NPRLab - A crash course to the basic usage

Pieter Jan Kerstens

January 1, 2015

Abstract

This manual provides a quick guide to using the NPRLab toolbox. It guides the user to all the basic steps in order to preprocess, estimate and plot nonparametric kernel estimators with NPRLab.

1 Introduction

NPRLab - NonParametric Regression Laboratory for MATLAB - is a toolbox to quickly run nonparametric regressions.[1] It is a spin-off of code written during my master thesis in mathematical engineering at KU Leuven in 2012. NPRLab is devised as a toolbox built around LS-SVMlab and as such it works very similar to LS-SVMlab.[2] In fact, users familiar with LS-SVMlab might only need a quick glance at this manual to get started. For all others this quickguide should walk you through the basic steps to get started.

Throughout, we assume the following data generating process:

$$Y_i = m(\mathbf{X}_i) + \epsilon_i, \tag{1}$$

$\forall i = 1, \ldots, n$ with $\mathbf{X}_i \in \mathbb{R}^d$, $Y_i \in \mathbb{R}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Note that we assume homoscedastic noise. NPRLab implements a class of "linear smoothers". An estimator $\hat{m}_n(X)$ is a linear smoother if, for each X there exists a vector $\mathbf{l(x)} = (l_1(X), \ldots, l_n(X))^T$ such that

$$\hat{m}_n(X) = \sum_{i=1}^{n} l_i(X) Y_i. \tag{2}$$

Denote the fitted values by $\hat{\mathbf{Y}} = (\hat{m}_n(X_1), \ldots, \hat{m}_n(X_n))^T$ and $\mathbf{Y} = (Y_1, \ldots, Y_n)^T$. We can then write the above as

$$\hat{\mathbf{Y}} = \mathbf{L}\mathbf{Y},$$

where $\mathbf{L}$ is a $n \times n$ smoother matrix with $L_{ij} = l_j(X_i)$. The form and construction of the smoother matrix depends on the specific nonparametric estimator. The class of linear smoothers is large and includes Nadaraya-Watson, Priestley-Chao, local polynomial regression, LS-SVM, penalized spline estimators, etc. Some of these are implemented in the toolbox, but others can easily be added by the user.

[1] NPRLab is freely available at http://kerstens.hopto.org/nprlab/ under the GNU GPL v3.

[2] LS-SVMlab is freely available at http://www.esat.kuleuven.be/sista/lssvmlab/.

2 Data preprocessing and estimation

We assume that a dataset is loaded into MATLAB containing a $n \times d$ matrix $\mathbf{X}$ and a $n \times 1$ vector $\mathbf{Y}$. Assuming the data needs no preprocessing, we initiate and train the model writing:

```
% Initialize and train a Nadaraya-Watson model
model = initnpr(X,Y,'nadaraya watson','original');
```

The fitted model is returned in a structure called `model`. In the above example, we have fitted a Nadaraya-Watson model. This can be changed by replacing the string `'nadaraya watson'` with one found in Table 1. In fact, the toolbox determines the selected smoother by comparing only the first two letters of the string. This is indicated by the third column in Table 1.

smoother name	string	shorthand
Nadaraya-Watson	`'nadaraya watson'`	`'na'`
Priestley-Chao	`'priestley-chao'`	`'pr'`
Local polynomial regression	`'local polynomial'`	`'lo'`
LS-SVM	`'ls-svm'`	`'ls'`

Table 1: Available linear smoothers

In some cases, better estimation results are obtained by normalizing the data: $\mathbf{X}$ and $\mathbf{Y}$ are transformed to have zero mean and unit variance. This can be done by replacing `'original'` by `'preprocess'` in the above code.

Now that the model is fitted, we can estimate it by

```
% Estimate the fitted model on a test vector X
Yhat = simnpr(model,X);
```

The vector `Yhat` contains the fitted values as in (2). Similarly, one can pass a different vector `Xt` to `simnpr` instead of `X`.

3 Data plotting

Once the model is fitted on the data, we can plot the resulting estimator together with the data and the underlying function (if available). Suppose we know the true underlying function $m(\mathbf{X}_i)$ which we represent by `Ytrue`. We can now plot both `Ytrue` and the fitted model by:

```
% Plot the trained model together with the true underlying
   function
plotnpr(model,[],Ytrue);
```

In case we do not know `Ytrue`, we only plot the fitted model by

```
% Plot the trained model
plotnpr(model);
```

The data is plotted in blue, the fitted model in red and the true underlying function in green.

