is checking for negative or positive grammar. Intuitively, it seems like it might be easier to define the properties that are correct in a grammar, as there are a set number of grammatical configurations that are correct and an infinite number of configurations that are incorrect. The problem is that describing all of the correct configurations for a grammar checker requires that for every check, it must look at every single rule to see if a given example is in the grammar. This process is necessarily slower than a system that uses a relatively few rules per check to see if something is *not* in the grammar. Since speed is not of great concern for the system in this paper, the rules checking could have been implemented either way, but for simplicity, we chose to implement rules that check for specific errors in grammar instead of using a model of correct grammar to find incorrect examples. For a small system, it is easier to describe a few things in English that are grammatically incorrect than every rule that is correct.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

2 Related Work

One approach to checking grammar relies on a technique called *aligned generation* (Bender et al., 2004). However, this process is not used in the everyday sort of grammar checking that might be used in a word processor, rather it is a complicated process that takes a fair amount of time and is used for generating language learning systems. The system takes *mal-rules* and *mal-lexical types and entries* given by the user and uses *feature structure grammar* analysis, which is an extensive search of multiple parse trees for errors based on the given rules. The majority of the work in the system is the parsing itself, in which the input sentence is put into every possible configuration, and then those configurations are rated, and an acceptable configuration is chosen. One concern with this method is that the process of creating the parse trees for analysis is potentially time consuming.

Finite state machine analysis has the interesting property of not being a rules based system, rather it is a bootstrapped learning system that uses regular expressions along with FSMs to attempt to judge the correctness of lexically determined phrases (Sofkova Hashemi et al., 2003). The phrases generated by the system's lexicon are strings mapped to a tag containing part-of-speech and other feature information. While this method has 92% recall, it only has approximately 45% precision. This could prove cumbersome for a word processor system, as the user could be presented with many cases that the checker flags as errors that are, in fact, correct. However, for the task described in this paper, the recall percentage is acceptable. The random rules generator described in this paper is an approximation of this type of analysis, but the system detailed in this paper has no lexigraphical aids.

The system that this paper attempts to emulate is the Granska rules matching system (Domeij et al., 1999), which makes a point of not using Hidden Markov models and simply using what the author calls *error rules* to locate errors and *helping rules* to attempt to determine the best correction, and thus the best fitting rule for a given error. The Granska system has precision and recall of approximately 80% for the problems that it was designed for, namely noun-phrase disagreement and incorrectly split compounds in Swedish. The issue with the Granska system, however, is that while it has good results for these two problems, it turns out that the methods used in Granska do not translate well to all problems in grammar.

3 Parts of a Grammar Checker

A typical grammar checker that might be found in a word processor consists of three different pieces. First, a processor has to be able to separate the input into individual sentences. Then, it needs a part-of-speech (POS) tagger that can accurately label the data that it has. Charniak has an excellent analysis of POS tagging (Charniak et al., 1993) that is used by the makers of the Granska system. The particular POS tagger that is used for this system was taken from the Stanford website `http://www-nlp.stanford.edu/links/statnlp.html` (Toutanova and Manning, 2000; Toutanova et al., 2003) and works in log linear time. The speed of this system substantially speeds up training and test, as tagging is a necessary preprocessing step.

One issue that is of some concern for this system is that of POS tagger granularity. Some of the grammatical errors in English are fairly fine grained (ie. *was* vs. *were*), and because a POS tagger may not differentiate between the two, it makes it very difficult to attempt to detect problems associated with them. From a tagging perspective, the sentence *I wish I was dead* is the same as *I wish I were dead*, while from the perspective of a grammar checker, the second is correct and the first is not. While this particular example is not difficult to correct, it is a recurring problem that highlights the fact that when hand-constructing error rules it is easy to for them to become over-trained. When the granularity of the POS tagger isn't fine enough, a grammar checker which relies solely on POS tags will not be able to distinguish between many pairs of grammatical and ungrammatical sentences such as the ones illustrated above.

Finally, the system needs a method of identifying grammatical errors. In the case of Granska (Domeij et al., 1999), they exclusively use error rules matching. Rules matching has the convenient properties of being fast, easy to implement, and accurate for

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

the set of problems that the rules are constructed for. The regular expression analyzer and aligned generation systems are more suitable for larger scale systems that attempt to evaluate grammars as a whole.

4 Procedure

4.1 Rule Construction by Hand

Taking heavily from the ideas of the Granska system, the grammar checker created for this project essentially searches for a set of grammatical conditions and then flags something as an error if those conditions are found. For example, for a noun-verb mismatch the checker searches for a noun and then a verb. If the noun is singular and the verb is plural or vice versa, the phrase is noted as incorrect and the rules that are violated are recorded. What makes the process of rule constructing difficult, is that no rule is ever without exception. In addition to looking for a noun and a verb, the checker must also be able to ignore any possible prepositional phrase in between.

The rules system takes a given sentence and then runs every single rule in sequence. Rules can be added or subtracted depending on which grammatical error the user is looking for. Essentially, every rule is a small finite state machine. Rather than using actual words, the rules only check the POS tags of words. The size of a given rule is the number of POS tags that the rule contains. For each sentence, the grammar checker invokes each rule, which then checks itself against the sentence. This method has been implemented as a depth-first search of the sentence. First, the rule checks to see whether the tag of the current word in the sentence matches the first tag in the rule. If it does, the checker cycles to the next word to see if its tag matches the next tag of the rule, and so on for the whole sentence. In the case of a wild card tag, the checker simply cycles until it detects that the tag it is considering is the next tag in the sequence of the rule. If the next tag is never found, then the machine simply returns false.

For example, one of the specific rules for noun-verb mismatch contains the POS tags: {NNS, PP, *, NN, VBZ}. This rule, containing 5 POS tags, is size 5, and the '*' symbol stands for a wild card. For the sentence *The dogs of war is released*, the rule identifies *dogs* as the noun, then the preposition, which is *of*. The next noun is the object of the preposition, *war*, but there may be any number of POS tags in between *of* and *war*, because of the wild card tag. After *war*, the verb, *is*, is associated with the noun *dogs*, which is grammatically incorrect. Since the POS tags in the sentence follow the sequence in the rule, the sentence is flagged.

In the case of the sentence, *The dogs is eating the food*, a different rule is needed to catch the mistake. A rule containing the sequence: {NNS, VBZ} would work in theory, but then the sentence *The problem with the dogs is that they are bad.* would also be flagged as incorrect even though it is not. Examples such as these necessitate different levels of rules. This system has three different classifications of rules: specific, general, and improbable. Specific rules, such as: {NNS, PP, *, NN, VBZ} have the longest definitions. Specific rules have the highest probability of finding actual errors and not mistaking good sentences for bad sentences. General rules typically are just a little simpler than specific rules. If a specific rule would examine five tags, a general rule would examine two tags with a wildcard like: {NNS, *, VBZ}. Improbable rules are rules that more often than not are actually grammatically correct, but could be incorrect, like the example: {NNS, VBZ}.

For some problems such as noun-verb disagreement, it's a simple matter to figure out that the system should be looking for a singular noun followed by a plural verb, or vice versa, but for something like the *they're, there, their* problem, it's more complicated. Some rules used for this system can be found at EduFind Online: `http://www.edufind.com/english/grammar/`, but require a subscription to use. While the definitions on EduFind Online are more like a grammar primer than a programmer's guide to grammar checking, the rules that it has are fairly comprehensive and can easily be converted to POS tag following rules. For example, the description given for nouns, in which the site lists rules for each different form of noun. The rules include which forms of verbs are correct which forms of nouns, as well as exceptions to each rule and example sentences for each rule.

This kind of rules matching for the English language can become very complicated, and for trickier grammatical errors, the process of defining specific rules can be very difficult. Trying to process higher

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

level/difficulty errors requires the test cases to be so specific that the entire point of having generalized rules is lost.

4.2 Random Rule Construction

To try to extend the kind of rules matching in Granska to a larger scale, the other method attempted for determining rules was random rules generation. Writing upwards of 100,000 rules by hand is a daunting process, so the system randomly assigns POS tags to rules of a user defined size. One issue with the generation process is that it could create a rule of size 5 such as {VB, VBG, VBP, VB, VBG}. While this rule is trivially incorrect, the fact remains that it is incorrect. Therefore, while this generation system does create rules that don't exist, it is always possible that a person will write a pattern that should not exist that must be marked as incorrect. However, the generator might make a rule such as: {NN, PP, *, NN, VB}, which is grammatically correct.

In order to remove all of the rules that reflect correct grammar, the system tests the randomly generated rules against a corpus of correct English and then eliminates all of the rules that generate flags in the corpus, thus leaving a set of tags that hopefully do not reflect proper grammar. Using sheer numbers, this method attempts to keep all of the rules that reflect whatever English isn't. This method takes away the issue of having to write out rules by hand at the expense of rule precision and transparency. For this method, each rule is weighted equally, and there are no specific, general, or improbable rule designations. This particular method also has a fairly long training process.

5 Results

One of the difficulties of the hand-constructed rules was actually measuring the effectiveness of the result. For a rule like noun-verb mismatch, it is very difficult to actually be able to tell how well the system can find errors, because it is easier to find corpora that are correct than corpora that intentionally make mistakes and then make note of those mistakes. The system was tested on 50 manually generated sentences that contained noun-verb mismatches followed by 50 sentences that were grammatically correct.

The specific rules for noun-verb mismatches flagged 22/50 of the incorrect sentences and 1/50 of the correct sentences as incorrect. General rules flagged 37/50 the incorrect sentences and 26/50 of the correct sentences. The improbable rules flagged 48/50 of the incorrect sentences and 39/50 of the correct sentences as incorrect. The precision and recall of the different rule types are shown in Figure 1.

The system was also tested on the *their/there/they're* and the *then/than* problem. Although these problems are actually contextual spelling errors, they can still be found with this system. The tests using these problems generated results similar to the noun-verb mismatch problem. The major issue with dealing with results for these specific rules, is that if a specific or general case doesn't trigger for a given data set, it is easy to simply write the rule that covers that particular problem, thus boosting the percentages. Having the system flag above 90% of the incorrect sentences with improbable rules is not an especially noteworthy or difficult task.

The random comparison algorithm was trained on approximately half of the translated Proust corpus from the Gutenberg Project, which totals approximately 100,000 words. For comparison, the system was trained for grammar rules containing two, three, four, five, and six POS tags (rules of size 2-6). Then the system was run on two grammatically correct paragraphs and a short message obtained from `http://faculty.washington.edu/sandeep/check/demofile.doc` that goes through Microsoft Word 2002 without causing any error messages. While the random system does trigger on both documents, it triggers at a significantly higher rate for the grammatically incorrect document, although it still misses a lot of rules.

Figure 2 shows a comparison of the number of rules triggered for a grammatically correct document versus a grammatically incorrect document. The results for Figure 2 are encouraging, as they suggest that the algorithm, despite not being perfect, has actually done something. For this graph, all of the duplicate rule triggers have been removed. At all levels, except for six where both are zero, the grammatically poor document has more triggers

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

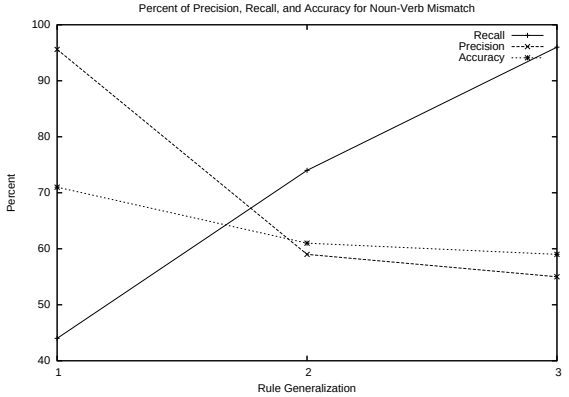


Figure 1: This graph shows precision and recall that each generalization of rule produced. On the x-axis, specific rules are 1, general rules are 2, and improbable rules are 3.

