Lecture 11: Machine Learning

Machine Learning Learning Methods for Classification and Model Selection

James Sears*
AFRE 891 SS24
Michigan State University

*Parts of these slides are adapted from **"Prediction and Machine-Learning in Econometrics"** by Ed Rubin, used under **CC BY-NC-SA 4.0**.

Table of Contents

Prologue

Packages we'll use today:

```
if (!require("DT")) remotes::install_github("rstudio/DT")

pacman::p_load(broom, data.table, DT, FNN, glmnet, ggrepel, ISLR, janitor
               magrittr, tidyverse, viridis, tibble)
```

Machine Learning for Classification

Classification Problems

Classification Problems are supervised learning methods for predicting **categorical labels**.

Classification Problems

Classification Problems are supervised learning methods for predicting **categorical labels**.

With categorical variables, MSE doesn't work—*e.g.*,

$$\mathbf{y} - \hat{\mathbf{y}} = \text{(Chihuahua) - (Blueberry muffin)} = \textbf{not math}$$

Instead we will use the **Training error rate:** the share of training predictions that we get wrong

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{y}_i)$$

Classification Example

Let's consider an example: the `Default` dataset from `ISLR`

default	student	balance	income
No	No	939.10	45,519
No	Yes	397.54	22,711
Yes	No	1,511.61	53,507
No	No	301.32	51,540
No	No	878.45	29,562
Yes	No	1,673.49	49,310
No	No	310.13	37,697
No	No	1,272.05	44,896
No	No	887.20	41,641

Classification Example

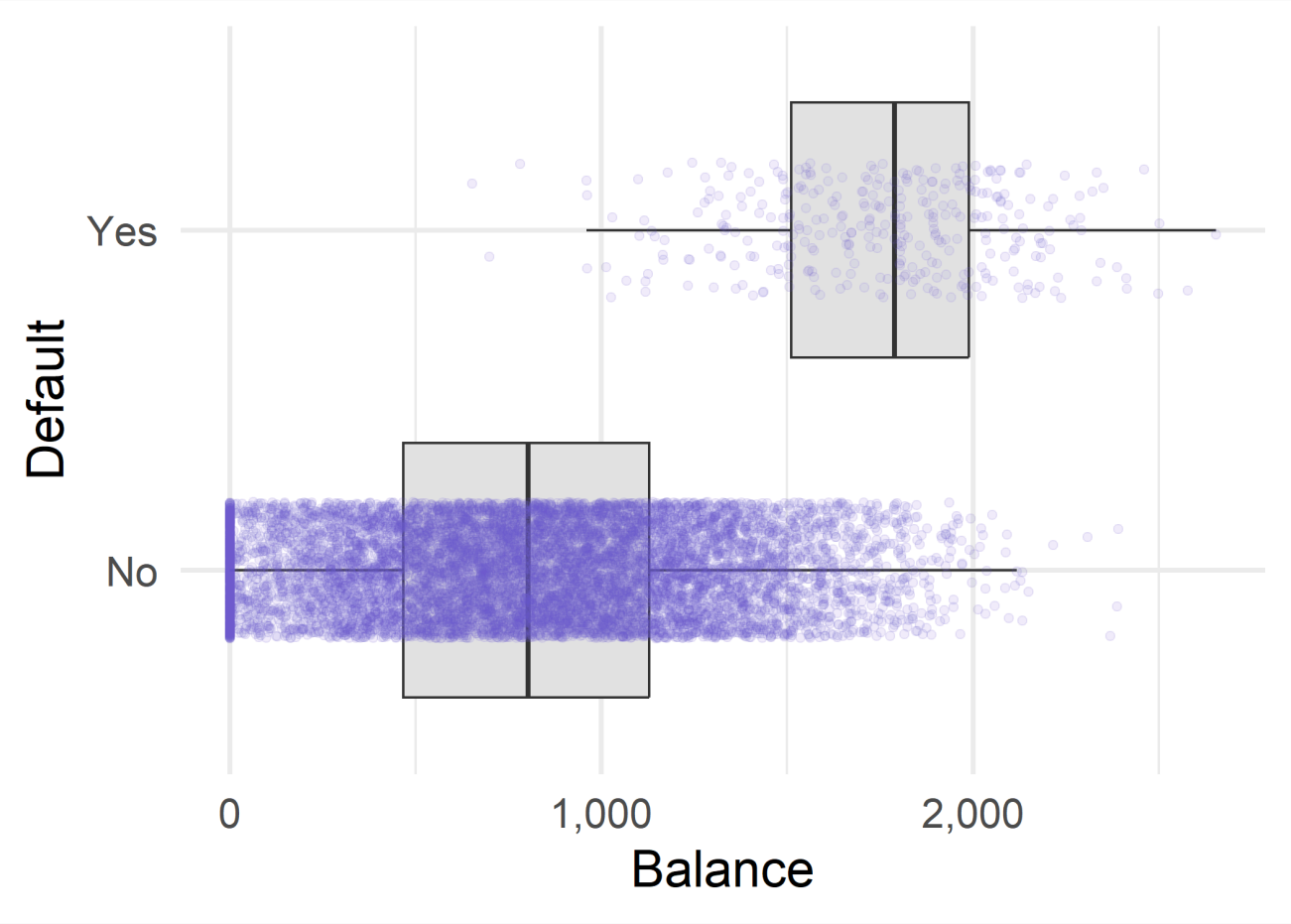
These data contain information on credit card default by ten thousand customers.

Let's first add in a binary indicator of whether or not an individual defaulted:

```
# Clean data
default_df ← ISLR::Default %>% dplyr::mutate(i_default = 1 * (default ==
```

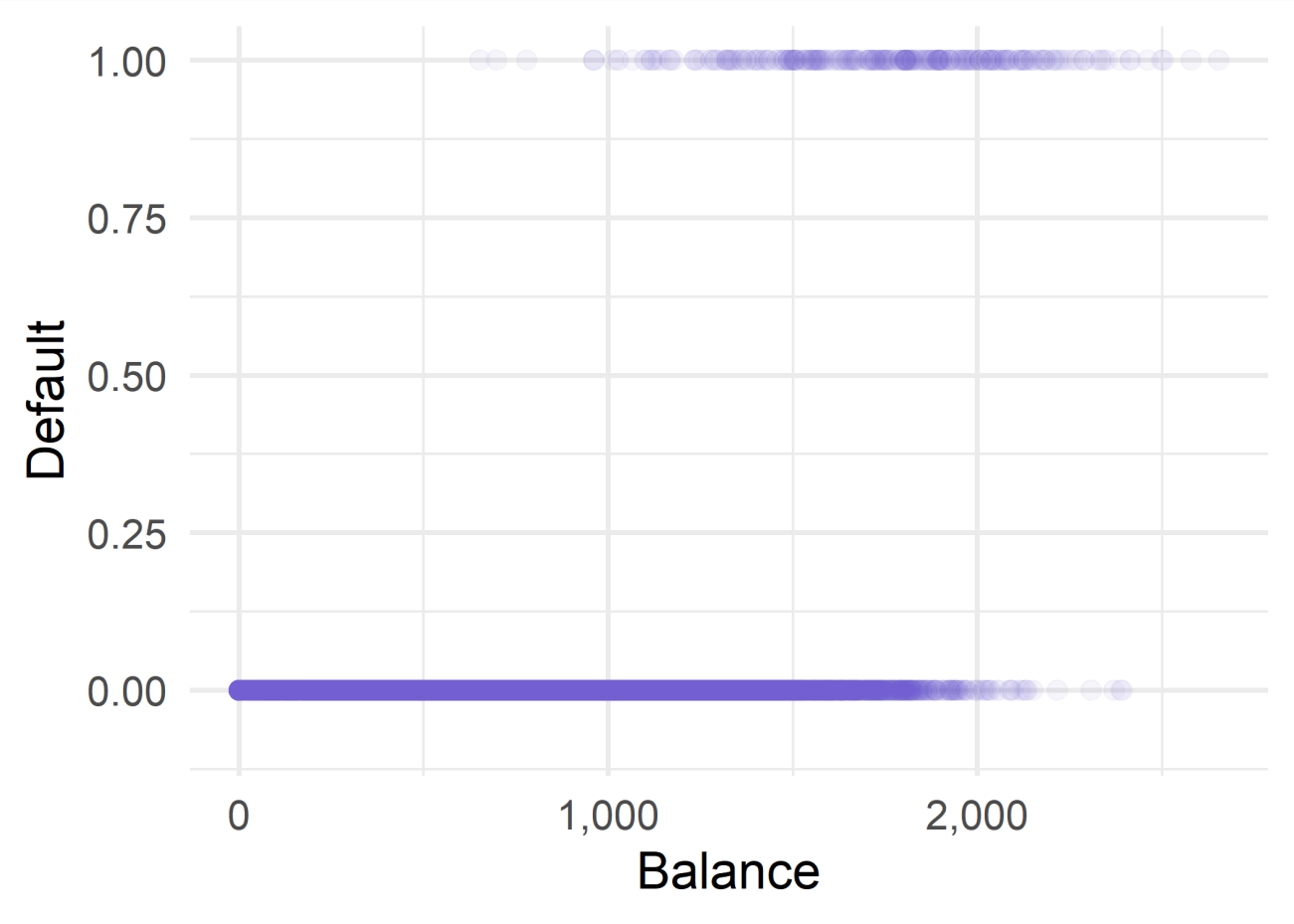
Classification Example

The data: The outcome, default, only takes two values (only 3.3% default).



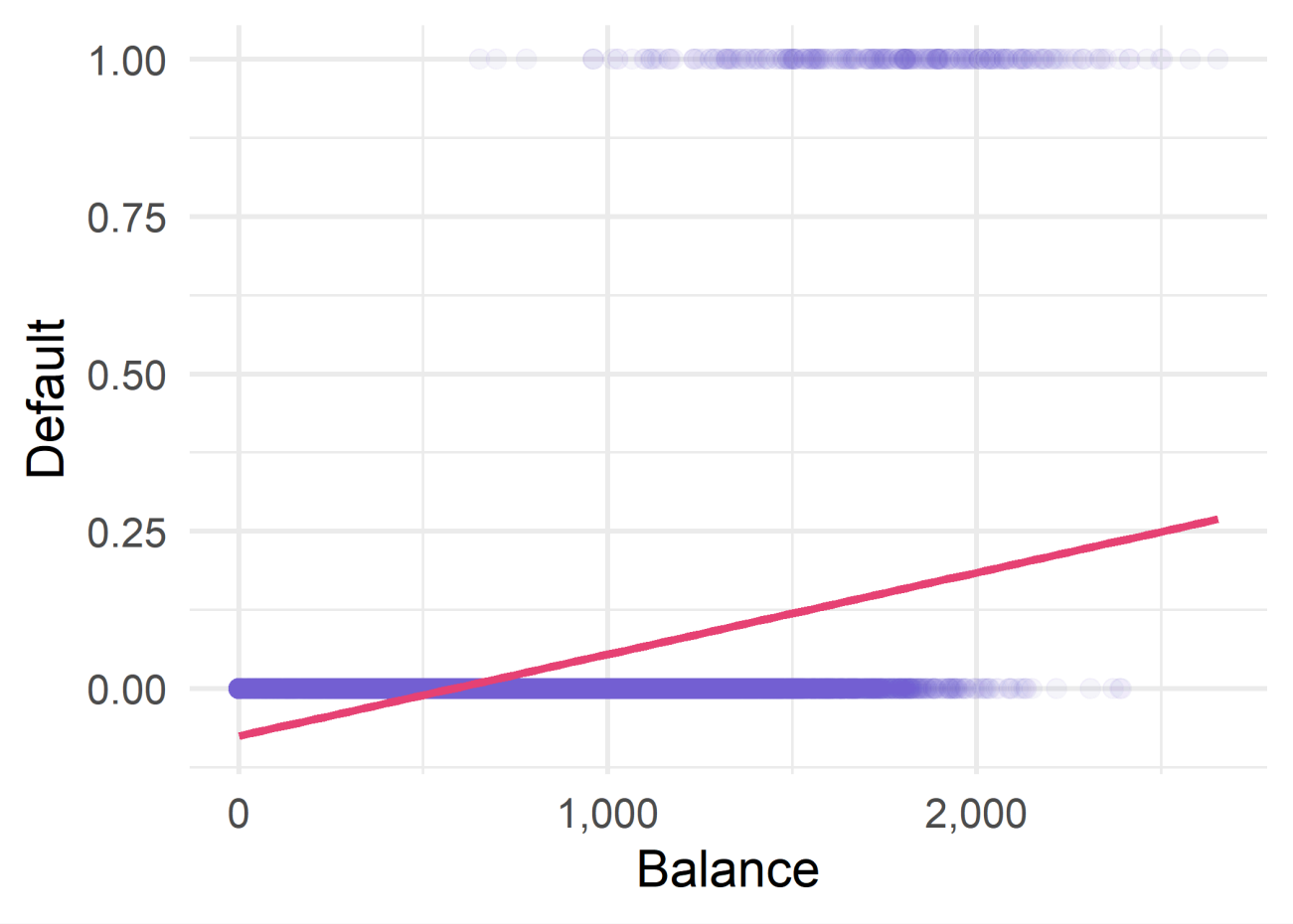
Classification Example

The data: The outcome, default, only takes two values (only 3.3% default).



