Lecture 009

Support vector machines

Edward Rubin

25 February 2021

Admin

Today

Topic Support vector machines

Upcoming

Readings

- *Today* *ISL* Ch. 9
- *Next* *100ML* Ch. 6

Project Project updates/questions?

Support vector machines

Support vector machines

Intro

Support vector machines (SVMs) are a *general class* of classifiers that essentially attempt to separate two classes of observations.

> SVMs have been shown to perform well in a variety of settings, and are often considered one of the best "out of the box" classifiers. *ISL, p. 337*

The support vector machine generalizes a much simpler classifier—the maximal margin classifier.

The maximal margin classifier attempts to separate the **two classes** in our prediction space using **a single hyperplane**.

Support vector machines

What's a hyperplane?

Consider a space with p dimensions.

A hyperplane is a $p-1$ dimensional **subspace** that is

1. **flat** (no curvature)
2. **affine** (may or may not pass through the origin)

Example In $p=1$ dimension, a *hyperplane* is a

Support vector machines

What's a hyperplane?

Consider a space with p dimensions.

A hyperplane is a $p - 1$ dimensional **subspace** that is

1. **flat** (no curvature)
2. **affine** (may or may not pass through the origin)

Example In $p = 1$ dimension, a *hyperplane* is a point.

Support vector machines

What's a hyperplane?

Consider a space with p dimensions.

A hyperplane is a $p - 1$ dimensional **subspace** that is

1. **flat** (no curvature)
2. **affine** (may or may not pass through the origin)

Example In $p = 2$ dimensions, a *hyperplane* is a

What's a hyperplane?

Consider a space with p dimensions.

A hyperplane is a $p - 1$ dimensional **subspace** that is

1. **flat** (no curvature)
2. **affine** (may or may not pass through the origin)

Example In $p = 2$ dimensions, a *hyperplane* is a line.

What's a hyperplane?

Consider a space with p dimensions.

A hyperplane is a $p - 1$ dimensional **subspace** that is

1. **flat** (no curvature)
2. **affine** (may or may not pass through the origin)

Example In $p = 3$ dimensions, a *hyperplane* is a (2D) plane.

Support vector machines

Hyperplanes

We can define a **hyperplane** in p dimensions by constraining the linear combination of the p dimensions.†

For example, in two dimensions a hyperplane is defined by

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

which is just the equation for a line.

Points $X = (X_1,\ X_2)$ that satisfy the equality *live* on the hyperplane.††

† Plus some offset ("intercept")

†† Alternatively: The hyperplane is composed of such points.

Support vector machines

Separating hyperplanes

More generally, in p dimensions, we defined a hyperplane by

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{A}$$

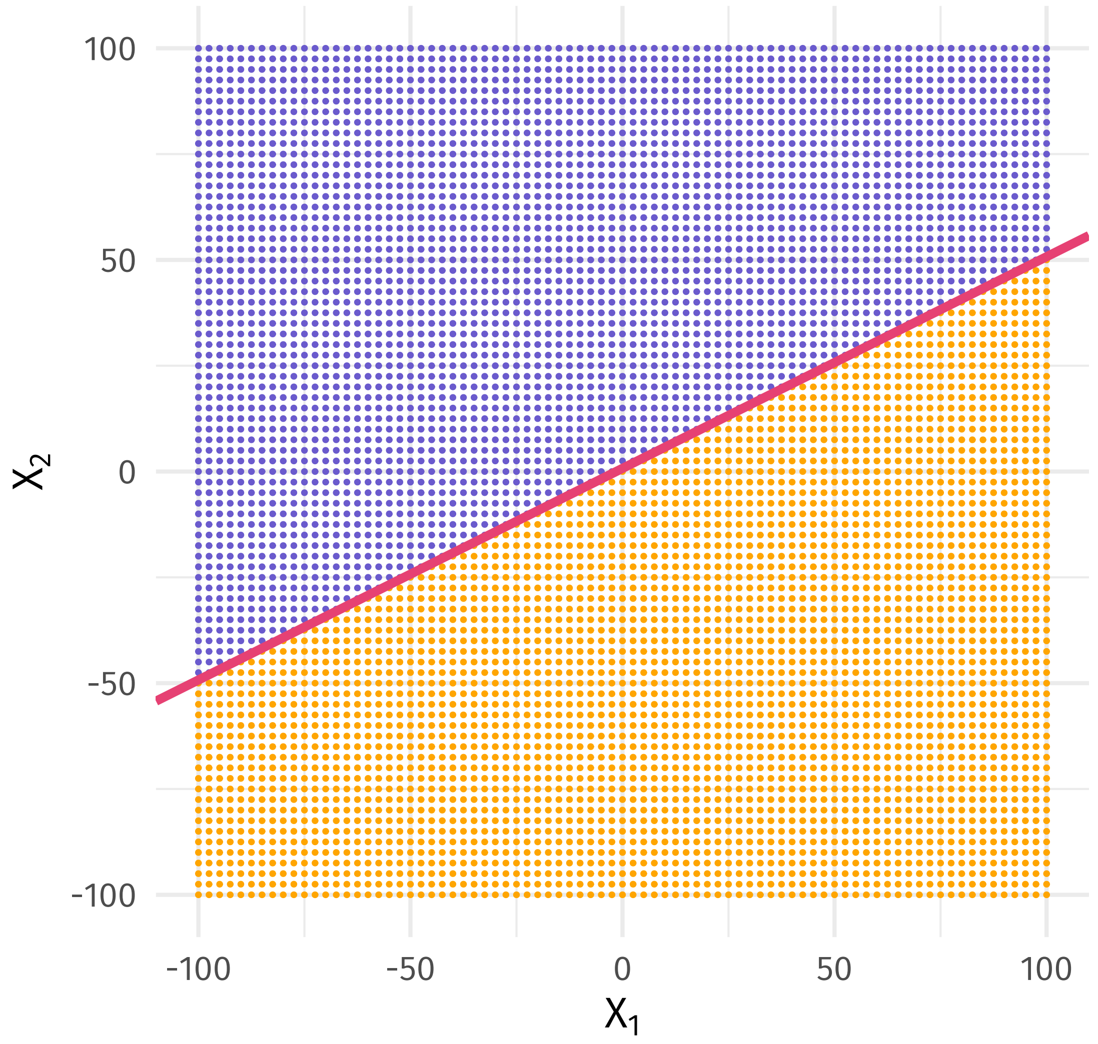
If $X = (X_1,\ X_2,\ \ldots,\ X_p)$ satisfies the equality, it is on the hyperplane.

Of course, not every point in the p dimensions will satisfy $\mathbf{A}$.

- If $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p > 0$, then X is ***above*** the hyperplane.
- If $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p < 0$, then X sits ***below*** the hyperplane.

The hyperplane *separates* the p-dimensional space into two "halves".

Ex: A **separating hyperplane** in two dimensions: $3 + 2X_1 - 4X_2 = 0$



Ex: A **separating hyperplane** in 3 dimensions: $3 + 2X_1 - 4X_2 + 2X_3 = 0$

Support vector machines

Separating hyperplanes and classification

Idea: *Separate* two classes of outcomes in the p dimensions of our predictor space using a separating hyperplane.

To make a prediction for observation $(x^o, y^o) = (x_1^o, x_2^o, \ldots, x_p^o, y^o)$:

We classify points that live "above" of the plane as one class, *i.e.*,

$$\text{If } \beta_0 + \beta_1 x_1^o + \cdots + \beta_p x_p^o > 0, \text{ then } \hat{y}^o = \text{Class 1}$$

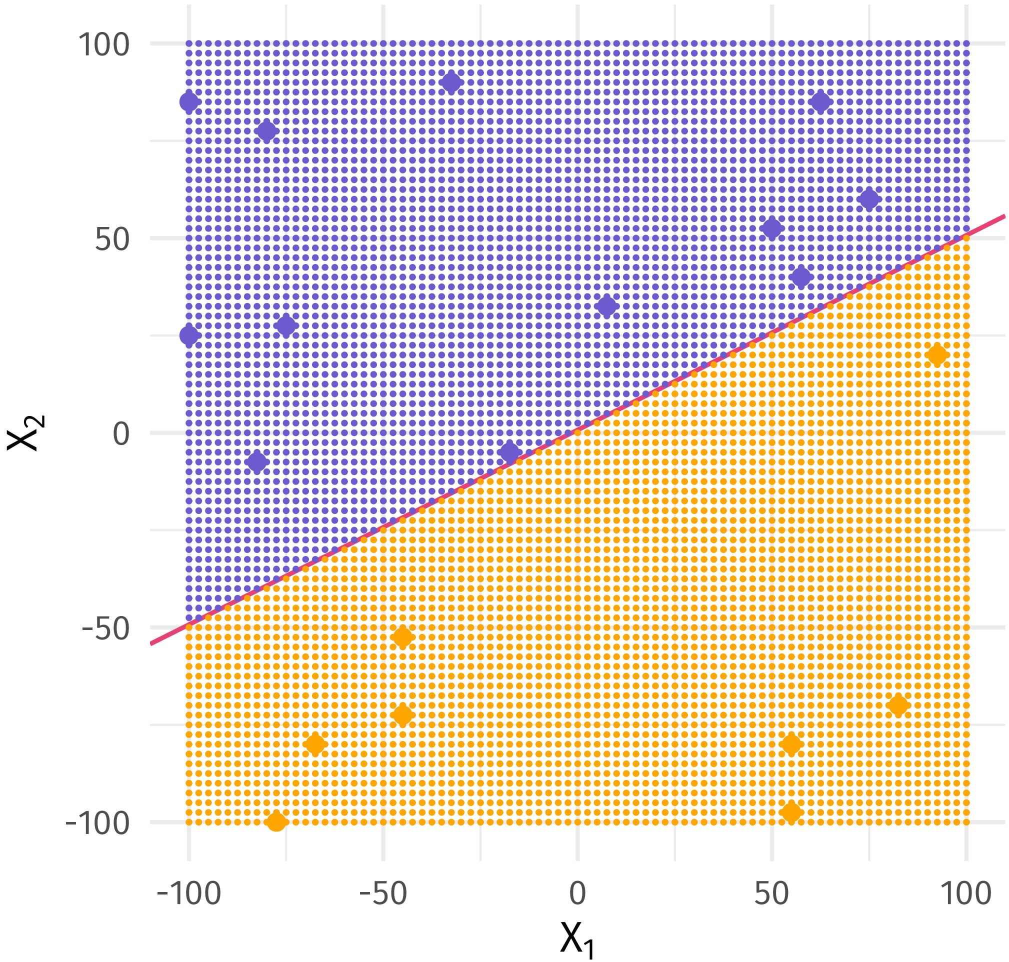
We classify points "below" the plane as the other class, *i.e.*,

$$\text{If } \beta_0 + \beta_1 x_1^o + \cdots + \beta_p x_p^o < 0, \text{ then } \hat{y}^o = \text{Class 2}$$

Note This strategy assumes a separating hyperplane exists.

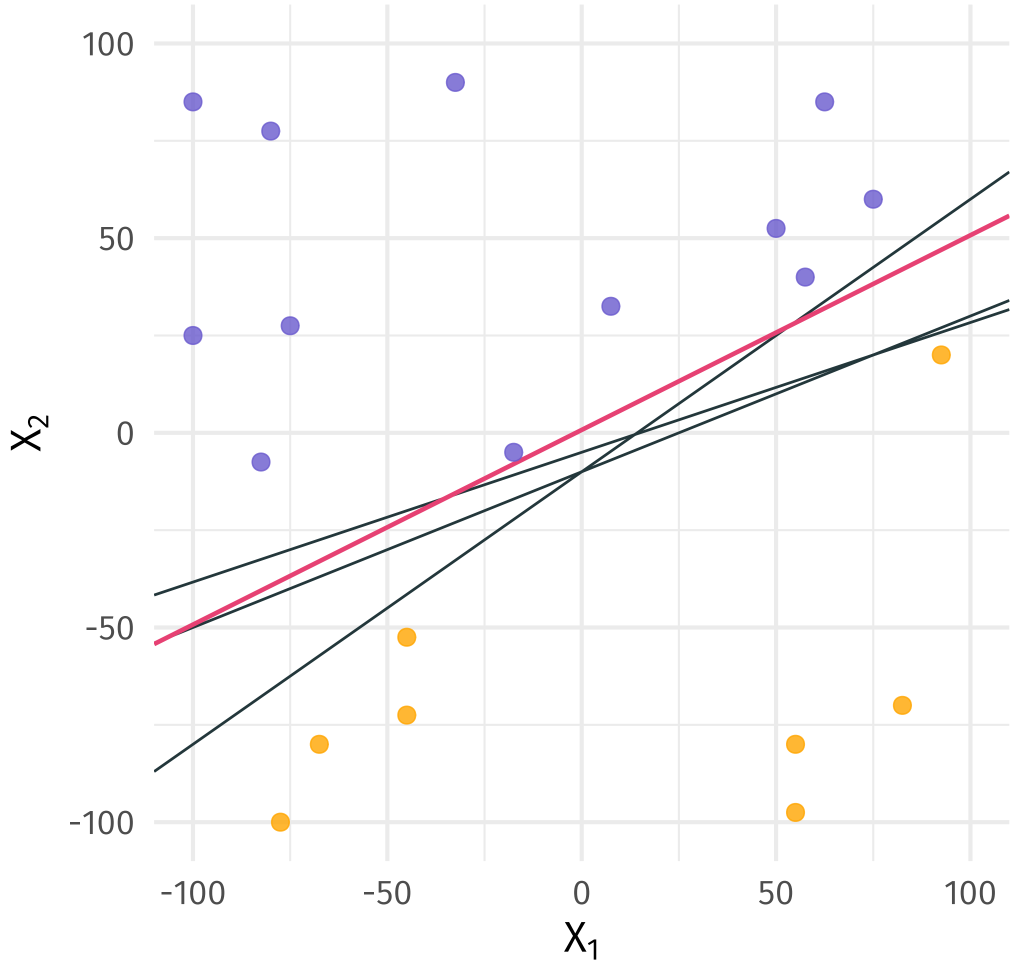
Support vector machines

If **a separating hyperplane** exists, then it defines a binary classifier.



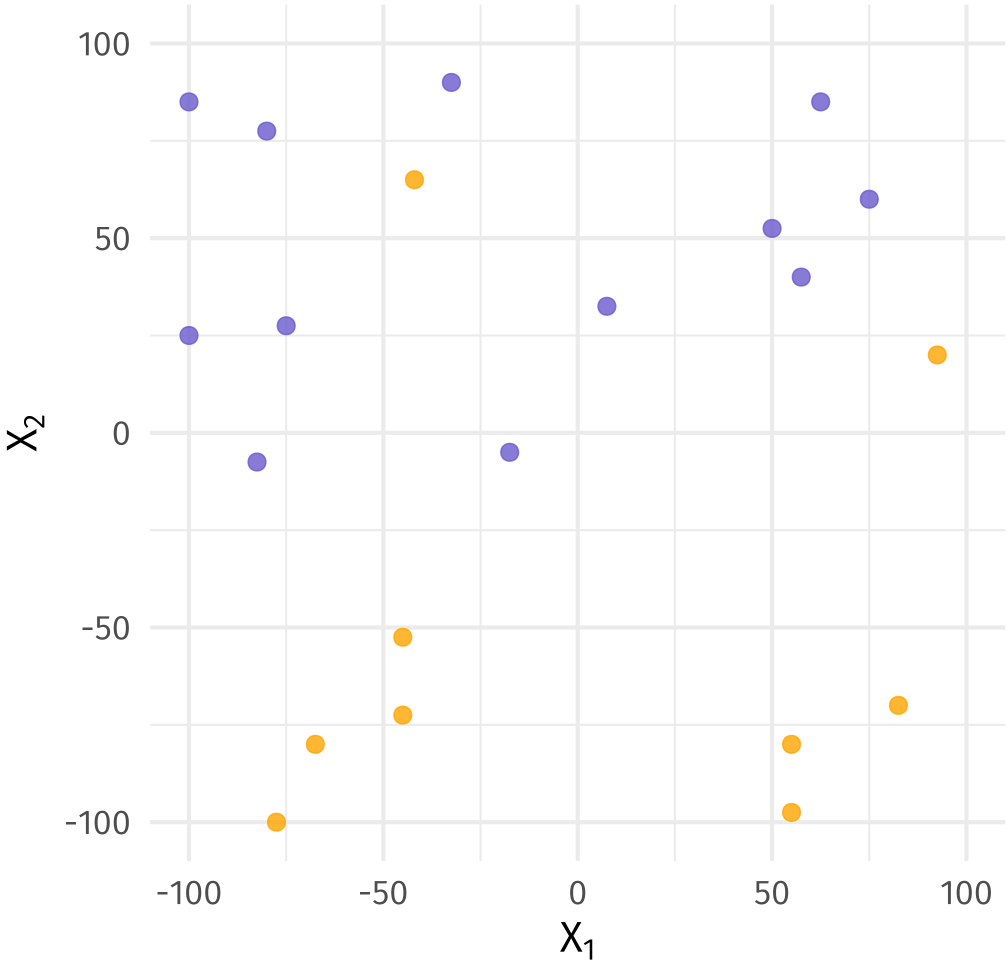
Support vector machines

If **a separating hyperplane** exists, then **many separating hyperplanes** exist.



Support vector machines

A **a separating hyperplane** may not exist.



Support vector machines

Decisions

Summary A given hyperplane

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0$$

produces a decision boundary.

We can determine any point's ($\mathbf{x}^o$) *side* of the boundary.

$$f(\mathbf{x}^o) = \beta_0 + \beta_1 x_1^o + \beta_2 x_2^o + \cdots + \beta_p x_p^o$$

We classify observationg $\mathbf{x}^o$ based upon whether $f(\mathbf{x}^o)$ is positive/negative.

