

# Support Vector Machines

Last modified: 2026-10-26

```
# The Python chunks on this page import numpy; reticulate provisions it.
# The Julia chunks run through JuliaCall.
library(reticulate)
py_require(c("numpy", "pandas"))
fmt2 <- function(v) formatC(v, format = "f", digits = 2)
fmt1 <- function(v) formatC(v, format = "f", digits = 1)
fmt3s <- function(v) formatC(signif(v, 3), format = "fg", digits = 3)
```

## i Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes, and drawing on Logan Sizemore's Spring 2025 lecture on support vector machines. Where this chapter goes past that lecture — the 1995 soft margin, and crediting Alexey Chervonenkis alongside Vapnik for the 1964 paper, which the lecture attributes to Vapnik alone — it follows the cited papers rather than the lecture. The lecture's gradient-descent derivation of the representer theorem for a general loss appears in the dual-problem section, as Theorem 2.2.

## 1 Hard-margin SVMs: the primal problem

Support vector machines come from the maximal-margin classifier Vladimir Vapnik and Alexey Chervonenkis published jointly in 1964 [1]. The kernel trick that extends them to non-linear classification arrived in 1992 [2], and the soft margin that lets them tolerate non-separable data shortly after [3].

SVMs were the dominant classifier of the late 1990s and early 2000s. Two things drove that: they are unusually easy to analyze in theory, and they performed well on the data sets of that time. Neural networks have since replaced them in most applications, but SVMs are still the right tool for some, and two of the ideas they introduce — the **max-margin objective** and the **kernel trick** — stayed in use after the method that introduced them faded.

We build the method up by starting with the most restrictive version and removing one assumption at a time. The starting point is a *supervised*, *linear*, *binary* classifier for *separable* classes, and we will drop three of those four assumptions in this order:

1. Separable, which soft-margin SVMs drop.
2. Linear, which the kernel trick drops.
3. Binary, which multiclass SVMs drop.

Supervised is the one assumption that stays.

**Exercise 1.1** (Reuse a fitted linear classifier). Everything in this lecture is built on a linear classifier these notes have already fit. A logistic regression on two inputs<sup>a</sup>  $x = (x_1, x_2)$  returned  $w = (2, -1)$  and  $b = 1$ .

Write the equation of its separating line in terms of  $x_1$  and  $x_2$ , and give two points on it.

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

#### Solution

*Solution 1.1.* The separating line of that fitted classifier is the set where  $h(x)$  vanishes,

$$\{x : h(x) = w \cdot x + b = 2x_1 - x_2 + 1 = 0\}.$$

Setting  $x_1 = 0$  gives  $x_2 = 1$ , so  $(0, 1)$  is on the line. Setting  $x_1 = 1$  gives  $x_2 = 3$ , so  $(1, 3)$  is on it too.

**Exercise 1.2** (Reuse a fitted linear classifier (part 2)). Use the setup of Exercise 1.1. Predict the class of each point  $(1, 1)$ ,  $(-1, 2)$  and  $(0, -1)$ , taking the positive class when  $h(x) > 0$ .

#### Solution

*Solution 1.2.* Evaluate  $h(x) = 2x_1 - x_2 + 1$  at each point:

- At  $(1, 1)$  it is  $2 - 1 + 1 = 2 > 0$ , so the prediction is the positive class.
- At  $(-1, 2)$  it is  $-2 - 2 + 1 = -3 < 0$ , so the prediction is the negative class.
- At  $(0, -1)$  it is  $0 + 1 + 1 = 2 > 0$ , so the prediction is the positive class.

**Exercise 1.3** (Reuse a fitted linear classifier (part 3)). Use the setup of Exercise 1.1. A support vector machine will use a function of the same form. Which of these does it keep from logistic regression, and which does it replace: the parameters, the separating hyperplane, the prediction rule, the objective that picks  $w$  and  $b$ ?

#### Solution

*Solution 1.3.* The SVM keeps the parameters  $w$  and  $b$ , the separating hyperplane  $\{x : w \cdot x + b = 0\}$ , and the prediction rule based on the sign of  $h(x)$ . It replaces the objective. Logistic regression picks  $w$  and  $b$  by minimizing the average cross-entropy<sup>a</sup> over the training set. The SVM picks them by making the margin as large as possible, which is a different style of optimization problem.

Everything about the *model* carries over; only the *objective* changes.

<sup>a</sup><https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

**Definition 1.1** (Decision boundary). A classifier's **decision boundary** is the set of inputs where its prediction changes from one class to another [4]. For a linear classifier it is the hyperplane  $\{x : h(x) = 0\}$ .

**Definition 1.2** (Decision region). The decision boundaries divide the input space into **decision regions**, each the set of inputs the classifier assigns to one class. For a linear binary classifier there are two:  $\{x : h(x) > 0\}$  and  $\{x : h(x) < 0\}$ .

**Example 1.1** (A linear classifier for fuel economy). In the `Auto` data [5], label a car  $+1$  if its `mpg` is above the median and  $-1$  otherwise. Classify it from its standardized `weight` and `horsepower` with the linear rule  $h(x) = w \cdot x + b$ , where  $w$  is the difference of the two class means and the hyperplane  $h(x) = 0$  is the perpendicular bisector of the segment joining them. Count how many cars fall on each side of that hyperplane, by actual label:

## 1.1 Python

```
import numpy as np
import pandas as pd

auto_dr = pd.read_csv("data/auto.csv").dropna().reset_index(drop=True)
# Standardize, so that weight and horsepower are on comparable scales.
Z_dr = auto_dr[["weight", "horsepower"]].to_numpy(float)
Z_dr = (Z_dr - Z_dr.mean(axis=0)) / Z_dr.std(axis=0, ddof=1)
y_dr = np.where(auto_dr["mpg"] > auto_dr["mpg"].median(), 1.0, -1.0)

# The boundary passes through the midpoint of the class means and is perpendicular to their difference
mean_hi_dr, mean_lo_dr = Z_dr[y_dr > 0].mean(axis=0), Z_dr[y_dr < 0].mean(axis=0)
w_dr = mean_hi_dr - mean_lo_dr
b_dr = -w_dr @ (mean_hi_dr + mean_lo_dr) / 2
h_dr = Z_dr @ w_dr + b_dr      # the sign of h says which decision region a car is in

print(pd.crosstab(np.where(h_dr > 0, "h > 0", "h < 0"), np.where(y_dr > 0, "high mpg", "low mpg"),
                  rownames=["region"], colnames=["actual"]))
```

|       | high mpg | low mpg |
|-------|----------|---------|
| h < 0 | 8        | 151     |
| h > 0 | 188      | 45      |

## 1.2 R

```
auto_dr <- tidyr::drop_na(read.csv("data/auto.csv"))
# Standardize, so that weight and horsepower are on comparable scales.
Z_dr <- scale(as.matrix(auto_dr[, c("weight", "horsepower")]))
y_dr <- ifelse(auto_dr$mpg > median(auto_dr$mpg), 1, -1)

# The boundary passes through the midpoint of the class means and is perpendicular to their difference
mean_hi_dr <- colMeans(Z_dr[y_dr > 0, ])
mean_lo_dr <- colMeans(Z_dr[y_dr < 0, ])
w_dr <- mean_hi_dr - mean_lo_dr
b_dr <- -sum(w_dr * (mean_hi_dr + mean_lo_dr)) / 2
h_dr <- drop(Z_dr %*% w_dr) + b_dr      # the sign of h says which decision region a car is in

table(region = ifelse(h_dr > 0, "h > 0", "h < 0"),
      actual = ifelse(y_dr > 0, "high mpg", "low mpg"))
```

|       | high mpg | low mpg |
|-------|----------|---------|
| h < 0 | 8        | 151     |
| h > 0 | 188      | 45      |

The line  $h(x) = 0$  is the decision boundary of Definition 1.1, and it cuts the plane into the two decision regions of Definition 1.2. The region  $h > 0$  holds 233 cars, 81% of them high-mpg; the region  $h < 0$  holds 159 cars, 95% of them low-mpg. The 53 cars on the wrong side of the line are the classifier's errors.

**Exercise 1.4** (Say which training sets a hyperplane can split). Logistic regression returns a hyperplane for any training set, but on some training sets every hyperplane misclassifies a point. Sketch a two-class training set in  $\mathbb{R}^2$  that some line classifies with no mistakes.

### Solution

*Solution 1.4.* Any two clouds of points that sit on opposite sides of some line, such as one class clustered at the upper right and the other at the lower left (Figure 1).

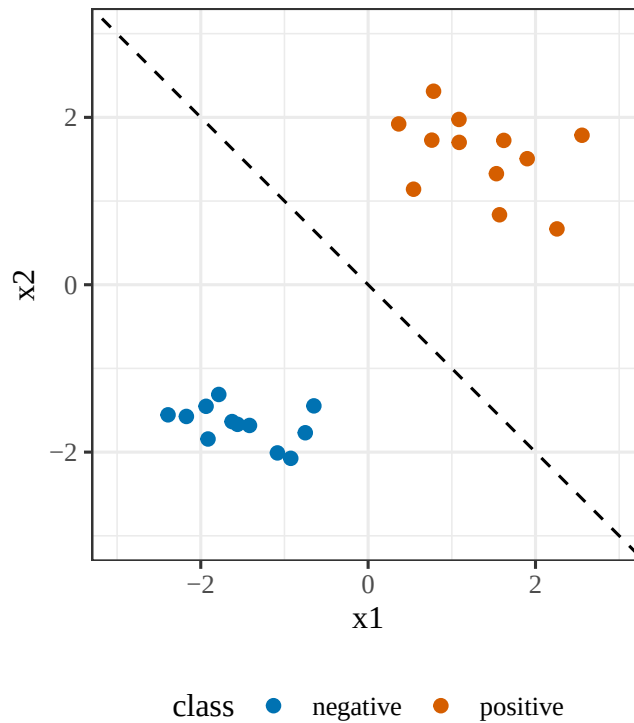


Figure 1: Two clouds of points that a line separates. The dashed line has every positive point on one side and every negative point on the other.

**Exercise 1.5** (Say which training sets a hyperplane can split (part 2)). Use the setup of Exercise 1.4. Sketch one that no line can.

### Solution

*Solution 1.5.* The four points of the XOR problem from logistic regression: one class at  $(1, 1)$  and  $(-1, -1)$ , the other at  $(1, -1)$  and  $(-1, 1)$ . Any line that puts both points of one class on the same side also puts at least one point of the other class there (Figure 2).

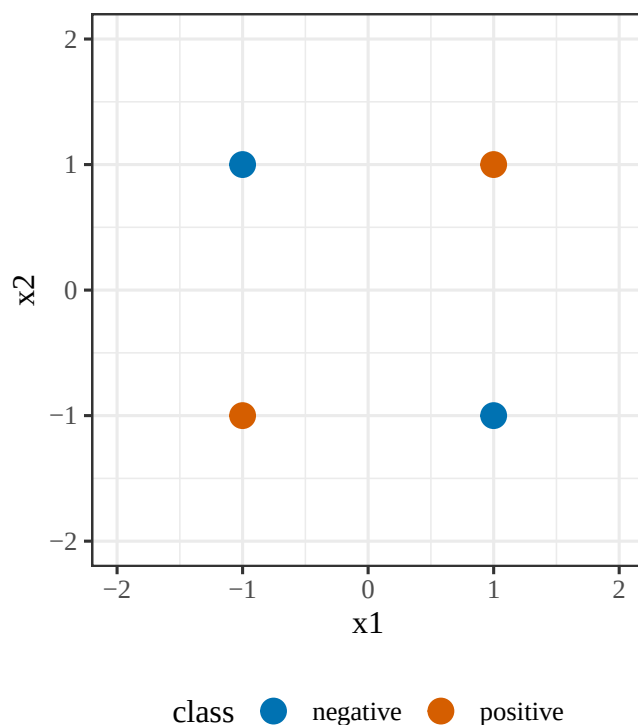


Figure 2: The four points of the XOR problem. Each class occupies two opposite corners of the square, so no line has one class on each side.

**Exercise 1.6** (Say which training sets a hyperplane can split (part 3)). Use the setup of Exercise 1.4. In one sentence, say what distinguishes the first from the second.

Solution

*Solution 1.6.* In the first set, some linear boundary puts each class entirely on its own side; in the second, no linear boundary does.

**Definition 1.3** (Linearly separable). A two-class dataset is **linearly separable** if there exists a linear decision boundary that separates the classes with no misclassifications [6].

**Example 1.2** (Ten cars that a line separates). Take the extremes of the **Auto** data [5]: the first five cars in the file with **mpg** at most 14 (label  $-1$ ) and the first five with **mpg** at least 30 (label  $+1$ ). Picking the extremes makes this set separable, and it is a toy example in that sense. Use weight in thousands of pounds and horsepower in hundreds as the two inputs, and check whether the perpendicular bisector of the two class means classifies all ten cars correctly:

### 1.3 Python

```

auto_ls = pd.read_csv("data/auto.csv").dropna().reset_index(drop=True)
cars_ls = pd.concat([auto_ls[auto_ls["mpg"] <= 14].head(5), auto_ls[auto_ls["mpg"] >= 30].head(5)])
# Rescaling the units keeps both inputs near 1.
X_sep = np.column_stack([cars_ls["weight"] / 1000, cars_ls["horsepower"] / 100])
y_sep = np.where(cars_ls["mpg"] >= 30, 1.0, -1.0)
print(cars_ls[["name", "mpg", "weight", "horsepower"]].to_string(index=False))

```

|  | name                    | mpg  | weight | horsepower |
|--|-------------------------|------|--------|------------|
|  | chevrolet impala        | 14.0 | 4354   | 220        |
|  | plymouth fury iii       | 14.0 | 4312   | 215        |
|  | pontiac catalina        | 14.0 | 4425   | 225        |
|  | plymouth 'cuda 340      | 14.0 | 3609   | 160        |
|  | buick estate wagon (sw) | 14.0 | 3086   | 225        |
|  | peugeot 304             | 30.0 | 2074   | 70         |
|  | fiat 124b               | 30.0 | 2065   | 76         |
|  | toyota corolla 1200     | 31.0 | 1773   | 65         |
|  | datsum 1200             | 35.0 | 1613   | 69         |
|  | datsum b210             | 31.0 | 1950   | 67         |

```

# y * h(x) > 0 means the car is on the side of the hyperplane that matches its label.
mean_hi_ls, mean_lo_ls = X_sep[y_sep > 0].mean(axis=0), X_sep[y_sep < 0].mean(axis=0)
w_ls = mean_hi_ls - mean_lo_ls
b_ls = -w_ls @ (mean_hi_ls + mean_lo_ls) / 2
print((y_sep * (X_sep @ w_ls + b_ls)).round(2))

```

```
[4.07 3.92 4.29 1.7  1.53 2.72 2.66 3.41 3.69 3.02]
```

## 1.4 R

```

auto_ls <- tidyr::drop_na(read.csv("data/auto.csv"))
cars_ls <- rbind(head(auto_ls[auto_ls$mpg <= 14, ], 5), head(auto_ls[auto_ls$mpg >= 30, ], 5))
# Rescaling the units keeps both inputs near 1.
X_sep <- cbind(cars_ls$weight / 1000, cars_ls$horsepower / 100)
y_sep <- ifelse(cars_ls$mpg >= 30, 1, -1)
cars_ls[, c("name", "mpg", "weight", "horsepower")]

```

```
# A tibble: 10 x 4
```

|    | name                    | mpg   | weight | horsepower |
|----|-------------------------|-------|--------|------------|
|    | <chr>                   | <dbl> | <int>  | <int>      |
| 1  | chevrolet impala        | 14    | 4354   | 220        |
| 2  | plymouth fury iii       | 14    | 4312   | 215        |
| 3  | pontiac catalina        | 14    | 4425   | 225        |
| 4  | plymouth 'cuda 340      | 14    | 3609   | 160        |
| 5  | buick estate wagon (sw) | 14    | 3086   | 225        |
| 6  | peugeot 304             | 30    | 2074   | 70         |
| 7  | fiat 124b               | 30    | 2065   | 76         |
| 8  | toyota corolla 1200     | 31    | 1773   | 65         |
| 9  | datsum 1200             | 35    | 1613   | 69         |
| 10 | datsum b210             | 31    | 1950   | 67         |

```

# y * h(x) > 0 means the car is on the side of the hyperplane that matches its label.
mean_hi_ls <- colMeans(X_sep[y_sep > 0, ])
mean_lo_ls <- colMeans(X_sep[y_sep < 0, ])
w_ls <- mean_hi_ls - mean_lo_ls
b_ls <- -sum(w_ls * (mean_hi_ls + mean_lo_ls)) / 2
round(y_sep * (drop(X_sep %*% w_ls) + b_ls), 2)

```

```
[1] 4.07 3.92 4.29 1.70 1.53 2.72 2.66 3.41 3.69 3.02
```

All ten products  $y_{(i)}h(x_{(i)})$  are positive, the smallest being 1.53, so this hyperplane classifies every car correctly and the set is linearly separable in the sense of Definition 1.3. Existence is all the definition asks for. Many other hyperplanes separate these ten cars too, which is why the next examples ask which one is best.

### i “SVM” names three models

“Support vector machine” is used loosely, for three models rather than one [5, Ch. 9]:

- A *maximal margin classifier* applies only to linearly separable data. This chapter calls it the hard-margin SVM.
- A *support vector classifier* also handles data that is not separable; its decision boundary is still linear. This chapter calls it the soft-margin SVM.
- A *support vector machine*, in the strict sense, has a non-linear decision boundary. The kernel trick turns either of the other two into one.

In practice all three get called support vector machines, so the name on its own does not tell you which of them is meant. This chapter uses “SVM” for all three, and says which one when it matters.

One thing does change before we start, and it is only a change of notation. From here on the binary labels are

$$y_{(i)} \in \{-1, +1\},$$

rather than the  $\{0, 1\}$  that logistic regression used.

Every constraint in this lecture is written in terms of the single product

$$y_{(i)}h(x_{(i)}),$$

which is positive on a correct prediction and negative on a wrong one. The notation choices page<sup>1</sup> gives the reason for the switch and explains how to read a source that uses the other coding.

**Exercise 1.7** (Say what makes one separating hyperplane better than another). Sketch two linearly separable clouds of points in  $\mathbb{R}^2$ , one class each, and draw three different hyperplanes that separate them. One of the three should be the one logistic regression would have returned.

Solution

*Solution 1.7.* Any three lines that leave one class entirely on each side will do.

<sup>1</sup><https://morrison-lab.github.io/macros/notation-choices.html#sec-binary-labels>

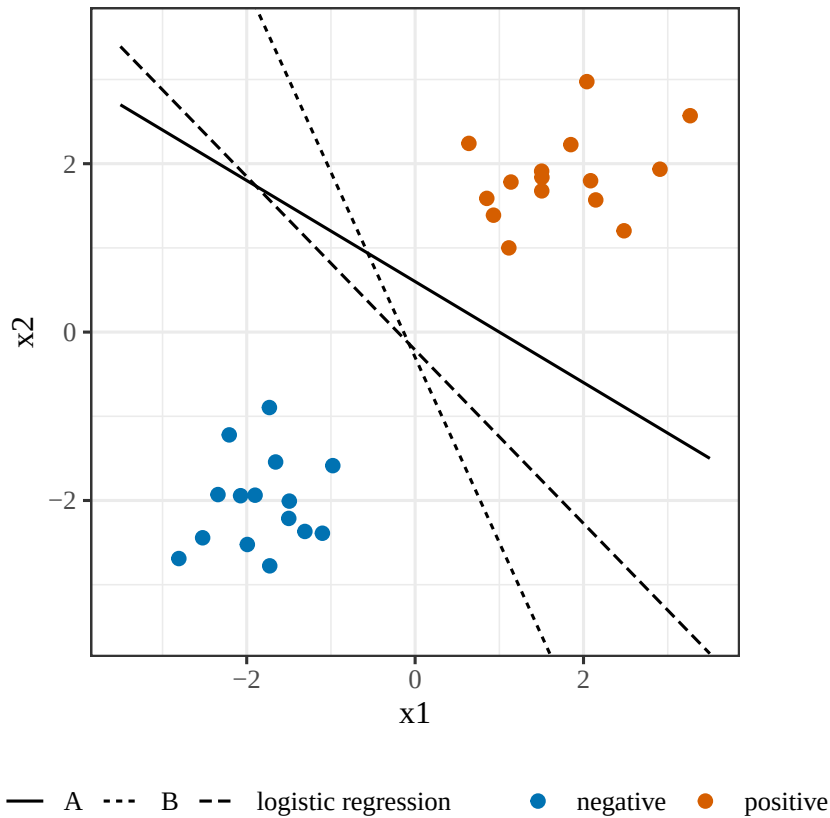


Figure 3: Two linearly separable clouds of points in  $\mathbb{R}^2$ . Each of the three lines leaves every positive point on one side and every negative point on the other; one of them is the line that logistic regression returns.

**Exercise 1.8** (Say what makes one separating hyperplane better than another (part 2)). Use the setup of Exercise 1.7.  
How many separating hyperplanes does a linearly separable training set admit?

**Solution**

*Solution 1.8.* Infinitely many. Separability says at least one exists, and any sufficiently small tilt or shift of a separating hyperplane still separates.

**Exercise 1.9** (Say what makes one separating hyperplane better than another (part 3)). Use the setup of Exercise 1.7.  
Every one of them classifies the training set perfectly, so training accuracy cannot choose between them. Name a property that can, and say why you would want it.

**Solution**

*Solution 1.9.* How much room it leaves. A hyperplane that passes very close to a training point classifies that point correctly and would misclassify a new point drawn a tiny distance to its other side. One that sits well clear of both classes tolerates that small variation. Training accuracy cannot tell them apart because it only asks which side of the boundary each point is on, never how far.

**Exercise 1.10** (Say what makes one separating hyperplane better than another (part 4)). Use the setup of Exercise 1.7.

Using that property, state in one sentence which hyperplane you want.

Solution

*Solution 1.10.* Take the hyperplane that is as far as possible from the nearest point of each class.

**Exercise 1.11** (Say what makes one separating hyperplane better than another (part 5)). Use the setup of Exercise 1.7.

Move one of the points far away from the boundary. Does your answer to Exercise 1.9 change?

Solution

*Solution 1.11.* It does not change. Solution 1.9 maximizes the distance to the nearest point, and a point that was not nearest to begin with does not become nearest by moving further away.

**Exercise 1.12** (Say what makes one separating hyperplane better than another (part 6)). Use the setup of Exercise 1.7.

Which points, then, actually determine where the hyperplane lies?

Solution

*Solution 1.12.* Only the closest points fix where the hyperplane lies, and the rest of the training set could be deleted without moving the hyperplane at all.

That is a strong claim, and it is the property the method is named for. Definition 1.4, Definition 1.5 and Definition 1.6 make it precise.

**Definition 1.4** (Margin). The **margin** of a separating hyperplane is the distance from it to the nearest training point.

**Definition 1.5** (Support vector). A **support vector** is a training point lying at distance exactly  $M$  from the hyperplane, where  $M$  is the margin of Definition 1.4.

**Definition 1.6** (Max-margin hyperplane). The **max-margin hyperplane** is the separating hyperplane whose margin is largest [7].

**Example 1.3** (The margin, support vectors and max-margin hyperplane of ten cars). For the ten cars of Example 1.2, find the max-margin hyperplane, its margin, and the support vectors. The helper `dual_svm` below solves the dual problem of the next section by repeatedly improving the two multipliers that violate its optimality conditions the most; treat it as a black box for now, since only its answer matters here. It returns the multipliers `alpha` and the offset `b`, and the weight vector is  $w = \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)}$ .

## 1.5 Python

```

def dual_svm(K, y, C=np.inf, tol=1e-6, max_iter=100_000):
    """Maximize the dual over alpha, given the kernel matrix K and labels y in {-1, +1}."""
    alpha, grad = np.zeros(len(y)), -np.ones(len(y))    # grad = Q alpha - 1, with Q = y y' * K
    Q = np.outer(y, y) * K
    for _ in range(max_iter):
        score = -y * grad
        up = ((y > 0) & (alpha < C)) | ((y < 0) & (alpha > 0))
        low = ((y > 0) & (alpha > 0)) | ((y < 0) & (alpha < C))
        i, j = np.flatnonzero(up)[np.argmax(score[up])], np.flatnonzero(low)[np.argmin(score[low])]
        if score[i] - score[j] < tol:
            break
        # Move alpha_i and alpha_j together, so that sum(alpha * y) stays 0 and the box 0 <= alpha <= C
        step = (score[i] - score[j]) / (K[i, i] + K[j, j] - 2 * K[i, j])
        step = min(step, C - alpha[i] if y[i] > 0 else alpha[i], alpha[j] if y[j] > 0 else C - alpha[j])
        alpha[i] += y[i] * step
        alpha[j] -= y[j] * step
        grad += (Q[:, i] * y[i] - Q[:, j] * y[j]) * step
    free = (alpha > 1e-7) & (alpha < C - 1e-7)
    return alpha, np.mean((y - (alpha * y) @ K)[free])    # b from the points strictly inside the box

K_mm = X_sep @ X_sep.T    # inner products of the ten cars
alpha_mm, b_mm = dual_svm(K_mm, y_sep)
w_mm = (alpha_mm * y_sep) @ X_sep

# The distance from a car to the hyperplane is |w.x + b| / ||w||.
dist_mm = np.abs(X_sep @ w_mm + b_mm) / np.linalg.norm(w_mm)
margin_mm = dist_mm.min()    # the margin is the smallest of these
is_sv_mm = dist_mm < margin_mm + 1e-4    # support vectors sit at exactly that distance
print(pd.DataFrame({"name": cars_ls["name"].to_numpy(), "distance": dist_mm.round(3), "support_vector": is_sv_mm}))

```

|   | name                    | distance | support_vector |
|---|-------------------------|----------|----------------|
| 0 | chevrolet impala        | 1.821    | False          |
| 1 | plymouth fury iii       | 1.757    | False          |
| 2 | pontiac catalina        | 1.908    | False          |
| 3 | plymouth 'cuda 340      | 0.865    | True           |
| 4 | buick estate wagon (sw) | 0.865    | True           |
| 5 | peugeot 304             | 0.895    | False          |
| 6 | fiat 124b               | 0.865    | True           |
| 7 | toyota corolla 1200     | 1.161    | False          |
| 8 | datsum 1200             | 1.261    | False          |
| 9 | datsum b210             | 1.011    | False          |

```

# The bisector of the class means (previous example) also separates the cars, with a smaller margin.
margin_ls = (y_sep * (X_sep @ w_ls + b_ls)).min() / np.linalg.norm(w_ls)
print(round(margin_mm, 4), round(margin_ls, 4))

0.8648 0.6134

```

## 1.6 R

```
dual_svm <- function(K, y, C = Inf, tol = 1e-6, max_iter = 1e5) {
  # Maximize the dual over alpha, given the kernel matrix K and labels y in {-1, +1}.
  alpha <- numeric(length(y))
  grad <- rep(-1, length(y)) # grad = Q alpha - 1, with Q = y y' * K
  Q <- outer(y, y) * K
  for (iter in seq_len(max_iter)) {
    score <- -y * grad
    up <- which((y > 0 & alpha < C) | (y < 0 & alpha > 0))
    low <- which((y > 0 & alpha > 0) | (y < 0 & alpha < C))
    i <- up[which.max(score[up])]
    j <- low[which.min(score[low])]
    if (score[i] - score[j] < tol) break
    # Move alpha_i and alpha_j together, so that sum(alpha * y) stays 0 and the box 0 <= alpha <= C holds
    step <- (score[i] - score[j]) / (K[i, i] + K[j, j] - 2 * K[i, j])
    step <- min(step, if (y[i] > 0) C - alpha[i] else alpha[i], if (y[j] > 0) alpha[j] else C - alpha[j])
    alpha[i] <- alpha[i] + y[i] * step
    alpha[j] <- alpha[j] - y[j] * step
    grad <- grad + (Q[, i] * y[i] - Q[, j] * y[j]) * step
  }
  free <- alpha > 1e-7 & alpha < C - 1e-7
  list(alpha = alpha, b = mean((y - drop((alpha * y) %*% K))[free])) # b from the points strictly in
}
```

```
K_mm <- X_sep %*% t(X_sep) # inner products of the ten cars
fit_mm <- dual_svm(K_mm, y_sep)
alpha_mm <- fit_mm$alpha
b_mm <- fit_mm$b
w_mm <- drop((alpha_mm * y_sep) %*% X_sep)

# The distance from a car to the hyperplane is |w.x + b| / ||w||.
dist_mm <- abs(drop(X_sep %*% w_mm) + b_mm) / sqrt(sum(w_mm^2))
margin_mm <- min(dist_mm) # the margin is the smallest of these
is_sv_mm <- dist_mm < margin_mm + 1e-4 # support vectors sit at exactly that distance
data.frame(name = cars_ls$name, distance = round(dist_mm, 3), support_vector = is_sv_mm)
```

```
# A tibble: 10 x 3
  name                distance support_vector
<chr>                <dbl> <lgl>
1 chevrolet impala    1.82 FALSE
2 plymouth fury iii   1.76 FALSE
3 pontiac catalina    1.91 FALSE
4 plymouth 'cuda 340  0.865 TRUE
5 buick estate wagon (sw) 0.865 TRUE
6 peugeot 304         0.895 FALSE
7 fiat 124b           0.865 TRUE
8 toyota corolla 1200  1.16 FALSE
9 datsun 1200         1.26 FALSE
10 datsun b210        1.01 FALSE
```

```
# The bisector of the class means (previous example) also separates the cars, with a smaller margin.
margin_ls <- min(y_sep * (drop(X_sep %*% w_ls) + b_ls)) / sqrt(sum(w_ls^2))
c(max_margin = margin_mm, bisector = margin_ls)
```

```
max_margin  bisector
0.8647659   0.6133909
```

The margin of Definition 1.4 of the max-margin hyperplane is 0.865, in the units of the scaled inputs. The 3 support vectors of Definition 1.5, which are the plymouth 'cuda 340, buick estate wagon (sw), fiat

124b, lie at exactly that distance, and every other car is farther away. By Definition 1.6 no separating hyperplane has a larger margin; the bisector of the class means has 0.613.

