Bootstrapping for Text Learning Tasks

Rosie Jones[1] rosie@cs.cmu.edu
Andrew McCallum[2,1] mccallum@justresearch.com
Kamal Nigam[1] knigam@cs.cmu.edu
Ellen Riloff[3] riloff@cs.utah.edu

[1]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[2]Just Research
4616 Henry Street
Pittsburgh, PA 15213

[3]Department of Computer Science
University of Utah
Salt Lake City, UT 84112

Abstract

When applying text learning algorithms to complex tasks, it is tedious and expensive to hand-label the large amounts of training data necessary for good performance. This paper presents bootstrapping as an alternative approach to learning from large sets of labeled data. Instead of a large quantity of labeled data, this paper advocates using a small amount of seed information and a large collection of easily-obtained unlabeled data. Bootstrapping initializes a learner with the seed information; it then iterates, applying the learner to calculate labels for the unlabeled data, and incorporating some of these labels into the training input for the learner. Two case studies of this approach are presented. Bootstrapping for information extraction provides 76% precision for a 250-word dictionary for extracting locations from web pages, when starting with just a few seed locations. Bootstrapping a text classifier from a few keywords per class and a class hierarchy provides accuracy of 66%, a level close to human agreement, when placing computer science research papers into a topic hierarchy. The success of these two examples argues for the strength of the general bootstrapping approach for text learning tasks.

1 Introduction

Text learning algorithms today are reasonably successful when provided with enough labeled or annotated training examples. For instance, text classifiers [Lewis, 1998; Joachims, 1998; Yang, 1999; Cohen and Singer, 1996; Schapire and Singer, 1999] reach high accuracy from large sets of class-labeled documents; information extraction algorithms [Califf, 1998; Riloff, 1993; Soderland, 1999; Freitag, 1998] perform well when given many tagged documents or large sets of rules as input. However, as more complex domains are considered, the requisite size of these training sets gets prohibitively large. Creating these training sets becomes tedious and expensive, since typically they must be labeled by a person. This leads us to consider learning algorithms that do not require such large amounts of labeled data.

While labeled data is difficult to obtain, *unlabeled* data is readily available and plentiful. Castelli and Cover [1996] show in a theoretical framework that unlabeled data can be used in some settings to improve classification, although it is exponentially less valuable than labeled data. Fortunately, unlabeled data can often be obtained by completely automated methods. Consider the problem of classifying news articles: a short Perl script and a night of automated Internet downloads can fill a hard disk with unlabeled examples of news articles. In contrast, it might take several days of human effort and tedium to label even one thousand of these, depending on the nature of the task.

However, one cannot learn to perform classification from just unlabeled data alone. By itself, unlabeled data describe the *domain* of the problem, but not the *task* over the domain. Thus, unlabeled data must be coupled with at least some information about the target function for the learning task. This target, or seed, information can come in many different forms, such as keywords or features which may appear in examples of the target classes, or a small number of examples of the target classes.

This paper advocates using a bootstrapping framework for text learning tasks that would otherwise require large training sets. The input to the bootstrapping process is a large amount of unlabeled data and a small amount of seed information to inform the learner about the specific task at hand. In this paper we consider seed information in the form of keywords associated with classes. Bootstrapping initializes a learner with the keywords. It then iterates, applying the learner to hypothesize labels for unlabeled data, and building a new learner from these bootstrapped labels.

We present two instantiations of the bootstrapping approach for different text learning tasks. The first case study is learning extraction patterns and dictionaries for information extraction, using keywords and a parser as the only knowledge supplied. We use a combination of two classifiers: a noun-phrase classifier and a noun-phrase context classifier based on extraction patterns. The seeding gives us a small number of relatively reliable training examples (noun phrases), which we then use to bootstrap the two classifiers. During the bootstrapping process the output of one classifier becomes the input for the other. The noun phrases help us identify good extraction patterns, and the extraction patterns help us identify new noun phrases. This combined bootstrapping is nested inside a higher level of bootstrapping, called *meta-bootstrapping*, which identifies the most reliable noun phrases generated by the inner bootstrapping loop.

We perform experimental evaluation of the information extraction bootstrapping algorithm by generating dictionaries for locations from corporate web pages used in the WebKB project [Craven *et al.*, 1998]. Meta-bootstrapping identifies 191 location phrases in the web pages. After 50 iterations, 76% of the hypothesized location phrases on the web pages were true locations. This algorithm has been described in isolation in [Riloff and Jones, 1999]; here we place it into the larger context of bootstrapping for text learning.

The second case study of bootstrapping is document classification using a naive Bayes classifier, where knowledge about the classes of interest is provided in the form of a few keywords per class and a class hierarchy. We present extensions to naive Bayes that succeed in this task without requiring sets of labeled training data. The extensions reduce the need for human effort by (1) using keyword matching to automatically assign approximate labels, (2) using a statistical technique called shrinkage that finds more robust parameter estimates by taking advantage of the hierarchy, and (3) increasing accuracy further by iterating Expectation-Maximization to probabilistically reassign approximate labels and incorporate unlabeled data.

Experimental evaluation of the document classification bootstrapping approach is performed on a data set of computer science research papers. A 70-leaf hierarchy of computer science and a few keywords for each class are provided as input. Thirty thousand unlabeled research papers are also provided. Keyword matching alone provides 45% accuracy. Our bootstrapping algorithm uses this as input and outputs a naive Bayes text classifier that achieves 66% accuracy. Interestingly, this accuracy approaches estimated human agreement levels of 72%.

This paper proceeds as follows. Section 2 discusses the general framework of the bootstrapping process and outlines the design space of bootstrapping algorithms. Section 3 presents one instantiation of bootstrapping for information extraction. A second example, for text classification, is given in Section 4. Related work is presented in Section 5, and discussion follows in Section 6.

2 Overview of Bootstrapping Algorithm

Bootstrapping is a general framework for improving a learner using unlabeled data. Typically, bootstrapping is an iterative process where labels for the unlabeled data are estimated at each round in the process, and the labels are then incorporated as training data into the learner. The bootstrapping described in this paper consists of the following steps:

- *Initialization*: A small number of hand-chosen seed information (in this paper, keywords) for each class are applied to the entire set of unlabeled examples, resulting in labels for those examples matching the keywords.
- *Iterate Bootstrapping*:
  - *Using Labels to Improve the Model*: The information from the labeled examples is used to generate a new, more reliable, model.
  - *Relabeling*: The model is used to generate new labels for the unlabeled data.

2.1 Initialization

In both instantiations of bootstrapping in this paper, seed information is provided in the form of keywords. The text-learning tasks we address involve the content and topics of documents and parts of documents, which previous research on feature selection has shown to be at least partially discernible from a few individual words within the text [Lewis and Ringuette, 1994; Cohen, 1996]. We do not require the keywords to label all examples, nor do we require every label they provide to be correct. Instead, we expect these labels to give us areas of greater-then-random confidence from which to begin a clustering process.

