Lecture 7: Core Machine Learning, part 1

LSE ME314: Introduction to Data Science and Machine Learning (https://github.com/me314-lse)

2025-07-23

Ryan Hübert

Slides last updated: 23 July 2025

Taking stock

What have we done so far?

- ➜ Principles of data science
- ➜ Introduction to data
- ➜ Regression
- ➜ Causal inference

Where are we going?

- ➜ Machine learning: supervised and unsupervised (3 lectures)
- ➜ "Unstructured" data: web, audio, visual and spatial (1 lecture)
- ➜ Text data: quantifying and analyzing texts, LLMs (1 lecture)
- ➜ Theory and data (1 lecture)

Who am I?

Associate professor in computational social science.

Political scientist by training with interests in "political economy."

Most of my research is on U.S. political institutions, esp. courts.

Dual persona: game theory + observational empirical analysis.

Empirical work on questions at methodological intersection of causal inference and machine learning.

- ➜ Do different judges cause legal cases to be resolved differently?
- ➜ Can we predict and model how judges make decisions?

Administrative reminders

Please put away your phones.

Your midsession assignment was due at 9:30 AM (25%)

Final exam next Friday, 1 August 2025 (75%).

- ➜ Details on time and date and format next week.
- ➜ Calculator allowed (but check regulations!).

Next Thursday's seminar will contain some exam review.

Exam regulations at UK uni may be different than you expect.

- ➜ In particular: we don't control many aspects of the process—review the information provided by SSO.

Administrative reminders

Percentage Mark	Letter Grade
80 and above	A+
70–79	A
65–69	A–
60–64	B+
50–59	B
48–49	B–
42–47	C+
40–41	C
Below 40	F

What is machine learning?

Beyond the hype

From Mitchell (1997), p. 2:

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Beyond the hype

Tasks: machine learns to do tasks you assign to it.

- ➜ A human defines the task, either explicitly or by setting the goals and controlling the environment.

Experience: it learns from the patterns in data or feedback.

- ➜ Uses statistics and algorithms to quantify patterns in data.
- ➜ Human tells it which techniques to use, how to use them.
- ➜ It builds a "model" based on what it sees, which can be used to do the assigned task.

Performance: it optimizes its performance against task-relevant metrics or rewards.

- ➜ It's performance on the tasks can be quantified and measured.
- ➜ Improving performance is partly its job, partly the human's job.

Beyond the hype

A quick digression on "artificial intelligence":

- ➜ Broadly speaking, ML is just a type of AI.
- ➜ Most modern examples of what we call "AI" are just (very fancy) ML models.
- ➜ ChatGPT is a sophisticated form of machine learning!

Feel free to tell your friends and family you learned how to build AI at LSE this summer ;)

Beyond the hype

A key idea I want you to keep in your mind:

The machine learns, but humans design the task, the learning process, and the goals.

Even "autonomous" AI is designed by humans.

➜ Distinction: degree of micromanagement by humans.

AFAICT humans will need to manage AI for foreseeable future.

It is why you need to learn this stuff.

Defining the tasks

Some common tasks that machines can learn to do:

- ➜ Classify things
- ➜ Predict or forecast things
- ➜ Group similar things

Architectures required for machines to learn these tasks involve forms of **statistical learning** and **algorithms**.

We'll look at some of these architectures.

But first: when would you use machine learning?

. . . depends on what you want to know.

Defining the tasks: what do you want to know?

You have a dataset with X and Y. What do you want to know?

Possibility #1:

- Dataset is *observationally* complete: you have values for all three variables for all observations.
- You are interested in getting an accurate quantitative estimate of the relationship between X and Y.
- Why? Maybe you want to know if X *causes* Y.
- Goal: estimate a treatment effect of X on Y.
- Trick: causal identification strategy like random assignment.
- Challenge: finding a good causal identification strategy.

Defining the tasks: what do you want to know?

You have a dataset with X and Y. What do you want to know?

Possibility #2:

- ➜ Dataset is incomplete: some values of Y are missing.
- ➜ You are interested in filling in these missing values.
- ➜ Why? Maybe Y is a crucial variable for your company's decision-making.
- ➜ Goal: predict the missing Y values.
- ➜ Trick: exploit relationship between X and Y to predict missing Ys.
- ➜ Challenge: you don't actually know the relationship.

Defining the tasks: what do you want to know?

Suppose you're a data scientist for a bank that issues loans.

You have some data, consisting of a few variables

- whether a customer defaulted on a loan (Y)
- whether a customer completed an online financial literacy course (X_1)
- a customer's income (X_2)
- a customer's age (X_3)

What do you want to know?

Defining the tasks: what do you want to know?

Suppose you are trying to determine if the online financial literacy course causes more customers to repay on time.

- ➜ This is a causal question: your focus is on measuring relationship between X_1 and Y.
- ➜ You need a complete dataset with all values of Y to do this.

Suppose instead, you are reviewing loan applications from new applicants and trying to decide whether to issue loans.

- ➜ You cannot see into the future, and so you have missing Y values for the loan applicants.
- ➜ But if you do have rich historical data on previous loans + data on the applicants: you can use historical data and ML to predict missing values of Y for the applicants.

Defining the tasks: what do you want to know?

Machine learning, at its core, is a set of techniques to fill in missing values for important variables.

Contrast this with causal inference, where core goal is learning about (causal) relationships between variables.

ML is about measurement; CI is about explanation.

Might sound narrow, but it is not!

- ➜ All of *learning* (by humans or by machines) is a process of filling in missing "data" about the world.
- ➜ Both are crucial for decision-making, research, etc.

Some vocabulary

ML is for filling in missing data. We need some specialized vocabulary to keep things straight.

Issue #1: what data is missing?

- **Response**: variable we're primarily interested in, and for which some values are missing.
  - Convention to call this variable Y in mathematical notation.
  - Could call it "outcome" but that's a bit awkward in ML.
- **Predictors** or **features**: variables machine uses to help learn about response variable.
  - Convention to call these variables X in mathematical notation, using indices if there are several, such as X_1, X_2, etc.
  - Ideally, we have no missing values for the predictors.

Side note: Y_i and Y are conceptually different, but the line is blurry.

Some vocabulary

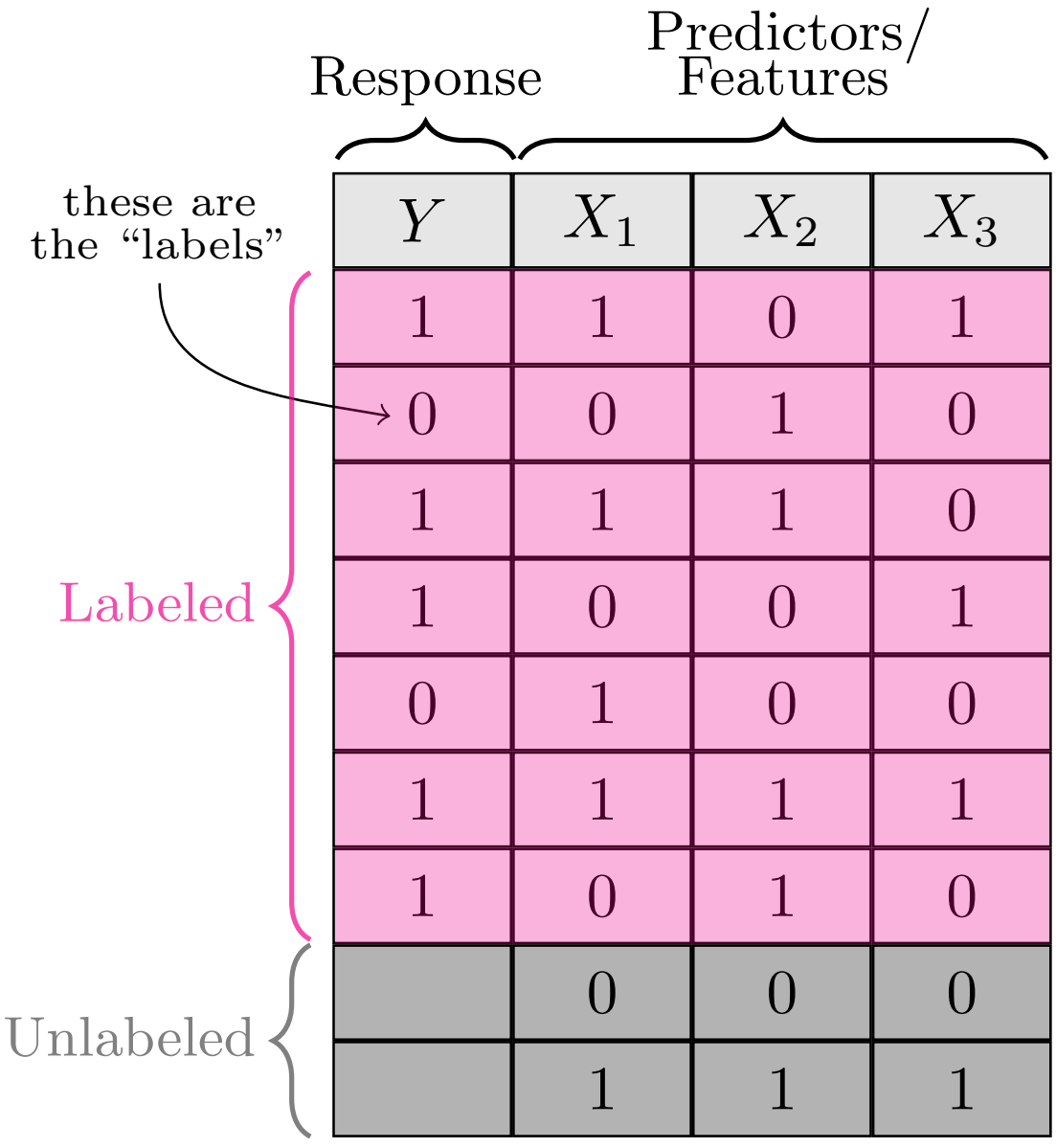
Issue #2: which observations have missing values for Y?

- **Unlabeled data**: observations for which we do not have values for the response variable.
  - Observations we have (but missing Y), or will get in future.
- **Labeled data**: observations for which we have values for the response variable.
  - This is data we currently have access to
  - **Training data**: *labeled* observations the machine uses to learn patterns (to "train" itself).
  - **Test (or validation) data**: *labeled* observations analyst uses to evaluate performance of machine's model after training.

Labeled and unlabeled data

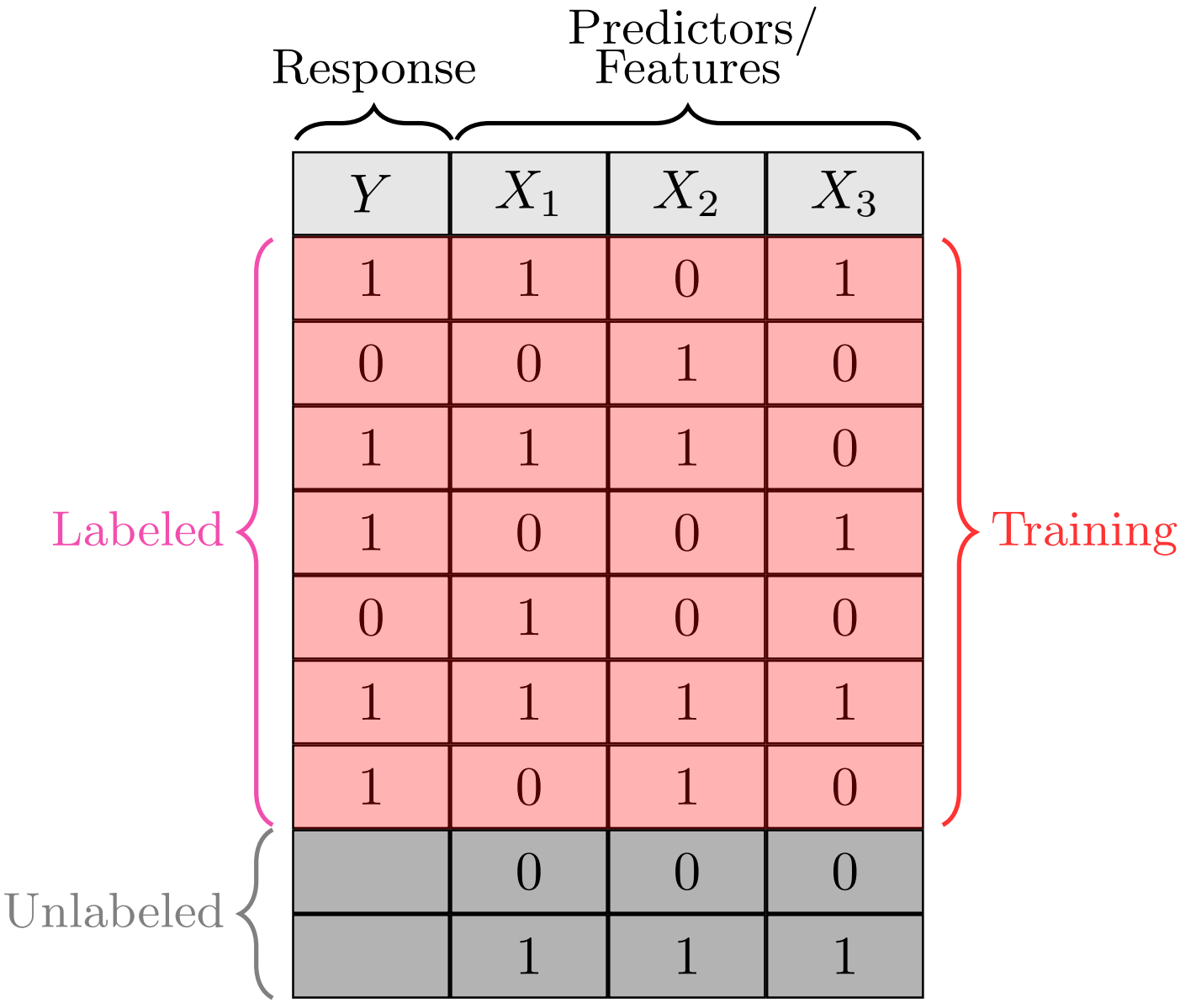
	Response	Predictors/ Features		
	Y	X_1	X_2	X_3
Observations	1	1	0	1
	0	0	1	0
	1	1	1	0
	1	0	0	1
	0	1	0	0
	1	1	1	1
	1	0	1	0
	0	0	0	0
	0	1	1	1

