

---

# MLSVM – Multi level Support Vector Machines

## User Guide

---

*Ehsan Sadrifaridpour (esadrfa@clemson.edu)*  
*Talayeh Razzaghi, Ilya Safro*

<https://github.com/esadr/mlsvm>

### Introduction

A Multilevel Support Vector Machines (SVM) is a fast scalable framework for SVM. This document is developed to provide a guide to users for installation, parameters, and use cases. The details of the framework are available at [1, 2].

### Contents

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Overall procedure</b>	<b>3</b>
<b>3</b>	<b>Input File Formats</b>	<b>3</b>
<b>4</b>	<b>Tools</b>	<b>3</b>
4.1	LibSVM file to PETSc Binary . . . . .	3
4.2	CSV file to PETSc Binary . . . . .	4
4.3	Normalize the data using z-score . . . . .	4
4.4	Calculate the k-Nearest Neighbors . . . . .	4
4.5	View PETSc binary file . . . . .	4
<b>5</b>	<b>Parameters</b>	<b>4</b>
<b>6</b>	<b>Outputs</b>	<b>4</b>
<b>7</b>	<b>Log Levels</b>	<b>6</b>
<b>8</b>	<b>Prediction</b>	<b>6</b>



## 1 Installation

The required packages which needs to be installed are PETSc [3], Metis [4], LibSVM [5], FLANN [6], and pyFLANN.

## 2 Overall procedure

Suppose you installed the framework successfully, you need to go through below steps to prepare a training data set, train a model(s), and predict the test data.

1. Save the data in PETSc binary format (Section 3).
2. For raw data, normalize the data using z-score function. In case the data were normalized in advance, please ignore this step. (Section 4.3).
3. Calculate the k-Nearest Neighbors (Section 4.4).
4. Set the parameters and run the MLSVM classifier (Section 5).
5. Use prediction tools to predict the test data (Section 8).

## 3 Input File Formats

The default data for the MLSVM framework is saved in a PETSc binary format. Using `mlsvm_libsvm_petsc`, you can convert the LibSVM data file format to PETSc Binary format (Section 4.1).

Using `mlsvm_csv_petsc` you can convert the CSV format. The label should be in the first column in the left and the data is filled in columns afterwards. The lines with “#” symbol in the beginning are considered as comments (Section 4.2).

## 4 Tools

There are steps such as converting, normalizing and calculating the k-Nearest Neighbors which are only used once for a data set. The results are saved in files and use for training the models. The details for each tool are explained in the following sections.

### 4.1 LibSVM file to PETSc Binary

The “`mlsvm_libsvm_petsc`” can be compiled using the `make`. The parameters “`-ds_p`” and “`-f`” are used to set the path and input file name respectively. The output includes two files. One for data and one for labels. The files will be saved at the same path using “`_data.dat`” extension for data file and “`_label.dat`” for the labels. An example is provided in the *docs/usecases* folder.

## 4.2 CSV file to PETSc Binary

It has the same parameters and outputs as “mlsvm\_libsvm\_petsc”. The input is CSV file format and the first column is the labels. The labels should be +1 or -1. The input name should not include the “.csv” extension.

## 4.3 Normalize the data using z-score

It normalizes the data using z-score. The whole data with file name such as X\_data.dat will pass as “-f X”. The “mlsvm\_zscore” stores the output as a matrix in file X\_zsc\_data.dat.

## 4.4 Calculate the k-Nearest Neighbors

We use the FLANN library [6]. You need to install the pyFlann and NumPy and SciPy packages as well. One convenient way is to use Anaconda library and use the “pip install pyflann”. The PY\_PATH environment variable is required. For using Anaconda, just set the PY\_PATH to “anaconda/bin/”

## 4.5 View PETSc binary file

The “petsc\_utility” folder includes both get\_mat\_info and get\_vec\_info which can be used to see the contents of a matrix and vector in PETSc binary format respectively.