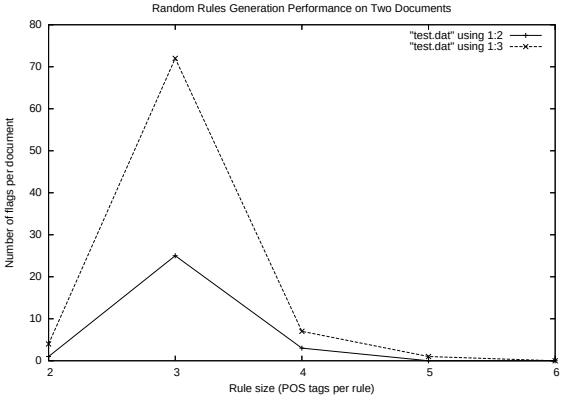


Figure 2: Random rules performance on two documents of 291 words. The upper line is the number of rule triggers for an intentionally incorrect message. The lower line is the number of rule triggers for two normal paragraphs of correct text.

than the grammatically sound document. So, while this method of random generation may not isolate rules, it may be a fairly decent measurement of the overall correctness of the grammar in a given document.

6 Conclusions

Grammar checking is not a simple problem. The Granska system works for two specific grammatical errors in Swedish, detecting them rapidly and accurately, and the two systems that were referenced earlier each had various problems that made them somewhat suboptimal. By writing out rules by hand, this system achieves results that are directly proportional to the number and accuracy of the rules that are written for a given problem. Some problems require more rules than others, but in order to hit every possible grammatical error this way, it is necessary to construct an unrealistic number of rules. A simple problem like noun-verb disagreement took this system 35 rules: 20 specific, 10 general, and 5 improbable. Describing something more complicated such as passive voice, or something more nebulous such as run-on sentences would require many more rules. On top of that, it is nearly impossible to tell if all of the rules of a given problem have been defined. On the other hand, the second random comparison method is dependent on many factors, such as the quality and size of the corpus that it's training on.

In conclusion, while it is possible to use a straight rules based system to create a grammar checker, it requires a large number of resources to create all of the rules necessary to properly define grammar problems. Using a random method obscures the rule creation process, but hopefully generates a rules set that has some bearing on what is grammatically incorrect. This system's random algorithm has a tendency to generate terminal cases that don't help define a grammar. By modifying the algorithm to include some stochastic processes, it may be possible to make the algorithm substantially better.

7 Future Work

The random comparison algorithm in this paper is fairly simple and could definitely use some adjustment. Currently, all rules that do not trigger on the training corpus are used while the rest are culled. This is a completely arbitrary decision, and it may be more effective to use a threshold greater than zero. Also, training from the Proust corpus was perhaps not the most efficient way to check for grammatically correct English. Generating rules from the Brown corpus and then testing them could generate very different results.

Finally, the main issue with a rules-based system is a lack of good ways to test it, short of having people purposefully write grammatically incorrect sen-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

tences and manually test them. This is both tedious and of limited use. There are a relatively few corpora that are intentionally incorrect and although the knowledge that the grammar checker won't misfire is useful, manual construction of rules can only be viable if there is a good body of data to test them on. Therefore, a good extension to this project would be to attempt to generate corpora that include grammatically incorrect sentences along with correct ones for the system to train on.

References

E. Bender, D. Flickinger, S. Oepen, A. Walsh, and T. Baldwin. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL/ICAL Symposium: NLP and Speech Technologies in Advance Language Learning Systems*.

E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. 1993. Equations for part-of-speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789.

R. Domeij, O. Knutsson, J. Carlberger, and V. Kann. 1999. Granska – an efficient hybrid system for Swedish grammar checking. In *Nordic Conference of Computational Linguistics*, pages 49–56.

S. Sofkova Hashemi, R. Cooper, and R. Andersson. 2003. Positive grammar checking: A finite state approach. In *CICLing-2003: Conference on Intelligent Text Processing and Computational Linguistics*.

K. Toutanova and C. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

Political Blog Analysis Using Bootstrapping Techniques

Fritz Heckel, Nick Ward
Department of Computer Science
Swarthmore College
Swarthmore, Pennsylvania, USA
{fwph,nward}@sccs.swarthmore.edu

Abstract

In the past few years, the form of Internet media known as "blogging" or weblogging has exploded, especially in the realm of politics. We propose and implement a system for performing qualitative text analysis of political blogs, with the ultimate goal of placing them on a map to categorize them according to political bias. Our system performed surprisingly well on the task of categorizing entire blogs, though the success is not entirely unqualified, and the system is not suitable for categorizing individual articles.

1 Introduction

Political bloggers are some of the most prolific writers of today's new media, generating thousands, if not millions, of articles a day. We can harness the sheer mass of blog articles to create a large and very dense corpus for use with machine learning techniques. Furthermore, as blogs are generally quite easy for a human to classify by political bent, it is effectively a self-labeled corpus.

Because of the sheer size of "the Blogosphere", it can be extremely difficult to navigate; information overload takes on a new meaning after spending several hours traversing the blog nets. We propose a system to help ease this task, by performing qualitative text analysis on blog articles. We hope to develop a system that not only categorizes blogs discretely, but also places them along a spectrum so that it is easy to compare different blogs with a glance.

We do not make any hypotheses as to the nature of the final blog classification results. We seek only some sort of document classification that reveals some interesting patterns in blogs. Whether those patterns manifest themselves as political affiliation, authorship, or some more subtle content-based qualifier, they will give some meta-information about a given blog.

2 Related Work

There has been effectively no work in the area of automatically classifying blogs based on their content. Most proposals focus on creating a pre-defined taxonomy of blog subject matter that would be integrated directly into RSS stream files as metadata; the current incarnation of such a system is a <taxo> XML tag that can be included in a blog's RSS feed (Beged-Dov et al., 2000). Note that this system requires manual entry of blog metadata by each blog author for every document that they create.

One example of a set of classifications that could be used with a taxonomy-based system has been proposed for blogs produced not by individuals but by corporations, businesses, or other organizations (Wackå, 2004). The author suggests that the primary division in this blog subdomain is between "External Blogs", which are used by the company or organization to promote their image, and "Internal Blogs", which are used by employees to collaborate and disseminate knowledge and company culture. The author mentions that any sort of top-down classification proposed for blogs, even within a constrained subdomain of "the Blogosphere", is doomed to failure simply because no one will agree

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

on what the fixed taxonomical classification standard will be. This is why automated classification systems are necessary.

Other extensions to existing blog metadata have been proposed, such as one to add Semantic Web-compatible content classifiers to existing RSS feeds (Karger and Quan, 2004). None of these methods appear to address the fact that every blogger would be required to conform to some sort of metadata standard in order to make their entries classifiable, nor that some bloggers might choose to intentionally mislabel their entries for some reason.

3 System Architecture

Our system is composed of three major parts: a data harvester, a training system for discovering domain-specific lexicons, and a categorization component. The first component uses Perl's XML::RSS and LWP::RobotUA modules to create a simple RSS aggregator which feeds entries into a MySQL database. The use of MySQL lets us continuously harvest blog articles from RSS feeds while simultaneously training our system, avoiding problems of file-locking and other race conditions, while the Perl modules give us pre-existing code to help us avoid reinventing the wheel.

The second component is based on the BASILISK system (Riloff and Thelen, 2002) created by Ellen Riloff. BASILISK first utilizes the AutoSlog (Riloff, 1996) system to generate extraction patterns from the training data, then takes a seed lexicon to discover a larger dictionary of words relating to a number of categories in the political domain. We re-used the top extraction patterns as templates to match in new documents; these combined with the words matching in the pattern will be used to create feature vectors for the categorization step.

The third component is based on a part of the SOMLib Digital Library system (Merkl and Rauber, 2000) created by the Department of Software Technology team at the Vienna University of Technology. Their unsupervised document classifier consists of a hierarchical feature map (HFM), a tree-like arrangement of several independent self-organizing maps (SOMs). The feature vectors derived from the second component will be used as input to these HFMs, which will be implemented using an existing SOM software module in the Python Robotics Project (Blank et al., 2002) (Blank et al., 2005). Developing in this pre-defined Python environment, with which both of us are experienced, sped the development of this component.

4 RSS Aggregation

We chose as sources for training data a number of high profile, high volume blogs for which we knew the political slant. The full list is shown in Table 1.

Table 1: Political Weblogs

Political Weblogs	
Category	Blog
Conservative	GOP Bloggers
	The Museum of Left Wing Lunacy
	Secure Liberty
	The Blue State
Liberal	Blog vs. Blog
	Pandagon
	Eschaton
	TalkLeft
	Kicking Ass

Most of these blogs are fairly high volume, and each one is clearly partisan. Over the course of 2 1/2 weeks, 962 blog articles were aggregated, totaling around 25,000 words. This is not a very large training corpus, but the unique nature of blogs would likely cause a larger corpus aggregated over a much longer time period to be far less useful: issues in blog-space may change rapidly, requiring retraining of the feature map on a regular basis.

5 Feature Training

We tried two different approaches to building features:

- Caseframe features (Sec. 5.1)
- Lexical features (Sec. 5.2)

We expected lexical features to be far more successful, as caseframes would tend to be more general,

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

and not necessarily even strongly related to the domain (ie, *he said*). Lexical features, on the other hand, would be words which tended to show up frequently throughout the corpus.

AutoSlog builds caseframes based on a number of simple heuristic patterns such as <**subj**> **passive-verb**, which will generate caseframes like <**person**> **gave**. When using AutoSlog-TS, these extraction patterns are generated for each noun phrase in each set of texts. Based on the number of times a pattern is found in the relevant texts and not in the irrelevant texts, each pattern is assigned frequency values. These can then be used to provide a probability that a caseframe is found in the text of an interesting domain. This serves as a score, and a number of patterns can be chosen based on their high scores. AutoSlog extracts patterns for each word specified in a target dictionary, rather than for every word in the corpus as AutoSlog-TS does. Frequencies are not calculated for these caseframes; instead they are used for extracting additional words, and the words are scored based on frequency.

5.1 Caseframe Features

To build caseframe features, or *extraction patterns*, we used AutoSlog-TS. AutoSlog-TS is capable of generating extraction patterns from text with no supervision. Our text corpora were composed of about 900 entries from the blogs mentioned above, totaling about 25,000 words, and an unrelated text composed of samples from the Corpus of Professional Spoken American English[1]. The samples from CPSA totaled about 75,000 words; ideally, we would have used a more balanced corpus.

Caseframes are extraction patterns: generally, they are composed of a verb phrase (or partial verb phrase) with one or more slots for an associated noun phrase. From these corpora, AutoSlog generated 855 caseframes which we used as individual buckets in a feature vector. By calling Sundance on each blog entry, we were able to find the number of times each extraction pattern occurred in the text; these values were used to build the actual feature vector which could then be fed into the SOM for training.

[1] http://www.athel.com/cpsa.html

5.2 Lexical Features

To build a lexicon for the domain, we started with a number of seed noun phrases in several categories relating to politics. Table 2 shows examples of categories and seed noun phrases for each. Ultimately, we found just seven seed noun phrases to be sufficient: *Bush, President, security, terrorism, terror, Congress*, and *Senate*.

Building the full lexicon then followed a simplified variant on the BASILISK method(Riloff and Thelen, 2002).

- Run the training corpus through AutoSlog, using the seed lexicon to generate extraction patterns
- Use Sundance with the new extraction patterns to discover additional lexicon candidates
- Choose some number of candidate noun phrases to add to the lexicon
- Remove common noun phrases from the list (such as he, she, this, etc.)
- Repeat until the lexicon size stabilizes.

We simplified the method by placing a threshold on the number of occurrences necessary for noun phrases to be added to the lexicon, rather than using a full probabilistic method. Using a threshold of five occurrences, we found that after six iterations of the algorithm, the lexicon had stabilized at 254 noun phrases for our training set. Some of the top noun phrases were, unsurprisingly, *Congress, Senate, and George Bush*. Other, more loaded in context, were *Pro-Choice President* or *Pro-Life President*.

Once again, the feature vector was created by counting the number of times the features– this time, the members of the lexicon– occurred in each blog entry. This vector's buckets represented number of occurrences of words instead of extraction patterns, and the feature vectors were fed into the map as with the extraction pattern features.

6 Categorization

6.1 Self-Organizing Maps

A self-organizing map (SOM) consists of a two-dimensional grid of nodes, each of which is initialized with a random vector in the feature space.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

Table 2: Categories and Seed Words

Categories and Seed words	
Category	Seeds
Issues	Social Security
	Iraq War
	Election Reform
	Gay Rights
Parties	Democrats
	Republicans
	GOP
Figures	George W. Bush
	John Kerry
	Paul Wolfowitz

The training set for a single SOM consists of a large number of vectors distributed throughout the feature space; therefore the SOM will effectively "learn" a simple clustering of that space. A conceptual depiction of a newly initialized 4x4 SOM can be seen in Figure 1. Note how the mapping of each SOM unit into the feature space is arbitrary.