Labeled and unlabeled data



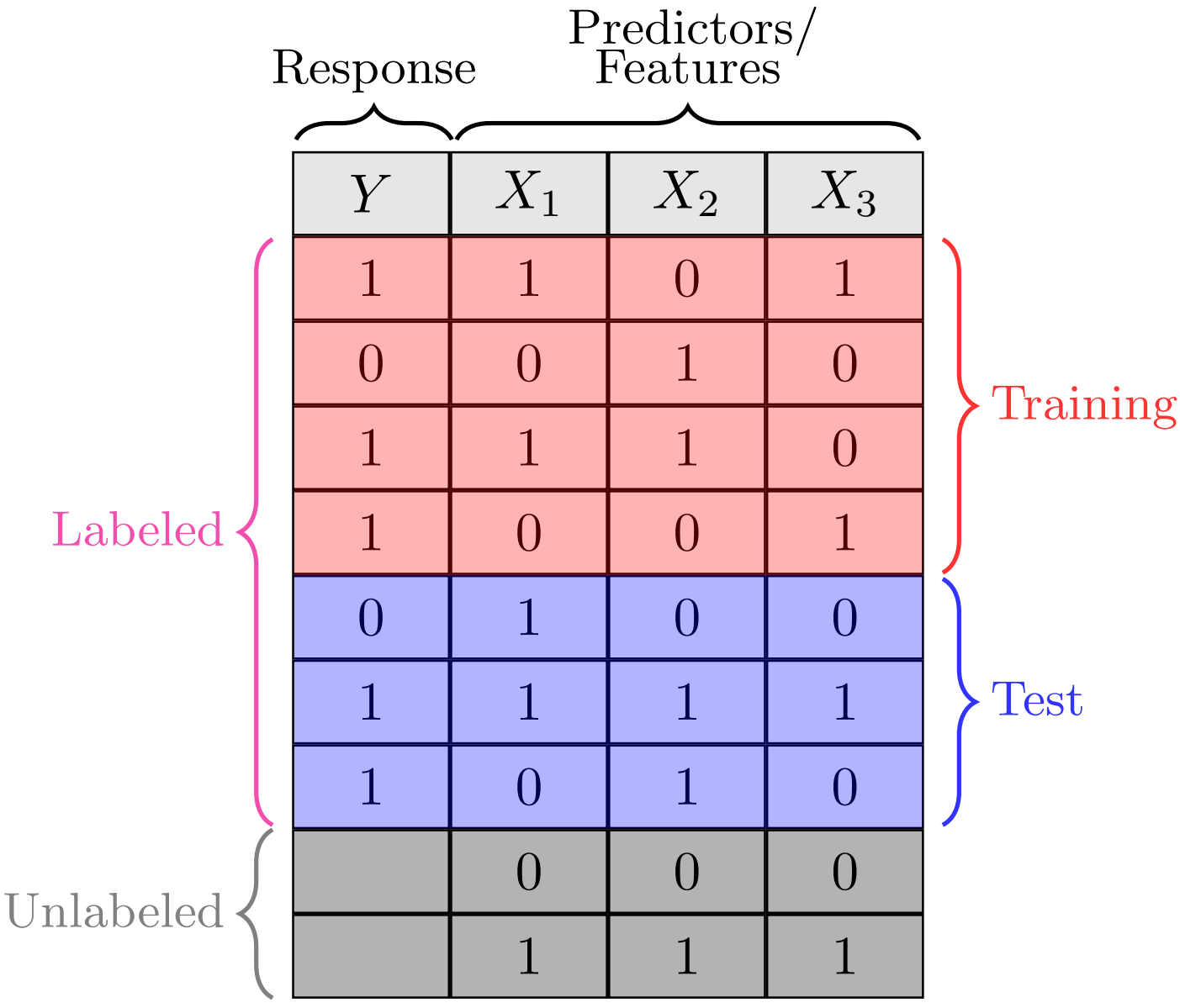
Y	X_1	X_2	X_3
1	1	0	1
0	0	1	0
1	1	1	0
1	0	0	1
0	1	0	0
1	1	1	1
1	0	1	0
	0	0	0
	1	1	1

Labeled and unlabeled data



Y	X_1	X_2	X_3
1	1	0	1
0	0	1	0
1	1	1	0
1	0	0	1
0	1	0	0
1	1	1	1
1	0	1	0
	0	0	0
	1	1	1

Labeled and unlabeled data



Types of machine learning

Consider two machine learning scenarios:

Scenario 1: variable Y is "well defined" and you have values for some observations.

For example:

- An email provider wants to know whether a received email is spam or not. Here, Y is well defined: $Y_i \in \{\text{spam}, \text{not spam}\}$.
- Hires a person to manually review 10,000 emails and label them as spam or not. The email provider now knows the value of Y for some of the emails (the labeled ones).

Types of machine learning

Consider two scenarios:

Scenario 2: variable Y is not "well defined," and you do not have a value for any of the observations.

For example:

- ➜ A school would like to customize instructional materials based on learning styles of enrolled students. Y is not well defined—unclear what the "learning styles" are ex ante.
- ➜ Has lots of data on each student and their performance in various classes, but not their "learning style." The school does not know the value of Y for any student.

Types of machine learning

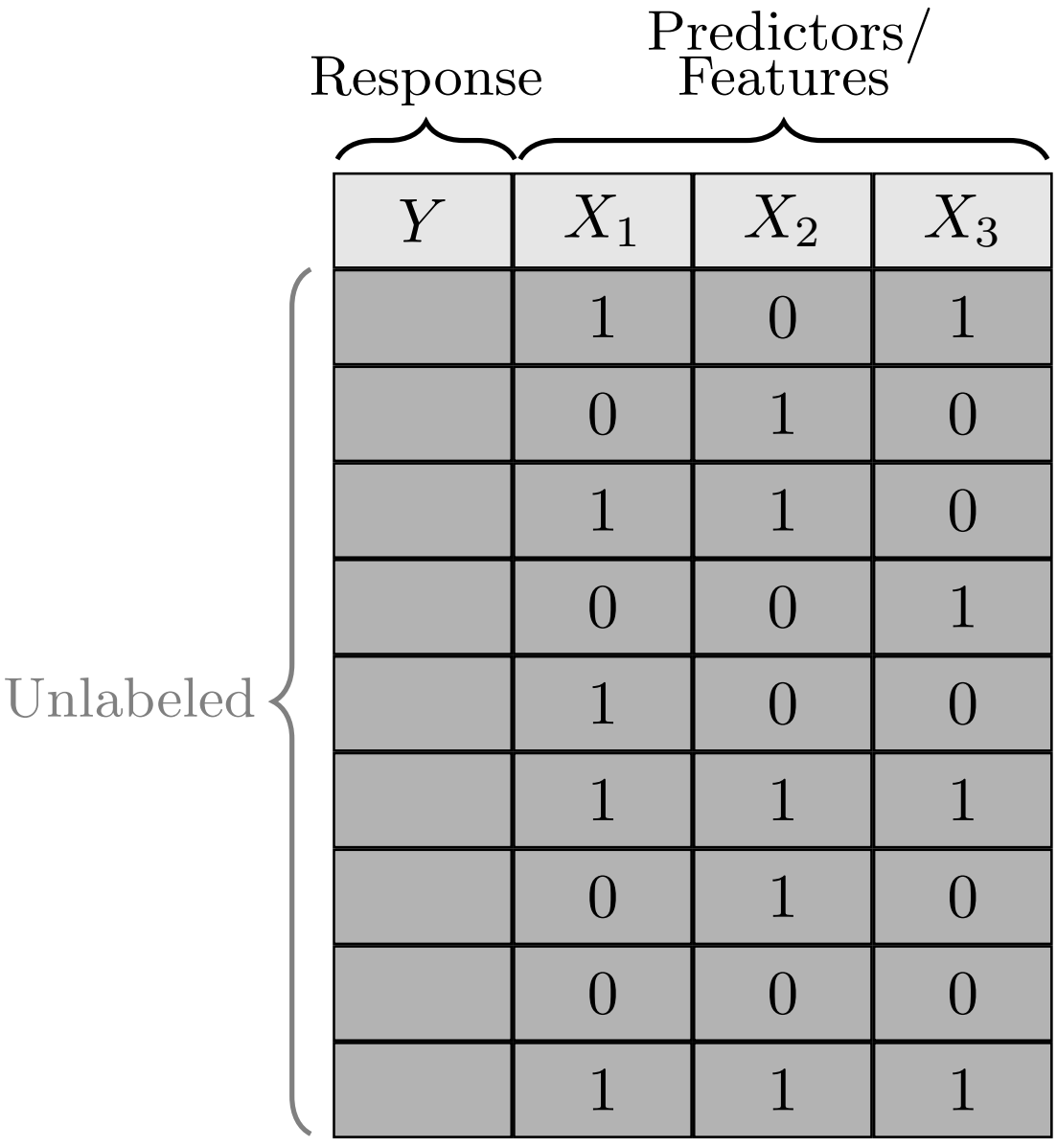
In scenario 1, the machine is doing **supervised learning**:

- ➜ Machine learns from existing labels of well-defined Y.
- ➜ Heavily constrained by data, i.e. external information about Y.
- ➜ Existing labels guide learning process directly via modeling choices.
- ➜ Example uses: forecasting/prediction and classification.

In scenario 2, the machine is doing **unsupervised learning**:

- ➜ Machine has no labeled response variable to learn from.
- ➜ Not constrained by labeled data.
- ➜ Constraints only come from analyst's modeling choices.
- ➜ Structure is imposed by algorithmic setup, not observed labels.
- ➜ Example use: grouping similar observations.

Types of machine learning



Aside: reinforcement learning

Supervised and unsupervised learning are the classical types of ML.

In this class, that's what we'll focus on.

➜ Supervised today and tomorrow; unsupervised next week.

But there is a third type of ML that is increasingly popular.

In **reinforcement learning**, the machine fills in missing values for Y using a reinforcement process.

➜ Good decisions are rewarded and/or bad decisions are punished.

➜ Model adapts over time, responding to rewards and punishments.

Roadmap of what is to come

Next 3 days: overview of ways you can get a machine to learn.

1 & 2: Supervised learning

- ➜ Mitchell's T: Types of supervised learning.
- ➜ Mitchell's P: performance metrics.
- ➜ Mitchell's E: how machine learns from experience.
- ➜ Revisiting Mitchell's P: building "robust" models.
  - ➜ The human's role in improving performance and filling in missing data.

3: Unsupervised learning and dimension reduction

Roadmap of what is to come

Warning: very brief overview — lots we cannot cover in only 3 days.

- ➜ My priority: broad conceptual understanding.
- ➜ You'll need to develop technical details in future courses.

Why would you need to use ML?

Before we get lost in the weeds. . . why use ML?

Brute force is possible: look at every observation and try to fill in missing values of Y by hand.

- ➜ This is what teachers do when they determine classify student work as "pass" or "fail."

Brute force not always practical.

- ➜ Google cannot hire enough people to manually review every email received at a Gmail address for spam.

In these cases, you can hand label small subset and let the machine learn patterns and apply to unlabeled sets!

Why would you need to use ML?

But the ethical considerations are tricky:

- ➜ Sometimes, hand labeling all observations is the only *ethical* thing to do—e.g., teachers failing students.
- ➜ Sometimes, it's not! Do you want Google/Microsoft/etc. to hire people to read all your emails?

Can take a hybrid approach: machine learning to get a score or index and then human judgment after considering the score.

- ➜ Some parole boards do this with "risk assessment instruments"
- ➜ "Human-in-the-chain" for battlefield decisions (scary!)

The are *huge* and *unresolved* moral and ethical debates about using ML/AI to make decisions, including right here at LSE.

Supervised learning overview

The main types of supervised learning

The response variable Y can be:

- ➜ continuous, such as $Y \in \mathbb{R}$ or $Y \in [-1, 1]$
- ➜ categorical (or discrete), such as $Y \in \{\text{blue}, \text{green}, \text{red}\}$

A special kind of categorical variable that takes two values is a binary variable, such as $Y \in \{\text{left}, \text{right}\}$.

As usual: always transform categorical variables into numbers.

- ➜ Binary variables are transformed to 0s and 1s.
- ➜ For example, if $Y \in \{\text{left}, \text{right}\}$, we might code left as 0 and right as 1, changing: $Y \in \{0, 1\}$.
- ➜ ≥ 3 categories: "dummy" the variable (or use `factor()`).

