HeroSVM Support Vector Machine User Guide

version 2.1 (August, 2005)

Jianxiong Dong, Ph.D.
Centre for Pattern Recognition and Machine Intelligence
Concordia University, Montreal Quebec, Canada

Copyright Information

Contents

Chapter 1

Introduction

1.1 What is Support Vector Machine

1.2 Support Vector Machine

Support vector machines (SVM) have recently generated a great interest in the community of machine learning due to its excellent generalization performance in a wide variety of learning problems, such as handwritten digit recognition (see [1] [2]), classification of web pages [3] and face detection [4]. Some classical problems such as multi-local minima, curse of dimensionality and overfitting in neural networks [5], seldom occur in support vector machines. However, training support vector machines is still a bottleneck, especially for a large-scale learning problem [2]. Therefore, it is important to develop a fast training algorithm for SVM in order to apply it to various engineering problems in other fields. HeroSVM package has been implemented based on our proposed method [6] [7].

In order to simplify the description of implementation, we give a simple introduction of support vector machine. The details are referred to Burge's tutorial [8]. Given that training samples $\{x_i, y_i\}$, $i = 1, \cdots, N$, $y_i \in \{-1, 1\}$, $x_i \in R^n$ where y_i is the class label, support vector machine first maps the data to the other Hilbert space $\mathcal{H}$ (also called feature space), using a mapping Φ,

$$\Phi : R^n \rightarrow \mathcal{H}. \tag{1.1}$$

The mapping Φ is implemented by a kernel function K that satisfies Mercer's conditions [9] such that $k(x_i, x_j) = \Phi(x_i)\cdot\Phi(x_j)$. Then, in the high-dimensional feature space $\mathcal{H}$, we find an optimal hyperplane by maximizing the margin and bounding the number of training errors. The decision function can be given by

$$\begin{aligned} f(x) &= \theta(w \cdot \Phi(x) - b) \\ &= \theta(\sum_{i=1}^{N} y_i \alpha_i \Phi(x_i) \cdot \Phi(x) - b) \end{aligned} \tag{1.2}$$

$$= \theta(\sum_{i=1}^{N} y_i \alpha_i k(x_i, x) - b).$$

where

$$\theta(u) = \begin{cases} 1 & \text{if } u > 0 \\ -1 & \text{otherwise} \end{cases} \tag{1.3}$$

If α_i is nonzero, the corresponding data x_i is called support vector. Training a SVM is to find $\alpha_i, i = 1, \cdots, N$, which can be achieved by minimizing the following quadratic cost function:

$$\begin{array}{rc} \text{maximize} & L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j). \\ \text{subject to} & 0 \leq \alpha_i \leq C \quad i = 1 \cdots N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{array} \tag{1.4}$$

where C is a parameter chosen by the user, a larger C corresponds to a higher penalty allocated to the training errors. Since kernel K is semi-positive definite and constraints define a convex set, the above optimization reduces to a convex quadratic programming. The weight W is uniquely determined, but with respect to the threshold b, there exist several solutions in the special cases (see [10] [11] [12]). Further, an interesting fact is that the solution is not changed if any non-support vector is removed from Eq.(1.4).

In the next chapter, we describe basic data structures and show users how to use the package. The function reference is given in the chapter 3.

Chapter 2

How to use HeroSVM

2.1 Design philosophy of HeroSVM

HeroSVM was implemented based on our proposed methods [6][7]. In order to facilitate the software portability and maintainess, an object-oriented method has been applied to design the package. Error handling was implemented for the robustness of software. HeroSVM is written using C++ language and developed under Microsoft visual C++ 6.0. In the current version, a dynamical link library in Windows or a shared library in Linux is provided to train SVM on a large-scale learning problem efficiently for research purpose in PC platform. We expect that HeroSVM can facilitate the training of support vector machine and solve some real-world problems in various engineering fields.

2.2 Components

The proposed SVM training algorithm consists of two components: parallel optimization and sequential optimization. Parallel optimization can be used to remove most non-support vectors quickly so that the computational cost for sequential optimization can be dramatically reduced. Sequential optimization is a working set algorithm, where several strategies such as kernel caching, shrinking are effectively integrated into it to speed up training.

2.3 Basic data structures

2.3.1 Kernel

In HeroSVM, some classical kernels such as radial basis function (RBF), polynomial and linear kernels have been implemented. Users can choose the above kernels or use their own customized kernel. Let a, b, c are kernel parameters. Then three classical kernels can be written as

$$\begin{aligned}
&\text{RBF} && \exp\left(-\tfrac{\|x_1-x_2\|^2}{2c}\right) && (2.1)\\
&\text{Polynomial} && ((<x_1,x_2>+b)/c)^a && (2.2)\\
&\text{Linear} && (<x_1,x_2>+b)/c && (2.3)
\end{aligned}$$

where $x_1 \in R^n$ and $x_2 \in R^n$, a is a positive integer, $< \cdot >$ and $\| \cdot \|$ denote dot product and Euclidean norm, respectively.

