

**Solution 1:**

- (a) The hyperplane is given by

$$\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_0 = 0.$$

Plugging in the values for the  $\theta$ s and solving for  $x_2$ , we get the decision boundary as function of  $x_1$ :

$$x_2 = -x_1 + 2.$$

- (b)  $(0.5, 0.5), (0, 1), (0, 3), (3, 0)$  are support vectors with slack value of  $\zeta^{(i)} = 0$  as they lie on the margin hyperplanes.

$(0, 0)$  is also a support vector with slack value of  $\zeta^{(i)} = 3$ .

Derivation: We use the equation from the constraint  $y_i(\theta^\top \mathbf{x}_i + \theta_0) \geq 1 - \zeta^{(i)}$  and plug in the values for the margin-violating point  $y_i = 1, x_1 = 0, x_2 = 0$ :

$$y_i(x_1 + x_2 - 2) = 1(0 + 0 - 2) \geq 1 - \zeta^{(i)} \Rightarrow \zeta^{(i)} \geq 3$$

- (c) Using  $\mathbf{x}^{(i)} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$ :

$$d(f, \mathbf{x}^{(i)}) = \frac{y^{(i)} f(\mathbf{x}^{(i)})}{\|\theta\|_2} = \frac{-1(0.5 + 0.5 - 2)}{\sqrt{2}} = \frac{1}{\sqrt{2}}$$

The distance is the same for all non-margin-violating support vectors.

- (d) Change point  $(0, 0)$  from  $+$  to  $-$  or remove  $(0, 0)$ .

**Solution 2:**

- Implementation of the PEGASOS algorithm:

```
#' @param y outcome vector
#' @param X design matrix (including a column of 1s for the intercept)
#' @param nr_iter number of iterations for the algorithm
#' @param theta starting values for thetas
#' @param lambda penalty parameter
#' @param alpha step size for weight decay
pegasos_linear <- function(
  y,
  X,
  nr_iter = 50000,
  theta = rnorm(ncol(X)),
  lambda = 1,
  alpha = 0.01)
{
  t <- 1
  n <- NROW(y)
```

```

while(t <= nr_iter){

  f_current = X%*%theta
  i <- sample(1:n, 1)

  # update
  theta <- (1 - lambda * alpha) * theta
  # add second term if within margin
  if(y[i]*f_current[i] < 1) theta <- theta + alpha * y[i]*X[i,]

  t <- t + 1

}

return(theta)
}

```

- Check on a simple example

```

## Check on a simple example
## -----

set.seed(2L)

C = 1

library(mlbench)
library(kernlab)
data = mlbench.twonorm(n = 100, d = 2)

data = as.data.frame(data)
X = as.matrix(data[, 1:2])
y = data$classes
par(mar = c(5,4,4,6))
plot(x = data$x.1, y = data$x.2, pch = ifelse(data$classes == 1, "-", "+"), col = "black",
      xlab = "x1", ylab = "x2")

# recode y
y = ifelse(y == "2", 1, -1)
mod_pegasos = pegasos_linear(y, cbind(1,X), lambda = C/(NROW(y)))

# Add estimated decision boundary:
abline(a = - mod_pegasos[1] / mod_pegasos[2],
       b = - mod_pegasos[2] / mod_pegasos[3], col = "#D55E00")

# Compare to logistic regression:
mod_logreg = glm(classes ~ ., data = data, family = binomial())
abline(a = - coef(mod_logreg)[1] / coef(mod_logreg)[2],
       b = - coef(mod_logreg)[2] / coef(mod_logreg)[3], col = "#56B4E9",
       lty = 3, lwd = 2)

# decision values
f_pegasos = cbind(1,X) %*% mod_pegasos

# How many wrong classified examples?
table(sign(f_pegasos * y))