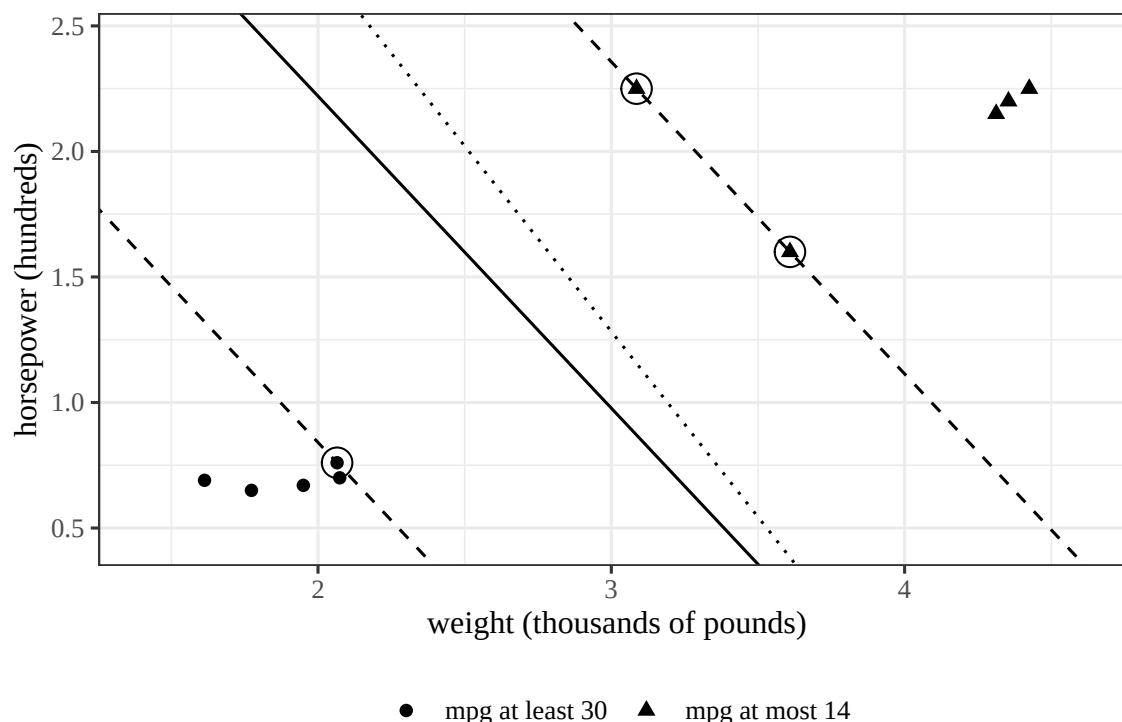


Figure 4: The ten cars of Example 1.2, by weight and horsepower. The solid line is the max-margin hyperplane and the dashed lines mark its margin. The circled cars are the support vectors, the only cars on the margin. The dotted line is the bisector of the class means, which also separates the cars but leaves a smaller margin.

### i Read

GeeksforGeeks explains support vector machines<sup>a</sup> [8].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/>

Knowing what we want is not the same as being able to compute it, so the rest of this lecture turns Definition 1.6 into an optimization problem.

Two warnings about the form of that problem. It will *not* be “minimize average loss over the training set”, which is the only form these notes have used so far; it will be a maximization subject to one constraint per training point. And actually *solving* such a problem is beyond the scope of this class. We will state it, understand what it asks for, and give it to a solver.

**Exercise 1.13** (Write the margin in terms of the parameters, in one dimension). Draw a real line with a few positive points to the right and a few negative points to the left. We want a linear function  $h(x) = wx + b$ , with  $w, b \in \mathbb{R}$ , that separates them. There is an easier way to do this in one dimension; we are doing it the way that generalizes to  $p$ .

Let  $x$  be the training point nearest the hyperplane, let  $y$  be its label, and let  $M$  be the margin of Definition 1.4.

What shape is a separating hyperplane in one dimension?

### Solution

*Solution 1.13.* A point. A hyperplane is one dimension short of the space it sits in, so in  $\mathbb{R}^1$  it is a single location on the line.

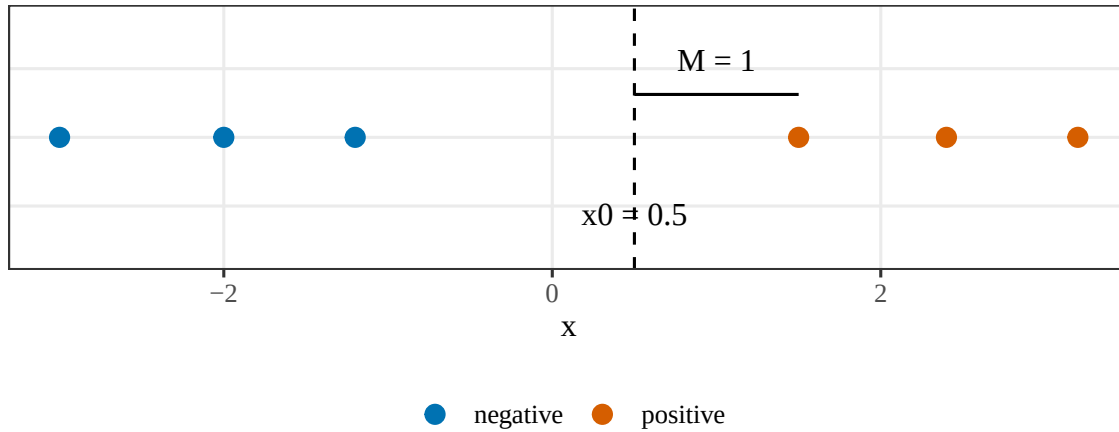


Figure 5: A separating hyperplane in one dimension is the single point  $x_0 = -b/w$ . The margin  $M$  is the distance from  $x_0$  to the nearest training point.

**Exercise 1.14** (Write the margin in terms of the parameters, in one dimension (part 2)). Use the setup of Exercise 1.13.

Where on the line does  $h$  vanish?

### Solution

*Solution 1.14.* At  $x_0 = -b/w$ , the root of  $w x + b$ .

**Exercise 1.15** (Write the margin in terms of the parameters, in one dimension (part 3)). Use the setup of Exercise 1.13.

Take  $y = 1$  and  $w > 0$ . Reading  $w$  as rise over run between that point and the hyperplane, write  $w$  in terms of  $h(x)$  and  $M$ .

### Solution

*Solution 1.15.* The rise from the hyperplane to the point is  $h(x) - h(x_0) = h(x)$ , since  $h(x_0) = 0$ . With  $y = 1$  and  $w > 0$  the positive points sit to the right of  $x_0$ , so the run is  $x - x_0 = M$ , and

$$w = \frac{h(x)}{M}.$$

**Exercise 1.16** (Write the margin in terms of the parameters, in one dimension (part 4)). Use the setup of Exercise 1.13.

Now take  $y = -1$ , still with  $w > 0$ , and do the same.

### Solution

*Solution 1.16.* With  $y = -1$  and  $w > 0$  the point sits to the left, so the run is  $x - x_0 = -M$  and the same ratio gives

$$w = \frac{-h(x)}{M}.$$

**Exercise 1.17** (Write the margin in terms of the parameters, in one dimension (part 5)). Use the setup of Exercise 1.13.

Combine Exercise 1.15 and Exercise 1.16 into a single equation covering both labels, and rearrange it to solve for  $M$ .

### Solution

*Solution 1.17.* In both cases the numerator is  $y h(x)$ , because multiplying by  $y$  flips the sign exactly when the point is on the negative side. So the two combine into

$$\frac{y h(x)}{M} = w \quad \text{that is} \quad y h(x) = wM \quad \text{that is} \quad \frac{y h(x)}{w} = M.$$

**Exercise 1.18** (Write the margin in terms of the parameters, in one dimension (part 6)). Use the setup of Exercise 1.13.

Check your equation when  $w < 0$ . Does it still hold?

### Solution

*Solution 1.18.* It fails. If  $w < 0$  then  $h$  is decreasing, so the positive points are the ones to the *left* of  $x_0$  and both runs in Solution 1.15 and Solution 1.16 flip sign. Redoing Solution 1.15 with  $y = 1$  and  $w < 0$  gives run  $x - x_0 = -M$  and so  $w = -h(x)/M$ , whence  $y h(x) = -wM$ , which is a positive number only because  $w$  is negative.

**Exercise 1.19** (Write the margin in terms of the parameters, in one dimension (part 7)). Use the setup of Exercise 1.13.

If it fails, repair it.

### Solution

*Solution 1.19.* The repair is to take the size of  $w$  rather than its signed value:

$$\frac{y h(x)}{M} = |w| \quad \text{that is} \quad y h(x) = |w| M \quad \text{that is} \quad \frac{y h(x)}{|w|} = M.$$

Both sides are now non-negative for a correctly classified point, whichever way  $h$  slopes<sup>a</sup>. Note that third form: the distance from a point to the hyperplane is the value of the linear function *divided by the size of the slope*<sup>b</sup>. That ratio carries over to  $p$  dimensions.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

<sup>b</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

**Theorem 1.1** (The margin of a point, in one dimension). *Let  $h(x) = wx + b$  with  $w \neq 0$ , and let  $x$  be a training point with label  $y \in \{-1, +1\}$  that lies on its own side of the hyperplane  $h(x) = 0$ , at distance  $M$  from it. Then*

$$\frac{y h(x)}{M} = |w|, \quad \text{that is} \quad y h(x) = |w| M, \quad \text{that is} \quad \frac{y h(x)}{|w|} = M.$$

### i Proof

*Proof.* This is the solution to Exercise 1.13. □

**Exercise 1.20** (Turn “largest margin” into a constrained optimization problem). Theorem 1.1 measured the margin of one point. Training has to choose  $w$  and  $b$  so that the *smallest* such measurement over the whole training set is as large as possible.

Write “every training point is at least  $M$  from the hyperplane, on its own side” as one inequality, indexed by  $i = 1, 2, \dots, N$ .

### Solution

*Solution 1.20.* Apply the third form of Theorem 1.1 to every point at once, with  $\geq$  in place of  $=$  because only the nearest point attains  $M$ :

$$\frac{y_{(i)}(wx_{(i)} + b)}{|w|} \geq M, \quad i = 1, 2, \dots, N.$$

**Exercise 1.21** (Turn “largest margin” into a constrained optimization problem (part 2)). Use the setup of Exercise 1.20.

Write the full problem: what is maximized, over which variables, subject to what.

### Solution

*Solution 1.21.* Maximize the margin over the parameters and the margin itself:

$$\begin{aligned} \arg \max_{w, b, M} \quad & M \\ \text{s.t.} \quad & \frac{y_{(i)}(wx_{(i)} + b)}{|w|} \geq M, \quad i = 1, 2, \dots, N. \end{aligned}$$

Note the form. There is no sum over the training set and no average; each training point contributes a *constraint* rather than a term.

**Exercise 1.22** (Turn “largest margin” into a constrained optimization problem (part 3)). Use the setup of Exercise 1.20.

The problem you wrote is convex<sup>a</sup>, so it has a unique optimal value  $M^*$ . It does not have a unique optimal  $(w^*, b^*)$ . Explain why, and describe the set of optima.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

### Solution

*Solution 1.22.* Because the constraints only ever see  $w$  and  $b$  through the ratio  $(wx_{(i)} + b)/|w|$ , and that ratio is unchanged when  $w$  and  $b$  are both multiplied by the same positive constant<sup>a</sup>  $c$ :

$$\begin{aligned}\frac{y_{(i)}(cwx_{(i)} + cb)}{|cw|} &= \frac{c y_{(i)}(wx_{(i)} + b)}{c |w|} \\ &= \frac{y_{(i)}(wx_{(i)} + b)}{|w|}.\end{aligned}$$

So if  $(w^*, b^*)$  is optimal then so is  $(cw^*, cb^*)$  for every  $c > 0$ , and the optima form a ray. Geometrically all of these describe the same hyperplane: they cross the axis at the same place and slope<sup>b</sup> the same way, differing only in how steeply  $h$  climbs away from it.

The problem is asking about the hyperplane, and  $h$ 's steepness is a detail of how we wrote the hyperplane down.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

<sup>b</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

**Proposition 1.1** (The max-margin program has a ray of optima). *Consider the program, over  $w \neq 0$ ,  $b$  and  $M$ , for a linearly separable training set  $(x_{(i)}, y_{(i)})$ ,  $i = 1, 2, \dots, N$ ,*

$$\begin{aligned}\arg \max_{w, b, M} \quad & M \\ \text{s.t.} \quad & \frac{y_{(i)}(wx_{(i)} + b)}{|w|} \geq M, \quad i = 1, 2, \dots, N.\end{aligned}$$

*If  $(w^*, b^*, M^*)$  is optimal, with  $M^* > 0$ , then so is  $(cw^*, cb^*, M^*)$  for every  $c > 0$ . The optima therefore form a ray, and all of its members describe the same hyperplane.*

### Proof

*Proof.* This is the solution to Exercise 1.20. □

**Exercise 1.23** (Pick one representative from the ray of optima). An infinite family of solutions is a nuisance for a solver, and by Proposition 1.1 every member of it describes the same hyperplane. So we are free to add one equation that selects a single member, provided it does not change which hyperplane comes out.

We choose this one: at the optimum, the support vectors satisfy

$$y_{(i)} h(x_{(i)}) = 1.$$

Explain why adding this equation loses no hyperplanes.

### Solution

*Solution 1.23.* Take any optimal  $(w^*, b^*)$ , with margin  $M^*$ . Its support vectors satisfy  $y_{(i)} h(x_{(i)}) = |w^*| M^*$  by Theorem 1.1, and that quantity is positive, so we may set  $c = 1/(|w^*| M^*) > 0$ . Rescaling by  $c$  makes the support vectors' value exactly 1.

So every hyperplane in the feasible set still has a representative, and now it has exactly one. The equation picks a member of the ray; it does not rule any ray out.

**Exercise 1.24** (Pick one representative from the ray of optima (part 2)). Use the setup of Exercise 1.23. Under it, what is  $M$  in terms of  $|w|$ ?

### Solution

*Solution 1.24.* At a support vector  $y_{(i)}h(x_{(i)}) = |w|M$ , and we have just declared that value to be 1, so

$$\frac{1}{M} = |w| \quad \text{that is} \quad 1 = |w|M \quad \text{that is} \quad \frac{1}{|w|} = M.$$

**Exercise 1.25** (Pick one representative from the ray of optima (part 3)). Use the setup of Exercise 1.23. Substitute that into the problem of Proposition 1.1. What is now being maximized, and over which variables?

### Solution

*Solution 1.25.*  $M$  is no longer a free variable — it is now determined by  $w$  — so it leaves the problem and its value takes the objective's place:

$$\begin{aligned} \arg \max_{w, b} \quad & \frac{1}{|w|} \\ \text{s.t.} \quad & \frac{y_{(i)}(wx_{(i)} + b)}{|w|} \geq \frac{1}{|w|}, \quad i = 1, 2, \dots, N. \end{aligned}$$

**Exercise 1.26** (Pick one representative from the ray of optima (part 4)). Use the setup of Exercise 1.23. Simplify the objective and the constraints into the simplest form you can.

### Solution

*Solution 1.26.* Maximizing  $1/|w|$  is minimizing  $|w|$ , and every constraint may be multiplied through by  $|w| > 0$ :

$$\begin{aligned} \arg \min_{w, b} \quad & |w| \\ \text{s.t.} \quad & y_{(i)}(wx_{(i)} + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

This form has a picture. Put a spike of height 1 at every positive point and a spike of height  $-1$  at every negative point. The constraints say the graph of  $h$  must reach or pass the tip of every spike, and the objective says to do it with the shallowest slope<sup>a</sup> available. A shallow line takes a long horizontal distance to climb from  $-1$  to 1, and that distance is the empty band around the boundary.

A line that breaks a constraint crosses a spike, dipping below the tip of an upward spike or rising above the tip of a downward one. That point is then either on the wrong side of the boundary or inside the margin, and the hard margin allows neither.

The picture also shows what separability is doing. If a positive point sits between two negative points, no line reaches all three spikes at once: the constraint set is empty and the problem has no solution. Separability is the condition that keeps it non-empty.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

**Theorem 1.2** (Fixing the scale of the max-margin program). *Let  $(w^*, b^*, M^*)$  be optimal for the program of Proposition 1.1, with  $w^* \neq 0$  and  $M^* > 0$ . Then  $(cw^*, cb^*)$ , with  $c = 1/(|w^*|M^*)$ , describes the same hyperplane and has  $y_{(i)}(cw^*x_{(i)} + cb^*) = 1$  at every support vector  $x_{(i)}$ . For any  $w \neq 0$  and  $b$  with  $y_{(i)}(wx_{(i)} + b) = 1$  at every support vector, the margin is  $M = 1/|w|$ , and the program becomes*

$$\begin{aligned} \arg \min_{w, b} \quad & |w| \\ \text{s.t.} \quad & y_{(i)}(wx_{(i)} + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

### i Proof

*Proof.* This is the solution to Exercise 1.23. □

### i What this chapter's symbols are called elsewhere

Almost every symbol in this chapter is spelled differently somewhere else, and two of the substitutions reuse symbols that we use for something else.

| Here                            | Also written                                                                        |
|---------------------------------|-------------------------------------------------------------------------------------|
| $h(x)$ , the linear function    | $y(x)$ in Bishop; $g(y)$ in Duda, Hart and Stork                                    |
| $w$ , the weight vector         | $\beta$ in ESL and ISL; $a$ in Duda, Hart and Stork, with the intercept included    |
| $b$ , the intercept             | $\beta_0$ in ESL and ISL                                                            |
| $M$ , the margin                | $M$ in ESL and ISL; $\gamma$ in Stanford's CS229 notes; $b$ in Duda, Hart and Stork |
| $y_{(i)}$ , the labels          | $t_n$ in Bishop; $z_k$ in Duda, Hart and Stork                                      |
| $\alpha_i$ , the dual variables | $a_n$ in Bishop                                                                     |

The two clashes are worth stating clearly. Duda, Hart and Stork write  $b$  for the **margin**, not the intercept, and their  $y$  is the *feature-mapped input* rather than the label, so two of their symbols mean something this chapter uses a different symbol for. Their margin satisfies  $b = 1/\|a\|$ , which is this chapter's  $M = 1/\|w\|_2$  written entirely in symbols we use for other things. Bishop's  $y(x)$  is likewise the decision function, not the label.

**Functional** and **geometric margin** are the one pair of names worth adopting even though this chapter does not use them. CS229 calls  $y_{(i)}h(x_{(i)})$  the functional margin,  $\hat{\gamma}$ , and calls  $y_{(i)}h(x_{(i)})/\|w\|_2$  the geometric margin,  $\gamma$ . This chapter's  $M$  is the geometric one. The distinction is exactly what Solution 1.20 turns on: the functional margin scales when you scale  $w$  and  $b$ , the geometric margin does not, and the ray of optima exists because only the geometric one is fixed.

Sources then use that freedom in two different ways. This chapter, CS229 and Bishop all set the support vectors' functional margin to 1 — Bishop calls the result the *canonical representation* of the hyperplane. ESL instead requires  $\|\beta\|_2 = 1$ , which makes the functional and geometric margins equal and lets it keep  $M$  in the objective rather than trading it for  $1/\|w\|_2$ . Both pick one point on the same ray.

**Exercise 1.27** (Carry the one-dimensional result up to  $p$  dimensions). Now let  $x, w \in \mathbb{R}^p$  and  $h(x) = w \cdot x + b$ . The equations of Theorem 1.1 no longer make sense:  $|w|$  was the size of a number, and  $w$  is now a vector.

$|w|$  measured the length of the slope<sup>a</sup>. What function gives the length of a vector?

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

### Solution

*Solution 1.27.* The  $\ell_2$  norm,  $\|w\|_2 = \sqrt{w_1^2 + \dots + w_p^2}$ . In one dimension it is exactly  $|w|$ , so this is a generalization rather than a replacement.

**Exercise 1.28** (Carry the one-dimensional result up to  $p$  dimensions (part 2)). Use the setup of Exercise 1.27. Rewrite the three forms of Theorem 1.1 with that substitution.

Solution

*Solution 1.28.*

$$\frac{y h(x)}{M} = \|w\|_2 \quad \text{that is} \quad y h(x) = \|w\|_2 M \quad \text{that is} \quad \frac{y h(x)}{\|w\|_2} = M.$$

**Exercise 1.29** (Carry the one-dimensional result up to  $p$  dimensions (part 3)). Use the setup of Exercise 1.27.

Check Exercise 1.28 geometrically. Draw the contour lines  $h(x) = 0$ ,  $h(x) = 1$ ,  $h(x) = 2$  and so on, and say what value  $h$  takes at a point sitting a distance  $M$  from the hyperplane on the positive side, then on the negative side.

Solution

*Solution 1.29.* The contours of  $h$  are hyperplanes parallel to the boundary, evenly spaced, and  $h$  climbs fastest in the direction of  $w$ . Moving a distance  $M$  from the boundary along that direction raises  $h$  by  $\|w\|_2 M$ , so a point sitting  $M$  away on the positive side has

$$h(x) = \|w\|_2 M,$$

and one sitting  $M$  away on the negative side has  $h(x) = -\|w\|_2 M$ . Multiplying by the label gives  $y h(x) = \|w\|_2 M$  in both cases, which is Solution 1.28.

**Exercise 1.30** (Carry the one-dimensional result up to  $p$  dimensions (part 4)). Use the setup of Exercise 1.27.

Rewrite the problem of Theorem 1.2 in  $p$  dimensions.

Solution

*Solution 1.30.*

$$\begin{aligned} \arg \min_{w, b} \quad & \|w\|_2 \\ \text{s.t.} \quad & y_{(i)}(w \cdot x_{(i)} + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

Nothing about the derivation used  $p = 1$  except the notation for length. We therefore solved the easy case by the method that extends to higher dimensions.

**Theorem 1.3** (The margin of a point, in  $p$  dimensions). *Let  $h(x) = w \cdot x + b$  with  $w \in \mathbb{R}^p$  and  $w \neq \vec{0}$ , and let  $x$  be a training point with label  $y \in \{-1, +1\}$  that lies on its own side of the hyperplane  $h(x) = 0$ , at distance  $M$  from it. Then*

$$\frac{y h(x)}{M} = \|w\|_2, \quad \text{that is} \quad y h(x) = \|w\|_2 M, \quad \text{that is} \quad \frac{y h(x)}{\|w\|_2} = M.$$

**i** Proof

*Proof.* This is the solution to Exercise 1.27. □

**Exercise 1.31** (Put the objective into the form a solver wants). The  $p$ -dimensional version of the program in Theorem 1.2, which Exercise 1.30 asks for, minimizes  $\|w\|_2$ . The form everyone actually writes down minimizes  $\frac{1}{2}\|w\|_2^2$  instead. Show that the two objectives have the same minimizer.

Solution

*Solution 1.31.*  $t \mapsto \frac{1}{2}t^2$  is strictly increasing on  $t \geq 0$ , and  $\|w\|_2 \geq 0$  always. A strictly increasing transformation of the objective reorders nothing, so the feasible point that minimizes one minimizes the other. The optimal *value* changes; the optimal  $(w, b)$  does not.

**Exercise 1.32** (Put the objective into the form a solver wants (part 2)). Use the setup of Exercise 1.31. Give two reasons to prefer the squared form.

Solution

*Solution 1.32.* The square root is gone, and with it the point of non-differentiability at  $w = 0$ .  $\|w\|_2$  has no gradient there, whereas  $\frac{1}{2}\|w\|_2^2$  is a polynomial<sup>a</sup> in the entries of  $w$  and is differentiable everywhere. And the squared form is *quadratic*, which together with the linear constraints puts the problem in a class solvers are built for.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial>

**Exercise 1.33** (Put the objective into the form a solver wants (part 3)). Use the setup of Exercise 1.31. What is the  $\frac{1}{2}$  for?

Solution

*Solution 1.33.* Convenience of the gradient:  $\nabla_w \frac{1}{2}\|w\|_2^2 = w$  rather than  $2w$ . It changes nothing about the solution.

**Exercise 1.34** (Put the objective into the form a solver wants (part 4)). Use the setup of Exercise 1.31. Write the resulting problem out in full.

Solution

*Solution 1.34.* The hard-margin primal problem:

$$\begin{aligned} \arg \min_{w, b} \quad & \frac{1}{2}\|w\|_2^2 \\ \text{s.t.} \quad & y_{(i)}(w^\top x_{(i)} + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

**Exercise 1.35** (Put the objective into the form a solver wants (part 5)). Use the setup of Exercise 1.31. Classify it: what is the objective as a function of  $(w, b)$ , and what are the constraints?

Solution

*Solution 1.35.* The objective is a convex<sup>a</sup> quadratic in  $w$  and does not involve  $b$  at all; the  $N$  constraints are linear in  $(w, b)$ . That makes it a convex quadratic program, which has a unique solution and standard solvers.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

**Definition 1.7** (Hard-margin primal problem). The **hard-margin primal problem** for a linearly separable training set is

$$\begin{aligned} \arg \min_{w, b} \quad & \frac{1}{2} \|w\|_2^2 \\ \text{s.t.} \quad & y_{(i)}(w \cdot x_{(i)} + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

Its solution is the max-margin hyperplane of Definition 1.6, scaled so that the support vectors of Definition 1.5 satisfy  $y_{(i)}(w \cdot x_{(i)} + b) = 1$ .

**Example 1.4** (Checking the primal problem on the ten cars). For the ten cars of Example 1.2, compare two feasible points of Definition 1.7: the max-margin  $(w, b)$  of Example 1.3, and the bisector of the class means, rescaled by the smallest factor that makes every constraint  $y_{(i)}(w \cdot x_{(i)} + b) \geq 1$  hold. For each, find the tightest constraint and the objective  $\frac{1}{2} \|w\|_2^2$ :

## 1.7 Python

```
# Constraint values y (w.x + b); the primal requires every one to be at least 1.
cons_mm = y_sep * (X_sep @ w_mm + b_mm)
# Dividing the bisector by its smallest constraint value makes that constraint exactly 1.
rescale_hp = 1 / (y_sep * (X_sep @ w_ls + b_ls)).min()
w_hp, b_hp = rescale_hp * w_ls, rescale_hp * b_ls
cons_hp = y_sep * (X_sep @ w_hp + b_hp)

print(pd.DataFrame({
    "candidate": ["max-margin", "rescaled bisector"],
    "tightest_constraint": [cons_mm.min(), cons_hp.min()],
    "objective": [0.5 * w_mm @ w_mm, 0.5 * w_hp @ w_hp],
}).round(4))
```

|   | candidate         | tightest_constraint | objective |
|---|-------------------|---------------------|-----------|
| 0 | max-margin        | 1.0                 | 0.6686    |
| 1 | rescaled bisector | 1.0                 | 1.3289    |

```
print((np.abs(cons_mm - 1) < 1e-4).sum()) # constraints that hold with equality
3
```

## 1.8 R

```
# Constraint values y (w.x + b); the primal requires every one to be at least 1.
cons_mm <- y_sep * (drop(X_sep %*% w_mm) + b_mm)
# Dividing the bisector by its smallest constraint value makes that constraint exactly 1.
rescale_hp <- 1 / min(y_sep * (drop(X_sep %*% w_ls) + b_ls))
w_hp <- rescale_hp * w_ls
b_hp <- rescale_hp * b_ls
cons_hp <- y_sep * (drop(X_sep %*% w_hp) + b_hp)

data.frame(
  candidate = c("max-margin", "rescaled bisector"),
  tightest_constraint = c(min(cons_mm), min(cons_hp)),
  objective = c(0.5 * sum(w_mm^2), 0.5 * sum(w_hp^2))
) |> dplyr::mutate(dplyr::across(where(is.numeric), \(v) round(v, 4)))

# A tibble: 2 x 3
  candidate      tightest_constraint objective
  <chr>          <dbl>          <dbl>
1 max-margin      1          0.669
2 rescaled bisector 1          1.33

sum(abs(cons_mm - 1) < 1e-4)          # constraints that hold with equality
[1] 3
```

Both candidates satisfy every constraint, and each has a tightest constraint equal to 1, so both are feasible. The objective separates them: 0.669 for the max-margin hyperplane against 1.329 for the rescaled bisector. Minimizing  $\frac{1}{2}\|w\|_2^2$  is the same as maximizing the margin, since the margin is  $1/\|w\|_2$  when the tightest constraint is 1: 0.865 here, which is the margin found in Example 1.3. The 3 constraints that hold with equality belong to the support vectors.

**Exercise 1.36** (Compare the margins of three training sets). Three training sets in  $\mathbb{R}^2$ :

1. A positive point at (1, 1) and a negative point at (−1, −1).
2. A positive point at (0, 5) and a negative point at (0, −5).
3. All four of those points at once.