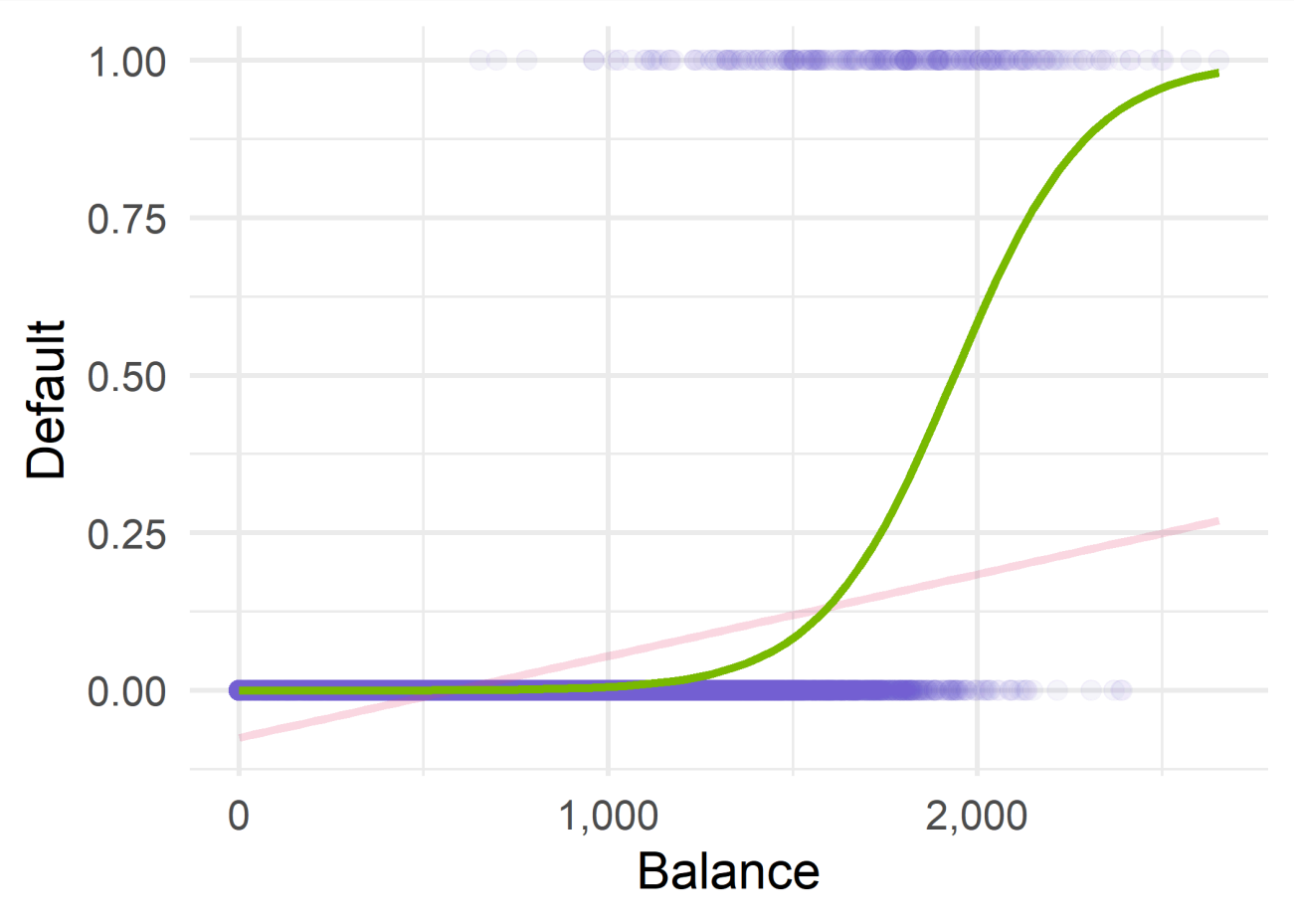
Classification Example

A **linear probability model** struggles with prediction in this one-predictor setting.



Classification Example

Logistic regression *appears* to offer an improvement.



Logistic Regression

How does logistic regression do using *both* balance and income as predictors?

Estimating the logistic regression:

```
est_logistic ← glm(
  i_default ~ balance + income,
  family = "binomial",
  data = default_df
)

# Get predicted Y = 1 if predicted probability is ⩾ 0.5
y_hat_logit ← as.numeric(predict(est_logistic, type = "response") ⩾ 0.5)
```

How Did We Do?

We guessed 97.37% of the observations correctly.

Q: 97.37% is pretty good, right?

A: It depends... Remember that 3.33% of the observations *actually* defaulted.

So we would get 96.67% right by guessing "No" for everyone.[†]

We *did* guess 32.43% of the defaults, which is clearly better than 0%.

Q: How would a machine learning approach perform?

A: Let's see!

† This idea is called the *null classifier*.

Bayes Classifier

The **Bayes Classifier** as the classifier that assigns an observation to its most probable groups, given the values of its predictors, *i.e.*,

Assign obs. i to the class j for which $\mathbf{Pr}(\mathbf{y} = j|\mathbf{X} = \mathbf{x}_0)$ is the largest

The Bayes classifier minimizes the **test error rate**.

- The share of test predictions that we get wrong:

Average $\mathbb{I}(y_0 \neq \hat{y}_0)$ in our **test data**

Bayes Classifier

Thinking back to the chihuahua/blueberry muffin problem, if the calculated conditional probabilities for a given picture were

- Pr(y = "chihuahua" | X = "orange and purple") = 0.3
- Pr(y = "blueberry muffin" | X = "orange and purple") = 0.4
- Pr(y = "squirrel" | X = "orange and purple") = 0.2
- Pr(y = "other" | X = "orange and purple") = 0.1

Then the Bayes classifier says we should predict **"blueberry muffin"**.

Bayes Classifier

More notes on the Bayes classifier:

1. In the **two-class case**, we're basically looking for $\Pr(\mathbf{y} = j \mid \mathbf{X} = x_0) > 0.5$ for one class.

2. The **Bayes decision boundary** is the point where the **probability is equal** between the most likely groups (*i.e.*, exactly 50% for two groups).

3. The Bayes classifier produces the lowest possible **test error rate**, which is called the **Bayes error rate**.

4. **Challenge:** the probabilities $\Pr(\mathbf{y} = j \mid \mathbf{X} = x_o)$ that the Bayes classifier relies upon are **unknown**. We have to **estimate them**.

K-Nearest Neighbors

One non-parametric way to estimate these unknown conditional probabilities: **K-nearest neighbors (KNN)**.

K-nearest neighbors (KNN) simply assigns a category based upon the nearest K neighbors votes (their values).

More formally: Using the K closest neighbors† to test observation $\mathbf{x_0}$, we calculate the share of the observations whose class equals j,

$$\hat{\Pr}(\mathbf{y} = j \mid \mathbf{X} = \mathbf{x_0}) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathbb{I}(\mathbf{y}_i = j)$$

These shares are our estimates for the unknown conditional probabilities.

We then assign observation $\mathbf{x_0}$ to the class with the highest probability.

† In $\mathbf{X}$ space.

K-Nearest Neighbors

We can use the `knn.reg()` function from the **FNN** package to obtain these predicted classifications for our defaulters.

First, we'll randomly sample 50% of our data as a **training** sample.

```
# add a row identifier into the default df
default_df ← mutate(default_df, id = row_number())

# randomly sample 50% of the data for training
train_df ← slice_sample(default_df, n = nrow(default_df)/2)
```

K Nearest Neighbors

Running KNN for an initial choice of K = 5, using balance and income as predictors, with half of the data used for testing

```
knn_5 ← knn.reg(
  train = select(train_df, balance, income) %>% as.matrix(),
  test = select(default_df, balance, income) %>% as.matrix(),
  #train = train_df$balance %>% as.matrix(),
  #test = test_df$balance %>% as.matrix(),
  y = train_df$i_default,
  k = 5
)
```

K Nearest Neighbors

How many defaults did we correctly predict?

```
default_df ← mutate(default_df,
                    phat_5 = knn_5$pred,
                    yhat = (phat_5 > 0.5) %>% as.numeric(),
                    correct = (i_default == yhat) %>% as.numeric(),
                    training = (row_number() %in% train_df$id) %>% as.nu
```

- Overall: 97.08%
- Correct Default: 20.72%
- Correct Non-Default: 99.71%

All told, slightly *worse* than the logistic regression.

Choice of K

The choice of K is very important—ranging from super flexible to inflexible.

