# Improved Natural Language Learning via Variance-Regularization Support Vector Machines

**Shane Bergsma**
University of Alberta
sbergsma@ualberta.ca

**Dekang Lin**
Google, Inc.
lindek@google.com

**Dale Schuurmans**
University of Alberta
dale@cs.ualberta.ca

## Abstract

We present a simple technique for learning better SVMs using fewer training examples. Rather than using the standard SVM regularization, we regularize toward low weight-variance. Our new SVM objective remains a convex quadratic function of the weights, and is therefore computationally no harder to optimize than a standard SVM. Variance regularization is shown to enable dramatic improvements in the learning rates of SVMs on three lexical disambiguation tasks.

## 1 Introduction

Discriminative training is commonly used in NLP and speech to scale the contribution of different models or systems in a combined predictor. For example, discriminative training can be used to scale the contribution of the language model and translation model in machine translation (Och and Ney, 2002). Without training data, it is often reasonable to weight the different models equally. We propose a simple technique that exploits this intuition for better learning with fewer training examples. We regularize the feature weights in a Support Vector Machine (Cortes and Vapnik, 1995) toward a low-variance solution. Since the new SVM quadratic program is convex, it is no harder to optimize than the standard SVM objective.

When training data is generated through human effort, faster learning saves time and money. When examples are labeled automatically, through user feedback (Joachims, 2002) or from textual pseudo-examples (Smith and Eisner, 2005; Okanohara and Tsujii, 2007), faster learning can reduce the lag before a new system is useful.

We demonstrate faster learning on lexical disambiguation tasks. For these tasks, a system predicts a label for a word in text, based on the word's context. Possible labels include part-of-speech tags, named-entity types, and word senses. A number of disambiguation systems make predictions with the help of N-gram counts from a web-scale auxiliary corpus, typically via a search-engine (Lapata and Keller, 2005) or N-gram corpus (Bergsma et al., 2009). When discriminative training is used to weigh the counts for classification, many of the learned feature weights have similar values. Good weights have low variance.

For example, consider the task of preposition selection. A system selects the most likely preposition given the context, and flags a possible error if it disagrees with the user's choice:

- I worked in Russia **from** 1997 to 2001.
- I worked in Russia ***during** 1997 to 2001.

Bergsma et al. (2009) use a variety of web counts to predict the correct preposition. They have features for COUNT(*in Russia* from), COUNT(*Russia* from *1997*), COUNT(from *1997 to*), etc. If these are high, **from** is predicted. Similarly, they have features for COUNT(*in Russia* during), COUNT(*Russia* during *1997*), COUNT(during *1997 to*). These features predict **during**. All counts are in the log domain. The task has thirty-four different prepositions to choose from. A 34-way classifier is trained on examples of correct preposition usage; it learns which context positions and sizes are most reliable and assigns feature weights accordingly.

A very strong unsupervised baseline, however, is to simply weight all the count features equally. In fact, in Bergsma et al. (2009), the supervised approach requires over 30,000 training examples before it outperforms this baseline. In contrast, we show that by regularizing a classifier toward equal weights, a supervised predictor outperforms the unsupervised approach after only ten examples, and does as well with 1000 examples as the standard classifier does with 100,000.

Section 2 first describes a general multi-class SVM. We call the base vector of information used by the SVM the *attributes*. A standard multi-class SVM creates features for the cross-product of attributes and classes. E.g., the attribute COUNT(*Russia* during *1997*) is not only a feature for predicting the preposition **during**, but also for predicting the 33 other prepositions. The SVM must therefore learn to disregard many irrelevant features. We observe that this is not necessary, and develop an SVM that only uses the relevant attributes in the score for each class. Building on this efficient framework, we incorporate variance regularization into the SVM's quadratic program.

We apply our algorithms to three tasks: preposition selection, context-sensitive spelling correction, and non-referential pronoun detection (Section 4). We reproduce Bergsma et al. (2009)'s results using a multi-class SVM. Our new models achieve much better accuracy with fewer training examples. We also exceed the accuracy of a reasonable alternative technique for increasing the learning rate: including the output of the unsupervised system as a feature in the SVM.

Variance regularization is an elegant addition to the suite of methods in NLP that improve performance when access to labeled data is limited. Section 5 discusses some related approaches. While we motivate our algorithm as a way to learn better weights when the features are counts from an auxiliary corpus, there are other potential uses of our method. We outline some of these in Section 6, and note other directions for future research.

## 2 Three Multi-Class SVM Models

We describe three max-margin multi-class classifiers and their corresponding quadratic programs. Although we describe linear SVMs, they can be extended to nonlinear cases in the standard way by writing the optimal function as a linear combination of kernel functions over the input examples.

In each case, after providing the general technique, we relate the approach to our motivating application: learning weights for count features in a discriminative web-scale N-gram model.