In general, keywords tend to have the nature of high precision and low recall. For example, Cohen [1996] showed that his system RIPPER, using only five words in the feature set, can grow rules which classify titles of AP news articles with 86% precision, at a recall level of 18%. This property is often an important feature of the seed information for bootstrapping, because it allows bootstrapping to iteratively trade off some precision for significantly higher recall.

The individual domain and learning task will determine the number of words in an example and their distribution, thus affecting precision and recall of the initial labeling. For this reason, our methods need to be robust to variations in the reliability of the initial set. The algorithm described in Section 3 assumes a relatively reliable initial set of examples. Given the domain and task (finding place-names as substrings of documents while avoiding false hits) this is a fair assumption. The task described in Section 4 involves distinguishing between documents in related fields in computer science. Here we might expect greater ambiguity in the keyword assignment, and the algorithm we use there is more robust to such ambiguity.

2.2 Bootstrapping

We can regard bootstrapping as iterative clustering. Given the initial class-based clusters of labeled examples provided as input, we re-cluster, and output a new set of labeled examples.

Bootstrapping can relabel examples in different ways. In Section 3 we describe an approach which relabels some examples with high confidence, effectively adding examples to the training pool iteratively. This algorithm relies on restarting the process at periodic intervals, to resume with the most confident predictions. In Section 4, we describe an algorithm that uses Expectation Maximization to relabel all examples probabilistically—expressing weaker confidence in those labels.

These are two points in a continuum we can trace between high-confidence labeling and low-confidence labeling, and using few or all labeled examples to retrain the models. Note that more computation is required when all examples are used to retrain models, and more confidently labeled examples are required when we use fewer.

3 Bootstrapping Dictionaries for Information Extraction

The wealth of on-line text has produced widespread interest in the problem of *information extraction*. Information extraction (IE) systems try to identify and extract specific types of information from natural language text. Most IE systems focus on information that is relevant to a particular domain or topic. For example, IE systems have been built to extract the names of perpetrators and victims of terrorist incidents, and the names of people and companies involved in corporate acquisitions. IE systems have also been developed to extract information about joint venture activities [MUC-5 Proceedings, 1993], microelectronics [MUC-5 Proceedings, 1993], job postings [Califf, 1998], rental ads [Soderland, 1999], and seminar announcements [Freitag, 1998].

Most information extraction systems rely on two dictionaries: a dictionary of extraction patterns and a semantic lexicon.[1] In the past few years, several techniques have been developed to automate the construction of these dictionaries. However, most of these methods rely on special training data. For example, AutoSlog [Riloff, 1993; 1996a], CRYSTAL [Soderland *et al.*, 1995], RAPIER [Califf, 1998], SRV [Freitag, 1998], and WHISK [Soderland, 1999] need a training corpus that includes annotations for the desired extractions, and PALKA [Kim and Moldovan, 1993] and LIEP [Huffman, 1996]) require manually defined keywords, frames, or object recognizers. AutoSlog-TS [Riloff, 1996b] has simpler needs but still requires a corpus of texts that have been labeled as relevant and irrelevant to the domain.

Using bootstrapping techniques described in section 2, we have developed an algorithm that can learn dictionaries for information extraction without any special training resources. Our bootstrapping technique generates extraction patterns and a semantic lexicon simultaneously, using only texts that are representative of the domain and small set of seed words. Our approach is based two observations.

1. *Objects that belong to a semantic category can be used to identify extraction patterns for that category.* For example, suppose we know that the words "schnauzer", "terrier", and "dalmation" all refer to dogs. We may then discover that the pattern "<X> barked" extracts many instances of these words and infer that it is a useful pattern for extracting references to dogs.

2. *Extraction patterns for a semantic category can be used to identify new members of that category.* For example, suppose we know that "<X> barked" is a good pattern for extracting dogs. Then we can infer that every noun phrase that it extracts is a reference to a dog. (This inference will not always be correct, but it should be correct more often than not.)

Our bootstrapping algorithm begins with a small set of seed words that belong to a semantic category of interest. These seed words are used to learn extraction patterns that reliably extract members of the same semantic class. The learned extraction patterns are then used to generate new category members, and the process repeats. We call this process *mutual bootstrapping* because it generates both a semantic lexicon and a dictionary of extraction patterns at the same time. We have also found it useful to introduce a second level of bootstrapping that retains only the most reliable lexicon entries produced by the mutual bootstrapping process and then re-starts it from scratch. This two-tiered bootstrapping process is more robust than a single level of bootstrapping and produces highly-quality dictionaries.

3.1 Mutual Bootstrapping

The mutual bootstrapping process begins with a text corpus and a small set of predefined seed words for the semantic category of interest. First, a set of candidate extraction patterns is generated by running AutoSlog [Riloff, 1993; 1996a] exhaustively over the text corpus. Given a noun phrase (NP) to extract, AutoSlog uses heuristics to generate a linguistic expression that represents relevant context for extracting the NP. This linguistic expression should be general enough to extract other relevant noun phrases as well. By applying AutoSlog exhaustively, it generates a pattern to extract every noun phrase in the corpus. Once all candidate extraction patterns have been generated, we apply them to the corpus and record the noun phrases (NPs) that they extract.

We now have all possible extraction patterns for the corpus[2] and all noun phrases that they extract, so one can view the next step as the process of labeling this data. That is, we want to determine which noun phrases belong to the semantic category of interest, and which extraction patterns will reliably extract members of the category. When we say that we "learn" an extraction pattern or a semantic lexicon entry, we really mean that we are labeling the extraction pattern or noun phrase as belonging to the semantic category.

[1]For our purposes, a semantic lexicon simply refers to a dictionary of words or phrases with semantic category labels.

[2]That is, all extraction patterns that AutoSlog can generate.

Generate all candidate extraction patterns from the training corpus using AutoSlog.

Apply the candidate extraction patterns to the training corpus and save the patterns with their extractions to *EPdata*

$SemLex = \{\text{seed\_words}\}$

$Cat\_EPlist = \{\}$

MUTUAL BOOTSTRAPPING LOOP

1. Score all extraction patterns in *EPdata*.
2. *best_EP* = the highest scoring extraction pattern not already in *Cat_EPlist*
3. Add *best_EP* to *Cat_EPlist*
4. Add *best_EP*'s extractions to *SemLex*.
5. Go to step 1

Figure 1: Mutual Bootstrapping Algorithm

Figure 1 outlines the mutual bootstrapping algorithm, which iteratively learns extraction patterns from the seed words and then exploits the learned extraction patterns to identify more words that belong to the semantic category. At each iteration, the algorithm saves the best extraction pattern for the category to a list (*Cat_EPlist*). All of its extractions are inferred to be category members and added to the semantic lexicon (*SemLex*). Then the next best extraction pattern is identified, based on both the original seed words and the new words that were just added to the lexicon, and the process repeats.

