Coursework - Machine Learning

CID: 02091191

February 2, 2022

Abstract

In this document we are presenting the work conducted during the first Machine Learning Coursework of the year.

1 First question

Let $\mathcal{D} = \{(x_i, y_i, z_i)\}_{i=1}^n$ a dataset with $n = 100$. Note that x and y are continuous variables whereas z is a discrete $\{0, 1\}$ variable. Consider the following model:

$$Y_i = aZ_i + (b_1 + b_2Z_i)\cos X_i + (c_1 + c_2Z_i)X_i + (d_1 + d_2Z_i)X_i^2 + (e_1 + e_2Z_i)X_i^3 + \varepsilon_i$$

where ε_i are independent and $\mathbb{E}[\varepsilon_i] = 0$ for all $i = 1, \dots, n$.

1. Ridge regression

Consider first a ridge regression model to infer the unknown parameters. Note that the model can be re-written in a matrix form as follows:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}}_{\boldsymbol{y}} = \underbrace{\begin{bmatrix} z_1 & \cos x_1 & z_1\cos x_1 & x_1 & z_1x_1 & x_1^2 & z_1x_1^2 & x_1^3 & z_1x_1^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ z_i & \cos x_i & z_i\cos x_i & x_i & z_ix_i & x_i^2 & z_ix_i^2 & x_i^3 & z_ix_i^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ z_n & \cos x_n & z_n\cos x_n & x_n & z_nx_n & x_n^2 & z_nx_n^2 & x_n^3 & z_nx_n^3 \end{bmatrix}}_{X} \cdot \underbrace{\begin{bmatrix} a \\ b_1 \\ b_2 \\ c_1 \\ c_2 \\ d_1 \\ d_2 \\ e_1 \\ e_2 \end{bmatrix}}_{\boldsymbol{\theta}} + \underbrace{\begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_i \\ \vdots \\ \varepsilon_n \end{bmatrix}}_{\boldsymbol{\varepsilon}}$$

i.e. $\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\varepsilon}$ where $\boldsymbol{y} \in \mathbb{R}^n$, $X \in \mathbb{R}^{n\times p}$, $\boldsymbol{\theta} \in \mathbb{R}^p$ and $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ ($p = 9$ is the number of features). The ridge regression model requires to minimise the mean squared error along with a L_2 regularisation term:

$$\|\boldsymbol{y} - X\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}\|^2$$

where $\|\cdot\|$ denotes the L_2 norm and $\lambda > 0$. Differenciating with respect to $\boldsymbol{\theta}$ and equalling to 0 gives:

$$-2X^T(\boldsymbol{y} - X\hat{\boldsymbol{\theta}}) + 2\lambda\hat{\boldsymbol{\theta}} = 0 \Longleftrightarrow \hat{\boldsymbol{\theta}} = (X^TX + \lambda I_p)^{-1}X^T\boldsymbol{y}$$

Note here that as X^TX is a Gram matrix, it is positive semi-definite. Thus, adding λI_p makes all the eigenvalues of $(X^TX + \lambda I_p)$ strictly positive, and therefore $(X^TX + \lambda I_p)$ is always invertible.

The aim is to find the best value of λ: if λ is too *small*, the ridge regression model is equivalent to a simple linear regression model, whereas if λ is too *large*, the minimisation problem is equivalent to minimising the L_2 norm of θ without taking into account the model. Hence, a trade-off has to be found for the value of λ. To do so, one may use a cross-validation procedure. As the dataset is relatively small (only $n = 100$ sample size), a leave-one-out cross validation (LOOCV) procedure seems appropriate (note that using a k-fold cross validation would have been suitable too, but the small size of the dataset makes the LOOCV procedure doable here). The LOOCV consists in, for a given value of λ, use all subsets consisting of a single data pair as the validation set, and average the overall mean squared error (MSE) obtained for each data pair. We apply this procedure to a grid of value for $\lambda \in [0.001, 1]$ and plot the average MSE obtained for each value of λ.

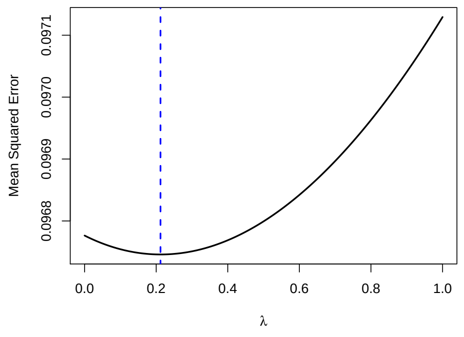


Figure 1.1: Average MSE for each value of λ from the LOOCV procedure

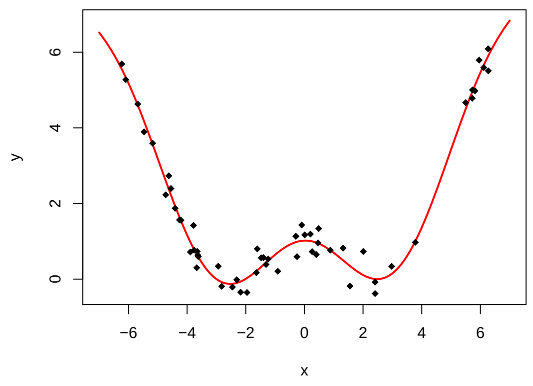
a	$8.597 \cdot 10^{-2}$
b_1	1.019
b_2	0.8631
c_1	$2.569 \cdot 10^{-2}$
c_2	$-1.635 \cdot 10^{-3}$
d_1	0.1206
d_2	$3.843 \cdot 10^{-3}$
e_1	$-6.853 \cdot 10^{-5}$
e_2	$3.097 \cdot 10^{-2}$

Table 1: Unknown parameters inferred from the ridge regression model with $\lambda = 0.2123$

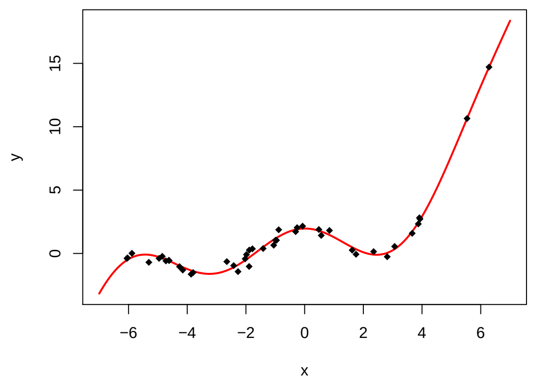
Figure 1.1 shows that the average MSE on validation sets is minimal when $\lambda = 0.2123$. Therefore, we may now fit a ridge regression model on the whole dataset with $\lambda = 0.2123$ and obtain estimates for the unknown parameters $\hat{\boldsymbol{\theta}}$ that are reported in Table 1.