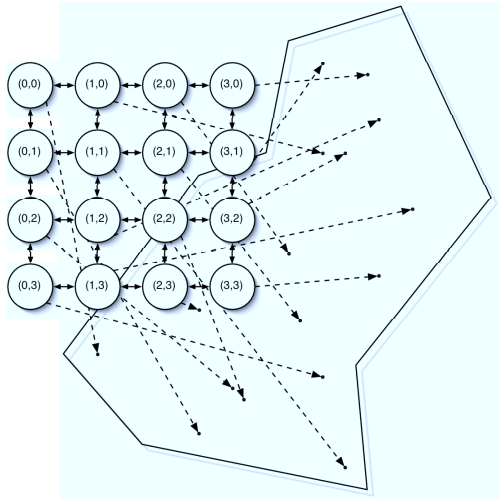


Figure 1: The units of a self-organizing map (SOM) in the standard two-dimensional grid configuration and their initial mapping into a feature space.

As each vector from the training set is input into the SOM, the unit whose state is closest to the input in the feature space by Euclidean distance "wins". The unit, and with some fall-off its neighbors, has its state adjusted to be closer to the input vector. After training, the nodes or units in the SOM will have clustered the feature space. The neighborhood function h_{ci} is given in Equation 1, where $\|r_c - r_i\|$ is the distance in feature space between two units' vectors, and $\sigma(t)$ is the width of the Gaussian. $\|r_c - r_i\|$ is merely a representation of whatever arbitrary distance metric is selected for a particular SOM implementation; no vector subtraction necessarily occurs. In our system, we chose to use standard Euclidean distance in the feature space, since we could guarantee that all of our input vectors would be the same length. $\sigma(t)$ decreases with time, so after many training iterations only the winning unit's position in feature space is updated (Merkl and Rauber, 2000).

$$h_{ci}(t) = e^{-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}} \quad (1)$$

By adjusting not just the best-match unit but also its neighbors, and by decreasing the influence on neighbors over time, the SOM units tend to detect clusters in the feature space. Hopefully, the end positions of the unit vectors subdivide that space in an interesting way.

The output of each unit in an SOM is associated with at most one cluster, although multiple units may represent a single informative cluster. The interpretation of the result of the training is entirely up to the user; the SOM can only give a clustering in terms of the feature space, as it does not "know" anything about the problem domain. In our case, feature vectors represent individual blog entries, so the clusters are interpreted as representing some abstract grouping of entries.

6.2 Hierarchical Feature Maps

The number of clusters is highly dependent on the architecture of an individual SOM. It can only cluster the feature space into at most as many clusters as it has units. This is where a hierarchical feature map (HFM) becomes useful: by using one SOM to divide the feature space into smaller sub-problems, each of those sub-problems can in turn be clustered by an SOM. The result is a highly specific clustering of the feature space.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

A typical HFM consists of a tree structure in several layers, as shown in Figure 2. The topmost layer of the HFM, and the root of the tree, consists of a single SOM. The individual units of an SOM at each layer pass feature vectors onto an entire SOM in the next layer down, all the way down to the base of the HFM.

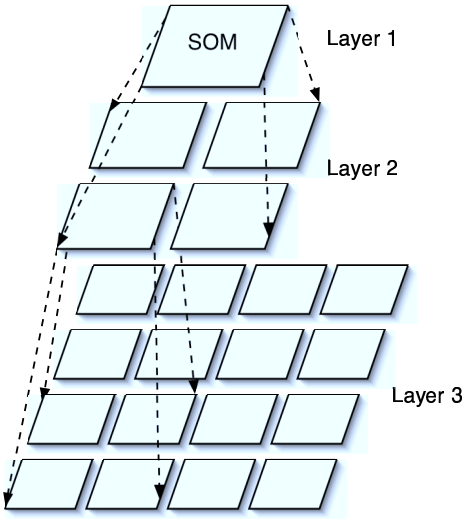


Figure 2: A hierarchical feature map (HFM) consists of multiple SOMs in a tree-like structure

To speed training and improve the accuracy of the results, the dimensionality of the feature space can be reduced between each layer on a unit-by-unit basis. If all of the input vectors that match have similar values along one feature space dimension, that dimension can be eliminated before passing the training subset onto the next SOM layer.

Note that it is possible for the dimensionality of the feature subspace being handled by different SOMs on the same HFM layer to vary. Some SOM units higher up in the HFM tree may be trained to make big clustering decisions that significantly reduce the dimensionality, while others may make no change to the dimensionality and pass vectors directly to their child SOMs.

Once the HFM has been trained, using it is simply a matter of inputting a feature vector. At each layer, the "winning" unit of the SOM will pass the vector down to its child SOM, until the bottom of the HFM tree is reached. The bottom-most units of an HFM are each interpreted as having some cluster-related meaning, based on the feature vectors.

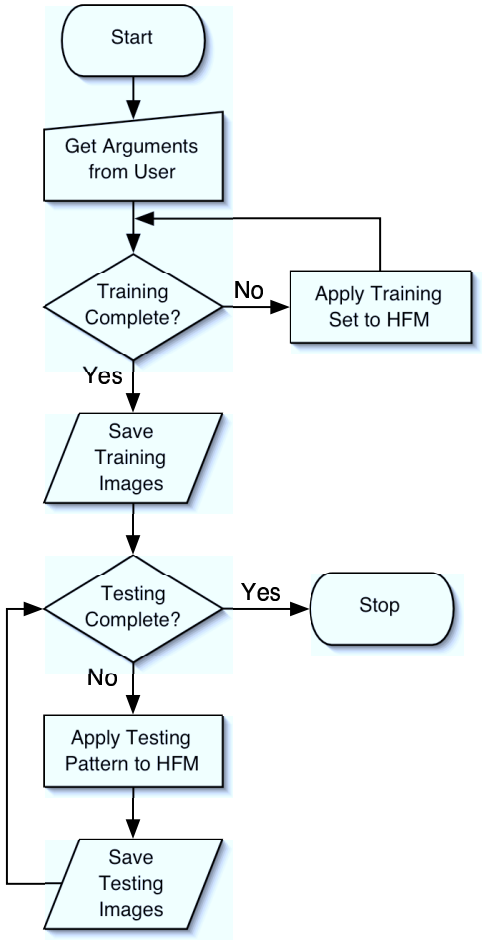


Figure 3: The training and testing process for an HFM

The complete process for training and testing a single HFM can take many iterations, depending on the size of the training and testing feature vector sets. The flowchart in Figure 3 demonstrates this procedurally.

6.3 Implementation

The blog aggregating software was written in Perl, using a MySQL database for storing information. Our spider used the XML::RSS and LWP::RobotUA Perl modules to fetch and parse blog RSS feeds. Some slight modifications to portions of the Sundance package were necessary to fix out of date code, and the process of running the two feature training algorithms was automated with a number of

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

Bash and Perl scripts, while feature vector information was generated using Perl.

Our implementation of HFMs was written entirely in the Python programming language. We chose to use Python so that we could use the existing SOM implementation written by Daniel Sproul '03 that is included as part of PyRo, the Python Robotics package (Blank et al., 2002).

The main functional unit of the system is the HFMNode class, which simply contains an SOM instance and a 2-D list of child HFMNodes. Because of the way in which the outputs of the SOM units at each HFM layer are passed on to the next layer, as described above in Section 6.2, the SOM and the list of child nodes must be the same size.

The HFMNodes are contained with the HFM class, which is merely a convenience class that holds a single HFMNode as the root of the HFM's tree. The HFM class also contains file I/O functionality, for reading input feature vectors generated during the lexical training steps described in Section 5.2.

6.4 Visualization

In order to examine the training and testing process, we needed some intuitive way of displaying the output of an HFM. Pyro's SOM implementation did have some visualization capability, but it was for live observation only, and suffered from data overload. In addition, our HFM implementation abstracted away from the SOM class to a large extent, so it was necessary to develop our own visualizer.

Each layer of a given HFM training or testing run can be output as a grid. This allows us to observe both the final categorization and the initial clustering decisions made by the SOMs in the lower-resolution layers of the HFM. Each grid cell in the output image represents one of the SOMs in that layer. The points drawn within cells are color-coded by blog, allowing us to distinguish the output. Each point represents a single tested or trained feature vector.

7 Results

To obtain our final results, we trained a 3-layer HFM consisting of 2x2 SOMs. The HFM was trained for 150 iterations using the entire corpus of blog entries from the nine blogs listed in Table 1. We then examined the testing results for each of the blogs individually.

The results from the caseframe features were unremarkable at best– the caseframe features did not extract a sufficient amount of information to cluster the blog entries in any manner. It is not necessary to cover that method any further, so the remainder of this section refers to our results using lexical features.

7.1 Training

Figures 4 and 5 contain examples of the output from the deepest layer of the HFM after training on the entire blog entry corpus. The primary feature of note is that the vast majority of the entries clustered along one of the diagonal axes of the HFM. The diagonal axis is inconsistent between training runs because the SOMs in the HFM have their initial positions in the feature space set randomly at runtime.

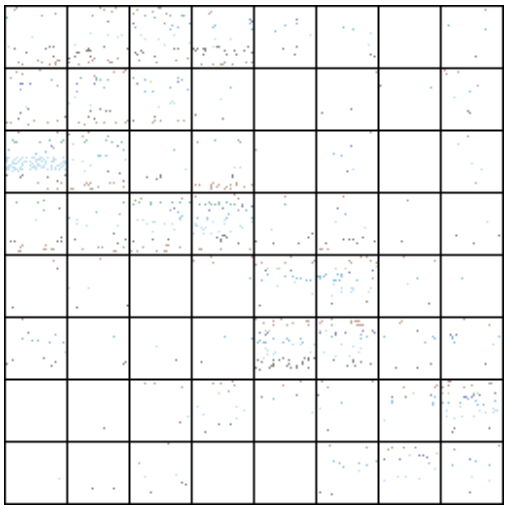


Figure 4: HFM training output for Run 1

The distributions of the training set are very similar between the two separate training runs, which implies that our HFM is probably learning the same distinguishing features. Note for example the strong cluster from a single blog that appears in cell (0, 2) of Figure 4 and in cell (7, 3) of Figure 5 as a denser stripe of points. This cluster from a single blog was assigned its own bucket between runs. The fact that we have consistent training of a randomly-initialized neural network structure is a very good sign that our testing results have some non-trivial meaning.

We wish to reiterate the somewhat black-box na-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

ture of network-based learning methods. These results are very much open to interpretation, although we believe that the patterns that can be seen in our results are not just mere chance.

7.2 Testing

The distribution of blogs shown in Table 3 is associated with the second training run shown in Figure 4. With the exception of Pandagon and The Museum of Left Wing Lunacy, the lower-left contains conservative blogs and the upper-right contains liberal blogs. It should be noted that this table reflects the primary concentration of each blog's testing output. Instapundit and Talking Points Memo are two blogs that were exclusively in our testing data set.

If we examine the results more closely, it becomes clear that our HFM did succeed in performing the basic liberal/conservative classification task. Pandagon is a particularly unique blog, and had the most diffuse results from the HFM. It is only barely concentrated with the conservative blogs, even though it is a liberal blog. There is not a strong explanation for why Pandagon emerged differently, though it is worth noting that the tone and nature of the articles on Pandagon are rather unique.

The Museum of Left Wing Lunacy, as an unashamedly conservative blog, was classified with the liberal blogs. However, a quick examination of their entries shows that they primarily quote other blogs, and mostly liberal blogs at that. This means that the plain text version which we analyzed contained mostly liberal-leaning text. This result might be avoided if we removed all quotations from the input entries.