- When K is **low**, the decision boundary becomes **overly flexible**, finding patterns in the data that don't match truth
- When K is **high**, things become **too rigid** and the decision boundary approaches linearity

Sensitivity to Choice of K

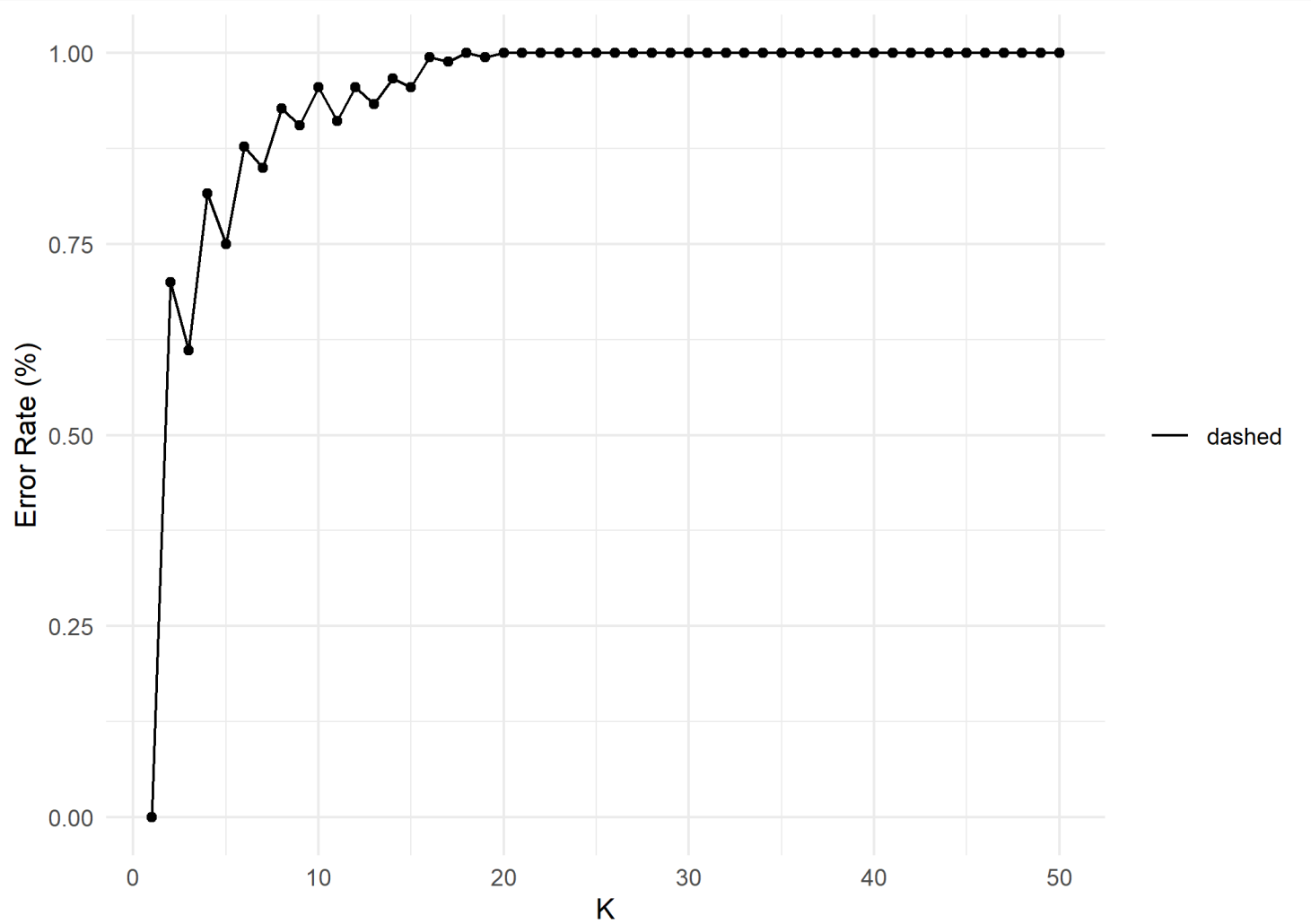
```
# function to estimate for a given k
est_bayes ← function(k_choice,df){
  knn ← knn.reg(
  train = select(train_df, balance, income) %>% as.matrix(),
  test = select(df, balance, income) %>% as.matrix(),
  #train = train_df$balance %>% as.matrix(),
  #test = test_df$balance %>% as.matrix(),
  y = train_df$i_default,
  k = k_choice
)
  df ← mutate(df, phat = knn$pred,
                   yhat = (phat > 0.5) %>% as.numeric(),
                   correct = (i_default == yhat) %>% as.numeric())

  # Error rate in training data
  error_train ← filter(df, training ==1, default == "Yes") %>% mutate(error = 1-correct, .k
  # Error rate in test data
  error_test ← filter(df, training ==0, default == "Yes") %>% mutate(error = 1-correct, .ke

  return(data.frame(k = k_choice,
                    error_train = error_train,
                    error_test = error_test))
}
# running over k in 1:50
bayes_k_range ← map(1:50, est_bayes, df = default_df) %>% list_rbind()
```

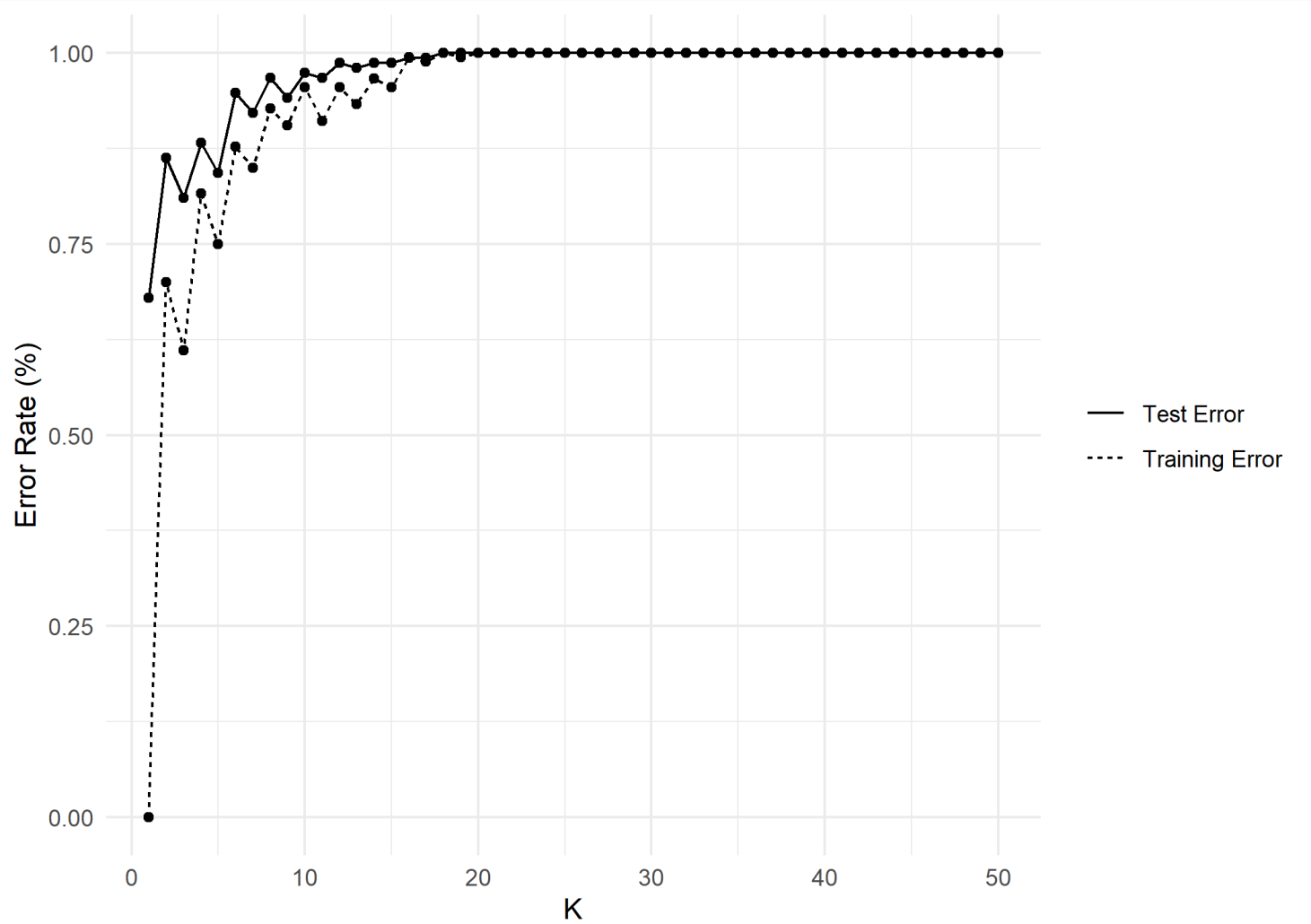
KNN Error Rates

Visualizing the error rates as a function of K: training



KNN Error Rates

Visualizing the error rates as a function of K: training + test



Bayes Decision Boundary

Recall that the **Bayes decision boundary** is the point where the **probability is equal** between the most likely groups (*i.e.*, exactly 50% for two groups)

Let's visualize it with a simpler example on a simulated dataset.

Bayes Decision Boundary

First, generate a random dataset with a regular grid (useful for plotting later)

```
# Generate data
set.seed(1234)
n_b ← 70
bayes_gen ← tibble(
  x1 = runif(n_b, 10, 90),
  x2 = x1 + rnorm(n_b, sd = 30),
  y = (x1 - 0.9 * x2 + rnorm(10) > 0) %>% as.numeric()
)
bayes_truth ← expand.grid(x1 = 1:100, x2 = 1:100) %>% as_tibble()
```

Bayes Decision Boundary

Running KNN for $k = 5$ on "truth" and adding the predictions to the data:

```
est_knn ← knn.reg(
  train = bayes_gen[,c("x1", "x2")],
  test = bayes_truth,
  y = bayes_gen$y,
  k = 5
)
bayes_truth$p ← est_knn$pred
bayes_truth$y ← as.numeric(est_knn$pred > 0.5)
```

Bayes Decision Boundary

Sampling data points and estimating KNN for $k = 5$ for two training data samples:

```
# Sample data points
bayes_sample ← sample_n(bayes_truth, size = 100) %>%
  mutate(y = rbernoulli(n = 100, p = p) * 1)
bayes_sample2 ← sample_n(bayes_truth, size = 100) %>%
  mutate(y = rbernoulli(n = 100, p = p) * 1)
# Train kNN
est_boundary ← knn.reg(
  train = bayes_sample[,c("x1", "x2")],
  test = bayes_truth[,c("x1", "x2")],
  y = bayes_sample$y,
  k = 5
)
est_boundary2 ← knn.reg(
  train = bayes_sample2[,c("x1", "x2")],
  test = bayes_truth[,c("x1", "x2")],
  y = bayes_sample2$y,
  k = 5
)
```

Bayes Decision Boundary

Let's also estimate for $k = 1$ and $k = 60$ on the first sample:

```
est_boundary_k1 ← knn.reg(
  train = bayes_sample[,c("x1", "x2")],
  test = bayes_truth[,c("x1", "x2")],
  y = bayes_sample$y,
  k = 1
)
est_boundary_k60 ← knn.reg(
  train = bayes_sample[,c("x1", "x2")],
  test = bayes_truth[,c("x1", "x2")],
  y = bayes_sample$y,
  k = 60
)
```