One may now use the parameters estimates to predict the response y for new values of x and z as:

$$\boldsymbol{y}_{\text{new}} = X_{\text{new}} \hat{\boldsymbol{\theta}}$$



(a) $z = 0$



(b) $z = 1$

Figure 1.2: Predicted values of y for $z \in \{0, 1\}$ and $x \in [-7, 7]$ in red line along data points in black

Figure 1.2 illustrates the regression lines drawn from the ridge regression fitting procedure both for $z = 0$ and $z = 1$ compared to the data points. The regression lines seem to fit correctly to the data, with a R^2 score of 0.9859. However, we should be careful as this corresponds to the score obtained on training set as there was no train-test split in this case. We may however note that some data points especially for x around 0 and $z = 0$ differ from the regression line.

2. Bayesian approach

Assume now that $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ for all $i = 1, \dots, n$, $\sigma^2 = 0.1$ and that the unknown parameters are *a priori* independent, so that we may consider a Gaussian prior for $\boldsymbol{\theta}$ with mean 0 and variance α:

$$p(\boldsymbol{\theta}) = \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{0}, \alpha I_p)$$

Using Bayes' Theorem, we may derive the posterior distribution for $\boldsymbol{\theta}$:

$$\begin{aligned} p(\boldsymbol{\theta}|X, \boldsymbol{y}, \sigma) &= \frac{\mathcal{N}_{\boldsymbol{y}}(X\boldsymbol{\theta}, \sigma I_n)\mathcal{N}_{\boldsymbol{\theta}}(\mathbf{0}, \alpha I_p)}{\int \mathcal{N}_{\boldsymbol{y}}(X\boldsymbol{\theta}, \sigma I_n)\mathcal{N}_{\boldsymbol{\theta}}(\mathbf{0}, \alpha I_p) d\boldsymbol{\theta}} \\ &= \mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{\mu}, \Sigma) \end{aligned}$$

according to the Lecture Notes [1] (p.31), with

$$\boldsymbol{\mu} = \left(X^T X + \frac{\sigma^2}{\alpha} I_p\right)^{-1} X^T \boldsymbol{y} \quad \text{and} \quad \Sigma = \sigma^2 \left(X^T X + \frac{\sigma^2}{\alpha} I_p\right)^{-1}$$

Then, as defined above, $\boldsymbol{\mu} \in \mathbb{R}^p$ is the vector of posterior mean of the 9 unknown parameters of the model. We may plot these posterior means for each parameter as a function of α, say for $\alpha \in [10^{-5}, 1]$.

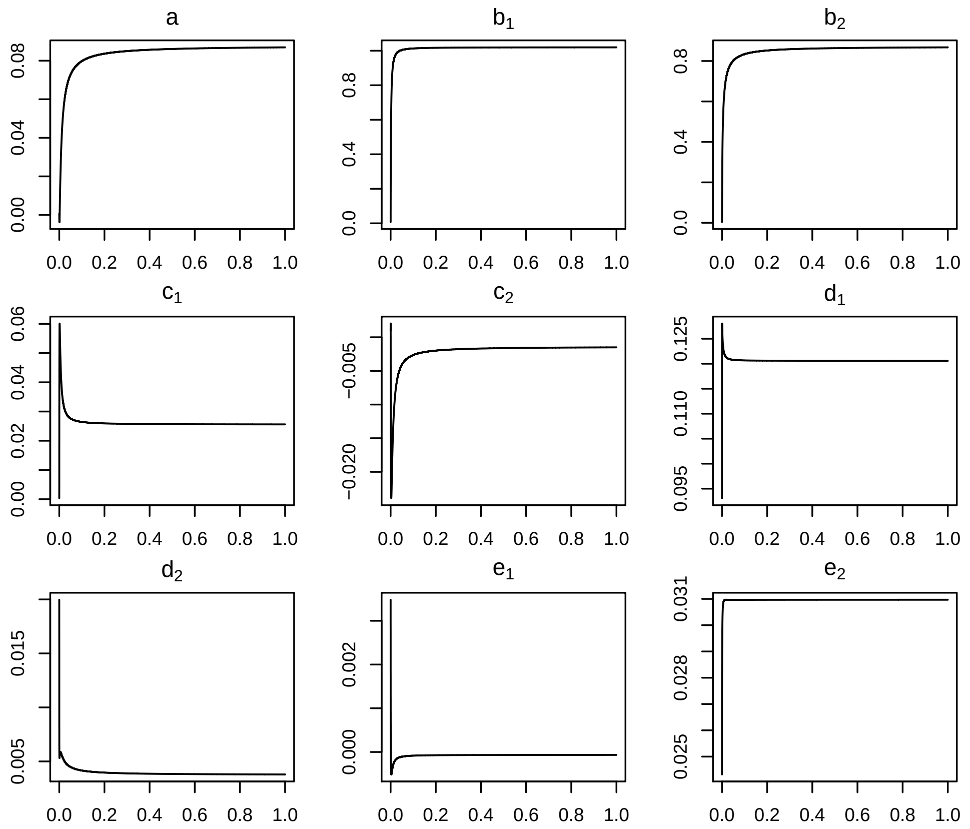


Figure 1.3: Posterior mean of each unknown parameter as functions of α for $\alpha \in [10^{-5}, 1]$ (α is on the x-axis, the posterior mean of each parameter is on y-axis)

Figure 1.3 first shows that plotting the posterior mean of each parameter as a function of α only for $\alpha \in [10^{-5}, 1]$ was sufficient and that plotting it for higher values of α would not have brought any additional information as they seem to converge relatively fast. We may notice that as α is close to 0, the posterior means of each parameter vary a lot and is relatively unstable compared to higher values of α: this is simply because in the expression of $\boldsymbol{\mu}$, α is at the denominator. In Bayesian terms, α being very close to 0 would mean that our prior assumption regarding the unknown parameters is that we *believe* that they are all close to zero as the variance is very small (i.e. the distribution is narrowed around 0, recall that the prior is a normal distribution with mean 0 and variance α). However, higher values of α could be compared as an *uninformative* prior, and in that case, it is equivalent to fit a simple linear regression as if the regularisation term was close to 0. Hence, the converged posterior means we observe on Figure 1.3 are the same estimates that we would obtain from fitting a simple linear regression to the dataset without any ridge regularisation. Finally, note that the ridge penalisation is equivalent to the Bayesian approach: in fact, λ in the ridge model is equivalent to $\frac{\sigma^2}{\alpha}$ in the Bayesian approach. As the previous LOOCV showed that the best value of λ was 0.2123, we may find the equivalent value of α, which would here be $\alpha = \frac{\sigma^2}{\lambda} = 0.4710$. On Figure 1.3, for α greater than this particular value, the posterior mean of all coefficients is nearly constant, maybe except from a which is still converging.