```

```
##
## -1 1
## 5 95

## compare to kernlab. we CANNOT expect a PERFECT match
## -----

mod_kernlab = ksvm(classes~.,
  data = data,
  kernel = "vanilladot",
  C = C,
  kpar = list(),
  scaled = FALSE)
f_kernlab = predict(mod_kernlab, newdata = data, type = "decision")
# How many wrong classified examples?
table(sign(f_kernlab * y))

##
## -1 1
## 5 95

# compare outputs
print(range(abs(f_kernlab - f_pegasos)))

## [1] 0.00014996 0.38049736

# compare coeffs
rbind(
  mod_pegasos,
  mod_kernlab = c(mod_kernlab@b,
    (params <- colSums(X[mod_kernlab@SVindex, ] *
      mod_kernlab@alpha[[1]] *
      y[mod_kernlab@SVindex])))
)

##
## mod_pegasos -0.05743352 -1.347267 -0.7917586
## mod_kernlab 0.09763532 -1.263707 -0.7747026

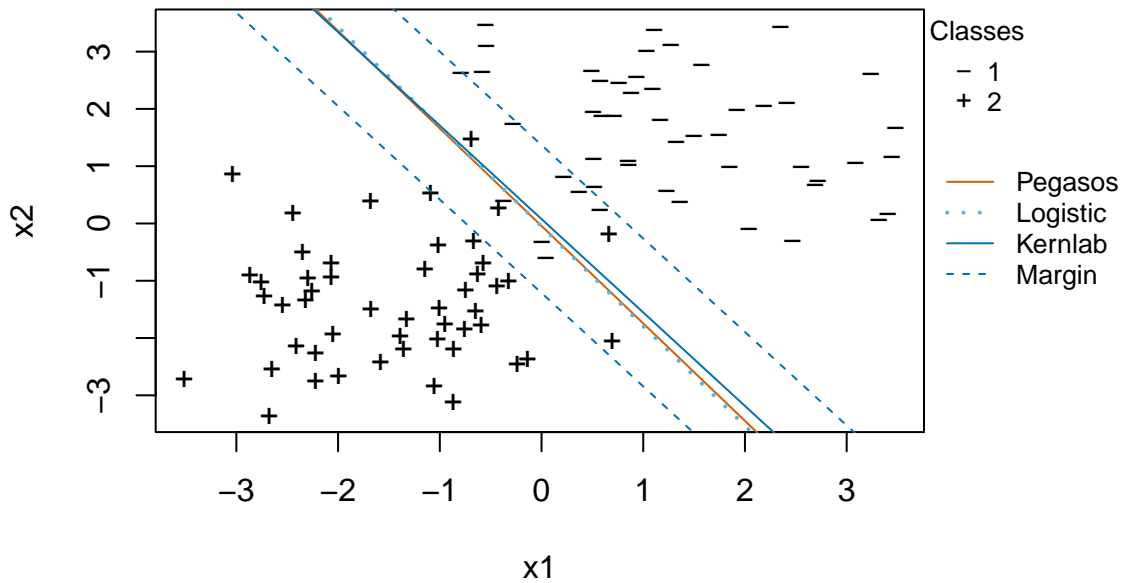
# seems we were reasonably close

# recompute margin
margin = 1 / sqrt(sum(params^2))

# compute value of intercept shift (the margin shift is in orthogonal direction
# to the decision boundary, so this has to be transformed first)
m = - params[1] / params[2]
t_0 = margin * m / (cos(atan(1/m)))

# add margins to visualization:
abline(a = - mod_kernlab@b / params[1],
  b = m, col = "#0072B2")
abline(a = - mod_kernlab@b / params[1] + t_0,
  b = m, col = "#0072B2", lty = 2)
abline(a = - mod_kernlab@b / params[1] - t_0,
  b = m, col = "#0072B2", lty = 2)
```

```
# add legends
legend(par('usr')[2], par('usr')[4], , bty='n', xpd=NA, legend=c("1","2"),
      pch=c("-", "+"), title="Classes", cex = 0.8)
legend(par('usr')[2], 1.8, , bty='n', xpd=NA,
      legend=c("Pegasos", "Logistic", "Kernlab", "Margin"),
      lty=c(1,3,1,2),
      col = c("#D55E00", "#56B4E9", "#0072B2", "#0072B2"),
      title="", cex = 0.8, lwd = c(1,2,1,1))
```



### Solution 3:

The polynomial kernel is defined as

$$k(x, \tilde{x}) = (x^T \tilde{x} + b)^d.$$

Furthermore, assume  $x \in \mathbb{R}^2$  and  $d = 2$ .

