



University  
of Manitoba

# **Matrix methods – Support vector machines**

**MATH 2740 – Mathematics of Data Science – Lecture 12**

**Julien Arino**

`julien.arino@umanitoba.ca`

**Department of Mathematics @ University of Manitoba**

**Fall 202X**

The University of Manitoba campuses are located on original lands of Anishinaabeg, Ininew, Anisininew, Dakota and Dene peoples, and on the National Homeland of the Red River Métis. We respect the Treaties that were made on these territories, we acknowledge the harms and mistakes of the past, and we dedicate ourselves to move forward in partnership with Indigenous communities in a spirit of Reconciliation and collaboration.

# Outline

**Support vector machines (SVM)**

**Linear SVM**

**Guess who's coming to dinner!**

**Soft-margin SVM**



# **Support vector machines (SVM)**

**Linear SVM**

**Guess who's coming to dinner!**

**Soft-margin SVM**

# Support vector machines (SVM)

We are given a training dataset of  $n$  points of the form

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$$

where  $\mathbf{x}_i \in \mathbb{R}^p$  and  $y_i = \{-1, 1\}$ . The value of  $y_i$  indicates the class to which the point  $\mathbf{x}_i$  belongs

We want to find a **surface**  $\mathcal{S}$  in  $\mathbb{R}^p$  that divides the group of points into two subgroups

Once we have this surface  $\mathcal{S}$ , any additional point that is added to the set can then be *classified* as belonging to either one of the sets depending on where it is with respect to the surface  $\mathcal{S}$



**Support vector machines (SVM)**

**Linear SVM**

**Guess who's coming to dinner!**

**Soft-margin SVM**

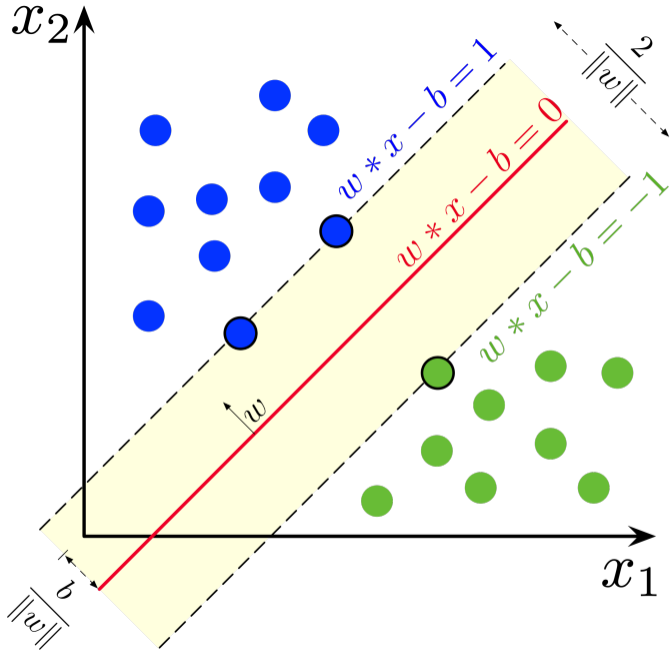
# Linear SVM

We are given a training dataset of  $n$  points of the form

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$$

where  $\mathbf{x}_i \in \mathbb{R}^p$  and  $y_i = \{-1, 1\}$ . The value of  $y_i$  indicates the class to which the point  $\mathbf{x}_i$  belongs

***Linear SVM*** – Find the “maximum-margin hyperplane” that divides the group of points  $\mathbf{x}_i$  for which  $y_i = 1$  from the group of points for which  $y_i = -1$ , which is such that the distance between the hyperplane and the nearest point  $\mathbf{x}_i$  from either group is maximized.



Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are the **support vectors**

Any **hyperplane** can be written as the set of points  $\mathbf{x}$  satisfying

$$\mathbf{w}^T \mathbf{x} - b = 0$$

where  $\mathbf{w}$  is the (not necessarily normalized) **normal vector** to the hyperplane (if the hyperplane has equation  $a_1 z_1 + \dots + a_p z_p = c$ , then  $(a_1, \dots, a_n)$  is normal to the hyperplane)

The parameter  $b/\|\mathbf{w}\|$  determines the offset of the hyperplane from the origin along the normal vector  $\mathbf{w}$

Remark: a hyperplane defined thusly is not a subspace of  $\mathbb{R}^p$  unless  $b = 0$ . We can of course transform the data so that it is...