Write  $M_i$  for the largest margin that any separating hyperplane achieves in scenario  $i$ . Which of  $M_1$ ,  $M_2$  and  $M_3$  are equal, and which is the smallest? Then name the support vectors in scenario 3.

Solution

*Solution 1.36.*

$$\begin{aligned}
 M_1 &= M_3 \\
 &= \sqrt{2} \\
 &\approx 1.414 \\
 &< M_2 \\
 &= 5.
 \end{aligned}$$

**Scenarios 1 and 2** have one point per class, so the max-margin hyperplane is the perpendicular bisector of the segment joining them and the margin is half that segment's length. In scenario 1 the points are  $\sqrt{2^2 + 2^2} = 2\sqrt{2}$  apart, giving  $M_1 = \sqrt{2}$  with the hyperplane  $x_1 + x_2 = 0$ . In scenario 2 they are 10 apart, giving  $M_2 = 5$  with the hyperplane  $x_2 = 0$ .

**Scenario 3** adds constraints without removing any. Adding a constraint can never enlarge a feasible set, so  $M_3 \leq \min(M_1, M_2) = \sqrt{2}$ . And the hyperplane  $x_1 + x_2 = 0$  achieves  $\sqrt{2}$  on all four points:  $(\pm 1, \pm 1)$  sit at distance  $\sqrt{2}$  and  $(0, \pm 5)$  at distance  $5/\sqrt{2} \approx 3.54$ . So  $M_3 = \sqrt{2}$  exactly.

**The support vectors in scenario 3** are (1, 1) and (−1, −1). The points at (0, 5) and (0, −5) sit at distance  $5/\sqrt{2} > \sqrt{2}$ , so by Definition 1.5 they are not support vectors, and deleting them would leave the solution untouched — even though on their own they admit a margin more than three times as

large.

Two points follow from this. A training set's margin is set by its *worst* pair, not by its typical spacing, so one point near the boundary is enough to make a widely spaced data set behave like a closely spaced one. And the solution depends on very few of the training points, a property the dual formulation of the same problem makes explicit.

### i Watch

Andrew Ng introduces SVMs in CS229, from 43:27<sup>a</sup>, and defines the functional and geometric margins, from 54:15<sup>b</sup> [9], Lecture 6.

<sup>a</sup><https://www.youtube.com/watch?v=IDwow4aOrtg&t=2607s>

<sup>b</sup><https://www.youtube.com/watch?v=IDwow4aOrtg&t=3255s>

### i Watch

- Hastie and Tibshirani explain the optimal separating hyperplane<sup>a</sup> [10], video 9.1.
- Abu-Mostafa teaches support vector machines<sup>b</sup> [11], Lecture 14.
- Weinberger explains the geometry of the margin, from 17:42<sup>c</sup> [12], Lecture 14.

<sup>a</sup><https://www.youtube.com/watch?v=Op0OyOuDjcQ>

<sup>b</sup><https://www.youtube.com/watch?v=eHsErIPJWUU>

<sup>c</sup><https://www.youtube.com/watch?v=xpHQ6UhMlx4&t=1062s>

## 2 Hard-margin SVMs: the dual problem

Every convex optimization problem has a paired problem, called its **dual**, built from the constraints rather than from the objective. Solving one solves the other. A full treatment belongs in a convex optimization course; we need only to see what the dual of Definition 1.7 looks like and why anyone would prefer it.

Three facts make the dual worth the extra work. It is the form in which the support vectors appear explicitly, so solving it is how you find them. It trades a problem in  $p + 1$  unknowns for one in  $N$ , which is better whenever the features outnumber the examples. And it is the only one of the two that survives the step to non-linear classifiers, where  $p$  can be infinite.

**Exercise 2.1** (Read the dual problem before deriving anything). Here is the dual of Definition 1.7. Deriving it needs Lagrange multipliers and is out of scope; reading it is not.

$$\begin{aligned} \arg \max_{\alpha \in \mathbb{R}^N} \quad & \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} x_{(i)} \cdot x_{(j)} \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \\ & \sum_{i=1}^N \alpha_i y_{(i)} = 0. \end{aligned}$$

How many unknowns does it have, and what does each one attach to?

### Solution

*Solution 2.1.*  $N$  unknowns, one per training example:  $\alpha_i$  belongs to the point  $x_{(i)}$ .

**Exercise 2.2** (Read the dual problem before deriving anything (part 2)). Use the setup of Exercise 2.1. Compare that with the primal's count.

#### Solution

*Solution 2.2.* The primal had  $p + 1$ , one per feature plus the intercept<sup>a</sup>. So the dual's size is set by how many examples you have and the primal's by how many features, and neither is smaller in general.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

**Exercise 2.3** (Read the dual problem before deriving anything (part 3)). Use the setup of Exercise 2.1. Where did  $w$  and  $b$  go?

#### Solution

*Solution 2.3.* Out of the problem entirely. They come back afterwards, computed from the  $\alpha_i$ .

**Exercise 2.4** (Read the dual problem before deriving anything (part 4)). Use the setup of Exercise 2.1. The training inputs<sup>a</sup> appear in exactly one place. Where, and in what form?

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

#### Solution

*Solution 2.4.* Only inside the inner products  $x_{(i)} \cdot x_{(j)}$ . Every one of the  $N^2$  coefficients of the objective is one such number, and the inputs<sup>a</sup> appear nowhere else.

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

**Exercise 2.5** (Read the dual problem before deriving anything (part 5)). Use the setup of Exercise 2.1. What does the problem therefore need to know about the data?

#### Solution

*Solution 2.5.* A solver therefore does not need the points themselves — it needs the matrix of pairwise inner products, and nothing more.

**Exercise 2.6** (Read the dual problem before deriving anything (part 6)). Use the setup of Exercise 2.1. Name one situation in which you would rather solve this than the primal, using only your answers to Exercise 2.1 and Exercise 2.3.

#### Solution

*Solution 2.6.* Whenever  $N < p$ , by Solution 2.2: fewer unknowns to solve for. Or whenever the inner products are cheap to compute but the coordinates are expensive to store, by Solution 2.3 — which is where the [kernel](#) trick applies.

**Definition 2.1** (Hard-margin dual problem). The **hard-margin dual problem** is

$$\begin{aligned} \arg \max_{\alpha \in \mathbb{R}^N} \quad & \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} x_{(i)} \cdot x_{(j)} \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \\ & \sum_{i=1}^N \alpha_i y_{(i)} = 0, \end{aligned}$$

whose solution determines the same hyperplane as Definition 1.7.

**Definition 2.2** (Dual variable). The **dual variable**  $\alpha_i$  is the unknown Definition 2.1 attaches to training example  $x_{(i)}$ , constrained to be non-negative.

**Example 2.1** (Dual variables of the ten cars). The solver of Example 1.3 maximizes Definition 2.1 over the dual variables  $\alpha_1, \dots, \alpha_{10}$ , one per car. Print them, check the constraints of the dual, and compare the dual objective with the primal objective  $\frac{1}{2} \|w\|_2^2$ :

## 2.1 Python

```
ay_dm = alpha_mm * y_sep # alpha_i * y_i, which also builds w
# The dual objective: sum of alphas minus a quadratic form in alpha * y.
dual_obj_dm = alpha_mm.sum() - 0.5 * ay_dm @ K_mm @ ay_dm
primal_obj_dm = 0.5 * w_mm @ w_mm

print(pd.DataFrame({"name": cars_ls["name"].to_numpy(), "alpha": alpha_mm.round(4)}))
```

|   | name                    | alpha  |
|---|-------------------------|--------|
| 0 | chevrolet impala        | 0.0000 |
| 1 | plymouth fury iii       | 0.0000 |
| 2 | pontiac catalina        | 0.0000 |
| 3 | plymouth 'cuda 340      | 0.4174 |
| 4 | buick estate wagon (sw) | 0.2512 |
| 5 | peugeot 304             | 0.0000 |
| 6 | fiat 124b               | 0.6686 |
| 7 | toyota corolla 1200     | 0.0000 |
| 8 | datsum 1200             | 0.0000 |
| 9 | datsum b210             | 0.0000 |

```
print("smallest alpha:", alpha_mm.min())
smallest alpha: 0.0

print("sum of alpha * y:", ay_dm.sum())
sum of alpha * y: 2.7755575615628914e-16

print("dual and primal objectives:", dual_obj_dm, primal_obj_dm)
dual and primal objectives: 0.6686099330013634 0.6686096260601547
```

## 2.2 R

```
ay_dm <- alpha_mm * y_sep # alpha_i * y_i, which also builds w
# The dual objective: sum of alphas minus a quadratic form in alpha * y.
dual_obj_dm <- sum(alpha_mm) - 0.5 * drop(t(ay_dm) %*% K_mm %*% ay_dm)
primal_obj_dm <- 0.5 * sum(w_mm^2)

data.frame(name = cars_ls$name, alpha = round(alpha_mm, 4))
```

```
# A tibble: 10 x 2
  name                alpha
  <chr>              <dbl>
1 chevrolet impala    0
2 plymouth fury iii   0
3 pontiac catalina    0
4 plymouth 'cuda 340  0.417
5 buick estate wagon (sw) 0.251
6 peugeot 304         0
7 fiat 124b           0.669
8 toyota corolla 1200  0
9 datsun 1200         0
10 datsun b210        0
```

```
cat("smallest alpha:", min(alpha_mm), "\n")
```

```
smallest alpha: 0
```

```
cat("sum of alpha * y:", sum(ay_dm), "\n")
```

```
sum of alpha * y: 2.775558e-16
```

```
cat("dual and primal objectives:", dual_obj_dm, primal_obj_dm, "\n")
```

```
dual and primal objectives: 0.6686099 0.6686096
```

Every dual variable is non-negative, and  $\sum_{i=1}^N \alpha_i y_{(i)}$  is  $3\text{e-}16$  in absolute value, which is zero up to rounding, so the solution satisfies the constraints of Definition 2.1. Only 3 of the ten dual variables are positive, and they belong to the support vectors of Example 1.3; the other cars have  $\alpha_i = 0$ . The dual objective is 0.6686 and the primal objective is 0.6686, so the dual problem recovers the same hyperplane at the same optimal value.

**Exercise 2.7** (Show that the optimal  $w$  is a weighted sum of the training inputs). Definition 1.7 searches over every  $w \in \mathbb{R}^p$ . You can show that its solution lies in a much smaller set using nothing more than inner products.

Split any weight vector  $w$  into two pieces,

$$w = w_{\parallel} + w_{\perp},$$

where

- $w_{\parallel} = \sum_{i=1}^N c_i x_{(i)}$  is a weighted sum of the training inputs, with real weights  $c_1, c_2, \dots, c_N$ ;
- $w_{\perp}$  is orthogonal to every training input:  $w_{\perp} \cdot x_{(i)} = 0$  for  $i = 1, 2, \dots, N$ .

Every  $w$  splits this way:  $w_{\parallel}$  is the orthogonal projection of  $w$  onto the set of weighted sums of the training inputs, and  $w_{\perp} = w - w_{\parallel}$  is the remainder.

Show that  $w \cdot x_{(i)} = w_{\parallel} \cdot x_{(i)}$  for every  $i$ .

**Solution**

*Solution 2.7.* Substitute the split and distribute the inner product:

$$\begin{aligned} w \cdot x_{(i)} &= (w_{\parallel} + w_{\perp})^{\top} x_{(i)} && \text{(split } w) \\ &= w_{\parallel} \cdot x_{(i)} + w_{\perp} \cdot x_{(i)} && \text{(inner product distributes over a sum)} \\ &= w_{\parallel} \cdot x_{(i)} + 0 && (w_{\perp} \text{ is orthogonal to } x_{(i)}) \\ &= w_{\parallel} \cdot x_{(i)}. \end{aligned}$$

**Exercise 2.8** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 2)). Use the setup of Exercise 2.7.

Suppose  $(w, b)$  satisfies every constraint of Definition 1.7. Does  $(w_{\parallel}, b)$ ?

Solution

*Solution 2.8.* Yes. The  $i$ th constraint sees  $w$  only through  $w \cdot x_{(i)}$ , so by Solution 2.7

$$y_{(i)}(w \cdot x_{(i)} + b) = y_{(i)}(w_{\parallel} \cdot x_{(i)} + b),$$

and each constraint holds for  $(w_{\parallel}, b)$  exactly when it holds for  $(w, b)$ . The constraints do not depend on  $w_{\perp}$  at all.

**Exercise 2.9** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 3)). Use the setup of Exercise 2.7.

Write  $\|w\|_2^2$  in terms of  $\|w_{\parallel}\|_2^2$  and  $\|w_{\perp}\|_2^2$ .

Solution

*Solution 2.9.* Write the squared norm<sup>a</sup> as an inner product and expand:

$$\begin{aligned} \|w\|_2^2 &= (w_{\parallel} + w_{\perp})^{\top} (w_{\parallel} + w_{\perp}) && \text{(squared norm as an inner product)} \\ &= w_{\parallel} \cdot w_{\parallel} + w_{\parallel} \cdot w_{\perp} + w_{\perp} \cdot w_{\parallel} + w_{\perp} \cdot w_{\perp} && \text{(distribute)} \\ &= \|w_{\parallel}\|_2^2 + 2 w_{\parallel} \cdot w_{\perp} + \|w_{\perp}\|_2^2 && \text{(inner product is symmetric).} \end{aligned}$$

The cross term vanishes, because  $w_{\parallel}$  is built from vectors that  $w_{\perp}$  is orthogonal<sup>b</sup> to:

$$\begin{aligned} w_{\parallel} \cdot w_{\perp} &= \left( \sum_{i=1}^N c_i x_{(i)} \right)^{\top} w_{\perp} && \text{(definition of } w_{\parallel}) \\ &= \sum_{i=1}^N c_i x_{(i)} \cdot w_{\perp} && \text{(inner product distributes over a sum)} \\ &= \sum_{i=1}^N c_i \cdot 0 && (w_{\perp} \text{ is orthogonal to each } x_{(i)}) \\ &= 0. \end{aligned}$$

So  $\|w\|_2^2 = \|w_{\parallel}\|_2^2 + \|w_{\perp}\|_2^2$ , which is Pythagoras' theorem.

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm>

<sup>b</sup><https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-orthogonal-matrix>

**Exercise 2.10** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 4)). Use the setup of Exercise 2.7.

Suppose  $(w, b)$  solves Definition 1.7 and  $w_{\perp} \neq 0$ . Find a feasible point with a smaller objective.

Solution

*Solution 2.10.*  $(w_{\parallel}, b)$  is feasible, by Solution 2.8. Its objective is

$$\begin{aligned}\frac{1}{2}\|w_{\parallel}\|_2^2 &= \frac{1}{2}\|w\|_2^2 - \frac{1}{2}\|w_{\perp}\|_2^2 \quad (@\text{sol-w-from-the-inputs-part3}) \\ &< \frac{1}{2}\|w\|_2^2 \quad (w_{\perp} \neq 0, \text{ so } \|w_{\perp}\|_2^2 > 0),\end{aligned}$$

smaller than the objective at  $(w, b)$ , which contradicts  $(w, b)$  being a solution.

**Exercise 2.11** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 5)). Use the setup of Exercise 2.7. What does that say about  $w_{\perp}$  at the optimum?

Solution

*Solution 2.11.* So  $w_{\perp} = 0$  at the optimum.  $w_{\perp}$  adds to the objective and changes no constraint.

**Exercise 2.12** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 6)). Use the setup of Exercise 2.7. What form does the optimal  $w$  take?

Solution

*Solution 2.12.* Every optimal  $w$  is therefore

$$\begin{aligned}w &= w_{\parallel} \\ &= \sum_{i=1}^N c_i x_{(i)}.\end{aligned}$$

**Exercise 2.13** (Show that the optimal  $w$  is a weighted sum of the training inputs (part 7)). Use the setup of Exercise 2.7. What does your argument say about the values of the weights  $c_i$ ?

Solution

*Solution 2.13.* Nothing. The argument shows that weights  $c_i$  exist, not what they are. They need not even be unique: when the training inputs<sup>a</sup> are linearly dependent, as they must be whenever  $N > p$ , different weights give the same  $w$ .

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

**Theorem 2.1** (The optimal weight vector is a weighted sum of the training inputs). *If  $(w, b)$  solves Definition 1.7, then there are real numbers  $c_1, c_2, \dots, c_N$  with*

$$w = \sum_{i=1}^N c_i x_{(i)};$$

*that is,  $w$  is a linear combination of the training inputs, and lies in their span.*

**i** Proof

*Proof.* By Solution 2.7 to Solution 2.10. □

The same argument works for any problem whose constraints or loss see  $w$  only through the inner products  $w \cdot x_{(i)}$  and whose penalty on  $\|w\|$  grows with it. That general statement is the representer theorem [13], Theorem 1.

**Example 2.2** (The weight vector as a sum of the support vectors). For the ten cars, get  $w$  in two ways. First, solve the three equations  $y_{(i)}(w \cdot x_{(i)} + b) = 1$  of the support vectors for  $(w_1, w_2, b)$ . Second, form the weighted sum  $\sum_{i=1}^N c_i x_{(i)}$  of Theorem 2.1 with  $c_i = \alpha_i y_{(i)}$ :

## 2.3 Python

```
sv_wi = np.flatnonzero(alpha_mm > 1e-6)      # the support vectors
# Route 1: three linear equations in (w1, w2, b), one per support vector.
sol_wi = np.linalg.solve(np.column_stack([X_sep[sv_wi], np.ones(len(sv_wi))]), y_sep[sv_wi])
# Route 2: the weighted sum of the inputs.
c_wi = alpha_mm * y_sep
w_sum_wi = c_wi @ X_sep

print(pd.DataFrame({"name": cars_ls["name"].to_numpy(), "c": c_wi.round(4)}).iloc[sv_wi])

      name      c
3  plymouth 'cuda 340 -0.4174
4  buick estate wagon (sw) -0.2512
6      fiat 124b  0.6686

print(sol_wi[:2], w_sum_wi, abs(sol_wi[:2] - w_sum_wi).max())

[-0.90095085 -0.72491891] [-0.90095064 -0.72491875] 2.0680144907636588e-07
```

## 2.4 R

```
sv_wi <- which(alpha_mm > 1e-6)      # the support vectors
# Route 1: three linear equations in (w1, w2, b), one per support vector.
sol_wi <- solve(cbind(X_sep[sv_wi, ], 1), y_sep[sv_wi])
# Route 2: the weighted sum of the inputs.
c_wi <- alpha_mm * y_sep
w_sum_wi <- drop(c_wi %*% X_sep)

data.frame(name = cars_ls$name, c = round(c_wi, 4))[sv_wi, ]

# A tibble: 3 x 2
  name      c
  <chr>    <dbl>
1 plymouth 'cuda 340 -0.417
2 buick estate wagon (sw) -0.251
3 fiat 124b  0.669

rbind(equations = sol_wi[1:2], weighted_sum = w_sum_wi)

      [,1]      [,2]
equations -0.9009508 -0.7249189
weighted_sum -0.9009506 -0.7249187

max(abs(sol_wi[1:2] - w_sum_wi))

[1] 2.068014e-07
```

The two routes agree to  $2e-07$ , so  $w = (-0.901, -0.725)$  is a weighted sum of the inputs. The weights are zero for the 7 cars that are not support vectors, and nonzero only for the 3 that are. In two dimensions any vector is a combination of two non-parallel inputs, so the span claim is automatic here; what the example shows is which weights the theorem uses. The example after Theorem 2.2 checks the span claim with 8 inputs in 57 dimensions, where it is not automatic.

Theorem 2.1 proves that the hard-margin optimum is a weighted sum of the training inputs, by way of the constrained primal. The same description is available for an unconstrained empirical risk, and the argument is different. It follows what gradient descent does.

Take a linear model whose loss sees each training point only through the score  $w \cdot x_{(i)}$ , and let

$$R(w) = \frac{1}{N} \sum_{i=1}^N L(w \cdot x_{(i)}, y_{(i)})$$

be its empirical risk. By the chain rule,

$$\nabla R(w) = \frac{1}{N} \sum_{i=1}^N g_i x_{(i)}, \quad g_i = L'(w \cdot x_{(i)}, y_{(i)}),$$

where  $L'$  is the derivative of the loss with respect to its first argument. Whatever the loss is, this gradient is a weighted sum of the training inputs. Each  $x_{(i)}$  reaches the loss only through the scalar  $w \cdot x_{(i)}$ , so differentiating that scalar leaves  $x_{(i)}$  behind as the outer factor.

**Theorem 2.2** (Gradient descent from zero stays in the span of the training inputs). *Let  $L$  be differentiable in its first argument. Run gradient descent on  $R$  from  $w^{(0)} = 0$  with step size  $\alpha$ :*

$$w^{(t+1)} = w^{(t)} - \alpha \nabla R(w^{(t)}).$$

*Then, for every  $t$ , there are real numbers  $c_1^{(t)}, \dots, c_N^{(t)}$  with*

$$w^{(t)} = \sum_{i=1}^N c_i^{(t)} x_{(i)}.$$

### i Proof

*Proof.* By induction on  $t$ .

*Base step.*  $w^{(0)} = 0 = \sum_{i=1}^N 0 \cdot x_{(i)}$ , a weighted sum with every coefficient zero.

*Inductive step.* Assume  $w^{(t)} = \sum_{i=1}^N c_i^{(t)} x_{(i)}$ . Substituting the gradient into the update, with  $g_i^{(t)}$  the value of  $g_i$  at  $w^{(t)}$ ,

$$w^{(t+1)} = \sum_{i=1}^N c_i^{(t)} x_{(i)} - \frac{\alpha}{N} \sum_{i=1}^N g_i^{(t)} x_{(i)} = \sum_{i=1}^N \underbrace{\left( c_i^{(t)} - \frac{\alpha}{N} g_i^{(t)} \right)}_{c_i^{(t+1)}} x_{(i)},$$

which is again a weighted sum of the training inputs. □

**Example 2.3** (Gradient descent on spam emails). In the Spambase data [14], take the first four spam and the first four non-spam emails, so  $N = 8$  inputs<sup>a</sup> with  $p = 57$  standardized features. Run gradient descent from  $w^{(0)} = 0$  on the logistic loss, with step size  $\alpha = 0.5$  for 200 steps, and update the coefficients  $c_i^{(t)}$  of the proof alongside the weights. Then compare  $w^{(t)}$  with  $\sum_{i=1}^N c_i^{(t)} x_{(i)}$ :

## 2.5 Python

```

spam_gs = pd.read_csv("data/spambase.csv")
feat_gs = spam_gs.drop(columns="Class").to_numpy(float)
feat_gs = (feat_gs - feat_gs.mean(axis=0)) / feat_gs.std(axis=0, ddof=1)
lab_gs = np.where(spam_gs["Class"] == 1, 1.0, -1.0)
pick_gs = np.r_[np.flatnonzero(lab_gs > 0)[:4], np.flatnonzero(lab_gs < 0)[:4]]
X_gs, y_gs = feat_gs[pick_gs], lab_gs[pick_gs]

h_gs = 0.5
w_gs, c_gs = np.zeros(X_gs.shape[1]), np.zeros(len(y_gs))
for _ in range(200):
    # g_i: derivative of the logistic loss with respect to the score w.x_i.
    g_gs = -y_gs / (1 + np.exp(y_gs * (X_gs @ w_gs)))
    w_gs = w_gs - h_gs * (g_gs @ X_gs) / len(y_gs) # gradient descent on the weights
    c_gs = c_gs - h_gs * g_gs / len(y_gs) # the same step, on the coefficients
print(X_gs.shape, np.linalg.matrix_rank(X_gs))

(8, 57) 8

print(np.linalg.norm(w_gs), np.abs(w_gs - X_gs.T @ c_gs).max())

3.2999845761032036 3.219646771412954e-15

```

## 2.6 R

```

spam_gs <- read.csv("data/spambase.csv")
feat_gs <- scale(as.matrix(spam_gs[, names(spam_gs) != "Class"]))
lab_gs <- ifelse(spam_gs$Class == 1, 1, -1)
pick_gs <- c(head(which(lab_gs > 0), 4), head(which(lab_gs < 0), 4))
X_gs <- feat_gs[pick_gs, ]
y_gs <- lab_gs[pick_gs]

h_gs <- 0.5
w_gs <- numeric(ncol(X_gs))
c_gs <- numeric(length(y_gs))
for (step in 1:200) {
    # g_i: derivative of the logistic loss with respect to the score w.x_i.
    g_gs <- -y_gs / (1 + exp(y_gs * drop(X_gs %*% w_gs)))
    w_gs <- w_gs - h_gs * drop(g_gs %*% X_gs) / length(y_gs) # gradient descent on the weights
    c_gs <- c_gs - h_gs * g_gs / length(y_gs) # the same step, on the coefficients
}
c(dim(X_gs), rank = qr(X_gs)$rank)

      rank
8  57   8

c(norm_w = sqrt(sum(w_gs^2)), max_diff = max(abs(w_gs - drop(t(X_gs) %*% c_gs))))

      norm_w      max_diff
3.299985e+00 2.220446e-15

```

The weight vector lives in  $\mathbb{R}^{57}$ , but the 8 training inputs span only 8 of those dimensions. After 200 steps  $w$  has norm 3.30, and it differs from the weighted sum  $\sum_{i=1}^N c_i^{(200)} x_{(i)}$  by at most  $2e-15$ , which is rounding error. So Theorem 2.2 holds here:  $w^{(200)}$  lies in the 8-dimensional span of the inputs, though nothing forced it to in  $\mathbb{R}^{57}$ .

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

So every iterate lies in the span of the training inputs, and so does anything the iterates converge to, since a span is closed. The argument did not use the particular loss, such as squared error. It used only that the loss sees each input through  $w \cdot x_{(i)}$ , and that the starting point was in the span. This is the representer theorem

in its simplest form [13]. With  $N$  coefficients rather than  $p$ , it matters whenever  $p$  is the larger of the two.

**i Remark** (What we still assume without proof)

*Remark 2.1* (What we still assume without proof). Theorem 2.1 says weights  $c_i$  exist; the derivation of Definition 2.1, which needs Lagrange multipliers and is out of scope, supplies a particular choice of them:

$$c_i = \alpha_i y_{(i)},$$

with  $\alpha$  the solution of Definition 2.1. Theorem 2.1 is proved; that identification is not, and the rest of this section assumes it without proof.

Writing  $c_i$  as  $\alpha_i y_{(i)}$  is always possible: set  $\alpha_i = c_i y_{(i)}$ , and then

$$\begin{aligned} \alpha_i y_{(i)} &= c_i y_{(i)}^2 \\ &= c_i, \end{aligned}$$

since  $y_{(i)}^2 = 1$ . What the dual derivation adds is that, for its choice of weights, the  $\alpha_i$  are non-negative, so each non-zero weight has the sign of its own point's label, and that they solve Definition 2.1.

**Exercise 2.14** (Check that the dual objective is the primal objective in disguise). The dual's two terms look unrelated to anything in Definition 1.7. They are not. Theorem 2.1 makes the optimal weight vector a weighted sum of the training inputs, and Remark 2.1 supplies the weights, without proof:

$$w = \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)},$$

each training input weighted by its own  $\alpha_i$  and signed by its label.

Substitute that into  $\frac{1}{2}\|w\|_2^2$  and expand. What do you get?

**Solution**

*Solution 2.14.* Write the squared norm as an inner product and substitute both copies of  $w$ :

$$\begin{aligned} \frac{1}{2}\|w\|_2^2 &= \frac{1}{2} w \cdot w \\ &= \frac{1}{2} \left( \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)} \right)^\top \left( \sum_{j=1}^N \alpha_j y_{(j)} x_{(j)} \right) \\ &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} x_{(i)} \cdot x_{(j)}. \end{aligned}$$

The second index has to be renamed, because the two sums run independently. That is where the double sum comes from, and why the training inputs meet each other pairwise.

**Exercise 2.15** (Check that the dual objective is the primal objective in disguise (part 2)). Use the setup of Exercise 2.14.

The dual *maximizes* the negative of that quantity. What is it therefore doing to  $\|w\|_2$ , and how does that compare with the primal?

**Solution**

*Solution 2.15.* Maximizing the negative of  $\frac{1}{2}\|w\|_2^2$  is minimizing  $\frac{1}{2}\|w\|_2^2$ , which is exactly the primal's objective. So the dual's second term is the primal's whole objective with its sign flipped, and the constraints contribute the first term.

**Exercise 2.16** (Check that the dual objective is the primal objective in disguise (part 3)). Use the setup of Exercise 2.14.

Using  $\alpha_i \geq 0$ , rewrite  $\sum_{i=1}^N \alpha_i$  as a norm<sup>a</sup> of the vector  $\alpha$ .

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm>

Solution

*Solution 2.16.* A non-negative number is its own absolute value, so

$$\begin{aligned}\sum_{i=1}^N \alpha_i &= \sum_{i=1}^N |\alpha_i| \\ &= \|\alpha\|_1.\end{aligned}$$

**Exercise 2.17** (Check that the dual objective is the primal objective in disguise (part 4)). Use the setup of Exercise 2.14.

Minimizing that norm<sup>a</sup> is the standard way to make a vector sparse. The dual maximizes it. Should you expect  $\alpha$  to be sparse or dense?

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm>

Solution

*Solution 2.17.* Dense, on that reasoning. An  $\ell_1$  penalty sparsifies when it is *minimized*; here it is being maximized, which pushes the  $\alpha_i$  up rather than down.

