Abstract

The named entity disambiguation task is to resolve the many-to-many correspondence between ambiguous names and the unique real-world entity. This task can be modeled as a classification problem, provided that positive and negative examples are available for learning binary classifiers. High-quality sense-annotated data, however, are hard to be obtained in streaming environments, since the training corpus would have to be constantly updated in order to accommodate the fresh data coming on the stream. On the other hand, few positive examples plus a large amount of unlabeled data may be easily acquired. Producing binary classifiers directly from this data, however, leads to poor disambiguation performance, and thus, we propose to enhance the quality of the classifiers using finer-grained variations of the well-known Expectation-Maximization (EM) algorithm. We conducted a systematic evaluation using Twitter streaming data, and the results show that our classifiers are extremely effective, providing improvements ranging from 1% to 20%, when compared to the current state-of-the-art biased SVMs, being 120 times faster.

1 Introduction

Human language is not exact. For instance, an entity may be referred by multiple names (i.e., polysemy), and also the same name may refer to different entities depending on the surrounding context (i.e., homonymy). The task of named entity disambiguation is to identify which names refer to the same entity in a textual collection (Sarmento et al., 2009). The emergence of new communication technologies, such as micro-blog platforms, brought a humongous amount of textual mentions with ambiguous entity names, raising an urgent need for novel disambiguation approaches and algorithms.

In this paper we address the named entity disambiguation task under a particularly challenging scenario. We are given a stream of messages from a micro-blog channel such as Twitter and a list of names used for mentioning a specific entity. Our problem is to continually monitor the stream and predict whether an incoming message containing indeed refers to (positive example) or not (negative example). This scenario brings some challenges that must be overcome. First, micro-blog messages are composed of a small amount of words and they are written in an informal, sometimes cryptic style. These characteristics make harder the identification of entities and the semantics of their relationships. Further, the scarcity of text in the messages makes it hard to properly characterize a common context for the entities. Second, as we need to monitor messages that keep coming at a fast pace, we cannot afford to gather information from external sources on-the-fly. Finally, fresh data coming in the stream introduces new patterns, quickly invalidating static disambiguation models.

We hypothesize that the lack of information in each individual message and from external sources

---

1The term entity refers to anything that has a distinct, separate (materialized or not) existence.

2Twitter is one of the fastest-growing micro-blog channels, and an authoritative source for breaking news (Jansen et al., 2009).
can be compensated by using information obtained from the large and diverse amount of text in a stream of messages taken as a whole, that is, thousands of messages per second coming from distinct sources.

The information embedded in such a stream of messages may be exploited for entity disambiguation through the application of supervised learning methods, for instance, with the application of binary classifiers. Such methods, however, suffer from a data acquisition bottleneck, since they are based on training datasets that are built by skilled human annotators who manually inspect the messages. This annotation process is usually lengthy and laborious, being clearly unfeasible to be adopted in data streaming scenarios. As an alternative to such manual process, a large amount of unlabeled data, augmented with a small amount of (likely) positive examples, can be collected automatically from the message stream (Liu et al., 2003; Denis, 1998; Comité et al., 1999; Letouzey et al., 2000).

Binary classifiers may be learned from such data by considering unlabeled data as negative examples. This strategy, however, leads to classifiers with poor disambiguation performance, due to a potentially large number of false-negative examples. In this paper we propose to refine binary classifiers iteratively, by performing Expectation-Maximization (EM) approaches (Dempster et al., 1977). Basically, a partial classifier is used to evaluate the likelihood of an unlabeled example being a positive example or a negative example, thus automatically and (continuously) creating a labeled training corpus. This process continues iteratively by changing the label of some examples (an operation we call label-transition), so that, after some iterations, the combination of labels is expected to converge to the one for which the observed data is most likely. Based on such an approach, we introduce novel disambiguation algorithms that differ among themselves on the granularity in which the classifier is updated, and on the label-transition operations that are allowed.

An important feature of the proposed approach is that, at each iteration of the EM-process, a new classifier (an improved one) is produced in order to account for the current set of labeled examples. We introduce a novel strategy to maintain the classifiers up-to-date incrementally after each iteration, or even after each label-transition operation. Indeed, we theoretically show that our classifier needs to be updated just partially and we are able to determine exactly which parts must be updated, making our disambiguation methods extremely fast.

To evaluate the effectiveness of the proposed algorithms, we performed a systematic set of experiments using large-scale Twitter data containing messages with ambiguous entity names. In order to validate our claims, disambiguation performance is investigated by varying the proportion of false-negative examples in the unlabeled dataset. Our algorithms are compared against a state-of-the-art technique for named entity disambiguation based on classifiers, providing performance gains up to 11% and being approximately 110 times faster.