# Linearly separable points

Let  $X_1$  and  $X_2$  be two sets of points in  $\mathbb{R}^p$

Then  $X_1$  and  $X_2$  are **linearly separable** if there exist  $w_1, w_2, \dots, w_p, k \in \mathbb{R}$  such that

- ▶ every point  $x \in X_1$  satisfies  $\sum_{i=1}^p w_i x_i > k$
- ▶ every point  $x \in X_2$  satisfies  $\sum_{i=1}^p w_i x_i < k$

where  $x_i$  is the  $i$ th component of  $x$

# Hard-margin SVM

If the training data is **linearly separable**, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible

The region bounded by these two hyperplanes is called the “margin”, and the maximum-margin hyperplane is the hyperplane that lies halfway between them

With a normalized or standardized dataset, these hyperplanes can be described by the equations

- ▶  $\mathbf{w}^T \mathbf{x} - b = 1$  (anything on or above this boundary is of one class, with label 1)
- ▶  $\mathbf{w}^T \mathbf{x} - b = -1$  (anything on or below this boundary is of the other class, with label -1)

Distance between these two hyperplanes is  $2/\|\mathbf{w}\|$

$\Rightarrow$  to maximize the distance between the planes we want to minimize  $\|\mathbf{w}\|$

The distance is computed using the distance from a point to a plane equation

We must also prevent data points from falling into the margin, so we add the following constraint: for each  $i$  either

$$\mathbf{w}^T \mathbf{x}_i - b \geq 1, \text{ if } y_i = 1$$

or

$$\mathbf{w}^T \mathbf{x}_i - b \leq -1, \text{ if } y_i = -1$$

(Each data point must lie on the correct side of the margin)

This can be rewritten as

$$y_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n$$

or

$$y_i(\mathbf{w}^T \mathbf{x}_i - b) - 1 \geq 0, \quad \text{for all } 1 \leq i \leq n$$

We get the optimization problem:

$$\text{Minimize } \|\mathbf{w}\| \text{ subject to } y_i(\mathbf{w}^T \mathbf{x}_i - b) - 1 \geq 0 \text{ for } i = 1, \dots, n$$

The  $\mathbf{w}$  and  $b$  that solve this problem determine the classifier,  $\mathbf{x} \mapsto \text{sgn}(\mathbf{w}^T \mathbf{x} - b)$  where  $\text{sgn}(\cdot)$  is the **sign function**.

The maximum-margin hyperplane is completely determined by those  $\mathbf{x}_i$  that lie nearest to it

These  $\mathbf{x}_i$  are the **support vectors**



**Support vector machines (SVM)**

**Linear SVM**

**Guess who's coming to dinner!**

**Soft-margin SVM**

# Our goal

Recall that our goal is

$$\textit{Minimize } \|\mathbf{w}\| \textit{ subject to } y_i(\mathbf{w}^T \mathbf{x}_i - b) - 1 \geq 0 \textit{ for } i = 1, \dots, n$$

## Writing the goal in terms of Lagrange multipliers

Using Lagrange multipliers  $\lambda_1, \dots, \lambda_n$ , we have the function

$$L_P := F(\mathbf{w}, b, \lambda_1, \dots, \lambda_n) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \lambda_i y_i (\mathbf{x}_i \mathbf{w} + b) + \sum_{i=1}^n \lambda_i$$

We have as many Lagrange multipliers as there are data points: there are that many inequalities that must be satisfied

The aim is to minimise  $L_P$  with respect to  $\mathbf{w}$  and  $b$  while the derivatives of  $L_P$  w.r.t.  $\lambda_i$  vanish and the  $\lambda_i \geq 0$ ,  $i = 1, \dots, n$

