# SVM+ and SVM+Multi-Task-Learning Software for Binary Classification

This software package implements SVM, SVM+ (Vapnik, 2006 [1]) and SVM+MTL (an SVM+ based Multi-Task Learning algorithm by Liang and Cherkassky, 2008 [2]) based on CVX (Matlab Software for Disciplined Convex Programming) and SVM-KM (SVM and Kernel Methods Matlab Toolbox).

## Background and Optimization Formulation

### 1. SVM

Given a training set $\{\{\mathbf{x}_i, y_i\}\}_{1\le i\le l}, \mathbf{x}_i \in R^d, y_i \in \{+1,-1\}$, SVM finds a maximum margin separating hyperplane $f(\mathbf{x}) = (\mathbf{w}, \mathbf{x}) + b$ between two classes. $f(\mathbf{x}) = (\mathbf{w}, \mathbf{x}) + b$ is also called decision function. SVM solves the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}(\mathbf{w},\mathbf{w}) + C\sum_{i=1}^{l}\xi_i \qquad \text{(OP1)}$$

subject to: $y_i((\mathbf{w},\mathbf{x}_i)+b) \ge 1-\xi_i$

$\xi_i > 0, i = 1,\dots,l$

$\xi_i, i=1,\dots l$ are called slack variables, which indicates the deviation from the margin borders. $(\mathbf{w},\mathbf{w})$ indicates the size of margin, which represents the model complexity of SVM. The coefficient $C$ controls the trade-off between complexity and proportion of nonseparable samples and must be selected by the user.

### 2. SVM+

Suppose that training data are the union of *t*>1 groups. Let us denote the indices from group r by $T_r = \{i_{n1},\dots,i_{n_r}\}, r = 1,\dots,t$. Then the total training data would be:

$\{\{X_r, Y_r\}, r = 1,\dots,t\}, \{X_r, Y_r\} = \{\{\mathbf{x}_{r_1}, y_{r_1}\},\dots,\{\mathbf{x}_{r_{nr}}, y_{r_{nr}}\}\}$ To account for the group information, Vapnik (2006) proposed to define the slacks for each group by some correcting function:

$$\xi_i = \xi_r(\mathbf{x}_i) = \varphi_r(\mathbf{x}_i, \mathbf{w}_r), i \in T_r, r = 1,\dots,t.$$

To define the correcting function $\xi_r(\mathbf{x}_i) = \phi_r(\mathbf{x}_i, \mathbf{w}_r)$ for group $T_r$, the input vectors $\mathbf{x}_i, i \in T_r$ are simultaneously mapped into two different Hilbert spaces: the decision space $\mathbf{z}_i = \Phi_z(\mathbf{x}_i) \in Z$ which defines the decision function and the correcting space $\mathbf{z}_i^r = \Phi_{z_r}(\mathbf{x}_i) \in Z_r$ which defines the set of correcting functions for a given group r. The correcting functions are represented by $\xi_r(\mathbf{x}_i) = (\mathbf{x}_i, \mathbf{w}_r) + d_r, r = \{1,...,t\}$

Compared to standard SVM, here the slack variables are restricted by the correcting functions, and the correcting functions represent additional information about the data. The goal is to find the decision function in decision space Z, $f(\mathbf{x}) = (\mathbf{w}, \Phi_Z(\mathbf{x})) + b$. SVM+ solves the following optimization problem:

$$\min_{w,w_1,..,w_t,b,d_1,..d_t} \frac{1}{2}(\mathbf{w},\mathbf{w}) + \frac{\gamma}{2}\sum_{r=1}^{t}(\mathbf{w}_r,\mathbf{w}_r) + C\sum_{r=1}^{t}\sum_{i\in T_r}\xi_i^r \qquad \text{(OP2)}$$

subject to:

$$y_i((\mathbf{w},\mathbf{z}_i)+b) \geq 1-\xi_i^r, i \in T_r, r=1,...,t$$

$$\xi_i^r \geq 0, i \in T_r, r=1,...,t$$

$$\xi_i^r = (\mathbf{z}_i^r, \mathbf{w}_r) + d_r, i \in T_r, r=1,...,t$$

## 3. SVM+MTL

Similar to SVM+, the input vectors $\mathbf{x}_i, i \in T_r$ are mapped simultaneously into two different Hilbert spaces: the decision space $\mathbf{z}_i = \Phi_z(\mathbf{x}_i) \in Z$ and the correcting space $\mathbf{z}_i^r = \Phi_{z_r}(\mathbf{x}_i) \in Z_r$ for each group r.

