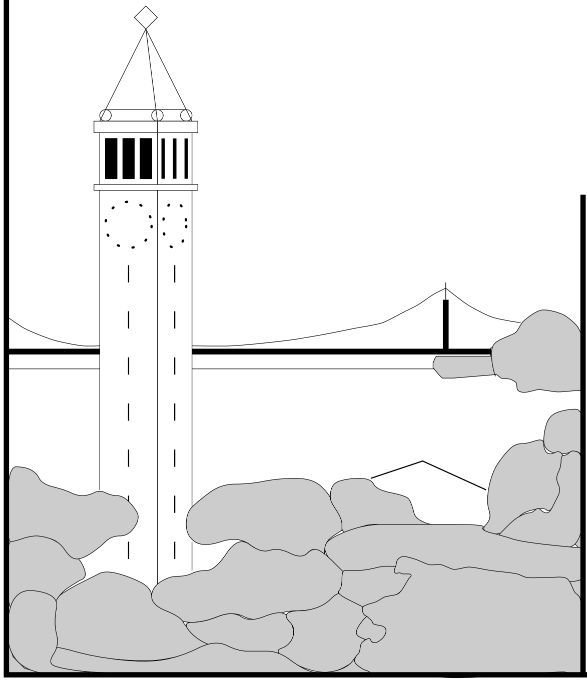
Experiments in Improving Unsupervised Word Sense Disambiguation

Jonathan Traupman and Robert Wilensky



Report No. UCB/CSD-03-1227

February 12, 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Experiments in Improving Unsupervised Word Sense Disambiguation

Jonathan Traupman
University of Calfornia, Berkeley
jont@cs.berkeley.edu

Robert Wilensky
University of California, Berkeley
wilensky@cs.berkeley.edu

February 12, 2003

1 Introduction

As with many problems in Natural Language Processing, word sense disambiguation is a difficult yet potentially very useful capability. Automatically determining the meanings of words with multiple definitions could benefit document classification, keyword searching, OCR, and many other applications that process text. Unfortunately, it is a challenge to design a system that can accurately cope with the idiosyncrasies of human language.

In this report we describe our attempts to improve the discrimination accuracy of the Yarowsky word sense disambiguation algorithm [32]. The first of these experiments used an iterative approach to re-train the classifier. Our hope was that a corpus labeled by an imperfect classifier would make training material superior to an unlabeled corpus. By using the classifier's output from one iteration as its training input in the next, we tried to boost the accuracy of each successive cycle.

Our second experiment used part-of-speech information as an additional knowledge source for the Yarowsky algorithm. We pre-processed our training and test corpora with a part-of-speech tagger and used these tags to filter possible senses and improve the predictive power of words' contexts. Since part-of-speech tagging is a relatively mature technology with high accuracy, we expected it to improve the accuracy of the much more difficult word sense disambiguation process.

The third experiment modified the training phase of the Yarowsky algorithm by replacing its assumption of a uniform distribution of senses for a word with a more realistic one. We exploit the fact that our dictionary lists senses roughly in order by frequency of use to create a distribution that allows more accurate training.

2 Related Work

Word sense disambiguation has a long history in the natural language processing community. It is expected that a successful word sense disambiguation system will be useful to many subfields of NLP, from machine translation to information retrieval. That nearly fifty years of research has yet to produce a disambiguator with high accuracy is evidence of this problem's enduring difficulty.

2.1 Early Systems

The earliest work on word sense disambiguation centered around machine translation. Without some method of determining the meanings of words in context, MT systems have virtually no hope of producing understandable translations. As early as 1960, Bar-Hillel [2] noted the difficulty of this problem in the appendix of his survey of contemporary machine translation research. He claimed that "no existing or imaginable program will enable a computer to determine" the sense of a word that humans "automatically" understand.

Over then next 25 years, researchers applied a variety of approaches to this problem. Katz and Fodor [13] proposed a linguistic theory of semantic structure that introduced the concept of selectional restrictions. In Katz and Fodor's theory, syntactic and semantic features of individual senses can restrict the possible meanings of ambiguous words. Wilks [31] implemented a translation system that used selectional restrictions, in the form of semantic templates, to distinguish between word senses. Selectional restrictions remain a key component of many word sense disambiguation systems today.

Quillian [25] introduced semantic networks, a graph of concepts and their relationships that is independent of syntax. Semantic networks have played a large role in NLP [28], including word sense disambiguation. Hayes [10] presented a word sense disambiguation system that combines semantic networks with selectional restrictions in the form of semantic frames. Though the heyday of semantic networks has passed, semantic network-like databases, such as WordNet [19], are important resources in modern word sense disambiguation systems.

2.2 Recent Systems

Most of the systems of the last 15 years use some form of machine learning to build a classifier from a large corpus. These systems typically run in two phases: a training phase, which builds a classifier from a large set of training examples, and a testing phase that evaluates the classifier on a previously unseen corpus.

Most classical machine learning techniques — decision trees, neural networks, naïve Bayes, and others — have been applied to the word sense disambiguation problem. Mooney [20] evaluates seven of these techniques and concludes that statistical methods (naïve Bayes and perceptron) outperform the others.

2.2.1 Supervised and Unsupervised Systems

Current corpus based approaches can be divided into two broad categories: supervised and unsupervised systems. The supervised systems, such as the examples mentioned above, require training material labeled with the correct sense of each ambiguous example word. While supervised learning algorithms for word sense disambiguation are comparatively well understood, obtaining labeled training corpora of sufficiently large size is a challenge.

In an unsupervised system, the words in the training material are not labeled with senses. The obvious advantage of this approach is that training material is readily available, and with the amount of text on the Internet, virtually unlimited in size. Unsupervised learning's downside is that it is a more difficult problem, since there is no ground truth to which the learning algorithm can refer. The performance of unsupervised systems is almost always inferior to that of the best supervised systems.

One source of difficulty with unsupervised methods is establishing the set of word senses for a given lexicon. One approach, used by Gale, Church, and Yarowsky [9], uses aligned bilingual corpora to distinguish senses that have different translations between French and English.

A large class of unsupervised systems use some form of machine readable dictionary to establish the possible senses for each word. Many of these systems rely on dictionaries that have semantic tags attached to each definition such as Roget's Thesaurus, the Longman's Dictionary of Contemporary English, or WordNet [19]. Yarowsky [32] describes the WSD system that is the foundation for our work. His program uses the category codes in Roget's Thesaurus as tags for senses. Cheng and Wilensky used the same algorithm with a more recent edition of Roget's in the design of the automatic document classification system IAGO [6]

Other systems, like the one by Karov and Edelman [12] do not require a dictionary with semantic tags. Instead they compute a similarity metric between sentences and dictionary definitions to choose the definition that best applies to the context. Cowie and Guthrie [8] also use a dictionary without semantic codes and use simulated annealing to choose definitions for ambiguous words. Their work is based on an earlier dictionary-based approach by Lesk [15].

Disambiguation is possible even without a dictionary. Schütze [27] describes a system that uses clustering in a high-dimension space to classify words according to their usage. While the results of such minimal-knowledge approaches can be impressive, a key problem is that the senses they discover in the text do not always correspond to words' conventional definitions.

2.2.2 Bootstrapping

Falling between supervised and unsupervised approaches are the bootstrapping systems. These systems automatically create a tagged corpus then train a supervised algorithm on the generated training data. Yarowsky [33] describes a system that starts with a small set of "seed" examples and iteratively labels more and more of an unlabeled corpus. Mihalcea and Moldovan [18] show a method for automatically creating large tagged corpora from information in WordNet and text found with a web search engine.

2.2.3 All Words and Single Word Systems

Word sense disambiguation systems can also be divided into *all-words* and *single-word* systems. An all-words system learns to disambiguate all words in a given, usually large, lexicon. A single-word system learns a separate classifier for each word it is to disambiguate and practical concerns usually limit it to a rather small vocabulary.

Because tagging large corpora with word sense information is time consuming and error prone, tagged training materials are scarce and often quite small. For this reason, many supervised systems are single-word systems that show theoretical abilities but are limited by practical concerns. Unsupervised systems are more often all-words systems, since training for additional words usually only requires additional computation time.

2.2.4 Multiple Knowledge Sources and Part of Speech

Syntactic structure, such as part-of-speech and inflected form, was an important knowledge source in the earliest selectional restriction and constraint satisfaction systems. For a time, however, syntax was regarded as less important than semantics, particularly in modern corpus based unsupervised systems. These systems often ignore syntactic structure entirely and view the sentence merely as a "bag of words." They rely on massive amounts of training data to compensate for any information lost by disregarding grammar.

Due to the development of highly accurate part-of-speech taggers [4], several recent word sense disambiguation systems use syntactic structure as a key feature. The LEXAS system of Ng and Lee [21] uses the part-of-speech of the ambiguous word as a filter for possible senses, and also uses the part-of-speech of surrounding words as a feature in their supervised classifier. Stevenson and Wilks [29] demonstrate that part-of-speech alone successfully disambiguates 92% of words in their corpus. Further work by Stevenson and Wilks [30] expand on this idea and use part-of-speech tagging as the first stage in a system that combines three partial taggers (the Yarowsky tagger, a selectional restriction tagger, and a simulated annealing tagger) with an examplar based voting system.

2.3 Further Reading

Good surveys of different techniques for word sense disambiguation may be found in Chapter 7 of Manning and Schütze's book [17], in Chapter 10 of Allen [1], and Ide and Véronis' introduction to the Special Issue on Word Sense Disambiguation in *Computational Linguistics* [11].

3 The Longman Dictionary of Contemporary English

Our classifier is based mainly on Yarowsky's [32] and therefore requires a machine readable dictionary with semantic codes for each definition. While Yarowsky used the categories from the Fourth Edition of Roget's International Thesaurus, we use the field and activator codes from the Third Edition of the Longman Dictionary of Contemporary English [16].

The Longman dictionary was designed as a learner's dictionary with definitions written with a limited vocabulary and a set of semantic markers denoting general concepts and/or specific fields attached to each definition. The same characteristics that make it a useful dictionary for ESL students also make it valuable for NLP research, so the Addison Wesley Company publishes an electronic version, the LDOCE3 NLP Database, specifically targeted at researchers.

3.1 LDOCE3 Format

The LDOCE3 database is in SGML format and contains the full text of the printed version of the dictionary. The dictionary is organized as a series of entries, each of which begin with a "head word," the word that would appear at the start of the entry in the dictionary. Words with multiple parts of speech have separate entries for each part-of-speech tag.

Each entry is divided into a series of senses, which correspond to definitions in the written dictionary. Some senses are further divided into subsenses, which provide finer gradations of meaning. Each sense or subsense contains the text of the dictionary definition, one or more semantic codes, and cross references, usage examples, or other optional information.

This structure is different than the one described by Stevenson and Wilks [30] because they were working with the 1978 First Edition, which grouped related senses into homographs. The Third Edition's notion of "sense" is roughly similar to the older edition's "homograph" and "subsense" to the older meaning of "sense." All of our disambiguation was done at the coarser sense or homograph level.

3.2 LDOCE3 Semantic Tags

Unlike earlier versions of the database, the third edition contains semantic tags for nearly every sense in the dictionary. There are about 1300 different tags used in the dictionary and they are divided into two sets: the *activator codes* and the *subject field codes*. Roughly, 70% of the codes are activator codes and the rest are field codes.

The field codes primarily annotate definitions for specialized or technical terms. These codes form a semantic hierarchy three levels deep with eleven top-level categories. Each field code is a one- to three-letter tag whose length indicates how deep in the hierarchy the code resides.

The more numerous activator codes are used to label definitions for words of more general meaning. These codes encompass general semantic concepts like "Everywhere" and "Angry". In some cases, an activator code, such as "Brave" can be used with words of opposite meanings, like "cowardly," if there are not enough words with opposite meanings to create a separate category.

Examples of both types of codes can be found in Table 1. The complete list of codes can be found in the user manual included with the LDOCE3 NLP Database.

3.3 Weaknesses of Longman's

While the amount of information contained in LDOCE3 is truly impressive, there are some limitations to its usefulness. For our application, its biggest problem is that it contains too much information, primarily in the form of more semantic codes than necessary.

Many of the definitions in LDOCE3 pertain only to word usages that conform to a specific lexical pattern. These senses are denoted by the SGML tag LEXUNIT in the database source. For example, one of the definitions for "rock" is for the lexical form "be on the rocks" meaning "a business or endeavor in dire trouble." This definitions does not apply except in its designated lexical context. Since our tagger cannot currently identify these lexical contexts, the net effect of the LEXUNIT tagged senses is to confuse the classifier and reduce its accuracy. For this reason, we discard these senses.

The assignment of semantic codes to senses can also be problematic. Most words have a single semantic code assigned to each sense, but many have multiple semantic codes per sense. In some cases, a sense will have both a field code and an activator code, in others multiple field or activator codes indicate that a sense overlaps semantic categories.

A similar situation arises with senses that are divided into subsenses. Since our system only discriminates at the sense level, these subsenses have the same effect as multiple codes assigned to a single sense. The classic Yarowsky algorithm uses only a single semantic code per sense, so we have modified it to handle senses with multiple semantic codes.

4 Classifier Algorithm

Our disambiguation algorithm is an adaptation of one due to Yarowsky [32], an unsupervised approach that assigns semantic codes provided by a machine readable dictionary. This algorithm works by collecting statistics about the frequencies of words, semantic codes, and word/code co-occurrences in a training corpus and then uses this data to find the most probable code to apply to a target word during disambiguation. The only data source besides the dictionary the Yarowsky algorithm uses is the context of the target word — the portion of the text that appears within a certain distance of it.

Code	Type	Meaning or Examples
A	Field	Arts
DN	Field	Daily Life/Nature
BFI	Field	Banking/Finance/Insurance
TEM	Field	Technology/Engineering/Mechanical
BORING	Activator	boring, tame, tedious
DO STH/TAKE ACTION	Activator	carry sth out, material, snatch
LEAVE A PLACE	Activator	walk off, take a hike, get away
SPEED	Activator	pace, speed, velocity

Table 1: Examples of Field and Activator semantic codes from LDOCE3.

4.1 Disambiguation

The Yarowsky algorithm assigns code c_{ML} to target word t, when the following is true:

$$\begin{aligned} c_{ML} &= \arg\max_{c_j \in C} p(c_j|T) \quad (1) \\ \text{where } C &= \{\text{codes in the entry for } t\} \\ T &= \{\text{words in the context of } t\} \end{aligned}$$

Using Bayes' rule, we can rearrange this equation to get:

$$c_{ML} = \arg\max_{c_j \in C} \frac{p(T|c_j)}{p(T)} p(c_j) \quad (2)$$

We must now estimate the probability of each semantic code, the context, and the context conditioned on a semantic code. All of these can be calculated from the training data, but it is not obvious how to compute the probabilities involving the context.

