Unsupervised Models for Named Entity Classification

Michael Collins and **Yoram Singer**
AT&T Labs–Research,
180 Park Avenue, Florham Park, NJ 07932
{mcollins,singer}@research.att.com

Abstract

This paper discusses the use of unlabeled examples for the problem of named entity classification. A large number of rules is needed for coverage of the domain, suggesting that a fairly large number of labeled examples should be required to train a classifier. However, we show that the use of *unlabeled* data can reduce the requirements for supervision to just 7 simple "seed" rules. The approach gains leverage from natural redundancy in the data: for many named-entity instances both the spelling of the name and the context in which it appears are sufficient to determine its type.

We present two algorithms. The first method uses a similar algorithm to that of (Yarowsky 95), with modifications motivated by (Blum and Mitchell 98). The second algorithm extends ideas from boosting algorithms, designed for supervised learning tasks, to the framework suggested by (Blum and Mitchell 98).

1 Introduction

Many statistical or machine-learning approaches for natural language problems require a relatively large amount of supervision, in the form of labeled training examples. Recent results (e.g., (Yarowsky 95; Brill 95; Blum and Mitchell 98)) have suggested that unlabeled data can be used quite profitably in reducing the need for supervision. This paper discusses the use of unlabeled examples for the problem of named entity classification.

The task is to learn a function from an input string (proper name) to its type, which we will assume to be one of the categories `Person`, `Organization`, or `Location`. For example, a good classifier would identify *Mrs. Frank* as a person, *Steptoe & Johnson* as a company, and *Honduras* as a location. The approach uses both *spelling* and *contextual* rules. A spelling rule might be a simple look-up for the string (e.g., a rule that *Honduras* is a location) or a rule that looks at words within a string (e.g., a rule that any string containing *Mr.* is a person). A contextual rule considers words surrounding the string in the sentence in which it appears (e.g., a rule that any proper name modified by an appositive whose head is *president* is a person). The task can be considered to be one component of the MUC (MUC-6, 1995) named entity task (the other task is that of segmentation, i.e., pulling possible people, places and locations from text before sending them to the classifier). Supervised methods have been applied quite successfully to the full MUC named-entity task (Bikel et al. 97).

At first glance, the problem seems quite complex: a large number of rules is needed to cover the domain, suggesting that a large number of labeled examples is required to train an accurate classifier. But we will show that the use of unlabeled data can drastically reduce the need for supervision. Given around 90,000 unlabeled examples, the methods described in this paper classify names with over 91% accuracy. The only supervision is in the form of 7 seed rules (namely, that *New York, California* and *U.S.* are locations; that any name containing *Mr.* is a person; that any name containing *Incorporated* is an organization; and that *I.B.M.* and *Microsoft* are organizations).

The key to the methods we describe is redundancy in the unlabeled data. In many cases, inspection of either the spelling or context alone is sufficient to classify an example. For example, in

.., says **Mr. Cooper**, a vice president of ..

both a spelling feature (that the string contains *Mr.*) and a contextual feature (that *president* modifies the string) are strong indications that **Mr. Cooper** is of type `Person`. Even if an example like this is not labeled, it can be interpreted as a "hint" that *Mr.* and *president* imply the same category. The unlabeled data gives many such "hints" that two features should predict the same label, and these hints turn out to be surprisingly useful when building a classifier.

We present two algorithms. The first method builds on results from (Yarowsky 95) and (Blum and

Mitchell 98). (Yarowsky 95) describes an algorithm for word-sense disambiguation that exploits redundancy in contextual features, and gives impressive performance. Unfortunately, Yarowsky's method is not well understood from a theoretical viewpoint: we would like to formalize the notion of redundancy in unlabeled data, and set up the learning task as optimization of some appropriate objective function. (Blum and Mitchell 98) offer a promising formulation of redundancy, also prove some results about how the use of unlabeled examples can help classification, and suggest an objective function when training with unlabeled examples. Our first algorithm is similar to Yarowsky's, but with some important modifications motivated by (Blum and Mitchell 98). The algorithm can be viewed as heuristically optimizing an objective function suggested by (Blum and Mitchell 98); empirically it is shown to be quite successful in optimizing this criterion.

The second algorithm builds on a boosting algorithm called AdaBoost (Freund and Schapire 97; Schapire and Singer 98). The AdaBoost algorithm was developed for supervised learning. AdaBoost finds a weighted combination of simple (weak) classifiers, where the weights are chosen to minimize a function that bounds the classification error on a set of training examples. Roughly speaking, the new algorithm presented in this paper performs a similar search, but instead minimizes a bound on the number of (unlabeled) examples on which two classifiers disagree. The algorithm builds two classifiers iteratively: each iteration involves minimization of a continuously differential function which bounds the number of examples on which the two classifiers disagree.

1.1 Additional Related Work

There has been additional recent work on inducing lexicons or other knowledge sources from large corpora. (Brin 98) describes a system for extracting *(author, book-title)* pairs from the World Wide Web using an approach that bootstraps from an initial seed set of examples. (Berland and Charniak 99) describe a method for extracting parts of objects from wholes (e.g., "speedometer" from "car") from a large corpus using hand-crafted patterns. (Hearst 92) describes a method for extracting hyponyms from a corpus (pairs of words in "isa" relations). (Riloff and Shepherd 97) describe a bootstrapping approach for acquiring nouns in particular categories (such as "vehicle" or "weapon" categories). The approach builds from an initial seed set for a category, and is quite similar to the decision list approach described in (Yarowsky 95). More recently, (Riloff and Jones 99) describe a method they term "mutual bootstrapping" for simultaneously constructing a lexicon and contextual extraction patterns. The method shares some characteristics of the decision list algorithm presented in this paper. (Riloff and Jones 99) was brought to our attention as we were preparing the final version of this paper.