Since the semantic lexicon is constantly growing, the extraction patterns need to be rescored after each iteration. The scoring measure counts how many unique lexicon entries a pattern extracts. This measure rewards generality: a pattern that extracts several different category members will score higher than a pattern that extracts only one or two category members, no matter how often. To score extractions, our scoring metric uses head phrase matching, which means that X matches Y if X is the rightmost substring of Y. For example, "New York" will match against "downtown New York" and "the financially stable New York". We stripped each NP of articles, common adjectives, and numbers before matching it with other NPs and saving it to the lexicon. We also used a small stopword list and a number recognizer to discard general terms such as pronouns and numbers.

We scored each extraction pattern with the *RlogF* metric previously used by AutoSlog-TS [Riloff, 1996b]. The score is computed as:

$$score(pattern_i) = R_i * log_2(F_i)$$

where F_i is the number of unique lexicon entries extracted by $pattern_i$, N_i is the total number of unique NPs extracted by $pattern_i$, and $R_i = \frac{F_i}{N_i}$. This metric was designed for information extraction tasks, where it is important to identify not only the most reliable extraction patterns but also patterns that will frequently extract relevant information (even if irrelevant information also will be extracted). For example, the pattern "kidnapped in <x>" will extract locations but it will also extract dates (e.g., "kidnapped in January"). The fact that it frequently extracts locations makes it essential to have in the dictionary, even if dates will also be extracted. The *RlogF* metric tries to strike a balance between reliability and frequency: R is high when the pattern's extractions are highly correlated with the semantic category, and F is high when the pattern extracts a large number of category members.

The mutual bootstrapping algorithm works quite well, producing good extraction patterns and lexicon entries. But the bootstrapping process can be led astray by extraction patterns that have an affinity for more than one type of semantic category. For example, suppose the pattern "shot in <x>" is one of the first extraction patterns learned for the location category. This pattern frequently extracts both locations and body parts (e.g., "shot in the back" or "shot in the arm"). All of its extractions are assumed to be locations so many references to body parts will be incorrectly added to the location lexicon. In the next iteration, extraction patterns will be rewarded for extracting references to body parts and the bootstrapping process can get derailed. To make the algorithm more robust, we introduce a second level of bootstrapping that retains only the most reliable lexicon entries learned by mutual bootstrapping and then restarts the process all over again.

Note that all dictionary entries and extraction patterns in the lexicons are considered to be positive examples, and all those outside the lexicons are considered to be negative examples. This means that high confidence is implicitly held in those positive examples (since at each iteration we may relabel negative examples as positive ones, but not vice versa). Since the bootstrapping alternates between two types of classifiers the complementary nature of the information they use can dissipate the effects of noise in the labeled examples so far. The way we quantify our confidence in each labeled example by its accuracy in the alternate classifier reflects the high-accuracy low-coverage bias of this approach to bootstrapping.

3.2 Multi-level Bootstrapping

On top of the mutual bootstrapping procedure, we introduce a second level of bootstrapping which we will call *meta-bootstrapping*. The outer bootstrapping mechanism (meta-bootstrapping) compiles the results from the inner bootstrapping process (mutual bootstrapping) and identifies the five most reliable lexicon entries. These five NPs are retained for the permanent semantic lexicon and the rest of the mutual bootstrapping process is discarded. The entire mutual bootstrapping process is then restarted from scratch. Figure 2 shows the meta-bootstrapping process.

To determine which NPs are most "reliable", we score each NP based on the number of unique patterns that extracted it. Intuitively, a noun phrase extracted by three different extractions patterns is more likely to belong to

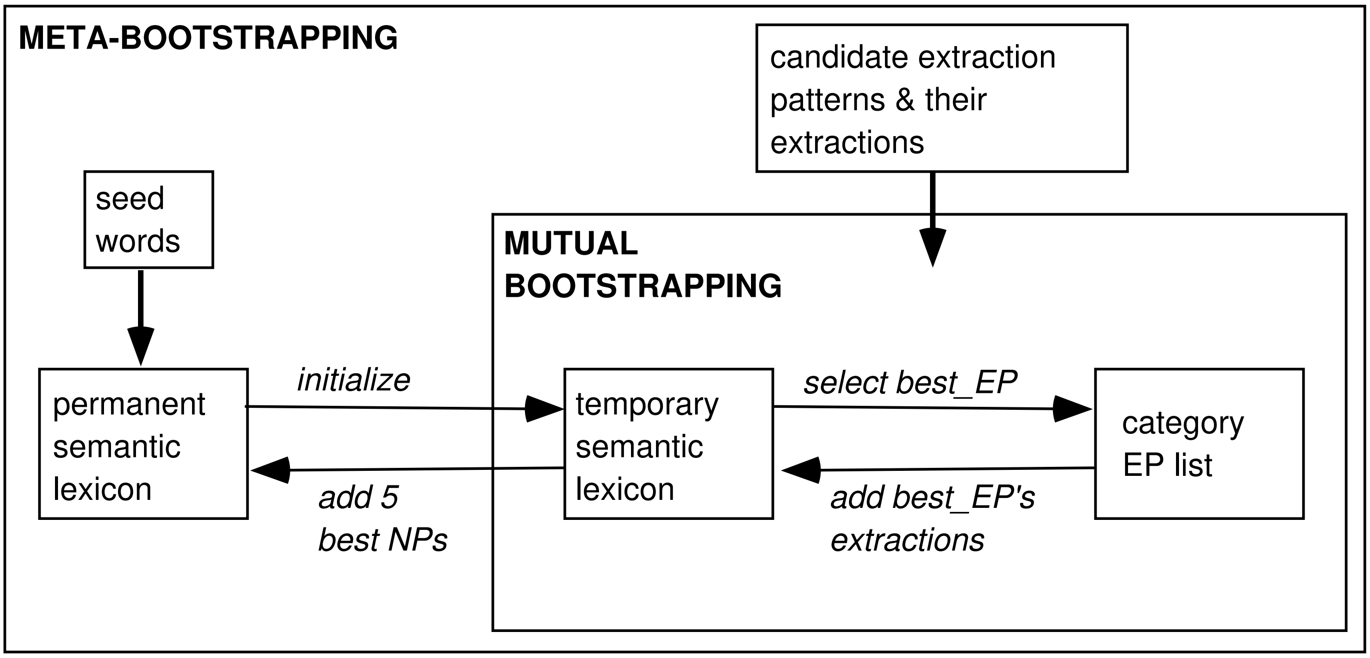

Figure 2: The Meta-Bootstrapping Process

the category than a noun phrase extracted by a single pattern. For tie-breaking purposes, we also add in a small factor that represents the strength of the patterns that extracted it. The scoring metric for noun phrases is shown below, where N_i is the number of unique patterns that extracted NP_i.