The reasoning is sound and the conclusion is wrong:  $\alpha$  is sparse, usually very sparse. Theorem 2.3 says why, and the cause has nothing to do with the  $\ell_1$  norm<sup>a</sup>. Treat this as a warning about reading an objective's shape and inferring a property of its solution.

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm>

**Exercise 2.18** (Find out why  $\alpha$  is sparse). Solution 2.14 predicted a dense  $\alpha$  and was wrong. The real cause is a condition every solution of a problem like this satisfies, called **complementary slackness**:

$$\alpha_i (y_{(i)} (w \cdot x_{(i)} + b) - 1) = 0, \quad i = 1, 2, \dots, N.$$

A product is zero when one of its factors is. Each  $i$  therefore has at most one non-zero factor, and the condition says which combinations are allowed.

Suppose  $\alpha_i > 0$ . What must be true of  $y_{(i)} h(x_{(i)})$ , and where does that put  $x_{(i)}$ ?

Solution

*Solution 2.18.* If  $\alpha_i > 0$  then the first factor is non-zero, so the second must vanish:

$$y_{(i)} (w \cdot x_{(i)} + b) = 1.$$

By the normalization of Theorem 1.2 that is exactly the condition for sitting on the margin, at distance  $M$  from the hyperplane.

**Exercise 2.19** (Find out why  $\alpha$  is sparse (part 2)). Use the setup of Exercise 2.18.

Suppose  $x_{(i)}$  sits strictly outside the margin, so that  $y_{(i)} h(x_{(i)}) > 1$ . What must  $\alpha_i$  be?

Solution

*Solution 2.19.* If  $y_{(i)}h(x_{(i)}) > 1$  then the second factor is strictly positive, so  $\alpha_i = 0$ .

**Exercise 2.20** (Find out why  $\alpha$  is sparse (part 3)). Use the setup of Exercise 2.18.

Parts 1 and 2 each state an implication. Write down the converse of Exercise 2.18 and decide whether it follows from complementary slackness.

Solution

*Solution 2.20.* The converse of Solution 2.18 is: if  $x_{(i)}$  lies on the margin then  $\alpha_i > 0$ . It does **not** follow. Complementary slackness only forbids the combination “ $\alpha_i > 0$  and off the margin”; it permits “ $\alpha_i = 0$  and on the margin”, since then both factors are zero and the product is zero for two reasons.

Such points exist. They are unusual, and they are the reason the two definitions of a [support vector](#) in use are not quite the same: Definition 1.5 calls a point a support vector when it lies at distance  $M$ , and many sources instead call it one when  $\alpha_i \neq 0$ . The second set is contained in the first and can be strictly smaller.

**Exercise 2.21** (Find out why  $\alpha$  is sparse (part 4)). Use the setup of Exercise 2.18.

Most points in a training set sit strictly outside the margin. Combine that with Exercise 2.19 to explain the sparsity of  $\alpha$ .

Solution

*Solution 2.21.* Solution 2.19 gives  $\alpha_i = 0$  for every point strictly outside the margin. That is almost all of them, so almost all  $\alpha_i$  are zero and  $\alpha$  is sparse. The sparsity comes from the *constraints* being slack, not from the objective’s shape. The  $\ell_1$  reasoning in Solution 2.14, correct about  $\ell_1$  norms<sup>a</sup>, said nothing about this problem.

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm>

**Exercise 2.22** (Find out why  $\alpha$  is sparse (part 5)). Use the setup of Exercise 2.18.

Using  $w = \sum_i \alpha_i y_{(i)} x_{(i)}$ , say which training points the fitted hyperplane actually depends on.

Solution

*Solution 2.22.* Only the points with  $\alpha_i \neq 0$  contribute a term to the sum, so  $w$  is a combination of those points alone. Every other training example could be deleted and the same  $w$  would result. That is Solution 1.11’s claim, now proved rather than argued from a picture, and the method is named after it.

**Theorem 2.3** (Only points on the margin carry weight). *Suppose  $w$ ,  $b$  and  $\alpha$  satisfy complementary slackness [15],*

$$\alpha_i (y_{(i)} (w \cdot x_{(i)} + b) - 1) = 0, \quad i = 1, 2, \dots, N,$$

*for a hard-margin training set scaled as in Definition 1.7, where  $h(x) = w \cdot x + b$ . If  $\alpha_i > 0$ , then  $y_{(i)}h(x_{(i)}) = 1$ , so  $x_{(i)}$  lies on the margin. Equivalently,  $\alpha_i = 0$  for every point strictly outside the margin, where  $y_{(i)}h(x_{(i)}) > 1$ .*

### **i** Proof

*Proof.* This is the solution to Exercise 2.18 and Exercise 2.19. □

**Exercise 2.23** (Get  $w$  and  $b$  back out of  $\alpha$ ). A solver gives you the optimal  $\alpha$ . You still have to classify a new point, which needs  $w$  and  $b$ . Write  $w$  in terms of  $\alpha$  and the training set.

### Solution

*Solution 2.23.*

$$w = \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)},$$

which by Theorem 2.3 is really a sum over the support vectors, the other terms being zero.

**Exercise 2.24** (Get  $w$  and  $b$  back out of  $\alpha$  (part 2)). Use the setup of Exercise 2.23. Take any  $i$  with  $\alpha_i \neq 0$ . Theorem 2.3 says what  $y_{(i)}(w \cdot x_{(i)} + b)$  equals. Solve that for  $b$ , using  $y_{(i)} \in \{-1, +1\}$ .

### Solution

*Solution 2.24.* Complementary slackness gives  $y_{(i)}(w \cdot x_{(i)} + b) = 1$ . Multiply both sides by  $y_{(i)}$ . Since  $y_{(i)} \in \{-1, +1\}$  we have  $y_{(i)}^2 = 1$ , so the left side simplifies and

$$w \cdot x_{(i)} + b = y_{(i)}, \quad \text{that is} \quad b = y_{(i)} - w \cdot x_{(i)}.$$

**Exercise 2.25** (Get  $w$  and  $b$  back out of  $\alpha$  (part 3)). Use the setup of Exercise 2.23. Your formula for  $b$  names a particular  $i$ . Would a different choice of  $i$  give a different  $b$ ?

### Solution

*Solution 2.25.* No. Every  $i$  with  $\alpha_i \neq 0$  satisfies the same equation, so each returns the same  $b$ . In floating-point arithmetic they return nearly the same  $b$ , so implementations average over all of them rather than picking one.

**Exercise 2.26** (Get  $w$  and  $b$  back out of  $\alpha$  (part 4)). Use the setup of Exercise 2.23. Is the set of **support vectors** determined by the training data?

### Solution

*Solution 2.26.* The **support vectors** need not be. Sitting on the margin is a geometric fact about a point and is determined, but which points carry non-zero  $\alpha_i$  can fail to be unique when more points lie on the margin than are needed to fix the hyperplane: the surplus weight can be shared among them in more than one way. So two runs of a solver may return different  $\alpha$  and the same classifier.

**Exercise 2.27** (Get  $w$  and  $b$  back out of  $\alpha$  (part 5)). Use the setup of Exercise 2.23. Are  $w$  and  $b$ ?

## Solution

*Solution 2.27.*  $w$  and  $b$  are determined — the primal is a strictly convex<sup>a</sup> problem in  $w$  with a unique solution, and Solution 2.24 then fixes  $b$ .

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

**Proposition 2.1** (Recovering  $w$  and  $b$  from the dual). *Let  $\alpha$  solve the hard-margin dual of Definition 2.1, and let  $(w, b)$  be the corresponding solution of Definition 1.7. Then*

$$w = \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)},$$

and, for every  $i$  with  $\alpha_i \neq 0$ ,

$$b = y_{(i)} - w \cdot x_{(i)}.$$

## i Proof

*Proof.* This is the solution to Exercise 2.23. The formula for  $b$  uses Theorem 2.3. □

**Example 2.4** (Recovering  $w$  and  $b$  for the ten cars). The solver of Example 1.3 returns the dual variables  $\alpha_i$  and the intercept  $b$ . Rebuild  $w = \sum_{i=1}^N \alpha_i y_{(i)} x_{(i)}$  from the dual variables alone, then apply the formula  $b = y_{(i)} - w \cdot x_{(i)}$  of Proposition 2.1 at every car with  $\alpha_i \neq 0$ , and compare each result with the solver's  $b$ :

## 2.7 Python

```
w_rb = (alpha_mm * y_sep) @ X_sep          # w = sum of alpha_i y_i x_i
sv_rb = np.flatnonzero(alpha_mm > 1e-6)    # the cars with alpha_i != 0
b_rb = y_sep[sv_rb] - X_sep[sv_rb] @ w_rb  # one value of b per support vector
print(np.abs(w_rb - w_mm).max())

0.0

print(pd.DataFrame({"name": cars_ls["name"].to_numpy()[sv_rb], "b": b_rb.round(6)}))

           name      b
0  plymouth 'cuda 340  3.411401
1  buick estate wagon (sw)  3.411401
2           fiat 124b  3.411401

print(b_mm)

3.4114010199672364
```

## 2.8 R

```
w_rb <- drop((alpha_mm * y_sep) %*% X_sep)  # w = sum of alpha_i y_i x_i
sv_rb <- which(alpha_mm > 1e-6)            # the cars with alpha_i != 0
b_rb <- y_sep[sv_rb] - drop(X_sep[sv_rb, , drop = FALSE] %*% w_rb)  # one value of b per support vect
max(abs(w_rb - w_mm))

[1] 0

data.frame(name = cars_ls$name[sv_rb], b = round(b_rb, 6))

# A tibble: 3 x 2
  name      b
  <chr> <dbl>
1  plymouth 'cuda 340  3.411401
2  buick estate wagon (sw)  3.411401
3           fiat 124b  3.411401
```

```

    <chr>                <dbl>
1 plymouth 'cuda 340    3.41
2 buick estate wagon (sw) 3.41
3 fiat 124b             3.41

```

```
b_mm
```

```
[1] 3.411401
```

The weight vector rebuilt from the dual variables matches the solver's  $w$ , with largest difference 0. Every one of the 3 support vectors gives the same  $b$  to six decimals, 3.4114, and it is the solver's  $b$ , so any one of them recovers the intercept.

### i Watch

Andrew Ng derives the dual problem in the CS229 lecture on [kernels](#), from 26:31<sup>a</sup> [9], Lecture 7.

<sup>a</sup><https://www.youtube.com/watch?v=8NYoQiRANpg&t=1591s>

### i Read

GeeksforGeeks explains the dual problem of the SVM<sup>a</sup> [16].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/dual-support-vector-machine/>

## 3 Soft-margin SVMs

**Exercise 3.1** (Find out what breaks when the classes overlap). Every problem so far assumed the training set is linearly separable. Real data usually is not. Look at Definition 1.7 and say precisely what goes wrong when no hyperplane separates the classes. It is not that the answer is bad.

### Solution

*Solution 3.1.* The feasible set is empty. Every point has to satisfy  $y_{(i)}(w \cdot x_{(i)} + b) \geq 1$ , and if no hyperplane gets all of them on the right side then no  $(w, b)$  satisfies all  $N$  constraints at once. The problem has no solution to return, good or bad. A solver reports infeasibility rather than a poor classifier.

**Exercise 3.2** (Find out what breaks when the classes overlap (part 2)). Use the setup of Exercise 3.1. A fix has to let a point sit closer to the boundary than the margin, or even on the wrong side, while still costing something. Propose a way to write that into the constraint  $y_{(i)}(w \cdot x_{(i)} + b) \geq 1$ .

### Solution

*Solution 3.2.* Give each point an allowance for being short of the margin:

$$y_{(i)}(w \cdot x_{(i)} + b) \geq 1 - \xi_i,$$

and charge for the allowances in the objective. With  $\xi_i$  free to grow, every point can be satisfied, so the feasible set is never empty again.

**Exercise 3.3** (Find out what breaks when the classes overlap (part 3)). Use the setup of Exercise 3.1. Your fix introduces a new unknown per training point. What should it be when the point is correct and far from the boundary?

Solution

*Solution 3.3.* Zero. A point at or beyond the margin already satisfies the original constraint, so it needs no allowance, and charging it for one it does not use would be arbitrary.

**Exercise 3.4** (Find out what breaks when the classes overlap (part 4)). Use the setup of Exercise 3.1. What would a negative  $\xi_i$  represent?

Solution

*Solution 3.4.* A negative  $\xi_i$  would describe a point sitting *beyond* the margin by  $|\xi_i|$ .

**Exercise 3.5** (Find out what breaks when the classes overlap (part 5)). Use the setup of Exercise 3.1. Should  $\xi_i$  be allowed to go negative?

Solution

*Solution 3.5.* No, and the reason matters more than the rule. Since the objective charges  $\sum_i \xi_i$  and we are minimizing, a negative allowance pays a *bonus*. The sum would stop measuring how badly the data violate the margin and start measuring how far every point sits from the boundary, so a correctly classified point far from the boundary would go on affecting the solution forever. That is precisely the behaviour Solution 1.8 removed. Requiring  $\xi_i \geq 0$  keeps the quantity a *penalty*.

**Definition 3.1** (Slack variable). The **slack variable**  $\xi_i \geq 0$  is the amount by which  $y_{(i)}h(x_{(i)})$  is allowed to be smaller than 1 [7].

The letter is a lower-case xi, said “zai”, “k-sai” or “k-see” depending on whom you ask.

**Example 3.1** (Slack of the fuel-economy classifier). Return to the classifier  $h$  of Example 1.1. Rescale  $(w, b)$  so that the two class means sit on the margin hyperplanes,  $h = +1$  and  $h = -1$ , then compute the slack  $\xi_i = \max(0, 1 - y_{(i)}h(x_{(i)}))$  that each car needs, and count the cars by how much slack they use:

### 3.1 Python

```
# The mean of the high-mpg cars has h = (w.w)/2 before rescaling, so this factor makes it exactly +1.
scale_sl = 2 / (w_dr @ w_dr)
w_sl, b_sl = scale_sl * w_dr, scale_sl * b_dr
f_sl = Z_dr @ w_sl + b_sl
xi_sl = np.maximum(0, 1 - y_dr * f_sl)      # the smallest slack that satisfies y h >= 1 - slack

print(pd.DataFrame({
    "slack": ["xi = 0", "0 < xi <= 1", "xi > 1"],
    "cars": [(xi_sl == 0).sum(), ((xi_sl > 0) & (xi_sl <= 1)).sum(), (xi_sl > 1).sum()],
}))
```

|   | slack       | cars |
|---|-------------|------|
| 0 | xi = 0      | 202  |
| 1 | 0 < xi <= 1 | 137  |

```
2      xi > 1      53
```

```
worst_sl = xi_sl.argmax()
print(auto_dr["name"][worst_sl], auto_dr["mpg"][worst_sl], xi_sl[worst_sl])

cadillac eldorado 23.0 2.158013540143878
```

### 3.2 R

```
# The mean of the high-mpg cars has h = (w.w)/2 before rescaling, so this factor makes it exactly +1.
scale_sl <- 2 / sum(w_dr^2)
w_sl <- scale_sl * w_dr
b_sl <- scale_sl * b_dr
f_sl <- drop(Z_dr %*% w_sl) + b_sl
xi_sl <- pmax(0, 1 - y_dr * f_sl) # the smallest slack that satisfies y h >= 1 - slack

data.frame(
  slack = c("xi = 0", "0 < xi <= 1", "xi > 1"),
  cars = c(sum(xi_sl == 0), sum(xi_sl > 0 & xi_sl <= 1), sum(xi_sl > 1))
)

# A tibble: 3 x 2
  slack      cars
  <chr>    <int>
1 xi = 0      202
2 0 < xi <= 1  137
3 xi > 1       53

worst_sl <- which.max(xi_sl)
c(auto_dr$name[worst_sl], auto_dr$mpg[worst_sl], xi_sl[worst_sl])
```

```
[1] "cadillac eldorado" "23" "2.15801354014388"
```

Cars with  $\xi_i = 0$  are on or beyond their own margin hyperplane, and need no slack. The 137 cars with  $0 < \xi_i \leq 1$  are on the correct side of the boundary but inside the margin, and the 53 with  $\xi_i > 1$  are misclassified. The worst offender is the cadillac eldorado, at 23 mpg, with slack 2.16. The total slack is  $\sum_{i=1}^N \xi_i = 143.6$ , the quantity that the soft-margin objective penalizes.

**Exercise 3.6** (Assemble the soft-margin program). Solution 3.1 has the relaxed constraint and Definition 3.1 names the new unknown. Turn them into a complete optimization problem. Write down every constraint the problem now carries, and count them.

Solution

*Solution 3.6.* Two families,  $N$  constraints each:

$$y_{(i)}(w \cdot x_{(i)} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \dots, N,$$

so  $2N$  in all, against the  $N$  of Definition 1.7.

**Exercise 3.7** (Assemble the soft-margin program (part 2)). Use the setup of Exercise 3.6. The objective has to trade a wide margin against small violations. Write it, introducing a constant<sup>a</sup>  $C$  for the trade-off between the two.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

### Solution

*Solution 3.7.*

$$\arg \min_{w, b, \xi} \quad \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i.$$

The first term is the one Solution 1.20 arrived at, and minimizing it widens the margin. The second counts the violations.  $C > 0$  says how many units of  $\frac{1}{2} \|w\|_2^2$  one unit of violation is worth.

**Exercise 3.8** (Assemble the soft-margin program (part 3)). Use the setup of Exercise 3.6.

Suppose we treated  $C$  as one more unknown and let the solver minimize over it as well. What value would it choose, and what classifier would come back?

### Solution

*Solution 3.8.* It would choose  $C = 0$ . Whenever  $\sum_i \xi_i > 0$  the objective is strictly increasing in  $C$ , so a solver free to pick  $C$  drives it to the smallest value the constraints allow. At  $C = 0$  the violations cost nothing, so each  $\xi_i$  can grow until its constraint is satisfied no matter what  $(w, b)$  is, and the objective is then minimized by  $w = 0$  — a classifier with no decision boundary at all.

So  $C$  cannot be fitted on the training data. It is a hyperparameter, fixed before the fit and chosen by comparing held-out performance across candidate values.

**Exercise 3.9** (Assemble the soft-margin program (part 4)). Use the setup of Exercise 3.6.

How many unknowns does the problem have now, and is it still a convex<sup>a</sup> problem?

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

### Solution

*Solution 3.9.*  $p + 1 + N$ : the  $p$  entries of  $w$ , the intercept<sup>a</sup>  $b$ , and one  $\xi_i$  per training point. It is still convex<sup>b</sup>. The objective is a convex quadratic in  $w$  plus a linear function of  $\xi$ , and every constraint is affine. It is strictly convex in  $w$  but not in  $b$  or  $\xi$ , so  $w$  is unique and the rest need not be — the same situation as Solution 2.23 describes for the hard margin.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

<sup>b</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

**Definition 3.2** (C-SVM primal problem). The **C-SVM primal problem**, for a constant  $C > 0$ , is

$$\begin{aligned} \arg \min_{w, b, \xi} \quad & \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_{(i)}(w \cdot x_{(i)} + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N \\ & \xi_i \geq 0, \quad i = 1, 2, \dots, N. \end{aligned}$$

Unlike Definition 1.7 it is feasible for every training set, separable or not.

**Example 3.2** (Evaluating the C-SVM primal objective). Each choice of  $C$  gives a different Definition 3.2 on the fuel-economy data of Example 1.1. Minimize it for  $C = 0.1$  and  $C = 1$  with the `dual_svm` helper of Example 1.3, which finds the same hyperplane through the dual problem, and evaluate the objective at the result. Compare with its value at the rescaled class-mean classifier of Example 3.1, which is feasible with its slacks  $\xi_i$ :

### 3.3 Python

```
K_cp = Z_dr @ Z_dr.T
rows_cp = []
for C_cp in (0.1, 1.0):
    alpha_cp, b_cp = dual_svm(K_cp, y_dr, C_cp, tol=1e-8)
    w_cp = (alpha_cp * y_dr) @ Z_dr
    xi_cp = np.maximum(0, 1 - y_dr * (Z_dr @ w_cp + b_cp)) # optimal slack for this (w, b)
    rows_cp.append([C_cp, 0.5 * w_cp @ w_cp + C_cp * xi_cp.sum(),
                    0.5 * w_cp @ w_cp + C_cp * xi_cp.sum(),
                    0.5 * w_sl @ w_sl + C_cp * xi_sl.sum(), int((xi_cp > 1e-6).sum())])
print(pd.DataFrame(rows_cp, columns=["C", "margin_term", "slack_term", "objective",
                                     "objective_class_means", "cars_with_slack"]).round(3))
```

|   | C   | margin_term | ... | objective_class_means | cars_with_slack |
|---|-----|-------------|-----|-----------------------|-----------------|
| 0 | 0.1 | 1.377       | ... | 14.851                | 144             |
| 1 | 1.0 | 2.543       | ... | 144.081               | 117             |

[2 rows x 6 columns]

### 3.4 R

```
K_cp <- Z_dr %*% t(Z_dr)
rows_cp <- t(sapply(c(0.1, 1), function(C_cp) {
  fit_cp <- dual_svm(K_cp, y_dr, C_cp, tol=1e-8)
  w_cp <- drop((fit_cp$alpha * y_dr) %*% Z_dr)
  xi_cp <- pmax(0, 1 - y_dr * (drop(Z_dr %*% w_cp) + fit_cp$b)) # optimal slack for this (w, b)
  c(C = C_cp, margin_term = 0.5 * sum(w_cp^2), slack_term = C_cp * sum(xi_cp),
    objective = 0.5 * sum(w_cp^2) + C_cp * sum(xi_cp),
    objective_class_means = 0.5 * sum(w_sl^2) + C_cp * sum(xi_sl),
    cars_with_slack = sum(xi_cp > 1e-6))
}))
round(rows_cp, 3)
```

|      | C   | margin_term | slack_term | objective | objective_class_means | cars_with_slack |
|------|-----|-------------|------------|-----------|-----------------------|-----------------|
| [1,] | 0.1 | 1.377       | 11.853     | 13.231    | 14.851                | 144             |
| [2,] | 1.0 | 2.543       | 114.005    | 116.547   | 144.081               | 117             |

The objective has two terms:  $\frac{1}{2}\|w\|_2^2$  rewards a wide margin and  $C \sum_{i=1}^N \xi_i$  charges for violations. At  $C = 0.1$  the minimizer has objective 13.2, against 14.9 for the class-mean classifier; at  $C = 1$  it is 116.5 against 144.1. Raising  $C$  from 0.1 to 1 changes the number of cars with positive slack from 144 to 117, because the slack term is now weighted ten times as heavily.

**Exercise 3.10** (Work out what  $C$  controls).  $C$  is the only parameter Definition 3.2 adds. Find out what changing it does. Which value of  $C$  recovers Definition 1.7: 0,  $-\infty$ , or  $\infty$ ?

Solution

*Solution 3.10.*  $C \rightarrow \infty$ , read as a limit rather than as a value.

**Exercise 3.11** (Work out what  $C$  controls (part 2)). Use the setup of Exercise 3.10. Say what happens to the  $\xi_i$  at that value.

### Solution

*Solution 3.11.* Every unit of violation then costs without limit, so the minimizer sets  $\xi_i = 0$  for every  $i$  whenever that is feasible, and the constraints collapse back to  $y_{(i)}(w \cdot x_{(i)} + b) \geq 1$ . At  $C = \infty$  the objective is undefined as soon as any  $\xi_i > 0$ , and on a training set that is not separable no finite  $C$  ever forces  $\xi = 0$  — the violations shrink as  $C$  grows but never reach zero.

**Exercise 3.12** (Work out what  $C$  controls (part 3)). Use the setup of Exercise 3.10. Take the two values you rejected in Exercise 3.10 in turn and say what goes wrong with each.

### Solution

*Solution 3.12.* At  $C = 0$  the violations are free. Each  $\xi_i$  can grow until its constraint holds regardless of  $(w, b)$ , so nothing stops  $w = 0$ , which is Solution 3.6's classifier with no boundary. At  $C = -\infty$ , and indeed at any  $C < 0$ , the objective rewards violations. Sending a single  $\xi_i \rightarrow \infty$  sends the objective to  $-\infty$ , so the problem has no minimum at all. This is Solution 3.1's penalty-into-bonus reversal arising through the cost  $C$  rather than through the sign of  $\xi_i$ .

**Exercise 3.13** (Work out what  $C$  controls (part 4)). Use the setup of Exercise 3.10. Is  $C$  positively correlated with the margin, negatively correlated, or neither? Argue from which of the two terms in the objective dominates.

### Solution

*Solution 3.13.* Negatively correlated. A large  $C$  makes the violation term dominate, so the solver accepts a large  $\frac{1}{2}\|w\|_2^2$  to avoid violations:  $\|w\|_2$  grows and the margin  $M = 1/\|w\|_2$  shrinks. A small  $C$  makes the margin term dominate, so the solver widens the margin and accepts the violations that come with it.

**Exercise 3.14** (Work out what  $C$  controls (part 5)). Use the setup of Exercise 3.10. Exercise 3.12 makes  $C$  sound like a scale between two extremes. Name the two failure modes at the ends of that scale, in the vocabulary of A First Model and Generalization.

### Solution

*Solution 3.14.* Large  $C$  overfits and small  $C$  underfits. A large  $C$  gives a narrow margin twisted to get the training points right, including the mislabelled ones, which is high variance<sup>a</sup>. A small  $C$  gives a wide margin that ignores much of the data, which is high bias<sup>b</sup>. Choosing  $C$  is choosing a point on the bias-variance trade-off, so it is settled on held-out data rather than on the training set.

<sup>a</sup><https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

<sup>b</sup><https://morrison-lab.github.io/sds/estimation.html#def-bias>

**Exercise 3.15** (Prove that the margin shrinks as  $C$  grows). Solution 3.11 argued from which term dominates. The claim can be proved outright, with no calculus and no differentiation of the solution. Choose a training set and write  $S = \sum_{i=1}^N \xi_i$  for the total violation. Take two values  $0 < C_1 < C_2$  of the penalty weight  $C$  and let  $(w_1, b_1, \xi_1)$  and  $(w_2, b_2, \xi_2)$  solve Definition 3.2 at  $C_1$  and  $C_2$ , with totals  $S_1$  and  $S_2$ . Each solution is optimal at its own value of  $C$ , so it scores at least as well there as the other solution does. Write these two facts as inequalities.

### Solution

*Solution 3.15.* The point  $(w_2, b_2, \xi_2)$  is feasible for the problem at  $C_1$ , and  $(w_1, b_1, \xi_1)$  is optimal there, so

$$\frac{1}{2}\|w_1\|_2^2 + C_1 S_1 \leq \frac{1}{2}\|w_2\|_2^2 + C_1 S_2.$$

Swapping the roles gives

$$\frac{1}{2}\|w_2\|_2^2 + C_2 S_2 \leq \frac{1}{2}\|w_1\|_2^2 + C_2 S_1.$$

The feasible set does not depend on  $C$ , so each solution can be tested at the other's value of  $C$ .

**Exercise 3.16** (Prove that the margin shrinks as  $C$  grows (part 2)). Use the setup of Exercise 3.15. Add them and cancel. Deduce the sign of  $(C_2 - C_1)(S_2 - S_1)$ , and state how the total violation changes with  $C$ .

### Solution

*Solution 3.16.* Adding the two and cancelling  $\frac{1}{2}\|w_1\|_2^2 + \frac{1}{2}\|w_2\|_2^2$  from both sides:

$$C_1 S_1 + C_2 S_2 \leq C_1 S_2 + C_2 S_1,$$

that is  $(C_2 - C_1)(S_2 - S_1) \leq 0$ . Since  $C_2 > C_1$  the first factor is positive, so  $S_2 \leq S_1$ : the total violation never increases as  $C$  increases.

**Exercise 3.17** (Prove that the margin shrinks as  $C$  grows (part 3)). Use the setup of Exercise 3.15. Put Exercise 3.16 back into the first inequality of Exercise 3.15 to compare  $\|w_1\|_2$  with  $\|w_2\|_2$ .

### Solution

*Solution 3.17.* From Solution 3.16,  $S_2 - S_1 \leq 0$ , and  $C_1 > 0$ , so  $C_1(S_2 - S_1) \leq 0$ . The first inequality of Solution 3.15 rearranges to

$$\frac{1}{2}\|w_1\|_2^2 \leq \frac{1}{2}\|w_2\|_2^2 + C_1(S_2 - S_1) \leq \frac{1}{2}\|w_2\|_2^2,$$

so  $\|w_1\|_2 \leq \|w_2\|_2$ .

**Exercise 3.18** (Prove that the margin shrinks as  $C$  grows (part 4)). Use the setup of Exercise 3.15. State what Exercise 3.16 and Exercise 3.17 together say about the margin, and say which word in Solution 3.13 they justify and which they do not.

### Solution

*Solution 3.18.*  $\|w\|_2$  never decreases as  $C$  increases, so the margin  $M = 1/\|w\|_2$  never increases: raising  $C$  gives a narrower margin and less total violation, exactly the trade Solution 3.11 described.

What is justified is the *direction*. What is not is the word “correlated”, which suggests a strict relationship: these are non-strict inequalities, so a range of  $C$  values can share one solution and leave the margin unchanged. Monotone is the accurate word.

