Machine Learning

Coursework 1

CID 01945214, Imperial College London Spring 2020

1 Question 1

We are interested in a dataset $D = \{X_{1,i}, X_{2,i}, Y_i\}_{i=1}^{100}$. We shuffle the data randomly before starting our study. Consider the following model :

$$Y_i = c + \sum_{j=1}^{2} \left(a_j \cos(X_{j,i}) + \sum_{p=1}^{4} b_{j,p} X_{j,i}^p \right) + \epsilon_i \tag{1.1}$$

where $i = 1, .., n$ with $n = 100$ the number of observations in our data set, and ϵ_i are independent random variables such that $\mathbb{E}(\epsilon_i) = 0$.

1. We start by having a look to the data. A scatter plot of $\mathbf{y}$ against $\mathbf{X_1}$ and $\mathbf{X_2}$ does not reveal any manifest outlier (figure 1). To estimate the error made by our model whose predictors are transformations of $\mathbf{X_1}$ and $\mathbf{X_2}$, we can then choose the sample Mean Squared Error (MSE) defined later in (1.2).

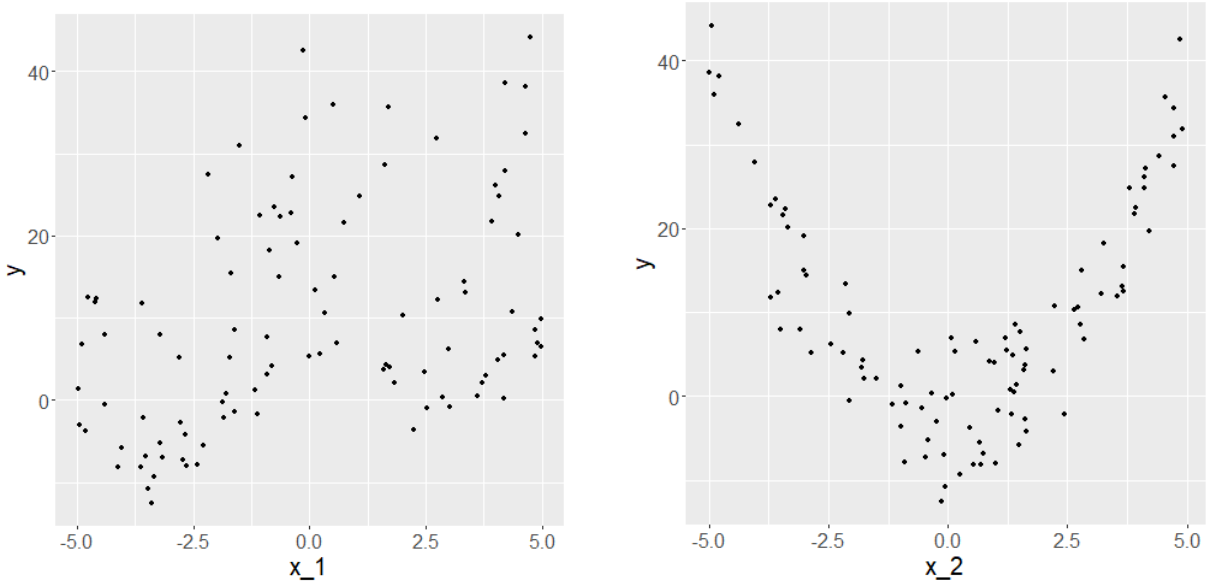

Figure 1: Scatter plot of $\mathbf{y}$ against $\mathbf{X_1}$ (left) and $\mathbf{X_2}$ (right)

To make it more explicit that the model (1.1) is linear, we introduce $\mathbf{Z}_1, .., \mathbf{Z}_{10}$ and $\boldsymbol{\theta} = (\theta_0, .., \theta_{10}) \in \mathbb{R}^{11}$ defined by :

$$\begin{aligned}
& & \theta_0 &= c & \\
Z_{1,i} &= \cos(X_{1,i}) & \theta_1 &= a_1 & \\
Z_{2,i} &= \cos(X_{2,i}) & \theta_2 &= a_2 & \\
Z_{2+p,i} &= X_{1,i}^p & \theta_{2+p} &= b_{1,p} & p = 1, .., 4 \\
Z_{6+p,i} &= X_{2,i}^p & \theta_{6+p} &= b_{2,p} & p = 1, .., 4
\end{aligned}$$

For $i = 1, .., n$. The model (1.1) can be written in function of $\mathbf{Z}_1, .., \mathbf{Z}_{10}$ and the parameter $\boldsymbol{\theta}$:

$$Y_i = \theta_0 + \sum_{j=1}^{10} \theta_j Z_{j,i}$$

This corresponds to a simple linear model in $\mathbf{Z}_1, .., \mathbf{Z}_{10}$. Our linear regressor is :

$$f(\mathbf{z}) = \theta_\mathbf{0} + \sum_{\mathbf{j=1}}^{\mathbf{10}} \theta_\mathbf{j} \mathbf{z_j}$$

Introducing the design matrix $\mathbf{Z} = [\mathbf{1} \quad \mathbf{Z}_1 \quad \dots \quad \mathbf{Z}_{10}]$, the Mean Squared Error (MSE) is defined by :

$$MSE(\boldsymbol{\theta}) = \frac{1}{n}(\mathbf{y} - \mathbf{Z}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{Z}\boldsymbol{\theta}) \tag{1.2}$$

We want to do a Ridge regression, the loss function that we use is :

$$\mathcal{L}(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta}||_2^2 \tag{1.3}$$

where λ is an hyper-parameter and the norm 2 is defined by $||\boldsymbol{\theta}||_2^2 = \theta_0^2 + .. + \theta_{10}^2$. The optimization problem we aim to solve is :

$$\boldsymbol{\theta}^* = arg \max_{\boldsymbol{\theta} \in \mathbb{R}^{11}} \mathcal{L}(\boldsymbol{\theta})$$

We calculate the gradient of the loss function in $\theta \in \mathbb{R}^{11}$:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{2}{n}(-\mathbf{Z}^T\mathbf{y} + \mathbf{Z}^T\mathbf{Z}\boldsymbol{\theta}) + 2\lambda\boldsymbol{\theta} \tag{1.4}$$

Setting $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = 0$, we obtain a local minimum :

$$\boldsymbol{\theta}^* = (\mathbf{Z}^T\mathbf{Z} + n\lambda\mathbf{I}_{11})^{-1}\mathbf{Z}^T\mathbf{y} \tag{1.5}$$

We calculate the Hessian of the loss function :

$$\mathbf{H}(\boldsymbol{\theta}) = \frac{2}{n}\mathbf{Z}^T\mathbf{Z} + 2\lambda\mathbf{I}_{11} \tag{1.6}$$

The eigenvalues of the Hessian are strictly larger than 0. Therefore the Hessian is positive definite and the Ridge optimization problem is convex. The solution $\boldsymbol{\theta}^*$ of the Ridge optimization problem obtained in equation 1.5 is then a global minimum of the loss function $\mathcal{L}(\boldsymbol{\theta})$.

