Graph-based Analysis of Semantic Drift in Espresso-like Bootstrapping Algorithms

Mamoru Komachi
NAIST, Japan
mamoru-k@is.naist.jp

Taku Kudo
Google Inc.
taku@google.com

Masashi Shimbo
NAIST, Japan
shimbo@is.naist.jp

Yuji Matsumoto
NAIST, Japan
matsu@is.naist.jp

Abstract

Bootstrapping has a tendency, called *semantic drift*, to select instances unrelated to the seed instances as the iteration proceeds. We demonstrate the semantic drift of bootstrapping has the same root as the topic drift of Kleinberg's HITS, using a simplified graph-based reformulation of bootstrapping. We confirm that two graph-based algorithms, the von Neumann kernels and the regularized Laplacian, can reduce semantic drift in the task of word sense disambiguation (WSD) on Senseval-3 English Lexical Sample Task. Proposed algorithms achieve superior performance to *Espresso* and previous graph-based WSD methods, even though the proposed algorithms have less parameters and are easy to calibrate.

1 Introduction

In recent years machine learning techniques become widely used in natural language processing (NLP). These techniques offer various ways to exploit large corpora and are known to perform well in many tasks. However, these techniques often require tagged corpora, which are not readily available to many languages. So far, reducing the cost of human annotation is one of the important problems for building NLP systems.

To mitigate the problem of hand-tagging resources, semi(or minimally)-supervised and unsupervised techniques have been actively studied. Hearst (1992) first presented a bootstrapping method which requires only a small amount of instances (seed instances) to start with, but can easily multiply the number of tagged instances with minimal human annotation cost, by iteratively applying the following phases: pattern induction, pattern ranking/selection, and instance extraction. Bootstrapping has been widely adopted in NLP applications such as word sense disambiguation (Yarowsky, 1995), named entity recognition (Collins and Singer, 1999) and relation extraction (Riloff and Jones, 1999; Pantel and Pennacchiotti, 2006).

However, it is known that bootstrapping often acquires instances not related to seed instances. For example, consider the task of collecting the names of common tourist sites from web corpora. Given words like "Geneva" and "Bali" as seed instances, bootstrapping would eventually learn *generic patterns* such as "pictures" and "photos," which also co-occur with many other unrelated instances. The subsequent iterations would likely acquire frequent words that co-occur with these generic patterns, such as "Britney Spears." This phenomenon is called *semantic drift* (Curran et al., 2007).

A straightforward approach to avoid semantic drift is to terminate iterations before hitting generic patterns, but the optimal number of iterations is task dependent and is hard to come by. The recently proposed *Espresso* (Pantel and Pennacchiotti, 2006) algorithm incorporates sophisticated scoring functions to cope with generic patterns, but as Komachi and Suzuki (2008) pointed out, Espresso still shows semantic drift unless iterations are terminated appropriately.

Another deficiency in bootstrapping is its sensitivity to many parameters such as the number of

Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 1011–1020,
Honolulu, October 2008.

seed instances, the stopping criterion of iteration, the number of instances and patterns selected on each iteration, and so forth. These parameters also need to be calibrated for each task.

In this paper, we present a graph-theoretic analysis of Espresso-like bootstrapping algorithms. We argue that semantic drift is inherent in these algorithms, and propose to use two graph-based algorithms that are theoretically less prone to semantic drift, as an alternative to bootstrapping.

After a brief review of related work in Section 2, we analyze in Section 3 a bootstrapping algorithm (*Simplified Espresso*) which can be thought of as a degenerate version of Espresso. Simplified Espresso is simple enough to allow an algebraic treatment, and its equivalence to Kleinberg's HITS algorithm (Kleinberg, 1999) is shown. An implication of this equivalence is that semantic drift in this bootstrapping algorithm is essentially the same phenomenon as topic drift observed in link analysis. Another implication is that semantic drift is inevitable in Simplified Espresso as it converges to the same score vector regardless of seed instances.

The original Espresso also suffers from the same problem as its simplified version does. It incorporates heuristics not present in Simplified Espresso to reduce semantic drift, but these heuristics have limited effect as we demonstrate in Section 3.3.

In Section 4, we propose two graph-based algorithms to reduce semantic drift. These algorithms are used in link analysis community to reduce the effect of topic drift. In Section 5 we apply them to the task of word sense disambiguation on Senseval-3 Lexical Sample Task and verify that they indeed reduce semantic drift. Finally, we conclude our work in Section 6.

2 Related Work

2.1 Overview of Bootstrapping

Bootstrapping (or self-training) is a general framework for reducing the requirement of manual annotation. Hearst (1992) described a bootstrapping procedure for extracting words in hyponym (*is-a*) relation, starting with three manually given lexico-syntactic patterns.