# Lagrange multipliers

We have already seen Lagrange multipliers

- ▶ in the intro math slides
- ▶ when we were studying PCA

# Maximisation using Lagrange multipliers (V1.0)

We want the max of  $f(x_1, \dots, x_n)$  under the constraint  $g(x_1, \dots, x_n) = k$

1. Solve

$$\begin{aligned}\nabla f(x_1, \dots, x_n) &= \lambda \nabla g(x_1, \dots, x_n) \\ g(x_1, \dots, x_n) &= k\end{aligned}$$

where  $\nabla = (\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n})$  is the **gradient operator**

2. Plug all solutions into  $f(x_1, \dots, x_n)$  and find maximum values (provided values exist and  $\nabla g \neq \mathbf{0}$  there)

$\lambda$  is the **Lagrange multiplier**

# The gradient

$f : \mathbb{R}^n \rightarrow \mathbb{R}$  function of several variables,  $\nabla = \left( \frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right)$  the gradient operator

Then

$$\nabla f = \left( \frac{\partial}{\partial x_1} f, \dots, \frac{\partial}{\partial x_n} f \right)$$

So  $\nabla f$  is a *vector-valued* function,  $\nabla f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ ; also written as

$$\nabla f = f_{x_1}(x_1, \dots, x_n) \mathbf{e}_1 + \dots + f_{x_n}(x_1, \dots, x_n) \mathbf{e}_n$$

where  $f_{x_i}$  is the partial derivative of  $f$  with respect to  $x_i$  and  $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$  is the standard basis of  $\mathbb{R}^n$

## Lagrange multipliers (V2.0)

However, the problem we were considering then involved a single multiplier  $\lambda$

Here we want  $\lambda_1, \dots, \lambda_n$

## Theorem 88 (Lagrange multiplier theorem)

*Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  be the objective function,  $g: \mathbb{R}^n \rightarrow \mathbb{R}^c$  be the constraints function, both being  $C^1$ . Consider the optimisation problem*

$$\begin{aligned} & \text{maximize } f(x) \\ & \text{subject to } g(x) = 0 \end{aligned}$$

*Let  $x^*$  be an optimal solution to the optimization problem, such that  $\text{rank}(Dg(x^*)) = c < n$ , where  $Dg(x^*)$  denotes the matrix of partial derivatives*

$$[\partial g_j / \partial x_k]$$

*Then there exists a unique Lagrange multiplier  $\lambda^* \in \mathbb{R}^c$  such that*

$$Df(x^*) = \lambda^{*T} Dg(x^*)$$

## Lagrange multipliers (V3.0)

Here we want  $\lambda_1, \dots, \lambda_n$

But we also are looking for  $\lambda_i \geq 0$

So we need to consider the so-called Karush-Kuhn-Tucker (KKT) conditions

# Karush-Kuhn-Tucker (KKT) conditions

Consider the optimisation problem

$$\begin{array}{ll}\text{maximize} & f(x) \\ \text{subject to} & g_i(x) \leq 0 \\ & h_i(x) = 0\end{array}$$

Form the Lagrangian

$$L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \mu^T \mathbf{g}(\mathbf{x}) + \lambda^T \mathbf{h}(\mathbf{x})$$

## Theorem 89

*If  $(\mathbf{x}^*, \mu^*)$  is a saddle point of  $L(\mathbf{x}, \mu)$  in  $\mathbf{x} \in \mathbf{X}$ ,  $\mu \geq \mathbf{0}$ , then  $\mathbf{x}^*$  is an optimal vector for the above optimization problem*

*Suppose that  $f(\mathbf{x})$  and  $g_i(\mathbf{x})$ ,  $i = 1, \dots, m$ , are convex in  $\mathbf{x}$  and that there exists  $\mathbf{x}_0 \in \mathbf{X}$  such that  $\mathbf{g}(\mathbf{x}_0) < 0$*

*Then with an optimal vector  $\mathbf{x}^*$  for the above optimization problem there is associated a non-negative vector  $\mu^*$  such that  $L(\mathbf{x}^*, \mu^*)$  is a saddle point of  $L(\mathbf{x}, \mu)$*

