Advanced Machine Learning
Practical 4b: Regression
(BLR, GPR & Gradient Boosting)

Professor: Aude Billard
Assistants: Nadia Figueroa, Ilaria Lauzana and Brice Platerrier
`E-mails:` aude.billard@epfl.ch, nadia.figueroafernandez@epfl.ch
ilaria.lauzana@epfl.ch, brice.platerrier@epfl.ch

Spring Semester 2017

1 Introduction

During this week's practical we will continue covering the different regression methods seen in class, namely Gaussian Process Regression (GPR) and Gradient Boosting. To ease into GPR we begin with Bayesian Linear Regression (BLR) which can also be used to better understand Relevant Vector Regression (RVR), from the previous practical. We then cover Gradient Boosting and learn how to tune their hyper-parameters.

2 ML_toolbox

`ML_toolbox` contains a set of methods and examples for easily learning and testing machine learning methods on your data in MATLAB. It is available in the following link:

`https://github.com/epfl-lasa/ML_toolbox`

From the course Moodle webpage (or the website), the student should download and extract the .zip file named `TP4a-Regression.zip` which contains the following files:

Code for each Task				
`TP4_blr.m`	`TP4_gpr.m`	`TP4_gradBoost.m`	`TP4_svr_vs_gpr.m`	`setup_TP4.m`

Before proceeding make sure that `./ML_toolbox` is at the same directory level as the TP directory `./TP4b-Regression` and `setup_TP4.m` as follows:

- 🟥 `My AML Practicals Folder`
  - 🟦 `ML_toolbox`
  - 🟦 `TP4b_Regression`
  - 🟩 `setup_TP4.m`

Now, to add these directories to your search path, type the following in the MATLAB command window:

```
>> setup_TP4.m
```

NOTE: To test that all is working properly with `ML_Toolbox` you can try out some examples of the toolbox; look in the **examples** sub-directory.

2.1 Regression Metrics

The metric you will use during this practical to compare the performance of each regressor will be the *Normalized Mean Square Error* ($NMSE$). If you have a vector $\hat{Y}$ consisting in n predictions and a vector Y of the observed values corresponding to these predictions, you can compute the *Mean Square Error* (MSE) of the predictor :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2 \tag{1}$$

The *Normalized Mean Square Error* ($NMSE$) is simply the MSE normalized by the variance of the observed values :

$$NMSE = \frac{MSE}{VAR(Y)} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2}{\frac{1}{n-1}\sum_{i=1}^{n}\left(Y_i - \mu\right)^2}, \tag{2}$$

where μ is the mean of the observed values : $\mu = \frac{1}{n}\sum_{i=1}^{n} Y_i$.

3 Bayesian Linear Regression

In Bayesian linear regression (BLR), referred to as *probabilistic regression* in the lecture slides, we seek to find the parameters of a linear regression model through a statistical approach. To recall, classical linear regression builds a linear model of the following form:

$$y = f(x) = \mathbf{w}^{\mathrm{T}}\, x \tag{3}$$

where $y \in \mathbb{R}$ and $\mathbf{w}, x \in \mathbb{R}^N$. Given a data set consisting of output-input pairs:

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(i)}, y^{(i)}), \ldots (x^{(M)}, y^{(M)}) = (X, \mathbf{y})$$

we are interested in finding the weights $\mathbf{w}$ of 3. To do so, we fit the given data to (3) by minimizing a *loss function*; typically the least square error: $||\,\mathbf{y} - \mathbf{w}^{\mathrm{T}} X||$. Minimizing the least square error results in a regressor function known as Ordinary Least Squares (OLS):

$$\mathbf{w}_{\mathrm{OLS}} = (\mathrm{X}^{\mathrm{T}}\, X)^{-1}\, \mathrm{X}^{\mathrm{T}}\, \mathbf{y} \tag{4}$$

In BLR, we assume that the values of y contain an additive noise ϵ that follows a zero-mean Gaussian distribution, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, this linear model has the following form:

$$\begin{aligned} y &= f(x) + \epsilon \\ &= \mathbf{w}^{\mathrm{T}} x + \epsilon \end{aligned} \tag{5}$$