The magnitude of $f(\mathbf{x}^o)$ tells us about our *confidence* in the classification.†

† Larger magnitudes are farther from the boundary.

Support vector machines

Which separating hyperplane?

Q How do we choose between the possible hyperplanes?

A *One solution:* Choose the separating hyperplane that is "farthest" from the training data points—maximizing "separation."

The **maximal margin hyperplane**[†] is the hyperplane that

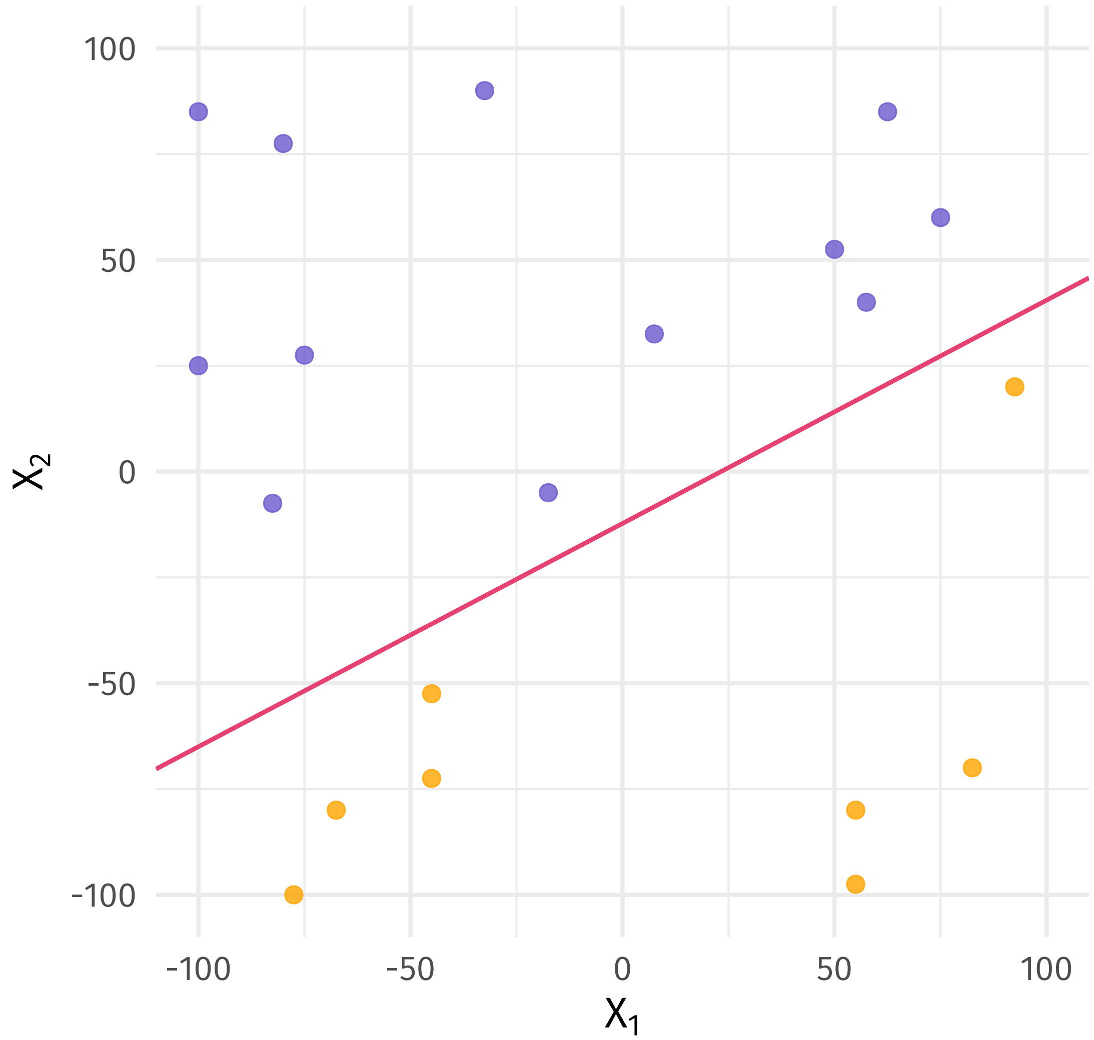
1. **separates** the two classes of obsevations
2. **maximizes** the **margin**—the distance to the nearest observation[††]

where *distance* is a point's perpendicular distance to the hyperplane.

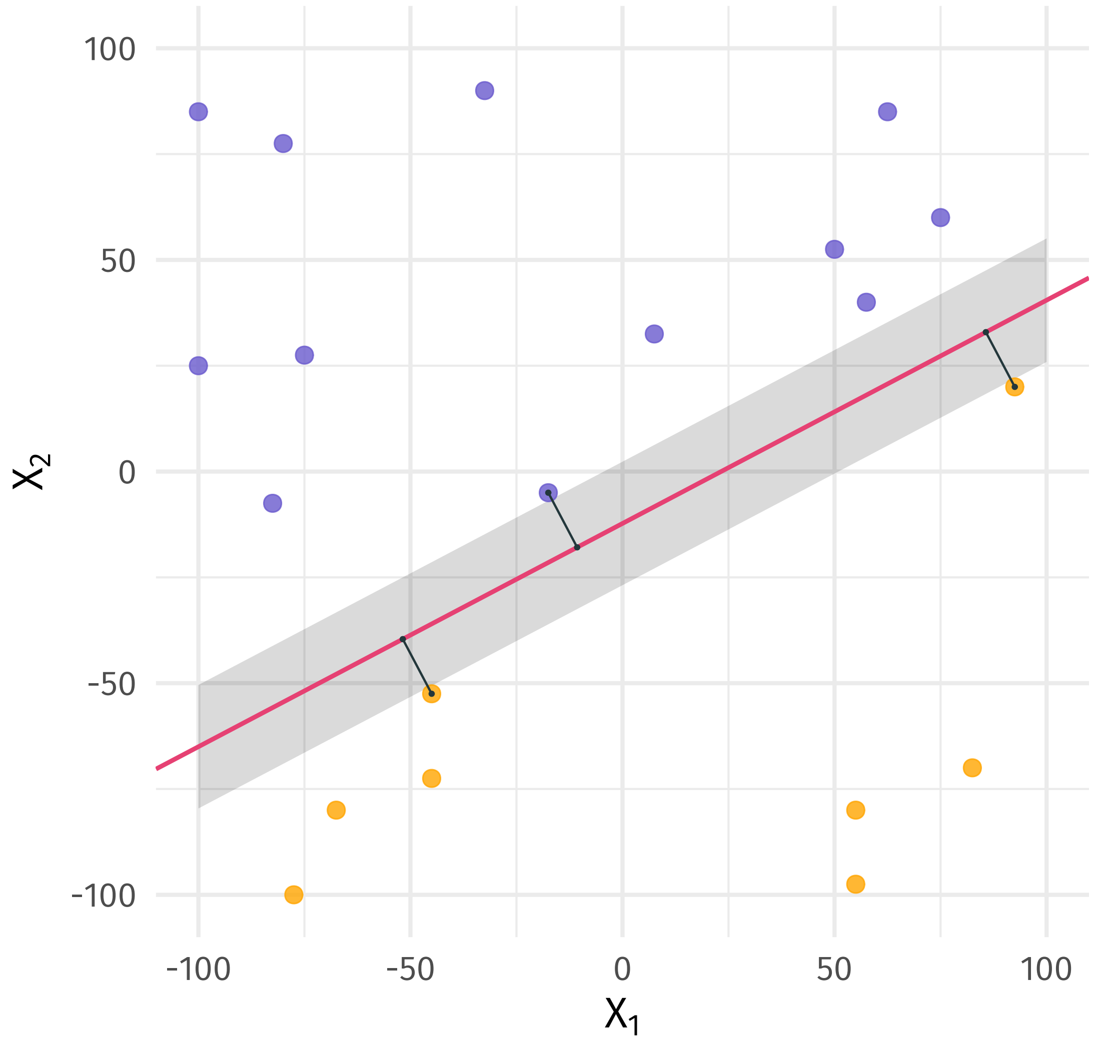
† AKA the *optimal separating hyperplane*

†† Put differently: The smallest distance to a training observation.

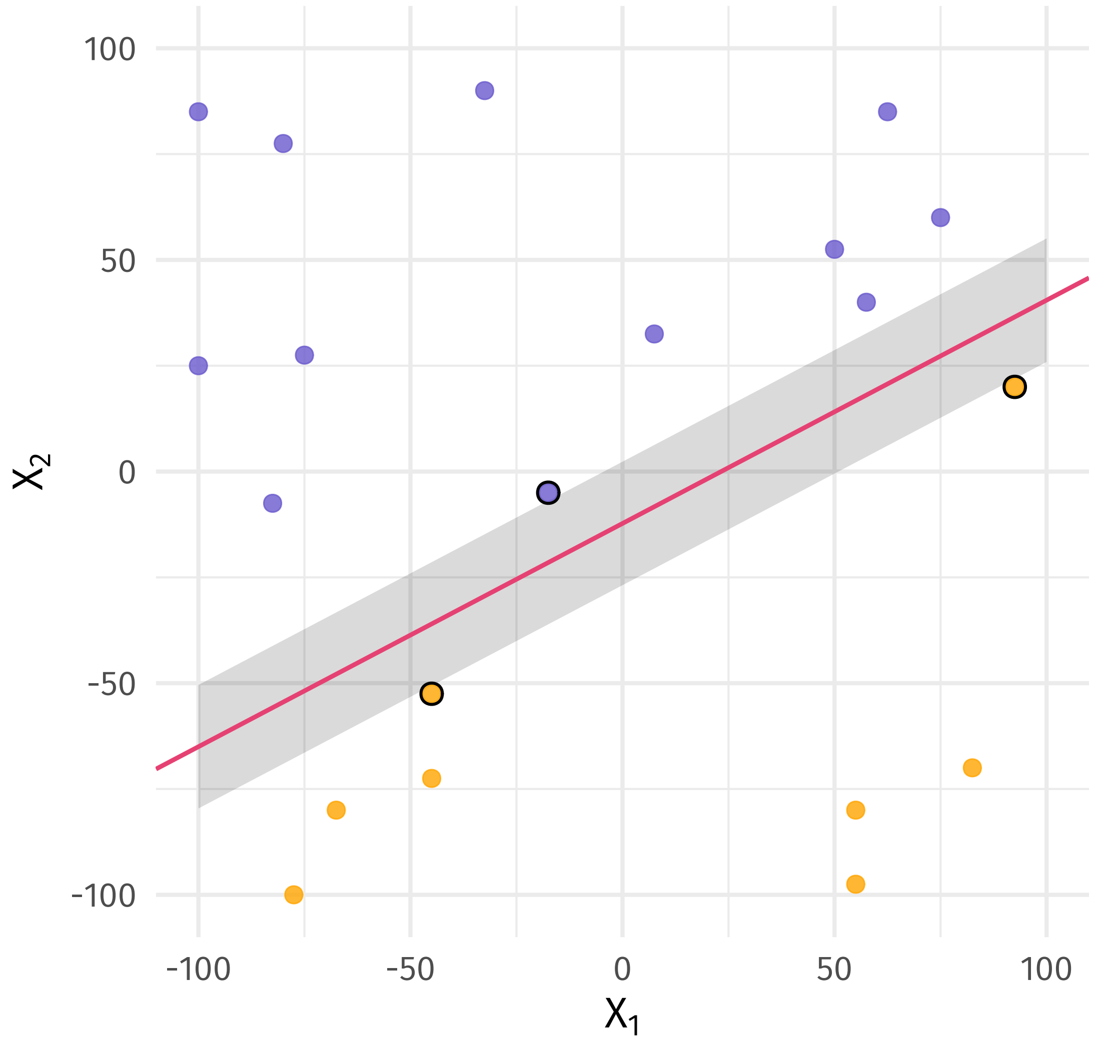
The **maximal margin hyperplane**...



…maximizes the **margin** between the hyperplane and training data…



...and is supported by three equidistant observations—the **support vectors**.



Support vector machines

The maximal margin hyperplane

Formally, the maximal margin hyperplane solves the problem:

Maximize the margin M over the set of $\{\beta_0,\ \beta_1,\ \ldots,\ \beta_p,\ M\}$ such that

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{1}$$

$$y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M \tag{2}$$

for all observations i.

(2) Ensures we separate (classify) observations correctly.

(1) allows us to interpret (2) as "distance from the hyperplane".

Support vector machines

Fake constraints

Note that our first "constraint"

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{1}$$

does not actually constrain $-1 \leq \beta_j \leq 1$ (or the hyperplane).

If we can define a hyperplane by

$$\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p} = 0$$

then we can also rescale the same hyperplane with some constant k

$$k\left(\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p}\right) = 0$$

The maximal margin classifier

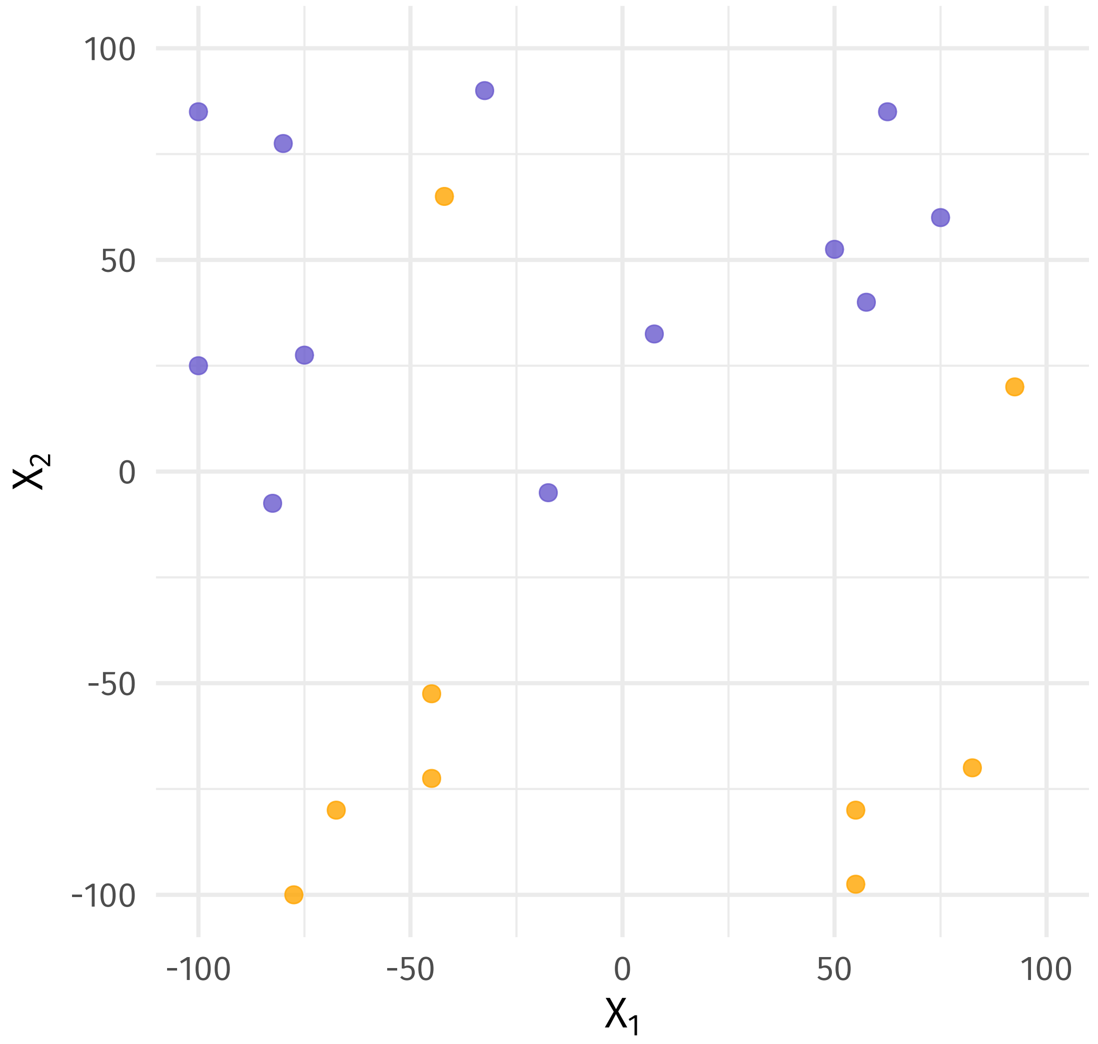
The maximal margin hyperplane produces the maximal margin classifier.

Notes

1. We are doing **binary classification**.
2. The decision boundary only uses the **support vectors**—very sensitive.
3. This classifier can struggle in **large dimensions** (big p).
4. A separating hyperplane does not always exist (**non-separable**).
5. Decision boundaries can be **nonlinear**.

Let's start by addressing non-separability...

Surely there's still a decent hyperplane-based classifier here, right?



Soft margins

When we cannot *perfectly* separate our classes, we can use **soft margins**, which are margins that "accept" some number of observations.