If we assume that the words in the context are independent, we have:

$$\begin{aligned} p(T|c_j) &= \prod_{t_i \in T} p(t_i|c_j) \quad (3) \\ p(T) &= \prod_{t_i \in T} p(t_i) \quad (4) \end{aligned}$$

The factors in these products are both estimated during the training phase. Combining these transformations gives us our final disambiguation equation:

$$c_{ML} = \arg\max_{c_j \in C} \frac{\prod_{t_i \in T} p(t_i|c_j)}{\prod_{t_i \in T} p(t_i)} p(c_j) \quad (5)$$

4.2 Training the Classifier

Training the Yarowsky classifier consists of estimating $p(t_i|c)$, $p(t_i)$, and $p(c_j)$ in the above equations. Estimating $p(t_i)$ is the easiest of the three. As we scan through the training corpus, we maintain a count, $count_{t_i}$, of the number of occurences of each unique word. Our estimate for $p(t_i)$ thus becomes:

$$p(t_i) = \frac{count_{t_i}}{\sum_{t_k \in TC} count_{t_k}} \quad (6)$$

where TC is the set of all words in the training corpus.

Estimating $p(t_i|c)$ requires that we count each time a word co-occurs with a particular code. We therefore maintain a matrix A whose entries $A_{t,c}$ contain the number of co-occurrences between a word t and a code c. To fill in the values this matrix, we scan through the training corpus until we encounter a word, t, that has the set of codes $C = \{c_0 \dots c_j \dots c_n\}$ listed in its dictionary entry. The words in the context, $T = \{t_0 \dots t_i \dots t_m\}$, all co-occur with the correct code for this instance of t. We should increment A_{t_i,c_j} for all $t_i \in T$ and the correct code c_j.

However, our training corpus is unlabeled, so we do not know which of the possible codes in C is the correct one. Therefore, we assume that *all* possible codes for t occur simultaneously with a uniform distribution. We update, A_{t_i,c_j}, for each $t_i \in T$ and $c_j \in C$ by incrementing it by a uniform weight:

$$\begin{aligned} A_{t_i,c_j} &\leftarrow A_{t_i,c_j} + w_j \quad (7) \\ \text{where each } w_j &= \frac{1}{|C|} \quad (8) \end{aligned}$$

To estimate $p(t_i|c_j)$ from the data in matrix A, we also need to count how many times each code co-occurs with any word. We maintain this count by incrementing a variable $count_{c_j}$ by the same factor w_j each time we see a word t that contains code c_j in its dictionary entry.

Once we have constructed the matrix A and the count of each code, we can estimate $p(t_i|c)$: [1]

$$p(t_i|c_j) = \frac{A_{t_i,c_j}}{count_{c_j}} \tag{9}$$

We reuse the same $count_{c_j}$ value to estimate $p(c_j)$:

$$p(c_j) = \frac{count_{c_j}}{\sum_{c_k \in C_{DICT}} count_{c_k}} \tag{10}$$

where C_{DICT} is the set of all semantic codes used in the dictionary.

Once training is complete, we use these three estimates in equation 5 above to classify new instances of ambiguous words.

4.3 Adding Part-of-Speech Information

Our experiment with part-of-speech information requires that we modify the standard Yarowsky algorithm. There are three main places where we wish to add part-of-speech information into the standard algorithm:

1. Limiting the choice of possible semantic codes, C, during disambiguation.

2. Limiting C during training.

3. Using the pair (t_i, p_i) instead of just t_i for context words during both training and disambiguation.

To add part-of-speech information to these three locations, we replace each word, t, with a tuple (t, p) of the word and its part-of-speech label in the above equations. The most likely semantic code, c_0, is thus determined by:

$$\begin{aligned} c_{ML} &= \arg\max_{c_j \in C} p(c_j) \prod_{(t_i,p_i) \in T} \frac{p(t_i, p_i|c_j)}{p(t_i, p_i)} \quad (11) \\ \text{where } C &= \{\text{semantic codes for } (t,p)\} \\ T &= \{(t_i, p_i) \text{ in the context of } t\} \end{aligned}$$

Each of the three uses of part-of-speech information can be independently switched on or off in our implementation. With all of them off, the algorithm reverts to the standard Yarowsky tagger.

Along with the use of part-of-speech data, we made several other adaptations to the standard Yarowsky algorithm to handle senses with multiple semantic codes as found in LDOCE3. These changes are described below in Sections 5.3.3 and 5.4.1.

4.4 Iterative Retraining

Our second modification to the standard algorithm uses an iterative approach that feeds the results of disambiguation back into the training step.

Under this system, the initial iteration is exactly like the standard Yarowsky algorithm: the classifier is trained on an unlabeled corpus. We take this classifier and use it to disambiguate all ambiguous words in the training corpus. The results of this disambiguation step is then used in another training step. While the normal Yarowsky algorithm weights each possible sense uniformly during training, the iterative approach weights them according to the likelihoods returned by the disambiguator. The hope is that the results of the first stage disambiguator are close to the correct sense and thus make better training examples than the uniformly distributed codes.

In some ways, this approach is similar to boosting [26]: we use the classifier to refine the training material in order to create a better classifier. However our approach differs from boosting in several fundamental ways. Boosting relies on a tagged corpus to

[1] This normalization is not completely correct. In order to ensure that the distribution of $p(t_i|c_j)$ sums to one, we should divide A_{t_i,c_j} by the number of times c_j co-occurs with a context word. We could maintain this count by incrementing $count_{c_j}$ by w_j once for each context word that co-occurs with c_j, rather than once for each target word that includes c_j as a possible code.

We use this less correct normalization for three reasons. First, it ensures that $p(T|c_j)$ sums to one: imagine that a code c_j always occurs with the same N context words. With the proper normalization, each $p(t_i|c_j) = \frac{1}{N}$, so $p(T|c_j) = \frac{1}{N^N}$ when T consists of these N words. With our normalization, $p(t_i|c_j) = 1$ so $p(T|c_j) = 1$.

Second, this normalization allows us to reuse $count_{c_j}$ in our calculation of $p(c_j)$. Finally, the larger denominators with the correct normalization can lead to numerical instability when calculating $p(T|c_j)$. Since this normalization factor is a constant, it does not affect our calculations of c_{ML}.

find examples that the originally classifier got wrong. It then retrains these failing cases using the ground truth examples from the labeled training set.

On the other hand, our system does not have tagged training materials and cannot find only the failing examples. Therefore, we retrain on all examples, using the output of the first iteration classifier as ground truth. Also unlike traditional boosting, we do not reuse the same training material from one iteration to the next. While the actual text remains the same, the distribution of senses assigned to each word varies considerably, based on the output of the previous generation's classifier.

Unfortunately, this scheme suffers from a fatal and rather obvious flaw. By using the classifier's output as training data, we reinforce the behavior of the original classifier. On words where its accuracy is high, our approach helps, but ones it frequently mislabels get worse. In essence, this iterative technique is overtraining, exactly the problem that boosting tries to avoid by emphasizing only mis-tagged examples during subsequent training iterations.

4.5 Sense Frequency Weighted Training

Our third and final experiment also involved altering the distribution of senses used during training. After observing the skew in the test set and the accuracy of a baseline classifier that always assigns the first sense listed in LDOCE3, we realized there might be benefit to weighting the training distribution by the order senses are listed in the dictionary. We replaced the uniform distribution of sense weights from equations 7 and 8 with the following distribution:

$$w_j = \frac{(\frac{1}{2})^j}{\sum_{k=1}^{M} w_k} \qquad (12)$$

In other words, the weight of the $j + 1^{st}$ sense is half of the j^{th} sense, and all the weights sum to one.

There is no rigorous justification for this weighting. We simply looked for a distribution that balanced our desire to emphasize senses listed earlier in the dictionary with the need to have all senses represented to some degree.

This scheme is easily adapted to use part of speech data. Just like with the standard algorithm, we use only the senses that agree with the labeled part-of-speech when constructing this distribution.

5 Implementation

Our word sense disambiguation system consists of several programs that implement different phases of preprocessing data, training the classifier, running the classifier to disambiguate a text, and measuring the results. The operation of our system follows the following steps:

1. Extract the dictionary and code files from LDOCE3.
2. Apply part-of-speech-tags to training and test corpora:
   - Detect sentence boundaries and place each sentence on its own line.
   - Run part-of-speech tagger.
3. Preprocess the training corpus:
   - Stem the words
   - Count words and sort by frequency
4. Run the training algorithm to build a Yarowsky-style classifier.
5. Apply the classifier to the test corpus.

We now describe each step of this process in detail.

5.1 Processing the Dictionary

Before we can use the information in the LDOCE3 database, we must first digest it into a more suitable format. LDOCE3 is provided in SGML format, which is structured, but slow and expensive to parse. We provide a simple program, `mkdict`, that processes the SGML into a more suitable format.

The output of `mkdict` is the file `dictionary.txt`. Each line of this file corresponding to an entry in Longman's and consists of a series of colon separated fields. The first field is the word, the second is its part-of-speech, and the third is the number of senses. The remaining fields are a list of the senses. Each of these senses is a slash (/) separated list. The first item is the number of semantic codes attached to the sense, and the subsequent items are numeric values representing the semantic codes.

In addition to the dictionary file, `mkdict` outputs a file named `codes.txt` that maps the semantic code strings to numeric codes. The file is simply a list of codes, with the numeric value given by the order in the list. For example, the first entry in `codes.txt`, "SLA," is represented by code "0" in the dictionary file.

`mkdict` also processes the part of speech labels to make them appropriate for the classifier. For instance, LDOCE3 contains sub-categories of verbs, like "auxiliary verb," that must be mapped to the standard "v" verb tag. Entries that are not a noun, verb, adjective, or adverb are given an "unknown" part-of-speech tag because the classifier does not care about any part of speech other than these main four.

The `mkdict` program need only be run once when setting up the system. Subsequent uses of either the training or disambiguator programs can use the same `dictionary.txt` and `codes.txt` files.

5.2 Part-of-Speech Tagging

Our part-of-speech experiment requires that both the training and testing corpora be labeled with part-of-speech tags. For part-of-speech tagging, we used the well-known Brill tagger [4, 5], which reads a corpus and outputs each word labeled with a part-of-speech tag. Brill reports its accuracy to be 95-97%.

Our test set confirms Brill's accuracy results. On average, part-of-speech tagging accuracy is 95.7%. Only three or our 18 test set words are tagged with less than 90% accuracy: float (83.2%), giant (50.3%), and promise (85.6%). Figure 1 charts the performance of the Brill tagger on each word in our test set.

The Brill tagger uses the Penn Treebank tag set, so it has far more part-of-speech tags than the four main ones our classifier uses. Therefore, the training and disambiguation programs must perform a simple mapping between the Penn tags and the four we use.

The implementation of the Brill tagger we use requires each sentence to be on its own line in the corpus. To perform the sentence boundary determination, we initially looked into using a sophisticated system like SATZ [22, 23], but were unable to use it because key lexical resources were unavailable. In the end, we created our own tool, `sbd`, that uses simple heuristic pattern matching to determine sentence boundaries. Its performance is not as good as SATZ, but is sufficient for our purposes. The other programs in our system do not place the same requirement on the corpus and, in fact, ignore sentence boundaries completely.

Like the dictionary processing, the corpus preprocessing need only be performed one time when the corpus is first used.

5.3 Training

We provide two programs to implement the Yarowsky algorithm: the training program, `train`, and the disambiguation program, `disambiguate`. The training program creates a classifier from the information extracted from LDOCE3 and a large training corpus. The disambiguator uses the training results to disambiguate a previously unseen test corpus. Obviously, the classifier must be trained before it can be used for disambiguation.

The `train` program begins by loading the dictionary, `codes.txt` file, and the stop list. It then performs some additional preprocessing and begins training the classifier. It outputs three files: `wordinfo.dat`, a file with information about word senses and frequencies, `codefreq.dat`, containing frequency data about the semantic codes, and `database.dat`, a Berkeley DB format database containing the word/sense collocation data.

5.3.1 Preprocessing

Before `train` actually starts training the classifier a small amount of additional preprocessing must be done. The program reads through the entire corpus and counts the occurrence of each word. The result of this word count is a list of all words that occur in the corpus, the dictionary, or the stop list.

The words are then sorted by frequency and assigned integer indices. These indices are used instead of strings in the word/code co-occurrence database for space efficiency. Sorting by frequency yields better locality in the database and thus improves performance of both training and disambiguation.

The `train` program can be instructed to halt after preprocessing by using the `-p` option. With this option, `train` will output the `wordinfo.dat` file, but

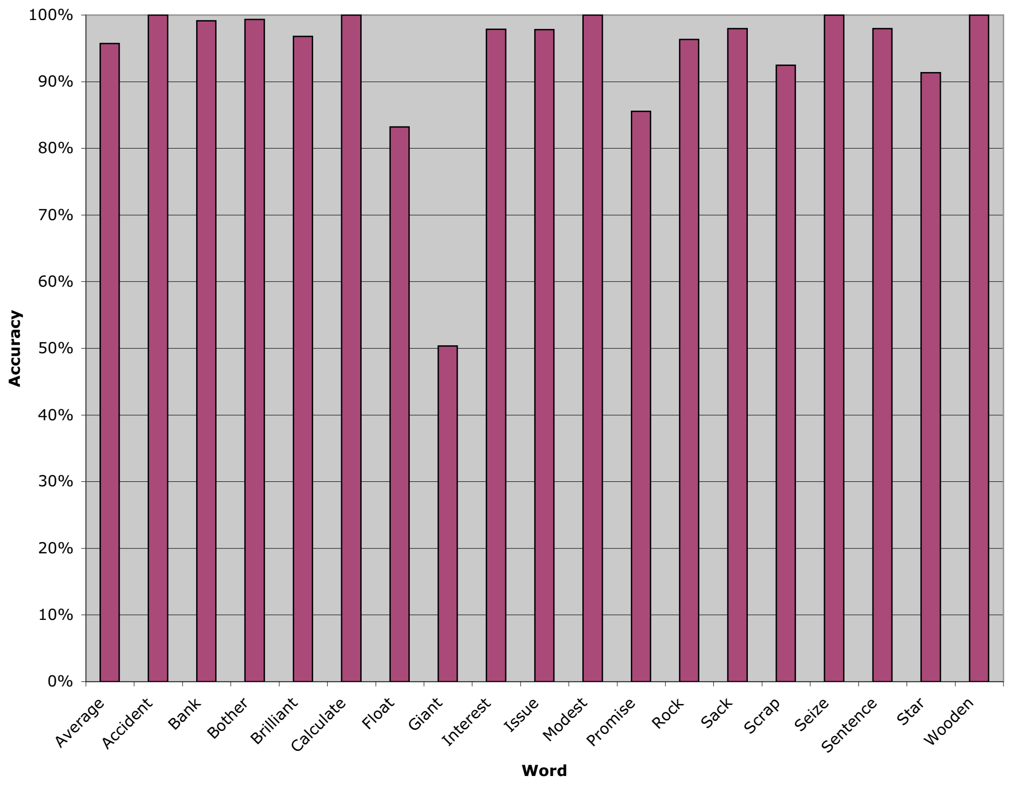


Figure 1: Accuracy of the Brill part-of-speech tagger on our test set.

not the database or code frequency data. Unlike the other preprocessing steps, the preprocessing in `train` must be performed each time the classifier is trained. Since preprocessing requires only about five minutes of CPU time at the start of a training run that may take several hours, it did not seem worth the effort to allow old preprocessing runs to be reused.