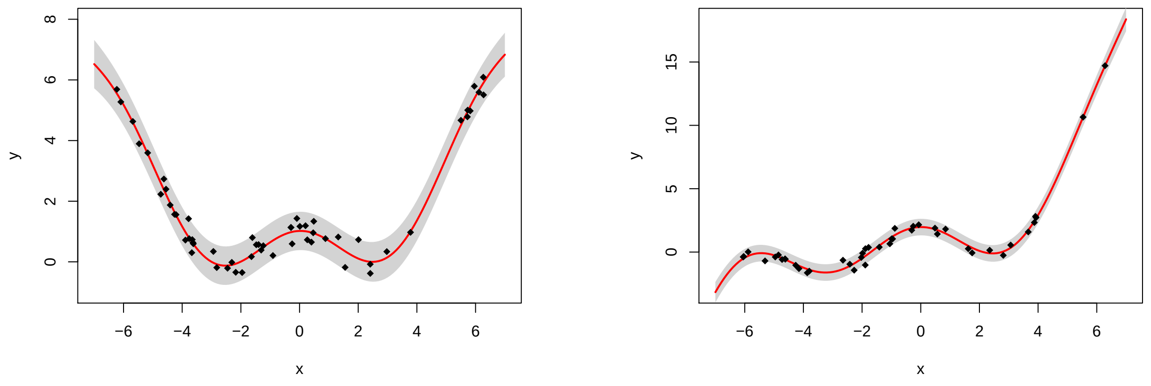
We now fix $\alpha = 1$ and we focus on predicting the response y for new values of $x \in [-7, 7]$ and $z \in \{0, 1\}$ using this Bayesian approach. One then must derive the posterior predictive distribution

$$\begin{aligned} p(y_{\text{new}}|x_{\text{new}}, X, \boldsymbol{y}, \sigma) &= \int p(y_{\text{new}}|x_{\text{new}}, \boldsymbol{\theta}, \sigma) p(\boldsymbol{\theta}|X, \boldsymbol{y}, \sigma) d\boldsymbol{\theta} \\ &= \mathcal{N}_{y_{\text{new}}}(x_{\text{new}}^T \boldsymbol{\mu}, x_{\text{new}}^T \Sigma x_{\text{new}} + \sigma^2) \end{aligned}$$

This result means that, given a new vector of features x_{new}, the posterior predictive distribution of the response associated with this data point is normally distributed around the predicted value $x_{\text{new}}^T \boldsymbol{\mu}$ with variance $x_{\text{new}}^T \Sigma x_{\text{new}} + \sigma^2$. Therefore, one may easily compute for all $x \in [-7, 7]$ and $z \in \{0, 1\}$ a confidence interval at a β-level as

$$\left[x_{\text{new}}^T \boldsymbol{\mu} - q_{1-\beta/2} \sqrt{x_{\text{new}}^T \Sigma x_{\text{new}} + \sigma^2},\ x_{\text{new}}^T \boldsymbol{\mu} + q_{1-\beta/2} \sqrt{x_{\text{new}}^T \Sigma x_{\text{new}} + \sigma^2} \right]$$

where x_{new} denotes the *design matrix form* of the features x, z as defined previously. Therefore, fixing $\alpha = 1$, we define a grid of values of x in $[-7, 7]$ and z in $\{0, 1\}$, compute their posterior predictive means and variance and finally compute the lower and upper bounds for a 95% confidence interval.



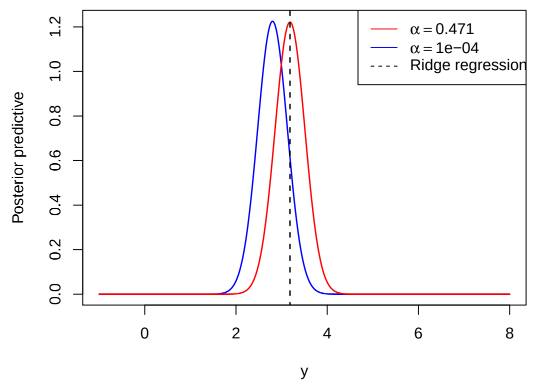
(a) $z = 0$ (b) $z = 1$

Figure 1.4: Mean of the posterior predictive distribution of y for $z \in \{0, 1\}$ and $x \in [-7, 7]$ in red line. 95% confidence interval area shaded in grey. Data points in black.

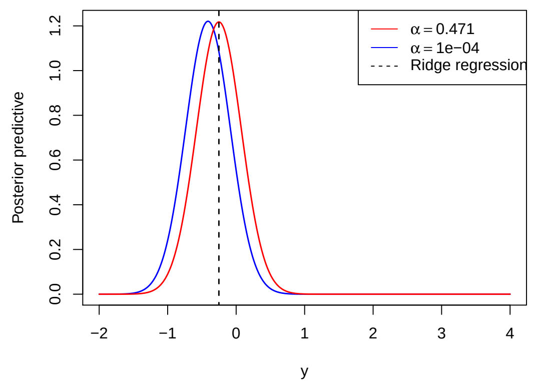
The fitted lines showed on Figure 1.4 are very similar to the ones obtained from fitting a ridge regression on Figure 1.2. We may derive the same conclusions about the goodness of fit. Moreover, we may add that the Bayesian approach has a small advantage compared to the ridge regression model, being that it provides a posterior predictive distribution for the response, whereas the ridge regression only provides *raw* fitted values. Hence, we may build a 95% confidence interval for each value of x and z which is represented in grey on Figure 1.4, and we note that all data points are within this interval.

3. Making new predictions

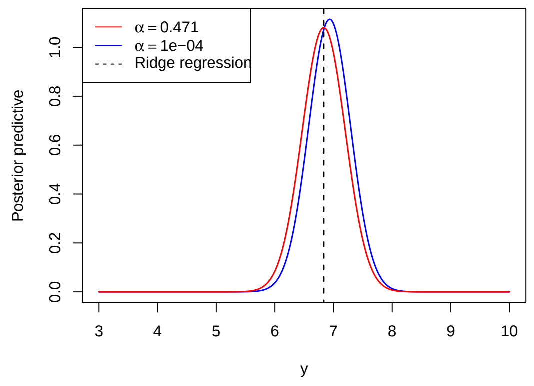
Assume now we are interested in making prediction for 4 new data points $(x, z) \in \{-5, 7\} \times \{0, 1\}$ using the ridge regression model and the Bayesian approach model fitted previously using two different values of α. We may first think about choosing $\alpha = \frac{\sigma^2}{\lambda} = 0.4710$ in order to check if the ridge regression and the Bayesian models agree on the predicted value, which should be the case here if our code was correctly implemented. Moreover, as we have seen on Figure 1.3 that increasing α would not change significantly the results from using $\alpha = 0.4710$, we may choose in the contrary a value of α very close to 0 and see how choosing a such *inappropriate* prior would give different results.