The idea: We will allow observations to be

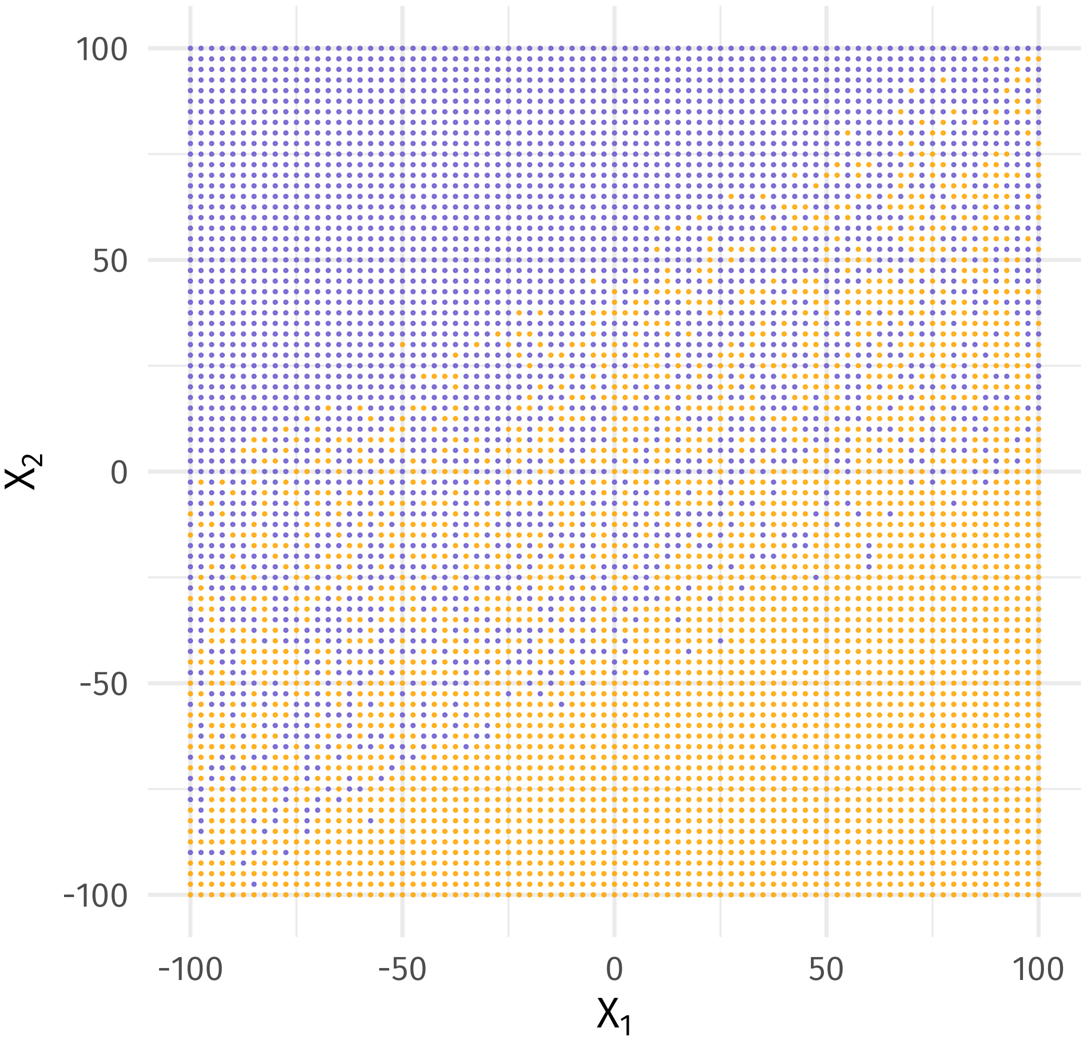
1. in the margin
2. on the wrong side of the hyperplane

but each will come with a price.

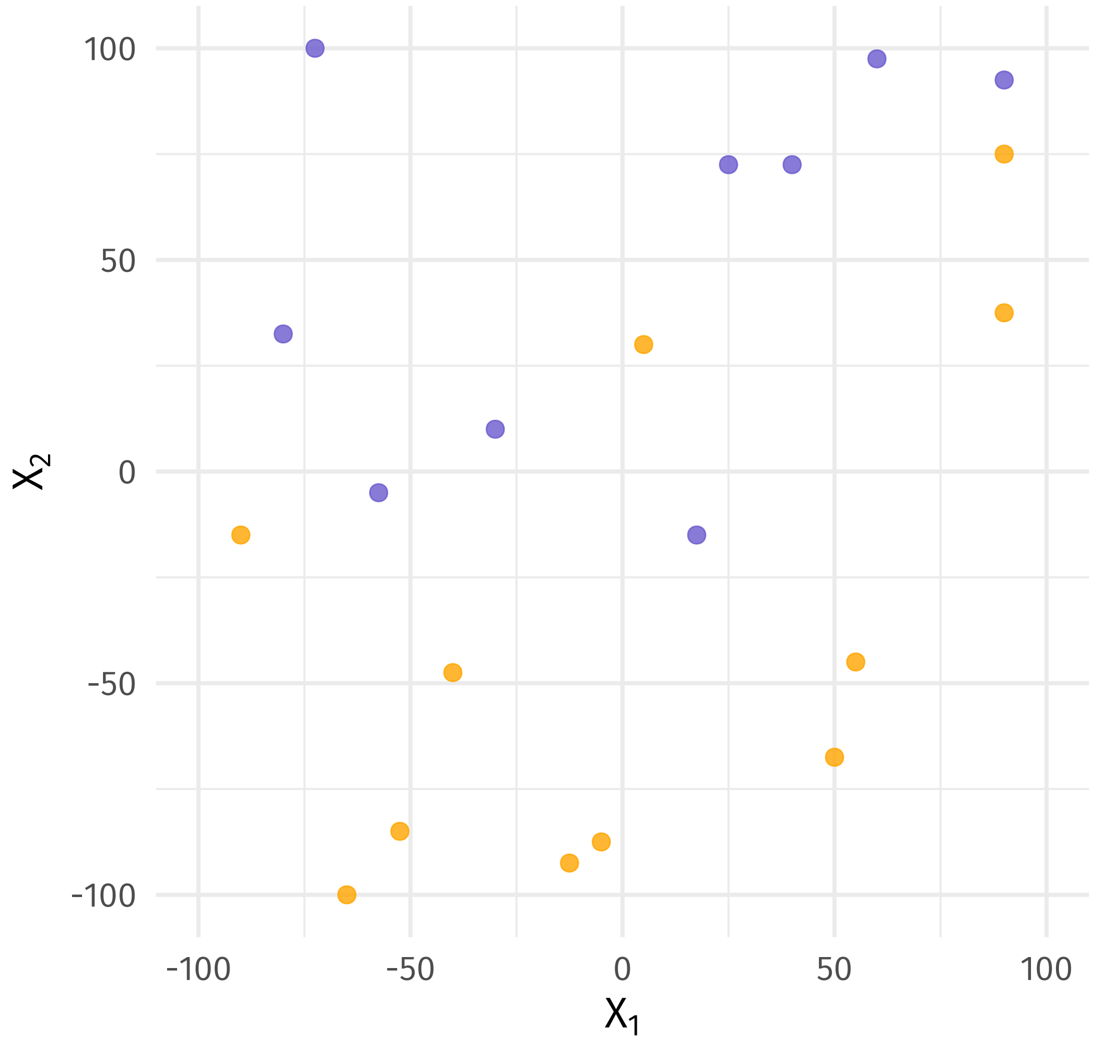
Using these *soft margins*, we create a hyperplane-based classifier called the **support vector classifier**.†

† Also called the *soft margin classifier*.

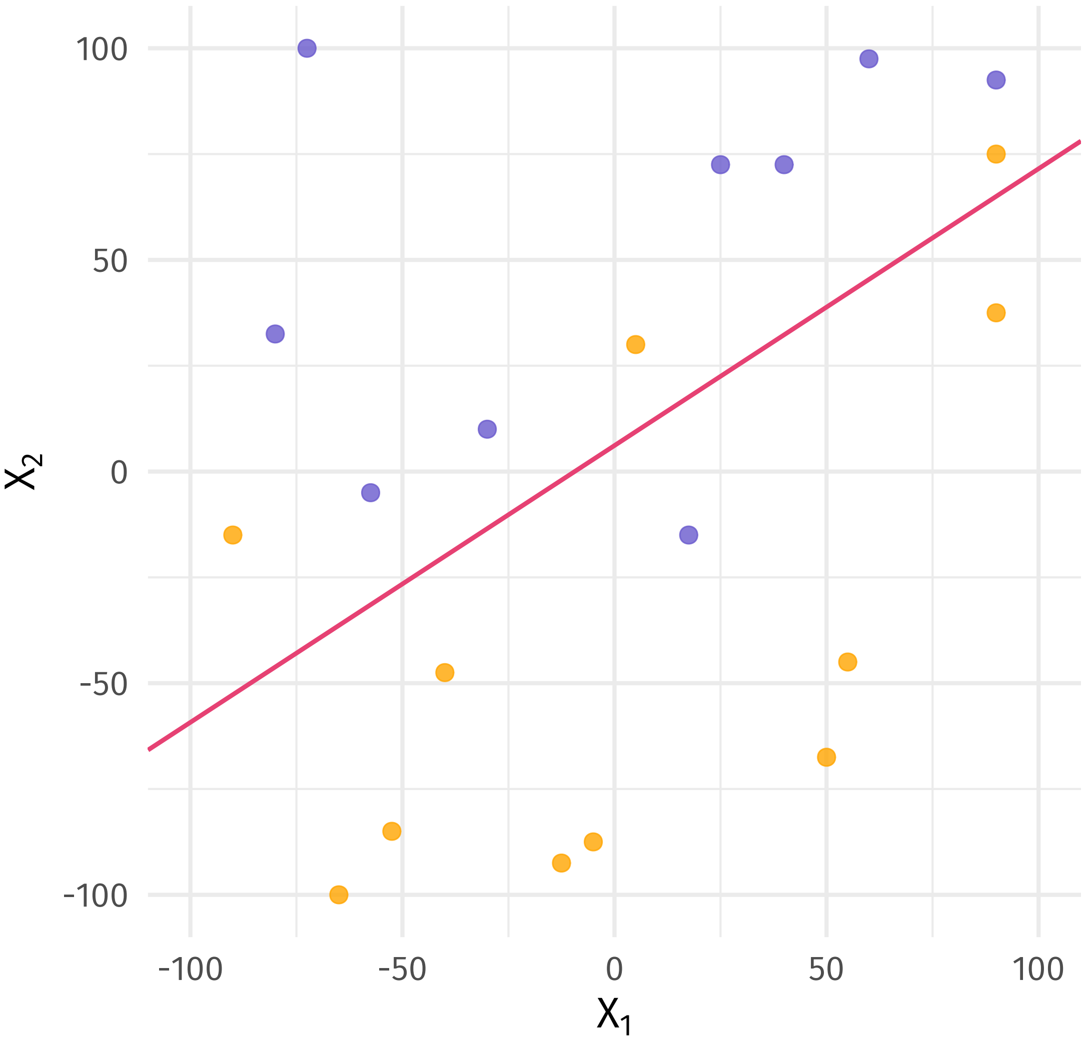
Our underlying population clearly does not have a separating hyperplane.



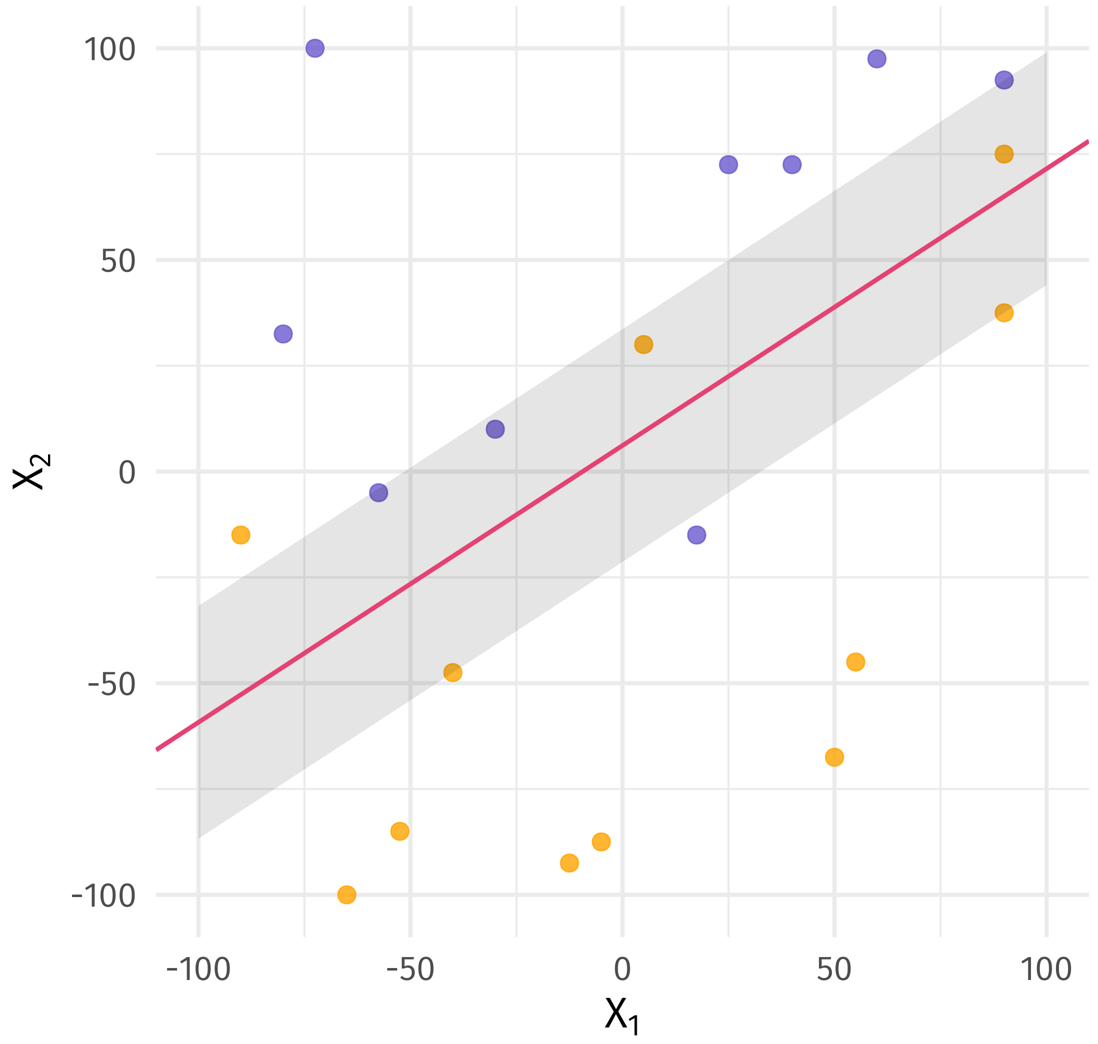
Our sample population also does not have a separating hyperplane.



