Scalable non-linear Support Vector Machine using hierarchical clustering

Asharaf S S K Shevade M Narasimha Murty
Computer Science and Automation, Indian Institute of Science, Bangalore-560012, India.
asharaf@csa.iisc.ernet.in shirish@csa.iisc.ernet.in mnm@csa.iisc.ernet.in

Abstract

This paper discusses a method for scaling SVM with Gaussian kernel function to handle large data sets by using a selective sampling strategy for the training set. It employs a scalable hierarchical clustering algorithm to construct cluster indexing structures of the training data in the kernel induced feature space. These are then used for selective sampling of the training data for SVM to impart scalability to the training process. Empirical studies made on real world data sets show that the proposed strategy performs well on large data sets.

1 Introduction

Even though Support Vector Machine(SVM) has gained huge acceptance from the machine learning community it has received very little attention from the Data Mining researchers due to its data size dependent training time complexity. Several incremental SVM[2] formulations have been proposed in the literature for scaling SVM to handle large data sets. Most of these techniques explore the structure of the SVM problem to impart incremental nature to the learning process.

Clustering Based SVM(CBSVM)[3] is a fairly new approach for scaling a two class linear SVM classifier to handle large data sets. This method employs a hierarchical clustering algorithm, namely BIRCH[4], operating in the data space as a preprocessing step to the SVM classifier. In CBSVM the SVM training process starts with a high level abstraction of the training data generated in data space by the clustering algorithm and is continued by selective declustering(expansion) of clusters near the boundary using the hierarchical indexing structures obtained from the clustering phase. But the applicability of this method is limited to SVM with linear kernel function. In this paper we address the problem of scaling a two class non-linear SVM classifier to handle large data sets. Here we propose a novel Kernel Based Hierarchical Clustering(KBHC) algorithm and its use for scaling SVMs with Gaussian kernel function to handle large data sets. The method proposed can also be extended to handle multi-category classification problem. The paper is organized as follows. In section 2 we discuss Clustering Based SVM. The proposed KBHC algorithm and its use for scaling SVM is discussed in Section 3. Empirical results are given in Section 4 and Section 5 concludes the paper.

2 Clustering Based SVM

Clustering Based SVM method achieves scalability of SVM with linear kernel function by selectively sampling the training set for a two class SVM using two hierarchical cluster indexing structures generated in data space. For scalability, these cluster indexing structures have to be generated from the training set using a clustering algorithm employing minimal number of data set scans. In CBSVM this is done using BIRCH with a single data set scan. BIRCH is a hierarchical clustering algorithm used to construct a hierarchical cluster description of the training data set in a tree structure called Clustering Feature(CF) tree. Any node in a CF tree contains entries called CF vectors each describing a cluster. The CF vectors are arranged to form a cluster hierarchy in the CF tree.

The key idea of CBSVM is to use the hierarchical clustering algorithm to get a finer description of the SVM training data closer to the decision boundary and coarser description of the data away from the decision boundary. Initially in CBSVM two hierarchical cluster trees(CF tree of BIRCH) are constructed from data sets corresponding to the two classes using BIRCH. Once the CF trees are constructed, CBSVM starts training the linear SVM from the CF entries of root nodes and obtains the corresponding Support Vectors(SVs). Now CBSVM selectively declusters only those clusters(CF vectors) near the boundary by using the next level of cluster hierarchy(CF tree). The declustering process expands a CF vector if its distance to the boundary is less than the maximum of the distances of Support Vectors from the boundary. The newly obtained training data set from this selective declustering phase is used for further training of SVM and this process continues till the declustering phase reaches the leaf nodes in the cluster hierarchy.

It may be noted that in CBSVM the BIRCH algorithm generates hierarchical cluster indexing structures in original data space and is used in the selective sampling process for training SVM with linear kernel function. To extend the applicability of this selective sampling strategy for SVM with non-linear kernel functions we need a suitable clustering algorithm that can construct the hierarchical indexing structures in the appropriate kernel induced feature space.

3 Kernel Based Hierarchical Clustering

The proposed hierarchical clustering algorithm is a kernalized form of BIRCH. Using a single data set scan this algorithm can generate a hierarchical cluster indexing structure similar to CF tree in a Gaussian kernel induced feature space. Similar to BIRCH the concept of Modified Clustering Feature(MCF) and MCF tree are at the core of KBHC. Here each Modified Clustering Feature is a triple that summarizes the information that we maintain about a cluster in the kernel induced feature space.

