Solution 1: SVM – Support Vectors and Separating Hyperplane

(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 θs 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: SVM – Optimization

- Implementation of the PEGASOS algorithm:

```r
#' @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)
```

```r
  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

```r
## 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))
```

```r
##
## -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])))
)

##                               x.1        x.2
## 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)
```

```r
# 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))
```