## KKT conditions

$$\frac{\partial}{\partial \mathbf{w}_\nu} L_P = \mathbf{w}_\nu - \sum_i^n \lambda_i y_i \mathbf{x}_{i\nu} = 0 \quad \nu = 1, \dots, p$$

$$\frac{\partial}{\partial b} L_P = - \sum_{i=1}^n \lambda_i y_i = 0$$

$$y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1 \geq 0 \quad i = 1, \dots, n$$

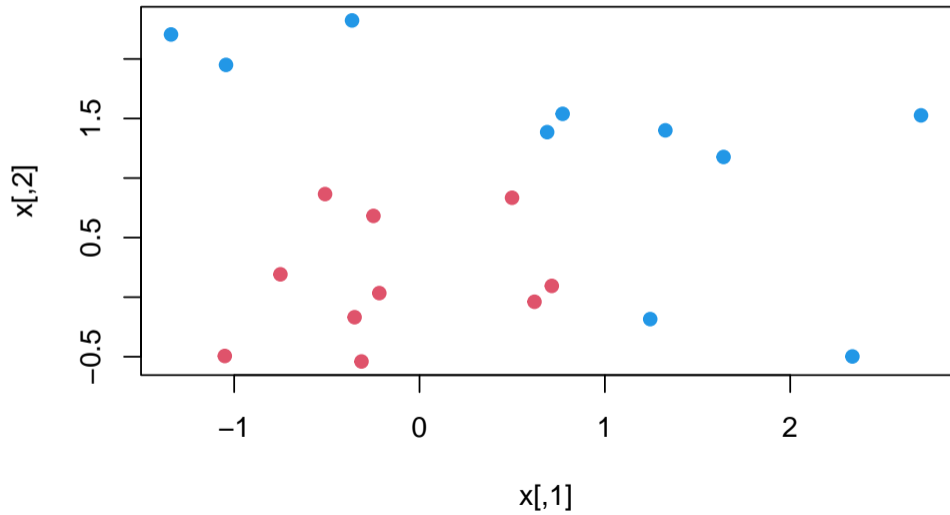
$$\lambda_i \geq 0 \quad i = 1, \dots, n$$

$$\lambda_i(y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1) = 0 \quad i = 1, \dots, n$$

# Numerical example

Example from here

```
set.seed(10111)
x = matrix(rnorm(40), 20, 2)
y = rep(c(-1, 1), c(10, 10))
x[y == 1,] = x[y == 1,] + 1
plot(x, col = y + 3, pch = 19)
```



## Linear SVM in matrix form (doing things “by hand”)

Let  $X \in \mathbb{R}^{n \times p}$  have rows  $\mathbf{x}_i^T$  and labels  $\mathbf{y} \in \{-1, 1\}^n$

Soft-margin (hinge-loss) primal objective in vector/matrix form:

$$\min_{\mathbf{w}, b} \lambda \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{x}_i^T \mathbf{w} - b))$$

We optimize this with a simple subgradient descent using base  $\mathbb{R}$

# Fit SVM via hinge-loss gradient descent

```
X <- x
Y <- as.numeric(y)
n <- nrow(X); p <- ncol(X)

# Initialize parameters
w <- rep(0, p)
b <- 0
lambda <- 0.05    # Regularization strength
lr <- 0.1          # learning rate
epochs <- 5000
```