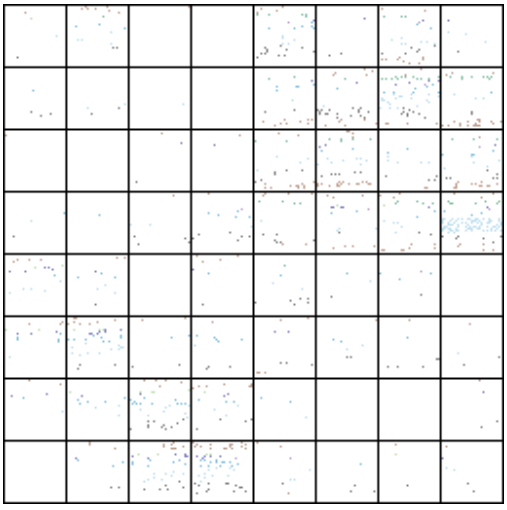


Figure 5: HFM training output for Run 2

Table 3: HFM output

HFM Output	
	Left Wing Lunacy (C) Blog vs. Blog (L) Eschaton (L) TalkLeft (L) Kicking Ass (L) Talking Points Memo
GOP Bloggers (C) Secure Liberty (C) Blue State Conservatives (C) Pandagon (L) Instapundit	

8 Future Work

Our implementation of BASILISK was somewhat less sophisticated than the original method, as we took a simpler approach to choosing words to be placed in the lexicon. Because many blog articles are very short– though may still contain a great deal of information– it seemed more important to avoid creating an overly small lexicon than one that was too large.

The original design of this project would have used both lexicon and extraction pattern data to generate the features, but we had difficulty in finding a feature representation which could concisely represent all of this data in a form which could provide useful results from the SOM. Feature maps seem to have served well in this task, though we do believe that our method could be refined significantly. We are not entirely satisfied with the feature vectors as they stand.

In addition, before calling this an unqualified success, further testing with a larger corpus must be performed.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

9 Conclusion

The combination of BASILISK and Self-Organizing Maps worked surprisingly well for this project. Given the ultimate sparsity of our feature vectors, we did not expect to achieve the level of performance that we did. Further exploration of this combination would certainly be worthwhile in the future.

References

Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopf, J., Galbraith, D., Guha, R.V., MacLeod, K., Miller, E., Swartz, A., and van der Vlist, E. (2000) "RSS 1.0 Modules: Taxonomy", RDF Site Summary 1.0 Modules, 20 Mar 2001, RSS-DEV Working Group, 01 Apr 2005, <http://web.resource.org/rss/1.0/modules/taxonomy/>.

Blank, D.S., Kumar, D., and Meeden, L. (2002) "Python robotics: An Environment for Exploring Robotics Beyond LEGOs", ACM Special Interest Group: Computer Science Education Conference, Reno, NV (SIGCSE 2003).

Blank, D.S., Kumar, D., Meeden, L. and Yanco, H. (2005) "Pyro: A Python-based Versatile Programming Environment for Teaching Robotics". To appear in the ACM Journal on Educational Resources in Computing (JERIC).

Karger, D. and Quan, D. (2004) "What Would It Mean to Blog on the Semantic Web?", International Semantic Web Conference 2004.

Merkl, D. and Rauber, A. (2000) "Document Classification with Unsupervised Neural Networks", Soft Computing in Information Retrieval, 2000, pp. 102-121.

Thelen, M. and Riloff, E. (2002) "A Bootstrapping Method for Learning Semantic Lexicons using Extraction Pattern Contexts", Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002).

Riloff, E. (1996) "Automatically Generating Extraction Patterns from Untagged Text", Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96) , 1996, pp. 1044-1049.

Wackå, Fredrik, "Six Types Of Business Blogs - A Classification" [Weblog entry], CorporateBloggingBlog, 10 Aug 2004, <http://www.corporateblogging.info/2004/08/six-types-of-business-blogs.asp>, 01 Apr 2005.

Kicking Ass, http://www.democrats.org/blog/

TalkLeft, http://www.talkleft.com

Eschaton, http://atrios.blogspot.com/

Pandagon, http://www.pandagon.net/

Blog vs. Blog, http://blog.battletothedeath.net

The Blue State Conservatives, http://www.radiobs.net/thebluestateconservatives/

Secure Liberty, http://secureliberty.org/

The Museum of Left Wing Lunacy, http://www.museumofleftwinglunacy.com

GOP Bloggers, http://www.gopbloggers.org/

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

Developing a Morphological Segmenter for Russian

America L. Holloway
Swarthmore College
Swarthmore, PA 19081
ahollow1@swarthmore.edu

Abstract

This paper presents an algorithm for developing a morphological segmenter for Russian. The segmenter can find multiple prefixes and suffixes for any given word. Therefore it is more suitable for a highly inflected language than a segmenter that is limited to at most one prefix or suffix. The segmenter requires a small hand segmented corpus to bootstrap from, and a larger unsegmented corpus from which to learn. The algorithm uses trigram probabilities, and Witten-Bell smoothing to predict the correct segmentation of a word. A filtering step is also used to weed out bad segmentations.

1 Introduction

Many languages, including Russian and Arabic, have a richer morphology than is found in English. In Russian, not only do verb endings change to reflect person, gender and number, noun endings also change (or in some cases are truncated) to reflect case. For example, the ending а is appended to a masculine noun to form the genitive singular. Furthermore, a word in Russian can often times be decomposed into smaller units, or morphemes, each of which carries its own meaning. These morphemes contribute to, and refine, the meaning of the entire word. For example, the noun председатель (*predsedatil*) means 'representative'. Literally, it can be translated as 'the one' (ель) 'who sits' (сидеть) 'before' (пред), or more figuratively, 'the one who represents' us. In general, it is not rare for a word to have multiple prefixes and suffixes. Multiple suffixes, in particular, are common. As an example, reflexive verbs will always have two suffixes. The first suffix -ся (*-cya*) indicates it is a reflexive verb, and the second suffix indicates what type of verb it is. These suffixes include -овать (*-ova*), -ать(*-at*) and -ить (*-ut*). Thus, to capture the morphology of such a language, it is important that any morphological analyzer be able to recognize multiple prefixes and suffixes.

The algorithm presented in this paper is adapted from the morphological segmenter for Arabic created by (Lee et al., 2003). Many existing morphological analyzers, for example (Goldsmith, 2000), identify only single suffixes. This type of system fails to capture the entire morphology of Russian. Recognizing multiple prefixes and suffixes is especially important for tasks such as aligning corpora, information retrieval and machine translation. This is because one Russian word may correspond to multiple words in a different language. Thus, our goal was to implement a segmenter that (1) could identify multiple affixes and (2) required few resources, in order to create a superior morphological analyzer specifically for Russian.

Our system requires only a small hand-segmented corpus to bootstrap the segmenter, and a larger, unsegmented corpus from which to gain new stems. For any Russian word, all possible segmentations are enumerated and the trigram probability of each is computed. The highest scoring segmentation is chosen as the correct one. The system performance is

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

surprisingly good given the small corpus and simple algorithm.

2 Related Work

As stated, this algorithm draws heavily from (Lee et al., 2003). They present a morphological segmenter for Arabic which identifies multiple prefixes and suffixes and requires only a small hand segmented corpus (110,000 words) and a large unsegmented corpus (155 million words). They also supplement their segmenter with an additional prefix/suffix list. The large unsegmented corpus is used to acquire new stems. They first divide the corpus into partitions. For each word, all possible segmentations are enumerated and the segmentations with the highest probabilities are kept. After each partition, the trigram probabilities are recomputed to take into account the new stems found. Each stem is also subjected to further testing to ensure that it does not contain a prefix or suffix. Stems are added to the list based upon the stem frequency (i.e. the number of times they are seen), the probability that a substring of the stem is a prefix or suffix, and contextual information. With just trigram probabilities alone, (Lee et al., 2003) are able to reduce the error from the baseline performance by half.

(Goldsmith, 2000) uses the notion of minimum description length (MDL) to implement a morphological segmenter, *Linguistica*, that is quite successful. *Linguistica* takes only a corpus and returns a list of stems, a list of suffixes and a list of signatures. A signature is a set of suffixes which can appear on the end of a stem. An example signature is the set (-NULL, -s). There are many stems, such as *apple* or *cow*, that are associated with this signature. A first analysis of the corpus can be as simple as splitting every word after each letter. Other heuristics are then employed to shrink the list of signatures.

Minimum description length is based on the notion that the number of letters in the morphological analysis of a corpus (e.g. a list of stems, suffixes and signatures) will be less than the number of letters in the original corpus. Accordingly, Goldsmith develops a description length to measure the size of the morphological analysis of a corpus. That is, he creates a description length to measure the size of the stem list, suffix list and signature list. After each heuristic is applied, the description length is computed. If the description length has decreased, the analysis is kept. Notably, *Linguistica* identifies only one suffix per word. For example, if the word *breathings* occurred in our corpus, the stem would be *breathing* and the suffix would be *-s* [1]. Thus *breathings* would be associated with the signature given above. Recall of 85.9% and precision of 90.4% is achieved for English.

Work using multilingual corpora to aid in morphological analysis has also been performed. (Yarowsky et al., 2001) use a lemmatizer and multilingual corpora to achieve a precision over 98% on a French corpus of 1.2 million words. (Hana et al., 2004) use a Czech-Russian aligned corpus. The system combines information from their own morphological segmenter, the Czech corpus and a part of speech tagger. Instead of detecting multiple prefixes or suffixes, they use the notion of paradigms. A paradigm is a list of suffixes, along with the corresponding part of speech, that can be appended to a certain class of stems. One interesting technique used to find the correct suffix of a word is the longest-suffix approach. Simply put, the correct suffix is usually the longest one. We have adopted this heuristic to increase our system performance.

3 Morphological Segmenter

3.1 Parsing Words

Before discussing the algorithm used to build the morphological segmenter, it is important to discuss what constitutes a prefix or a suffix. Two categories of suffixes are distinguished by the segmenter: suffixes that change the part of speech, and suffixes that preserve part of speech, but reflect a change in case, or person.

As an example of the first type of suffix, consider the suffix -ение (*-enie*). This is appended onto the end of a verb to form the corresponding noun. Hence, the noun обсуждение (meaning 'discussion') is derived from the verb обсуждать (meaning 'to discuss'). To then form the genitive or possessive form of the noun, the ending -ие (*-iye*) changes to -ия (*-iya*). This is an example of the second type of suffix which preserves the part of speech.

[1] This example is taken from (Goldsmith, 2000)

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

count			prefix	stem	suffix(es)
7	&	&	#N	угл	+ов
16	&	&	#N	каз	2+ан +ие
17	&	&	#N	завод	+N
23	&	&	#раз	дел	+ить
29	&	&	#бес	платн	+ые
35	&	&	#N	север	+е
14	&	&	#N	тысяч	+ами

Table 1: Morphologically Segmented List of Russian Words

prefix	stem	suffix
N	заработк	е
N	заработке	N
за	работк	е
за	работке	N

Table 2: All possible suffix-prefix segmentations

In general, noun or adjective prefixes are harder to discern than verbal prefixes or suffixes. A verbal prefix is often used to denote aspect. However with nouns (and adjectives) a prefix neither changes the part of speech, nor the case, person or number. Instead we chose to define a prefix as a morpheme that refines or adds to the meaning of the word. For example, appending the preposition без (meaning 'without' or 'short of') to the adjective умный (meaning 'of the mind') gives the adjective безумный which means 'crazy'. In general however, the presence of a preposition at the beginning of a word does not necessarily mean it is acting as a prefix. Thus our method of determining prefixes for nouns and adjectives is inherently subjective. To account for this, when creating the small hand segmented corpus, a verb was determined to have a prefix if it was shown to have one in the Oxford Russian Dictionary. Nouns were determined to have a prefix, again, if a prefix was shown in the Oxford Russian Dictionary, or if it was clear from the meaning. The subjective nature of determining whether or not a noun contains a prefix actually hurt the performance of the segmenter and is discussed in the Results section.

3.2 Bootstrapping

A small hand segmented corpus of 474 Russian words was used to bootstrap the segmenter. Each word was split into prefix(es), stem and suffix(es). We adopt the convention that a pound sign (#) precedes every prefix, and a plus sign (+) precedes every suffix. In order for every word to have at least one prefix and suffix, the letter N is used for the null prefix and suffix. Finally, for the purpose of calculating trigram probabilities, two symbols (& &) were placed at the beginning of each word. Table 1 shows a sample of the corpus used to bootstrap the segmenter. From the corpus we create a static list of suffixes and prefixes, and a list of stems to which we will be adding. The smaller corpus is also used for initial trigram probabilities.

3.3 Building the Segmenter

The larger corpus consists of approximately 40,297 words and is split into 403 partitions of 100 words each. The number of words in the partition was arbitrarily chosen. We first read in an entire partition. Then for each word w, all possible segmentations of w are enumerated, and the probability for each segmentation is calculated. Only the segmentation with the highest probability is kept. The stem is then added to a list of possible stems. When the frequency (i.e. the number of times the stem has been seen) passes a given threshold, the stem is added to the list of accepted stems. Since the larger corpus is relatively small, the threshold value was set at 2.