First, go inside the petsc\_utility folder. Then “make get\_mat\_info” and “make get\_vec\_info”. The data file is saved as a matrix and the label file is saved as a vector. The output of these tools will be as large as the content of the file which probably clutter your terminal. Hence, try to export the standard output to text files or using less or more commands to handle the output. The parameter to specify the input file is “-i” instead of “-f” and you need to pass the full path along with the file name completely. You can find a use case in docs.

# 5 Parameters

The parameters are explained in the params.xml file. The main parameters are explained in Table 1

# 6 Outputs

There are two set of outputs.

- One is the standard text results of the information regarding the results of trainings, validations and predictions. The parameters which are used, size of data at each level, The type and volume of logs can be customized. The details are explained in section 7.

Table 1: List of Parameters

--nn_n	Set the number of nearest neighbors
--nn_d	Set the distance type for nearest neighbors
-s	Set the random seed. The random seed will increase at the beginning of each experiment inside your run.
-x	Set the number of experiment which includes the whole k-fold cross validation
-k	Set the number of folds in the k-fold cross validation. The default is 5 or 10.
--ds_p	Set the path to data set files
-f, --ds_f	Set the name of data set. Remember, this is the name before _data.dat or _zsc_data.dat
--tmp_p	Set the path to temporary folder which is used to store the test data files
--cs_pi	Set the maximum number of features in the training data. The default is 300. Please set this value to the maximum number of features you have in the training data, otherwise the performance degrades.
-t, --cs_t	Set the threshold for the coarsening to stop. This value is for each class not both classes together.
-q, --cs_q	Set the threshold for adding fine points as seed to increase the number of seeds. The default value is 0.4
-r, --cs_r	Set the number of fractions which are allows to participate in each aggregates in the coarsening.
--cs_m	Set the maximum level of coarsening to stop it from infinite loops. It will be deprecated in future versions.
--cs_we	Set the threshold for filtering the weak edges.
-v	Set the fraction of training data to be used as the validation data. The range is between [0,1] and the default is 0.1
--rf_2nd	The 1 adds the distant 2 neighbors during the refinement. The default value is 0.
--pr_start	Set a threshold for starting the partitioning when the number of points reaches to this number in both classes during the refinement
--pr_max	Set the maximum number of data points in each partition for each class
--mv_id	Set the type of majority voting in case of partitioning (partitioning produces multiple models).

- The trained models and the summary of the models in files are saved in the `svm_models` folder. The summary file includes the information for the number of experiments and the number of k-folds in the first line. The rest of the summary file maintain the information for the best level at a specific fold of an experiment and the number of models. The number of models are 1 normally unless the partitioning has happened which has produced multiple models. These models would be used for prediction of unseen data points in future.

## 7 Log Levels

All the steps of the framework can generate a spectrum of logs from almost nothing to an extensive logs. There are macros in the “`config_logs.h`” which controls the volume of logs for the main parts of the framework. The default values are suggested as comments in the file. For some data set, more details in the log might be required which can easy set in the file. A clean and complete compile of the code is required after changing the log levels.

For large logs, exporting the standard output to a file would help to reduce the difficulty to search and analyze the logs later. We cover required commands in section 9.

## 8 Prediction

The prediction happens in the end of each level on both validation and test data. The results printed in the standard output. A stand-alone program to predict just one model has not developed yet! Please email me if you are interested to have a stand alone program for prediction.

## 9 Use Cases

In the docs folder inside the GitHub repository, you can find the sample run commands for different parts.

## References

- [1] E. Sadrifaridpour, T. Razzaghi, and I. Safro. Engineering multilevel support vector machines. *ArXiv e-prints*, July 2017.
- [2] E. Sadrifaridpour, S. Jeerreddy, K. Kennedy, A. Luckow, T. Razzaghi, and I. Safro. Algebraic multigrid support vector machines. *ArXiv e-prints*, November 2016.
- [3] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc Web page. <http://www.mcs.anl.gov/petsc>, 2016.
- [4] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
- [5] 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>, 2011.
- [6] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09*, pages 331–340. INSTICC Press, 2009.