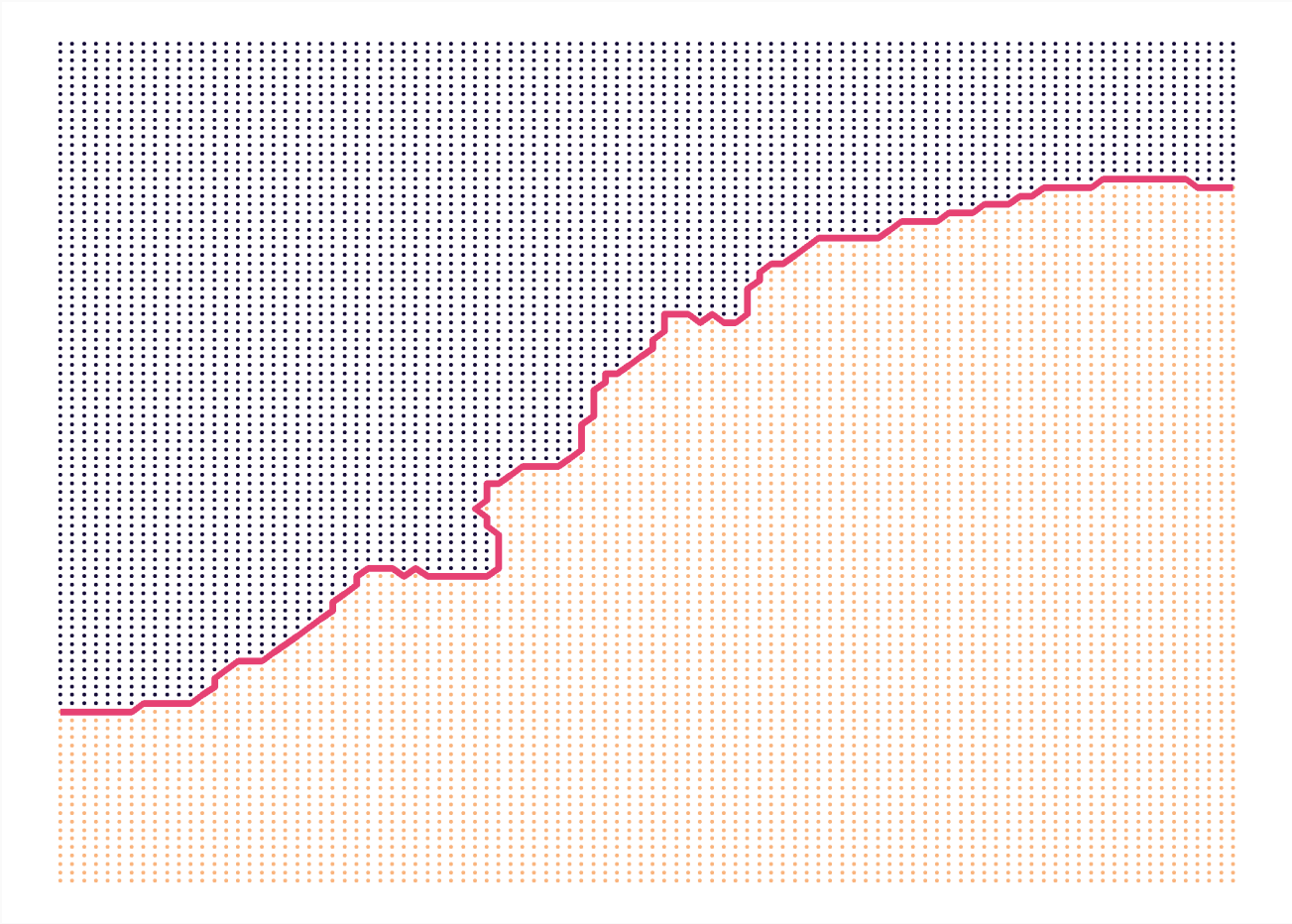
Bayes Decision Boundary

Adding estimates to the full dataset:

```
bayes_truth ← bayes_truth %>%
  mutate(
    y_hat = as.numeric(est_boundary$pred > 0.5),
    y_hat2 = as.numeric(est_boundary2$pred > 0.5),
    y_hat_k1 = as.numeric(est_boundary_k1$pred > 0.5),
    y_hat_k60 = as.numeric(est_boundary_k60$pred > 0.5)
  )
```

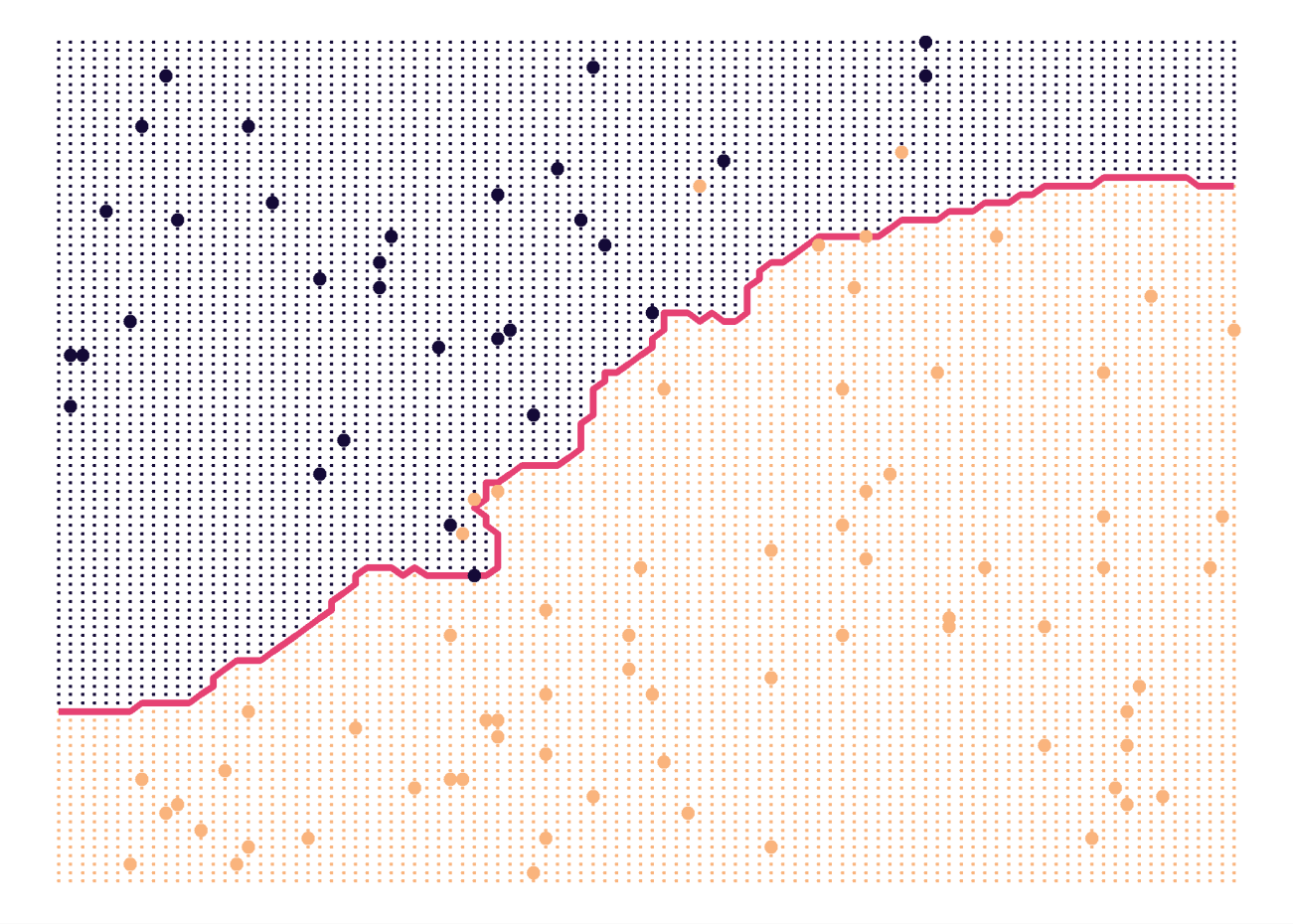
Bayes Decision Boundary

The **Bayes decision boundary** between classes A and B



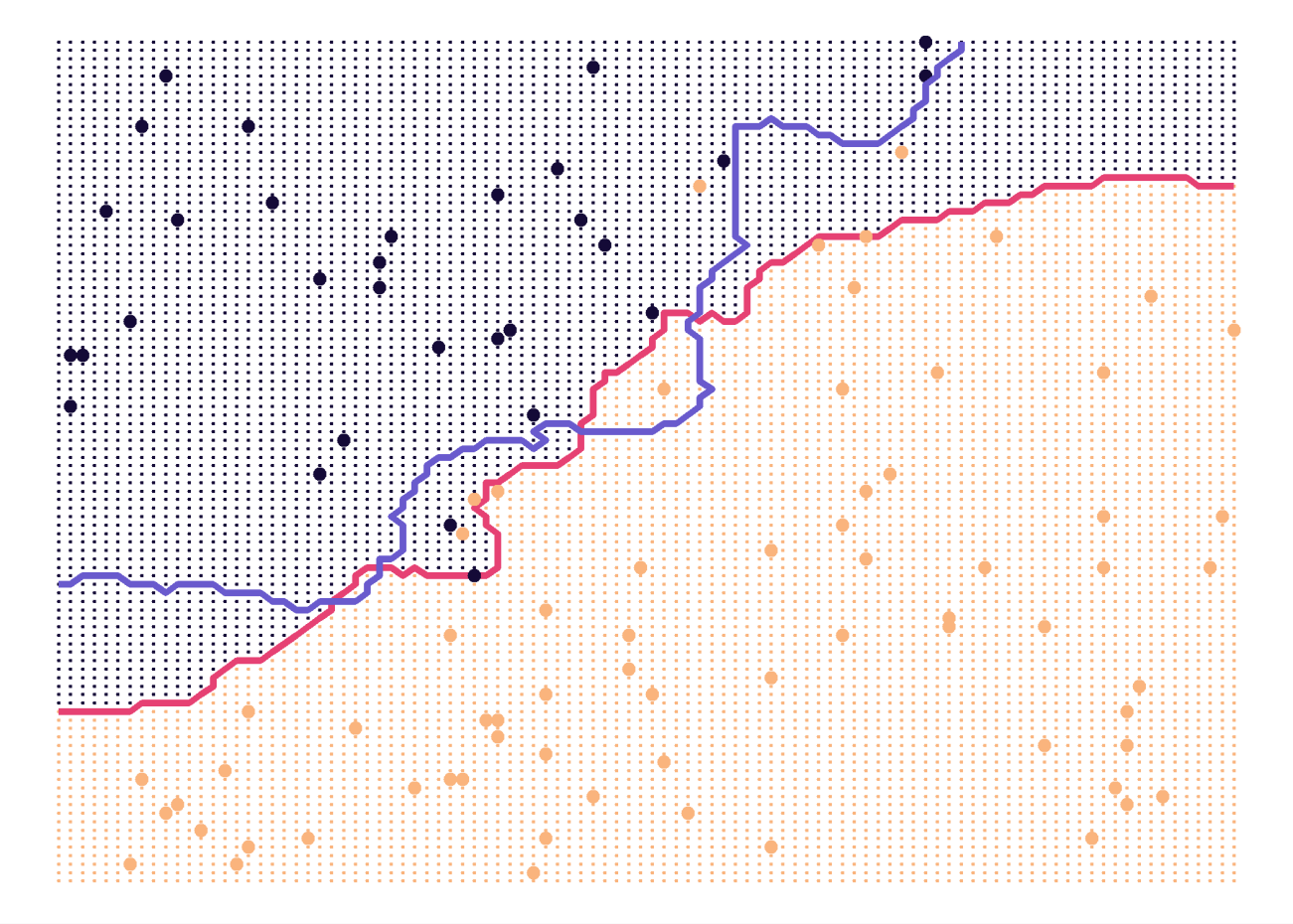
Bayes Decision Boundary

Now we sample...