The main types of supervised learning

The fact that Y can be continuous or categorical gives us two main types of supervised learning:

1. **Regression** uses machine learning to guess the value of a continuous variable—Y is continuous.

2. **Classification** uses machine learning to classify observations into known categories—Y is categorical.

We refer to the "guesses" the machine makes as **predictions**.

Note: "regression" is a confusing term with multiple meanings, e.g. we'll use regression-based techniques to do classification.

The main types of supervised learning

Examples of continuous supervised classification:

- Company forecasting product sales to guide production.
- Real estate company (e.g., RightMove, Zillow, etc.) predicting future home prices.
- Electric utility forecasting energy consumption to determine whether to take emergency measures.
- Health officials predicting vaccine take-up during a global pandemic.
- Etc.

The main types of supervised learning

Examples of classification:

- Email provider classifying incoming mail as “spam” or “not spam”.
- Parole board classifying post-release outcomes as “will re-offend” or “won’t re offend”.
- Social media company classifying user posts as “hate speech” or “not hate speech”.
- Bank classifying potential loan outcomes as “default” or “not default”.
- Etc.

All binary examples, but classification with more categories is possible!

Performance metrics: continuous response (regression)

Quantifying performance

With supervised learning, we want to build a model that does a "good job" predicting values of Y.

"Good job" = model's predictions not too different from the truth.

A variety of **performance metrics** quantify this.

Remember: we only know the truth for *some* observations.

- ➜ We can only quantify performance on labeled data.
- ➜ An issue we revisit in detail tomorrow.

Why talk about performance metrics before we even know how to use ML to *create* predictions?

- ➜ To keep our eyes on the prize.
- ➜ You already know one way to create predictions. . .

Evaluating linear regression for prediction

Suppose I have labeled data with a predictor and a response:

	Y	X
1	2.145000	-0.1128886
2	5.982464	0.7975038
3	1.368951	-0.5276081
4	-1.216584	-0.2566335
5	-4.858001	1.3644160
6	1.820742	0.5826611

Suppose I assume a DGP where: $Y = \beta_0 + \beta_1 X$.

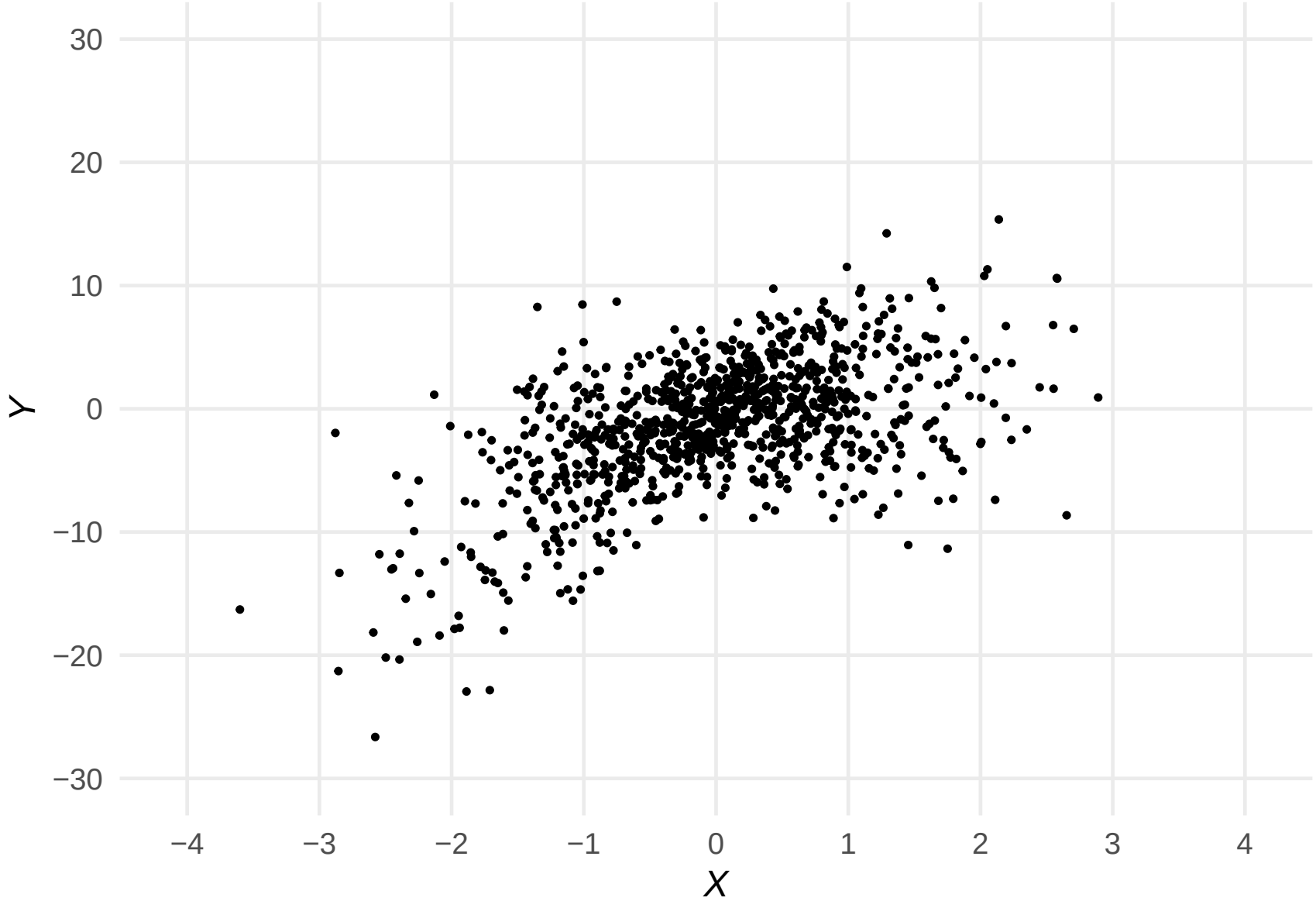
- That is: we assume the true relationship between X and Y is captured by this equation.

Evaluating linear regression for prediction

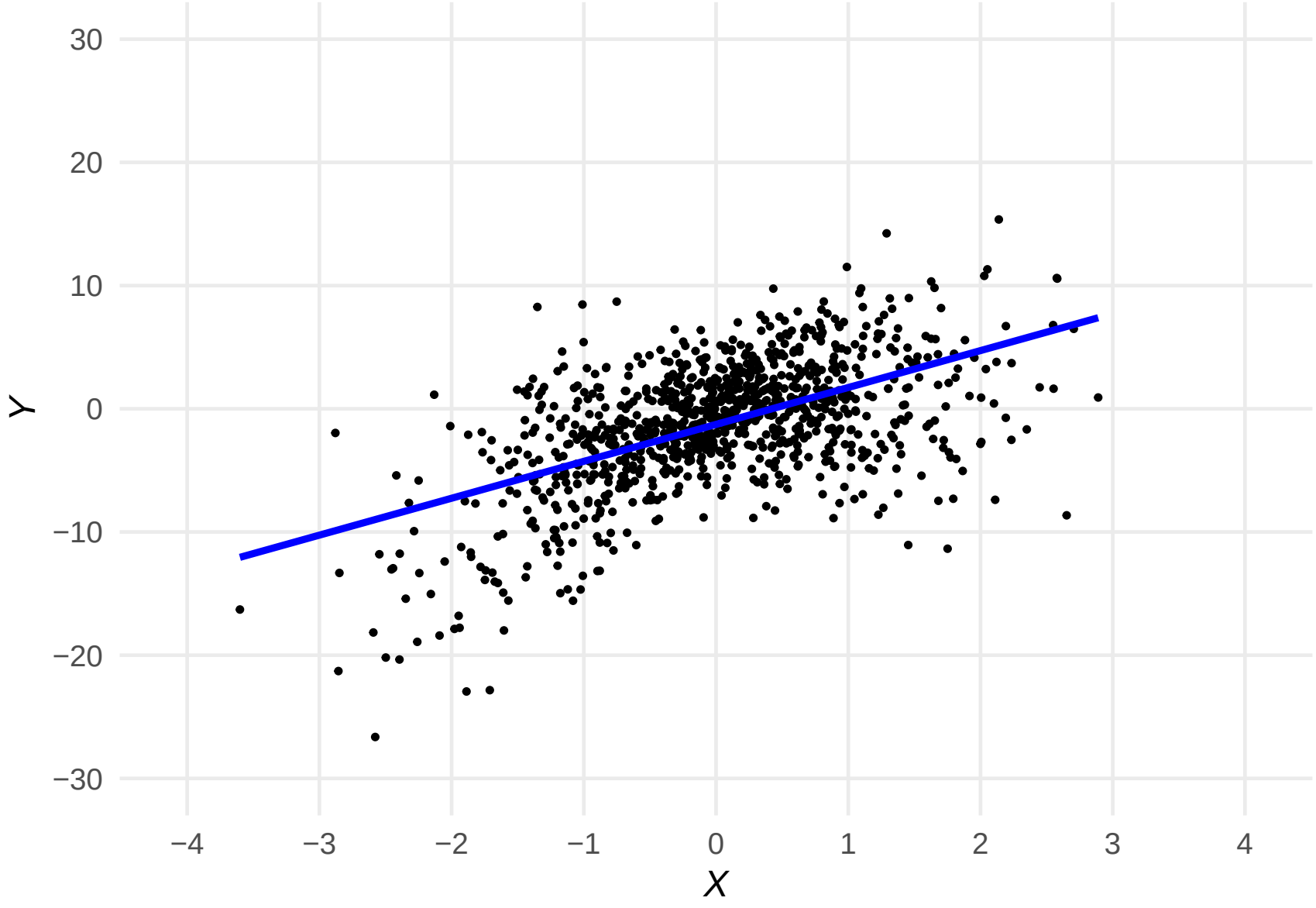
```r
mod <- lm(Y ~ X, data = df)
summary(mod)
```

```
Call:
lm(formula = Y ~ X, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-17.6467  -2.3956   0.4201   2.9427  13.5798

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.2685     0.1373  -9.241   <2e-16 ***
X             2.9956     0.1424  21.037   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.34 on 998 degrees of freedom
Multiple R-squared:  0.3072,    Adjusted R-squared:  0.3065
F-statistic: 442.6 on 1 and 998 DF,  p-value: < 2.2e-16
```

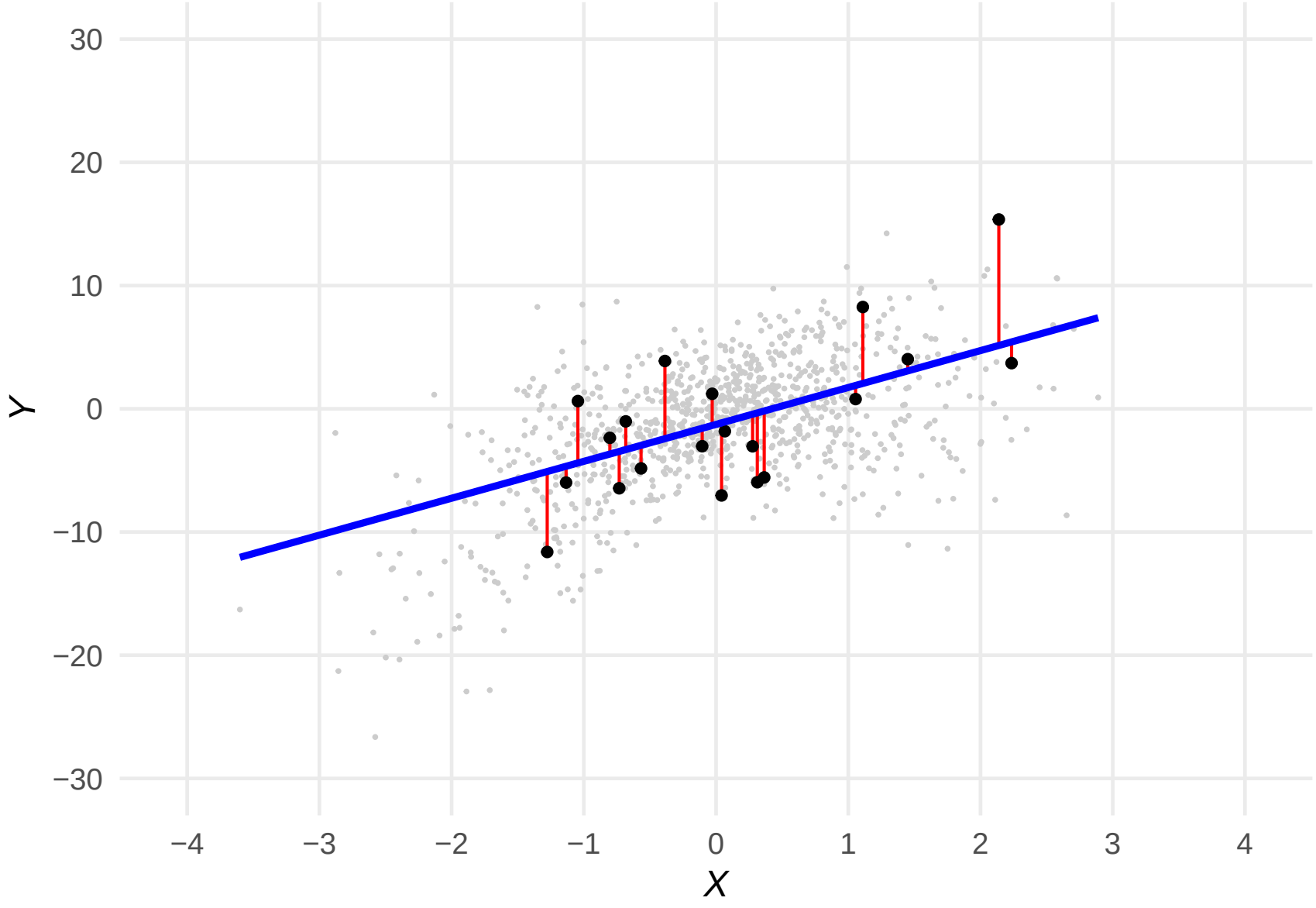
Evaluating linear regression for prediction



Continuous predictions with regression



Continuous predictions with regression



Quantifying performance of linear regression

The **mean squared error (MSE)** is commonly used to measure performance of predictions.