Given the data points $\{x_i\}_{i=1}^{N}$ we find μ, the data space representation of the best prototype in the least square error sense for a cluster C defined in the kernel induced feature space, by solving the following problem:

$$\min_{\mu} J(\mu) = \min_{\mu} \sum_{x_i \in C} ||\phi(x_i) - \phi(\mu)||^2 \tag{1}$$

where ϕ is the non-linear transformation implicitly achieved by the kernel function. The objective function in equation (1) can now be expanded using Mercer kernel $K(x,y) = \phi(x)\cdot\phi(y)$ giving the dot product in the kernel induced feature space. One of the common kernel functions used is the Gaussian kernel given by $K(x_i,x_j) = e^{-q||x_i - x_j||^2}$. Here $||\cdot||$ represents the L_2 norm and q is a user given parameter. For Gaussian kernel we can write the objective function as

$$\min_{\mu} J(\mu) = \min_{\mu} \sum_{x_i \in C} (2 - 2K(x_i,\mu)) \tag{2}$$

The solution of this problem results in the fixed point equation

$$\mu = \frac{\sum\limits_{x_i \in C} K(x_i,\mu)x_i}{\sum\limits_{x_i \in C} K(x_i,\mu)} \tag{3}$$

This minimization is performed by starting with an initial guess for μ and then iteratively recomputing the μ value until convergence is met. The convergence criteria used can be based on a user given $Tolerance$ defining a threshold for the change in μ values in consecutive iterations.

The MCF vector representing a cluster C_i in the Gaussian kernel induced feature space can now be defined as a triple: $\mathbf{MCF_i} = (\mathbf{N_i}, \mathbf{LS_i}, \mu_\mathbf{i})$, here N_i is the number of data points in the cluster C_i and LS_i is the Linear Sum which is detailed in the next section. For each cluster found, KBHC maintains only its MCF vector from which the needed statistics used in computations involving that cluster can be obtained.

3.1 Merging Clusters

KBHC constructs the hierarchical cluster abstraction of a data set by arranging the MCF vectors in the form of an MCF tree. In this cluster hierarchy the MCF vectors corresponding to the lower level clusters are merged to form the higher level abstractions. The merging process involved here can be explained as follows. Let $MCF_j = (N_j, LS_j, \mu_j),\ \ j = 1...m$ be the MCF vectors of m disjoint clusters, $\{C_j\}_{j=1}^{m}$, which are to be merged. The KBHC algorithm finds the MCF Vector $MCF_{new} = (N_{new}, LS_{new}, \mu_{new})$ for the merged cluster C_{new} by minimizing the objective function given in equation (1) until convergence is met. Here $N_{new} = \sum_{j=1}^{m} N_j$. The initial value of μ_{new} used in the minimization process is computed as

$$\mu_{new}^{(0)} = \frac{\sum_{j=1}^{m} N_j \mu_j}{\sum_{j=1}^{m} N_j} \tag{4}$$

At iteration t, the algorithm computes the new value of the prototype vector $\mu_{new}^{(t)}$ as

$$\mu_{new}^{(t)} = \frac{\sum\limits_{x_k \in C_{new}} K(x_k, \mu_{new}^{(t-1)})x_k}{\sum\limits_{x_k \in C_{new}} K(x_k, \mu_{new}^{(t-1)})} \tag{5}$$

The iterations continue till

$$||\mu_{new}^{(t-1)} - \mu_{new}^{(t)}|| < Tolerance$$

To simplify this computation process we will now make an assumption

$$K(x_k, \mu_{new}^{(t)}) \approx K(\mu_j, \mu_{new}^{(t)}) \quad \forall x_k \in C_j \quad j = 1,2...m \tag{6}$$

With this assumption it requires only one kernel evaluation per cluster and also reduces the space requirement significantly because it has to maintain only the MCF vector for any cluster instead of all the data points belonging to that cluster. Empirically this is found to be a good approximation. Using this assumption we can write equation (5) as

$$\mu_{new}^{(t)} \approx \frac{\sum_{j=1}^{m}\left(K(\mu_j, \mu_{new}^{(t-1)}) \sum_{x_k \in C_j} x_k\right)}{\sum_{j=1}^{m} N_j K(\mu_j, \mu_{new}^{(t-1)})} \tag{7}$$

Note that a linear sum appears in equation (7) and this is defined as the LS_i entry of the MCF vector representing cluster C_i, where $LS_i = \sum_{x_k \in C_i} x_k$

