Multi-Component Word Sense Disambiguation*

Massimiliano Ciaramita **Mark Johnson**
Brown University
Department of Cognitive and Linguistic Sciences
Providence, RI 02912
{massi@brown.edu,mark_johnson@brown.edu}

Abstract

This paper describes the system MC-WSD presented for the English Lexical Sample task. The system is based on a multicomponent architecture. It consists of one classifier with two components. One is trained on the data provided for the task. The second is trained on this data and, additionally, on an external training set extracted from the Wordnet glosses. The goal of the additional component is to lessen sparse data problems by exploiting the information encoded in the ontology.

1 Introduction

One of the main difficulties in word sense classification tasks stems from the fact that word senses, such as Wordnet's synsets (Fellbaum, 1998), define very specific classes[1]. As a consequence training instances are often too few in number to capture extremely fine-grained semantic distinctions. Word senses, however, are not just independent entities but are connected by several semantic relations; e.g., the *is-a*, which specifies a relation of inclusion among classes such as "car is-a vehicle". Based on the is-a relation Wordnet defines large and complex hierarchies for nouns and verbs.

These hierarchical structures encode potentially useful world-knowledge that can be exploited for word sense classification purposes, by providing means for generalizing beyond the narrowest synset level. To disambiguate an instance of a noun like "bat" a system might be more successful if, instead of limiting itself to applying what it knows about the concepts "bat-mammal" and "bat-sport-implement", it could use additional knowledge about other "animals" and "artifacts".

Our system implements this intuition in two steps. First, for each sense of an ambiguous word we generate an additional set of training instances from the Wordnet glosses. This data is not limited to the specific synset that represents one of the senses of the word, but concerns also other synsets that are semantically similar, i.e., close in the hierarchy, to that synset. Then, we integrate the task-specific and the external training data with a multicomponent classifier that simplifies the system for hierarchical word sense disambiguation presented in (Ciaramita et al., 2003). The classifier consists of two components based on the averaged multiclass perceptron (Collins, 2002; Crammer and Singer, 2003). The first component is trained on the task-specific data while the second is trained on the former and on the external training data. When predicting a label for an instance the classifier combines the predictions of the two components. Cross-validation experiments on the training data show the advantages of the multicomponent architecture.

In the following section we describe the features used by our system. In Section 3 we explain how we generated the additional training set. In Section 4 we describe the architecture of the classifier and in Section 5 we discuss the specifics of the final system and some experimental results.

2 Features

We used a set of features similar to that which was extensively described and evaluated in (Yoong and Hwee, 2002). The sentence with POS annotation "A-DT newspaper-NN and-CC now-RB a-DT *bank-NN* have-AUX since-RB taken-VBN over-RB" serves as an example to illustrate them. The word to disambiguate is *bank* (or *activate* for (7)).

1. part of speech of neighboring words $\mathtt{P}_\mathtt{x}$, $\mathtt{x} \in \{-3, -2, -1, 0, +1, +2, +3\}$; e.g., $\mathtt{P}_{-1} = \mathtt{DT}$, $\mathtt{P}_0 = \mathtt{NN}$, $\mathtt{P}_{+1} = \mathtt{AUX}$, ...
2. words in the same sentence WS or passage WC; e.g., $\mathtt{WS} = \mathtt{have}_\mathtt{v}$, $\mathtt{WS} = \mathtt{over}_\mathtt{r}$, $\mathtt{WS} = \mathtt{newspaper}_\mathtt{n}$, ...
3. n-grams:
   - $\mathtt{NG}_\mathtt{x}$, $\mathtt{x} \in \{-2, -1, +1, +2\}$; e.g., $\mathtt{NG}_{-2} = \mathtt{now}$, $\mathtt{NG}_{+1} = \mathtt{have}$, $\mathtt{NG}_{+2} = \mathtt{take}$

* We would like to thank Thomas Hofmann and our colleagues in the Brown Laboratory for Linguistic Information Processing (BLLIP).

[1] 51% of the noun synsets in Wordnet contain only 1 word.