Suppose you trained your model on a dataset with n observations. Then:

$$\text{MSE} = \frac{1}{n}\sum_i (\underbrace{Y_i - \widehat{Y}_i}_{\text{error } (\hat{\varepsilon}_i)})^2 = \frac{1}{n}\sum_i \hat{\varepsilon}_i^2$$

Higher the MSE $\Rightarrow$ predictions are further from true values.

Side note: you saw something like this before...

- MSE is just the mean of the squared residuals: $\frac{1}{n}\text{SSR}$.
- And thus, OLS minimizes the MSE in addition to SSR.
- (Of course, assuming the regression model is correct.)

Quantifying performance of linear regression

Minor technical issue: some people don't like the fact that MSE is not in the same units as the Y variable.

➜ So, instead they might report **root MSE (RMSE)**.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_i (Y_i - \widehat{Y}_i)^2}$$

Bigger technical issue: (R)MSE is very sensitive to outliers and can thus make you think your model is worse than it is.

➜ Instead, you can calculate **mean absolute error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_i |Y_i - \widehat{Y}_i|$$

But, for all three: lower numbers means better performance.

Quantifying performance of linear regression

	Y	X	Prediction
1	2.145000	-0.1128886	-1.6066358
2	5.982464	0.7975038	1.1205693
3	1.368951	-0.5276081	-2.8489846
4	-1.216584	-0.2566335	-2.0372431
5	-4.858001	1.3644160	2.8188322
6	1.820742	0.5826611	0.4769784

Quantifying performance of linear regression

	Y	X	Prediction	Residual
1	2.145000	-0.1128886	-1.6066358	3.7516356
2	5.982464	0.7975038	1.1205693	4.8618951
3	1.368951	-0.5276081	-2.8489846	4.2179356
4	-1.216584	-0.2566335	-2.0372431	0.8206591
5	-4.858001	1.3644160	2.8188322	-7.6768336
6	1.820742	0.5826611	0.4769784	1.3437636

Quantifying performance of linear regression

```
mse <- (1/(nrow(df))) * sum((df$Y - df$Prediction)^2)
print(mse)
```

```
[1] 18.80203
```

```
rmse <- sqrt(mse)
print(rmse)
```

```
[1] 4.33613
```

```
mae <- (1/(nrow(df))) * sum(abs(df$Y - df$Prediction))
print(mae)
```

```
[1] 3.367366
```

Evaluating linear regression for prediction

Another way to calculate MSE/RMSE/MAE is using the linear regression model object (the `lm()` object).

```
mod.mse <- mean(mod$residuals^2)
print(mod.mse)
```

```
[1] 18.80203
```

```
mod.rmse <- sqrt(mean(mod$residuals^2))
print(mod.rmse)
```

```
[1] 4.33613
```

```
mod.mae <- mean(abs(mod$residuals))
print(mod.mae)
```

```
[1] 3.367366
```

Comparing models

MSE, MAE, etc. don't have a obvious benchmarks.

- ➜ Can't say "If MSE is below X, then we have a 'good' model."
- ➜ Unlike, e.g., statistical significance (p-value < 0.05).

So, why are these performance metrics useful?

- ➜ Mostly useful for examining *relative* performance.
- ➜ Compare predictions on same data for two different models.
- ➜ Which MSE is smaller?

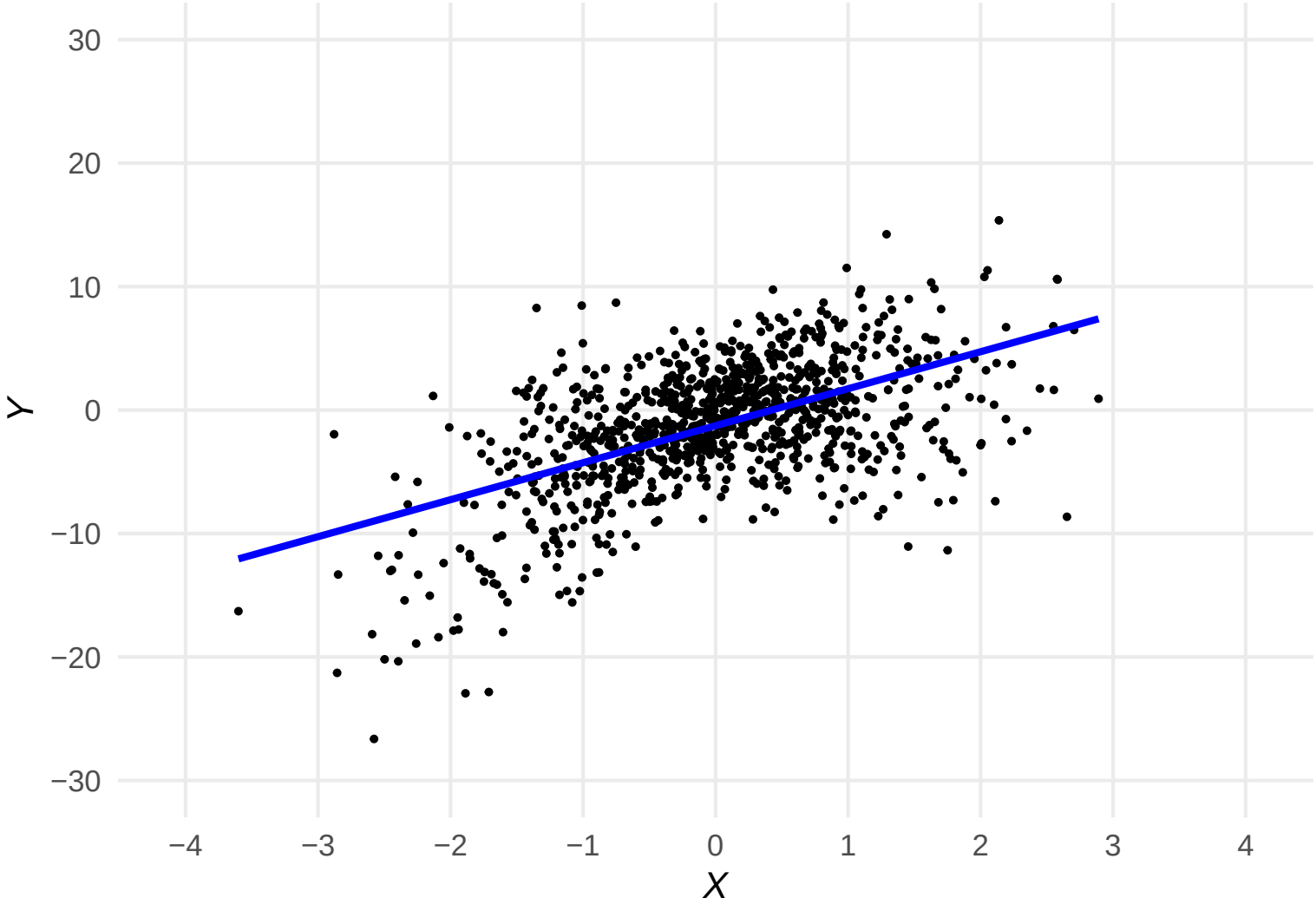
In real-world ML, this is often how statistics like this are used.

- ➜ If so, really doesn't matter if you choose MSE, MAE or RMSE.

We've only looked at one model, so there's no comparison (yet)...

Continuous predictions with regression

. . . but something doesn't look right:



Continuous predictions with regression

Relationship between X and Y does not seem to be linear.

Maybe we do not have a "good" model?

Looks like relationship between X and Y might be quadratic?

Maybe this formula is a better assumption about the DGP?

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2$$

Let's estimate in R and evaluate.

Continuous predictions with regression

```
mod2 <- lm(Y ~ X + I(X^2), data = df)
mod2.mse <- mean(mod2$residuals^2)
print(mod2.mse)
```

```
[1] 17.06452
```

Compare to the previous MSE:

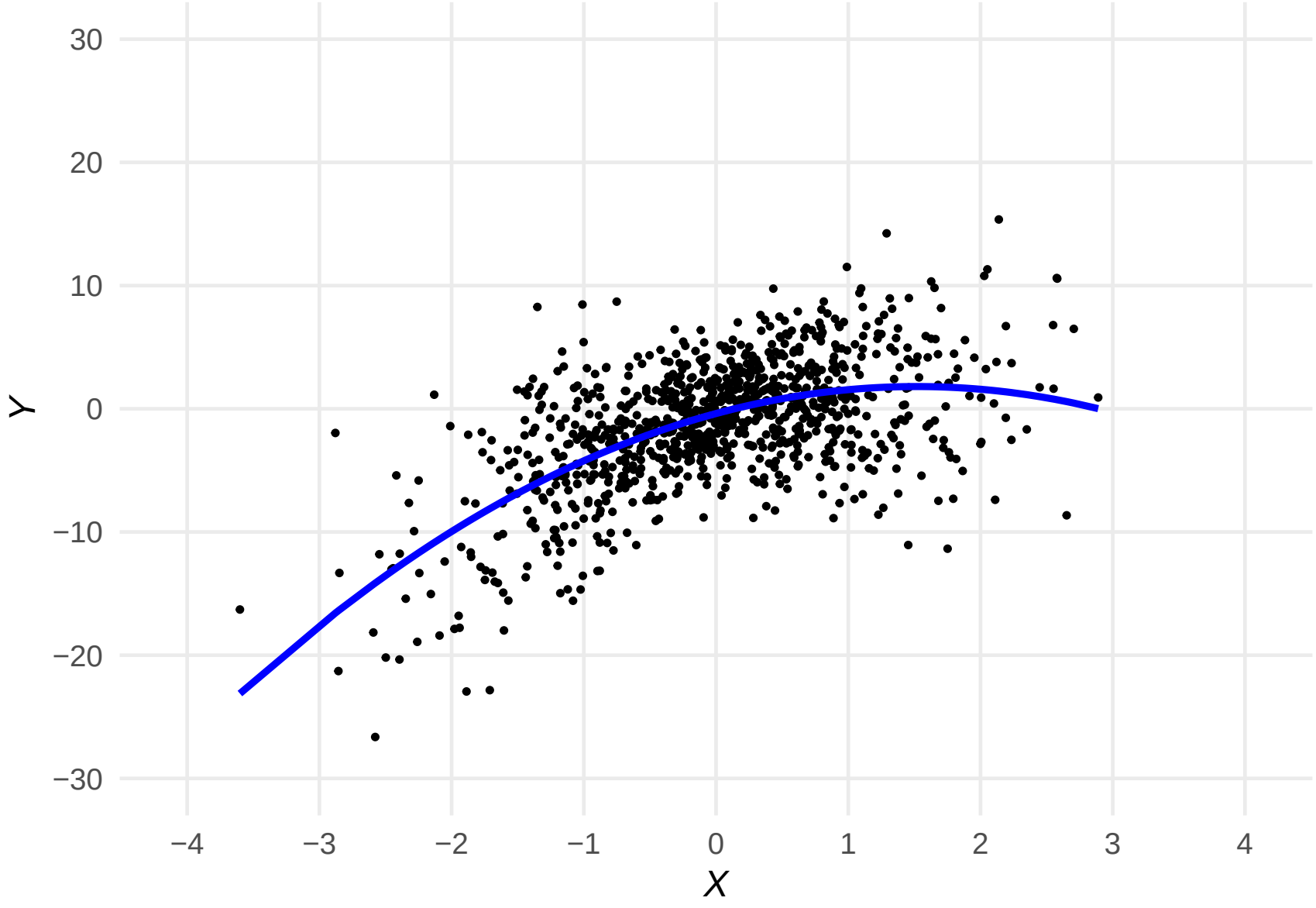
```
print(mod.mse)
```

```
[1] 18.80203
```

This new model does a better job predicting.

- ➜ More technically: it has a smaller MSE.
- ➜ Remember: lower MSE means predictions closer to the truth (on average).

Continuous predictions with regression



Performance metrics: categorical response (classification)

Quantifying performance of classifiers

Classifiers predict which class each observation is in.

- ➜ Recall: for classification Y has to be categorical.

In this case, MSE isn't the best way to measure performance.

- ➜ E.g., it assumes a continuous response, not categorical.

To build ideas, let's consider a simple example:

- ➜ A classification task with three classes, A, B and C.
- ➜ 100 labeled observations (i.e., we know each obs's class).
- ➜ A classifier has predicted each observation's class.

Confusion matrix

A **confusion matrix** contains counts of observations that are:

1. truly in each class, and
2. predicted to be in each class by the classifier.

Predicted class

True class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

With this matrix, you can calculate several metrics

Standard performance metrics

Accuracy: how correct is the classifier's identifications overall?

$$\text{Accuracy} = \frac{\text{Correctly classified}}{\text{Total number classified}}$$

How correct is my classifier's classifications of a particular class?