The idea of learning with a bootstrapping method was adopted for many tasks. Yarowsky (1995) presented an unsupervised WSD system which rivals supervised techniques. Abney (2004) presented a thorough discussion on the Yarowsky algorithm. He extended the original Yarowsky algorithm to a new family of bootstrapping algorithms that are mathematically well understood.

Li and Li (2004) proposed a method called *Bilingual Bootstrapping*. It makes use of a translation dictionary and a comparable corpus to help disambiguate word senses in the source language, by exploiting the asymmetric many-to-many sense mapping relationship between words in two languages.

Curran et al. (2007) presented an algorithm called *Mutual Exclusion Bootstrapping*, which minimizes semantic drift using mutual exclusion between semantic classes of learned instances. They prepared a list of so-called *stop classes* similar to a stop word list used in information retrieval to help bound the semantic classes. Stop classes are sets of terms known to cause semantic drift in particular semantic classes. However, stop classes vary from task to task and domain to domain, and human intervention is essential to create an effective list of stop classes.

A major drawback of bootstrapping is the lack of principled method for selecting optimal parameter values (Ng and Cardie, 2003; Banko and Brill, 2001). Also, there is an issue of generic patterns which deteriorates the quality of acquired instances. Previously proposed bootstrapping algorithms differ in how they deal with the problem of semantic drift. We will take recently proposed Espresso algorithm as the example to explain common configuration for bootstrapping in detail.

2.2 The Espresso Algorithm

Pantel and Pennachiotti (2006) proposed a bootstrapping algorithm called Espresso to learn binary semantic relations such as *is-a* and *part-of* from a corpus. What distinguishes Espresso from other bootstrapping algorithms is that it benefits from generic patterns by using a principled measure of instance and pattern reliability. The key idea of Espresso is recursive definition of pattern-instance scoring metrics. The reliability scores of pattern p and instance i, denoted respectively as $r_\pi(p)$ and

$r_\iota(i)$, are given as follows:

$$r_\pi(p) = \frac{\sum_{i \in I} \frac{pmi(i,p)}{\max\ pmi} r_\iota(i)}{|I|} \quad (1)$$

$$r_\iota(i) = \frac{\sum_{p \in P} \frac{pmi(i,p)}{\max\ pmi} r_\pi(p)}{|P|} \quad (2)$$

where

$$pmi(i,p) = \log_2 \frac{|i,p|}{|i,*||*,p|} \quad (3)$$

is pointwise mutual information between i and p, P and I are sets of patterns and instances, and $|P|$ and $|I|$ are the numbers of patterns and instances, respectively. $|i,*|$ and $|*,p|$ are the frequencies of pattern p and instance i in a given corpus, respectively, and $|i,p|$ is the frequency of pattern p which co-occurs with instance i. $\max\ pmi$ is a maximum value of the pointwise mutual information over all instances and patterns. The intuition behind these definitions is that a reliable pattern co-occurs with many reliable instances, and a reliable instance co-occurs with many reliable patterns.

Espresso and other bootstrapping methods iterate the following three phases: *pattern induction, pattern ranking/selection, and instance extraction*.

We describe these phases below, along with the parameters that controls each phase.

Phase 1. Pattern Induction Induce patterns from a corpus given seed instances. Patterns may be surface text patterns, lexico-syntactic patterns, and/or just features.

Phase 2. Pattern Ranking/Selection Create a pattern ranker from a corpus using instances as features and select patterns which co-occur with seed instances for the next instance extraction phase. The main issue here is to avoid ranking generic patterns high and to choose patterns with high relatedness to the seed instances. Parameters and configurations: (a) *a pattern scoring metrics* and (b) *the number of patterns to use for extraction of instances*.

Phase 3. Instance Extraction Select high-confidence instances to the seed instance set. It is desirable to keep only high-confidence instances at this phase, as they are used as seed instances for the next iteration. Optionally, instances can be cumulatively obtained on each iteration to retain highly relevant instances learned in early iterations. Parameters and configurations: (c) *instance scoring metrics*, (d) *whether to retain extracted instances on each iteration or not*, and (e) *the number of instances to pass to the next iteration*.

input:
seed vector $\mathbf{i}_0$
pattern-instance co-occurrence matrix M
output:
instance and pattern score vectors $\mathbf{i}$ and $\mathbf{p}$

$\mathbf{i} = \mathbf{i}_0$
loop
  $\mathbf{p} \leftarrow M\mathbf{i}$
  Normalize $\mathbf{p}$
  $\mathbf{i} \leftarrow M^T\mathbf{p}$
  Normalize $\mathbf{i}$
  if $\mathbf{i}$ and $\mathbf{p}$ have both converged **then**
    return $\mathbf{i}$ and $\mathbf{p}$
  end if
end loop

Figure 1: A simple bootstrapping algorithm

Bootstrapping iterates the above three phases several times until stopping criteria are met. Acquired instances tend to become noisy as the iteration proceeds, so it is important to terminate before semantic drift occurs. Thus, we have another configuration: (f) *stopping criterion*.

