LINEAR MODELS

Preliminaries

1 – How to plot the vector \( \mathbf{w} \) of coordinates \( (w_1, w_2) \)?

There are several ways:
   
   a) Plot the arrow from \((0,0)\) to \((w_1, w_2)\) – ie plot the vector \( \mathbf{w} \) itself.
   
   b) Plot the arrow from \((0,0)\) to \((1, w_2 / w_1)\) – ie plot the vector \( \mathbf{w} \) rescaled by \(1/w_1\) (note that if \(w_1\) is negative the direction will stay the same but the arrow will point to the opposite sense).

   c) Plot the arrow from \((0,0)\) to \((a, a w_2 / w_1)\) – ie plot the vector \( \mathbf{w} \) rescaled by \(a/w_1\) (note that if \(a/w_1\) is negative the direction will stay the same but the arrow will point to the opposite sense).

   d) Plot the arrow from \((x,y)\) to \((x + a, y + a w_2/w_1)\) – ie plot the vector \( \mathbf{w} \) rescaled by \(a/w_1\) and starting at \((x,y)\) instead of \((0,0)\).

We used method c for the plots in Exercise 1, d for the plots in Exercise 3. Note that, for methods b-c-d, the direction of the line supported by \( \mathbf{w} \) does not change, only the sense and the length of the vector \( \mathbf{w} \). Therefore the legends are abusive when they say \( \mathbf{w} \), they should say rescaled \( \mathbf{w} \).

2 – How to find the intersection point between two lines of slopes \( a_1, a_2 \) and intersects \( b_1, b_2 \)?

We need to find \( (x,y) \) such that \( y = a_1 x + b_1 \) and \( y = a_2 x + b_2 \)

So \((a_2 - a_1) x + (b_2 - b_1) = 0\) and

\[
   x = - (b_2 - b_1) / (a_2 - a_1)
\]

\[
   y = b_1 - a_1 (b_2 - b_1)/(a_2 - a_1)
\]
Exercise 1: PCA

Generate the following data in R and visualize it.

N = 200; D = 2;
X = matrix(0,nrow=N,ncol=D);
X[,1]=10+10*rnorm(N); # 1st dimension
X[,2]=2*X[,1]+20+10*rnorm(N); # 2nd dimension

Question 1

Refer to the slides.

Question 2

# Run PCA
pcamodel = prcomp(X);
names(pcamodel);

# Check the results
eigen_vectors = pcamodel$rotation; # the principal components
eigen_values = (pcamodel$sdev)^2; # the associated variances
rotated_data = pcamodel$x; # the data coordinates in the new space

Question 3

# Divide the plot in 2 parts
par(mfrow=c(1,2));

# On the left, plot the data and the principal components
mu = colMeans(X);
plot(X[,1],X[,2],xlab="X1",ylab="X2",col="blue",asp=1);
PC1 = eigen_vectors[,1];
arrows(mu[1],mu[2],mu[1]+15,mu[2]+15*PC1[2]/PC1[1],col="red",lty=1,lwd=2);
PC2 = eigen_vectors[,2];
arrows(mu[1],mu[2],mu[1]+15,mu[2]+15*PC2[2]/PC2[1],col="orange",lty=1,lwd=2);
legend("topleft",legend=c("PC1","PC2"),fill=c("red","orange"));
# On the right, plot the rotated data
plot(rotated_data[,1],rotated_data[,2],xlab="PC1",ylab="PC2",col="blue",asp=1);

Question 4

C = cov(X); C

\[
\begin{bmatrix}
[1,] & [2,] \\
[1,] & 76.2453 & 155.1534 \\
[2,] & 155.1534 & 430.4627
\end{bmatrix}
\]

V = cov(rotated_data); V

\[
\begin{bmatrix}
PC1 & PC2 \\
PC1 & 4.888112e+02 & 8.799943e-15 \\
PC2 & 8.799943e-15 & 1.789684e+01
\end{bmatrix}
\]
The diagonal of $V$ is close to 0, indicating that the two variables, ie the two principal components are decorrelated.

Question 5

This is a tricky question depending on the package used to compute PCA. If you used pracomp or princomp, the eigen values are not directly returned. Instead the standard deviations of the principal components are returned. They must be squared to obtain the eigen values.

The link should now be clear: the eigen values are the squared standard deviations, ie the variances, of the new variables, ie of the principal components. They should be found in the diagonal of $V$.

\[
\begin{array}{ccc}
V & & \\
PC1 & PC2 \\
PC1 & 4.888112e+02 & 8.799943e-15 \\
PC2 & 8.799943e-15 & 1.789684e+01 \\
\end{array}
\]

eigen_values
[1] 488.81122  17.89684
Exercise 2: PCA versus LDA

Generate the following data in R and visualize it with colours (points of class 1 in blue and points of class 2 in red for example).

\[
\begin{align*}
N1 &= 200; \ N2 = 300; \ D = 2; \\
x1 &= 10 + 10 \times rnorm(N1); \ # \ 1^{st} \ dimension \ for \ points \ of \ class \ 1 \\
y1 &= 2 \times x1 + 20 + 10 \times rnorm(N1); \ # \ 2^{nd} \ dimension \ for \ points \ of \ class \ 1 \\
x2 &= 10 + 10 \times rnorm(N2); \ # \ 1^{st} \ dimension \ for \ points \ of \ class \ 2 \\
y2 &= x2 - 15 + 10 \times rnorm(N2); \ # \ 2^{nd} \ dimension \ for \ points \ of \ class \ 2 \\
\end{align*}
\]

outliers = matrix(0, nrow=Noutliers, ncol=D); \ # \ class \ 2 \ outliers
\begin{align*}
\text{outliers}[1,1] &= 35 + 5 \times rnorm(Noutliers); \ # \ 1^{st} \ dimension \ for \ outliers \\
\text{outliers}[2,1] &= -80 + 10 \times rnorm(Noutliers); \ # \ 2^{nd} \ dimension \ for \ outliers \\
\end{align*}

\[
X = \begin{pmatrix}
\end{pmatrix}
\]

\[
X[1,1] = c(x1, x2); \ # \ complete \ data \ – \ 1^{st} \ dimension \\
X[2,1] = c(y1, y2); \ # \ complete \ data \ – \ 2^{nd} \ dimension \\
\]

