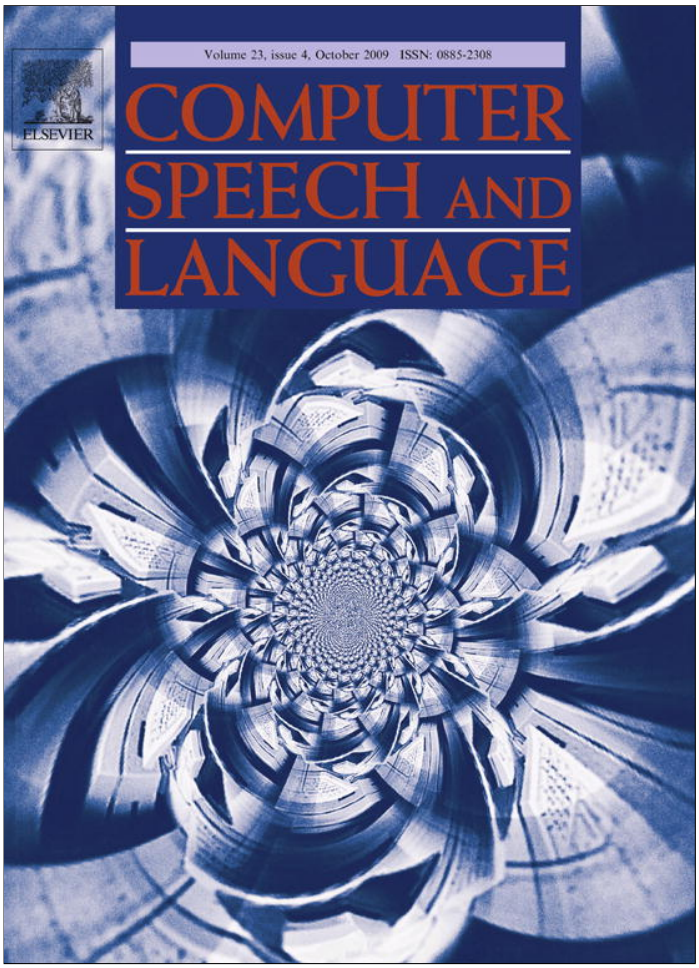

Available online at www.sciencedirect.com
ScienceDirect
Computer Speech and Language 23 (2009) 464–478
COMPUTER SPEECH AND LANGUAGE
www.elsevier.com/locate/csl

Label propagation via bootstrapped support vectors for semantic relation extraction between named entities

Zhou GuoDong *, Qian LongHua, Zhu QiaoMing

Jiangsu Provincial Key Lab for Computer Information Processing Technology, School of Computer Science and Technology, Soochow Univ. 1 ShiZi Street, Suzhou 215006, China

Received 5 March 2008; received in revised form 15 October 2008; accepted 3 March 2009
Available online 18 March 2009

Abstract

This paper proposes a semi-supervised learning method for semantic relation extraction between named entities. Given a small amount of labeled data, it benefits much from a large amount of unlabeled data by first bootstrapping a moderate number of weighted support vectors from all the available data through a co-training procedure on top of support vector machines (SVM) with feature projection and then applying a label propagation (LP) algorithm via the bootstrapped support vectors and the remaining hard unlabeled instances after SVM bootstrapping to classify unseen instances. Evaluation on the ACE RDC corpora shows that our method can integrate the advantages of both SVM bootstrapping and label propagation. It shows that our LP algorithm via the bootstrapped support vectors and hard unlabeled instances significantly outperforms the normal LP algorithm via all the available data without SVM bootstrapping. Moreover, our LP algorithm can significantly reduce the computational burden, especially when a large amount of labeled and unlabeled data is taken into consideration.

Keywords: Semi-supervised learning; Semantic relation extraction; Bootstrapped support vectors; SVM bootstrapping; Label propagation

1. Introduction

With the dramatic increase in the amount of textual information available in digital archives and WWW, there has been a growing interest in information extraction (IE), which attempts to identify relevant information (usually of pre-defined types) from text documents in a certain domain and put them in a structured format. According to the scope of the NIST automatic content extraction (ACE, 2000–2005) program, current research in IE has three main objectives: entity detection and tracking (EDT), relation detection and characterization (RDC), and event detection and characterization (EDC). The EDT task entails the detection of entity mentions and chaining them together by identifying their coreference. In the ACE vocabulary, entities

* Corresponding author. Tel.: +86 13402679766; fax: +86 512 65241071.
E-mail addresses: gdzhou@suda.edu.cn (Z. GuoDong), qianlonghua@suda.edu.cn (Q. LongHua), qmzhu@suda.edu.cn (Z. QiaoMing).

0885-2308/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.csl.2009.03.001

are objects in the world, mentions are references to them, and relations are semantic relationships between entities. Entities can be of five types: persons, organizations, locations, facilities and geo-political entities (GPE: geographically defined regions that indicate a political boundary, e.g. countries, states, cities, etc.). Mentions have three levels: names, nomial expressions or pronouns.

This paper will focus on semantic relation extraction between named entities, in particular, the ACE RDC task, which detects and classifies implicit and explicit relations[1] between entities identified by the EDT task. For example, we want to determine whether a person is at a location, based on the evidence in the context. Normally, RDC can be either done in one step or separated into two steps: relation detection, which identifies whether there exists a relationship between two entities, and relation classification, which classifies the identified relationship into a specific semantic relation. In the ACE task, various relations are defined in a hierarchical way. For example, ACE 2003 defines five relations types, including AT, NEAR, PART, ROLE and SOCIAL, which are further separated into 24 relation sub-types, including AT.BASED-IN, AT.LOCATED and AT.RESIDENCE. Extraction of semantic relationships between entities can be very useful for applications such as question answering (Hirschman and Gaizauskas, 2001), e.g. to answer the query "who is the president of the United States?", and information retrieval (Strzalkowski and Carballo, 1998; Gao et al., 2004), e.g. to expand the query "George W. Bush" with "the president of the United States" via his relationship with the country "the United States".

During the last decade, much research has been done to address the relation extraction task and can be classified into three categories: supervised learning (Zelenko and Aone, 2003; Culotta and Sorensen, 2004; Zhao and Grishman, 2005; Zhang et al., 2006; Zhou et al., 2005, 2006, 2007; Maslennikov and Chua, 2007), semi-supervised learning (Hearst, 1992; Brin, 1998; Agichtein and Gravano, 2000; Zhang, 2004; Etzioni et al., 2005; Chen et al., 2006; Xu et al., 2007), and unsupervised learning (Hasegawa et al., 2004; Zhang et al., 2005). Among them, supervised learning-based methods have been shown to be effective and perform much better than the other two alternatives. However, their performance much depends on the availability of a large amount of manually labeled high-quality data and annotating large corpora with relation instances is expensive and tedious. On the other hand, unsupervised learning-based methods normally perform very poorly, though they do not depend on the availability of any manually labeled data. To achieve better balance between human efforts and extraction performance, semi-supervised learning has been drawing more and more attention recently in semantic relation extraction and other NLP applications as well.

This paper proposes a semi-supervised learning method for semantic relation extraction between named entities. Given a small amount of labeled data, our proposed method benefits much from the availability of a large number of unlabeled data in two phases. First, a moderate number of weighted support vectors are bootstrapped from all the available labeled and unlabeled data using a co-training procedure on top of support vector machines (SVM) with feature projection. Then, a label propagation (LP) algorithm is applied to classifying unseen instances by modeling the natural clustering structure inherent in both the labeled and unlabeled data via the bootstrapped support vectors and the remaining hard unlabeled instances after SVM bootstrapping. Compared with previous ones, our method can integrate the advantages of both SVM bootstrapping in learning critical instances for the labeling function and label propagation in modeling the natural clustering structure inherent in both the labeled and unlabeled data to smooth the labeling function.