The fact that we assume the noise follows a $\mathcal{N}(0, \sigma^2)$ means that we are putting a prior distribution over the noise. We can compute the likelihood of (5) as (assuming i.i.d.):

$$p(\mathbf{y}\,|X, \mathbf{w}; \sigma^2) = \mathcal{N}(\mathbf{y} - \mathbf{w}^{\mathrm{T}} X; I\sigma^2) \tag{6}$$

$$= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} ||\,\mathbf{y} - \mathbf{w}^{\mathrm{T}} X||\right) \tag{7}$$

By following a **Maximum Likelihood Estimation (MLE)** approach one could estimate the weights $\mathbf{w}$ as follows:

$$\nabla_{\mathbf{w}} \log p(\mathbf{y}\,|X, \mathbf{w}) = -\frac{1}{\sigma^2} \mathrm{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{w}^{\mathrm{T}} X) \tag{8}$$

$$\mathbf{w}_{\mathrm{ML}} = (\mathrm{X}^{\mathrm{T}} X)^{-1} \mathrm{X}^{\mathrm{T}} \mathbf{y} \tag{9}$$

Interestingly, this **ML** yields the same result as the least square estimate 4, meaning that it does not take into account the prior on the noise. To consider priors on variables, one must compute the **Maximum A Posterori (MAP)** estimate. In BLR we make use of the fact that the likelihood function is Gaussian and add a prior Gaussian distribution over the weights, $\mathbf{w}$. The posterior of (5) is then:

$$\overbrace{p(\mathbf{w}\,|X, \mathbf{y})}^{posterior} \propto \overbrace{p(\mathbf{y}\,|X, \mathbf{w})}^{likelihood}\, \overbrace{p(\mathbf{w})}^{prior} \tag{10}$$

where $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_w)$ is the prior distribution on the weights and $p(\mathbf{w}\,|X, \mathbf{y})$ is the posterior distribution. Finding the parameters $\mathbf{w}_{\mathrm{MAP}}$ of BLR consists of computing the expectation over the posterior:

$$\begin{aligned} \mathbf{w}_{\mathrm{MAP}} &= E\{p(\mathbf{w}\,|X, \mathbf{y})\} = \frac{1}{\sigma^2} A^{-1} X\, \mathbf{y} \\ \text{with} \quad A &= \sigma^{-2} X\, \mathrm{X}^{\mathrm{T}} + \Sigma_w^{-1} \end{aligned} \tag{11}$$

To predict the output y^* of a new testing point x^*, we take the expectation of the following distribution:

$$p(y^*|x^*, X, \mathbf{y}) = \mathcal{N}(\frac{1}{\sigma^2} x^{*T} A^{-1} X\, \mathbf{y}; x^{*T} A^{-1} x^*) \tag{12}$$

which yields:

$$\begin{aligned} y^* &= E\{p(y^*|x^*, X, \mathbf{y})\} \\ &= \frac{1}{\sigma^2} x^{*T} A^{-1} X\, \mathbf{y} \end{aligned} \tag{13}$$

Not only do we take into account the variance in the noise σ^2 and the uncertainty in the weights Σ_w, we can also compute the uncertainty of the prediction through the variance:

$$var\{p(y^*|x^*, X, \mathbf{y})\} = x^{*T} A^{-1} x^* \tag{14}$$

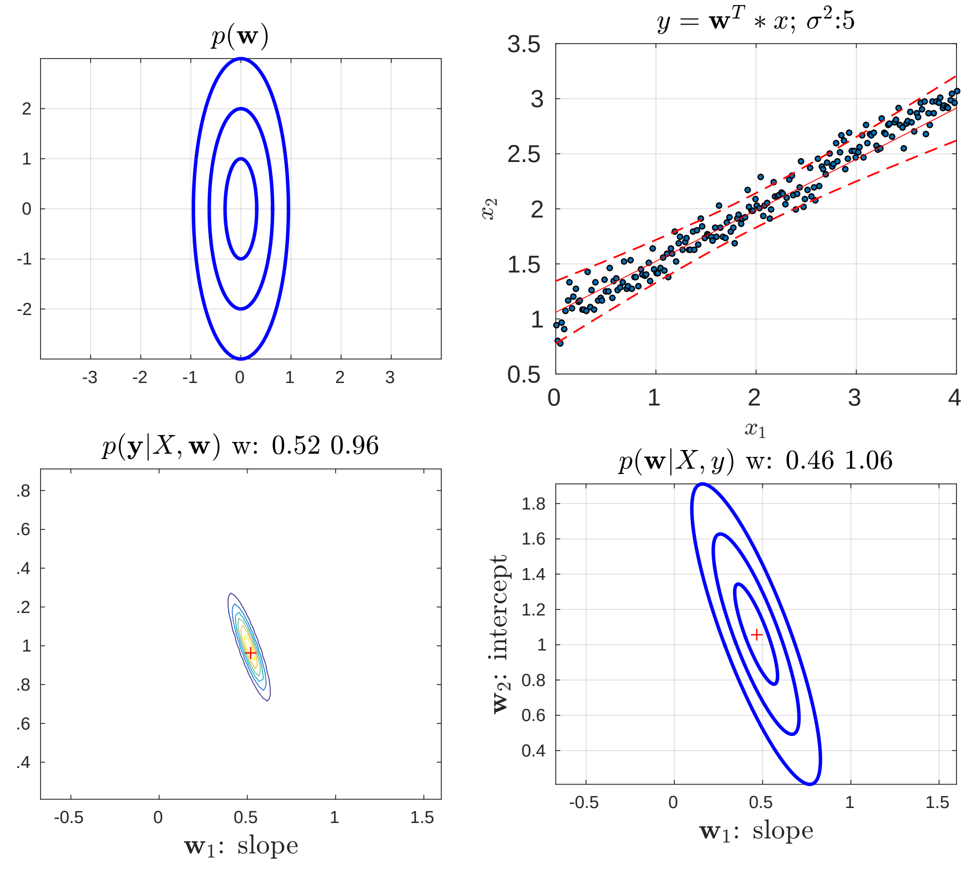


Figure 1: Bayesian Linear Regression on noisy line signal.

Hyper-parameters of BLR

There are two hyper-parameters for Bayesian Linear Regression:

- σ^2: variance of the measurement noise, $p(\mathbf{y}\,|X, \mathbf{w}; \sigma^2)$
- Σ_w: prior uncertainty on the parameters of $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_w)$.

`Illustrative example:` Open the MATLAB script **TP4_blr.m** and load the noisy line data in sub-block `(1a)` and train a BLR model by running code block `(2)`. You should see the a set of plots as in Figure 1 which clearly depict the relation between the prior, likelihood and posterior distributions in BLR. For a more detailed explanation on BLR you can consult Chapter 2 from William's Gaussian Process book.

TASK 1: Analyze the effect of the hyper-parameters of BLR

1. How does the prior on the weights $\mathbf{w}$ change your estimated regressive signal?
2. How does the prior on the noise ϵ change your estimated regressive signal?
3. How do the prior parameters adapt as a function of the number of sample of your training data?

`Hint: Follow code block 3 in TP4_blr.m for question 1+2 (play around with the hyper-parameters) and code block 4 for question 3.`

4 Gaussian Process Regression

Gaussian Process Regression (GPR) is the non-linear kernelized versino of BLR. Rather than modeling a distribution over data-points, GPR models a distribution over functions:

$$\begin{aligned} y &= f(x) + \epsilon \qquad (15) \\ &= \mathbf{w}^{\mathrm{T}}\,\phi(x) + \epsilon \end{aligned}$$