### 2.1 Standard Multi-Class SVM

We define a $K$-class SVM following Crammer and Singer (2001). This is a generalization of binary SVMs (Cortes and Vapnik, 1995). We have a set $\{(\bar{x}^1, y^1), ..., (\bar{x}^M, y^M)\}$ of $M$ training examples. Each $\bar{x}$ is an $N$-dimensional attribute vector, and $y \in \{1, ..., K\}$ are classes. A classifier, $H$, maps an attribute vector, $\bar{x}$, to a class, $y$. $H$ is parameterized by a $K$-by-$N$ matrix of weights, $\mathbf{W}$:

$$H_{\mathbf{W}}(\bar{x}) = \operatorname*{argmax}_{r=1}^{K}\{\bar{W}_r \cdot \bar{x}\} \qquad (1)$$

where $\bar{W}_r$ is the $r$th row of $\mathbf{W}$. That is, the predicted label is the index of the row of $\mathbf{W}$ that has the highest inner-product with the attributes, $\bar{x}$.

We seek weights such that the classifier makes few errors on training data and generalizes well to unseen data. There are $KN$ weights to learn, for the cross-product of attributes and classes. The most common approach is to train $K$ separate one-versus-all binary SVMs, one for each class. The weights learned for the $r$th SVM provide the weights $\bar{W}_r$ in (1). We call this approach **OvA-SVM**. Note in some settings various one-versus-one strategies may be more effective than one-versus-all (Hsu and Lin, 2002).

The weights can also be found using a single constrained optimization (Vapnik, 1998; Weston and Watkins, 1998). Following the soft-margin version in Crammer and Singer (2001):

$$\begin{aligned} &\min_{\mathbf{W},\xi^1,\ldots,\xi^M} && \frac{1}{2}\sum_{i=1}^{K}||\bar{W}_i||^2 + C\sum_{i=1}^{m}\xi^i \\ &\text{subject to :} && \xi^i \geq 0 \\ &\forall r \neq y^i, && \bar{W}_{y^i}\cdot\bar{x}^i - \bar{W}_r\cdot\bar{x}^i \geq 1-\xi^i \qquad (2) \end{aligned}$$

The constraints require the correct class to be scored higher than other classes by a certain margin, with slack for non-separable cases. Minimizing the weights is a form of regularization. Tuning the $C$-parameter controls the emphasis on regularization versus separation of training examples.

We call this the **K-SVM**. The K-SVM outperformed the OvA-SVM in Crammer and Singer (2001), but see Rifkin and Klautau (2004). The popularity of K-SVM is partly due to convenience; it is included in popular SVM software like SVM-*multiclass*[1] and LIBLINEAR (Fan et al., 2008).

Note that with two classes, K-SVM is less efficient than a standard binary SVM. A binary classifier outputs class 1 if $(\bar{w} \cdot \bar{x} > 0)$ and class 2 otherwise. The K-SVM encodes a binary classifier using $\bar{W}_1 = \bar{w}$ and $\bar{W}_2 = -\bar{w}$, therefore requiring twice the memory of a binary SVM. However, both binary and 2-class formulations have the same solution (Weston and Watkins, 1998).

[1] http://svmlight.joachims.org/svm_multiclass.html

#### 2.1.1 Web-Scale N-gram K-SVM

K-SVM was used with N-gram models in Bergsma et al. (2009). For preposition selection, attributes were web counts of patterns filled with 34 prepositions, corresponding to the 34 classes. Each preposition serves as the *filler* of each *context pattern*. Fourteen patterns were used for each filler: all five 5-grams, four 4-grams, three 3-grams, and two 2-grams spanning the position to be predicted. There are $N = 14{*}34 = 476$ total attributes, and therefore $KN = 476 * 34 = 16184$ weights in **W**.

This K-SVM classifier can potentially exploit very subtle information. Let $\bar{W}_{in}$ and $\bar{W}_{before}$ be weights for the classes **in** and **before**. Notice some of the attributes weighted in the inner products $\bar{W}_{before} \cdot \bar{x}$ and $\bar{W}_{in} \cdot \bar{x}$ will be for counts of the preposition *after*. Relatively high counts for a context with *after* should deter us from choosing **in** more than from choosing **before**. These correlations can be encoded in the classifier via the corresponding weights on *after*-counts in $\bar{W}_{in}$ and $\bar{W}_{before}$. How useful are these correlations and how much training data is needed before they can be learned and exploited effectively?

We next develop a model that, for each class, only scores those attributes deemed to be directly relevant to the class. Our experiments thus empirically address these questions for different tasks.

### 2.2 SVM with Class-Specific Attributes

Suppose we can partition our attribute vectors into sub-vectors that only include attributes that we declare as relevant to the corresponding class: $\bar{x} = (\bar{x}_1, ..., \bar{x}_K)$. We develop a classifier that only uses the class-specific attributes in the score for each class. The classifier uses an $N$-dimensional weight vector, $\bar{w}$, which follows the attribute partition, $\bar{w} = (\bar{w}_1, ..., \bar{w}_K)$. The classifier is:

$$H_{\bar{w}}(\bar{x}) = \underset{r=1}{\overset{K}{\operatorname{argmax}}} \{\bar{w}_r \cdot \bar{x}_r\} \tag{3}$$

We call this classifier the **CS-SVM** (an SVM with **C**lass-**S**pecific attributes).