**Lemma 3.1** (The total violation never increases with  $C$ ). Choose a training set, and take two values  $0 < C_1 < C_2$  of the penalty weight  $C$ . Let  $(w_1, b_1, \xi_1)$  and  $(w_2, b_2, \xi_2)$  solve Definition 3.2 at  $C_1$  and  $C_2$ ,

with total violations  $S_1 = \sum_{i=1}^N \xi_{1,i}$  and  $S_2 = \sum_{i=1}^N \xi_{2,i}$ . Then  $S_2 \leq S_1$ .

### i Proof

*Proof.* This is the solution to Exercise 3.15 and Exercise 3.16. □

**Example 3.3** (The total violation of the fuel-economy classifier as  $C$  grows). Fit Definition 3.2 on the fuel-economy data of Example 1.1 at three values of  $C$  with the `dual_svm` helper, as in Example 3.2, and compute the total violation  $S = \sum_{i=1}^N \xi_i$  of each fit:

## 3.5 Python

```
K_vm = Z_dr @ Z_dr.T
rows_vm = []
for C_vm in (0.01, 0.1, 1.0):
    alpha_vm, b_vm = dual_svm(K_vm, y_dr, C_vm, tol=1e-8)
    w_vm = (alpha_vm * y_dr) @ Z_dr
    S_vm = np.maximum(0, 1 - y_dr * (Z_dr @ w_vm + b_vm)).sum() # total violation at the optimal sl
    rows_vm.append([C_vm, S_vm, np.linalg.norm(w_vm)])
print(pd.DataFrame(rows_vm, columns=["C", "total_violation", "norm_w"]).round(3))
```

|   | C    | total_violation | norm_w |
|---|------|-----------------|--------|
| 0 | 0.01 | 150.942         | 0.842  |
| 1 | 0.10 | 118.533         | 1.660  |
| 2 | 1.00 | 114.005         | 2.255  |

## 3.6 R

```
K_vm <- Z_dr %*% t(Z_dr)
rows_vm <- t(sapply(c(0.01, 0.1, 1), function(C_vm) {
  fit_vm <- dual_svm(K_vm, y_dr, C_vm, tol = 1e-8)
  w_vm <- drop((fit_vm$alpha * y_dr) %*% Z_dr)
  S_vm <- sum(pmax(0, 1 - y_dr * (drop(Z_dr %*% w_vm) + fit_vm$b))) # total violation at the optimal
  c(C = C_vm, total_violation = S_vm, norm_w = sqrt(sum(w_vm^2)))
}))
round(rows_vm, 3)
```

|      | C    | total_violation | norm_w |
|------|------|-----------------|--------|
| [1,] | 0.01 | 150.942         | 0.842  |
| [2,] | 0.10 | 118.533         | 1.660  |
| [3,] | 1.00 | 114.005         | 2.255  |

The total violation falls from 150.94 at  $C = 0.01$  to 118.53 at  $C = 0.1$  and 114.00 at  $C = 1$ , so it never increases as  $C$  increases, as Lemma 3.1 says.

**Theorem 3.1** (The margin never increases with  $C$ ). In the setting of Lemma 3.1,  $\|w_1\|_2 \leq \|w_2\|_2$ . Hence the margin  $M = 1/\|w\|_2$  never increases as  $C$  increases.

### i Proof

*Proof.* This is the solution to Exercise 3.17 and Exercise 3.18. □

**Exercise 3.19** (Predict the soft-margin dual). Definition 2.1 has three ingredients: an objective, a sign constraint on each  $\alpha_i$ , and one equality constraint. Definition 3.2 changed the primal in one way only, by letting each point use an allowance of slack at cost  $C$  per unit. Which of the three ingredients do you expect to change?

Solution

*Solution 3.19.* Only the sign constraint. The objective and the equality constraint  $\sum_i \alpha_i y_{(i)} = 0$  come through unchanged, which is worth noticing: the soft margin adds one bound and nothing else.

**Exercise 3.20** (Predict the soft-margin dual (part 2)). Use the setup of Exercise 3.19. The change is a second bound on each  $\alpha_i$ . Write the constraint you expect, and say which number bounds it.

Solution

*Solution 3.20.*

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N.$$

The cost  $C$  is the bound. It is the same constant<sup>a</sup>, doing the same job seen from the other side: in the primal it is the cost of a unit of violation, and in the dual it is the most any one point may pay. A constraint of this shape, holding a variable between two constants, is called a box constraint.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

**Exercise 3.21** (Predict the soft-margin dual (part 3)). Use the setup of Exercise 3.19. Your constraint caps how large any single  $\alpha_i$  can be. Given  $w = \sum_i \alpha_i y_{(i)} x_{(i)}$ , say what that cap stops one badly placed training point from doing.

Solution

*Solution 3.21.* It stops that point from dominating  $w$ . In Definition 2.1 nothing bounds  $\alpha_i$  above, so a single point in the wrong place can carry an arbitrarily large weight and pull the hyperplane toward it. Capping every  $\alpha_i$  at  $C$  bounds one point's contribution to  $w$  by  $C \|x_{(i)}\|_2$ , so a mislabelled example can move the solution only so far.

That is the robustness the soft margin gives, and the dual is where it is visible as a single inequality.

**Exercise 3.22** (Predict the soft-margin dual (part 4)). Use the setup of Exercise 3.19. Check Exercise 3.20 against Exercise 3.10: what does your constraint become as  $C \rightarrow \infty$ ?

Solution

*Solution 3.22.* The upper bound disappears and  $0 \leq \alpha_i \leq C$  becomes  $\alpha_i \geq 0$ , which is Definition 2.1. The two problems agree in the limit in the dual exactly as they do in the primal.

**Definition 3.3** (C-SVM dual problem). The **C-SVM dual problem**, for a constant  $C > 0$ , is

$$\begin{aligned}
& \arg \max_{\alpha \in \mathbb{R}^N} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} x_{(i)} \cdot x_{(j)} \\
& \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \\
& \quad \sum_{i=1}^N \alpha_i y_{(i)} = 0,
\end{aligned}$$

whose solution determines the same hyperplane as Definition 3.2.

**Example 3.4** (Solving the C-SVM dual: American versus other cars). In the `Auto` data [5], label a car +1 if its `origin` is 1 (American) and -1 otherwise, and use the standardized `weight` and `horsepower` of Example 1.1 as inputs. Maximize Definition 3.3 with the `dual_svm` helper of Example 1.3 for  $C = 0.1$  and  $C = 1$ . Check the constraints, sort the dual variables into  $\alpha_i = 0$ ,  $0 < \alpha_i < C$  and  $\alpha_i = C$ , and compare the dual objective with the primal objective of Definition 3.2 at the  $w$  and slacks that the dual variables imply:

### 3.7 Python

```

y_cd = np.where(auto_dr["origin"] == 1, 1.0, -1.0)
rows_cd = []
for C_cd in (0.1, 1.0):
    alpha_cd, b_cd = dual_svm(K_cp, y_cd, C_cd, tol=1e-8)
    w_cd = (alpha_cd * y_cd) @ Z_dr # w from the dual variables
    xi_cd = np.maximum(0, 1 - y_cd * (Z_dr @ w_cd + b_cd)) # optimal slack for this (w, b)
    at_zero, at_C = alpha_cd < 1e-7, alpha_cd > C_cd - 1e-7
    rows_cd.append([C_cd, alpha_cd.sum() - 0.5 * (alpha_cd * y_cd) @ K_cp @ (alpha_cd * y_cd),
                    0.5 * w_cd @ w_cd + C_cd * xi_cd.sum(), abs((alpha_cd * y_cd).sum()),
                    at_zero.sum(), (~at_zero & ~at_C).sum(), at_C.sum()])
print(pd.DataFrame(rows_cd, columns=["C", "dual_objective", "primal_objective", "abs_sum_alpha_y",
                                     "alpha_0", "alpha_between", "alpha_C"]))

```

|   | C   | dual_objective | primal_objective | ... | alpha_0 | alpha_between | alpha_C |
|---|-----|----------------|------------------|-----|---------|---------------|---------|
| 0 | 0.1 | 19.952291      | 19.952291        | ... | 181     | 3             | 208     |
| 1 | 1.0 | 187.594365     | 187.594365       | ... | 201     | 3             | 188     |

[2 rows x 7 columns]

### 3.8 R

```

y_cd <- ifelse(auto_dr$origin == 1, 1, -1)
rows_cd <- t(sapply(c(0.1, 1), function(C_cd) {
  fit_cd <- dual_svm(K_cp, y_cd, C_cd, tol=1e-8)
  ay_cd <- fit_cd$alpha * y_cd
  w_cd <- drop(ay_cd %*% Z_dr) # w from the dual variables
  xi_cd <- pmax(0, 1 - y_cd * (drop(Z_dr %*% w_cd) + fit_cd$b)) # optimal slack for this (w, b)
  at_zero <- fit_cd$alpha < 1e-7
  at_C <- fit_cd$alpha > C_cd - 1e-7
  c(C = C_cd, dual_objective = sum(fit_cd$alpha) - 0.5 * drop(t(ay_cd) %*% K_cp %*% ay_cd),
    primal_objective = 0.5 * sum(w_cd^2) + C_cd * sum(xi_cd), abs_sum_alpha_y = abs(sum(ay_cd)),
    alpha_0 = sum(at_zero), alpha_between = sum(!at_zero & !at_C), alpha_C = sum(at_C))
}))
rows_cd

```

|      | C   | dual_objective | primal_objective | abs_sum_alpha_y | alpha_0 | alpha_between |
|------|-----|----------------|------------------|-----------------|---------|---------------|
| [1,] | 0.1 | 19.95229       | 19.95229         | 1.179612e-16    | 181     | 3             |

|      |         |           |           |              |     |   |
|------|---------|-----------|-----------|--------------|-----|---|
| [2,] | 1.0     | 187.59436 | 187.59436 | 1.110223e-16 | 201 | 3 |
|      | alpha_C |           |           |              |     |   |
| [1,] |         | 208       |           |              |     |   |
| [2,] |         | 188       |           |              |     |   |

The constraint  $\sum_{i=1}^N \alpha_i y_{(i)} = 0$  holds up to rounding, at most  $1e-16$  in absolute value, and the box constraint  $0 \leq \alpha_i \leq C$  holds by construction. At  $C = 1$  the dual objective is 187.59 and the primal objective is 187.59: the two problems have the same optimal value, as Definition 3.3 states. Of the 392 cars, 201 have  $\alpha_i = 0$  at  $C = 1$ , and so play no part in  $w$ ; 3 lie strictly between 0 and  $C$ , on the margin, and 188 are at the cap  $C$ , inside the margin or misclassified. Raising  $C$  from 0.1 to 1 raises the cap on each  $\alpha_i$  and lowers the number of cars at it from 208 to 188.

**Exercise 3.23** (Sort the training points by their  $\alpha_i$ ). Theorem 2.3 sorted the hard-margin training points into two kinds using one complementary-slackness condition. The soft-margin problem has two conditions, one per family of constraints in Definition 3.2:

$$\alpha_i (y_{(i)} h(x_{(i)}) - 1 + \xi_i) = 0, \quad (C - \alpha_i) \xi_i = 0.$$

The box constraint  $0 \leq \alpha_i \leq C$  leaves three cases. Take each in turn and say what the two conditions force, and where the point sits.

$\alpha_i = 0$ .

Solution

*Solution 3.23.*  $\alpha_i = 0$ . The second condition has  $C - \alpha_i = C \neq 0$ , so  $\xi_i = 0$ ; the first holds whatever the bracket is. With no slack, the original constraint  $y_{(i)} h(x_{(i)}) \geq 1$  applies, so the point sits on or outside the margin. It contributes no term to  $w = \sum_i \alpha_i y_{(i)} x_{(i)}$  and could be deleted without changing the fit. These are the **non-support vectors**, and they are the majority.

**Exercise 3.24** (Sort the training points by their  $\alpha_i$  (part 2)). Use the setup of Exercise 3.23.  $0 < \alpha_i < C$ .

Solution

*Solution 3.24.*  $0 < \alpha_i < C$ . Both factors outside the brackets are non-zero, so both brackets must vanish:  $\xi_i = 0$  from the second, and then  $y_{(i)} h(x_{(i)}) = 1$  from the first. The point sits exactly on the margin, unviolated, and carries weight. These are the **margin support vectors**.

**Exercise 3.25** (Sort the training points by their  $\alpha_i$  (part 3)). Use the setup of Exercise 3.23.  $\alpha_i = C$ . This case does not fix  $\xi_i$ ; say what values it allows and what each of them means<sup>a</sup>, including the boundary between a point that is still classified correctly and one that is not.

<sup>a</sup><https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Solution

*Solution 3.25.*  $\alpha_i = C$ . The second condition reads  $0 \cdot \xi_i = 0$  and so says nothing;  $\xi_i$  is free to be any non-negative number. The first gives  $y_{(i)} h(x_{(i)}) = 1 - \xi_i$ .

- $\xi_i = 0$  puts the point on the margin, like case 2, but paying the maximum cost.
- $0 < \xi_i < 1$  puts it inside the margin and still on the correct side of the boundary, since  $y_{(i)} h(x_{(i)}) > 0$ .
- $\xi_i = 1$  puts it exactly on the boundary, where  $h(x_{(i)}) = 0$  and the classifier has no answer.
- $\xi_i > 1$  puts it on the wrong side: the point is misclassified.

$\xi_i = 1$  is therefore the boundary between a violation that is merely inside the margin and one that is an error. The points with  $\xi_i > 0$  are the **non-margin support vectors**.

The vocabulary is worth stating carefully: the three names partition by  $\alpha_i$ , so a point with  $\alpha_i = C$  and  $\xi_i = 0$  lies on the margin while being a non-margin **support vector** by name. It is a boundary case rather than a contradiction, and it is the soft-margin version of the mismatch Solution 2.18 found between the two definitions of a support vector.

**Exercise 3.26** (Sort the training points by their  $\alpha_i$  (part 4)). Use the setup of Exercise 3.23. Proposition 2.1 found  $b$  from a single training point. Which of your three kinds can still be used for that, and why do the other two fail?

Solution

*Solution 3.26.* Only the margin **support vectors** of case 2. Proposition 2.1 gives  $b = y_{(i)} - w \cdot x_{(i)}$ , which solves  $y_{(i)}h(x_{(i)}) = 1$  for  $b$ , and case 2 is the only one that supplies that equation. Case 3 gives  $y_{(i)}h(x_{(i)}) = 1 - \xi_i$ , one equation in the two unknowns  $b$  and  $\xi_i$ , and  $\xi_i$  is not known until  $b$  is. Case 1 gives an inequality, which fixes nothing.

Both kinds of **support vector** contribute to  $w$ ; only one kind can give  $b$ .

**Exercise 3.27** (Sort the training points by their  $\alpha_i$  (part 5)). Use the setup of Exercise 3.23. Read  $\alpha_i$  as how hard  $x_{(i)}$  is pushing on the decision boundary, and describe each of the three kinds in those terms.

Solution

*Solution 3.27.*  $\alpha_i$  is the force  $x_{(i)}$  exerts on the boundary, capped at  $C$ .

- $\alpha_i = 0$ : the point exerts no force. It lies outside the margin and has no influence on the fit.
- $0 < \alpha_i < C$ : the point exerts force below the maximum and holds the margin at exactly the right distance.
- $\alpha_i = C$ : the point exerts the maximum force allowed, and that is not enough to keep it outside the margin. The margin moves past it, and other points with force left hold the margin in place.

**Exercise 3.28** (Measure the slack of one point that violates the margin). Work in one dimension again, as Exercise 1.13 did, and under the scale Theorem 1.2 fixed. Take  $h(x) = 2x - 6$ , so that  $w = 2$  and  $b = -6$ .

Where is the hyperplane, and where are the two margin boundaries, the places where  $h(x) = 1$  and  $h(x) = -1$ ?

Solution

*Solution 3.28.* The hyperplane is the root of  $h$ , at  $x = 3$ . The margin boundaries are at  $h(x) = 1$ , so  $x = 3.5$ , and at  $h(x) = -1$ , so  $x = 2.5$ .

**Exercise 3.29** (Measure the slack of one point that violates the margin (part 2)). Use the setup of Exercise 3.28.

What is the margin  $M$ ?

Solution

*Solution 3.29.* By Theorem 1.2,

$$\begin{aligned} M &= 1/|w| \\ &= 1/2, \end{aligned}$$

which is the distance from  $x = 3$  to either boundary.

**Exercise 3.30** (Measure the slack of one point that violates the margin (part 3)). Use the setup of Exercise 3.28.

A training point with  $y = +1$  sits at  $x = 3.4$ . Find the smallest  $\xi$  that satisfies its constraint in Definition 3.2.

Solution

*Solution 3.30.*

$$\begin{aligned} h(3.4) &= 2(3.4) - 6 \\ &= 0.8, \end{aligned}$$

and  $y = +1$ , so  $y h(x) = 0.8$ . The constraint  $y h(x) \geq 1 - \xi$  needs  $\xi \geq 0.2$ , so the smallest admissible slack is  $\xi = 0.2$ .

**Exercise 3.31** (Measure the slack of one point that violates the margin (part 4)). Use the setup of Exercise 3.28.

Is the point classified correctly?

Solution

*Solution 3.31.* The point is classified correctly:  $h(3.4) = 0.8 > 0$  puts it on the positive side. It has merely moved inside the margin.

**Exercise 3.32** (Measure the slack of one point that violates the margin (part 5)). Use the setup of Exercise 3.28.

Do the same for a point with  $y = +1$  at  $x = 2.8$ .

Solution

*Solution 3.32.*  $h(2.8) = -0.4$ , so  $y h(x) = -0.4$  and  $\xi = 1.4$ . This point is on the negative side of the hyperplane and carries a positive label, so it is misclassified.

**Exercise 3.33** (Measure the slack of one point that violates the margin (part 6)). Use the setup of Exercise 3.28.

In each of Exercise 3.29 and Exercise 3.30, measure the distance along the line from the point to the margin boundary it should have reached. Compare each distance with the slack you found, and state the relationship between the two.

Solution

*Solution 3.33.* In Solution 3.29 the point should have reached  $x = 3.5$  and stopped at  $x = 3.4$ , a shortfall of 0.1, against a slack of 0.2. In Solution 3.30 the shortfall is  $3.5 - 2.8 = 0.7$ , against a slack of 1.4. Both times the slack is the shortfall divided by  $M = 1/2$ :

$$\xi_i = \frac{\text{distance short of the margin boundary}}{M}.$$

Equivalently  $\xi_i = |w| \times \text{shortfall}$ , which is the same statement because Theorem 1.2 makes  $M$  and  $|w|$  reciprocal.

**Exercise 3.34** (Measure the slack of one point that violates the margin (part 7)). Use the setup of Exercise 3.28.

Your answer to Exercise 3.31 says what a slack variable is measured in. Use it to explain why  $\xi_i = 1$  is the value Solution 3.23 treats as special.

#### Solution

*Solution 3.34.* A slack variable is a distance measured in margins rather than in the units of  $x$ . So  $\xi_i = 1$  says that the point falls exactly one margin short of its margin boundary — and one margin back from the margin boundary is the hyperplane itself. So  $\xi_i = 1$  separates a point still on the right side from a misclassified one, and the threshold is 1 for every dataset and every scaling of the features.

#### i The $\nu$ -SVM, and why we might use $\nu$ instead of $C$

$C$  has one practical drawback: its useful range depends on the data. It is a cost in units of  $\frac{1}{2}\|w\|_2^2$  per unit of violation, so a rescaling of the features moves the range of sensible values, and there is no interval you can search that is right for every problem.

The  $\nu$ -SVM replaces  $C$  with a parameter  $\nu \in [0, 1]$  that has a direct meaning for the training set. At the optimum,  $\nu$  is

- an upper bound on the fraction of margin errors, the points with  $\xi_i > 0$ ;
- a lower bound on the fraction of support vectors.

So  $\nu = 0.1$  asks for at most 10% of the training points inside the margin and guarantees at least 10% of them are support vectors. The margin widens as  $\nu$  grows, which is the opposite direction from  $C$ . The two formulations fit the same family of classifiers;  $\nu$  only changes which one you ask for and how you ask. The reason to know both is that software offers both, and a value of  $\nu$  transfers between datasets in a way a value of  $C$  does not.

#### i Watch

- Hastie and Tibshirani formulate the soft margin, from 4:08<sup>a</sup> [10], video 9.2.
- Abu-Mostafa explains how SVMs handle non-separable data, from 45:58<sup>b</sup> [11], Lecture 15.
- Starmer explains support vector machines<sup>c</sup> [17], including the soft margin.

<sup>a</sup><https://www.youtube.com/watch?v=pjvnCEfAswc&t=248s>

<sup>b</sup><https://www.youtube.com/watch?v=XUj5JbQihIU&t=2758s>

<sup>c</sup><https://www.youtube.com/watch?v=efR1C6CvbmE>

#### i Read

GeeksforGeeks explains hard and soft margins<sup>a</sup> [18].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/using-a-hard-margin-vs-soft-margin-in-svm/>

## 4 In code

**Exercise 4.1** (Solve for the slack). The C-SVM primal problem of Definition 3.2 is a constrained problem, which looks like it needs a quadratic-programming solver. Choose values of the weights  $w$  and the intercept  $b$ . Which value of  $\xi_i$  satisfies both constraints on it and makes the objective as small as possible?

Solution

*Solution 4.1.* Both constraints bound  $\xi_i$  from below: the first by  $1 - y_{(i)}(w \cdot x_{(i)} + b)$ , and the second by 0. The objective grows with  $\xi_i$ , since  $C > 0$ , so the best choice is the larger of the two bounds:

$$\xi_i = \max(0, 1 - y_{(i)}(w \cdot x_{(i)} + b))$$

**Exercise 4.2** (Remove the slack from the problem). Make the assumptions of Exercise 4.1, and use the value of  $\xi_i$  found there. Substitute that value back, and write the **primal problem** as an unconstrained problem in  $w$  and  $b$  alone.

Solution

*Solution 4.2.* Substituting it for each  $\xi_i$  leaves

$$\arg \min_{w, b} \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \max(0, 1 - y_{(i)}(w \cdot x_{(i)} + b))$$

with no constraints left.

**Exercise 4.3** (Does the problem still need a quadratic-programming solver?). Make the assumptions of Exercise 4.1, and consider the unconstrained problem of Exercise 4.2. Does it still need a quadratic-programming solver?

Solution

*Solution 4.3.* No. An unconstrained convex<sup>a</sup> objective can be minimized by (sub)gradient descent (Gradient Descent), which is how the `svm_subgradient` function fits it.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

**Proposition 4.1** (The optimal slack is the hinge loss). Choose values of the weights  $w$  and the intercept  $b$  in Definition 3.2. The slack that satisfies both of its constraints and minimizes the objective is

$$\xi_i = \max(0, 1 - y_{(i)}(w \cdot x_{(i)} + b)).$$

Substituting it, the C-SVM primal problem is the unconstrained problem

$$\arg \min_{w, b} \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \max(0, 1 - y_{(i)}(w \cdot x_{(i)} + b)).$$

**i** Proof

*Proof.* This is the solution to Exercise 4.1 and Exercise 4.2. □

**Example 4.1** (The optimal slack of the class-mean classifier). Take the rescaled class-mean classifier  $(w, b)$  of Example 3.1 and  $C = 1$ . Evaluate the objective of Definition 3.2 twice: once with the slack  $\xi_i = \max(0, 1 - y_{(i)}h(x_{(i)}))$  of Proposition 4.1, and once with every slack increased by 0.1, which still satisfies both constraints. Then evaluate the unconstrained objective of Proposition 4.1:

## 4.1 Python

```
C_os = 1.0
margin_os = 0.5 * w_sl @ w_sl
obj_opt_os = margin_os + C_os * xi_sl.sum() # objective at the optimal slack
obj_pad_os = margin_os + C_os * (xi_sl + 0.1).sum() # a feasible but larger slack
obj_unc_os = margin_os + C_os * np.maximum(0, 1 - y_dr * (Z_dr @ w_sl + b_sl)).sum()
print(obj_opt_os, obj_pad_os, obj_unc_os)

144.08114431320323 183.2811443132032 144.08114431320323
```

## 4.2 R

```
C_os <- 1
margin_os <- 0.5 * sum(w_sl^2)
obj_opt_os <- margin_os + C_os * sum(xi_sl) # objective at the optimal slack
obj_pad_os <- margin_os + C_os * sum(xi_sl + 0.1) # a feasible but larger slack
obj_unc_os <- margin_os + C_os * sum(pmax(0, 1 - y_dr * (drop(Z_dr %*% w_sl) + b_sl)))
c(obj_opt_os, obj_pad_os, obj_unc_os)
```

```
[1] 144.0811 183.2811 144.0811
```

The optimal slack gives objective 144.08. Adding 0.1 to each of the 392 slacks raises it to 183.28, so any larger slack is worse, and the unconstrained objective, 144.08, equals the constrained one at the optimal slack.

**Definition 4.1** (Hinge loss). The **hinge loss** of a classifier  $h$  at a training point  $(x_{(i)}, y_{(i)})$ , with  $y_{(i)} \in \{-1, +1\}$ , is

$$\max(0, 1 - y_{(i)}h(x_{(i)})).$$

It is 0 for a point on the correct side of the margin, and grows linearly as the point moves past it [19]. It is the optimal slack  $\xi_i$  of Proposition 4.1, and logistic regression compared it with the logistic loss in an earlier exercise.

**Example 4.2** (Hinge loss on three cars). For the rescaled classifier  $h$  of Example 3.1, compute the hinge loss  $\max(0, 1 - y_{(i)}h(x_{(i)}))$  and the misclassification loss (1 if  $y_{(i)}h(x_{(i)}) < 0$ , else 0) of the first car in each of three groups: beyond its margin, inside its margin on the correct side, and misclassified. Then average each loss over all cars:

## 4.3 Python

```
yh_hl = y_dr * f_sl # y h(x): positive when correct, at least 1 beyond the m
hinge_hl = np.maximum(0, 1 - yh_hl)
zero_one_hl = (yh_hl < 0).astype(float)
first_hl = [np.flatnonzero(yh_hl >= 1)[0], np.flatnonzero((yh_hl >= 0) & (yh_hl < 1))[0], np.flatnonzero(yh_hl < 0)[0]]

print(pd.DataFrame({"name": auto_dr["name"], "y_h": yh_hl.round(2), "hinge": hinge_hl.round(2),
                    "misclassified": zero_one_hl}).iloc[first_hl])

name y_h hinge misclassified
```

```

1      buick skylark 320  1.66  0.00      0.0
0  chevrolet chevelle malibu  0.90  0.10      0.0
15      plymouth duster -0.29  1.29      1.0

```

```
print(hinge_hl.mean(), zero_one_hl.mean())
```

```
0.3662992006749351 0.13520408163265307
```

#### 4.4 R

```

yh_hl <- y_dr * f_sl # y h(x): positive when correct, at least 1 beyond the margin
hinge_hl <- pmax(0, 1 - yh_hl)
zero_one_hl <- as.numeric(yh_hl < 0)
first_hl <- c(which(yh_hl >= 1)[1], which(yh_hl >= 0 & yh_hl < 1)[1], which(yh_hl < 0)[1])

data.frame(name = auto_dr$name, y_h = round(yh_hl, 2), hinge = round(hinge_hl, 2),
           misclassified = zero_one_hl)[first_hl, ]

```

```
# A tibble: 3 x 4
```

|   | name                      | y_h   | hinge | misclassified |
|---|---------------------------|-------|-------|---------------|
|   | <chr>                     | <dbl> | <dbl> | <dbl>         |
| 1 | buick skylark 320         | 1.66  | 0     | 0             |
| 2 | chevrolet chevelle malibu | 0.9   | 0.1   | 0             |
| 3 | plymouth duster           | -0.29 | 1.29  | 1             |

```
c(mean_hinge = mean(hinge_hl), error_rate = mean(zero_one_hl))
```

```
mean_hinge error_rate
0.3662992 0.1352041
```

The first car costs nothing, since it is beyond its margin. The second is classified correctly but sits inside the margin, so the hinge loss charges it 0.10 where the misclassification loss charges 0. The third is misclassified, and the hinge loss of 1.29 exceeds the misclassification loss of 1. Averaged over all 392 cars, the hinge loss is 0.366 and the error rate is 0.135. The hinge loss is never below the misclassification loss, so it is an upper bound on the error rate, and it equals the slack  $\xi_i$  of Example 3.1.