Espresso uses Equations (1) for (a) and (2) for (c) respectively, whereas other parameters rely on the tasks and need calibration. Even though Espresso greatly improves recall while keeping high precision by using these pattern and instance scoring metrics, Komachi and Suzuki (2008) observed that extracted instances matched against generic patterns may become erroneous after tens of iterations, showing the difficulty of applying bootstrapping methods to different domains.

3 Analysis of an Espresso-like Bootstrapping Algorithm

3.1 Simplified Espresso

Let us consider a simple bootstrapping algorithm illustrated in Figure 1, in order to elucidate the cause

of semantic drift.

As before, let $|I|$ and $|P|$ be the numbers of instances and patterns, respectively. The algorithm takes a seed vector $\mathbf{i}_0$, and a pattern-instance co-occurrence matrix M as input. $\mathbf{i}_0$ is a $|I|$-dimensional vector with 1 at the position of seed instances, and 0 elsewhere. M is a $|P| \times |I|$-matrix whose (p, i)-element $[M]_{pi}$ holds the (possibly reweighted) number of co-occurrence of pattern p and instance i in the corpus. If both $\mathbf{i}$ and $\mathbf{p}$ have converged, the algorithm returns the pair of $\mathbf{i}$ and $\mathbf{p}$ as output.

This algorithm, though simple, can encode Espresso's update formulae (1) and (2) as Steps 3 through 6 if we pose

$$[M]_{pi} = \frac{pmi(i,p)}{\max pmi}, \tag{4}$$

and normalize $\mathbf{p}$ and $\mathbf{i}$ in Steps 4 and 6 by

$$\mathbf{p} \leftarrow \mathbf{p}/|I| \quad \text{and} \quad \mathbf{i} \leftarrow \mathbf{i}/|P|, \tag{5}$$

respectively.

This specific instance of the algorithm of Figure 1, obtained by specialization through Equations (4) and (5), will be henceforth referred to as *Simplified Espresso*. Indeed, it is an instance of the original Espresso in which the iteration is not terminated until convergence, all instances are carried over to the next iteration, and instances are not cumulatively learned.

3.2 Simplified Espresso as Link Analysis

Let n denote the number of times Steps 2–10 are iterated. Plugging (4) and (5) into Steps 3–6, we see that the score vector of instances after the nth iteration is

$$\mathbf{i}_n = A^n \mathbf{i}_0 \tag{6}$$

where

$$A = \frac{1}{|I||P|} M^T M. \tag{7}$$

Suppose matrix A is irreducible; i.e., the graph induced by taking A as the adjacency matrix is connected. If n is increased and $\mathbf{i}_n$ is normalized on each iteration, $\mathbf{i}_n$ tends to the principal eigenvector of A. This implies that no matter what seed instances are input, the algorithm will end up with the same ranking of instances, if it is run until convergence. Because $A = \frac{M^T M}{|I||P|}$, the principal eigenvector of A is identical to the authority vector of HITS (Kleinberg, 1999) algorithm run on the graph induced by M. [1] This similarity of Equations (1), (2) and HITS is not discussed in (Pantel and Pennacchiotti, 2006).

As a consequence of the above discussion, semantic drift in simplified Espresso seems to be inevitable as the iteration proceeds, since the principal eigenvector of A need not resemble seed vector $\mathbf{i}_0$. A similar phenomenon is reported for HITS and is known as *topic drift*, in which pages of the dominant topic are ranked high regardless of the given query. (Bharat and Henzinger, 1998)

Unlike HITS and Simplified Espresso, however, Espresso and other bootstrapping algorithms (Yarowsky, 1995; Riloff and Jones, 1999), incorporate heuristics so that only patterns and instances with high confidence score are carried over to the next iteration.

3.3 Convergence Process of Espresso

To investigate the effect of semantic drift on Espresso with and without the heuristics of selecting the most confident instances on each iteration (i.e., the original Espresso and Simplified Espresso of Section 3.2), we apply them to the task of word sense disambiguation of word "bank" in the Senseval-3 Lexical Sample (S3LS) Task data.[2] There are 394 instances of word "bank" and their occurring context in this dataset, and each of them is annotated with its true sense. Of the ten senses of *bank,* the most frequent is the bank as in "bank of the river." We use the standard training-test split provided with the data set.