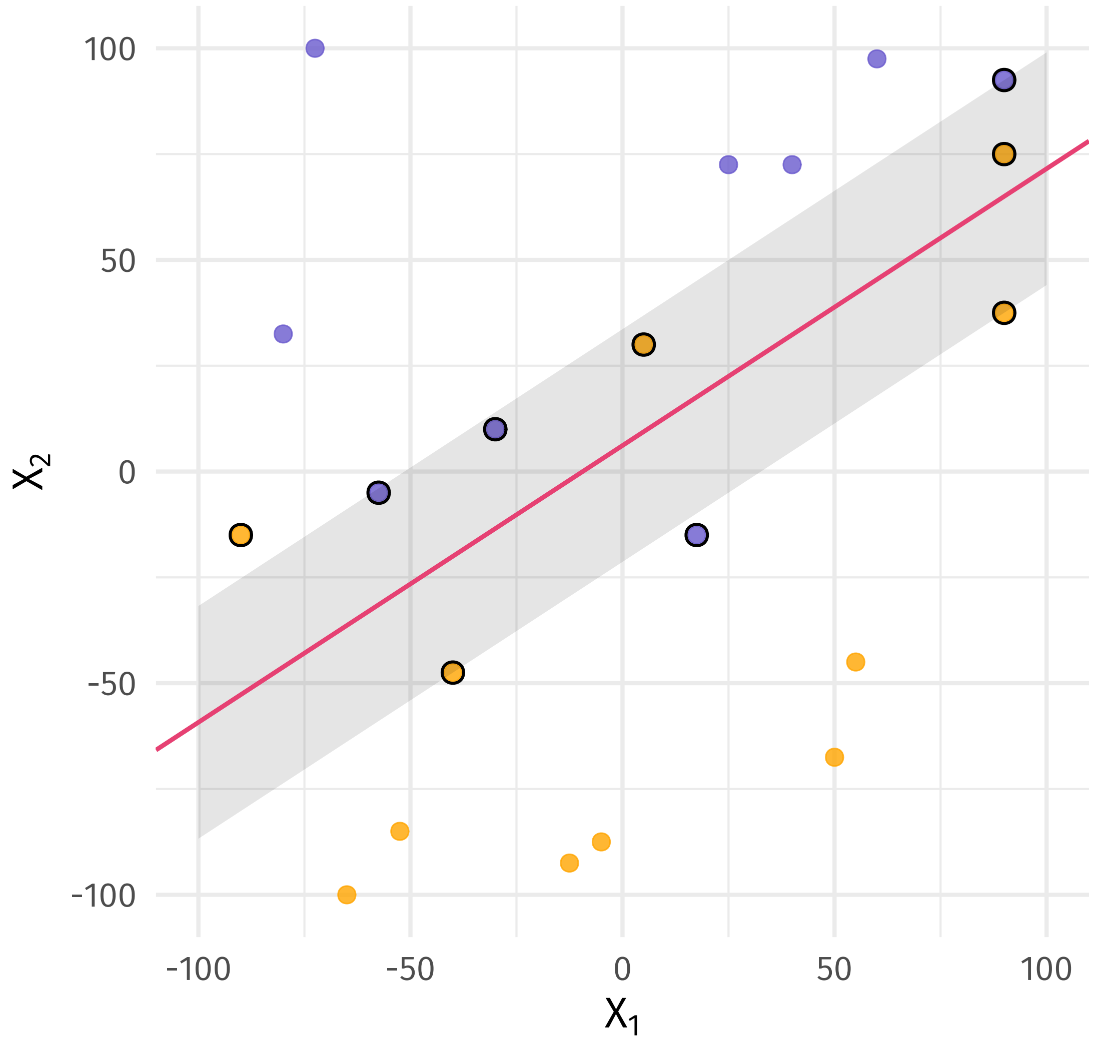
Our **hyperplane**



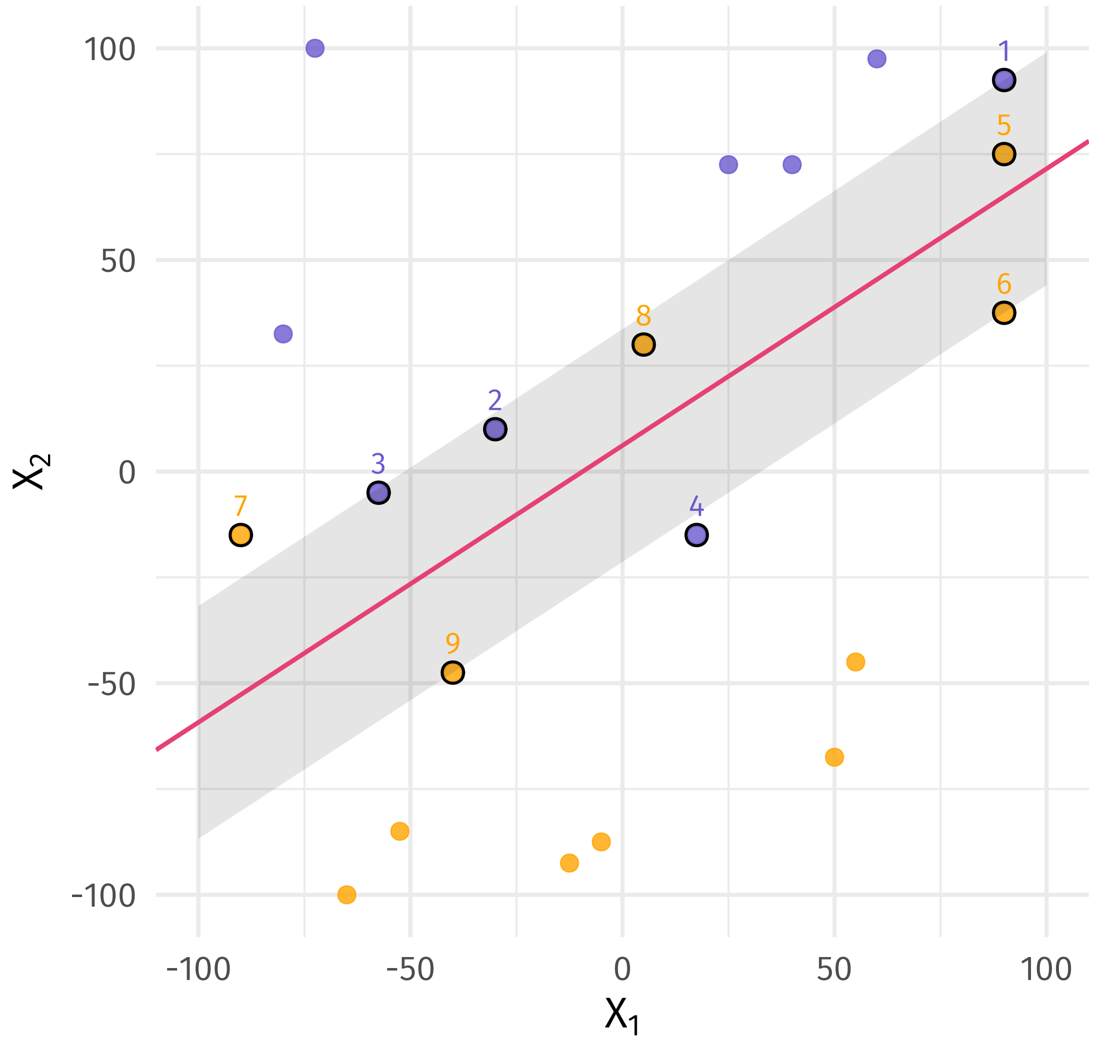
Our **hyperplane** with **soft margins**...



Our **hyperplane** with **soft margins** and **support vectors**.



Support vectors: on (i) the margin or (ii) on the wrong side of the margin.



Support vector classifier

The support vector classifier selects a hyperplane by solving the problem

Maximize the margin M over the set $\{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_n, M\}$ s.t.

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{3}$$

$$y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M\left(1 - \epsilon_i\right) \tag{4}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C \tag{5}$$

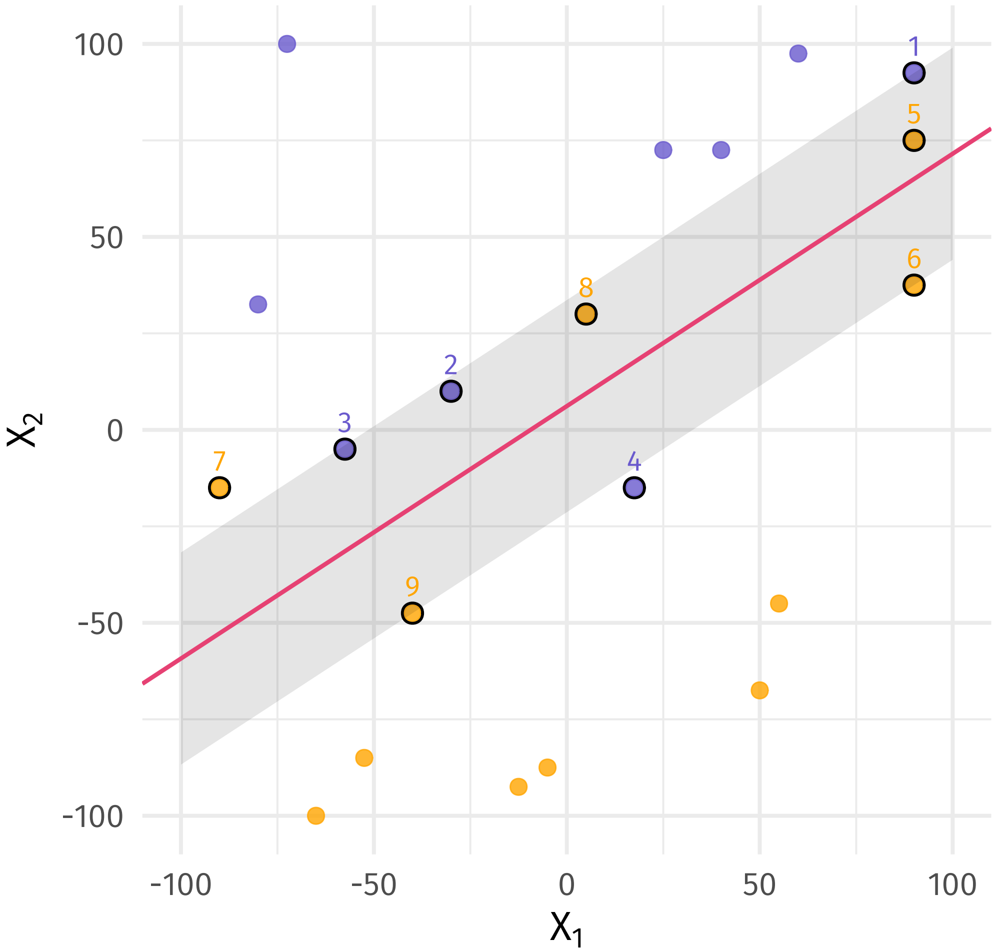
The ϵ_i are slack variables that allow i to *violate* the margin or hyperplane. C gives is our budget for these violations.

Let's consider constraints (4) and (5) work together...

$$y_i\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{ip}\right) \geq M\left(1-\epsilon_i\right) \tag{4}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^n \epsilon_i \leq C \tag{5}$$

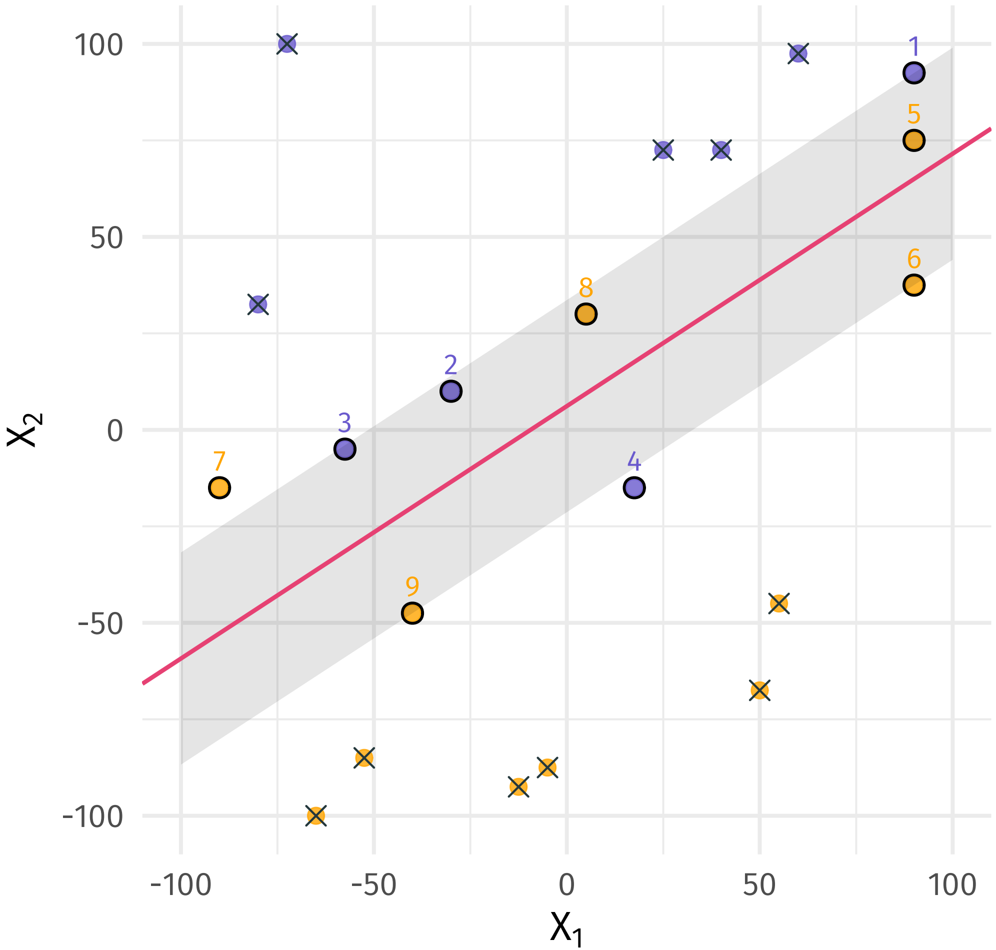
$$y_i\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{ip}\right) \geq M\left(1-\epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



For $\epsilon_i = 0$:

- $M\left(1-\epsilon_i\right) > 0$
- Correct side of hyperplane
- Correct side of margin (or on margin)
- No cost (C)
- Distance $\geq M$
- *Examples?*

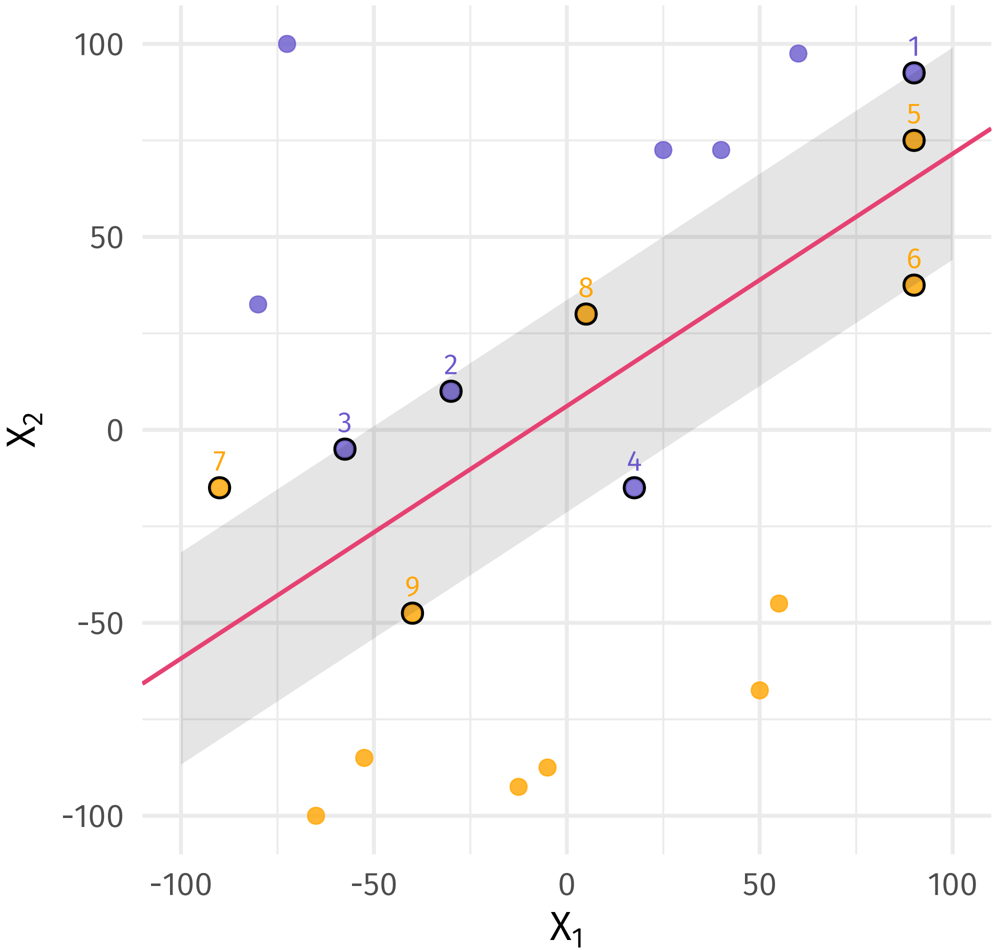
$$y_i\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{ip}\right) \geq M\left(1-\epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



For $\epsilon_i = 0$:

- $M\left(1-\epsilon_i\right) > 0$
- Correct side of hyperplane
- Correct side of margin (or on margin)
- No cost (C)
- Distance $\geq M$
- Correct side of margin: ($\times$)
- *On margin:* 1, 6, 9

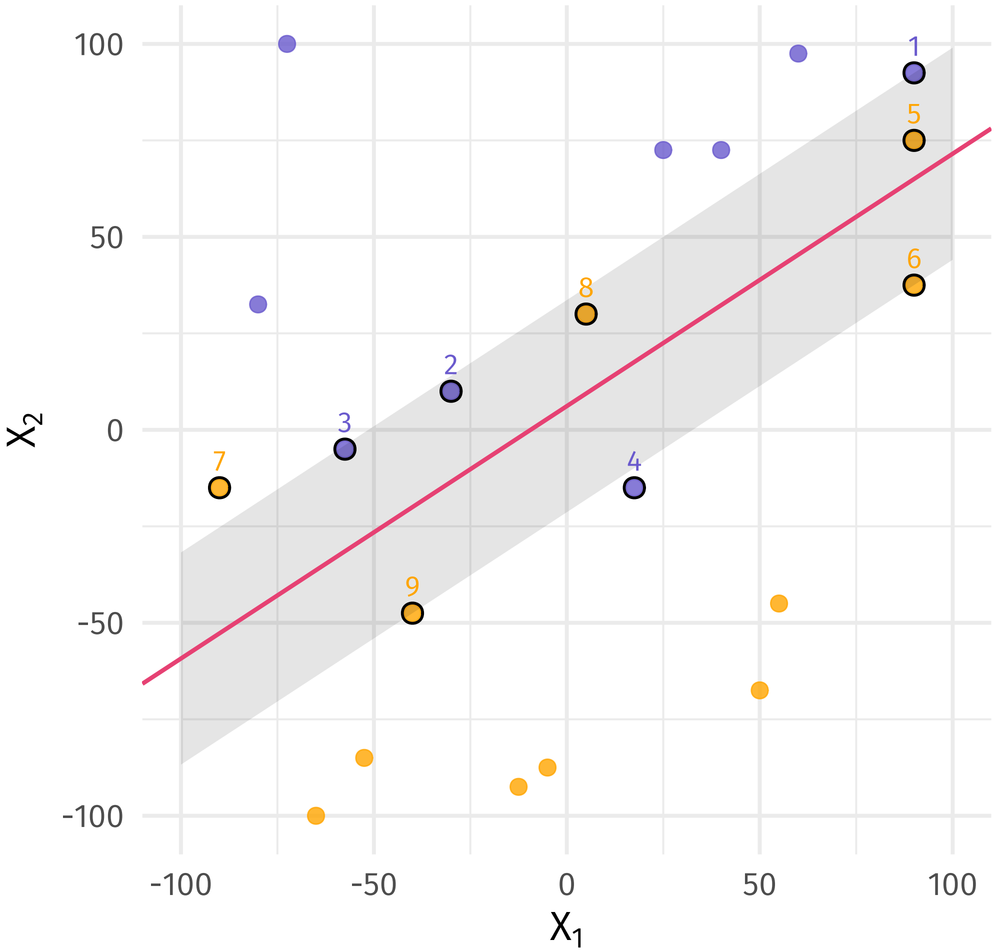
$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M\left(1 - \epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



For $0 \leq \epsilon_i \leq 1$:

- $M\left(1 - \epsilon_i\right) > 0$
- Correct side of hyperplane
- Wrong side of the margin (*violates margin*)
- Pays cost ϵ_i
- Distance $< M$
- *Examples?*