$$score(NP_i) = \sum_{k=1}^{N_i} 1 + (.01 * score(pattern_k))$$

The main advantage of meta-bootstrapping comes from re-evaluating the extraction patterns after each mutual bootstrapping process. For example, after the first mutual bootstrapping run, five new words are added to the permanent semantic lexicon. Then mutual bootstrapping is restarted from scratch with the original seed words plus these five new words. Now, the best pattern selected by mutual bootstrapping might be different from the best pattern selected last time. This produces a snowball effect because its extractions are added to the temporary semantic lexicon which is the basis for choosing the next extraction pattern. In practice, what happens is that the ordering of the patterns changes (sometimes dramatically) between subsequent runs of mutual bootstrapping. In particular, more general patterns seem to float to the top as the permanent semantic lexicon grows.

3.3 Evaluation

We evaluated our bootstrapping algorithm by generating dictionaries for locations using two text collections: corporate web pages used in the WebKB project [Craven *et al.*, 1998] and terrorism news articles from the MUC-4 corpus [MUC-4 Proceedings, 1992]. For training, we used 4160 of the web pages and 1500 of the terrorism texts. We preprocessed the web pages first by removing HTML tags and adding periods to separate independent phrases.[3] AutoSlog generated 19,690 candidate extraction patterns from the web page training set, and 14,064 candidate extraction patterns from the terrorism training set.[4] The seed word lists that we used are shown in Figure 3. We used different location seeds for the two text collections because the terrorism articles were mainly from Latin America while the web pages were much more international. The seed words must be frequent in the training texts for the bootstrapping algorithm to work well.

Web Location:	*australia canada china england france germany japan mexico switzerland united_states*
Terr. Location:	*bolivia city colombia district guatemala honduras neighborhood nicaragua region town*

Figure 3: Seed Word Lists

We ran the meta-bootstrapping algorithm (outer bootstrapping) for 50 iterations. The extraction patterns produced by the last iteration were the output of

[3] Web pages pose a problem for NLP systems because separate lines do not always end with a period (e.g., list items and headers). We used several heuristics to insert periods whenever an independent line or phrase was suspected.

[4] AutoSlog actually generated many more extraction patterns, but for practical reasons we only used the patterns that appeared with frequency ≥ 2.

	Iter 1	*Iter 10*	*Iter 20*	*Iter 30*	*Iter 40*	*Iter 50*
Web Location	5/5 (1)	46/50 (.92)	88/100 (.88)	129/150 (.86)	163/200 (.82)	191/250 (.76)
Terr. Location	5/5 (1)	32/50 (.64)	66/100 (.66)	100/150 (.67)	127/200 (.64)	158/250 (.63)

Table 1: Accuracy of the Semantic Lexicons

Web Location Patterns	*Terrorism Location Patterns*
offices in <x>	living in <x>
facilities in <x>	traveled to <x>
operations in <x>	become in <x>
loans in <x>	sought in <x>
operates in <x>	presidents of <x>
locations in <x>	parts of <x>
producer in <x>	to enter <x>
states of <x>	condemned in <x>
seminars in <x>	relations between <x>
activities in <x>	ministers of <x>
consulting in <x>	part in <x>
countries of <x>	taken in <x>
rep. of <x>	returned to <x>
outlets in <x>	process in <x>
consulting in <x>	involvement in <x>
customers in <x>	intervention in <x>
diensten in <x>	linked in <x>
distributors in <x>	operates in <x>
services in <x>	kidnapped in <x>
expanded into <x>	refuge in <x>
partners in <x>	democracy in <x>
located in <x>	prevailing in <x>
plant in <x>	billion in <x>
dienste in <x>	groups in <x>
packages in <x>	wave in <x>
shipyards in <x>	outskirts of <x>
support in <x>	prevails in <x>
dedicated in <x>	percent in <x>
testing in <x>	stay in <x>
manager in <x>	heard throughout

Figure 4: Top 25 extraction patterns for locations

the system, along with the permanent semantic lexicon. For each meta-bootstrapping iteration, we ran the mutual bootstrapping procedure (inner bootstrapping) until it produced 10 patterns that extracted at least one new NP (i.e., not currently in the semantic lexicon). But there were two exceptions: (1) if the best pattern had score < 0.7 then mutual bootstrapping stopped, or (2) if the best pattern had score > 1.8 then mutual bootstrapping continued.

Figure 4 shows the top 20 extraction patterns produced by meta-bootstrapping after 50 iterations. Most of these extraction patterns are clearly useful patterns for extracting noun phrases that represent locations. It is also interesting to note that the location patterns generated for the web pages are very different from the location patterns generated for the terrorism articles. The two text collections contain very different vocabulary, and the locations mentioned in the texts are very different too. The web pages are widely international and mention many countries and U.S. cities, while the terrorism articles are mostly from Latin America and focus on cities and towns therein. One of the strengths of a corpus-based algorithm is that it learns dictionary entries that are most important for the texts represented in the training corpus.

To evaluate the quality of the semantic lexicons, we manually inspected each word. We judged a word to belong to the category if it was a specific category member (e.g., "Japan" is a specific location) or a general referent for the category (e.g., "the area" is a referent for locations). Although referents are meaningless in isolation, they are useful for information extraction tasks because a coreference resolver should be able to find their antecedent. The referents were also very useful during bootstrapping because a pattern that extracts "the area" will probably also extract specific locations.

Table 1 shows the accuracy of the semantic lexicons after the 1st iteration of meta-bootstrapping and after each 10th iteration. Each cell shows the number of true category members among the entries generated thus far. For example, 50 phrases were added to the web location lexicon after the tenth iteration and 46 of those (92%) were true location phrases. Table 1 shows that meta-bootstrapping identified 191 location phrases in the web pages and 158 location phrases in the terrorism articles. The density of good phrases was also quite high. Even after 50 iterations, 76% of the hypothesized location phrases on the web pages were true locations, and 63% of the hypothesized locations phrases in the terrorism articles were true locations.

In summary, multi-level bootstrapping can produce high-quality extraction patterns and semantic lexicon entries using only raw texts and a small set of seed words as input. Our bootstrapping approach has two advantages over previous techniques for learning information extraction dictionaries: both a semantic lexicon and a dictionary of extraction patterns are acquired simultaneously, and no special training resources are needed.