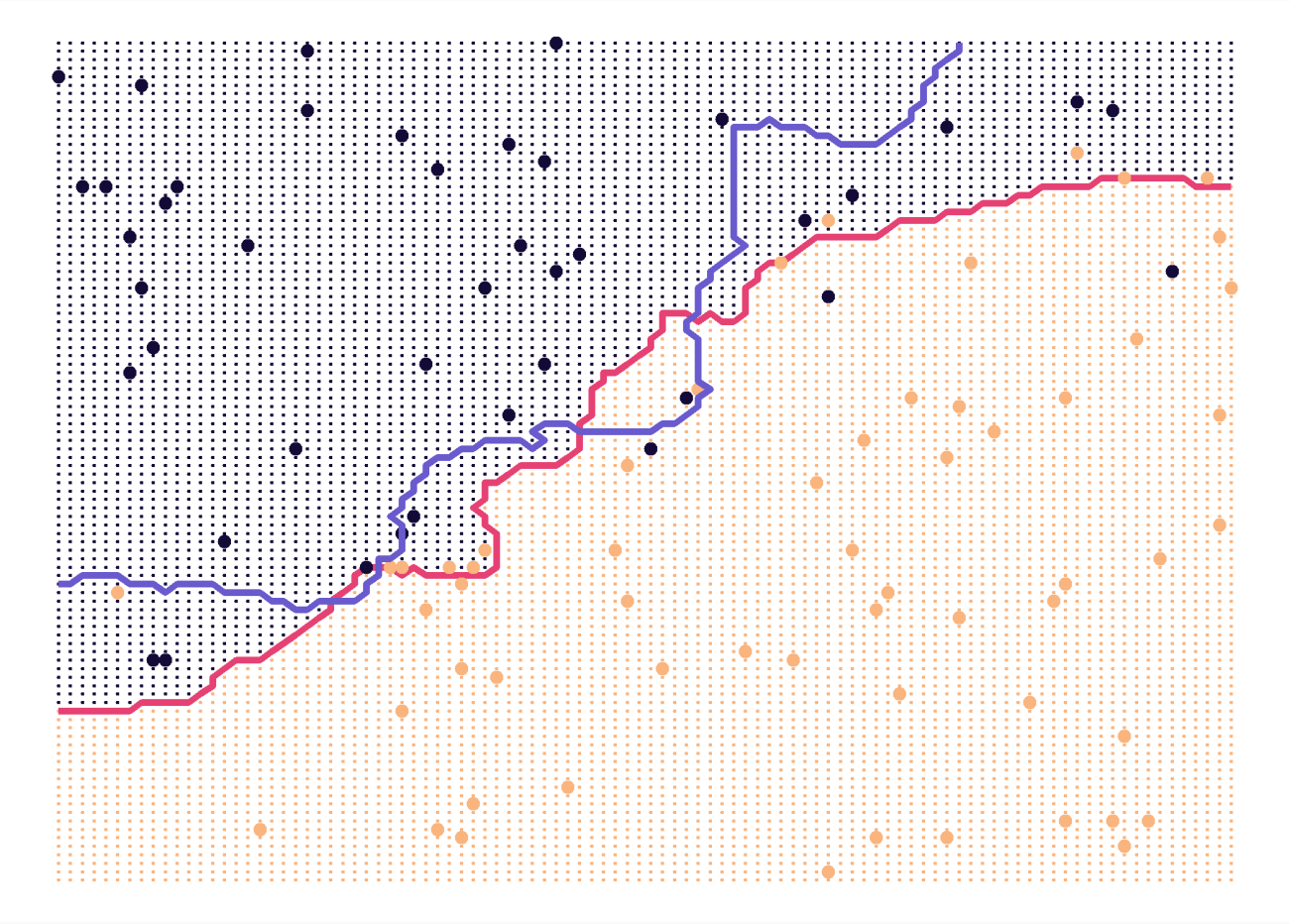
Bayes Decision Boundary

... and our first sample gives us an **estimated decision boundary**.



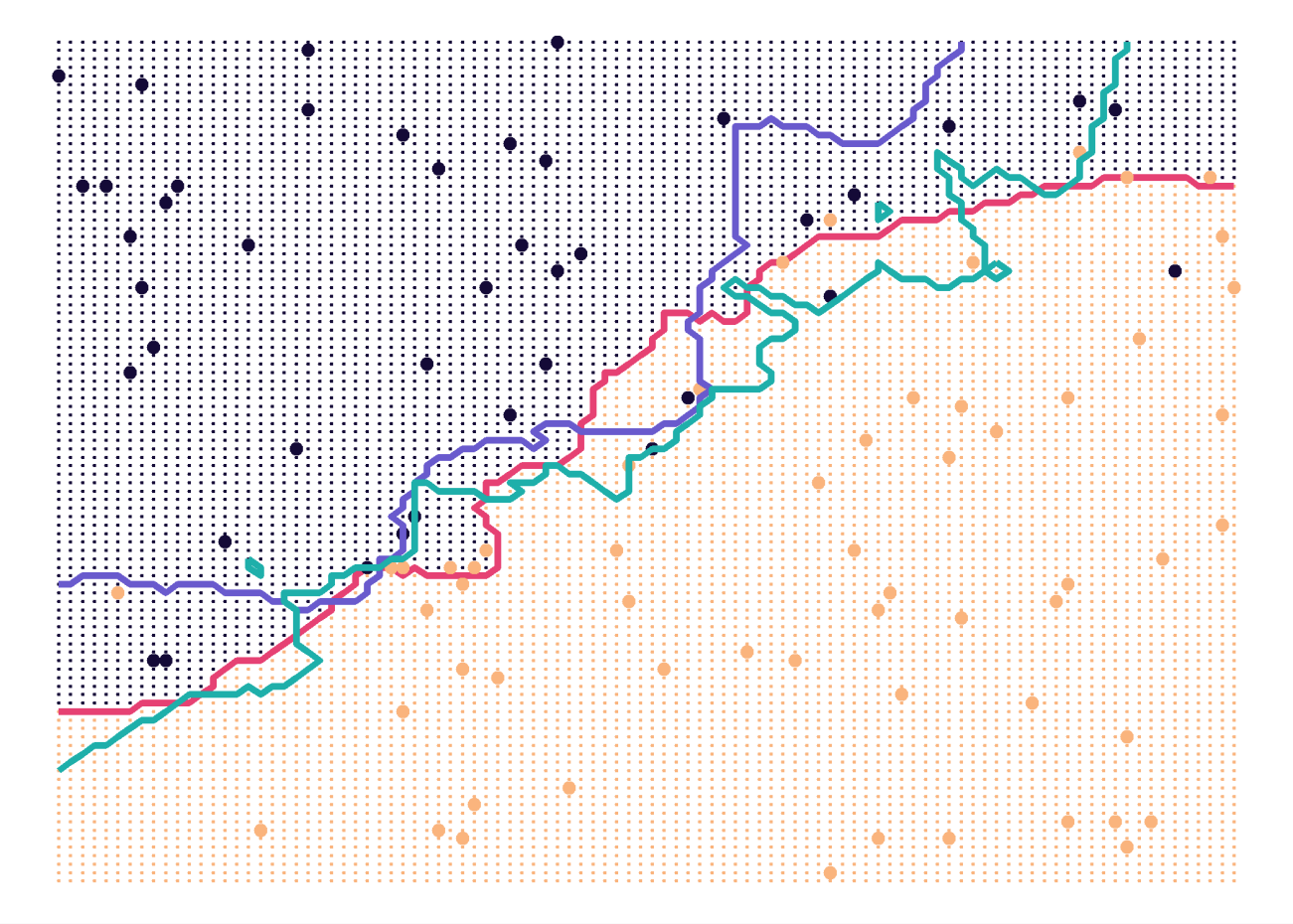
Bayes Decision Boundary

Resample...



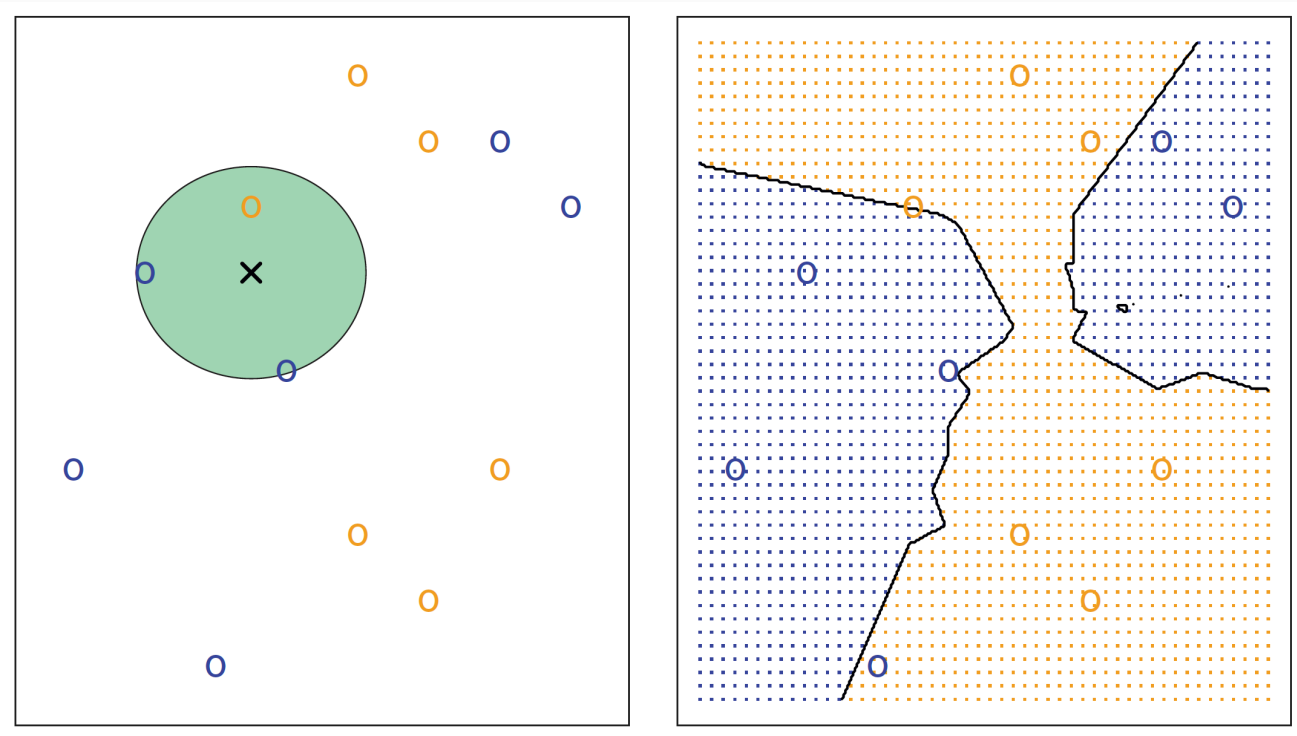
Bayes Decision Boundary

And the second sample gives us another **estimated decision boundary**.