Then, we have to choose the hyper-parameter λ. To do so, we perform a cross-validation with $n\_folds = 10$ folds for a grid of values of λ. We partition the data into 10 folds with 10 observations each. Then, for each fold, a Ridge regression is trained on 90 observations (the 100 observations in the data set minus the 10 observations in the considered fold). The mean squared error is calculated on the 10 observations of the fold.

For each value of λ in the grid, We thus obtain 10 errors (one for each fold). The cross validation error is defined as the mean of these 10 errors. We will select the value of λ that minimizes this cross validation error. We start by trying several values of λ with different scales, and then we "zoom" on the region of interest where we can find a local minimum of the MSE (figure 2). This process gives an approximation of a local minimum of the cross validation error :

$$\hat{\lambda} = 2.5 \times 10^{-3}$$

Now, we can fit a Ridge regression to the whole data set with an hyper-parameter set equal to $\hat{\lambda}$. The inferred parameters calculated with equation 1.5 are reported in table 1.

θ_0^*	θ_1^*	θ_2^*	θ_3^*	θ_4^*	θ_5^*	θ_6^*	θ_7^*	θ_8^*	θ_9^*	θ_{10}^*
c^*	a_1^*	a_2^*	$b_{1,1}^*$	$b_{1,2}^*$	$b_{1,3}^*$	$b_{1,4}^*$	$b_{2,1}^*$	$b_{2,2}^*$	$b_{2,3}^*$	$b_{2,4}^*$
-0.0042	3.9139	0.5118	1.0502	-0.3534	0.0000	0.0148	-0.2767	1.5393	0.0098	-0.0018

Table 1: Inferred parameters using Ridge regression

For new values of X_1 and X_2, we can create a design matrix $\mathbf{Z}_{new}$ as before. Then we can use the learnt parameter $\boldsymbol{\theta}^*$ (table 1) to make predictions at new values :

$$\hat{\mathbf{y}} = \mathbf{Z}_{new}\boldsymbol{\theta}^*$$

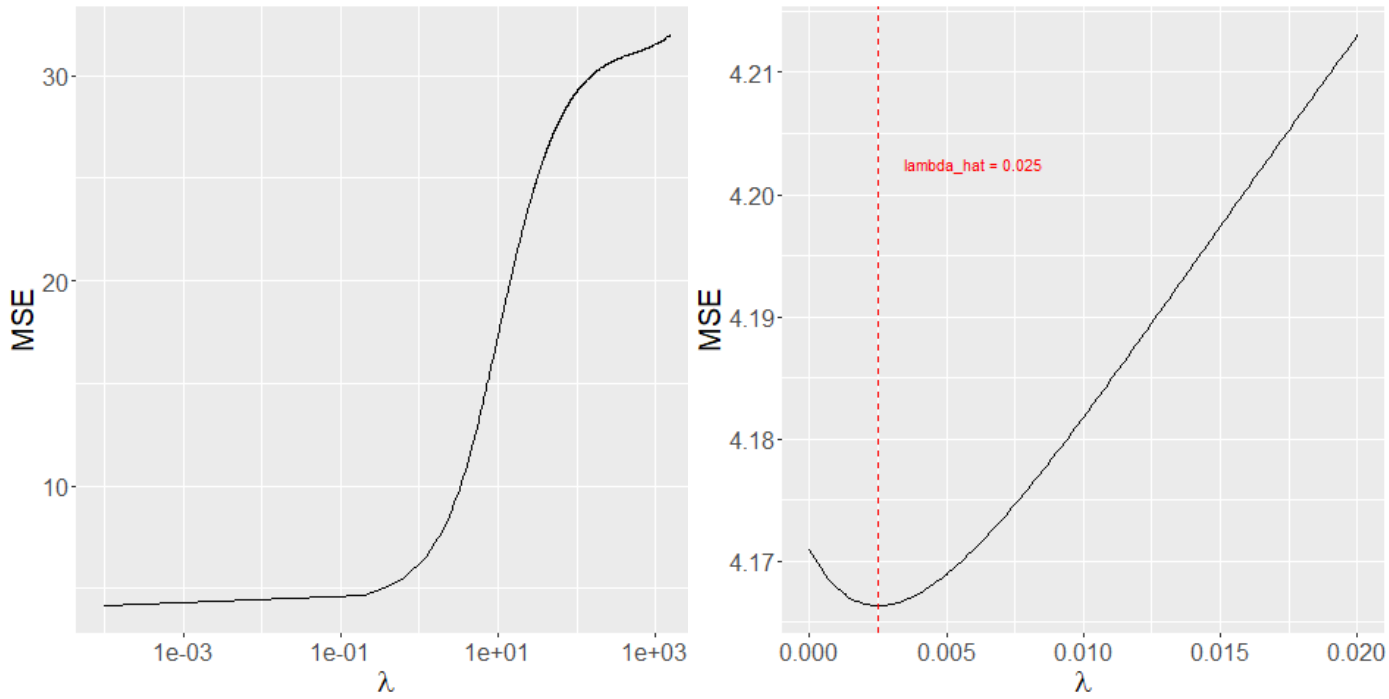

Figure 2: Cross validation mean squared error for different values of λ

The predicted values of Y for four observations using the inferred model are reported in table 2.

X_1	-2.1	2.9	-0.9	3.8
X_2	4.4	-4.5	0.3	3.9
$\hat{y}$	23.13	27.99	1.75	21.01

Table 2: Prediction using Ridge regression

2. Suppose now that $\epsilon \sim \mathcal{N}(0, 4)$. Assuming that the parameters are a priori independent and considering a Gaussian prior distribution with mean 0 and variance α for each parameter, we aim to perform a Bayesian inference of the model parameters.

From the lecture, we know that the joint posterior distribution of the 11 model parameters $(\theta_0, .., \theta_{10})$ is :

$$\boldsymbol{\theta}|\mathbf{Z}, \mathbf{y} \sim \mathcal{N}_\theta(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1.7}$$

Where

$$\boldsymbol{\mu} = \left(\mathbf{Z}^T\mathbf{Z} + \frac{4}{\alpha}\mathbf{I}_{11}\right)^{-1}\mathbf{Z}^T\mathbf{y} \qquad \boldsymbol{\Sigma} = 4\left(\mathbf{Z}^T\mathbf{Z} + \frac{4}{\alpha}\mathbf{I}_{11}\right)^{-1}$$

We plot the posterior mean as a function of α in figure 3. A logarithmic scale is used for α to observe both the behaviour in the neighborhood of 0 and $+\infty$. Below is given an interpretation for this plot using key values of α reported in table 3 :

- When $\alpha \simeq 0$, the covariance matrix of the a priori distribution tends towards $0_{M_{11}(\mathbb{R})}$. Then, the posterior mean tends toward the a priori mean which is $0_{\mathbb{R}^{11}}$. In this case, the observation is almost not informative compared to the a priori.
- When α increases, the observation becomes more informative and has an impact on the posterior mean. For example, when $\alpha = 1$, we observe different values for each parameter $\theta_0, .., \theta_{10}$. As all the parameters have the same a piori distribution, it illustrates the impact of the observations in the parameters inference.

- When $\alpha = \frac{4}{n\lambda} = 16$, the maximum a posteriori estimate is equal to the solution of the Ridge least squares problem. Therefore, we observe that the inferred parameters for this value of λ are exactly the same as the inferred parameters of question 1 reported in table 3.