3.3.1 Segmenting Words

Given any Russian word w, we wish to find all possible prefixes and suffixes of w. To find all possible prefixes of a given Russian word w, we compare substrings of w against the list of prefixes. The first substring is simply the first letter of w. The next substring is the first two letters of w, then the first three, and so on, until we come to the end of the word. We do the same for suffixes except we begin at the last letter of w. We then enumerate all possible prefix-suffix combinations. The null prefix (suffix) is always a possible prefix (suffix) for every word. Table 2 shows all the prefix-suffix combinations for the word заработке (*zarabotke*), the prepositional form of the word заработок meaning 'earnings'.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

3.3.2 Filtering

Often longer suffixes include within them shorter suffixes. For example, the word живому (*zhivomy*) has two possible suffixes: -ому or -у. In general however, the longest suffix is usually the correct one. A suffix on the end of a word of length 5 is more likely to be the correct one, than a suffix that is only of length 1. Thus, we give preference to longer suffixes. If a word has one (or more) compound suffixes, we consider only the compound suffixes and disregard any other possible segmentation of the word with only one suffix (including the null suffix).

We also provide to the system a list of 8 default suffixes. If a word contains one of these suffixes, all other segmentations of the word are disregarded except for this one. Hence there will be only one segmentation for the word, the segmentation with the default suffix.

In Russian, certain word endings will almost always indicate a suffix. For example, the genitive ending for masculine adjectives is ого (*-ovo*). An adjective will never have this ending unless it is in genitive case, and very few nouns have this ending. So few, that it is worth making ого a default suffix.

3.4 Probabilities

Given any Russian word w and any possible segmentation of w into morphemes $m_1 m_2 m_3 ... m_k$, the probability of the segmentation is given as:

$$P(\&) * P(\&|\&) * P(m_1|\&\&) * ... * P(m_k|\&\& m_1 ... m_{k-1}) \quad (1)$$

We can simplify this expression using a second-order Markov assumption. This makes computing the probability of morpheme m_i easier, since the probability of seeing m_i can be estimated given the previous two morphemes instead of all preceding morphemes. Also, Since every word begins with &&, we can consider $P(\&)$ and $P(\&|\&)$ to be constants and thus disregard them. This gives

$$P(m_1|\&\&) * P(m_2|\& m_1) * ... * P(m_k|m_{k-2} m_{k-1}) \quad (2)$$

We use the maximum likelihood estimate (MLE) shown below to calculate $P(m_i|m_{i-2} m_{i-1})$.

$$P(m_i|m_{i-2} m_{i-1}) = \frac{C(m_{i-2} m_{i-1} m_i)}{C(m_{i-2} m_{i-1})} \quad (3)$$

Witten-Bell discounting (Witten and Bell, 1991) is used for smoothing. The probability of seeing $m_{i-2} m_{i-1} m_i$ for the first time can be approximated by the number of times we saw previous trigrams for the first time. Let $m_{i-2} m_{i-1} m_i$ be a trigram that has never before been seen. Then $P(m_i|m_{i-2} m_{i-1})$ can be expressed as:

$$P(m_i|m_{i-2} m_{i-1}) = \frac{T}{Z(N+T)} \quad (4)$$

where T is the number of unique trigrams observed before, N is the total number of trigrams seen before, and Z is the number of zero trigrams. The probability of seeing $m_{i-2} m_{i-1} m_i$ is given by the number of previous times we saw a trigram for the first time (T) divided by the number of times a new trigram could have have been seen for the first time (N+T). We then distribute this probability evenly to all of the zero trigrams by dividing by Z. Since we need to know the value of Z in advance, we must read an entire partition first, segment all the words, and keep track of how many segmentations result in a stem that has never before been seen.

4 Results

To evaluate the segmenter, the hand tagged corpus was split into 9 different sets. Each set contains a different 50 lines from the corpus to test on, and the remaining 428 lines from which to train. Thus, the first set used the first 50 lines from which to test, the second set used the second 50 lines from which to test, and so on. The last set, set 9, was tested on the last 77 lines.

The segmenter was trained on the hand tagged corpus, and then asked to segment the appropriate 50 lines. The segmenter was evaluated according to recall and precision. Table 3 shows the performance of the segmenter on sets 1 through 9. The first column shows the recall of the segmenter (i.e. of the correct prefixes and suffixes, how many did the segmenter find). The second column shows the precision (i.e. of the prefixes and suffixes postulated by the segmenter, which were correct). The third and

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

	including null		excluding null	
Test Set	Recall	Precision	Recall	Precision
1	77.6%	81.90 %	75.6%	76.92%
2	90.52%	92.85 %	81.25%	78.94%
3	78.26%	83.33 %	64.29%	78.95%
4	74.55%	75.93 %	62.69%	72.73%
5	81.65%	86.53 %	87.93%	79.17%
6	81.65%	86.53 %	87.93%	79.17%
7	82.20%	86.61 %	68.57%	81.67%
8	82.46%	84.55 %	78.13%	77.19%
9	85.45%	83.03 %	77.42%	81.18%
Average	**88.74%**	**84.58 %**	**75.98 %**	**78.43 %**

Table 3: The first two columns show recall and precision when the null prefix/suffix is included. The last two columns show recall and precision disregarding the null prefix/suffix

fourth column show the recall and precision without taking the null prefix and suffix into account.

4.1 Discussion of Errors

Given the small size of the training corpora, and the simple nature of the algorithm, the results are encouraging. A majority of the mis-segmentations stem from a few key errors. One of the biggest problems was the small size of the hand tagged corpus. A few stems were seen once or twice and hence the corresponding suffix had an extremely high probability. For example the suffix -ь was seen only once with the word лечь. Since the probability of a segmentation was determined using trigram counts (Equation 3), the probability of the suffix -ь was 1.

One disheartening result is that the segmenter failed to find any prefix save one. However, since there were so few prefixes in the hand-tagged corpus, performance was not hurt too drastically. The poor prefix performance can be attributed to the subjective nature of prefixes. A word was considered to have a prefix if (1) it was shown with a prefix in the dictionary, or (2) the prefix contributed to the meaning of the word and taking away the prefix gave another related word. Thus, two words w_1 and w_2 may both have the same first two letters, yet only w_1 has a prefix. This, combined with the small corpus size and overwhelming probability of the null prefix, accounts for the system's preference for the null prefix.

5 Conclusion and Future Work

The segmenter does surprisingly well taking into account the small corpora size and the rather simple algorithm. In general, it is easy to detect a majority of suffixes, either because they are very unique, or because they are rather long. It is a small subset of suffixes such as -о, -е and -а that are difficult to identify. Thus, focusing on identifying these suffixes would result in major system gain. Another area of interest is a more uniform way of segmenting words into prefix(es), stem and suffix(es). In particular, changing the method of prefix identification so that every word with a particular first few letters are considered to have the same prefix, even if this prefix does not contribute to the meaning of the word.

6 Acknowledgments

We would like to thank Nastassia Herasimovich for helping segment Russian words and Professor Wicentowski for all of his (very much needed) help.

References

Goldsmith J. 2000. *Unsupervised learning of the morphology of a natural language* Computational Linguistics, 27(1).

Hana J., Feldman A. and Brew C. 2004. *A Resource-light approach to Russian morphology: Tagging Russian using Czech resources.* Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

Witten I.H. and Bell T.C. 1991. *"The Zero-Freqency Problem: Estimating the probabilities of novel events in adaptive text compression"* IEEE Transactions on Information Theory, 37(4), p. 1085-1094.

Yarowsky D., Ngai G. and Wicentowski R. 2001. *Inducing Multilingual Text Analysis Tools via Robust Projection across Aligned Corpora.* Proceedings of the HLT 2001, pages 161-168.

Lee Y., Papineni K. and Roukos S. 2003. *Language Model Based Arabic Word Segmentation.* Proceedings of the 41st Annual Meeting of the Association, pages 399-406.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 28–33
Computer Science Department, Swarthmore College

Report on Political Leaning Classification

Ben Mitchell & Zach Pezzementi
Computer Science Department
Swarthmore College
Swarthmore, PA 19081
{mitchell,zap}@cs.swarthmore.edu

Abstract

The tasks of document classification and sentiment classification have been explored in the literature, but to our knowledge the task of political classification has not. We use a modified form of a document classification algorithm (Hu and Liu, 2004) to classify newspapers as liberal, conservative, or neutral based on their text. By using a cosine similarity metric in our feature space, we were able to achieve distances that separated openly liberal from openly conservative papers. According to the same metric, we found Time and Newsweek to be fairly centrist, as their distances from liberal and conservative papers were about the same, while the Chicago Tribune displayed a distinct liberal bias. This feature space shows promise for further sentiment or document classification work.

1 Introduction

Document classification is the task of grouping a set of documents based on their content, usually into a fixed number of predefined categories. Document classification schemes have been developed for use in specific domains, such as classifying news stories (Yang et al., 1999) or grouping web posted job openings (Cohen and Hirsh, 1998), as well as more generic algorithms designed to work across many domains (Schohn and Cohn, 2000). The classes are usually broad topics, picked in advance (for example, classifying sports articles as being about baseball, football, or basketball). A fairly simple Bayesian bag-of-words model has been shown to be successful in document classification tasks (Baker and McCallum, 1998).

Sentiment classification attempts to group documents according to the sentiment of the author with respect to the subject. Most previous studies have defined the sentiment classification task as integrating aspects of document classification and text summarization (Fei et al., 2004; Hu and Liu, 2004; Pang et al., 2002). The goal is typically to classify each document (often a product review) as being a member of one of two classes, either positive or negative, though attempts at more complex classification schemes have been made(Yi et al., 2003).

We expected the problem of political sentiment classification to require somewhat different techniques from those used in document classification or standard sentiment classification. Firstly, in typical sentiment classification tasks, the text used as input is written specifically to communicate the information the algorithm is trying to extract. A product review, for example, is written with the intention of expressing the sentiment of the reviewer with respect to the product being reviewed. The sentiment we are trying to detect, on the other hand, is not necessarily stated explicitly within the text. Similarly, most document classifiers need only identify the main topic of a passage in order to make their classification decision, whereas we specifically want to avoid distinguishing between articles based primarily on their main topic. To help avoid classifying based on con-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

Nation	Freq	*National Review*	Freq
dlc	61	guevara	33
un	60	u.n.	29
durbin	42	gannon	26
henry	40	official	26
trotsky	39	chavez	23
falluja	35	pollack	22
guernica	32	kim	20
deutscher	31	ortega	19
nevada	28	mithal	15
women's	25	post-war	14

Table 1: **Top Ten Most Frequent Words Which Occur in Only One Corpus**

tent, we limited our data to articles on a single topic; we chose the United States' war in Iraq as a topic since it was frequently in the news and was also a subject of political contention.

Preliminary tests suggested that unigram probabilities are insufficient for our classification task (see Table 1). The results in this table represent the top ten words in each of two corpora, where words are ranked by number of occurrences, and words that appeared in both corpora were eliminated. To a human observer, there does not appear to be a strong signal of political leaning in these data. For this reason, we used a more complex feature space to do our classification.

2 Procedure

The features we deal with for classifying documents are distributions of association rule confidences as described in (Hu and Liu, 2004). For a given document consisting of a set of words W divided into a set of sentences S, an association rule expresses the likelihood that two separate word phrases X and Y will occur in the same sentence, with an implication that the presence of X causes Y to appear. The rule is defined $X \rightarrow Y$ where $X \subset W$, $Y \subset W$ and $X \cap Y = \emptyset$. That is, both X and Y are word phrases that do not overlap. For our purposes, X and Y are always single words. Two statistics, support s and confidence c, are calculated for each possible word association (every pair of words which occur together in at least one sentence). Support is a measurement of the number of times we see a place in the text where the two words could be associated, and it is defined as the percent of sentences in the corpus that contain either X or Y, $\frac{occ(X \cup Y)}{|S|}$ where $occ(w)$ is the number of sentences containing w. Confidence is then a measurement of how strongly we believe the presence of X causes the presence of Y, and it is measured as the percent of sentences containing X which also contain Y, $\frac{occ(X \wedge Y)}{occ(X)}$.