4 Bootstrapping a Text Classifier

In the previous section, we showed how an application of the bootstrapping framework successfully created an information extractor. Now, we turn to the problem of text classification. A variety of text classification algorithms, such as naive Bayes [Lewis, 1998], support vector machines [Joachims, 1998], k-nearest neighbor [Yang, 1999], rule-learning algorithms [Cohen and Singer, 1996], and Boosting [Schapire and Singer, 1999] have proven adept at text classification when provided with large amounts of labeled training data. In recent work [Nigam *et al.*, 1999], we have shown that text classification error can be

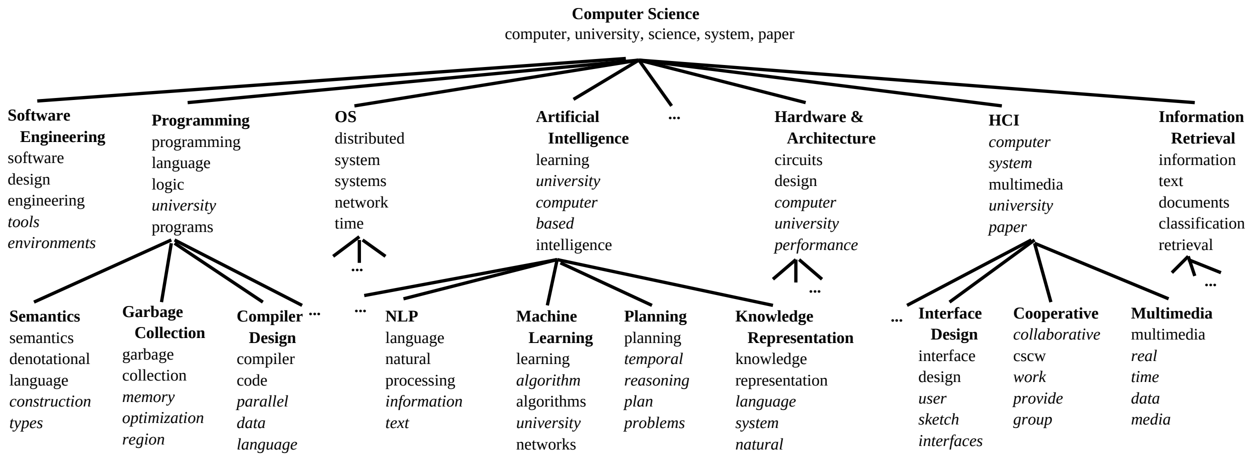

Figure 5: A subset of Cora's topic hierarchy. Each node contains its title, and the five most probable words, as calculated by naive Bayes and shrinkage with vertical word redistribution [Hofmann & Puzicha, 1998]. Words among the initial keywords for that class are indicated in plain font; others are in italics.

reduced by up to 30% when a small amount of labeled documents is augmented with a large collection of unlabeled documents. Here, we extend this approach and replace the task information given the labeled documents with seed information in another form.

In this study, knowledge about the classes of interest is provided in the form of a few keywords per class and a class hierarchy. Keywords are typically generated more quickly and easily than even a small number of labeled documents. Many classification problems naturally come with hierarchically-organized classes. Our algorithm proceeds by using the keywords to generate preliminary labels for some documents by term-matching. Then these labels, the hierarchy and all the unlabeled documents become the input to a bootstrapping algorithm that generates a naive Bayes classifier.

The bootstrapping algorithm used here combines hierarchical shrinkage and Expectation-Maximization (EM) with unlabeled data. EM is an iterative algorithm for maximum likelihood estimation in parametric estimation problems with missing data. In our scenario, the class labels of the documents are treated as missing data. Here, EM works by first training a classifier with only the documents preliminarily-labeled by the keywords, and then uses the classifier to re-assign probabilistically-weighted class labels to all the documents by calculating the expectation of the missing class labels. It then trains a new classifier using all the documents and iterates.

We further improve the classifier by also incorporating *shrinkage*, a statistical technique for improving parameter estimation in the face of sparse data. When classes are provided in a hierarchical relationship, shrinkage is used to estimate new parameters by using a weighted average of the specific (but unreliable) local class estimates and the more general (but also more reliable) ancestors of the class in the hierarchy. The optimal weights in the average are calculated by an EM process that runs simultaneously with the EM that is re-estimating the class labels.

Experimental results show that the bootstrapping algorithm described here uses the unlabeled data, keywords, and class hierarchy to output a classifier that is close to human agreement levels. The experimental domain in this paper is topic identification of computer science research papers. This domain originates as part of the *Ra* research project, an effort to build domain-specific search engines on the Web with machine learning techniques. Our demonstration system, Cora, is a search engine over computer science research papers [McCallum *et al.*, 1999]. The bootstrapping classification algorithm described in this paper is used in Cora to place research papers into a Yahoo-like hierarchy of the field of computer science. The search engine, with the hierarchy, is publicly available at *www.cora.justresearch.com*.

4.1 Generating Preliminary Labels with Keywords

The first step in the bootstrapping process is to use the keywords to generate preliminary labels for as many of the unlabeled documents as possible. Each class is given just a few keywords. Figure 5 shows examples of the number and type of keywords given in our experimental domain—the human-provided keywords are shown in the nodes in non-italic font.

In this paper, we generate preliminary labels from the keywords by term-matching in a rule-list fashion: for each document, we step through the keywords and place the document in the category of the first keyword that matches. If an extensive keyword list were carefully chosen, this method could be reasonably accurate. However, finding enough keywords to obtain broad coverage and finding sufficiently specific keywords to obtain high accuracy is very difficult; it requires intimate knowledge of the data and a lot of trial and error.

As a result, keyword matching is both an inaccurate and incomplete classification method. Keywords tend to be high-precision and low-recall; this brittleness will leave many documents unlabeled. Some documents will incorrectly match keywords from the wrong class. In general we expect the brittleness of the keywords to be the dominating factor in overall error. In our experimental domain, for example, 59% of the unlabeled documents are left unlabeled by keyword matching.

Another method of priming bootstrapping with keywords would be to take each set of keywords as a labeled mini-document containing just a few words. This could then be used as input to any standard learning algorithm. Testing this, and other keyword labeling approaches, is an area of ongoing research.

4.2 The Bootstrapping Algorithm

The goal of the bootstrapping step is to generate a naive Bayes classifier from the inputs: the (inaccurate and incomplete) preliminary labels, the unlabeled data and the class hierarchy. One straightforward method would be to simply take the unlabeled documents with preliminary labels, and treat this as labeled data in a standard supervised setting. This approach provides only minimal benefit for three reasons: (1) the labels are rather noisy, (2) the sample of preliminarily-labeled documents is skewed from the regular document distribution (*i.e.* it includes only documents with known keywords), and (3) data is sparse in comparison to the size of the feature space. Adding the remaining unlabeled data and running EM helps counter the first and second of these reasons. Adding hierarchical shrinkage to naive Bayes helps counter the first and third of these reasons. We begin a detailed description of our bootstrapping algorithm with a short overview of standard naive Bayes text classification, then proceed to add EM to incorporate the unlabeled data, and conclude by explaining hierarchical shrinkage. An outline of the entire algorithm is presented in Table 2.