The weights can be determined using the follow (soft-margin) optimization:

$$\begin{aligned} \min_{\bar{w},\xi^1,...,\xi^m} \quad & \frac{1}{2}\bar{w}^T\bar{w} + C\sum_{i=1}^{m}\xi^i \\ \text{subject to :} \quad & \xi^i \geq 0 \\ \forall r \neq y^i, \quad & \bar{w}_{y^i} \cdot \bar{x}^i_{y^i} - \bar{w}_r \cdot \bar{x}^i_r \geq 1 - \xi^i \end{aligned} \tag{4}$$

There are several advantages to this formulation. Foremost, rather than having $KN$ weights, it can have only $N$. For linear classifiers, the number of examples needed to reach optimum performance is at most linear in the number of weights (Vapnik, 1998; Ng and Jordan, 2002). In fact, both the total number and number of *active* features per example decrease by $K$. Thus this reduction saves far more memory than what could be obtained by an equal reduction in dimensionality via pruning infrequent attributes.

Also, note that unlike the K-SVM (Section 2.1), in the binary case the CS-SVM is completely equivalent (thus equally efficient) to a standard SVM.

We will not always *a priori* know the class associated with each attribute. Also, some attributes may be predictive of multiple classes. In such cases, we can include ambiguous attributes in every sub-vector (needing $N$+$D(K$-$1)$ total weights if $D$ attributes are duplicated). In the degenerate case where every attribute is duplicated, CS-SVM is equivalent to K-SVM; both have $KN$ weights.

#### 2.2.1 Optimization as a Binary SVM

We could solve the optimization problem in (4) directly using a quadratic programming solver. However, through an equivalent transformation into a binary SVM, we can take advantage of efficient, custom SVM optimization algorithms.

We follow Har-Peled et al. (2003) in transforming a multi-class example into a set of binary examples, each specifying a constraint from (4). We extend the attribute sub-vector corresponding to each class to be $N$-dimensional. We do this by substituting zero-vectors for all the other sub-vectors in the partition. The attribute vector for the $r$th class is then $\bar{z}_r = (\bar{0}, ..., \bar{0}, \bar{x}_r, \bar{0}, ..., \bar{0})$. This is known as Kesler's Construction and has a long history in classification (Duda and Hart, 1973; Crammer and Singer, 2003). We then create binary rank constraints for a ranking SVM (Joachims, 2002) (ranking SVMs reduce to standard binary SVMs). We create $K$ instances for each multi-class example $(\bar{x}^i, y^i)$, with the transformed vector of the true class, $\bar{z}_{y^i}$, assigned a higher-rank than all the other, equally-ranked classes, $\bar{z}_{\{r \neq y^i\}}$. Training a ranking SVM using these constraints gives the same weights as solving (4), but allows us to use efficient, custom SVM software.[2] Note the K-SVM

[2] One subtlety is whether to use a single slack, $\xi^i$, for all K-1 constraints per example $i$ (Crammer and Singer, 2001), or a different slack for each constraint (Joachims, 2002). Us-

can also be trained this way, by including every attribute in every sub-vector, as described earlier.

#### 2.2.2 Web-Scale N-gram CS-SVM

Returning to our preposition selection example, an obvious attribute partition for the CS-SVM is to include as attributes for predicting preposition $r$ only those counts for patterns filled with preposition $r$. Thus $\bar{x}_{in}$ will only include counts for context patterns filled with *in* and $\bar{x}_{before}$ will only include counts for context patterns filled with *before*. With 34 sub-vectors and 14 attributes in each, there are only $14 * 34 = 476$ total weights. In contrast, K-SVM had $16184$ weights to learn.

It is instructive to compare the CS-SVM in (3) to the unsupervised SUMLM approach in Bergsma et al. (2009). That approach can be written as:

$$H(\bar{x}) = \underset{r=1}{\overset{K}{\operatorname{argmax}}}\{\bar{1} \cdot \bar{x}_r\} \tag{5}$$

where $\bar{1}$ is an $N$-dimensional vector of ones. This is CS-SVM with all weights set to unity. The counts for each preposition are simply summed, and whichever one scores the highest is taken as the output (actually only a subset of the counts are used, see Section 4.1). As mentioned earlier, this system performs remarkably well on several tasks.

### 2.3 Variance Regularization SVMs

Suppose we choose our attribute partition well and train the CS-SVM on a sufficient number of examples to achieve good performance. It is a reasonable hypothesis that the learned weights will be predominantly positive. This is because each sub-vector $\bar{x}_r$ was chosen to only include attributes that are predictive of class $r$. Unlike the classifier in (1) which weighs positive and negative evidence together for each class, in CS-SVM, negative evidence only plays a roll as it contributes to the score of competing classes.

If all the attributes are equally important, the weights should be equal, as in the unsupervised approach in (5). If some are more important than others, the training examples should reflect this and the learner can adjust the weights accordingly.[3] In the absence of this training evidence, it is reasonable to bias the classifier toward an equal-weight solution.

ing the former may be better as it results in a tighter bound on empirical risk (Tsochantaridis et al., 2005).