As in any kernel method $\phi(x)$ is a non-linear transformation. Following the same MAP estimation procedure as in BLR, to predict the output y^* of a new testing point x^*, we take the expectation of the following distribution:

$$p(y^*|x^*, X, \mathbf{y}) = \mathcal{N}(\frac{1}{\sigma^2}\phi(x^*)^T A^{-1}\Phi(X)\,\mathbf{y}; \phi(x^*)^T A^{-1}\phi(x^*)) \qquad (16)$$

$$\text{with} \quad A = \sigma^{-2}\Phi(X)\Phi(X)^T + \Sigma_w^{-1} \qquad (17)$$

by defining the kernel function as $k(x, x') = \phi(x)^T\Sigma_w\phi(x)$, this yields,

$$\begin{aligned} y^* = E\{p(y^*|x^*, X, \mathbf{y})\} &= \sum_{i=1}^{M} \alpha_i\, k(x, x^i) \qquad (18) \\ \text{with} \quad \boldsymbol{\alpha} &= [K(X, X) + \epsilon_{\sigma^2} I]^{-1}\,\mathbf{y} \end{aligned}$$

Hence, the Gaussian process regression function $f(x)$ is a linear combination of weighed kernel functions evaluated on test points. (18) has a similar structure to SVR/RVR, however in GPR **all the points** are used in the computation of the predicted signal; i.e. $\alpha_i > 0$. As any other kernel-method, one can use many types of kernel functions. In this practical, we will be solely considering the radial basis function (also known as the Gaussian or Squared Exponential kernel),

$$k(x, x') = \exp\left(-\frac{||x - x'||}{l}\right) \qquad (19)$$

Hyper-parameters of GPR

Using this kernel, we have two hyper-parameters to tune for GPR:

- ϵ_{σ^2} : variance of the signal noise.
- l : variance of the kernel, also known as *kernel width* or *length-scale.*

`Illustrative example:` Open the MATLAB script **TP4_gpr.m**, load the noisy sinusoidal data in sub-block `(1a)` and train a GPR model by running code block `(2)`. You should see the plots in Figure 2. For a more detailed explanation on GPR you can consult Chapter 2 from William's Gaussian Process book.

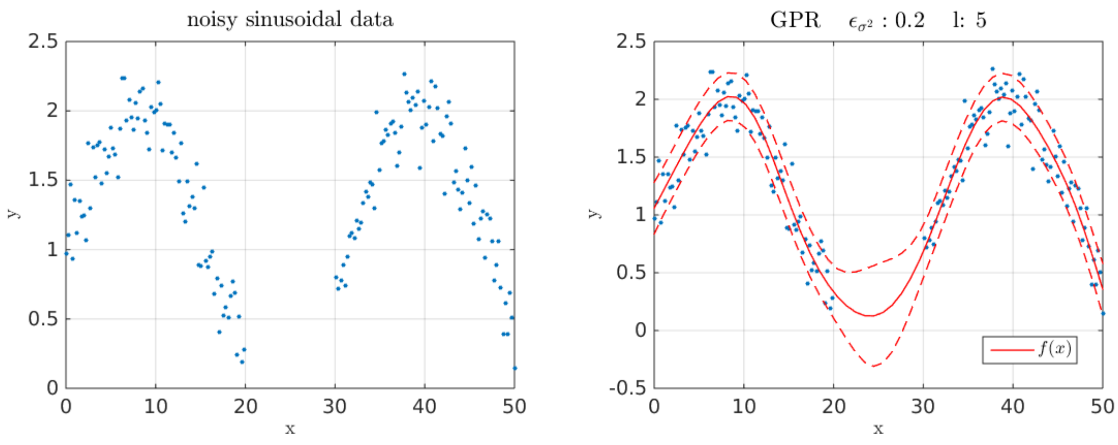


Figure 2: Sinusoidal data with a hole.

TASK 2: Analyze the effect of the hyper-parameters of GPR

1. What is the impact of ϵ on the regressor function ?
2. What is the impact of the kernel width l on the regressor function ?
3. Can one find a suitable kernel width l through Grid Search + Cross-Validation?