The `-f` option allows the user to specify a file where `train` will dump the list of all words in the training corpus sorted by frequency. This option is a useful tool for creating stop lists tailored for a particular corpus.

5.3.2 Stemming

Like most dictionaries, LDOCE3 only contains entries for the root form of inflected words. While "bike" is in the dictionary, "biking" and "bikes" are not. In order to reduce data sparseness and control the size of the collocation database, we wish to transform each word in the corpus into its stem form.

We use the *morphy* stemmer from WordNet as the foundation of our stemming algorithm. The morphy stemmer uses both the unstemmed word and its part-of-speech label in deciding the correct base form for a word.

Our stemming algorithm proceeds as follows:

1. Use morphy to find the stem of a word/part-of-speech pair.

2. Lookup the returned stem in LDOCE3. If the stem exists in the dictionary, return it.

3. If the stem is not in LDOCE3, the word ends in "ing" or "ings," and is tagged as a noun, use morphy to find the stem of the word with a verb

part-of-speech tag. If the stem returned by morphy is in LDOCE3, return the stem and change its part-of-speech tag from noun to verb.

4. If the stem is not in LDOCE3, the word ends in "ing" or "ed," and is tagged as an adjective, use morphy to find the stem of the word with a verb part-of-speech tag. If the stem returned by morphy is in LDOCE3, return the stem and change its part-of-speech tag from adjective to verb.

5. Otherwise, returned the word unstemmed.

Steps 3 and 4 are necessary because the Brill tagger labels gerunds and participles as nouns and adjectives, respectively. WordNet contains separate entries for the gerund and participle forms of verbs, so morphy will return the word unchanged. However, LDOCE3 does not, in general, contain separate entries for gerunds and participles, so the stem returned by morphy (still in gerund or participle form) will appear to have no entry in LDOCE3. Since most gerunds and participles are easily identified, we can retry stemming them with morphy with a verb part-of-speech tag. If the resulting verb stem is in LDOCE3, we return it and permanently change the word's part-of-speech tag to verb. Otherwise, we return the original word and part-of-speech tag unmodified. This approach allows us to use the sense codes for participles and gerunds that have their own entries in LDOCE3 (e.g. "yearning") while using the verb-form senses for ones that are not listed in the dictionary.

This stemming algorithms corrects a number of flaws from our earlier approach, a stemmer based on the well-known Porter algorithm [24]. Our Porter stemmer variant often returned stems that were nonwords. In particular, it handled inflected words whose stem ends in -y very poorly: "buried" becomes "buri" not "bury." Being unaware of the part-of-speech tags, our earlier stemmer also did not transform the tag from noun or adjective to verb when stemming participles and adjectives. In cases where the stem has both noun and verb senses (e.g. "rock" as the stem for "rocking"), this behavior would cause the training and disambiguation processes to choose from the less appropriate noun set of senses in the case of gerunds and from all the senses with participles.

5.3.3 Training the Classifier

Once the preprocessing and stemming are done, training the classifier is a fairly straightforward application of our modified Yarowsky algorithm. The `train` program iterates through each word in the corpus and updates the frequency counts of the word's semantic codes and the co-occurrence counts of the word's codes with each of the other words in the context window. The complete training algorithm is given in pseudocode in Figure 2.

5.3.4 Senses with Multiple Semantic Codes

Because the Yarowsky algorithm assumes that each sense has only a single semantic code assigned to it, we need to modify it to handle the multiple semantic codes in some LDOCE3 senses. In the case where a word's senses each have only one semantic code, we proceed like the standard algorithm: each code's global count is incremented by the inverse of the number of senses. If a word has five senses each with a single semantic code, each code will have its global count increased by 0.2.

If one of these senses has multiple codes, this increment is further divided by the number of codes attached to the sense. A sense from the previous example that has two semantic codes will have each of these codes' counts incremented by $0.2/2 = 0.1$. The same values are used for updating both the global semantic code counts and the word/code co-occurrence counts in the database. We believe this mechanism strikes a sound balance between the need to count all codes attached to a word while not allowing senses with multiple codes to dominate the training.

5.3.5 Support for Part-of-Speech Information

As can be seen in Figure 2, we have added support for part-of-speech information in two places. As each target word is processed for training, we examine its part-of-speech label and use it to discard any senses listed under entries with differing parts of speech in the dictionary. In addition, the part-of-speech label for context words is used along with the word as the index into the collocation matrix. We maintain separate collocation counts for each part-of-speech tag that a context word can assume.

```
declare $wordcnt$           // total count of all words
declare $count[]$           // individual word counts
declare $codecnt[]$         // count of semantic codes
declare $A[][]$             // Co-location array

for each word $w$ in the training corpus do
    $p \leftarrow$ the part of speech of $w$
    $ns \leftarrow$ the number of senses in the dictionary entry for $(w, p)$
    $count[w] \leftarrow count[w] + 1$
    $wordcnt \leftarrow wordcnt + 1$

    for each sense $s$ in the dictionary entry for $(w, p)$ do
        $nc \leftarrow$ the number of codes in sense $s$

        for each code $c$ in sense $s$ do
            $codecnt[c] \leftarrow codecnt[c] + 1/(nc \cdot ns)$

            for each word $t$ in the context of $w$ do
                $A[t][c] \leftarrow A[t][c] + 1/(nc \cdot ns)$
            end for
        end for
    end for
end for

save $wordcnt$
save count[]
save codecnt[]
save A[][]
```

Figure 2: Training algorithm.

The use of part-of-speech data is turned off using the `-no_target_pos` and `-no_context_pos` switches on the `train` command line. Disabling the use of context part-of-speech causes the program to store collocation between words and semantic codes instead of between word/part-of-speech pairs and semantic codes. Disabling target part-of-speech forces `train` to use all semantic codes for a given target word, not just the codes that agree with the tagged part-of-speech. Turning on both of these switches completely eliminates the use of part-of-speech data during training, and the program reverts to an implementation of the standard Yarowsky algorithm.

5.3.6 Support for Iterative Re-training

The support for iterative re-training is not shown in the pseudocode, but is very straightforward. In the algorithm above, the code frequency and collocation matrix entries are incremented by a uniform amount (subject to the scaling described in Section 5.3.4). Iterative retraining replaces these uniform weights with the likelihood distribution returned by the disambiguation. Since the disambiguator returns the likelihood of each sense, we still need to scale this value by the number of semantic codes in the sense.

5.3.7 Support for Sense Frequency Weighted Training

The changes necessary to implement training with weights distributed according to sense frequency are minimal. In the pseudo-code in Figure 2, we replace the factor $1/ns$ in the update statements with the weights calculated as described in Section 4.5.

5.3.8 Optimizations

Training the classifier is far and away the most time and resource intensive component of our system, so we have added several optimizations to make it run as fast as possible. One of these, sorting the words to improve database locality, was mentioned above. We also designed the database entries to be as small as possible to maximize the amount of data that could be cached in RAM by Berkeley DB.

Even with much of the database paged into RAM, each Berkeley DB operation is quite slow — several orders of magnitude slower than a normal memory reference. To reduce the number of these operations and thus speed up the database, we implemented a simple caching procedure for the training phase.

When a corpus file is initially loaded and parsed, we attach a cache structure to each non-stopped word. This structure, initially empty, is a linked list containing tuples of semantic codes and co-occurrence increments. When the training algorithm updates the word/code co-occurrence count for a word in the context, it does not load the old value from the database, add the increment, and push it back into the database, as a naïve implementation would. Instead, it adds the increment to the tuple containing the proper code. It creates a new tuple in the cache if one does not already exist. Once the program is done processing each word in the file, it iterates through the cache and adds each cached increment to the appropriate word/code entry in the database.

This optimization resulted in a 10-20% speedup in training time, for two reasons. First, it can reduce the total number of database operations. If a word co-occurs with the same semantic code from two different words in its context — and one must believe this happens if Yarowsky's assumption that semantic codes can indicate topic is true — then two or more database operations are folded into a single one plus some cheap cache manipulations. Second, all database operations are batched together in a single phase, and several updates for a single word are performed sequentially, both of which improve database locality and reduce the amount of time spent doing database operations.

While we did not do a detailed analysis of the cause and effect of these optimizations, we did observe a noticeable speedup in the still very long training time.

5.4 Disambiguation

The structure of the disambiguation program, `disambiguate`, is very similar to that of the training program. The word counting and sorting operations are not necessary during disambiguation, because this information is all contained in `wordinfo.dat`. The same stemming and stop list procedures are performed as during training.

The disambiguation algorithm itself, as shown in Figure 3, is essentially the inverse of the training algorithm. For each ambiguous word, the algorithm accumulates evidence from code frequency data and from the word/code co-occurrence data. The sense that has the largest amount of evidence in its favor is chosen as the sense for the word.

Unlike the training algorithm, the disambiguation process does not use any sort of cache to speed up database operations. Since most disambiguation is done on smaller corpora and with only a limited set of target words, the performance implications of not caching are minor.

5.4.1 Handling Multiple Codes per Sense

Like the training algorithm, the stock Yarowsky disambiguation algorithm needed to be modified to support multiple semantic codes attached to a sense. We explored two possibilities for handling this case. In both cases, we run the standard Yarowsky disambiguation algorithm to calculate evidence for each possible semantic code. We then use one of the following methods to choose a sense based on the semantic code evidence:

1. Choose the sense that has the semantic code with the greatest amount of evidence. If more than one sense includes the most likely semantic code, report all of them.

```
load $wordcnt$          // total number of words in training corpus
load $count[]$          // word counts
load $codecnt[]$        // count of semantic codes
load $A[][]$            // co-location array

$wordcnt \leftarrow$ the total word count

for each word $w$ in the testing corpus do
    $p \leftarrow$ the part of speech of $w$
    declare $evidence[]$          // evidence for each sense

    for each sense $s$ in the dictionary entry for $(w, p)$ do
        declare $code\_evidence[]$          // evidence for each code in $s$

        for each code $c$ in sense $s$ do
            $code\_evidence[c] \leftarrow codecnt[c]/wordcnt$

            for each word $t$ in the context of $w$ do
                $code\_evidence[c] \leftarrow code\_evidence[c] \cdot (A[t][c] * wordcnt)/(codecnt[c] * count[t])$
            end for
        end for
        $evidence[s] \leftarrow \max_c code\_evidence[c]$
    end for
    return $\arg\max_s evidence[s]$
end for
```

Figure 3: Disambiguation algorithm.

2. Average the evidence for each sense's codes to form an evidence figure for the entire sense. Assign the sense with the highest evidence to the word.

After experimenting with both of these options, we chose option 1. Option 2, while possessing a sense of mathematical correctness, causes senses with multiple codes to be chosen less often then they ought to be. Since the multiple semantic codes in a sense are frequently only distantly related, averaging tends to scale down the total evidence for a sense by a factor of the number of senses. Even if a single code in such a sense has very high evidence, it will often be beat by a much less likely sense with only a single semantic code.

The choice of option 1 does have one serious shortcoming: it renders our disambiguator incapable of discriminating between two senses that have a semantic code in common. The use of part-of-speech information significantly reduces this error, since it allows such senses to be distinguished if they occur with different parts of speech.

5.4.2 Integrating Part-of-Speech Information

Part-of-speech information can be used in roughly the same places during disambiguation as when training. The pseudocode above already includes the necessary additions to the Yarowsky algorithm.

The `disambiguate` program also has the same two switches for controlling the use of part-of-speech information. Both have the same effect as during training: `-no_target_pos` disables the elimination of semantic codes incompatible with the tagged part-of-speech. The `-no_context_pos` switch turns off the use of part-of-speech tag when looking up collocation in the database. The setting of the `-no_context_pos` flag in the training and disambiguation phases must be the same, but the `-no_target_pos` flag can be set independently. Enabling both options during both training and disambiguation results in a standard Yarowsky classifier.

5.4.3 Support for Iterative Re-training

The disambiguation half of the Yarowsky algorithm requires no substantive modifications to support iterative retraining. The only modifications we made is an option to tell `disambiguate` to disambiguate all words in a corpus and output the distribution of sense likelihoods for each ambiguous word. Normally, we disambiguate only specified target words and produce a more human-readable output.

5.4.4 Unique Identification of Senses

In some cases, it is impossible to uniquely identify the correct sense of an ambiguous word. Often, two separate senses in LDOCE3 will have either the same semantic code or will have a semantic code in common. Since the disambiguator deals only in semantic codes, it cannot make a further distinction in this case and is forced to output both senses. For example, the word "rock" has one sense labeled with the codes HEG and DN, meaning "stone," and another labeled just with HEG, meaning "gem." If the disambiguator determines that the most likely semantic code is HEG, it cannot further distinguish between these two senses and is forced to list both of them. On the other hand, if DN is the most probably code, then it can choose the "stone" sense with confidence.

Senses with codes in common frequently occur because senses with different parts-of-speech have similar semantics. It is just this sort of case where the use of part-of-speech information proves most valuable. The additional knowledge gained from the part-of-speech tags allow us to completely disambiguate these cases where the standard tagger would be unable discriminate between them.

5.4.5 Smoothing

An important contributor to the accuracy of the disambiguation program is the smoothing of the data collected during training. Because the training set is only a finite sample of English text, it may not be representative of true usage. This issue is especially troublesome with words that occur very infrequently in the training corpus. For example, if a word that appears only twice in the training corpus occurs once within the context of a target word with a certain semantic code, is it a statistically significant indicator of that code, or just a random fluke that they co-occurred?

We use two approaches to smoothing the training data. Both techniques work by discarding evidence from certain context words during disambiguation. The evidence a context word, t_i, contributes towards a sense c_j is the term $p(t_i|c_j)/p(t_i)$ in equation 5.

Our first technique is to ignore evidence below a certain threshold. The rational for this smoothing approach is that low values of evidence for a particular code are often just noise due to the small sample size. Too much of this noise in a large context window can appear to indicate a false correlation between the context and a particular code. Only counting strong evidence reduces the effect of this noise. We discovered through empirical studies that the optimal value for this parameter is $p(t_i|c_j)/p(t_i) \geq 1.1$. In other words, a sense must co-occur with a particular context word approximately 1.1 times more often than random chance in order for the evidence of this co-occurrence to count in determining the target word's most likely semantic code.

This technique subsumes an earlier technique we tried: discarding evidence less than 1. Manning and Schütze described this tactic [17, p. 246], but it was not clearly mentioned in Yarowsky's original paper. However, as long as the threshold for retaining evidence is greater than 1, evidence less than one will be automatically discarded.

The final smoothing technique addresses the concern stated above: that evidence from infrequently occurring words is often unreliable. To address this problem, we simply ignore any evidence from context words that occurred less than a certain threshold in the training corpus. Our experiments set the optimal value of this parameter to 10.

In addition to smoothing, both the training and the disambiguation programs have a final parameter that affects their performance: the size of the context window. The same empirical testing that established the smoothing parameter values indicated that the optimal context size for our corpora was ± 25 non-stopped words around the target word. Smaller contexts did not provide enough evidence for accurate disambiguation and larger contexts allowed distant, topically unrelated words to contribute inaccurate evidence.