2.3.2 The size of working set

Usually the total training set is much larger than the number of final support vectors. The size of the working set should be large enough to contain these support vectors. Before the training, the number of support vectors is unknown. Therefore, users can estimate it in terms of the number of training samples. Experiments [6] have shown that generalization performance is insensitive to this paper if it is large enough.

2.3.3 Training data format

Parallel optimization needs two files. One sequentially stores the feature vectors of all training samples, which are shared by all classes. The other stores corresponding labels of training samples. A sample is labeled by an integer[1] in an interval from 0 to $m-1$, where m is the number of classes. After parallel optimization, training sets for each class are collected. Two files for each class will be generated. One is called feature vector file. The other is called target file, which sequentially stores target values (-1.0 or 1.0). The training method is one-against-the-rest. Note that feature vectors are stored in binary format and each component is represented as a 4-byte float data type. For example,

```
/* feature vector file */
Input feature vector of sample  1
Input feature vector of sample  2
...
Input feature vector of sample  n


/* label file */
Label of sample 1
Label of sample 2
...
Label of sample n
```

[1] This integer is called class label of a sample.

2.3.4 Summary information

After training for each class ends, the summary information will be stored in a file . We describe its format with an example as follows:

```
class Label = 0
User-specified kernel is used
The size of working set is 2000
The size of the training set is 7291
b_low = 0.528602 b_up = 0.507482
cache_hit = 6219030
total kernel evaluations  = 8928917
Actual kernel evaluations = 2709887
cache hit ratio = 0.696504
C= 10.0000
Bias = 0.518042
Iterations = 91
Training time (CPU seconds): 13.390000
Max alpha = 4.608595
Number of support vectors : 330
Number of bounded support vector: 0
|W|^2 = 159.62
margin of separation = 0.158301
```

In the above example, Bias means b in Eq.(1.2). The cache hit ratio can be calculated by

$$\text{cache hit ration} = \frac{\text{cache\_hit}}{\text{total kernel evaluations}} \tag{2.4}$$

b_low and b_up are two thresholds in modified SMO [11], and margin is equal to $1/ \parallel w \parallel^2$. If alpha value of one support vector is equal to C, we call it bounded support vector.

2.3.5 Save training results

We save training results into two files. One is used to store kernel paramters, support vectors and the corresponding α. The other (index file) is to store the sequential number of support vectors on the training set in order to merge them during the testing stage and reduce unnecessary kernel re-evaluations. Users can read training results with C language as follows:

```
fp = fopen(file name, ‘‘rb’’);
fread(&C, sizeof(float), 1, fp);
//for user-specified kernel, skip the following three statements.
fread(&kernel_para1, sizeof(float), 1, fp);
```

```
fread(&kernel_para2, sizeof(float), 1, fp);
fread(&kernel_para3, sizeof(float), 1, fp);
fread(&threshold, sizeof(float), 1, fp);
fread(&sv, sizeof(int), 1, fp);
for ( i = 0; i < sv; i++, vec += dim)
{
   //read the support vector
   fread(vec, sizeof(float), dim, fp);
   //read the target value of the above support vector
   fread(&target[i], sizeof(float), 1, fp);
   fread(&alpha[i], sizeof(float), 1, fp);
}
```

The format of the index file is illustrated by

```
sequential number of support vector 1
sequential number of support vector 2
...
sequential number of support vector n
```

2.4 Thread issues

Although a single thread is considered, parallel optimization of the proposed algorithm is suitable for multi-thread programming in a multi-processor platform. In the next version, multi-thread programming will be supported to speed up the training on a multi-processor's platform.

Chapter 3

Function Reference

This chapter describes the interface function reference. These functions are sorted by name. For each routine, we refer to format of xlib reference manual, including declarations of the arguments, return type and description.

3.1 SvmInit

Name

SvmInit – Set SVM training parameters and allocate the memory.

Header file

SVMTrain.h

Synopsis

```
int SvmInit(int nDim,
    int nWorkingSetSize ,
    int nTrainingSetSize ,
    float kernel_Para1,
    float kernel_Para2,
    float kernel_Para3,
    unsigned short int nKernelType,
    int ClassNum,
    float C,
    int TrainingType,
    int nApplicationType);
```

Parameters

nDim
The parameter specifies the dimension of input feature vector.

nWorkingSetSize
Size of working set

nTrainingSetSize
Number of training samples

kernel_Para1
Kernel parameter

kernel_Para2
Kernel parameter

kernel_Para3
Kernel parameter

nKernelType
Kernel type. There are four categories as follows:

1	RBF kernel
2	Polynomial kernel
3	Linear kernel

ClassNum
Number of classes

C
Upper bound of alpha in eq. (1.4).