`Hint: Use sinusoidal + simpler dataset loaded with sub-block (1b) for question 1. Follow code block 3 in TP4_gpr.m for question 2 (with the sinusoidal data). Run code block 4 for question 3.`

5 Gradient Boosting

As in classification, another approach to obtaining non-linear functions is through boosting. Gradient Boosting is a powerful method that combines AdaBoost + Gradient Descent, by combining a set of *weak* learners into a stronger learning through an iterative approach targeted at minimizing a *loss function* $L(y, f(x))$. The strong regressor function $f(x)$ is thus composed of a weighted sum of weak regressors $h_i(x)$:

$$f(x) = \sum_{i=1}^{N} \gamma_i h_i(x) + b \tag{20}$$

where b is a base constant value that is used to initialize the model, where γ_i is the weight for each weak learner $h_i(x)$, which could be any type of function estimator. In this tutorial we will use decision trees, which are the most used in practice, and squared loss function $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$. Given a dataset

$$(x^{(1)}, y^{(1)}), \dots, (x^{(i)}, y^{(i)}), \dots (x^{(M)}, y^{(M)}) = (X, \mathbf{y})$$

estimating (20) involves iterating the following 4 steps for $i = \{1, \dots, N\}$ iterations, once $f_0(x) = b$ is initialized:

1. Compute current residuals:

$$r^{(j)} = -\left[\frac{\partial L(y^{(j)}, f_{i-1}(x^{(j)}))}{\partial f_{i-1}(x^{(j)})}\right] \quad \text{for} \quad j = \{1 \dots M\}$$

2. Fit a weak learner $h_i(x)$ on the residuals :

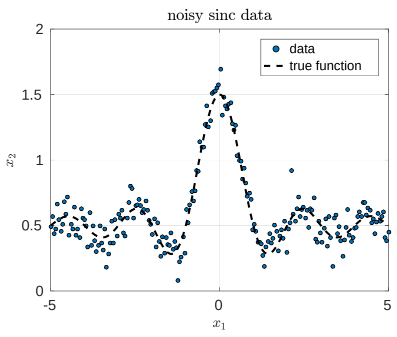
$$(x^{(1)}, r^{(1)}), \dots, (x^{(i)}, r^{(i)}), \dots (x^{(M)}, r^{(M)}) = (X, \mathbf{r})$$

3. Compute weight for current step:

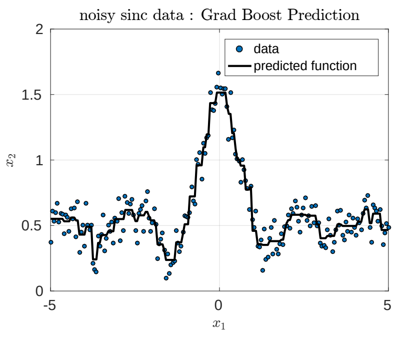
$$\gamma_i = \arg\min_{\gamma} \sum_{j=1}^{M} L(y^{(j)}, f_{i-1}(x^{(j)}) + \gamma h_i(x^{(j)}))$$

4. Update function: $f_i(x) = f_{i-1}(x) + \gamma_i h_i(x)$ and go back to step 1.

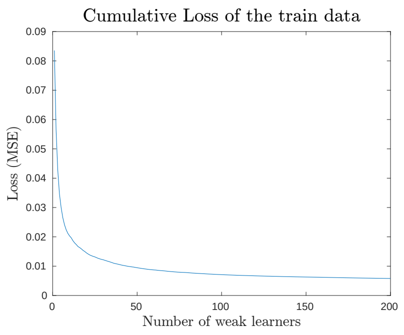
In the literature, gradient boosting (GB) is also referred to as gradient boosted decision trees (GBDT) or least square boost (LS Boost). For a more detailed description of the algorithm we refer to this tutorial and this very intuitive interactive visualization.



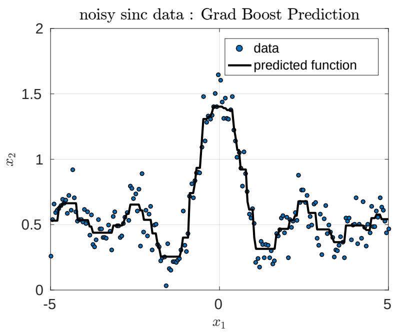
(a) Noisy Sinc Dataset



(b) Predicted Regressor with 200 weak learners



(c) Mean Square Error throughout iterations



(d) Predicted Regressor with 100 weak learners

Figure 3: Gradient Boosting on noisy sinc dataset with Decision Trees as *weak learners*.