6 Testing Methodology

To evaluate the effectiveness of our modifications, we compared several variants of our algorithm and compared the results. The ten classifiers tested are described in Table 2.

The classifiers are broadly divided into three categories. The first category is the two baseline algorithms, described in detail in Section 6.3 below. The next category is the classifiers that use the standard uniform weighting of senses during training. We test four variants in this category, each using a different amount of part-of-speech information:

a. No part-of-speech information (standard Yarowsky tagger).

b. Part-of-speech information used to limit target word senses during disambiguation.

c. Part-of-speech data used to limit target word senses during both disambiguation and training.

d. Part-of-speech data used both to limit target senses and in the collocation database during both training and disambiguation.

The last category consists of classifiers that use dictionary order during training to distribute the sense weights according to frequency as described in Section 4.5. Within this category, we test the same four variants as above.

The overall results of our tests can be found in Figure 4. All of these tests use the same training and disambiguation corpora.

To test the iterative re-training approach, we ran the system for five iterations and charted improvements and regressions for each cycle. Results of this experiment are shown in Figure 12 and described in more detail in Section 7.5.

6.1 Training and Test Corpora

All training and testing runs used the same corpora. The training set consisted of approximately ten million words from the Microsoft Encarta 97 electronic encyclopedia [7]. Yarowsky demonstrated in his original paper on this algorithm [32] that using general interest training material, such as this encyclopedia, contributes to higher accuracy on a wider variety of text than using more specialized corpora such as newswire data.

We extracted our test set from a 14 million word corpus of AP newswire stories from January-May 1991. We tested the algorithms on 18 words, chosen to have a mixture of parts of speech and degrees of ambiguity. The words and their characteristics are described in Tables 3–6. For each of the words, we extracted between 50 and 700 usage examples from the AP newswire corpus. The examples were chosen randomly and we expect them to reflect the distribution of the words' usages in the overall test corpus text.

Several of the words in our test set will be recognized as coming from the first SENSEVAL competition [14]. We decided to use words from the SENSEVAL resources because they have been judged "good" words to use for evaluating word sense disambiguation systems by a panel of experts in the field. We also hoped to leverage the publicly available test sets for these words to make our hand tagging task easier. Unfortunately, the SENSEVAL resources' typically small contexts (± 10 words) and use of British English [2] proved to be a poor fit for our classifier. We therefore kept the same words but took new examples from our AP newswire corpus.

6.2 Scoring

Each of these 18 test sets were hand tagged with the correct sense using a utility program we wrote called `mkmaster`. This program produces an "answer key" file with the correct code for each usage of the test word in the testing corpus. Each instance of a word can be tagged with multiple tags if more than one seemed appropriate. If two senses overlapped, we made our best possible judgment of the correct

[2] For example, the SENSEVAL test set for "float" contained many uses of the vehicle sense such as "milk float" that never occur in our training materials

Variant	Description
Baseline 1	Baseline algorithm that always returns the first sense listed in the dictionary for a given word.
Baseline 2	Baseline algorithm that returns the first sense listed in the dictionary for a given word and part-of-speech tag.
Classifier 1a	Standard Yarowsky classifier that does not use part-of-speech information and trains with a uniform sense distribution.
Classifier 1b	Classifier that uses the part-of-speech tag to limit senses during disambiguation. Uses a uniform sense distribution during training.
Classifier 1c	Classifier that uses the part-of-speech tag to limit senses during both training and disambiguation. Uses a uniform sense distribution during training.
Classifier 1d	Classifier that uses the part-of-speech tag to limit senses during training and disambiguation as well as using part-of-speech information in the sense/context word co-occurrence data. Uses a uniform sense distribution during training.
Classifier 2a	Classifier that does not use part-of-speech data. Uses the frequency weighting of senses described in Section 4.5 during training.
Classifier 2b	Classifier that uses the part-of-speech tag to limit senses during disambiguation. Uses a distribution weighted by sense frequency during training.
Classifier 2c	Classifier that uses the part-of-speech tag to limit senses during training and disambiguation. Uses a distribution weighted by sense frequency during training.
Classifier 2d	Classifier that uses the part-of-speech tag to limit senses during training and disambiguation as well as using part-of-speech information in the sense/context word co-occurrence data. Uses a distribution weighted by sense frequency during training.

Table 2: Classifiers used to evaluate the part-of-speech modifications and the use of dictionary order to weight senses by frequency during training.

sense according to the definitions and usage examples in LDOCE3. In cases where multiple senses seemed equally appropriate, we tagged the word with both senses. If none of the senses seemed relevant or the correct sense would be an eliminated LEXUNIT sense (see Section 3.3), we marked the example "none of the above" and that instance was not used to score the algorithm.

Scoring of the algorithm is handled by another utility program, `score`. This program reads in the answer key produced by `mkmaster` and the output of the disambiguator and outputs a summary of the disambiguator's accuracy and precision on individual words and average accuracy. Its output is either a human readable table or a CSV file suitable for post-processing by most spreadsheet software.

Because both the answer key and the disambiguator output can contain multiple senses for each instance of a word, scoring is slightly more complex than just counting how many instances match and dividing it by the total number of target words in the corpus. Each instance of a tagged word receives a score according to the following formula:

$$s = \frac{|D \cap C|}{|D|}$$

where D is the set of senses output by the disambiguator, and C is the set of senses in the answer key for the instance. In other words, the score for an instance is the number of senses that appear in both the disambiguator output and the answer key divided by the total number of senses the disambiguator produced.

In the simple case of the answer key and the disambiguator both giving a single sense for a word, `score` gives a score of 1.0 for the instance if the answers match and 0.0 if they don't. However, if the disambiguator returns multiple senses, it only gets partial

Word	Codes	Examples	Definition
accident(n)	ACCIDENT	324	unintentional damage or injury
	CHANCE/BY CHANCE	8	something that happens without planning or intention
bank(n)	BFB	616	a business that keeps and lends money
	DN	43	land along the side of a river or lake a large pile of earth, snow, sand, etc.
	LINE	0	a large number of machines, etc. arranged in a row
	DGG	0	money in a gambling game that can be won
	TTR	0	a slope in a road to make it easier to turn
bank(v)	BFB	2	to put or keep money in bank
	TTA TTC	1	to make a plane or car move to one side while turning
	LINE	0	to arrange something into a pile or rows
	DN	0	to form a mass of cloud, mist, etc.
	BURN	0	to cover a fire with fuel to keep it going
	DEPEND ON/RELY ON	1	to depend on something happening
bother(v)	DO STH/TAKE ACTION	38	make the effort to do something
	WORRIED	88	make someone feel slightly worried or upset
	DISTURB	20	to annoy someone by interrupting
	HURT/CAUSE PAIN	15	upset, frighten, or harass someone
bother(n)	PROBLEM	1	trouble or difficulty that has been caused by small problems
	ANNOY	0	a person or job that is annoying to deal with
brilliant(adj)	BRIGHT	19	strong light or color
	GOOD ENOUGH	42	extremely good, clever, or skillful
	GOOD/EXCELLENT	4	very good (not used for people)
	SUCCESSFUL	15	very successful; a brilliant career
brilliant(n)	DCJ	0	a precious stone cut to shine
calculate(v)	FIND OUT	132	to find out how much s/t will cost, how long s/t will take, etc.
	GUESS	21	to guess something using as many facts as you can
	DEPEND ON/RELY ON	0	depend on something
float(v)	ON/ON TOP OF	52	to stay on the surface of water
	SLOW	17	to move slowly through the air
	HEAR	1	for sounds, smells to move through the air
	PEC	4	allow a currency to change with the market
	SUGGEST	7	to suggest an idea or plan
	BFS	5	to sell shares in a company
	PAY FOR	0	to write a check without sufficient funds to cover it
	GRACEFUL	1	to move gracefully
	CHANGE YOUR MIND	2	to move from one thing to another, especially jobs

Table 3: Test set words (1 of 4).

Word	Codes	Examples	Definition
float(n)	TTC	24	a vehicle(often in a parade)
	DFD	0	a desert made with ice cream
	DSO	0	a light object that floats on water
	DSS	0	a light object used during swimming
	BBT	0	a small amount of cash kept on hand
giant(adj)	BIG	74	something large
giant(n)	RF	1	a tall mythical character
	BBC	56	a large, successful company
	TALL PERSON	0	a very big man
	GOOD AT	11	someone who is very good at something
interest(n)	INTERESTED	69	the desire to find out more about something a quality of something that holds your attention something that you enjoy doing
	BFL	197	a charge make for borrowing money money paid for keeping money in a bank
	ADVANTAGE OWN	112	things that bring advantages to someone or something
	BBC	39	a share in a company, business, etc.
	GROUP OF PEOPLE B	2	a group of people in the same business who share goals, ideas, etc.
interest(v)	INTERESTED	0	to make someone feel interested
issue(n)	SUBJECT	411	a subject of discussion
	TCN	27	a particular edition of a magazine or newspaper
	AVAILABLE	87	a new set of something, such as shares or stamps
	PROVIDE	0	the act of officially providing people with something
issue(v)	SAY/STATE	134	to officially make a statement, give an order, etc.
	PROVIDE	14	to provide something for each member of a group
	SELL	19	to officially produce something, such as shares
	APPEAR	0	to come out of somewhere
modest(adj)	MODEST	5	unwilling to talk proudly about your achievements
	LITTLE/NOT MUCH	153	not very big, expensive, etc., especially less than what would be expected
	SHOW LET SB SEE STH	2	shy about showing your body or attracting sexual interest, because you are embarrassed
	SEXY	1	modest clothing covers the body in a way that does not attract sexual interest

Table 4: Test set words (2 of 4).

Word	Codes	Examples	Definition
promise(v)	PROMISE	105	to tell someone that you will definitely do something
	EXPECT	3	to make you expect that something will happen
promise(n)	PROMISE	63	a statement that you will definitely do something
	SIGN/INDICATION	8	signs that something or someone will be successful
rock(n)	HEG DN	29	a large stone
	APM	69	a type of music
	DF	1	a hard sweet food made in long round pieces
	HEG	0	a diamond or other jewel
rock(v)	MOVE/CHANGE POSITION	1	to move gently back and forth
	SHOCK	5	to make people feel very shocked or surprised
	HE	1	shake from an explosion or earthquake
sack(n)	TA D TM	31	a bag, or the amount that a sack can hold
sack(v)	LEAVE A JOB OR ORGANIZATION	11	fire someone
	DSA	0	know down the quarterback in football
	ATTACK PMA	7	destroy a conquered city or country
scrap(n)	PIECE	8	a small piece of paper, cloth, information, etc.
	USE STH	13	materials that are no longer used, but can be recycled
	FIGHT	1	a short fight or argument
scrap(v)	STOP STH THAT IS HAPPENING	101	to decide to stop using a plan or system
	USE STH	6	to get rid of an old item and recycle its parts
	FIGHT	0	to have a short fight or argument

Table 5: Test set words (3 of 4).

Word	Codes	Examples	Definition
seize(v)	TAKE STH FROM SB TAKE ST FROM SOMEWHERE	6	to take hold of something suddenly and violently
	CONTROL	74	to control of a place suddenly and quickly using military force
	TAKE STH FROM SOMEWHERE	186	if the police seize something, they take away illegal goods
	CATCH	10	to suddenly catch someone and make sure they cannot get away
	INTERESTED	7	to suddenly become very interested in an idea, excuse, etc.
	STOP MOVING	0	if an engine or machine seizes, it suddenly stops moving, often due to a lack of oil
sentence(n)	SLG WORD, PHRASE, OR SENTENCE	13	a group of words that express a complete thought or question
	SCT PUNISH	243	a punishment issued by a judge
sentence(v)	SCT PUNISH	277	to legally issue a punishment
star(n)	HA	21	astronomical object
	FAMOUS ACTOR/ACTRESS PERFORM AP AM	98	a famous and successful performer in entertainment or sports the main actor or actress in a movie or play
	CF SLA PM	9	a shape with four or move points a mark in this shape a piece of cloth or metal in this shape worn to designate rank
	DLT SUCCESSFUL	0	a mark used in a system of judging hotels and restaurants someone who is successful at a job, etc.
star(v)	ACTOR/ACTRESS AP AM	71	act the main part in a play or movie
	WRITE	0	place a star-shaped mark next to something written
wooden(adj)	T D	180	made of wood
	EXPRESS AP	0	showing little emotion

Table 6: Test set words (4 of 4).

credit based on how many answers it gave and how many matched. If the disambiguator returns two potential senses, but only one matches with the senses in the answer key for that instance, it will receive a score of 0.5. If both senses match the answer key, it will receive a score of 1.0.

This scheme does not penalize the disambiguator for failing to match all senses listed in the answer key for an instance. We give multiple correct tags to ambiguous words only when we could not decide among equally appropriate senses. If the disambiguator returns at least one of the correct senses (and no incorrect ones) it receives full credit for a correct response.

The score for a complete corpus is the average score for all instances of the target word in that corpus.

6.3 Baseline Classifiers

To fairly evaluate our algorithm, we compared our results with those from two baseline taggers. Both baseline taggers assign the most frequently occurring sense to each instance of a test word. Since we do not have actual statistics for how often each sense of a word occurs, we rely on the fact that LDOCE3 lists word senses and parts-of-speech roughly in order by frequency. As can be seen from the distribution of senses in our test set, this approximation is reasonably accurate.

Baseline 1 chooses the first sense from the first part-of-speech entry for a word. It uses no part-of-speech information. Baseline 2 uses the part-of-speech tags to limit possible senses, like several of our classifier variants. It returns the first sense listed in the dictionary entry corresponding to each word's part-of-speech tag.

7 Results

As can be seen in Figure 4, the overall trend is that the more part-of-speech information is used, the better the results. Using dictionary order weighting during training also shows a small benefit: each classifier that uses this weighting by approximate frequency outperformed the corresponding classifier that used a uniform distribution of sense weights.

Both variants that do not use part-of-speech information did very poorly. The standard Yarowsky classifier (1a), has an average accuracy of only 25.4%, while the version that uses dictionary order weighting at training does only slightly better at 27%. Both significantly underperform both baselines.

7.1 Part-of-speech Results

Using part-of-speech to limit the choice of senses during disambiguation increases the accuracy significantly. Classifier 1b achieves an average accuracy of 39.0% and classifier 2b, which uses training weights distributed by the sense's dictionary order, is even higher at 41.3%. Both of these classifiers still underperform both baseline classifiers.

Using part-of-speech to limit the senses during both training and disambiguation does not produce a benefit. Both classifiers 1c and 2c underperform the variant that uses part-of-speech to filter senses only at disambiguation. The accuracy of classifier 1c is 38.4% and 2c is 41.0%, a slightly less than the performance of variants 1b and 2b, respectively.