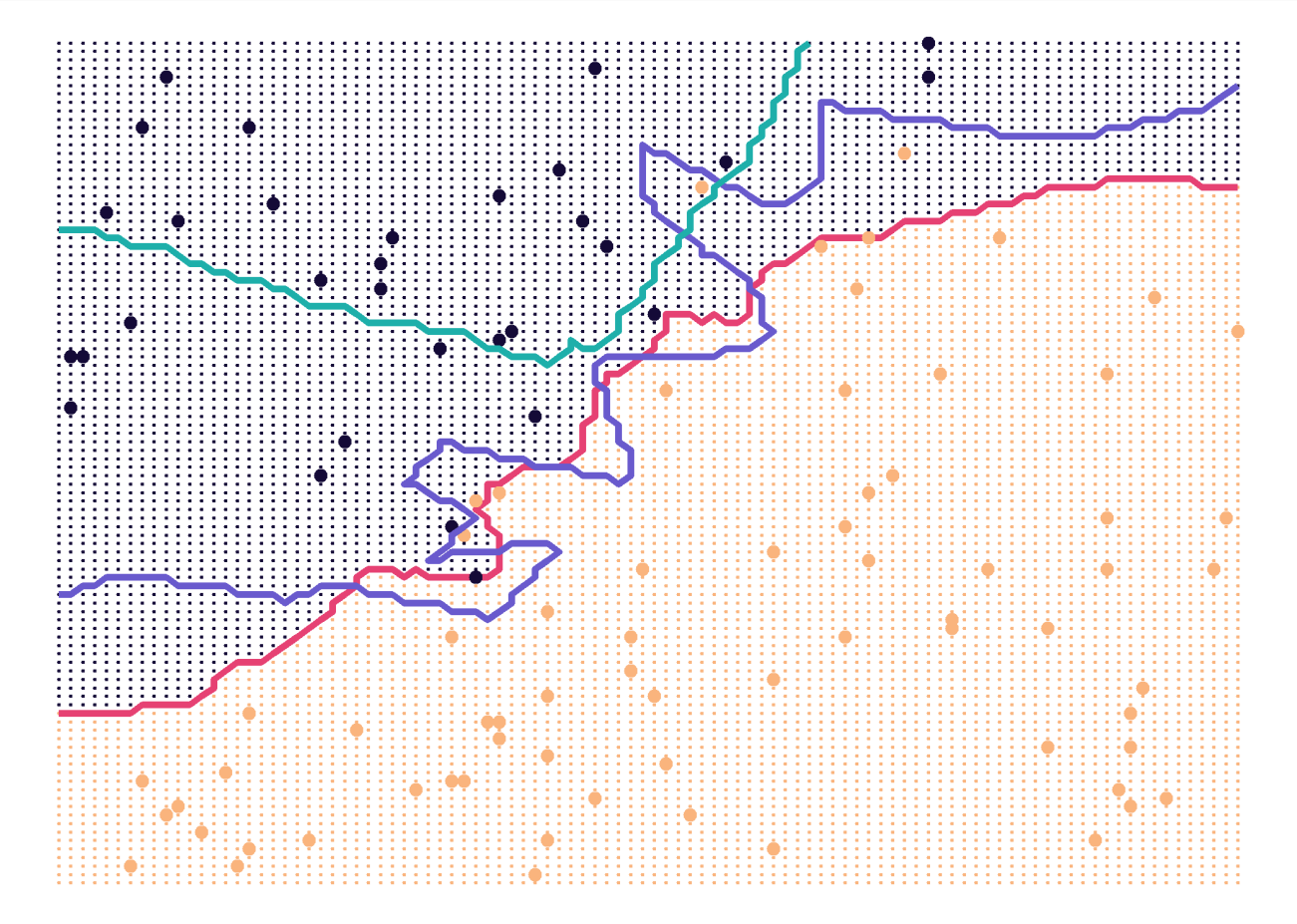
KNN in action

Left: K=3 estimation for "x". *Right:* KNN decision boundaries.



Source: ISL

Decision boundaries: **Bayes**, **K=1**, and **K=60**

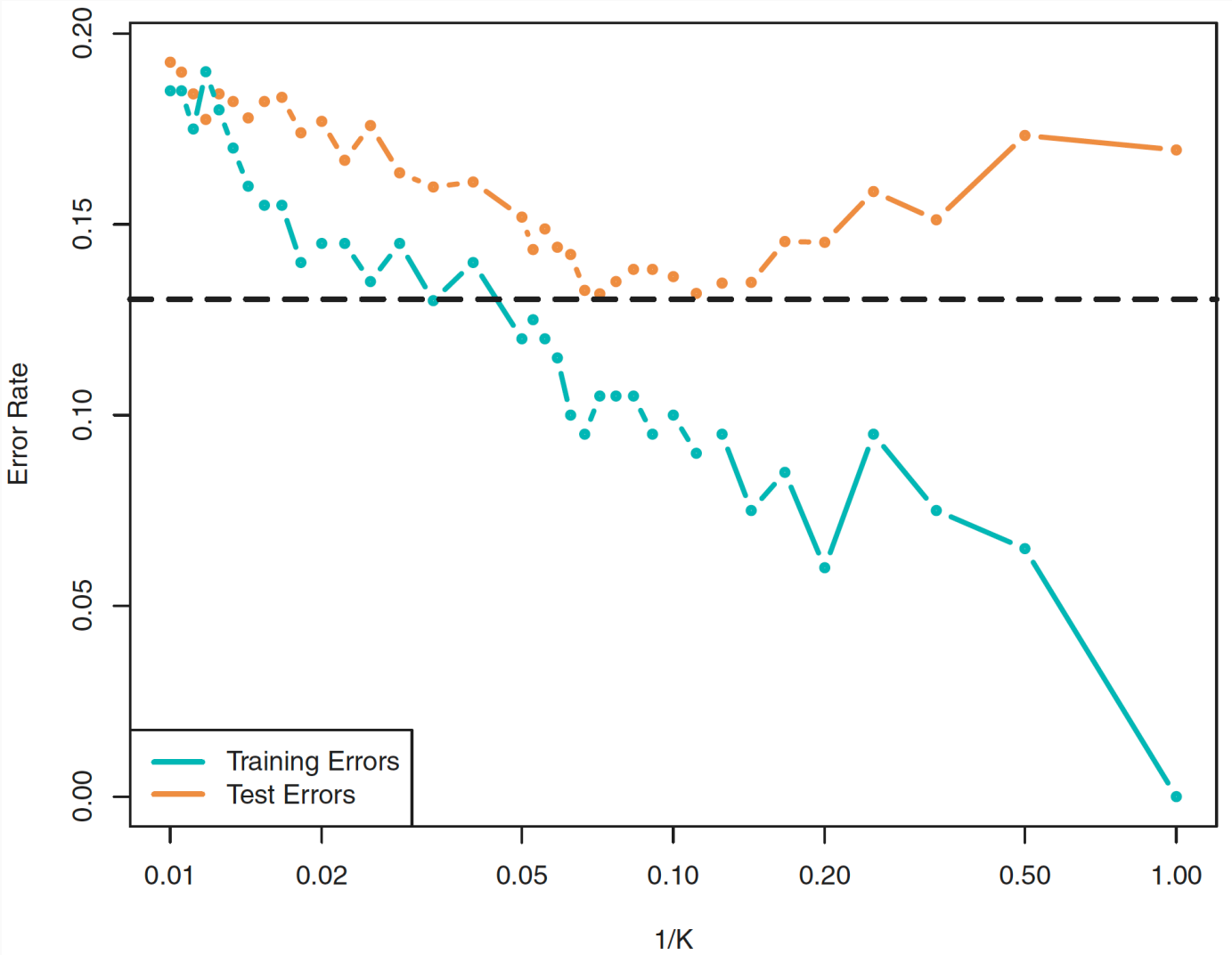


Choice of K

The choice of K is very important—ranging from super flexible to inflexible.

- When K is **low**, the decision boundary becomes **overly flexible**, finding patterns in the data that don't match truth
- When K is **high**, things become **too rigid** and the decision boundary approaches linearity

KNN error rates, as K increases



Source: ISL

Model Selection and Regularization

Model Selection and Regularization

As economists, we often use **Economic Theory** to specify our econometric models.

$$f(L, K) = Y = AL^{\alpha}K^{\beta}$$

Which we can easily represent in R with the formula `y ~ log(L) + log(K)`

Model Selection and Regularization

However, there are plenty of times where we aren't sure **which variables** to plausibly include as explanatory variables

$$Weather_{it} = f(Temp,\ Precip,\ Wind,\ Pressure, \dots)$$

In cases like this, we can use machine learning to aid in **model selection**.

Model Selection and Regularization

Two main approaches:

1. Subset-selection methods

1. Algorithmically search for the "best" subset of our p predictors
2. Estimate the linear models via least squares

These methods assume we need to choose a model before we fit it...

2. Shrinkage methods

- Fit a model that contains all p predictors
- Simultaneously: shrink† coefficients toward zero

Idea: Penalize the model for coefficients as they move away from zero.

† Synonyms for *shrink*: constrain or regularize

Shrinkage Methods

Two most commonly used shrinkage methods are **Ridge Regression** and **LASSO**

Shrinkage Methods

Two most commonly used shrinkage methods are **Ridge Regression** and **LASSO**

Ridge Regression

- Uses all variables
- Shrinks all coefficients towards zero

LASSO

- Shrinks some coefficients to exactly zero
- i.e. uses only a subset of variables

Shrinkage Methods: Ridge Regression

Recall Least-squares regression gets $\hat{\beta}_j$'s by minimizing RSS, *i.e.*,

$$\min_{\hat{\beta}} \text{RSS} = \min_{\hat{\beta}} \sum_{i=1}^{n} e_i^2 = \min_{\hat{\beta}} \sum_{i=1}^{n} \left(y_i - \underbrace{\left[\hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + \cdots + \hat{\beta}_p x_{i,p} \right]}_{=\hat{y}_i} \right)^2$$

Ridge regression makes a small change

- Adds a hi-medgrn[shrinkage penalty] = the sum of squared coefficients $\left(\lambda \sum_j \beta_j^2 \right)$
- Minimizes the (weighted) sum of **RSS and the shrinkage penalty**

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Shrinkage Methods: Ridge Regression

$$\min_{\hat{\beta}^R} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$\lambda \ (\geq 0)$ is a **tuning parameter** for the harshness of the penalty.
$\lambda = 0$ implies **no penalty:** we are back to least squares.
Size of penalty changes with **units of** $\mathbf{x}_j \Rightarrow$ standardize!
Each value of λ produces a new set of coefficients.

Ridge's approach to the bias-variance tradeoff: **Balance**

- Reducing RSS, *i.e.*, $\sum_i \left(y_i - \hat{y}_i \right)^2$
- Reducing coefficients (ignoring the intercept)

λ determines how much ridge "cares about" these two quantities.

Shrinkage Methods: Lasso

LASSO simply replaces ridge's **squared** coefficients with **absolute values**.

$$\min_{\hat{\beta}^L} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Lasso's penalty does not increase with the size of β_j (always pay λ to increase $|\beta_j|$ by one unit)
The only way to avoid lasso's penalty is to **set coefficients to zero**.

This feature has **two benefits:**

1. Some coefficients will be **set to zero**—we get **"sparse" models**.
2. Lasso can be used for subset/feature **selection**.

λ and Penalization

Choosing a **good** value for λ is key for Ridge and LASSO.

- If λ is too small, then our model is essentially back to OLS.
- If λ is too large, then we shrink all of our coefficients too close to zero (or remove all our variables entirely).

Q: So what do we do?

A: Cross validate!

(You saw that coming, right?)

Example

Let's get some practice with Lasso and Ridge using more credit card data:

```
credit_df ← ISLR::Credit %>% clean_names()
```

```
##    id income limit rating cards age education gender student married
## 1 218 12.456  5395    392     3  65        14   Male      No     Yes
## 2 187 36.472  3806    309     2  52        13   Male      No      No
## 3 121 27.241  1402    128     2  67        15 Female      No     Yes
## 4  23 20.103  2631    213     3  61        10   Male      No     Yes
## 5  61 35.510  5198    364     2  35        20 Female      No      No
## 6  71 24.889  3954    318     4  75        12   Male      No     Yes
##           ethnicity balance
## 1         Caucasian     955
## 2 African American     188
## 3             Asian       0
## 4 African American       0
## 5             Asian     631
## 6         Caucasian     357
```

Example

First, we'll want to perform some cleaning, converting the categorical variables to dummies

```
credit_std ← mutate(credit_df,
                    gender_female = (gender == "Female") %>% as.numeric(
                    student_yes = (student == "Yes") %>% as.numeric(),
                    married_yes = (married == "Yes") %>% as.numeric(),
                    ethnicity_asian = (ethnicity == "Asian") %>% as.nume
                    ethnicity_caucasian = (ethnicity == "Caucasian") %>%
) %>%
  select(-gender, -student, -married, -ethnicity)
```

Example

For Ridge and Lasso, we will use `glmnet()` from the **glmnet** package.

