

# Discriminative Training for Near-Synonym Substitution

Liang-Chih Yu<sup>1</sup>, Hsiu-Min Shih<sup>2</sup>, Yu-Ling Lai<sup>2</sup>, Jui-Feng Yeh<sup>3</sup> and Chung-Hsien Wu<sup>4</sup>

<sup>1</sup>Department of Information Management, Yuan Ze University

<sup>2</sup>Department of Mathematics, National Chung Cheng University

<sup>3</sup>Department of CSIE, National Chia-Yi University

<sup>4</sup>Department of CSIE, National Cheng Kung University

Contact: lcyu@saturn.yzu.edu.tw

## Abstract

Near-synonyms are useful knowledge resources for many natural language applications such as query expansion for information retrieval (IR) and paraphrasing for text generation. However, near-synonyms are not necessarily interchangeable in contexts due to their specific usage and syntactic constraints. Accordingly, it is worth to develop algorithms to verify whether near-synonyms do match the given contexts. In this paper, we consider the near-synonym substitution task as a classification task, where a classifier is trained for each near-synonym set to classify test examples into one of the near-synonyms in the set. We also propose the use of discriminative training to improve classifiers by distinguishing positive and negative features for each near-synonym. Experimental results show that the proposed method achieves higher accuracy than both pointwise mutual information (PMI) and n-gram-based methods that have been used in previous studies.

## 1 Introduction

Near-synonym sets represent groups of words with similar meaning, which are useful knowledge resources for many natural language applications. For instance, they can be used for query expansion in information retrieval (IR) (Moldovan and Mihalcea, 2000; Bhogal et al., 2007), where a query term can be expanded by its near-synonyms to improve the recall rate. They can also be used in an intelligent thesaurus that can automatically suggest alternative words to avoid repeating the same word in the composing of text when there are suitable alternatives in its

synonym set (Inkpen and Hirst, 2006; Inkpen, 2007). These near-synonym sets can be derived from manually constructed dictionaries such as WordNet (called synsets) (Fellbaum, 1998), EuroWordNet (Rodríguez et al., 1998), or clusters derived using statistical approaches (Lin, 1998).

Although the words in a near-synonym set have similar meaning, they are not necessarily interchangeable in practical use due to their specific usage and collocational constraints. Pearce (2001) presented an example of collocational constraints for the context “\_\_\_ coffee”. In the given near-synonym set {strong, powerful}, the word “strong” is more suitable than “powerful” to fill the gap, since “powerful coffee” is an anti-collocation. Inkpen (2007) also presented several examples of collocations (e.g. ghastly mistake) and anti-collocations (e.g. ghastly error). Yu *et al.* (2007) described an example of the context mismatch problem for the context “\_\_\_ under the bay” and the near-synonym set {bridge, overpass, viaduct, tunnel} that represents the meaning of a physical structure that connects separate places by traversing an obstacle. The original word (target word) in the given context is “tunnel”, and cannot be substituted by the other words in the same set since all the substitutions are semantically implausible. Accordingly, it is worth to develop algorithms to verify whether near-synonyms do match the given contexts. Applications can benefit from this ability to provide more effective services. For instance, a writing support system can assist users to select an alternative word that best fits a given context from a list of near-synonyms.

In measuring the substitutability of words, the co-occurrence information between a target word

(the gap) and its context words is commonly used in statistical approaches. Edmonds (1997) built a lexical co-occurrence network from 1989 Wall Street Journal to determine the near-synonym that is most typical or expected in a given context. Inkpen (2007) used the pointwise mutual information (PMI) formula to select the best near-synonym that can fill the gap in a given context. The PMI scores for each candidate near-synonym are computed using a larger web corpus, the Waterloo terabyte corpus, which can alleviate the data sparseness problem encountered in Edmonds' approach. Following Inkpen's approach, Gardiner and Dras (2007) also used the PMI formula with a different corpus (the Web 1T 5-gram corpus) to explore whether near-synonyms differ in attitude.

Yu *et al.* (2007) presented a method to compute the substitution scores for each near-synonym based on n-gram frequencies obtained by querying Google. A statistical test is then applied to determine whether or not a target word can be substituted by its near-synonyms. The dataset used in their experiments are derived from the OntoNotes corpus (Hovy *et al.*, 2006; Pradhan *et al.*, 2007), where each near-synonym set corresponds to a *sense pool* in OntoNotes. Another direction to the task of near-synonym substitution is to identify the senses of a target word and its near-synonyms using word sense disambiguation (WSD), comparing whether they were of the same sense (McCarthy, 2002; Dagan *et al.*, 2006). Dagan *et al.* (2006) described that the use of WSD is an indirect approach since it requires the intermediate sense identification step, and thus presented a sense matching technique to address the task directly.