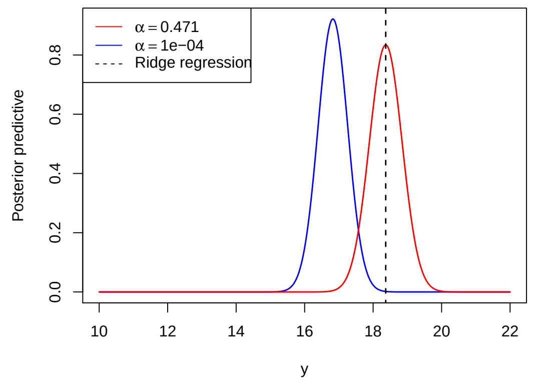
(a) $(x, z) = (-5, 0)$



(b) $(x, z) = (-5, 1)$



(c) $(x, z) = (7, 0)$



(d) $(x, z) = (7, 1)$

Figure 1.5: Comparison of posterior predictive distributions of the Bayesian model using $\alpha = 0.4710$ and $\alpha = 10^{-4}$ for the four given new data points, along with the ridge regression predicted value

Figure 1.5 shows that for all new data points (x, z), the ridge regression model and the Bayesian model with $\alpha = 0.4710$ give the same predicted value, which is consistent as the Bayesian model with a such value of α is equivalent to a ridge regression model with a $\frac{\sigma^2}{\alpha} = \lambda$ penalty coefficient. Furthermore, we may notice that for $\alpha = 10^{-4}$, the Bayesian model gives significantly different values of y especially for $(x, z) = (7, 1)$. This is because we chose a prior distribution for the model parameters very narrowed around 0, and thus the maximum a posterior (MAP) estimates of the parameters are different and close to zero. In other terms, it would be similar to solve the optimisation problem:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\boldsymbol{y} - X\boldsymbol{\theta}\|^2 + \frac{\sigma^2}{\alpha}\|\boldsymbol{\theta}\|^2 \simeq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{\sigma^2}{\alpha}\|\boldsymbol{\theta}\|^2$$

as $\alpha \to 0$, and does not take into account the fitting optimisation anymore.

2 Second question

2.1 Data description

Consider the Million Song dataset (here a subset of the original dataset) which contains in our case $n = 12,523$ samples and $p = 90$ attributes about songs, along with a last column which corresponds to the year the song was released. The 90 features correspond to 12 timbre averages and 78 timbre covariances measured on segments of a song. The aim of this question is to determine if a song has been produced before or after 1980.

We may first define the dataset $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{n}$ such that for all $i = 1, \dots, n$, $x^{(i)} \in \mathbb{R}^p$ and $y^{(i)} \in \{0, 1\}$, with $n = 12,523$, $p = 90$ and $y^{(i)} = 1$ if the song has been released in or after 1980 and $y^{(i)} = 0$ if the song has been released before 1980. This choice has been made so that each class has (approximately) the same prior probability (proportion of songs). We then face a binary classification problem.

Before considering any classification model, we may first focus on the dataset. As mentioned before, the dataset contains $n = 12,523$ samples, $p = 90$ features and one target variable which is whether or not a song has been produced before 1980. We may note that all features are continuous. Since the dataset is high dimensional, plotting any correlation or pair plot would not be readable. Therefore, we suggest here to reduce the dimension of the dataset in order to visualise it more easily. To do so, we apply a principal component analysis (PCA) to the 90 features after standardising as they may have different range of values (packages `factoextra` and `stats` in `R` to fit the PCA, functions `prcomp` and `get_pca_ind`). The PCA procedure indicates that 4 principal components would explain 31.55% of variance. We then may plot the projection of all data points in the new space made by the 4 principal components.

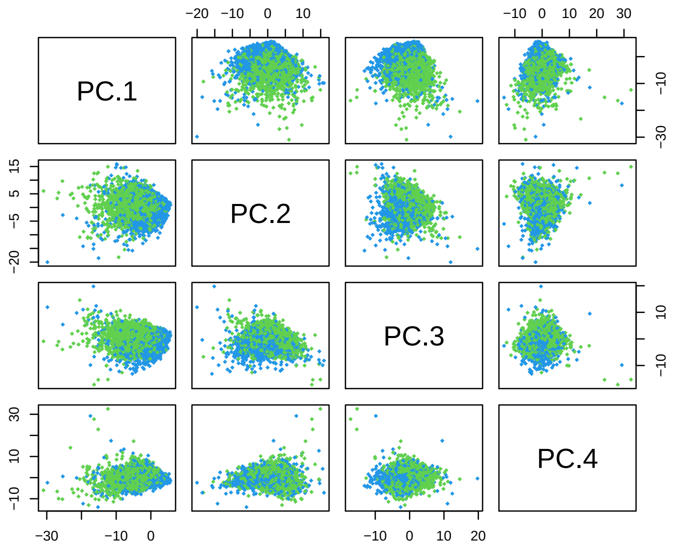


Figure 2.1: Pair plot of data points in the four dimensional principal components space. Green points are songs released before 1980 and blue points are songs released in or after 1980

Figure 2.1 shows that data points are not linearly separable in the projection space. However, we may notice some *trend* between songs released before and after 1980, which infers that a classification procedure would not be vain.

To begin with classification, we first divide our dataset in a train set and a test set using the `caret` package and function `createDataPartition` that creates a list of indices for the train set randomly in the initial data and providing the same proportion of each class than in the initial dataset in both train and test sets. We use 80% of the dataset as train set and the remaining 20% as test set, and in both sets, the proportion of $y^{(i)}$ labelled with 1 is 0.495 which ensures balance between both sets.

2.2 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) classifier seems appropriate as all ${x^{(i)}}_{i=1}^{n}$ are continuous. It assumes that the class-conditional densities are Gaussian with a shared covariance matrix Σ

$$x|y = k \sim \mathcal{N}(\mu_k, \Sigma), \quad k \in \{0, 1\}$$

Note that in our dataset, multiple features do not seem to be normally distributed but are right skewed, this aspect will be discussed later in this report. The decision boundary is given by $p(y = 1|x) = p(y = 0|x)$, and is usually written as

$$\log \frac{p(y=1|x)}{p(y=0|x)} = \omega^T x + b$$

where $\omega = \Sigma^{-1}(\mu_0 - \mu_1)$ and $b = (\mu_1 - \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0) + 2 \log \frac{\pi_1}{\pi_0}$, which can be shown easily using Bayes' Theorem and use the Gaussian assumptions for $x|y$. Therefore, the decision boundary is a hyperplane that best separates the data as they may be not linearly separable. We may use the training set to derive the decision boundary, and then classify each new data point according to the sign of $\omega^T x_{\text{new}} + b$. To apply this method, we need to estimate π_k, μ_k and Σ as

$$\hat{\pi_k} = \frac{n_k}{n}, \quad \hat{\mu_k} = \frac{1}{n_k} \sum_{i, y^{(i)} = k} x^{(i)}, \quad \hat{\Sigma} = \frac{1}{n} \sum_{k=0}^{1} \sum_{i, y^{(i)} = k} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T$$

where n_k is the number of elements in the training set from class k. Note that the LDA procedure does not require any standardising pre-processing as it does not rely on distances, and does not require any parameter tuning neither. However, we use cross-validation in order to estimate better the real performance of this model on the training set. In real cases, this is a good practice in order to choose between multiple models. The cross-validation procedure is a k-fold with 5 folds using accuracy as the metric, as the training set may be too large to consider a LOOCV.

