cjlin1 / libsvm

No description or website provided.

1,002 commits **1** branch **30** releases **9** contributors

cjlin1 libsvm.jar for 3.21 release Latest commit `2fdc614` on Dec 14, 2015

java	libsvm.jar for 3.21 release	6 months ago
matlab	Catch the exception to display the error message	11 months ago
python	in get_sv_coef() we used xrange to save the storage but in python 3	6 months ago
svm-toy	fix the Makefile in the ./svm-toy/qt	10 months ago
tools	modify grid.py to support path containing space character.	2 years ago
windows	correct the modes of the windows binaries files from last commit	6 months ago
COPYRIGHT	change version number in files and 2013 to 2014 in COPYRIGHT	3 years ago
FAQ.html	FAQ for the release of libsvm 3.21	6 months ago
Makefile	Fix typo "-W1" to the correct option "-Wl".	4 years ago
Makefile.win	change windows binaries from 32-bit to 64-bit	6 months ago
README	change windows binaries from 32-bit to 64-bit	6 months ago
heart_scale	This commit was generated by cvs2svn to compensate for changes in r2,	13 years ago
svm-predict.c	Add return 0 into print_null in svm-predict.c,	4 years ago
svm-scale.c	better description about when to use -l 0 in svm-scale	11 months ago
svm-train.c	[Fix] Change 'long int elements, j;' long int into size_t	2 years ago
svm.cpp	Fix an issue where select_working_set() does not work correctly	11 months ago
svm.def	Regenerate the binary files after fixing svm.def	4 years ago
svm.h	change version in .h files to 3.21 for the new release	6 months ago