The rest of this paper is as follows. In Section 2, we review related semi-supervised learning work in semantic relation extraction. Then, the LP algorithm with SVM bootstrapping is proposed in Section 3 while Section 4 shows the experimental results. Finally, we conclude our work in Section 5.

2. Related work

Since the labeled data is usually expensive to obtain, there has been a growing interest in semi-supervised learning, aiming at inducing classifiers by leveraging a small amount of labeled data and a large amount of unlabeled data. Related work in semantic relation extraction using semi-supervised learning can be mainly

[1] In ACE (http://www.ldc.upenn.edu/Projects/ACE), explicit relations occur in text with explicit evidence suggesting the relationships. Implicit relations need not have explicit supporting evidence in text, though they should be evident from a reading of the document.

classified into two categories: bootstrapping-based (Hearst, 1992; Brin, 1998; Agichtein and Gravano, 2000; Zhang, 2004; Etzioni et al., 2005; Xu et al., 2007) and LP (label propagation)-based (Chen et al., 2006).

Currently, bootstrapping-based methods dominate semi-supervised learning in semantic relation extraction. Generally, they work by iteratively classifying unlabeled instances and adding confidently classified ones into the labeled data using a model learnt from the augmented labeled data in the previous loop. As a pioneer, Hearst (1992) used a small set of seed patterns in a bootstrapping fashion to mine pairs of hypernym–hyponym nouns. Brin (1998) proposed a bootstrapping-based method on top of a self-developed pattern matching-based classifier to exploit the duality between patterns and relations. Agichtein and Gravano (2000) shared much in common with Brin (1998). It employed an existing pattern matching-based classifier (i.e., SNoW as proposed in Carlson et al. (1999)) instead. Zhang (2004) approached relation classification by bootstrapping on top of SVM. For a given target relation, Etzioni et al. (2005) bootstrapped a rule template containing words that describe the class of the arguments (e.g. "the company") and a small set of seed patterns (e.g. "has acquired"). Xu et al. (2007) bootstrapped from a small set of *n*-ary relation instances as seeds to automatically learn pattern rules from parsed data, using a bottom-up pattern extraction method with a new rule representation composed on top of the rules for projections of the relation. Although bootstrapping-based methods have achieved certain success in the literature, one problem is that they may not be able to well capture the inherent natural clustering structure among the unlabeled data.

As an alternative to the bootstrapping-based methods, Chen et al. (2006) employed a LP-based method in semantic relation extraction. Compared with bootstrapping, the LP algorithm can effectively exploit the natural clustering structure in both the labeled and unlabeled data. The rationale behind this algorithm is that the instances in high-density areas tend to carry the same labels. The LP algorithm has also been successfully applied in other NLP applications, such as word sense disambiguation (Niu et al., 2005), text classification (Szummer and Jaakkola, 2001; Blum and Chawla, 2001; Belkin and Niyogi, 2002; Zhu and Ghahramani, 2002; Zhu et al., 2003; Blum et al., 2004), and information retrieval (Yang et al., 2006). However, one problem is its huge computational burden, especially when a large amount of labeled and unlabeled data is taken into consideration.

Besides, Bunescu and Mooney (2007) learnt to extract relations from the Web using minimal supervision by extracting bags of sentences containing the pairs, given a few pairs of relation instances (both positive and negative). It transformed this weakly-labeled multiple instance learning problem into a standard supervised problem by properly controlling the relative influence of false negative vs. false positive and eliminating two types of bias due to words that are correlated with the arguments of a relation instance and those that are specific to a relation instance. Therefore, the minimal supervision proposed by Bunescu and Mooney (2007) actually employed a supervised learning method.

In order to take the advantages of both bootstrapping and label propagation, our proposed method propagates labels via bootstrapped support vectors and the remaining hard unlabeled instances after SVM bootstrapping. Evaluation on the ACE RDC corpora shows that our method can not only significantly reduce the computational burden in the normal LP algorithm via all the available data but also well capture the natural clustering structure inherent in both the labeled and unlabeled data via the bootstrapped support vectors and hard unlabeled instances only.

3. Label propagation with SVM bootstrapping

The idea behind our LP algorithm with SVM bootstrapping is that, instead of propagating labels through all the available labeled and unlabeled data, our method propagates labels only through critical instances in both the labeled and unlabeled data. Similar to support vectors in SVM, critical instances in this paper refer to the support vectors in label propagation, which play a critical role in propagating labels. Fig. 1 shows a brief flow chart in training and testing of our LP algorithm with SVM bootstrapping. The key behind our LP algorithm is how to find critical instances in both the labeled and unlabeled data.

In this paper, we use SVM as the underlying classifier to bootstrap a moderate number of weighted support vectors for this purpose. This is based on an assumption that the natural clustering structure in both the labeled and unlabeled data can be well preserved through the critical instances, including the weighted support

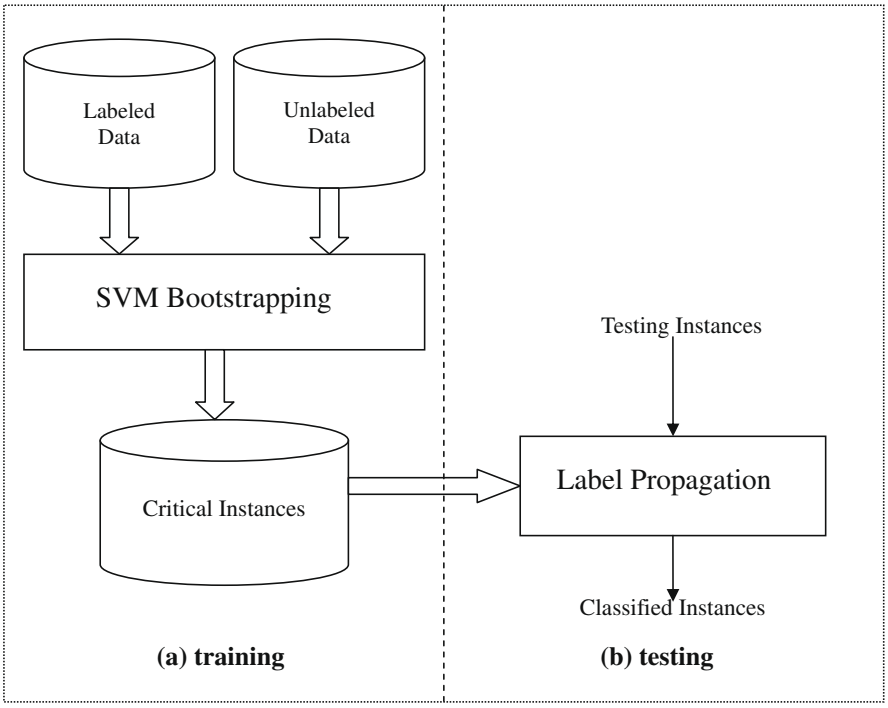

Fig. 1. The architecture of our LP algorithm with SVM bootstrapping.

vectors bootstrapped from all the available labeled and unlabeled data and those hard unlabeled instances (i.e., those remaining instances in the unlabeled data after SVM bootstrapping). The main reason why we choose SVM is that it represents the state-of-the-art in machine learning research and there are good implementations available. Another reason is that we can adopt the weighted support vectors returned by the bootstrapped SVMs plus the remaining hard unlabeled instances as the critical instances, via which label propagation is done. In particular, the binary SVM-Light (Joachims, 1998) is selected as our classifier. For efficiency, we apply the one vs. others strategy, which builds K classifiers so as to separate one class from all others, to handle the multi-category classification problem in semantic relation extraction. Moreover, K sets of weighted support vectors plus hard unlabeled instances are bootstrapped as the critical instances, one for each category (including all the relation categories plus the non-relation in semantic relation extraction), and applied in a label propagation procedure. Finally, the label with the largest LP value is selected as the final output.