In this paper, we consider the near-synonym substitution task as a classification task, where a classifier is trained for each near-synonym set to classify test examples into one of the near-synonyms in the set. However, near-synonyms share more common context words (features) than semantically dissimilar words in nature. Such similar contexts may decrease classifiers' ability to discriminate among near-synonyms. Therefore, we propose the use of a supervised discriminative training technique (Ohler *et al.*, 1999; Kuo and Lee, 2003; Zhou and He, 2009) to improve classifiers by distinguishing positive and negative features for each near-synonym. To

**Sentence:** This will make the \_\_\_\_\_ message easier to interpret.

**Original word:** error

**Near-synonym set:** {error, mistake, oversight}

Figure 1. Example of a near-synonym set and a sentence to be evaluated.

our best knowledge, this is the first study that uses discriminative training for near-synonym or lexical substitution. The basic idea of discriminative training herein is to adjust feature weights according to the minimum classification error (MCE) criterion. The features that contribute to discriminating among near-synonyms will receive a greater positive weight, whereas the noisy features will be penalized and might receive a negative weight. This re-weighting scheme helps increase the separation of the correct class against its competing classes, thus improves the classification performance.

The proposed supervised discriminative training is compared with two unsupervised methods, the PMI-based (Inkpen, 2007) and n-gram-based (Yu *et al.*, 2007) methods. The goal of the evaluation is described as follows. Given a near-synonym set and a sentence with one of the near-synonyms in it, the near-synonym is deleted to form a gap in this sentence. Figure 1 shows an example. Each method is then applied to predict an answer (best near-synonym) that can fill the gap. The possible candidates are all the near-synonyms (including the original word) in the given set. Ideally, the correct answers should be provided by human experts. However, such data is usually unavailable, especially for a large set of test examples. Therefore, we follow Inkpen's experiments to consider the original word as the correct answer. The proposed methods can then be evaluated by examining whether they can restore the original word by filling the gap with the best near-synonym.

The rest of this work is organized as follows. Section 2 describes the PMI and n-gram-based methods for near-synonym substitution. Section 3 presents the discriminative training technique. Section 4 summarizes comparative results. Conclusions are finally drawn in Section 5.

## 2 Unsupervised Methods

### 2.1 PMI-based method

The mutual information can measure the co-occurrence strength between a near-synonym and the words in a given context. A higher mutual information score indicates that the near-synonym fits well in the given context, thus is more likely to be the correct answer. The pointwise mutual information (Church and Hanks, 1991) between two words  $x$  and  $y$  is defined as

$$PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}, \quad (1)$$

where  $P(x, y) = C(x, y)/N$  denotes the probability that  $x$  and  $y$  co-occur;  $C(x, y)$  is the number of times  $x$  and  $y$  co-occur in the corpus, and  $N$  is the total number of words in the corpus. Similarly,  $P(x) = C(x)/N$ , where  $C(x)$  is the number of times  $x$  occurs in the corpus, and  $P(y) = C(y)/N$ , where  $C(y)$  is the number of times  $y$  occurs in the corpus. Therefore, (1) can be re-written as

$$PMI(x, y) = \log_2 \frac{C(x, y) \cdot N}{C(x) \cdot C(y)}. \quad (2)$$

Inkpen (2007) computed the PMI scores for each near-synonym using the Waterloo terabyte corpus and a context window of size  $2k$  ( $k=2$ ). Given a sentence  $s$  with a gap,  $s = \dots w_1 \dots w_k \text{ \_\_\_ } w_{k+1} \dots w_{2k} \dots$ , the PMI score for a near-synonym  $NS_i$  to fill the gap is defined as

$$PMI(NS_j, s) = \sum_{i=1}^k PMI(NS_j, w_i) + \sum_{i=k+1}^{2k} PMI(NS_j, w_i). \quad (3)$$

The near-synonym with the highest score is considered as the answer. In this paper, we use the Web 1T 5-gram corpus to compute PMI scores, the same as in (Gardiner and Dras, 2007). The frequency counts  $C(\cdot)$  are retrieved from this corpus in the same manner within the 5-gram boundary.