Most notable, however, is the improvement in accuracy when the part of speech of the context words is used in the collocation database (classifiers 1d and 2d). Intuitively, we expected this technique to yield little improvement, since it increases the size and thus the sparseness of the collocation matrix. Apparently, part of speech is a more important factor for determining semantic code co-occurrence than we had expected. The version that uses uniform training weights (1d) has an accuracy of 59.1% while using training weights distributed according to the senses' dictionary order (2d) once again gives us a small improvement to 60.6%. Both of these classifiers outperform both baselines.

Those familiar with this algorithm will also note that our overall accuracy is far lower than the 92% claimed by Yarowsky in his original paper on this algorithm. We attribute this discrepancy to several factors. First, the degree of ambiguity in our test corpus is significantly higher than in Yarowsky's study: 4.6 versus 3.3. Second, we deliberately chose "hard" words; namely, those that do not typically relate topically to the sentences that contain them, such as "issue" and "interest," for much of our test set.

Finally, the Yarowsky algorithm was originally designed for disambiguating nouns, while we test on nouns, verbs, and adjectives. The fact that many

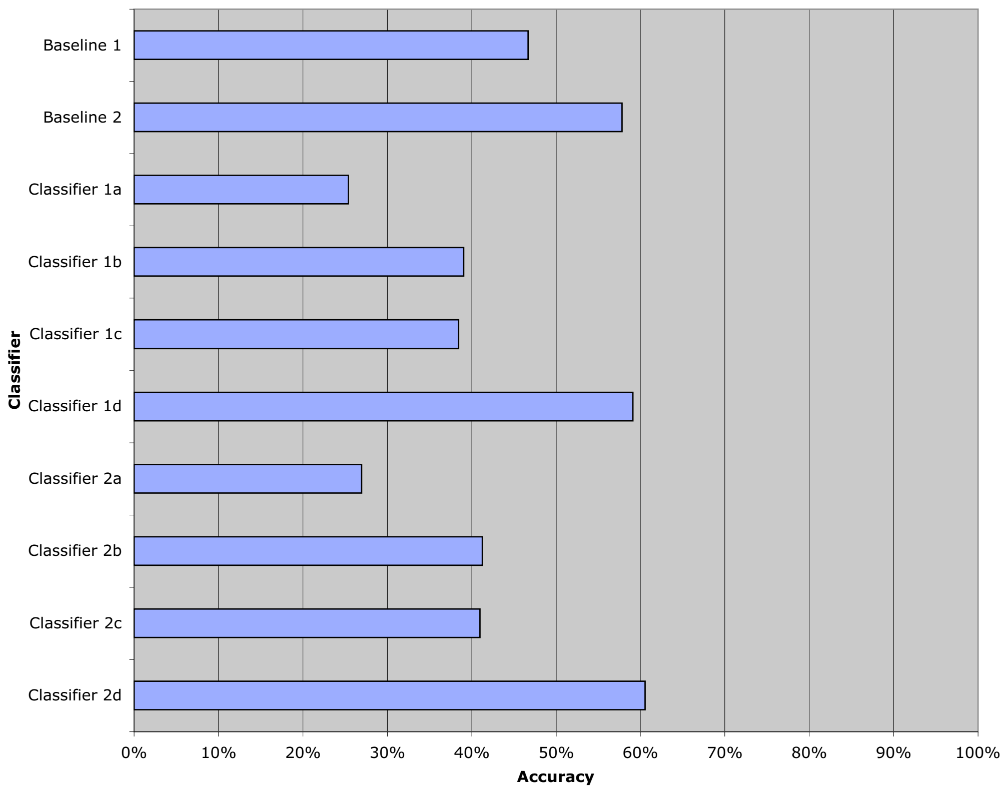


Figure 4: Comparison of tagger accuracy.

words contain senses with the same semantic codes listed under more than one part-of-speech particularly cripple the standard Yarowsky version of our algorithm. In the absence of part-of-speech data, the algorithm cannot distinguish between multiple senses containing the same semantic code, and must list all of them in its output. This phenomenon is particularly noticeable in the words "bank," "sentence," and "star" in our test set: the accuracy nearly doubles when part-of-speech filtering is enabled.

Our results show the indisputable benefit of part-of-speech data to a classifier of this sort. Of the 18 words in our test set, all but three showed some degree of improvement when part-of-speech information was used. The three exceptions are "giant," "seize," and "calculate." The performance decline with "giant" is due to an interaction between the labeling of senses in LDOCE3 and our part-of-speech tagger's performance on this word and is discussed in more depth below. We have no convincing explanation for "calculate" or "seize" — as words with a single part-of-speech, we did not expect much improvement, but did not think we would see the performance decrease dramatically as more part-of-speech information was used.

7.2 Results of Training with Dictionary Order Sense Weights

The results of the using dictionary order to weight the sense during training were positive, as shown in Figure 7. Within each pair of classifiers the version that weights the senses by approximate frequency during training on average outperforms the one that assumes uniformly distributed senses.

Using dictionary order to weight the senses during training improved the accuracy of at least one classifier variant on all but one of our eighteen test set words. When not using part-of-speech information (the "a" variants), weighting senses by frequency improved the performance on seven of the words. With twelve of the words, these non-uniform weights improved both the classifiers that filter using part-of-speech during disambiguation only ("b" variants) and the ones that filter during both training and disambiguation ("c" variants). Finally, the classifiers that use part-of-speech both as a filter and in the context ("d" variants) also improved with twelve of the words in our test set. The largest improvement, 17.4 percentage points, occurred when comparing classifiers 1c and 2c applied to the word "modest." The largest decline happened with the word "issue," where classifier 2d was 12.8 percentage points less accurate than Classifier 1d.

On average, weighting by dictionary order at training increases the accuracy of all variants of our algorithm by a few percentage points. The classifier variant that uses part-of-speech tags to filter during both training disambiguation improved the most with this technique, from 38.4% to 41.0%. The version that also uses part-of-speech in the context improved the least, from 59.1% to 60.6%.

We did not experiment extensively with alternative distributions during training. Possibly, one that more accurately models the actual distribution of senses will show a greater benefit.

7.3 Detailed Results

Because of the number of variants we are testing (two baselines and eight classifiers), it is difficult to see all of the relationships between them at once. We provide seven charts, each showing a different type of comparison:

Figure 5 Presents the four part-of-speech usage variants trained with uniformly distributed senses.

Figure 6 Presents the four part-of-speech usage variants trained with senses weighted by dictionary order.

Figure 7 Compares the performance of pairs of classifiers using the same type of part-of-speech data where one uses uniformly weighted senses during training and the other uses senses weighted by dictionary order.

Figure 8 Compares the classifiers using uniformly weighted senses during training with Baseline 1.

Figure 9 Compares the classifiers using uniformly weighted senses during training with Baseline 2.

Figure 10 Compares the classifiers using senses distributed by dictionary order during training with Baseline 1.

Figure 11 Compares the classifiers using senses distributed by dictionary order during training with Baseline 2.

Each of the words in our test set had slightly different characteristics, which influenced each algorithm's ability to discriminate its senses. By examining the performance of these classifiers on individual words, we can gain some insight into each version's strengths and weaknesses.

7.3.1 Accident

"Accident" shows poor performance with Classifiers 1a-c. The standard Yarowsky classifier (1a) performs at about its average accuracy with this word, while the two classifiers that use part-of-speech to filter senses (1b and 1c) are well below their average performance with "accident." Using part-of-speech as a filter has no effect with this word, since it has only a single part-of-speech.

Using part-of-speech both as a filter and in the context (Classifier 1d) has a dramatic effect: it increases the accuracy from 25.9% to 86.1%, the second highest single-word accuracy for classifiers using uniform training weights.

Distributing training weights by dictionary order at training also shows a large benefit. All four classifiers in this category outperformed their counterparts using uniform weights. The greatest gains were for the variants a-c, though Classifier 2d did outperform 1d by 3.3%.

All eight versions of our algorithm underperformed the baseline, primarily because the first listed sense is the correct one in 97.5% of our test set examples.

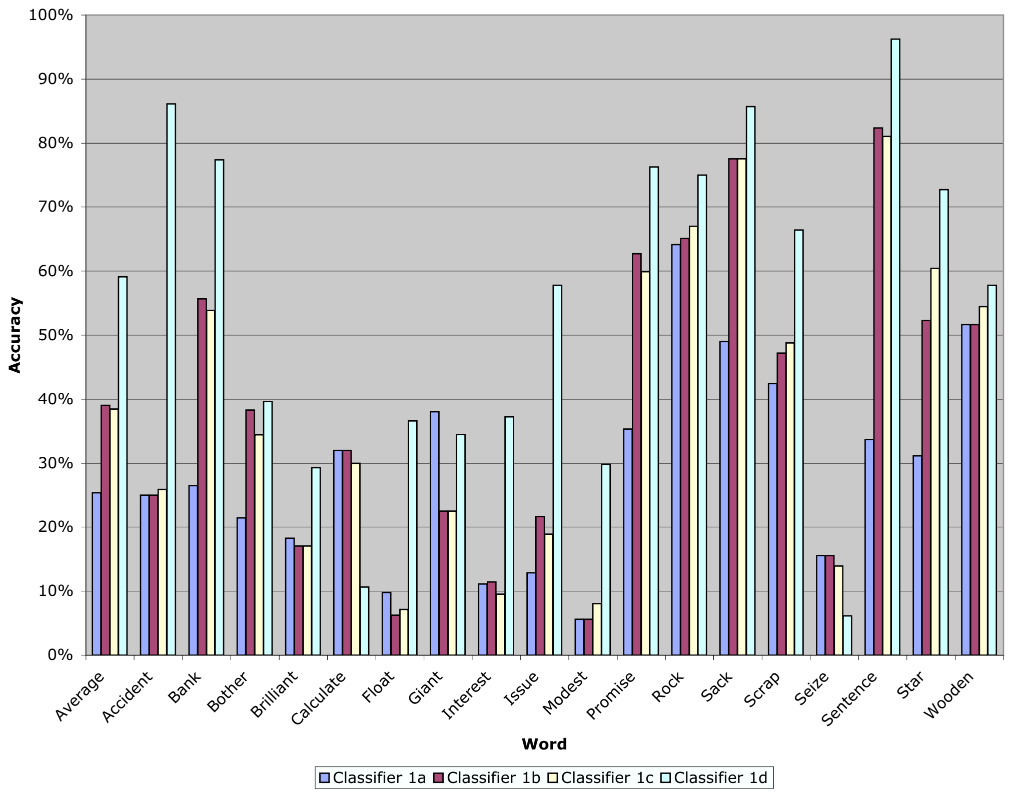


Figure 5: Accuracy of the four classifier variants that use uniformly distributed sense weights during training.

7.3.2 Bank

"Bank" is exactly the sort of word we expected our classifiers to discriminate well. It occurs mainly as a noun and is frequently related to the topic of the sentence in which it appears. For the most part, this expectation proves true. Each of the eight variants of our classifier performed with better than its average accuracy.

Using part-of-speech to filter senses during disambiguation resulted in a large improvement, primarily because senses in both noun and verb entries for "bank" contain the semantic code BFB. Without using the part-of-speech tags, classifiers 1a and 2a cannot distinguish between these two senses, roughly halving their accuracy compared to variants 1b and 2b.

Variants 1c and 2c, which use the part-of-speech tags to filter during training, had slightly lower performance than classifiers 1b and 2b. The difference between classifiers 1b and 2b was greater than that between 1c and 2c. One possible explanation for this behavior is that the most common tag, BFB, appears in both the noun and verb entries and skews the classifier in its favor by occurring twice during training when part-of-speech filtering is not used. Two other words that show large declines in accuracy between version 1b and 1c, "promise" and "sentence," also have their most common semantic code occurring in both noun and verb senses. Since the using dictionary order to weight senses at training already skews the classifier in favor of these most common senses, we don't see as large a drop in performance between variants 2b and 2c.

Using part-of-speech in the context (classifiers 1d and 2d) also gives significant benefits. Classifiers 1d

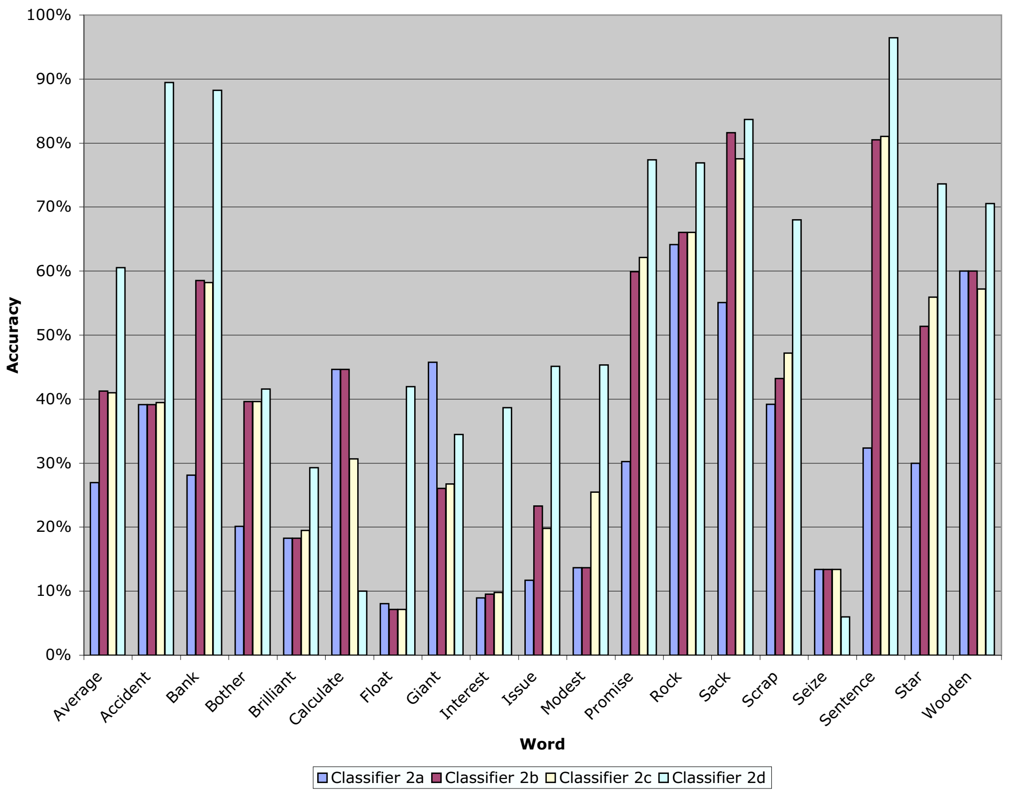


Figure 6: Accuracy of the four classifier variants that use sense weights distributed by dictionary order during training.

and 2d both exceed their average performance by a sizable margin with this word and outperform the other six varieties of classifier.

None of the eight variants of our algorithm outperform either baseline classifier. Once again, the large skew in the test set (nearly 93% of examples are noun occurrences of BFB) allows the baselines to achieve very high accuracy. Both baselines have roughly the same accuracy, since the test examples are 99% nouns, and the part-of-speech tagger is very accurate with this word.

7.3.3 Bother

"Bother" has both a noun and verb sense, but occurs mainly as a verb in our test set. Results for both baseline classifiers are thus similar. The most frequently occurring sense is the second listed verb sense, so both baselines are fairly inaccurate at 24.7%.