- (a) Derive the explicit feature map  $\phi$  taking into account that the following equation holds:

$$k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle$$

**Solution:**

$$\begin{aligned}
k(x, \tilde{x}) &= \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} + b \right)^2 \\
&= (x_1 \tilde{x}_1 + x_2 \tilde{x}_2 + b)^2 \\
&= (x_1 \tilde{x}_1 + x_2 \tilde{x}_2)^2 + 2(x_1 \tilde{x}_1 + x_2 \tilde{x}_2)b + b^2 \\
&= x_1^2 \tilde{x}_1^2 + 2x_1 \tilde{x}_1 x_2 \tilde{x}_2 + x_2^2 \tilde{x}_2^2 + 2bx_1 \tilde{x}_1 + 2bx_2 \tilde{x}_2 + b^2 \\
&= \left\langle \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \\ \sqrt{2}bx_1 \\ \sqrt{2}bx_2 \\ b \end{pmatrix}, \begin{pmatrix} \tilde{x}_1^2 \\ \sqrt{2}\tilde{x}_1 \tilde{x}_2 \\ \tilde{x}_2^2 \\ \sqrt{2}b\tilde{x}_1 \\ \sqrt{2}b\tilde{x}_2 \\ b \end{pmatrix} \right\rangle \\
&= \langle \phi(x), \phi(\tilde{x}) \rangle
\end{aligned}$$

(b) Describe the main differences between the kernel method and the explicit feature map.

**Solution:**

Using the kernel method reduces the computational costs of computing the scalar product in the higher-dimensional features space after calculating the feature map.

**Solution 4:**

(a) Prior distribution (assuming the same notation as in the lecture):

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$

with  $\mathbf{m} = m(\mathbf{x})$  and  $\mathbf{K}$  defined by the entries  $\mathbf{K}_{ij} = k(x_i, x_j)$ . NB: Note the (in-)finite Gaussian property of a GP.

(b) Note that the posterior distribution  $\mathbf{f}|\mathbf{y}, \mathbf{x}$  in this case is different from the one of  $\mathbf{f}_*|\mathbf{x}_*, \mathbf{x}, \mathbf{y}$  and also from the marginal distribution of  $\mathbf{y} \sim \mathcal{N}(\mathbf{m}, \mathbf{K} + \sigma^2 \mathbf{I})$ ! We have:

$$\begin{aligned}
p(\mathbf{f}|\mathbf{y}) &\propto p(\mathbf{y}|\mathbf{f}) \cdot p(\mathbf{f}) \\
&\propto \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{f})^\top (\sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \mathbf{f})\right) \cdot \exp\left(-\frac{1}{2}(\mathbf{f} - \mathbf{m})^\top \mathbf{K}^{-1} (\mathbf{f} - \mathbf{m})\right) \\
&\propto \exp\left(-\frac{1}{2}\left\{\mathbf{f}^\top \underbrace{((\sigma^2 \mathbf{I})^{-1} + \mathbf{K}^{-1})}_{=: \mathbf{K}_{post}^{-1}} \mathbf{f} - 2\mathbf{f}^\top \underbrace{((\sigma^2 \mathbf{I})^{-1} \mathbf{y} + \mathbf{K}^{-1} \mathbf{m})}_{=: \tilde{\mathbf{f}}}\right\}\right) \quad (1) \\
&\propto \exp\left(-\frac{1}{2}\{\mathbf{f}^\top \mathbf{K}_{post}^{-1} \mathbf{f} - 2\mathbf{f}^\top \tilde{\mathbf{f}}\}\right)
\end{aligned}$$

by removing all constant factors that do not depend on  $\mathbf{f}$  as we only need to know the density up to a constant of proportionality. By extending the proportionality, we can get a quadratic form in  $\mathbf{f}$ :

$$\begin{aligned}
p(\mathbf{f}|\mathbf{y}) &\propto \exp\left(-\frac{1}{2}\{\mathbf{f}^\top \mathbf{K}_{post}^{-1} \mathbf{f} - 2\mathbf{f}^\top \tilde{\mathbf{f}}\}\right) \\
&\propto \exp\left(-\frac{1}{2}\{\mathbf{f}^\top \mathbf{K}_{post}^{-1} \mathbf{f} - 2\mathbf{f}^\top \mathbf{K}_{post}^{-1} \underbrace{\mathbf{K}_{post} \tilde{\mathbf{f}}}_{:= \mathbf{f}_{post}}\}\right) \quad (2) \\
&\propto \exp\left(-\frac{1}{2}(\mathbf{f} - \mathbf{f}_{post})^\top \mathbf{K}_{post}^{-1} (\mathbf{f} - \mathbf{f}_{post})\right)
\end{aligned}$$

which is the so-called *kernel* of a multivariate normal distribution  $\mathcal{N}(\mathbf{f}_{post}, \mathbf{K}_{post})$ , i.e.,  $\mathbf{f}|\mathbf{y} \sim \mathcal{N}(\mathbf{f}_{post}, \mathbf{K}_{post})$ .

