LAB SESSION 1

The aim of this lab session is double : first if you are not familiar with `R`, to give you a quick-start. Second to give you the opportunity to implement some of the methods and principles described in the two first lectures.

This lab session is not marked, you are not supposed to hand out your work in the end. It is a training step to make you ready for the second lab session which will be marked.

1 Introduction to R

If you have never used `R` before or for a quick reminder, open tutorial `Rtutorial.html` in a web browser and follow the instructions.

N.B if you are a student from Telecom you have already seen this tutorial last year. This is just a rehearsal.

2 Bayesian linear regression

The linear model is one of the simplest in the supervised context of regression. Given a training set $(x_i, y_i)_{i=1,\dots,n}$, with $x_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$, the first goal is to learn a linear regression function of the kind $h(x) = \sum_{j=1}^{p} \phi_j(x)\theta_j = \langle \phi(x), \theta \rangle$ where $\phi = (\phi_1, \dots, \phi_p)$ is a vector of basis functions that are fixed in advance and $\theta \in \mathbb{R}^p$ is the regression parameter that we want to learn. The estimated $\widehat{\theta}(x_{1:n}, y_{1:n})$ should fit the data well, *i.e.* the empirical error

$$R_n(\theta, x_{1:n}, y_{1:n}) = \sum_{i=1}^{n} (y_i - \langle \theta, \phi(x_i) \rangle)^2$$

should be small. Also to prevent over-fitting, one would like the estimator $\widehat{\theta}$ to be robust, *i.e.* ideally $\|\theta\|_1$ or $\|\theta\|_2$ should remain moderate.

In a probabilistic setting the y_i's are viewed as realizations of random variables Y_i's such that

$$Y_i = \langle \theta, \phi(x_i) \rangle + \epsilon_i, \ i \in \mathbb{N} \qquad \text{with } (\epsilon_i)_{i \in \mathbb{N}} \overset{\text{i.i.d}}{\sim} \mathcal{N}(0, \beta^{-1}). \tag{1}$$

Here the noise precision β is viewed as a fixed and known model parameter.

In a Bayesian setting one defines a prior $\boldsymbol{\pi}$ on $\boldsymbol{\theta}$. In this particular case a Gaussian prior is convenient because it is conjugate. Remind from last lecture that choosing a prior precision of the kind αI_p and zero prior mean yields an easy-to-handle posterior distribution The Bayesian linear model considered in this lab session is thus

$$\begin{cases} \boldsymbol{\theta} \sim \mathcal{N}(0_p, \alpha^{-1} I_p) \\ \mathcal{L}(Y_i | \theta) = \mathcal{N}(\langle \theta, \phi(x_i) \rangle, \beta^{-1}) \end{cases} \tag{2}$$

where α, β are hyper-parameters and $\mathcal{L}(Y_i|\theta)$ is the conditional distribution of Y_i given $\boldsymbol{\theta} = \theta$.

Remind from last lecture that the posterior distribution takes the form

$$\begin{gathered} \boldsymbol{\pi}(\,\cdot\,|y_{1:n}) = \mathcal{N}(m_n, S_n) \text{ with} \\ m_n = \Big(\frac{\alpha}{\beta} + \Phi^\top \Phi\Big)^{-1} \Phi^\top y_{1:n}\,, \qquad S_n = \big(\alpha I_p + \beta \Phi^\top \Phi\big)^{-1}. \end{gathered} \tag{3}$$

2.1 Posterior distribution in a linear model

Write a function returning the posterior mean and variance as a list, according to the following model.