# labels to be used for logistic regression
t = c( rep(1, N1), rep(-1, N2) );

Question 1

# load the MASS library
library(MASS)

# Run PCA
pcamodel = prcomp(X);
w pca = pcamodel$rotation[,1]; \ # \ the \ first \ principal \ component

# Run LDA
ldamodel = lda(t ~ X);
w lda = ldamodel$scaling;

Question 2

# Plot the data and both vectors
mu = colMeans(X);
x_color = t; x_color[t==1] = "green"; x_color[t==-1] = "red";
plot(X[,1],X[,2],xlab="X1",ylab="X2",col=x_color,asp=1);
arrows(mu[1],mu[2],mu[1]+15,mu[2]+15*wpca[2]/wpca[1],col="blue",lt y=1,lwd=2);
arrows(mu[1],mu[2],mu[1]+15,mu[2]+70*wlda[2]/wlda[1],col="black",lt y=1,lwd=2);
legend("topleft",legend=c("w_PCA","w_LDA"),fill=c("blue","black"));

Question 2

A good way to check the discriminant power of these projection axes is to project the data and to plot the densities for every class separately.

# Project the data onto the PCA axis: \( y = X w_{PCA} \)

cpa_proj = pcamodel$x[,1];

# Project the data onto the LDA axis: \( y = X w_{LDA} \)

lda_proj = as.vector( X %*% wlda );
# Load the library for density comparison
library(sm)

# Divide the plot in two parts
par(mfrow=c(1,2));

# On the left, plot the densities of the two classes when projected on $w_{PCA}$
sm.density.compare(pca_proj,t,col=c("red","green"),lty=c(1,1));
title(main="PCA projection");

# On the left, plot the densities of the two classes when projected on $w_{LDA}$
sm.density.compare(lda_proj,t,col=c("red","green"),lty=c(1,1));
title(main="LDA projection");

As can be seen, LDA has a much higher predictive power.
Exercise 3: Least squares and logistic regression

Generate the following data in R and visualize it.

N1 = 200; N2 = 300; Noutliers = 60; D = 2;

x1=10+10*rnorm(N1); # 1st dimension for points of class 1
y1=2*x1+20+10*rnorm(N1); # 2nd dimension for points of class 1
x2=10+10*rnorm(N2); # 1st dimension for points of class 2
y2=x2-15+10*rnorm(N2); # 2nd dimension for points of class 2
outliers=matrix(0,nrow=Noutliers,ncol=D); # class 2 outliers
outliers[,1]=35+5*rnorm(Noutliers); # 1st dimension for outliers
outliers[,2]=-80+10*rnorm(Noutliers); # 2nd dimension for outliers

X=matrix(0,nrow=N1+N2+Noutliers,ncol=D)
X[,1]=c(x1,x2,outliers[,1]); # complete data – 1st dimension
X[,2]=c(y1,y2,outliers[,2]); # complete data – 2nd dimension

# labels to be used for logistic regression
tlr=c( rep(1,N1) , rep(0,N2+Noutliers) );

# labels to be used for least squares
tls=c( rep(1,N1) , rep(-1,N2+Noutliers) );

Question 1

# least squares
lsmodel=lm(tls~X);
wls = lsmodel$coefficients;

# logistic regression
lrmodel=glm(tlr~X,family="binomial");
wlr = lrmodel$coefficients;

# compute the intersection between the 2 decision boundaries \( \mathbf{w}_{LS}^T \mathbf{x} + w_0 = 0 \) and \( \mathbf{w}_{LR}^T \mathbf{x} + w_0 = 0 \) to plot everything from there
  # here \( w[1] + w[2] x1 + w[3] x2 = 0 \)
  # so \( x2 = - (w[2]/w[3]) x1 - w[1]/w[3] \)
als = - wls[2]/wls[3]; bls = - wls[1]/wls[3];
alr = - wlr[2]/wlr[3]; blr = - wlr[1]/wlr[3];
inter = vector(); inter[1] = - (blr-bls)/(alr-als);
inter[2] = bls - als*(blr-bls)/(alr-als);

# plot both vectors w and both decision boundaries
x_color = tls; x_color[tls==1] = "green"; x_color[tls==-1] = "red";
plot(X[,1],X[,2],xlab="X1",ylab="X2",col=x_color,asp=1);
plot_xlim = par("usr")[1:2]; plot_ylim = par("usr")[3:4];
  # plot w_{LS} starting from the intersection
arrows(inter[1],inter[2],inter[1]+30,inter[2]+30*wls[3]/wls[2],col="blue",lty=2);
  # plot w_{LS}^T x + w_0 = 0
abline(a=-wls[1]/wls[3],b=-wls[2]/wls[3],col="blue",lwd=2);
  # plot w_{LR} starting from the intersection
arrows(inter[1],inter[2],inter[1]+100,inter[2]+100*wlr[3]/wlr[2],col="black",lty=2);
  # plot w_{LR}^T x + w_0 = 0
abline(a=-wlr[1]/wlr[3],b=-wlr[2]/wlr[3],col="black",lwd=2);
  # plot legend
legend("topleft",legend=c("w_{LS}","decision boundary (LS)"),col=c("blue","blue","black"),lty=2);
legend("topleft",legend=c("w_{LR}","decision boundary (LR)"),col=c("blue","blue","black"),lty=2,ncol=3,lwd=2);
The problem using least squares is that we try to minimise the distance between \( \mathbf{w}^T \mathbf{x} + w_0 \) and our target \( t \in \{0, 1\} \). However, in the dataset we have, although the data is linearly separable, there is a cluster that is far away from the rest.