- (c) In order to get the posterior predictive distribution for a new sample  $x_*$  from the same data-generating process, we could derive

$$p(y_*|x_*, \mathbf{y}, \mathbf{x}) = \int p(y_*|x_*, \mathbf{x}, \mathbf{y}, \mathbf{f}) \cdot p(\mathbf{f}|\mathbf{y}, \mathbf{x}) d\mathbf{f}.$$

This is feasible but cumbersome. Alternatively, we can make use of the fact that the joint distribution of  $\mathbf{y}$  and  $y_*$  is known (cf. slides on noisy GP):

$$\begin{pmatrix} \mathbf{y} \\ y_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{m} \\ m_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} + \sigma^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*^\top & K_{**} \end{pmatrix}\right),$$

with  $m_* = m(x_*)$ ,  $\mathbf{K}_* = k(x_*, \mathbf{x})$  and  $K_{**} = k(x_*, x_*)$ . The conditional distribution can then be derived using the rule of conditioning for Gaussian distributions:

$$y_*|x_*, \mathbf{x}, \mathbf{y} \sim \mathcal{N}(m_* + \mathbf{K}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \mathbf{m}), K_{**} - \mathbf{K}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{K}_*).$$

- (d) To implement a GP with squared exponential kernel and  $\ell = 1$ , we need the inverse of  $\mathbf{K}$ .  $\mathbf{x}$  being a vector implies that we have only one feature and thus the entries of our matrix  $\mathbf{K}$  are

$$\mathbf{K} = \begin{pmatrix} 1 & \exp(-0.5(x^{(1)} - x^{(2)})^2) \\ \exp(-0.5(x^{(2)} - x^{(1)})^2) & 1 \end{pmatrix}.$$

The inverse of  $\mathbf{K}$  is then given by

$$\frac{1}{1 - \exp(-(x^{(1)} - x^{(2)})^2)} \begin{pmatrix} 1 & -\exp(-0.5(x^{(1)} - x^{(2)})^2) \\ -\exp(-0.5(x^{(2)} - x^{(1)})^2) & 1 \end{pmatrix}.$$

If we have a noisy GP, we would have to add  $\sigma^2 \mathbf{I}_2$  to  $\mathbf{K}$  with resulting inverse

$$\mathbf{K}_y^{-1} = \frac{1}{(1 + \sigma^2)^2 - \exp(-(x^{(1)} - x^{(2)})^2)} \begin{pmatrix} 1 + \sigma^2 & -\exp(-0.5(x^{(1)} - x^{(2)})^2) \\ -\exp(-0.5(x^{(2)} - x^{(1)})^2) & 1 + \sigma^2 \end{pmatrix}.$$

Assuming a zero mean GP, we can derive  $\frac{\partial \mathbf{K}_y}{\partial \theta}$  with  $\theta = \sigma^2$ , which gives us the identity matrix. We can thus maximize the marginal likelihood (slide on *Gaussian Process Training*), by finding  $\sigma^2$  that yields

$$\text{tr}(\mathbf{K}_y^{-1} \mathbf{y} \mathbf{y}^\top \mathbf{K}_y^{-1} - \mathbf{K}_y^{-1}) = 0.$$

This can be solved analytically (though quite tedious). We will use a root-finding function for this. For the posterior predictive distribution we can make use of the results from the previous exercise.

```
library(kernlab)

# set seed, define n, true (unknown) sigma
set.seed(4212)
n <- 2
sigma <- 1

# define kernel with l = 1
kernel_fun <- function(x)
  kernelMatrix(kernel = rbfdot(sigma = 1/2),
               x = x)
kernel_fun_pred <- function(x,y)
  kernelMatrix(kernel = rbfdot(sigma = 1/2),
               x = x, y = y)

# draw data according to the generating process:
x <- rnorm(n)
K <- kernel_fun(x)
K_y <- K + diag(rep(sigma^2,2))
(y <- t(mvtnorm::rmvnorm(1, sigma = K_y)))
```