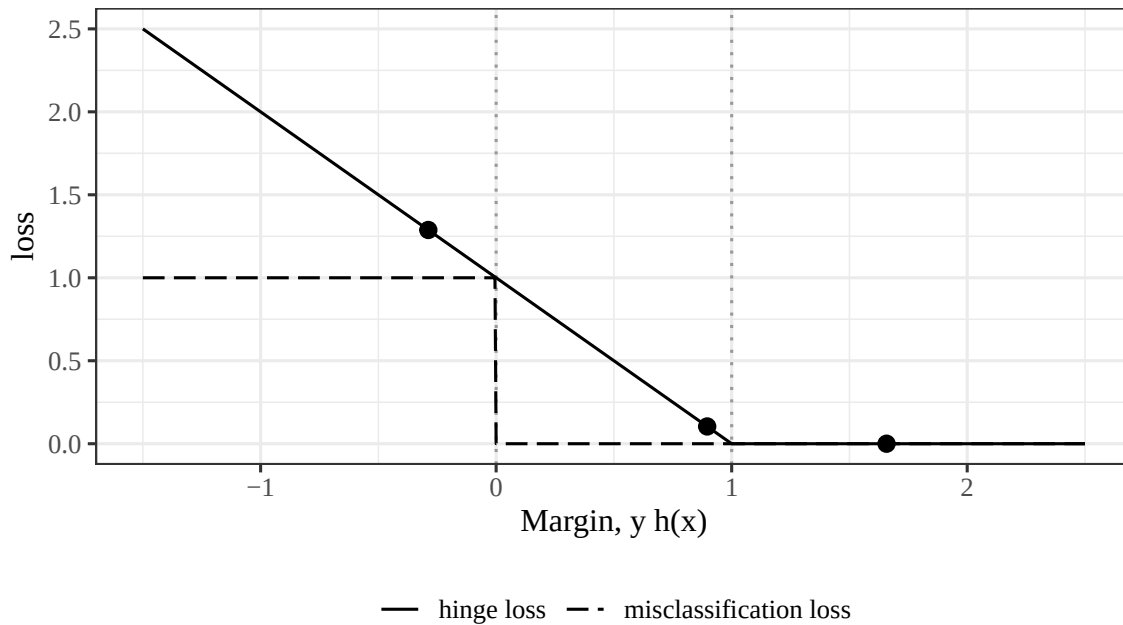


Figure 6: The hinge loss and the misclassification loss as functions of  $y h(x)$ . The dotted lines mark  $y h(x) = 0$ , the decision boundary, and  $y h(x) = 1$ , the margin. The three points are the three cars of Example 4.2: one beyond its margin, one inside it, and one misclassified.

#### **i** Read

GeeksforGeeks explains the hinge loss<sup>a</sup> [20].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/hinge-loss-relationship-with-support-vector-machines/>

**Exercise 4.4** (Step across the hinge’s corner). Proposition 4.1 turned Definition 3.2 into one unconstrained objective,

$$J(w, b) = \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \max(0, 1 - y_{(i)}(w \cdot x_{(i)} + b)),$$

which gradient descent could minimize but for one problem. Where, as a function of  $w$  and  $b$ , is  $J$  not differentiable?

#### Solution

*Solution 4.4.* Point  $i$ ’s hinge term has a corner where  $y_{(i)}(w^\top x_{(i)} + b) = 1$ , that is, where the point sits exactly on the margin. For each  $i$  that is one hyperplane in  $(w, b)$  space, and  $J$  is not differentiable on the union of those  $N$  hyperplanes.

**Exercise 4.5** (Step across the hinge’s corner (part 2)). Use the setup of Exercise 4.4. Away from those points, give the gradient of one point’s hinge term with respect to  $w$  and  $b$ , and then the gradient of  $J$ .

### Solution

*Solution 4.5.* Where  $y_{(i)}(w^\top x_{(i)} + b) > 1$  the term is 0, and so is its gradient. Where  $y_{(i)}(w^\top x_{(i)} + b) < 1$  the term is  $C(1 - y_{(i)}(w^\top x_{(i)} + b))$ , with gradient  $-Cy_{(i)}x_{(i)}$  in  $w$  and  $-Cy_{(i)}$  in  $b$ . Writing  $V$  for the set of points that violate the margin,

$$\nabla_w J = w - C \sum_{i \in V} y_{(i)} x_{(i)}, \quad \frac{\partial J}{\partial b} = -C \sum_{i \in V} y_{(i)}.$$

A subgradient method steps against these, recomputing  $V$  at every step.

**Exercise 4.6** (Step across the hinge’s corner (part 3)). Use the setup of Exercise 4.4. At a point where  $J$  is not differentiable, a *subgradient* method uses the derivative from either side. Explain why the choice makes no practical difference.

### Solution

*Solution 4.6.* Each iterate is a deterministic function of the data, so landing exactly on a corner means<sup>a</sup> the data satisfy one particular equation, which data drawn from a continuous distribution<sup>b</sup> do with probability zero. And should it happen, either one-sided derivative is a valid subgradient; treating a point exactly on the margin as inactive picks one of them.

The corners still matter near the optimum. A **support vector** that sits exactly on the margin has  $y_{(i)}h(x_{(i)}) = 1$ , so the optimum itself lies on the corner set, and the iterates keep crossing corners around it rather than landing on one. That crossing limits the step size.

<sup>a</sup><https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

<sup>b</sup><https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

**Proposition 4.2** (The gradient of the C-SVM objective away from its corners). *Let  $J(w, b)$  be the objective of Proposition 4.1, and let  $V = \{i : y_{(i)}(w \cdot x_{(i)} + b) < 1\}$  be the set of points that violate the margin. At every  $(w, b)$  with no training point exactly on the margin,  $J$  is differentiable, and*

$$\nabla_w J = w - C \sum_{i \in V} y_{(i)} x_{(i)}, \quad \frac{\partial J}{\partial b} = -C \sum_{i \in V} y_{(i)}.$$

### i Proof

*Proof.* This is the solution to Exercise 4.4 and Exercise 4.5. □

**Exercise 4.7** (Write the loop). Using the gradient of Proposition 4.2, write a function `svm_subgradient(X, y, C, h, steps)` that starts from  $w = 0$  and  $b = 0$  and takes `steps` subgradient steps of size  $\alpha$ , treating a point exactly on the margin as inactive. Then simulate two overlapping classes to try it on: 100 points labelled  $-1$  around  $(-1.5, 0)$  and 100 labelled  $+1$  around  $(1.5, 0)$ , each coordinate normal with standard deviation 1.

### Solution

*Solution 4.7.*

## 4.5 Python

```
import numpy as np

def svm_subgradient(X, y, C, h=1e-4, steps=20_000):
    w = np.zeros(X.shape[1])
    b = 0.0
    for _ in range(steps):
        offside = y * (X @ w + b) < 1      # where the hinge is active
        w = w - h * (w - C * (y[offside] @ X[offside]))
        b = b + h * C * y[offside].sum()
    return w, b

rng = np.random.default_rng(0)
n = 100
X = np.vstack([rng.normal([-1.5, 0], 1.0, (n, 2)),
               rng.normal([1.5, 0], 1.0, (n, 2))])
y = np.repeat([-1.0, 1.0], n)
```

## 4.6 R

```
svm_subgradient <- function(X, y, C, h = 1e-4, steps = 20000) {
  w <- rep(0, ncol(X))
  b <- 0
  for (step in seq_len(steps)) {
    offside <- y * drop(X %*% w + b) < 1      # where the hinge is active
    w <- w - h * (w - C * colSums(X[offside, , drop = FALSE] * y[offside]))
    b <- b + h * C * sum(y[offside])
  }
  list(w = w, b = b)
}

set.seed(0)
n <- 100
X <- rbind(cbind(rnorm(n, -1.5), rnorm(n, 0)),
           cbind(rnorm(n, 1.5), rnorm(n, 0)))
y <- rep(c(-1, 1), each = n)
```

## 4.7 Julia

```

using Random, LinearAlgebra

function svm_subgradient(X, y, C; h = 1e-4, steps = 20_000)
    w = zeros(size(X, 2))
    b = 0.0
    for _ in 1:steps
        offside = y .* (X * w .+ b) .< 1      # where the hinge is active
        w = w .- h .* (w .- C .* vec(sum(X[offside, :] .* y[offside], dims = 1)))
        b = b + h * C * sum(y[offside])
    end
    return w, b
end;

Random.seed!(0);
n = 100;
X = vcat([-1.5 0.0] .+ randn(n, 2), [1.5 0.0] .+ randn(n, 2));
y = [fill(-1.0, n); fill(1.0, n)];

```

**Exercise 4.8** (Predict what  $C$  does, then check). Run the `svm_subgradient` of Solution 4.7 at  $C = 0.01, 0.1$  and  $1$ . For each  $C$ , find the margin width  $2/\|w\|$ , the number of support vectors (points with  $y_{(i)}h(x_{(i)}) \leq 1$ ), and the number of misclassified points (points with  $y_{(i)}h(x_{(i)}) < 0$ ). Before you run it, predict which way each of the three numbers moves as  $C$  grows.

Solution

*Solution 4.8.*

## 4.8 R

```

# Each tab's svm_subgradient at the three values of C, keeping the margin,
# support-vector and misclassified counts for the table and the prose.
svm_r <- t(apply(c(0.01, 0.1, 1), function(cost) {
    fit <- svm_subgradient(X, y, cost)
    f <- y * drop(X %*% fit$w + fit$b)
    c(2 / sqrt(sum(fit$w^2)), sum(f <= 1 + 1e-6), sum(f < 0))
}))

```

## 4.9 Python

```

# Each tab's svm_subgradient at the three values of C, keeping the margin,
# support-vector and misclassified counts for the table and the prose.
svm_py = []
for C in (0.01, 0.1, 1.0):
    w, b = svm_subgradient(X, y, C)
    f = y * (X @ w + b)
    svm_py.append([2 / np.linalg.norm(w),
                    int(np.sum(f <= 1 + 1e-6)), int(np.sum(f < 0))])
svm_py = np.array(svm_py)

```

## 4.10 Julia

```
svm_jl = [begin
    w, b = svm_subgradient(X, y, C)
    f = y .* (X * w .+ b)
    [2 / norm(w), count(f .<= 1 + 1e-6), count(f .< 0)]
end for C in (0.01, 0.1, 1.0)];
svm_jl = permutedims(hcat(svm_jl...));

# One row per C (0.01, 0.1, 1); columns: margin, support vectors,
# misclassified.
svm_py <- py$svm_py
svm_jl <- JuliaCall::julia_eval("svm_jl")
svm_all <- list(Py = svm_py, R = svm_r, Jl = svm_jl)
# fmt2() comes from the page's setup chunk, which lintr cannot see from this
# file.
# nolint start: object_usage_linter.
margins <- function(i) {
    vals <- sapply(svm_all, function(m) fmt2(m[i, 1]))
    paste0("$", vals, "$", collapse = " / ")
}
# nolint end
svs <- function(i) {
    vals <- sapply(svm_all, function(m) m[i, 2])
    paste0("$", vals, "$", collapse = " / ")
}
misclassified <- function(m) paste0("$", m[, 3], "$", collapse = ", ")
sv_ratio <- sapply(svm_all, function(m) m[1, 2] / m[3, 2])
# The claims the prose makes in words.
for (m in svm_all) {
    stopifnot(
        all(diff(m[, 1]) < 0),           # margin falls as C grows
        all(diff(m[, 2]) < 0),           # so does the support-vector count
        diff(range(m[, 3])) <= 1         # misclassified wanders by one point
    )
}
stopifnot(
    all(sv_ratio > 2.5 & sv_ratio < 3.5), # roughly a factor of three
    # not even the same direction in all three
    length(unique(sapply(svm_all, function(m) sign(m[2, 3] - m[1, 3])))) > 1
)
```

The three languages draw their own data, so read the trends rather than the digits:

| $C$  | margin (Py / R / Jl) | support vectors |
|------|----------------------|-----------------|
| 0.01 | 2.90 / 3.01 / 3.00   | 101 / 94 / 100  |
| 0.1  | 1.50 / 1.68 / 1.56   | 50 / 49 / 53    |
| 1.0  | 1.06 / 0.99 / 1.11   | 32 / 35 / 36    |

As  $C$  grows, the margin narrows, the support-vector count falls, and the misclassified count barely moves. Raising  $C$  makes each unit of slack more expensive, so the optimizer uses less of it, and using less slack means a narrower margin. The hard-margin case is  $C \rightarrow \infty$ , where slack costs too much to use at all. A wide margin swallows more points, and every point inside the margin is a support vector, so the support-vector count follows the margin; this is the  $\nu$ -SVM's lower bound on the fraction of support vectors seen from the other direction. The misclassified count does not follow either, because  $C$  trades margin width against margin violations, and a point can sit inside the margin and still be on the correct side of the boundary.

**Exercise 4.9** (Predict what  $C$  does, then check (part 2)). Use the setup of Exercise 4.8. Which of those predictions does Exercise 3.15 already settle?

Solution

*Solution 4.9.* Only the margin. Theorem 3.1 proves that the margin never increases as  $C$  grows, which Solution 3.11 argued from which term dominates. Neither says anything about the support-vector or misclassified counts.

**Exercise 4.10** (Predict what  $C$  does, then check (part 3)). Use the setup of Exercise 4.8. Run it in one language, and compare.

Solution

*Solution 4.10.* The margin and the support-vector count move as predicted in all three languages, and the support-vector count falls by roughly a factor of three — 3.2 in Python, 2.7 in R and 2.8 in Julia. The misclassified count changes by a single point — 11, 12, 11 in Python, 11, 10, 11 in R, 17, 16, 16 in Julia — with no consistent direction, and not even the same direction in all three. These classes really do overlap, so there is a minimum error that no linear boundary can go below, and  $C$  does not trade accuracy.

## 4.11 On the step size

**Exercise 4.11** (Choose the step size). The `svm_subgradient` of Solution 4.7 uses the step size  $\alpha = 10^{-4}$ , a thousand times smaller than the 0.1 that Gradient Descent uses. The hinge terms are piecewise *linear* in  $w$  and  $b$ . Where  $J$  is differentiable, what is its Hessian<sup>a</sup> with respect to  $(w, b)$ ?

<sup>a</sup><https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

Solution

*Solution 4.11.*

```
# The step-size measurements, on the Python tab's data.
with np.errstate(over="ignore", invalid="ignore"):
    step_c1 = [np.linalg.norm(svm_subgradient(X, y, 1.0, h=h)[0])
               for h in (1e-4, 0.001, 0.01, 0.03, 0.1, 0.3, 2.1)]
    step_c100 = [np.linalg.norm(svm_subgradient(X, y, 100.0, h=h)[0])
                 for h in (1e-4, 0.01)]
    step_finite_19 = [bool(np.isfinite(np.linalg.norm(
        svm_subgradient(X, y, C, h=1.9)[0])))
                     for C in (0.01, 0.1, 1.0, 100.0)]
    step_finite_21 = [bool(np.isfinite(np.linalg.norm(
        svm_subgradient(X, y, C, h=2.1)[0])))
                     for C in (0.01, 0.1, 1.0, 100.0)]

step_c1 <- unlist(py$step_c1)
step_c100 <- unlist(py$step_c100)
stopifnot(
  all(unlist(py$step_finite_19)),      # below 2 it never overflows
  !any(unlist(py$step_finite_21)),    # at 2.1 it overflows for every C
  !is.finite(step_c1[7]),
  step_c100[2] / step_c100[1] > 2     # "times too large"
)
```

Away from the corners, each hinge term is linear in  $(w, b)$ , so it contributes no curvature at all. The only curvature comes from  $\frac{1}{2}\|w\|_2^2$ , so the Hessian is

$$\nabla^2 J = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix},$$

with the identity in the  $w$  block and 0 for  $b$ .

**Exercise 4.12** (Choose the step size (part 2)). Use the setup of Exercise 4.11. What is  $L$ , the largest eigenvalue of that Hessian<sup>a</sup>, as in Gradient Descent?

<sup>a</sup><https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

Solution

*Solution 4.12.* Its largest eigenvalue is  $L = 1$ .

**Exercise 4.13** (Choose the step size (part 3)). Use the setup of Exercise 4.11. What bound on  $\alpha$  would that give?

Solution

*Solution 4.13.* With  $L = 1$ , the quadratic bound  $\alpha < 2/L$  is  $\alpha < 2$ .

**Exercise 4.14** (Choose the step size (part 4)). Use the setup of Exercise 4.11. Would that bound depend on  $C$  or  $N$ ?

Solution

*Solution 4.14.* Neither  $C$  nor  $N$  appears in it. But  $\nabla_w J$  jumps at the corners, so  $J$  is not a quadratic and that result does not apply directly; Solution 4.16 shows that the bound holds anyway.

**Exercise 4.15** (Choose the step size (part 5)). Use the setup of Exercise 4.11. Write the  $w$  update as  $w_{k+1} = a w_k + r_k$ , where  $a$  is a number and  $r_k$  collects the hinge terms.

Solution

*Solution 4.15.* The update of Solution 4.7 is

$$\begin{aligned} w_{k+1} &= w_k - \alpha \left( w_k - C \sum_{i \in V_k} y_{(i)} x_{(i)} \right) \\ &= (1 - \alpha) w_k + \alpha C \sum_{i \in V_k} y_{(i)} x_{(i)}, \end{aligned}$$

where  $V_k$  is the set of points violating the margin at step  $k$ . So  $a = 1 - \alpha$  and  $r_k = \alpha C \sum_{i \in V_k} y_{(i)} x_{(i)}$ .

**Exercise 4.16** (Choose the step size (part 6)). Use the setup of Exercise 4.11. For which  $\alpha$  do the iterates stay bounded?

Solution

*Solution 4.16.* Whichever points violate the margin,  $\|r_k\|_2 \leq R = \alpha C \sum_{i=1}^N \|x_{(i)}\|_2$ . Starting from  $w_0 = 0$  and unrolling,

$$w_k = \sum_{j=0}^{k-1} a^{k-1-j} r_j, \quad \text{so} \quad \|w_k\|_2 \leq R \sum_{j=0}^{k-1} |a|^j \leq \frac{R}{1-|a|} \quad \text{when } |a| < 1.$$

So the iterates stay bounded when  $|1 - \alpha| < 1$ , that is, when  $0 < \alpha < 2$ .

**Exercise 4.17** (Choose the step size (part 7)). Use the setup of Exercise 4.11. Does your answer agree with Exercise 4.13 and Exercise 4.14?

Solution

*Solution 4.17.* The bound  $0 < \alpha < 2$  agrees with Solution 4.13.  $C$  and  $N$  appear only in  $R$ , never in  $a$ , so the threshold depends on neither. When  $|a| > 1$ , nothing damps the  $r_k$ : `svm_subgradient` overflows at  $\alpha = 2.1$  for every  $C$  tried, and at  $\alpha = 1.9$  it never does. The intercept<sup>a</sup> gets no such contraction, since its Hessian<sup>b</sup> entry is 0; what keeps  $b$  in check is the hinge, not curvature.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

<sup>b</sup><https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

**Exercise 4.18** (Choose the step size (part 8)). Use the setup of Exercise 4.11. In Gradient Descent, a step size past the bound made the iterates overflow. Say in advance what happens here at  $C = 1$  as  $\alpha$  grows from 0.001 toward the bound: will the returned  $\|w\|$  stay close to its converged value?

Solution

*Solution 4.18.* No. In Gradient Descent the iterates overflowed; here they mostly do not, but staying finite is not the same as being right, and the important failure happens well below  $\alpha = 2$ . As  $\alpha$  grows, the iterates start jumping across the corners, changing which points violate the margin from one step to the next, and they settle into an orbit around the optimum rather than at it. The answer that comes back is finite, believable, and wrong. Measured on the Python tab's data at  $C = 1$ , where the converged

$\|w\|$  is 1.88:

| $\alpha$ | 0.001 | 0.01 | 0.03 | 0.1  | 0.3  | 2.1      |
|----------|-------|------|------|------|------|----------|
| $\ w\ $  | 1.88  | 1.88 | 1.89 | 3.38 | 13.9 | overflow |

**Exercise 4.19** (Choose the step size (part 9)). Use the setup of Exercise 4.11. Should the range of usable step sizes grow or shrink as  $C$  grows?

Solution

*Solution 4.19.* It shrinks as  $C$  grows, since  $C$  scales  $r_k$  and so the size of each jump. At  $C = 100$ , `svm_subgradient` needs  $\alpha = 10^{-4}$  to get  $\|w\| = 2.09$ ; at  $\alpha = 0.01$  it returns 14.7, with nothing to tell you that is 7 times too large. The step size of  $10^{-4}$  avoids that silent-wrong-answer regime, rather than the overflow.

## 4.12 Soft-margin SVMs on real data

**Exercise 4.20** (Classify fuel economy with a soft margin). The Auto data of James et al. [5] give the fuel efficiency (mpg) of 392 cars. Download `auto.csv`<sup>a</sup> to load it in Python or R. Label a car +1 if its mpg is above the median mpg of the training cars and -1 otherwise. Use every third car (the 1st, 4th, 7th, and so on) as test data and the other cars as training data. Predict the label from `horsepower` and `weight`, each standardized with the mean and standard deviation of the training cars. Fit the `svm_subgradient` of Solution 4.7 to the training cars at  $C = 0.01, 0.1$  and 1. For each  $C$ , find the number of support vectors and the test misclassification rate.

<sup>a</sup><https://morrison-lab.github.io/lds/chapters/data/auto.csv>

Solution

*Solution 4.20.*

### 4.12.1 Python

```
import pandas as pd

auto_py = pd.read_csv("data/auto.csv").dropna()
is_test_py = (np.arange(len(auto_py)) % 3 == 0)
feat = auto_py[["horsepower", "weight"]].to_numpy(dtype=float)
mu, sd = feat[~is_test_py].mean(axis=0), feat[~is_test_py].std(axis=0, ddof=1)
Xa_py = (feat - mu) / sd
mpg = auto_py["mpg"].to_numpy()
ya_py = np.where(mpg > np.median(mpg[~is_test_py]), 1.0, -1.0)

auto_svm_py = []
for C in (0.01, 0.1, 1.0):
    w, b = svm_subgradient(Xa_py[~is_test_py], ya_py[~is_test_py], C)
    f_train = ya_py[~is_test_py] * (Xa_py[~is_test_py] @ w + b)
    f_test = ya_py[is_test_py] * (Xa_py[is_test_py] @ w + b)
    auto_svm_py.append([C, int(np.sum(f_train <= 1 + 1e-6)),
                        float(np.mean(f_test < 0))])
pd.DataFrame(auto_svm_py, columns=["C", "support_vectors", "test_error"])
```

|   | C    | support_vectors | test_error |
|---|------|-----------------|------------|
| 0 | 0.01 | 166             | 0.145038   |
| 1 | 0.10 | 105             | 0.122137   |
| 2 | 1.00 | 85              | 0.114504   |

#### 4.12.2 R

```
auto <- read.csv("data/auto.csv")
auto <- tidyr::drop_na(auto)
is_test <- (seq_len(nrow(auto)) - 1) %% 3 == 0
feat <- auto |>
  dplyr::select(horsepower, weight) |>
  as.matrix()
Xa <- scale(feat, center = colMeans(feat[!is_test, ]),
            scale = apply(feat[!is_test, ], 2, sd))
ya <- ifelse(auto$mpg > median(auto$mpg[!is_test]), 1, -1)

auto_fits <- lapply(c(0.01, 0.1, 1), function(cost) {
  svm_subgradient(Xa[!is_test, ], ya[!is_test], cost)
})
auto_svm <- t(sapply(auto_fits, function(fit) {
  f_train <- ya[!is_test] * drop(Xa[!is_test, ] %*% fit$w + fit$b)
  f_test <- ya[is_test] * drop(Xa[is_test, ] %*% fit$w + fit$b)
  c(support_vectors = sum(f_train <= 1 + 1e-6),
    test_error = mean(f_test < 0))
})))
tibble::tibble(C = c(0.01, 0.1, 1), tibble::as_tibble(auto_svm))

# A tibble: 3 x 3
      C support_vectors test_error
  <dbl>         <dbl>         <dbl>
1  0.01           166          0.145
2  0.1            105          0.122
3  1              85          0.115

# The claims the prose makes: both languages agree, and the support-vector
# count falls as C grows.
auto_py <- matrix(as.numeric(unlist(py$auto_svm_py)), ncol = 3, byrow = TRUE)
stopifnot(
  all(auto_py[, 2] == auto_svm[, 1]),
  all(abs(auto_py[, 3] - auto_svm[, 2]) < 1e-9),
  all(diff(auto_svm[, 1]) < 0)
)
```

Both languages read the same file and use the same standardization and the same deterministic algorithm, so they print the same table.

**Exercise 4.21** (Classify fuel economy with a soft margin (part 2)). Use the setup of Exercise 4.20. Plot the training cars with the boundary and margin for  $C = 1$ .

Solution

*Solution* 4.21. Figure 7 shows the training cars with the boundary and margin for  $C = 1$ .

```
library(ggplot2)
fit <- auto_fits[[3]]
svm_df <- tibble::tibble(
```

```

horsepower = Xa[!is_test, 1],
weight = Xa[!is_test, 2],
mpg_group = ifelse(ya[!is_test] > 0, "above median mpg", "below median mpg")
)
svm_lines <- tibble::tibble(
  intercept = as.numeric((c(-1, 0, 1) - fit$b) / fit$w[2]),
  slope = as.numeric(-fit$w[1] / fit$w[2]),
  kind = c("margin", "boundary", "margin")
)
# The fit works in z-space; the tick labels show horsepower and pounds.
hp_center <- attr(Xa, "scaled:center")["horsepower"]
hp_scale <- attr(Xa, "scaled:scale")["horsepower"]
wt_center <- attr(Xa, "scaled:center")["weight"]
wt_scale <- attr(Xa, "scaled:scale")["weight"]
hp_ticks <- pretty(svm_df$horsepower * hp_scale + hp_center)
wt_ticks <- pretty(svm_df$weight * wt_scale + wt_center)
ggplot(svm_df, aes(horsepower, weight)) +
  geom_point(aes(shape = mpg_group), size = 2) +
  geom_abline(aes(intercept = intercept, slope = slope, linetype = kind),
    data = svm_lines, show.legend = FALSE) +
  scale_shape_manual(
    values = c("above median mpg" = 1, "below median mpg" = 2)
  ) +
  scale_linetype_manual(values = c(boundary = "solid", margin = "dashed")) +
  scale_x_continuous(
    breaks = (hp_ticks - hp_center) / hp_scale, labels = hp_ticks
  ) +
  scale_y_continuous(
    breaks = (wt_ticks - wt_center) / wt_scale,
    labels = scales::label_comma()(wt_ticks)
  ) +
  coord_cartesian(
    xlim = range(svm_df$horsepower),
    ylim = range(svm_df$weight)
  ) +
  labs(x = "horsepower", y = "weight (lb)",
    shape = NULL) +
  theme(legend.position = "bottom")

```

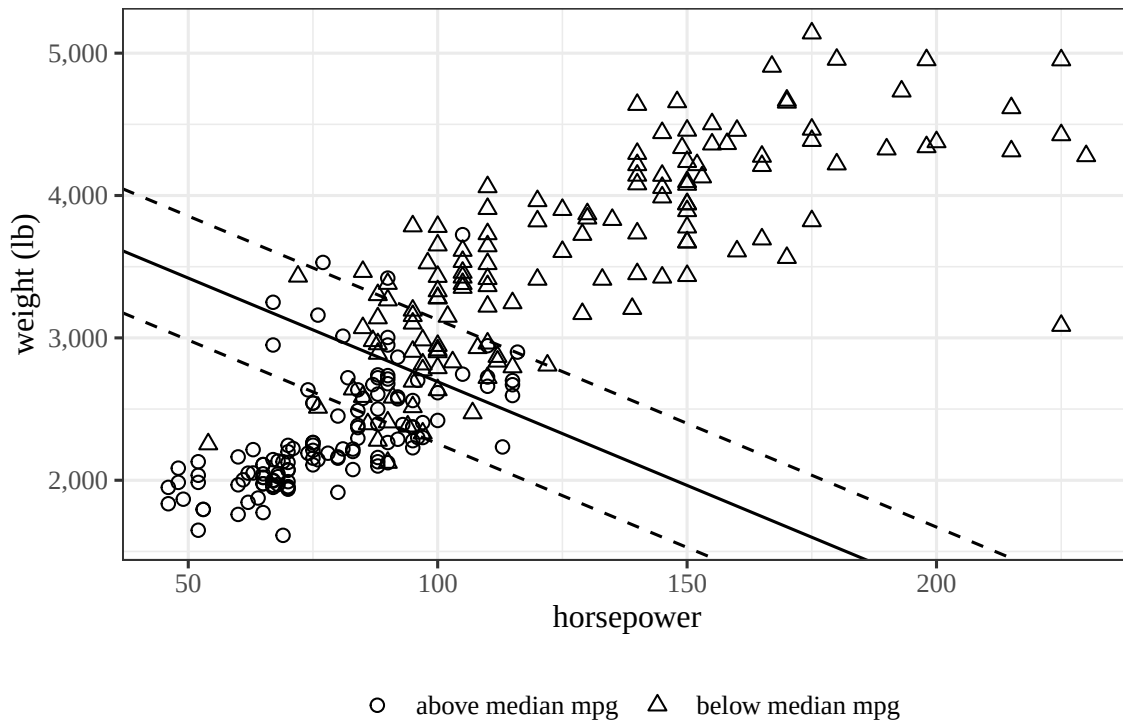


Figure 7: Training cars from the Auto data, by horsepower and weight (the SVM is fitted to the standardized values). Circles are cars with above-median mpg and triangles are the others. The solid line is the soft-margin SVM boundary at  $C = 1$ , and the dashed lines mark its margin.

**Exercise 4.22** (Classify fuel economy with a soft margin (part 3)). Use the setup of Exercise 4.20. Which way does the number of [support vectors](#) move as  $C$  grows?

Solution

*Solution 4.22.* The [support-vector](#) count falls from 166 at  $C = 0.01$  to 105 at  $C = 0.1$  and 85 at  $C = 1$ , as Solution 4.10 found for simulated data: a wider margin has more points inside it.

**Exercise 4.23** (Classify fuel economy with a soft margin (part 4)). Use the setup of Exercise 4.20. Does the test misclassification rate also move in one direction?

Solution

*Solution 4.23.* The test misclassification rate is 0.145, 0.122 and 0.115, which need not move in one direction, because  $C$  trades how wide the margin is against margin violations and not accuracy.