We can see how least squares always tries to find a \( \mathbf{w} \) that models the whole dataset by passing through it. All green points should have a score of 1, and all red points a score of -1. This of course is not going to be possible with Least squares. In fact, points that are “easy” to classify will be penalized because their scores will be below -1 or over +1 and therefore far from -1 or +1!!!

In the next figure, we plot the scores \( \mathbf{w}^T \mathbf{x} + w_0 \) obtained using Least squares and Logistic regression.
# Create the grids
xgrid = seq(-133/1.4,133/1.4,by=1);
ygrid = seq(-150/1.4,150/1.4,by=1);
gridls = matrix(NA,nrow=length(xgrid),ncol=length(ygrid));
for(i in 1:length(xgrid)) for(j in 1:length(ygrid)) gridls[i,j] = wls[1] + wls[2]*xgrid[i] + wls[3]*ygrid[j];
gridlr = matrix(NA,nrow=length(xgrid),ncol=length(ygrid));
col_pal = colorRampPalette(c('dark blue','white','dark red'));

# Plot the scores for Least squares
mini = min(gridls); maxi = max(gridls); lim = max(abs(mini),abs(maxi));
filled.contour(x=xgrid,y=ygrid,z=gridls,levels=seq(-lim,lim,length.out=52),asp=1,col=col_pal(51),xlab="X1",ylab="X2",main="Least squares",key.title=title(main="w.x + w0"),plot.axes={
points(X[,1],X[,2],col=x_color); axis(1); axis(2); });
# Plot the scores for Logistic regression
mini = min(gridlr); maxi = max(gridlr); lim = max(abs(mini),abs(maxi));
filled.contour(x=xgrid,y=ygrid,z=gridlr,levels=seq(-lim,lim,length.out=52),asp=1,col=col_pal(51),xlab="X1",ylab="X2",main="Logistic regression",key.title=title(main="w.x + w0"),plot.axes={points(X[,1],X[,2],col=x_color); axis(1); axis(2); });
We can see that, for Least squares, the scores are around -1 and 1. For Logistic regression however, the scores are 0 around the change of class and quite high in absolute value everywhere else, reflecting the fact that the prediction is confident. Indeed a high absolute score translates to a probability close to 1 or to 0.

# Plot the probability for Logistic regression
problr = 1 / (1 + exp(-gridlr));
filled.contour(x=xgrid,y=ygrid,z=problr,levels=seq(0,1,length.out=52),asp=1,col=col_pal(51),xlab="X1",ylab="X2",main="Logistic regression",key.title=title(main=expression(paste(sigma,"(w.x + w0)",sep="")",cex.main=0.99),plot.axes={points(X[,1],X[,2],col=x_color); axis(1); axis(2); });
Here we take the outliers out. The two classes are now Gaussian.

# Create dataset without outliers and run LS and LR on it
X2=matrix(0, nrow=N1+N2, ncol=D)
X2[,1]=c(x1,x2); # complete data – 1st dimension
X2[,2]=c(y1,y2); # complete data – 2nd dimension
tlr2=c( rep(1,N1) , rep(0,N2) );
tls2=c( rep(1,N1) , rep(-1,N2) );
lsmodel2=lm(tls2~X2);
wls2 = lsmodel2$coefficients;
lrmodel2=glm(tlr2~X2,family="binomial");
wlr2 = lrmodel2$coefficients;

# Replot both vectors w and both decision boundaries, and add the ones computed without outliers
x_color = tls; x_color[tls==1] = "green"; x_color[tls==-1] = "red";
plot(X[,1],X[,2],xlab="X1",ylab="X2",col=x_color,asp=1);
arrows(inter[1],inter[2],inter[1]+30,inter[2]+30*wls[3]/wls[2],col="blue",lty=2);
abline(a=-wls[1]/wls[3],b=-wls[2]/wls[3],col="blue",lwd=2);
arrows(inter[1],inter[2],inter[1]+100,inter[2]+100*wlr[3]/wlr[2],col="black",lty=2);
abline(a=-wlr[1]/wlr[3],b=-wlr[2]/wlr[3],col="black",lwd=2);
  # plot w_LS starting from the intersection (no outliers)
arrows(inter[1],inter[2],inter[1]+100,inter[2]+100*wls2[3]/wls2[2],col="orange",lty=2);
  # plot w_LS^T x + w_0 = 0 (no outliers)
abline(a=-wls2[1]/wls2[3],b=-wls2[2]/wls2[3],col="orange",lty=3,lwd=2);
  # plot w_LR starting from the intersection (no outliers)
arrows(inter[1],inter[2],inter[1]+100,inter[2]+100*wlr2[3]/wlr2[2],col="purple",lty=2);
  # plot w_LR^T x + w_0 = 0 (no outliers)
abline(a=-wlr2[1]/wlr2[3],b=-wlr2[2]/wlr2[3],col="purple",lty=3,lwd=2);
  # plot legend
legend("topleft",legend=c("w_LS","decision boundary (LS)","w_LR","decision boundary (LR)","w_LS (no outliers)","boundary (LS, no outliers)","w_LR (no outliers)","boundary (LR, no outliers)"),col=c("blue","blue","black","black","orange","orange","purple","purple"),lty=c(2,1,2,1,3,2,3),lwd=c(2,2,2,2,2,2,2),cex=0.9);
Logistic regression is only concerned with the decision boundary. It does not try to find a $w$ that models all the points, instead it finds a $w$ such that the decision boundary is correct.
Exercise 4: Regularized linear regression

Generate the following data in R and visualize it.