### 2.2 N-gram-based method

The n-grams can capture contiguous word associations in given contexts. Given a sentence with a gap, the n-gram scores for each near-synonym

are computed as follows. First, all possible n-grams containing the gap are extracted from the sentence. Each near-synonym then fills the gap to compute a normalized frequency according to

$$Z(ng\text{ram}_{NS_j}^i) = \frac{\log(C(ng\text{ram}_{NS_j}^i) + 1)}{\log C(NS_j)}, \quad (4)$$

where  $C(ng\text{ram}_{NS_j}^i)$  denotes the frequency of an n-gram containing a near-synonym,  $C(NS_j)$  denotes the frequency of a near-synonym, and  $Z(ng\text{ram}_{NS_j}^i)$  denotes the normalized frequency of an n-gram, which is used to reduce the effect of high frequencies on measuring n-gram scores. All of the above frequencies are retrieved from the Web 1T 5-gram corpus.

The n-gram score for a near-synonym to fill the gap is computed as

$$NGRAM(NS_j, s) = \frac{1}{R} \sum_{i=1}^R Z(ng\text{ram}_{NS_j}^i), \quad (5)$$

where  $NGRAM(NS_j, s)$  denotes the n-gram score of a near-synonym, which is computed by averaging the normalized frequencies of all the n-grams containing the near-synonym, and  $R$  is the total number of n-grams containing the near-synonym. Again, the near-synonym with the highest score is the proposed answer. We herein use the 4-gram frequencies to compute n-gram scores as Yu *et al.* (2007) reported that the use of 4-grams achieved higher performance than the use of bigrams and trigrams.

## 3 Discriminative Training

### 3.1 Classifier

The supervised classification technique can also be applied to for near-synonym substitution. Each near-synonym in a set corresponds to a class. The classifiers for each near-synonym set are built from the labeled training data, i.e., a collection of sentences containing the near-synonyms. Such training data is easy to obtain since it requires no human annotation. The training data used herein is collected by extracting the 5-grams containing the near-synonyms from the Web 1T 5-gram corpus. The features used for training are the words occurring in the context of the near-synonyms in the 5-grams.

	Example		
	1	2	3
$g_1(x, M) = \cos(x, m_1)$	0.9*	0.6*	0.8
$g_2(x, M) = \cos(x, m_2)$	0.3	0.5	0.3*
$g_3(x, M) = \cos(x, m_3)$	0.2	0.4	0.1
$\max_{j \neq k} g_i(x, M) =$	0.3	0.5	0.8
$d_k(x, M) =$	-0.6	-0.1	0.5
$l_k(x, M) =$ ( $\gamma=5$ )	0.047	0.378	0.924

Table 1. Examples of correct classification and misclassification. \* denotes the scores of the correct class.

For each near-synonym set, an  $F \times K$  feature-class matrix, denoted as  $M$ , is created for classification. The rows represent the  $F$  distinct words (features) and the columns represent the  $K$  near-synonyms (classes) in the same set. Each entry in the matrix,  $m_{ij}$ , represents a weight of word  $i$  respect to near-synonym  $j$ , which is computed as the number of times word  $i$  appears in the contexts of near-synonym  $j$  divided by the total number of context words of near-synonym  $j$ . This frequency-based weight can then be transformed into a probabilistic form, i.e., divided by the sum of the weights of word  $i$  respect to all near-synonyms. Each test sentence is also transformed into an  $F$ -dimensional feature vector. Let  $x = [x_1, \dots, x_i, \dots, x_F]$  denote the feature vector of an input sentence. The classification is performed by computing the cosine similarity between  $x$  and the column vectors (near-synonyms) in the matrix, defined as

$$\begin{aligned}
NS_j^\wedge &= \arg \max_j \cos(x, m_j) \\
&= \arg \max_j \frac{x \cdot m_j}{\|x\| \|m_j\|} \\
&= \arg \max_j \frac{\sum_{i=1}^F x_i m_{ij}}{\sqrt{\sum_{i=1}^F x_i^2} \sqrt{\sum_{i=1}^F m_{ij}^2}},
\end{aligned} \tag{6}$$

where  $m_j$  is the  $j$ -th column vector in the matrix  $M$ . The near-synonym with the highest cosine similarity score,  $NS_j^\wedge$ , is the predicted class of the classifier.