## 5 Non-linear SVMs: the kernel trick

### Watch

- Abu-Mostafa presents the kernel trick, from 15:53<sup>a</sup> [11], Lecture 15.
- Andrew Ng explains the kernel trick, from 28:56<sup>b</sup> [9], Lecture 7.
- Hastie and Tibshirani explain feature expansion and the SVM<sup>c</sup> [10], video 9.3.
- Their Python lab<sup>d</sup> fits SVMs in code, including a non-linear SVM with a radial kernel<sup>e</sup>.
- Starmer explains the polynomial kernel<sup>f</sup> [21] and the radial kernel<sup>g</sup> [22].

<sup>a</sup><https://www.youtube.com/watch?v=XUj5JbQihlU&t=953s>

<sup>b</sup><https://www.youtube.com/watch?v=8NYoQiRANpg&t=1736s>

<sup>c</sup><https://www.youtube.com/watch?v=02icdqOJsH4>

<sup>d</sup><https://www.youtube.com/watch?v=94wKFGS4Zm4>

<sup>e</sup><https://www.youtube.com/watch?v=94wKFGS4Zm4&t=690s>

<sup>f</sup><https://www.youtube.com/watch?v=Toet3EiSFcM>

<sup>g</sup>[https://www.youtube.com/watch?v=Qc5IyLW\\_hns](https://www.youtube.com/watch?v=Qc5IyLW_hns)

**Exercise 5.1** (Ask what a linear boundary cannot do). Every classifier built so far is a hyperplane in  $x$ . Linear regression's non-linear section had the same limit and removed it with a feature transformation  $D : \mathbb{R}^p \rightarrow \mathbb{R}^{p^*}$ , fitting a linear model in  $D(x)$  instead of in  $x$ . Sketch a training set in  $\mathbb{R}^2$  that no line separates, but that a circle does.

### Solution

*Solution 5.1.* A ring of positive points at radius 2 around the origin, and negative points scattered inside radius 1. No line separates near from far, but the circle  $\|x\|_2 = 1.5$  does (Figure 8).

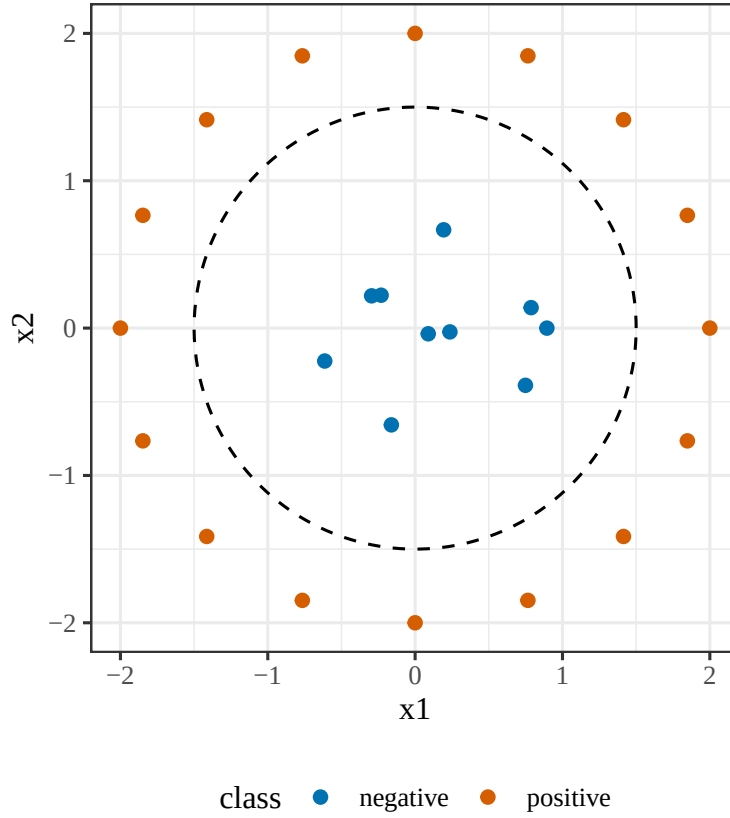


Figure 8: A ring of positive points around a cluster of negative points. No line separates the classes, but the dashed circle of radius 1.5 does.

**Exercise 5.2** (Ask what a linear boundary cannot do (part 2)). Use the setup of Exercise 5.1. Propose a transformation  $\tilde{D} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$  under which your two classes become linearly separable.

Solution

*Solution 5.2.*  $\tilde{D}(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$ . The third coordinate is  $\|x\|_2^2$ , which is small for the inner class and large for the ring, so a horizontal plane in the third coordinate now separates the two classes.

**Exercise 5.3** (Ask what a linear boundary cannot do (part 3)). Use the setup of Exercise 5.1. What has to change in Definition 3.3 to fit a hyperplane in  $\tilde{D}(x)$ -space instead of in  $x$ -space?

Solution

*Solution 5.3.* Every  $x_{(i)}$  is replaced by  $\tilde{D}(x_{(i)})$ , everywhere it appears:

$$\arg \max_{\alpha \in \mathbb{R}^N} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} \tilde{D}(x_{(i)})^\top \tilde{D}(x_{(j)}).$$

Nothing else in Definition 3.3 changes: the box constraint and the equality constraint are conditions on  $\alpha$ , and  $\alpha$  still has one entry per training point regardless of what space those points lie in.

**Exercise 5.4** (Find out what the dual actually costs). Solution 5.1’s dual needs  $\tilde{D}(x_{(i)})^\top \tilde{D}(x_{(j)})$  for every pair of training points. Take the polynomial<sup>a</sup> transformation of degree<sup>b</sup>  $d$  on  $p$  input<sup>c</sup> features, whose output dimension is  $p^* = O(p^d)$ .

Computing  $\tilde{D}(x)$  once costs  $O(p^*)$  work. Computing  $\tilde{D}(x_{(i)})^\top \tilde{D}(x_{(j)})$  from already-computed  $\tilde{D}(x_{(i)})$  and  $\tilde{D}(x_{(j)})$  costs another  $O(p^*)$ . Put the two together: what does computing one entry of the dual’s double sum cost?

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial>

<sup>b</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree>

<sup>c</sup><https://morrison-lab.github.io/mds/model-validation.html#def-data-set>

#### Solution

*Solution 5.4.*  $O(p^*)$  to compute  $\tilde{D}(x_{(i)})$ ,  $O(p^*)$  to compute  $\tilde{D}(x_{(j)})$ , and  $O(p^*)$  for the dot product<sup>a</sup> itself:  $O(p^*)$  in total, since three additive  $O(p^*)$  terms are still  $O(p^*)$ .

<sup>a</sup><https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-dot-product>

**Exercise 5.5** (Find out what the dual actually costs (part 2)). Use the setup of Exercise 5.4. MNIST digit images are  $28 \times 28$  pixels, so  $p = 784$ . Take  $d = 4$ . Using  $p^* = O(p^d)$ , is  $p^*$  in the thousands, the millions, or the billions?

#### Solution

*Solution 5.5.*  $p^* = O(784^4) \approx 3.8 \times 10^{11}$ : the billions. Every one of those  $\tilde{D}(x_{(i)})$  vectors would need to be stored, and 377 billion floating-point numbers is far past what a training run keeps for one data point.

**Exercise 5.6** (Find out what the dual actually costs (part 3)). Use the setup of Exercise 5.4. The dual objective needs this quantity for every pair  $(i, j)$  among  $N$  training points. Multiply Exercise 5.4’s per-pair cost by the number of pairs<sup>a</sup>, and say whether the bottleneck is  $\tilde{D}$  or the number of points.

<sup>a</sup><https://morrison-lab.github.io/mds/sets.html#def-ordered-pair>

#### Solution

*Solution 5.6.*  $O(N^2)$  pairs<sup>a</sup>, each costing  $O(p^*)$ , for  $O(N^2 p^*)$  total. With  $p^*$  in the billions and  $N$  in the tens of thousands, the  $p^*$  factor dominates completely: the bottleneck is computing  $\tilde{D}$  at all, not the number of points.

<sup>a</sup><https://morrison-lab.github.io/mds/sets.html#def-ordered-pair>

**Exercise 5.7** (Find out what the dual actually costs (part 4)). Use the setup of Exercise 5.4. Exercise 5.5 says explicit transformation is unworkable at that  $p^*$ . Name precisely which quantity the dual needs – not  $\tilde{D}(x_{(i)})$  and  $\tilde{D}(x_{(j)})$  separately, but the one number they combine into – since that is the quantity worth computing a shortcut for.

#### Solution

*Solution 5.7.* The inner product  $\tilde{D}(x_{(i)})^\top \tilde{D}(x_{(j)})$  itself, a single real number<sup>a</sup>. The dual objective never uses  $\tilde{D}(x_{(i)})$  or  $\tilde{D}(x_{(j)})$  on their own – only their inner product enters the sum – so a method that

produces that one number without first producing the two  $p^*$ -dimensional vectors would sidestep the cost in Solution 5.5 entirely.

<sup>a</sup><https://morrison-lab.github.io/mds/number-systems.html#def-real-numbers>

**Definition 5.1** (Kernel function). A **kernel function**  $K(x, z)$  for a map  $\tilde{D} : \mathbb{R}^p \rightarrow \mathbb{R}^{p^*}$  is a function computing

$$K(x, z) = \tilde{D}(x) \cdot \tilde{D}(z)$$

in time that does not scale with  $p^*$  [23].

**Example 5.1** (A quadratic kernel on two cars). For  $x, z \in \mathbb{R}^2$ , the function  $K(x, z) = (1 + x \cdot z)^2$  is a kernel for the map  $\tilde{D}(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$  into  $\mathbb{R}^6$ . Take the first two cars in the standardized Auto data [5] of Example 1.1, and compare  $K(x, z)$  with the inner product  $\tilde{D}(x) \cdot \tilde{D}(z)$  computed in the six-dimensional space:

## 5.1 Python

```
def phi_kf(v):
    # The explicit feature map: all monomials of degree at most 2, with the weights that make the kernel
    return np.array([1, np.sqrt(2) * v[0], np.sqrt(2) * v[1], v[0] ** 2, np.sqrt(2) * v[0] * v[1], v[1] ** 2])

x_kf, z_kf = Z_dr[0], Z_dr[1]
kernel_kf = (1 + x_kf @ z_kf) ** 2 # one inner product in 2 dimensions
inner_kf = phi_kf(x_kf) @ phi_kf(z_kf) # an inner product in 6 dimensions
print(kernel_kf, inner_kf)

6.579531810298261 6.5795318102982625
```

## 5.2 R

```
phi_kf <- function(v) {
  # The explicit feature map: all monomials of degree at most 2, with the weights that make the kernel
  c(1, sqrt(2) * v[1], sqrt(2) * v[2], v[1]^2, sqrt(2) * v[1] * v[2], v[2]^2)
}

x_kf <- Z_dr[1, ]
z_kf <- Z_dr[2, ]
kernel_kf <- (1 + sum(x_kf * z_kf))^2 # one inner product in 2 dimensions
inner_kf <- sum(phi_kf(x_kf) * phi_kf(z_kf)) # an inner product in 6 dimensions
c(kernel = kernel_kf, feature_space = inner_kf)

      kernel feature_space
6.579532      6.579532
```

Both routes give 6.5795, as Definition 5.1 requires. The kernel needs one inner product of the two original coordinates and a square, however many coordinates  $D$  has, while the explicit route builds 6 coordinates for each car. With  $p$  inputs<sup>a</sup> the degree-2 map has  $\binom{p+2}{2}$  coordinates, which grows like  $p^2/2$ , but the kernel still costs one inner product in  $p$  dimensions.

<sup>a</sup><https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

### Read

GeeksforGeeks explains the kernel trick<sup>a</sup> [24].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/kernel-trick-in-support-vector-classification/>

**Exercise 5.8** (Use a kernel without ever writing down  $D$ ). Suppose  $K$  is a [kernel function](#) for some map  $\tilde{D}$ , in the sense of Definition 5.1, and you are handed  $K$  but never told what  $\tilde{D}$  is. Definition 3.3 solves for  $\alpha$  using only inner products of training points. Rewrite it with  $K(x_{(i)}, x_{(j)})$  in place of  $x_{(i)} \cdot x_{(j)}$ .

### Solution

*Solution 5.8.*

$$\arg \max_{\alpha \in \mathbb{R}^N} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_{(i)} y_{(j)} K(x_{(i)}, x_{(j)}) \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^N \alpha_i y_{(i)} = 0.$$

**Exercise 5.9** (Use a kernel without ever writing down  $\tilde{D}$  (part 2)). Use the setup of Exercise 5.8. Does solving it require knowing  $\tilde{D}$ ?

### Solution

*Solution 5.9.* No. Every appearance of  $\tilde{D}$  in Solution 5.1's dual was inside an inner product, and  $K$  evaluates each of those directly. The optimization never asks for  $\tilde{D}(x_{(i)})$  on its own.

**Exercise 5.10** (Use a kernel without ever writing down  $\tilde{D}$  (part 3)). Use the setup of Exercise 5.8. Solution 5.4 found the dual's bottleneck is computing  $\tilde{D}$ , not the number of training points. Say what Exercise 5.8 has done to that bottleneck.

### Solution

*Solution 5.10.* It has removed it. Computing  $K(x_{(i)}, x_{(j)})$  costs whatever  $K$ 's own formula costs, with no dependence on  $p^*$ , so the  $O(p^*)$  step Solution 5.4 identified as the bottleneck is gone from the calculation entirely — not made cheaper, but skipped. The classifier can now work in a space too large to write a vector down in, because no vector in that space is ever computed.

**Exercise 5.11** (Use a kernel without ever writing down  $\tilde{D}$  (part 4)). Use the setup of Exercise 5.8. Not every function of two arguments is a kernel function for some  $\tilde{D}$ . Take  $K(x, z) = x \cdot z - 1$  on  $x, z \in \mathbb{R}$ , so that  $K(x, x) = x^2 - 1$ . Explain why no  $\tilde{D}$  can produce this  $K$ , using only the definition  $K(x, z) = \tilde{D}(x) \cdot \tilde{D}(z)$ .

### Solution

*Solution 5.11.* For any  $x$ ,  $K(x, x) = \tilde{D}(x) \cdot \tilde{D}(x) = \|\tilde{D}(x)\|_2^2$ , a squared length, which is never negative. Take  $x = 0$ :  $K(0, 0) = 0^2 - 1 = -1 < 0$ . No real vector has a negative squared length, so no  $\tilde{D}$  satisfies  $K(x, z) = \tilde{D}(x) \cdot \tilde{D}(z)$  for this  $K$ . Being a valid inner product is a real constraint on which functions qualify as [kernels](#), not a formality.

### i Three kernels to know by name

Definition 5.1 only requires  $K$  to compute some  $\tilde{D}$ 's inner product quickly. It says nothing about what  $\tilde{D}$  is, and in practice you pick  $K$  without ever naming  $\tilde{D}$  at all. Three choices cover most uses. **The linear kernel**,  $K(x, z) = x \cdot z$ , is  $\tilde{D} = \text{the identity}$ . Using it in Solution 5.8' dual reproduces Definition 3.3 exactly, and Definition 2.1 in the further limit  $C \rightarrow \infty$ , where the box constraint's upper bound stops binding: every SVM built so far is the special case of a kernel SVM where the kernel is linear.

**The polynomial kernel**,  $K(x, z) = (x \cdot z + 1)^d$ , corresponds to a  $\tilde{D}$  whose coordinates are every monomial in  $x$ 's entries up to degree  $d$ , the same family linear regression's non-linear section names.  $p^*$  for this  $\tilde{D}$  is exactly the count that exercise found, and  $K$  computes its inner product in  $O(p)$  time regardless.

**The RBF kernel** (also called the Gaussian kernel),

$$K(x, z) = \exp\left\{-\frac{\|x - z\|_2^2}{2\sigma^2}\right\},$$

corresponds to a  $\tilde{D}$  mapping into an infinite-dimensional space. There is no finite  $p^*$  to be intractable here: the inner product exists only through  $K$ , since no computer stores an infinite vector.  $\sigma$  is a hyperparameter, chosen the same way  $C$  is.

A useful, informal way to read the RBF kernel is as a similarity score:  $K(x, z)$  is largest, equal to 1, when  $x = z$ , and decreases as  $\|x - z\|_2$  grows. The linear and polynomial kernels do not read that way. The linear kernel gives  $K(x, x) = 1$  but  $K(x, (2, 0)) = 2$  for  $x = (1, 0)$ , so moving  $z$  further along  $x$ 's direction raises  $K$ , and the polynomial kernel grows the same way. The linear kernel measures how well  $x$  and  $z$  point the same way, scaled by their lengths:  $x \cdot z = \|x\|_2 \|z\|_2 \cos \theta$ , where  $\theta$  is the angle between them. The polynomial kernel measures the same thing for  $\tilde{D}(x)$  and  $\tilde{D}(z)$ . Dividing out the lengths of  $\tilde{D}(x)$  and  $\tilde{D}(z)$ ,

$$\frac{K(x, z)}{\sqrt{K(x, x) K(z, z)}},$$

gives a score between  $-1$  and  $1$  for any kernel whose  $K(x, x)$  and  $K(z, z)$  are positive, by the Cauchy-Schwarz inequality, since  $K(x, z) = \tilde{D}(x) \cdot \tilde{D}(z)$  (Definition 5.1). For the linear kernel that score is  $\cos \theta$ . The RBF kernel already has  $K(x, x) = 1$ , so dividing changes nothing, so it reads as a similarity score as it stands. Which functions can serve as a kernel at all is a large topic of its own, touched on in Exercise 5.11 and pursued no further here.

### i Read

GeeksforGeeks explains the common kernel functions<sup>a</sup> [25].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/major-kernel-functions-in-support-vector-machine-svm/>

**Exercise 5.12** (Compare points with the RBF kernel). Let  $K(x, z) = \exp\{-\|x - z\|_2^2/(2\sigma^2)\}$  be the RBF kernel with bandwidth  $\sigma = 1$ .

Which of the following is the smallest?

- (A)  $K((0, 0), (1, 2))$
- (B)  $K((0, 0), (-3, 0))$
- (C)  $K((2, 1), (5, 4))$

### Solution

*Solution 5.12.* The RBF kernel depends on its inputs only through the squared distance  $\|x - z\|_2^2$ , and it decreases as that distance grows. So the smallest value belongs to the pair that is furthest apart.

```
# The RBF kernel with bandwidth sigma = 1, written out from its definition.
rbf <- function(x, z, sigma = 1) {
  exp(-sum((x - z)^2) / (2 * sigma^2))
}

# One value per pair, in the order A, B, C, so we can compare them.
rbf_values <- c(
  A = rbf(c(0, 0), c(1, 2)),
  B = rbf(c(0, 0), c(-3, 0)),
  C = rbf(c(2, 1), c(5, 4))
)
rbf_values
```

```
      A      B      C
0.0820849986 0.0111089965 0.0001234098
```

The squared distances are 5, 9 and 18, so the values are 0.0821, 0.0111 and 0.000123. The smallest is (C), the pair at the greatest distance.

**Exercise 5.13** (Compare points with the RBF kernel (part 2)). Use the setup of Exercise 5.12. Does your answer to Exercise 5.12 change if  $\sigma$  changes?

Solution

*Solution 5.13.* No. For any  $\sigma > 0$ ,  $K$  is a decreasing function of  $\|x - z\|_2^2$ , so the ranking of the three values is the ranking of the distances. The bandwidth changes how fast  $K$  falls off with distance, not the order.

**Exercise 5.14** (Compare points with the RBF kernel (part 3)). Use the setup of Exercise 5.12. The RBF kernel's  $\tilde{D}$  maps into an infinite-dimensional space. Is that a recipe for overfitting<sup>a</sup>?

<sup>a</sup><https://morrison-lab.github.io/rme/chapters/predictor-selection.html#def-overfitting>

Solution

*Solution 5.14.* Not by itself.  $\tilde{D}$  places the points on a surface inside the large space, so the ambient dimension is huge but the points are not free to go anywhere in it. Computing  $\tilde{D}(x) \cdot \tilde{D}(z)$  costs only  $O(p)$  operations through  $K$  for the same reason: the data's intrinsic dimension is still  $p$ . The max-margin objective also penalizes  $\|w\|$ , which regularizes the fit, so a larger feature space does not by itself mean<sup>a</sup> a more flexible classifier. The bandwidth  $\sigma$  and the cost  $C$  are what control flexibility here, and they are chosen by validation, as in Solution 3.11.

<sup>a</sup><https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

**Exercise 5.15** (Classify a point when  $w$  cannot be written down). Solution 5.8 never computes  $\tilde{D}(x_{(i)})$ , so  $w = \sum_i \alpha_i y_{(i)} \tilde{D}(x_{(i)})$  from Proposition 2.1 is a sum of vectors you do not have, possibly infinite-dimensional ones under the RBF kernel. The trained classifier still has to predict a label for a new point  $x$ , which needs the sign of  $h(x) = w^\top \tilde{D}(x) + b$ . Substitute  $w$ 's definition into  $w^\top \tilde{D}(x)$  and distribute the sum over the inner product.

Solution

*Solution 5.15.*

$$w^\top \tilde{D}(x) = \sum_{i=1}^N \alpha_i y_{(i)} \tilde{D}(x_{(i)})^\top \tilde{D}(x).$$

**Exercise 5.16** (Classify a point when  $w$  cannot be written down (part 2)). Use the setup of Exercise 5.15. Rewrite the result using  $K$ .

Solution

*Solution 5.16.*

$$w^\top \tilde{D}(x) = \sum_{i=1}^N \alpha_i y_{(i)} K(x_{(i)}, x).$$

**Exercise 5.17** (Classify a point when  $w$  cannot be written down (part 3)). Use the setup of Exercise 5.15. Proposition 2.1 recovered  $b$  from a margin support vector  $x_{(k)}$  as  $b = y_{(k)} - w \cdot x_{(k)}$ , which solves  $y_{(k)} h(x_{(k)}) = 1$  for  $b$ . Do the same here: substitute Exercise 5.16's expression<sup>a</sup> for  $w^\top \tilde{D}(x_{(k)})$  and solve for  $b$ .

<sup>a</sup><https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Solution

*Solution 5.17.* Solution 5.16 with  $x = x_{(k)}$  gives  $w^\top \tilde{D}(x_{(k)}) = \sum_i \alpha_i y_{(i)} K(x_{(i)}, x_{(k)})$ . Substituting into  $y_{(k)} h(x_{(k)}) = 1$ , that is  $y_{(k)}(w^\top \tilde{D}(x_{(k)}) + b) = 1$ , and dividing by  $y_{(k)} \in \{-1, +1\}$ :

$$b = y_{(k)} - \sum_{i=1}^N \alpha_i y_{(i)} K(x_{(i)}, x_{(k)}),$$

for any margin support vector  $x_{(k)}$ , meaning  $0 < \alpha_k < C$ .

**Exercise 5.18** (Classify a point when  $w$  cannot be written down (part 4)). Use the setup of Exercise 5.15. Write the full decision rule – the sign that is actually computed to classify a new point – using only  $K$ , the training labels, the  $\alpha_i$ , and  $b$ . Confirm no  $\tilde{D}$  appears anywhere in it.

Solution

*Solution 5.18.* Classify  $x$  by the sign of

$$\sum_{i=1}^N \alpha_i y_{(i)} K(x_{(i)}, x) + b.$$

Only  $K$ , the  $y_{(i)}$ , the  $\alpha_i$ , and  $b$  appear.  $\tilde{D}$  was used to derive the rule and never appears in it.

**Exercise 5.19** (Classify a point when  $w$  cannot be written down (part 5)). Use the setup of Exercise 5.15. At training time the number of parameters was fixed by  $p$  (for  $w$ ) or by  $N$  (for  $\alpha$ ). Exercise 5.18's rule needs  $K(x, x_{(i)})$  for training points  $x_{(i)}$ . What has to be kept around after training to make a prediction, that an ordinary parametric model would not need?

### Solution

*Solution 5.19.* The training points themselves, or at least the ones with  $\alpha_i \neq 0$  – the **support vectors** – together with their labels and  $\alpha_i$  values. An ordinary parametric model, such as Definition 3.2’s own  $w$  and  $b$ , is a fixed-size summary of the training data that can be kept without keeping the data. A **kernel SVM**’s decision rule instead recomputes  $K$  against stored training points every time it predicts, which makes it a **non-parametric** model (Kinds of Learning Problems and Features): the number of things that must be stored, and the cost of a single prediction, both grow with  $N$  rather than staying fixed.

**Theorem 5.1** (The decision rule of a kernel SVM). *Let  $K$  be a **kernel** for the map  $D$  as in Definition 5.1, let  $\alpha$  solve the  $C$ -SVM dual with  $K$  in place of  $x_{(i)} \cdot x_{(j)}$ , and let  $w = \sum_{i=1}^N \alpha_i y_{(i)} \tilde{D}(x_{(i)})$ . Then, for every  $x$ ,*

$$w \cdot \tilde{D}(x) + b = \sum_{i=1}^N \alpha_i y_{(i)} K(x_{(i)}, x) + b,$$

where, for any  $k$  with  $0 < \alpha_k < C$ ,

$$b = y_{(k)} - \sum_{i=1}^N \alpha_i y_{(i)} K(x_{(i)}, x_{(k)}).$$

The rule classifies  $x$  by the sign of the left side. It uses  $K$ , the labels, the coefficients  $\alpha_i$  and  $b$ , but never  $\tilde{D}$  itself, and only the training points with  $\alpha_i \neq 0$  enter it.

### i Proof

*Proof.* This is the solution to Exercise 5.16. □

**Exercise 5.20** (Decide which problem to hand the solver). Definition 3.2 has  $p + 1 + N$  unknowns. Definition 3.3 has  $N$ . Neither is uniformly cheaper: which is better depends on  $p$  against  $N$ . When  $N \gg p$ , which problem has fewer unknowns?

### Solution

*Solution 5.20.* The **primal problem**, with its  $p + 1$  non-slack unknowns against the dual’s  $N$ .

**Exercise 5.21** (Decide which problem to hand the solver (part 2)). Use the setup of Exercise 5.20. Is the **kernel** trick useful in that regime?

### Solution

*Solution 5.21.* The **kernel** trick gives no benefit here: its entire point is avoiding a large  $p^*$ , and when  $N \gg p$  there is no large  $p^*$  to avoid, so the primal is both cheaper and, being linear in the original features, simpler to fit and to interpret.

**Exercise 5.22** (Decide which problem to hand the solver (part 3)). Use the setup of Exercise 5.20. When a **kernel** maps into a  $p^*$  with  $p^* \gg N$  – the RBF kernel’s infinite-dimensional case included – which problem is even expressible without  $\tilde{D}$ ?

### Solution

*Solution 5.22.* Only the dual. The primal's unknown is  $w \in \mathbb{R}^p$ , which under Definition 5.1's whole premise is never written down – under the RBF [kernel](#) it is not even finite-dimensional. The dual's unknown is  $\alpha \in \mathbb{R}^N$ , which exists and is finite whatever  $\tilde{D}$  is, because Solution 5.9 showed the optimization only ever touches  $\tilde{D}$  through  $K$ .

**Exercise 5.23** (Decide which problem to hand the solver (part 4)). Use the setup of Exercise 5.20. Theorem 5.1 found that a [kernel](#) SVM has to keep training points at prediction time. Does the primal, fit without a kernel, share that cost?

### Solution

*Solution 5.23.* No. A primal fit without a [kernel](#) returns an explicit  $w \in \mathbb{R}^p$  and  $b$ , a fixed-size summary exactly like Definition 3.2's own solution, so prediction is one inner product with  $w$  and needs no training data afterward. The storage cost Theorem 5.1 found is specifically the cost of working through  $K$  rather than through an explicit  $w$ .

**Exercise 5.24** (Decide which problem to hand the solver (part 5)). Use the setup of Exercise 5.20. Solution 5.4 found the dual's per-pair cost scales with  $p^*$  for an explicit  $\tilde{D}$ , and  $K$  removes that dependence. What does the dual's cost still scale with, that the primal's does not?

### Solution

*Solution 5.24.*  $N$ . Both terms of the dual's objective are sums over training points – the linear term over  $N$  points once, the quadratic term over  $N \times N$  pairs<sup>a</sup> – so evaluating it costs  $O(N^2)$  [kernel](#) evaluations regardless of how cheap each one is, while the primal's cost per iteration scales with  $p$  and  $N$  separately rather than with their product. A large training set makes the dual expensive precisely where the kernel trick made a large  $p^*$  cost nothing.

<sup>a</sup><https://morrison-lab.github.io/mds/sets.html#def-ordered-pair>

### **i** When to use which

Exercise 5.20 works out the two directions in full; this collects the conclusions.

Prefer the primal when  $N \gg p^*$ : it scales with  $N$  once per term rather than once per pair, it returns an explicit  $w$  that needs no training data to use afterward, and it has no access to kernels at all, since a kernel's whole benefit is a  $\tilde{D}$ -space too large to fit a primal in.

Prefer the dual when  $p^* \gg N$ : it is the only form the kernel trick applies to, since only the dual touches  $\tilde{D}$  solely through inner products, it scales with  $p^*$  not at all, and it produces the  $\alpha_i$ , which Solution 3.23 read off to sort the training points into support vectors and non-support vectors – a distinction  $w$  alone does not show. Its cost is the  $O(N^2)$  term Solution 5.20 found, and the non-parametric storage cost of Theorem 5.1.