TrainingType
Specify the optimization step

0	Parallel optimization
1	Sequential optimization

nApplicationType
Application type for the output of error message

0	Console application
1	Window application

Return Value

If succeed, return 0; otherwise -1

Remarks

See Also

SvmClean

Example

```
/* create a SVM with RBF kernel for parallel optimization*/
SvmInit(576, 8000, 60000, 1.0, 0.0, 0.3, 1, 10, 10.0, 0, 0);
```

3.2 SvmTrain_Parallel

Name

SvmTrain_Parallel – Train support vector machines with multi-classes to remove non-support vectors quickly.

Header file

SVMTrain.h

Synopsis

```
int SvmTrain_Parallel(char* szDataFilePathName, char* szLabelFilePathName,
  char* PositiveSamplesFilePathName, char* SaveFilePath);
```

Parameters

szDataFilePathName
: a pointer to feature vector filename

szLabelFilePathName
: a pointer to labelling filename. This file consists of identity numbers of samples.

PositiveSamplesFilePathName
: a pointer to positive sample file name. This file contains one sample of each class.

SaveFilePath
: Saved file path.

Return Value

If succeed, return 0; otherwise -1

Remarks

Feature vector and label files are both read/written in binary mode.

See Also

Example

```
SvmTrain_Parallel(‘‘d:\\data\\feature.dat’’, ‘‘d:\\data\\label.dat’’,
‘‘d:\\data\\pos_samples.dat’’, ‘‘d:\\data\\");
```

3.3 SvmTrain_Sequential

Name

SvmTrain_Sequential – Train support vector machines sequentially

Header file

SVMTrain.h

Synopsis

```
int SvmTrain_Sequential(char* szDataFilePathName, char* szTargetFilePathName,
    int nClassID, char* szSummaryFilePathName, char* szSvFilePathName,
    char* szSvIndexFilePathName);
```

Parameters

szDataFilePathName
: a pointer to feature vector filename

szTargetFilePathName
: a pointer to target filename. This file consists of target values of samples (-1.0 or 1.0).

nClassID
: Identity number of one class

szSummaryFilePathName
: a pointer to summary filename

szSvFilePathName
: a pointer to a filename. This file stores support vectors and trained parameters.

szSvIndexFilePathName
: a pointer to index filename

Return Value

If succeed, return 0; otherwise -1

Remarks

Feature vector and target files are both read/written in binary mode.

See Also

Example

3.4 SvmClean

Name

SvmClean – Free dynamically allocated memory in terms of training type.

Header file

SVMTrain.h

Synopsis

```
void SvmClean(int TrainingType);
```

Parameters

TrainingType
0: Parallel optimization
1: Sequential optimization

Return Value

NONE

Remarks

See Also

SvmInit

Example

```
SvmClean(0);
```

Chapter 4

Appendix

The chapter contains contact information and an example that shows how to use HeroSvm.

Jianxiong Dong, Ph.D.
Centre of Pattern Recognition and Machine Intelligence
1455 de Maisonneuve Blvd. West Suite EV003.403
Montreal, Quebec H3G 1M8
Canada

E-mail: jdongca2003@gmail.com
homepage: http://www.cenparmi.concordia.ca/~jdong

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "global.h"
#include "SVMTrain.h"

extern void GeneratePositiveSamples( );
extern void MergeOrderIndex( );
extern void GenerateMergeSet( );
extern void GenerateFinalSvIndex(  );
extern void CreateSvmTrainingDictionary( );
extern void LoadSv( );
extern void SvmTest( );

//Step 1
int TestParallelOptimization( )
{
    int nRes;
```

```
    nRes  = SvmInit(DIM, 8000, TotalSamplesNumber,
                    (float)1.0, (float)0.0, (float)0.3,
                     1, nClass, 10.0, 0, 0);
    if ( nRes )
    {
       printf("\n Intialization fails");
       return -1;
    }

    char dataFileName[256];
    char labelFileName[256];
    char posSampleFileName[256];

    strcpy(dataFileName, FilePath);
    strcat(dataFileName, "feature.dat");
    strcpy(labelFileName, FilePath);
    strcat(labelFileName, "label.dat");
    strcpy(posSampleFileName, FilePath);
    strcat(posSampleFileName, "pos_samples.dat");
    nRes = SvmTrain_Parallel(dataFileName,
                             labelFileName,
                             posSampleFileName,
                             FilePath);
    if ( nRes )
    {
        printf("\n Training fails");
        return -1;
    }

    SvmClean(0);
    return 0;
}

//step 2:
int TestSequentialOptimization( )
{
    char fname1[200];
    char fname2[200];
    char buf[200];
    char fname3[200];
    char fname4[200];
```