All eight of our classifier variants perform fairly poorly with this word; not surprising, since "bother" is not often topically related to its context. The use of part-of-speech tags to filter during disambiguation (classifiers 1b and 2b) does offers a significant benefit. Using other part-of-speech information does not help much, and adding part-of-speech filtering at training causes performance to degrade when used with uniform sense weights (classifiers 1b and 1c).

The results of using dictionary order to weight the senses at training is also mixed. Variant 2a underperformed variant 1a, but the other three variants improved to some degree when moving from uniform to non-uniform weight training.

All versions that use part-of-speech information

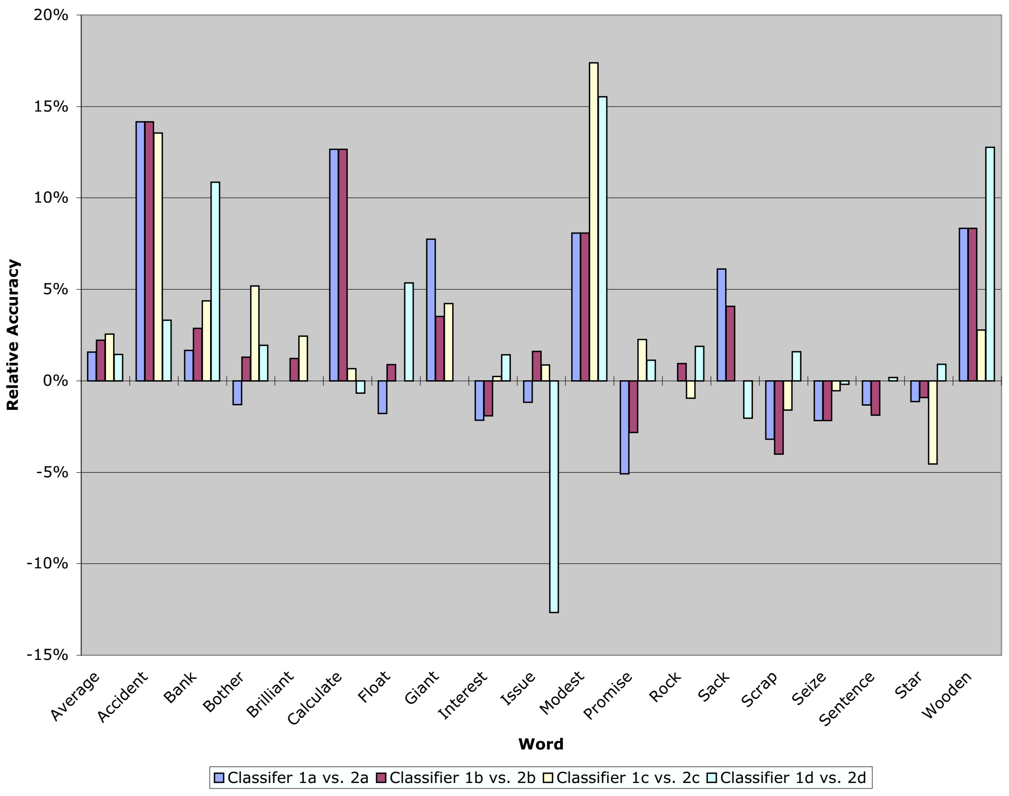


Figure 7: Comparison of classifiers trained with senses weighted by dictionary order as described in Section 4.5 to ones trained on a uniform distribution of sense weights. Each bar represents the difference in performance between a classifier that uses senses weighted by dictionary order during training to one that assumes uniformly weighted senses. The amount and type of part-of-speech information by the classifiers is the same in both members of each pair being compared.

(1b-d, 2b-d) outperformed the baseline, but this is more a reflection of the baseline's extremely low accuracy with this word than because of high performance by our classifiers.

7.3.4 Brilliant

Brilliant always occurs in our test set as an adjective, but because of inaccuracies in the part-of-speech tagger, performance declines slightly when we filter out the noun sense at disambiguation time (classifier 1b). We do not see the same decline in performance between classifiers 2a and 2b because the benefit of training with senses weighted by dictionary order counteracts the slight inaccuracies introduced by the part-of-speech tagger. In both the uniform and non-uniform weight versions, using part-of-speech information in the context (classifiers 1d and 2d) yields a decent benefit.

Weighting by dictionary order at training has a slightly positive effect on the "b" and "c" variant classifiers, but neither helps nor hurts the other versions.

The baseline performances for this word are low: both are at 23.2%. Classifiers 1d and 2d outperform this baseline accuracy, but the other six variants do not.

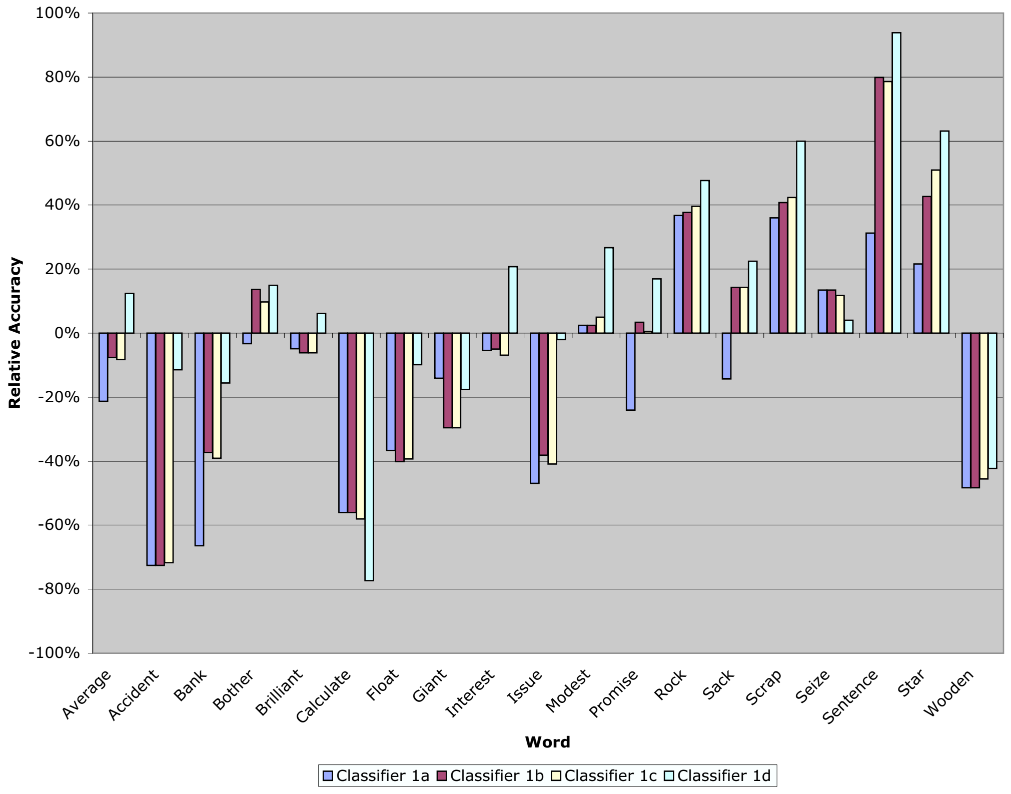


Figure 8: Comparison to Baseline 1 of the performance of the four variants using uniform training weights.

7.3.5 Calculate

This word is the only one in our test set where our classifiers consistently perform worse the more part-of-speech information we use. Adding part-of-speech filtering during training has no effect regardless of whether dictionary order is used or not during training, since calculate only occurs as a noun. However, adding part-of-speech filtering during training drops the accuracy of both uniform and frequency weighted versions to around 30%.[3] After adding part-of-speech information to the context, performance of both versions of the classifier plunge to around 10%.

Both baselines perform identically with accuracies of 88% and outperform all variants of our algorithm.

Weighting senses by their dictionary order during training appears to be a benefit for the first two variants (a and b): 2a and 2b turn in a score of 44.7% while 1a and 1b are lower at 32%. However, using this weighting does not improve the "c" variants much, and degrades the performance of the "d" variants.

7.3.6 Float

Adding part-of-speech data to the classifiers appears, at first, to be a bad idea with "float:" the variants that use part-of-speech as a filter (1b, 1c, 2b, 2c) underperform the ones that use no part-of-speech information (1a and 2a). Most likely, this behavior is

[3]If the word has no non-verb senses, how can limiting the senses by part-of-speech during training possibly have any effect? We must keep in mind that this is an *all words* classifier, and so the performance for one word is effected by the training of all words with which it shares semantic codes. The use of part-of-speech filtering during training affects other words that share semantic codes with "calculate" and thus indirectly affect it.

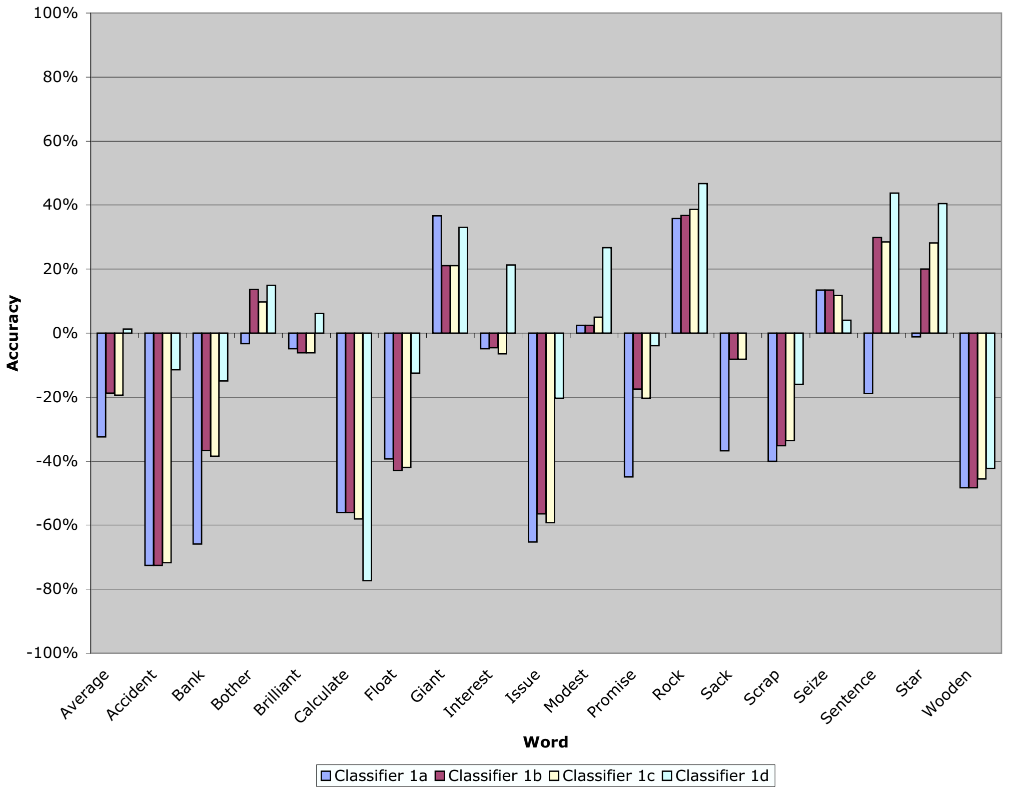


Figure 9: Comparison to Baseline 2 of the performance of the four variants using uniform training weights.

caused by the rather low performance of the Brill tagger with this word (83.2%).

However, adding part-of-speech information to the sense/context co-occurrence matrix causes a dramatic improvement. Variants 1d and 2d have 3.7 and 5 times higher performance than variants 1a and 2a, respectively.

Using dictionary order to weight senses by frequency during training also appears to be a mixed bag with this word. training version using no part-of-speech information slightly underperforms the uniform version. The variants that use part-of-speech tags as filters (b and c) have roughly the same performance regardless of the distribution of sense weights at training. However, the non-uniform variant 2d is 5.4 percentage points more accurate than the uniform version.

With "float," Baseline 1 correctly tags 46.1% of the test set examples, and Baseline 2 is slightly better at 49.1%. The baselines outperform all eight variants of our algorithm.

7.3.7 Giant

"Giant" demonstrates the Achilles' heal of our techniques: its need for good lexical resources and our assumption of accurate part-of-speech tagging.

While LDOCE3 in general has a good choice of senses for each word, in the case of "giant," its definitions left something to be desired. The noun senses of giant were all very similar. For example, LDOCE3 distinguishes between two senses of the meaning "large person" depending on whether the person being described is a fictional character or not. While potentially useful for the ESL students the dictionary targets, it is not a distinction that is easy for

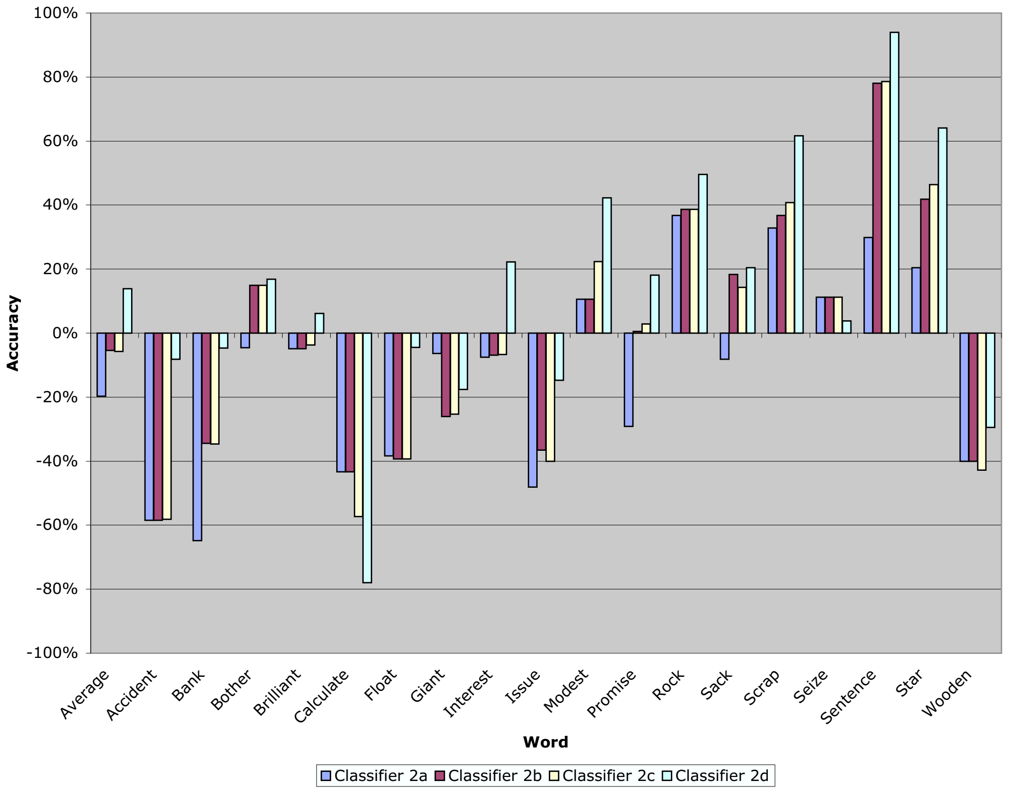


Figure 10: Comparison to Baseline 1 of the performance of the four variants using training weights distributed by dictionary order.

our algorithms to recognize. Additionally, the adjective entry has only a single sense meaning "big," a sense strangely lacking from the noun entry. One could use the "big company" meaning for the phrase "a pharmaceutical giant" but not for "giant electronics firm." Similarly, there is no good sense for "the old redwood was a giant among the newly planted saplings."