2 The Problem

2.1 The Data

971,746 sentences of New York Times text were parsed using the parser of (Collins 96).[1] Word sequences that met the following criteria were then extracted as named entity examples:

• The word sequence was a sequence of consecutive proper nouns (words tagged as NNP or NNPS) within a noun phrase, and whose last word was head of the noun phrase.

• The NP containing the word sequence appeared in one of two contexts:

1. There was an appositive modifier to the NP, whose head is a singular noun (tagged NN). For example, take

> ..., says Maury Cooper, a vice president at S.&P.

In this case, *Maury Cooper* is extracted. It is a sequence of proper nouns within an NP; its last word *Cooper* is the head of the NP; and the NP has an appositive modifier (*a vice president at S.&P.*) whose head is a singular noun (*president*).

2. The NP is a complement to a preposition, which is the head of a PP. This PP modifies another NP, whose head is a singular noun. For example,

> ... fraud related to work on a federally funded sewage plant in Georgia

In this case, *Georgia* is extracted: the NP containing it is a complement to the preposition *in*; the PP headed by *in* modifies the NP *a federally funded sewage plant*, whose head is the singular noun *plant*.

In addition to the named-entity string (*Maury Cooper* or *Georgia*), a contextual predictor was also extracted. In the appositive case, the contextual

[1]Thanks to Ciprian Chelba for running the parser and providing the data.

predictor was the head of the modifying appositive (*president* in the *Maury Cooper* example); in the second case, the contextual predictor was the preposition together with the noun it modifies (*plant_in* in the *Georgia* example). From here on we will refer to the named-entity string itself as the *spelling* of the entity, and the contextual predicate as the *context*.

2.2 Feature Extraction

Having found (*spelling*, *context*) pairs in the parsed data, a number of features are extracted. The features are used to represent each example for the learning algorithm. In principle a feature could be an arbitrary predicate of the (*spelling*, *context*) pair; for reasons that will become clear, features are limited to querying either the *spelling* or *context* alone. The following features were used:

full-string=x The full string (e.g., for *Maury Cooper*, `full-string=Maury_Cooper`).

contains(x) If the spelling contains more than one word, this feature applies for any words that the string contains (e.g., *Maury Cooper* contributes two such features, `contains(Maury)` and `contains(Cooper)`.

allcap1 This feature appears if the spelling is a single word which is all capitals (e.g., `IBM` would contribute this feature).

allcap2 This feature appears if the spelling is a single word which is all capitals or full periods, and contains at least one period. (e.g., `N.Y.` would contribute this feature, `IBM` would not).

nonalpha=x Appears if the spelling contains any characters other than upper or lower case letters. In this case `nonalpha` is the string formed by removing all upper/lower case letters from the spelling (e.g., for *Thomas E. Petry* `nonalpha=.`, for *A.T.&T.* `nonalpha=..&.`).

context=x The *context* for the entity. The *Maury Cooper* and *Georgia* examples would contribute `context=president` and `context=plant_in` respectively.

context-type=x `context-type=appos` in the appositive case, `context-type=prep` in the PP case.

Table 1 gives some examples of entities and their features.

3 Unsupervised Algorithms based on Decision Lists

3.1 Supervised Decision List Learning

The first unsupervised algorithm we describe is based on the decision list method from (Yarowsky 95). Before describing the unsupervised case we first describe the supervised version of the algorithm:

Input to the learning algorithm: n labeled examples of the form $(\mathbf{x}_i, y_i)$. y_i is the label of the ith example (given that there are k possible labels, y_i is a member of $\mathcal{Y} = \{1 \ldots k\}$). $\mathbf{x}_i$ is a set of m_i features $\{x_{i1}, x_{i2} \ldots x_{im_i}\}$ associated with the ith example. Each x_{ij} is a member of $\mathcal{X}$, where $\mathcal{X}$ is a set of possible features.

Output of the learning algorithm: a function $h : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$ where $h(x, y)$ is an estimate of the conditional probability $p(y|x)$ of seeing label y given that feature x is present. Alternatively, h can be thought of as defining a decision list of rules $x \rightarrow y$ ranked by their "strength" $h(x, y)$.

The label for a test example with features $\mathbf{x}$ is then defined as

$$f(\mathbf{x}) = \arg \max_{x \in \mathbf{x}, y \in \mathcal{Y}} h(x, y) \qquad (1)$$

In this paper we define $h(x, y)$ as the following function of counts seen in training data:

$$h(x, y) = \frac{Count(x, y) + \alpha}{Count(x) + k\alpha} \qquad (2)$$

$Count(x, y)$ is the number of times feature x is seen with label y in training data, $Count(x) = \sum_{y \in \mathcal{Y}} Count(x, y)$. α is a smoothing parameter, and k is the number of possible labels. In this paper $k = 3$ (the three labels are `person`, `organization`, `location`), and we set $\alpha = 0.1$. Equation 2 is an estimate of the conditional probability of the label given the feature, $P(y|x)$. [2]

3.2 An Unsupervised Algorithm

We now introduce a new algorithm for learning from unlabeled examples, which we will call **DL-CoTrain** (**DL** stands for decision list, the term **Cotrain** is taken from (Blum and Mitchell 98)). The

[2] (Yarowsky 95) describes the use of more sophisticated smoothing methods. It's not clear how to apply these methods in the unsupervised case, as they required cross-validation techniques: for this reason we use the simpler smoothing method shown here.