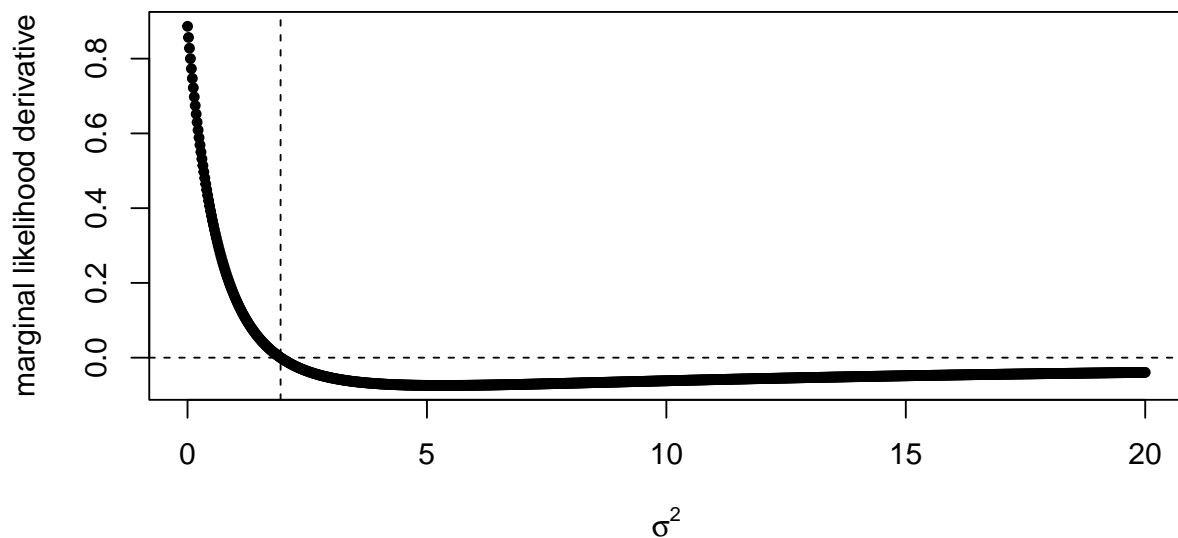
```
##           [,1]
## [1,] 2.012317
## [2,] 1.866819

# function to find the best sigma^2
root_fun <- function(sigmaSq){
  K_y_inv <- solve(K + diag(rep(sigmaSq,2)))
  0.5*sum(diag(K_y_inv*%*%y*%*%t(y)*%*%K_y_inv - K_y_inv))
}

# get the best sigma
(bestSigmaSq <- uniroot(f = root_fun, interval = c(0,20)))$root

## [1] 1.943684

# plot the optimization problem and best sigma
possible_sigvals <- seq(0.001,20,l=1000)
plot(possible_sigvals, sapply(possible_sigvals, root_fun),
     xlab = expression(sigma^2), ylab = "marginal likelihood derivative",
     pch = 20)
abline(h=0, lty=2)
abline(v=bestSigmaSq$root, lty=2)
```



```
# function to draw samples from the predictive posterior
draw_from_pred_posterior <- function(number_samples, y, x, xstar, sigmaSq = 1)
{
  # invert noisy K
  K_y_inv <- solve(kernel_fun(x) + diag(rep(sigmaSq,2)))
  # get the other K's for new data
  Kstar <- kernel_fun_pred(x,xstar)
  Kstarstar <- kernel_fun(xstar)
  # draw samples according to Ex. (d)
  rnorm(number_samples,
```

```

    mean = as.numeric(t(Kstar) %*% K_y_inv %*% y),
    sd = sqrt(as.numeric(Kstarstar - t(Kstar) %*% K_y_inv %*% Kstar))
  )
}

# draw enough samples to get a feeling for the distribution
samples_posterior <-
  draw_from_pred_posterior(number_samples = 1000, sigmaSq = bestSigmaSq$root,
    y = y, x = x, xstar = 0)

# plot the distribution
hist(samples_posterior, breaks=50, xlab=expression(y["*"]^b))

```

**Histogram of samples\_posterior**

