Exercise Session 2: Function Estimation and Time Series Prediction

Dries Hendrikx Michaël Fanuel Marcin Orchel Johan Suykens

This document describes a step-by-step guide for the application of support vector machine methods. The goal of the exercise sessions is to provide you with experience and teach you how to tackle future application problems. The MATLAB scripts and toolboxes, the course documents, the referred papers and all the datasets used in the exercise sessions are available for academic purposes on `https://toledo.kuleuven.be`.

- The exam consists of an oral discussion on the basis of the reports written for the 3 exercise sessions. It is important to show that you have understanding about the problem and that you can work in a constructive manner towards getting good solutions to given problems.
- The reports have to be written *individually*.
- The reports contain the solutions to the exercises *and* the homework problems. All the questions that need to be answered in the report are indicated by the grey box, which reads 'Questions'.
- A good report finds a good balance between *visuals* on the one hand and to-the-point *explanations* on the other hand. Use figures and tables to explain things, rather than only long and elaborate sentences. Make sure to include a few key equations in the textual part.
- There is *no* page limit for the reports. Keep in mind however that the quality of the report is assessed, rather than the quantity.
- Write a *separate* report for every exercise session. At the end of the semester, you have to hand in 3 separate reports.

1 Exercises

1.1 Support vector machine for function estimation

In this exercise the support vector machine (SVM) toolbox is used for regression. You can find it on Toledo (`svm_toolbox_1.zip`). Download this toolbox and add it to your MATLAB path. Consequently, execute:

```
>> uiregress
```

Executing this script opens a GUI, which allows you to create datasets and do SVM regression on them. In order to add data points, click `Data`, which creates a cursor that can be used to define the locations of the data points. A left mouse click adds a data point, a right mouse click removes the data cursor. Remove the data cursor before clicking `Regress`, otherwise the GUI will add an additional data point under the `Regress` button.

Questions

- Construct a dataset where a linear kernel is better than any other kernel (around 20 data points). What is the influence of `e` (try small values such as $0.10, 0.25, 0.50, \ldots$) and of `Bound` (try larger increments such as $0.01, 0.10, 1, 10, 100$). Where does the sparsity property come in?
- Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.
- In what respect is SVM regression different from a classical least squares fit?

1.2 A simple example: the sinc function

We proceed with the least squares based variant of the support vector machine (LS-SVM), using the LS-SVMlab toolbox. More information on the use of the toolbox can be found in the general introduction on Toledo. Before we start experimenting with the toolbox ourselves, let's first have a look at a demo by typing

```
>> demofun
```

Go through this introductory example in order to gain more feeling with the syntax and the possibilities of the LS-SVMlab toolbox.

1.2.1 Regression of the sinc function

To get an intuitive idea what function estimation is about, the exercises start with a simple toy example. An artificial dataset is constructed from the sinc function (with white noise):

```
>> X = (-3:0.01:3)';
>> Y = sinc(X) + 0.1.*randn(length(X), 1);
```

Consequently, we create the training and the test sets:

```
>> Xtrain = X(1:2:end);
>> Ytrain = Y(1:2:end);
>> Xtest = X(2:2:end);
>> Ytest = Y(2:2:end);
```

In this exercise, we construct a LS-SVM regression model with the RBF kernel.

Questions

- Try out a range of different `gam` and `sig2` parameter values (e.g., `gam` $= 10, 10^3, 10^6$ and `sig2` $= 0.01, 1, 100$) and visualize the resulting function estimation on the test set data points. Discuss the resulting function estimation. Report the mean squared error for every combination (`gam`, `sig2`).
- Do you think there is one optimal pair of hyperparameters? Argument why (not).
- Tune the `gam` and `sig2` parameters using the `tunelssvm` procedure. Use multiple runs: what can you say about the hyperparameters and the results? Use both the `simplex` and `gridsearch` algorithms and report differences.

1.2.2 Application of the Bayesian framework

In addition to the approach outlined above, the Bayesian framework can also be used to tune and to analyze the LS-SVM regressor. The basic result from the Bayesian framework for the LS-SVM is the derivation of the probability that the data points are generated by the given model. This is called the posterior probability. This probability criterion is expressed as a number. There are 3 variants: the posterior with respect to the model parameters `alpha` and `b`, the posterior with respect to the regularization constant `gam` and the posterior with respect to the choice of the kernel and its parameter `sig2`. The cost (negative logarithm of the posteriors) is computed by the function call

```
>> sig2     = 0.4;
>> gam      = 10;
>> crit_L1 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 1);
>> crit_L2 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 2);
>> crit_L3 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

The model can be optimized with respect to these criteria:

```
>> [~,alpha,b] = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 1);
>> [~,gam]     = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 2);
>> [~,sig2]    = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

For regression, the error bars can be computed using Bayesian inference using

```
>> sig2e = bay_errorbar({Xtrain, Ytrain, 'f', gam, sig2}, 'figure');
```

Questions

- Discuss in a schematic way how parameter tuning works using the Bayesian framework. Illustrate this scheme by interpreting the function calls denoted above.

1.3 Automatic Relevance Determination

In addition to parameter tuning, the Bayesian framework can also be used to select the most relevant inputs by Automatic Relevance Determination (ARD). The following procedure uses this criterion for backward selection for a three dimensional input selection task, constructed as (use tuned `gam` and `sig2` parameter values):

```
>> X = 6.*rand(100, 3) - 3;
>> Y = sinc(X(:,1)) + 0.1.*randn(100,1);
>> [selected, ranking] = bay_lssvmARD({X, Y, 'f', gam, sig2});
```

Questions

- Visualize the results in a simple figure.
- How can you do input selection in a similar way using the `crossvalidate` function instead of the Bayesian framework?