### 3.2 Minimum classification error criterion

According to the decision rule of the classifier, a classification error will occur when the near-synonym with the highest cosine score is not the correct class. Table 1 shows some examples, where Example 3 is an example of misclassification. On the other hand, although Example 2 is a correct classification, it might be an ambiguous case to classifiers since the scores are close among classes. Therefore, if we can increase the separation of the correct class from its competing ones, then the classification performance can be improved accordingly. This can be accomplished by adjusting the feature weights of the matrix  $M$  that have direct influence on the computation of cosine scores. The discriminative training performs the adjustment in the training phase according to the minimum classification error criterion. The detailed steps are as follows.

Given an input vector  $x$ , the classifier computes the cosine scores between  $x$  and each class using (6). The discriminant function for a class can thus be defined as the cosine measure; that is,

$$g_j(x, M) = \cos(x, m_j). \tag{7}$$

where  $j$  denotes a class in  $K$ . Since the correct class of each input vector is known in the training phase, we can determine whether or not the input vector is misclassified by comparing the discriminant function (cosine score) of the correct class against its competing classes. In the case of misclassification, the cosine score of the correct class will be smaller than the competing cosine scores. Let  $k$  be the correct class of  $x$ , the misclassification function can be defined as

$$d_k(x, M) = -g_k(x, M) + G_k(x, M), \tag{8}$$

where  $g_k(x, M)$  is the discriminant function for the correct class  $k$ , and  $G_k(x, M)$  is the anti-discriminant function that represents the other  $K-1$  competing classes, defined as

$$G_k(x, M) = \left[ \frac{1}{K-1} \sum_{j \neq k} g_j(x, M)^\eta \right]^{\frac{1}{\eta}}, \tag{9}$$

When  $\eta = 1$ , the anti-discriminant function  $G_k(x, M)$  is the average of all the competing cosine scores. With the increase of  $\eta$ ,  $G_k(x, M)$  is gradually dominated by the biggest

competing class. In the extreme case, i.e.,  $\eta \rightarrow \infty$ , the anti-discriminant function becomes

$$G_k(x, M) = \max_{j \neq k} g_j(x, M). \quad (10)$$

The misclassification function in (8) can thus be rewritten as

$$d_k(x, M) = -g_k(x, M) + \max_{j \neq k} g_j(x, M), \quad (11)$$

In this case, the classification error is determined by comparing the discriminant function of the correct class against that of the biggest competing class. Obviously,  $d_k(x, M) > 0$  implies a classification error. For instance, in Example 3, the discriminant function for the correct class is  $g_2(x, M) = 0.3$ , and that of the biggest competing class is  $\max(g_1(x, M), g_3(x, M)) = 0.8$ , thus the classification error is  $d_k(x, M) = 0.5$ . On the other hand, the classification error will be a negative value for correct classifications, as shown in Example 1 and 2.

Intuitively, a greater classification error also results in a greater loss. We herein use the sigmoid function as the class loss function; that is,

$$l_k(x, M) = l(d_k) = \frac{1}{1 + \exp^{-\gamma d_k}}, \quad (12)$$

where  $\gamma$  is a constant that controls the slope of the sigmoid function. The sigmoid function maps the values of classification error within the range of 0 to 1. For correct classifications, a greater separation of the correct class from the biggest competing class leads to a greater negative value of  $d_k$ , i.e., a smaller classification error, resulting in a smaller loss tends asymptotically to 0 (Example 1), whereas a moderate loss is yielded if the separation was close (Example 2). For the cases of misclassification, a greater separation leads to a greater positive value of  $d_k$ , i.e., a greater classification error, resulting in a greater loss tends asymptotically to 1 (Example 3). The overall loss of the entire training set can then be estimated as

$$L(M) = \frac{1}{Q} \sum_{k=1}^K \sum_{x \in C_k} l_k(x, M), \quad (13)$$

where  $C_k$  denotes the set of training vectors of class  $k$ , and  $Q$  is the total number of vectors in the training set. The goal now is to minimize the loss. According to the above discussions on the

three examples, to minimize the loss is to minimize the classification error, and to improve the separation of the correct class against its competing classes. This can be accomplished by adjusting the feature weights of the matrix  $M$  to distinguish positive and negative features for each class. We herein adopt a gradient descent method such as the generalized probabilistic descent (GPD) algorithm (Katagiri et al., 1998) to update the matrix  $M$ . The detailed steps are as follows.