2 Related Work

In the context of databases, traditional entity disambiguation methods rely on similarity functions over attributes associated to the entities (de Carvalho et al., 2011). Obviously, such an approach is unfeasible for the scenario we consider here. Still on databases, (Bhattacharya and Getoor, 2007; Dong et al., 2005) propose graph-based disambiguation methods that generate clusters of co-referent entities using known relationships between entities of several types (e.g., marriage, co-authorship). Methods to disambiguate person names in e-mail (Minkov et al., 2006) and Web pages (Bekkerman and McCallum, 2005; Wan et al., 2005) have employed similar ideas. In e-mails, information taken from the header of the messages leads to establish relationships between users and building a co-reference graph. In Web pages, reference information come naturally from links. Such graph-based approach could hardly be applied to the context we consider, in which the implied relationships between entities mentioned in a given micro-blog message are not clearly defined.

In the case of textual corpora, traditional disambiguation methods represent entity names and their context (i.e., words, phrases and other names occurring near them) as weighted vectors (Bagga and Baldwin, 1998; Pedersen et al., 2005). To evaluate whether two names refer to the same entity, these methods compute the similarity between these vectors. Clusters of co-referent names are then built based on such similarity measure. Although effec-
tive for the tasks considered in these papers, the simplistic BOW-based approaches they adopt are not suitable for cases in which the context is harder to capture due to the small number of terms available or to informal writing style. This is the case in micro-blog messages, but also in other types of textual corpora. To address these problems, some authors argue that contextual information may be enriched with knowledge from external sources, such as search results and the Wikipedia (Cucerzan, 2007; Bunescu and Pasca, 2006; Han and Zhao, 2009). While such a strategy is feasible in an off-line setting, two problems arise when monitoring streams of micro-blog messages. First, gathering information from external sources through the Internet can be costly and, second, informal mentions to named entities make it hard to look for related information in such sources.

The disambiguation methods we propose fall into a learning scenario known as PU (positive and unlabeled) learning (Liu et al., 2003; Denis, 1998; Comité et al., 1999; Letouzey et al., 2000), in which a classifier is built from a set of positive examples plus unlabeled data (i.e., positive and negative examples). Most of the approaches for PU learning, such as the biased-SVM approach (Li and Liu, 2003), are based on extracting negative examples from unlabeled data. We notice that existing approaches for PU learning, such as the biased-SVM approach (Li and Liu, 2003), are not likely to scale given the restrictions imposed by streaming data. Thus, we propose highly incremental alternate approaches, which are able to process large-scale streaming data.

3 Disambiguation in Streaming Data

Consider a stream of messages from a micro-blog channel such as Twitter and let $n_1, n_2, \ldots, n_N$ be names used for mentioning a specific entity $e$ in these messages. Our problem is to continually monitor the stream and predict whether an incoming message containing $n_i$ indeed refers to $e$ or not.

This task may be accomplished through the application of classification techniques. In this case, we are given an input data set called the training corpus (denoted as $D$) which consists of examples of the form $<e, m, c>$, where $e$ is the entity, $m$ is a message containing the entity name (i.e., any $n_i$), and $c \in \{\varnothing, \oplus\}$ is a binary variable that specifies whether or not the entity name in $m$ refers to the desired real-world entity $e$. The training corpus is used to produce a classifier that relates textual patterns in $m$ to the value of $c$. The test set (denoted as $T$) consists of a set of records of the form $<e, m, ?>$, for which only the entity and the message are known. The classifier is used to indicate which messages in $T$ refer to (or not) the desired entity.

Supervised classifiers, however, are subject to a data acquisition bottleneck, since the creation of a training corpus requires skilled human annotators to manually inspect the messages. The cost associated with this annotation process may render vast amounts of examples unfeasible. In many cases, however, the acquisition of some positive examples is relatively inexpensive. For instance, as we are dealing with messages collected from micro-blog channels, we may exploit profiles (or hashtags) that are known to be strongly associated with the desired entity. Let us consider, as an illustrative example, the profile associated with a company (i.e., @bayer).

3.1 Expectation-Maximization Approach

In this paper we hypothesize that it is worthwhile to enhance the reliability of unlabeled examples, provided that this type of data is inexpensive and the enhancement effort will be then rewarded with an improvement in disambiguation performance. Thus, we propose a new approach based on the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). We assume two scenarios:

- the training corpus $D$ is composed of a small set of truly positive examples plus a large amount of unlabeled examples.
- the training corpus $D$ is composed of a small set of potentially positive examples plus a large amount of unlabeled examples.
In both scenarios, unlabeled examples are initially treated as negative ones, so that classifiers can be built from $\mathcal{D}$. Therefore, in both scenarios, $\mathcal{D}$ may contain false-negatives. In the second scenario, however, $\mathcal{D}$ may also contain false-positives.

