SWAT-MP: Supervised WSD and Affective Text Tagging

Phil Katz
Department of Computer Science
Swarthmore College
Swarthmore, PA
katz@cs.swarthmore.edu

Matt Singleton
Department of Computer Science
Swarthmore College
Swarthmore, PA
msingle1@sccs.swarthmore.edu

Abstract

In this paper, we describe our Word Sense Disambiguation system for SEMEVAL-1 task 5: Multilingual Chinese-English Lexical Sample Task. We implement methods based on Bayesian calculations, cosine comparison of word-frequency vectors, decision lists, and Latent Semantic Analysis. We also implement a simple classifier combination system that combines these classifiers into one WSD module. The results of the SEMEVAL-1 competition are discussed briefly.

1 Introduction

In this section, we will discuss a word sense disambiguation system that implements four different context-based classifiers: a Naïve Bayesian classifier, a Decision List classifier, and a Nearest Neighbor Cosine classifier. The system combines the classifications from these four classifiers into a single guess as to the sense of a word. Our system is an extension of the system described in (Wicentowski et al., 2004).

The task, a multilingual Chinese-English lexical sample task, consists of instances of ambiguous Chinese words surrounded by context, sense-tagged with the correct English translation. The instances are divided into training data, for which the correct translation is provided, and test data, for which the task is to provide the correct translation.

In section 2 we present the implementation of each classifier and of the classifier combination system. In section 3 we present the results from SEMEVAL-1.

2 Methods

2.1 The Classifiers

As discussed previously, our system consists of four unique classifiers. All of the classifiers require the creation of a *term-document matrix*, which contains a column for each training instance of an ambiguous word, and a row for each feature that can occur in the context of an ambiguous word

2.1.1 Naïve Bayes

The Naïve Bayes classifier is based on one of the simplest, most fundamental probabilistic rules: Bayes' Theorem.

$$Pr(A|B) = \frac{Pr(B|A) * Pr(A)}{Pr(B)}$$

Given a term-document matrix, it is very straightforward to implement a Naïve Bayes classifier. The goal is to calculate, for a given context *B*, the probability of a sense *A* occurring. It is simple to calculate the global probability of *A* in the training data, and the probability of *B* is 1. In order to calculate Pr(B|A), we assume that

$$Pr(B|A) = \sum_{i=1}^{n} Pr(B_i|A)$$

where there are *n* different features B_i in context *B*. Pr(B_i|A) can be calculated from the training data as the frequency with which B_i occurs in the context of sense *A*.

Feature	Confidence	Sense
Prev–Financial	99%	Financial
Next–Shot	98%	Basketball
WordBag-Bond	96%	Financial
WordBag-Water	95%	River

Figure 1: A small piece of an example Decision List.

The classifier uses a basic probability equation to calculate the similarity between an instance, *A*, and a sense (from the training data), B_i:

$$Sim(A, B_i) = P(B_i) * P(A|B_i)$$

The classifier returns the sense with the highest similarity to the test data.

2.1.2 Decision List

The Decision List classifier constructs a decision list during training and applies that Decision List during testing. A Decision List is a data structure that can best be visualized as a series of questions asked of the input (Does the input have this feature?). If the answer is yes, then there is an associated classification with that node that is selected; if the answer is no, it moves on to the next node.

The Decision List is created by counting the occurances of each feature in the training set and the occurances of each feature in the context of a given sense. If a feature occurs only in the context of words with the "Financial" sense, the Decision List would have 100% confidence that the feature indicates a financial context. The Decision List is sorted by confidence so that it checks for the features in which it is most confident before the lower confidence features.

Figure 1 shows an example Decision List for *bank*; if the context of the test set does not have any of the features in the Decision List, the classifier simply chooses the most common sense with a confidence of the probability of that sense.

2.1.3 Nearest Neighbor Cosine

The Nearest Neighbor Cosine classifier uses the context vectors created for each sense during training, and for the ambiguous instance during testing. The cosines between the ambiguous vector and each of the sense vectors are calculated, and the sense that is the "nearest" (largest cosine) is selected by the classifier. A more intricate method of cosine comparison, based upon Latent Semantic Analysis, is the fourth and final classifier.

2.1.4 Latent Semantic Analysis

In the process of Latent Semantic Analysis, Singular Value Decomposition (SVD) is used to reduce a term-document (or term-term) matrix of term occurances, W, into three matrices: a left matrix, U, a singular value, or eigenvalue, matrix of eignenvalues down the diagonal, S, and a right matrix, V. The matrix W is constructed with terms as the row dimension, and documents as the column dimension, representing the count of each term in each particular document. Once SVD is implemented, an n by m W matrix will be broken down into an n by m U matrix, an m by m S matrix, and an m by m V matrix.