Let the feature weights of the matrix  $M$  be the parameter set to be adjusted. The adjustment is performed iteratively according to the following update formula.

$$M^{(t+1)} = M^{(t)} - \varepsilon_t \nabla l_k(x^{(t)}, M^{(t)}), \quad (14)$$

where  $t$  denotes the  $t$ -th iteration,  $\varepsilon_t$  is an adjustment step of a small positive real number, and  $\nabla l_k(x^{(t)}, M^{(t)})$  is the gradient of the loss function, which is computed by the following two parts

$$\nabla l_k(x^{(t)}, M^{(t)}) = \frac{\partial l_k}{\partial d_k} \frac{\partial d_k(x^{(t)}, M^{(t)})}{\partial m_{ij}}, \quad (15)$$

where

$$\frac{\partial l_k}{\partial d_k} = \gamma l_k(d_k)(1 - l_k(d_k)), \quad (16)$$

and from (7), (8), and (9),

$$\frac{\partial d_k(x^{(t)}, M^{(t)})}{\partial m_{ij}} = \begin{cases} -x_i, & \text{if } j = k \\ x_i \frac{G_k(x^{(t)}, M) g_j(x^{(t)}, M)^{\eta-1}}{\sum_{j \neq k} g_j(x^{(t)}, M)^\eta}, & \text{if } j \neq k \end{cases}, \quad (17)$$

where  $x_i$  is an element of the input vector  $x$ . By replacing  $\nabla l_k(x_t, M_t)$  in (14) with the two parts in (15), at each iteration each feature weight  $m_{ij}$  in  $M$  is adjusted by

$$m_{ij}^{(t+1)} = \begin{cases} m_{ij}^{(t)} + \varepsilon_t \frac{\partial l_k}{\partial d_k} x_i, & \text{if } j = k \\ m_{ij}^{(t)} - \varepsilon_t \frac{\partial l_k}{\partial d_k} x_i \frac{G_k(x^{(t)}, M) g_j(x^{(t)}, M)^{\eta-1}}{\sum_{j \neq k} g_j(x^{(t)}, M)^\eta}, & \text{if } j \neq k \end{cases}. \quad (18)$$

The weight  $x_i$  represents whether or not a dimension word occurs in an input sentence. A zero

weight indicates that the dimension word does not occur in the input sentence, thus the corresponding dimension of each column vector will not be adjusted. On the contrary, the corresponding dimension of the column vector of the correct class ( $j = k$ ) is adjusted by adding a value, while those of the competing classes ( $j \neq k$ ) are adjusted by subtracting a value from them. After a sequence of adjustments over the training set, the positive and negative features can be distinguished by adjusting their weights that result in a greater positive or negative value for each of them. The separation of the correct class against its competing ones can thus be increased.

The weight adjustment in (18) is in proportion to the adjustment step  $\varepsilon_t$  and the slope of the sigmoid function  $\partial l_k / \partial d_k$ . The adjustment step  $\varepsilon_t$  can be determined empirically. As (16) shows, the slope  $\partial l_k / \partial d_k$  converges asymptotically to zero as the classification error  $d_k$  approaches to a very large (or small) value. This leads to a small weight adjustment. For instance, the weight adjustment in Example 1 is small due to a small value of  $d_k$ , or, say, due to a large separation between the correct class and its competing ones. This is reasonable because classifiers often perform well in such cases. Similarly, the weight adjustment in Example 3 (misclassification) is also small due to a large value of  $d_k$ . A greater adjustment is not employed because such a large separation is difficult to be reduced significantly. Additionally, over-adjusting some features may introduce negative effects on other useful features in the matrix. Therefore, discriminative training is more effective on the cases with a moderate value of  $d_k$ , like Example 2. Such cases usually fall within the decision boundary and tend to be confusing to classifiers. Hence, improving the separation on such cases helps significantly improve the classification performance.

## 4 Experimental Results

### 4.1 Experiment setup

**1) Data:** The near-synonym sets used for experiments included the seven sets (Exp1) and the eleven sets (Exp2) used in the previous studies (Edmonds, 1997; Inkpen, 2007; Gardiner and Dras, 2007), as shown in Table 2. The Web 1T 5-gram corpus was used to build classifiers,