We henceforth denote Espresso with the following filtering strategy as *Filtered Espresso* to stress the distinction from Simplified Espresso. For Filtered Espresso, we cleared all but the 100 top-scoring instances in the instance vector on each iteration, and the number of non-zeroed instance scores

[1] As long as the relative magnitude of the components of vector $\mathbf{i}_n$ is preserved, the vector can be normalized in any way on each iteration. Hence HITS and Simplified Espresso use different normalization but both converge to the principal eigenvector of A.

[2] http://www.senseval.org/senseval3/data.html

grows by 100 on each iteration. On the other hand, we cleared all but the 20 top-scoring patterns in the pattern vector on each iteration, and the number of non-zeroed pattern scores grows by 1 on each iteration following (Pantel and Pennacchiotti, 2006).[3] The values of other parameters (b), (d), (e) and (f) remains the same as those for simplified Espresso in Section 3.1.

The task of WSD is to correctly predict the senses of test instances whose true sense is hidden from the system, using training data and their true senses. To predict the sense of a given instance i, we apply k-nearest neighbor algorithm.

Given a test instance i, its sense is predicted with the following procedure:

1. Compute the instance-pattern matrix M from the entire set of instances. We defer the details of this step to Section 5.2.

2. Run Simplified- and Filtered Espresso using the given instance i as the only seed instance.

3. After the termination of the algorithm, select k training instances with the highest scores in the score vector $\mathbf{i}$ output by the algorithm.

4. Since the selected k instances are training instances, their true senses are accessible. Choose the majority sense s from these k instances, and output s as the prediction for the given instance i. When there is a tie, output the sense of the instance with the highest score in $\mathbf{i}$. Note that only Step 4 uses sense information.

Figure 2 shows the convergence process of Simplified- and Filtered Espresso. X-axis indicates the number of bootstrapping iterations and Y-axis indicates the recall, which in this case equals precision, as the coverage is 100% in all cases.

[3] We conducted preliminary experiment to find these parameters to maximize the performance of Filtered Espresso. (These numbers are different from the original Espresso (Pantel and Pennacchiotti, 2006).) The number of initial patterns is relatively large because of a data sparseness problem in WSD, unlike relation extraction and named entity recognition. Also, WSD basically uses more features than relation extraction and thus it is hard to determine the stopping criterion based on the number and scores of patterns, as (Pantel and Pennacchiotti, 2006) does.

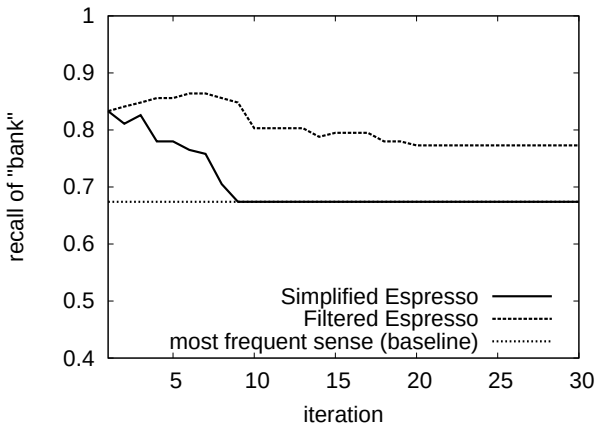

Figure 2: Recall of Simplified- and Filtered Espresso

Simplified Espresso tends to select the most frequent sense as the iteration proceeds, and after nine iterations it selects the most frequent sense ("the bank of the river") regardless of the seed instances. As expected from the discussion in Section 3.2, generic patterns gradually got more weight and semantic drift occurred in later iterations. Indeed, the ranking of the instances after convergence was identical to the HITS authority ranking computed from instance-pattern matrix M (i.e., the ranking induced by the dominant eigenvector of $M^T M$).

On the other hand, Filtered Espresso suffers less from semantic drift. The final recall achieved was 0.773 after convergence on the 20th iteration, outperforming the most-frequent sense baseline by 0.10. However, a closer look reveals that the filtering heuristics is limited in effectiveness.

Figure 3 plots the learning curve of Filtered Espresso on the set of test instances. We show recall ($\frac{|correct\ instances|}{|total\ true\ instances|}$) of each sense to see how Filtered Espresso tends to select the most frequent sense. If semantic drift takes place, the number of instances predicted as the most frequent sense should increase as the iteration proceeds, resulting in increased recall on the most frequent sense and decreased recall on other senses. Figure 3 exactly exhibit this trend, meaning that Filtered Espresso is not completely free from semantic drift. Figure 2 also shows that the recall of Filtered Espresso starts to decay after the seventh iteration.