**Definition 3.1:** The label-transition operation $x^{\ominus \rightarrow \ominus}$ turns a negative example $x^{\ominus} \in \mathcal{D}$ into a positive one $x^{\oplus}$. The training corpus $\mathcal{D}$ becomes $\{(\mathcal{D} - x^{\ominus}) \cup x^{\oplus}\}$. Similarly, the label-transition operation $x^{\ominus \rightarrow \oplus}$, turns a positive example $x^{\oplus} \in \mathcal{D}$ into a negative one $x^{\ominus}$. The training corpus $\mathcal{D}$ becomes $\{(\mathcal{D} - x^{\ominus}) \cup x^{\oplus}\}$.

Our Expectation Maximization (EM) methods employ a classifier which assigns to each example $x \in \mathcal{D}$ a probability $\alpha(x, \ominus)$ of being negative. Then, as illustrated in Algorithm 1, label-transition operations are performed, so that, in the end of the process, it is expected that the assigned labels converge to the combination for which the data is most likely. In the first scenario only operations $x^{\ominus \rightarrow \ominus}$ are allowed, while in the second scenario operations $x^{\ominus \rightarrow \ominus}$ are also allowed. In both cases, a crucial issue that affects the effectiveness of our EM-based methods concerns the decision of whether or not performing the label-transition operation. Typically, a transition threshold $\alpha_{\text{min}}$ is employed, so that a label-transition operation $x^{\ominus \rightarrow \ominus}$ is always performed if $x$ is a negative example and $\alpha(x, \ominus) \leq \alpha_{\text{min}}$. Similarly, operation $x^{\ominus \rightarrow \ominus}$ is always performed if $x$ is a positive example and $\alpha(x, \ominus) > \alpha_{\text{min}}$.

**Algorithm 1** Expectation-Maximization Approach.

**Given:**
- $\mathcal{D}$: training corpus
- $\mathcal{R}$: a binary classifier learned from $\mathcal{D}$

**Expectation step:**
- perform transition operations on examples in $\mathcal{D}$

**Maximization step:**
- update $\mathcal{R}$ and $\alpha(x, \ominus) \ \forall x \in \mathcal{D}$

The optimal value for $\alpha_{\text{min}}$ is not known in advance. Fortunately, data distribution may provide hints about proper values for $\alpha_{\text{min}}$. In our approach, instead of using a single value for $\alpha_{\text{min}}$, which would be applied to all examples indistinctly, we use a specific $\alpha_{\text{min}}$ threshold for each example $x \in \mathcal{D}$. Based on such an approach, we introduce fine-grained EM-based methods for named entity disambiguation under streaming data. A specific challenge is that the proposed methods perform several transition operations during each EM iteration, and each transition operation may invalidate parts of the current classifier, which must be properly updated. We take into consideration two possible update granularities:

- the classifier is updated after each EM iteration.
- the classifier is updated after each label-transition operation.

**Incremental Classifier:** As already discussed, the classifier must be constantly updated during the EM process. In this case, well-established classifiers, such as SVMs (Joachims, 2006), have to be learned entirely from scratch, replicating work by large. Thus, we propose as an alternative the use of Lazy Associative Classifiers (Veloso et al., 2006).

**Definition 3.2:** A classification rule is a specialized association rule $\{X \rightarrow c\}$ (Agrawal et al., 1993), where the antecedent $X$ is a set of terms (i.e., a termset), and the consequent $c$ indicates if the prediction is positive or negative (i.e., $c \in \{\ominus, \oplus\}$).

The domain for $X$ is the vocabulary of $\mathcal{D}$. The cardinality of rule $\{X \rightarrow c\}$ is given by the number of terms in the antecedent, that is $|X|$. The support of $X$ is denoted as $\sigma(X)$, and is the number of examples in $\mathcal{D}$ having $X$ as a subset. The confidence of $\{X \rightarrow c\}$ is denoted as $\theta(X \rightarrow c)$, and is the conditional probability of $c$ given the termset $X$, that is, $\theta(X \rightarrow c) = \frac{\sigma(X \cup c)}{\sigma(X)}$.

The classifier is denoted as $\mathcal{R}$, and it is composed of a set of rules $\{X \rightarrow c\}$ extracted from $\mathcal{D}$. Specifically, $\mathcal{R}$ is represented as a pool of entries with the form $<\text{key}, \text{properties}>$, where $\text{key}=\{X, c\}$ and $\text{properties}=\{\sigma(X), \sigma(X \cup c), \theta(X \rightarrow c)\}$. Each entry in the pool corresponds to a rule, and the key is used to facilitate fast access to rule properties.