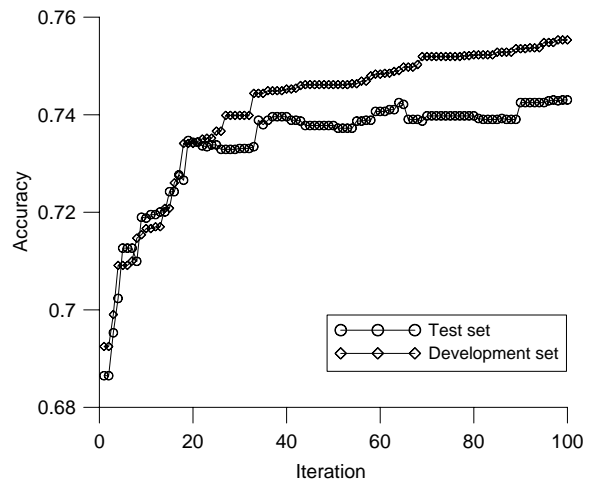


Figure 2. The change of classification accuracy during discriminative training.

where the corpus was randomly split into a training set, a development set, and a test set with an 8:1:1 ratio. For efficiency consideration, we randomly sampled up to 100 5-grams from the test set for each near-synonym. This sampling procedure was repeated five times for evaluation of the classifiers.

**2) Classifiers:** The classifiers were implemented using PMI, n-grams, and discriminative training (DT) methods, respectively.

**PMI:** Given a near-synonym set and a test 5-gram with a gap, the PMI scores for each near-synonym were calculated using (3), where the size of the context window  $k$  was set to 2. The frequency counts between each near-synonym and its context words were retrieved from the training set.

**NGRAM:** For each test 5-gram with a gap, all possible 4-grams containing the gap were first extracted (excluding punctuation marks). The averaged 4-gram scores for each near-synonym were then calculated using (5). Again, the frequency counts of the 4-grams were retrieved from the training set.

**DT:** For each near-synonym set, the matrix  $M$  was built from the training set. Each 5-gram in the development set was taken as input to iteratively compute the cosine score, loss, classification error, respectively, and finally to adjust the feature weights of  $M$ . The parameters of DT included  $\eta$  for the anti-discriminative function,  $\gamma$

No.	Near-synonym set	No. of cases	Accuracy (%)			
			NGRAM	PMI	COS	DT
1.	difficult, hard, tough	300	58.60	61.67	60.13	63.13
2.	error, mistake, oversight	300	68.47	78.33	77.20	79.20
3.	job, task, duty	300	68.93	70.40	74.00	75.67
4.	responsibility, burden, obligation, commitment	400	69.80	66.95	68.75	69.55
5.	material, stuff, substance	300	70.20	67.93	71.07	75.13
6.	give, provide, offer	300	58.87	66.47	64.13	68.27
7.	settle, resolve	200	69.30	68.10	77.10	84.10
<b>Exp1</b>		<b>2,100</b>	<b>66.33</b>	<b>68.50</b>	<b>69.94</b>	<b>72.89</b>
1.	benefit, advantage, favor, gain, profit	500	71.44	69.88	69.44	71.36
2.	low, gush, pour, run, spout, spurt, squirt, stream	800	65.45	65.00	68.68	71.08
3.	deficient, inadequate, poor, unsatisfactory	400	65.65	69.40	70.35	74.35
4.	afraid, aghast, alarmed, anxious, apprehensive, fearful, frightened, scared, terror-stricken*	789	49.84	44.74	47.00	49.33
5.	disapproval, animadversion*, aspersion*, blame, criticism, reprehension*	300	72.80	79.47	80.00	82.53
6.	mistake, blooper, blunder, boner, contretemps*, error, faux pas*, goof, slip, solecism*	618	62.27	59.61	68.41	71.65
7.	alcoholic, boozier*, drunk, drunkard, lush, sot	433	64.90	80.65	77.88	84.34
8.	leave, abandon, desert, forsake	400	65.85	66.05	69.35	74.35
9.	opponent, adversary, antagonist, competitor, enemy, foe, rival	700	58.51	59.51	63.31	67.14
10.	thin, lean, scrawny, skinny, slender, slim, spare, svelte, willowy*, wiry	734	57.74	61.99	55.72	64.58
11.	lie, falsehood, fib, prevarication*, rationalization, untruth	425	57.55	63.58	69.46	74.21
<b>Exp2</b>		<b>6,099</b>	<b>61.69</b>	<b>63.32</b>	<b>65.15</b>	<b>69.26</b>

Table 2. Accuracy of classifiers on Exp1 (7 sets) and Exp2 (11 sets). The words marked with \* are excluded from the experiments because their 5-grams are very rare in the corpus.

for the sigmoid function, and  $\varepsilon_i$  for the adjustment step. The settings,  $\eta = 25$ ,  $\gamma = 35$ , and  $\varepsilon_i = 10^{-3}$ , were determined by performing DT for several iterations through the training set. These settings were used for the following experiments.