By imposing thresholds on both c and s (*c-thresh* and *s-thresh*), we select for a given document a number of association rules which both occur somewhat frequently (high support) and have fairly strong causality (high confidence). We further filter these rules by requiring that the "term-sentence frequency" of the second term in the rule, Y, be smaller than a third threshold, *t-thresh*. The term-sentence frequency of a word is defined as the number of sentences containing that word divided by the total number of sentences in the document. This restriction eliminates unimportant rules on very common words like "the" and "of," which would otherwise have very high confidence. The particular values used for these thresholds were *s-thresh*=0.01, *c-thresh*=0.1, and *t-thresh*=0.2. The number of association rules which pass this final threshold define the length of our feature vector for a given document. Some sample rules are given in Table 3. To compare two documents, we use one of several methods to calculate the distance between the feature vectors for the documents. Few rules in a given document's vector occur in other documents' rule-sets as well, so the vectors tend to be fairly distant in the feature space. This means that actual similarity scores will be low, but by comparing relative distances between various publication pairs, we can establish which other publications are more similar to a each other, and which are less.

Our corpora are built from articles obtained via Infotrac both from publications with open political leanings and from those who claim to be balanced or impartial, as shown in Table 2. We restrict our search to articles covering the war in Iraq to minimize variation in the data due solely to topic. This is accomplished by searching the full text for articles containing both "Iraq" and "war."

We calculate distances between articles in the feature space to determine similarity. Distances within

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

Corpus Name	Publication	Size
Liberal		
AProspect	American Prospect	171
Nation	The Nation	110
Nation2	The Nation	102
WashMonth	Washington Monthly	215
Conservative		
Review	The National Review	67
Economist	The Economist	91
Economist2	The Economist	80
WashTimes	The Washington Times	61
"Impartial"		
Time	Time	117
Newsweek	Newsweek	106
ChicTrib	The Chicago Tribune	92
ChicTribBig	The Chicago Tribune	288

Table 2: **Corpus Naming Conventions with Sizes, in thousands of sentences**

humble	→	bush
exhausted	→	has
humble	→	foreign
rare	→	or
multiply	→	but
avert	→	be
attached	→	be
136000	→	by
hat	→	up
beacon	→	it
instruments	→	other
omar	→	an
resounding	→	not
fortune	→	his
fails	→	he

Table 3: **Sample High Scoring Rules from the Intersection of Review and Economist**

the feature space are calculated by one of three distance metrics. Simple cosine similarity is the first. Since $A \cdot B = |A||B| \cos(\theta)$, the cosine of the angle between two feature vectors can be found by computing the dot product of the vectors and dividing by the sum of their lengths; this value can be used as a measure of similarity. We also calculate binary cosine similarity, which is found by converting each non-zero value of the vectors to a 1 and then finding cosine similarity. Our third metric is Euclidean distance in the feature space; we tried using both normalized and un-normalized vectors. The results in Table 9 were generated using the un-normalized vectors, but there did not appear to be a noticeable difference in political-leaning correlation between the two methods.

3 Results

We did several experiments, and the results for each of them are presented here. The naming conventions shown in Table 2 are used to represent our corpora in further figures.

Tables 5, 6, 7 show the data from the cosine similarity comparisons arranged for ease of readability, along with some numerical analyses of those data. For each corpus, the first column shows the publication to which it is being compared. The second column shows that publication's political leaning. The third shows the raw cosine similarity value calculated for those two corpora. The mean and standard deviation over all pairings were calculated, and the fourth column contains the number of standard deviations between that mean and the number in the third column. These tables are sorted according to raw cosine similarity, such that publications more similar to a given corpus appear higher up. The raw data from which these tables are derived is included in Table 8. The raw data for Euclidean distance and binary cosine distance was included in Tables 9 and 10 respectively, but no further analysis was conducted on them since there appeared to be no interesting correlations.

Two corpora were constructed from Chicago Tribune articles, ChicTrib and ChicTribBig. The big corpus contains three times as many articles as the other, and includes *all* of the articles that make up the small one. Table 4 shows cosine similarities of both the small and large corpora to all other corpora. The table is sorted on the second column, and the last column of that table shows the difference in similarity between the second and third columns of that row. This table demonstrates the effect of increased corpus size on cosine similarity. The corpora Nation2 and Economist2 were second corpora taken from The Nation and The Economist respec-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

	ChicTrib	ChicTribBig	Difference
Time	0.0502	0.0508	+0.0006
WashMonth	0.0492	0.0546	+0.0054
Newsweek	0.0482	0.0508	+0.0026
AProspect	0.0481	0.0530	+0.0049
Nation2	0.0478	0.0502	+0.0024
Nation	0.0459	0.0475	+0.0016
Economist	0.0443	0.0440	-0.0003
Review	0.0442	0.0437	-0.0005
WashTimes	0.0428	0.0366	-0.0062
Economist2	0.0402	0.0395	-0.0007

Table 4: **Cosine Similarities for the Small and Large Chicago Tribune Corpora, sorted from top to bottom by ChicTrib similarity score**

tively so that we could test the similarity of different articles from the same publication. As one might expect, these split corpora were more similar to each other than they were to corpora of other publications. These two "second" corpora were composed of the same number of articles as the "first" corpora, and the first and second corpora did not contain any of the same articles and contained the same number of articles. The second corpora were not included in the larger tables, but the second corpora have similarity scores to other publications comparable to those of the equivalent first corpora.

4 Discussion

While the actual numbers returned by the cosine similarity metric are very small, what we are interested in is the relationships between the numbers. The reason that all the similarity scores are so low is that our rules are generated on a per-document basis; this means that each document is likely to generate many rules which are not generated by any other document in the corpus. Because of this, the document vectors tend to have many "dimensions" (each corresponding to a rule) in which there will never be any overlap. It is this sparsity of rules that are common to multiple publications that causes the similarity scores to be so low. We could determine which rules were generated by only a single corpus and thrown them away, but this process would require our method to deal with all our corpora at once, and for this experiment we wanted to use a method that worked explicitly with only two corpora at a time. Using only two corpora means that a new corpus can be analyzed and compared to any number of existing corpora with relatively little work; working with all the corpora at once would force us to re-analyze every corpus each time we wanted add a new one.

It is therefore not a problem that all our similarity scores seem very small. What is important is the differences between those scores, and how those differences correspond to the differences between the publications perceived by humans. As shown in Tables 5, 6, and 7, the cosine similarity metric gives results that roughly correspond to the "desired" values. Binary cosine similarity and Euclidean distance do not appear to give as meaningful results; there is simply no correlation between political leaning and similarity score (see Tables 8, 9, and 10 for the raw data). The cosine similarity metric gives exclusively higher similarities between publications which are openly liberal than it does between openly liberal and openly conservative publications. The same is not true for the openly conservative publications; similarity between conservative publications is not significantly higher than similarity between conservative and liberal publications. The Washington Times in particular has low similarity to all other publications. This dissimilarity may indicate that it is written in a different style, or that it represents a distinct political category, but it most likely indicates a data scarcity problem, since this was our smallest corpus. As shown in Table 2, the conservative corpora for some reason were all smaller than the liberal corpora, at least in terms of number of sentences, despite the fact that all corpora except ChicTribBig contained exactly 80 articles. This is probably at least part of the reason that the conservative publications have lower similarity to each other; with smaller corpora, there are likely to be fewer rules that overlap.

It is also interesting that the Economist showed up as being closer to the liberal publications than the conservative ones; despite our original label of the publication as "conservative," further investigation has revealed that parts of it are generally considered to be liberal. Had our metric not indicated this to begin with, we would not have known to re-examine our label.

The purportedly impartial publications tested

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

were Time, Newsweek, and the Chicago Tribune. The Chicago Tribune exhibits larger similarities to liberal than to conservative publications. Time and Newsweek, however, both appear fairly balanced in their similarities. The apparently liberal slant of the Chicago Tribune may be in part due to the fact that the liberal corpora contained somewhat longer articles on average. However, tripling the size of the Chicago Tribune corpus makes it more similar to the liberal publications and less similar to some conservative publications, indicating that similarity is not merely a function of corpus size (see Table 4). Additionally, the original Chicago Tribune corpus was the smallest of the three impartial corpora (see Table 2), so the fact that it came out as more liberal goes counter to the trend of bigger being equated with more liberal.

5 Conclusions & Future Work

The method outlined in this paper seems to provide at least some ability to rank the similarity of publications, and the similarities it reports correspond with the political agendas that human readers ascribe to those publications. While these results are encouraging, there is still much work to be done in the area of political sentiment classification.

In the future, we would like to analyze publications which claimed to be impartial but are widely thought to have a political leaning, such as the New York Times, the Wall Street Journal, and the Washington Post. Comparison between these publications and publications with known leanings would be interesting.

We would like to test more and larger corpora, and try to find better values for our constants, possibly by training them using machine learning techniques. We would also like to do more statistical analysis on the results of those tests. This analysis would help to demonstrate more clearly the utility of our method. We especially would like to get more data from the Washington Times, since the WashTimes corpus had very low similarity scores to all of the other publications in the corpus. More experimentation is needed to determine why this is the case, but we did not have a large enough corpus to split that corpus in half and do a self-similarity test, which would be the first test we would do.

It is important to note that our algorithm does nothing to specifically isolate features relating to politics. The fact that the resulting feature space seems able to separate liberal publications from conservative ones may therefore come as some surprise. This result is probably due primarily to the fact that all of the articles dealt with the same general subject-matter. If this had not been the case, it is doubtful that similar results would be obtained, simply because the data would be to scarce for political leaning to dominate article topic. Table 3 indicates that the information captured by our features relates primarily to topic and writing style. The rules generated surprisingly do not look much more meaningful to a human than those in Table 1, but our results show it to be nonetheless sufficient for the task of political leaning classification. Adding further processing that does specifically address politics could produce even better results. One such modification could be to learn a set of words which could be considered important to the domain, such as "politically-charged words". Rules containing those words could be weighted more heavily for intersection in order to focus classification to that domain. Similar modifications could be made to focus on domains other than politics instead, making this technique one of general use in any classification task.

References

L. Douglas Baker and Andrew Kachites McCallum. 1998. Distributional clustering of words for text classification. In *Proceedings of 21st ACM International Conference on Research and Development in Information Retrieval (SIGIR-98)*.

William W. Cohen and Haym Hirsh. 1998. Joins that generalize: text classification using WHIRL. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *Proceedings of KDD-98, 4th International Conference on Knowledge Discovery and Data Mining*, pages 169–173, New York, US. AAAI Press, Menlo Park, US.

Zhongchao Fei, Jian Liu, and Gengfeng Wu. 2004. Sentiment classification using phrase patterns. In *The Fourth International Conference on Computer and Information Technology (CIT'04)*, pages 79–86, Wuhuan, China, Sept.

Minqing Hu and Bing Liu. 2004. Mining opinion features in customer reviews. In *Proceedings of Nine-*

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

AProspect (L)				Nation (L)				WashMonth (L)			
Name	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*
WashMonth	*L*	0.0623	+2.460	AProspect	*L*	0.0558	+1.390	AProspect	*L*	0.0623	+2.460
Nation	*L*	0.0558	+1.390	WashMonth	*L*	0.0546	+1.190	Time	*I*	0.0547	+1.210
Time	*I*	0.0510	+1.210	Economist	*C*	0.0497	+0.384	Nation	*L*	0.0546	+1.190
Newsweek	*I*	0.0502	+0.466	Review	*C*	0.0494	+0.334	Newsweek	*I*	0.0528	+0.894
Review	*C*	0.0501	+0.450	Newsweek	*I*	0.0479	+0.087	Review	*C*	0.0503	+0.483
Economist	*C*	0.0491	+0.285	ChicTrib	*I*	0.0459	-0.242	Economist	*C*	0.0496	+0.367
ChicTrib	*I*	0.0482	+0.137	Time	*I*	0.0455	-0.308	ChicTrib	*I*	0.0493	+0.318
WashTimes	*C*	0.0370	-1.708	WashTimes	*C*	0.0342	-2.169	WashTimes	*C*	0.0364	-1.807

Table 5: **Similarity of Liberal Publications to All Publications, sorted by similarity**
P is political leaning. Raw is cosine similarity. Scaled is number of standard deviations from the mean.