```
int  classNum = nClass;
int  i;
int nRes;
char fname[256];

int SvOfClass[nClass];
FILE* fp;

strcpy(fname, FilePath);
strcat(fname, "SvNum.dat");
fp = fopen(fname, "rb");
fread(SvOfClass, sizeof(int), nClass, fp);
fclose(fp);

int SizeOfWorkingSet;

for ( i = 0; i < nClass; i++)
{
    printf("\n Class = %d\n", i);
    SizeOfWorkingSet = SvOfClass[i];

    nRes = SvmInit(DIM, SizeOfWorkingSet, SvOfClass[i],
                   (float)1.0, (float)0.0, (float)0.3,
                   1, nClass, 10.0, 1, 0);
    if ( nRes )
    {
       printf("\n Intialization fails");
       return -1;
    }
    itoa(i, fname1, 10);
    strcpy(buf, FilePath);
    strcpy(fname4, buf);
    strcat(fname4, fname1);
    strcat(fname4, ".dat");
    strcat(fname1, ".tgt");
    strcat(buf, fname1);

    itoa(i, fname2, 10);
    strcpy(fname3, fname2);
    strcat(fname3, ".index");
    strcat(fname2, "_res.dat");
```

```
        strcpy(fname, FilePath);
        strcat(fname, "info2.txt");
        nRes = SvmTrain_Sequential(fname4, buf, i,
                        fname, fname2, fname3);
        if ( nRes )
        {
           printf("\n  SVM training fails");
           SvmClean(1);
           return -1;
        }
        SvmClean(1);
   }
   return 0;
}

int main( )
{
    printf("\n 1. Collect positive samples");
    printf("\n 2. Parallel optimization");
    printf("\n 3. Merge index");
    printf("\n 4. Generate training sets for sequential optimization");
    printf("\n 5. Sequential optimization");
    printf("\n 6. Create recognition dictionary for svm");
    printf("\n 7. Svm Testing\n");
    printf("\n Please activate the specified step: ");

    int step;

    scanf("%d", &step);
    switch(step)
    {
        case 1:
            GeneratePositiveSamples( );
            break;
        case 2:
            TestParallelOptimization( );
            break;
        case 3:
            MergeOrderIndex( );
            break;
```

```
        case 4:
            GenerateMergeSet( );
            break;
        case 5:
            TestSequentialOptimization( );
            break;
        case 6:
            GenerateFinalSvIndex( );
            CreateSvmTrainingDictionary( );
            break;
        case 7:
            LoadSv( );
            SvmTest( );
            break;
    }
    return 0;
}
```

Bibliography

[1] B.Scholkopf, C.J.C. Burges, and V. Vapnik, "Extracting support data for a given task," in *Proceedings, First International Conference on Knowledge Discovery and Data Mining,* U.M. Fayyad and R. Uthurusamy, Eds., pp. 252–257, AAAI Press, Menlo Park, CA, 1995.

[2] D. DeCoste and B. Scholkopf, "Training invariant support vector machines," *Machine Learning*, vol. 46, no. 1–3, pp. 161–190, 2002.

[3] T. Joachims, "Text categorization with support vector machine," in *Proceedings of European Conference on Machine Learning(ECML)*, 1998.

[4] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proceedings of the 1997 conference on Computer Vision and Pattern Recognition(CVPR'97)*, Puerto Rico, June 17–19, 1997.

[5] C.M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[6] Jian-xiong Dong, Krzyzak A., and Suen C.Y., A fast SVM Training Algorithm. *Proceedings of International workshop on Pattern Recognition with Support Vector Machines.* S.-W. Lee and A. Verri (Eds.) Springer Lecture Notes in Computer Science LNCS 2388, pp. 53–67, Niagara Falls, Canada, August 10, 2002.

[7] Jian-xiong Dong, Krzyzak A., and Suen C.Y., Fast SVM Training Algorithm with Decomposition on Very Large Datasets, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, pp. 603–618, April 2005.

[8] C.J.C. Burges, "A tutorial on support vector machines for pattern recognition," in *Data Mining and Knowledge Discovery,* pp. 121–167, 1998.

[9] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London,* vol. A(209), pp. 415–446, 1909.

[10] C.J.C. Burges and D.J. Crisp, "Uniqueness of the SVM solution," in *Advances in Neural Information Processing Systems,* to appear in NIPS 12.

[11] S.S. Keerthi, S.K. Shevade, C. Bhattachayya, and K.R.K. Murth, "Improvements to Platt's SMO Algorithm for SVM classifier design," *Neural Computation*, vol. 13, pp. 637–649, March, 2001

[12] Lin Chih-Jen, " Formulations of support vector machines: A note from an optimization point of view," *Neural Computation*, vol. 13, pp. 307–317, 2001.