N.B. Argument `feature_map` below is a function, *e.g.* for a polynomial basis function model `feature_map = function(x){c(1, x,x^2,x^3, x^4)}`.

```
glinear_fit <- function(Alpha, Beta, data, feature_map, target)
   ## Alpha: prior precision on theta
   ## Beta: noise precision
   ## data: the input variables (x): a matrix with n rows
#### where n is the sample size
   ##feature_map: the basis function, returning a vector of
#### size p equal to the dimension of theta
   ## target: the observed values y: a vector of size n
{
  Phi <- ## complete:
  ## ? apply
  ## ? t
    p = ncol(Phi)
    posterior_variance <- ## complete
    posterior_mean <- ## complete
    return(list(mean=posterior_mean, cov=posterior_variance))
}
```

To test your model,

1. Generate a dataset $(x_{1:n}, Y_{1:n})$ according to model (1) with fixed $\theta_0 \in \mathbb{R}^5$ and $\beta_0 > 0$ and the polynomial basis function `Fmap <- function(x){c(1, x,x^2,x^3, x^4)}`. For the input data $x_{1:n}$, you may *e.g.* generate n uniform random variables in $[-3, 3]$. As default values for a start, you may take *e.g.* $\theta_0 = (5, 2, 1, -1, -0.1)$ $\beta_0 = 1$ and $n = 100$.
2. Compute the posterior distribution of $\boldsymbol{\theta}$ and check that the posterior mean converges to θ_0 for large sample sizes. How does the posterior variance behave ? For plotting, you may adapt the following code to your purposes

```
## dummy plotting example
xx <- 1:100
yy <- sin(xx/10) * xx/10
pp <- yy + rnorm(n=100)
interv <- sqrt(abs(yy))
lsup <- yy+ 1.96*interv
linf <- yy- 1.96*interv
plot(xx, yy, type='l', lwd=3, ylim=range(lsup,linf))
lines(xx, lsup, col='red')
lines(xx, linf, col='red')
points(xx, pp, pch=19,col='blue')
legend('top', legend=c('estimate', 'credible levels', 'data'),
     col=c('black', 'red', 'blue'),
     pch= c(NA, NA, NA, 19),
     lwd=c(3,1,1,NA)
     )
```

3. Generate a dataset of size $N = 1000$ and for $n \in \{1, \dots, N\}$ compute the posterior means and variance of $\boldsymbol{\theta}$ using the first n values of the dataset. Concerning the covariance, only keep track of the diagonal entries, *i.e.* the posterior variances of the $\boldsymbol{\theta}_j$'s. Plot on the same figure, for $j = 1, \dots, 4$, the true value $\theta_{0,j}$ the graph (as a function of n) of the mean estimators of $\mathbb{E}(\boldsymbol{\theta}_j | x_{1:n})$ and centered 95% posterior credible intervals for each value of n.

2.2 Predictive distribution

1. Recall the definition of the posterior predictive distribution of Y_{new} at a new input point x_{new}. What is its expression in the Bayesian linear model ?
2. Write a function returning the mean and variance of the predictive distribution, at new input points x_{new}, according to the following model

```
 glinear_pred <- function(Alpha, Beta, fitted, data, feature_map)
   ## Alpha: prior pecision for theta
   ## Beta: noise variance
   ## fitted: the output of glinear_fit: the posterior mean and
#### variance of the parameter theta.
   ## data: new input data where the predictive distribution
#### of Y must be computed
   ## feature map: the vector of basis functions
{
   Phi_transpose <- ## complete
   pred_mean <- ## complete
   pred_variance <- ## complete
   return(list(mean = pred_mean, variance = pred_variance))

}
```

3. Take $(x_{new,i}), i \in \{1, \dots, 200\}$ on a regular grid on the interval $[-3, 3]$. (see `? seq`). Compute the posterior predictive distribution of the $Y_{new,i}$'s using the function `glinear_pred`. On the other hand generate new targets $Y_{new,i}$ using the same true parameters as in Section 2.1. Plot on the same graph :
   - the posterior predictive mean and posterior credible intervals for the predictive distribution as a function of x_{new},
   - the true regression function
   - the generated 'true' labels $Y_{new,i}$.

2.3 Empirical Bayes for linear regression

Until now, the hyper parameter α for the prior precision on $\boldsymbol{\theta}$ has been set to arbitrary values. Also, the parameter β (noise precision) is needed to fit the model, whereas the true value β_0 is in general unknown. Finally, the number of basis functions (*i.e.* the model dimension) has been set to $p = 5$ both for data simulation and mode fitting. However in practice, the data may not come exactly from a polynomial basis function model and even though, the dimension is unknown. Let us denote by $\gamma = (\alpha, \beta, p)$ the unknown parameters (except from θ) which need to be chosen.