- When $\alpha \to +\infty$, the maximum a posteriori is equal to the solution of the unregularized simple linear model ($\lambda = 0$ in equation 1.5). the a priori becomes uninformative and the observations determine the maximum a posteriori.

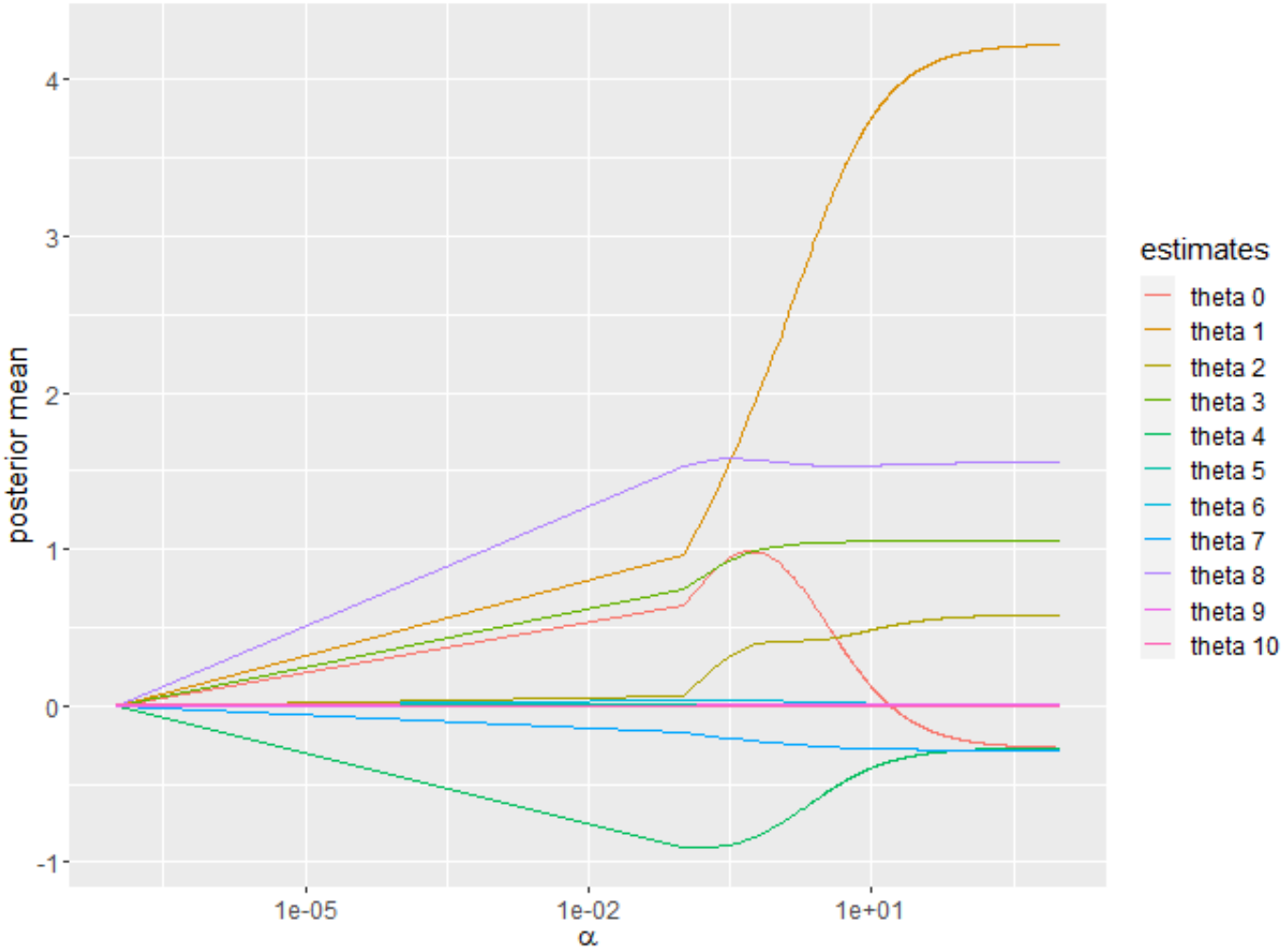

Figure 3: Posterior mean as a function of α

α	θ_0^{MAP}	θ_1^{MAP}	θ_2^{MAP}	θ_3^{MAP}	θ_4^{MAP}	θ_5^{MAP}	θ_6^{MAP}	θ_7^{MAP}	θ_8^{MAP}	θ_9^{MAP}	θ_{10}^{MAP}
10^{-8}	0.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.9202	2.3047	0.4115	1.0174	-0.7634	0.0018	0.0310	-0.2379	1.5582	0.0066	-0.0021
16	-0.0042	3.9139	0.5118	1.0502	-0.3534	0.0000	0.0148	-0.2767	1.5393	0.0098	-0.0018
10^8	-0.2722	4.2292	0.5826	1.0512	-0.2670	-0.0000	0.0114	-0.2818	1.5538	0.0103	-0.0024

Table 3: Maximum A Posteriori (MAP) for different values of α

From the lecture, we know that the predictive posterior distribution of a new set of observations $\mathbf{Z}_{new}$ is :

$$\mathbf{y}_{new}|\mathbf{Z}, \mathbf{y} \sim \mathcal{N}(\mathbf{Z}_{new}^T \boldsymbol{\mu}, \mathbf{Z}_{new}^T \boldsymbol{\Sigma} \mathbf{Z}_{new} + \sigma^2 \mathbf{I}_4)$$

Where $\mathbf{Z}_{new}$ is the design matrix associated to the new observations, created like in question 1.1.

We compute :

$$\mathbf{Z}_{new}^T\boldsymbol{\mu} = \begin{pmatrix} 25.8098 \\ 29.3381 \\ -0.5020 \\ 20.9535 \end{pmatrix} \quad \mathbf{Z}_{new}^T\boldsymbol{\Sigma}\mathbf{Z}_{new} + \sigma^2\mathbf{I}_4 = \begin{pmatrix} 4.3906 & -0.0058 & -0.0147 & 0.1290 \\ -0.0058 & 4.7578 & -0.0461 & 0.0869 \\ -0.0147 & -0.0461 & 4.2454 & -0.0827 \\ 0.1290 & 0.0869 & -0.0827 & 4.3222 \end{pmatrix}$$

2 Question 2

Consider a binary classification problem where, under class $\mathcal{C}_1$, $\mathbf{X}$ is uniformly distributed on the p-dimensional unit cube $[0,1]^p$ while, under class $\mathcal{C}_2$, $\mathbf{X}$ is uniformly distributed on the p-dimensional cube $[0,1+\epsilon]^p$ with $p > 0$. We denote Y the response random variable of the classifier. We denote f the density of $\mathbf{X}$ and π the mass density of Y. We assume equal prior probabilities for the two classes, i.e $\pi(y = C_i) = 1/2$, $i = 1, 2$.

