
--- MATLAB/OCTAVE interface of LIBSVM ---

Table of Contents
=================

- Introduction
- Installation
- Usage
- Returned Model Structure
- Other Utilities
- Examples
- Additional Information

Introduction
============

This tool provides a simple interface to LIBSVM, a library for support vector
machines (http://www.csie.ntu.edu.tw/~cjlin/libsvm). It is very easy to use as
the usage and the way of specifying parameters are the same as that of LIBSVM.

Installation
============

On Windows systems, pre-built binary files are already in the directory '..\windows', so no need to conduct installation. Now we provide binary files only for 64bit MATLAB on Windows. If you would like to re-build the package, please rely on the following steps.

We recommend using make.m on both MATLAB and OCTAVE. Just type 'make' to build 'libsvmread.mex', 'libsvmwrite.mex', 'svmtrain.mex', and 'svmpredict.mex'.

On MATLAB or Octave:

```
        >> make
```

If make.m does not work on MATLAB (especially for Windows), try 'mex -setup' to choose a suitable compiler for mex. Make sure your compiler is accessible and workable. Then type 'make' to start the installation.

Example:

```
      matlab>> mex -setup
      (ps: MATLAB will show the following messages to setup default compiler.)
      Please choose your compiler for building external interface (MEX) files:
```

```
      Would you like mex to locate installed compilers [y]/n? y
      Select a compiler:
      [1] Microsoft Visual C/C++ version 7.1 in C:\Program
Files\Microsoft Visual Studio
      [0] None
      Compiler: 1
      Please verify your choices:
      Compiler: Microsoft Visual C/C++ 7.1
      Location: C:\Program Files\Microsoft Visual Studio
      Are these correct?([y]/n): y

      matlab>> make
```

On Unix systems, if neither make.m nor 'mex -setup' works, please use Makefile and type 'make' in a command window. Note that we assume your MATLAB is installed in '/usr/local/matlab'. If not, please change MATLABDIR in Makefile.

Example:

```
        linux> make
```

To use octave, type 'make octave':

Example:

```
      linux> make octave
```

For a list of supported/compatible compilers for MATLAB, please check the following page:

http://www.mathworks.com/support/compilers/current_release/

Usage
=====

```
matlab> model = svmtrain(training_label_vector, training_instance_matrix
[, 'libsvm_options']);

        -training_label_vector:
            An m by 1 vector of training labels (type must be double).
        -training_instance_matrix:
            An m by n matrix of m training instances with n features.
            It can be dense or sparse (type must be double).
        -libsvm_options:
            A string of training options in the same format as that of
LIBSVM.

matlab> [predicted_label, accuracy, decision_values/prob_estimates] =
svmpredict(testing_label_vector, testing_instance_matrix, model [,
'libsvm_options']);

        -testing_label_vector:
            An m by 1 vector of prediction labels. If labels of test
            data are unknown, simply use any random values. (type must be
double)
```

-testing_instance_matrix:
An m by n matrix of m testing instances with n features.
It can be dense or sparse. (type must be double)
-model:
The output of svmtrain.
-libsvm_options:
A string of testing options in the same format as that of LIBSVM.

Returned Model Structure

The 'svmtrain' function returns a model which can be used for future prediction. It is a structure and is organized as [Parameters, nr_class, totalSV, rho, Label, ProbA, ProbB, nSV, sv_coef, SVs]:

-Parameters: parameters
-nr_class: number of classes; = 2 for regression/one-class svm
-totalSV: total #SV
-rho: -b of the decision function(s) wx+b
-Label: label of each class; empty for regression/one-class SVM
-ProbA: pairwise probability information; empty if -b 0 or in one-class SVM
-ProbB: pairwise probability information; empty if -b 0 or in one-class SVM
-nSV: number of SVs for each class; empty for regression/one-class SVM
-sv_coef: coefficients for SVs in decision functions
-SVs: support vectors

If you do not use the option '-b 1', ProbA and ProbB are empty matrices. If the '-v' option is specified, cross validation is conducted and the returned model is just a scalar: cross-validation accuracy for classification and mean-squared error for regression.

More details about this model can be found in LIBSVM FAQ (http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html) and LIBSVM implementation document (http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf).