3.1. SVM bootstrapping

We modify the SVM bootstrapping algorithm BootProject (Zhang, 2004) to bootstrap weighted support vectors using a co-training procedure. Given a small amount of labeled data for each category and a large amount of unlabeled data, the modified BootProject algorithm bootstraps on top of the binary SVM-Light by iteratively classifying unlabeled instances using classifiers learnt from the augmented labeled data in the previous loop and moving confidently classified ones to the labeled data until no enough unlabeled instances (fine-tuned to 10 in this paper) can be classified confidently. Moreover, this algorithm generates multiple overlapping "views" by projecting from the original feature space and multiple classifiers are trained on the augmented labeled data corresponding to the multiple projected views to vote on the remaining unlabeled instances and those with the highest probability of being correctly labeled (with the classifier agreement of more than a threshold (fine-tuned to 70% in this paper), and the absolute SVM confidence value on average bigger than a threshold (fine-tuned to 1.0 in this paper) are chosen to augment the labeled data with a weight λ^{i} (λ is the decay factor and fine-tuned to 0.9 while i is the number of loops). Please note that all the hyper-parameters, including those above, are fine-tuned using 2-fold cross-validation on the ACE RDC 2003 training data.

Assume:
L: labeled data;
U: unlabeled data;
S: batch size (100 in our experiments);
P: number of views(feature projections);
r: number of classes (including all the relation (sub)types plus the non-relation)

BEGIN
REPEAT
FOR i = 1 to P DO
Generate projected feature space F_i from the original feature space F;
Project both L and U onto F_i, thus generate L_i and U_i;
Train SVM classifier SVM_{ij} on L_i for each class $r_j (j = 1 \ldots r)$;
Run SVM_{ij} on U_i for each class $r_j (j = 1 \ldots r)$
END FOR
Find (at most) S instances in U with the highest agreement (with threshold 70% in our experiments) and the highest average SVM-returned confidence value (with threshold 1.0 in our experiments);
Move them from U to L;
UNTIL no enough unlabeled instances (less than 10 in our experiments) can be confidently classified;
Return all the (positive and negative) support vectors included in all the latest SVM classifiers SVM_{ij} with their respective weight (absolute $a \times y$) information as the set of bootstrapped support vectors to act as the labeled data in the LP algorithm;
Return U (the hard cases which can not be confidently classified) to act as the unlabeled data in the LP algorithm;
END

Fig. 2. The algorithm for bootstrapping support vectors.

Fig. 2 shows the modified BootProject algorithm for bootstrapping support vectors. During the bootstrapping process, the support vectors included in all the SVM classifiers are bootstrapped (i.e. updated) at each loop. When the bootstrapping process stops, all the bootstrapped (positive and negative) support vectors included in the SVM classifiers are returned with their respective weights (absolute $a \times y$)[2] to act as the labeled data in the LP algorithm and all the remaining hard unlabeled instances (i.e. those hard cases which can not be confidently classified in the bootstrapping process) in the unlabeled data are returned to act as the unlabeled data in the LP algorithm. Through SVM bootstrapping, our LP algorithm will only depend on the critical instances (i.e. weighted support vectors, bootstrapped from all the available data, and the hard unlabeled instances, remaining in the unlabeled data after SVM bootstrapping), instead of all the available labeled and unlabeled data.

In this paper, two types of views are explored: flat feature views and tree views.

3.1.1. Flat feature views

We employ the same feature set as described in a state-of-the-art feature-based system (Zhou et al., 2005):

(1) words in the context of the two mentions: according to their positions, words are separated into four categories: the words in the two mentions, the words between the two mentions, the words before the first mention, and the words after the second mention,
(2) entity types of both the mentions, which can be PERSON, ORGANIZATION, FACILITY, LOCATION and GPE (i.e. geo-political entity) in the ACE RDC 2003 corpus,
(3) mention levels of the two mentions, which can be NAME, NOMIAL and PRONOUN,
(4) overlap relationship between the two mentions,
(5) base phrase chunking: the head words of the phrases between the two mentions and the phrase path in between,

[2] a is the weight of support vector and $y = 1$ or -1 indicating positive or negative support vectors.

(6) dependency parsing: information about the words, part-of-speeches and phrase labels of the words on which the mentions are dependent in the dependency tree,
(7) syntactic parsing: the phrase path information between the two mentions in the parse tree,
(8) semantic information: semantic information from various resources, such as WordNet, is used to classify important words into different semantic lists according to their indicating relationships.

Given above various lexical, syntactic and semantic features, multiple overlapping feature views are generated using random feature projection (Zhang, 2004). For each feature projection in bootstrapping support vectors, a feature is randomly selected with probability p and therefore the eventually projected feature space has $p * F$ features on average, where F is the size of the original feature space. In this paper, p and the number of different views are fine-tuned to 0.5 and 11, respectively[3] using 2-fold cross-validation on the ACE RDC 2003 training data.

Due to superior performance, a polynomial kernel ($d = 2$) is applied to computing the similarity between two feature vectors in this paper instead of the linear kernel, as applied in Zhou et al. (2005).

3.1.2. Tree views

We explore seven kinds of tree views as our feature projections, including five parse tree spaces proposed in Zhang et al. (2006) and two more chunking tree spaces:

(1) Minimum complete tree (MCT): the complete sub-tree rooted by the nearest common ancestor of the two considered entities.
(2) Path-enclosed tree (PT): the smallest common sub-tree including the two entities. In other words, the sub-tree is enclosed by the shortest path linking the two entities in the parse tree.
(3) Chunking tree (CT): the base phrase list extracted from the PT by pruning out all the internal structures of the PT while only keeping the root node and the base phrase list for generating the chunking tree.
(4) Context-sensitive path tree (CPT): the PT extended with the 1st left sibling of the node of entity 1 and the 1st right sibling of the node of entity 2. If the sibling is unavailable, it moves to the parent of current node and repeats the same process until the sibling is available or the root is reached.
(5) Context-sensitive chunking tree (CCT): the CT extended with the 1st left sibling of the node of entity 1 and the 1st right sibling of the node of entity 2. If the sibling is unavailable, the same process as generating the CPT is applied. Then we do a further pruning process to guarantee that the context structure in the CCT is still a list of base phrases.
(6) Flattened PT (FPT): the PT with the single in-and-out arcs of non-terminal nodes (except POS nodes), and the non-terminal nodes having the same syntactic phrase type with their father removed.
(7) Flattened CPT (FCPT): the CPT with the single in-and-out arcs of non-terminal nodes (except POS nodes), and the non-terminal nodes having the same syntactic phrase type with their father removed.

In this paper, a context-sensitive convolution tree kernel, similar to the one proposed in Zhou et al. (2007), is applied as our tree kernel to computing the similarity between two trees:

$$K_{CTK}(T[1], T[2]) = \sum_{i=1}^{m} w_i \sum_{\substack{n_1^i[1] \in N_1^i[1] \\ n_1^i[2] \in N_1^i[2]}} \Delta(n_1^i[1], n_1^i[2]) \quad (1)$$

where

(1) $n_1^i[j] = (n_1 n_2, \ldots, n_i)[j]$ is a root node path with length i in tree $T[j]$ taking into account the $i-1$ ancestral nodes $n_2^i[j]$ of $n_i[j]$ in $T[j]$.