README

```
Libsvm is a simple, easy-to-use, and efficient software for SVM
classification and regression. It solves C-SVM classification, nu-SVM
classification, one-class-SVM, epsilon-SVM regression, and nu-SVM
regression. It also provides an automatic model selection tool for
C-SVM classification. This document explains the use of libsvm.

Libsvm is available at
http://www.csie.ntu.edu.tw/~cjlin/libsvm
Please read the COPYRIGHT file before using libsvm.

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

- Quick Start
- Installation and Data Format
- `svm-train' Usage
- `svm-predict' Usage
```

```
- `svm-scale' Usage
- Tips on Practical Use
- Examples
- Precomputed Kernels
- Library Usage
- Java Version
- Building Windows Binaries
- Additional Tools: Sub-sampling, Parameter Selection, Format checking, etc.
- MATLAB/OCTAVE Interface
- Python Interface
- Additional Information

Quick Start
===========

If you are new to SVM and if the data is not large, please go to
`tools' directory and use easy.py after installation. It does
everything automatic -- from data scaling to parameter selection.

Usage: easy.py training_file [testing_file]

More information about parameter selection can be found in
`tools/README.'

Installation and Data Format
============================

On Unix systems, type `make' to build the `svm-train' and `svm-predict'
programs. Run them without arguments to show the usages of them.

On other systems, consult `Makefile' to build them (e.g., see
'Building Windows binaries' in this file) or use the pre-built
binaries (Windows binaries are in the directory `windows').

The format of training and testing data file is:

<label> <index1>:<value1> <index2>:<value2> ...
.
.
.

Each line contains an instance and is ended by a '\n' character.  For
classification, <label> is an integer indicating the class label
(multi-class is supported). For regression, <label> is the target
value which can be any real number. For one-class SVM, it's not used
so can be any number.  The pair <index>:<value> gives a feature
(attribute) value: <index> is an integer starting from 1 and <value>
is a real number. The only exception is the precomputed kernel, where
<index> starts from 0; see the section of precomputed kernels. Indices
must be in ASCENDING order. Labels in the testing file are only used
to calculate accuracy or errors. If they are unknown, just fill the
first column with any numbers.

A sample classification data included in this package is
`heart_scale'. To check if your data is in a correct form, use
`tools/checkdata.py' (details in `tools/README').

Type `svm-train heart_scale', and the program will read the training
data and output the model file `heart_scale.model'. If you have a test
set called heart_scale.t, then type `svm-predict heart_scale.t
heart_scale.model output' to see the prediction accuracy. The `output'
file contains the predicted class labels.

For classification, if training data are in only one class (i.e., all
labels are the same), then `svm-train' issues a warning message:
`Warning: training data in only one class. See README for details,'
which means the training data is very unbalanced. The label in the
training data is directly returned when testing.

There are some other useful programs in this package.

svm-scale:

        This is a tool for scaling input data file.
```

```
svm-toy:

        This is a simple graphical interface which shows how SVM
        separate data in a plane. You can click in the window to
        draw data points. Use "change" button to choose class
        1, 2 or 3 (i.e., up to three classes are supported), "load"
        button to load data from a file, "save" button to save data to
        a file, "run" button to obtain an SVM model, and "clear"
        button to clear the window.

        You can enter options in the bottom of the window, the syntax of
        options is the same as `svm-train'.

        Note that "load" and "save" consider dense data format both in
        classification and the regression cases. For classification,
        each data point has one label (the color) that must be 1, 2,
        or 3 and two attributes (x-axis and y-axis values) in
        [0,1). For regression, each data point has one target value
        (y-axis) and one attribute (x-axis values) in [0, 1).

        Type `make' in respective directories to build them.

        You need Qt library to build the Qt version.
        (available from http://www.trolltech.com)

        You need GTK+ library to build the GTK version.
        (available from http://www.gtk.org)

        The pre-built Windows binaries are in the `windows'
        directory. We use Visual C++ on a 32-bit machine, so the
        maximal cache size is 2GB.

`svm-train' Usage
=================

Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
        0 -- C-SVC              (multi-class classification)
        1 -- nu-SVC             (multi-class classification)
        2 -- one-class SVM
        3 -- epsilon-SVR        (regression)
        4 -- nu-SVR             (regression)
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
        4 -- precomputed kernel (kernel values in training_set_file)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default
0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : quiet mode (no outputs)