The goal is to find the *t* decision functions

$$f_r(\mathbf{x}) = (\Phi_z(\mathbf{x}),\mathbf{w}) + b + (\Phi_{z_r}(\mathbf{x}),\mathbf{w}_r) + d_r, r=1,...,t$$

Each decision function includes two parts: common deciision function $(\Phi_z(\mathbf{x}),\mathbf{w}) + b$ and unique correcting function $(\Phi_{z_r}(\mathbf{x}),\mathbf{w}_r) + d_r$. Common decision function is defined in the decision space Z and unique correcting function defined in the correcting space $Z_r$, so the final decision function actually involves two spaces: decision space and correcting space.

The *t* tasks are related in the sense that decision functions for different tasks share a common decision function. SVM+MTL solves the following optimization problem.

$$\min_{w,b} \frac{1}{2}(\mathbf{w},\mathbf{w}) + \frac{\gamma}{2}\sum_{r=1}^{t}(\mathbf{w}_r,\mathbf{w}_r) + C\sum_{r=1}^{t}\sum_{i\in T_r}\xi_i^r \qquad \text{(OP3)}$$

st: $y_i^r((\mathbf{w},\mathbf{z}_i)+b+(\mathbf{w}_r,\mathbf{z}_i^r)+d_r)\geq 1-\xi_i^r, i\in T_r, r=1,\ldots,t$

$\xi_i^r \geq 0, i\in T, r=1,\ldots,t$

## Software Installation:

### System requirements:

1, Type of OS: Microsoft Windows

2, Type of language: Matlab

### Installation:

1, Download the software package.

2, Add directories SVM-KM and cvx to the path of Matlab

3, Go to directory cvx/

4, Run the command cvx_setup to configure the software.

5, When cvx_setup completes, it may ask you to save the current path so that cvx will automatically be available when you start Matlab. If you do not do so, you will have to re-run cvx_setup every time you start Matlab.

## Software Description

```
1, [classifier] = trainSVMplusMTL(data,method,para)
   A unified training function
   input:
      data:  input data (t groups)
             data{i}.examples: An n by d matrix of m training instances with
d features (i=1,…,t)
             data{i}.labels: An n by 1 vector of training labels(+1 or -1)
(i=1,…,t)

      method: 'svm'     -- SVM
              'svmplus' -- SVM+(gamma)
              'svmmtl'  -- SVM+MTL
      para:  control parameters
             .C [1 by 1] parameter C for all methods
             .g [1 by 1] parameter gamma for SVM+ and SVM+MTL
             .kernel1 [string] decision space kernel for SVM+ and SVM+MTL
                 'jcb' linear kernel
                 'poly' polynomial kernel
                 'gaussian' rbf kernel
                .kerneloption1 [1 by 1] decision space kernel parameter
                .kernel2 [string] correcting space kernel for SVM+ and SVM+MTL
                 'jcb' linear kernel
                 'poly' polynomial kernel
                 'gaussian' rbf kernel
                .kerneloption2 [1 by 1] correcting space kernel parameter
```

```
  Output:
      Classifier
             A struct variable passed to functions explained below.
2, [pred] = predSVM(classifier,data)
   A function to compute classification accuracy for SVM method by
using ‘classifier’ trained by trainSVMplusMTL
   Inputs:
     classifier: [struct] model trained by trainSVMplusMTL
      data: test data (1 group)
             data{1}.examples: An n by d matrix of m training instances with
d features
             data{1}.labels: An n by 1 vector of training labels(+1 or -1)
    Output:
       pred: [1 by 1] classification accuracy

3,  [pred] = predSVMplus(classifier,data)
   A function to compute classification accuracy for SVM+ method by
using ‘classifier’ trained by trainSVMplusMTL
   Inputs:
     classifier: [struct] model trained by trainSVMplusMTL
      data: test data (t groups)
             data{i}.examples: An n by d matrix of m training instances with
d features (i=1,…,t)
             data{i}.labels: An n by 1 vector of training labels(+1 or -1)
(i=1,…,t)
    Output:
       pred: [1 by 1] classification accuracy

4, [pred] = predSVMMTL(classifier,data)
   A function to compute classification accuracy for SVM+MTL method by
using ‘classifier’ trained by trainSVMplusMTL
   Inputs:
     classifier: [struct] model trained by trainSVMplusMTL
      data: test data (t groups)
             data{i}.examples: An n by d matrix of m training instances with
d features (i=1,…,t)
             data{i}.labels: An n by 1 vector of training labels(+1 or -1)
(i=1,…,t)
    Output:
       pred: [1 by 1] classification accuracy
```