In `R`, we use the `caret` and `MASS` package to define the LDA model and the cross-validation procedure. The LDA classifier obtains 0.7574 accuracy average on all 5 validation sets. The results on test set will be described in the **Results and performances** section.

2.3 K-Nearest Neighbours

The LDA classifier is a generative approach and a parametric model, whereas a k-nearest neighbours (KNN) classifier is a discriminative approach and a non-parametric model. No training is needed for a KNN classifier: for a test input x_{new} we compute the distances to all other points and classify the new input on a simple majority decision using the k nearest neighbours. Hence, as the algorithm relies on distances, it is essential to centre and scale data points according to the training set:

$$\tilde{x_j} = \frac{x_j - \bar{x_j}}{\sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_j^{(i)} - \bar{x_j})^2}}, \quad x_j = \begin{bmatrix} x_j^{(1)} \\ \vdots \\ x_j^{(n)} \end{bmatrix}, \quad \bar{x_j} = \frac{1}{n} \sum_{i=1}^{n} x_j^{(i)}, \quad j = 1, \ldots, p$$

Note that standardisation on test set will use the same means and variances as on training set. Although KNN is non-parametric, there are still parameters that need to be chosen wisely. First, the number of neighbours to take into account for the majority vote k, and second the distance function that may have great effect on the performance of the algorithm. To optimise these parameters, we may use a cross-validation procedure, here a k-fold cross validation with 3 folds using accuracy as the metric. We therefore define a grid of values to be tested for k and the distance function. The two distance functions considered here are the usual Euclidean distance and the Manhattan distance, which are both Minkowski distances with respectively $l = 2$ and $l = 1$.

$$d_{\text{Eucl}}(s,t) = \sqrt{\sum_{d=1}^{p}(s_d - t_d)^2} \quad d_{\text{Manh}}(s,t) = \sum_{d=1}^{p}|s_d - t_d|$$

Considering the Manhattan distance and not only the Euclidean distance is important here because the dataset is high dimensional (90 features) and as this article [2] suggests, one should prefer L^l distances with small l in high dimensions rather than high values of l. The grid of values chosen for k is $\{3, 5, \ldots, 95, 97\}$. Note that the number of folds considered in KNN is less than for the LDA classifier because the hyperparameter tuning would take a higher amount of time. The KNN algorithm used in `R` is from the `kknn` library. The cross-validation procedure took around 5 hours, and gave the following results:

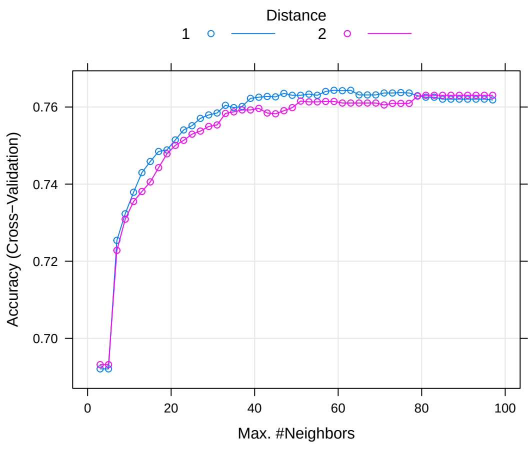


Figure 2.2: Average accuracy on validation sets for all values of k and distances

Figure 2.2 hence shows that the best parameters for the KNN classifier are $k = 63$ and the Manhattan distance (testing higher values of k would not improve the model). With these hyperparameters, the KNN classifier obtains a 0.7643 accuracy average on validation sets. The results on test set will be described in the **Results and performances** section.

2.4 Naive Bayes

The Naive Bayes (NB) classifier is a generative approach and parametric model. As all features are continuous, we may consider that the appropriate distributions are Gaussian. The assumptions are similar to the LDA classifier, but are even stronger:

$$x|y = k \sim \mathcal{N}(\mu_k, \Sigma), \quad k \in \{0,1\}, \quad \Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_p^2)$$

In the Naive Bayes case, we make the LDA assumptions in addition to the *naive* assumption that the covariance matrix is diagonal, and this approach works surprisingly well in many cases. All features are therefore assumed independent given the label. Hence,

$$p(x|y = k) = \prod_{j=1}^{p} p(x_j|y = k) = \prod_{j=1}^{p} \Phi(x_j; \mu_j, \sigma_j^2)$$

For the given dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, the joint probability for a single data point $x^{(i)}, y^{(i)}$ is

$$p\left(x^{(i)}, y^{(i)} \mid \theta, \pi\right) = p\left(y^{(i)} \mid \pi\right) \prod_{j=1}^{p} p\left(x_j^{(i)} \mid \theta_j\right) = \prod_{k=0}^{1} \pi_k^{1_{(y^{(i)}=k)}} \prod_{j=1}^{p} \prod_{k=0}^{1} p\left(x_j^{(i)} \mid \theta_{jk}\right)^{1_{(y^{(i)}=k)}}$$

where π_k is the prior for class k (all priors sum to 1). Hence, the log-likelihood of the whole data assuming independence is:

$$\log p(\mathcal{D} \mid \theta, \pi) = \sum_{k=0}^{1} n_k \log \pi_k + \sum_{k=0}^{1} \sum_{j=1}^{p} \sum_{i:y^{(i)}=k} \log p\left(x_j^{(i)} \mid \theta_{jk}\right)$$

Therefore, parameters π and θ of the Naive Bayes model can easily be estimated using maximum likelihood. Once these parameters are learnt from training data, one may easily calculate the probability of a new data point x to be from class 1 (and similarly from class 0):

$$\begin{aligned} p(y=1 \mid x) &= \frac{p(x \mid y=1)p(y=1)}{p(x)} \\ &= \frac{\left(\prod_{j=1}^{p} p\left(x_j \mid y=1\right)\right)\pi_1}{\left(\prod_{j=1}^{p} p\left(x_j \mid y=1\right)\right)\pi_1 + \left(\prod_{j=1}^{p} p\left(x_j \mid y=0\right)\right)\pi_0} \end{aligned}$$

In `R`, we use the `caret` package along with the `klaR` package to define and fit the Naive Bayes model on the training set, using a k-fold cross-validation with 5 folds with accuracy as the metric in order to better estimate the real accuracy of our model on new data. This Naive Bayes classifier obtains 0.6629 accuracy average on all 5 validation sets. The results on test set will be described in the **Results and performances** section.