1. (a) Let consider a Bayes classifier that chooses the class with the greatest probability given the observation $\mathbf{x}$:

$$arg\max_{1,2} \pi(Y = C_i|\mathbf{x}) \tag{2.1}$$

It means that, given an observation $\mathbf{x}$, the Bayes classifier predicts the class $\hat{y}$ such that :

$$\hat{y} = \begin{cases} \mathcal{C}_2 & \text{if } \pi(Y = \mathcal{C}_2|\mathbf{x}) > \pi(Y = \mathcal{C}_1|\mathbf{x}) \\ \mathcal{C}_1 & \text{otherwise} \end{cases} \tag{2.2}$$

From Bayes theorem, it follows that, for $i = 1, 2$:

$$\pi(Y = C_i|\mathbf{x}) \propto f(\mathbf{x}|Y = C_i)\pi(Y = C_i) \tag{2.3}$$

The decision rule 2.2 can then be written :

$$\hat{y} = \begin{cases} \mathcal{C}_2 & \text{if } f(\mathbf{x}|Y = \mathcal{C}_2)\pi(Y = \mathcal{C}_2) > f(\mathbf{x}|Y = \mathcal{C}_1)\pi(Y = \mathcal{C}_1) \\ \mathcal{C}_1 & \text{otherwise} \end{cases} \tag{2.4}$$

With uniform prior π, it becomes :

$$\hat{y} = \begin{cases} \mathcal{C}_2 & \text{if } f(\mathbf{x}|Y = \mathcal{C}_2) > f(\mathbf{x}|Y = \mathcal{C}_1) \\ \mathcal{C}_1 & \text{otherwise} \end{cases} \tag{2.5}$$

We know that $f(\mathbf{x}|Y = \mathcal{C}_2) = \left(\frac{1}{1+\epsilon}\right)^p \mathbb{1}_{[0,1+\epsilon]^p}(\mathbf{x})$ and $f(\mathbf{x}|Y = \mathcal{C}_1) = \mathbb{1}_{[0,1]^p}(\mathbf{x})$. Therefore, as $\epsilon > 0$ and $p \geq 1$, we have $\left(\frac{1}{1+\epsilon}\right)^p < 1$ and then $f(\mathbf{x}|Y = \mathcal{C}_2) > f(\mathbf{x}|Y = \mathcal{C}_1)$ if and only if $\mathbf{x} \in [0,1+\epsilon]^p \setminus [0,1]^p$ or equivalently if and only if $\exists i = 1, .., p$, $x_i \in (1, 1+\epsilon]$. The rule followed by the Bayes classifier is then :

$$\hat{y} = \begin{cases} \mathcal{C}_2 & \text{if } \exists i = 1, .., p,\ x_i \in (1, 1+\epsilon] \\ \mathcal{C}_1 & \text{otherwise} \end{cases} \tag{2.6}$$

(b) The Bayes error rate for the Bayes classifier is defined by :

$$B = 1 - \mathbb{E}\left\{\max_{i=1,2} \pi(Y = C_i|\mathbf{X})\right\} \tag{2.7}$$

Following what we did in part (a), we have :

$$\mathbb{E}\left\{\max_{i=1,2} \pi(Y = C_i|\mathbf{X})\right\} = \begin{cases} \mathbb{E}\left\{\pi(Y = \mathcal{C}_2|\mathbf{X})\right\} & \text{if } \mathbf{X} \in [0,1+\epsilon]^p \setminus [0,1]^p \\ \mathbb{E}\left\{\pi(Y = \mathcal{C}_1|\mathbf{X})\right\} & \text{otherwise} \end{cases} \tag{2.8}$$

Therefore, the Bayes error rate can be written :

$$B = 1 - \mathbb{E}\left\{\pi(Y = \mathcal{C}_1|\mathbf{X})\mathbb{1}_{[0,1]^p}(\mathbf{X})\right\} - \mathbb{E}\left\{\pi(Y = \mathcal{C}_2|\mathbf{X})\mathbb{1}_{[0,1+\epsilon]^p \setminus [0,1]^p}(\mathbf{X})\right\} \tag{2.9}$$

As the marginal density of X is strictly positive on $[0, 1+\epsilon]^p$, we can use Bayes theorem to compute :

$$\begin{aligned}
\mathbb{E}\left\{\pi(Y=\mathcal{C}_2|\mathbf{X})\mathbb{1}_{[0,1+\epsilon]^p\setminus[0,1]^p}(\mathbf{X})\right\} &= \int_{[0,1+\epsilon]^p\setminus[0,1]^p} \pi(Y=\mathcal{C}_2|\mathbf{x})f(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int_{[0,1+\epsilon]^p\setminus[0,1]^p} \frac{\pi(Y=\mathcal{C}_2)f(\mathbf{x}|Y=\mathcal{C}_2)}{f(\mathbf{x})} f(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= 1/2 \int_{[0,1+\epsilon]^p\setminus[0,1]^p} \frac{1}{(1+\epsilon)^p}\mathrm{d}\mathbf{x} \\
&= 1/2 \left[1 - \int_{[0,1]^p} \frac{1}{(1+\epsilon)^p}\mathrm{d}\mathbf{x}\right] \\
&= 1/2 \left[1 - \frac{1}{(1+\epsilon)^p}\right]
\end{aligned}$$

Similarly we compute :

$$\begin{aligned}
\mathbb{E}\left\{\pi(Y=\mathcal{C}_1|\mathbf{X})\mathbb{1}_{[0,1]^p}(\mathbf{X})\right\} &= \int_{[0,1]^p} \pi(Y=\mathcal{C}_1|\mathbf{x})f(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int_{[0,1]^p} \frac{\pi(Y=\mathcal{C}_1)f(\mathbf{x}|Y=\mathcal{C}_1)}{f(\mathbf{x})} f(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= 1/2 \int_{[0,1]^p} \mathrm{d}\mathbf{x} \\
&= 1/2
\end{aligned}$$

Therefore, plug-in the last two results in 2.7 gives the following Bayes error rate for the Bayes classifier :

$$B(\epsilon, p) = \frac{1}{2(1+\epsilon)^p} \tag{2.10}$$

We plot the Bayes error rate for $\epsilon = 0.1, 1$ and 2, results are presented in figure 4.

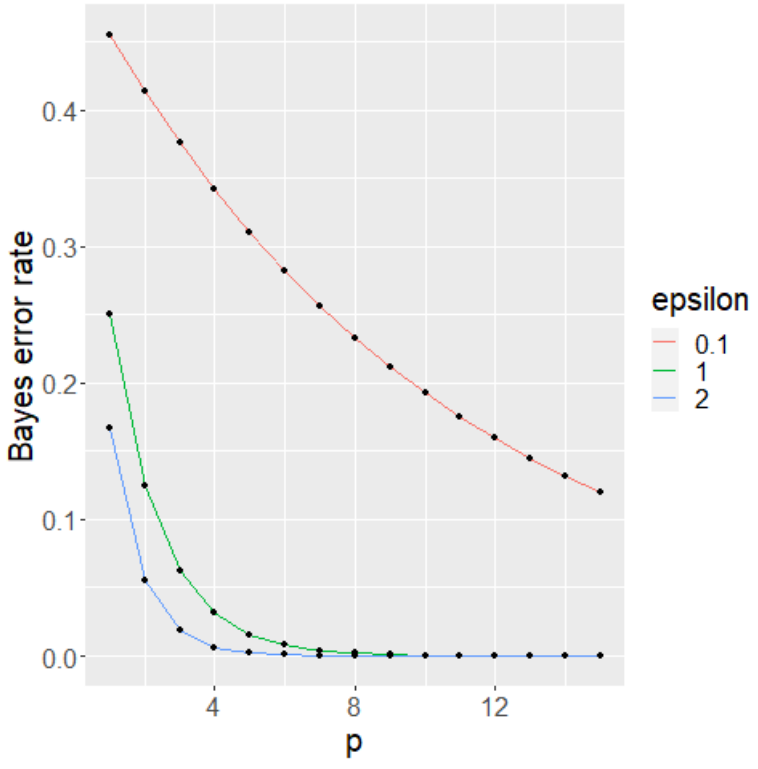

Figure 4: Bayes error rate as a function of the dimension p for $\epsilon = 0.1, 1$ and 2

Two important trends are noticed on figure 4 : for a given ϵ the Bayes error rate decreases when p increases, and for a given p the Bayes error rate decreases when ϵ increases.