Sentence	Entities (Spelling/Context)	Features
But Robert Jordan, a partner at Steptoe & Johnson who took ...	Robert Jordan/partner	full-string=Robert_Jordan contains(Robert) contains(Jordan) context=partner context-type=appos
	Steptoe & Johnson/partner_at	full-string=Steptoe_&_Johnson contains(Steptoe) contains(&) contains(Johnson) nonalpha=& context=partner_at context-type=prep
By hiring a company like A.T.&T. ...	A.T.&T./company_like	full-string=A.T.&T. allcap2 nonalpha=..&. context=company_like context-type=prep
Hanson acquired Kidde Incorporated, parent of Kidde Credit, for ...	Kidde Incorporated/parent	full-string=Kidde_Incorporated contains(Kidde) contains(Incorporated) context=parent context-type=appos
	Kidde-Credit/parent_of	full-string=Kidde_Credit contains(Kidde) contains(Credit) context=parent_of context-type=prep

Table 1: Some example named entities and their features.

input to the unsupervised algorithm is an initial, "seed" set of rules. In the named entity domain these rules were

```
full-string=New_York        →    Location
full-string=California      →    Location
full-string=U.S.            →    Location
contains(Mr.)               →    Person
contains(Incorporated)      →    Organization
full-string=Microsoft       →    Organization
full-string=I.B.M.          →    Organization
```

Each of these rules was given a strength of 0.9999. The following algorithm was then used to induce new rules:

1. Set $n = 5$. (n is the maximum number of rules of each type induced at each iteration.)

2. **Initialization**: Set the *spelling* decision list equal to the set of seed rules.

3. Label the training set using the current set of *spelling* rules. Examples where no rule applies are left unlabeled.

4. Use the labeled examples to induce a decision list of *contextual* rules, using the method described in section 3.1.

   Let $Count'(x)$ be the number of times feature x is seen with some known label in the training data. For each label (`Person`, `Organization` and `Location`), take the n contextual rules with the highest value of $Count'(x)$ whose *unsmoothed*[3] strength is above some threshold p_{min}. (If fewer than n rules have precision greater than p_{min}, we keep only those rules which exceed the precision threshold.) p_{min} was fixed at 0.95 in all experiments in this paper.

   Thus at each iteration the method induces at most $n \times k$ rules, where k is the number of possible labels ($k = 3$ in the experiments in this paper).

5. Label the training set using the current set of *contextual* rules. Examples where no rule applies are left unlabeled.

6. On this new labeled set, select up to $n \times k$ spelling rules using the same method as in step 4. Set the *spelling* rules to be the seed set plus the rules selected.

7. If $n < 2500$ set $n = n + 5$ and return to step 3. Otherwise, label the training data with the combined spelling/contextual decision list, then induce a final decision list from the labeled examples where all rules (regardless of strength) are added to the decision list.

3.3 The Algorithm in (Yarowsky 95)

We can now compare this algorithm to that of (Yarowsky 95). The core of Yarowsky's algorithm is as follows:

1. **Initialization**: Set the decision list equal to the set of seed rules.

2. Label the training set using the current set of rules.

3. Use the labels to learn a decision list $h(x, y)$ where h is defined by the formula in equation 2, with counts restricted to training data examples that have been labeled in step 2.

[3]Note that taking the top n most frequent rules already makes the method robust to low count events, hence we do not use smoothing, allowing low-count high-precision features to be chosen on later iterations.

Set the decision list to include *all* rules whose (smoothed) strength is above some threshold p_{min}.

4. Return to step 2.

There are two differences between this method and the **DL-CoTrain** algorithm:

• The **DL-CoTrain** algorithm is rather more cautious, imposing a gradually increasing limit on the number of rules that can be added at each iteration.

• The **DL-CoTrain** algorithm has separated the *spelling* and *contextual* features, alternating between labeling and learning with the two types of features. Thus an explicit assumption about the redundancy of the features — that either the spelling or context alone should be sufficient to build a classifier — has been built into the algorithm.

To measure the contribution of each modification, a third, intermediate algorithm, **Yarowsky-cautious** was also tested. **Yarowsky-cautious** does not separate the spelling and contextual features, but does have a limit on the number of rules added at each stage. (Specifically, the limit n starts at 5 and increases by 5 at each iteration.)

The first modification – cautiousness – is a relatively minor change. It was motivated by the observation that the (Yarowsky 95) algorithm added a very large number of rules in the first few iterations. Taking only the highest frequency rules is much "safer", as they tend to be very accurate. This intuition is born out by the experimental results.

The second modification is more important, and is discussed in the next section.

3.4 Justification for the Separation of Contextual and Spelling Features

An important reason for separating the two types of features is that this opens up the possibility of theoretical analysis of the use of unlabeled examples. (Blum and Mitchell 98) describe learning in the following situation:

• Each example is represented by a feature vector x drawn from a set of possible values (an instance space) X. The task is to learn a classification function $f : X \rightarrow Y$ where Y is a set of possible labels.

• The features can be separated into two types: $X = X_1 \times X_2$ where X_1 and X_2 correspond to two different "views" of an example. In the named entity task, X_1 might be the instance space for the spelling features, X_2 might be the instance space for the contextual features. By this assumption, each element $x \in X$ can also be represented as $(x_1, x_2) \in X_1 \times X_2$.

• Each view of the example is sufficient for classification. That is, there exist functions f_1 and f_2 such that for any example $x = (x_1, x_2)$, $f(x) = f_1(x_1) = f_2(x_2)$. We never see an example $x = (x_1, x_2)$ in training or test data such that $f_1(x_1) \neq f_2(x_2)$.

Thus the method makes the fairly strong assumption that the features can be partitioned into two types such that each type alone is sufficient for classification.