2.5 Results and performances

		True 0	**True** 1
Pred.	0	972	305
	1	292	935

(a) LDA

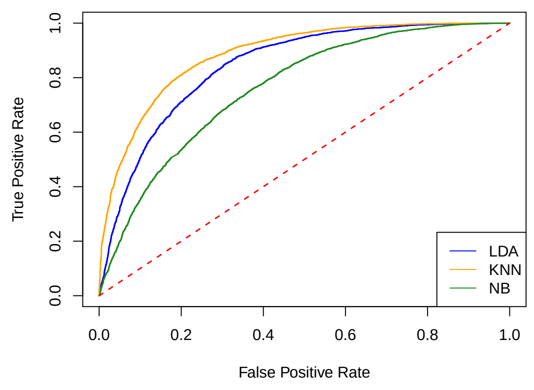
		True 0	**True** 1
Pred.	0	1061	295
	1	203	945

(b) KNN ($k = 63$, Manhattan distance)

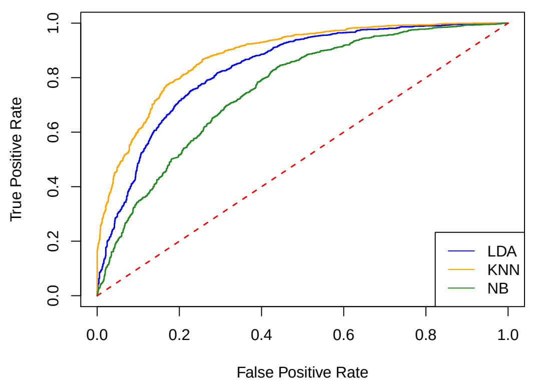
		True 0	**True** 1
Pred.	0	642	240
	1	622	1000

(c) Naive Bayes

Table 2: Confusion matrices obtained from LDA, KNN and Naive Bayes classifiers on test set



(a) Train set



(b) Test set

Figure 2.3: ROC curves for the LDA, KNN and Naive Bayes classifiers on both train and test sets

First, we may recall the average accuracy obtained for each model on validation sets: 0.7574 for the LDA, 0.7643 for the KNN and 0.6629 on the Naive Bayes. Therefore, without looking at the test set at all, we would have chosen the KNN classifier as the best in terms of accuracy on validation sets.

Table 2 illustrates the confusion matrices obtained on test set for the three models, which we may derive easily the **accuracy**: 0.7616 for the LDA, 0.8011 for the KNN and 0.6558 for the Naive Bayes. Therefore, we obtain a similar result to the accuracy on validation sets, and the KNN performs better without any overfitting. However, before asserting that the KNN has *more than 80% accuracy*, one should remain careful as it looks higher than the 0.7643 accuracy obtained on validation sets and it may be a *lucky* result.

We may also have focused on different performance measures: sensitivity, specificity,... Here, we suggest to focus on **F1 score**, which is defined as the harmonic mean between precision (what proportion of positive identifications was actually correct?) and recall (what proportion of actual positives was identified correctly?). Using figures from Table 2, the F1 score of LDA is 0.7580, the one of KNN is 0.7410 and the one of Naive Bayes is 0.6988. Thus, in terms of F1 score, the LDA classifier performs better. Note that this measure of performance is more commonly used in medicine or biology diagnosis where we want to give more importance to false negatives and false positives (i.e. not to misclassify a patient saying he does not have a disease whereas he actually has it). However, the F1 score is a good measure for comparing model on both precision and recall simultaneously.

Now, we may focus on the **ROC curves** showed on Figure 2.3 obtained for all three models on both train and test sets. We may easily see that the ROC curve obtained for the KNN classifier shows a better fit and generalisation than the one of the LDA which itself shows better fit and generalisation to the test set than the Naive Bayes classifier for any threshold we may choose for the boundary. Hence, the area under the curve (AUC) on test set of the KNN classifier is 0.8806, the one of the LDA is 0.8342 and the one of the Naive Bayes is 0.7575. As we may prefer models where the AUC is close to 1, we thus prefer the KNN classifier in terms of ROC curve and AUC.

Let us focus on the **computational cost** and the complexity of each algorithm, assuming n is the sample size and p is the number of feature (the number of class being 2 as we work on binary classification, it is not taken into account here). A KNN classifier does not require any training. However, to make a new prediction, it requires to compute the distances to all the n others data points which are p dimensional: thus, a KNN classifier has no training time complexity but has a prediction/testing time complexity of $\mathcal{O}(np)$. A LDA classifier requires to estimate μ_k, Σ and π_k during the training procedure, which is in terms of complexity $\mathcal{O}(np^2)$ due to Σ. Moreover, in terms of prediction, it requires to compute $\omega^T x + b$ and check its sign, which is $\mathcal{O}(p)$ as the vectors are p dimensional. Finally, the Naive Bayes has a training complexity of $\mathcal{O}(np)$ (similar to LDA but Σ is diagonal) and a prediction complexity of $\mathcal{O}(p)$. Therefore, if one may spend some time training the model but needs high speed predictions, it should prefer the LDA or Naive Bayes classifiers, and therefore the LDA classifier as it is more accurate. However, if there is no need of speed for the prediction task, we may of course prefer the KNN algorithm which has the best performances overall.

As mentioned before, one of the LDA and Naive Bayes assumptions is that features are normally distributed. However, one may notice that even if all features are continuous in our dataset, some of them are not normally distributed but may be sampled from a Gamma distribution for example. We therefore tried to modify such features by applying a log transformation only to these features, letting the other *almost* normally distributed features as before, and fitted the same LDA, KNN and Naive Bayes models. The accuracy of the LDA classifier on test set becomes 0.7586, the accuracy of the KNN becomes 0.7636 and the one of the Naive Bayes becomes 0.6729, which only improves slightly but not significantly for the two generative approaches and in the contrary decreases for the KNN model.

References

[1] Dr Sarah Filippi (January 2022) *MATH70091 - Machine Learning lecture notes*, Imperial College London MSc Statistics resources

[2] Charu C. Aggarwal, Alexander Hinneburg, Daniel A. Keim (2001) *On the Surprising Behavior of Distance Metrics in High Dimensional Space*, IBM T. J. Watson Research Center, Institute of Computer Science, University of Halle, [URL]