For given $\epsilon > 0$, $p \geq 1$, the zone of uncertain prediction is the p-dimensional unit cube $[0,1]^p$. Here, "uncertain" means that an observation in $[0,1]^p$ can belong to each classes (with strictly positive probability).

As we saw before, the predicted class for an observation in $[0,1]^p$ is $\mathcal{C}_1$ (Equation 2.6). Therefore, the errors made by our classifier concern the observations of $\mathcal{C}_2$ in $[0,1]^p$. We calculate the probability that an observation $\mathbf{X}$ of $\mathcal{C}_2$ belongs to $[0,1]^p$:

$$\begin{aligned}\mathbb{P}(\mathbf{X} \in [0,1]^p | Y = \mathcal{C}_2) &= \int_{[0,1]^p} \frac{1}{(1+\epsilon)^p} \mathrm{d}\mathbf{x} \\ &= \frac{1}{(1+\epsilon)^p}\end{aligned}$$

For a given $\epsilon > 0$, $\frac{1}{(1+\epsilon)^p}$ is a strictly decreasing function of p. Then, when the dimension p increases, the probability $\mathbb{P}(\mathbf{X} \in [0,1]^p | Y = \mathcal{C}_2)$ decreases. In other terms, the probability that an observation of class $\mathcal{C}_2$ belongs to the zone of uncertain prediction decreases when p increases. Consequently, our classifier will make less prediction mistakes (on average) in higher dimension p.

Recall that the Bayes error rate is interpreted as the lowest possible test error rate of the Bayes classifier. Therefore, the previous argument shows that the Bayes error rate decreases when p increases for a given ϵ. We can do the same reasoning if we fix $p > 0$ and we increase ϵ. Indeed, for a given p, $\frac{1}{(1+\epsilon)^p}$ is a strictly decreasing function of ϵ, and the previous arguments still hold. It shows that the Bayes error rate decreases when ϵ increases for a given p.

2. We generate a random dataset of $n = 1000$ observations for $p = 4$ and $\epsilon = 0.5$. To do so, we draw the class Y with the uniform a priori distribution and then we draw $\mathbf{X}$ conditioning on Y. Repeating the process 1000 times gives a sample with independent observation. This dataset will be used for training, validation and testing. The error rate of our model is defined as the ratio $\frac{\text{number of observations misclassified}}{\text{total number of observations}}$.

(a) We want to use cross-validation to train a logistic regression and a QDA classifiers. We partition the training data in n_folds (approximately) equally sized folds. Then, for each fold, a model is trained on the observations not in the fold and evaluated on the observations in the fold. With this process, we obtain a set of errors $E = (e_1, .., e_{n\_folds})$ where e_i is the error of the model evaluated on the i^{th} fold. We define the cross validation error as the average $\bar{E}$ of all these errors. A pseudo-code for the cross validation procedure of an arbitrary model is given in Algorithm 1.

Algorithm 1 Cross Validation for a given model

input: n_folds number of folds, training data $data\_train$, a model $model$ to fit
$CV\_errors = list()$
Partition $data\_train$ in n_folds (approximately) equally sized folds and store each fold in a list $folds$
For $k = 1..n\_folds$ do :
    $fold = folds[k]$
    $data\_train\_k = data\_train[-fold]$
    $fit(model, data\_train\_k)$
    $CV\_errors.append(error(model, fold))$
$CV\_error = mean(CV\_errors)$
$\sigma\_CV = sd(CV\_errors)$
return CV_error, σ_CV

To train the logistic regression and QDA classifier with cross-validation in R, I use the functions *trainControl* and *train* from the package *caret*. We choose a number of folds equal to 10. The cross validation error rate and the standard deviation of the errors $\sigma^{CV}_{errors} := \sigma(E)$ are reported in table 4.

	CV error	σ^{CV}_{errors}
Logistic regression	26.40 %	2.95 %
QDA classifier	17.90 %	5.47 %

Table 4: Cross Validation error and standard deviation of the errors calculated on each fold for logistic regression and QDA classifier

(b) We want to implement a k nearest neighbours classifier. To do so, we need to find a number of neighbours k which obtains good performance. If k is too small, the classifier will overfit the training data and will have poor generalization performance. If k is too high, the model is biased and will also obtain bad performances on unseen data. We use a nested cross-validation to pick an appropriate hyper-parameter k.

The idea is to perform an inner cross validation to select an hyper-parameter $\hat{k}$ which obtains good performance. To reduce the risk of overfitting, this inner cross-validation is nested inside an outer cross validation that evaluates the performance of the classifier with hyper-parameter set equal to $\hat{k}$. In the following, we say that a set of observations is labelled if we "know" the labels of the observations. A labelled set is used by a k nearest neighbours classifier to predict the labels of unseen observations.

We partition the training data in n_folds_outer (approximately) equally sized folds. For each fold, the data set minus the fold will be a labelled subset for the outer cross validation. We called these sets the *outer_subset*s. Then, we partition each of these *outer_subset*s in n_folds_inner (approximately) equally sized folds. For each fold, the *outer_subset* minus the fold will be a labelled subset for the inner cross validation. We call these sets the *inner_subset*s.

We perform a grid search for the parameter k in the inner cross-validation. For several numbers of neighbours k, a k nearest neighbours classifier is used to predict the observations in the inner folds from the *inner_subset*s. For each value of k, we obtain a set of inner errors $E^k_{inner} = (e^k_1, .., e^k_{n\_folds\_inner})$. Taking the mean of them gives an average inner cross-validation error $E^k_{CV\_inner}$. We then select the parameter $\hat{k}$ that minimizes this average cross-validation error :

$$\hat{k} = arg \max_{k \in list\_k} E^k_{inner} \tag{2.11}$$

Afterward, we set the number of neighbours equal to $\hat{k}$ and we evaluate the performance of the classifier in the outer cross validation. We obtain a set of errors $E^{\hat{k}}_{outer} = (e^{\hat{k}}_1, .., e^{\hat{k}}_{n\_folds\_outer})$ where $e^{\hat{k}}_i$ is the error of the model evaluated on the i^{th} outer fold. We define the nested cross validation error as the average $\bar{E}^{\hat{k}}_{outer}$ of these errors.