Result of Prediction

The function 'svmpredict' has three outputs. The first one, predictd_label, is a vector of predicted labels. The second output, accuracy, is a vector including accuracy (for classification), mean squared error, and squared correlation coefficient (for regression). The third is a matrix containing decision values or probability estimates (if '-b 1' is specified). If k is the number of classes in training data, for decision values, each row includes results of predicting k(k-1)/2 binary-class SVMs. For classification, k = 1 is a special case. Decision value +1 is returned for each testing instance, instead of an empty vector. For probabilities, each row contains k values indicating the probability that the testing instance is in each class.

Note that the order of classes here is the same as 'Label' field
in the model structure.

Other Utilities

A matlab function libsvmread reads files in LIBSVM format:

```
[label_vector, instance_matrix] = libsvmread('data.txt');
```

Two outputs are labels and instances, which can then be used as inputs
of svmtrain or svmpredict.

A matlab function libsvmwrite writes Matlab matrix to a file in LIBSVM
format:

```
libsvmwrite('data.txt', label_vector, instance_matrix]
```

The instance_matrix must be a sparse matrix. (type must be double)
For 32bit and 64bit MATLAB on Windows, pre-built binary files are ready
in the directory `..\windows', but in future releases, we will only
include 64bit MATLAB binary files.

These codes are prepared by Rong-En Fan and Kai-Wei Chang from National
Taiwan University.

Examples

Train and test on the provided data heart_scale:

```
matlab> [heart_scale_label, heart_scale_inst] =
libsvmread('../heart_scale');
matlab> model = svmtrain(heart_scale_label, heart_scale_inst, '-c 1 -g
0.07');
matlab> [predict_label, accuracy, dec_values] =
svmpredict(heart_scale_label, heart_scale_inst, model); % test the
training data
```

For probability estimates, you need '-b 1' for training and testing:

```
matlab> [heart_scale_label, heart_scale_inst] =
libsvmread('../heart_scale');
matlab> model = svmtrain(heart_scale_label, heart_scale_inst, '-c 1 -g
0.07 -b 1');
matlab> [heart_scale_label, heart_scale_inst] =
libsvmread('../heart_scale');
matlab> [predict_label, accuracy, prob_estimates] =
svmpredict(heart_scale_label, heart_scale_inst, model, '-b 1');
```

To use precomputed kernel, you must include sample serial number as
the first column of the training and testing data (assume your kernel
matrix is K, # of instances is n):

```
matlab> K1 = [(1:n)', K]; % include sample serial number as first column
matlab> model = svmtrain(label_vector, K1, '-t 4');
matlab> [predict_label, accuracy, dec_values] = svmpredict(label_vector,
K1, model); % test the training data
```

We give the following detailed example by splitting heart_scale into 150 training and 120 testing data. Constructing a linear kernel matrix and then using the precomputed kernel gives exactly the same testing error as using the LIBSVM built-in linear kernel.

```
matlab> [heart_scale_label, heart_scale_inst] =
libsvmread('../heart_scale');
matlab>
matlab> % Split Data
matlab> train_data = heart_scale_inst(1:150,:);
matlab> train_label = heart_scale_label(1:150,:);
matlab> test_data = heart_scale_inst(151:270,:);
matlab> test_label = heart_scale_label(151:270,:);
matlab>
matlab> % Linear Kernel
matlab> model_linear = svmtrain(train_label, train_data, '-t 0');
matlab> [predict_label_L, accuracy_L, dec_values_L] =
svmpredict(test_label, test_data, model_linear);
matlab>
matlab> % Precomputed Kernel
matlab> model_precomputed = svmtrain(train_label, [(1:150)',
train_data*train_data'], '-t 4');
matlab> [predict_label_P, accuracy_P, dec_values_P] =
svmpredict(test_label, [(1:120)', test_data*train_data'],
model_precomputed);
matlab>
matlab> accuracy_L % Display the accuracy using linear kernel
matlab> accuracy_P % Display the accuracy using precomputed kernel
```

Note that for testing, you can put anything in the testing_label_vector. For more details of precomputed kernels, please read the section ``Precomputed Kernels'' in the README of the LIBSVM package.

Additional Information

======================

This interface was initially written by Jun-Cheng Chen, Kuan-Jen Peng, Chih-Yuan Yang and Chih-Huai Cheng from Department of Computer Science, National Taiwan University. The current version was prepared by Rong-En Fan and Ting-Fan Wu. If you find this tool useful, please cite LIBSVM as follows

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

For any question, please contact Chih-Jen Lin <cjlin@csie.ntu.edu.tw>, or check the FAQ page:

http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#/Q9:_MATLAB_interface