**3) Evaluation metric:** The answers proposed by each classifier are the near-synonyms with the highest score. The correct answers are the near-synonyms originally in the gap of the test 5-grams. The performance is measured by the accuracy, which is defined as the number of correct answers made by each classifier, divided by the total number of test 5-grams.

In the following sections, we first demonstrate the effect of DT on classification performance, followed by the comparison of the classifiers.

#### 4.2 Evaluation on discriminative training

This experiment is to investigate the performance change during discriminative training. Figure 2 shows the accuracy at the first 100 iterations for both development set and test set, with the 8th set in Exp2 as an example. The accuracy increased rapidly in the first 20 iterations, and stabilized after the 40th iteration. The discriminative training is stopped until the accuracy has not been changed over 30 iterations or the 300th iteration has been reached.

Features	Near-synonym set		
	mistake	error	oversight
made	<b>0.076</b>	-0.004	-0.005
biggest	<b>0.074</b>	-0.001	-0.004
message	-0.004	<b>0.039</b>	-0.010
internal	0.001	<b>0.026</b>	-0.001
supervision	-0.001	-0.006	<b>0.031</b>
audit	-0.002	-0.003	<b>0.028</b>

Table 3. Example of feature weights after discriminative training.

### 4.3 Comparative results

Table 2 shows the comparative results of the classification accuracy for the 18 near-synonym sets (Exp1 + Exp2). The accuracies for each near-synonym set were the average accuracies of the five randomly sampled test sets. The cosine measure without discrimination training (COS) was also considered for comparison. The results show that NGRAM performed worst among the four classifiers. The major reason is that not all 4-grams of the test examples can be found in the corpus. Instead of contiguous word associations used by NGRAM, PMI considers the words in the contexts independently to select the best synonyms. The results show that PMI achieved better performance than NGRAM. The two supervised methods, COS and DT, outperformed the two unsupervised methods, NGRAM and PMI. As indicated in the bold numbers, using the supervised method alone (without DT), COS yielded higher average accuracy by 5% and 2% over NGRAM and PMI, respectively, on Exp1, and by 6% and 3%, respectively, on Exp2. When DT was employed, the average accuracy was further improved by 4% and 6% on Exp1 and Exp2, respectively, compared with COS.

The use of DT can improve the classification accuracy mainly because it can adjust the feature weights iteratively to improve the separation between the correct class and its competing ones, which helps tackle the ambiguous test examples that fall within the decision boundary. Table 3 presents several positive and negative features for the near-synonym set {mistake, error, oversight}. The feature weights were adjusted ac-

Exp1	Rank 1	Rank 2	Diff.
NGRAM	66.33%	79.35%	+19.63%
PMI	68.50%	88.99%	+29.91%
COS	69.94%	89.93%	+28.58%
DT	72.89%	91.06%	+24.93%
Exp2	Rank 1	Rank 2	Diff.
NGRAM	61.69%	68.48%	+11.01%
PMI	63.32%	79.11%	+24.94%
COS	65.15%	80.52%	+23.59%
DT	69.26%	82.86%	+19.64%

Table 4. Accuracy of Rank 1 and Rank 2 for each classifier.

ording to their contributions to discriminating among the near-synonyms in the set. For instance, the features “made” and “biggest” both received a positive value for the class “mistake”, and a negative value for the competing classes “error” and “oversight”. These positive and negative weights help distinguish useful features from noisy ones for classifier training. On the other hand, if the feature weights were evenly distributed among the classes, these features would not be unlikely to contribute to the classification performance.

### 4.4 Accuracy of Rank 1 and Rank 2

The accuracy presented in Table 2 was computed based on the classification results at Rank 1, i.e., a test sample was considered correctly classified only if the near-synonym with the highest score was the word originally in the gap of the test sample. Similarly, the accuracy at Rank 2 can be computed by considering the top two near-synonyms proposed by each classifier. That is, if either the near-synonym with the highest or the second highest score was the correct answer, the test sample was considered correctly classified. Table 4 shows the accuracy of Rank 1 and Rank 2 for each classifier. The results show that the improvement of Rank 1 accuracy on Exp1 was about 20 to 30 percentage points, and was 25.76 in average. For Exp2, the average improvement was 19.80 percentage points



## 5 Conclusion

This work has presented the use of discriminative training for near-synonym substitution. The discriminative training can improve classification performance by iteratively re-weighting the positive and negative features for each class. This helps improve the separation of the correct class against its competing ones, making classifiers more effective on ambiguous cases close to the decision boundary. Experimental results show that the supervised discriminative training technique achieves higher accuracy than the two unsupervised methods, the PMI-based and n-gram-based methods. The availability of a large labeled training set also encourages the use of the proposed supervised method.

