Advanced Machine Learning
Practical 4b Solution: 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:

<table>
<thead>
<tr>
<th>Code for each Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP4_blr.m</td>
</tr>
</tbody>
</table>

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:

```
1  >> 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:

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2
\]

(1)

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

\[
NMSE = \frac{MSE}{\text{VAR}(Y)} = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2 \frac{1}{\frac{1}{n-1} \sum_{i=1}^{n} (Y_i - \mu)^2},
\]

(2)

where $\mu$ 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:

\[
y = f(x) = w^T x
\]

(3)

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

\[
(x^{(1)}, y^{(1)}), \ldots, (x^{(i)}, y^{(i)}), \ldots (x^{(M)}, y^{(M)}) = (X, y)
\]

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

\[
w_{\text{OLS}} = (X^T X)^{-1} X^T y
\]

(4)
In BLR, we assume that the values of $y$ contain an additive noise $\epsilon$ that follows a zero-mean Gaussian distribution, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, this linear model has the following form:

$$y = f(x) + \epsilon$$

$$= w^T x + \epsilon$$  \hspace{1cm} (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.):

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

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

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

$$\nabla_w \log p(y | X, w) = -\frac{1}{\sigma^2} X^T (y - w^T X)$$

$$w_{ML} = (X^T X)^{-1} X^T y$$  \hspace{1cm} (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, $w$. The posterior of (5) is then:

$$\frac{\text{posterior}}{p(w | X, y)} \propto \frac{\text{likelihood}}{p(y | X, w)} \frac{\text{prior}}{p(w)}$$  \hspace{1cm} (10)

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

$$w_{MAP} = E\{p(w | X, y)\} = \frac{1}{\sigma^2} A^{-1} X y$$

$$\text{with } A = \sigma^{-2} X X^T + \Sigma_w^{-1}$$  \hspace{1cm} (11)

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

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

which yields:

$$y^* = E\{p(y^* | x^*, X, y)\}$$

$$= \frac{1}{\sigma^2} x^T A^{-1} X y$$  \hspace{1cm} (13)

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

$$\text{var}\{p(y^* | x^*, X, y)\} = x^T A^{-1} x^*$$  \hspace{1cm} (14)
Hyper-parameters of BLR

There are two hyper-parameters for Bayesian Linear Regression:

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

Illustrative example: Open the MATLAB script \texttt{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 $w$ change your estimated regressive signal?
2. How does the prior on the noise $\epsilon$ 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.

**Solution Task 1:**
The solution to Bayesian Linear Regression will depend a lot on the prior information you give it. In this case the prior on the weights $w = [0, 0]^T$ of the linear function is zero. This means that before seeing any data our best guess of what the regressor function should look like is a threshold line with a zero bias, which is not very informative. Then there is the variance $\Sigma_p$ associated with the prior guess on the value of $w$, which is a way of encoding how much confidence we have in our prior $w$. In Figure 2 we illustrate the effect $\Sigma_w$ has on the BLR solution, $w_{MAP}$.

Figure 2: **Effect of $\Sigma_p$.** (a) The width of the uncertainty tube (dashed lines) is equivalent to a variance of 5. The extreme high value of $\Sigma_p$ means that that we completely ignore the effect of the prior information and the solution is the same as OLS. (b) $\Sigma_p = I$ is identity, as a result the prior information is taken into consideration and we can see as a result that the offset $w_2$ has shifted by down to zero. (c) $\Sigma_p = 0.001$; we are extremely confident in our prior which totally overshadows the likelihood function. (d) illustration of the effect of $\sigma^2$, compared with (a).

To get a sense of how the prior parameters adapt as a function of the number of points we can look at the rate of converge of the slop and intercept of $w$ for both the OLS and BLR solutions. In Figure 3 we illustrate the convergence rates.
Figure 3: (a) Average noise an prior uncertainty; slow convergence to the true $w$. (b) Very small uncertainty, both in noise and prior; the convergence rate is nearly the same as (b). (c) Uncertainty in measurement is 10 times smaller than the weight prior. Because of this actual data values are taken more into consideration and the both BLR and OLS converge to the same solution. (d) 100 times less uncertainty in prior on weights as oppose to the uncertainty set for observations. As a result many data points will be necessary to overcome the overly confident prior $p(w)$.