Once the classifier $\mathcal{R}$ is extracted from $\mathcal{D}$, rules are collectively used to approximate the likelihood of an arbitrary example being positive ($\oplus$) or negative ($\ominus$). Basically, $\mathcal{R}$ is interpreted as a poll, in which each rule $\{X \rightarrow c\} \in \mathcal{R}$ is a vote given by $X$ for $\oplus$ or $\ominus$. Given an example $x$, a rule $\{X \rightarrow c\}$ is only considered a valid vote if it is applicable to $x$. 


Definition 3.3: A rule \( \{ \mathcal{X} \rightarrow c \} \in \mathcal{R} \) is said to be applicable to example \( x \) if \( \mathcal{X} \subseteq x \), that is, if all terms in \( \mathcal{X} \) are present in example \( x \).

We denote as \( \mathcal{R}_x \) the set of rules in \( \mathcal{R} \) that are applicable to example \( x \). Thus, only and all the rules in \( \mathcal{R}_x \) are considered as valid votes when classifying \( x \). Further, we denote as \( \mathcal{R}_x^c \) the subset of \( \mathcal{R}_x \) containing only rules predicting \( c \). Votes in \( \mathcal{R}_x^c \) have different weights, depending on the confidence of the corresponding rules. Weighted votes for \( c \) are averaged, giving the score for \( c \) with regard to \( x \) (Equation 1).

Finally, the likelihood of \( x \) being a negative example is given by the normalized score (Equation 2).

\[
s(x, c) = \sum \frac{\theta(\mathcal{X} \rightarrow c)}{|\mathcal{R}_x^c|}, \text{ with } c \in \{\ominus, \oplus\} \tag{1}
\]

\[
\alpha(x, \ominus) = \frac{s(x, \ominus)}{s(x, \ominus) + s(x, \oplus)} \tag{2}
\]

Training Projection and Demand-Driven Rule Extraction: Demand-driven rule extraction (Velasco et al., 2006; Santana et al., 2011) is a recent strategy used to avoid the huge search space for rules, by projecting the training corpus according to the example being processed. More specifically, rule extraction is delayed until an example \( x \) is given for classification. Then, terms in \( x \) are used as a filter that configures the training corpus \( \mathcal{D} \) so that just rules that are applicable to \( x \) can be extracted. This filtering process produces a projected training corpus, denoted as \( \mathcal{D}_x \), which contains only terms that are present in \( x \). As shown in (Menezes et al., 2010), the number of rules extracted using this strategy grows polynomially with the size of the vocabulary.

Extending the Classifier Dynamically: With demand-driven rule extraction, the classifier \( \mathcal{R} \) is extended dynamically as examples are given for classification. Initially \( \mathcal{R} \) is empty; a subset \( \mathcal{R}_{x_i} \) is appended to \( \mathcal{R} \) every time an example \( x_i \) is processed. Thus, after processing a sequence of \( m \) examples \( \{x_1, x_2, \ldots, x_m\} \), \( \mathcal{R} = \{\mathcal{R}_{x_1} \cup \mathcal{R}_{x_2} \cup \ldots \cup \mathcal{R}_{x_m}\} \).

Before extracting rule \( \{\mathcal{X} \rightarrow c\} \), it is checked whether this rule is already in \( \mathcal{R} \). In this case, while processing an example \( x \), if an entry is found with a key matching \( \{\mathcal{X}, c\} \), then the rule in \( \mathcal{R} \) is used instead of extracting it from \( \mathcal{D}_x \). Otherwise, the rule is extracted from \( \mathcal{D}_x \) and then it is inserted into \( \mathcal{R} \).

Incremental Updates: Entries in \( \mathcal{R} \) may become invalid when \( \mathcal{D} \) is modified due to a label-transition operation. Given that \( \mathcal{D} \) has been modified, the classifier \( \mathcal{R} \) must be updated properly. We propose to maintain \( \mathcal{R} \) up-to-date incrementally, so that the updated classifier is exactly the same one that would be obtained by re-constructing it from scratch.

Lemma 3.1: Operation \( x^{\ominus \rightarrow \ominus} \) (or \( x^{\ominus \rightarrow \oplus} \)) does not change the value of \( \sigma(\mathcal{X}) \), for any termset \( \mathcal{X} \).

Proof: The operation \( x^{\ominus \rightarrow \ominus} \) changes only the label associated with \( x \), but not its terms. Thus, the number of examples in \( \mathcal{D} \) having \( \mathcal{X} \) as a subset is essentially the same as in \( \{(\mathcal{D} - x^{\ominus}) \cup x^{\ominus}\} \). The same holds for operation \( x^{\ominus \rightarrow \oplus} \).

Lemma 3.2: Operation \( x^{\ominus \rightarrow \ominus} \) (or \( x^{\ominus \rightarrow \oplus} \)) changes the value of \( \sigma(\mathcal{X} \cup c) \) iff termset \( \mathcal{X} \subset x \).