Review (C)				WashTimes (C)				Economist (C?)			
Name	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*
Time	*I*	0.0518	+0.730	ChicTrib	*I*	0.0429	-0.736	Nation	*L*	0.0497	+0.384
Newsweek	*I*	0.0507	+0.546	Newsweek	*I*	0.0413	-1.000	WashMonth	*L*	0.0496	+0.367
WashMonth	*L*	0.0503	+0.483	Time	*I*	0.0411	-1.032	AProspect	*L*	0.0491	+0.285
AProspect	*L*	0.0501	+0.450	Review	*C*	0.0376	-1.609	Review	*C*	0.0476	+0.038
Nation	*L*	0.0494	+0.334	AProspect	*L*	0.0370	-1.708	Time	*I*	0.0454	-0.324
Economist	*C*	0.0476	+0.038	Economist	*C*	0.0368	-1.741	ChicTrib	*I*	0.0443	-0.506
ChicTrib	*I*	0.0443	-0.505	WashMonth	*L*	0.0364	-1.807	Newsweek	*I*	0.0442	-0.522
WashTimes	*C*	0.0376	-1.609	Nation	*L*	0.0342	-2.169	WashTimes	*C*	0.0368	-1.741

Table 6: **Similarity of Conservative Publications to All Publications, sorted by similarity**
P is political leaning. Raw is cosine similarity. Scaled is number of standard deviations from the mean.

Time (I)				Newsweek (I)				ChicTrib (I)			
Name	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*	*Name*	*P*	*Raw*	*Scaled*
Newsweek	*I*	0.0548	+1.223	Time	*I*	0.0548	+1.224	Time	*I*	0.0503	+0.483
WashMonth	*L*	0.0547	+1.207	WashMonth	*L*	0.0528	+0.894	WashMonth	*L*	0.0492	+0.318
Review	*C*	0.0518	+0.730	Review	*C*	0.0507	+0.548	Newsweek	*I*	0.0482	+0.137
AProspect	*L*	0.0510	+0.598	AProspect	*L*	0.0502	+0.466	AProspect	*L*	0.0481	+0.137
ChicTrib	*I*	0.0503	+0.482	ChicTrib	*I*	0.0482	+0.137	Nation	*L*	0.0459	-0.242
Nation	*L*	0.0455	-0.308	Nation	*L*	0.0479	+0.087	Economist	*C*	0.0443	-0.506
Economist	*C*	0.0454	-0.324	Economist	*C*	0.0442	-0.522	Review	*C*	0.0442	-0.506
WashTimes	*C*	0.0411	-1.032	WashTimes	*C*	0.0413	-1.000	WashTimes	*C*	0.0402	-0.736

Table 7: **Similarity of Impartial Publications to All Publications, sorted by similarity**
P is political leaning. Raw is cosine similarity. Scaled is number of standard deviations from the mean.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

	AProspect	ChicTrib	Economist	Nation	Newsweek	Time	WashMonth	WashTimes	Review
AProspect	1.0000	0.0482	0.0491	0.0558	0.0502	0.0510	0.0623	0.0370	0.0501
ChicTrib	0.0482	1.0000	0.0443	0.0459	0.0482	0.0503	0.0493	0.0429	0.0443
Economist	0.0491	0.0443	1.0000	0.0497	0.0442	0.0454	0.0496	0.0368	0.0476
Nation	0.0558	0.0459	0.0497	1.0000	0.0479	0.0455	0.0546	0.0342	0.0494
Newsweek	0.0502	0.0482	0.0442	0.0479	1.0000	0.0548	0.0528	0.0413	0.0507
Time	0.0510	0.0503	0.0454	0.0455	0.0548	1.0000	0.0547	0.0411	0.0518
WashMonth	0.0623	0.0493	0.0496	0.0546	0.0528	0.0547	1.0000	0.0364	0.0503
WashTimes	0.0370	0.0429	0.0368	0.0342	0.0413	0.0411	0.0364	1.0000	0.0376
Review	0.0501	0.0443	0.0476	0.0494	0.0507	0.0518	0.0503	0.0376	1.0000

Table 8: **Simple Cosine Similarity**

	AProspect	ChicTrib	Economist	Nation	Newsweek	Time	WashMonth	WashTimes	Review
AProspect	0.0000	216.6922	212.5889	242.6386	234.4846	229.0122	286.4664	182.5949	211.1575
ChicTrib	273.4382	0.0000	216.0229	246.8631	237.6160	232.0051	290.7928	183.7058	214.8406
Economist	273.0869	219.9037	0.0000	246.6635	237.8007	232.3097	290.4779	184.5227	214.2705
Nation	271.4702	218.7671	214.2648	0.0000	236.5015	231.3484	289.3130	183.9661	212.8550
Newsweek	272.1748	218.4070	215.1057	245.6556	0.0000	230.4619	289.3109	183.1170	213.0973
Time	272.2653	218.5251	215.2385	246.1693	235.9112	0.0000	289.3462	183.4231	213.1218
WashMonth	267.2290	215.4656	211.6379	241.6557	233.1303	227.3656	0.0000	181.7377	210.0709
WashTimes	274.7550	221.4264	217.8855	248.9037	239.6326	234.0268	292.0342	0.0000	216.8134
Review	274.1146	220.5114	216.4998	247.5112	238.1333	232.5915	291.3607	184.8660	0.0000

Table 9: **Euclidean Distance in Feature Space**

	AProspect	ChicTrib	Economist	Nation	Newsweek	Time	WashMonth	WashTimes	Review
AProspect	1.0000	0.1301	0.1321	0.1477	0.1372	0.1339	0.1651	0.0952	0.1328
ChicTrib	0.1301	1.0000	0.1195	0.1228	0.1325	0.1385	0.1330	0.1042	0.1205
Economist	0.1321	0.1195	1.0000	0.1343	0.1211	0.1205	0.1325	0.0943	0.1277
Nation	0.1477	0.1228	0.1343	1.0000	0.1297	0.1229	0.1452	0.0883	0.1333
Newsweek	0.1372	0.1325	0.1211	0.1297	1.0000	0.1467	0.1407	0.1041	0.1351
Time	0.1339	0.1385	0.1205	0.1229	0.1467	1.0000	0.1436	0.1045	0.1349
WashMonth	0.1651	0.1330	0.1325	0.1452	0.1407	0.1436	1.0000	0.0960	0.1304
WashTimes	0.0952	0.1042	0.0943	0.0883	0.1041	0.1045	0.0960	1.0000	0.0955
Review	0.1328	0.1205	0.1277	0.1333	0.1351	0.1349	0.1304	0.0955	1.0000

Table 10: **Binary-valued Cosine Similarity**

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

teenth National Conference on Artificial Intelligence (AAAI-2004), San Jose, USA, July.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, July.

Greg Schohn and David Cohn. 2000. Less is more: Active learning with support vector machines. In *Proc 17th International Conf. on Machine Learning*, pages 839–846. Morgan Kaufmann, San Francisco, CA.

Yiming Yang, Jaime G. Carbonell, Ralf D. Brown, Thomas Pierce, Brian T. Archibald, and Xin Liu. 1999. Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, 14(4):32–43, July.

Jeonghee Yi, Tetsuya Nasukawa, Razvan Bunescu, and Wayne Niblack. 2003. Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Melbourne, Florida.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 34–41
Computer Science Department, Swarthmore College

Word Alignment of Parallel Texts

Joshua Berney
Department of Computer Science
Swarthmore College
berney@cs.swarthmore.edu

Jason Perini
Department of Computer Science
Swarthmore College
perini@cs.swarthmore.edu

Abstract

We induced a word-aligned dictionary of English and French using parallel texts. Our texts were the Hansards corpus and a small literary text corpus. We performed phrase alignment by the use of identical words in both texts as anchor points and improved the distribution of our anchor points with lexically similar words. We then performed statistical word-alignment using ϕ statistical correlation to locate translation word pairs in the parallel corpora. Our results show that ϕ correlation works reasonably well when a large number of small parallel phrases are available.

1 Introduction

The goal of this project is to induce a translation dictionary between two similar languages using parallel corpora. The two languages we chose were English and French, as they mostly share the same character set and have significant linguistic similarities. One ready source of large blocks of parallel texts in English and French are classic literary works that have been translated. These have the advantages of being in the public domain and are long documents with consistent word usage and translation style throughout. In addition, the translation of a literary text will leave a large number of words untouched and untranslated such as characters' names, locations, etc. We will need these and any French-English cognates in our phrase alignment algorithm. The disadvantages of using literary works is that the translators, in an attempt to reproduce the style of the original texts, are less likely to produce exact translations, will use less common words, and will repeat words less often.

The literary text we used was "Swann's Way", the first volume of Marcel Proust's *Remembrance of Things Past*, which is approximately 200,000 words in French and English. We took the text from the Project Gutenberg website[1]. We also ran our system against part of the Hansards corpus[2], which is the proceedings of the Canadian Parliament and is in both English and French. This corpus was appealing because it was already split into sentence alignments and was very large (approximately 1 million words).

We started with phrase and sentence alignments using anchor points, which were words that are identical in either text. We then increased the number of anchor points we used by finding likely matches using lexical similarity. Armed with a large number of aligned phrases, we then match likely pairs using the ϕ statistic correlation method.

2 Previous Work

Dmitriy's (2005) work has a number of similarities to ours in his intentions, his system induces dictionaries for languages with few machine translation resources from parallel texts in linguistically similar languages. He aligned his text on a character-to-character basis, not word tokens, and he then per-

[1] Project Gutenberg main site: http://www.gutenberg.org. The specific Proust text can be found at: http://www.gutenberg.org/etext/2650 and http://www.gutenberg.org/etext/7178

[2] http://www.isi.edu/natural-language/download/hansard/1

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College

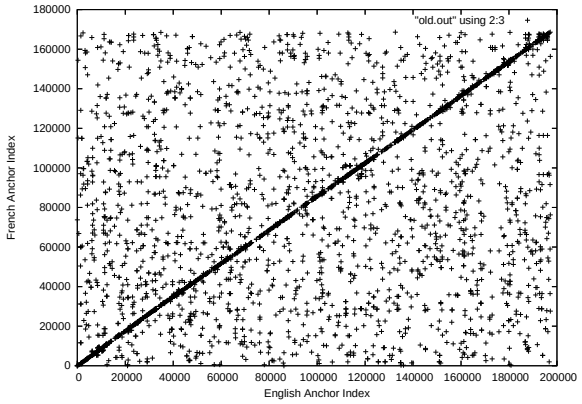


Figure 1: Output from the original anchor alignment procedure: English vs. French Anchor Position

forms a number of post-processing steps to improve the matches he receives from the GIZA++ software, which is a statistical alignment model updated by Franz Joseph Och. One of these steps is the use of lexically similar words, determined by edit distance, as 'seed' words for the alignment models.

Melamed (1999) discusses a much more complicated alignment process for bilingual parallel corpora. He uses cognates and lexically similar words with lexical similarity being determined by the Longest Common Subsequence Ratio method. The alignment is further refined using methods taken from signal noise filtering, as well as several-pass segment alignment and subsection deviations.

Gale and Church (1991) discuss using the ϕ statistic for determining word correspondences. However, their paper is preliminary and provides only vague numbers. There does not appear to be any followup work.

3 Phrase Alignment

The first step in our system is finding equivalent phrases in the source and target languages. This is done to reduce the total number of comparisons that must be made to find translations and to avoid false matches of words that are very far apart from each other in the text. Most post-segment-alignment matching algorithms increase much faster than O(n), where n is the number of words in a corpus, our phrase-alignment method can significantly reduce the time required later in the system.

Our algorithm relies on the fact that some words are exactly lexically similar in the source and target languages, typically nouns. Common examples of such words are places, names, and recently developed concepts. Using these words, we can divide the source and target texts into equivalent phrases. Before beginning the main algorithm, we standardize or eliminate most punctuation. Next, we locate the indices of words that are exactly the same in the target and source language and record their indices. We limit the minimum length of words to exclude which are exactly lexically similar, but are actually different words, such as the English 'a' and French 'a' (the English 'a' is an indefinite article whereas 'a' in French can mean the singular third-person conjugation of 'avoir', 'to have'). We also limit the number of occurrences of words in hopes of limiting the number of times one word appears very close to itself and hence creates possible confusion over the actual anchor pair matching.