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:

$$ y = f(x) + \epsilon = w^T \phi(x) + \epsilon $$

(15)

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:

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

(16)

with

$$ A = \sigma^{-2} \Phi(X) \Phi(X)^T + \Sigma_w^{-1} $$

(17)
by defining the kernel function as $k(x, x') = \phi(x)^T \Sigma_w \phi(x)$, this yields,

$$y^* = E\{p(y^*|x^*, X, y)\} = \sum_{i=1}^{M} \alpha_i k(x, x^i) \quad (18)$$

with $\alpha = [K(X, X) + \epsilon\sigma^2 I]^{-1} y$

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),

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

**Hyper-parameters of GPR**

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

- $\epsilon\sigma^2$ : 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 4. For a more detailed explanation on GPR you can consult Chapter 2 from William’s Gaussian Process book.

![Illustrative Example](image)
TASK 2: Analyze the effect of the hyper-parameters of GPR

1. What is the impact of $\epsilon$ 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.

Solution Task 2:

What is the impact of $\epsilon$ on the regressor function? Intuitively the $\epsilon \sigma^2$ can be thought in terms of how much trust we place in our training data. If you set the value of $\epsilon \sigma^2$ high it means that your training values $y$ are possibly very corrupted and do not accurately reflect the underlying function.

![Diagram](image1)

Figure 5: (a) $\epsilon \sigma^2$ is high; the value of the regressor function $f(x)$ does not go all the way to the blue data points. (b) $\epsilon \sigma^2$ is low; all training data points are considered to be noiseless values of the underlying function, so the regressor predicts exactly the training point values.

If instead your set $\epsilon \sigma^2$ to be a small value, then the values $y$ accurately reflect the underlying function you are trying to regress. In Figure 5 we illustrate the effect this has on the output of your learned GPR function.
In Figure 6 we illustrate the effect \( \epsilon_{\sigma^2} \) has on the regressors prediction.

![Figure 6: Effect of noise variance on regressor](image)

Figure 6: **Effect of noise variance on regressor.** The kernel width is kept constant at \( l = 5 \) and the noise parameter \( \epsilon \) is varied from 0.002 to 15. The shape of the regression line seems to be slightly effected. As the noise level increases to infinity the value of the regressor function tends to zero.
What is the impact of the kernel width $l$ on the regressor function? As the variance of the kernel function goes to zero, the kernel function $k(x', x') = 1$ will only be equal to one when the test point $x'$ is equal to training point $x$; it will be zero otherwise $k(x, x) = 0$. As a result, the regressor function will only return a non-zero value when a test point is within a very small distance from a training point (determined by $l$). When the variance of the kernel tends to infinity, the Gram matrix will be equal to one, $K(X, X) = 1$, the regressor will end up giving the mean value of the data.

Optimal kernel parameter through grid search + CV? Yes, by following the same K-fold Cross-Validation procedure seen in the classification practical, we can find the best kernel width $l$ for our data. By running code block 4 you should get the plot in Figure 9 for the sinusoidal with hole dataset. As can be seen, the optimal values lie within the range of $l = [5 - 10]$. 

Figure 7: **Effect of noise variance on regressor #2.** A more systematic evaluation of the effect of noise on the regressor.
Figure 8: **Effect of kernel variance on regressor.** (a) Very small variances causes the regressor function to be zero everywhere except on test points. (b)-(c) Variance is within the scale of the input space, which results in normal behaviour. (c) The variance, kernel width, is extremely large. We can see that the regressor value lies exactly on top of the mean of the output data, $y$. 
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 $h_i(x)$:

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

where $b$ is a base constant value that is used to initialize the model, where $\gamma_i$ 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

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

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

1. Compute current residuals:

$$r^{(j)} = - \left[ \frac{\partial L(y^{(j)}, f_{i-1}(x^{(j)}))}{\partial f_{i-1}(x^{(j)})} \right] \hspace{1cm} \text{for} \hspace{0.5cm} j = \{1 \ldots M\}$$
2. Fit a weak learner $h_i(x)$ on the residuals:

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

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](#).

**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 10.

![Figure 10: Gradient Boosting on noisy sinc dataset with Decision Trees as weak learners.](#)
**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.

**Solution Task 3:**
Performance of Gradient Boosting on Hole dataset is shown in Figure 11.

![Noisy Hole Dataset](image1)

![Predicted Regressor with 100 weak learners](image2)

![Cumulative Loss of the train data](image3)

![Predicted Regressor with 20 weak learners](image4)

Figure 11: Gradient Boosting on noisy hole dataset with Decision Trees as *weak learners*. 
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 which you can perform a grid search with 10-fold cross-validation to find the best hyperparameters 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).
Figure 12: Example of result with manual parameter selection.