N = 200;
Z = matrix(0,nrow=N,ncol=2);
Z[,1] = 20*runif(N);
Z[,2] = 20*runif(N);
X = matrix(0,nrow=N,ncol=6);
X[,1] = Z[,1] + (1/16)*rnorm(N);
X[,2] = -Z[,1] + (1/16)*rnorm(N);
X[,3] = Z[,1] + (1/16)*rnorm(N);
X[,4] = Z[,2] + (1/16)*rnorm(N);
X[,5] = -Z[,2] + (1/16)*rnorm(N);
X[,6] = Z[,2] + (1/16)*rnorm(N);
t = Z[,1] + 0.1*Z[,2] + rnorm(N);

Question 1

$$w_{R(\lambda)} = \frac{w_{LS}}{1 + \lambda}$$

Ridge

Question 2

# Standardize the data
for(k in 1:ncol(X)) X[,k] = (X[,k] - mean(X[,k])) / sd(X[,k]);
# Run Least squares
lsmodel = lm(t~X);
wls = lsmodel$coefficients;

# Plot the weights
cols = c("black","red","green","blue","orange","magenta");
barplot(wls,col=c("grey",cols));

The weights look completely wrong…

Question 3

To compute the bias $b_0$ and the variance $v_0$ of $w_{LS}$, we need to repeat the process for several datasets.

First, let us get rid of the low eigen values to reduce the variance and estimate the bias more accurately.
# Generate 10000 similar datasets and record the weights
T = 10000;
WLS0 = matrix(NA,nrow=T,ncol=ncol(X)+1);
for(i in 1:T) {
  Zi = matrix(0,nrow=N,ncol=2);
  Zi[,1] = 20*runif(N);
  Zi[,2] = 20*runif(N);
  Xi = matrix(0,nrow=N,ncol=6);
  Xi[,1] = Zi[,1] + (1/16)*rnorm(N);
  Xi[,2] = -Zi[,1] + (1/16)*rnorm(N);
  Xi[,3] = Zi[,1] + (1/16)*rnorm(N);
  Xi[,4] = Zi[,2] + (1/16)*rnorm(N);
  Xi[,5] = -Zi[,2] + (1/16)*rnorm(N);
  Xi[,6] = Zi[,2] + (1/16)*rnorm(N);
  for(k in 1:ncol(Xi)) Xi[,k] = (Xi[,k] - mean(Xi[,k])) / sd(Xi[,k]);
  ti = Zi[,1] + 0.1*Zi[,2] + rnorm(N);
  sv = eigen(t(cbind(rep(1,N),Xi))%*%cbind(rep(1,N),Xi));
  invcov = sv$vectors[,1:3] %*% diag(1/sv$values[1:3]) %*% t(sv$vectors[,1:3]);
  coeffi = invcov %*% t(cbind(rep(1,N),Xi)) %*% ti;
  WLS0[i,] = coeffi;
}

# Since the bias should be 0, we can use WLS0 to estimate the true estimator
truth = colMeans(WLS0);

Now, we use a more realistic case to compute the bias $b_0$ and the variance $v_0$.

# Generate 100 similar datasets and record the weights
T = 100;
WLS = matrix(NA,nrow=T,ncol=ncol(X)+1);
for(i in 1:T) {
  Zi = matrix(0,nrow=N,ncol=2);
  Zi[,1] = 20*runif(N);
  Zi[,2] = 20*runif(N);
  Xi = matrix(0,nrow=N,ncol=6);
  Xi[,1] = Zi[,1] + (1/16)*rnorm(N);
  Xi[,2] = -Zi[,1] + (1/16)*rnorm(N);
  Xi[,3] = Zi[,1] + (1/16)*rnorm(N);
  Xi[,4] = Zi[,2] + (1/16)*rnorm(N);
  Xi[,5] = -Zi[,2] + (1/16)*rnorm(N);
  Xi[,6] = Zi[,2] + (1/16)*rnorm(N);
\[ Xi[,6] = Zi[,2] + (1/16)\text{n} \text{rnorm}(N); \]
\[
\text{for}(k \text{ in } 1:\text{ncol}(X)) \text{ Xi[,}k\text{] = (Xi[,}k\text{] - mean(Xi[,}k\text{])) / sd(Xi[,}k\text{]);}\]
\[
ti = Zi[,1] + 0.1*Zi[,2] + \text{n} \text{rnorm}(N); \]
\[
\text{lsmodeli = } \text{l}m(ti\sim Xi); \]
\[
\text{WLS[i,]} = \text{lsmodeli}\$\text{coefficients};\]

# Compute the bias and the variance. The bias \( b_0 \) should be 0 therefore we can use WLS to estimate the true estimator
\[
b0 = \text{colMeans}(WLS) - \text{truth}; \]
\[
v0 = \text{diag}(\text{cov}(WLS));\]

Question 4

library(glmnet);