A pseudo-code for the cross validation of an arbitrary model is given in Algorithm 2.
To implement the k nearest neighbours classifier in R, I compute myself the nested cross validation and I use the functions *trainControl* and *train* from the package *caret* to predict with the k-nn classifier. We choose a number of outer folds equal to 10 and a number of inner folds equal to 5. The cross validation error rate and the standard deviation of the outer errors $\sigma^{CV}_{errors} := \sigma(E^{\hat{k}}_{outer})$ are reported in table 4. The hyper-parameter selected by the nested CV is $\hat{k} = 10$.

	CV error rate	σ^{CV}_{errors}	$\hat{k}$
k-nn	17.8 %	2.63 %	11

Table 5: Cross Validation error rate and standard deviation of the errors calculated on each fold for logistic regression and QDA classifier

Algorithm 2 nested Cross Validation

input: n_folds_outer number of folds in the outer CV, n_folds_inner number of folds in the inner CV, training data $data\_train$, a model $model$ to fit, a list of hyper-parameters values $list\_k$
$CV\_errors\_outer = list()$
Partition $data\_train$ in n_folds_outer (approximately) equally sized folds and store each fold in a list $folds\_outer$
For $i = 1..n\_folds\_outer$ do :
  $fold\_outer\_i = folds\_outer[i]$
  $data\_train\_outer\_i = data\_train[-fold\_outer\_i]$
  $CV\_errors\_k = list()$
  Partition $data\_train$ in n_folds_inner (approximately) equally sized folds and store
  each fold in a list $folds\_inner$
  For k in $list\_k$ do :
    $CV\_errors\_k\_inner = list()$
    For $j = 1..n\_folds\_inner$ do :
      $data\_train\_inner\_j = data\_train\_i[-fold\_inner]$
      $fit(model(k), data\_train\_inner\_j)$
      $CV\_errors\_k\_inner.append(error(model(k), data\_train\_inner\_j))$
    $CV\_errors\_k.append(mean(CV\_errors\_k\_inner))$
  $\hat{k} = arg \min_{k \in list\_k} CV\_errors\_k$
  $fit(model(\hat{k}), data\_train\_i)$
  $CV\_errors\_outer.append(error(model(\hat{k}, fold\_outer\_i)))$
$nested\_CV\_error = mean(CV\_errors\_outer)$
$nested\_\sigma\_CV = sd(CV\_errors\_outer)$
return $nested\_CV\_error$, $nested\_\sigma\_CV$

(c) We want to compare the three classifiers (logistic, QDA and k nearest neighbours). The criteria that we use for the comparison are the performance, the variability and the computational cost. The figures used for the comparison are summarised in table 2.

- In terms of performance, the k nearest neighbours obtains slightly lower average error rate than QDA classifier. Both of k nearest neighbours and QDA classifier obtain much lower average error rate than logistic regression.

- To test the variability of the algorithms, we have computed the standard deviation σ^{CV}_{errors} of the errors inside the cross validation procedure. It is observed that the QDA classifier has an higher σ^{CV}_{errors} than the k nearest neighbours classifier and the logistic regression. The performances of QDA classifier are more variable than those of QDA classifier or logistic regression.

- To test the computational cost of our 3 algorithms, we computed the time for one cross-validation procedure thanks to the package *tictoc*. It is striking that the k nearest neighbours needs much more time for a cross-validation procedure. With only (well chosen) 6 different values of hyper-parameter k to try and 1000 observations, the k nearest neighbours nested cross validation procedure lasts more than one minute, whereas the two other algorithms need less than one second for one cross validation procedure. This can be explained by the fact that the nested cross-validation procedure described in algorithm 2 has a temporal complexity in $O(n\_folds\_outer \times K \times n\_folds\_inner)$ where K is the number of hyper-parameter values k to try. On the other hand, the classic cross validation procedure described in algorithm 1 and used for QDA and Logistic regression has a temporal complexity in $O(n\_folds)$.

Among the three classifiers, I would recommend to use the k nearest neighbours classifier. On one hand, the logistic regression obtains much poorer performances and thus is not really relevant as a predictive model used alone (nevertheless, it could be used for a bagging procedure). On the other hand, the k nearest neighbours classifier obtains slightly better performances than QDA classifier and has a much lower variability.

One may be concerned regarding the computational cost of the k nearest neighbours. Nevertheless, once the cross validation procedure is done to select k, the prediction is very fast because we have only 1000 observations.

	CV error	σ^{CV}_{errors}	CV time (s)
Logistic regression	26.40 %	2.95 %	0.67
QDA classifier	17.90 %	5.47 %	0.58
k-nn	17.80 %	2.63 %	93.44

Table 6: Cross Validation error, standard deviation of the errors calculated on each fold and Cross Validation time for logistic regression and QDA classifier

(d) We want to use the Bayes classifier whose decision rule is given in equation 2.2. We separate the data set in n_folds folds, and we compute the error rate of the Bayes classifier in each fold. It gives us a set of errors $E_{Bayes} = (e_1, .., e_{n\_folds})$ where e_i is the error of the Bayes classifier on the i^{th} fold. We define the cross validation error rate of the Bayes classifier as the average $\bar{E}_{Bayes}$ of these errors. The cross validation error rate and the standard deviation of the errors $\sigma^{CV}_{errors} := \sigma(E_{Bayes})$ are reported in table 2.

	CV error	σ^{CV}_{errors}
Bayes classifier	10.30 %	2.53 %

Table 7: Error rate of logistic regression and QDA classifier

We can now complete the table 2 with the results obtained by our Bayes classifier. It can be observed in table 2 that the Bayes classifier obtains much lower cross validation error rate. It also has a lower variability and a lower cross validation procedure time than the other classifiers. For these reasons, the Bayes classifier would be our final recommended classifier in this problem.

	CV error	σ^{CV}_{errors}	CV time (s)
Logistic regression	26.40 %	2.95 %	0.67
QDA classifier	17.90 %	5.47 %	0.58
k-nn	17.80 %	2.63 %	93.44
Bayes classifier	10.30 %	2.53 %	0.07

Table 8: Cross Validation error, standard deviation of the errors calculated on each fold and Cross Validation time for logistic regression, QDA classifier, k-nn and Bayes classifier

The error rate of our Bayes classifier and the theoretical Bayes error (equation 2.10) are reported in table 2. It can be observed that our Bayes classifier obtains an error rate very close to the theoretical Bayes error rate.

Our Bayes classifier error rate would converge to the Bayes error rate if the number of observations in the training data set tends towards $+\infty$. The fact that we only evaluate the Bayes classifier on $n = 1000$ observations explains why we observe an error rate slightly different than the theoretical Bayes error rate.