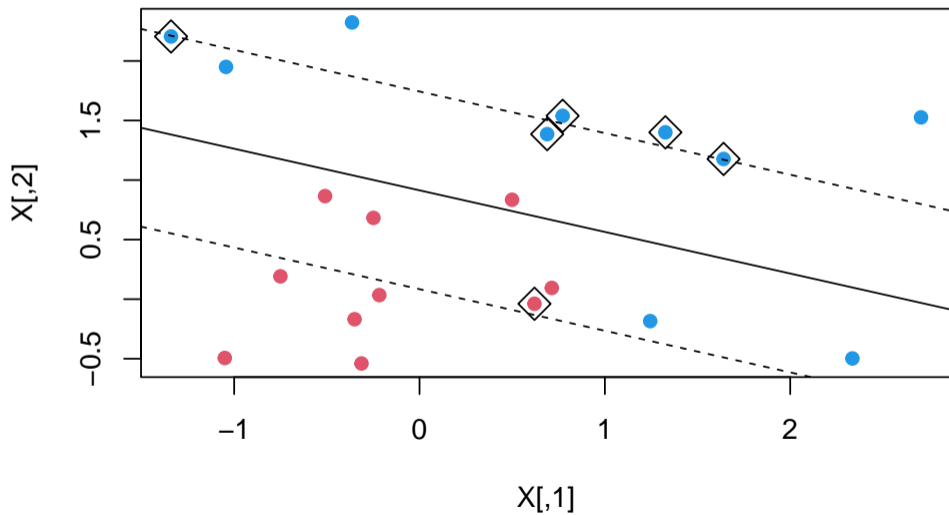
```

for (t in 1:epochs) {
  margins <- Y * (as.vector(X %*% w) - b)
  active <- margins < 1 # points violating margin
  if (any(active)) {
    grad_w <- 2 * lambda * w -
      colSums( (Y[active] * X[active, , drop = FALSE]) ) / n
    grad_b <- sum(Y[active]) / n
  } else {
    grad_w <- 2 * lambda * w
    grad_b <- 0
  }
  # parameter update
  w <- w - lr * grad_w
  b <- b - lr * grad_b
  # mild decay for stability
  if (t %% 1000 == 0) lr <- lr * 0.9
}

```

```
cat("w:", paste(round(w, 3), collapse = ", "),  
    " b:", round(b, 3), "\n")
```

```
## w: 0.421, 1.205 b: 1.101
```

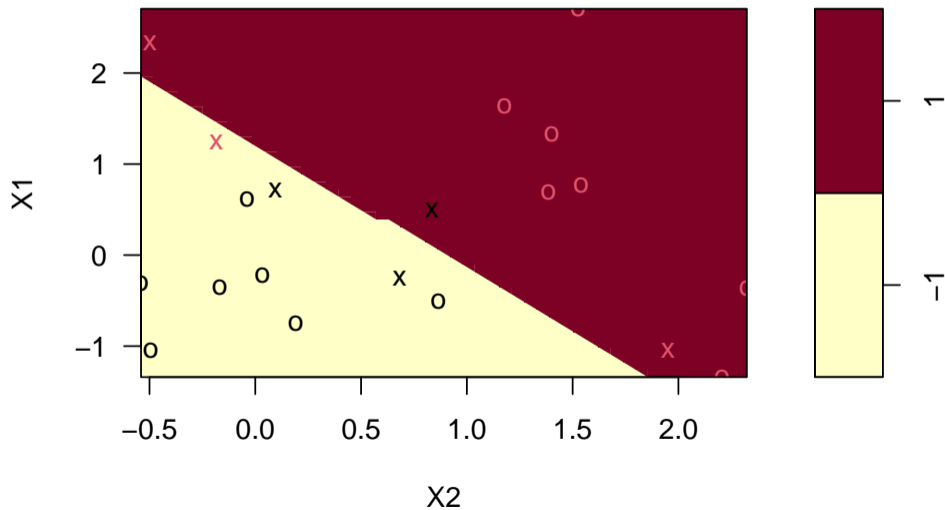


```
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10,
             scale = FALSE)
print(svmfit)

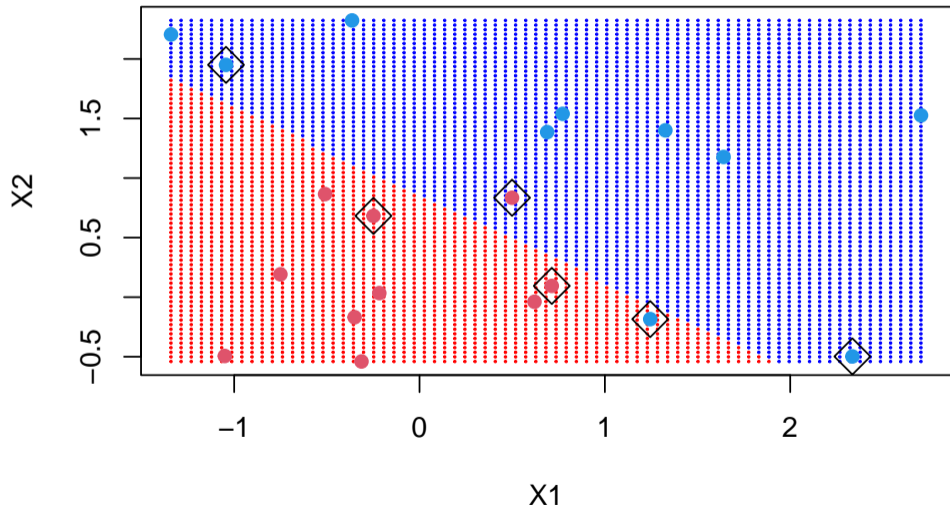
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FA
##
##
## Parameters:
##      SVM-Type:  C-classification
##      SVM-Kernel: linear
##           cost:  10
##
## Number of Support Vectors:  6

plot(svmfit, dat)
```