- **Precision** (for a class k): what proportion of observations classified in class k are truly in that class?

$$\text{Precision}_k = \frac{\text{Correctly classified as } k}{\text{Total number classified as } k}$$

- **Recall** (for a class): what proportion of observations in the class are correctly classified by the classifier?

$$\text{Recall} = \frac{\text{Correctly classified as } k}{\text{Total number truly in class } k}$$

Standard performance metrics

For a class k, you can also construct a composite measure of precision and recall (technically, the harmonic mean).

$$\mathrm{F1}_k = 2 \times \frac{\mathrm{Precision}_k \times \mathrm{Recall}_k}{\mathrm{Precision}_k + \mathrm{Recall}_k}$$

And since precision and recall are defined on a class-by-class basis, you can also create an F1 that is an average across classes

➜ You have to decide how to average for your specific purpose
See https://en.wikipedia.org/wiki/F-score#Extension_to_multi-class_classification

Warning: lots of people report precision, recall and F1 without explicitly saying which class – use context clues to figure it out.

Confusion matrix

True class \ Predicted class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

What is the accuracy of this classifier?

$$\text{Accuracy} = \frac{35 + 24 + 16}{120} = 0.625$$

In other words: what proportion of the observations were classified correctly by the classifier? $\approx 63\%$

Confusion matrix

True class \ Predicted class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

What is the precision of this classifier for class A?

$$\text{Precision}_A = \frac{35}{35+6+8} = 0.714$$

In other words: of the observations that my classifier classified as A, what proportion were actually in class A? $\approx 71\%$

Confusion matrix

True class \ Predicted class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

What is the recall of this classifier for class A?

$$\text{Recall}_A = \frac{35}{35 + 10 + 5} = 0.700$$

In other words: of the observations that are in class A, what proportion were classified correctly by the classifier? $\approx 70\%$

Confusion matrix

True class \ Predicted class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

Finally, what is the F1 of this classifier for class A?

$$\mathrm{F1}_A \approx 2 \times \frac{0.714 \times 0.700}{0.714 + 0.700} \approx 0.707$$

Again: this is a sort of average of precision and recall.

Confusion matrix

Predicted class

True class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

Can calculate precision, recall and F1 for the other classes too:

$$\text{Precision}_B = \frac{24}{10+24+7} = 0.585 \quad \text{Recall}_B = \frac{24}{6+24+9} = 0.615 \quad \text{F1}_B \approx 2\times\frac{0.585\times 0.615}{0.585+0.615} \approx 0.600$$

$$\text{Precision}_C = \frac{16}{5+9+16} = 0.533 \quad \text{Recall}_C = \frac{16}{8+7+16} = 0.516 \quad \text{F1}_C \approx 2\times\frac{0.533\times 0.516}{0.533+0.516} \approx 0.524$$

Which performance metric do I use?

Deciding which metric to focus on depends on your task

Example: spam filter

- ➜ If you want to make sure that messages sent to junk folder are actually spam: *precision*
- ➜ If you want to make sure that all spam ends up in the junk folder: *recall*

For an email provider, precision is (probably) more important

Report multiple metrics as they give different information

- ➜ Accuracy is often misleading. . .

Why you should report multiple statistics: spam

True class		Predicted class: Not Spam	Predicted class: Spam	
	Not Spam	**920**	60	**980**
	Spam	20	**20**	**40**
		940	**80**	**1020**

$$\text{Accuracy} = \frac{940}{1020} = 0.92$$

$$\text{Recall}_S = \frac{20}{20 + 20} = 0.50$$

$$\text{Precision}_S = \frac{20}{20 + 60} = 0.25$$

92% are accurately classified, but 50% of spam messages are getting through the filter and 75% of messages in the junk mail folder *aren't spam*(!)

Why you should report multiple statistics: illegal content

		Predicted class: Legal	Predicted class: Illegal	
True class	Legal	**850**	50	**900**
	Illegal	100	**50**	**150**
		950	**100**	**1050**

$$\text{Accuracy} = \frac{900}{1050} = 0.86$$

$$\text{Recall}_I = \frac{50}{50 + 100} = 0.33$$

$$\text{Precision}_I = \frac{50}{50 + 50} = 0.50$$

86% are accurately classified, but 66% of illegal content is getting through the filter and 50% of blocked content isn't illegal

Binary classification

Binary classification tasks are common, e.g.

- ➜ Spam versus not spam
- ➜ Illegal content versus legal content

We have some special (but confusing) terminology for such scenarios

		Predicted class: Positive	Predicted class: Negative
True class	Positive	True Positives	False Negatives (Type II error)
True class	Negative	False Positives (Type I error)	True Negatives

The confusing part: what's "positive" and what's "negative"?

- ➜ Convention: whatever you code as $Y_i = 1$ is "positive".

Predicted probabilities versus predicted classes

Classification tasks often yield *two* kinds of predictions.

For example, assume Y can be one of the classes in a set $\mathcal{K}$.

Then, for a observation i and a class $k \in \mathcal{K}$, we might get:

1. **Predicted probabilities**: $\widehat{\Pr}(Y_i = k \mid \mathbf{X}_i) \equiv \hat{p}_{ki}$ for all $k \in \mathcal{K}$
   - ➜ E.g., an algorithm might yield a model that tells you: there is a 0.78 probability that a specific email is spam.
   - ➜ It didn't directly tell you if it is "spam" or "not spam."
2. **Predicted class**: $\widehat{Y}_i \in \mathcal{K}$

Some algorithms estimate $\hat{p}_{ki}$ first, then $\widehat{Y}_i$ is derived from it.

Other algorithms estimate $\widehat{Y}_i$ first, then $\hat{p}_{ki}$ can be backed out of it.

Classification after prediction

Many common algorithms (like regression, shortly) do $\hat{p}_{ki} \Rightarrow \widehat{Y}_i$.

But how do they do it?

One intuitive option: to classify observation i, choose a class k with the highest $\hat{p}_{ik}$. In math: $\widehat{Y}_i = \arg\max_{k \in \mathcal{K}} \hat{p}_{ki}$.

- ➜ Typically what is done if there are ≥ 3 classes in $\mathcal{K}$.

For binary classification: implies choosing class with $\hat{p}_{ki} > 0.5$.

But using 0.5 doesn't always work well.

- ➜ For example: if you are trying to classify unbalanced classes (e.g., illegal content), 0.5 threshold won't give best predictions.
- ➜ All your predicted probabilities will be very low or very high.
- ➜ But even with relatively balanced classes, 0.5 won't be best.

Classification after prediction

Common approach (with balanced classes): choose threshold t that maximizes $\mathrm{TPR}(t) - \mathrm{FPR}(t)$.

- **True Positive Rate (TPR)**: $\frac{\text{true positives}}{\text{true positives+false negatives}}$
- **False Positive Rate (FPR)**: $\frac{\text{false positives}}{\text{false positives+true negatives}}$

Some boring terminology issues:

- TPR is recall for the positive class—called **sensitivity**
- FPR is 1 - recall for the negative class—called **specificity**

Because classes are usually imbalanced and tasks are different, this is an art.

Note: most R/python packages do something automatically (read the docs!).

ROC curves

Again note: TPR/FPR depend on threshold chosen.

So, for binary classification: common way to visualize how TPR and FPR vary as you change the threshold t is the **receiver operating characteristic (or ROC) curve**.

Example data set:

	y	x1	x2
1	1	1.3709584	2.3250585
2	0	-0.5646982	0.5241222
3	0	0.3631284	0.9707334
4	1	0.6328626	0.3769734
5	0	0.4042683	-0.9959334
6	1	-0.1061245	-0.5974829

ROC curves

Suppose we use a classifier that generates the following predicted probabilities.

	y	x1	x2	pred.prob
1	1	1.3709584	2.3250585	0.7940119
2	0	-0.5646982	0.5241222	0.2933375
3	0	0.3631284	0.9707334	0.5385136
4	1	0.6328626	0.3769734	0.5974207
5	0	0.4042683	-0.9959334	0.5087907
6	1	-0.1061245	-0.5974829	0.3822767

For example:

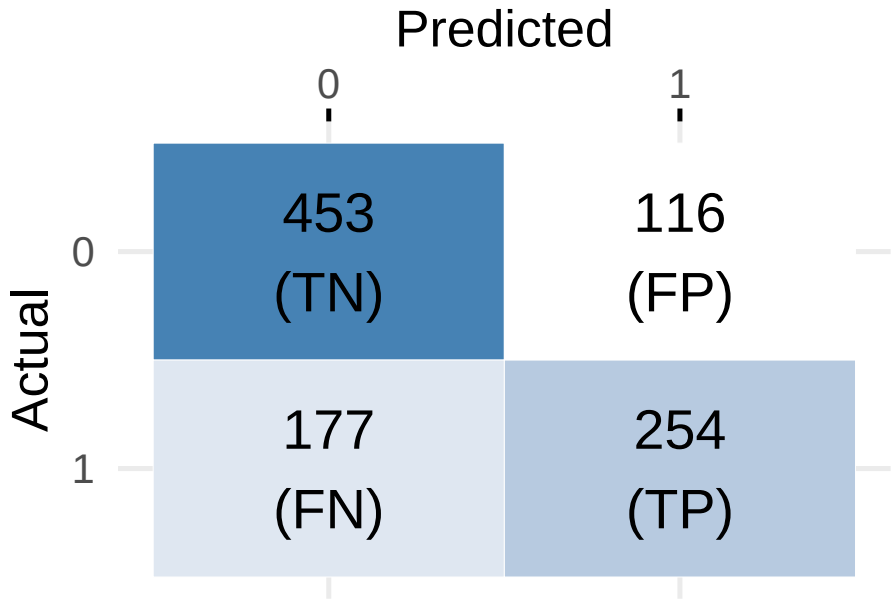
- For observation 1: predict a 0.794 probability that $Y_1 = 1$.
- For observation 2: predict a 0.293 probability that $Y_2 = 1$.

Seems pretty good so far.

ROC curves

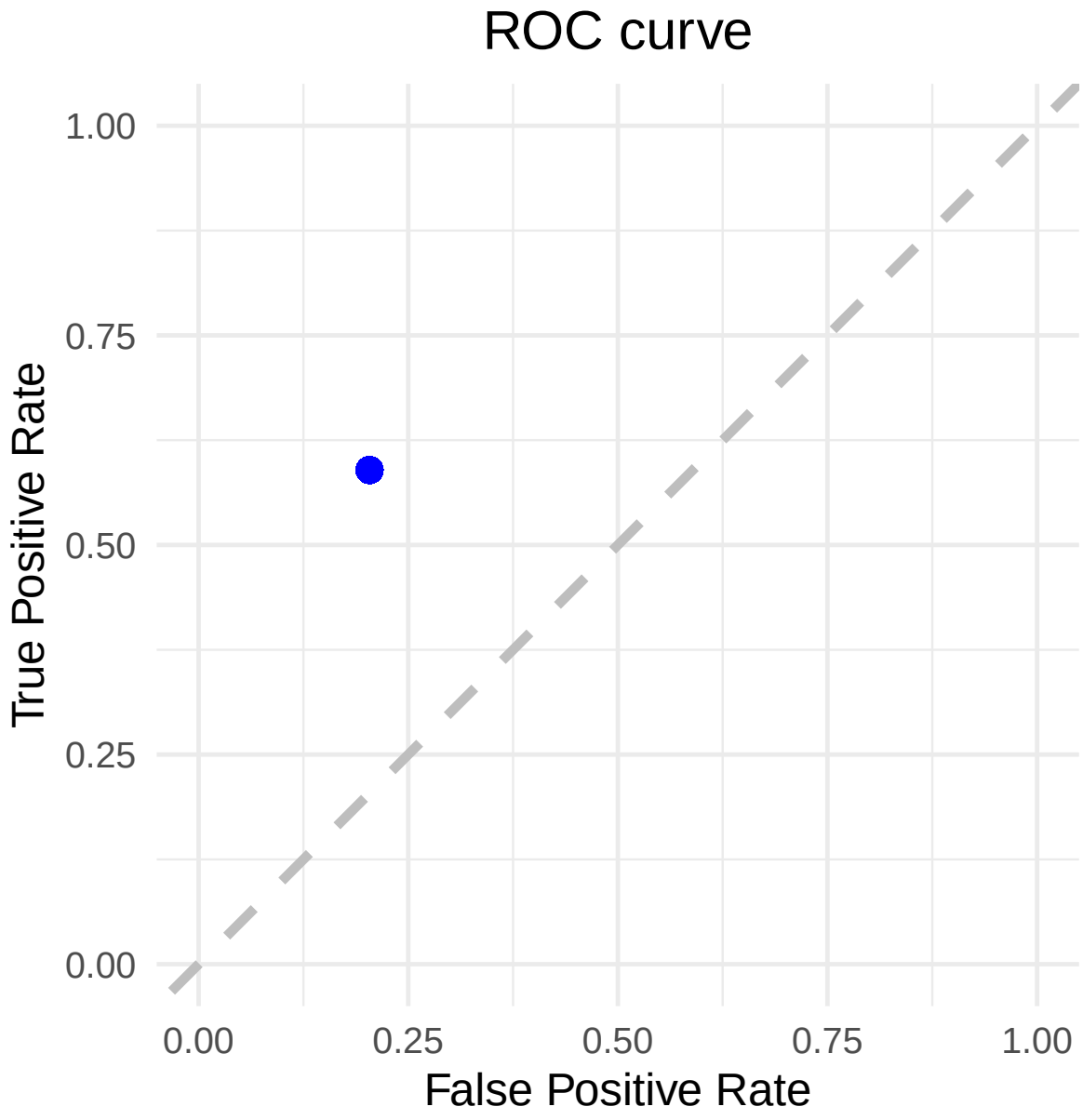
Now we want to predict classifications.