# Ridge regression
\[
\text{rmodel} = \text{glmnet}(x=X,y=t,\text{family}="\text{gaussian}";\text{standardize} = \text{FALSE}, \text{nlambda}=200, \text{alpha}=0);\]
\[
\text{plot(rmodel, xvar="lambda", lwd=2, col=cols);} \]
\[
\text{legend("topright", legend=paste("X", 1:ncol(X), sep=""), fill=cols);} \]
# LASSO
lmodel =
glmnet(x=X,y=t,family="gaussian",standardize=FALSE,nlambda=200,alpha=1);
plot(lmodel,xvar="lambda",lwd=2,col=cols);
legend("topright",legend=paste("X",1:ncol(X),sep=""),fill=cols);
# Elastic Net
enmodel =
glmnet(x=X,y=t,family="gaussian",standardize=FALSE,nlambda=200,alpha=0.5);
plot(enmodel,xvar="lambda",lwd=2,col=cols);
legend("topright",legend=paste("X",1:ncol(X),sep=""),fill=cols);
Question 5

# Generate test data
testN = 200;
testZ = matrix(0, nrow=testN, ncol=2);
testZ[,1] = 20*runif(testN);
testZ[,2] = 20*runif(testN);
testX = matrix(0, nrow=testN, ncol=6);
testX[,1] = testZ[,1] + (1/16)*rnorm(testN);
testX[,2] = -testZ[,1] + (1/16)*rnorm(testN);
testX[,3] = testZ[,1] + (1/16)*rnorm(testN);
testX[,4] = testZ[,2] + (1/16)*rnorm(testN);
testX[,5] = -testZ[,2] + (1/16)*rnorm(testN);
testX[,6] = testZ[,2] + (1/16)*rnorm(testN);
for(k in 1:ncol(testX)) testX[,k] = (testX[,k] - mean(testX[,k])) / sd(testX[,k]);
testt = testZ[,1] + 0.1*testZ[,2] + rnorm(testN);

# Predict the targets using Least squares and compute the mean-squared error
pls = wls[1] + (testX %*% wls[2:7]);
els = sum((testt - pls)^2) / testN;

# Predict the targets using Ridge regression and compute the mean-squared error for every \( \lambda \)
wr = rmodel$beta;
pr = matrix(NA,nrow=testN,ncol=length(rmodel$lambda));
er = vector("numeric",length=length(rmodel$lambda));
for(i in 1:ncol(pr)) {
    pr[,i] = rmodel$a0[i] + (testX %*% wr[,i]);
    er[i] = sum((testt - pr[,i])^2) / testN;
}
minstepr = which.min(er);

# Predict the targets using LASSO and compute the mean-squared error for every \( \lambda \)
wl = lmodel$beta;
pl = matrix(NA,nrow=testN,ncol=length(lmodel$lambda));
el = vector("numeric",length=length(lmodel$lambda));
for(i in 1:ncol(pl)) {
    pl[,i] = lmodel$a0[i] + (testX %*% wl[,i]);
    el[i] = sum((testt - pl[,i])^2) / testN;
}
minstepl = which.min(el);

# Predict the targets using Ridge regression, compute the mean-squared error for every \( \lambda \) and find the minimum
wen = enmodel$beta;
pen = matrix(NA,nrow=testN,ncol=length(enmodel$lambda));
een = vector("numeric",length=length(enmodel$lambda));
for(i in 1:ncol(pen)) {
    pen[,i] = enmodel$a0[i] + (testX %*% wen[,i]);
    een[i] = sum((testt - pen[,i])^2) / testN;
}
minstepen = which.min(een);

# Plot the results
mini = min(c(els,er,el,een));
maxi = max(c(els,er,el,een));
plot(c(1,nlambdas),c(els,els),ylim=c(mini,maxi),col="black",type="l",lwd=2,xlab="n
th step on the path",ylab="mean-squared-error",main="mean-squared-error");
points(er,col="blue",type="l",lwd=2);
abline(v=minstepr,col="blue",lty=2,lwd=2);
points(el,col="red",type="l",lwd=2);
abline(v=minstepl,col="red",lty=2,lwd=2);
points(een,col="green",type="l",lwd=2);
abline(v=minstepen,col="green",lty=2,lwd=2);
legend(x=100,y=32.57,legend=c("LS","Ridge","LASSO","Elastic Net"),fill=c("black","blue","red","green"));

Question 6

# Generate 100 similar datasets and record the weights
nlambdas=200;
WR = array(NA,dim=c(ncol(X),nlambdas,T));
RLAMBDA = matrix(NA,nrow=nlambdas+1,ncol=T);
WL = array(NA,dim=c(ncol(X),nlambdas,T));
LLAMBDA = matrix(NA,nrow=nlambdas+1,ncol=T);
WEN = array(NA,dim=c(ncol(X),nlambdas,T));
ENLAMBDA = matrix(NA,nrow=nlambdas+1,ncol=T);
for(i in 1:T) {
    # Generate data
    Zi = matrix(0,nrow=N,ncol=2);
    Zi[,1] = 20*runif(N);
    Zi[,2] = 20*runif(N);
    Xi = matrix(0,nrow=N,ncol=6);
    Xi[,1] = Zi[,1] + (1/16)*rnorm(N);
    Xi[,2] = -Zi[,1] + (1/16)*rnorm(N);
    Xi[,3] = Zi[,1] + (1/16)*rnorm(N);
    Xi[,4] = Zi[,2] + (1/16)*rnorm(N);
    Xi[,5] = -Zi[,2] + (1/16)*rnorm(N);
    Xi[,6] = Zi[,2] + (1/16)*rnorm(N);
    for(k in 1:ncol(Xi)) Xi[,k] = (Xi[,k] - mean(Xi[,k])) / sd(Xi[,k]);
    ti = Zi[,1] + 0.1*Zi[,2] + rnorm(N);

    # Run Ridge regression
    rmodeli =
glmnet(x=Xi,y=ti,family="gaussian",standardize=FALSE,nlambda=nlambdas,alpha=0);
    RLAMBDA[1:length(rmodeli$lambda),i] = rmodeli$lambda;
    RLAMBDA[nlambdas+1,i] = length(rmodeli$lambda);
    for(j in 1:ncol(X)) WR[j,1:ncol(rmodeli$beta),i] = rmodeli$beta[,j];