- $\texttt{NG}_{\texttt{x,y}}, (\texttt{x}, \texttt{y}) \in \{(-2,-1),(-1,+1),(+1,+2)\}$; e.g., $\texttt{NG}_{-2,-1} = \texttt{now\_a}$, $\texttt{NG}_{+1,+2} = \texttt{have\_take}$

4. syntactically governing elements under a phrase $\texttt{G}_1$; e.g., $\texttt{G}_1 = \texttt{take\_S}$
5. syntactically governed elements under a phrase $\texttt{G}_2$; e.g., $\texttt{G}_2 = \texttt{a\_NP}$, $\texttt{G}_2 = \texttt{now\_NP}$
6. coordinates $\texttt{OO}$; e.g., $\texttt{OO} = \texttt{newspaper}$
7. features for verbs, e.g, "... *activate* the pressure":
   - number of arguments $\texttt{VN}$; e.g., $\texttt{VN} = 1$
   - syntactic type of arguments $\texttt{VA}$; e.g., $\texttt{VA} = \texttt{NP}$
8. morphology/spelling:
   - prefixes/suffixes up to 4 characters $\texttt{MP}/\texttt{MS}$; e.g., $MP = b, MP = ba, MS = nk, MS = ank$
   - uppercase characters $\texttt{MU}$; e.g., $\texttt{MU} = 0$
   - number/type of word's components $\texttt{MK}/\texttt{MC}$; e.g., $\texttt{MK} = 1, \texttt{MC} = \texttt{bank}$

The same features were extracted from the given test and training data, and the additional dataset. POS and other syntactic features were extracted from parse trees. Training and test data, and the Wordnet glosses, were parsed with Charniak's parser (Charniak, 2000). Open class words were morphologically simplified with the "morph" function from the Wordnet library "wn.h". When it was not possible to identify the noun or verb in the glosses [2] we only extracted a limited set of features: $\texttt{WS}$, $\texttt{WC}$, and morphological features. Each gloss provides one training instance per synset. Overall we found approximately 200,000 features.

3 External training data

There are 57 different ambiguous words in the task: 32 verbs, 20 nouns, and 5 adjectives. For each word w a training set of pairs $(x_i, y_i)_{i=1}^n$, $y_i \in Y(w)$, is generated from the task-specific data; x_i is a vector of features and $Y(w)$ is the set of possible senses for w. Nouns are labeled with Wordnet 1.71 synset labels, while verbs and adjectives are annotated with the Wordsmyth's dictionary labels. For nouns and verbs we used the hierarchies of Wordnet to generate the additional training data. We used the given sense map to map Wordsmyth senses to Wordnet synsets. For adjectives we simply used the task-specific data and a standard flat classifier.[3]

For each noun, or verb, synset we generated a fixed number k of other semantically similar synsets. For each sense we start collecting synsets among the descendants of the sense itself and work our way up the hierarchy following the paths from the sense to the top until we found k synsets. At each level we look for the closest k descendants of the current synset as follows - this is the "closest_descendants()" function of Algorithm 1 above. If there are k or less descendants we collect them all. Otherwise, we take the closest k around the synset exploiting the fact that when ordered, using the synset IDs as keys, similar synsets tend to be close to each other[4]. For example, synsets around "Rhode_Islander" refer to other American states' inhabitants' names:

	Synset_ID	Nouns
	109127828	Pennsylvanian
→	109127914	Rhode_Islander
	109128001	South_Carolinian

Algorithm 1 Find k Closest Neighbors

```
1: input Q = {y}, N_y = ∅, k
2: repeat
3:    u ← head[Q]
4:    CD_u ← closest_descendants(u, k)
5:    for each s ∈ CD_u do
6:       if |N_y| < k then
7:          N_y ← N_y ∪ s
8:       end if
9:    end for
10:   for each v : v is a parent of u do
11:      ENQUE(Q,v)
12:   end for
13:   DEQUE(Q)
14: until |N_y| = k or Q = ∅
```

Algorithm 1 presents a schematic description of the procedure. For each sense y of a noun, or verb, we produced a set N_y of $k = 100$ similar neighbor synsets of y. We label this set with $\hat{y}$, thus for each set of labels $Y(w)$ we induce a set of pseudo-labels $\hat{Y}(w)$.For each synset in N_y we compiled a training instance from the Wordnet glosses. At the end of this process, for each noun or verb, there is an additional training set $(x_i, \hat{y}_i)^m$.

4 Classifier

4.1 Multiclass averaged perceptron

Our base classifier is the multiclass averaged perceptron. The multiclass perceptron (Crammer and Singer, 2003) is an on-line learning algorithm which

[2] E.g., the example sentence for the noun synset *relegation* is "He has been *relegated* to a post in Siberia",

[3] We used Wordnet 2.0 in our experiments using the Wordnet sense map files to map synsets from 1.71 to 2.0.