- Suppose we predict $\widehat{Y}_i = 1$ if predicted prob. > 0.5?
- How many would we predict correctly? Incorrectly?

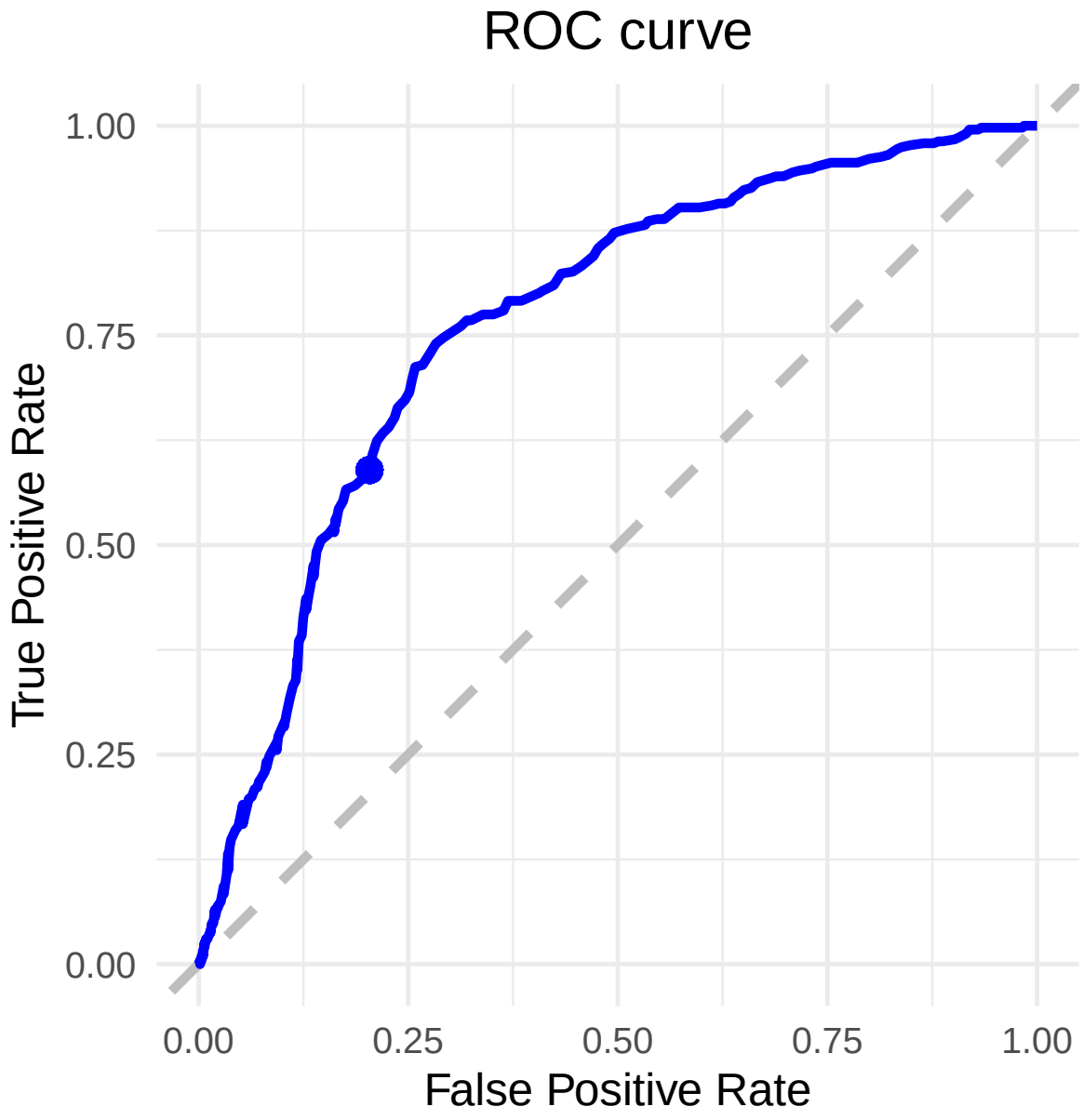


$\text{TPR} = \frac{TP}{TP+FN} = \frac{254}{254+177} = 0.589$ and $\text{FPR} = \frac{FP}{FP+TN} = \frac{116}{116+453} = 0.204$

ROC curves


ROC curve
True Positive Rate
False Positive Rate
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00

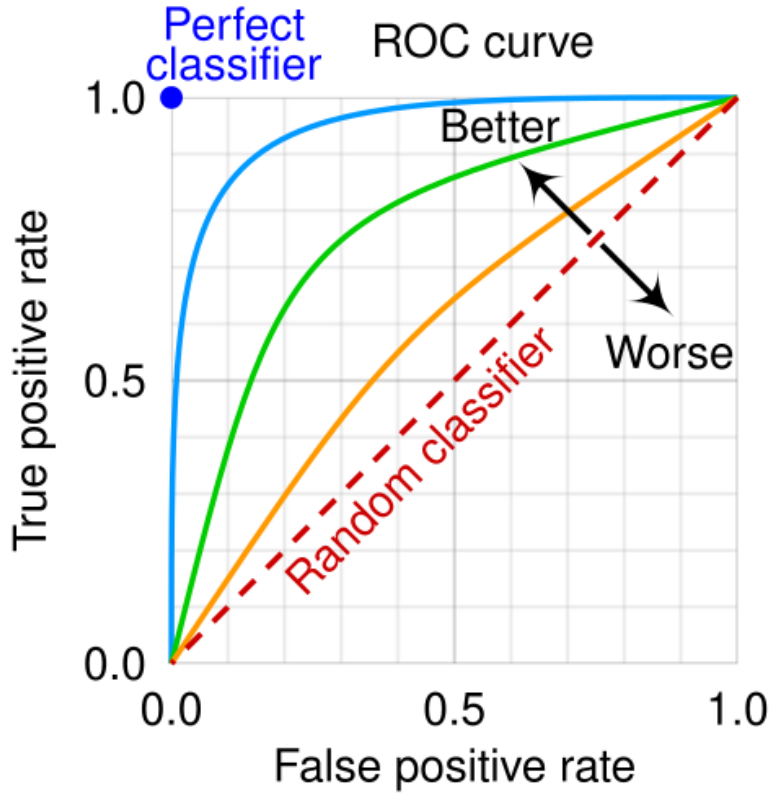
ROC curves


ROC curve
True Positive Rate
False Positive Rate
0.00
0.25
0.50
0.75
1.00
0.00
0.25
0.50
0.75
1.00

ROC curves

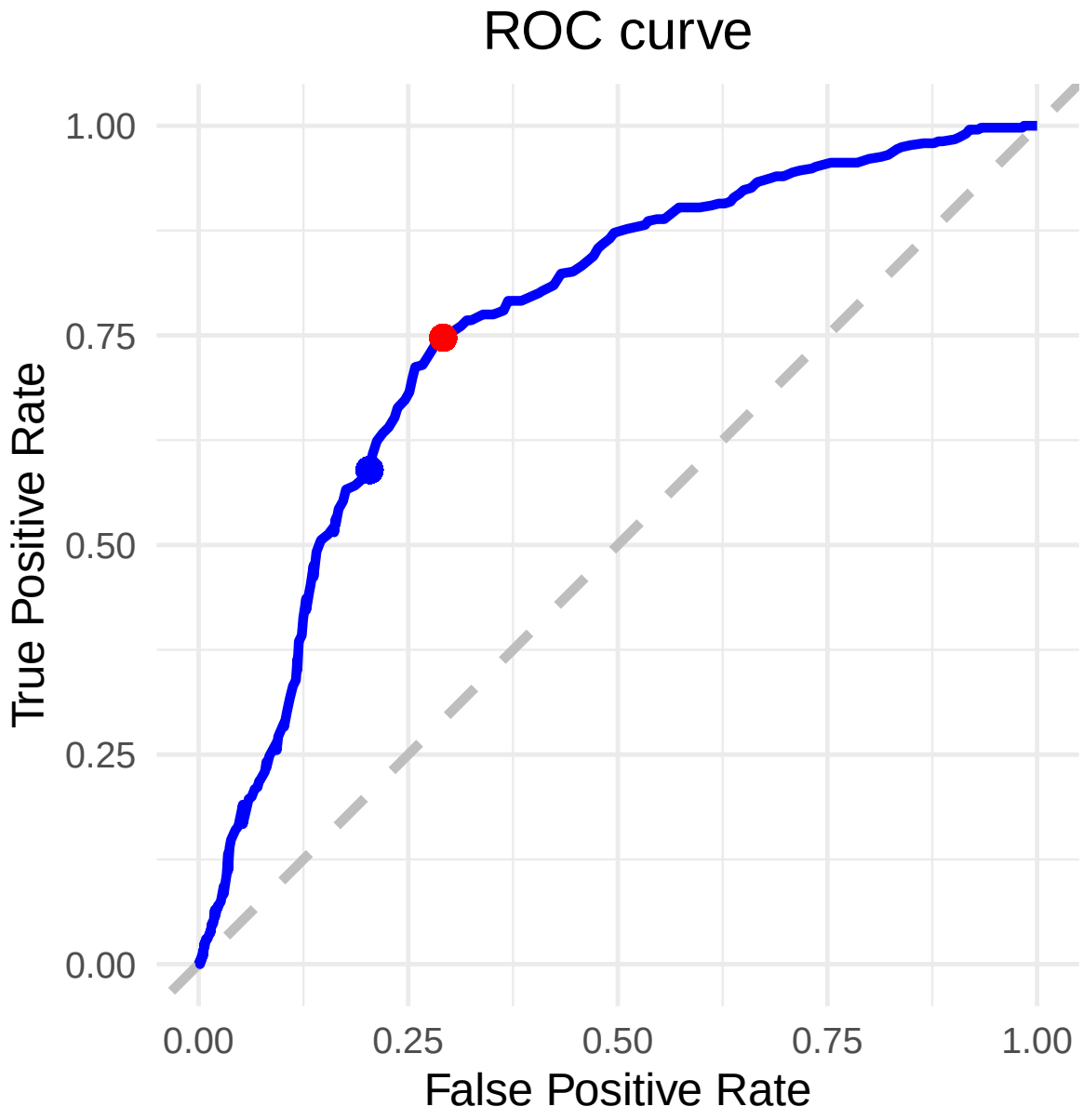
The *overall* shape gives us a good sense for how good the classifier does across all ts.

➜ If the classifier mostly classifies correctly *regardless of* t, then it's very good!



From Wikipedia

ROC curves


ROC curve
True Positive Rate
False Positive Rate
0.00
0.25
0.50
0.75
1.00
0.00
0.25
0.50
0.75
1.00

Area under the curve (AUC)

One common performance metric used for binary classifiers is the **area under the curve (AUC)**.

Literally: the area under the ROC curve.

Values range from 0 to 1:

- ➜ AUC of 0.5: model performs as good as random guessing (quite bad!).
- ➜ AUC of 1: model performs perfectly and never misclassifies.
- ➜ AUC less than 0.5: model performs worse than guessing

Downside: prioritizes types of errors equally — might want other performance metrics if you are concerned about one class or the other (e.g. illegal content example).

Side note: performance of predictions

We've been concerned with quantifying performance on predicted classifications (the $\widehat{Y}_i$s).

Sometimes, you want to know how well the predicted probabilities perform (the $\hat{p}_{ki}$s).

➜ Can't use confusion matrix, precision, recall, etc.

Continuous measures (MSE/RMSE/MAE) don't work well here.

There are some other measures you can use, e.g.:

Log-loss (**cross entropy**), for binary case with $\hat{p}_i = \widehat{\Pr}(Y_i = 1 \mid \mathbf{X}_i)$:

$$\text{LL} = -\frac{1}{n}\sum_i (Y_i \log(\hat{p}_i) + (1 - Y_i)\log(1 - \hat{p}_i))$$

(Not going to dwell on technical details here.)

Doing prediction: architecture of an algorithm

How does the machine learn?

"Machine learning" – machine is *learning* patterns in the data.

How does the machine "learn"?

By minimizing errors in its predictions:

- ➜ Minimizing the number of misclassifications.
- ➜ Minimizing residuals for regression.

Different **algorithms** contain different approaches to minimizing errors.

- ➜ Algorithm: a set of procedures rooted in statistical reasoning and computational principles.

Algorithms

Some algorithms start with an assumed DGP, and use a specified process to estimate empirical model of DGP.

- ➜ E.g., linear regression, naive Bayes.
- ➜ Typically, the specified process can be analytically shown to minimize errors with a fixed number of parameters.

Other algorithms don't start with an assumed DGP—they use rules or procedures to search for patterns or splits that minimize prediction error in a more flexible, data-driven way.

- ➜ E.g., decision trees and random forests.
- ➜ Often rely on heuristics or greedy search, with no fixed number of parameters.

Both types generate **(estimated) models**.

Models

A **model** is a formal, mathematical representation of a relationship between inputs and outputs.

- ➜ Inputs are the predictors and output is the response.
- ➜ Represent mathematically as *functions* e.g. $Y = f(X, \theta)$.

Conceptual distinction between theoretical and estimated models:

- ➜ A theoretical model is an assumption about the DGP.
- ➜ For a theoretical model, you don't have specific values for all parameters; just a general idea of relationships.
- ➜ The machine then calculates ("estimates") values for all parameters using the data.
- ➜ Again: not all ML algorithms start with theoretical models about the DGP, but they all end with *estimated* models!