    # Run LASSO
    lmodeli =
glmnet(x=Xi,y=ti,family="gaussian",standardize=FALSE,nlambda=nlambdas,alpha=1);
    LLAMBDA[1:length(lmodeli$lambda),i] = lmodeli$lambda;
    LLAMBDA[nlambdas+1,i] = length(lmodeli$lambda);
    for(j in 1:ncol(X)) WL[j,1:ncol(lmodeli$beta),i] = lmodeli$beta[,j];

    # Run Elastic Net
    enmodeli =
glmnet(x=Xi,y=ti,family="gaussian",standardize=FALSE,nlambda=nlambdas,alpha=0.5);
    ENLAMBDA[1:length(enmodeli$lambda),i] = enmodeli$lambda;
    ENLAMBDA[nlambdas+1,i] = length(enmodeli$lambda);
for(j in 1:ncol(X)) WEN[j,1:ncol(enmodeli$beta),i] = enmodeli$beta[j,];
}

# Compute the bias and variance for Ridge regression for every \( \lambda \)
 mWR = apply(WR,FUN=mean,MARGIN=c(1,2));
 rb = mWR -
 matrix(rep(truth[2:length(truth)],nrow=ncol(X),ncol=ncol(mWR));
 rv = (WR[,1] - mWR)^2; for(i in 2:T) rv = rv + (WR[,i] - mWR)^2; rv = rv / (T-1);

# Compute the bias and variance for LASSO for every \( \lambda \)
 lkeep =1:min(LLAMBDA[nlambdas+1,]);
 mWL = apply(WL[lkeep,],FUN=mean,MARGIN=c(1,2));
 lb = mWL -
 matrix(rep(truth[2:length(truth)],length(lkeep)),nrow=ncol(X),ncol=length(lkeep));
 lv = (WL[,lkeep,1] - mWL)^2; for(i in 2:T) lv = lv + (WL[,lkeep,i] - mWL)^2; lv = lv / (T-1);

# Compute the bias and variance for Elastic Net for every \( \lambda \)
 enkeep =1:min(ENLAMBDA[nlambdas+1,]);
 mWEN = apply(WEN[,enkeep,],FUN=mean,MARGIN=c(1,2));
 enb = mWEN -
 matrix(rep(truth[2:length(truth)],length(enkeep)),nrow=ncol(X),ncol=length(enkeep));
 env = (WEN[,enkeep,1] - mWEN)^2; for(i in 2:T) env = env + (WEN[,enkeep,i] - mWEN)^2; env = env / (T-1);

# Plot the bias for w1, the coefficient associated to X1
 mini = min(c(b0[2],rb[1,],lb[1,],enb[1,]));
 maxi = max(c(b0[2],rb[1,],lb[1,],enb[1,]));
 plot(c(1,nlambdas),c(b0[2],b0[2]),main="bias for w1",xlab="nth step on the path",ylab="bias",col="black",ylim=c(mini,maxi),type="l",lwd=2);
 points(rb[1,],col="blue",type="l",lwd=2);
 points(lb[1,],col="red",type="l",lwd=2);
 points(enb[1,],col="green",type="l",lwd=2);
 legend("bottomright",legend=c("LS","Ridge","LASSO","Elastic Net"),fill=c("black","blue","red","green"));
# Plot the variance for w1, the coefficient associated to X1
mini = min(c(v0[2],rv[1,],lv[1,],env[1,]));
maxi = max(c(v0[2],rv[1,],lv[1,],env[1,]));
plot(c(1,nlambdas),log(c(v0[2],v0[2])),main="log(variance) for w1",xlab="nth step on the path",ylab="log(variance)",col="black",ylim=c(max(-10,log(mini)),log(maxi)),type="l",lwd=2);
points(log(rv[1,]),col="blue",type="l",lwd=2);
points(log(lv[1,]),col="red",type="l",lwd=2);
points(log(env[1,]),col="green",type="l",lwd=2);
legend("right", legend=c("LS", "Ridge", "LASSO", "Elastic Net"), fill=c("black", "blue", "red", "green"));

log(variance) for w1

log(variance)

nth step on the path

nth step on the path
Exercise 5: Kernels and Kernel regression

Generate the following data in R and visualize it.

N = 500;
x = seq(-10,10,length.out=N); x = x - mean(x);
t = x^2 - 2*x + 20*rnorm(N); t = t - mean(t);
testx = seq(-10,10,by=0.1);

Question 1
Refer to the slides.

Question 2
Refer to the slides.

Question 3

# Run Least squares
lsmodel = lm(t~x);
wls = lsmodel$coefficients;

# Predict new values
testt = wls[2]*testx + wls[1];

# Plot the results
plot(x,t,main="linear regression",xlab="x",ylab="t");
points(testx,testt,col="blue");
Question 4

# Compute the linear kernel matrix
K = x%*%t(x);

# Compute the coefficients
s = eigen(K);
Km1 = (1/s$values[1]) * (matrix(s$vectors[,1],nrow=N,ncol=1) %*% matrix(s$vectors[,1],nrow=1,ncol=N));
a = Km1 %*% t;

# Compute the test kernel matrix
testK = ((testx%*%t(x)) + 1)^2;

# Compute the predictions
ktestt = testK %*% a;
# Compute the polynomial kernel matrix
pK = ((x%*%t(x)) + 1)^2;

# Compute the coefficients
ps = eigen(pK);
pKm1 = ps$vectors[,1:3] %*% diag(1/ps$values[1:3]) %*% t(ps$vectors[,1:3]);
pKm1 = ps$vectors[,1:3] %*% diag(1/ps$values[1:3]) %*% t(ps$vectors[,1:3]);