[3] This suggests that the modified BootProject algorithm in the bootstrapping phase outperforms the SelfBoot algorithm (with $p = 1.0$ and $m = 1$) which uses all the features as the only view. In the NLP literature, co-training has also shown to typically outperform self-bootstrapping.

(2) $\Delta(n_1^i[1], n_1^i[2])$ measures the common context-sensitive sub-trees with root node paths $n_1^i[1]$ and $n_1^i[2]$[4], m defines the maximal length of a root node path and w_i is the weight for a context-sensitive sub-tree with a root node path of length i.

Obviously, this tree kernel takes the ancestral information (i.e. the root node path) of sub-trees into consideration and $\Delta(n_1^i[1], n_1^i[2])$ is computed recursively as follows:

(1) If the context-sensitive productions (context-sensitive grammar (CSG) rules with root node paths as their left hand sides) rooted at $n_1^i[1]$ and $n_1^i[2]$ are different, return $\Delta(n_1^i[1], n_1^i[2]) = 0$; Otherwise go to Step 2.
(2) If both$n_1^i[1]$ and $n_1^i[2]$ are POS tags, $\Delta(n_1^i[1], n_1^i[2]) = \lambda$; Otherwise go to Step 3.
(3) Calculate $\Delta(n_1^i[1], n_1^i[2]) = 0$, recursively as:

$$\Delta(n_1^i[1], n_1^i[2]) = \lambda \prod_{k=1}^{\#ch(n_1^i[1])} (1 + \Delta(ch(n_1^i[1], k), ch(n_1^i[2], k)) \quad (2)$$

where $ch(n_1^i[j], k)$ is the kth context-sensitive child of the context-sensitive sub-tree rooted at $n_1^i[j]$ with $\#ch(n_1^i[j])$ the number of the context-sensitive children. Here, λ ($0 < \lambda < 1$) is the decay factor in order to make the kernel value less variable with respect to different sizes of context-sensitive sub-trees.

For both fair comparison and convenience, the similarity between two original tree spaces is computed by linearly interpolating the seven individual tree kernels using above seven tree spaces (i.e. tree views) with equal weights.

3.2. Label propagation

In the LP algorithm (Zhu and Ghahramani, 2002), the labeled and unlabeled data are represented as a connected graph. Given the labeled data (e.g. the above bootstrapped support vectors with their weights) and the unlabeled data (e.g. the remaining hard unlabeled instances after SVM bootstrapping plus all the unseen test instances for evaluation), the LP algorithm first represents labeled and unlabeled instances as vertices in a connected graph, then propagates the label information from any vertex to nearby vertex through weighted edges and finally infers the labels of unlabeled instances until a global stable stage is achieved.

Fig. 3 presents the label propagation algorithm. Here, each vertex corresponds to an instance, and the edge between any two instances x_i and x_j is weighted by w_{ij} to measure their similarity. In principle, larger edge weights allow labels to travel through easier. Thus the closer the instances are, the more likely they have similar labels. The algorithm first calculates the weight w_{ij} using a kernel (e.g. either a polynomial kernel between two feature vectors or a tree kernel between two trees), then transforms it to $t_{ij} = p(j \rightarrow i) = w_{ij} / \sum_{k=1}^{n} w_{ij}$, which measures the probability of propagating a label from instance x_j to instance x_i, and finally normalizes t_{ij} row by row using $\bar{t}_{ij} = t_{ij} / \sum_{k=1}^{n} t_{ik}$ to maintain the class probability interpretation of the labeling matrix Y.

During the label propagation process, the label distribution of the labeled data is clamped in each loop using the normalized weights of the bootstrapped support vectors and acts like forces to push out labels through the unlabeled data. With this push originates from the labeled data, the label boundaries will be pushed much faster along edges with larger weights and settle in gaps along those with lower weights. Ideally, we can expect that w_{ij} across different classes should be as small as possible and w_{ij} within the same class as big as possible. In this way, label propagation happens within the same class most likely.

This algorithm has been shown to converge to a unique solution, and the initialization of the unlabeled data is not important since it does not affect its estimation (Zhu and Ghahramani, 2002). However, the initialization of the unlabeled data actually helps the algorithm converge more rapidly in practice. In this paper, each row in the unlabeled data is initialized to the average similarity with the labeled instances (normalized).

[4] That is, each root node path n_1^i encodes the identity of a context-sensitive sub-tree rooted at n_1^i and, if there are two root node paths in the tree with the same label sequence, the summation will go over both of them.

Assume:

Y: the $n*r$ labeling matrix, where y_{ij} represents the probability of vertex $x_i (i = 1 \ldots n)$ with label $r_j (j = 1 \ldots r)$ (including the non-relation label);

Y_L: the top l rows of Y^0. Y_L corresponds to the l labeled instances;

Y_U: the bottom u rows of Y^0. Y_U corresponds to the u unlabeled instances;

$\overline{T}$: a $n*n$ matrix, with $\bar{t}_{ij}$ is the probability of jumping from vertex x_i to vertex x_j;

BEGIN (the algorithm)

Initialization:

1) Set the iteration index $t = 0$;

2) Let Y^0 be the initial soft labels attached to each vertex;

3) Let Y_L^0 be consistent with the labeling in the labeled (including all the relation types/sub-types plus the non-relation) data, where y_{ij}^0 = the normalized weight[6] of the bootstrapped support vector if x_i has label r_j (Please note that r_j can be the non-relation label) and 0 otherwise;

4) Initialize Y_U^0;

REPEAT

Propagate the labels of any vertex to nearby vertices by $Y^{t+1} = \overline{T} Y^t$;

Clamp the labeled data, that is, replace Y_L^{t+1} with Y_L^0;

UNTIL Y converges(e.g. Y_L^{t+1} converges to Y_L^0);

Assign each unlabeled instance with a label: for $x_i (l \prec i \leq n)$, find its label with $\arg\max_j y_{ij}$;

END (the algorithm)

Fig. 3. The LP algorithm. [6] In SVM, the weight of a support vector measures its importance to the separating hyperplane. Assume svw_{ij} is the weight of a bootsrapped support vector x_i for label r_j, then y_{ij}^0 is initialized to $suw_{ij}/\sum_j suw_{ij}$. In this way, y_{ij}^0 can be treated as a probability.

4. Experimentation

This paper uses the ACE RDC 2003 and 2004 corpora provided by LDC for evaluation. The ACE corpora are gathered from various newspapers, newswires and broadcasts.

4.1. Experimental setting

In the ACE RDC 2003 corpus, the training data consists of 674 annotated text documents (~300 k words) and 9683 instances of relations while the test data consists of 97 documents (~50 k words) and 1386 instances of relations. The ACE RDC 2003 task defines five relation types, which are further separated into 24 relation subtypes, between five entity types, i.e. person, organization, location, facility and GPE. Compared with the ACE RDC 2003 task, the ACE RDC 2004 task defines two more entity types (i.e. weapon and vehicle) and seven relation types, which are further separated into 23 relation subtypes between the seven entity types. The ACE RDC 2004 corpus from LDC contains 451 documents and 5702 relation instances. For evaluation, we randomly choose 361 documents, which contain 4553 relation instances, as the training data and the remaining 90 documents, which contain 1149 relation instances as the test data. Furthermore, all the hyper-parameters in this paper are fine-tuned using 2-fold cross-validation on the training data of the ACE RDC 2003 corpus.

Similar to previous work in the literature, such as Zhou et al. (2005) and Zhang et al. (2006), we iterate over all pairs of entity mentions occurring in the same sentence to generate potential relation instances. In addition, we only measure the performance of semantic relation extraction on "true" mentions with "true" chaining of coreference (i.e. as annotated by the corpus annotators) in both corpora. All the sentences in both corpora are parsed using the Charniak parser (Charniak, 2001) with the boundaries of all the entity mentions kept. Finally, we explicitly model the argument order of the two mentions involved. For example, when comparing two mentions m1 and m2, we distinguish between m1-ROLE.Citizen-Of-m2 (m1 occurs before m2) and m2-ROLE.