Models

For example, you have some data on Y and X.

- ➜ You decide you think there's a linear relationship between Y and X in the DGP.
  - ➜ So your f looks like: $Y = \beta_0 + \beta_1 X + \varepsilon$.
- ➜ But you don't know the precise linear relationship in your data, as captured by the values of β_0 and β_1.
- ➜ You give your data to your machine and ask it to calculate an estimated model, something like: $\widehat{Y}_i = 0.25 + 1.61 X_i$.
- ➜ With this estimated model, you can do some stuff: prediction, inference, etc.

Parameters

Models typically have **parameters**:

1. Fixed parameters that are learned from the data.

   - ➜ These are estimated directly through model training.
   - ➜ They adjust automatically to minimize some loss or error.
   - ➜ Example: β coefficients in regression.
   - ➜ You don't always know how many there are in advance (e.g., trees).

2. Fixed parameters that are set in advance by the analyst.

   - ➜ These are called **hyperparameters**.
   - ➜ They are not "learned" by the machine — you specify them before training begins.
   - ➜ They control the model's structure/complexity/learning process.
   - ➜ E.g., the maximum depth of a decision tree (in random forests).

Classification with regression

Linear probability model

For *binary* classification where $Y_i \in \{0, 1\}$, then linear regression gives us a quantity that *approximates* a probability.

Mathematically, suppose we have a model that assumes:

$$\mathbb{E}[Y_i|X_{1i}, ..., X_{Ji}] = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji}$$

Since $Y_i \in \{0, 1\}$ implies $E[Y_i|\mathbf{X}_i] = \Pr(Y_i = 1|\mathbf{X}_i)$:

$$\Pr(Y_i = 1|X_{1i}, ..., X_{Ji}) = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji}$$

This is known as the **linear probability model**.

In practice: regular regression with a binary response variable.

Pros and cons

This is super easy: you already know how to estimate linear regressions.

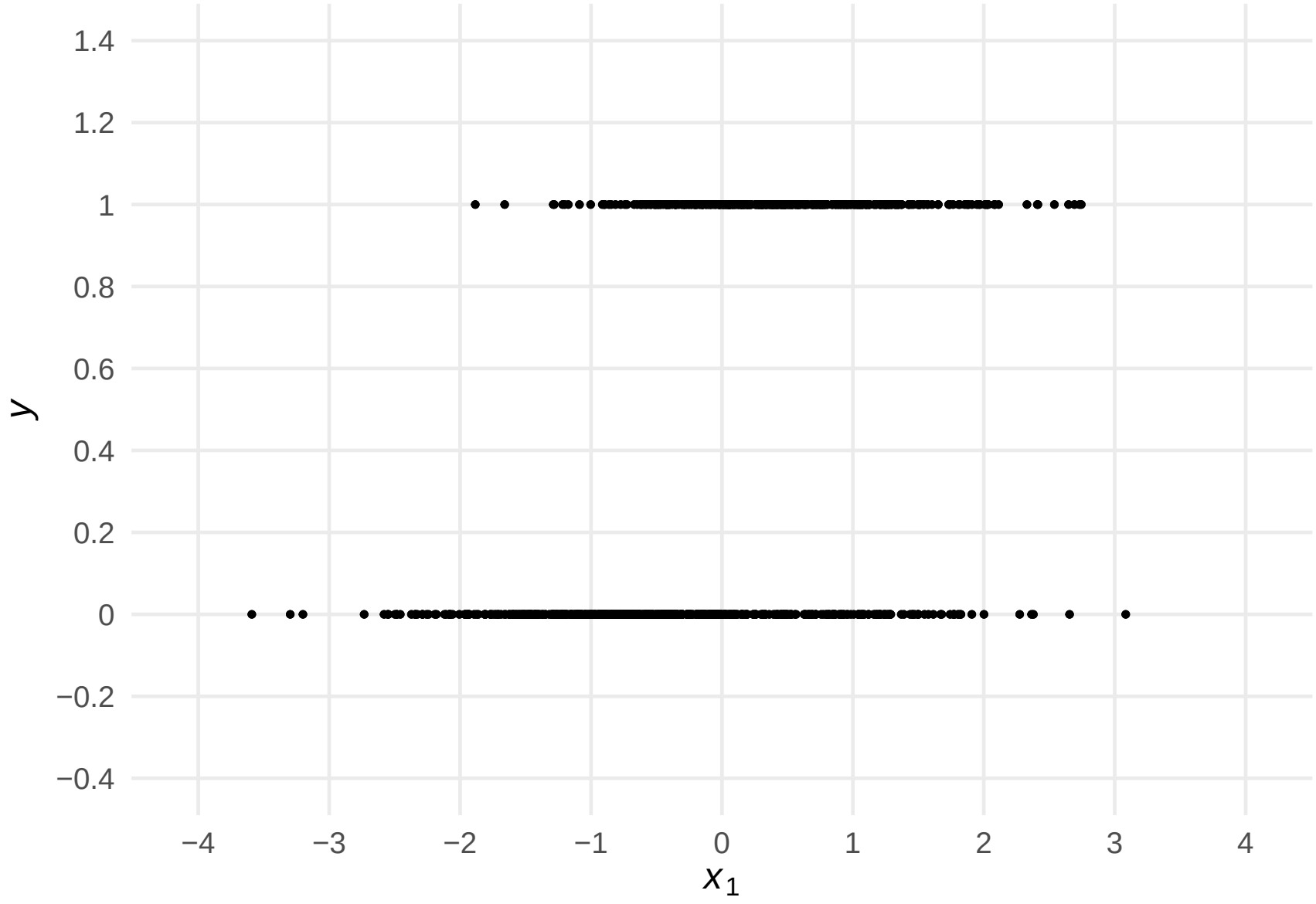
But there's a problem: $\beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji}$ is only an *approximation* of $\Pr(Y_i = 1 | X_{i1}, ..., X_{Ji})$.

Why?

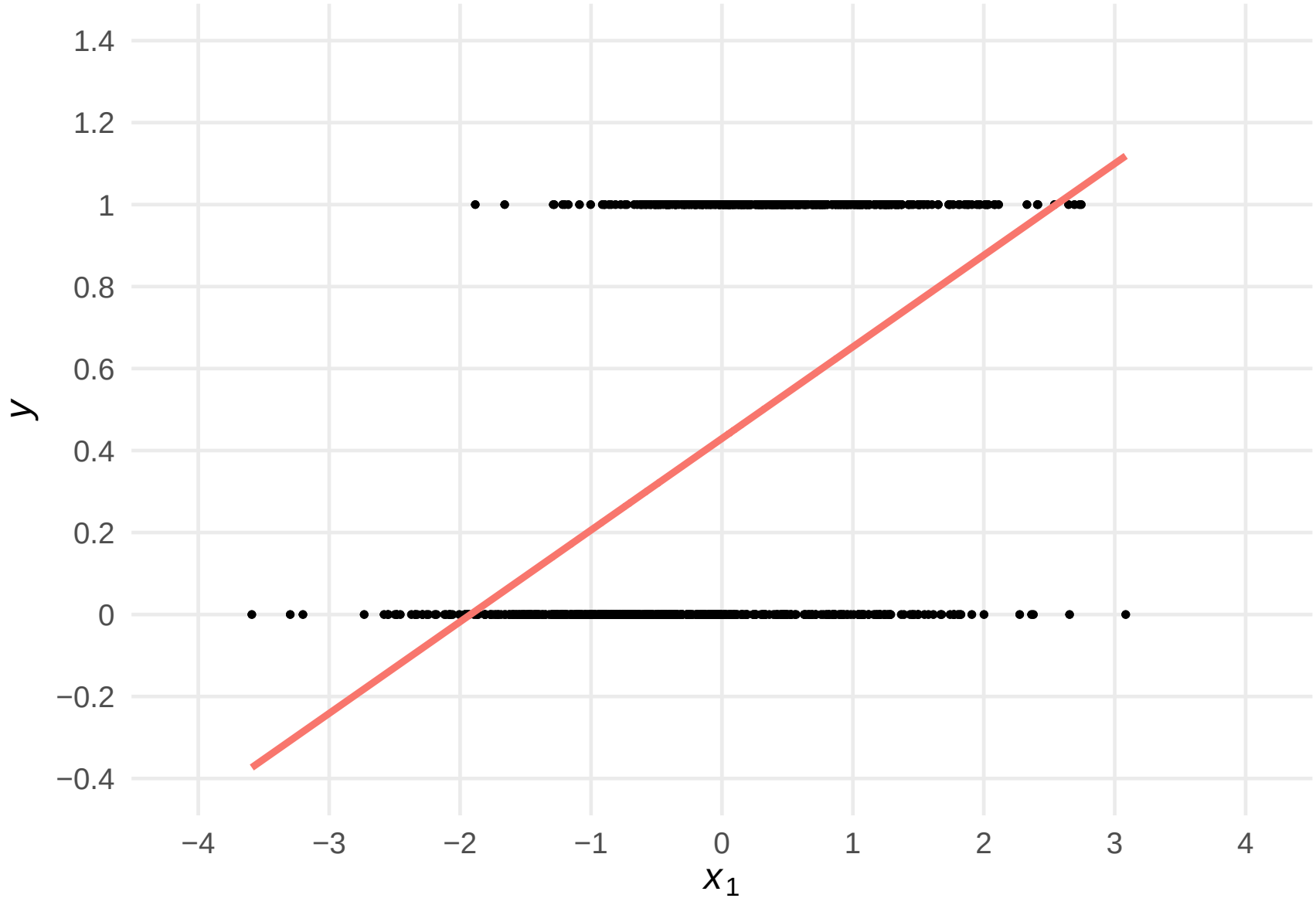
1. This kind of linear function is not necessarily bounded between 0 and 1. It is illegal to have probabilities outside $[0, 1]$!

2. Regression's assumption on error term is wrong for binary response, $Y_i \in \{0, 1\}$.

Example



Example



Logistic regression ("logit")

The standard way to modify linear regression to constrain predictions to be between 0 and 1 is **logistic regression** (or "logit").

$$\Pr(Y_i = 1|X_1, X_2, ..., X_J) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji})}}$$

Some algebra gives you:

$$\log\left(\frac{\Pr(Y_i = 1|X_{1i}, ..., X_{Ji})}{1 - \Pr(Y_i = 1|X_{1i}, ..., X_{Ji})}\right) = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji}$$

And since $\hat{p}_i = \Pr(Y_i = 1|X_{1i}, ..., X_{Ji})$:

$$\underbrace{\log\left(\frac{\hat{p}_i}{1 - \hat{p}_i}\right)}_{\text{log-odds}} = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_J X_{Ji}$$

Note: this $\log$ is to the base e—the natural log $\ln(\cdot)$.

Logistic regression ("logit")

```r
est.logit.model <- glm(y ~ x1, data = df, family = binomial)
print(summary(est.logit.model))
```

```
Call:
glm(formula = y ~ x1, family = binomial, data = df)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.36422    0.07285   -5.00 5.74e-07 ***
x1           1.17290    0.08989   13.05  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1361.2  on 999  degrees of freedom
Residual deviance: 1124.3  on 998  degrees of freedom
AIC: 1128.3

Number of Fisher Scoring iterations: 4
```

Logistic regression ("logit")

R has given us estimates: $\beta_0 \approx -0.228$ and $\beta_1 \approx 1.050$.

So, we have an estimated model we can use to make predictions:

$$\log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right) \approx -0.228 + 1.050\, X_1$$

How do we make a prediction for a specific observation where $X_1 = 2$?

$$\log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right) \approx -0.228 + 1.050 \times 2$$

Some boring algebra gets you $\hat{p}_i \approx 0.867$.

Logistic regression ("logit")

Of course, don't have to calculate by hand—let's use R:

```
predict(est.logit.model, newdata = data.frame(x1 = 2))
```

```
       1
1.981587
```

Oops, this is not between 0 and 1 either! It's the log-odds.