# Compute the test kernel matrix
testpK = ((testx%*%t(x)) + 1)^2;

# Compute the predictions
pktestt = testpK %*% pa;

# Plot the results
plot(x,t,main="kernel regression",xlab="x",ylab="t");
points(testx,ktestt,col="blue");
points(testx,pktestt,col="red");
legend("top",legend=c("linear","polynomial"),fill=c("blue","red"));
kernel regression

- Linear
- Polynomial
Exercise 6: Kernels and Kernel classification

Generate the following data in R and visualize it with colours (points of class 1 in blue and points of class 2 in red for example).

\[ N = 1000; \sigma = 0.5; \]
\[ x_{11} = -1 + \sigma \cdot \text{rnorm}(N/4); x_{12} = 1 + \sigma \cdot \text{rnorm}(N/4); \]
\[ x_{21} = 1 + \sigma \cdot \text{rnorm}(N/4); x_{22} = -1 + \sigma \cdot \text{rnorm}(N/4); \]
\[ x_{31} = -1 + \sigma \cdot \text{rnorm}(N/4); x_{32} = -1 + \sigma \cdot \text{rnorm}(N/4); \]
\[ x_{41} = 1 + \sigma \cdot \text{rnorm}(N/4); x_{42} = 1 + \sigma \cdot \text{rnorm}(N/4); \]
\[ X = \text{cbind}(c(x_{11},x_{21},x_{31},x_{41}),c(x_{12},x_{22},x_{32},x_{42})); \]
\[ \text{for}(i \text{ in } 1:2) \ X[,i] = X[,i] - \text{mean}(X[,i]); \]
\[ t = \text{c(rep}(1,N/2),\text{rep}(-1,N/2)); \]

Question 1

# Run logistic regression
\[ tlr = t; tlr[t==-1] = 0; \]
\[ \text{lrmodel} = \text{glm}(tlr~X,\text{family}="\text{binomial}"); \]
\[ \text{wlr} = \text{lrmodel}\$\text{coefficients}; \]

# Plot the data and the scores
\[ \text{xgrid} = \text{seq}(-3,3,\text{length.out}=200); \]
\[ \text{ygrid} = \text{seq}(-3.4,3.4,\text{length.out}=220); \]
\[ \text{z} = \text{matrix}(\text{NA},\text{nrow}=\text{length(xgrid)},\text{ncol}=\text{length(ygrid)}); \]
\[ \text{for}(i \text{ in } 1:\text{length(xgrid)}) \{ \]
\[ \text{for}(j \text{ in } 1:\text{length(ygrid)}) \{ \]
\[ \text{z}[i,j] = wlr[2]*xgrid[i] + wlr[3]*ygrid[j] + wlr[1]; \]
\[ \} \]
\[ \text{maxi} = \text{max}(\text{abs}(z)); \]
\[ \text{x\_color} = t; \text{x\_color}[t==1] = "green"; \text{x\_color}[t==-1] = "cyan"; \]
\[ \text{col\_pal} = \text{colorRampPalette}(\text{c}('dark blue','white','dark red')); \]
\[ \text{filled.contour}(x=xgrid,y=ygrid,z=z,\text{levels}=\text{seq}(-\text{maxi},\text{maxi},\text{length.out}=52),\text{asp}=1,\text{col}=\text{col\_pal}(51),\text{xlab}="X1",\text{ylab}="X2", \text{main}="Logistic regression",\text{key.title}=\text{title}(\text{main}="w.x + w0"),\text{plot.axes}=\{ \]
\[ \text{points}(X[,1],X[,2],\text{col}=\text{x\_color}); \text{axis}(1); \text{axis}(2); \} ); \]
Question 2

# Compute the linear kernel matrix
K = X%*%t(X);

# Compute the coefficients
s = eigen(K);
Km1 = s$vectors[,1:2] %*% diag(1/s$values[1:2]) %*% t(s$vectors[,1:2]);
a = Km1 %*% t;

# Plot the data and the scores
xgrid = seq(-3,3,length.out=200);
ygrid = seq(-3.4,3.4,length.out=220);
z = matrix(NA,nrow=length(xgrid),ncol=length(ygrid));
for(i in 1:length(xgrid)) {  
}
testpoints = cbind(rep(xgrid[i], length(ygrid)), ygrid);
testK = testpoints%*%t(X);
z[i,] = as.vector(testK %*% a);
}

maxi = max(abs(z));
x_color = t; x_color[t==1] = "green"; x_color[t==-1] = "cyan";
filled.contour(x=xgrid, y=ygrid, z=z, levels=seq(-maxi, maxi, length.out=52), asp=1, col=col_pal(51), xlab="X1", ylab="X2", main="Linear kernel classification", key.title=title(main="testK * a"), plot.axes={ points(X[,1], X[,2], col=x_color); axis(1); axis(2); });

Question 3

# Compute the polynomial kernel matrix
K = ((X%*%t(X)) + 1)^2;

# Compute the coefficients
s = eigen(K);
Km1 = s$vectors[,1:6] %*% diag(1/s$values[1:6]) %*% t(s$vectors[,1:6]);
a = Km1 %*% t;