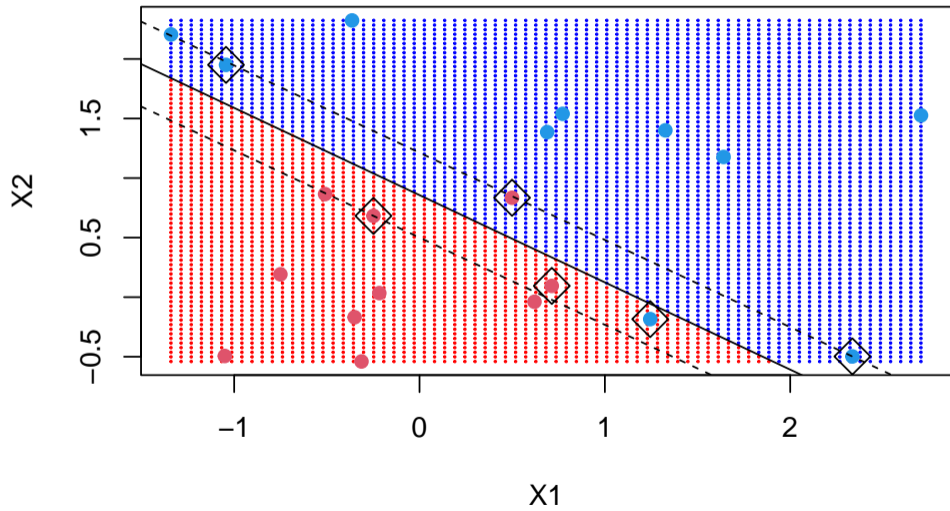
**SVM classification plot**



```
make.grid = function(x, n = 75) {  
  grange = apply(x, 2, range)  
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n)  
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n)  
  expand.grid(X1 = x1, X2 = x2)  
}  
xgrid = make.grid(x)  
ygrid = predict(svmfit, xgrid)  
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = .2)  
points(x, col = y + 3, pch = 19)  
points(x[svmfit$index,], pch = 5, cex = 2)
```



```
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 = svmfit$rho
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```





Support vector machines (SVM)

Linear SVM

Guess who's coming to dinner!

**Soft-margin SVM**

## Soft-margin SVM

To extend SVM to cases in which the data are not linearly separable, the **hinge loss** function is helpful

$$\max \left( 0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b) \right)$$

$y_i$  is the  $i$ th target (i.e., in this case, 1 or -1), and  $\mathbf{w}^T \mathbf{x}_i - b$  is the  $i$ -th output

This function is zero if the constraint is satisfied, in other words, if  $\mathbf{x}_i$  lies on the correct side of the margin

For data on the wrong side of the margin, the function's value is proportional to the distance from the margin

The goal of the optimization then is to minimize

$$\lambda \|\mathbf{w}\|^2 + \left[ \frac{1}{n} \sum_{i=1}^n \max \left( 0, 1 - y_i (\mathbf{w}^T \mathbf{x}_i - b) \right) \right]$$

where the parameter  $\lambda > 0$  determines the trade-off between increasing the margin size and ensuring that the  $\mathbf{x}_i$  lie on the correct side of the margin

Thus, for sufficiently small values of  $\lambda$ , it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not

## Soft-margin: slack variables and primal form

Another common formulation introduces slack variables  $\xi_i \geq 0$  to allow violations of the margin. The soft-margin (primal) problem is:

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to

$$y_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \dots, n$$

Here  $C > 0$  controls the penalty for margin violations. Large  $C$  — fewer violations (closer to hard-margin). Small  $C$  — wider margin but more violations

## Hinge loss and relation to slack variables

The hinge loss for a single sample is  $\ell_{\text{hinge}}(t) = \max(0, 1 - t)$  with  $t = y_i(\mathbf{w}^T \mathbf{x}_i - b)$

Minimizing the empirical hinge loss plus a regularizer is equivalent to the slack-variable primal above when the loss is replaced by the constrained slack formulation. Concretely,

$$\frac{1}{n} \sum_{i=1}^n \ell_{\text{hinge}}(y_i(\mathbf{w}^T \mathbf{x}_i - b)) + \lambda \|\mathbf{w}\|^2$$

and the primal with  $C$  are related: roughly  $C \approx 1/(2n\lambda)$  (constants depend on exact formulation)

## Worked R example: soft-margin behaviour

We'll simulate a slightly noisy two-class dataset and fit linear SVMs with different cost ( $C$ ) values to see the effect on the margin and support vectors

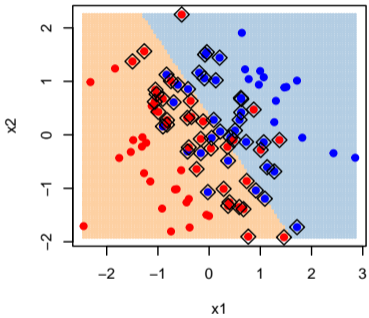
## R: simulate data and fit SVMs