Bayes error rate	9.88 %
Bayes classifier error rate	10.30 %

Table 9: Theoretical Bayes error rate (2.10) and error rate of Bayes classifier for $\epsilon = 0.5$, $p = 4$

A R code - Question 1

```
###   MATH97131- Machine Learning   ###

# Coursework 1 - Spring 2021 #


rm(list = objects())
par(mfrow = c(1,1))
setwd("C:/Users/robin/Dropbox/Applications/Overleaf/Coursework_1_ML")

library(xtable)
library(ggplot2)
library(tidyverse)

data = read.csv("01945214.csv")
head(data)
dim(data)

# shuffle data

set.seed(42)
NumOfDataPairs = dim(data)[1]
s = sample.int(NumOfDataPairs)
data_shuffled = data[s,]


# creation design matrix

y = data$y
x_1 = data$X.1
x_2 = data$X.2

png(filename = "y_x_1.png")
g = ggplot(data = data)
g = g + geom_point(aes(x = x_1, y = y)) + theme(text = element_text(size=20))
g
graphics.off()

png(filename = "y_x_2.png")
g = ggplot(data = data)
g = g + geom_point(aes(x = x_2, y = y)) + theme(text = element_text(size=20))
g
graphics.off()

X = cbind(1, cos(x_1), cos(x_2))

PolyOrder = 4

for (p in 1:PolyOrder){
  X = cbind(X, x_1^p)
}

for (p in 1:PolyOrder){
  X = cbind(X, x_2^p)
```

```
}

# Compute average MSE using a k folds CV procedure for a given lambda.

k_cv_lambda = function(X,y,k=10 ,lambda=0){
  # X is the matrix of features (including cosinus and polynomials)
  # y is a vector containing the targets
  # Lambda is the parameter for ridge regression

  NumOfDataPairs = length(y)
  size_test_set = NumOfDataPairs %/% k

  # Initialize CV variable for storing results
  CV = matrix(nrow=k, ncol=1)

  for (i in 0:(k-1)){

    s = seq(i * size_test_set + 1, size_test_set * (i+1))

    # Create training design matrix and target data, leaving one out each time
    Train_X = X[-s, ]
    Train_y = y[-s]

    # Create testing design matrix and target data
    Test_X = X[s, ]
    Test_y = y[s]

    # Learn the optimal parameters using MSE loss

    Paras_hat = solve( t(Train_X) %*% Train_X + NumOfDataPairs * lambda*diag(dim(X)[2
    Pred_y = Test_X %*% Paras_hat

    # Calculate the MSE of prediction using training data
    CV[i+1] = mean((Pred_y - Test_y)^2)

  }
  return(mean(CV))
}


## Find best parameter lambda

png(filename = "MSE_lambda_dezoom.png")
lambdalist = seq(0.0001,1000,by=0.1)
MSE_lambda = sapply(lambdalist, function(lambda) k_cv_lambda(X,y,k=10,lambda=lambda))
data_lambda = data.frame(lambda = lambdalist, mse = MSE_lambda)
g = ggplot(data_lambda, aes(x = lambda, y = mse)) + geom_line()
g = g + scale_x_log10() + theme(legend.position="right", text = element_text(size=20
g
dev.off()

png(filename = "MSE_lambda_zoom.png")
lambdalist_2 = seq(0,0.02,by=0.0001)
MSE_lambda = sapply(lambdalist_2, function(lambda) k_cv_lambda(X,y,k=10,lambda=lambda
data_lambda = data.frame(lambda = lambdalist_2, mse = MSE_lambda)
lambda <-lambdalist_2[which.min(MSE_lambda)]
```

```
g = ggplot(data_lambda, aes(x = lambda, y = mse)) + geom_line()
g = g + theme(legend.position="right", text = element_text(size=20)) + xlab(expressio
g = g + geom_vline(xintercept = lambda, color = 'red', lty = 2)
g = g + annotate(geom="text", x=0.005, y=4.2025, label= "lambda_hat_=_0.025",
         color="red")
g
dev.off()

# predictions for new observations

Paras_hat_lambda = solve( t(X) %*% X + NumOfDataPairs * lambda * diag(dim(X)[2]) ) %
xtable(round(t(Paras_hat_lambda),digits = 4), digits = 4)

X1 = c(-2.1, 2.9, -0.9, 3.8)
X2 = c(4.4, -4.5, 0.3, 3.9)

new_X = cbind(1, cos(X1), cos(X2))

PolyOrder = 4

for (p in 1:PolyOrder){
  new_X = cbind(new_X, X1^p)
}

for (p in 1:PolyOrder){
  new_X = cbind(new_X, X2^p)
}

Pred_y = new_X %*% Paras_hat_lambda
xtable(round(t(Pred_y), digits = 4))


# 2

sigma = 2
posterior_mean = function(alpha,X,y){
  gram = t(X) %*% X
  d = dim(gram)[1]
  I = diag(rep(1,d))
  return (solve(gram + (sigma^2/alpha) * I) %*% t(X) %*% y)
}

posterior_mean(alpha = 2,X,y)

png(filename = "plot_post_mean.png", width = 650)

list_alpha = seq(0.0000001,1000,by=0.1)

post_means = sapply(1:11, function(i) sapply(list_alpha, function(alpha) posterior_m

dat = data.frame(alpha = list_alpha, post_means)
colnames(dat) = c("alpha", paste("theta",letters[1:11]))
dat = dat %>%  gather(key = "estimates", value = "value", -alpha)

g = ggplot(dat, aes(x = alpha, y = value)) + geom_line(aes(color = estimates))
g = g + scale_x_log10() + theme(legend.position="right", text = element_text(size=15
```

```
g = g + scale_color_discrete(labels = paste("theta",0:10))
g

dev.off()

posterior_mean(alpha=1,X,y)

alpha_ult = sigma^2/(lambda * NumOfDataPairs)

table = data.frame(
Paras_hat_lambda,
posterior_mean(alpha=0.0000000001,X,y),
posterior_mean(alpha=1,X,y),
posterior_mean(alpha=alpha_ult,X,y),
posterior_mean(alpha=100000000,X,y)
)

xtable(t(table), digits = 4)

mylm = lm(y~X-1)
mylm$coefficients

# predictions alpha = 1

alpha = 1
mu = solve(t(X) %*% X + (sigma^2/alpha) * NumOfDataPairs * diag(dim(X)[2])) %*% t(X)
Eps = sigma^2 * solve(t(X) %*% X + (sigma^2/alpha) * diag(dim(X)[2]))

mean_pred = new_X %*% mu
sd_pred = new_X %*% Eps %*% t(new_X) + sigma^2 * diag(4)

round(mean_pred, digits = 4)
round(sd_pred, digits = 4)
```