Given a set of Bayesian models $\{M_\gamma, \gamma \in \Gamma\}$ with $M_\gamma = \{P_\theta, \theta \in \Theta_\gamma, \boldsymbol{\pi}_\gamma\}$ and a dataset $z_{1:n}$ (here $z_{1:n} = (y_{1:n}, x_{1:n})$, empirical Bayes consists in selecting γ^* as a maximizer of the *model evidence*

$$p(z_{1:n}|\gamma) = \int_{\Theta_\gamma} p_\theta(z_{1:n}) \pi_\gamma(\theta) \, \mathrm{d}\theta.$$

1. Show that the log-evidence of for $\gamma = (p, \alpha, \beta)$ is

$$\log p(y_{1:n}|p,\alpha,\beta) = \frac{-n}{2}\log(2\pi) - \frac{1}{2}\log\det\Sigma - \frac{1}{2}y_{1:n}^\top \Sigma^{-1} y_{1:n} \ ,$$

with $\Sigma = \left(\alpha^{-1}\Phi\Phi^\top + \beta^{-1}I_n\right)$.
hint : the evidence is the marginal density of $Y_{1:n}$ evaluated at $y_{1:n}$, where $Y_{1:n}$ is viewed as a component of the random vector $(Y_{1:n}, \boldsymbol{\theta})$ and where $\boldsymbol{\theta} \sim \boldsymbol{\pi}_\gamma$. Write $Y_{1:n}$ as a function of $(\boldsymbol{\theta}, \epsilon)$ and conclude.

2. (homework) Show that the log-evidence can be written

$$\log p(y_{1:n}|p,\alpha,\beta) = \frac{-n}{2}\log(2\pi) + \frac{n}{2}\log(\beta) + \frac{p}{2}\log(\alpha) - \frac{1}{2}\log(\det A) - \frac{\beta}{2}\|y_{1:n} - \Phi m_n\|^2 - \frac{\alpha}{2}m_n^\top m_n$$

with $A = \alpha I + \beta\Phi^\top\Phi$. This alternative expression is particularly useful when $n \gg p$ because it does not require inverting a $n \times n$ matrix.
To do so, use the previous result and the identities

$$\det(I_p + A^\top B) = det(I_n + AB^\top) \ \text{ for } A, B \in \mathbb{R}^{n\times p}$$
$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1} \text{ for } A, B, D, C \text{ such that the products are well defined}$$

3. implement a function `logevidence` computing the log-evidence of the hyper parameters according to the model below. Notice that the dimension p of the model is implicit and can be deduced from the argument `feature_map`.

```
logevidence <- function(Alpha, Beta, data ,feature_map, target)
   ## Alpha: prior precision for theta
   ## Beta: noise precision
   ## data: the input points x_{1:n}
   ## feature_map: the vector of basis functions
   ## target: the observed values y: a vector of size n.
{
   Phi_transpose <- ## complete the code
   if(is.vector(Phi_transpose)){
      Phi_transpose = matrix(Phi_transpose,nrow=1)
    }
    ## avoids undesired matrix-> vector conversions for one
    ## dimensional feature maps
   Phi <- t(Phi_transpose)
   N <- nrow(Phi)
   p <- ncol(Phi)

   ### complete the code

   return(res)
}
```

4. Use the function `logevidence` to choose the parameter α when setting all other unknown parameters to their true values ($\beta = \beta_0, p = 5$). Proceed by grid-search, *i.e.* compute the log-evidence for α varying on a regularly spaced grid and determine the maximizer. Plotting the log-evidence curve is a good idea.
hint : The functions `sapply` and **`which.max`** may be useful.