If this were the only problem, it would likely lead to a wildly over-accurate score for this word. Since "giant" is mainly used as an adjective in our test set, and it has only one adjectival sense, we *should* be choosing the only proper tag. Unfortunately, this word is one where the Brill tagger performs poorly: it correctly tags only 50.3% of the test set examples. Almost always, it applies the noun tag to instances of "giant," even when it is used as an adjective. This fact is also apparent from the baseline results. While the baseline algorithm that ignores part-of-speech does reasonably well (52.1%) choosing the listed-first, adjectival "big" sense, when part-of-speech tagging is enabled, performance drops off a precipice (1.4%).

Because of these problems, the variants that use no part-of-speech data are the best performing with "giant." The variants that use part-of-speech tag as filters are all significantly less accurate than either of the no part-of-speech versions. Adding part-of-speech information to the context recovers some of the performance, but both versions 1d and 2d are still less accurate than either 1a or 2a.

Training with senses weighted according to dictionary order provides a sizable benefit (7.7% points) to the variants not using part-of-speech data, and a small benefit to the versions that use part-of-speech

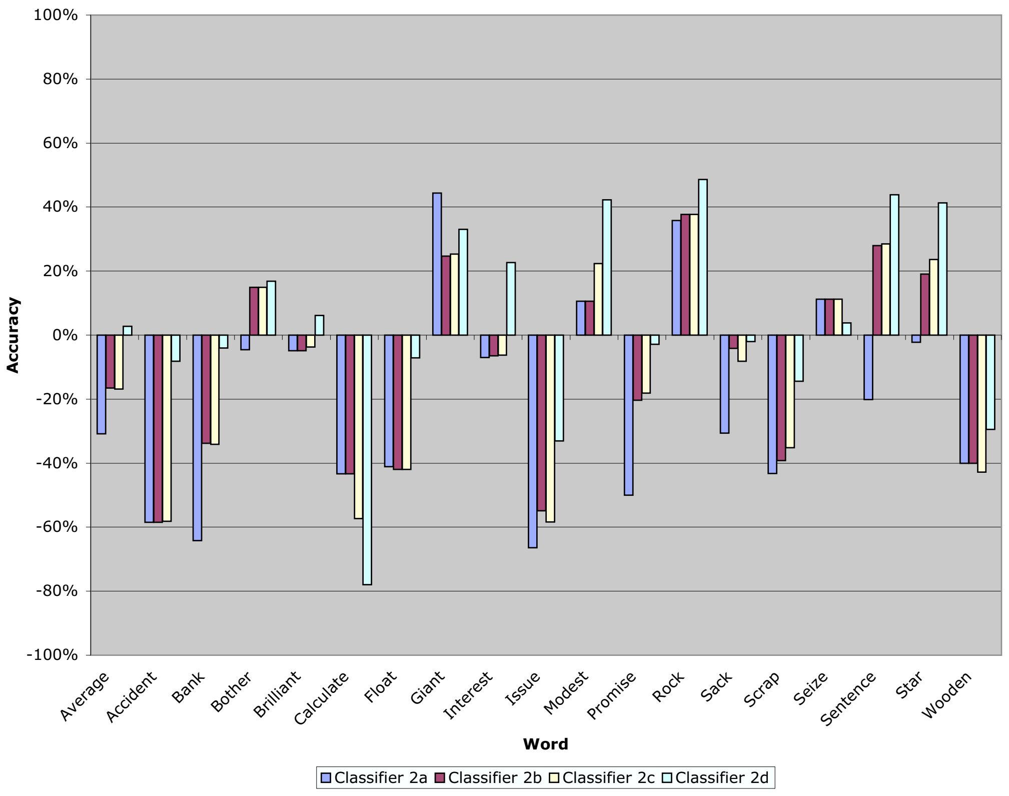


Figure 11: Comparison to Baseline 2 of the performance of the four variants using training weights distributed by dictionary order.

tags to filter senses. Changing the training weights makes no difference either way with the "d" variants.

Baseline 1 outperforms all versions of our algorithm with "giant," but baseline 2, with its miserable 1.4% accuracy, is trounced by all.

In general, part-of-speech data gives a big boost to the Yarowsky algorithm's performance. Unfortunately, though, when the part-of-speech tagger is incorrect, it *guarantees* that we will assign the wrong sense for a word. Since start-of-the-art part-of-speech taggers can achieve 97-98% accuracy, this should not be too much of a problem, but this case exhibits what happens when it goes wrong.

7.3.8 Interest

Using part-of-speech data as a filter during disambiguation gives only a very slight improvement with "interest." Using it as a filter during training as well gives mixed results: it helps the version trained with sense weights distributed by dictionary order slightly and hurts the version that uses uniformly distributed sense weights.

Adding part-of-speech information to the context once again helps a great deal, though. Performance increases from the 10% range to nearly 40% with the use of part-of-speech data in the context.

Training with senses weighted by their dictionary order has very little benefit with this word: variants 2a and 2b underperform their uniform weight counterparts, and 2c performs essentially the same as 1c.

There is a slight benefit to using non-uniform weights with the “d” variant: 2d’s performance is 1.4% points better than 1d.

Because the first listed sense is not the most frequently occurring, the baselines for this word are both quite low. Nevertheless, they still manage to beat six of the eight classifiers. However, variants 1d and 2d outperform both baselines by a comfortable margin.

7.3.9 Issue

Like “interest,” this word is fairly non-topical: it often occurs in contexts completely unrelated to its senses. Unlike “interest,” though, the test set examples are more evenly distributed among its two parts-of-speech and eight senses. As expected of a non-topical word with such a high degree of ambiguity, the classifiers that do not use part-of-speech (the “a” variants) perform very poorly.

Filtering at disambiguation helps significantly, no doubt because the part-of-speech tagger is highly accurate with “issue” and therefore cuts the ambiguity from eight senses to four for each example. Doing filtering during training does not help, though, and makes the accuracy slightly worse.

Once again, using part-of-speech in the context causes a dramatic increase in performance. Curiously, this behavior seems to fit the pattern established by “float” and “interest:” non-topical words with two parts-of-speech and low accuracies with the variants a-c. Though this phenomenon is notable, we have no real explanation for it.

The performance on “issue” demonstrates that training with a sense weight distribution based on dictionary order is not always beneficial: variant 2d is 12.7% points less accurate than 1d. The “a” variants also show less accuracy with the non-uniform weights, though the “b” and “c” variants both improve slightly with it.

The two baselines are both rather high with this word (59.8% and 78.2% respectively) and outperform all eight variants of our algorithm.

7.3.10 Modest

“Modest” has only one part-of-speech, so not unexpectedly, filtering on the part-of-speech tag during disambiguation has no effect. Filtering during training, does help both variants 1c and 2c to post more accurate scores than 1b and 2b. Using part-of-speech information in the context also has a large benefit with this word.

Weighting senses by dictionary order at training has a strongly positive effect on all our classifiers’ performances with “modest.” The “c” and “d” variants show the strongest benefit and post the two largest gains for this technique.

Because the second listed sense accounts for 95% of the test set examples, both baselines perform poorly. They are beaten by all eight versions of our algorithm.

7.3.11 Promise

The two most common senses for “promise,” one a noun and one a verb, both have the same semantic code, so the variants that do not use part-of-speech data are once again at a disadvantage with this word. Even if the classifier chooses the correct semantic code, it cannot distinguish between the noun and verb senses that both use that code, so can be at best 50% accurate. Part-of-speech filtering during disambiguation exactly addresses this problem. The performance of variants 1b and 2b are respectively about twice that of 1a and 2a, despite relatively low part-of-speech tagger performance on this word.

Like “bank,” which has a similar profile of senses, using part-of-speech tags to filter during training helps the classifier trained with senses weighted by dictionary order, but hurts the one that assumes uniformly distributed sense frequencies. Both training distribution variants benefit from using part-of-speech information in the context, leading to respectable accuracies of 76.3% and 77.4% respectively.

Baseline 1, at 59.3% outperforms the “a” variants, which use no part-of-speech, and is roughly even with the “b” and “c” variants. The “d” variants both significantly outperform baseline 1. Baseline 2, with 80.2% accuracy, outperforms all versions of our classifier.

7.3.12 Rock

Our classifiers show a small benefit to using filtering during disambiguation with “rock.” Filtering during training as well gives a slight improvement when uniform weights are used, but no change in either direction with the variants that use dictionary order to weight senses during training. Adding part-of-speech information to the context produces a significant benefit, but not as dramatic an improvement as we have seen with other words.

Weighting senses by dictionary order is of mixed benefit: no change for the "a" variants, small improvements in the "b" and "d" variants, and a small degradation with the "c" variants.

Both baseline 1 and baseline 2 are fairly inaccurate with this word. All eight versions of our algorithm outperform both baselines.

7.3.13 Sack

Using part-of-speech as a filter during disambiguation provides the greatest benefit with "sack," while using it in the context also helps. However, using it as a filter during training is no help: variant 1c performs no better or worse than 1b, while 2c underperforms 2b. Dictionary order weights at training has mixed benefits: it helps variants "a" and "b," but harms the "d" variants.

Baseline 1, with an accuracy of 63.3%, beats variants 1a and 2a, but is less accurate than the other six. Baseline 2, significantly more accurate at 85.7%, outperforms all but variant 1d, whose performance it equals.

7.3.14 Scrap

"Scrap" shows a steady increase in performance as more part-of-speech information is used in the classifier. The "d" variants show the biggest jump in accuracy, with the others giving smaller benefits.

Dictionary order weights seem to hurt more than help with "scrap." Variants 2a-c each underperform their respective uniform weight version. However, version 2d is slightly more accurate than 1d and, with 68% accuracy, turns in the highest score among our classifiers on this word.

Because the noun senses are listed ahead of the more common verb senses, baseline 1 is very inaccurate and all eight versions of our algorithm beat it. Part-of-speech tagging fixes this problem, so baseline 2 beats all versions of the algorithm.

7.3.15 Seize

"Seize" only has one part-of-speech, so as expected, using filtering during disambiguation does nothing. However, performance slides downhill from there. Performing filtering during training, causes the uniform weight classifier's (1c) performance to slide, while it has no effect on the version trained with senses weighted by dictionary order (2c). Adding part-of-speech to the context has the same negative effect as with "calculate:" performance plunges by more than half.

Training with sense weights distributed according to dictionary order also is no help with this word: all four non-uniform variants perform worse than their equivalents that assumed a uniform distribution of sense frequencies.

Even the baselines perform poorly with "seize." Both have an accuracy of only 2.2% and are are beaten be even the dismal performances of the eight versions of our algorithm.

7.3.16 Sentence

Noun and verb senses with common semantic codes, a relatively skewed distribution of senses, and a high performance from the part-of-speech tagger all combine to allow part-of-speech filtering during disambiguation to show a very large improvement in accuracy over the classifiers that use no part-of-speech information. Variant 1b's accuracy is more than double the 33.7% of 1a, and the non-uniform versions show similar results.

Like "bank" and "promise," which also have overlapping codes and high skew, filtering using part-of-speech at training only helps when used in conjunction with the training weight modification. The same technique with uniform weights causes performance to drop slightly.

Finally, adding part-of-speech information to the context further raises performance to an excellent 96.2% for variant 1d and 96.4% for variant 2d.

Training with senses distributed by their dictionary order is not nearly as effective as part-of-speech information with this word. The "a" and "b" variants both show less accuracy with non-uniform weights, while the "c" and "d" variants show little effect either way.

Because the first listed sense is by far the least common, baseline 1 performs poorly, with a score of 2.4%. Baseline 2 does slightly better, because it gets almost all of the verb examples right, since "sentence" only has a single verb sense. Its score of 52.5% outperforms variants 1a and 2a, but is worse than that of the other six classifiers.

7.3.17 Star

Classifier performance with "star" improves with each additional part-of-speech element, with the "b" and "d" variants showing the largest jumps. Training with senses weighted by dictionary order, though, only benefits the "d" variants. It reduces the performance of the other three types of classifiers.

Both baselines are quite inaccurate: baseline 1 at 9.5% and baseline 2 at 32.3%. Except for baseline 2 outperforming the two variants that do not use part-of-speech information (1a and 2a), our algorithms do better than the baselines on this word.

7.3.18 Wooden

Not only does "wooden" have only a single part-of-speech, but all 180 test set examples occur with its first sense meaning "made of wood." Therefore, both baselines are 100% accurate and outperform all of our algorithm variants.

Part-of-speech information used as a disambiguation filter has no effect, since there is only one possible sense. Filtering at training helps when assuming a uniform distribution of senses, but hurts when senses are weighted by their dictionary order. Both uniform and non-uniform versions see some benefit from using part-of-speech in the context.

All four types of classifiers strongly benefit from training with sense weights distributed by dictionary order. The "d" variants benefit the most, while the "c" variants show the least improvement.

7.4 Summary of Results

While we did not succeed in creating an especially successful new classifier, we have shown that simple, low cost approaches can improve an already well-established algorithm. Both using part-of-speech data and using training weights distributed by dictionary order improved the performance of the Yarowsky algorithm.

7.4.1 The Part-of-Speech Experiment

Overall, the use of part-of-speech tagging as a preprocessing step prior to applying the Yarowsky algorithm appears to have a significant benefit. Not surprisingly, words that have the same semantic code appearing in senses with different parts-of-speech show the greatest benefits from using part-of-speech tags to limit senses during disambiguation. All words with multiple parts-of-speech showed some benefit from this technique, except where the part-of-speech tagger's accuracy was too low. Doing a similar type of filtering during training, on the other hand, does not benefit most words and actually decreases the average accuracy across the whole test set.

Of particular interest is the improvement gained by using part-of-speech information in the context. Using this information increases the size of the collocation matrix by about a factor of 1.5. Since the number of training examples remains the same, the sparseness of this matrix will also increase. Despite this increase in data sparseness, using part-of-speech data in the context showed a clear benefit of the same order as the more obvious use as a filter during disambiguation. Clearly, the usages of the words that co-occur with a particular sense, not just the words themselves, must be significant indicators of the proper sense of the target.

We are particularly pleased that the use of part-of-speech data has few harmful effects: the standard classifier outperforms our variants by a sizable margin in only three cases. The low risk, high return, and very small cost of this technique make it a useful augmentation of the standard Yarowsky algorithm.

7.4.2 The Dictionary Order Weight Training Experiment

Training with a sense distribution weighted by dictionary order also yielded increased accuracy compared to the uniformly weighted senses used to train in the standard Yarowsky classifier. However, this technique gave a much smaller benefit than using part-of-speech information. On average, results improved 1.5-2.5 percentage points when we used non-uniform weight training.