Proof: Comes directly from Lemmas 3.1 and 3.2.

From Lemmas 3.1 to 3.3, the number of rules that have to be updated due to a label-transition operation is bounded by the number of possible termsets in \( x \). The following theorem states exactly the rules in \( \mathcal{R} \) that have to be updated due to a transition operation.

Theorem 3.4: All rules in \( \mathcal{R} \) that must be updated due to \( x^{\ominus \rightarrow \ominus} \) (or \( x^{\ominus \rightarrow \oplus} \)) are those in \( \mathcal{R}_x \).

Proof: From Lemma 3.3, all rules \( \{\mathcal{X} \rightarrow c\} \in \mathcal{R} \) that have to be updated due to operation \( x^{\ominus \rightarrow \ominus} \) (or \( x^{\ominus \rightarrow \oplus} \)) are those for which \( \mathcal{X} \subseteq x \). By definition, \( \mathcal{R}_x \) contains only and all such rules.

Updating \( \theta(\mathcal{X} \rightarrow \ominus) \) and \( \theta(\mathcal{X} \rightarrow \oplus) \) is straightforward. For operation \( x^{\ominus \rightarrow \ominus} \), it suffices to iterate on \( \mathcal{R}_x \), incrementing \( \sigma(\mathcal{X} \cup \ominus) \) and decrementing \( \sigma(\mathcal{X} \cup \oplus) \). Similarly, for operation \( x^{\ominus \rightarrow \oplus} \), it suffices to iterate on \( \mathcal{R}_x \), incrementing \( \sigma(\mathcal{X} \cup \ominus) \) and decrementing \( \sigma(\mathcal{X} \cup \ominus) \). The corresponding values for \( \theta(\mathcal{X} \rightarrow \ominus) \) and \( \theta(\mathcal{X} \rightarrow \oplus) \) are simply obtained by computing \( \frac{\sigma(\mathcal{X} \cup \ominus)}{\sigma(\mathcal{X})} \) and \( \frac{\sigma(\mathcal{X} \cup \ominus)}{\sigma(\mathcal{X})} \), respectively.
3.2 Best Entropy Cut Method

In this section we propose a method for finding the activation threshold, $\alpha_{min}^x$, which is a fundamental step of our Expectation-Maximization approach.

**Definition 3.4:** Let $c^y \in \{\emptyset, \Theta\}$ be the label associated with an example $y \in D_x$. Consider $N_{\emptyset}(D_x)$ the number of examples in $D_x$ for which $c^y = \emptyset$. Similarly, consider $N_{\Theta}(D_x)$ the number of examples in $D_x$ for which $c^y = \Theta$.

**Entropy Minimization:** Our method searches for a threshold $\alpha_{min}^x$ that provides the best entropy cut in the probability space induced by $D_x$. Specifically, given examples $\{y_1, y_2, \ldots, y_k\}$ in $D_x$, our method first calculates $\alpha(y_i, \emptyset)$ for all $y_i \in D_x$. Then, the values for $\alpha(y_i, \Theta)$ are sorted in ascending order. In an ideal case, there is a cut $\alpha_{min}^x$ such that:

$$c^y = \begin{cases} \emptyset & \text{if } \alpha(y_i, \emptyset) \leq \alpha_{min}^x \\ \Theta & \text{otherwise} \end{cases}$$

However, there are more difficult cases, for which it is not possible to obtain a perfect separation in the probability space. Thus, we propose a more general method to find the best cut in the probability space. The basic idea is that any value for $\alpha_{min}^x$ induces two partitions over the space of values for $\alpha(y, \emptyset)$ (i.e., one partition with values that are lower than $\alpha_{min}^x$, and another partition with values that are higher than $\alpha_{min}^x$). Our method sets $\alpha_{min}^x$ to the value that minimizes the average entropy of these two partitions. Once $\alpha_{min}^x$ is calculated, it can be used to activate a label-transition operation. Next we present the basic definitions in order to detail this method.

**Definition 3.5:** Consider a list of pairs $O = \{\ldots, <c^y, \alpha(y_i, \emptyset)>, <c^y, \alpha(y_j, \Theta)>, \ldots\}$, such that $\alpha(y_i, \emptyset) \leq \alpha(y_j, \Theta)$. Also, consider $f$ a candidate value for $\alpha_{min}^x$. In this case, $O_f(\leq)$ is a sub-list of $O$, that is, $O_f(\leq) = \{\ldots, <c^y, \alpha(y_i, \emptyset)>, \ldots\}$, such that for all pairs in $O_f(\leq)$, $\alpha(y, \emptyset) \leq f$. Similarly, $O_f(\geq) = \{\ldots, <c^y, \alpha(y_i, \Theta)>, \ldots\}$, such that for all pairs in $O_f(\geq)$, $\alpha(y, \Theta) > f$. In other words, $O_f(\leq)$ and $O_f(\geq)$ are partitions of $O$ induced by $f$.