A R code

```
### Problem 1

data <- read.csv("02091191.csv")
head(data)

# Design matrix for regression
y <- data$y
design_matrix <- function(x, z){
  X <- c()
  X <- cbind(X, z)
  X <- cbind(X, cos(x))
  X <- cbind(X, z * cos(x))
  X <- cbind(X, x)
  X <- cbind(X, z * x)
  X <- cbind(X, x^2)
  X <- cbind(X, z * x^2)
  X <- cbind(X, x^3)
  X <- cbind(X, z * x^3)
  X
}
X <- design_matrix(data$x, data$z)

# LOOCV to estimate the best regularization parameter lambda
n <- dim(X)[1]
p <- dim(X)[2]
# LOOCV is doable here because we don't have a big n
LOOCV <- function(lambda){
  CV <- c()
  for(i in 1:n){
    X_train <- X[-i,]
    y_train <- y[-i]
    X_test <- X[i,]
    y_test <- y[i]
    beta_hat <- solve(t(X_train) %*% X_train + lambda * diag(p), t(X_train) %*% y_train)
    y_pred <- X_test %*% beta_hat
    CV <- c(CV, (y_pred - y_test)^2)
  }
  mean_CV <- mean(CV)
  sd_CV   <- sd(CV)
  results <- data.frame(mean_CV, sd_CV)
  return(results)
}

err <- c()
lambdas <- seq(0, 1, by=0.0001)
results <- lapply(lambdas, LOOCV)
for(i in 1:length(results)){
  err <- c(err, results[[i]][[1]])
}
plot(lambdas, err, type="l", lwd=2, xlab=expression(lambda),
     ylab="Mean Squared Error")
```

```
lambda_star <- lambdas[err == min(err)]
lines(c(lambda_star, lambda_star), c(0, 1), lty=2, col="blue", lwd=2)

# Fit a ridge regression using this value of lambda
beta_hat <- solve(t(X) %*% X + lambda_star * diag(p)) %*% t(X) %*% y
mse <- mean((y - X %*% beta_hat)^2)
r2 <- 1 - sum((y - X %*% beta_hat)^2) / sum((y - mean(y))^2)

# Predict for X in [-7, 7] and Z=0,1
x <- seq(-7, 7, by=0.1)
z0 <- rep(0, length(x))
z1 <- rep(1, length(x))
X_z0 <- design_matrix(x, z0)
X_z1 <- design_matrix(x, z1)
y_z0 <- X_z0 %*% beta_hat
y_z1 <- X_z1 %*% beta_hat

plot(x, y_z1, type="l", lwd=2, col="red", xlab=expression(x), ylab=expression(y))
points(data[data$z == 1,]$x, data[data$z == 1,]$y, pch=18)

plot(x, y_z0, type="l", lwd=2, col="red", xlab=expression(x), ylab=expression(y),
     ylim=c(min(data[data$z == 0,]$y), max(y_z0)))
points(data[data$z == 0,]$x, data[data$z == 0,]$y, pch=18)


# Posterior mean for each parameter (Bayesian approach)
sigma <- sqrt(0.1)
alphas <- seq(0.00001, 1, by=0.00001)
posterior_mean <- function(alpha) solve(t(X) %*% X + sigma^2/alpha * diag(p)) %*% t(X) %*% y
mus <- sapply(alphas, posterior_mean)
parameter <- c(expression(a), expression(b[1]), expression(b[2]), expression(c[1]),
               expression(c[2]), expression(d[1]), expression(d[2]), expression(e[1]),
               expression(e[2]))
par(mfrow=c(3,3), par(mar = c(2, 2, 2, 2)))
for(i in 1:9){
  plot(alphas, mus[i,], type="l", lwd=1.1, main=parameter[i])
}
mus[,dim(mus)[2]] # parameters estimates

# Posterior predictive distribution
alpha <- 1
# Estimate mu and Sigma
mu <- solve(t(X) %*% X + sigma^2/alpha * diag(p)) %*% t(X) %*% y
Sigma <- sigma^2 * solve(t(X) %*% X + sigma^2/alpha * diag(p))
# Create new design matrices
x <- seq(-7, 7, by=0.1)
z0 <- rep(0, length(x))
z1 <- rep(1, length(x))
X_z0 <- design_matrix(x, z0)
X_z1 <- design_matrix(x, z1)
pred_z0 <- X_z0 %*% mu
pred_z1 <- X_z1 %*% mu
```

```
post_mean_z0 <- X_z0 %*% mu
post_mean_z1 <- X_z1 %*% mu
post_var_z0 <- c()
post_var_z1 <- c()
for(i in 1:dim(X_z0)[1]){
  post_var_z0 <- c(post_var_z0, X_z0[i,] %*% Sigma %*% X_z0[i,] + sigma^2)
  post_var_z1 <- c(post_var_z1, X_z1[i,] %*% Sigma %*% X_z1[i,] + sigma^2)
}
upper_95_z0 <- post_mean_z0 + qnorm(0.975) * sqrt(post_var_z0)
lower_95_z0 <- post_mean_z0 + qnorm(0.025) * sqrt(post_var_z0)
upper_95_z1 <- post_mean_z1 + qnorm(0.975) * sqrt(post_var_z1)
lower_95_z1 <- post_mean_z1 + qnorm(0.025) * sqrt(post_var_z1)

plot(x, post_mean_z0, type="l", col="red", ylim=c(-1, 8), ylab="y")
polygon(c(x, rev(x)), c(upper_95_z0, rev(lower_95_z0)),
        col = "#D3D3D3", lty=0)
lines(x, post_mean_z0, col="red", lwd=2)
points(data[data$z == 0,]$x, data[data$z == 0,]$y, pch=18)

plot(x, post_mean_z1, type="l", col="red", ylab="y")
polygon(c(x, rev(x)), c(upper_95_z1, rev(lower_95_z1)),
        col = "#D3D3D3", lty=0)
lines(x, post_mean_z1, lwd=2, col="red")
points(data[data$z == 1,]$x, data[data$z == 1,]$y, pch=18)


# (-5, 0), (-5, 1), (7, 0), (7, 1) predictions
alpha1 <- sigma^2/lambda_star # Optimal according to cross validation in 1
alpha2 <- 1e-5
mu1 <- solve(t(X) %*% X + sigma^2/alpha1 * diag(p)) %*% t(X) %*% y
mu2 <- solve(t(X) %*% X + sigma^2/alpha2 * diag(p)) %*% t(X) %*% y
Sigma1 <- sigma^2 * solve(t(X) %*% X + sigma^2/alpha1 * diag(p))
Sigma2 <- sigma^2 * solve(t(X) %*% X + sigma^2/alpha2 * diag(p))
# Create new design matrices
X_new <- design_matrix(x=c(-5, -5, 7, 7), z=c(0, 1, 0, 1))
# Compute posterior mean (estimate prediction) and variance of the response
y_new_pred1 <- X_new %*% mu1
y_new_pred2 <- X_new %*% mu2
y_new_var1 <- c()
y_new_var2 <- c()
for(i in 1:dim(X_new)[1]){
  y_new_var1 <- c(y_new_var1, X_new[i,] %*% Sigma1 %*% X_new[i,] + sigma^2)
  y_new_var2 <- c(y_new_var2, X_new[i,] %*% Sigma2 %*% X_new[i,] + sigma^2)
}
# Plot the predictive distribution for these data points
pred_ridge <- X_new %*% beta_hat
for(i in 1:dim(X_new)[1]){
  t <- seq(10, 22, by=0.01)
  dis1 <- dnorm(t, mean=y_new_pred1[i], sd=sqrt(y_new_var1[i]))
  dis2 <- dnorm(t, mean=y_new_pred2[i], sd=sqrt(y_new_var2[i]))
  plot(t, dis2, type="l", col="blue", lwd=1.5,
       xlab="y", ylab="Posterior predictive")
  lines(t, dis1, col="red", lwd=1.5)
  lines(c(pred_ridge[i], pred_ridge[i]), c(-10, 10), lty=2, lwd=1.5)
  legend("topleft", legend=c(expression(alpha==0.4710), expression(alpha==1e-4),
                              "Ridge regression"),
         col=c("red", "blue", "black"), lty=c(1, 1, 2))