[3]E.g., the true preposition might be better predicted by the counts of patterns that tend to include the preposition's grammatical object, i.e., patterns that include more right-context.

Rather than the standard SVM regularization that minimizes the norm of the weights as in (4), we therefore regularize toward weights that have low variance. More formally, we can regard the set of weights, $w_1, ..., w_N$, as the distribution of a discrete random variable, $W$. We can calculate the mean and variance of this variable from its distribution. We seek a variable that has low variance.

We begin with a more general objective and then explain how a specific choice of covariance matrix, $\mathbf{C}$, minimizes the variance of the weights. We propose the regularizer:

$$\begin{aligned} &\min_{\bar{w},\xi^1,\ldots,\xi^m} && \frac{1}{2}\bar{w}^T\mathbf{C}\bar{w} + C\sum_{i=1}^{m}\xi^i \\ &\text{subject to}: && \xi^i \geq 0 \\ &\forall r \neq y^i, && \bar{w}_{y^i} \cdot \bar{x}^i_{y^i} - \bar{w}_r \cdot \bar{x}^i_r \geq 1 - \xi^i \end{aligned} \tag{6}$$

where $\mathbf{C}$ is a normalized covariance matrix such that $\sum_{i,j} C_{i,j} = 0$. This ensures uniform weight vectors receive zero regularization penalty. Since all covariance matrices are positive semi-definite, the quadratic program (QP) remains convex in $\bar{w}$, and thus amenable to general purpose QP-solvers.

Since the unsupervised system in (5) has zero weight variance, the SVM learned in (6) should do as least as well as (5) as we tune the $C$-parameter on development data. That is, as $C$ approaches zero, variance minimization becomes the sole objective of (6), and uniform weights are produced.

We use covariance matrices of the form:

$$\mathbf{C} = \operatorname{diag}(\bar{p}) - \bar{p}\bar{p}^T \tag{7}$$

where $\operatorname{diag}(\bar{p})$ is the matrix constructed by putting $\bar{p}$ on the main diagonal. Here, $\bar{p}$ is an arbitrary $N$-dimensional weighting vector, such that $p \geq 0$ and $\sum_i p_i = 1$. $\bar{p}$ dictates the contribution of each $w_i$ to the mean and variance of the weights in $\bar{w}$. It is easy to see that $\sum_{i,j} C_{i,j} = \sum_i p_i - \sum_i \sum_j p_i p_j = 0$.

We now show that $\bar{w}^T(\operatorname{diag}(\bar{p}) - \bar{p}\bar{p}^T)\bar{w}$ expresses the variance of the weights in $\bar{w}$ with respect to the probability weighting $\bar{p}$. The variance of a random variable with mean $E[W] = \mu$ is:

$$\operatorname{Var}[W] = E[(W-\mu)^2] = E[W^2] - E[W]^2$$

The mean of the weights using probability weighting $\bar{p}$ is $E[W] = \bar{w}^T\bar{p} = \bar{p}\bar{w}$. Also, $E[W^2] = \bar{w}^T\operatorname{diag}(\bar{p})\bar{w}$. Thus:

$$\begin{aligned} \operatorname{Var}[W] &= \bar{w}^T\operatorname{diag}(\bar{p})\bar{w} - (\bar{w}^T\bar{p})(\bar{p}\bar{w}) \\ &= \bar{w}^T(\operatorname{diag}(\bar{p}) - \bar{p}\bar{p})\bar{w} \end{aligned}$$

In our experiments, we deem each weight to be equally important to the variance calculation, and set $p_i = \frac{1}{N}, \forall i = 1, \ldots, N$.

The goal of the regularization in (6) using $\mathbf{C}$ from (7) can be regarded as directing the SVM toward a good unsupervised system, regardless of the constraints (training examples). In some unsupervised systems, however, only a subset of the attributes are used. In other cases, distinct subsets of weights should have low variance, rather than minimizing the variance across all weights. There are examples of these situations in Section 4.

We can account for these cases in our QP. We provide separate terms in our quadratic function for the subsets of $\bar{w}$ that should have low variance. Suppose we create $L$ subsets of $\bar{w}$: $\tilde{\omega}_1, ...\tilde{\omega}_L$, where $\tilde{\omega}_j$ is $\bar{w}$ with elements set to zero that are not in subset $j$. We then minimize $\frac{1}{2}(\tilde{\omega}_1^T \mathbf{C}_1 \tilde{\omega}_1 + ... + \tilde{\omega}_L^T \mathbf{C}_L \tilde{\omega}_L)$. If the terms in subset $j$ have low variance, $\mathbf{C}_j$ = $\mathbf{C}$ from (7) is used. If the subset corresponds to attributes that are not *a priori* known to be useful, an identity matrix can instead be used, $\mathbf{C}_j = \mathbf{I}$, and these weights will be regularized toward zero as in a standard SVM.[4]

Variance regularization therefore exploits extra knowledge by the system designer. The designer decides which weights should have similar values, and the SVM is biased to prefer this solution.