B R code - Question 2

```
###   MATH97131- Machine Learning   ###

# Coursework 1 - Spring 2021 #

rm(list = objects())
par(mfrow = c(1,1))
setwd("C:/Users/robin/Dropbox/Applications/Overleaf/Coursework 1 ML")

# 1

list_eps = c(0.1, 1, 2)
list_p = seq(1,15,by=1)
ord_eps_1 = 1/(2*(1+list_eps[1])^list_p)

png(filename = "Bayes_error_rate.png")

plot(list_p,
     ord_eps_1,
     type='b',
     ylim = c(0,0.5),
     lwd = 2,
     xlab = "p",
     ylab = "Bayes Error Rate")


for (i in 2:length(list_eps)){
  ord_eps_i = 1/(2*(1+list_eps[i])^list_p)
  lines(list_p, ord_eps_i, col=i, type = 'b', lwd = 2)
}

legend('topright',
       legend = c("epsilon = 0.1", "epsilon = 1", "epsilon = 2"),
       col=1:3,
       lwd=2,
       cex=1.3)

dev.off()


# 2

set.seed(01945214)

n = 1000
p = 4
eps = 0.5

x = matrix(nrow = n,ncol = p)
y = c()

for (i in 1:n){
  u = runif(1)
  if (u < 0.5){
    y = c(y,1)
```

```
    x[i,] = runif(p)
  }
  if (u > 0.5){
    y = c(y,2)
    x[i,] = runif(p, min = 0, max = 1 + eps)
  }
}

data = data.frame(x,y = as.factor(y))
head(data)
dim(data)

s = 1:800
data_train = data[s,]
data_test = data[-s,]


##   Part a   ##

library(caret)
library(tictoc)

# training logistic regression

tic()

train_control_log_reg = trainControl(method = "cv", number = 10)
log_reg = train(y ~ .,
                data = data,
                trControl = train_control_log_reg,
                method = "glm",
                family=binomial())
toc()

log_reg$results$Accuracy


log_reg_shuffle = function(data){
  rows = sample(nrow(data))
  data_shuffle = data[rows,]
  return( train(y ~ .,
                data = data,
                trControl = train_control_log_reg,
                method = "glm",
                family=binomial())$results$Accuracy )
}

variability_log_reg = replicate(10, log_reg_shuffle(data))

round(1-mean(variability_log_reg), digits = 4)
round(1-min(variability_log_reg), digits = 4)
round(1-max(variability_log_reg), digits = 4)
round(sd(variability_log_reg), digits = 6)


# training qda
```

```
tic()
train_control_qda = trainControl(method = "cv", number = 10)
qda_clf = train(y ~ .,
                data = data,
                trControl = train_control_qda,
                method = "qda")
toc()

qda_clf$results$Accuracy

qda_shuffle = function(data){
  rows = sample(nrow(data))
  data_shuffle = data[rows,]
  return( train(y ~ .,
                data = data,
                trControl = train_control_qda,
                method = "qda")$results$Accuracy )
}

variability_qda = replicate(10, qda_shuffle(data))

round(1-mean(variability_qda), digits = 4)
round(1-min(variability_qda), digits = 4)
round(1-max(variability_qda), digits = 4)
round(sd(variability_qda), digits = 6)


##   Part b   ##

# function to create the train/test sets for outer CV

create_outer_CV_sets = function(X, y, n_folds = 10){

  # create subsets for outer cross-validation

  NumOfDataPairs = length(y)
  size_test_set = NumOfDataPairs %/% n_folds

  outer_CV_train_sets = list()
  outer_CV_test_sets = list()
  outer_CV_train_sets_label = list()
  outer_CV_test_sets_label = list()

  for (i in 0:(n_folds-1)){

    s = seq(i * size_test_set + 1, size_test_set * (i+1))

    # Create training design matrix and target data
    X_train = X[-s, ]
    y_train = y[-s]

    # Create testing design matrix and target data
    X_test = X[s, ]
    y_test = y[s]
```

```
    outer_CV_train_sets[[i+1]] = X_train
    outer_CV_train_sets_label[[i+1]] = y_train

    outer_CV_test_sets[[i+1]] = X_test
    outer_CV_test_sets_label[[i+1]] = y_test

  }
  return(list(outer_CV_train_sets, outer_CV_train_sets_label, outer_CV_test_sets, ou
}


# Nested CV for knn

k_nn_cv <- function(x, y, n_folds_outer = 10, n_folds_inner = 5, list_k = 1:20){
  # X is the matrix of features
  # y is a vector containing the targets

  outer_CV_train_sets = create_outer_CV_sets(x, y, n_folds = n_folds_outer)[[1]]
  outer_CV_train_sets_label = create_outer_CV_sets(x, y, n_folds = n_folds_outer)[[2]

  outer_CV_test_sets = create_outer_CV_sets(x, y, n_folds_outer)[[3]]
  outer_CV_test_sets_label = create_outer_CV_sets(x, y, n_folds_outer)[[4]]

  cv_k = c()

  for (k in list_k){

    cv_errors_k = c()

    for (i in 1:length(outer_CV_train_sets)){

      inner_CV_train_set = outer_CV_train_sets[[i]]
      inner_CV_train_set_label = outer_CV_train_sets_label[[i]]
      inner_CV_test_set = outer_CV_test_sets_label[[i]]

      inner_data = data.frame(x_inner = inner_CV_train_set, y_inner = as.factor(inner

      trControl = trainControl(method  = "cv",
                               number  = n_folds_inner)

      fit_knn_k = train(y_inner ~ .,
                  method     = "knn",
                  tuneGrid = data.frame(k = k),
                  trControl  = trControl,
                  metric     = "Accuracy",
                  data       = inner_data)

      cv_errors_k = c(cv_errors_k, fit_knn_k$results$Accuracy)

    }

    cv_k = c(cv_k, mean(cv_errors_k))
  }

  best_k = list_k[which.max(cv_k)]
```

```
  outer_data = data.frame(x_outer = x, y_outer = as.factor(y))

  trControl = trainControl(method  = "cv",
                           number  = n_folds_outer)

  fit_knn_best_k = train(y_outer ~ .,
                method     = "knn",
                tuneGrid = data.frame(k = best_k),
                trControl  = trControl,
                metric     = "Accuracy",
                data       = outer_data)

  return(list(fit_knn_best_k,best_k))
}

tic()
fitted_best_knn = k_nn_cv(x, y)
toc()

knn_shuffle = function(data){
  rows = sample(nrow(data))
  data_shuffle = data[rows,]
  x = data_shuffle[,1:4]
  y = data_shuffle$y
  return( k_nn_cv(x, y,list_k = 9:15) )
}

variability_knn = replicate(10, knn_shuffle(data))

#variability_knn = replicate(10, k_nn_cv(x, y,list_k = 9:15))

variability_knn_accuracy = sapply(variability_knn[1,], function(model) model$results$
variability_knn_k = variability_knn[2,]
variability_knn_accuracy
variability_knn_k

round(1-mean(variability_knn_accuracy), digits = 4)
round(1-min(variability_knn_accuracy), digits = 4)
round(1-max(variability_knn_accuracy), digits = 4)
round(sd(variability_knn_accuracy), digits = 6)

summary(as.factor(as.numeric(variability_knn_k)))

data_outer = data.frame(x_outer = x, y_outer = y)

# d

library(klaR)
library("htmltools")

tic()
trControl = trainControl(method  = "cv",
                         number  = 10)
naiveBayes_clf = train(y ~ .,
                       method     = "nb",
                       trControl  = trControl,
```

```
                          metric      = "Accuracy",
                          data        = data)
toc()

naiveBayes_clf$results$Accuracy

Bayes_shuffle = function(data){
  rows = sample(nrow(data))
  data_shuffle = data[rows,]
  return( train(y ~ .,
      method      = "nb",
      trControl   = trControl,
      metric      = "Accuracy",
      data        = data_shuffle)$results$Accuracy[1] )
}

variability_bayes = replicate(10, Bayes_shuffle(data))

round(1-mean(variability_bayes), digits = 4)
round(1-min(variability_bayes), digits = 4)
round(1-max(variability_bayes), digits = 4)
round(sd(variability_bayes), digits = 6)

bayes_error = 1/(2*(1+eps)^p)
round(bayes_error, digits = 4)
round(1-mean(variability_bayes), digits = 4)
```