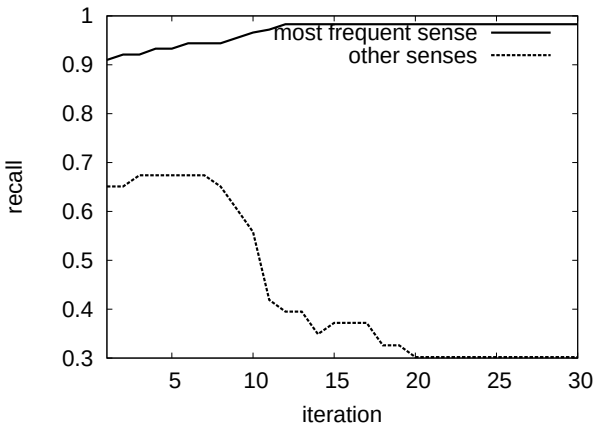

Figure 3: Recall of Filtered Espresso on the instances having "bank of the river" and other senses

4 Two Graph-based Algorithms for Exploiting Generic Patterns

We explore two graph-based methods which have the advantage of Espresso to harness the property of generic patterns by the mutual recursive definition of instance and pattern scores. They also have less parameters than bootstrapping, and are less prone to semantic drift.

4.1 Von Neumann Kernel

Kandola et al. (2002) proposed the *von Neumann kernels* for measuring similarity of documents using words. If we apply the von Neumann kernels to the pattern-instance co-occurrence matrix instead of the document-word matrix, the relative importance of an instance to seed instances can be estimated.

Let $A = M^T M$ be the instance similarity matrix obtained from pattern-instance matrix M, and λ be the principal eigenvalue of A. The *von Neumann kernel matrix* K_β with diffusion factor β ($0 \leq \beta < \lambda^{-1}$) is defined as follows:

$$K_\beta \;=\; A\sum_{n=0}^{\infty}\beta^n A^n = A(I-\beta A)^{-1}. \quad (8)$$

The similarity between two instances i, j is given by the (i, j) element of K_β. Hence, the i-th column vector can be used as the score vector for seed instance i.

Ito et al. (2005) showed that the von Neumann kernels represent a mixture of the co-citation relatedness and Kleinberg's HITS importance. They compute the weighted sum of all paths between two nodes in the co-citation graph induced by $A = M^T M$. The $(M^T M)^n$ term of smaller n corresponds to the relatedness to the seed instances, and the $(M^T M)^n$ term of larger n corresponds to HITS importance. The von Neumann kernels calculate the weighted sum of $(M^T M)^n$ from $n = 1$ to ∞, and therefore smaller diffusion factor β results in ranking by relatedness, and larger β returns ranking by HITS importance.

In NLP literature, Schütze (1998) introduced the notion of first- and second-order co-occurrence. First-order co-occurrence is a context which directly co-occurs with a word, whereas second-order co-occurrence is a context which occurs with the (contextual) words that co-occur with a word. Higher-order co-occurrence information is less sparse and more robust than lower-order co-occurrence, and thus is useful for a proximity measure.

Given these definitions, we see that the $(M^T M)^n$ term of smaller n corresponds to lower-order co-occurrence, which is accurate but sparse, and the $(M^T M)^n$ term of larger n corresponds to higher-order co-occurrence, which is dense but possibly giving too much weight on unrelated instances extracted by generic patterns.

As a result, it is expected that setting diffusion factor β to a small value prevents semantic drift and also takes higher order pattern vectors into account. We verify this claim in Section 5.3.

4.2 Regularized Laplacian Kernel

The von Neumann kernels can be regarded as a mixture of relatedness and importance, and diffusion factor β controls the trade-off between relatedness and importance. In practice, however, setting the right parameter value becomes an issue. We solve this problem by the regularized Laplacian (Smola and Kondor, 2003; Chebotarev and Shamis, 1998), which are stable across diffusion factors and can safely benefit from generic patterns.

Let G be a weighted undirected graph whose adjacency (weight) matrix is a symmetric matrix A. The (combinatorial) graph Laplacian L of a graph G is defined as follows:

$$L = D - A \quad (9)$$

where D is a diagonal matrix, and the ith diagonal

Table 1: Recall of predicted labels of *bank*

algorithm	MFS	others
Simplified Espresso	100.0	0.0
Filtered Espresso	100.0	30.2
Filtered Espresso (optimal stopping)	94.4	67.4
von Neumann kernels	92.1	65.1
regularized Laplacian	92.1	62.8

element $[D]_{ii}$ is given by

$$[D]_{ii} = \sum_j [A]_{ij}. \tag{10}$$

Here, $[A]_{ij}$ stands for the (i, j) element of A. By replacing A with $-L$ in Equation (8) and deleting the first A, we obtain a *regularized Laplacian kernel* [4].

$$R_\beta = \sum_{n=0}^{\infty} \beta^n (-L)^n = (I + \beta L)^{-1} \tag{11}$$

Again, $\beta(\geq 0)$ is called the diffusion factor.