Citizen-Of-m1 (m2 occurs before m1). Note that, in the ACE RDC 2003 task, 6 of these 24 relation subtypes are symmetric. In this way, we model relation extraction in the ACE RDC 2003 task as a multi-category classification task with 43 (24X2-6 + 1) classes, two for each relation subtype (except the symmetric subtypes) plus a "NONE" class for the case where the two mentions are not related. For the ACE RDC 2004 task, 6 of these 23 relation subtypes are symmetric. In this way, we model relation extraction in the ACE RDC 2004 task as a multi-category classification task with 41 (23X2-6 + 1) classes, two for each relation subtype (except the six symmetric subtypes) plus a "NONE" class for the case where the two mentions are not related. In particular, the RDC task is done in one step instead of separating into two steps containing relation detection and relation classification.

Finally, we mainly report our performance on the relation detection and classification of the 24 and 23 relation subtypes of the ACE RDC 2003 and 2004 corpora, respectively, using F1-measure and the difference between precision and recall (for saving space and revealing more characteristics on the ACE RDC 2003 and 2004 corpora). When necessary, we also present the performance on the relation detection task and the RDC task on the five and seven relation types of the ACE RDC 2003 and 2004 corpora. For computational burden, we also compares the training and testing time of different settings on a PC with a 2.13 GHz Intel Core 2 CPU and 2.0 GB memory. To see whether an improvement is significant, we also conduct significance testing using paired *t*-testing. In this paper, '$***$', '$**$' and '$*$' denote *p*-values of an improvement smaller than 0.01, in-between (0.01, 0,05) and bigger than 0.05, which mean significantly better, moderately better and slightly better, respectively.

4.2. Experimentation and discussion

The experimentation is organized in the following order.

(1) Evaluating the LP algorithm over different kinds of labeled data without any help from unlabeled data in the learning process to show the impact of different kinds of labeled data.
(2) Evaluating the LP algorithm over different kinds of labeled data with the help from unlabeled data in the learning process to compare different ways in incorporating unlabeled data.
(3) Comparing our LP algorithm with other state-of-the-art machine learning methods to validate the advantages of our LP algorithm.
(4) Comparing our LP algorithm over different kernels to evaluate the complementary nature of different kernels on our LP algorithm.
(5) Comparing our LP algorithm with other state-of-the-art machine learning methods on computational burden.

Table 1 presents the performance of the LP algorithm over different kinds of labeled data using the state-of-the-art polynomial kernel without any help from unlabeled data. Here, the support vectors are returned by SVM over the whole training data of each ACE corpus. For fair comparison, K-means (i.e. K cluster centriods) are determined using the K-means clustering algorithm, which uses the most common iterative refinement heuristic (known as Lloyd's algorithm) and starts by randomly partitioning the input points into *k* initial sets, on both positive and negative training data with Ks equal to the numbers of positive and negative SVs (support vectors), respectively, and the weight of each mean equal to the number of instances belonging to the cluster. Table 1 shows that

(1) LP over weighted SVs performs best. It shows that the LP algorithm over the weighted SVs outperfoms the one over all the labeled data by 0.6_{**} and 0.8_{**} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively. This means that the weighted SVs can better model the natural clustering structure of all the training data. This may be due to that the weighted SVs can well represent all the labeled data and that it is better to propagate labels along the weighted support vectors on the decision boundary instead of all the labeled data.
(2) LP over weighted K-means performs only slightly worse than the one over all the labeled data. This means that weighted K-means can well preserve the natural clustering structure of all the labeled data.

Table 1
LP algorithm over different kinds of labeled data using a state-of-the-art polynomial kernel without any help from unlabeled data in the learning process (i.e. only test instances are included in the label propagation process). Note: (1) all the training data (including 9683 positive instances and 4553 positive instances on the ACE 2003 and 2004 corpora, respectively) are used as the labeled data in this experimentation. (2) 1356 and 623 SVs are returned on the ACE 2003 and 2004 corpora, respectively by SVM with weight bigger than 0.01. For fair comparison, the same numbers are used as Ks in K-means clustering.

Data/*F*-measure (P-R%)	Relation detection	RDC on types	RDC on subtypes
(a) ACE 2003 corpus			
Weighted SVs	75.4 (+13.3)	68.5 (+12.6)	58.2 (+11.9)
Un-weighted SVs	74.9 (+11.5)	67.5 (+11.0)	57.4 (+10.2)
Weighted K-means	74.8 (+10.8)	67.1 (+10.3)	57.1 (+9.7)
Un-weighted K-means	74.5 (+10.3)	66.4 (+9.8)	56.4 (+9.5)
All the labeled instances	75.1 (+8.1)	67.8 (+7.6)	57.6 (+7.4)
(b) ACE 2004 corpus			
Weighted SVs	77.3 (+14.9)	71.2 (+14.1)	63.5 (+13.2)
Un-weighted SVs	76.6 (+14.3)	69.6 (+12.4)	62.1 (+12.1)
Weighted K-means	76.7 (+12.5)	69.8 (+12.0)	62.3 (+11.3)
Un-weighted K-means	76.2 (+11.8)	68.6 (+11.2)	61.2 (+10.8)
All the labeled instances	76.9 (+9.5)	70.4 (+9.1)	62.7 (+8.6)

(3) LPs over weighted SVs and K-means moderately outperform those over un-weighted SVs and K-means, e.g. by 0.6_{**} and 0.7_{**} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 corpus, respectively. This means that proper weighting of SVs and K-means is necessary in modeling the natural clustering structure of all the labeled data.
(4) Among all the settings, the precision is always much higher than the recall. This applies to all the later experiments in this paper, as shown in Table 2–4, which is also consistent with previous work in the literature, such as Zhou et al. (2005) and Zhang et al. (2006). We have attempted to achieve better balance between precision and recall by adjusting relevant hyper-parameters. However, our exploration shows that the gain in the recall always fails to (compensate?) the loss in the precision. This may be due to the special characteristics of the ACE 2003 and 2004 corpora, e.g. their high quality (ACE, 2000–2005) and the data sparseness problem (Zhou et al., 2006).
(5) It is also interesting to note that, although the ACE 2004 corpus defines more relations and its size is smaller than that of the ACE 2003 corpus, the performance on the ACE 2004 corpus is consistently much higher than that on the ACE 2003 corpus. This may be due to that the relations in the ACE

Table 2
LP algorithm over different kinds of labeled data using a state-of-the-art polynomial kernel with the help from unlabeled data (All the performances are averaged over five times). Note that only part of the training data (400 positive instances and relevant negative instances in the same sentences) is used as the labeled data while the remaining part is used as the unlabeled data.

Data/*F*-measure (P-R%)	Relation detection	RDC on types	RDC on subtypes
(a) ACE 2003 corpus			
Weighted SVs	64.5 (+14.9)	54.2 (+14.2)	40.7 (+13.3)
Bootstrapped weighted SVs	66.2 (+11.1)	56.8 (+10.4)	44.8 (10.0)
Bootstrapped weighted SVs and hard unlabeled instances	67.3 (+11.3)	58.1 (+10.7)	46.5 (+10.1)
All the labeled data	64.3 (+9.8)	54.1 (+9.2)	40.4 (+9.7)
All the labeled data and unlabeled data	65.4 (+8.2)	55.6 (+7.8)	43.6 (+8.1)
(b) ACE 2004 Corpus			
Weighted SVs	64.9 (+15.6)	56.0 (+15.2)	46.2 (+14.5)
Bootstrapped weighted SVs	66.5 (+12.7)	58.5 (+12.0)	49.8 (+11.5)
Bootstrapped weighted SVs and hard unlabeled instances	67.7 (+12.8)	59.8 (+12.2)	51.9 (+11.8)
All the labeled data	64.6 (+11.6)	55.6 (+11.0)	45.9 (+11.6)
All the labeled data and unlabeled data	65.9 (+9.3)	58.3 (+8.6)	49.3 (+8.7)

Table 3
Comparison of our LP algorithm with other state-of-the-art methods on different sizes of the labeled data using a state-of-the-art polynomial kernel with the help from unlabeled data (#*P*, the number of positive instances in the labeled data; All the performances are averaged over five times).