One consequence of being able to regularize different subsets of weights is that we can also apply variance regularization to the standard multiclass SVM (Section 2.1). We can use an identity $\mathbf{C}_i$ matrix for all *irrelevant* weights, i.e., weights that correspond to class-attribute pairs where the attribute is not directly relevant to the class. In our experiments, however, we apply variance regularization to the more efficient CS-SVM.

We refer to a CS-SVM trained using the variance minimization quadratic program as the **VAR-SVM**.

#### 2.3.1 Web-Scale N-gram VAR-SVM

If variance regularization is applied to all weights, attributes COUNT(*in Russia* during), COUNT(*Russia* during *1997*), and COUNT(during *1997 to*) will be encouraged to have similar weights in the score for class **during**. Furthermore, these will be weighted similarly to other patterns, filled with other prepositions, used in the scores for other classes.

Alternatively, we could minimize the variance separately over all 5-gram patterns, then over all 4-gram patterns, etc., or over all patterns with a filler in the same position. In our experiments, we took a very simple approach: we minimized the variance of all attributes that are weighted equally in the unsupervised baselines. If a feature is not included in a baseline, it is regularized toward zero.

## 3 Experimental Details

We use the data sets from Bergsma et al. (2009). These are the three tasks where web-scale N-gram counts were previously used as features in a standard K-SVM. In each case a classifier makes a decision for a particular word based on the word's surrounding context. The attributes of the classifier are the log counts of different fillers occurring in the context patterns. We retrieve counts from the web-scale Google Web 5-gram Corpus (Brants and Franz, 2006), which includes N-grams of length one to five. We apply add-one smoothing to all counts. Every classifier also has bias features (for every class). We simply include, where appropriate, attributes that are always unity.

We use LIBLINEAR (Fan et al., 2008) to train K-SVM and OVA-SVM, and SVM$^{rank}$ (Joachims, 2006) to train CS-SVM. For VAR-SVM, we solve the primal form of the quadratic program directly in CPLEX (2005), a general optimization package.

We vary the number of training examples for each classifier. The $C$-parameters of all SVMs are tuned on development data. We evaluate using **accuracy**: the percentage of test examples that are classified correctly. We also provide the accuracy of the majority-class baseline and best unsupervised system, as defined in Bergsma et al. (2009).

As an alternative way to increase the learning rate, we augment a classifier's features using the output of the unsupervised system: For each class, we include one feature for the sum of all counts (in the unsupervised system) that predict that class. We denote these augmented systems with a $+$ as in K-SVM$^+$ and CS-SVM$^+$.

## 4 Applications

### 4.1 Preposition Selection

Preposition errors are common among new English speakers (Chodorow et al., 2007). Systems that can reliably identify these errors are needed in word processing and educational software.

[4]Weights must appear in $\geq$1 subsets (possibly only in the $\mathbf{C}_j = \mathbf{I}$ subset). Each occurs in at most one in our experiments. Note it is straightforward to express this as a single covariance matrix regularizer over $\bar{w}$; we omit the details.

| System | Training Examples 10 | 100 | 1K | 10K | 100K |
|---|---|---|---|---|---|
| OvA-SVM | 16.0 | 50.6 | 66.1 | 71.1 | 73.5 |
| K-SVM | 13.7 | 50.0 | 65.8 | 72.0 | 74.7 |
| K-SVM$^+$ | 22.2 | 56.8 | 70.5 | 73.7 | **75.2** |
| CS-SVM | 27.1 | 58.8 | 69.0 | 73.5 | 74.2 |
| CS-SVM$^+$ | 39.6 | 64.8 | 71.5 | 74.0 | 74.4 |
| VAR-SVM | **73.8** | **74.2** | **74.7** | **74.9** | 74.9 |

Table 1: Accuracy (%) of preposition-selection SVMs. Unsupervised accuracy is 73.7%.

In our experiments, a classifier must choose the correct preposition among 34 candidates, using counts for filled 2-to-5-gram patterns. We use 100K training, 10K development, and 10K test examples. The unsupervised approach sums the counts of all 3-to-5-gram patterns for each preposition. We therefore regularize the variance of the 3-to-5-gram weights in VAR-SVM, and simultaneously minimize the norm of the 2-gram weights.

#### 4.1.1 Results

The majority-class is the preposition **of**; it occurs in 20.3% of test examples. The unsupervised system scores 73.7%. For further perspective on these results, note Chodorow et al. (2007) achieved 69% with 7M training examples, while Tetreault and Chodorow (2008) found the human performance was around 75%. However, these results are not directly comparable as they are on different data.

Table 1 gives the accuracy for different amounts of training data. Here, as in the other tasks, K-SVM mirrors the learning rate in Bergsma et al. (2009). There are several distinct phases among the relative ranking of the systems. For smaller amounts of training data ($\leq$1000 examples) K-SVM performs worst, while VAR-SVM is statistically significantly better than all other systems, and always exceeds the performance of the unsupervised approach.[5] Augmenting the attributes with sum counts (the + systems) strongly helps with fewer examples, especially in conjunction with the more efficient CS-SVM. However, VAR-SVM clearly helps more. We noted earlier that VAR-SVM is guaranteed to do as well as the unsupervised system on the development data, but here we confirm that it can also exploit even small amounts of training data to further improve accuracy.