Future work will investigate on the use of multiple features for discriminating among near-synonyms. For instance, the predicate-argument structure, which can capture long-distance information, will be combined with currently used local contextual features to boost the classification performance. More experiments will also be conducted to evaluate classifiers' ability to rank multiple answers.

## References

- J. Bhogal, A. Macfarlane, and P. Smith. 2007. A Review of Ontology based Query Expansion. *Information Processing & Management*, 43(4):866-886.
- K. Church and P. Hanks. 1991. Word Association Norms, Mutual Information and Lexicography. *Computational Linguistics*, 16(1):22-29.
- I. Dagan, O. Glickman, A. Gliozzo, E. Marmorshtein, and C. Strapparava. 2006. Direct Word Sense Matching for Lexical Substitution. In *Proc. of COLING/ACL-06*, pages 449-456.
- P. Edmonds. 1997. Choosing the Word Most Typical in Context Using a Lexical Co-occurrence Network. In *Proc. of ACL-97*, pages 507-509.
- C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.
- M. Gardiner and M. Dras. 2007. Exploring Approaches to Discriminating among Near-Synonyms, In *Proc. of the Australasian Technology Workshop*, pages 31-39.
- E. H. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. 2006. OntoNotes: The 90% Solution. In *Proc. of HLT/NAACL-06*, pages 57-60.
- D. Inkpen. 2007. Near-Synonym Choice in an Intelligent Thesaurus. In *Proc. of NAACL/HLT-07*, pages 356-363.
- D. Inkpen and G. Hirst. 2006. Building and Using a Lexical Knowledge-base of Near-Synonym Differences. *Computational Linguistics*, 32(2):1-39.
- S. Katagiri, B. H. Juang, and C. H. Lee. 1998. Pattern Recognition Using a Family of Design Algorithms based upon the Generalized Probabilistic Descent Method, *Proc. of the IEEE*, 86(11):2345-2373.
- H. K. J. Kuo and C. H. Lee. 2003. Discriminative Training of Natural Language Call Routers, *IEEE Trans. Speech and Audio Processing*, 11(1):24-35.
- D. Lin. 1998. Automatic Retrieval and Clustering of Similar Words. In *Proc. of ACL/COLING-98*, pages 768-774.
- D. McCarthy. 2002. Lexical Substitution as a Task for WSD Evaluation. In *Proc. of the SIGLEX/SENSEVAL Workshop on Word Sense Disambiguation at ACL-02*, pages 109-115.
- D. Moldovan and R. Mihalcea. 2000. Using Wordnet and Lexical Operators to Improve Internet Searches. *IEEE Internet Computing*, 4(1):34-43.
- U. Ohler, S. Harbeck, and H. Niemann. 1999. Discriminative Training of Language Model Classifiers, In *Proc. of Eurospeech-99*, pages 1607-1610.
- D. Pearce. 2001. Synonymy in Collocation Extraction. In *Proc. of the Workshop on WordNet and Other Lexical Resources at NAACL-01*.
- S. Pradhan, E. H. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. 2007. OntoNotes: A Unified Relational Semantic Representation. In *Proc. of the First IEEE International Conference on Semantic Computing (ICSC-07)*, pages 517-524.
- H. Rodríguez, S. Climent, P. Vossen, L. Bloksma, W. Peters, A. Alonge, F. Bertagna, and A. Roventint. 1998. The Top-Down Strategy for Building EuroWordNet: Vocabulary Coverage, Base Concepts and Top Ontology, *Computers and the Humanities*, 32:117-159.
- L. C. Yu, C. H. Wu, A. Philpot, and E. H. Hovy. 2007. OntoNotes: Sense Pool Verification Using Google N-gram and Statistical Tests, In *Proc. of the OntoLex Workshop at the 6th International Semantic Web Conference (ISWC-07)*.
- D. Zhou and Y. He. 2009. Discriminative Training of the Hidden Vector State Model for Semantic Parsing, *IEEE Trans. Knowledge and Data Engineering*, 21(1):66-77.