Use of Supprot Vector Learning for Chunk Identification

Taku Kudoh and **Yuji Matsumoto**
Graduate School of Information Science, Nara Institute of Science and Technology
{*taku-ku, matsu*}*@is.aist-nara.ac.jp*

1 Introduction

In this paper, we explore the use of Support Vector Machines (SVMs) for CoNLL-2000 shared task, chunk identification. SVMs are so-called large margin classifiers and are well-known as their good generalization performance. We investigate how SVMs with a very large number of features perform with the classification task of chunk labelling.

2 Support Vector Machines

Support Vector Machines (SVMs), first introduced by Vapnik (Cortes and Vapnik, 1995; Vapnik, 1995), are relatively new learning approaches for solving two-class pattern recognition problems. SVMs are well-known for their good generalization performance, and have been applied to many pattern recognition problems. In the field of natural language processing, SVMs are applied to text categorization, and are reported to have achieved high accuracy without falling into over-fitting even with a large number of words taken as the features (Joachims, 1998; Taira and Haruno, 1999)

First of all, let us define the training data which belongs to either positive or negative class as follows:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \qquad \mathbf{x}_i \in \mathbf{R}^n,\ y_i \in \{+1, -1\}$$

$\mathbf{x}_i$ is a feature vector of the i-th sample represented by an n dimensional vector. y_i is the class (positive($+1$) or negative(-1) class) label of the i-th data. In basic SVMs framework, we try to separate the positive and negative examples by hyperplane written as:

$$(\mathbf{w} \cdot \mathbf{x}) + b = 0 \qquad \mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}.$$

SVMs find the "optimal" hyperplane (optimal parameter $\mathbf{w}, b$) which separates the training data into two classes precisely. What "optimal" means? In order to define it, we need to consider the **margin** between two classes. Figures 1 illustrates this idea. The solid lines show two possible hyperplanes, each of which correctly separates the training data into two classes. The two dashed lines parallel to the separating hyperplane show the boundaries in which one can move the separating hyperplane without misclassification. We call the distance between each parallel dashed lines as **margin**. SVMs take a simple strategy that finds the separating hyperplane which maximizes its margin. Precisely, two dashed lines and margin (d) can be written as:

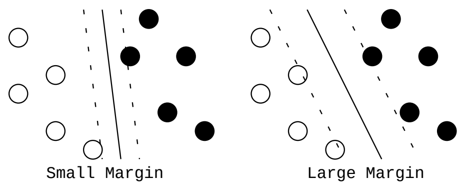

Figure 1: Two possible separating hyperplanes

$$(\mathbf{w} \cdot \mathbf{x}) + b = \pm 1, \quad d = 2/\|\mathbf{w}\|.$$

SVMs can be regarded as an optimization problem; finding $\mathbf{w}$ and b which minimize $\|\mathbf{w}\|$ under the constraints: $y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1$.

Furthermore, SVMs have potential to cope with the linearly unseparable training data. We leave the details to (Vapnik, 1995), the optimization problems can be rewritten into a dual form, where all feature vectors appear in their dot product. By simply substituting every dot product of $\mathbf{x_i}$ and $\mathbf{x_j}$ in dual form with any Kernel function $K(\mathbf{x_i}, \mathbf{x_j})$, SVMs can handle non-linear hypotheses. Among the many kinds of Kernel functions available, we will focus on the

d-th polynomial kernel:

$$K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Use of d-th polynomial kernel function allows us to build an optimal separating hyperplane which takes into account all combination of features up to d.

We believe SVMs have advantage over conventional statistical learning algorithms, such as Decision Tree, and Maximum Entropy Models, from the following two aspects:

- SVMs have high generalization performance independent of dimension of feature vectors. Conventional algorithms require careful feature selection, which is usually optimized heuristically, to avoid overfitting.
- SVMs can carry out their learning with all combinations of given features without increasing computational complexity by introducing the Kernel function. Conventional algorithms cannot handle these combinations efficiently, thus, we usually select "important" combinations heuristically with taking the trade-off between accuracy and computational complexity into account.

3 Approach for Chunk Identification

The chunks in the CoNLL-2000 shared task are represented with IOB based model, in which every word is to be tagged with a chunk label extended with I (inside a chunk), O (outside a chunk) and B (inside a chunk, but the preceding word is in another chunk). Each chunk type belongs to I or B tags. For example, NP could be considered as two types of chunk, I-NP or B-NP. In training data of CoNLL-2000 shared task, we could find 22 types of chunk [1] considering all combinations of IOB-tags and chunk types. We simply formulate the chunking task as a classification problem of these 22 types of chunk.