The **key arguments** for `glmnet()` are

- `x` a [matrix] of predictors
- `y` outcome variable as a vector
- `standardize` (`T` or `F`)
- `alpha` elasticnet parameter
  - `alpha=0` gives ridge
  - `alpha=1` gives lasso

- `lambda` tuning parameter (sequence of numbers)
- `nlambda` alternatively, R picks a sequence of values for λ

Example

We just need to define a **decreasing sequence** for λ, and then we're set.

```
# Define our range of lambdas (glmnet wants decreasing range)
lambdas ← 10^seq(from = 5, to = -2, length = 100)
```

Estimating ridge regression:

```
# Fit ridge regression
est_ridge = glmnet(
  x = credit_std %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_std$balance,
  standardize = T,
  alpha = 0, # alpha = 0 for Ridge
  lambda = lambdas
)
```

Ridge

The `glmnet` output (`est_ridge` here) contains estimated coefficients for λ.

Use `coef(s = lambda)` to retrieve estimated coefficients for a **specific lambda**

For example, the smallest lambda (0.01)

```
coef(est_ridge, s = 0.01)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)         -484.5225957
## income                -7.8000469
## limit                  0.1763554
## rating                 1.3529988
## cards                 16.6769557
## age                   -0.6161700
## education             -1.0438640
## gender_female        -10.6555578
## student_yes          425.0254323
```

Ridge

Looking at the largest lambda (1e+05) highlights how all the coefficients have been driven close to zero:

```
coef(est_ridge, s = 1e+05)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)          5.104357e+02
## income               2.727528e-02
## limit                7.794611e-04
## rating               1.165415e-02
## cards                1.322199e-01
## age                  8.316248e-05
## education           -5.282613e-03
## gender_female        8.975916e-02
## student_yes          1.812122e+00
## married_yes         -2.548000e-02
## ethnicity_asian     -4.613786e-02
## ethnicity_caucasian -1.342033e-02
```

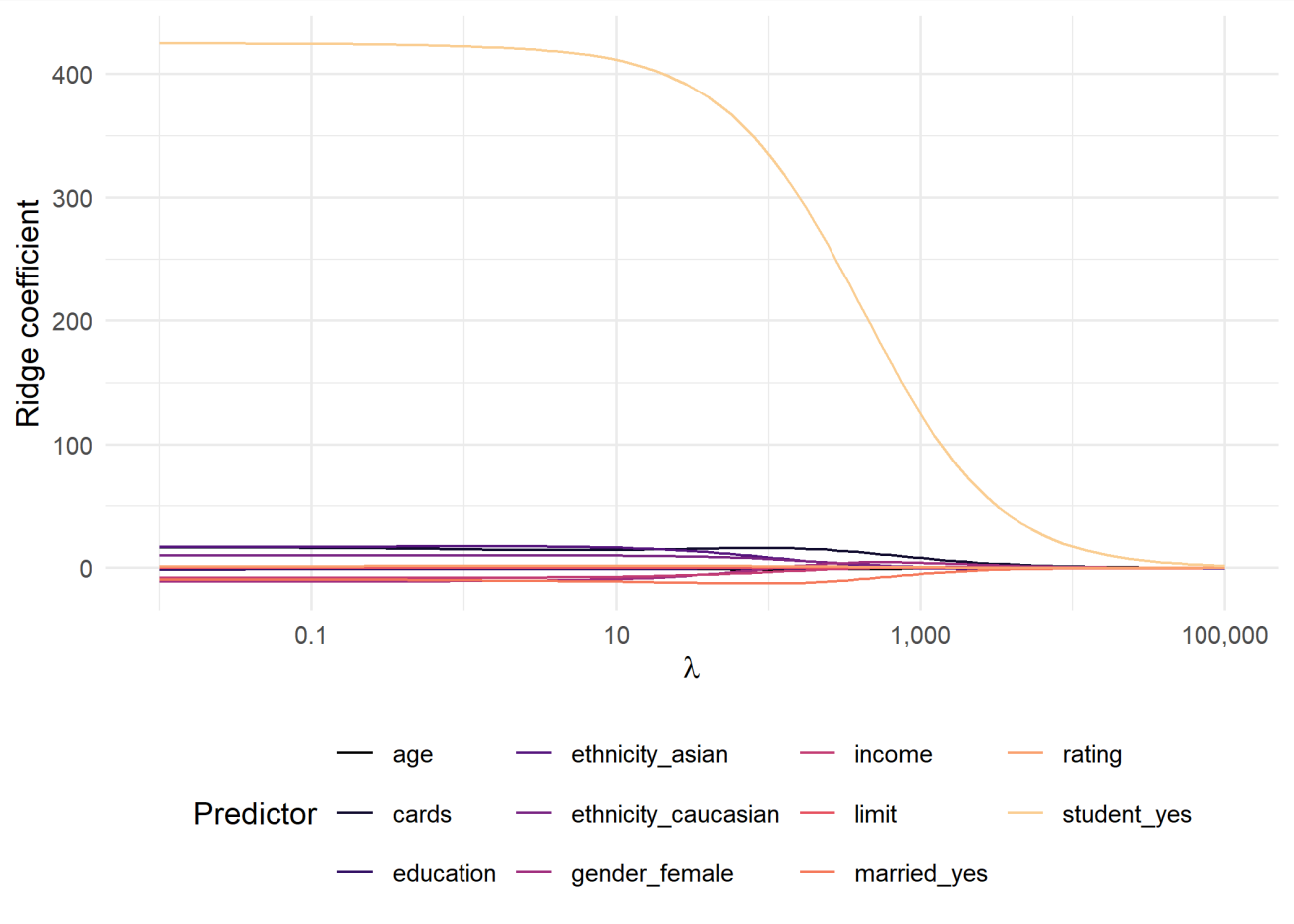
Ridge

Retrieving all the Ridge regression coefficients for λ between 0.01 and 100,000 and formatting long:

```
# convert to data frame with variables on the columns
ridge_df ← est_ridge %>% coef() %>% t() %>% as.matrix() %>% as.data.frame
# remove intercept and add in lambdas
ridge_df %<>% dplyr::select(-1) %>% mutate(lambda = est_ridge$lambda)
# Convert to long format
ridge_df %<>% pivot_longer(-lambda, names_to = "variable", values_to = "c
  arrange(lambda, variable)
```

Ridge

Plotting the coefficients for λ between 0.01 and 100,000



Ridge and Lasso

`glmnet` also provides convenient cross-validation function: `cv.glmnet()`.

```
# Cross validation
ridge_cv ← cv.glmnet(
  x = credit_std %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_std$balance,
  alpha = 0,
  standardize = T,
  lambda = lambdas,
  # New: How we make decisions and number of folds
  type.measure = "mse", # loss function
  nfolds = 5 # number of folds to use (LOOCV if set at sample size)
)
```

Ridge and Lasso

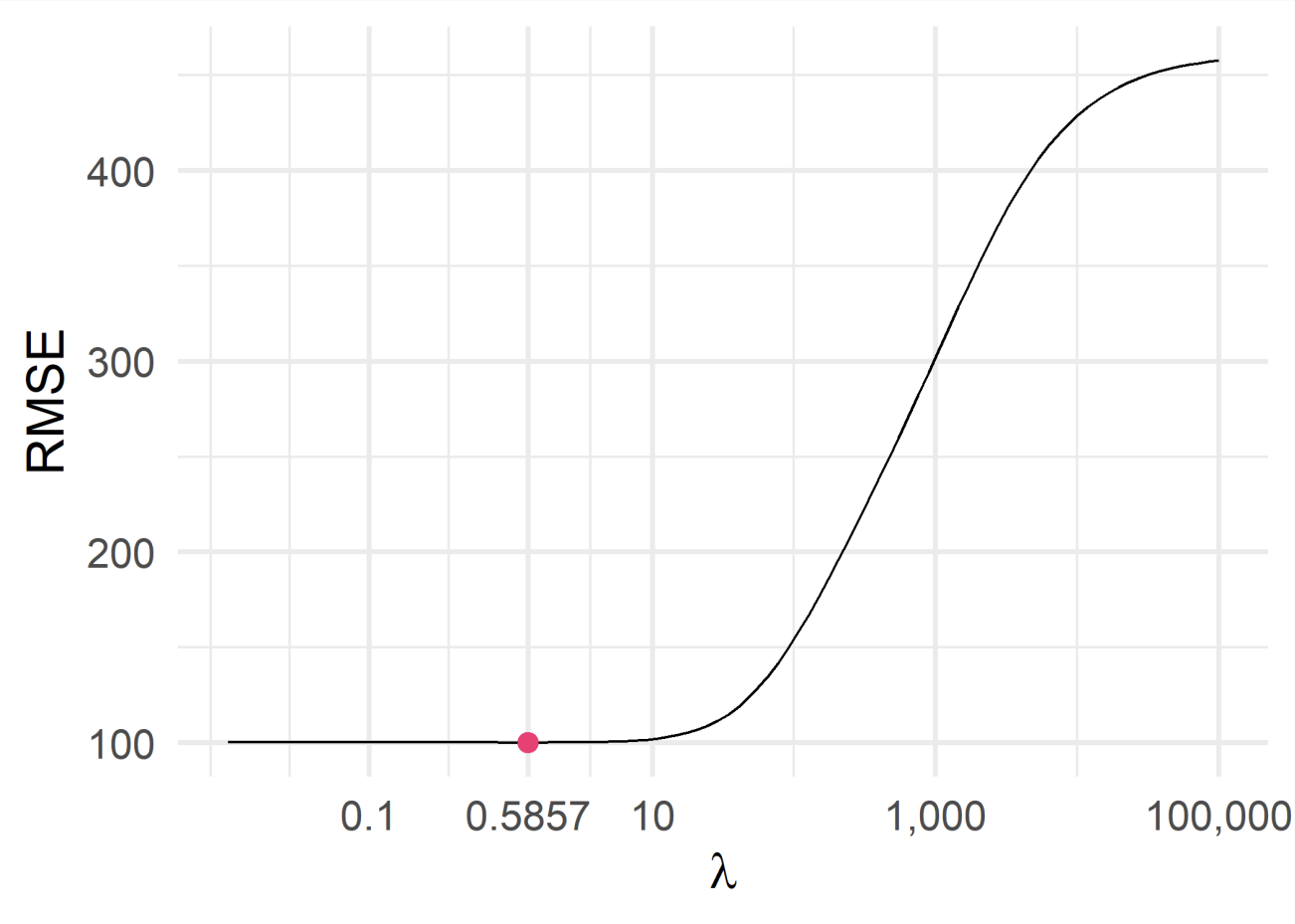
Which choice of lambda gives lowest cross-validated error?

```
ridge_cv$lambda.min
```

```
## [1] 1.321941
```