Our method works as follows. Firstly, it calculates the entropy in $O$, as shown in Equation 3. Then, it calculates the sum of the entropies in each partition induced by $f$, according to Equation 4. Finally, it sets $\alpha_{min}^x$ to the value of $f$ that minimizes $E(O) - E(O_f)$, as illustrated in Figure 1.

$$E(O) = - \left( \frac{N_{\emptyset}(O)}{|O|} \times \log \frac{N_{\emptyset}(O)}{|O|} \right)$$

$$E(O_f) = \frac{|O_f(\leq)|}{|O|} \times E(O_f(\leq)) + \frac{|O_f(\geq)|}{|O|} \times E(O_f(\geq))$$

3.3 Disambiguation Algorithms

In this section we discuss four algorithms based on our incremental EM approach and following our Best Entropy Cut method. They differ among themselves on the granularity in which the classifier is updated and on the label-transition operations allowed:

- **A1:** the classifier is updated incrementally after each EM iteration (which may comprise several label-transition operations). Only operation $x \Theta \rightarrow \Theta$ is allowed.
- **A2:** the classifier is updated incrementally after each EM iteration. Both operations $x \Theta \rightarrow \Theta$ and $x \Theta \rightarrow \Theta$ are allowed.
- **A3:** the classifier is updated incrementally after each label-transition operation. Only operation $x \Theta \rightarrow \Theta$ is allowed.
- **A4:** the classifier is updated incrementally after each label-transition operation. Both operations $x \Theta \rightarrow \Theta$ and $x \Theta \rightarrow \Theta$ are allowed.
4 Experimental Evaluation

In this section we analyze our algorithms using standard measures such as F-score and AUC values. For each positive+unlabeled (PU) corpus used in our evaluation we randomly selected \(x\%\) of the positive examples (P) to become unlabeled ones (U). This procedure enables us to control the uncertainty level of the corpus. For each level we have a different TPR-FPR combination, enabling us to draw ROC curves, as depicted in Fig. 2. We repeated this procedure five times, so that five executions were performed for each uncertainty level. Our tables show the average for the five runs. Significance tests were performed (\(p<0.05\)) and the best results, including statistical ties, are shown in bold.

4.1 Baselines and Collections

We used the B-SVM algorithm (Liu et al., 2003) as baseline. This algorithm uses a soft-margin SVM as the underlying classifier, which is re-constructed from scratch after each EM iteration. B-SVM employs a single transition threshold \(\alpha_{min}\) for the entire corpus, instead of a different threshold \(\alpha_{min}^x\) for each \(x \in D\). It is representative of the state-of-the-art for learning classifiers from PU data.

We employed two different Twitter collections. The first collection, ORGANIZATIONS, is composed of 10 corpora \(^3\) (O\(_1\) to O\(_{10}\)). Each corpus contains messages in English mentioning the name of an organization (Bayer, Renault, Southwest Airlines, among others). All messages were labeled by five annotators. Label \(\oplus\) means that the message is associated with the organization, whereas label \(\ominus\) means that the message is not related to the organization.

The other collection, SOCCER TEAMS, contains 6 large-scale PU corpora (ST\(_1\) to ST\(_6\)). Each corpus contains messages in Portuguese mentioning the name/mascot of a Brazilian soccer team (Fluminense Football Club, Clube Atlético Mineiro, among others). Both collections are summarized in Table 1.

4.2 Results

All experiments were performed on a Linux PC with an Intel Core 2 Duo 2.20GHz and 4GBytes RAM. Next we discuss the disambiguation performance and the computational efficiency of our algorithms.

\(^3\)http://nlp.uned.es/weps/

![Figure 2: ROC curve for different uncertainty levels.](image)

Table 1: Characteristics of each collection.

|     | P  | U  |
|-----|----|----|
| O\(_1\) | 329 | 119 |
| O\(_2\) | 335 | 133 |
| O\(_3\) | 292 | 148 |
| O\(_4\) | 314 | 143 |
| O\(_5\) | 295 | 172 |
| O\(_6\) | 273 | 165 |
| O\(_7\) | 349 | 116 |
| O\(_8\) | 404 | 55  |
| O\(_9\) | 33  | 425 |
| O\(_{10}\) | 404 | 10  |

Table 2: For each evaluated algorithm we show the corresponding average AUC value (the higher the better).