## 6 Multiclass SVMs

**Exercise 6.1** (Notice what every SVM built so far assumes). Every problem in this chapter has used  $y_{(i)} \in \{-1, +1\}$ .

Point to the exact place that assumption enters Definition 3.2, and say what breaks if a training set instead has  $y_{(i)} \in \{1, 2, 3\}$ .

### Solution

*Solution 6.1.* In every constraint,  $y_{(i)}(w \cdot x_{(i)} + b) \geq 1 - \xi_i$ . The product  $y_{(i)}h(x_{(i)})$  is read as a signed margin, positive when  $h$  and  $y_{(i)}$  agree in sign, and that reading needs  $y_{(i)} = \pm 1$ . With  $y_{(i)} \in \{1, 2, 3\}$  the same product is not a margin at all — multiplying  $h(x_{(i)})$  by 2 or 3 changes its scale for no geometric reason, and the constraint no longer says anything about which side of the hyperplane  $x_{(i)}$  falls on.

**Exercise 6.2** (Notice what every SVM built so far assumes (part 2)). Use the setup of Exercise 6.1. Propose the simplest possible fix: pick two of the three classes, discard every training point belonging to the third, and fit an ordinary binary SVM. What is wrong with this as a solution to the three-class problem?

### Solution

*Solution 6.2.* It discards data, and it only ever answers a question between two of the three classes. Presented with a new point, a classifier fit on classes 1 and 2 has no way to say “this is class 3” — it was never shown a class 3 example and has no notion that one exists. Generalizing this idea to every pair of classes is a real strategy, one-versus-one, but picking one pair and discarding the rest is not. One-versus-one is defined later in this section, in Definition 6.2.

**Exercise 6.3** (Notice what every SVM built so far assumes (part 3)). Use the setup of Exercise 6.1. A less wasteful idea keeps every training point in every fit, by changing how points get their  $\pm 1$  labels rather than which points are used. Propose one.

### Solution

*Solution 6.3.* Relabel the training set once per class: to fit a classifier for class  $m$ , set  $y_{(i)}^{(m)} = +1$  for every point actually labelled  $m$ , and  $y_{(i)}^{(m)} = -1$  for every other point, whatever its true class. Every training point takes part in every fit, just with a different label each time.

**Definition 6.1** (One-versus-rest). **One-versus-rest** classification fits one binary SVM per class,  $m = 1, \dots, C$ , on the relabelling  $y_{(i)}^{(m)} = +1$  if  $y_{(i)} = m$  and  $-1$  otherwise, and classifies a new point  $x$  by

$$\arg \max_m h^{(m)}(x) = \arg \max_m \sum_{i=1}^N \alpha_i^{(m)} y_{(i)}^{(m)} K(x, x_{(i)}) + b^{(m)}.$$

**Example 6.1** (One-versus-rest on the three origins of the cars). The Auto data [5] record where each car was made: 1 American, 2 European, 3 Japanese. Standardize `weight`, `horsepower`, `acceleration` and `year`, and use a linear kernel. Fit one SVM per origin with the `dual_svm` helper of Example 1.3 at  $C = 1$ , relabelling the cars  $+1$  for that origin and  $-1$  for the rest, classify each car by the largest score  $h^{(m)}$ , and tabulate the result against the true origin (these are training cars, so the table describes the fit, not performance on new data):

## 6.1 Python

```

feat_ov = auto_dr[["weight", "horsepower", "acceleration", "year"]].to_numpy(float)
feat_ov = (feat_ov - feat_ov.mean(axis=0)) / feat_ov.std(axis=0, ddof=1)
K_ov = feat_ov @ feat_ov.T # linear kernel
org_ov = auto_dr["origin"].to_numpy()

score_ov = np.zeros((len(org_ov), 3))
for m in (1, 2, 3):
    y_m = np.where(org_ov == m, 1.0, -1.0) # origin m versus the rest
    alpha_m, b_m = dual_svm(K_ov, y_m, 1.0, tol=1e-8)
    score_ov[:, m - 1] = K_ov @ (alpha_m * y_m) + b_m #  $h^{(m)}(x) = \sum_i \alpha_i y_i K(x, x_i) + b$ 
pred_ov = score_ov.argmax(axis=1) + 1 # the class with the largest score
print(pd.crosstab(org_ov, pred_ov, rownames=["origin"], colnames=["predicted"]))

```

|        | predicted | 1   | 2 | 3  |
|--------|-----------|-----|---|----|
| origin |           |     |   |    |
| 1      |           | 235 | 0 | 10 |
| 2      |           | 43  | 3 | 22 |
| 3      |           | 40  | 0 | 39 |

## 6.2 R

```

feat_ov <- scale(as.matrix(auto_dr[, c("weight", "horsepower", "acceleration", "year")]))
K_ov <- feat_ov %*% t(feat_ov) # linear kernel
org_ov <- auto_dr$origin

score_ov <- sapply(1:3, function(m) {
  y_m <- ifelse(org_ov == m, 1, -1) # origin m versus the rest
  fit_m <- dual_svm(K_ov, y_m, 1, tol = 1e-8)
  drop(K_ov %*% (fit_m$alpha * y_m)) + fit_m$b #  $h^{(m)}(x) = \sum_i \alpha_i y_i K(x, x_i) + b$ 
})
pred_ov <- max.col(score_ov, ties.method = "first") # the class with the largest score
table(origin = org_ov, predicted = factor(pred_ov, levels = 1:3))

```

|        | predicted | 1 | 2  | 3 |
|--------|-----------|---|----|---|
| origin |           |   |    |   |
| 1      | 235       | 0 | 10 |   |
| 2      | 43        | 3 | 22 |   |
| 3      | 40        | 0 | 39 |   |

One-versus-rest, Definition 6.1, fit 3 SVMs, each on all 392 cars. It labels 70.7% of the cars with their true origin. The European cars are the hardest: only 3 of the 68 are recovered, against 235 of 245 American and 39 of 79 Japanese cars.

### i Read

GeeksforGeeks explains one-versus-rest classification<sup>a</sup> [26].

<sup>a</sup><https://www.geeksforgeeks.org/machine-learning/one-vs-rest-strategy-for-multi-class-classification/>

**Exercise 6.4** (Question the argmax rule). Definition 6.1 classifies  $x$  by which of the  $C$  classifiers gives  $h^{(m)}(x)$  the largest value, not by which one returns +1. Suppose two classifiers,  $h^{(1)}$  and  $h^{(2)}$ , both return a positive value at some point  $x$ :  $x$  looks like class 1 to the first fit and like class 2 to the second. What would picking “the first classifier that says +1” do at such a point, and why is the argmax rule better?

### Solution

*Solution 6.4.* “First classifier that says +1” would answer class 1 or class 2 depending only on which one happens to be checked first — an arbitrary tie-break presented as a decision. The argmax rule instead asks each classifier not just for a sign but for a signed distance from its own hyperplane, and picks the class whose fit is most confident. That uses information the plain sign discards.

**Exercise 6.5** (Question the argmax rule (part 2)). Use the setup of Exercise 6.4. Is it possible for *every*  $h^{(m)}(x)$  to be negative at some point  $x$ ?

### Solution

*Solution 6.5.* Yes:  $x$  can sit on the wrong side of every one of the  $C$  hyperplanes, if it is genuinely hard to place.

**Exercise 6.6** (Question the argmax rule (part 3)). Use the setup of Exercise 6.4. If so, what does the argmax rule still return there?

### Solution

*Solution 6.6.* The rule still returns whichever  $m$  is *least negative* — the classifier that was closest to calling  $x$  its own class, even though none of them did. The rule always returns some class; it never refuses to answer.

**Exercise 6.7** (Question the argmax rule (part 4)). Use the setup of Exercise 6.4.  $h^{(1)}$  and  $h^{(2)}$  come from two entirely separate fits, each with its own  $\alpha^{(m)}$ ,  $b^{(m)}$ , and even its own support vectors. Nothing in Definition 3.3 ties their scales together. Explain why comparing  $h^{(1)}(x)$  against  $h^{(2)}(x)$  by size is therefore a rough rule, not a guarantee, and name the training-set imbalance Definition 6.1’s relabelling introduces that could make the rough rule worse: for  $C = 10$  balanced classes, what fraction of the training points does each fit label  $-1$ ?

### Solution

*Solution 6.7.* Each  $h^{(m)}$  is the solution of an independent<sup>a</sup> optimization problem, so nothing forces  $\|w^{(m)}\|_2$  to be the same across  $m$ , and  $h^{(m)}(x)$ ’s scale depends on it: two classifiers can each be equally confident and still report different-sized numbers, or one can report a large number while being the less trustworthy fit. Picking the largest raw value therefore compares numbers that were never built to be compared, so it is a rough rule that tends to work rather than a rule that is provably correct. With  $C = 10$  balanced classes, each fit labels 1/10 of the training set +1 and the other 9/10 of it  $-1$ . That imbalance grows with  $C$ : at  $C = 100$  each fit sees a training set that is 99% negative. An SVM fit on such a skew has little reason to place its margin generously toward the minority class, since almost every point it is penalized for misclassifying is already on the  $-1$  side — which is the concrete mechanism behind the standard complaint that one-versus-rest is asymmetric.

<sup>a</sup><https://morrison-lab.github.io/pds/independence.html#def-indpt>

**Definition 6.2** (One-versus-one). **One-versus-one** classification fits one binary SVM for every pair of classes,  $\binom{C}{2} = C(C-1)/2$  of them, each trained on only the points belonging to that pair. A new point is classified by evaluating all  $\binom{C}{2}$  classifiers and taking a majority vote among their predictions [27].

**Example 6.2** (One-versus-one on the three origins of the cars). Repeat Example 6.1 with one-versus-one classification. Fit an SVM for each of the  $\binom{3}{2} = 3$  pairs of origins, each on only the cars of that pair, let every SVM cast a vote for one of its two origins on every car, and give each car the origin with the most votes (a three-way tie, one vote each, goes to the lowest origin):

### 6.3 Python

```

votes_ov1 = np.zeros((len(org_ov), 3))
for p, q in ((1, 2), (1, 3), (2, 3)):
    idx = np.flatnonzero((org_ov == p) | (org_ov == q)) # only the cars of this pair
    y_pq = np.where(org_ov[idx] == p, 1.0, -1.0)
    alpha_pq, b_pq = dual_svm(K_ov[np.ix_(idx, idx)], y_pq, 1.0, tol=1e-8)
    s_pq = K_ov[:, idx] @ (alpha_pq * y_pq) + b_pq # score of every car, not only the pair's
    votes_ov1[s_pq > 0, p - 1] += 1
    votes_ov1[s_pq <= 0, q - 1] += 1
pred_ov1 = votes_ov1.argmax(axis=1) + 1 # most votes; ties go to the lowest origin
print(pd.crosstab(org_ov, pred_ov1, rownames=["origin"], colnames=["predicted"]))

```

|        | predicted | 1   | 2  | 3  |
|--------|-----------|-----|----|----|
| origin |           |     |    |    |
| 1      |           | 211 | 7  | 27 |
| 2      |           | 31  | 15 | 22 |
| 3      |           | 23  | 3  | 53 |

### 6.4 R

```

votes_ov1 <- matrix(0, nrow(K_ov), 3)
for (pair in list(c(1, 2), c(1, 3), c(2, 3))) {
    idx <- which(org_ov %in% pair) # only the cars of this pair
    y_pq <- ifelse(org_ov[idx] == pair[1], 1, -1)
    fit_pq <- dual_svm(K_ov[idx, idx], y_pq, 1, tol = 1e-8)
    s_pq <- drop(K_ov[, idx] %*% (fit_pq$alpha * y_pq)) + fit_pq$b # score of every car, not only the
    votes_ov1[cbind(seq_along(s_pq), ifelse(s_pq > 0, pair[1], pair[2]))] <-
        votes_ov1[cbind(seq_along(s_pq), ifelse(s_pq > 0, pair[1], pair[2]))] + 1
}
pred_ov1 <- max.col(votes_ov1, ties.method = "first") # most votes; ties go to the lowest origin
table(origin = org_ov, predicted = factor(pred_ov1, levels = 1:3))

```

|        | predicted | 1   | 2  | 3  |
|--------|-----------|-----|----|----|
| origin |           |     |    |    |
| 1      |           | 211 | 7  | 27 |
| 2      |           | 31  | 15 | 22 |
| 3      |           | 23  | 3  | 53 |

One-versus-one, Definition 6.2, trained 3 SVMs on 313, 324 and 147 cars, fewer than the 392 that each one-versus-rest SVM used. It labels 71.2% of the cars with their true origin, against 70.7% for one-versus-rest. The vote was tied three ways for 9 cars.

**Exercise 6.8** (Compare the two strategies' costs). Definition 6.1 fits  $C$  classifiers. Definition 6.2 fits  $C(C-1)/2$ . For  $C = 3$ , how many classifiers does each strategy fit?

Solution

*Solution 6.8.* At  $C = 3$ : one-versus-rest fits 3, one-versus-one fits  $\binom{3}{2} = 3$  — equal.

**Exercise 6.9** (Compare the two strategies' costs (part 2)). Use the setup of Exercise 6.8. How many classifiers does each strategy fit for  $C = 8$ ?

Solution

*Solution 6.9.* At  $C = 8$ : one-versus-rest fits 8, one-versus-one fits  $\binom{8}{2} = 28$ . The two agree only at  $C = 3$ ; one-versus-one grows much faster afterward.

**Exercise 6.10** (Compare the two strategies' costs (part 3)). Use the setup of Exercise 6.8. Definition 6.2's classifiers each train on the points belonging to only two classes, while Definition 6.1's train on the whole dataset every time. Assuming the  $N$  training points are spread evenly across the  $C$  classes, how many points does one one-versus-one fit see?

Solution

*Solution 6.10.*  $2N/C$ : each classifier trains on the points from exactly two of the  $C$  classes, and each class contributes  $N/C$  points on average.

**Exercise 6.11** (Compare the two strategies' costs (part 4)). Use the setup of Exercise 6.8. Exercise 5.4 found the dual's cost scales like  $O(N^2)$  in the number of training points it is fit on. Using Exercise 6.9's answer, and treating  $C$  as fixed while  $N$  grows, compare how the *total* cost across all classifiers scales with  $N$  for each strategy.

Solution

*Solution 6.11.* One-versus-rest fits  $C$  classifiers, each on all  $N$  points, for a total cost of  $C \cdot O(N^2) = O(N^2)$ , since  $C$  is fixed.

One-versus-one fits  $\binom{C}{2}$  classifiers, each on  $2N/C$  points, for a total cost of

$$\begin{aligned} \binom{C}{2} \cdot O((2N/C)^2) &= O(C(C-1) \cdot N^2/C^2) \\ &= O(N^2) \end{aligned}$$

as well, since  $C$  is again fixed and  $(C-1)/C \rightarrow 1$  contributes no growth in  $N$ .

Both scale the same way in  $N$  for fixed  $C$ : the  $O(N^2)$  per-fit cost dominates either way, and one-versus-one's extra fits are exactly compensated by each one seeing fewer points. The difference in expense at a given  $C$  (Solution 6.9) is therefore about the constant<sup>a</sup> in front, not about how either strategy scales as the dataset grows.

<sup>a</sup><https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

**Exercise 6.12** (Compare the two strategies' costs (part 5)). Use the setup of Exercise 6.8. One-versus-one needs  $\binom{C}{2}$  predictions, each choosing between two specific classes, and a new point can end up with votes split across three or more classes with no clear winner. Propose a rule for breaking a tie.

Solution

*Solution 6.12.* Any tie-breaking rule has to say something when the simple vote does not decide, so a common choice is to fall back to Definition 6.1's own argmax: compute  $h^{(m)}(x)$  for the tied classes only, using each class's one-versus-rest classifier, and take the largest. This uses one-versus-one's votes when they decide the question and one-versus-rest's confidence score only to resolve what one-versus-one left

open, rather than committing to one strategy exclusively.

**Exercise 6.13** (Fit one classifier of a one-versus-rest scheme by hand). A three-class training set has one point per class:

$$\mathcal{X} = \{([3, 0], 1), ([-1, 0], 2), ([-3, 2], 3)\}.$$

Find  $w$ ,  $b$  and  $M$  for the hard-margin ( $C = \infty$ ) classifier that Definition 6.1 fits for class 1.

Write down the relabelling this classifier trains on: which point gets  $y = +1$ , and which get  $y = -1$ ?

Solution

*Solution 6.13.*  $y = +1$  for  $[3, 0]$ , and  $y = -1$  for both  $[-1, 0]$  and  $[-3, 2]$ , per Definition 6.1's relabelling for  $m = 1$ .

**Exercise 6.14** (Fit one classifier of a one-versus-rest scheme by hand (part 2)). Use the setup of Exercise 6.13.

With only one positive point, the optimal separating hyperplane is the perpendicular bisector between it and whichever negative point is nearest to it. Compute the distance from the class-1 point to each negative point, and identify the two [support vectors](#).

Solution

*Solution 6.14.* Distance from  $[3, 0]$  to  $[-1, 0]$ :

$$\begin{aligned}\sqrt{(3 - (-1))^2 + (0 - 0)^2} &= \sqrt{16} \\ &= 4.\end{aligned}$$

Distance from  $[3, 0]$  to  $[-3, 2]$ :

$$\begin{aligned}\sqrt{(3 - (-3))^2 + (0 - 2)^2} &= \sqrt{36 + 4} \\ &= \sqrt{40} \\ &\approx 6.32.\end{aligned}$$

$[-1, 0]$  is closer, so it and  $[3, 0]$  are the two [support vectors](#) — provided the resulting hyperplane does not cut inside  $[-3, 2]$ 's margin, which Solution 6.18 checks.

**Exercise 6.15** (Fit one classifier of a one-versus-rest scheme by hand (part 3)). Use the setup of Exercise 6.13.

The margin  $M$  is half the distance between the two support vectors. Compute it.

Solution

*Solution 6.15.*

$$\begin{aligned}M &= 4/2 \\ &= 2.\end{aligned}$$

**Exercise 6.16** (Fit one classifier of a one-versus-rest scheme by hand (part 4)). Use the setup of Exercise 6.13.

Theorem 1.3, under the scale Theorem 1.2 fixed, ties  $M$  to  $\|w\|_2$  by  $M = 1/\|w\|_2$ . Use that to find  $\|w\|_2$ .

### Solution

*Solution 6.16.*

$$\begin{aligned}\|w\|_2 &= 1/M \\ &= 1/2.\end{aligned}$$

**Exercise 6.17** (Fit one classifier of a one-versus-rest scheme by hand (part 5)). Use the setup of Exercise 6.13.

$w$  points along the line joining the two **support vectors**, so  $w$  is a positive multiple of the vector from the negative **support vector** to the positive one. Use Exercise 6.16's  $\|w\|_2$  to find  $w$  itself. Then the midpoint of the two support vectors lies on the hyperplane, where  $h(x) = 0$ ; use it to find  $b$ .

### Solution

*Solution 6.17.* The vector from  $[-1, 0]$  to  $[3, 0]$  is  $[4, 0]$ , so  $w = c[1, 0]$  for some  $c > 0$ . Then  $\|w\|_2 = c = 1/2$ , and  $w = [\frac{1}{2}, 0]$ .

The midpoint of the two **support vectors** is  $[1, 0]$ . It lies on the hyperplane, so  $h([1, 0]) = 0$ :

$$\begin{aligned}w^\top [1, 0] + b &= \frac{1}{2} + b \\ &= 0\end{aligned} \quad \Rightarrow \quad b = -\frac{1}{2}.$$

Checking the class-1 **support vector**'s constraint at equality:

$$\begin{aligned}y h([3, 0]) &= \frac{1}{2}(3) - \frac{1}{2} \\ &= 1,\end{aligned}$$

as it should.

**Exercise 6.18** (Fit one classifier of a one-versus-rest scheme by hand (part 6)). Use the setup of Exercise 6.13.

Check your answer: does the class-3 point satisfy its constraint with strict inequality, confirming it is not a **support vector** for this classifier?

### Solution

*Solution 6.18.* For  $[-3, 2]$ ,  $y = -1$  and

$$\begin{aligned}h([-3, 2]) &= \frac{1}{2}(-3) + 0(2) - \frac{1}{2} & y h(x) &= (-1)(-2) \\ &= -2, & &= 2.\end{aligned}$$

Since  $2 > 1$ , the constraint is satisfied with strict inequality, confirming  $[-3, 2]$  is not a **support vector** and the choice in Solution 6.14 was right.

## 7 Strengths and weaknesses

**Exercise 7.1** (Weigh what an SVM is good at against its drawbacks). Looking back over the chapter: name three properties that make the SVM attractive to use;

### Solution

*Solution 7.1.* In favor:

- $C$  is a direct, interpretable handle on overfitting. It is the cost of one unit of slack, and raising it

narrows the margin while lowering it widens it, so a single scalar moves the model along the whole range from “tolerate nothing” to “tolerate almost anything”.

- The kernel decouples the shape of the decision boundary from the cost of computing it. Swapping the linear kernel for an RBF kernel changes what the model can express without changing the optimization problem being solved.
- The problem is convex, so every local optimum is a global one, and there is no dependence on where the solver started. The  $w$  that comes back is unique (Solution 3.6), though  $b$  and the slack need not be.

**Exercise 7.2** (Weigh what an SVM is good at against its drawbacks (part 2)). Use the setup of Exercise 7.1.

name three that make it costly or limited.

Solution

*Solution 7.2.* Against:

- $C$ , the [kernel](#), and the kernel’s own parameters (such as the RBF’s  $\sigma$ ) are all hyperparameters with no default that works everywhere, so fitting an SVM means<sup>a</sup> running a search over them rather than a single fit.
- The formulation is binary. Multiclass problems have to be decomposed into binary ones, and every scheme for doing so has a drawback.
- The fitted model is hard to read. With a non-linear kernel there is no  $w$  to inspect, only the [support vectors](#) and their coefficients (Theorem 5.1), and “this point resembles those training points” is a weaker explanation than a coefficient per feature.

<sup>a</sup><https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

## 8 Further reading

- James et al. [5], Chapter 9
- Hastie et al. [28], Chapter 12
- Zhou [29], Support Vector Machines<sup>2</sup> — slides on *An Introduction to Statistical Learning*, Chapter 9, with an interactive demonstration and code in R
- Starmer [30], on support vector classifiers and machines — an illustrated walk-through with little algebra
- Lange [31], Section 17.5 — an introduction to reproducing [kernel](#) Hilbert spaces, the setting behind kernel methods

### Watch

The Data Science Learning Community’s book clubs discuss *An Introduction to Statistical Learning*, Chapter 9, in recorded meetings:

- the R book club, in part 1<sup>a</sup> and part 2<sup>b</sup> [32];
- the Python book club, in part 1<sup>c</sup> and part 2<sup>d</sup> [33].

<sup>a</sup><https://www.youtube.com/watch?v=susVQLM0XsE&list=PL3x6DOfs2NGjlhpKMiWsDcKL2nngwXROq>

<sup>b</sup><https://www.youtube.com/watch?v=TjyZeF-3CDg&list=PL3x6DOfs2NGjlhpKMiWsDcKL2nngwXROq>

<sup>c</sup><https://www.youtube.com/watch?v=Ikn4q7di0LU&list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp>

<sup>d</sup><https://www.youtube.com/watch?v=oGLlIT7LpRk&list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp>

<sup>2</sup><https://ucla-biostat-212a.github.io/2026winter/slides/09-svm/svm.html>

## References

- [1] V. N. Vapnik and A. Ya. Chervonenkis, “A note on one class of perceptrons,” *Automation and Remote Control*, vol. 25, 1964.
- [2] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on computational learning theory (COLT ’92)*, ACM, 1992, pp. 144–152. doi: 10.1145/130385.130401<sup>3</sup>.
- [3] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995, doi: 10.1007/BF00994018<sup>4</sup>.
- [4] Wikipedia contributors, “Decision boundary — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Decision\\_boundary](https://en.wikipedia.org/wiki/Decision_boundary)
- [5] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: <https://www.statlearning.com/>
- [6] Wikipedia contributors, “Linear separability — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Linear\\_separability](https://en.wikipedia.org/wiki/Linear_separability)
- [7] Wikipedia contributors, “Support vector machine — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)
- [8] GeeksforGeeks, “Support vector machine (SVM) algorithm.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/>
- [9] A. Ng *et al.*, “CS229: Machine learning, autumn 2018.” Stanford Online, lecture videos on YouTube, 2018. Available: <https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShTMGgzqpfVfbU>
- [10] T. Hastie, R. Tibshirani, and J. Taylor, “Statistical learning with python.” Stanford Online, lecture videos on YouTube, 2023. Available: <https://www.youtube.com/playlist?list=PLoROMvodv4rPP6braWoRt5UCXYZ71GZIQ>
- [11] Y. S. Abu-Mostafa, “Learning from data: CS 156 lectures.” Caltech, lecture videos on YouTube, 2012. Available: <https://www.youtube.com/playlist?list=PLD63A284B7615313A>
- [12] K. Weinberger, “CS4780: Machine learning for intelligent systems, spring 2017.” Cornell University, lecture videos on YouTube, 2017. Available: <https://www.youtube.com/playlist?list=PLl8OIHZGYOQ7bkVbuRthEsaLr7bONzbXS>
- [13] B. Schölkopf, R. Herbrich, and A. J. Smola, “A generalized representer theorem,” in *Computational learning theory (COLT 2001 / EuroCOLT 2001)*, in Lecture notes in computer science, vol. 2111. Springer, 2001, pp. 416–426. doi: 10.1007/3-540-44581-1\_27<sup>5</sup>.
- [14] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, “Spambase.” UCI Machine Learning Repository, 1999. doi: 10.24432/C53G6X<sup>6</sup>.
- [15] Wikipedia contributors, “Karush–kuhn–tucker conditions — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Karush%E2%80%93kuhn%E2%80%93tucker\\_conditions](https://en.wikipedia.org/wiki/Karush%E2%80%93kuhn%E2%80%93tucker_conditions)
- [16] GeeksforGeeks, “Dual support vector machine.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/dual-support-vector-machine/>
- [17] J. Starmer, “Support vector machines part 1 (of 3): Main ideas.” StatQuest, video on YouTube, 2019. Available: <https://www.youtube.com/watch?v=efR1C6CvbmE>
- [18] GeeksforGeeks, “Using a hard margin vs soft margin in SVM.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/using-a-hard-margin-vs-soft-margin-in-svm/>
- [19] Wikipedia contributors, “Hinge loss — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Hinge\\_loss](https://en.wikipedia.org/wiki/Hinge_loss)
- [20] GeeksforGeeks, “Hinge-loss & relationship with support vector machines.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/hinge-loss-relationship-with-support-vector-machines/>
- [21] J. Starmer, “Support vector machines part 2: The polynomial kernel (part 2 of 3).” StatQuest, video on YouTube, 2019. Available: <https://www.youtube.com/watch?v=Toet3EiSFcM>

---

<sup>3</sup><https://doi.org/10.1145/130385.130401>

<sup>4</sup><https://doi.org/10.1007/BF00994018>

<sup>5</sup>[https://doi.org/10.1007/3-540-44581-1\\_27](https://doi.org/10.1007/3-540-44581-1_27)

<sup>6</sup><https://doi.org/10.24432/C53G6X>

- [22] J. Starmer, “Support vector machines part 3: The radial (RBF) kernel (part 3 of 3).” StatQuest, video on YouTube, 2019. Available: [https://www.youtube.com/watch?v=Qc5IyLW\\_hns](https://www.youtube.com/watch?v=Qc5IyLW_hns)
- [23] Wikipedia contributors, “Kernel method — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Kernel\\_method](https://en.wikipedia.org/wiki/Kernel_method)
- [24] GeeksforGeeks, “Kernel trick in support vector classification.” GeeksforGeeks, tutorial article, 2024. Available: <https://www.geeksforgeeks.org/machine-learning/kernel-trick-in-support-vector-classification/>
- [25] GeeksforGeeks, “Major kernel functions in support vector machine (SVM).” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/major-kernel-functions-in-support-vector-machine-svm/>
- [26] GeeksforGeeks, “One-vs-rest strategy for multi-class classification.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/one-vs-rest-strategy-for-multi-class-classification/>
- [27] Wikipedia contributors, “Multiclass classification — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Multiclass\\_classification](https://en.wikipedia.org/wiki/Multiclass_classification)
- [28] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. Available: <https://hastie.su.domains/ElemStatLearn/>
- [29] J. Zhou, “Biostat 212A: Statistical machine learning a, winter 2026.” UCLA, course webpage with lecture slides, 2026. Available: <https://ucla-biostat-212a.github.io/2026winter/>
- [30] J. Starmer, *The StatQuest illustrated guide to machine learning*. StatQuest Publications, 2022. Available: <https://statquest.org/statquest-store/>
- [31] K. Lange, *Numerical analysis for statisticians*, 2nd ed. in Statistics and computing. Springer, 2010. doi: 10.1007/978-1-4419-5945-4<sup>7</sup>.
- [32] Data Science Learning Community, “Introduction to statistical learning using R (islr06).” Book club meeting recordings on YouTube, 2023. Available: <https://www.youtube.com/playlist?list=PL3x6DOfs2NGjIhpKMiWsDcKL2nngwXROq>
- [33] Data Science Learning Community, “Introduction to statistical learning using Python (islp03).” Book club meeting recordings on YouTube, 2025. Available: <https://www.youtube.com/playlist?list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp>

---

<sup>7</sup><https://doi.org/10.1007/978-1-4419-5945-4>