• x_1 and x_2 are not correlated too tightly. (For example, there is not a deterministic function from x_1 to x_2.)

Now assume we have n pairs $(x_{1,i}, x_{2,i})$ drawn from $X_1 \times X_2$, where the first m pairs have labels y_i, whereas for $i = m+1...n$ the pairs are unlabeled. In a fully supervised setting, the task is to learn a function f such that for all $i = 1...m$, $f(x_{1,i}, x_{2,i}) = y_i$. In the cotraining case , (Blum and Mitchell 98) argue that the task should be to induce functions f_1 and f_2 such that

1. $f_1(x_{1,i}) = f_2(x_{2,i}) = y_i$ for $i = 1...m$
2. $f_1(x_{1,i}) = f_2(x_{2,i})$ for $i = m+1...n$

So f_1 and f_2 must (1) correctly classify the labeled examples, and (2) must agree with each other on the unlabeled examples. The key point is that the second constraint can be remarkably powerful in reducing the complexity of the learning problem.

(Blum and Mitchell 98) give an example that illustrates just how powerful the second constraint can be. Consider the case where $|X_1| = |X_2| = N$ and N is a "medium" sized number so that it is feasible to collect $O(N)$ unlabeled examples. Assume that the two classifiers are "rote learners": that is, f_1 and f_2 are defined through look-up tables that list a label for each member of X_1 or X_2. The problem is a binary classification problem. The problem can be represented as a graph with $2N$ vertices corresponding to the members of X_1 and X_2. Each unlabeled pair $(x_{1,i}, x_{2,i})$ is represented as an edge between nodes corresponding to $x_{1,i}$ and $x_{2,i}$ in the graph. An edge indicates that the two features must have the same label. Given a sufficient number of randomly drawn unlabeled examples (i.e., edges), we will induce two completely connected components that together span the entire graph. Each vertex within a connected component must have the same label — in the binary classification case, we need a *single* labeled example to identify which component should get which label.

(Blum and Mitchell 98) go on to give PAC results for learning in the cotraining case. They also describe an application of cotraining to classifying web pages (the two feature sets are the words on the page, and other pages pointing to the page). The method halves the error rate in comparison to a method using the labeled examples alone.

Limitations of (Blum and Mitchell 98): While the assumptions of (Blum and Mitchell 98) are useful in developing both theoretical results and an intuition for the problem, the assumptions are quite limited. In particular, it may not be possible to learn functions $f_1(x_{1,i}) = f_2(x_{2,i})$ for $i = m + 1...n$: either because there is some noise in the data, or because it is just not realistic to expect to learn perfect classifiers given the features used for representation. It may be more realistic to replace the second criteria with a softer one, for example (Blum and Mitchell 98) suggest the alternative

1. $f_1(x_{1,i}) = f_2(x_{2,i}) = y_i$ for $i = 1...m$
2. The choice of f_1 and f_2 must minimize the number of examples for which $f_1(x_{1,i}) \neq f_2(x_{2,i})$.

Alternatively, if f_1 and f_2 are probabilistic learners, it might make sense to encode the second constraint as one of minimizing some measure of the distance between the distributions given by the two learners. The question of what soft function to pick, and how to design algorithms which optimize it, is an open question, but appears to be a promising way of looking at the problem.

The **DL-CoTrain** algorithm can be motivated as being a greedy method of satisfying the above 2 constraints. At each iteration the algorithm increases the number of rules, while maintaining a high level of agreement between the spelling and contextual decision lists. Inspection of the data shows that at $n = 2500$, the two classifiers both give labels on 44,281 (49.2%) of the unlabeled examples, and give the *same* label on 99.25% of these cases. So the success of the algorithm may well be due to its success in maximizing the number of unlabeled examples on which the two decision lists agree. In the next section we present an alternative approach that builds two classifiers while attempting to satisfy the above constraints as much as possible. The algorithm, called **CoBoost**, has the advantage of being more general than the decision-list learning algorithm, and, in fact, can be combined with almost any supervised machine learning algorithm.

Input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$; $\mathbf{x}_i \in 2^{\mathcal{X}}$, $y_i = \pm 1$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Get weak hypothesis $h_t : 2^{\mathcal{X}} \to \mathbb{R}$ by training weak learner using distribution D_t.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)} / Z_t$$

where $Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$.

Output final hypothesis:
$$f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

Figure 1: The AdaBoost algorithm for binary problems (Schapire and Singer 98).

4 A Boosting-based algorithm

This section describes an algorithm based on boosting algorithms, which were previously developed for supervised machine learning problems. We first give a brief overview of boosting algorithms. We then discuss how we adapt and generalize a boosting algorithm, AdaBoost, to the problem of named entity classification. The new algorithm, which we call **CoBoost**, uses labeled and unlabeled data and builds two classifiers in parallel. (We would like to note though that unlike previous boosting algorithms, the **CoBoost** algorithm presented here is not a boosting algorithm under Valiant's (Valiant 84) Probably Approximately Correct (PAC) model.)

4.1 The AdaBoost algorithm

This section describes AdaBoost, which is the basis for the **CoBoost** algorithm. AdaBoost was first introduced in (Freund and Schapire 97); (Schapire and Singer 98) gave a generalization of AdaBoost which we will use in this paper. For a description of the application of AdaBoost to various NLP problems see the paper by Abney, Schapire, and Singer in this volume.

The input to AdaBoost is a set of training examples $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\rangle$. Each $\mathbf{x}_i \in 2^{\mathcal{X}}$ is the set of features constituting the ith example. For the moment we will assume that there are only two possible labels: each y_i is in $\{-1, +1\}$. AdaBoost is given access to a *weak learning* algorithm, which accepts as input the training examples, along with a distribution over the instances. The distribution specifies the relative weight, or importance, of each example — typically, the weak learner will attempt to minimize the weighted error on the training set, where the distribution specifies the weights.