Both the regularized Laplacian and the von Neumann kernels compute all the possible paths in a graph, and consequently they can calculate influence between nodes in a long distance in the graph. Also, Equations (9) and (10) show that the negative Laplacian $-L$ can be regarded as a modification to the graph G with the weight of self-loops re-weighted to negative values. In this modified graph, if an instance co-occurs with a pattern which also co-occurs with a large number of other instances, a self-loop of a node in the instance similarity graph induced by $M^T M$ will receive a higher negative weight. In other words, instances co-occurring with generic patterns will get less weight in the regularized Laplacian than in the von Neumann kernels.

5 Experiments and Results

5.1 Experiment 1: Reducing Semantic Drift

We test the von Neumann kernels and the regularized Laplacian on the same task as we used in Section 3.3; i.e., word sense disambiguation of word "bank." During the training phase, a pattern-instance matrix M was constructed using the training and testing data from Senseval-3 Lexical Sample (S3LS) Task. The (i, j) element of M of both kernels is set to pointwise mutual information of a pattern i and an instance j, just the same as in Espresso. Recall is used in evaluation.[5] The diffusion parameter β is set to 10^{-5} and 10^{-2} for the von Neumann kernels and the regularized Laplacian, respectively.

Table 1 illustrates how well the proposed methods reduce semantic drift, just the same as the experiment of Figure 3 in Section 3.3. We evaluate the recall on predicting the most frequent sense (MFS) and the recall on predicting other less frequent senses (others). For Filtered Espresso, two results are shown: the result on the seventh iteration, which maximizes the performance (Filtered Espresso (optimal stopping)), and the one after convergence. As in Section 3.3, if semantic drift occurs, recall of prediction on the most frequent sense increases while recall of prediction on other senses declines. Even Filtered Espresso was affected by semantic drift, which is again a consequence of the inherent graphical nature of Espresso-like bootstrapping algorithms. On the other hand, both proposed methods succeeded to balance the most frequent sense and other senses. Filtered Espresso at the optimal number of iterations achieved the best performance. Nevertheless, the number of iterations has to be estimated separately.

5.2 Experiment 2: WSD Benchmark Data

We conducted experiments on the task of word sense disambiguation of S3LS data, this time not just on the word "bank" but on all target nouns in the data, following (Agirre et al., 2006). We used two types of patterns.

Unordered single words (bag-of-words) We used all single words (unigrams) in the provided context from S3LS data sets. Each word in the context constructs one pattern. The pattern corresponding to a word w is set to 1 if it appears in the context of instance i. Words were lowercased and preprocessed with the Porter Stemmer[6].

[4] It has been reported that normalization of A improves performance in application (Johnson and Zhang, 2007), so we normalize L by $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.

[5] Again, recall equals precision in this case as the coverage is 100% in all cases.

[6] http://tartarus.org/~martin/PorterStemmer/def.txt

Table 2: Comparison of WSD algorithms

algorithm	Recall
most frequent sense	54.5
HyperLex (Véronis, 2004)	64.6
PageRank (Agirre et al., 2006)	64.5
Simplified Espresso	44.1
Filtered Espresso	46.9
Filtered Espresso (optimal stopping)	66.5
von Neumann kernels ($\beta = 10^{-5}$)	**67.2**
regularized Laplacian ($\beta = 10^{-2}$)	67.1

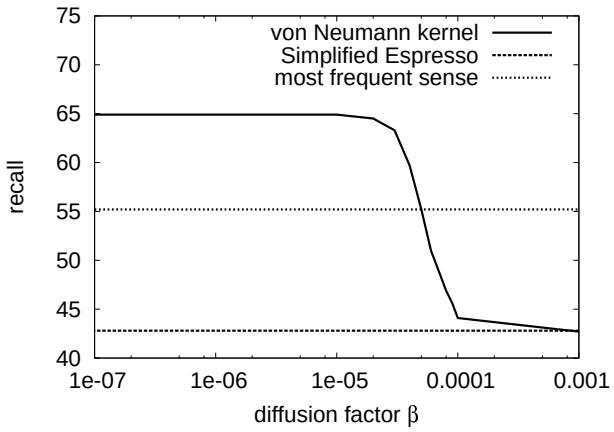

Figure 4: Recall of the von Neumann kernels with a different diffusion factor β on S3LS WSD task

Local collocations A local collocation refers to the ordered sequence of tokens in the local, narrow context of the target word. We allowed a pattern to have wildcard expressions like "sale of * *interest* in * *" for the target word *interest*. We set the window size to ± 3 by a preliminary experiment.