5. Proceed similarly with β and check that the chosen β^* is close to β_0.
6. proceed similarly with the polynomial order of the regression, by computing the model evidence for a polynomial order ranging from 0 to 7. You may copy-paste the following basis functions

```
F7 <- function(x){c(1, x, x^2, x^3, x^4,x^5, x^6, x^7)}
F6 <- function(x){c(1, x, x^2, x^3, x^4,x^5, x^6)}
F5 <- function(x){c(1, x, x^2, x^3, x^4,x^5)}
F4 <- function(x){c(1, x, x^2, x^3, x^4)}
F3 <- function(x){c(1, x, x^2, x^3)}
F2 <- function(x){c(1, x, x^2)}
F1 <- function(x){c(1, x)}
F0 <- function(x){1}
listF=list(F0,F1,F2,F3,F4,F5,F6,F7)
```

7. perform a joint optimization over p (polynomial order $+1$), α, β. A joint optimization over (α, β) can easily be achieved using the optimization routine `optim` from `R`. The `"L-BFGS-B"` method (passed as argument `method` to `optim`) allows for box constraints.
   Discuss the results.
8. **model misspecification** : instead of generating data from the polynomial basis linear model, fix as true regression function the sinusoidal $h_0(x) = \sin(x)$. Follow the same lines as above and discuss the results.

3 Naive Bayes (optional)

The goal is to predict the income class of a population (less or greater than $50K$) using several predictors, some of them real valued, some of them categorical.

Before starting, you should know that naive Bayes is implemented in `R` package `e1071`

```
install.packages("e1071")
library(e1071)
? naivebayes
```

The goal of the remainder of this section is to let you implement naive Bayes yourself (with some guidance). In the end you'll be able to compare your results with those obtained with the `e1071` package.

We use the simplest possible models for the features (conditionally to the class) : the numerical features are modeled as Gaussian variables, which estimated mean and variances are the empirical mean and variance. As for the categorical variables, the estimated probability of each category is be defined as the empirical proportion of that category (again, conditionally to a class).

3.1 Data loading and preprocessing

Download the `adult` data from the UCI ML database :

```
 install.packages('RCurl')
library(RCurl)
urlfile <- address
```

with `address ='https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'` Then load the dataset into `R` :

```
 # download the file
downloaded <- getURL(urlfile, ssl.verifypeer=FALSE)
```

```
# treat the text data as a steam so we can read from it
connection <- textConnection(downloaded)
# parse the downloaded data as CSV
dataset <- read.csv(connection, header=FALSE, as.is=FALSE)
```

Inspect the dataset :

```
# preview the first 5 rows
head(dataset)
```

Be careful : some columns are converted factors. It's OK. (see `help(factor)`). Those columns correspond to categorical variables.

```
 ## which columns are factors ? (thus, categorical ?)
indx <- sapply(adult, is.factor)
Lev <- lapply(adult[indx], function(x) levels(x))
Lev
## a list which i'th element is the vector of categories (stored as
## strings) for the i'th categorical variable.
```

The predicted variable is V15 : income $> 50K$ or income $\leq 50K$.

Create a training and a testing set, and separate the classes from the features in the training set

```
N = dim(adult)[1]
p = dim(adult)[2] - 1

Ntrain = floor(2*N/3)
training = adult[1:Ntrain,]
test = adult[(Ntrain+1):N,]

Class = training[,p+1]
Features = training[,1:p]
```

3.2 Defining a prior distribution for the classes based on the dataset

Complete the code :

```
   Nclass = length(levels(Class))
classPrior=rep(0,Nclass)
for (i in 1:Nclass){
   classPrior[i] = ##complete
}
classPrior
```

3.3 Training the model

Complete the code of the training function below. The returned value should be a list which j^{th} element corresponds to the j^{th} feature and is defined as

- if the j^{th} feature is numeric : a matrix with K columns and 2 rows, with K the number of classes. The k^{th} column is the mean and variance of the estimated normal distribution for the j^{th} feature given the class k.