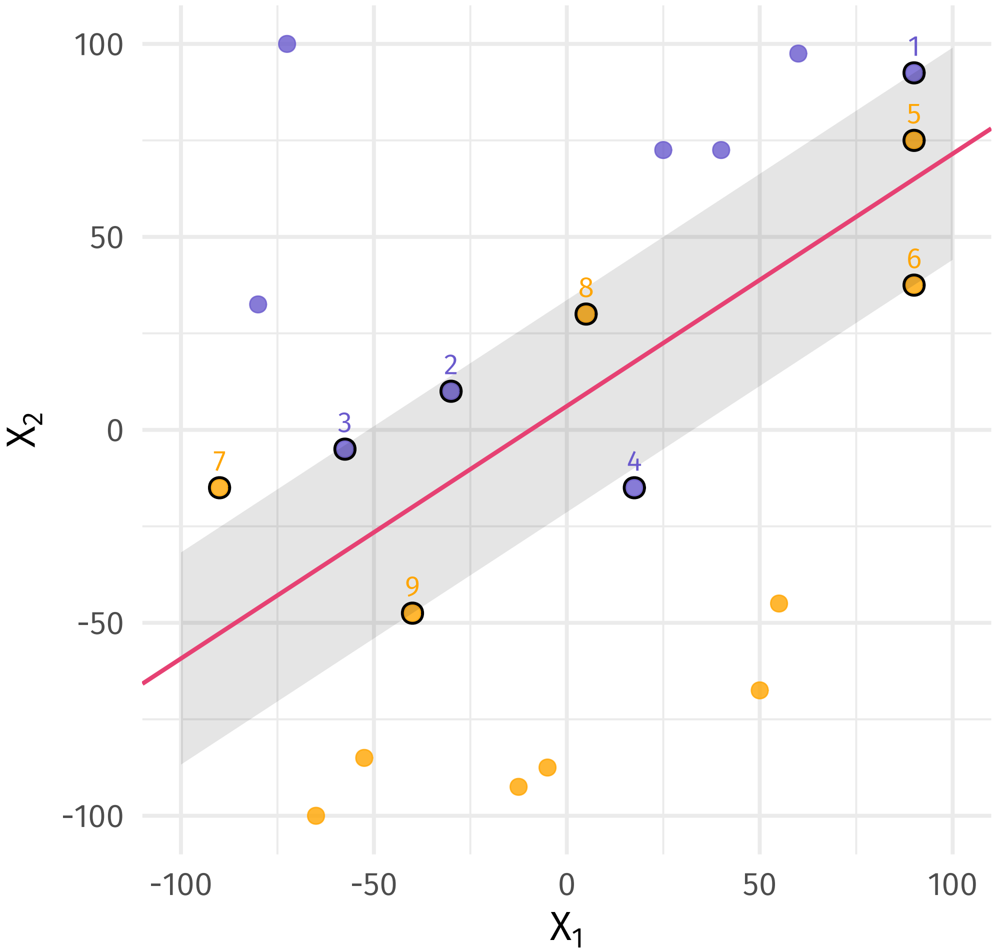
$$y_i\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{ip}\right)\geq M\left(1-\epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n}\epsilon_i \leq C$$



For $\mathbf{0} \leq \epsilon_i \leq \mathbf{1}$:

- $M\left(1-\epsilon_i\right)>0$
- Correct side of hyperplane
- Wrong side of the margin (*violates margin*)
- Pays cost ϵ_i
- Distance $< M$
- *Ex:* 2, 3

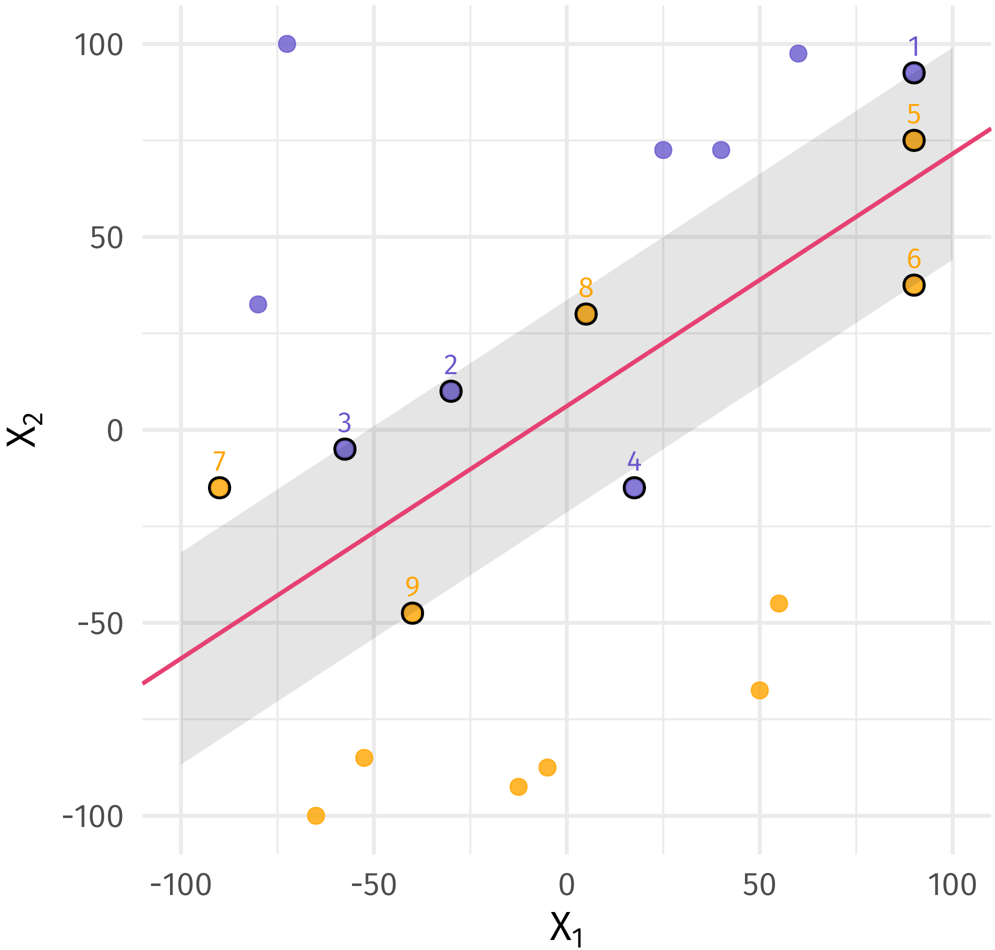
$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M\left(1 - \epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



For $\epsilon_i \geq 1$:

- $M\left(1 - \epsilon_i\right) < 0$
- Wrong side of hyperplane
- Pays cost ϵ_i
- Distance $\lesseqqgtr M$
- *Examples?*

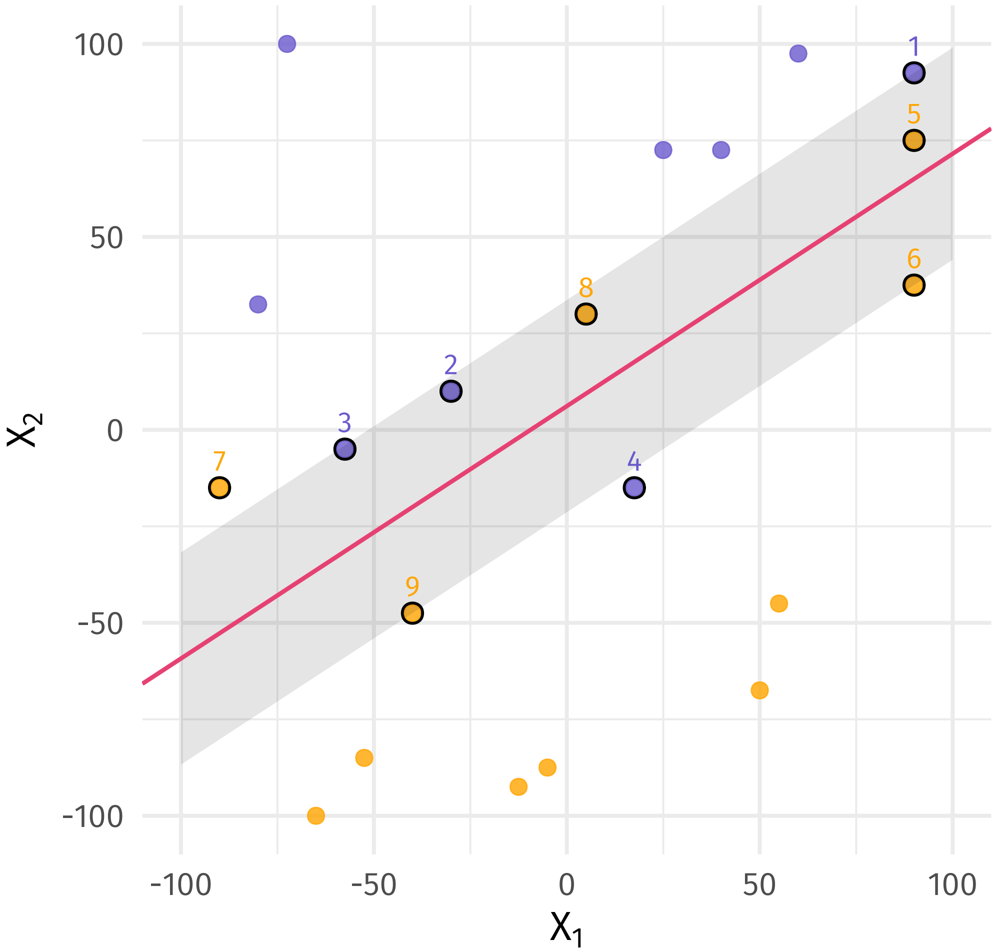
$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M\left(1 - \epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



For $\epsilon_i \geq 1$:

- $M\left(1 - \epsilon_i\right) < 0$
- Wrong side of hyperplane
- Pays cost ϵ_i
- Distance $\lesseqgtr M$
- *Ex:* 4, 5, 7, 8

$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M\left(1 - \epsilon_i\right), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$



Support vectors

- On margin
- Violate margin
- Wrong side of hyperplane

determine the classifier.

Support vector classifier

The tuning parameter C determines how much *slack* we allow.

C is our budget for violating the margin—including observations on the wrong side of the hyperplane.

Case 1: $C = 0$

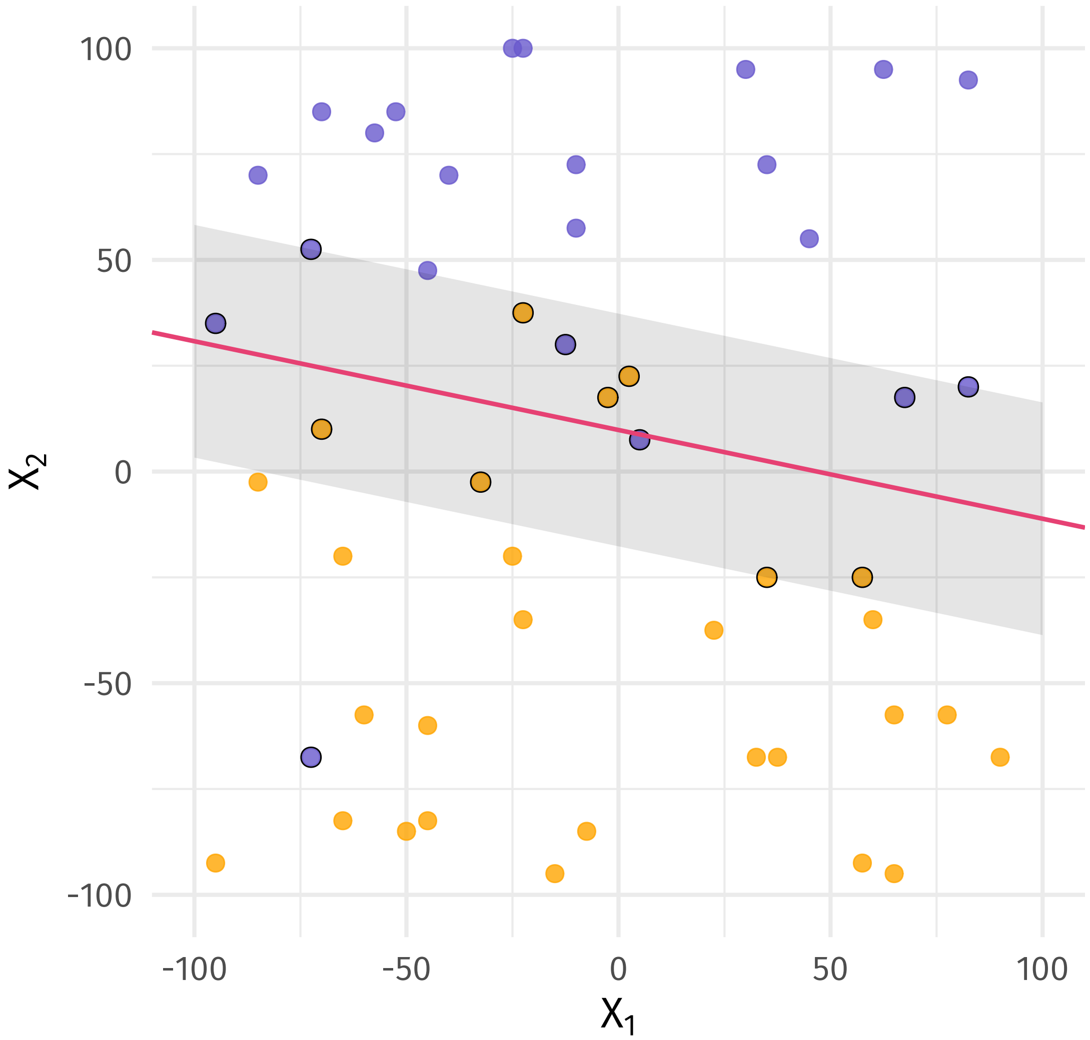
- We allow no violations.
- Maximal margin hyperplane.
- Trains on few obs.

Case 2: $C > 0$

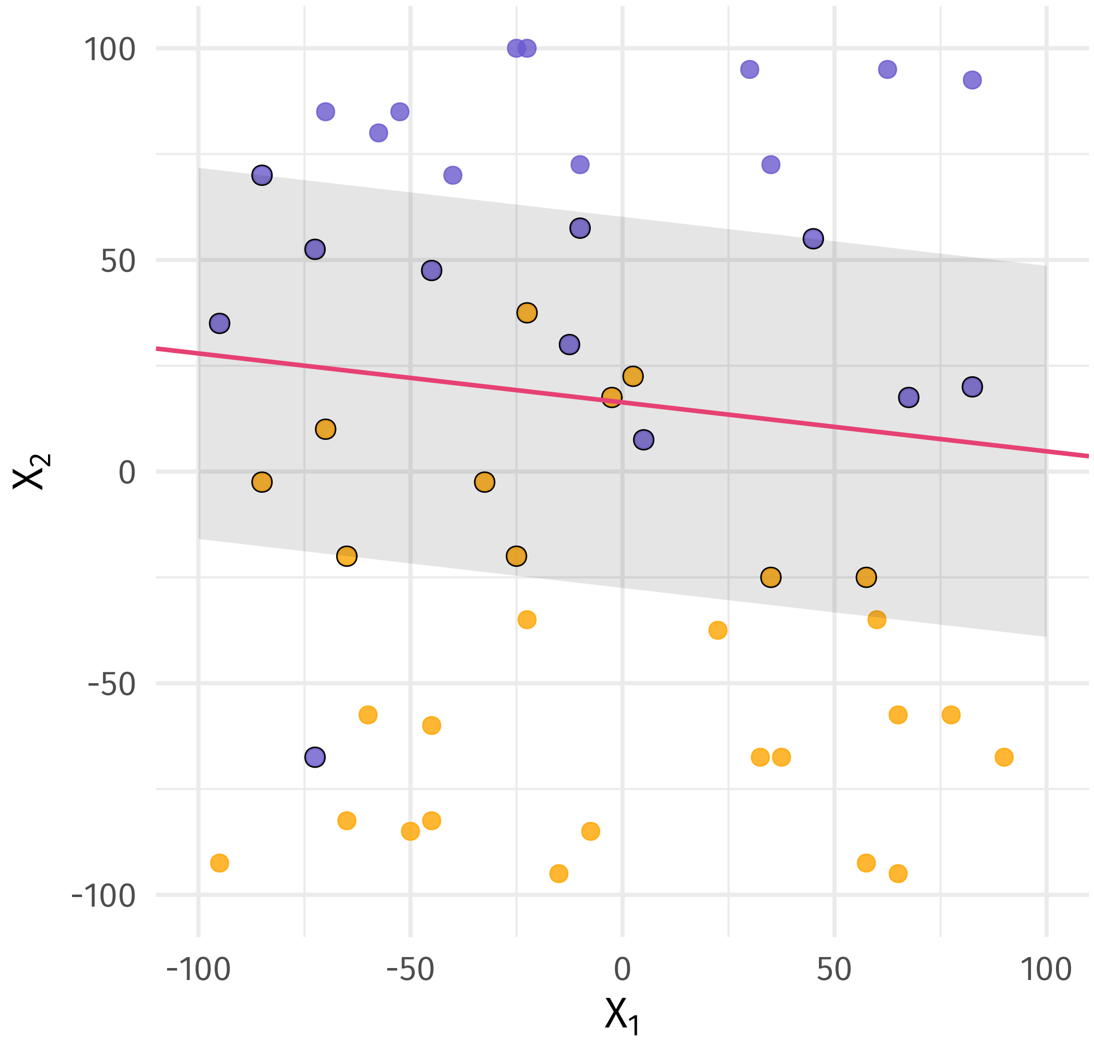
- $\leq C$ violations of hyperplane.
- *Softens* margins
- Larger C uses more obs.