The k in the -g option means the number of attributes in the input data.

option -v randomly splits the data into n parts and calculates cross
validation accuracy/mean squared error on them.

See libsvm FAQ for the meaning of outputs.

`svm-predict' Usage
===================
```

```
Usage: svm-predict [options] test_file model_file output_file
options:
-b probability_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM
only 0 is supported

model_file is the model file generated by svm-train.
test_file is the test data you want to predict.
svm-predict will produce output in the output_file.

`svm-scale' Usage
=================

Usage: svm-scale [options] data_filename
options:
-l lower : x scaling lower limit (default -1)
-u upper : x scaling upper limit (default +1)
-y y_lower y_upper : y scaling limits (default: no y scaling)
-s save_filename : save scaling parameters to save_filename
-r restore_filename : restore scaling parameters from restore_filename

See 'Examples' in this file for examples.

Tips on Practical Use
=====================

* Scale your data. For example, scale each attribute to [0,1] or [-1,+1].
* For C-SVC, consider using the model selection tool in the tools directory.
* nu in nu-SVC/one-class-SVM/nu-SVR approximates the fraction of training
  errors and support vectors.
* If data for classification are unbalanced (e.g. many positive and
  few negative), try different penalty parameters C by -wi (see
  examples below).
* Specify larger cache size (i.e., larger -m) for huge problems.

Examples
========

> svm-scale -l -1 -u 1 -s range train > train.scale
> svm-scale -r range test > test.scale

Scale each feature of the training data to be in [-1,1]. Scaling
factors are stored in the file range and then used for scaling the
test data.

> svm-train -s 0 -c 5 -t 2 -g 0.5 -e 0.1 data_file

Train a classifier with RBF kernel exp(-0.5|u-v|^2), C=10, and
stopping tolerance 0.1.

> svm-train -s 3 -p 0.1 -t 0 data_file

Solve SVM regression with linear kernel u'v and epsilon=0.1
in the loss function.

> svm-train -c 10 -w1 1 -w-2 5 -w4 2 data_file

Train a classifier with penalty 10 = 1 * 10 for class 1, penalty 50 =
5 * 10 for class -2, and penalty 20 = 2 * 10 for class 4.

> svm-train -s 0 -c 100 -g 0.1 -v 5 data_file

Do five-fold cross validation for the classifier using
the parameters C = 100 and gamma = 0.1

> svm-train -s 0 -b 1 data_file
> svm-predict -b 1 test_file data_file.model output_file

Obtain a model with probability information and predict test data with
probability estimates

Precomputed Kernels
===================

Users may precompute kernel values and input them as training and
testing files.  Then libsvm does not need the original
```

```
training/testing sets.

Assume there are L training instances x1, ..., xL and.
Let K(x, y) be the kernel
value of two instances x and y. The input formats
are:

New training instance for xi:

<label> 0:i 1:K(xi,x1) ... L:K(xi,xL)

New testing instance for any x:

<label> 0:? 1:K(x,x1) ... L:K(x,xL)

That is, in the training file the first column must be the "ID" of
xi. In testing, ? can be any value.

All kernel values including ZEROs must be explicitly provided.  Any
permutation or random subsets of the training/testing files are also
valid (see examples below).

Note: the format is slightly different from the precomputed kernel
package released in libsvmtools earlier.

Examples:

        Assume the original training data has three four-feature
        instances and testing data has one instance:

        15  1:1 2:1 3:1 4:1
        45      2:3     4:3
        25          3:1

        15  1:1     3:1

        If the linear kernel is used, we have the following new
        training/testing sets:

        15  0:1 1:4 2:6  3:1
        45  0:2 1:6 2:18 3:0
        25  0:3 1:1 2:0  3:1

        15  0:? 1:2 2:0  3:1

        ? can be any value.

        Any subset of the above training file is also valid. For example,

        25  0:3 1:1 2:0  3:1
        45  0:2 1:6 2:18 3:0

        implies that the kernel matrix is

                [K(2,2) K(2,3)] = [18 0]
                [K(3,2) K(3,3)] = [0  1]

Library Usage
=============

These functions and structures are declared in the header file
`svm.h'.  You need to #include "svm.h" in your C/C++ source files and
link your program with `svm.cpp'. You can see `svm-train.c' and
`svm-predict.c' for examples showing how to use them. We define
LIBSVM_VERSION and declare `extern int libsvm_version; ' in svm.h, so
you can check the version number.

Before you classify test data, you need to construct an SVM model
(`svm_model') using training data. A model can also be saved in
a file for later use. Once an SVM model is available, you can use it
to classify new data.

- Function: struct svm_model *svm_train(const struct svm_problem *prob,
                                        const struct svm_parameter *param);
```

```
This function constructs and returns an SVM model according to
the given training data and parameters.

struct svm_problem describes the problem:

    struct svm_problem
    {
            int l;
            double *y;
            struct svm_node **x;
    };

where `l' is the number of training data, and `y' is an array containing
their target values. (integers in classification, real numbers in
regression) `x' is an array of pointers, each of which points to a sparse