## Software Limitations

The software package solves dual problems of SVM, SVM+ and SVM+MTL using CVX (Matlab Software for Disciplined Convex Programming). The dimensionality of the dual optimization problem is given by the training sample size of the original problem. Therefore, the training sample size determines execution time. For a problem with about 300 samples (shown in the example section), using 5-fold double resampling for tuning complexity parameters, execution of all 3 methods, SVM, SVM+ and SVM+MTL, took ~ 10 hours on our computer(Intel Core 2 Duo @ 3GHz, 4GB of RAM, Microsoft Windows XP Professional x64 Edition, Matlab 7.6.0) . For a problem with more than 2000 samples, using typical 5-fold or 10-fold cross-validation, computational time would be prohibitively long.

## Model Selection

Each of the methods, SVM, SVM+ and SVM+MTL, has complexity parameter(s) that need to be optimally tuned for a given data set. SVM has a single parameter $C$ when linear kernel is used, and two parameters $C$, $\sigma$ when RBF kernel is used. Both SVM+ and SVM+MTL have 3 parameters $C$ (as in standard linear SVM), $\gamma$ and $\sigma$ (RBF width parameter). Estimation of prediction (test) error and model selection (parameter tuning) is performed via standard double-resampling procedure:

1. For estimating prediction error of a particular method, use 5-fold cross-validation, so that 80% of data samples are used for training and 20% of the data are used as test data.
2. For each training fold, parameter tuning (model selection) is performed via resampling within the training fold.

## Example

### Objective

This section describes empirical comparisons of various modeling approaches for classification, such as single SVM (sSVM), multiple SVM (mSVM, where SVM classifiers are estimated separately for each group separately), SVM+ and SVM+MTL. All comparisons use linear kernels for sSVM and mSVM. The common decision space for SVM+ and SVM+MTL uses linear kernel while the unique correction space use RBF(Gaussian) kernel. All methods under comparison use the same experimental procedure (double resampling) described in model selection.

## Data Description

This dataset is Ljubljana breast cancer dataset available at UCI machine learning repository. It consists of 286 instances, each with 9 attributes. The dataset contains 9 instances with missing values, so only the remaining 277 instances are used for modeling. The goal is to predict the class (no-recurrence-events or recurrence-events) from 9 attributes. Variable 'age' was selected to separate the data into 3 different groups: group 1( $age < 47$ , 94 instances), group 2($47 \le age < 55$, 93 instances) and group 3($age \ge 55$, 90 instances). Possible choices of parameters for sSVM and mSVM are $\mathrm{C} = [0.1\ 1\ 10\ 100]$ and $\sigma = [0.25\ 0.5\ 1\ 2\ 4]$ . Possible choices of parameters for SVM+ and SVM+MTL are $\mathrm{C} = [0.1\ 1\ 10\ 100]$ , $\gamma = [10\ 1\ 0.1\ 0.01\ 0.001]$ and $\sigma = [0.25\ 0.5\ 1\ 2\ 4]$ . Final comparison results show an average test error rate (for each method) and its standard deviation. The average test error rate and its standard deviation are calculated from 5 estimated test errors in the 5-fold cross-validation procedure.