The MCF tree is a height balanced tree with two parameters: branching factor B and threshold T. Here each node contains at most B entries of the form $\{[MCF_i, child_i]\}_{i=1}^{B}$. Here MCF_i is the MCF of a subcluster of the cluster represented by this node and $child_i$ is the pointer to this subcluster if it is a non-leaf node and *NULL* pointer if it is a leaf node. So a node represents a cluster made up of all the subclusters represented by its MCF entries. All the entries in a leaf node must satisfy a *threshold requirement*, with respect to a threshold value T: the radius R of the cluster has to be less than T. The radius R_i of a cluster C_i is given by

$$R_i = \left(\frac{\sum_{x_j \in C_i} \|\phi(x_j) - \phi(\mu_i)\|^2}{N_i}\right)^{\frac{1}{2}} \tag{8}$$

For each cluster C_i this is computed using the $\mu_l,\ l = 1...r$ values of its r subclusters. Now using the assumption given in equation (6) we can write equation (8) as

$$R_i = \left(\frac{\sum_{l=1}^{r} N_l(2 - 2K(\mu_l, \mu_i))}{\sum_{l=1}^{r} N_l}\right)^{\frac{1}{2}}$$

The tree size is a function of T. The larger the value of T is, the smaller the tree size is. Once the size of the pattern vector used is known, the parameters B and T can be determined based on the size of the tree that can be accommodated in the available memory and in KBHC algorithm we treat them as user given.

The MCF tree is dynamically built as new data points are inserted. The construction of the MCF tree starts with a single leaf node having only one MCF vector representing a singleton cluster containing any randomly chosen initial data point. Then tree is grown by inserting the data points one at a time as in BIRCH. Except the cluster merging process as explained above the MCF tree construction of KBHC proceeds in the same manner as the CF tree construction in BIRCH.

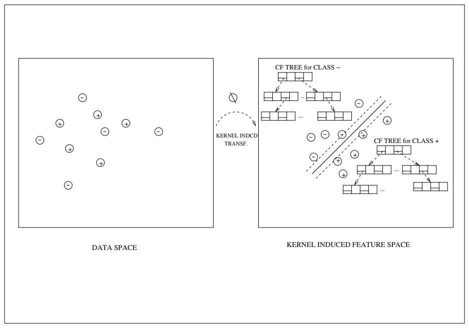

Figure 1. Data Space to Kernel Induced Feature Space Transformation(Implicitly done by the Kernel Function) and the MCF Trees constructed for positive class and negative class in Kernel Induced Feature Space

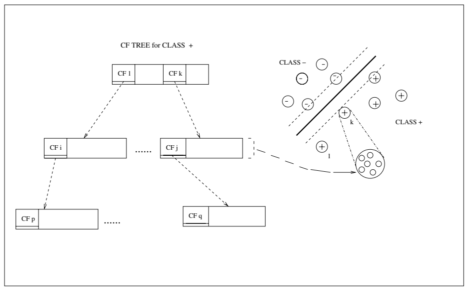

Figure 2. Declustering of Clusters(MCF vectors) near the decision boundary of the SVM to get the training set for subsequent SVM training

3.2 KBHC - CBSVM

CBSVM using Kernel Based Hierarchical Clustering is aimed at scaling SVMs with Gaussian kernel function to handle large data sets which are not linearly separable. It employs selective sampling for training as in CBSVM. Here KBHC is used to construct two hierarchical cluster indexing structures(MCF trees) in Gaussian kernel induced feature space from the data sets corresponding to the two classes as shown in Figure 1. This method starts the SVM training process with the μ values computed from the MCF entries of the root node. Once an SVM training round is over, the training set for the subsequent iterations is obtained by selectively declustering the MCF entries corresponding to those clusters that are near the boundary as shown in Figure 2. The declustering phase expands a cluster represented by MCF_i if $(D_i - R_i) < D_{ms}$. Here D_i is the distance of this cluster from the decision boundary obtained (it is the absolute value of the decision function obtained for μ_i), R_i is the radius of this cluster as given in equation (8) and D_{ms} is the maximum of the distances of the support vectors from the decision boundary obtained by previous SVM training. The selective sampling process and retraining of SVM continues till the leaf nodes of the MCF trees are encountered. The soft margin SVM used here takes care of the clusters that are falling on the wrong side of the decision boundary.