representation (array of svm_node) of one training vector.

For example, if we have the following training data:

LABEL    ATTR1    ATTR2    ATTR3    ATTR4    ATTR5
-----    -----    -----    -----    -----    -----
  1        0        0.1      0.2      0        0
  2        0        0.1      0.3     -1.2      0
  1        0.4      0        0        0        0
  2        0        0.1      0        1.4      0.5
  3       -0.1     -0.2      0.1      1.1      0.1

then the components of svm_problem are:

l = 5

y -> 1 2 1 2 3

x -> [ ] -> (2,0.1) (3,0.2) (-1,?)
     [ ] -> (2,0.1) (3,0.3) (4,-1.2) (-1,?)
     [ ] -> (1,0.4) (-1,?)
     [ ] -> (2,0.1) (4,1.4) (5,0.5) (-1,?)
     [ ] -> (1,-0.1) (2,-0.2) (3,0.1) (4,1.1) (5,0.1) (-1,?)

where (index,value) is stored in the structure `svm_node':

    struct svm_node
    {
            int index;
            double value;
    };

index = -1 indicates the end of one vector. Note that indices must
be in ASCENDING order.

struct svm_parameter describes the parameters of an SVM model:

    struct svm_parameter
    {
            int svm_type;
            int kernel_type;
            int degree;     /* for poly */
            double gamma;   /* for poly/rbf/sigmoid */
            double coef0;   /* for poly/sigmoid */

            /* these are for training only */
            double cache_size; /* in MB */
            double eps;     /* stopping criteria */
            double C;       /* for C_SVC, EPSILON_SVR, and NU_SVR */
            int nr_weight;          /* for C_SVC */
            int *weight_label;      /* for C_SVC */
            double* weight;         /* for C_SVC */
            double nu;      /* for NU_SVC, ONE_CLASS, and NU_SVR */
            double p;       /* for EPSILON_SVR */
            int shrinking;  /* use the shrinking heuristics */
            int probability; /* do probability estimates */
    };

svm_type can be one of C_SVC, NU_SVC, ONE_CLASS, EPSILON_SVR, NU_SVR.
```

```
C_SVC:              C-SVM classification
NU_SVC:             nu-SVM classification
ONE_CLASS:          one-class-SVM
EPSILON_SVR:        epsilon-SVM regression
NU_SVR:             nu-SVM regression

kernel_type can be one of LINEAR, POLY, RBF, SIGMOID.

LINEAR:     u'*v
POLY:       (gamma*u'*v + coef0)^degree
RBF:        exp(-gamma*|u-v|^2)
SIGMOID:    tanh(gamma*u'*v + coef0)
PRECOMPUTED: kernel values in training_set_file

cache_size is the size of the kernel cache, specified in megabytes.
C is the cost of constraints violation.
eps is the stopping criterion. (we usually use 0.00001 in nu-SVC,
0.001 in others). nu is the parameter in nu-SVM, nu-SVR, and
one-class-SVM. p is the epsilon in epsilon-insensitive loss function
of epsilon-SVM regression. shrinking = 1 means shrinking is conducted;
= 0 otherwise. probability = 1 means model with probability
information is obtained; = 0 otherwise.

nr_weight, weight_label, and weight are used to change the penalty
for some classes (If the weight for a class is not changed, it is
set to 1). This is useful for training classifier using unbalanced
input data or with asymmetric misclassification cost.

nr_weight is the number of elements in the array weight_label and
weight. Each weight[i] corresponds to weight_label[i], meaning that
the penalty of class weight_label[i] is scaled by a factor of weight[i].

If you do not want to change penalty for any of the classes,
just set nr_weight to 0.

*NOTE* Because svm_model contains pointers to svm_problem, you can
not free the memory used by svm_problem if you are still using the
svm_model produced by svm_train().

*NOTE* To avoid wrong parameters, svm_check_parameter() should be
called before svm_train().

struct svm_model stores the model obtained from the training procedure.
It is not recommended to directly access entries in this structure.
Programmers should use the interface functions to get the values.

    struct svm_model
    {
            struct svm_parameter param;     /* parameter */
            int nr_class;           /* number of classes, = 2 in regression/one class svm */
            int l;                  /* total #SV */
            struct svm_node **SV;           /* SVs (SV[l]) */
            double **sv_coef;       /* coefficients for SVs in decision functions (sv_coef[k-1][l]) */
            double *rho;            /* constants in decision functions (rho[k*(k-1)/2]) */
            double *probA;          /* pairwise probability information */
            double *probB;
            int *sv_indices;        /* sv_indices[0,...,nSV-1] are values in [1,...,num_traning_data]
to indicate SVs in the training set */

            /* for classification only */

            int *label;             /* label of each class (label[k]) */
            int *nSV;               /* number of SVs for each class (nSV[k]) */
                                    /* nSV[0] + nSV[1] + ... + nSV[k-1] = l */
            /* XXX */
            int free_sv;            /* 1 if svm_model is created by svm_load_model*/
                                    /* 0 if svm_model is created by svm_train */
    };

param describes the parameters used to obtain the model.

nr_class is the number of classes. It is 2 for regression and one-class SVM.

l is the number of support vectors. SV and sv_coef are support
vectors and the corresponding coefficients, respectively. Assume there are
```