This slight but noticeable improvement suggests that more refinement is necessary before we can recommend non-uniform weight training always be used.

7.4.3 Comparison to the Baselines

Both of the baseline classifiers were surprisingly successful with this data set. When we look at the comparison graphs, two facts are immediately apparent. The first, that the Yarowsky algorithm is slightly worse than both baselines with LDOCE3 and our data sets. The second, that adding additional knowledge to the original algorithm significantly improves its accuracy and raises its performance well above the first and equal to or slightly above the second.

By using easily obtainable knowledge sources — the part-of-speech tags and the relative sense frequencies — we were able to construct an algorithm capable of slightly outperforming both baselines. Though the initial algorithm significantly underperformed, we were able to improve it with little additional computer resources and no additional human intervention.

Interestingly, these same two knowledge sources allowed us to construct a classifier that performs nearly as well the most sophisticated version of our algorithm: namely, baseline 2. Baseline 2's average performance of 57.8% is nearly the same as that of classifier 2d at 60.6%, but the baseline algorithm requires less than 1% of the time and storage needed by the Yarowsky-based classifiers. However, the variance of baseline 2's accuracy is significantly higher than our best classifier: 0.12 versus 0.08, which may affect its usefulness.

7.5 The Iterative Re-training Experiment

Before we achieved success with using part-of-speech information, we experimented with other methods for improving performance on the Yarowsky algorithm. Most promising of these was performing iterative retraining of the algorithm. The results of this experiment are summarized in Figure 12.[4]

Results from the approach are mixed. Not surprisingly, words that were disambiguated accurately by the standard Yarowsky classifier improved with this method, while those that could not be disambiguated well got worse. Unfortunately, performance on some successfully discriminated words, like "rock" and "promise" also declined with this method.

In hindsight, these results are obvious: by assuming our first stage algorithm is correct, we reinforce its behavior on subsequent iterations. While individual words often showed significant improvement or degradation, averaged across several test words, this approach showed a slight decline in overall performance.

The missing piece that may allow this technique to work more successfully is an automated method for discriminating words where the classifier performed well from those where it did poorly. Such a discrimination procedure would allow this technique to work more like a classical boosting approach. However, instead of retraining wrong examples, we would reinforce the classifier where it performed correctly.

We tried a few possible methods for automatically gauging the classifier's accuracy, but met with no success. If such a decision procedure could be found, it could allow us to boost the performance on many words, while leaving other less accurate ones unchanged.

8 Further Work

Our experiments with these classifiers suggest several areas of further investigation that may yield fruitful results. There are unexploited opportunities to improve performance further along with interesting applications of this technology to other problems.

8.1 Formalizing these Results

Our empirical results contained many interesting patterns in the data. Some have obvious causes: for example, it is clear how filtering on the part-of-speech tags during disambiguation helps words that have senses with common semantic codes. Others are less clear: we would like to know the validity of our supposition about why filtering at training tends to be harmful with words with senses that have common semantic codes.

Finally, we still do not have a satisfying explanation for why using part-of-speech with context words is so effective despite the dramatic increase in the sparseness of co-occurrence data. It would also be

[4] Please note that the testing corpora, answer keys, and scoring algorithm in this experiment were significantly different and less well-defined than in the part-of-speech data experiment. Comparison of the results of these two experiments, aside from average trends, is not possible

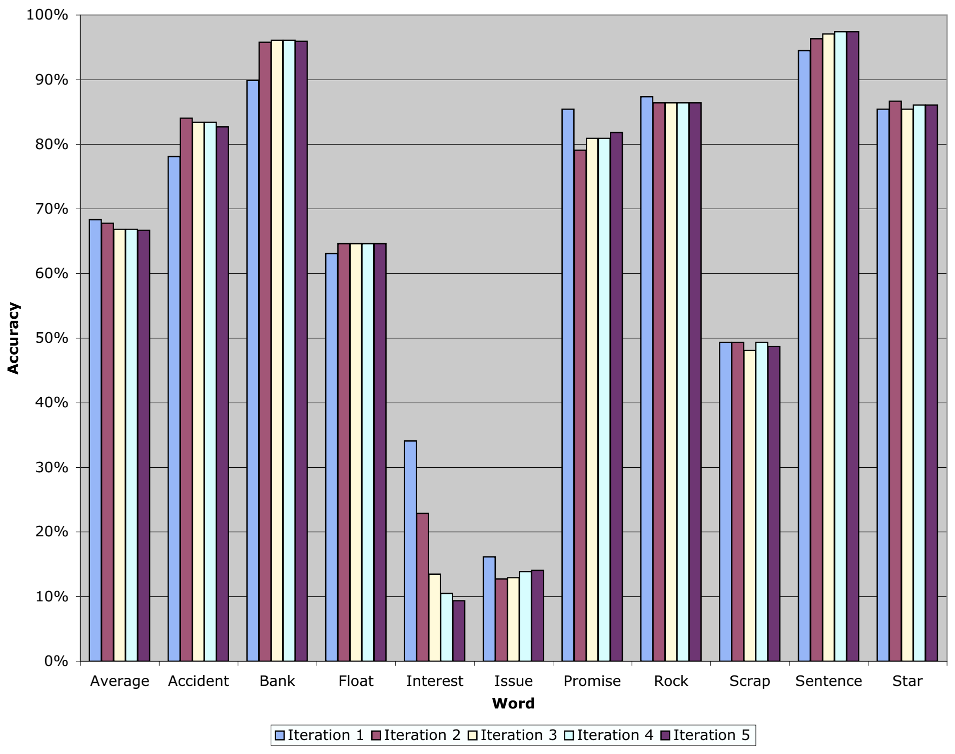


Figure 12: Iterative re-training results.

very helpful to understand just what went wrong with using part-of-speech data with "calculate" and "seize."

8.2 Additional Knowledge Sources

LDOCE3 contains a wealth of lexical information. We are only exploiting a very small portion of it in our system. There are sections that detail usage examples or limit where a sense can be used. There are the often metaphorical "LEXUNIT" senses that we currently ignore. This information, as well as information contained in other lexical resources, may prove useful for further improvements to our classifier.

For instance, if usage examples are provided, it may be possible to determine that certain senses always occur in set phrases, with certain grammatical agreement, or have some other characteristic that can be used to narrow down the choice of semantic codes for an unlabeled word.

8.3 Building a Better Semantic Model

The activator and field codes of LDOCE3 provide a convenient set of semantic tags for use with a Yarowsky classifier. However, using them in the same way as thesaurus category codes does not realize their full potential.

The field codes in LDOCE3 are arranged in a three level hierarchy. For example, the "financial institution" meaning of bank has the code BFB, for "business/finance/banking." During training, words that co-occur with the BFB sense will become more likely to indicate this sense of bank during disambiguation. However, words that co-occur with related codes like BF (business/finance) and BFL (business/finance/loans) may not. Since the field codes are arranged by semantic similarity, perhaps words that co-occur with related codes can also be counted as evidence during disambiguation. The main problem we foresee with this approach is that it may end up favoring field codes over activator codes because of all the additional evidence being introduced through the field code hierarchy.

One could also imagine a similar but less hierarchical relatedness graph being applied to the LDOCE3 activator codes. Since there are only about 1000 such codes, it may be feasible to create such a simple semantic network by hand. It would be far more interesting, however, to try to build up such a network through automated means.

8.4 Applications

Word sense disambiguation, while an interesting problem, is rarely an end in itself. Originally, it was a tool for improving the performance of machine translation systems. While it still plays a role in MT, there are other far more feasible applications for it.

One possible use for our system is an automatic text classification system like IAGO [6] which used the standard Yarowsky algorithm. Presumably, a classifier with higher accuracy will also make a better document classifier.

An exciting possibility is to use this classifier to assist in image recognition. Several colleagues are building a system that clusters images based on keyword descriptions [3]. We are exploring the possibility of using the classification capabilities of this algorithm to improve their system.

9 Conclusion

In an application as complex as natural language processing, the more information that can be brought to bear, the better the results are likely to be. We experimented with three possible techniques for improving the accuracy of the well-known Yarowsky classifier: using part-of-speech information in several places, replacing the assumption during training of uniformly distributed senses with a distribution weighted according to sense frequency, and bootstrapping the algorithm with training material generated by previous versions of the classifier.

Of these three approaches, using part-of-speech information provides the most dramatic benefit: it more than doubled the performance of the standard Yarowsky algorithm — from 25.4% to 59.1% accuracy. Words with senses fairly evenly distributed amongst their parts-of-speech show even greater benefit: disambiguation accuracy on "issue" improved nearly five-fold, from 12.9% to 57.8%.

These benefits come at very little cost: the Brill tagger is efficient, so it does not contribute greatly to the training or disambiguation time. There were no serious degradation when using part-of-speech data: our system was worse than the Yarowsky tagger in only three of eighteen cases.

Our main weakness is the accuracy of the Brill tagger and the size and composition of the training data. A more accurate part-of-speech tagger — there are several available that claim 98% or greater accuracy — would probably improve our technique somewhat. Replacing the Brill tagger would be an easy modification. Larger and more varied training material, while sometimes hard to obtain, would also be trivial to add to our system.

Our second modification, training using a weighting of senses distributed according to dictionary order, produces a slight additional improvement. This modification gives a 1.5-2.5 percentage point increase in the accuracy of the standard Yarowsky algorithm and all three of our variants that use part-of-speech information. While its benefits are small, its performance cost is essentially zero, and it appears to have low risk of performance degradation on any individual words.

These same two knowledge sources also allowed us to construct a classifier that performs nearly as well as our best Yarowsky-based algorithm but with far lower computational complexity. The second baseline algorithm, which simply chooses the first sense listed in LDOCE3 for a given word's part-of-speech, had nearly the same average accuracy as our best variant. This simple classifier does not require any training (which takes almost six hours for the Yarowsky-based ones) or a large database of word/sense co-occurrence information. It also disambiguates faster, since it does not need to examine any context. Baseline 2 does exhibit higher variance in its accuracy on our test set, which may limit its usefulness in general.

Our last experiment, iterative retraining, was not as successful. The average performance stays the same or got slightly worse with each re-training operation. We still believe that there is potential for improvement with this technique, if we can find a reliable metric for evaluating classifier performance.

By adding additional, easy to obtain knowledge sources — an accurate part-of-speech tagger, and the relative sense frequencies from LDOCE3 — to an established disambiguation algorithm, we were able to improve discrimination accuracy with virtually no additional cost. We demonstrated that we can achieve a significant accuracy improvement while retaining the elegant simplicity of the Yarowsky algorithm. For some applications, these same two knowledge sources can be used without any machine learning and still achieve reasonably high accuracy, as with baseline 2.

We expect that similar systems that exploit combinations of simple techniques will further increase discrimination accuracy without the need to resort to excessively large complex word sense disambiguation systems.

Acknowledgements

Many thanks to Professor Marti Hearst for taking the time to review several drafts of this work and offer many helpful suggestions. In particular, her suggestion to use dictionary order as a basis for the two baseline algorithms greatly improved the results discussion.

Thanks also to Professor Sadao Kurohashi of the University of Tokyo for hosting me as an NSF/Monbukagakusho visiting researcher for two months in the summer of 2001. Thanks to his and his lab's helpful suggestions and excellent resources, I was able to make great progress on key phases of this project.

This research was supported by the Digital Libraries Initiative under grant NSF CA98-17353.

References

[1] James Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc., 1995.

[2] Yehoshua Bar-Hillel. Automatic translation of languages. *Advances in Computers*, 1:91–163, 1960.

[3] Kobus Barnard, Pinar Duygulu, and David Forsyth. Clustering art. In *Computer Vision and Pattern Recognition*, 2001.

[4] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, IT, 1992.

[5] Eric Brill. Some advances in transformation-based part of speech tagging. In *AAAI, Vol. 1*, pages 722–727, 1994.

[6] Isaac Cheng and Robert Wilensky. An experiment in enhancing information access by natural language processing. Technical Report CSD-97-963, University of California, Berkeley, Computer Science Division, 1997.

[7] Microsoft Corporation. Encarta 97 encyclopedia, 1997.

[8] Jim Cowie and Louise Guthrie. Lexical disambiguation using simulated annealing. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, 1992.

[9] William A. Gale, Kenneth W. Church, and David Yarowsky. A method for disambiguating word senses in a large corpus. *Computers and the Humanities*, 26:415–439, 1993.

[10] Philip J. Hayes. On semantic nets, frames, and associations. In *Proceedings of the International Joint Converence on Artificial Intelligence (IJCAI)*, 1977.

[11] Nancy Ide and Jean Véronis. Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics*, 24(1), 1998.

[12] Yael Karov and Shimon Edelman. Similarity-based word sense disambiguation. *Computational Linguistics*, 20(10), 1997.

[13] Jerrold J. Katz and Jerry A. Fodor. The structure of a semantic theory. *Language*, 39(2), 1963.

[14] Adam Kilgarriff. Senseval: An excercise in evaluating word sense disambiguation programs. In *LREC*, pages 581–588, Granada, May 1998.

[15] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of ACM SIGDOC Conference*, pages 24–26, 1986.

[16] Longman. *Dictionary of Contemporary English.* Addison Wesley, third edition, 1997.

[17] Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.

[18] Rada Mihalcea and Dan I. Moldovan. An automatic method for generating sense tagged corpora. In *AAAI/IAAI*, pages 461–466, 1999.

[19] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to wordnet: an on-line lexical database. *International Journal of Lexicography (special issue)*, 3(4):235–312, 1990.

[20] Raymond J. Mooney. Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1996.

[21] Hwee Tou Ng and Hian Beng Lee. Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 40–47, San Francisco, 1996. Morgan Kaufmann Publishers.

[22] David Palmer. SATZ - an adaptive sentence segmentation system. Master's thesis, University of California, Berkeley, Computer Science Division, December 1994. Available as UCB Technical Report CSD-94-845.

[23] David Palmer and Marti Hearst. Adaptive sentence boundary disambiguation. Technical Report CSD-94-797, University of California, Berkeley, Computer Science Division, February 1994.

[24] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[25] M. Ross Quillian. Semantic memory. In Marvin Minsky, editor, *Semantic Information Processing.* 1968.

[26] Robert E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[27] Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1), 1998.

[28] R. F. Simmons. Semantic networks: Their computation and use for understanding english sentences. In R. C. Schank and K.M. Kirby, editors, *Computer Models of Thought and Language.* 1973.

[29] Mark Stevenson and Yorick Wilks. The grammar of sense: Using part-of-speech tags as a first step in semantic disambiguation. *Journal of Natural Language Engineering*, 1997.

[30] Mark Stevenson and Yorick Wilks. The interaction of knowledge sources in word sense disambiguation. *Computational Linguistics*, 27(3):321–349, 2001.

[31] Yorick Wilks. An intelligent analyzer and understander or english. *Communications of the ACM*, 18(5), 1975.

[32] David Yarowsky. Word-sense disambiguation using statistical models of roget's categories trained on large corpora. In *Proceedings of COLING-92*, pages 454–460, Nantes, France, 1992.

[33] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Meeting of the Association for Computational Linguistics*, pages 189–196, 1995.