The weak learner for two-class problems computes a weak hypothesis h from the input space into the reals ($h : 2^{\mathcal{X}} \to \mathbb{R}$), where the sign[4] of $h(\mathbf{x})$ is interpreted as the predicted label and the magnitude $|h(\mathbf{x})|$ is the confidence in the prediction: large numbers for $|h(\mathbf{x})|$ indicate high confidence in the prediction, and numbers close to zero indicate low confidence. The weak hypothesis can abstain from predicting the label of an instance $\mathbf{x}$ by setting $h(\mathbf{x}) = 0$. The final strong hypothesis, denoted $f(\mathbf{x})$, is then the sign of a weighted sum of the weak hypotheses, $f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$, where the weights α_t are determined during the run of the algorithm, as we describe below.

Pseudo-code describing the generalized boosting algorithm of Schapire and Singer is given in Figure 1. Note that Z_t is a normalization constant that ensures the distribution D_{t+1} sums to 1; it is a function of the weak hypothesis h_t and the weight for that hypothesis α_t chosen at the tth round. The normalization factor plays an important role in the AdaBoost algorithm. Schapire and Singer show that the training error is bounded above by

$$\frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i\left(\sum_{t=1}^{T}\alpha_t h_t(\mathbf{x}_i)\right)\right) = \prod_t Z_t \ . \quad (3)$$

Thus, in order to greedily minimize an upper bound on training error, on each iteration we should search for the weak hypothesis h_t and the weight α_t that minimize Z_t.

In our implementation, we make perhaps the simplest choice of weak hypothesis. Each h_t is a function that predicts a label ($+1$ or -1) on examples containing a particular feature x_t, while abstaining on other examples:

$$h_t(\mathbf{x}) = \begin{cases} \pm 1 & x_t \in \mathbf{x} \\ 0 & x_t \notin \mathbf{x} \end{cases} \ .$$

The prediction of the strong hypothesis can then be written as

$$f(\mathbf{x}) = \text{sign}\left(\sum_{t:x_t \in \mathbf{x}} \alpha_t h_t(\mathbf{x})\right) \ .$$

[4]We define $\text{sign}(0) = 0$.

We now briefly describe how to choose h_t and α_t at each iteration. Our derivation is slightly different from the one presented in (Schapire and Singer 98) as we restrict α_t to be positive. Z_t can be written as follows

$$\begin{aligned} Z_t &= \sum_{i:x_t \notin \mathbf{x}_i} D_t(i) \\ &+ \sum_{i:x_t \in \mathbf{x}_i} D_t(i) \exp(-y_i \alpha_t h_t(\mathbf{x}_i)) \ . \end{aligned} \quad (4)$$

Let

$$\begin{aligned} W_0 &= \sum_{i:h_t(\mathbf{x}_i)=0} D_t(i) \ , \\ W_+ &= \sum_{i:h_t(\mathbf{x}_i)=y_i} D_t(i) \ , \\ W_- &= \sum_{i:h_t(\mathbf{x}_i)=-y_i} D_t(i) \ . \end{aligned}$$

Following the derivation of Schapire and Singer, providing that $W_+ > W_-$, Equ. (4) is minimized by setting

$$\alpha_t = \frac{1}{2}\ln\left(\frac{W_+}{W_-}\right) \ . \quad (5)$$

Since a feature may be present in only a few examples, W_- can be in practice very small or even 0, leading to extreme confidence values. To prevent this we "smooth" the confidence by adding a small value, ϵ, to both W_+ and W_-, giving $\alpha_t = \frac{1}{2}\ln\left(\frac{W_+ + \epsilon}{W_- + \epsilon}\right)$.

Plugging the value of α_t from Equ. (5) and h_t into Equ. (4) gives

$$Z_t = W_0 + 2\sqrt{W_+ W_-} \quad (6)$$

In order to minimize Z_t, at each iteration the final algorithm should choose the weak hypothesis (i.e., a feature x_t) which has values for W_+ and W_- that minimize Equ. (6), with $W_+ > W_-$.

4.2 The CoBoost algorithm

We now describe the **CoBoost** algorithm for the named entity problem. Following the convention presented in earlier sections, we assume that each example is an instance pair of the from $(\mathbf{x}_{1,i}, \mathbf{x}_{2,i})$ where $\mathbf{x}_{j,i} \in 2^{\mathcal{X}_j}, j \in \{1, 2\}$. In the named-entity problem each example is a (*spelling*,*context*) pair. The first m pairs have labels y_i, whereas for $i = m+1, \ldots, n$ the pairs are unlabeled. We make the assumption that for each example, both