We report the results of Filtered Espresso both after convergence, and with its optimal number of iterations to show the upper bound of its performance.

Table 2 compares proposed methods with Espresso with various configurations. The proposed methods outperform by a large margin the most frequent sense baseline and both Simplified- and Filtered Espresso. This means that the proposed methods effectively prevent semantic drift.

Also, Filtered Espresso without early stopping shows more or less identical performance to Simplified Espresso. It is implied that the heuristics of filtering and early stopping is a crucial step not to select generic patterns in Espresso, and the result is consistent with the experiment of convergence process of Espresso in Section 3.3.

Filtered Espresso halted after the seventh iteration (Filtered Espresso (optimal stopping)) is comparable to the proposed methods. However, in bootstrapping, not only the number of iterations but also a large number of parameters must be adjusted for each task and domain. This shortcoming makes it hard to adapt bootstrapping in practical cases. One of the main advantages of the proposed methods is that they have only one parameter β and are much easier to tune.

It is suggested in Sections 3.3 and 4.1 that Espresso and the von Neumann kernel with large β converge to the principal eigenvector of A, though the result does not seem to support this claim (both Simplified- and Filtered Espresso are 10 points lower than the most frequent sense baseline). The reason seems to be because Espresso and the von Neumann kernels use pointwise mutual information as a weighting factor so that the principal eigenvector of A may not always represent the most frequent sense.[7]

We also show the results of previous graph-based methods (Agirre et al., 2006), based on HyperLex (Véronis, 2004) and PageRank (Brin and Page, 1998). The experimental set-up is the same as ours in that they do not use the sense tags of training corpus to construct a co-occurrence graph, and they use the sense tags of all the S3LS training corpus for mapping senses to clusters. However, these methods have seven parameters to tune in order to achieve the best performance, and hence are difficult to optimize.

5.3 Experiment 3: Sensitivity to a Different Diffusion Factor

Figure 4 shows the performance of the von Neumann kernels with a diffusion factor β. As expected, smaller β leads to relatedness to seed instances, and larger β asymptotically converges to the HITS authority ranking (or equivalently, Simplified

[7] A similar but more extreme case is described in (Ito et al., 2005) in which the use of a normalized weight matrix M results in an unintuitive principal eigenvector.

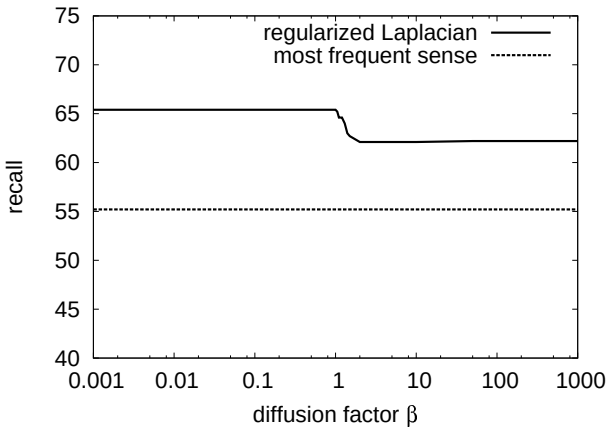

Figure 5: Recall of the regularized Laplacian with a different diffusion factor β on S3LS WSD task

Espresso).

One of the disadvantages of the von Neumann kernels over the regularized Laplacian is their sensitivity to parameter β. Figure 5 illustrates the performance of the regularized Laplacian with a diffusion factor β. The regularized Laplacian is stable for various values of β, while the von Neumann kernels change their behavior drastically depending on the value of β. However, β in the von Neumann kernels is upper-bounded by the reciprocal $1/\lambda$ of the principal eigenvalue of A, and the derivatives of kernel matrices with respect to β can be used to guide systematic calibration of β (see (Ito et al., 2005) for detail).

6 Conclusion and Future Work

This paper gives a graph-based analysis of semantic drift in Espresso-like bootstrapping algorithms. We indicate that semantic drift in bootstrapping is a parallel to topic drift in HITS. We confirm that the von Neumann kernels and the regularized Laplacian reduce semantic drift in the Senseval-3 Lexical Sample task. Our proposed methods have only one parameters and are easy to calibrate.