k classes. For data in class j, the corresponding sv_coef includes (k-1) y*alpha vectors,
where alpha's are solutions of the following two class problems:
1 vs j, 2 vs j, ..., j-1 vs j, j vs j+1, j vs j+2, ..., j vs k
and y=1 for the first j-1 vectors, while y=-1 for the remaining k-j
vectors. For example, if there are 4 classes, sv_coef and SV are like:

```
+-+-+-+--------------------+
|1|1|1|                    |
|v|v|v|  SVs from class 1  |
|2|3|4|                    |
+-+-+-+--------------------+
|1|2|2|                    |
|v|v|v|  SVs from class 2  |
|2|3|4|                    |
+-+-+-+--------------------+
|1|2|3|                    |
|v|v|v|  SVs from class 3  |
|3|3|4|                    |
+-+-+-+--------------------+
|1|2|3|                    |
|v|v|v|  SVs from class 4  |
|4|4|4|                    |
+-+-+-+--------------------+
```

See svm_train() for an example of assigning values to sv_coef.

rho is the bias term (-b). probA and probB are parameters used in
probability outputs. If there are k classes, there are k*(k-1)/2
binary problems as well as rho, probA, and probB values. They are
aligned in the order of binary problems:
1 vs 2, 1 vs 3, ..., 1 vs k, 2 vs 3, ..., 2 vs k, ..., k-1 vs k.

sv_indices[0,...,nSV-1] are values in [1,...,num_traning_data] to
indicate support vectors in the training set.

label contains labels in the training data.

nSV is the number of support vectors in each class.

free_sv is a flag used to determine whether the space of SV should
be released in free_model_content(struct svm_model*) and
free_and_destroy_model(struct svm_model**). If the model is
generated by svm_train(), then SV points to data in svm_problem
and should not be removed. For example, free_sv is 0 if svm_model
is created by svm_train, but is 1 if created by svm_load_model.