|     | A1 | A2 | A3 | A4 | B-SVM |
|-----|----|----|----|----|-------|
| O\(_1\) | 0.68 | 0.68 | **0.70** | **0.72** | 0.66  |
| O\(_2\) | **0.71** | **0.72** | **0.71** | **0.72** | **0.71** |
| O\(_3\) | 0.69 | 0.72 | **0.74** | **0.74** | 0.69  |
| O\(_4\) | 0.73 | 0.73 | **0.75** | **0.75** | 0.70  |
| O\(_5\) | 0.65 | 0.68 | 0.69 | **0.72** | 0.63  |
| O\(_6\) | 0.70 | 0.70 | **0.72** | **0.72** | 0.70  |
| O\(_7\) | **0.68** | **0.70** | **0.69** | **0.69** | 0.65  |
| O\(_8\) | 0.77 | 0.78 | 0.70 | **0.82** | 0.75  |
| O\(_9\) | 0.70 | **0.74** | 0.71 | **0.65** | 0.66  |
| O\(_{10}\) | 0.74 | 0.76 | 0.74 | **0.79** | 0.76  |

**ORGANIZATIONS Corpora:** Table 2 shows average AUC values for each algorithm. Algorithm A4 was the best performer in all cases, suggesting the benefits of (i) enabling both types of label-transition operations, and (ii) keeping the classifier
Table 3: For each evaluated algorithm we show the corresponding average AUC value (the higher the better).

|       | A1  | A2  | A3  | A4  | B-SVM |
|-------|-----|-----|-----|-----|-------|
| ST1   | 0.65| 0.67| 0.67| 0.69| 0.64  |
| ST2   | 0.55| 0.58| 0.59| 0.59| 0.57  |
| ST3   | 0.62| 0.63| 0.64| 0.67| 0.61  |
| ST4   | 0.57| 0.59| 0.59| 0.59| 0.55  |
| ST5   | 0.74| 0.74| 0.77| 0.77| 0.72  |
| ST6   | 0.68| 0.70| 0.71| 0.72| 0.68  |

Table 4: For each evaluated algorithm we show the corresponding average execution time, in seconds.

|       | A1   | A2   | A3   | A4   | B-SVM |
|-------|------|------|------|------|-------|
| ST1   | 2,738| 2,795| 3,083| 3,168| 388,184|
| ST2   | 2,086| 2,159| 2,488| 2,516| 297,556|
| ST3   | 1,565| 1,610| 2,102| 2,125| 268,216|
| ST4   | 847  | 883  | 1,199| 1,217| 139,100|
| ST5   | 1,304| 1,372| 1,604| 1,646| 192,576|
| ST6   | 1,369| 1,412| 1,658| 1,700| 196,922|

Table 5: Messages per second (m/sec). For each algorithm we show the corresponding average F-score.

| stream | A4 m/sec | A4 F  | B-SVM m/sec | B-SVM F |
|--------|----------|-------|-------------|---------|
| L1     | 1.8      | 182.2 | 0.59        | 2.0     | 0.55  |
| L2     | 2.6      | 194.3 | 0.82        | 2.1     | 0.74  |
| L3     | 1.3      | 175.3 | 0.71        | 1.6     | 0.58  |
| L4     | 1.6      | 185.8 | 0.89        | 1.4     | 0.85  |
| L5     | 0.8      | 134.6 | 0.73        | 1.3     | 0.62  |
| L6     | 1.2      | 128.6 | 0.71        | 1.6     | 0.62  |

up-to-date after each label-transition operation. Further, algorithm A3 performed better than algorithm A2 in most of the cases, indicating the importance of keeping the classifier always up-to-date. On average A1 provides gains of 4% when compared against B-SVM, while A4 provides gains of more than 20%.

**SOCCER TEAMS Corpora:** Table 3 shows average AUC values for each algorithm. Again, algorithm A4 was the best performer, providing gains that are up to 13% when compared against the baseline. Also, algorithm A3 performed better than algorithm A2, and the effectiveness of Algorithm A1 is similar to the effectiveness of the baseline.

Since the SOCCER TEAMS collection is composed of large-scale corpora, in addition to high effectiveness, another important issue to be evaluated is computational performance. Table 4 shows the results obtained for the evaluation of our algorithms. As it can be seen, algorithm A1 is the fastest one, since it is the simplest one. Even though being slower than algorithm A1, algorithm A4 runs, on average, 120 times faster than B-SVM.

Our last set of experiments emulates a real data streaming scenario. For this purpose, we manually built a separate labeled dataset for each corpus in the ST collection. Each dataset contains 1,000 messages labeled as ⊥ or ⊤ by three annotators. These datasets are identified from L1 to L6, and the rates at which messages come in these datasets are shown in Table 5 (column stream). In this experiment, we use each ST corpus as training data and the corresponding dataset L as test set. Table 5 also shows the number of messages processed by algorithms A4 and B-SVM (to save space and to keep the table more readable, only results obtained by the best algorithm and by the baseline are shown). The F-score for each algorithm is also shown in the table. In this case, the training corpus is first submitted to our EM process, and then it is used as training data, and the F-score is computed using the corresponding test set L.