#*P*/*F*-measure (P–R%)		400	800	1200	1600	2000
(a) ACE 2003 corpus						
Relation detection	Our LP	67.3 (+11.3)	69.1 (+10.6)	70.4 (+9.7)	71.9(+8.9)	73.2 (+8.2)
	Normal LP	65.4 (+8.2)	68.7 (+7.2)	69.8 (+6.9)	71.2 (+6.3)	72.1 (+5.5)
	SVM	59.6 (+22.3)	63.7 (+20.8)	66.1 (+20.1)	67.8 (+18.9)	69.2 (+18.0)
	BootProject	63.1 (+12.7)	66.4 (+11.3)	67.8 (+10.8)	69.3 (+9.2)	70.2 (+9.1)
RDC on types	Our LP	58.1 (+10.7)	62.5 (+10.1)	64.9 (+8.7)	66.9 (+7.5)	68.3 (+7.3)
	Normal LP	55.6 (+7.8)	60.6 (+6.7)	63.2 (+6.6)	65.4 (+5.7)	66.7 (+5.0)
	SVM	51.3 (+20.1)	56.1 (+18.6)	58.3 (+18.1)	60.8 (+17.2)	62.5 (+16.8)
	BootProject	54.3 (+12.0)	58.7 (+10.7)	61.3 (+10.4)	62.9 (+8.9)	64.2 (+8.2)
RDC on sub-types	Our LP	46.5 (+10.1)	49.3 (+9.2)	51.4 (+8.6)	53.1 (+7.7)	54.5 (+7.1)
	Normal LP	43.6 (+8.1)	46.7 (+7.0)	49.7 (+6.6)	51.6 (+6.0)	53.1 (+5.7)
	SVM	35.6 (+18.8)	41.3 (+18.1)	44.1 (+17.5)	46.2 (+17.2)	48.3 (+16.4)
	BootProject	40.3 (+11.2)	43.5 (+9.8)	46.1 (+9.3)	48.1 (+8.3)	49.6 (+7.9)
(b) ACE 2004 corpus						
Relation detection	Our LP	67.7 (+12.8)	71.2 (+11.7)	72.8 (+10.2)	74.5 (+9.4)	75.4 (+9.1)
	Normal LP	65.9 (+9.3)	69.5 (+8.6)	71.6 (+7.1)	73.2 (+6.1)	74.0 (+5.8)
	SVM	60.7 (+24.2)	66.0 (+22.6)	68.5 (+21.8)	71.0 (+21.0)	72.2 (+20.5)
	BootProject	63.8 (+13.2)	68.1 (+11.7)	70.2 (+10.9)	72.1 (+9.6)	72.9 (+9.3)
RDC on types	Our LP	59.8 (+12.2)	64.4 (+10.5)	66.3 (+9.0)	67.7 (+8.1)	68.3 (+7.5)
	Normal LP	58.3 (+8.6)	62.5 (+7.2)	64.1 (+6.8)	65.7 (+5.9)	66.5 (+5.4)
	SVM	54.1 (+22.9)	58.9 (+21.3)	61.4 (+20.7)	63.5 (+20.0)	64.9 (+19.2)
	BootProject	56.8 (+12.6)	61.0 (+10.3)	62.6 (+9.8)	64.6 (+8.7)	65.8 (+8.3)
RDC on sub-types	Our LP	51.9 (+11.8)	54.8 (+9.8)	56.9 (+8.6)	58.8 (+8.0)	60.1 (+7.4)
	Normal LP	49.3 (+9.0)	51.9 (+7.9)	54.5 (+7.5)	55.6 (+6.6)	58.2 (+6.1)
	SVM	40.7 (+20.5)	46.1 (+19.2)	49.2 (+18.0)	51.6 (+17.8)	53.4 (+17.5)
	BootProject	45.8 (+12.5)	48.7 (+10.7)	51.1 (+10.1)	52.9 (+9.0)	54.5 (+8.4)

Table 4
Comparison of our LP algorithm on different sizes of the labeled data using different kernels on the RDC of the ACE 2003 and 2004 subtypes. (#*P*, the number of positive instances in the labeled data; all the performances are averaged over five times).

#*P*/*F*-measure (P–R%)	Polynomial kernel	Convolution tree kernel	Composite kernel
(a) ACE 2003 corpus			
400	46.5 (+10.1)	44.1 (+14.3)	48.7 (+12.9)
800	49.3 (+9.2)	47.5 (+13.5)	51.4 (+12.1)
1200	51.4 (+8.6)	50.3 (+12.8)	53.6 (+11.4)
1600	53.1 (+7.7)	52.6 (+12.0)	55.0 (+10.5)
2000	54.5 (+7.1)	54.1 (+11.3)	56.3 (+9.7)
(b) ACE 2004 corpus			
400	51.9 (+11.8)	49.6 (+15.4)	54.2 (+13.8)
800	54.8 (+9.8)	53.1 (+14.2)	57.1 (+12.6)
1200	56.9 (+8.6)	55.7 (+13.1)	59.0 (+11.8)
1600	58.8 (+8.0)	58.2 (+12.3)	61.3 (+11.0)
2000	60.1 (+7.4)	60.3 (+11.7)	62.4 (+10.3)

2004 corpus are better defined and thus easier to be differentiated than those in the ACE the 2003 corpus. This also applies to all the later experiments in the paper and is consistent with previous research in the literature.

Table 2 further presents the performance of the LP algorithm over different kinds of labeled data using the state-of-the-art polynomial kernel with the help from the unlabeled data. Here, for both corpora, we randomly choose enough sentences from the training data to include 400 positive instances[5] and use all the positive and negative instances in these sentences as the labeled data while the instances in the remaining sentences of the training data are used as the unlabeled data. Table 2 shows that

(1) SVM bootstrapping significantly improves the LP algorithm, e.g. by 4.1_{***} and 3.6_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively (bootstrapped weighted SVs vs. weighted SVs). This suggests the usefulness of SVM bootstrapping in capturing the additional information in the unlabeled data and that LP can significantly benefit from SVM bootstrapping via bootstrapped weighted SVs. The reason why SVM bootstrapping is useful may be due to that bootstrapping can capture the additional information in the unlabeled data, which may not be well preserved by the labeled data. This always happens when the labeled data is not large enough, such as the case in this paper.
(2) LP over all the labeled and unlabeled data significantly outperforms the one over all the labeled data, by 3.2_{***} and 3.4_{***} in F-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively (all the labeled and unlabeled data vs. all the labeled data). This indicates that the unlabeled data contain much additional information about the natural clustering structure and that LP can much benefit from the unlabeled data.
(3) It is interesting to note that the hard unlabeled instances are very useful in preserving the natural clustering structure. For example, the hard unlabeled instances significantly improve the performance by 1.7_{***} and 2.1_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively (bootstrapped weighted SVs vs. bootstrapped weighted SVs and hard unlabeled instances). The reason why the hard unlabeled instances are useful is due to that the labeled data and the unlabeled data may not be well preserved by the bootstrapped SVs and that the remaining hard unlabeled instances in the unlabeled data after SVM bootstrapping include some important information about the data. That is, the bootstrapped SVs are unable to well represent those hard unlabeled instances and thus inclusion of them in the LP algorithm can help a lot.
(4) LP over bootstrapped weighted SVs and hard unlabeled instances significantly outperforms the one over all the labeled and unlabeled data, e.g. by 2.9_{***} and 2.6_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively. This indicates that those critical instances may better represent the natural clustering structure than all the available data in the LP algorithm, and that it is better to propagate labels via those critical instances instead of all the available labeled and unlabeled data.