CS-SVM outperforms K-SVM except with 100K examples, while OvA-SVM is better than K-SVM for small amounts of data.[6] K-SVM performs best with all the data; it uses the most expressive representation, but needs 100K examples to make use of it. On the other hand, feature augmentation and variance regularization provide diminishing returns as the amount of training data increases.

| System | Training Examples 10 | 100 | 1K | 10K | 100K |
|---|---|---|---|---|---|
| CS-SVM | 86.0 | 93.5 | 95.1 | **95.7** | 95.7 |
| CS-SVM$^+$ | 91.0 | 94.9 | 95.3 | **95.7** | 95.7 |
| VAR-SVM | **94.9** | **95.3** | **95.6** | **95.7** | **95.8** |

Table 2: Accuracy (%) of spell-correction SVMs. Unsupervised accuracy is 94.8%.

### 4.2 Context-Sensitive Spelling Correction

Context-sensitive spelling correction, or real-word error/malapropism detection (Golding and Roth, 1999; Hirst and Budanitsky, 2005), is the task of identifying errors when a misspelling results in a real word in the lexicon, e.g., using *site* when *sight* or *cite* was intended. Contextual spell checkers are among the most widely-used NLP technology, as they are included in commercial word processing software (Church et al., 2007).

For every occurrence of a word in a pre-defined confusion set (e.g. {*cite, sight, cite*}), the classifier selects the most likely word from the set. We use the five confusion sets from Bergsma et al. (2009); four are binary and one is a 3-way classification. We use 100K training, 10K development, and 10K test examples for each, and average accuracy across the sets. All 2-to-5 gram counts are used in the unsupervised system, so the variance of all weights is regularized in VAR-SVM.

#### 4.2.1 Results

On this task, the majority-class baseline is much higher, 66.9%, and so is the accuracy of the top unsupervised system: 94.8%. Since four of the five sets are binary classifications, where K-SVM and CS-SVM are equivalent, we only give the accuracy of the CS-SVM (it does perform better on the one 3-way set). VAR-SVM again exceeds the unsupervised accuracy for all training sizes, and generally

[5]Significance is calculated using a $\chi^2$ test over the test set correct/incorrect contingency table.

[6]Rifkin and Klautau (2004) argue OvA-SVM is as good as K-SVM, but this is "predicated on the assumption that the classes are 'independent'," i.e., that examples from class 0 are no closer to class 1 than to class 2. This is not true of this task (e.g. $\bar{x}_{before}$ is closer to $\bar{x}_{after}$ than $\bar{x}_{in}$, etc.).

| System | Training Examples 10 | 100 | 1K |
|---|---|---|---|
| CS-SVM | 59.0 | 71.0 | 84.3 |
| CS-SVM$^+$ | 59.4 | 74.9 | **84.5** |
| VAR-SVM | **70.2** | 76.2 | **84.5** |
| VAR-SVM+FreeB | 64.2 | **80.3** | **84.5** |

Table 3: Accuracy (%) of non-referential detection SVMs. Unsupervised accuracy is 80.1%.

performs as well as the augmented CS-SVM$^+$ using an order of magnitude less training data (Table 2). Differences from $\leq$1K are significant.

### 4.3 Non-Referential Pronoun Detection

Non-referential detection predicts whether the English pronoun *it* refers to a preceding noun ("*it* lost money") or is used as a grammatical placeholder ("*it* is important to..."). This binary classification is a necessary but often neglected step for noun phrase coreference resolution (Paice and Husk, 1987; Bergsma et al., 2008; Ng, 2009).

Bergsma et al. (2008) use features for the counts of various fillers in the pronoun's context patterns. If *it* is the most common filler, the pronoun is likely non-referential. If other fillers are common (like *they* or *he*), it is likely a referential instance. For example, "*he* lost money" is common on the web, but "*he* is important to" is not. We use the same fillers as in previous work, and preprocess the N-gram corpus in the same way.

The unsupervised system picks non-referential if the difference between the summed count of *it* fillers and the summed count of *they* fillers is above a threshold (note this no longer fits (5), with consequences discussed below). We thus separately minimize the variance of the *it* pattern weights and the *they* pattern weights. We use 1K training, 533 development, and 534 test examples.

#### 4.3.1 Results

The most common class is **referential**, occurring in 59.4% of test examples. The unsupervised system again does much better, at 80.1%.

Annotated training examples are much harder to obtain for this task and we experiment with a smaller range of training sizes (Table 3). The performance of VAR-SVM exceeds the performance of K-SVM across all training sizes (bold accuracies are significantly better than either CS-SVM for $\leq$100 examples). However, the gains were not as large as we had hoped, and accuracy remains worse than the unsupervised system when not using all the training data. When using all the data, a fairly large C-parameter performs best on development data, so regularization plays less of a role.