[4] This likely depends on the fact that the IDs encode the location in the hierarchy, even though we don't know how the IDs are generated.

Algorithm 2 Multiclass Perceptron

1: **input** training data $(x_i, y_i)_{i=1}^n$, $\mathbf{V}$
2: **repeat**
3: **for** $i = 1, ..., n$ **do**
4: $E_i = \{r \in Y : \langle v_r, x_i \rangle > \langle v_{y_i}, x_i \rangle\}$
5: **if** $|E_i| > 0$ **then**
6: $v_{y_i} \leftarrow v_{y_i} + x_i$
7: **for** $r \in E_i$ **do**
8: $v_r \leftarrow v_r - \frac{1}{|E_i|} x_i$
9: **end for**
10: **end if**
11: **end for**
12: **until** no more mistakes

extends to the multiclass case the standard perceptron. It takes as input a training set $(x_i, y_i)_{i=1}^n$, $x_i \in \mathbb{R}^d$, and $y_i \in Y(w)$. In the multiclass perceptron, one introduces a weight vector $v_y \in \mathbb{R}^d$ for every $y \in Y(w)$, and defines H by the so-called winner-take-all rule

$$H(x; \mathbf{V}) = \arg\max_{y \in Y} \langle v_y, x \rangle \,. \quad (1)$$

Here $\mathbf{V} \in \mathbb{R}^{|Y(w)| \times d}$ refers to the matrix of weights, with every column corresponding to one of the weight vectors v_y. The algorithm is summarized in Algorithm 2. Training patterns are presented one at a time. Whenever $H(x_i; \mathbf{V}) \neq y_i$ an update step is performed; otherwise the weight vectors remain unchanged. To perform the update, one first computes the error set E_i containing those class labels that have received a higher score than the correct class:

$$E_i = \{r \in Y : \langle v_r, x_i \rangle > \langle v_{y_i}, x_i \rangle\} \quad (2)$$

We use the simplest case of uniform update weights, $-\frac{1}{|E_i|}$ for $r \in E_i$.

The perceptron algorithm defines a sequence of weight matrices $\mathbf{V}^{(0)}, \ldots, \mathbf{V}^{(n)}$, where $\mathbf{V}^{(i)}$ is the weight matrix after the first i training items have been processed. In the standard perceptron, the weight matrix $\mathbf{V} = \mathbf{V}^{(n)}$ is used to classify the unlabeled test examples. However, a variety of methods can be used for regularization or smoothing in order to reduce the effect of overtraining. Here we used the *averaged perceptron* (Collins, 2002), where the weight matrix used to classify the test data is the average of all of the matrices posited during training, i.e., $\mathbf{V} = \frac{1}{n} \sum_{i=1}^n \mathbf{V}^i$.

4.2 Multicomponent architecture

Task specific and external training data are integrated with a two-component perceptron. The discriminant function is defined as:

$$H(x; \mathbf{V}, \mathbf{M}) = \arg\max_{y \in Y(w)} \lambda_y \langle v_y, x \rangle + \lambda_{\hat{y}} \langle m_{\hat{y}}, x \rangle$$

The first component is trained on the task-specific data. The second component learns a separate weight matrix $\mathbf{M}$, where each column vector represents the set label $\hat{y}$, and is trained on both the task-specific and the additional training sets. Each component is weighted by a parameter λ; here $\lambda_{\hat{y}}$ is simply equal to $1 - \lambda_y$. We experimented with two values for λ_y, namely 1 and 0.5. In the former case only the first component is used, in the latter they are both used, and their contributions are equally weighted.

Algorithm 3 Multicomponent Perceptron

1: **input** $(x_i, y_i)_{i=1}^n$, $\mathbf{V} = 0$,$(x_j, \hat{y_j})_{j=1}^m$, $\mathbf{M} = 0$,
2: **for** $t = 1, ..., T$ **do**
3: train $\mathbf{M}$ on $(x_j, \hat{y_j})_{j=1}^m$ and $(x_i, y_i)_{i=1}^n$
4: train $\mathbf{V}$ on $(x_i, y_i)_{i=1}^n$
5: **end for**

The training procedure for the multicomponent classifier is described in Algorithm 3. This is a simplification of the algorithm presented in (Ciaramita et al., 2003). The two algorithms are similar except that convergence, if the data is separable, is clear in this case because the two components are trained individually with the standard multiclass perceptron procedure. Convergence is typically achieved in less than 50 iterations, but the value for T to be used for evaluation on the unseen test data was chosen by cross-validation. With this version of the algorithm the implementation is simpler especially if several components are included.