Table 3 presents the performance of our LP algorithm (i.e. over bootstrapped weighted SVs and hard unlabeled instances) on different sizes of the labeled data (measured by the number of positive instances, sampled in the same way as Table 2) with other state-of-the-art machine learning methods: SVM, the original SVM bootstrapping algorithm BootProject (i.e. bootstrapping on top of SVM with feature projection, as proposed in Zhang (2004)) and the normal LP algorithm (i.e. over all the available labeled and unlabeled data). Table 3 shows that:

(1) Inclusion of unlabeled data using semi-supervised learning, including BootProject, the normal LP algorithm and our LP algorithm, consistently improves the performance, although semi-supervised learning has shown to typically fail to improve the performance when a lot of (enough) labeled data is available (Nigam, 2001). For example, even the worst-performed BootProject algorithm significantly outperforms SVM by 4.7_{***} and 5.1_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004

[5] This means ~10 positive instances per ACE relation subtype on average. Moreover, since different relation subtypes distribute much unevenly, each relation subtype is guaranteed to have at least five positive instances by picking up more sentences containing positive instances for a minor subtype to randomly replace the sentences containing positive instances for a major subtype if the minor subtype has less than five positive instances.

corpora, respectively, when 400 positive instances are used in the labeled data. This may be due to the deficiency of labeled data in the ACE 2003 and 2004 corpora. Actually, most of relation subtypes in the two corpora suffer from the data sparseness problem (Zhou et al., 2006).

(2) The LP algorithms outperform both the state-of-the-art SVM classifier and BootProject. Especially, when a small amount of labeled data is available, the performance improvements by the LP algorithms are significant. For example, even the lower-performed normal LP algorithm significantly outperforms BootProject by 3.3_{***} and 3.5_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively, when 400 positive instances are used in the labeled data. This indicates the usefulness of the natural clustering structure in both labeled and unlabeled data and the powerfulness of the LP algorithm in modeling such information.

(3) Our LP algorithm via bootstrapped support vectors and hard unlabeled instances significantly outperforms the normal LP algorithm via all the available labeled and unlabeled data (without SVM bootstrapping), especially when only a small amount of labeled data is available. For example, our LP algorithm significantly outperforms the normal LP algorithm by 2.9_{***} and 2.6_{***} in *F*-measure on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively, when 400 positive instances are used in the labeled data. This implies that bootstrapped weighted support vectors and hard unlabeled instances may represent the inherent structure (e.g. the decision boundary from where label propagation is done) better than the full set of data – an interesting result worthy of more quantitative and qualitative justifications in future work.

(4) Above comparison of SVM, SVM bootstrapping, the normal LP algorithm and our LP algorithm with SVM bootstrapping shows that both bootstrapping and label propagation contribute much to the performance improvement.

All the above experiments are done using a state-of-the-art feature-based polynomial kernel. To show the effect of kernel methods, Table 4 compares the polynomial kernel with the state-of-the-art context-sensitive convolution tree kernel, as described in Section 3.1, in our LP algorithm, on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora. Here, the SVM-Light regularization parameter C, the decay factor λ and the maximal root path length m of the tree kernel are fine-tuned to 2.4, 0.4 and 3, respectively while the weights for different context-sensitive sub-trees w_1, w_2 and w_3 are fine-tuned to 0.7, 0.2 and 0.1, respectively, using 2-fold cross-validation on the training data of the ACE RDC 2003 corpus. It shows that the tree kernel can achieve the same tendency as the polynomial kernel. Comparison of the tree kernel with the polynomial kernel shows that the tree kernel performs worse than the polynomial kernel when only a small amount of training data is available. However, the tree kernel catches up with the polynomial kernel when the labeled data size increases to a certain extent. This suggests that the tree kernel is much more sensitive to the size of labeled data than the polynomial kernel. This also suggests that, although the tree kernel may suffer more from the data sparseness problem, it has the potential for better performance in exploring structured parse tree information, which is critical for further performance improvement in semantic relation extraction. A good solution to take the advantages of both the polynomial and tree kernels is to integrate them via a composite kernel, which has been proven useful in semantic relation extraction (Zhang et al., 2006) and other NLP applications, such as semantic role labeling (Moschitti, 2004):

$$K_{Comp} = \gamma \cdot K_{Poly} + (1 + \gamma) \cdot K_{CTK} \tag{3}$$

where K_{Poly} is the polynomial kernel and K_{CTK} is our proposed context-sensitive convolution tree kernel. The weight $\gamma (0 \leqslant \gamma \leqslant 1)$ is fine-tuned to 0.6 using 2-fold cross-validation on the ACE RDC 2003 training data. Table 4 shows that the polynomial kernel and the tree kernel are quite complementary and their integration is quite effective in LP with SVM bootstrapping. For example, the composite kernel outperforms the polynomial kernel and the convolution tree kernel by 2.2_{***} and 4.6_{***}, respectively, on the RDC of the relation subtypes in the ACE RDC 2003 corpus, when 400 positive instances are used in the labeled data. This is not surprising since the tree kernel focuses on capturing syntactic structured information while the (feature-based) polynomial kernel covers different knowledge sources, which may be hard to be covered by the tree kernel.

Finally, Table 5 compares the computational burden of our LP algorithm with related machine learning methods when 400 positive instances in the labeled data are used on the RDC of the relation subtypes in

Table 5
Computational burden (in minutes) of different machine learning methods with the help from unlabeled data when 400 positive instances in the labeled data are used on the RDC of the relation subtypes in the ACE 2003 and 2004 corpora, respectively.

Methods/computational burden (in minutes): polynomial kernel/tree kernel	ACE 2003		ACE 2004	
	Training	Testing	Training	Testing
Our LP with SVM bootstrapping over the labeled and unlabeled data	628/1241	4.7/11	481/934	3.0/6.5
Normal LP over the labeled and unlabeled data	0/0	41/92	0/0	22/48
SVM	2.8/8.2	3.1/6.8	2.5/6.5	1.9/4.2
BootProject with SVM bootstrapping over the labeled and unlabeled data	628/1241	28/47	481/934	17/31

the ACE 2003 and 2004 corpora. It shows that our LP algorithm can achieve the same level of testing time as SVM at the expense of a bigger computational burden in training due to its employment of the modified BootProject, which bootstraps a moderate number of weighted supported vectors on top of SVM via a co-training procedure from the labeled and unlabeled data. Benefiting from the one-time training procedure, our LP algorithm can significantly reduce the testing time by about 85%, compared with the normal LP algorithm, which applies all the available (both labeled and unlabeled) data in the label propagation process and requires no training. Our experiments show that the critical instances (i.e. bootstrapped weighted support vectors and hard unlabeled instances) normally occupy only about 35% (average 32% for the polynomial kernel and average 39% for the tree kernel) of all the available labeled and unlabeled data. This explains that, through SVM bootstrapping, our LP algorithm can significantly reduce the computational burden since it only depends on the critical instances. Although our LP algorithm needs a SVM bootstrapping process, this process is worthwhile, considering that the SVM bootstrapping process is only applied once in the learning process and that the computational burden is significantly reduced in the label propagation process. Please note that the two algorithms with SVM bootstrapping (i.e. LP with SVM bootstrapping over the labeled and unlabeled data and BootProject with SVM bootstrapping over the labeled and unlabeled data) have the same training time. This is due to that these two algorithms have the same training process, i.e. SVM bootstrapping. The difference between the two algorithms lies in the testing process: our LP algorithm with SVM bootstrapping applies the critical instances (including the bootstrapped support vectors contained in the final bootstrapped SVM classifiers and the remaining hard unlabeled instances after SVM bootstrapping) in the label propagation process, while the BootProject algorithm with SVM bootstrapping applies the final bootstrapped SVM classifiers in a majority voting process.