`Illustrative example:` Open the MATLAB script **TP4_gradBoost.m**, load the noisy sinc data in sub-block `(1a)` and train a Gradient Boosting model by running code block `(2)`. You should see the plots in Figure 3.

TASK 3: Apply Gradient Boosting on more challenging datasets

1. How sensitive is it to noise?
2. How sensitive is it to missing data?

`Hint: Use the hole dataset from the GPR exercises. Play around with the noise values when generating the datasets.`

6 Cross-Validation for Model Comparison (Optional: TO DO AT HOME)

6.1 Comparison of SVR vs. GPR

In this section you will compare the results of SVR and GPR using different metrics. For this, you will use two real-world datasets related to wines from the north of Portugal. The goal is to grade wine quality based on physicochemical tests. Each dataset is composed of 11 real input variables and one output variable the wine quality (between 0 and 10). A more detail description of these dataset is available here `http://archive.ics.uci.edu/ml/datasets/Wine+Quality`.

Open **TP4_svr_vs_gpr.m** to start.

Goals : Try to see for each regression technique and each dataset :

- Which one seems to yield the best precision ?
- Which one is the easiest to setup ?
- Do they always converge to the same solution ?
- Is there any difference if you compare the performance with different metrics ?
- What is the computational cost of each method and how do they compare?
- Is the performance changing if you remove a certain percentage of data ?
- Is the performance changing if you use only some features (attributes) ?
  Feature Selection can be a good way to reduce the error. You can either select a subset of attributes or perform some dimensionality reduction technique we have seen in the second practical.

You first load the dataset and plot it. Then you also normalized it to have a comparable influence of each attribute. Then you perform **GPR** and **SVR** (ϵ-SVR or ν-SVR) for

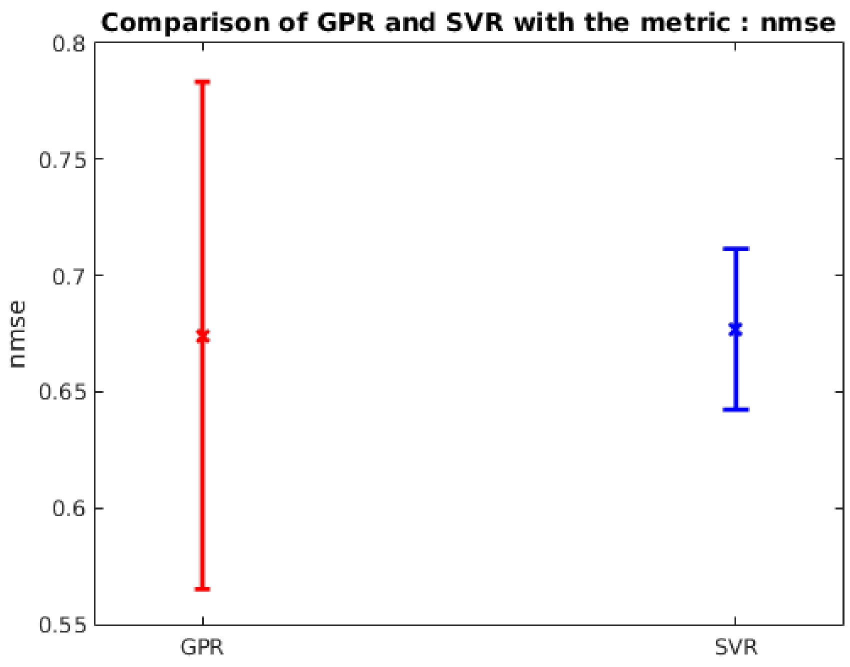


Figure 4: **Example of result with manual parameter selection**.

which you can perform a grid search with 10-fold cross-validation to find the best hyper-parameters or simply enter the parameters you want. Choose a good range for the grid search if you use it.
Finally, you can compare the performance using one of the proposed metrics (MSE or $NMSE$).