Beside the regularized Laplacian, many other kernels based on the eigenvalue regularization of the Laplacian matrix have been proposed in machine learning community (Kondor and Lafferty, 2002; Nadler et al., 2006; Saerens et al., 2004). One such kernel is the commute-time kernel (Saerens et al., 2004) defined as the pseudo-inverse of Laplacian. Despite having no parameters at all, it has been reported to perform well in many collaborative filtering tasks (Fouss et al., 2007). We plan to test these kernels in our task as well.

Another research topic is to investigate other semi-supervised learning techniques such as co-training (Blum and Mitchell, 1998). As we have described in this paper, self-training can be thought of a graph-based algorithm. It is also interesting to analyze how co-training is related to the proposed algorithm.

Bootstrapping algorithms have been used in many NLP applications. Two major tasks of bootstrapping are word sense disambiguation and named entity recognition. In named entity recognition task, instances are usually retained on each iteration and added to seed instance set. This seems to be because named entity recognition suffers from semantic drift more severely than word sense disambiguation. Even though this problem setting is different from ours, it needs to be verified that the graph-based approaches presented in this paper are also effective in named entity recognition.

Acknowledgements

We thank anonymous reviewers for helpful comments and for making us aware of Abney's work. The first author is partially supported by the Japan Society for Promotion of Science (JSPS), Grant-in-Aid for JSPS Fellows.

References

Steven Abney. 2004. Understanding the Yarowsky Algorithm. *Computational Linguistics*, 30(3):365–395.

Eneko Agirre, David Martínez, Oier López de Lacalle, and Aitor Soroa. 2006. Two graph-based algorithms for state-of-the-art WSD. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 585–593.

Michele Banko and Eric Brill. 2001. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33.

Krishna Bharat and Monika R. Henzinger. 1998. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st ACM SIGIR Conference*.

Avrim Blum and Tom Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the Workshop on Computational Learning Theory (COLT)*, pages 92–100. Morgan Kaufmann.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

Pavel Yu Chebotarev and Elena V. Shamis. 1998. On proximity measures for graph vertices. *Automation and Remote Control*, 59(10):1443–1459.

Michael Collins and Yoram Singer. 1999. Unsupervised Models for Named Entity Classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–110.

James R. Curran, Tara Murphy, and Bernhard Scholz. 2007. Minimising semantic drift with Mutual Exclusion Bootstrapping. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 172–180.

François Fouss, Luh Yen, Pierr Dupont, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369.

Marti Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 539–545.

Takahiko Ito, Masashi Shimbo, Taku Kudo, and Yuji Matsumoto. 2005. Application of Kernels to Link Analysis. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 586–592.

Rie Johnson and Tong Zhang. 2007. On the Effectiveness of Laplacian Normalization for Graph Semi-supervised Learning. *Journal of Machine Learning Research*, 8:1489–1517.

Jaz Kandola, John Shawe-Taylor, and Nello Cristianini. 2002. Learning Semantic Similarity. In *Advances in Neural Information Processing Systems 15*, pages 657–664.

Jon Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632.

Mamoru Komachi and Hisami Suzuki. 2008. Minimally Supervised Learning of Semantic Knowledge from Query Logs. In *Proceedings of the 3rd International Joint Conference on Natural Language Processing*, pages 358–365.

Risi Imre Kondor and John Lafferty. 2002. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the 19th International Conference on Machine Learning (ICML-2002)*.

Hang Li and Cong Li. 2004. Word Translation Disambiguation Using Bilingual Bootstrapping. *Computational Linguistics*, 30(1):1–22.

Boaz Nadler, Stephane Lafon, Ronald Coifman, and Ioannis Kevrekidis. 2006. Diffusion maps, spectral clustering and eigenfunctions of fokker-planck operators. *Advances in Neural Information Processing Systems 18*, pages 955–962.

Vincent Ng and Claire Cardie. 2003. Weakly Supervised Natural Language Learning Without Redundant Views. In *Proceedings of the HLT-NAACL 2003*, pages 94–101.

Patrick Pantel and Marco Pennacchiotti. 2006. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 113–120.

Ellen Riloff and Rosie Jones. 1999. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intellligence (AAAI-99)*, pages 474–479.

Marco Saerens, François Fouss, Luh Yen, and Pierre Dupont. 2004. The principal component analysis of a graph, and its relationship to spectral clustering. In *Proceedings of European Conference on Machine Learning (ECML 2004)*, pages 371–383. Springer.

Heinrich Schütze. 1998. Automatic Word Sense Discrimination. *Computational Linguistics*, 24(1):97–123.

Alex J. Smola and Risi Imre Kondor. 2003. Kernels and Regularization of Graphs. In *Proceedings of the 16th Annual Conference on Learning Theory*, pages 144–158.

Jean Véronis. 2004. HyperLex: Lexical Cartography for Information Retrieval. *Computer Speech & Language*, 18(3):223–252.

David Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196.