After development experiments, we speculated that the poor performance relative to the unsupervised approach was related to class bias. In the other tasks, the unsupervised system chooses the highest summed score. Here, the difference in *it* and *they* counts is compared to a *threshold*. Since the bias feature is regularized toward zero, then, unlike the other tasks, using a low $C$-parameter does not produce the unsupervised system, so performance can begin below the unsupervised level.

Since we wanted the system to learn this threshold, even when highly regularized, we removed the regularization penalty from the bias weight, letting the optimization freely set the weight to minimize training error. With more freedom, the new classifier (VAR-SVM+FreeB) performs worse with 10 examples, but exceeds the unsupervised approach with 100 training points. Although this was somewhat successful, developing better strategies for bias remains useful future work.

## 5 Related Work

There is a large body of work on regularization in machine learning, including work that uses positive semi-definite matrices in the SVM quadratic program. The graph Laplacian has been used to encourage geometrically-similar feature vectors to be classified similarly (Belkin et al., 2006). An appealing property of these approaches is that they incorporate information from unlabeled examples. Wang et al. (2006) use Laplacian regularization for the task of dependency parsing. They regularize such that features for distributionally-similar words have similar weights. Rather than penalize pairwise differences proportional to a similarity function, we simply penalize weight variance.

In the field of computer vision, Tefas et al. (2001) (binary) and Kotsia et al. (2009) (multi-class) also regularize weights with respect to a covariance matrix. They use labeled data to find the sum of the sample covariance matrices from each class, similar to linear discriminant analysis. We propose the idea in general, and instantiate with a different $\mathbf{C}$ matrix: a variance regularizer over $\bar{w}$. Most importantly, our instantiated covariance matrix does not require labeled data to generate.

In a Bayesian setting, Raina et al. (2006) model feature correlations in a logistic regression classifier. They propose a method to construct a covariance matrix for a multivariate Gaussian prior on the classifier's weights. Labeled data for other, related tasks is used to infer potentially correlated features on the target task. Like in our results, they found that the gains from modeling dependencies diminish as more training data is available.

We also mention two related online learning approaches. Similar to our goal of regularizing toward a good unsupervised system, Crammer et al. (2006) regularize $\bar{w}$ toward a (different) target vector at each update, rather than strictly minimizing $||\bar{w}||^2$. The target vector is the vector learned from the cumulative effect of previous updates. Dredze et al. (2008) maintain the variance of each weight and use this to guide the online updates. However, covariance between weights is not considered.

We believe new SVM regularizations in general, and variance regularization in particular, will increasingly be used in combination with related NLP strategies that learn better when labeled data is scarce. These may include: using more-general features, e.g. ones generated from raw text (Miller et al., 2004; Koo et al., 2008), leveraging out-of-domain examples to improve in-domain classification (Blitzer et al., 2007; Daumé III, 2007), active learning (Cohn et al., 1994; Tong and Koller, 2002), and approaches that treat unlabeled data as labeled, such as bootstrapping (Yarowsky, 1995), co-training (Blum and Mitchell, 1998), and self-training (McClosky et al., 2006).

## 6 Future Work

The primary direction of future research will be to apply the VAR-SVM to new problems and tasks. There are many situations where a system designer has an intuition about the role a feature will play in prediction; the feature was perhaps added with this role in mind. By biasing the SVM to use features as intended, VAR-SVM may learn better with fewer training examples. The relationship between attributes and classes may be explicit when, e.g., a rule-based system is optimized via discriminative learning, or annotators justify their decisions by indicating the relevant attributes (Zaidan et al., 2007). Also, if features are *a priori* thought to have different predictive worth, the attribute *values* could be scaled such that variance regularization, as we formulated it, has the desired effect.

Other avenues of future work will be to extend the VAR-SVM in three directions: efficiency, representational power, and problem domain.

While we optimized the VAR-SVM objective in CPLEX, general purpose QP-solvers "do not exploit the special structure of [the SVM optimization] problem," and consequently often train in time super-linear with the number of training examples (Joachims et al., 2009). It would be useful to fit our optimization problem to efficient SVM training methods, especially for linear classifiers.

VAR-SVM's representational power could be extended by using non-linear SVMs. Kernels can be used with a covariance regularizer (Kotsia et al., 2009). Since $\mathbf{C}$ is positive semi-definite, the square root of its inverse is defined. We can therefore map the input examples using $(\mathbf{C}^{-\frac{1}{2}}\bar{x})$, and write an equivalent objective function in terms of kernel functions over the transformed examples.

Also, since structured-prediction SVMs build on the multi-class framework (Tsochantaridis et al., 2005), variance regularization can be incorporated naturally into more complex prediction tasks, such as parsers, taggers, and aligners.

VAR-SVM may also help in new domains where annotated data is lacking. VAR-SVM should be stronger cross-domain than K-SVM; regularization with domain-neutral prior-knowledge can offset domain-specific biases. Learned weight vectors from other domains may also provide cross-domain regularization guidance.