We tune C via CV to balance low bias (low C) and low variance (high C).

Starting with a low budget (C).



Now for a high budget (C).

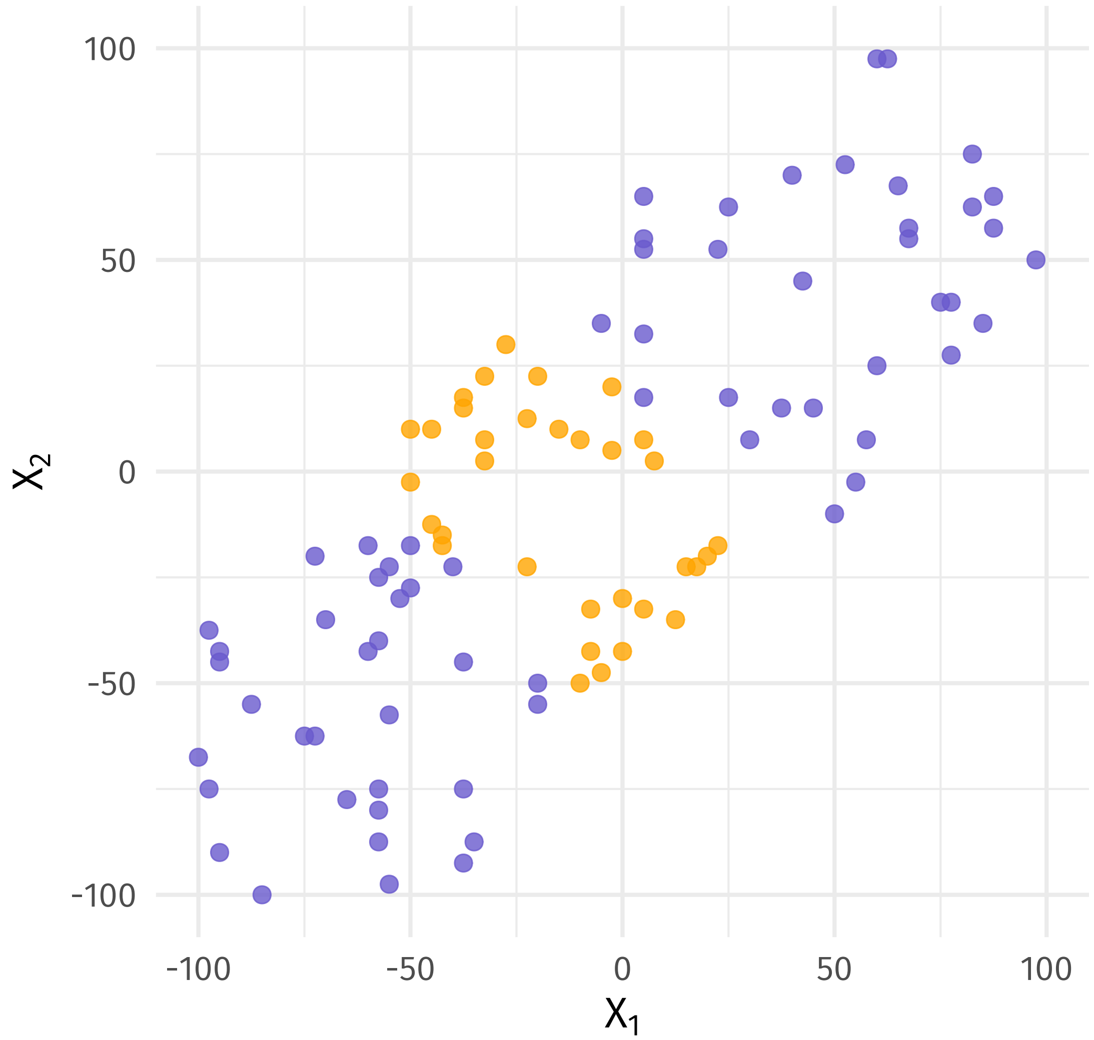


The **support-vector classifier** extends the **maximal-margin classifier**:

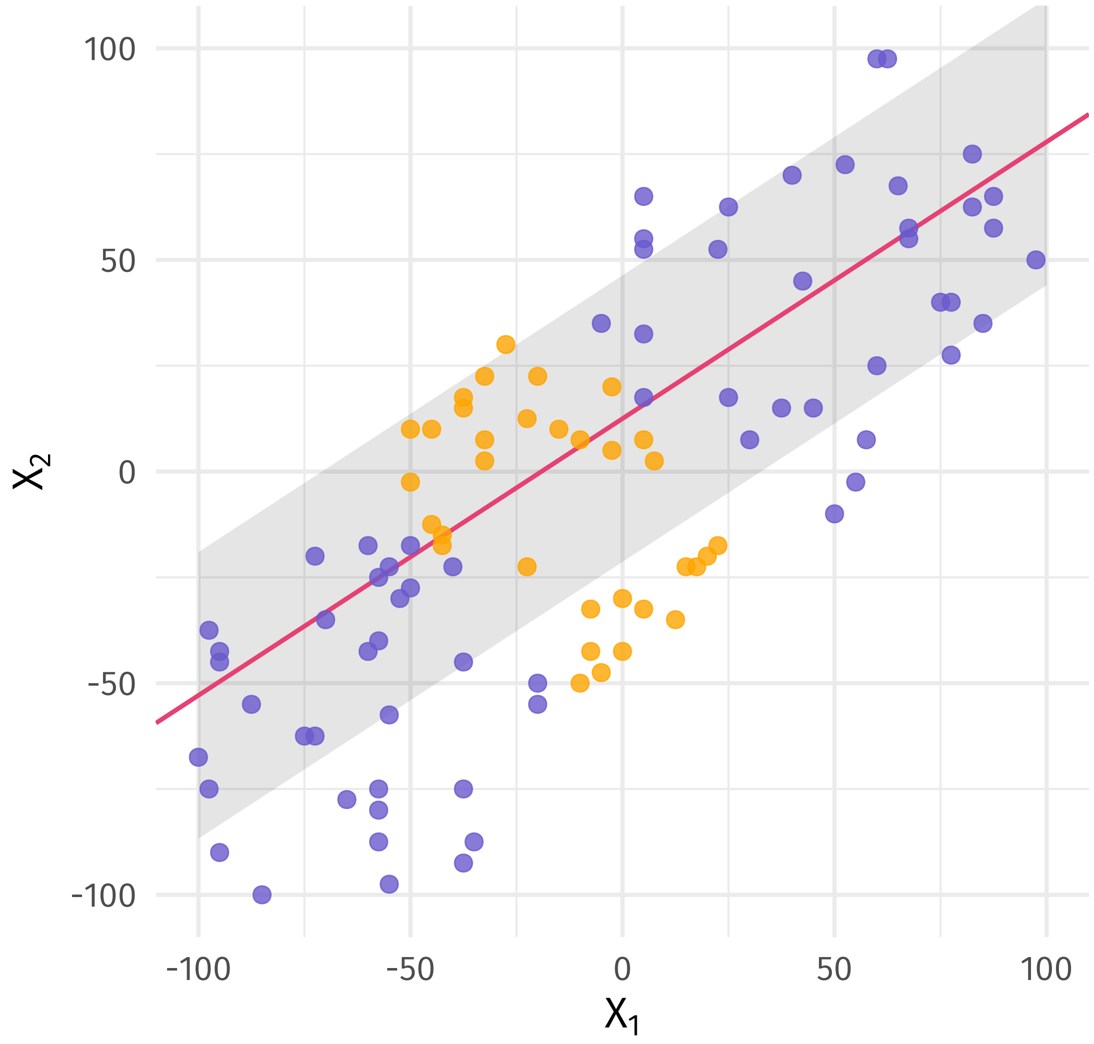
1. Allowing for **misclassification**
   - Observations on the *wrong side of the hyperplane*.
   - Situations where there is no separating hyperplane.
2. Permitting **violations of the margin**.
3. Typically using **more observations** to determine decision boundary.

However, we still are using a (single) linear boundary between our classes.

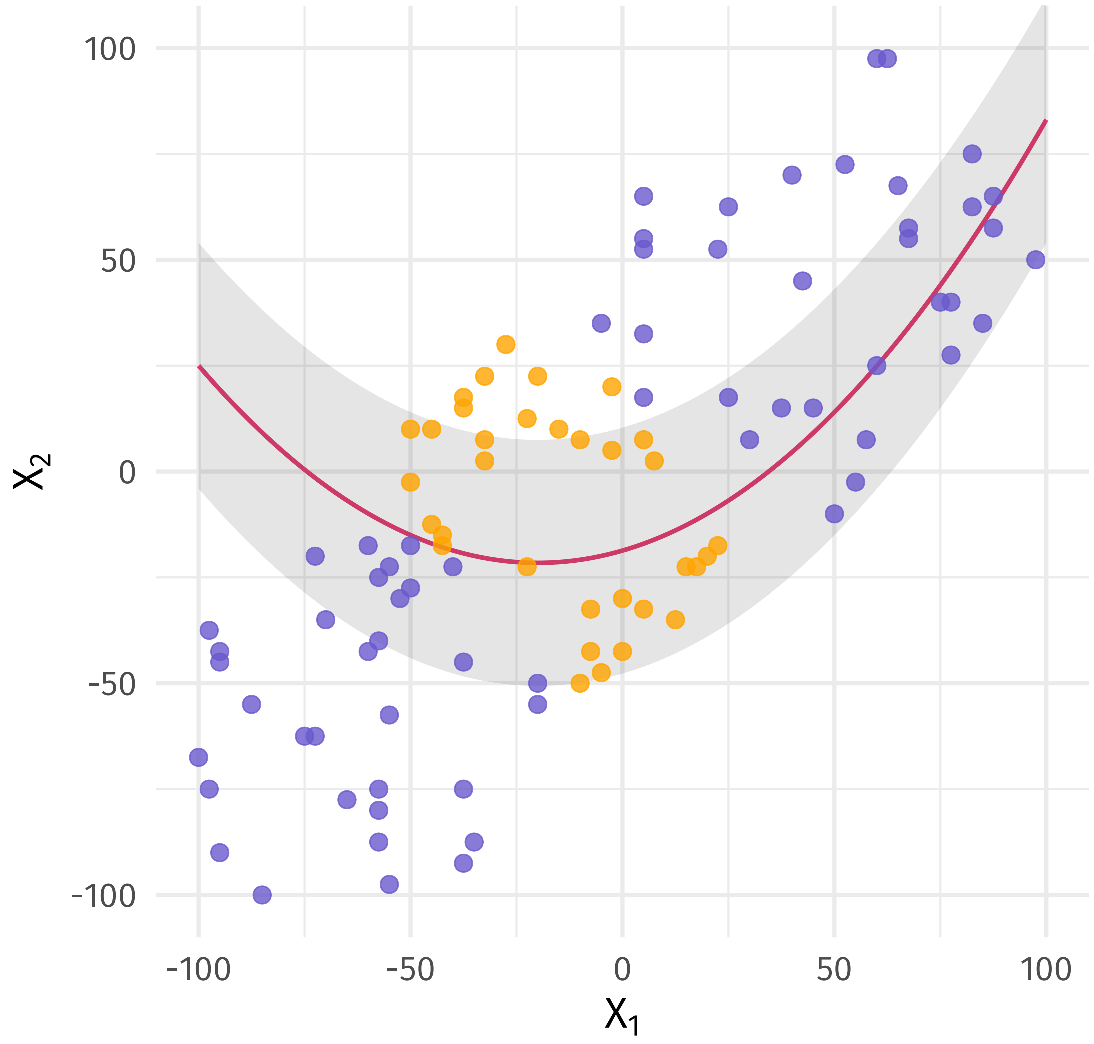
Ex: Some data



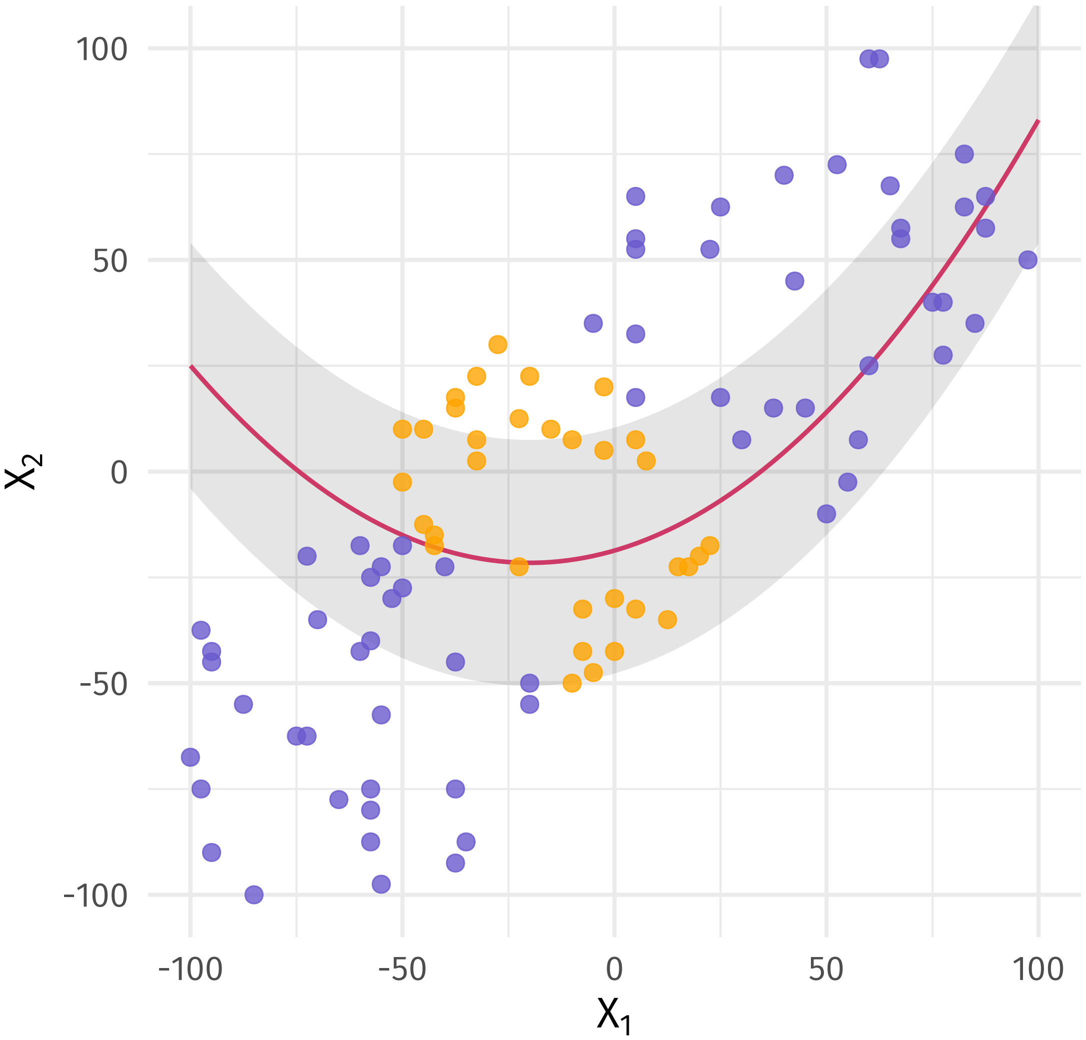
Ex: Some data don't really work with *linear* decision boundaries.



Ex: Some data may have *non-linear* decision boundaries.



Ex: We could probably do even better with more flexibility.



Support vector machines

Flexibility

In the regression setting, we increase our model's flexiblity by adding polynomials in our predictors, *e.g.*, $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2 + \hat{\beta}_3 x_i^3$.

We can apply a very similar idea to our support vector classifier.

Previously: Train the classifier on $X_1, X_2, \ldots, X_p$.

Idea: Train the classifier on $X_1, X_1^2, X_2, X_2^2 \ldots, X_p, X_p^2$ (and so on).

The new classifier has a **linear decision boundary** in the expanded space.

The boundary is going to be **nonlinear** within the original space.

Support vector machines

Introducing

The support vector machine runs with this idea of expanded flexiblity.

(Why stop at quadratic functions—or polynomials?)

Support vector machines **train a support vector classifier** on **expanded feature**† **spaces** that result from applying kernels to the original features.

† feature = predictor

Support vector machines

Dot products

It turns out that solving the support vector classifier only involves the **dot product** of our observations.

The **dot product** of two vectors is defined as

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = a_1 b_1 + a_2 b_2 + \cdots + a_p b_p = \sum_{i=1}^{p} a_i b_i$$

Ex: The dot product of $\boldsymbol{a}$ = (1,2) and $\boldsymbol{b}$ = (3,4) is $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ = 1×3 + 2×4 = 11.

Dot products are often pitched as a measure of two vectors' similarity.

Support vector machines

Dot products and the SVC

We can write the linear support vector classifier as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

and we fit the (n) α_i and β_0 with the training observations' dot products.†

As you might guess, $\alpha_i \neq 0$ only for support-vector obsevations.

† The actually fitting is beyond what we're doing today.