$\mathbf{x}_{1,i}$ and $\mathbf{x}_{2,i}$ alone are sufficient to determine the label y_i. The learning task is to find two classifiers $f_1 : 2^{\mathcal{X}_1} \rightarrow \{-1,+1\}$ $f_2 : 2^{\mathcal{X}_2} \rightarrow \{-1,+1\}$ such that $f_1(\mathbf{x}_{1,i}) = f_2(\mathbf{x}_{2,i}) = y_i$ for examples $i = 1,\ldots,m$, and $f_1(\mathbf{x}_{1,i}) = f_2(\mathbf{x}_{2,i})$ as often as possible on examples $i = m+1,\ldots,n$. To achieve this goal we extend the auxiliary function that bounds the training error (see Equ. (3)) to be defined over unlabeled as well as labeled instances. Denote by $g_j(\mathbf{x}) = \sum_t \alpha_t^j h_t^j(\mathbf{x}), j \in \{1,2\}$ the *unthresholded* strong-hypothesis (i.e., $f_j(\mathbf{x}) = \mathrm{sign}(g_j(\mathbf{x}))$). We define the following function:

$$\begin{aligned} Z_{\mathrm{CO}} \stackrel{def}{=} & \sum_{i=1}^{m} \exp(-y_i g_1(\mathbf{x}_{1,i})) \\ + & \sum_{i=1}^{m} \exp(-y_i g_2(\mathbf{x}_{2,i})) \\ + & \sum_{i=m+1}^{n} \exp(-f_2(\mathbf{x}_{2,i}) g_1(\mathbf{x}_{1,i})) \\ + & \sum_{i=m+1}^{n} \exp(-f_1(\mathbf{x}_{1,i}) g_2(\mathbf{x}_{2,i})) \ . \quad (7) \end{aligned}$$

If Z_{CO} is small, then it follows that the two classifiers must have a low error rate on the labeled examples, and that they also must give the same label on a large number of unlabeled instances. To see this, note that the first two terms in the above equation correspond to the function that AdaBoost attempts to minimize in the standard supervised setting (Equ. (3)), with one term for each classifier. The two new terms force the two classifiers to agree, as much as possible, on the unlabeled examples. Put another way, the minimum of Equ. (7) is at 0 when: 1) $\forall i : \mathrm{sign}(g_1(\mathbf{x}_i)) = \mathrm{sign}(g_2(\mathbf{x}_i))$; 2) $|g_j(\mathbf{x}_i)| \rightarrow \infty$; and 3) $\mathrm{sign}(g_j(\mathbf{x}_i)) = y_i$ for $i = 1,\ldots,m$. In fact, Z_{CO} provides a bound on the sum of the classification error of the labeled examples and the number of disagreements between the two classifiers on the unlabeled examples. Formally, let ϵ_1 (ϵ_2) be the number of classification errors of the first (second) learner on the training data, and let ϵ_{CO} be the number of unlabeled examples on which the two classifiers disagree. Then, it can be verified that

$$\epsilon_1 + \epsilon_2 + 2\epsilon_{\mathrm{CO}} \leq Z_{\mathrm{CO}} \ .$$

We can now derive the **CoBoost** algorithm as a means of minimizing Z_{CO}. The algorithm builds two classifiers in parallel from labeled and unlabeled data. As in boosting, the algorithm works in rounds. Each round is composed of two stages; each stage updates one of the classifiers while keeping the other classifier fixed. Denote the unthresholded classifiers after $t-1$ rounds by g_j^{t-1} and assume that it is the turn for the first classifier to be updated while the second one is kept fixed. We first define "pseudo-labels", $\tilde{y}_i$, as follows:

$$\tilde{y}_i = \begin{cases} y_i & 1 \leq i \leq m \\ \mathrm{sign}(g_2^{t-1}(\mathbf{x}_{2,i})) & m < i \leq n \end{cases}$$

Thus the first m labels are simply copied from the labeled examples, while the remaining $(n-m)$ examples are taken as the current output of the second classifier. We can now add a new weak hypothesis h_t^1 based on a feature in $\mathcal{X}_1$ with a confidence value α_t^1. h_t^1 and α_t^1 are chosen to minimize the function

$$Z_{\mathrm{CO}}^1 = \sum_{i=1}^{n} \exp(-\tilde{y}_i(g_1^{t-1}(\mathbf{x}_i) + \alpha_t^1 h_t^1(\mathbf{x}_{1,i}))) \ . \quad (8)$$

We now define, for $1 \leq i \leq n$, the following *virtual* distribution,

$$D_t^1(i) = \frac{1}{Z_t^1} \exp(-\tilde{y}_i g_1^{t-1}(\mathbf{x}_{1,i})),$$

As before, Z_t^1 is a normalization constant. Equ. (8) can now be rewritten[5] as

$$\sum_{i=1}^{n} D_t^1(i) \exp(-\tilde{y}_i \alpha_t^1 h_t^1(\mathbf{x}_{1,i})) \ ,$$

which is of the same form as the function Z_t used in AdaBoost. Using the virtual distribution $D_t^1(i)$ and pseudo-labels $\tilde{y}_i$, values for W_0, W_+ and W_- can be calculated for each possible weak hypothesis (i.e., for each feature $x \in \mathcal{X}_1$); the weak hypothesis with minimal value for $W_0 + 2\sqrt{W_+ W_-}$ can be chosen as before; and the weight for this weak hypothesis $\alpha_t = \frac{1}{2}\ln\left(\frac{W_+ + \epsilon}{W_- + \epsilon}\right)$ can be calculated. This procedure is repeated for T rounds while alternating between the two classifiers. The pseudo-code describing the algorithm is given in Fig. 2.

The **CoBoost** algorithm described above divides the function Z_{CO} into two parts: $Z_{\mathrm{CO}} = Z_{\mathrm{CO}}^1 + Z_{\mathrm{CO}}^2$. On each step **CoBoost** searches for a feature and a weight so as to minimize *either* Z_{CO}^1 or Z_{CO}^2. In

[5] up to a constant factor Z_t^1 which does not affect the minimization of Equ. (8) w.r.t. h_t and α_t.