}



### Problem 2

ROC <-
  function(y, s)
  {
    yav <- rep(tapply(y, s, mean), table(s))
    rocx <- cumsum(yav)
    rocy <- cumsum(1 - yav)
    area <- sum(yav * (rocy - 0.5 * (1 - yav)))
    list(x = c(0, rocx)/sum(y), y = c(0, rocy)/sum(1 - y), area =
           area/(sum(y) * sum(1 - y)))
  }

data <- read.csv("dataQ2.csv")
head(data)

# PCA to visualise data
res.pca <- prcomp(data[,-c(1)], scale=TRUE)
library(factoextra)
library(dplyr)
res.ind <- get_pca_ind(res.pca)
coord <- res.ind$coord
no_coord <- 4
coord <- data.frame(cbind(coord[,1:no_coord], as.numeric(data$V1>=1980) + 3))
coord <- sample_n(coord, dim(coord)[1])
colnames(coord) <- c("PC.1", "PC.2", "PC.3", "PC.4")
pairs(coord[,1:no_coord], col=coord[,(no_coord+1)], cex=0.7, pch=18)

# Train-test split
library(caret)
set.seed(42)
train_index <- createDataPartition(data$V1, p=0.8, list=FALSE)
train <- data[train_index,]
test <- data[-train_index,]
# Check that data is correctly distributed in both sets wrt to the response
mean(train$V1 >= 1980)
mean(test$V1 >= 1980)
# Transform the response in categorical variable
# 1 if after 1980, 0 if before 1980
train$V1 <- as.factor(as.numeric(train$V1 >= 1980))
test$V1 <- as.factor(as.numeric(test$V1 >= 1980))

# LDA classifier
library(MASS)
library(caret)
```

```
trControl <- trainControl(method  = "cv",
                          number  = 5)

model.lda <- train(V1 ~ .,
                   method     = "lda",
                   trControl  = trControl,
                   metric     = "Accuracy",
                   data       = train)
model.lda # Gets a good accuracy on average validation sets
summary(model.lda)

lda.predict.test <- predict(model.lda, newdata=test)
confusionMatrix(lda.predict.test, test$V1)
f1.lda <- 935/(935 + 0.5*(292+305))

# ROC
lda.posterior <- predict(model.lda, test, type="prob")
lda.posterior.train <- predict(model.lda, train, type="prob")
lda.roc <- ROC(as.numeric(test$V1)-1, lda.posterior[,2])
lda.roc.train <- ROC(as.numeric(train$V1)-1, lda.posterior.train[,2])
lda.auc <- lda.roc$area
lda.auc.train <- lda.roc.train$area

# KNN

trControl <- trainControl(method  = "cv",
                          number  = 3,
                          verboseIter=TRUE)
# Standardizing better than normalizing as features are not bounded
# It's a choice, the range of all features are different
# Normalizing would break completely and lose some information
model.knn <- train(V1 ~ .,
                   method     = "kknn",
                   tuneGrid   = expand.grid(kmax=seq(3, 97, by=2), distance=c(1, 2),
                                            kernel=c("optimal")),
                   trControl  = trControl,
                   metric     = "Accuracy",
                   data       = train,
                   preProcess = c("center","scale"),
                   verbose=TRUE)
model.knn # Gets the recap of mean (on all validation sets) accuracy (k=63)
summary(model.knn)
plot(model.knn)

knn.predict.test <- predict(model.knn, newdata=test)
confusionMatrix(knn.predict.test, test$V1)
f1.knn <- 887/(887 + 0.5*(353+267))

# ROC
knn.posterior <- predict(model.knn, test, type="prob")
knn.posterior.train <- predict(model.knn, train, type="prob")
knn.roc <- ROC(as.numeric(test$V1)-1, knn.posterior[,2])
knn.roc.train <- ROC(as.numeric(train$V1)-1, knn.posterior.train[,2])
plot(knn.roc$x, knn.roc$y, type="l")
knn.auc <- knn.roc$area
knn.auc.train <- knn.roc.train$area


# Naive Bayes
library(klaR)

trControl <- trainControl(method  = "cv",
                          number  = 5)

model.nb <- train(V1 ~ .,
                  method     = "nb",
                  trControl  = trControl,
                  tuneGrid   = expand.grid(usekernel=c(FALSE), fL=c(0),
                                           adjust=c(1)),
                  metric     = "Accuracy",
                  data       = train)
model.nb # Gets a bad accuracy on average validation sets
summary(model.nb)

nb.predict.test <- predict(model.nb, newdata=test)
confusionMatrix(nb.predict.test, test$V1)
f1.nb <- 1000/(1000 + 0.5*(622+240))

# ROC
nb.posterior <- predict(model.nb, test, type="prob")
nb.posterior.train <- predict(model.nb, train, type="prob")
nb.posterior[is.na(nb.posterior)] <- 0.5
nb.posterior.train[is.na(nb.posterior.train)] <- 0.5
nb.roc <- ROC(as.numeric(test$V1)-1, nb.posterior[,2])
nb.roc.train <- ROC(as.numeric(train$V1)-1, nb.posterior.train[,2])
plot(nb.roc$x, nb.roc$y, type="l")
nb.auc <- nb.roc$area
nb.auc.train <- nb.roc.train$area

# Voting classifier with equal weights
vote.predict.test <- as.factor(as.numeric((as.numeric(nb.predict.test)-1 +
                                   as.numeric(knn.predict.test)-1 +
                                   as.numeric(lda.predict.test)-1)/3 > 0.5))
confusionMatrix(vote.predict.test, test$V1)


# ROC Curve final
plot(lda.roc$x, lda.roc$y, type="l", lwd=1.5, col="blue", xlab="False Positive Rate",
     ylab="True Positive Rate")
lines(knn.roc$x, knn.roc$y, lwd=1.5, col="orange")
lines(nb.roc$x, nb.roc$y, lwd=1.5, col="forestgreen")
lines(c(0, 1), c(0, 1), lty=2, lwd=1.5, col="red")
legend("bottomright", legend=c("LDA", "KNN", "NB"), col=c("blue", "orange", "forestgreen"),
       lty=1)
```

```
plot(lda.roc.train$x, lda.roc.train$y, type="l", lwd=1.5, col="blue", xlab="False Positive Rat
     ylab="True Positive Rate")
lines(knn.roc.train$x, knn.roc.train$y, lwd=1.5, col="orange")
lines(nb.roc.train$x, nb.roc.train$y, lwd=1.5, col="forestgreen")
lines(c(0, 1), c(0, 1), lty=2, lwd=1.5, col="red")
legend("bottomright", legend=c("LDA", "KNN", "NB"), col=c("blue", "orange", "forestgreen"),
       lty=1)
```