## Code:

```
%   By Feng Cai, Double Resampling
%   Jul.23,2008
%   A Test aiming to compare sSVM, mSVM, SVM+ and SVM+MTL
%   A test on Ljubljana breastcancer dataset from uci

clear all;

[age, men, inv, nod, deg, bre, qua, irr, rec, cla]= textread('breastcancer.dat', ...
    '%f%f%f%f%f%f%s%f%f%f','delimiter',',');

data = [age, men, inv, nod, deg, bre, irr, cla, rec];

% # of rows
N1 = size(data, 1);

% # of columns
N2 = size(data,2);

% remove missing values
alldata = [];
for i = 1:N1
    if size(find(data(i,:)==10000),2)<1 && data(i,2)~=2
        alldata = [alldata; data(i,:)];
    end
end
```

```
% # of rows
N1 = size(alldata, 1);

% # of columns
N2 = size(alldata,2);

alldatag = [alldata(:,1) alldata(:,3:N2-1) alldata(:,2) alldata(:,N2)];
alldata = alldatag;

% order all data by increasing age
alldata = sortrows(alldata);


% scaled input variables to zero mean 1 variance
for i = 1:N2-2
    tmp1 = min(alldata(:,i));
    tmp2 = max(alldata(:,i));
    alldata(:,i) = (alldata(:,i)-tmp1)/(tmp2-tmp1);
end

train = [];
test = [];
finalaccuracy = [];
finalaccuracy_m = [];
finalaccuracy_plus = [];
finalaccuracy_mtl = [];

% variables to record optimal parameter for each partition
C_sSVM = zeros(5,1);
C_mSVM_1 = zeros(5,1);
C_mSVM_2 = zeros(5,1);
C_SVMplus = zeros(5,1);
gama_SVMplus = zeros(5,1);
sigma_SVMplus = zeros(5,1);
C_SVMmtl = zeros(5,1);
gama_SVMmtl = zeros(5,1);
sigma_SVMmtl = zeros(5,1);


for i = 1:5
    testindex = [];
    trainindex = [];
    % compute the index
    for j = 1:N1
        if mod(j,5)==mod(i,5)
            testindex = [testindex j];
        else
            trainindex = [trainindex j];
        end
    end

    % train data for this partition
    train = alldata(trainindex,:);
    traing = alldatag(trainindex,:);

    % test data for this partition
    test = alldata(testindex,:);
    testg = alldatag(testindex,:);

    % construct the test data
    testdata{1}.examples = [];
    testdata{1}.labels = [];
    testdata{2}.examples = [];
    testdata{2}.labels = [];
    traindata{1}.examples = [];
    traindata{1}.labels = [];
    traindata{2}.examples = [];
    traindata{2}.labels = [];

    % format the test data to use the interface your provided
    % testdata will be the test data for this partition
```

```matlab
for j = 1:size(test,1)
    if test(j,N2-1)==1
        testdata{2}.examples = [testdata{2}.examples; test(j,1:N2-2)];
        if test(j,N2)==1
            testdata{2}.labels = [testdata{2}.labels; 1];
        else
            testdata{2}.labels = [testdata{2}.labels; -1];
        end
    else
        testdata{1}.examples = [testdata{1}.examples; test(j,1:N2-2)];
        if test(j,N2)==1
            testdata{1}.labels = [testdata{1}.labels; 1];
        else
            testdata{1}.labels = [testdata{1}.labels; -1];
        end
    end
end

% format the traindata to use the interface you provided
% traindata will be training data for this partition
for j = 1:size(train,1)
    if train(j,N2-1)==1
        traindata{2}.examples = [traindata{2}.examples; train(j,1:N2-2)];
        if train(j,N2)==1
            traindata{2}.labels = [traindata{2}.labels; 1];
        else
            traindata{2}.labels = [traindata{2}.labels; -1];
        end
    else
        traindata{1}.examples = [traindata{1}.examples; train(j,1:N2-2)];
        if train(j,N2)==1
            traindata{1}.labels = [traindata{1}.labels; 1];
        else
            traindata{1}.labels = [traindata{1}.labels; -1];
        end
    end
end

% resampling to tune the model parameter
C = [0.1 1 10 100];
gama = [10 1 0.1 0.01 0.001];
sigma = [0.25 0.5 1 2 4];
N3 = size(train,1);

% double resampling
for j = 1:5
    testindex = [];
    trainindex = [];

    for k = 1:N3
        if mod(k,5)==mod(j,5)
            testindex = [testindex k];
