Using MATLAB and the LS-SVMlab Toolbox

Alex Lambert[*] Sonny Achten[†] Konstantinos Kontras[‡]
Francesco Tonin[§] Yingyi Chen[¶] Johan Suykens

Academic Year 2022-2023

The exercises for the course Support Vector Machines: Methods and Applications (`H02D3a` or `H00H3a`) will be solved using the MATLAB environment. For additional general information on how to use this environment, see the website `https://www.mathworks.com/`. The documentation is available under the link `support` → `documentation`.

1 MATLAB: An absolute crash course

- Example commands for the MATLAB prompt are notated as

```
>> ...
```

  while comments are given as

```
>> % ...
```

- For additional help on `command`, type

```
>> help command
```

- To clear all variables from the workspace, use

```
>> clear
```

- MATLAB is a high-level development environment, it is not required to bother about declarations, memory assignment and the like.
- One can pass the orders to MATLAB via the command prompt. An alternative is to execute a series of commands in batch mode. For the latter, type

```
>> edit script
```

  An editor will open with the file `script.m`. After saving the file, one can execute the script:

[*]alex.lambert@esat.kuleuven.be
[†]sonny.achten@esat.kuleuven.be
[‡]konstantinos.kontras@esat.kuleuven.be
[§]francesco.tonin@esat.kuleuven.be
[¶]yingyi.chen@esat.kuleuven.be

```
>> script
```

- MATLAB has quite extensive possibilities for visualization. Figures are made in a new figure-window, which can be created by

  ```
  >> H = figure;
  ```

  and can be referred to by its handle `H`. To close that window, type

  ```
  >> close(H)
  ```

  There is always a current window, which is the one last used. To close the current window, type

  ```
  >> close
  ```

  To close all windows, type

  ```
  >> close all
  ```

  Information on different plotting commands can be found by the help of `plot`, `plot3`, `line`, `surf`, ...

- To install the SVM and LS-SVMlab toolboxes go to Toledo website → SVM Exercises Course → Course Documents. Download the toolboxes and unzip them in your personal account or hard disk, e.g. in the directories `C:/SVMcourse/SVM` and `C:/SVMcourse/LS-SVMlab`. Then set your MATLAB path to:

  ```
  >> addpath('C:/SVMcourse/SVM');
  >> addpath('C:/SVMcourse/LS-SVMlab');
  ```

2 Using LS-SVMlab

2.1 Classification

- *Loading the data.* Before using the LS-SVMlab toolbox, we have to load our data into MATLAB. When loaded, multiple variables appear in the MATLAB workspace:

  – `Xtrain` - data points of the *training set* (matrix of size $n \times d$),
  – `Ytrain` - labels corresponding to the data points of the training set (matrix of size $n \times 1$),
  – `Xtest`: data points of the *test set* (matrix of size $m \times d$),
  – `Ytest`: labels corresponding to the data points of the test set (matrix of size $m \times 1$).

  where n is the number of data points in the training set, m is the number of data points in the test set and d is the dimensionality of the data points. Always make sure that your data is split up into a training set and a test set, and make sure that none of the test set data is involved in the training procedure.

- *Training the model.* To train a classification model in LS-SVMlab, use the following syntax:

Table 1: Classification using LS-SVMlab: parameters of the training procedure.

Parameter	Explanation	Values
Xtrain	Data points of the training set	
Ytrain	Labels of the training data points	
gam	Regularization constant	
kpar	Kernel parameter(s)	Parameters used in exercise session: • [] (linear, no parameters) • [t ; degree] (polynomial) • sig2 (RBF kernel)
'kernel'	Kernel type	Kernels used in exercise session: • 'lin_kernel' • 'poly_kernel' • 'RBF_kernel'

```
>> [alpha, b] = trainlssvm({Xtrain, Ytrain, 'c', gam, kpar,
    'kernel'});
```

where 'c' indicates that we are building a classification model. More information on how to use the syntax can be found in table 1.

- *Visualization of the model.* A plot of the LS-SVM classifier can be generated using:

```
>> plotlssvm({Xtrain, Ytrain, 'c', gam, kpar, 'kernel'}, {alpha,
    b});
```

  Note that the model first has to be trained (i.e., alpha and b have to be computed) before a plot can be generated. The plot shows the classification boundary and the training data points. Visualization is only possible for the two-dimensional case.

- *Classification using resulting model.* Classification using a (trained) LS-SVM classifier can be done using:

```
>> Yest = simlssvm({Xtrain, Ytrain, 'c', gam, kpar, 'kernel'}, {
    alpha, b}, Xtest);
```

  Note again that the model has to be trained first, before it can be simulated on 'new' data (the new data is here indicated by Xtest).

- *Automatic parameter tuning.* The tuning procedure of the hyperparameter γ and possible kernel parameters can be implemented in an automatic way using:

```
>> [gam,sig2,cost] = tunelssvm({Xtrain, Ytrain, 'c', [], [],
    'kernel'}, 'algorithm', 'crossvalidatelssvm',{10, 'misclass'});
```

  where 'algorithm' can be chosen as 'simplex' (Nelder-Mead method) or 'gridsearch' (brute force gridsearch).

2.2 Function estimation

- *Loading the data.* As for classification, the data is loaded in variables Xtrain, Ytrain, Xtest and Ytest. In the function estimation case, the Y variables contain function values (rather than class labels).

- *Training the model.* To train a function estimation model in LS-SVMlab, use the following syntax:

```
>> [alpha, b] = trainlssvm({Xtrain, Ytrain, 'f', gam, kpar,
    'kernel'});
```

where `'f'` indicates that we are building a function estimation model. The parameters are the same as for the classification case, and can be found in table 1.

- *Visualization of the model.* A plot of the LS-SVM function estimator can be generated using:

```
>> plotlssvm({Xtrain, Ytrain, 'f', gam, kpar, 'kernel'}, {alpha
    , b});
```

Note that the model first has to be trained (i.e., `alpha` and `b` have to be computed) before a plot can be generated. Visualization is only possible for the two-dimensional case. The plot shows the estimated function values for the training data points.

- *Function estimation using resulting model.* Function estimation using a (trained) LS-SVM function estimator can be done using:

```
>> Yest = simlssvm({Xtrain, Ytrain, 'f', gam, kpar, 'kernel'},
    {alpha, b}, Xtest);
```

Note again that the model has to be trained first, before it can be simulated on 'new' data (the new data is here indicated by `Xtest`).

- *Automatic parameter tuning.* The tuning procedure of the hyperparameter γ and possible kernel parameters can be implemented in an automatic way using:

```
>> [gam,sig2,cost] = tunelssvm({Xtrain, Ytrain, 'f', [], [],
    'kernel'}, 'algorithm', 'crossvalidatelssvm',{10, 'mse'});
```

where `'algorithm'` can be chosen as `'simplex'` (Nelder-Mead method) or `'gridsearch'` (brute force gridsearch).

2.3 Remark

It is important to note that by default the data is preprocessed internally for as well training as simulation. For additional information on the used preprocessing, type:

```
>> help prelssvm
```

This is important to remember as preprocessing directly affects the chosen hyperparameters and the performance on the test set.