## 7 Conclusion

We presented variance-regularization SVMs, an approach to learning that creates better classifiers using fewer training examples. Variance regularization incorporates a bias for known good weights into the SVM's quadratic program. The VAR-SVM can therefore exploit extra knowledge by the system designer. Since the objective remains a convex quadratic function of the weights, the program is computationally no harder to optimize than a standard SVM. We also demonstrated how to design multi-class SVMs using only class-specific attributes, and compared the performance of this approach to standard multi-class SVMs on the task of preposition selection.

While variance regularization is most helpful on tasks with many classes and features, like preposition selection, it achieved gains on all our tasks when training with smaller sample sizes. It should be useful on a variety of other NLP problems.

## References


Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 7:2399–2434.

Shane Bergsma, Dekang Lin, and Randy Goebel. 2008. Distributional identification of non-referential pronouns. In *ACL-08: HLT*.

Shane Bergsma, Dekang Lin, and Randy Goebel. 2009. Web-scale N-gram models for lexical disambiguation. In *IJCAI*.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *COLT*.

Thorsten Brants and Alex Franz. 2006. The Google Web 1T 5-gram Corpus Version 1.1. LDC2006T13.

Martin Chodorow, Joel R. Tetreault, and Na-Rae Han. 2007. Detection of grammatical errors involving prepositions. In *ACL-SIGSEM Workshop on Prepositions*.

Kenneth Church, Ted Hart, and Jianfeng Gao. 2007. Compressing trigram language models with Golomb coding. In *EMNLP-CoNLL*.

David Cohn, Les Atlas, and Richard Ladner. 1994. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.*, 20(3):273–297.

CPLEX. 2005. IBM ILOG CPLEX 9.1. `www.ilog.com/products/cplex/`.

Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*, 3:951–991.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *JMLR*, 7:551–585.

Hal Daumé III. 2007. Frustratingly easy domain adaptation. In *ACL*.

Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *ICML*.

Richard O. Duda and Peter E. Hart. 1973. *Pattern Classification and Scene Analysis*. John Wiley & Sons.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874.

Andrew R. Golding and Dan Roth. 1999. A Winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3):107–130.

Sariel Har-Peled, Dan Roth, and Dav Zimak. 2003. Constraint classification for multiclass classification and ranking. In *NIPS*.

Graeme Hirst and Alexander Budanitsky. 2005. Correcting real-word spelling errors by restoring lexical cohesion. *Nat. Lang. Eng.*, 11(1):87–111.

Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE Trans. Neur. Networks*, 13(2):415–425.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. 2009. Cutting-plane training of structural SVMs. *Mach. Learn.*, 77(1):27–59.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *KDD*.

Thorsten Joachims. 2006. Training linear SVMs in linear time. In *KDD*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *ACL-08: HLT*.

Irene Kotsia, Stefanos Zafeiriou, and Ioannis Pitas. 2009. Novel multiclass classifiers based on the minimization of the within-class variance. *IEEE Trans. Neur. Networks*, 20(1):14–34.

Mirella Lapata and Frank Keller. 2005. Web-based models for natural language processing. *ACM Trans. Speech and Language Processing*, 2(1):1–31.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *HLT-NAACL*.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *HLT-NAACL*.

Andrew Y. Ng and Michael I. Jordan. 2002. Discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*.

Vincent Ng. 2009. Graph-cut-based anaphoricity determination for coreference resolution. In *NAACL-HLT*.

Franz J. Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *ACL*.

Daisuke Okanohara and Jun'ichi Tsujii. 2007. A discriminative language model with pseudo-negative samples. In *ACL*.

Chris D. Paice and Gareth D. Husk. 1987. Towards the automatic recognition of anaphoric features in English text: the impersonal pronoun "it". *Computer Speech and Language*, 2:109–132.

Rajat Raina, Andrew Y. Ng, and Daphne Koller. 2006. Constructing informative priors using transfer learning. In *ICML*.

Ryan Rifkin and Aldebaro Klautau. 2004. In defense of one-vs-all classification. *JMLR*, 5:101–141.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: training log-linear models on unlabeled data. In *ACL*.

Anastasios Tefas, Constantine Kotropoulos, and Ioannis Pitas. 2001. Using support vector machines to enhance the performance of elastic graph matching for frontal face authentication. *IEEE Trans. Pattern Anal. Machine Intell.*, 23:735–746.

Joel R. Tetreault and Martin Chodorow. 2008. The ups and downs of preposition error detection in ESL writing. In *COLING*.

Simon Tong and Daphne Koller. 2002. Support vector machine active learning with applications to text classification. *JMLR*, 2:45–66.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484.

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. John Wiley & Sons.

Qin Iris Wang, Colin Cherry, Dan Lizotte, and Dale Schuurmans. 2006. Improved large margin dependency parsing via local constraints and Laplacian regularization. In *CoNLL*.

Jason Weston and Chris Watkins. 1998. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*.

Omar Zaidan, Jason Eisner, and Christine Piatko. 2007. Using "annotator rationales" to improve machine learning for text categorization. In *NAACL-HLT*.