The naive Bayes framework

We use the framework of multinomial naive Bayes text classification [Lewis, 1998; McCallum and Nigam, 1998]. It is useful to think of naive Bayes as estimating the parameters of a probabilistic generative model for text documents. In this model, first the class of the document is selected. The words of the document are then generated based on the parameters for the class-specific multinomial (*i.e.* unigram model). Thus, the classifier parameterizes the class prior probabilities and the class-conditioned word probabilities. Each class, c_j, has a document frequency relative to all other classes, written $\mathrm{P}(c_j)$. For every word w_t in the vocabulary V, $\mathrm{P}(w_t|c_j)$ indicates the frequency that the classifier expects word w_t to occur in documents in class c_j.

In the standard supervised setting, learning the parameters is accomplished using a set of labeled training documents, $\mathcal{D}$. To estimate the word probability parameters, $\mathrm{P}(w_t|c_j)$, we count the frequency that word w_t occurs in all word occurrences for documents in class c_j. We supplement this with Laplace smoothing that primes each estimate with a count of one to avoid probabilities of zero. Define $N(w_t, d_i)$ to be the count of the number of times word w_t occurs in document d_i, and define $\mathrm{P}(c_j|d_i) \in \{0, 1\}$, as given by the document's class label. Then, the estimate of the probability of word w_t in class c_j is:

$$\mathrm{P}(w_t|c_j) = \frac{1 + \sum_{d_i \in \mathcal{D}} N(w_t, d_i)\mathrm{P}(c_j|d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{d_i \in \mathcal{D}} N(w_s, d_i)\mathrm{P}(c_j|d_i)}. \quad (1)$$

The class prior probability parameters are set in the same way, where $|\mathcal{C}|$ indicates the number of classes:

$$\mathrm{P}(c_j) = \frac{1 + \sum_{d_i \in \mathcal{D}} \mathrm{P}(c_j|d_i)}{|\mathcal{C}| + |\mathcal{D}|}. \quad (2)$$

Given an unlabeled document and a classifier, we determine the probability that the document belongs in class c_j using Bayes' rule and the naive Bayes assumption—that the words in a document occur independently of each other given the class. If we denote $w_{d_{i,k}}$ to be the kth word in document d_i, then classification becomes:

$$\begin{aligned} \mathrm{P}(c_j|d_i) &\propto \mathrm{P}(c_j)\mathrm{P}(d_i|c_j) \\ &\propto \mathrm{P}(c_j) \prod_{k=1}^{|d_i|} \mathrm{P}(w_{d_{i,k}}|c_j). \end{aligned} \quad (3)$$

Empirically, when given a large number of training documents, naive Bayes does a good job of classifying text documents [Lewis, 1998]. More complete presentations of naive Bayes for text classification are provided by Mitchell [1997] and McCallum and Nigam [1998].

- **Inputs:** A collection $\mathcal{D}$ of unlabeled documents, a class hierarchy, and a few keywords for each class.
- Generate preliminary labels for as many of the unlabeled documents as possible by term-matching with the keywords in a rule-list fashion.
- Initialize all the λ_j's to be uniform along each path from a leaf class to the root of the class hierarchy.
- Iterate the EM algorithm:
  - **(M-step)** Build the maximum likelihood multinomial at each node in the hierarchy given the class probability estimates for each document (Equations 1 and 2). Normalize all the λ_j's along each path from a leaf class to the root of the class hierarchy so that they sum to 1.
  - **(E-step)** Calculate the expectation of the class labels using the classifier created in the M-step (Equation 3). Increment the new λ_j's by attributing each word probabilistically to the ancestors of each class.
- **Output:** A naive Bayes classifier that takes an unlabeled document and predicts a class label.

Table 2: An outline of the bootstrapping algorithm described in Sections 4.1 and 4.2.

Adding unlabeled data with EM

In the standard supervised setting, each document comes with a label. In our bootstrapping scenario, the preliminary keyword labels are both incomplete and inaccurate—the keyword matching leaves many documents unlabeled, and labels some incorrectly. In order to use the entire data set in a naive Bayes classifier, we use the Expectation-Maximization (EM) algorithm to generate probabilistically-weighted class labels for all the documents. This results in classifier parameters that are more likely given all the data.

EM is a class of iterative algorithms for maximum likelihood estimation in problems with incomplete data [Dempster *et al.*, 1977]. Given a model of data generation, and data with some missing values, EM iteratively uses the current model to estimate the missing values, and then uses the missing value estimates to improve the model. Using all the available data, EM will locally maximize the likelihood of the parameters and give estimates for the missing values. In our scenario, the class labels of the unlabeled data are treated as the missing values.

In implementation, EM is an iterative two-step process. Initially, the parameter estimates are set in the standard naive Bayes way from just the preliminarily labeled documents. Then we iterate the E- and M-steps. The E-step calculates probabilistically-weighted class labels, $\mathrm{P}(c_j|d_i)$, for every document using the classifier and Equation 3. The M-step estimates new classifier parameters using all the documents, by Equations 1 and 2, where $\mathrm{P}(c_j|d_i)$ is now continuous, as given by the E-step. We iterate the E- and M-steps until the classifier converges. The initialization step identifies each mixture component with a class and seeds EM so that the local maxima that it finds correspond well to class definitions.

In previous work [Nigam *et al.*, 1999], we have shown this technique significantly increases text classification accuracy when given limited amounts of labeled data and large amounts of unlabeled data. The expectation here is that EM will both correct and complete the labels for the entire data set.

Improving sparse data estimates with shrinkage

Even when provided with a large pool of documents, naive Bayes parameter estimation during bootstrapping will suffer from sparse data because naive Bayes has so many $(|V||C| + |C|)$ parameters to estimate. Using the provided class hierarchy, we can integrate the statistical technique *shrinkage* into the EM algorithm to help alleviate the sparse data problem.

Consider trying to estimate the probability of the word "intelligence" in the class NLP. This word should clearly have non-negligible probability there; however, with limited training data we may be unlucky, and the observed frequency of "intelligence" in NLP may be very far from its true expected value. One level up the hierarchy, however, the Artificial Intelligence class contains many more documents (the union of all the children). There, the probability of the word "intelligence" can be more reliably estimated.

Shrinkage calculates new word probability estimates for each leaf class by a *weighted average* of the estimates on the path from the leaf to the root. The technique balances a trade-off between specificity and reliability. Estimates in the leaf are most specific but unreliable; further up the hierarchy estimates are more reliable but unspecific. We can calculate mixture weights for the averaging that are guaranteed to maximize the likelihood of the data with the EM algorithm.