Algorithm	CBSVM		KBHC-CBSVM	
Data Set/Param	Intr Det	IJCNN1	Intr Det	IJCNN1
B	30	35	30	30
T1	6.16	6.213	0.000052	0.00024
T2	6.295	6.217	0.00062	0.0002
q	-	-	0.004	0.0015
Tolerance	-	-	0.0001	0.0001

Table 1. Parameters used with Intrusion Detection Data Set and IJCNN1 Data Set for CBSVM and KBHC-CBSVM algorithms

Algorithm	# TR	TT	# SVs	CT	GP
SVM	494021	3790	8612	1382	92.0734
CBSVM	5049	73	1372	146	78.7521
KBHC-CBSVM	4961	98	1863	275	91.7837

Table 2. Results on Intrusion Detection Data Set. The abbreviations used are: number of training points(# TR), training time(TT) in seconds, number of support vectors(# SVs), classification time(CT) for test data in seconds and generalization performance(GP) on test data in percentage

3.3 The Algorithm

The KBHC-CBSVM algorithm employs two sets of user defined parameters viz. a) The parameters B, T and $Tolerance$ used in MCF tree construction and b) The parameter q of the Gaussian kernel function. Now we can summarize the KBHC-CBSVM algorithm as

algorithm KBHC-CBSVM(Training data,Parameters)

1. **Construct two MCF trees from positive and negative class examples respectively as explained in section 3.1**
2. **Perform SVM training using the selective sampling strategy discussed in section 3.2**

4 Experimental Results

Experiments are done with two real world data sets viz. Intrusion Detection data set used in KDD-CUP 99 contest from the UCI KDD archive available at *"http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html"* and IJCNN1 data set from the LIBSVM page available at *"http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/ binary.html"*. All the experiments are done using LIBSVM Version 2.8 [1] implementation with its default C value equal to 1. For SVM on the full data set and KBHC-CBSVM algorithm we used Gaussian Kernel function and linear kernel was used for CBSVM. We compared the results obtained with SVM on the full data set, with original CBSVM algorithm and with the proposed KBHC-CBSVM algorithm. The parameters used in the experiments are given in Table 1. All the experiments were done on a Intel Xeon(TM) 3.06GHz machine with 2GB RAM.

a) Intrusion Detection Data Set: The 10% Intrusion Detection data set available in UCI KDD archive having 494021 training patterns and 311029 test set patterns was used for the experiments. We have considered it as a two class problem with a training set having 97276 patterns in normal class and 396745 in attack class(all attack types treated as equal). Only the 38 numerical features available were considered and they were normalized to have values between zero and one by dividing them by their maximum values. The results obtained are given in Table 2.

Algorithm	# TR	TT	# SVs	CT	GP
SVM	49990	516	8969	462	92.7885
CBSVM	9050	28	1557	39	90.4974
KBHC-CBSVM	8887	33	3097	93	93.1746

Table 3. Results on IJCNN1 Data Set. The abbreviations used are: number of training points(# TR), training time(TT) in seconds, number of support vectors(# SVs), classification time(CT) for test data in seconds and generalization performance(GP) on test data in percentage

b) IJCNN1 Data Set: This data set pertains to a two class problem with 49990 patterns in the training set and 91701 patterns in the test set. Each pattern here is described using 22 numerical features. The results obtained on this data set are shown in Table 3.

From all these empirical results we have observed that when handling non-linear decision boundaries in data sets the proposed method outperforms CBSVM in terms of the generalization performance at the expense of some additional computation time.

5 Conclusions

A Gaussian kernel based hierarchical clustering algorithm and its use for scaling SVM to handle large data sets having non-linear decision boundaries is proposed in this paper. Some merits of the proposed method are: **a**) It can handle large data sets with linear/non-linear decision boundaries, **b**) It generates the required indexing structures in the Gaussian kernel induced feature space using a single data set scan and hence the method is scalable.

References

[1] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. http://www.csie.ntu.edu.tw/∼cjlin/libsvm, 2001.

[2] C. P. Diehl and G. Cauwenberghs. SVM incremental learning, Adaptation and Optimization. In *Proc. IEEE Int. Joint Conf. Neural Networks(IJCNN 2003)*, pages 20–23, Portland, Oregon, July 2003.

[3] H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. In *Proc. 2003 ACM SIGKDD*, pages 306 – 315, Washington, DC, Aug 2003.

[4] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD Int.Conf. on Management of Data*, pages 103–114, Montreal, Canada, ACM Press, NewYork, 1996.