# Plot the data and the scores
xgrid = seq(-3,3,length.out=200);
ygrid = seq(-3.4,3.4,length.out=220);
z = matrix(NA,nrow=length(xgrid),ncol=length(ygrid));
for(i in 1:length(xgrid)) {
    testpoints = cbind(rep(xgrid[i],length(ygrid)),ygrid);
    testK = ((testpoints%*%t(X)) + 1)^2;
    z[i,] = as.vector(testK %*% a);
}
maxi = max(abs(z));
x_color = t; x_color[t==1] = "green"; x_color[t==-1] = "cyan";
filled.contour(x=xgrid,y=ygrid,z=z,levels=seq(-maxi,maxi,length.out=52),asp=1,col=col_pal(51),xlab="X1",ylab="X2",
main="Polynomial kernel classification",key.title=title(main="testK * a"),plot.axes={ points(X[,1],X[,2],col=x_color); axis(1); axis(2); });
Question 4

# Compute the Gaussian kernel matrix
diff = matrix(NA, nrow=N, ncol=N);
for(i in 1:N) {
    rowSums((matrix(rep(X[i,], each=N), nrow=N, ncol=2) - X)^2);
}
width = 0.5;
K = exp(-diff / (2*width^2));

# Compute the coefficients
s = eigen(K); #Km1 = solve(K);
Km1 = s$vectors[,1:100] %*% diag(1/s$values[1:100]) %*% t(s$vectors[,1:100]);
a = Km1 %*% t;
# Plot the data and the scores
xgrid = seq(-3,3,length.out=200);
ygrid = seq(-3.4,3.4,length.out=220);
z = matrix(NA,nrow=length(xgrid),ncol=length(ygrid));
for(i in 1:length(xgrid)) {
    testpoints = cbind(rep(xgrid[i],length(ygrid)),ygrid);
    testdiff = matrix(NA,nrow=nrow(testpoints),ncol=N);
    for(k in 1:nrow(testpoints)) {
        testdiff[k,] = rowSums((matrix(rep(testpoints[k,],each=N),nrow=N,ncol=2) - X)^2);
    }
    testK = exp(-testdiff / (2*width^2));
    z[i,] = as.vector(testK %*% a);
}
maxi = max(abs(z));
x_color = t; x_color[t==1] = "green"; x_color[t==-1] = "cyan";
filled.contour(x=xgrid,y=ygrid,z=z,levels=seq(-maxi,maxi,length.out=52),asp=1,col=col_pal(51),xlab="X1",ylab="X2",main="Gaussian kernel classification",key.title=title(main="testK * a"),plot.axes={ points(X[,1],X[,2],col=x_color); axis(1); axis(2); });
Question 5

1) Dual formulation

When we saw kernel regression, we rewrote $w$ as a linear combination of the training points $X = \{x_n\}$, so that $w^T y$ became a linear combination of $x_n^T y$. To kernelise PCA, you have to find where the kernel may replace the dot product $x_n^T y$ in the equations.

In PCA, we want to find a new vector $u$, the principal component, and project our data on it

$$y = \text{transformed}(x) = x^T u$$

So now imagine that we write $u$ as a linear combination of the training points $X$. 
\[ u = \sum_{n=1}^{N} a_n x_n = X^T a \]

This gives us:
\[ y = \text{transformed}(x) = x^T \left( \sum_{n=1}^{N} a_n x_n \right) = \sum_{n=1}^{N} a_n x_n^T x_n \]

We can now replace the dot product with any kernel we like \( \varnothing \):
\[ y = \text{transformed}(x) = \sum_{n=1}^{N} a_n K(x, x_n) \]

Now instead of learning the principal component \( u \), we learn the coefficients \( a \). Remember that \( u \) is learnt to maximise the variance and we showed that \( u \) has to be an eigenvector of the correlation matrix. What about \( a \)?

2) First point of view

The variance is
\[
\sum_{n=1}^{N} y_n^T y_n = \sum_{n=1}^{N} (x_n^T u)^T (x_n^T u) = \sum_{n=1}^{N} u^T x_n x_n^T u
\]
\[
= u^T \left( \sum_{n=1}^{N} x_n x_n^T \right) u = u^T (X^T X) u = (X^T a)^T (X^T X)(X^T a)
\]
\[
= a^T X^T X X^T a = a^T (XX^T)^2 a = a^T (XX^T) a = a^T K^2 a
\]

where \( K \) is the kernel matrix between training points.

We want the resulting \( u \) to be normalised
\[ u^T u = 1 \iff (X^T a)^T (X^T a) = 1 \iff a^T XX^T a = 1 \iff a^T (XX^T) a = 1 \iff a^T K a = \]

Maximising the variance with respect to \( a \) therefore means
\[ \frac{d}{da} \left( a^T K^2 a + \lambda \left( 1 - a^T K a \right) \right) = 0 \iff 2K^2 a - 2\lambda K a = 0 \iff K^2 a = \lambda K a \]

\[ \iff K a = \lambda a \]

So, much in the same way, \( a \) will be an eigenvector of the kernel matrix \( K \).

3) second point of view

If \( C \) is the covariance matrix, then

\[ C u = \lambda u \]

\[ (X^T X) u = \lambda u \]

\[ (X^T X)(X^T a) = \lambda (X^T a) \]

\[ X (X^T X)(X^T a) = \lambda X (X^T a) \]

\[ (XX^T)(XX^T)a = \lambda (XX^T)a \]

\[ (XX^T)^2 a = \lambda (XX^T)a \]

\[ (XX^T)a = \lambda a \]

\[ K a = \lambda a \]

So, much in the same way, \( a \) will be an eigenvector of the kernel matrix \( K \).

4) Which eigen vector?

The variance will then be

\[ \sum_{n=1}^{N} y_n^T y_n = a^T K^2 a = a^T K (K a) = \lambda a^T K a = \lambda \]

So, again, \( a \) must be the eigen vector corresponding to the largest eigen value.

5) How about test points?

How to project a new point \( x \)?

\[ y = \text{transformed}(x) = \sum_{n=1}^{N} a_n K(x, x_n) \]