Examining a plot of English vs. French position generated from the above algorithm (Figure 1), we see a relatively clear line through the origin (number of English words, number of French words) and many scattered points throughout the plot. Considering the solid line in the figure and the structure of language, we make the assumption that a linear relation exists between the location of a given English word and the French equivalent. Similarly, the location of an English word should be approximately linearly related to the location of the equivalent French word. However, we must also consider there will be places where more English words per French word occur than normal or vice versa. We are also concerned with some target language sentences being out of order with respect to source sentences.

Combining these concepts, we say a given pair anchor points determined from the above algorithm must satisfy:

$$\$source\_word\_index = c \times \$target\_word\_index + \gamma + a\beta \quad (1)$$

where $c = \frac{\$number\_source\_words}{\$number\_target\_words}$, β is some constant, a varies from -1 to 1,

$$\gamma_{new} = \alpha \times c \times \$target\_word\_index + (1-\alpha) \times \gamma_{old} \quad (2)$$

for each valid anchor pair examined in order of occurrence and α is some constant.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College

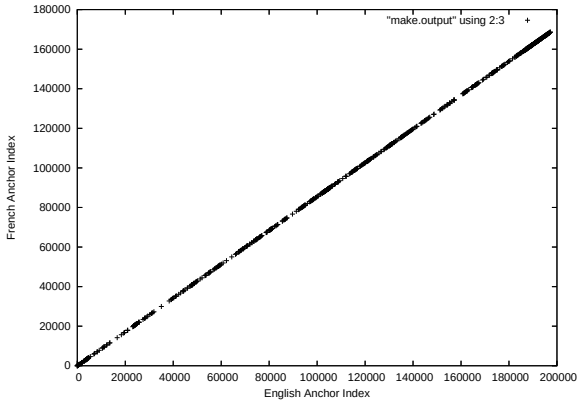


Figure 2: Output from the refined anchor alignment procedure: English vs. French Anchor Position

$source_word_index is our approximation of where the source word should appear. γ represents the current drift – that is the amount of deviation from a linear translation. β is a small constant that allows for some error in our approximation method. In our implement a does not exist, we instead test that $source_word_index without the β term fits within the interval of β. Equation 2 reflects that γ must be updated as anchor pairs as examined. We start with an initial value of γ at 0. Then we iterate through potential anchor word pairs in order of their source index. When a valid pair is found, the second equation is executed and γ_{new} is used until it is updated by finding a new valid pair. Empirically, we have found $\alpha = 0.15$ and $\beta = 40$ work well for the Proust corpus. Due to the nature of the Hansards, it has been difficult to determine optimal values.

Occasionally, a word that appears infrequently in the text will occur very close to itself. Consider the case of a character in a novel only encountered once. This can lead to the algorithm picking up several matches for the same word in a very small region of the text. When this occurs, we take only the first alignment for the word in the region. The output we receive is shown in Figure 2. We use this output for our later steps.

4 Lexical Word Alignment

For texts in similar languages, such as English and French, using lexical similarities can improve the alignment accuracy of other methods by finding words that are likely to be matches. Very good matches are added to the anchor point list along with the lexically identical words and then the anchor point list is passed onto the ϕ statistic correlation method. One simple method of finding lexically similar words is to measure their Levenshtein distances.

The Levenshtein distance between two words is the number of character alterations needed to change one word into another. Each substitution, insertion, or deletion of a character adds to the edit distance of the words. The specific implementation of the Levenshtein algorithm we used was written by Eli Bendersky (2003). It employs a (M+1) x (N+1) matrix where M and N are the lengths of the two strings. The algorithm starts with the word in the source language and calculates the cost for any move, following the least costly path until the minimum transformation cost from one of the strings to the other is found.

The way we used the Levenshtein distance measure to find potential matches followed a partial bag-of-words approach. We looked at the two phrases surrounding an anchor word as unordered list of words, calculating the Levenshtein distance of each word against every other word. We took several steps to speed this process up and avoid calculating distances uselessly. We decided that finding words with a greater Levenshtein distance than 3 changes would result in too many false matches, and so we limited the length difference between two measured words. Since our phrases could be fairly long, oftentimes over 200 words, we found that shortening the window around the anchor word we looked at to between 60 and 80 words in either direction reduced the running time while keeping the algorithm from possibly finding matches where they would be unlikely to occur. We decided that translations would rarely move a word over 120 words from the word it was translated from. However, while most of the words in most phrases will be analyzed by the algorithm looking at the anchor points at the ends of the phrase, this will result in the middle portions of some large phrases being ignored. To capture these 'lost' words we ran the entire lexical matching system through several iterations, using the words we decided were very good matches as new anchors, thus reducing the size of the phrases.

The way we decided whether a match was 'very

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College

good' was if the words in the match met three requirements:

1. They had a Levenshtein distance of three or less.

2. The ratio between the frequency of the words was not too large in the corpus. For example, if one word appears only two times, the word it is matched with should not appear a hundred times. For words with lexical distance of 1, the ratio is 2:5; for 2, 3:5; for 3, 4:5.

3. The words, if they have a close numerical ratio, should usually be matched with each other. In other words, the matched words should not appear apart from each other too frequently. For words with Levenshtein distance of 1, neither word in the match can appear more than 30 times the number of times the match appears; for 2, 20; for 3, 10.

These requirements were applied with differing strictness depending on their Levenshtein distance. Words that were very similar to each other were allowed to vary in their unmatched appearances and numerical ratio more than words that were less lexically similar.

5 ϕ^2 Word Alignment

The ϕ statistic is used to determine correlation between two binary variables. After separating the corpus into phrases, it is a generally good approximation that a given word will appear only once per phrase. By relating the occurrence (one or zero) of a word in a phrase we can hope to find the equivalent translated word in the target language.

The general form of the phi statistic is

$$\phi = \frac{ad - bc}{\sqrt{efgh}} \tag{3}$$

where

	X^-	X^+	Total
Y^-	a	b	e
Y^+	c	d	f
Total	g	h	n

It can be seen ϕ is close to 1 if x and y frequently do and do not occur in conjunction, near 0 if there is no correlation, and if one rarely occurs when the other occurs ϕ is close to -1. In practice, however, computing the square root is relatively computationally intensive. Furthermore, we make the assumption that words will not be negatively related, that is, the existence of one word in a source phrase should not imply that some other word does occur in the target phrase. Making these assumptions, computation time can be decreased by computing

$$\phi^2 = \frac{(ad - bc)^2}{efgh} \tag{4}$$

.

An issue with using the ϕ statistic is computation time. We must compute the ϕ value for every source, target word pair. At initialization, we determine the binary occurrence, either a word does or does not exist per phrase, for each word in the source and target corpora. We iterate through each source phrase for each source word counting the binary occurrence of each target word in the equivalent target phrases. From this, we learn d and using the pre-computed binary occurrences for the entire corpus we can determine the values of all variables. For each source word and target word that occurs in some parallel phrase to the source word, we compute a phi score. We take the highest phi score and treat this as a translation for the source word. A final refinement is to only consider words source words which occur greater than two times. If we consider source words that only occur once, we will frequently receive a large list of false good matches.

This algorithm works well for fairly limited size corpus (<300,000 words), but as the size increases the number of phrases a word occurs in increases approximately linearly and thus the number of phi ranks that must be computed increases very rapidly. This has limited the size of corpus that may be used for training. We believe in future work this problem can be eliminated.

6 Data and Results

We primarily used two corpora for testing: sections from the 2001 Hansards and Swann's Way by Marcel Proust. Each of these documents are available

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College

online in French and English. We also used a sentence by sentence alignment of the 2001 Hansards.

6.1 Phrase Alignment

Phrase alignment has been found to be reasonably precise. Due to the nature of phrase alignment we have no standard data to which we can compare our performance, but examination of parallel phrases reflect that it is generally good at picking out appropriate anchor points. One problem is that not enough anchor points are selected. For the Proust corpus of approximately 200k words, 1000 anchor points are found which translates into phrases of around 200 words. Increasing the parameters to allow the algorithm to locate more anchor points greatly decreases the quality of phrases.

6.2 Lexical Word Alignment

The lexical word alignment was only somewhat successful. It did not end up adding many new anchor points to our phrase alignments, as we needed to constrain the matches greatly in order to reach a high accuracy rate (approximately 70-80% correct). We only allowed matches of up to a Levenshtein distance of 3 and small variations in their occurrence ratios. This generally resulted in the introduction of 400-600 new anchor points to a system with an average of 4000 anchor points produced from using identical words and our anchor phrase alignment algorithm. All these results are on the Proust corpus.

6.3 ϕ^2 Word Alignment

ϕ^2 word alignment was tested using our phrase alignment system for the Proust and Hansards corpora and using the sentence-aligned Hansards corpus. Determining the total number of possible words pairs would be by definition hence we do not include recall numbers. Regardless of the phrase alignment method, if we only considered alignment words with a ϕ^2 value greater than 0.5 almost 80% of the words pairs were correct. However, using our anchor point based alignment system, we will receive less than 100 of $>$4000 unique words which occur twice, resulting in 90% precision, but a very low recall. Yet, when using the sentence-alignment Hansards corpus of 60,000 sentence pairs we find 4000 translation pairs with a precision of 83% as found from a randomly selected sample of 50 word pairs.

7 Conclusion

Two of the defining features of our dictionary induction system were the two texts we used and the ϕ correlation. As discussed in the results section, we have found that ϕ correlation works well with a large number of small parallel phrases. Our system would work best on literary texts with many proper nouns, which would give us better anchor point coverage. Using the sentence-aligned Hansards text showed us how critical having a well-aligned work is and pointed towards one of the problems we had with the non-sentence aligned literary work.

The results of our system are very promising. While we found that lexical alignment did not improve our results greatly, we found that a well aligned corpora can be used to produce a very good translational dictionary using a statistical method. Future work using ϕ word alignment for sentence aligned parallel corpora could provide a highly accurate translation dictionary using no knowledge of the text other than they are linguistically related.

8 Future Work

8.1 Phrase Alignment

Phrase alignment based on sentence boundaries should be examined in depth. While using lexically identical words yields accurate parallel phrases, it fails to yield enough of them. This is especially crucial for ϕ^2 word alignment where increasing the number of phrases and decreasing their size improves the accuracy and running time of the algorithm. Cursory examination of the number of periods in the French vs. English version of Proust's corpus shows many more English than French sentences. However, we can use the fact that on average a given number of English words occur per French word and compare sentence lengths to determine sentence by sentence alignment. As is shown in the results section for ϕ^2 word alignment, if we could improve parallel phrase alignment, we would receive much better results.

8.2 Lexical Word Alignment

The lexical word alignment could undoubtedly be improved in its accuracy and its production of accurately matched words by more tweaking of the

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College

various thresholds and restraints placed on the results. As for larger plans, using language-specific rules and morphological knowledge would be good, but would mean that we could not easily port the system to other linguistically similar languages. A more general approach that would greatly increase the accuracy of our matches would be to use a corpus with part-of-speech tagging, either pre-tagged or done with a readily available part-of-speech tagger. Lastly, testing the lexical word alignment algorithm on other texts and more importantly, different types of texts would reveal further improvements that could be applied to our system.

8.3 ϕ^2 Word Alignment

Several improvements can be made to the ϕ^2 word alignment algorithm. Clearly, this method will work better with large amounts of data. However, we are limited in the amount of data it can currently handle due to the algorithm computing ϕ^2 values for each target word in a target phrase parallel to a source phrase in which the source word exists. Examining only the first approximately 10 phrases in which a source word appears, we can determine the target words that might be translations of the given source word. From this, we can examine the rest of the phrases for a source word and only compute ϕ^2 values for the target words we have picked out. This would significantly reduce time required to run this algorithm and all us to examine very large corpora.

The ϕ correlation statistic is intended for use with binary variables. Many sentences will contain multiple occurrences of a given word. This additional information should be taken into account either by using a different correlation statistic or somehow incorporating this information to the existing ϕ rank statistic.

References

Genzel, D. 2005. Inducing a bilingual dictionary from a parallel corpus in related languages. Submitted to *ACL-05*.

Melamed, D. 1999. Bitext maps and alignment via pattern recognition. *Computational Linguistics* 25(1).

Gale, W. and Church, K. 1991. Identifying Word Correspondences in Parallel Texts *Proceedings of the 4th Speech and Natural Language Workshop*.

Bendersky, E. *Levenshtein Distance Algorithm: Perl Implementation*, `http://www.merriampark.com/ldperl.htm`

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 42–47
Computer Science Department, Swarthmore College