## 5 Conclusions

In this paper we have introduced a novel EM approach, which employs a highly incremental underlying classifier based on association rules, completely avoiding work replication. Further, two label-transition operations are allowed, enabling the correction of false-negatives and false-positives. We proposed four algorithms based on our EM approach. Our algorithms employ an entropy minimization method, which finds the best transition threshold for each example in D. All these properties make our algorithms appropriate for named entity disambiguation under streaming data scenarios.

Our experiments involve Twitter data mentioning ambiguous named entities. We have shown that three of our algorithms achieve significantly higher disambiguation performance when compared against the baseline, providing gains ranging from 1% to 20%. Also importantly, our algorithms are more than 120 times faster than the baseline.
Acknowledgments

This research was sponsored by UOL (www.uol.com.br) through its UOL Bolsa Pesquisa program, process number 20110215172500, and partially supported by CNPq, CAPES, Finep, and Fapemig.

References

R. Agrawal, T. Imielinski, and A. Swami. 1993. Mining association rules between sets of items in large databases. In SIGMOD, pages 207–216. ACM.

A. Bagga and B. Baldwin. 1998. Entity-based cross-document coreferencing using the vector space model. In ACL, pages 79–85. Association for Computational Linguistics.

R. Bekkerman and A. McCallum. 2005. Disambiguating web appearances of people in a social network. In WWW, pages 463–470. ACM.

I. Bhattacharya and L. Getoor. 2007. Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data, 1.

R. Bunescu and M. Pasca. 2006. Using encyclopedic knowledge for named entity disambiguation. In ACL. Association for Computational Linguistics.

F. De Comité, F. Denis, R. Gilleron, and F. Letouzey. 1999. Positive and unlabeled examples help learning. In ATL, pages 219–230.

S. Cucerzan. 2007. Large-scale named entity disambiguation based on wikipedia data. In EMNLP-CoNLL, pages 708–716.

M. de Carvalho, A. Laender, M. Gonçalves, and A. Soares. 2011. A genetic programming approach to record deduplication. IEEE Trans. on Knowledge and Data Engineering.

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

F. Denis. 1998. Pac learning from positive statistical queries. In ALT, pages 112–126.

X. Dong, A. Halevy, and J. Madhavan. 2005. Reference reconciliation in complex information spaces. In SIGMOD, pages 85–96. ACM.

A. Han and J. Zhao. 2009. Named entity disambiguation by leveraging wikipedia semantic knowledge. In CIKM, pages 215–224. ACM.

B. Jansen, M. Zhang, K. Sobel, and A. Chowdury. 2009. Twitter power: Tweets as electronic word of mouth. JASIST, 60(11):2169–2188.

T. Joachims. 2006. Training linear svms in linear time. In SIGKDD, pages 217–226. ACM.

F. Letouzey, F. Denis, and R. Gilleron. 2000. Learning from positive and unlabeled examples. In ALT, pages 71–85.

X. Li and B. Liu. 2003. Learning to classify texts using positive and unlabeled data. In IJCAI, pages 587–592.

B. Liu, Y. Dai, X. Li, W. Lee, and P. Yu. 2003. Building text classifiers using positive and unlabeled examples. In ICDM, pages 179–188.

G. Menezes, J. Almeida, F. Belém, M. Gonçalves, A. Lacerda, E. de Moura, G. Pappa, A. Veloso, and N. Ziviani. 2010. Demand-driven tag recommendation. In ECML/PKDD, pages 402–417.

E. Minkov, W. Cohen, and A. Ng. 2006. Contextual search and name disambiguation in email using graphs. In SIGIR, pages 27–34. ACM.

T. Pedersen, A. Purandare, and A. Kulkarni. 2005. Name discrimination by clustering similar contexts. In Int'l Conf. on Computational Linguistics and Intelligent Text Processing, pages 226–237.

I. Santana, J. Gomide, A. Veloso, R. Ferreira, and W. Meira Jr. 2011. Learning sentiment streams with training expansion and demand-driven projection. In SIGIR, pages 475–484.

L. Sarmento, A. Kehlenbeck, E. Oliveira, and L. Ungar. 2009. An approach to web-scale named-entity disambiguation. In MLDM, pages 689–703.

A. Veloso, W. Meira Jr., M. de Carvalho, B. Pössas, S. Parthasarathy, and M. Zaki. 2002. Mining frequent itemsets in evolving databases. In SDM.

A. Veloso, W. Meira Jr., and M. Zaki. 2006. Lazy associative classification. In ICDM, pages 645–654. IEEE Computer Society.

A. Wan, J. Gao, M. Li, and B. Ding. 2005. Person resolution in person search results: Webhawk. In CIKM, pages 163–170. ACM.