        else
            trainindex = [trainindex k];
        end
    end

    % partition on the training data from outer loop
    traintmp = train(trainindex,:);
    traintmpg = traing(trainindex,:);
    testtmp = train(testindex,:);
    testtmpg = traing(testindex,:);

    traintune{1}.examples = [];
    traintune{1}.labels = [];
    traintune{2}.examples = [];
    traintune{2}.labels = [];
    testtune{1}.examples = [];
    testtune{1}.labels = [];
    testtune{2}.examples = [];
    testtune{2}.labels = [];
    for k = 1:size(traintmp,1)
        if traintmp(k,N2-1)==1
            traintune{2}.examples = [traintune{2}.examples; traintmp(k,1:N2-2)];
            if traintmp(k,N2)==1
                traintune{2}.labels = [traintune{2}.labels; 1];
            else
                traintune{2}.labels = [traintune{2}.labels; -1];
            end
        else
            traintune{1}.examples = [traintune{1}.examples; traintmp(k,1:N2-2)];
            if traintmp(k,N2)==1
                traintune{1}.labels = [traintune{1}.labels; 1];
            else
                traintune{1}.labels = [traintune{1}.labels; -1];
            end
        end
    end

    for k = 1:size(testtmp,1)
        if testtmp(k,N2-1)==1
            testtune{2}.examples = [testtune{2}.examples; testtmp(k,1:N2-2)];
            if testtmp(k,N2)==1
                testtune{2}.labels = [testtune{2}.labels; 1];
            else
                testtune{2}.labels = [testtune{2}.labels; -1];
            end
        else
            testtune{1}.examples = [testtune{1}.examples; testtmp(k,1:N2-2)];
            if testtmp(k,N2)==1
                testtune{1}.labels = [testtune{1}.labels; 1];
            else
                testtune{1}.labels = [testtune{1}.labels; -1];
            end
        end
    end

    % compute accuracy for different model parameters
    for inx1 = 1:size(C,2)
            % setting paramters
        para.C = C(inx1);
        para.g = 1;
        para.kernel1 = 'jcb';   % decision space kernel linear kernel
        para.kerneloption1 =1;  % decision space kernel parameter
        para.kernel2 ='gaussian'; % correcting space kernel rbf kernel
        para.kerneloption2 = 1;   % correcting space kernle parameter

        svm_classifier = trainSVMplusMTLtest(traintune,'svm',para);
        accuracy(i,j,inx1) = predSVM(svm_classifier,testtune);

        svm_classifier1 = trainSVMplusMTLtest(traintune(1),'svm',para);
        accuracy_m1(i,j,inx1) = predSVM(svm_classifier1,testtune(1));

        svm_classifier2 = trainSVMplusMTLtest(traintune(2),'svm',para);
        accuracy_m2(i,j,inx1) = predSVM(svm_classifier2,testtune(2));

        for inx2 = 1:size(gama,2)
            for inx3 = 1:size(sigma,2)
                para.g = gama(inx2);
                para.kerneloption2 = sigma(inx3);

                svmp_classifier = trainSVMplusMTLtest(traintune,'svmplus',para);
                accuracy_plus(i,j,inx1,inx2,inx3) = predSVMplus(svmp_classifier,testtune);


                svmmtl_classifier = trainSVMplusMTLtest(traintune,'svmmtl',para);
                accuracy_mtl(i,j,inx1,inx2,inx3) = predSVMMTL(svmmtl_classifier,testtune);
            end
        end
    end
end
```

```
% select optimal parameters
innersvm_maxaccuracy(i) = 0;
opt_C(i) = 0.001;
innersvmm1_maxaccuracy(i) = 0;
opt_Cm1(i) = 0.001;
innersvmm2_maxaccuracy(i) = 0;
opt_Cm2(i) = 0.001;
innersvmplus_maxaccuracy(i) = 0;
opt_Cplus(i) = 0.1;
opt_gamaplus(i) = 10;
opt_sigmaplus(i) = 0.5;
innersvmmtl_maxaccuracy(i) = 0;
opt_Cmtl(i) = 0.1;
opt_gamamtl(i) = 10;
opt_sigmamtl(i) = 0.5;

for inx1 = 1:size(C,2)
    acc_svm = mean(accuracy(i,:,inx1));
    if innersvm_maxaccuracy(i)<acc_svm
        innersvm_maxaccuracy(i)=acc_svm;
        opt_C(i) = C(inx1);
    end

