A Parallel Mixture of SVMs for Very Large Scale Problems

Ronan Collobert*  
Université de Montréal, DIRO  
CP 6128, Succ. Centre-Ville  
Montréal, Québec, Canada  
collobert@iro.umontreal.ca

Samy Bengio  
IDIAP  
CP 592, rue du Simplon 4  
1920 Martigny, Switzerland  
bengio@idiap.ch

Yoshua Bengio  
Université de Montréal, DIRO  
CP 6128, Succ. Centre-Ville  
Montréal, Québec, Canada  
bengioy@iro.umontreal.ca

Abstract

Support Vector Machines (SVMs) are currently the state-of-the-art models for many classification problems but they suffer from the complexity of their training algorithm which is at least quadratic with respect to the number of examples. Hence, it is hopeless to try to solve real-life problems having more than a few hundreds of thousands examples with SVMs. The present paper proposes a new mixture of SVMs that can be easily implemented in parallel and where each SVM is trained on a small subset of the whole dataset. Experiments on a large benchmark dataset (Forest) as well as a difficult speech database, yielded significant time improvement (time complexity appears empirically to locally grow linearly with the number of examples). In addition, and that is a surprise, a significant improvement in generalization was observed on Forest.

1 Introduction

Recently a lot of work has been done around Support Vector Machines [9], mainly due to their impressive generalization performances on classification problems when compared to other algorithms such as artificial neural networks [3, 6]. However, SVMs require to solve a quadratic optimization problem which needs resources that are at least quadratic in the number of training examples, and it is thus hopeless to try solving problems having millions of examples using classical SVMs.

In order to overcome this drawback, we propose in this paper to use a mixture of several SVMs, each of them trained only on a part of the dataset. The idea of an SVM mixture is not new, although previous attempts such as Kwok’s paper on Support Vector Mixtures [5] did not train the SVMs on part of the dataset but on the whole dataset and hence could not overcome the

*Part of this work has been done while Ronan Collobert was at IDIAP, CP 592, rue du Simplon 4, 1920 Martigny, Switzerland.
time complexity problem for large datasets. We propose here a simple method to train such a mixture, and we will show that in practice this method is much faster than training only one SVM, and leads to results that are at least as good as one SVM. We conjecture that the training time complexity of the proposed approach with respect to the number of examples is sub-quadratic for large data sets. Moreover this mixture can be easily parallelized, which could improve again significantly the training time.

The organization of the paper goes as follows: in the next section, we briefly introduce the SVM model for classification. In section 3 we present our mixture of SVMs, followed in section 4 by some comparisons to related models. In section 5 we show some experimental results, first on a toy dataset, then on two large real-life datasets. A short conclusion then follows.

2 Introduction to Support Vector Machines

Support Vector Machines (SVMs) [9] have been applied to many classification problems, generally yielding good performance compared to other algorithms. The decision function is of the form

\[ y = \text{sign} \left( \sum_{i=1}^{N} y_i \alpha_i K(x, x_i) + b \right) \]  

(1)

where \( x \in \mathbb{R}^d \) is the \( d \)-dimensional input vector of a test example, \( y \in \{-1, 1\} \) is a class label, \( x_i \) is the input vector for the \( i^{th} \) training example, \( y_i \) is its associated class label, \( N \) is the number of training examples, \( K(x, x_i) \) is a positive definite kernel function, and \( \alpha = \{\alpha_1, \ldots, \alpha_N\} \) and \( b \) are the parameters of the model. Training an SVM consists in finding \( \alpha \) that minimizes the objective function

\[ Q(\alpha) = -\sum_{i=1}^{N} \alpha_i + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \]  

(2)

subject to the constraints

\[ \sum_{i=1}^{N} \alpha_i = 0 \]  

(3)

and

\[ 0 \leq \alpha_i \leq C \quad \forall i. \]  

(4)

The kernel \( K(x, x_i) \) can have different forms, such as the Radial Basis Function (RBF):

\[ K(x_i, x_j) = \exp \left( \frac{-||x_i - x_j||^2}{\sigma^2} \right) \]  

(5)

with parameter \( \sigma \).

Therefore, to train an SVM, we need to solve a quadratic optimization problem, where the number of parameters is \( N \). This makes the use of SVMs for large datasets difficult: computing \( K(x_i, x_j) \) for every training pair would require \( O(N^2) \) computation, and solving may take up to \( O(N^3) \). Note however that current state-of-the-art algorithms appear to have training time complexity scaling much closer to \( O(N^2) \) than \( O(N^3) \) [2].

3 A New Conditional Mixture of SVMs

In this section we introduce a new type of mixture of SVMs. The output of the mixture for an input vector \( x \) is computed as follows:

\[ f(x) = h \left( \sum_{m=1}^{M} w_m(x) s_m(x) \right) \]  

(6)
where $M$ is the number of experts in the mixture, $s_m(x)$ is the output of the $m^{th}$ expert given input $x$, $w_m(x)$ is the weight for the $m^{th}$ expert given by a “gater” module taking also $x$ in input, and $h$ is a transfer function which could be for example the hyperbolic tangent for classification tasks. Here each expert is an SVM, and we took a neural network for the gater in our experiments. In the proposed model, the gater is trained to minimize the cost function

$$C = \sum_{i=1}^{N} [f(x_i) - y_i]^2 .$$  \tag{7}$$

To train this model, we propose a very simple algorithm:

1. Divide the training set into $M$ random subsets of size near $N/M$.
2. Train each expert separately over one of these subsets.
3. Keeping the experts fixed, train the gater to minimize (7) on the whole training set.
4. Reconstruct $M$ subsets: for each example $(x_i, y_i)$,
   - sort the experts in descending order according to the values $w_m(x_i)$,
   - assign the example to the first expert in the list which has less than $(N/M + c)$ examples*, in order to ensure a balance between the experts.
5. If a termination criterion is not fulfilled (such as a given number of iterations or a validation error going up), goto step 2.