Input: $\{(\mathbf{x}_{1,i}, \mathbf{x}_{2,i})\}_{i=1}^{n}, \{y_i\}_{i=1}^{m}$
Initialize: $\forall i, j : g_j^0(\mathbf{x}_i) = 0$.
For $t = 1, \ldots, T$ and for $j = 1, 2$:

- Set pseudo-labels:

$$\tilde{y}_i = \begin{cases} y_i & 1 \le i \le m \\ \mathrm{sign}(g_{3-j}^{t-1}(\mathbf{x}_{3-j,i})) & m < i \le n \end{cases}$$

- Set virtual distribution:

$$D_t^j(i) = \frac{1}{Z_t^j} \exp\left(-\tilde{y}_i g_j^{t-1}(\mathbf{x}_{j,i})\right)$$

where $Z_t^j = \sum_{i=1}^{n} \exp(-\tilde{y}_i g_j^{t-1}(\mathbf{x}_{j,i}))$.
- Get a weak hypothesis $h_t^j : 2^{\mathcal{X}_j} \to \mathbb{R}$. by training weak learner j using distribution D_t^j.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$\forall i : g_j^t(\mathbf{x}_{j,i}) = g_j^{t-1}(\mathbf{x}_{j,i}) + \alpha_t h_t^j(\mathbf{x}_{j,i}) \ .$$

Output final hypothesis:
$$f(\mathbf{x}) = \mathrm{sign}\left(\sum_{j=1}^{2} g_j^T(\mathbf{x}_j)\right)$$

Figure 2: The **CoBoost** algorithm.

practice, this greedy approach almost always results in an overall decrease in the value of Z_{CO}. Note, however, that there might be situations in which Z_{CO} in fact increases.

One implementation issue deserves some elaboration. Note that in our formalism a weak-hypothesis can abstain. In fact, during the first rounds many of the predictions of g_1, g_2 are zero. Thus corresponding pseudo-labels for instances on which g_j abstain are set to zero and these instances do not contribute to the objective function. Each learner is free to pick the labels for these instances. This allow the learners to "bootstrap" each other by filling the labels of the instances on which the other side has abstained so far.

The **CoBoost** algorithm just described is for the case where there are two labels: for the named entity task there are three labels, and in general it will be useful to generalize the **CoBoost** algorithm to the multiclass case. Several extensions of AdaBoost for multiclass problems have been suggested (Freund and Schapire 97; Schapire and Singer 98). In this work we extended the AdaBoost.MH (Schapire and Singer 98) algorithm to the cotraining case. AdaBoost.MH maintains a distribution over instances *and* labels; in addition, each weak-hypothesis outputs a confidence *vector* with one confidence value for each possible label. We again adopt an approach where we alternate between two classifiers: one classifier is modified while the other remains fixed. Pseudo-labels are formed by taking seed labels on the labeled examples, and the output of the fixed classifier on the unlabeled examples. AdaBoost.MH can be applied to the problem using these pseudo-labels in place of supervised examples.

For the experiments in this paper we made a couple of additional modifications to the **CoBoost** algorithm. The algorithm in Fig. (2) was extended to have an additional, innermost loop over the (3) possible labels. The weak hypothesis chosen was then restricted to be a predictor in favor of this label. Thus at each iteration the algorithm is forced to pick features for the `location`, `person` and `organization` in turn for the classifier being trained. This modification brings the method closer to the **DL-CoTrain** algorithm described earlier, and is motivated by the intuition that all three labels should be kept healthily populated in the unlabeled examples, preventing one label from dominating — this deserves more theoretical investigation.

We also removed the `context-type` feature type when using the **CoBoost** approach. This "default" feature type has 100% coverage (it is seen on every example) but a low, baseline precision. When this feature type was included, **CoBoost** chose this default feature at an early iteration, thereby giving non-abstaining pseudo-labels for all examples, with eventual convergence to the two classifiers agreeing by assigning the same label to almost all examples. Again, this deserves further investigation.

Finally, we would like to note that it is possible to devise similar algorithms based with other objective functions than the one given in Equ. (7), such as the likelihood function used in maximum-entropy problems and other generalized additive models (Lafferty 99). We are currently exploring such algorithms.

5 An EM-based approach

The Expectation Maximization (EM) algorithm (Dempster, Laird and Rubin 77) is a common approach for unsupervised training; in this section we describe its application to the named entity problem. A generative model was applied (similar to naive Bayes) with the three labels as hidden variables on unlabeled examples, and observed variables on (seed) labeled examples. The model was parameterized such that the joint probability of a *(label, feature-set)* pair $P(y_i, \mathbf{x_i})$ is written as

$$\begin{aligned} P(y_i, \mathbf{x_i}) &= P(y_i, x_{i1} \dots x_{im_i}) \\ &= P(y_i)P(m_i)\prod_{j=1}^{m_i} P(x_{ij}|y_i) \end{aligned} \quad (9)$$

The model assumes that $(y, \mathbf{x})$ pairs are generated by an underlying process where the label is first chosen with some prior probability $P(y_i)$; the number of features m_i is then chosen with some probability $P(m_i)$; finally the features are independently generated with probabilities $P(x_{ij}|y_i)$.

We again assume a training set of n examples $\{\mathbf{x_1} \dots \mathbf{x_n}\}$ where the first m examples have labels $\{y_1 \dots y_m\}$, and the last $(n-m)$ examples are unlabeled. For the purposes of EM, the "observed" data is $\{(\mathbf{x_1}, y_1) \dots (\mathbf{x_m}, y_m), \mathbf{x_{m+1}} \dots \mathbf{x_n}\}$, and the hidden data is $\{y_{m+1} \dots y_n\}$. The likelihood of the observed data under the model is

$$\prod_{i=1}^{m} P(y_i, \mathbf{x_i}) \times \prod_{i=m+1}^{n} \sum_{y=1}^{k} P(y, \mathbf{x_i}) \quad (10)$$

where $P(y_i, \mathbf{x_i})$ is defined as in (9). Training under this model involves estimation of parameter values for $P(y)$, $P(m)$ and $P(x|y)$. The maximum likelihood estimates (i.e., parameter values which maximize 10) can not be found analytically, but the EM algorithm can be used to hill-climb to a local maximum of the likelihood function from some initial parameter settings. In our experiments we set the parameter values randomly, and then ran EM to convergence.