    acc_svm1 = mean(accuracy_m1(i,:,inx1));
    if innersvmm1_maxaccuracy(i)<acc_svm1
        innersvmm1_maxaccuracy(i) = acc_svm1;
        opt_Cm1(i) = C(inx1);
    end

    acc_svm2 = mean(accuracy_m2(i,:,inx1));
    if innersvmm2_maxaccuracy(i)<acc_svm2
        innersvmm2_maxaccuracy(i) = acc_svm2;
        opt_Cm2(i) = C(inx1);
    end

    for inx2 = 1:size(gama,2)
        for inx3 = 1:size(sigma,2)
            acc_svmplus = mean(accuracy_plus(i,:,inx1,inx2,inx3));
            if innersvmplus_maxaccuracy(i)<acc_svmplus
                innersvmplus_maxaccuracy(i)=acc_svmplus;
                opt_Cplus(i) = C(inx1);
                opt_gamaplus(i) = gama(inx2);
                opt_sigmaplus(i) = sigma(inx3);
            end

            acc_svmmtl = mean(accuracy_mtl(i,:,inx1,inx2,inx3));
            if innersvmmtl_maxaccuracy(i)<acc_svmmtl
                innersvmmtl_maxaccuracy(i)=acc_svmmtl;
                opt_Cmtl(i) = C(inx1);
                opt_gamamtl(i) = gama(inx2);
                opt_sigmamtl(i) = sigma(inx3);
            end
        end
    end
end

para.g = 1;
para.kernel1 = 'jcb';   % decision space kernel linear kernel
para.kerneloption1 =1;  % decision space kernel parameter
para.kernel2 ='gaussian'; % correcting space kernel rbf kernel
para.kerneloption2 = 1;   % correcting space kernle parameter

% compute the prediction accuracy
para.C = opt_C(i);
C_sSVM(i,1) = opt_C(i);
svm_classifier = trainSVMplusMTLtest(traindata,'svm',para);
svm_pred = predSVM(svm_classifier,testdata);
finalaccuracy = [finalaccuracy; svm_pred];

para.C = opt_Cm1(i);
```

```
    C_mSVM_1(i,1) = opt_Cm1(i);
    svm_classifier1 = trainSVMplusMTLtest(traindata(1),'svm',para);
    svm_pred1 = predSVM(svm_classifier1,testdata(1));

    para.C = opt_Cm2(i);
    C_mSVM_2(i,1) = opt_Cm2(i);
    svm_classifier2 = trainSVMplusMTLtest(traindata(2),'svm',para);
    svm_pred2 = predSVM(svm_classifier2,testdata(2));

    finalaccuracy_m = [finalaccuracy_m; (svm_pred1*size(testdata{1}.examples,1)...
        +svm_pred2*size(testdata{2}.examples,1))/(size(testdata{1}.examples,1)+...
        size(testdata{2}.examples,1))];

    para.C = opt_Cplus(i);
    C_SVMplus(i,1) = opt_Cplus(i);
    para.g = opt_gamaplus(i);
    gama_SVMplus(i,1) = opt_gamaplus(i);
    para.kerneloption2 = opt_sigmaplus(i);
    sigma_SVMplus(i,1) = opt_sigmaplus(i);
    svmp_classifier = trainSVMplusMTLtest(traindata,'svmplus',para);
    svmp_pred = predSVMplus(svmp_classifier,testdata);
    finalaccuracy_plus = [finalaccuracy_plus; svmp_pred];

    para.C = opt_Cmtl(i);
    C_SVMmtl(i,1) = opt_Cmtl(i);
    para.g = opt_gamamtl(i);
    gama_SVMmtl(i,1) = opt_gamamtl(i);
    para.kerneloption2 = opt_sigmamtl(i);
    sigma_SVMmtl(i,1) = opt_sigmamtl(i);
    svmmtl_classifier = trainSVMplusMTLtest(traindata,'svmmtl',para);
    svmmtl_pred = predSVMMTL(svmmtl_classifier,testdata);
    finalaccuracy_mtl = [finalaccuracy_mtl; svmmtl_pred];
end

fprintf('The prediction accuracy results:\n');
fprintf('sSVM:    %f ', mean(finalaccuracy));
fprintf([setstr(177), ' %f\n'], std(finalaccuracy));
fprintf('SVM+:    %f ', mean(finalaccuracy_plus));
fprintf([setstr(177), ' %f\n'], std(finalaccuracy_plus));
fprintf('mSVM:    %f ', mean(finalaccuracy_m));
fprintf([setstr(177), ' %f\n'], std(finalaccuracy_m));
fprintf('SVM+MTL: %f ', mean(finalaccuracy_mtl));
fprintf([setstr(177), ' %f\n'], std(finalaccuracy_mtl));
```

## The final result:

```
The prediction accuracy results:
sSVM:    0.707272 ± 0.062184
SVM+:    0.750650 ± 0.048384
mSVM:    0.703962 ± 0.016182
SVM+MTL: 0.765258 ± 0.034435
```

## Reference

[1] Vapnik, V. Empirical Inference Science Afterword of 2006, Springer, 2006.
[2] Liang, L. and Cherkassky, V. Connection between SVM+ and Multi-Task Learning, IJCNN, 2008.