Once the decomposition process is completed, the matrices are organized by magnitude of highest eigenvalues. Then, we dropped low magnitude dimensions of the decomposed matrices. Choosing which dimension (vectors) to drop was an empirical decision based off of determining which dimension magnitude maximized the success rate for a particular language. This process is described further in the results section.

After the matrices have been dimensionally reduced, we are left with three matrices, each with different semantic meaning. The right matrix, V, can be viewed as a "meaning" by document matrix, in which each document has "counts" for the amount of semantic meaning within. This set of vectors can be used to disambiguate any future document. To implement this method, we utilized the SVDLIBC library. This package uses the Single-Vector Lanczos method and quickly decomposed our large matrices.

To disambiguate, the classifier will *fold* the targeted document into the correct "meaning" space such that a term vector of the disambiguation target is transformed into a meaning vector. This process is done by multiplying the test document vector by U and S^{-1}. Once this vector is transformed, it can be compared with the "meaning"-document matrix using nearest neighbor cosine similarity to determine to which document it is most similar. By reduc-

ing the term dimension into a "meaning" dimension, we am hoping to remove the noise from the term-document matrix such that the important factors in disambiguation will stand out and improve the accuracy of our cosine similarity scores.

2.2 Classifier Combination

The classifier combination algorithm that we implement is based on a simple voting system. Each classifier returns a score for each sense: the Naïve Bayes classifier returns a probability, the cosine-based classifiers return a cosine distance, and the decision list classifier returns the weight associated with the first feature that sense has in common with the test instance. The scores from each classifier are normalized to the range [0,1], multiplied by a constant representing the overall accuracy of that classifier (determined empirically), and summed for each sense. The combiner chooses the sense with the highest summed score. We also implemented a simple voting system, where the sense that is chosen by the most classifiers is the sense chosen by the combiner, but found our combination algorithm to be more accurate (in cross-validation).

2.3 Context Features

Our classifier combination system used a number of features from the surrounding context of an ambiguous word, including: unigrams, bigrams, trigrams, and a simple weighting system on the ten surrounding words.

2.4 Additional Methods

There are three more significant computational methods that were used to improve the performance of the classifiers.

2.4.1 TF*IDF

TF*IDF (Term Frequency-Inverse Document Frequency) is a method for adjusting the frequency of words based on their importance to a document in a corpus. TF*IDF, at a high level, decreases the value of a word that occurs more times in a corpus, but increases the value of a word that occur in less different documents. The equation used for TF*IDF is:

$$tf_i \cdot idf_i = \frac{n_i}{\sum_j n_j} \cdot \log\left(\frac{|D|}{|D : t_i \epsilon D|}\right)$$

where n_i is the number of occurances of a term t_i, and *D* is the set of all training documents.

TF*IDF was used for the Nearest Neighbor Cosine classifier, in an attempt to minimize the noise from words such as *and* that were extremely common, but common across all training instances.

2.4.2 Alpha Smoothing

Alpha smoothing is a technique that is used to attempt to improve the information gained from low-frequency words. We used alpha smoothing in the Naïve Bayes classifier and the Decision List classifier. To implement alpha smoothing, we added a very small number to the frequency count of each feature (and divided the final product by this alpha value times the size of the feature set to maintain accurate probabilities). This small number has almost no effect on more frequent words, but boosts the score of less common, yet potentially equally informative, words.

2.4.3 K-Nearest-Neighbor

One variant on nearest-neighbor cosine comparison that we implemented is K-nearest-neighbor cosine comparison. For this, instead of treating each sense in the training data as one vector, we treat each individual training instance as a seperate vector, and find the *k* nearest training instances to the test instance. We then choose the most frequent sense among those *k* instances. Ultimately, we found that the inclusion of a K-nearest-neighbor classifier actually lowered our accuracy in cross validation so we removed the K-nearest-neighbor classifier from our final system.

3 Results

System	Micro-average	Macro-average
Rank1	.716578	.749236
Rank2	.712299	.746824
Rank3	.710160	.748761
SWAT-MP	.657754	.692487
Rank5	.375401	.431243
Rank6	.336898	.395993

Figure 2: The results of the SEMEVAL-1 task 5, annonymised

References

Richard Wicentowski, Emily Thomforde, and Adrian Packel. 2004. The swarthmore college `senseval-3` system. In *Proceedings of Senseval-3, Third International Workshop on Evaluating Word Sense Disambiguation Systems*.