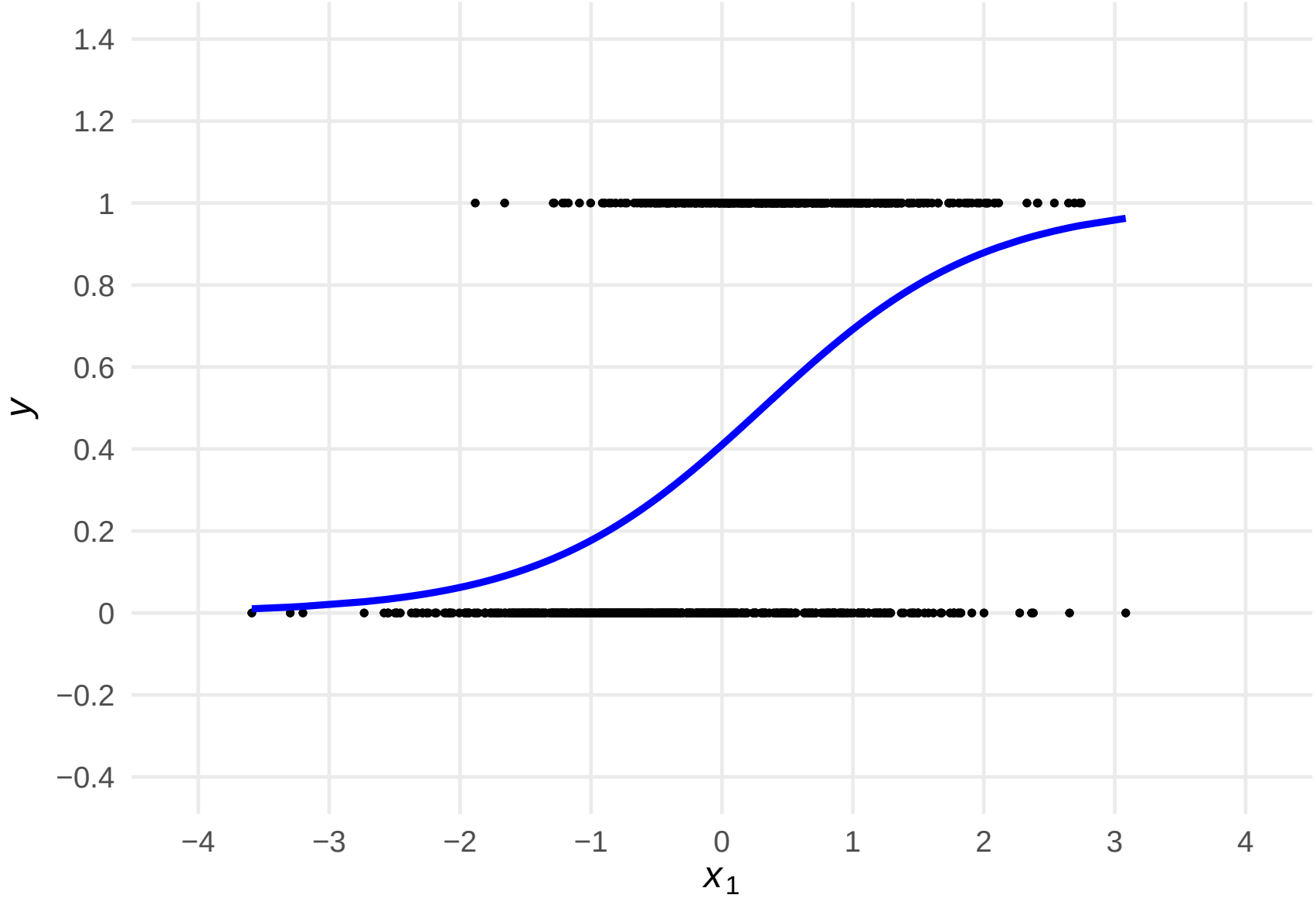
Need to ask for the predicted probability:

```
predict(est.logit.model, newdata = data.frame(x1 = 2),
        type = "response")
```

```
        1
0.8788502
```

By default, R `predict()` gives you $\log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right)$, not $\hat{p}_i$.

Logistic regression ("logit")



Side note: prediction versus inference

Sometimes you hear people say: "If your dependent variable is binary, you always have to use logit (or probit)."

This is not true. That's why I showed you LPM!

If your goal is predicting the response variable, then yes, you should use probably logit.

But if your goal is inference (e.g., estimating causal effects, $\hat{\beta}_1$), then LPM is fine and probably preferrable.

I'll add: even the benefit of logit over LPM for prediction is overstated.

Classification with naive Bayes

DGP for a naive Bayes classifier

Given predictors $X_{1i}, \ldots, X_{Ji}$, we want to predict the observation's class.

Let's keep assuming Y_i is binary, so $Y_i = \{0, 1\}$.

➜ Keep in mind: this is not necessary to use naive Bayes.

This means we'll, again, estimate $\hat{p}_i = \Pr(Y_i = 1 \mid \mathbf{X}_i)$.

Using Bayes' theorem, we can write:

$$\hat{p}_i = \Pr(Y_i = 1 \mid \mathbf{X}_i) = \frac{\Pr(Y_i = 1) \Pr(\mathbf{X}_i \mid Y_i = 1)}{\Pr(\mathbf{X}_i)}$$

DGP for a naive Bayes classifier

We can expand out the denominator to get:

$$\hat{p}_i = \Pr(Y_i = 1 \mid \mathbf{X}_i) = \frac{\Pr(Y_i = 1)\Pr(\mathbf{X}_i \mid Y_i = 1)}{\Pr(Y_i = 1)\Pr(\mathbf{X}_i \mid Y_i = 1) + \Pr(Y_i = 0)\Pr(\mathbf{X}_i \mid Y_i = 0)}$$

The right hand side has four quantities in it:

- Two **priors**, $\Pr(Y_i = 1)$ and $\Pr(Y_i = 0)$.
  - These capture *class prevalence*.
- Two **likelihood functions**, $\Pr(\mathbf{X}_i \mid Y_i = 1)$ and $\Pr(\mathbf{X}_i \mid Y_i = 0)$.
  - These capture *class-specific models (DGPs) for the predictors*.

Class prevalence is easy to estimate, but not the class-specific models.

Specifying a class specific model

$\Pr(\mathbf{X}_i \mid Y_i = 1)$ and $\Pr(\mathbf{X}_i \mid Y_i = 0)$ are class specific models.

To make progress, we need to make some more specific assumptions.

1. We make things simple by making a **naive assumption**:

➜ Conditional on class, each predictor is drawn independently.

From the rules of probability this means that:

$$\Pr(\mathbf{X}_i \mid Y_i = 1) = \prod_{j=1}^{J} \Pr(X_{ji} \mid Y_i = 1) \quad \Pr(\mathbf{X}_i \mid Y_i = 0) = \prod_{j=1}^{J} \Pr(X_{ji} \mid Y_i = 0)$$

Specifying a class specific model

$\Pr(\mathbf{X}_i \mid Y_i = 1)$ and $\Pr(\mathbf{X}_i \mid Y_i = 0)$ are class specific models.

To make progress, we need to make some more specific assumptions.

2. We assume $X_{ji} \mid Y_i$ is distributed according to a specific distribution:

   - ➜ Continuous predictors: normal distribution.
   - ➜ Count data predictors: multinomial distribution.
   - ➜ Binary predictors: Bernoulli distribution.

Wikipedia tells you the formula for $\Pr(X_{ji} \mid \cdot)$ for each.

E.g., for a binary X_j:

$$\Pr(X_{ji} \mid Y_i = 1) = \Pr(X_{ji} = 1 \mid Y_i = 1)^{X_{ji}} \left(1 - \Pr(X_{ji} = 1 \mid Y_i = 1)\right)^{1 - X_{ji}}$$

Estimating a naive Bayes classifier

So far, I've walked you through assumptions on a DGP based on:

- Bayes theorem, plus
- a "naive" assumption that individual predictors are drawn independently, conditional on class, plus
- specific distributions based on the type of data.

The four quantities can all be estimated with a dataset using standard results from statistics (review day 3/4).

First, class prevalence:

$$\widehat{\Pr}(Y_i = 1) = \frac{1}{n}\sum_i Y_i \quad \text{and} \quad \widehat{\Pr}(Y_i = 0) = \frac{1}{n}\sum_i (1 - Y_i)$$

Estimating a naive Bayes classifier

Second, the class prevalence.

- Let's assume a binary Y and continuous X_j.
- Different calculations for other kinds of data, same process.

Assume a normal distribution for the likelihood function:

$$\Pr(X_{ji} \mid Y_i = 1) = \underbrace{\frac{1}{\sqrt{2\pi\sigma_{j,1}^2}} \exp\left(-\frac{(X_{ji} - \mu_{j,1})^2}{2\sigma_{j,1}^2}\right)}_{\text{pdf of } \; X_{ji} \mid Y_i = 1 \sim \mathcal{N}(\mu_{j,1}, \sigma_{j,1}^2)}$$

For each predictor X_j we estimate sample mean of X_j among observations where $Y_i = 1$:

$$\hat{\mu}_{j,1} = \frac{1}{\sum_i Y_i} \sum_i Y_i X_{ji}$$

As well as the sample variance:

$$\hat{\sigma}_{j,1}^2 = \frac{1}{\sum_i Y_i - 1} \sum_i Y_i \left(X_{ji} - \hat{\mu}_{j,1}\right)^2$$

A very simple example

Binary response and predictors, $Y_i, X_{1i}, X_{2i} \in \{0, 1\}$.

A preview:

	y	x1	x2
1	1	1	1
2	0	0	0
3	1	1	0
4	0	0	0
5	1	0	0
6	1	0	0
7	0	0	1
8	1	0	1
9	0	0	0
10	0	0	0

A very simple example

There are five parameters to estimate:

- The prior:
  - $\Pr(Y_i = 1)$
- Four likelihoods:
  - $\Pr(X_{1i} = 1 \mid Y_i = 1)$
  - $\Pr(X_{1i} = 1 \mid Y_i = 0)$
  - $\Pr(X_{2i} = 1 \mid Y_i = 1)$
  - $\Pr(X_{2i} = 1 \mid Y_i = 0)$

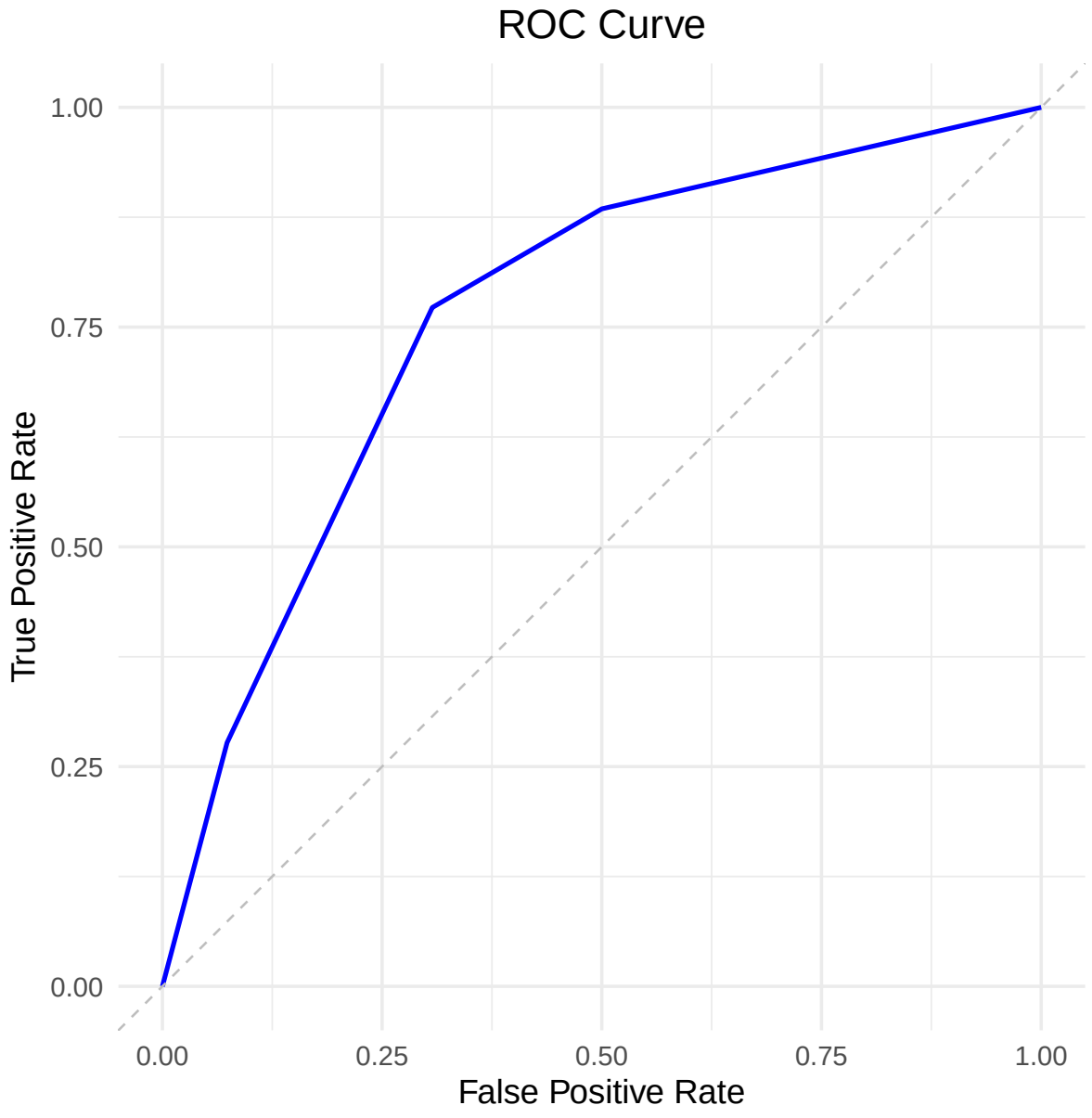
A very simple example

Can calculate things by hand, but also can run in R:

```
library(e1071)
model <- naiveBayes(y ~ x1 + x2, data = df, laplace = 0)
df$pred.probs <- predict(model, newdata = df, type = "raw")[,1]
df$pred.labels <- predict(model, newdata = df, type = "class")
print(head(df[,c("y","pred.labels")]))
```

```
  y pred.labels
1 1           1
2 0           0
3 1           1
4 0           0
5 1           0
6 1           0
```

A very simple example



Naive Bayes classifier

The assumptions of naive Bayes are obviously wrong

- ➜ The simplicity of the model makes it easier to estimate, but also more limited in its ability to provide accurate classifications.
- ➜ One issue: because it (purposefully) ignores correlations between predictors in observations, it can yield incorrect estimates of the proportion of observations in a class.
- ➜ But it still largely works for classifying individual observations.

It is very flexible!

Practical reality: not used often.