Given parameter estimates, the label for a test example $\mathbf{x}$ is defined as

$$f(\mathbf{x}) = \arg\max_{y \in \{1 \dots k\}} p(x, y) \quad (11)$$

We should note that the model in equation 9 is deficient, in that it assigns greater than zero probability to some feature combinations that are impossible. For example, the independence assumptions mean that the model fails to capture the dependence between specific and more general features (for example the fact that the feature `full-string=New_York` is always seen with the features `contains(New)` and `contains(York)` and is never seen with a feature such as `contains(Group)`). Unfortunately, modifying the model to account for these kind of dependencies is not at all straightforward.

Learning Algorithm	Accuracy (Clean)	Accuracy (Noise)
Baseline	45.8%	41.8%
EM	83.1%	75.8%
(Yarowsky 95)	81.3%	74.1%
Yarowsky-cautious	91.2%	83.2%
DL-CoTrain	91.3%	83.3%
CoBoost	91.1%	83.1%

Table 2: Accuracy for different learning methods. The baseline method tags all entities as the most frequent class type (organization).

6 Evaluation

88,962 (*spelling*,*context*) pairs were extracted as training data. 1,000 of these were picked at random, and labeled by hand to produce a test set. We chose one of four labels for each example: `location`, `person`, `organization`, or `noise` where the `noise` category was used for items that were outside the three categories. The numbers falling into the `location`, `person`, `organization` categories were 186, 289 and 402 respectively.

123 examples fell into the `noise` category. Of these cases, 38 were temporal expressions (either a day of the week or month of the year). We excluded these from the evaluation as they can be easily identified with a list of days/months. This left 962 examples, of which 85 were noise. Taking N_c to be the number of examples an algorithm classified correctly (where all gold standard items labeled `noise` were counted as being incorrect), we calculated two measures of accuracy:

$$\text{Accuracy : Noise} = \frac{N_c}{962} \quad (12)$$

$$\text{Accuracy : Clean} = \frac{N_c}{962 - 85} \quad (13)$$

See Tab. 2 for the accuracy of the different methods. Note that on some examples (around 2% of the test set) **CoBoost** abstained altogether; in these cases we labeled the test example with the baseline, `organization`, label. Fig. (3) shows learning curves for **CoBoost**.

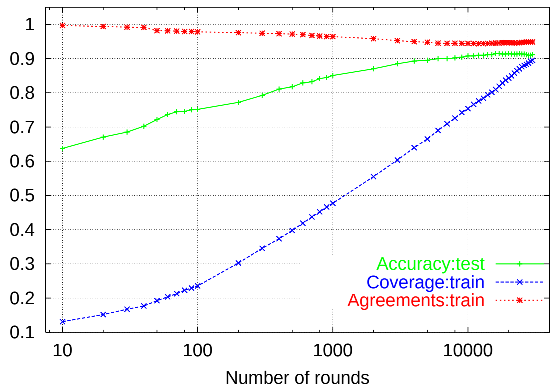

Figure 3: Learning curves for **CoBoost**. The graph gives the accuracy on the test set, the coverage (proportion of examples on which both classifiers give a label rather than abstaining), and the proportion of these examples on which the two classifiers agree. With each iteration more examples are assigned labels by both classifiers, while a high level of agreement ($> 94\%$) is maintained between them. The test accuracy more or less asymptotes.

7 Conclusions

Unlabeled examples in the named-entity classification problem can reduce the need for supervision to a handful of seed rules. In addition to a heuristic based on decision list learning, we also presented a boosting-like framework that builds on ideas from (Blum and Mitchell 98). The method uses a "soft" measure of the agreement between two classifiers as an objective function; we described an algorithm which directly optimizes this function. We are currently exploring other methods that employ similar ideas and their formal properties. Future work should also extend the approach to build a complete named entity extractor — a method that pulls proper names from text and then classifies them. The contextual rules are restricted and may not be applicable to every example, but the spelling rules are generally applicable and should have good coverage. The problem of "noise" items that do not fall into any of the three categories also needs to be addressed.

References

M. Berland and E. Charniak. 1999. Finding Parts in Very Large Corpora. In *Proceedings of the the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*.

D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. 1997. Nymble: a High-Performance Learning Name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 194-201.

A. Blum and T. Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*.

E. Brill. 1995. Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the Third Workshop on Very Large Corpora*.

S. Brin. 1998. Extracting Patterns and Relations from the World Wide Web. In *WebDB Wokshop at EDBT '98*.

M. Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184-191.

A.P. Dempster, N.M. Laird, and D.B. Rubin, (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm, *Journal of the Royal Statistical Society*, Ser B, 39, 1-38.

Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

M. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*.

Michael Kearns. Thoughts on hypothesis boosting. Unpublished manuscript, December 1988.

J. Lafferty. Additive Models, Boosting, and Inference for Generalized Divergences. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 1999.

Proceedings of the Sixth Message Understanding Conference (MUC-6). Morgan Kaufmann, San Mateo, CA.

E. Riloff and J. Shepherd. 1997. A Corpus-Based Approach for Building Semantic Lexicons. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*.

E. Riloff and R. Jones. 1999. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. *In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998. To appear, *Machine Learning*.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

D. Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods.In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA, pp. 189-196.