- Function: double svm_predict(const struct svm_model *model,
                               const struct svm_node *x);

This function does classification or regression on a test vector x
given a model.

For a classification model, the predicted class for x is returned.
For a regression model, the function value of x calculated using
the model is returned. For an one-class model, +1 or -1 is
returned.

- Function: void svm_cross_validation(const struct svm_problem *prob,
        const struct svm_parameter *param, int nr_fold, double *target);

This function conducts cross validation. Data are separated to
nr_fold folds. Under given parameters, sequentially each fold is
validated using the model from training the remaining. Predicted
labels (of all prob's instances) in the validation process are
stored in the array called target.

The format of svm_prob is same as that for svm_train().

- Function: int svm_get_svm_type(const struct svm_model *model);

This function gives svm_type of the model. Possible values of
svm_type are defined in svm.h.

- Function: int svm_get_nr_class(const svm_model *model);

```
    For a classification model, this function gives the number of
    classes. For a regression or an one-class model, 2 is returned.

- Function: void svm_get_labels(const svm_model *model, int* label)

    For a classification model, this function outputs the name of
    labels into an array called label. For regression and one-class
    models, label is unchanged.

- Function: void svm_get_sv_indices(const struct svm_model *model, int *sv_indices)

    This function outputs indices of support vectors into an array called sv_indices.
    The size of sv_indices is the number of support vectors and can be obtained by calling svm_get_nr_sv.
    Each sv_indices[i] is in the range of [1, ..., num_traning_data].

- Function: int svm_get_nr_sv(const struct svm_model *model)

    This function gives the number of total support vector.

- Function: double svm_get_svr_probability(const struct svm_model *model);

    For a regression model with probability information, this function
    outputs a value sigma > 0. For test data, we consider the
    probability model: target value = predicted value + z, z: Laplace
    distribution e^(-|z|/sigma)/(2sigma)

    If the model is not for svr or does not contain required
    information, 0 is returned.

- Function: double svm_predict_values(const svm_model *model,
                                      const svm_node *x, double* dec_values)

    This function gives decision values on a test vector x given a
    model, and return the predicted label (classification) or
    the function value (regression).

    For a classification model with nr_class classes, this function
    gives nr_class*(nr_class-1)/2 decision values in the array
    dec_values, where nr_class can be obtained from the function
    svm_get_nr_class. The order is label[0] vs. label[1], ...,
    label[0] vs. label[nr_class-1], label[1] vs. label[2], ...,
    label[nr_class-2] vs. label[nr_class-1], where label can be
    obtained from the function svm_get_labels. The returned value is
    the predicted class for x. Note that when nr_class = 1, this
    function does not give any decision value.

    For a regression model, dec_values[0] and the returned value are
    both the function value of x calculated using the model. For a
    one-class model, dec_values[0] is the decision value of x, while
    the returned value is +1/-1.

- Function: double svm_predict_probability(const struct svm_model *model,
            const struct svm_node *x, double* prob_estimates);

    This function does classification or regression on a test vector x
    given a model with probability information.

    For a classification model with probability information, this
    function gives nr_class probability estimates in the array
    prob_estimates. nr_class can be obtained from the function
    svm_get_nr_class. The class with the highest probability is
    returned. For regression/one-class SVM, the array prob_estimates
    is unchanged and the returned value is the same as that of
    svm_predict.

- Function: const char *svm_check_parameter(const struct svm_problem *prob,
                                            const struct svm_parameter *param);

    This function checks whether the parameters are within the feasible
    range of the problem. This function should be called before calling
    svm_train() and svm_cross_validation(). It returns NULL if the
    parameters are feasible, otherwise an error message is returned.

- Function: int svm_check_probability_model(const struct svm_model *model);
```

```
    This function checks whether the model contains required
    information to do probability estimates. If so, it returns
    +1. Otherwise, 0 is returned. This function should be called
    before calling svm_get_svr_probability and
    svm_predict_probability.

- Function: int svm_save_model(const char *model_file_name,
                               const struct svm_model *model);

    This function saves a model to a file; returns 0 on success, or -1
    if an error occurs.

- Function: struct svm_model *svm_load_model(const char *model_file_name);

    This function returns a pointer to the model read from the file,
    or a null pointer if the model could not be loaded.

- Function: void svm_free_model_content(struct svm_model *model_ptr);

    This function frees the memory used by the entries in a model structure.

- Function: void svm_free_and_destroy_model(struct svm_model **model_ptr_ptr);

    This function frees the memory used by a model and destroys the model
    structure. It is equivalent to svm_destroy_model, which
    is deprecated after version 3.0.

- Function: void svm_destroy_param(struct svm_parameter *param);

    This function frees the memory used by a parameter set.

- Function: void svm_set_print_string_function(void (*print_func)(const char *));

    Users can specify their output format by a function. Use
        svm_set_print_string_function(NULL);
    for default printing to stdout.

Java Version
============

The pre-compiled java class archive `libsvm.jar' and its source files are
in the java directory. To run the programs, use

java -classpath libsvm.jar svm_train <arguments>
java -classpath libsvm.jar svm_predict <arguments>
java -classpath libsvm.jar svm_toy
java -classpath libsvm.jar svm_scale <arguments>

Note that you need Java 1.5 (5.0) or above to run it.

You may need to add Java runtime library (like classes.zip) to the classpath.
You may need to increase maximum Java heap size.

Library usages are similar to the C version. These functions are available:

public class svm {
        public static final int LIBSVM_VERSION=321;
        public static svm_model svm_train(svm_problem prob, svm_parameter param);
        public static void svm_cross_validation(svm_problem prob, svm_parameter param, int nr_fold,
double[] target);
        public static int svm_get_svm_type(svm_model model);
        public static int svm_get_nr_class(svm_model model);
        public static void svm_get_labels(svm_model model, int[] label);
        public static void svm_get_sv_indices(svm_model model, int[] indices);
        public static int svm_get_nr_sv(svm_model model);
        public static double svm_get_svr_probability(svm_model model);
        public static double svm_predict_values(svm_model model, svm_node[] x, double[] dec_values);
        public static double svm_predict(svm_model model, svm_node[] x);
        public static double svm_predict_probability(svm_model model, svm_node[] x, double[]
prob_estimates);
        public static void svm_save_model(String model_file_name, svm_model model) throws IOException
        public static svm_model svm_load_model(String model_file_name) throws IOException
        public static String svm_check_parameter(svm_problem prob, svm_parameter param);
        public static int svm_check_probability_model(svm_model model);
        public static void svm_set_print_string_function(svm_print_interface print_func);
```

```
}

The library is in the "libsvm" package.
Note that in Java version, svm_node[] is not ended with a node whose index = -1.

Users can specify their output format by

        your_print_func = new svm_print_interface()
        {
                public void print(String s)
                {
                        // your own format
                }
        };
        svm.svm_set_print_string_function(your_print_func);

Building Windows Binaries
=========================

Windows binaries are available in the directory `windows'. To re-build
them via Visual C++, use the following steps:

1. Open a DOS command box (or Visual Studio Command Prompt) and change
to libsvm directory. If environment variables of VC++ have not been
set, type

""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\amd64\vcvars64.bat""

You may have to modify the above command according which version of
VC++ or where it is installed.

2. Type

nmake -f Makefile.win clean all

3. (optional) To build shared library libsvm.dll, type

nmake -f Makefile.win lib

4. (optional) To build 32-bit windows binaries, you must
        (1) Setup "C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\vcvars32.bat" instead of
vcvars64.bat
        (2) Change CFLAGS in Makefile.win: /D _WIN64 to /D _WIN32

Another way is to build them from Visual C++ environment. See details
in libsvm FAQ.

- Additional Tools: Sub-sampling, Parameter Selection, Format checking, etc.
============================================================================

See the README file in the tools directory.

MATLAB/OCTAVE Interface
=======================

Please check the file README in the directory `matlab'.

Python Interface
================

See the README file in python directory.

Additional Information
======================

If you find LIBSVM helpful, please cite it as

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

LIBSVM implementation document is available at
http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf
```

```
For any questions and comments, please email cjlin@csie.ntu.edu.tw

Acknowledgments:
This work was supported in part by the National Science
Council of Taiwan via the grant NSC 89-2213-E-002-013.
The authors thank their group members and users
for many helpful discussions and comments. They are listed in
http://www.csie.ntu.edu.tw/~cjlin/libsvm/acknowledgements
```