Note that step 2 of this algorithm can be easily implemented in parallel as each expert can be trained separately on a different computer. Note also that step 3 can be an approximate minimization (as usually done when training neural networks).

4 Other Mixtures of SVMs

The idea of mixture models is quite old and has given rise to very popular algorithms, such as the well-known Mixture of Experts [4] where the cost function is similar to equation (7) but where the gater and the experts are trained, using gradient descent or EM, on the whole dataset (and not subsets) and their parameters are trained simultaneously. Hence such an algorithm is quite demanding in terms of resources when the dataset is large, if training time scales like $O(N^p)$ with $p > 1$.

In the more recent Support Vector Mixture model [5], the author shows how to replace the experts (typically neural networks) by SVMs and gives a learning algorithm for this model. Once again the resulting mixture is trained jointly on the whole dataset, and hence does not solve the quadratic barrier when the dataset is large.

In another divide-and-conquer approach [7], the authors propose to first divide the training set using an unsupervised algorithm to cluster the data (typically a mixture of Gaussians), then train an expert (such as an SVM) on each subset of the data corresponding to a cluster, and finally recombine the outputs of the experts. Here, the algorithm does indeed train separately the experts on small datasets, like the present algorithm, but there is no notion of a loop reassigning the examples to experts according to the prediction made by the gater of how well each expert performs on each example. Our experiments suggest that this element is essential to the success of the algorithm.

Finally, the Bayesian Committee Machine [8] is a technique to partition the data into several subsets, train SVMs on the individual subsets and then use a specific combination scheme based on the covariance of the test data to combine the predictions. This method scales linearly in the

*where $c$ is a small positive constant. In the experiments, $c = 1$. 
number of training data, but is in fact a transductive method as it cannot operate on a single test example. Like in the previous case, this algorithm assigns the examples randomly to the experts (however the Bayesian framework would in principle allow to find better assignments).

Regarding our proposed mixture of SVMs, if the number of experts grows with the number of examples, and the number of outer loop iterations is a constant, then the total training time of the experts scales linearly with the number of examples. Indeed, given \( N \) the total number of examples, choose the number of expert \( M \) such that the ratio \( \frac{N}{M} \) is a constant \( r \); Then, if \( k \) is the number of outer loop iterations, and if the training time for an SVM with \( r \) examples is \( O(r^3) \) (empirically, \( \beta \) is slightly above 2), the total training time of the experts is \( O(kr^3 \times M) = O(kr^3 - 1 N) \), where \( k, r \) and \( \beta \) are constants, which gives a total training time of \( O(N) \). In particular for \( \beta = 2 \) that gives \( O(krN) \). The actual total training time should however also include \( k \) times the training time of the gater, which may potentially grow more rapidly than \( O(N) \). However, it did not appear to be the case in our experiments, thus yielding apparent linear training time. Future work will focus on methods to reduce the gater training time and guarantee linear training time per outer loop iteration.

5 Experiments

In this section, we present three sets of experiments comparing the new mixture of SVMs to other machine learning algorithms. Note that all the SVMs in these experiments have been trained using SVMTorch [2].

5.1 A Toy Problem

In the first series of experiments, we first tested the mixture on an artificial toy problem for which we generated 10,000 training examples and 10,000 test examples. This problem had two non-linearly separable classes and had two input dimensions. On Figure 1 we show the decision surfaces obtained first by a linear SVM, then by a Gaussian SVM, and finally by the proposed mixture of SVMs. Moreover, in the latter, the gater was a simple linear function and there were two linear SVMs in the mixture\(^1\). This artificial problem thus shows clearly that the algorithm seems to work, and is able to combine, even linearly, very simple models in order to produce a non-linear decision surface.

5.2 A Large-Scale Realistic Problem: Forest

For a more realistic problem, we did a series of experiments on part of the UCI Forest dataset\(^2\). We modified the 7-class classification problem into a binary classification problem where the goal was to separate class 2 from the other 6 classes. Each example was described by 54 input features, each normalized by dividing by the maximum found on the training set. The dataset had more than 500,000 examples and this allowed us to prepare a series of experiments as follows:

- We kept a separate test set of 50,000 examples to compare the best mixture of SVMs to other learning algorithms.
- We used a validation set of 10,000 examples to select the best mixture of SVMs, varying the number of experts and the number of hidden units in the gater.
- We trained our models on different training sets, using from 100,000 to 400,000 examples.
- The mixtures had from 10 to 50 expert SVMs with Gaussian kernel and the gater was an MLP with between 25 and 500 hidden units.

\(^1\)Note that the transfer function \( h(\cdot) \) was still a \( \tanh(\cdot) \).

\(^2\)The Forest dataset is available on the UCI website at the following address: ftp://ftp.ics.uci.edu/pub/machine-learning-databases/covtype/covtype.data.