More formally, let $\{\mathrm{P}^1(w_t|c_j), \ldots, \mathrm{P}^k(w_t|c_j)\}$ be word probability estimates, where $\mathrm{P}^1(w_t|c_j)$ is the estimate using training data just in the leaf, $\mathrm{P}^{k-1}(w_t|c_j)$ is the estimate at the root using all the training data, and $\mathrm{P}^k(w_t|c_j)$ is the uniform estimate ($\mathrm{P}^k(w_t|c_j) = 1/|V|$). The interpolation weights among c_j's "ancestors" (which we define to include c_j itself) are written $\{\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^k\}$, where $\sum_{i=1}^k \lambda_j^i = 1$. The new word probability estimate based on shrinkage, denoted $\check{\mathrm{P}}(w_t|c_j)$, is then

$$\check{\mathrm{P}}(w_t|c_j) = \lambda_j^1 \mathrm{P}^1(w_t|c_j) + \ldots + \lambda_j^k \mathrm{P}^k(w_t|c_j). \quad (4)$$

The λ_j vectors are calculated during EM. In the E-step of bootstrapping, for every word of training data in class c_j, we determine the expectation that each ancestor was responsible for generating it. In the M-step, we normalize the sum of these expectations to obtain new mixture weights λ_j.

One can think of hierarchical shrinkage as defining a different generative model than the one in Section 4.2. As before, a class is selected first. Then, for each word position in the document, an ancestor of the class (including itself) is selected according to the averaging weights. Then, the word itself is chosen based on the parameters of that ancestor. For every possible generative model of this type, there is a corresponding model of the simple naive Bayes type that yields the same document distribution. It is created by "flattening" the hierarchy according to the weights. A more complete description of hierarchical shrinkage for text classification is presented by McCallum *et al.* [1998].

4.3 Experimental Results

In this section, we provide empirical evidence that bootstrapping a text classifier from unlabeled data can produce a high-accuracy text classifier. As a test domain, we use computer science research papers. We have created a 70-leaf hierarchy of computer science topics, part of which is shown in Figure 5. Creating the hierarchy took about 60 minutes, during which we examined conference proceedings, and explored computer science sites on the Web. Selecting a few keywords associated with each node took about 90 minutes. A test set was created by expert hand-labeling of a random sample of 625 research papers from the 30,682 papers in the Cora archive at the time we began these experiments. Of these, 225 (about one-third) did not fit into any category, and were

Method	# Lab	# P-Lab	# Unlab	Acc
Keyword	—	—	—	45%
NB	100	—	—	30%
NB	399	—	—	47%
NB+EM+S	—	12,657	18,025	66%
NB	—	12,657	—	47%
NB+S	—	12,657	—	63%
Human	—	—	—	72%

Table 3: Classification results with different techniques: keyword matching, human agreement, naive Bayes (NB), and naive Bayes combined with hierarchical shrinkage (S), and EM. The classification accuracy (Acc), and the number of labeled (Lab), keyword-matched preliminarily-labeled (P-Lab), and unlabeled (Unlab) documents used by each method are shown.

discarded—resulting in a 400 document test set. Labeling these 400 documents took about six hours. Some of these papers were outside the area of computer science (*e.g.* astrophysics papers), but most of these were papers that with a more complete hierarchy would be considered computer science papers. The class frequencies of the data are not too skewed; on the test set, the most populous class accounted for only 7% of the documents.

Each research paper is represented as the words of the title, author, institution, references, and abstract. A detailed description of how these segments are automatically extracted is provided elsewhere [McCallum *et al.*, 1999; Seymore *et al.*, 1999]. Words occurring in fewer than five documents and words on a standard stoplist were discarded. No stemming was used. Bootstrapping was performed using the algorithm outlined in Table 2.

Table 3 shows classification results with different classification techniques used. The rule-list classifier based on the keywords alone provides 45%. (The 43% of documents in the test set containing no keywords cannot be assigned a class by the rule-list classifier, and are counted as incorrect.) As an interesting time comparison, about 100 documents could have been labeled in the time it took to generate the keyword lists. Naive Bayes accuracy with 100 labeled documents is only 30%. With 399 labeled documents (using our test set in a leave-one-out-fashion), naive Bayes reaches 47%. When running the bootstrapping algorithm, 12,657 documents are given preliminary labels by keyword matching. EM and shrinkage incorporate the remaining 18,025 documents, "fix" the preliminary labels and leverage the hierarchy; the resulting accuracy is 66%. As an interesting comparison, agreement on the test set between two human experts was 72%.

A few further experiments reveal some of the inner-workings of bootstrapping. If we build a naive Bayes classifier in the standard supervised way from the 12,657 preliminarily labeled documents the classifier gets 47% accuracy. This corresponds to the performance for the first iteration of bootstrapping. Note that this matches the accuracy of traditional naive Bayes with 399 labeled training documents, but that it requires less than a quarter the human labeling effort. If we run bootstrapping without the 18,025 documents left unlabeled by keyword matching, accuracy reaches 63%. This indicates that shrinkage and EM on the preliminarily labeled documents is providing substantially more benefit than the remaining unlabeled documents.

One explanation for the small impact of the 18,025 documents left unlabeled by keyword matching is that many of these do not fall naturally into the hierarchy. Remember that about one-third of the 30,000 documents fall outside the hierarchy. Most of these will not be given preliminary labels by keyword matching. The presence of these outlier documents skews EM parameter estimation. A more inclusive computer science hierarchy would allow the unlabeled documents to benefit classification more.

However, even without a complete hierarchy, we could use these documents if we could identify these outliers. Some techniques for robust estimation with EM are discussed by McLachlan and Basford [1988]. One specific technique for these text hierarchies is to add extra leaf nodes containing uniform word distributions to each interior node of the hierarchy in order to capture documents not belonging in any of the predefined topic leaves. This should allow EM to perform well even when a large percentage of the documents do not fall into the given classification hierarchy. A similar approach is also planned for research in topic detection and tracking (TDT) [Baker *et al.*, 1999]. Experimentation with these techniques is an area of ongoing research.

5 Related Work

Other research efforts in text learning have also used bootstrapping approaches. Brin [1998] uses a bootstrapping approach on the World Wide Web to extract book title and author pairs. From a small set of seeded books, his DIPRE algorithm searches the Web for known pairs and learns new patterns that are common for these pairs. Then, these new patterns are used to identify new books, in an iterative fashion.

The preliminary labeling by keywords used in this paper is similar to the seed collocations used by Yarowsky [1995]. There, in a word sense disambiguation task, a bootstrapping algorithm is seeded with some examples of common collocations with the particular sense of some word (*e.g.* the seed "life" for the biological sense of "plant").