Ridge

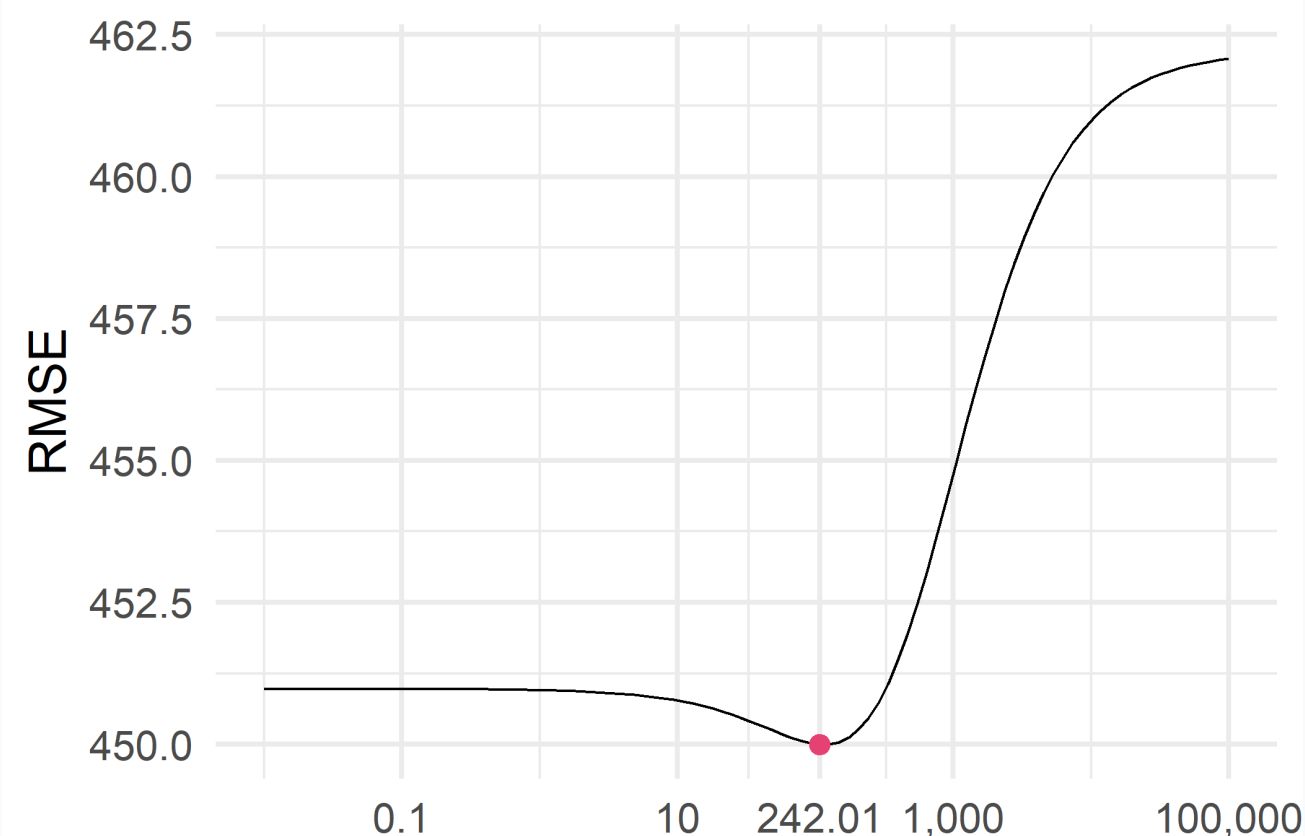
Visualizing Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



Ridge

Often, you will have a minimum more obviously far from the extremes.

For example, if we re-run our Ridge regression but omit the first three predictors (income, limit, and rating):

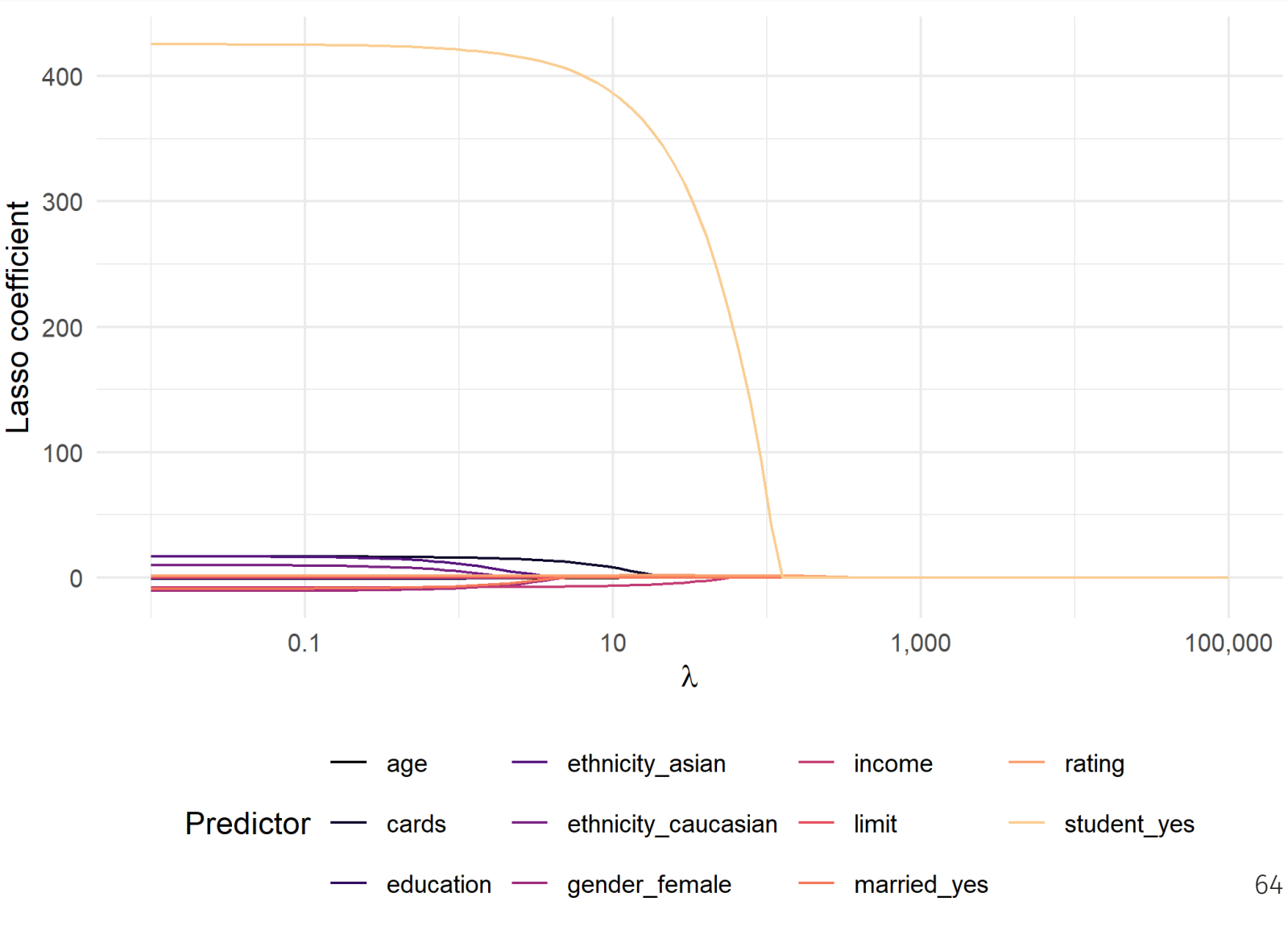


Lasso

How do our results differ if we perform Lasso?

```
est_lasso ← glmnet(
  x = credit_std %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_std$balance,
  alpha = 1, # Lasso
  standardize = T,
  lambda = lambdas
)
```

Lasso Coefficients for Range of λ



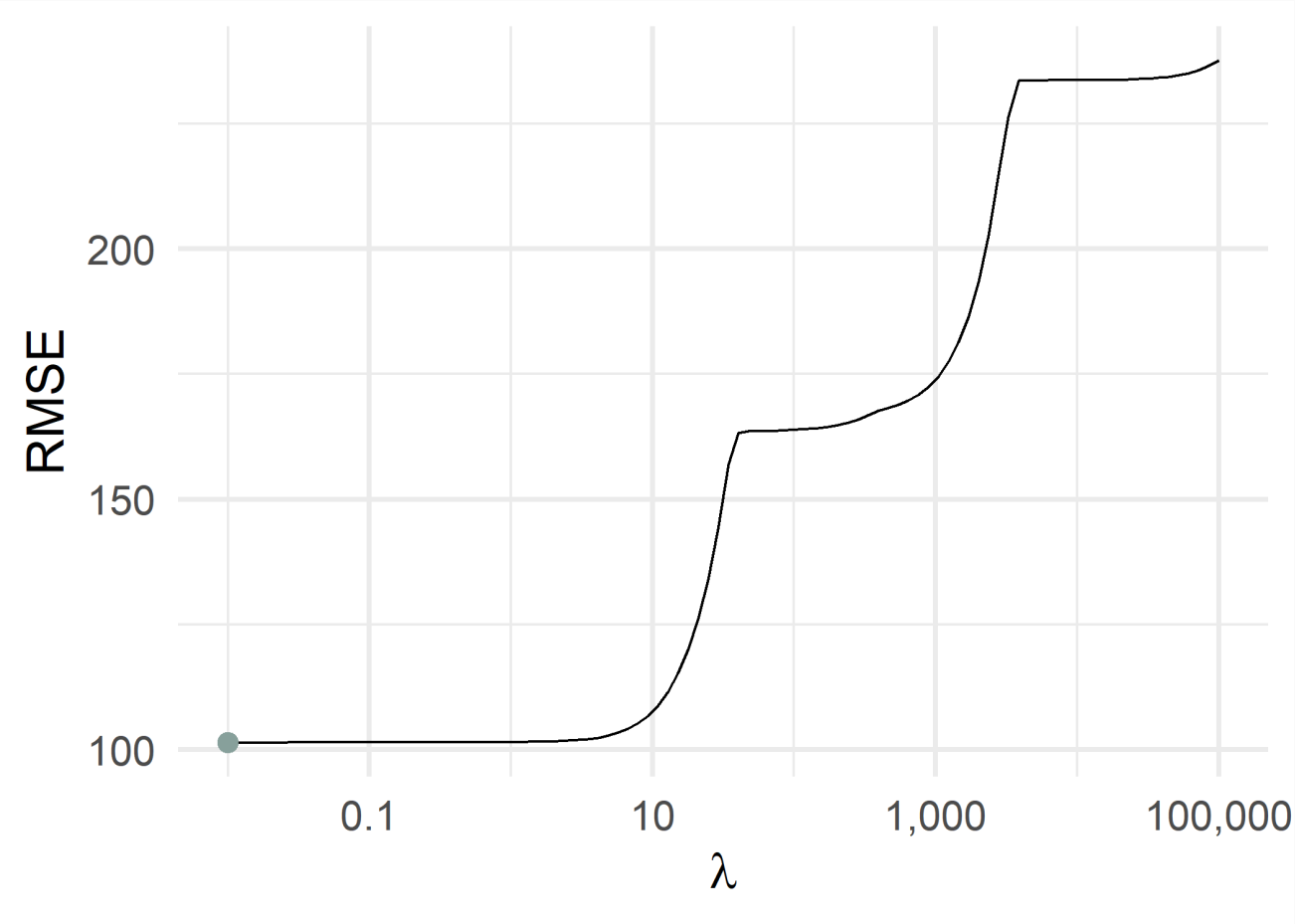
Lasso

We can also cross validate λ with `cv.glmnet()`:

```
lasso_cv ← cv.glmnet(
  x = credit_std %>% dplyr::select(-balance, -id) %>% as.matrix(),
  y = credit_std$balance,
  alpha = 1, # Lasso
  standardize = F,
  lambda = lambdas,
  # New: How we make decisions and number of folds
  type.measure = "mse", # loss function
  nfolds = 5 # number of folds to use (LOOCV if set at sample size)
)
```

Lasso

Cross-validated RMSE and λ: Which λ minimizes CV RMSE?



Which to Use: Ridge or Lasso?

Ridge regression

+ shrinks $\hat{\beta}_j$ *near* 0
- many small $\hat{\beta}_j$
- doesn't work for selection
- difficult to interpret output
+ better when all $\beta_j \neq 0$

Best: p is large & $\beta_j \approx \beta_k$

Lasso

+ shrinks $\hat{\beta}_j$ to 0
+ many $\hat{\beta}_j = 0$
+ great for selection
+ sparse models easier to interpret
- implicitly assumes some $\beta = 0$

Best: p is large & many $\beta_j \approx 0$

> [N]either ridge... nor the lasso will universally dominate the other.

ISL, p. 224

elasticnet, which combines Ridge and Lasso.

Part 2: Machine Learning Methods for Classification and Model Selection

1. Machine Learning for Classification
2. Model Selection and Regularization