4 Confidence band estimation

Suppose we have an estimator $\hat{m}_n$ for our model m. One may wonder: How good is this estimate? The problem of finding confidence bands for an estimator addresses this question. In particular, given $\alpha \in (0,1)$ we look for a bound g_α such that

$$\mathbb{P}\left[\sup_x |\hat{m}_n(x) - m(x)| \leq g_\alpha\right] \geq 1 - \alpha \tag{3}$$

at least for large samples. In words: we look for a bound g_α such that with a probability of at least $1-\alpha$ our true function lies entirely within this bound. This bound g_α is computed using the volume-of-tube approximation as proposed by Sun and Loader (1994) and implemented in the toolbox. The general idea is that we construct a tube around the estimator and then search the minimal width c of this tube such that $\hat{m}_n$ satisfies (3). In a one-dimensional setting with $d = 1$, we look for a c that satisfies

$$\alpha \approx \frac{\kappa_0}{\pi}\left(1 + \frac{c^2}{\nu}\right)^{-\nu/2} + \frac{\zeta_0}{2}\mathbb{P}(|t_\nu| > c) \tag{4}$$

for a given α and with t_ν a t-distributed random variable with $\nu = n - tr(L)$ degrees of freedom. κ_0 and ζ_0 are geometric constants. κ_0 can be computed using quasi-Monte Carlo integration and $\zeta_0 = 2$ in a one-dimensional setting.

The extension to a multi-dimensional setting (i.e. $d > 1$) is more complicated and is given by:

$$\begin{aligned}\alpha \approx \kappa_0 \frac{\Gamma((d+1)/2)}{\pi^{(d+1)/2}}\mathbb{P}\left(F_{d+1,\nu} > \frac{c^2}{d+1}\right) + \frac{\zeta_0}{2}\frac{\Gamma(d/2)}{\pi^{d/2}}\mathbb{P}\left(F_{d,\nu} > \frac{c^2}{d}\right) \\ + \frac{\kappa_2 + \zeta_1 + m_0}{2\pi}\frac{\Gamma((d-1)/2)}{\pi^{(d-1)/2}}\mathbb{P}\left(F_{d-1,\nu} > \frac{c^2}{d-1}\right)\end{aligned} \tag{5}$$

with Γ the Gamma function, F the F-distribution with 2 parameters and $\nu = tr((I-L)(I-L)^T)^2/tr(((I-L)(I-L)^T)^2)$. Again, we search for c that satisfies (5) for a given confidence level α and the geometric constants $\kappa_0, \zeta_0, \kappa_2, \zeta_1$ and m_0 can all be computed using quasi-Monte Carlo integration techniques. We refer the interested reader to Sun and Loader (1994) for details.

The main computational burden is the computation of the geometric constants. The toolbox takes advantage of the parallel nature of the computations and computes these in parallel when the Parallel Computing toolbox is installed. Sun and Loader (1994)

have found that not all terms of the sum in (5) are equally important. In particular, the 2nd term involving ζ_2 and the 3rd term with κ_2, ζ_1 and m_0 are less important when the bandwidth h is small. This result has also been verified empirically by Kerstens (2012).

Confidence bands are then constructed by

$$\hat{m}_n(x) \pm (c + \delta)\sqrt{\mathbb{V}_x},$$

with c the solution of (4) or (5) depending on d, δ a bias correction computed by undersmoothing and $\mathbb{V}_x$ the estimated variance of the linear smoother.

Computation and plotting of the confidence bands by the toolbox is now done by the commands

```
% Compute 100*(1-alpha)% confidence bands by volume-of-tube
   approximation
ci = cinpr(model,alpha);
% Plot the trained model together with the true underlying
   function and
% confidence bands
plotnpr(model,ci,Ytrue);
```

5 Adding custom linear smoother

Custom written linear smoothers by the user can easily be added to the toolbox as follows. The custom linear smoother must accept the following parameters:

1. $\mathbf{X}, \mathbf{Y}$: the data to fit containing n data points,
2. **newX**: a $nX \times d$ matrix of evaluation points,
3. h: a bandwidth parameter of the kernel function.

Furthermore, the custom linear smoother must return:

1. **newY**: a $nX \times 1$ vector of fitted points evaluated at **newX**,
2. **L**: a $nX \times n$ smoother matrix as in (2),
3. h: the same bandwidth parameter of the kernel function.

A custom linear smoother, called `'mylinearsmoother'`, is represented by a `.m` file of the form:

```
function [newY,L,h,varargout] = mylinearsmoother(X,Y,newX,h,
   varargin)
% Implementation of 'mylinearsmoother'
end
```

and added to the toolbox by placing it in the `addins` subfolder. No additional steps are needed.

6 Concluding remarks

We hope that by reading to the end of this quickguide, the user has enough information concerning the working of the toolbox to get started. Every `*.m` file of this toolbox contains more detailed information regarding its use and we hope that this together with this quickguide allows the user to explore the more advanced possibilities of the toolbox.

References

P. J. Kerstens. Uniform confidence intervals for nonparametric regression. Master's thesis, KU Leuven, 2012.

J. Sun and C. R. Loader. Simultaneous confidence bands for linear regression and smoothing. *The Annals of Statistics*, 22(3):1328–1345, 09 1994. doi: 10.1214/aos/1176325631. URL http://dx.doi.org/10.1214/aos/1176325631.