- if the j^{th} feature is categorical : a matrix with K columns. The k^{th} column is the estimated parameter of a categorical model. So the number of rows is the number of categories for the j^{th} feature.

```
train_naivebayes <- function(Class, predictors, laplace = 1)
## Class: vector of target classes for the classification problem
###### (a factor)
## predictors: the features (x) of the training data
## laplace: an integer(0 or 1): 1 for laplace regularization,
###### i.e. for assigning an additional observation to each class
###### (avoids zeros)
{
   Nclass = length(levels(Class))
   p = dim(predictors)[2]
   indClass=list()
   ## select indices corresponding to each class k
   for(k in 1:Nclass){
      indClass[[k]]= which(Class==levels(Class)[k])
   }
   fitted=list()
   for (j in 1:p){
      if(is.factor(predictors[,j])){
         ## fit a multinomial model:
         ## store the estimated probability for a class given
         ## a predictor in a matrix (nrows: number of levels,
         ## ncol: number of classes)
         nlev_j = length(levels(predictors[,j]))
         fitted[[j]]=matrix( nrow= nlev_j, ncol=Nclass)
      }
      else{ ##fit a normal model: store estimated mean and variance
         fitted[[j]]=matrix(0,nrow = 2, ncol = Nclass)
      }
      for(k in 1:Nclass){
         X = predictors[indClass[[k]], j]
         if( is.factor(X)){

             fitted[[j]][,k] = ## complete
         }
         else{
            fitted[[j]][,k] = ## complete
         }
      }
   }
   return(fitted)
}
```

3.4 Class prediction : posterior probabilities

The prediction step consists in computing the conditional probability of each class given a new input point, using the Bayes formula and the conditional independence assumption on the features. Complete

the code below :

```
  predict_naivebayes <- function(model, xnew, classPrior = NULL,
                                 labels = NULL)
{
   ## model: the output of function train_naivebayes
   ## xnew: the new input points (a matrix with as any rows
#### as test data)
   ## classPrior: the prior probabilitie of each class.
#### if NULL, a uniform prior will be imposed.
   ## labels: the target class labels. If NULL, the class labels
#### contained in the 'model' agument will be used.

   p <- ncol(xnew)
   if(is.null(labels)){
      nclass <- ncol(model[[1]])
      labels <- as.character(1:nclass)
   }
   else{nclass<- length(labels)}
   if(is.null(classPrior)){
      classPrior <- rep(1/nclass, nclass)
     }

   ntest <- nrow(xnew)
   posteriorProb<- matrix(0, nrow = ntest, ncol = nclass,)
   for( k in 1:nclass){
      marglikelihoods= matrix(1,nrow= ntest, ncol=p)
      for(j in 1:p){
         if(is.factor(xnew[,j])){
            inds <- sapply(xnew[,j],
                           function(x){which(x == levels(x))})
            marglikelihoods[,j] <- ## complete
         }
         else{
           marglikelihoods[,j] <-
           ## complete
         }

      }
      posteriorProb[,k] <- classPrior[k] *
         exp(apply(log(marglikelihoods), 1, sum ))

   }
   normalize = apply(posteriorProb,1,sum)
   posteriorProb = posteriorProb/normalize
   ## M/v: each row M[i,] is divided by v[i]
   return(posteriorProb)
}
```

3.5 Testing your model

Try out your functions by adapting the following code chunks to your purposes

```
fitted_naive <- train_naivebayes(Class=Class,
                          predictors = training[,-15],
                          laplace=1)
Pclass <- predict_naivebayes(fitted_naive,xnew = test[,-15],
                       classPrior = classPrior,
                       labels = levels(Class))
mapClass = apply(Pclass, 1, which.max)
mapClass = factor(mapClass, labels=c(" <=50K", " >50K"))
table(mapClass,testClass)
```

Compare with the output of the `e1071` package :

```
 library(e1071)
 m <- naiveBayes(x = training[, 1:14], y=training[ , 15],laplace =1)
 predm <- predict(m, test, type ="class")
 probm <- predict(m, test, type ="raw")
 table(predm, testClass)
```

You should obtain almost the same result. Why not exactly the same ?

```
which(predm !=mapClass)
Pclass[286,]
probm[286,]
test[286,]
```