Basically, SVMs are binary classifiers, thus we must extend SVMs to multi-class classifiers in order to classify these 22 types of chunks. It is known that there are mainly two approaches to extend from a binary classification task to those with K classes. First approach is often used and typical one *"one class vs. all others"*. The idea is to build K classifiers that separate one class among from all others. Second approach is *pairwise classification*. The idea is to build $K \times (K-1)/2$ classifiers considering all pairs of classes, and final class decision is given by their majority voting. We decided to construct pairwise classifiers for all the pairs of chunk labels, so that the total number of classifiers becomes $\frac{22 \times 21}{2} = 231$. The reasons that we use pairwise classifiers are as follows:

- Some experiments report that combination of pairwise classifier perform better than K classifier (Kreßel, 1999).
- The amount of training data for a pair is less than the amount of training data for separating one class with all others.

For the features, we decided to use all the information available in the surrounding contexts, such as the words, their POS tags as well as the chunk labels. More precisely, we give the following for the features to identify chunk label c_i at i-th word:

$$\begin{array}{rl} w_j, t_j & (j = i-2,\, i-1,\, i,\, i+1,\, i+2) \\ c_j & (j = i-2,\, i-1) \end{array}$$

where w_i is the word appearing at i-th word, t_i is the POS tag of w_i, and c_i is the (extended) chunk label at i-th word. Since the chunk labels are not given in the test data, they are decided dynamically during the tagging of chunk labels. This technique can be regarded as a sort of Dynamic Programming (DP) matching, in which the best answer is searched by maximizing the total certainty score for the combination of tags. In using DP matching, we decided to keep not all ambiguities but a limited number of them. This means that a beam search is employed, and only the top N candidates are kept for the search for the best chunk tags. The algorithm scans the test data from left to right and calls the SVM classifiers for all pairs of chunk tags for obtaining the certainty score. We defined the certainty score as the number of votes for the class (tag) obtained through the pairwise voting.

[1] Precisely, the number of combination becomes 23. However, we do not consider I-LST tag since it dose not appear in training data.

Since SVMs are vector based classifier, they accept only numerical values for their features. To cope with this constraints, we simply expand all features as a binary-value taking either 0 or 1. By taking all words and POS tags appearing in the training data as features, the total dimension of feature vector becomes as large as 92837. Generally, we need vast computational complexity and memories to handle such a huge dimension of vectors. In fact, we can reduce these complexity considerably by holding only indices and values of non-zero elements, since the feature vectors are usually sparse, and SVMs only require the evaluation of dot products of each feature vectors for their training.

In addition, although we could apply some cut-off threshold for the number of occurrence in the training set, we decided to use everything, not only POS tags but also words themselves. The reasons are that we simply do not want to employ a kind of "heuristics", and SVMs are known to have a good generalization performance even with very large features.

4 Results

We have applied our proposed method to the test data of CoNLL-2000 shared task, while training with the complete training data. For the kernel function, we use the 2-nd polynomial function. We set the beam width N to 5 tentatively. SVMs training is carried out with the SVM^{light} package, which is designed and optimized to handle large sparse feature vector and large numbers of training examples(Joachims, 2000; Joachims, 1999a). It took about 1 day to train 231 classifiers with PC-Linux (Celeron 500Mhz, 512MB).

Figure 1 shows the results of our experiments. The all the values of the chunking F-measure are almost 93.5. Especially, our method performs well for the chunk types of high frequency, such as NP, VP and PP.

test data	precision	recall	$F_{\beta=1}$
ADJP	79.22%	69.63%	74.12
ADVP	80.86%	80.48%	80.67
CONJP	62.50%	55.56%	58.82
INTJ	100.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	93.72%	94.02%	93.87
PP	96.60%	97.94%	97.26
PRT	80.58%	78.30%	79.43
SBAR	89.29%	84.11%	86.62
VP	93.76%	93.84%	93.80
all	93.45%	93.51%	93.48

Table 1: The results per chunk type with our proposed SVMs based method

5 Discussion

In this paper, we propose Chunk identification analysis based on Support Vector Machines.

Although we select features for learning in very straight way — using all available features such as the words their POS tags without any cut-off threshold for the number of occurrence, we archive high performance for test data.

When we use other learning methods such as Decision Tree, we have to select feature set manually to avoid over-fitting. Usually, these feature selection depends on heuristics, so that it is difficult to apply them to other classification problems in other domains.

Memory based learning method can also handle all available features. However, the function to compute the distance between the test pattern and the nearest cases in memory is usually optimized in an ad-hoc way

Through our experiments, we have shown the high generalization performance and high feature selection abilities of SVMs.

References

C. Cortes and Vladimir N. Vapnik. 1995. Support Vector Networks. *Machine Learning*, 20:273–297.

Thorsten Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning (ECML)*.

Thorsten Joachims. 1999a. Making Large-Scale Support Vector Machine Learning Practical. In *Advances in Kernel Mathods*. MIT Press.

Thorsten Joachims. 2000. SVM^{light} version 3.02. http://www-ai.cs.uni-dortmund.de/ SOFTWARE/SVM_LIGHT/svm_light.eng.html.

Ulrich H.-G Kreßel. 1999. Pairwise Classification and Support Vector Machines. In *Advances in Kernel Mathods*. MIT Press.

Hirotoshi Taira and Masahiko Haruno. 1999. Feature Selection in SVM Text Categorization. In *AAAI-99*.

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.