```
set.seed(2025)
library(e1071)
# simulate 2D data with some overlap
n = 100
x = matrix(rnorm(2*n), n, 2)
y = ifelse(x[,1] + 0.6*x[,2] + rnorm(n, sd=0.8) > 0, 1, -1)
dat = data.frame(x1 = x[,1], x2 = x[,2], y = as.factor(y))

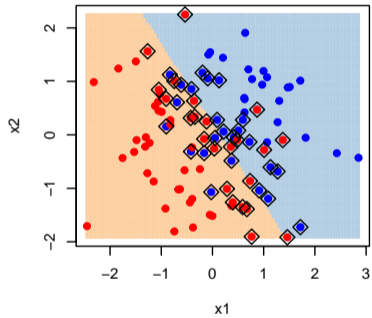
# fit linear SVMs with different cost values
svm_lowC  = svm(y ~ ., data = dat, kernel = 'linear', cost = 0.1, scale = FA
svm_medC  = svm(y ~ ., data = dat, kernel = 'linear', cost = 1,    scale = FA
svm_highC = svm(y ~ ., data = dat, kernel = 'linear', cost = 100, scale = FA

svm_list = list(low = svm_lowC, med = svm_medC, high = svm_highC)
```

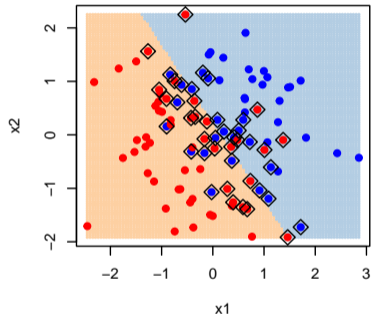
**cost = 0.1 (soft)**



**cost = 1**



**cost = 100 (harder)**



## Interpretation

Observe how smaller cost (left) yields a smoother, wider-margin decision boundary with more misclassified points but fewer support vectors sometimes; larger cost (right) fits data more tightly, reducing training error but possibly overfitting

Use cross-validation to select  $C$  in practice (next example uses `tune()`)

## R: tuning cost with cross-validation

```
set.seed(2025)
tune.out = tune(svm, y ~ ., data = dat,
               kernel = 'linear',
               ranges = list(cost = c(0.01, 0.1, 1, 10, 100)))
print(tune.out)
best = tune.out$best.model
```

**SVM classification plot**