Generalizing

Recall: Linear support vector classifier $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$

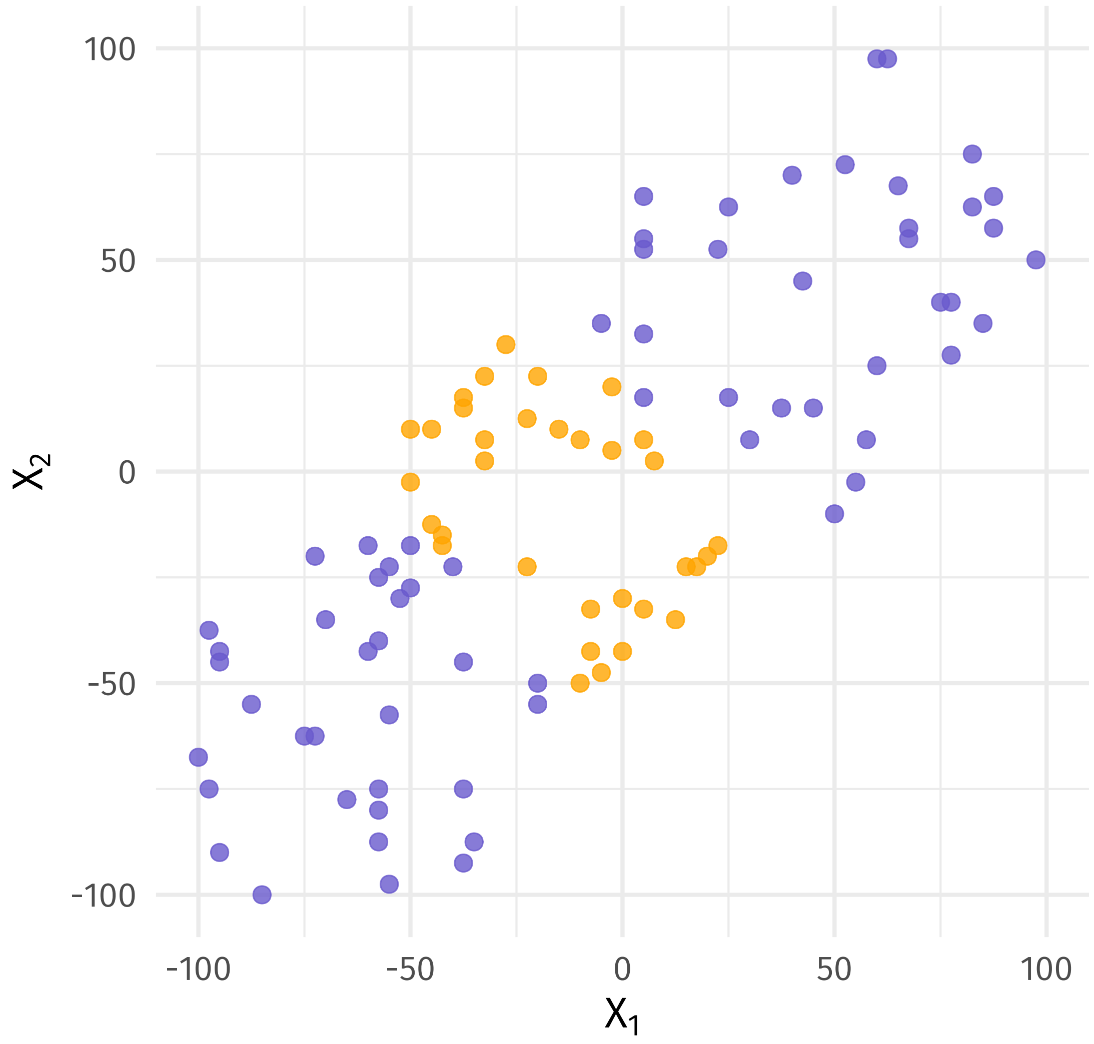
Support vector machines generalize this linear classifier by simply replacing $\langle x, x_i \rangle$ with (non-linear) **kernel functions**† $K(x_i, x_{i'})$.

These magical **kernel functions** are various ways to measure similarity between observations.

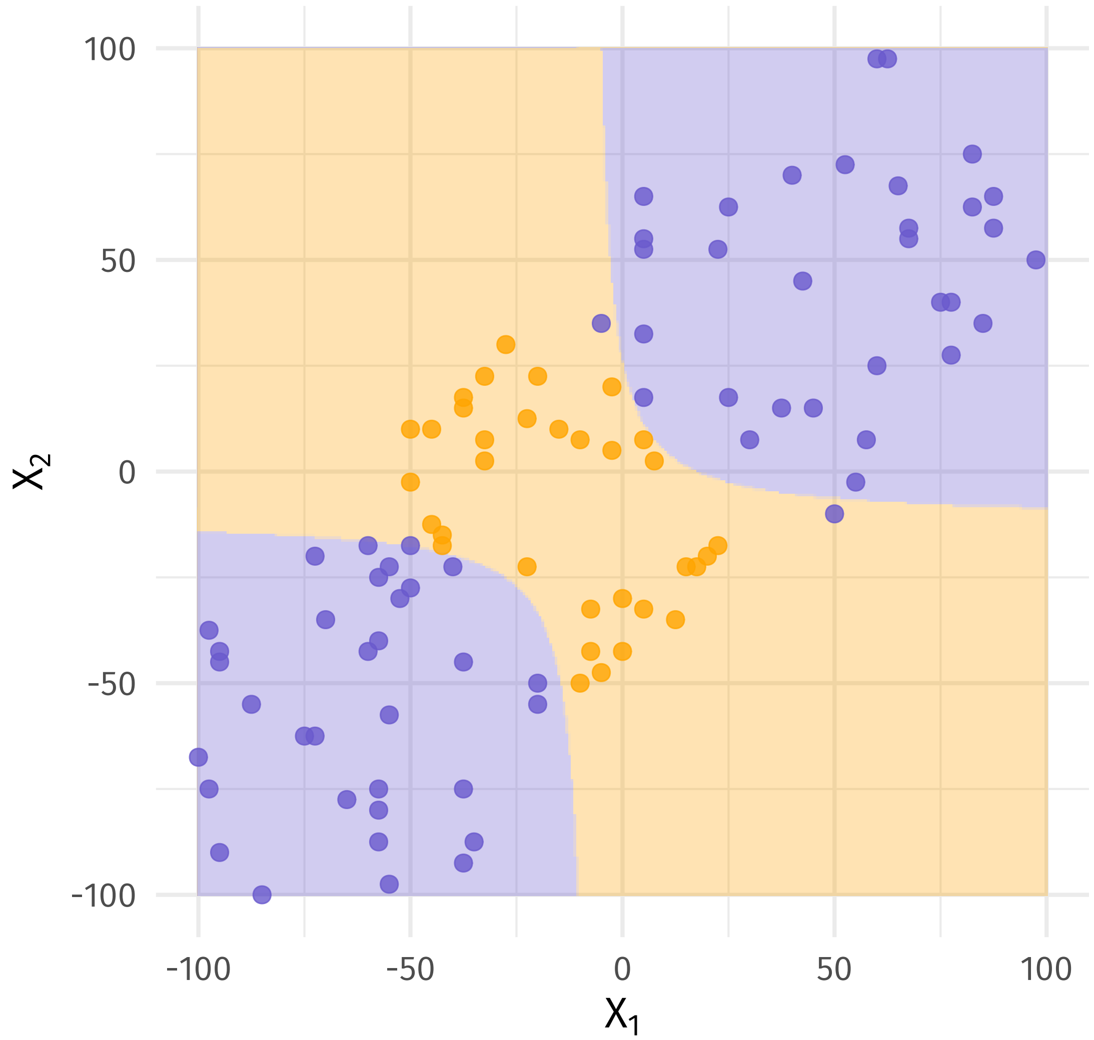
- *Linear kernel:* $K(x_i, x_{i'}) = \sum_{i=1}^{p} x_{i,j} x_{i',j}$ (back to SVC)
- *Polynomial kernel:* $K(x_i, x_{i'}) = \left(1 + \sum_{i=1}^{p} x_{i,j} x_{i',j}\right)^2$
- *Radial kernel:* $K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} \left(x_{i,j} - x_{i',j}\right)^2\right)$

† Or just *kernels*.

Our example data.

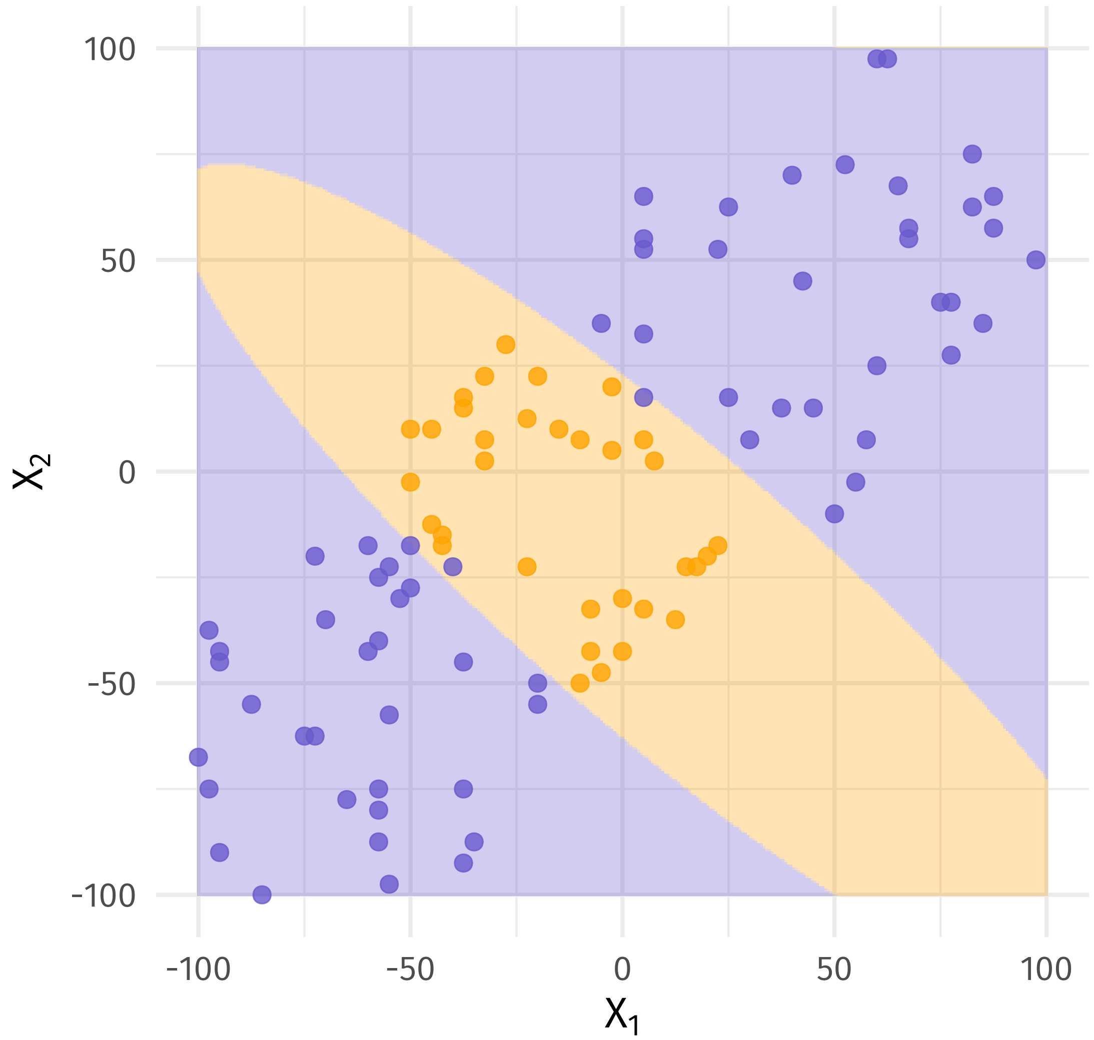


With a linear kernel *plus* and interaction between X_1 and X_2.[†]

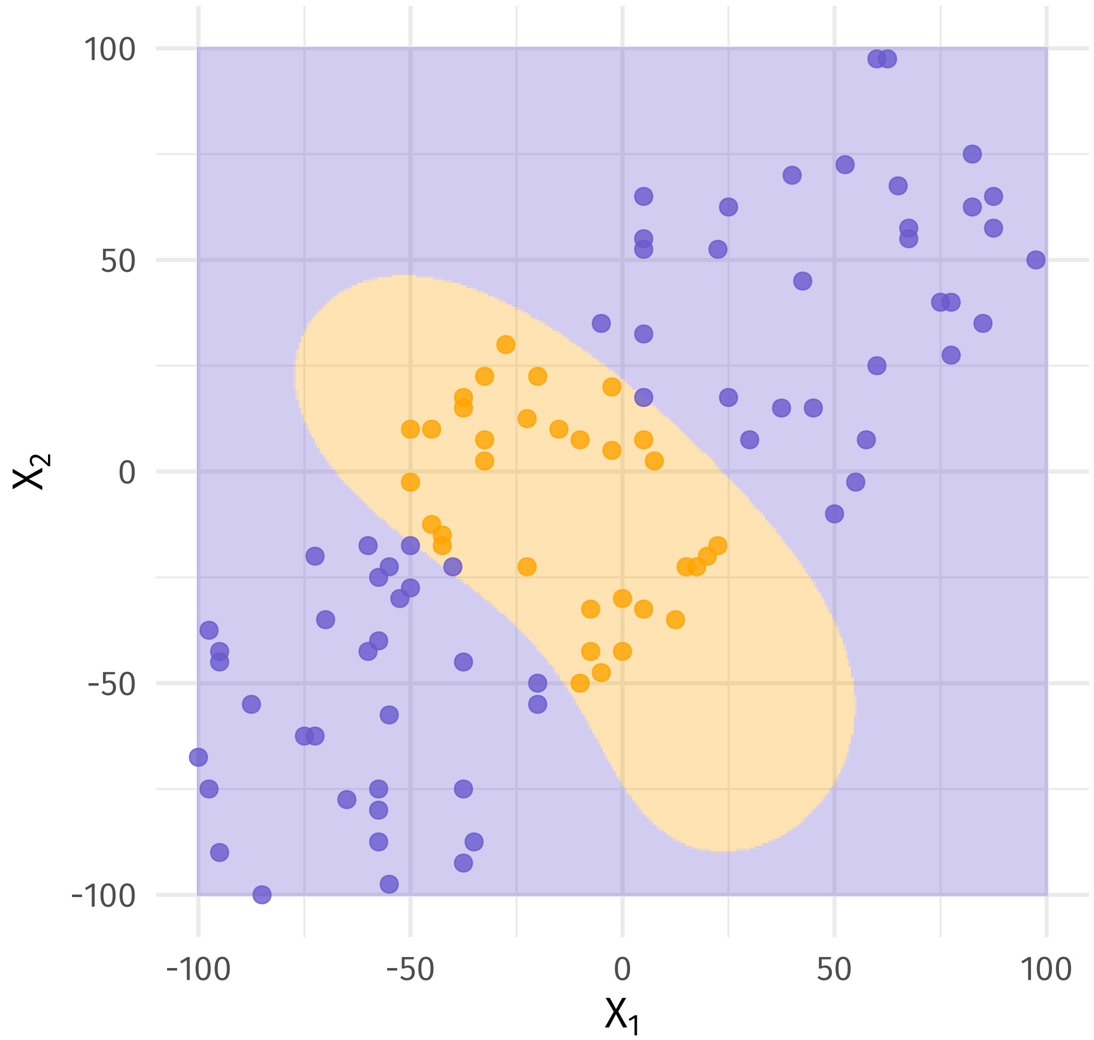


† Exciting!!

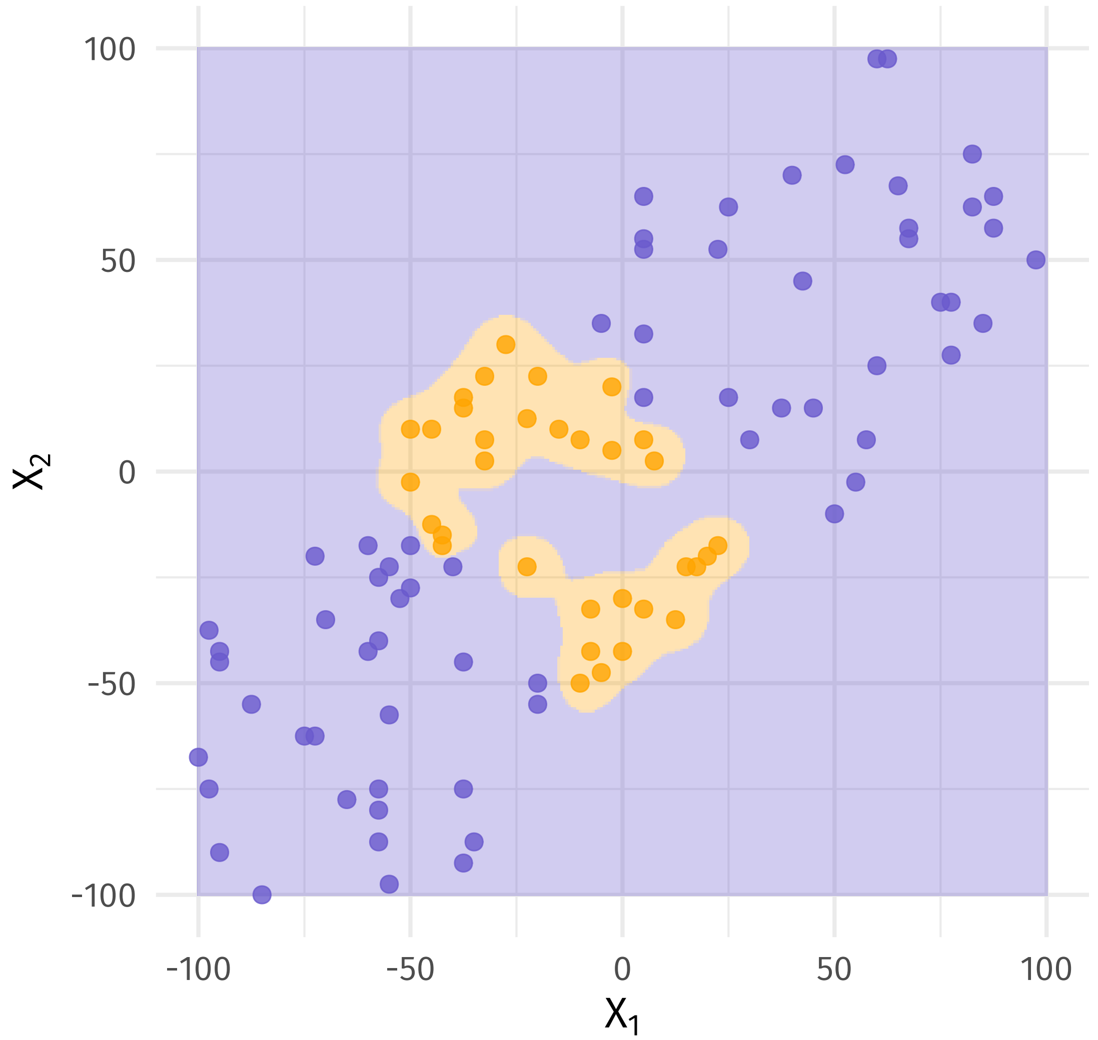
Polynomial kernel (of degree 2).