5. Conclusion

The label propagation (LP) algorithm has the potential of well modeling the natural clustering structure in both the labeled and unlabeled data. However, the normal LP suffers from large computation burden, especially when a large amount of labeled and unlabeled data is taken into consideration. This paper proposes a new effective and efficient LP algorithm in semantic relation extraction by bootstrapping a moderate number of weighted support vectors from all the available labeled and unlabeled data through a SVM-based co-training procedure with feature projection. Here, a random feature projection technique is applied to generating multiple overlapping feature views for a state-of-the-art polynomial kernel while several parse tree and chunk tree views are applied for a state-of-the-art tree kernel. Then, a LP algorithm is applied to propagating labels via the bootstrapped support vectors and the hard unlabeled instances to classify unseen test instances.

Evaluation on the ACE RDC corpora shows that our LP algorithm with SVM bootstrapping can take the advantages of both SVM bootstrapping and label propagation. It also shows that our LP algorithm much outperforms the normal LP algorithm and significantly outperforms the state-of-the-art bootstrapping algorithm, especially when a small amount of labeled data is available. Finally, it shows that our LP algorithm can also significantly reduce the computation burden in the testing (i.e. label propagation) process at the expense of one-time SVM bootstrapping in the learning process, compared with the normal one.

In future work, we will explore better ways of determining the critical instances for label propagation and better metrics for measuring the similarity between two instances.

Acknowledgements

This research is supported by Projects 60673041 and 60873150 under the National Natural Science Foundation of China and Project 2006AA01Z147 under the "863" National High-Tech Research and Development of China.

References

ACE, 2000–2005. Automatic Content Extraction. <http://www.ldc.upenn.edu/Projects/ACE/>.
Agichtein, E., Gravano, L., 2000. Snowball: Extracting Relations from Large Plain-text Collections. ACMDL'2000.
Belkin, M., Niyogi, P., 2002. Using Manifold Structure for Partially Labeled Classification. NIPS'15.
Blum, A., Chawla, S., 2001. Learning from Labeled and Unlabeled Data Using Graph Mincuts. ICML'2001.
Blum, A., Lafferty, J., Rwebangira, R., Reddy, R., 2004. Semi-supervised Learning Using Randomized Mincuts. ICML'2004.
Brin, S., 1998. Extracting patterns and relations from world wide web. In: Proceedings of WebDB Workshop at Sixth International Conference on Extending Database Technology. pp. 172–183.
Bunescu, R.C., Mooney, R.J., 2007. Learning to Extract Relations from the Web Using Minimal Supervision. ACL'2007, pp. 576–583.
Carlson, A.J., Cumby, C.M., Rosen, J.L., Roth, D., 1999. The Snow Learning Achitecture. Technical Report UIUCDCS-R-99-2101, UIUC CS Dept.
Charniak, E., 2001. Immediate-head Parsing for Language Models. ACL'2001, pp. 129–137.
Chen, J.X., Ji, D.H., Tan, C.L., Niu, Z.Y., 2006. Relation Extraction Using Label Propagation Based Semi-Supervised Learning. COLING-ACL'2006, pp. 129–136.
Culotta, A., Sorensen, J., 2004. Dependency Tree Kernels for Relation Extraction. ACL'2004, pp. 423–429
Etzioni, O., Cafarella, M., Downey, D., et al., 2005. Unsupervised named entity extraction from the web: an experimental study. Artificial Intelligence 165 (1), 91–134.
Gao, J.F., Nie, J.Y., Wu, G.G., Cao, G.H., 2004. Dependence Language Model for Information Retrieval. SIGIR'2004, Sheffield, UK, July 25–29, 2004.
Hasegawa, T., Sekine, S., Grishman, R., 2004. Discovering Relations Among Named Entities form Large Corpora. ACL'2004.
Hearst, M.A., 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. ACL'1992.
Hirschman, L., Gaizauskas, R., 2001. Natural Language question answering: the view from here. Natural Language Engineering 7 (4), 275–300.
Joachims, T., 1998. Text Categorization with Support Vector Machine: Learning with Many Relevant Features. ECML'1998, pp. 137–142.
Maslennikov, M., Chua, T.S., 2007. A Multi-resolution Framework for Information Extraction From Free Text. ACL'2007, pp. 592–599.
Moschitti, A., 2004. A study on Convolution Kernels for Shallow Semantic Parsing. ACL'2004, pp. 335–342.
Nigam, K.P., 2001. Using Unlabeled Data to Improve Text Classification. Technical Report CMU-CS-01-126.
Niu, Z.Y., Ji, D.H., Tan, C.L., 2005. Word Sense Disambiguation Using Label Propagation Based Semi-supervised Learning. ACL'2005, pp. 395–402.
Strzalkowski, T., Carballo, J.P., 1998. Natural Language Information Retrieval: TREC-5 Report. 5th Text Retrieval Conference.
Szummer, M., Jaakkola, T., 2001. Partially Labeled Classification with Markov Random Walks. NIPS 14.
Xu, F.Y., Uszkoreit, H., Li, H., 2007. A Seed-Driven Bottom-up Machine Learning Framework for Extracting Relations of Various Complexity. ACl'2007, pp. 584–591.
Yang, L.P., Ji, D.H., Zhou, G.D., Nie, Y., 2006. Document Re-ranking Using Cluster Validation and Label Propagation. CIKM'2006.
Zelenko, D., Aone, C. Richardella, 2003. Kernel methods for relation extraction. Journal of Machine Learning Research 3 (Feb), 1083–1106.
Zhang, M., Su, J., Wang, D.M., Zhou, G.D., Tan, C.L., 2005. Discovering Relations from a Large Raw Corpus Using Tree Similarity-based Clustering. IJCNLP'2005, pp. 378–389.
Zhang, M., Zhang, J., Su, J., Zhou, G.D., 2006. A Composite Kernel to Extract Relations between Entities with both Flat and Structured Features. COLING-AC'2006, pp. 825–832.
Zhang, Z., 2004. Weakly Supervised Relation Classification for Information Extraction. CIKM'2004.
Zhao, S.B., Grishman, R., 2005. Extracting Relations with Integrated Information Using Kernel Methods. ACL'2005, pp. 419–426.
Zhou, G.D., Su, J., Zhang, J., Zhang, M., 2005. Exploring Various Knowledge in Relation Extraction. ACL'2005, pp. 427–434.
Zhou, G.D., Su, J., Zhang, M., 2006. Modeling Commonality Among Related Classes in Relation Extraction. COLING-ACL'2006, pp. 121–128.
Zhou, G.D., Zhang, M., Ji, D.H., Zhu, Q.M., 2007. Tree Kernel-based Relation Extraction with Context-sensitive Structured Parse Tree Information. EMNLP-CoNLL'2007.
Zhu, X., Ghahramani, Z., 2002. Learning from Labeled and Unlabeled Data with Label Propagation. Technical Report. CMU-CALD-02-107.
Zhu, X., Ghahramani, Z., Lafferty, J., 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. ICML'2003.