The co-training algorithm [Blum and Mitchell, 1998] for classification works in cases where the feature space is separable into naturally redundant and independent parts. For example, web pages can be thought of as the text on the web page, and the collection of text in hyperlink anchors to that page.

Unlabeled data has also been used in document classification by Nigam *et al.* [1999], where, as in this work, EM is used to fill in the "missing" class labels on the unlabeled data. Unlike this work, Nigam *et al.* begin with a few labeled documents instead of keywords. More importantly, they also describe two methods of overcoming problems when the results of maximum likelihood clustering do not correspond well to class labels.

6 Discussion

The two problems studied in this paper work at very different granularities. In the document classification domain, our granularity of interest is the entire document, and our feature set consists of potentially every word in the vocabulary. This means that a single keyword can match one of several hundred words in a document, and is chosen from one of tens of thousands of words in the vocabulary. We can view the seeding using these keywords as finding better than random starting points, which we then exploit using Expectation-Maximization (EM) and shrinkage over the class hierarchy.

In the information extraction task, the granularity of interest is at the phrase-level. A phrase contains relatively few words, so a match with a keyword is a much stronger indicator of class membership. The feature set we use consists of all possible phrases with exact match, so our vocabulary is large, but the number of possible features for each example is very small. In this situation, we can view the seeding from keywords as providing a small set of highly accurately labeled data to begin bootstrapping.

Thus, the effect of initial labels is likely to be different for the two tasks; each example in the extraction task contains only a few words, whereas in classification an example typically contains several hundred words. Thus, we expect the significance of individual keywords to be lower in the document setting than in the extraction setting, but coverage to be higher.

In future work we propose to examine the effect of the domain and task on the applicability of the seeded bootstrapping framework. Pure unsupervised clustering [Duda and Hart, 1973] can be used to find underlying regularities in the data domain. If there is a strong correspondence between these regularities and our target task, there may be no need for seeds, except to speed up convergence by providing a good starting point for the clustering. However, if our dataset could have multiple solutions when unsupervised clustering is performed, and only a subset of these correspond to partitions that perform our task well, seeding can help us to initialize our clustering with a point close to the target local maximum. We hypothesize that many local maxima exist in text learning tasks, which makes the seeding essential.

We will test this hypothesis in two ways. We will perform unsupervised clustering on the datasets (*e.g.* by providing random initial seeds) then labeling the clusters post-facto with the seeds. If the clusters produced this way perform as well as those produced using seeding, then our approach finds us the global maximum for our task. If this is not the case (as we expect) we will perform experiments on providing seedwords of successively poorer quality (*i.e.* seedwords which are more and more ambiguous or loosely related to the target class) and examine how the quality of the initial labeling using those seedwords affects our final task-specific results.

This paper has considered text learning in cases without labeled training data. To this end, we advocate using the general bootstrapping process. From a small amount of seed information and a large collection of unlabeled data, bootstrapping iterates the steps of building a model from generated labels, and obtaining labeled data from the model. In two case studies in information extraction and text classification, the bootstrapping paradigm proves successful at learning without labeled data.

Acknowledgements

This research is supported in part by the National Science Foundation under grants IRI-9509820, IRI-9704240, and SBR-9720374 and the DARPA HPKB program under contract F30602-97-1-0215.

References

[Baker *et al.*, 1999] D. Baker, T. Hofmann, A. McCallum, and Y. Yang. A hierarchical probabilistic model for novelty detection in text. Technical report, Just Research, 1999. http://www.cs.cmu.edu/~mccallum.

[Blum and Mitchell, 1998] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT '98*, 1998.

[Brin, 1998] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at EDBT '98*, 1998.

[Califf, 1998] M. E. Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, Tech. Rept. AI98-276, Artificial Intelligence Laboratory, The University of Texas at Austin, 1998.

[Castelli and Cover, 1996] V. Castelli and T. M. Cover. The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory*, 42(6):2101–2117, November 1996.

[Cohen and Singer, 1996] W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In *SIGIR '96*, 1996.

[Cohen, 1996] W. W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.

[Craven *et al.*, 1998] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[Duda and Hart, 1973] R. Duda and P. Hart. *Pattern classification and scene analysis*. Wiley, 1973.

[Freitag, 1998] Dayne Freitag. Multistrategy Learning for Information Extraction. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[Huffman, 1996] S. Huffman. Learning information extraction patterns from examples. In Stefan Wermter, Ellen Riloff, and Gabriele Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 246–260. Springer-Verlag, Berlin, 1996.

[Joachims, 1998] T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In *ECML-98*, 1998.

[Kim and Moldovan, 1993] J. Kim and D. Moldovan. Acquisition of Semantic Patterns for Information Extraction from Corpora. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications*, pages 171–176, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[Lewis and Ringuette, 1994] David D. Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.

[Lewis, 1998] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *ECML-98*, 1998.

[McCallum and Nigam, 1998] A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998. Tech. rep. WS-98-05, AAAI Press. http://www.cs.cmu.edu/~mccallum.

[McCallum *et al.*, 1998] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text clasification by shrinkage in a hierarchy of classes. In *ICML-98*, 1998.

[McCallum *et al.*, 1999] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Using machine learning techniques to build domain-specific search engines. In *IJCAI-99*, 1999. To appear.

[McLachlan and Basford, 1988] G.J. McLachlan and K.E. Basford. *Mixture Models*. Marcel Dekker, New York, 1988.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[MUC-4 Proceedings, 1992] MUC-4 Proceedings. *Proceedings of the Fourth Message Understanding Conference (MUC-4)*. Morgan Kaufmann, San Mateo, CA, 1992.

[MUC-5 Proceedings, 1993] MUC-5 Proceedings. *Proceedings of the Fifth Message Understanding Conference (MUC-5)*. Morgan Kaufmann, San Francisco, CA, 1993.

[Nigam *et al.*, 1999] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 1999. To appear.

[Riloff and Jones, 1999] E. Riloff and R. Jones. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 1044–1049. The AAAI Press/MIT Press, 1999. (to appear).

[Riloff, 1993] E. Riloff. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816. AAAI Press/The MIT Press, 1993.

[Riloff, 1996a] E. Riloff. An Empirical Study of Automated Dictionary Construction for Information Extraction in Three Domains. *Artificial Intelligence*, 85:101–134, 1996.

[Riloff, 1996b] E. Riloff. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044–1049. The AAAI Press/MIT Press, 1996.

[Schapire and Singer, 1999] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning Journal*, 1999. To appear.

[Seymore *et al.*, 1999] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999. In submission.

[Soderland *et al.*, 1995] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1314–1319, 1995.

[Soderland, 1999] Stephen Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Machine Learning*, 1999. (to appear).

[Yang, 1999] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1999. To appear.

[Yarowsky, 1995] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL-95*, 1995.