1.4 Robust regression

In situations where the data is corrupted with non-Gaussian noise or outliers, it becomes important to incorporate robustness into the estimation. Consider the following simple example:

```
>> X = (-6:0.2:6)';
>> Y = sinc(X) + 0.1.*rand(size(X));
```

Outliers can be added via:

```
>> out = [15 17 19];
>> Y(out) = 0.7+0.3*rand(size(out));
>> out = [41 44 46];
>> Y(out) = 1.5+0.2*rand(size(out));
```

Let's say we first train a LS-SVM regressor model, without giving special attention to the outliers. We can implement this as:

```
>> model   = initlssvm(X, Y, 'f', [], [], 'RBF_kernel');
>> costFun = 'crossvalidatelssvm';
>> model   = tunelssvm(model, 'simplex', costFun, {10, 'mse';});
>> plotlssvm(model);
```

We explicitly write out the code here, since the object oriented notation is used.

If we were to train a robust LS-SVM model, using robust crossvalidation, we can implement this as:

```
>> model   = initlssvm(X, Y, 'f', [], [], 'RBF_kernel');
>> costFun = 'rcrossvalidatelssvm';
>> wFun    = 'whuber';
>> model   = tunelssvm(model, 'simplex', costFun, {10, 'mae';}, wFun);
>> model   = robustlssvm(model);
>> plotlssvm(model);
```

Train and plot a LS-SVM regression model, without giving special attention to the outliers: tune the parameters (using `tunelssvm`), train the regressor model (using `trainlssvm`) and plot the result (using `plotlssvm`). What is the influence of the outlying points?

Questions

- Visualize and discuss the results. Compare the non-robust version with the robust version. Do you spot any differences?

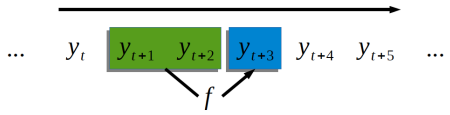

Figure 1: Schematic representation of the nonlinear auto-regressive model.

- Why in this case is the mean absolute error (`'mae'`) preferred over the classical mean squared error (`'mse'`)?
- Try alternatives to the weighting function `wFun` (e.g., `'whampel'`, `'wlogistic'` and `'wmyriad'`. Report on differences. Check the user's guide of LS-SVMlab for more information.

2 Homework problems

2.1 Introduction: time series prediction

The linear autoregressive (AR) model of a process Z_t with $t = 1, 2, \ldots$ is given by

$$\hat{z}_t = a_1 z_{t-1} + a_2 z_{t-2} + \cdots + a_n z_{t-n} \tag{1}$$

with $a_i \in \mathbb{R}$ for $i = 1, \ldots, n$ and n the model order. At the same time, the nonlinear variant (NAR) is desscribed as:

$$\hat{z}_t = f(z_{t-1}, z_{t-2}, \ldots, z_{t-n}), \tag{2}$$

A depiction of these processes can be found in figure 1. Time series identification can be written as a classical black-box regression modeling problem:

$$\hat{y}_t = f(x_t) \tag{3}$$

with $y_t = z_t$ and $x_t = [z_{t-1}, z_{t-2}, \ldots, z_{t-n}]^\top$. In a nutshell, any provided sequence Z_t can be mapped into a regression problem.

2.2 Logmap dataset

Let's now apply time series prediction on the `logmap` dataset. As always, we first have to load the data:

```
>> load logmap.mat
```

Two variables are loaded into the workspace: `Z` (training data) and `Ztest` (test data). First, we have to map our sequence `Z` into a regression problem. This can be done using the command `windowize`:

```
>> order = 10;
>> X = windowize(Z, 1:(order + 1));
>> Y = X(:, end);
>> X = X(:, 1:order);
```

Now, a model can be built using these data points:

```
>> gam = 10;
>> sig2 = 10;
>> [alpha, b] = trainlssvm({X, Y, 'f', gam, sig2});
```

It is straightforward to predict the next data points using the `predict` function of the LS-SVMlab toolbox. In order to call the function, we first have to define the starting point of the prediction:

```
>> Xs = Z(end-order+1:end, 1);
```

Naturally, this is the last point of the training set. The test set `Ztest` presents data points after this point, which we will try to predict. This can be implemented as follows:

```
>> nb = 50;
>> prediction = predict({X, Y, 'f', gam, sig2}, Xs, nb);
```

where `nb` indicates how many time points we want to predict. Here, we define this number equal to the number of data points in the test set. Finally, the performance of the predictor can be checked visually:

```
>> figure;
>> hold on;
>> plot(Ztest, 'k');
>> plot(prediction, 'r');
>> hold off;
```

In this figure, the data points of the test set, that is, the actual data points that we want to predict are depicted in black, while the prediction is presented in red. Try to assess the performance of the prediction: do you think it's good? Is there still some room for improvements? As indicated numerous times before, the parameters `gam` and `sig2` can be optimized using crossvalidation. In the same way, one can optimize `order` as a parameter.

Questions

- As indicated numerous times before, the parameters `gam` and `sig2` can be optimized using crossvalidation. In the same way, one can optimize `order` as a parameter. Define a strategy to tune these 3 parameters.
- Do time series prediction using the optimized parameter settings. Visualize your results. Discuss.

2.3 Santa Fe dataset

In this exercise, we apply time series prediction on the Santa Fe laser dataset.

Questions

- Does `order` $= 50$ for the utilized auto-reressive model sounds like a good choice?
- Would it be sensible to use the performance of this recurrent prediction on the validation set to optimize hyperparameters and the model order?
- Tune the parameters (`order`, `gam` and `sig2`) and do time series prediction. Visualize your results. Discuss.