4.3 Multilabel cases

Often, several senses of an ambiguous word are very close in the hierarchy. Thus it can happen that a synset belongs to the neighbor set of more than one sense of the ambiguous word. When this is the case the training instance for that synset is treated as a multilabeled instance; i.e., $\hat{y_i}$ is actually a set of labels for x_i, that is, $\hat{y_i} \subset \hat{Y}(w)$. Several methods can be used to deal with multilabeled instances, here we use a simple generalization of Algorithm 2. The error set for a multilabel training instance is defined as:

$$E_i = \{r \in Y : \exists y \in y_i, \langle v_r, x_i \rangle > \langle v_y, x_i \rangle\} \quad (3)$$

which is equivalent to the definition in Equation 2 when $|y_i| = 1$. The positive update of Algorithm 2 (line 6) is also redefined. The update concerns a set

word	$\lambda_y = 1$	$\lambda_y = 0.5$	word	$\lambda_y = 1$	$\lambda_y = 0.5$	word	$\lambda_y = 1$	$\lambda_y = 0.5$
appear	86.1	85.5	audience	84.8	86.8	encounter	72.9	75.0
arm	85.9	87.5	bank	82.9	82.1	watch	77.1	77.9
ask	61.9	62.7	begin	57.0	61.5	hear	65.6	68.7
lose	53.1	52.5	eat	85.7	85.0	party	77.1	79.0
expect	76.6	75.9	mean	76.5	77.5	image	66.3	67.8
note	59.6	60.4	difficulty	49.2	54.2	write	68.3	65.0
plan	77.2	78.3	disc	72.1	74.1	paper	56.3	57.7

Table 1. Results on several words from the cross-validation experiments on the training data. Accuracies are reported for the best value of T, which is then chosen as the value for the final system, together with the value λ_y that performed better. On most words the multicomponent model outperforms the flat one

of labels $Y_i \subseteq \hat{Y}(w)$ such that there are incorrect labels wich achieved a better score; i.e., $Y_i = \{y \in y_i : \exists r \notin y_i, \langle v_r, x_i \rangle > \langle v_y, x_i \rangle\}$. For each $y \in Y_i$ the update is equal to $+\frac{1}{|Y_i|}$, which, again, reduces to the former case when $|Y_i| = 1$.

5 Results

Table 1 presents results from a set of experiments performed by cross-validation on the training data, for several nouns and verbs. For 37 nouns and verbs, out of 52, the two-component model was more accurate than the flat model[5]. We used the results from these experiments to set, separately for each word, the parameters T, which was equal to 13.9 on average, and λ_y. For adjectives we only set the parameter T and used the standard "flat" perceptron. For each word in the task we separately trained one classifier. The system accuracy on the unseen test set is summarized in the following table:

Measure	Precision	Recall
Fine all POS	71.1	71.1%
Coarse all POS	78.1	78.1%
Fine verbs	72.5	72.5%
Coarse verbs	80.0	80.0%
Fine nouns	71.3	71.3%
Coarse nouns	77.4	77.4%
Fine adjectives	49.7	49.7%
Coarse adjectives	63.5	63.5%

Overall the system has the following advantages over that of (Ciaramita et al., 2003). Selecting the external training data based on the most similar k synsets has the advantage, over using supersenses, of generating an equivalent amount of additional data for each word sense. The additional data for each synset is also more homogeneous, thus the model should have less variance[6]. The multicomponent architecture is simpler and has an obvious convergence proof. Convergence is faster and training is efficient. It takes less than one hour to build and train all final systems and generate the complete test results. We used the averaged version of the perceptron and introduced an adjustable parameter λ to weigh each component's contribution separately.

[5] Since λ_y is an adjustable parameter it is possible that, with different values for λ_y, the multicomponent model would achieve even better performances.

[6] Of course the supersense level, or any other level, can simply be added as an additional component.

References

E. Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000)*.

M. Ciaramita, T. Hofmann, and M. Johnson. 2003. Hierarchical Semantic Classification: Word Sense Disambiguation with World Knowledge. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 1–8.

K. Crammer and Y. Singer. 2003. Ultraconservative Online Algorithms for Multiclass Problems. *Journal of Machine Learning Research*, 3.

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

K.L Yoong and T.N. Hwee. 2002. An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*.