And now for the radial kernel!



Very small γ forces radial kernel to be *even more* local.



More generalizing

So why make a big deal of kernels? Anyone can transform variables.

While anyone can transform variables, you cannot transform variables to cover all spaces that our kernels cover.

For example, the feature space of the radial kernel is infinite dimensional.†

† And implicit

Support vector machines

In R

As you probably guessed, `parsnip` offers several SVM options.†

- **Linear**††
  `svm_linear(cost)` with `"LiblineaR"` engine

- **Polynomial**
  `svm_linear(cost, degree, scale_factor)` with `"kernlab"` engine

- **Radial**
  `svm_rbf(cost, rbf_sigma, scale_factor)` with `"kernlab"` engine

You can also find more kernels in the actual packages (or other packages).

† `e1071` is another popular R package for SVMs, but it does not work with `tidymodels`.
†† Remember that you can still add interactions with the linear kernel.

Note: Costs have units. You often want to center/scale your variables.

Support vector machines

In R

For a real example let's go back to the heart disease dataset.

- Load the data.
- Add a recipe **with normalization**.
- Define the CV splits.

In R

For a real example let's go back to the heart disease dataset.

```
# Load the dataset
heart_df = read_csv("Heart.csv") %>%
  dplyr::rename(id = X1, HeartDisease = AHD) %>%
  janitor::clean_names()
# Define the recipe w/ imputation, dummies, and normalization
heart_recipe = recipe(heart_disease ~ ., data = heart_df) %>%
  update_role(id, new_role = "id variable") %>%
  step_medianimpute(all_predictors() & all_numeric()) %>%
  step_modeimpute(all_predictors() & all_nominal()) %>%
  step_dummy(all_predictors() & all_nominal()) %>%
  step_normalize(all_predictors())
# Define CV
set.seed(12345)
heart_splits = heart_df %>% vfold_cv(v = 5)
```

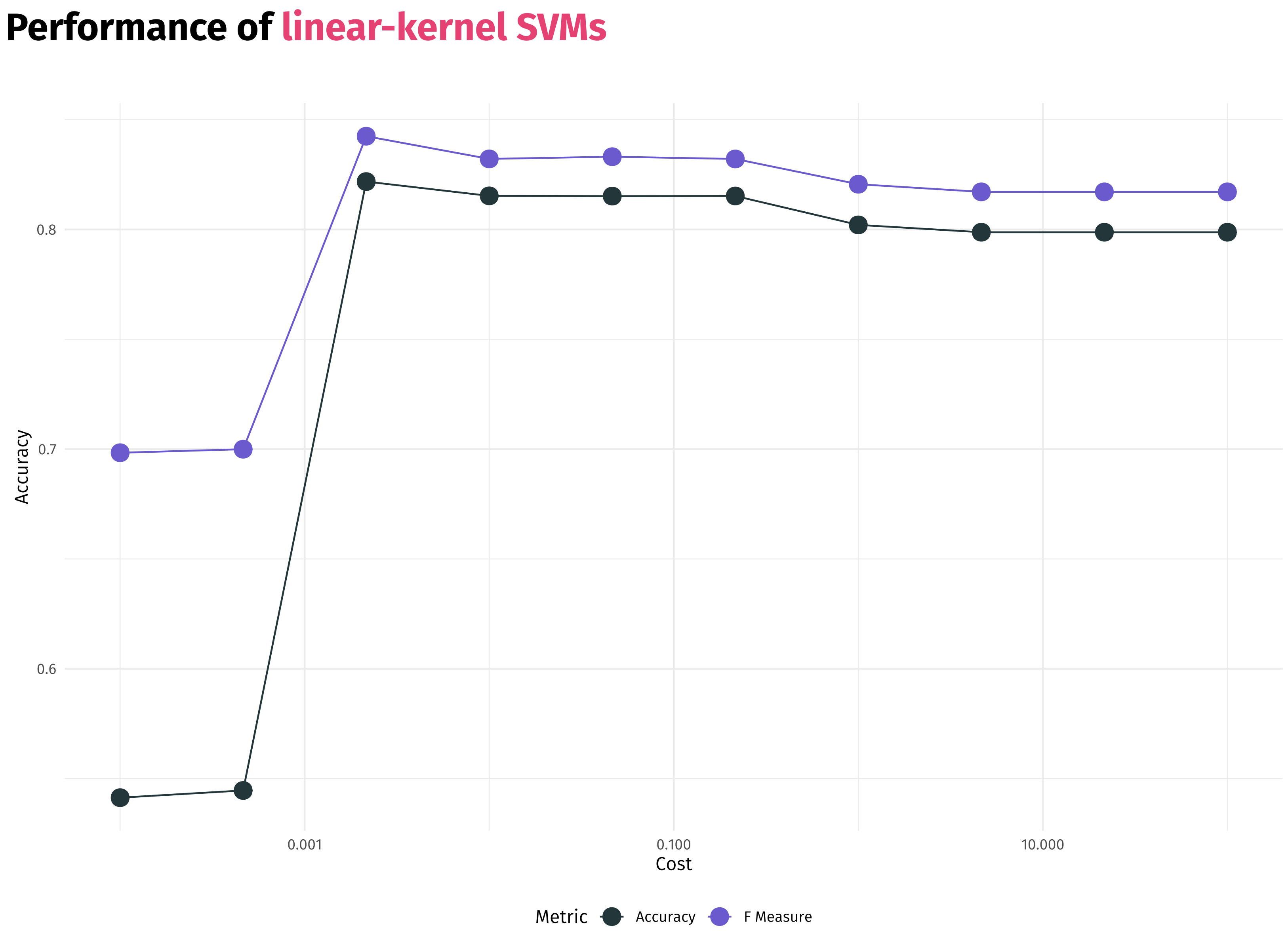
Note The actual linear SVM function `linear_svm()` is under development and isn't working quite right. First-degree polynomial *is* linear.

SVM with linear kernel

Define the model and workflow—tuning C.

```r
# The linear-SVM model
heart_linear = svm_poly(
  mode = "classification",
  cost = tune(),
  degree = 1
) %>% set_engine("kernlab")
# The linear-SVM workflow
wf_linear_svm = workflow() %>%
  add_model(heart_linear) %>%
  add_recipe(heart_recipe)
```

```r
# Tune the linear SVM
tuning_linear_svm = tune_grid(
  wf_linear_svm,
  heart_splits,
  grid = data.frame(
    cost = 10^seq(-4, 2, length = 10)
  ),
  metrics = metric_set(
    f_meas, accuracy
  )
)
```



Performance of linear-kernel SVMs
Accuracy
0.8
0.7
0.6
0.001
0.100
10.000
Cost
Metric
Accuracy
F Measure

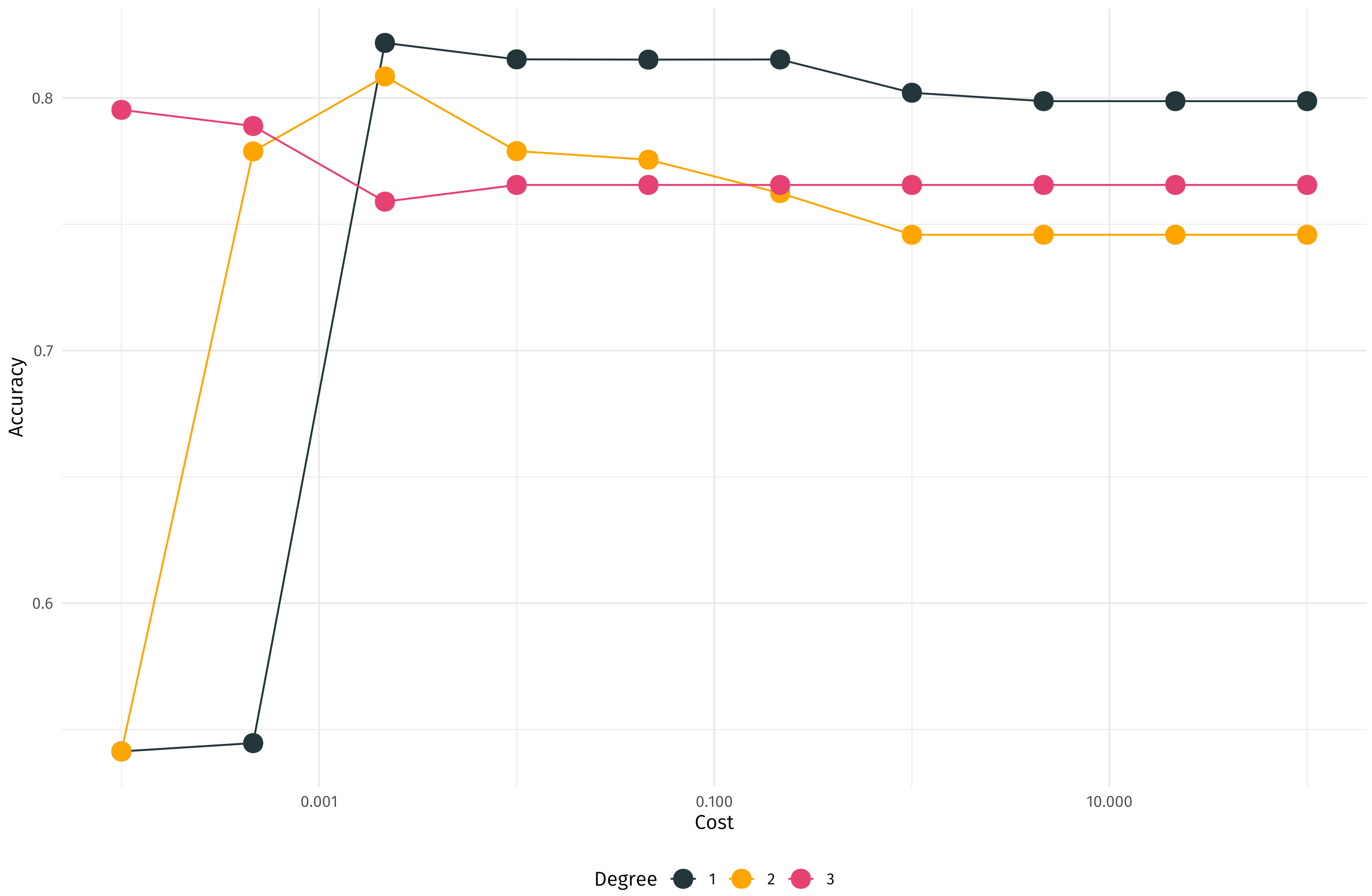
SVM with polynomial kernel

Define the model and workflow—tuning C *and* `degree`.

```r
# The linear-SVM model
heart_poly = svm_poly(
  mode = "classification",
  cost = tune(),
  degree = tune()
) %>% set_engine("kernlab")
# The linear-SVM workflow
wf_poly_svm = workflow() %>%
  add_model(heart_poly) %>%
  add_recipe(heart_recipe)
```

```r
# Tune the linear SVM
tuning_poly_svm = tune_grid(
  wf_poly_svm,
  heart_splits,
  grid = expand_grid(
    cost = 10^seq(-4, 2, length = 10),
    degree = 1:3
  ),
  metrics = metric_set(
    f_meas, accuracy
  )
)
```

Accuracy of polynomial-kernel SVMs



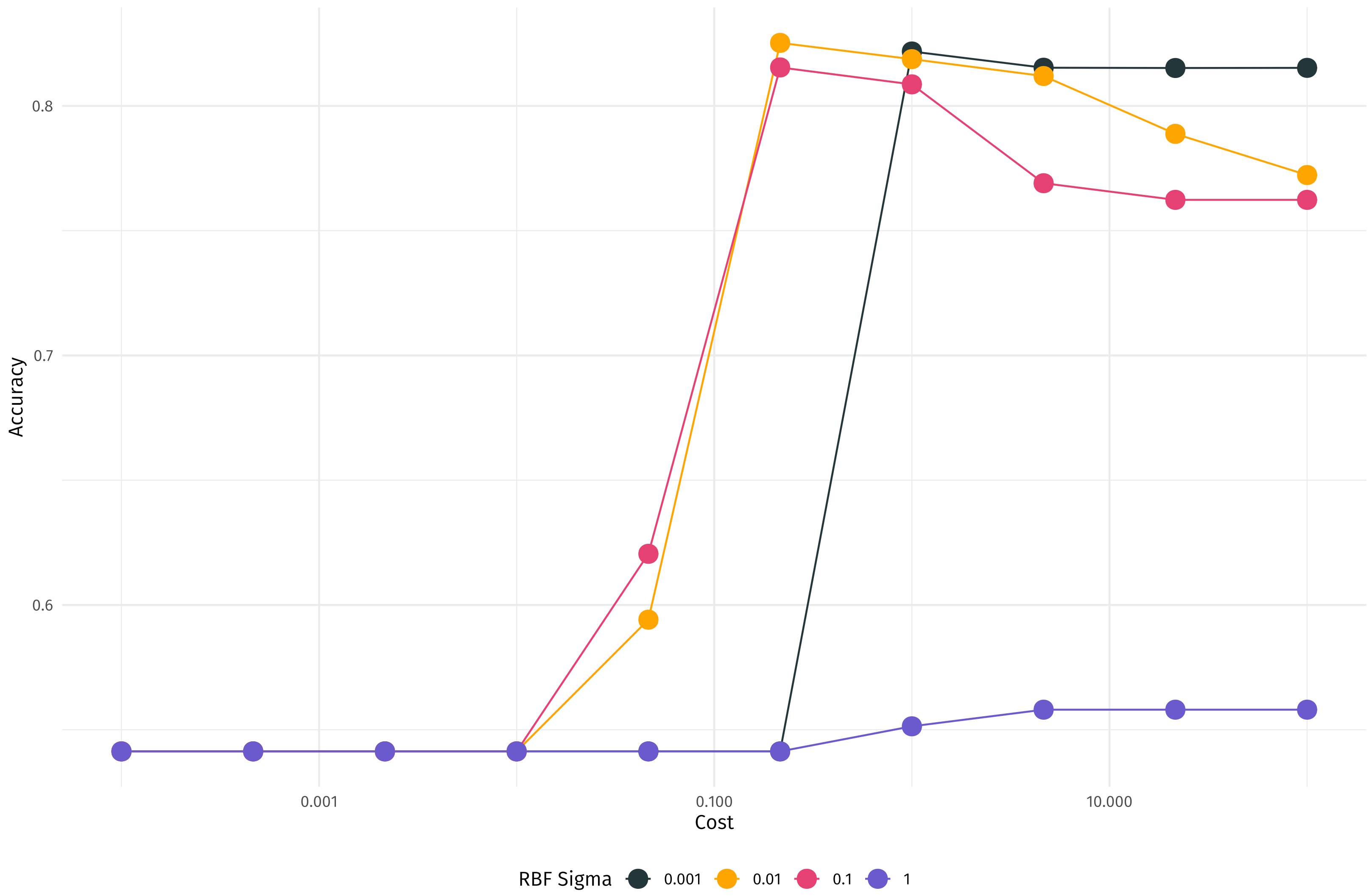
SVM with RBF kernel

Define the model and workflow—tuning C *and* `degree`.

```r
# The linear-SVM model
heart_rbf = svm_rbf(
  mode = "classification",
  cost = tune(),
  rbf_sigma = tune()
) %>% set_engine("kernlab")
# The linear-SVM workflow
wf_rbf_svm = workflow() %>%
  add_model(heart_rbf) %>%
  add_recipe(heart_recipe)
```

```r
# Tune the linear SVM
tuning_rbf_svm = tune_grid(
  wf_rbf_svm,
  heart_splits,
  grid = expand_grid(
    cost = 10^seq(-4, 2, length = 10),
    rbf_sigma = 10^c(-3:0)
  ),
  metrics = metric_set(
    f_meas, accuracy
  )
)
```

Accuracy of RBF-kernel SVMs



Multi-class classification

You will commonly see SVMs applied in settings with $K > 2$ classes.

What can we do? We have options!

One-versus-one classification

- Compares each *pair* of classes, one pair at a time.
- Final prediction comes from the most-common pairwise prediction.

One-versus-all classification

- Fits K unique SVMs—one for each class: k *vs.* not k.
- Predicts the class for which $f_k(x)$ is largest.

More material

Visualizing decision boundaries

- From `scikit-learn`
- From sub-subroutine

The `kernlab` paper (describes kernel parameters)

A Practical Guide to Support Vector Classification

Next:

- Prep: A nice neural-net video
- Fun: A neural-net playground

SV Regression

Extending SVM regression problems

You can extend this SVM idea to regression settings.

Recall OLS regression chooses βs to minimize SSE-based loss.

SV regression chooses βs to minimize loss based upon residuals *larger than some defined magnitude* (think: the margin).

Sources

These notes draw upon

- An Introduction to Statistical Learning (*ISL*)
  James, Witten, Hastie, and Tibshirani

Table of contents