Lecture 006

Classification

Edward Rubin
13 February 2020

Admin

Admin

Material

Last time Shrinkage methods

- Ridge regression
- (The) lasso
- Elasticnet

Today Classification methods

- Introduction to classification
- Linear probability models
- Logistic regression

Also: Class will end today at 11:30am.[†]

[†] 🎉 ?

Admin

Upcoming

Readings *Today* *ISL* Ch. 4

Problem sets

- *Shrinkage methods* Due today
- *Classification* Due next week

Classification

Classification

Intro

Regression problems seek to predict the number an outcome will take—integers (*e.g.*, number of cats), reals (*e.g.*, home/cat value), *etc.* †

Classification problems instead seek to predict the category of an outcome

- **Binary outcomes**
  success/failure; true/false; A or B; cat or *not cat*; *etc.*

- **Multi-class outcomes**
  yes, no, *or maybe*; colors; letters; type of cat;†† *etc.*

This type of outcome is often called a *qualitative* or *categorical* response.

† Maybe: Binary indicators... †† It turns out, all of machine learning is about cats.

Classification

Examples

For the past few weeks, we've been immersed in regression problems.

It's probably helpful to mention a few **examples of classification problems**.

- Using life/criminal history (and demographics?):
  Can we predict whether a defendant is **granted bail**?

- Based upon a set of symptoms and observations:
  Can we predict a patient's **medical condition**(s)?

- From the pixels in an image:
  Can we classify images as **bagel, puppy, or other**?

Classification

Approach

One can imagine two[†] related **approaches to classification**

1. Predict **which category** the outcome will take.

2. Estimate the **probability of each category** for the outcome.

That said, the general approach will

- Take a set of training observations $(x_1, y_1),\ (x_2, y_2),\ \dots,\ (x_n, y_n)$
- Build a classifier $\hat{y}_o = f(x_o)$

all while balancing bias and variance.[††]

[†] At least. [††] Sound familiar?

Q If everything is so similar, can't we use regression methods?

Q If everything is so similar, can't we use regression methods?

A *Sometimes. Other times:* No. Plus you still need new tools.

Classification

Why not regression?

Regression methods are not made to deal with **multiple categories**.

Ex. Consider three medical diagnoses: stroke, overdose, and seizure.

Regression needs a numeric outcome—how should we code our categories?

Option 1

$$Y = \begin{cases} 1 & \text{if stroke} \\ 2 & \text{if overdose} \\ 3 & \text{if seizure} \end{cases}$$

Option 2

$$Y = \begin{cases} 1 & \text{if overdose} \\ 2 & \text{if stroke} \\ 3 & \text{if seizure} \end{cases}$$

Option 3

$$Y = \begin{cases} 1 & \text{if seizure} \\ 2 & \text{if stroke} \\ 3 & \text{if overdose} \end{cases}$$

The categories' ordering is unclear—let alone the actual valuation.
The choice of ordering and valuation can affect predictions. 😿

Classification

Why not regression?

As we've seen, **binary outcomes** are simpler.

Ex If we are only choosing between stroke and overdose

Option 1

$$Y = \begin{cases} 0 & \text{if stroke} \\ 1 & \text{if overdose} \end{cases}$$

and

Option 2

$$Y = \begin{cases} 0 & \text{if overdose} \\ 1 & \text{if stroke} \end{cases}$$

will provide the same results.

Classification

Why not regression?

In these **binary outcome** cases, we *can* apply linear regression.

These models are called linear probability models (LPMs).

The **predictions** from an LPM

1. estimate the conditional probability $y_i = 1$, *i.e.*, $\Pr(y_o = 1 \mid x_o)$
2. are not restricted to being between 0 and 1[†]
3. provide an ordering—and a reasonable estimate of probability

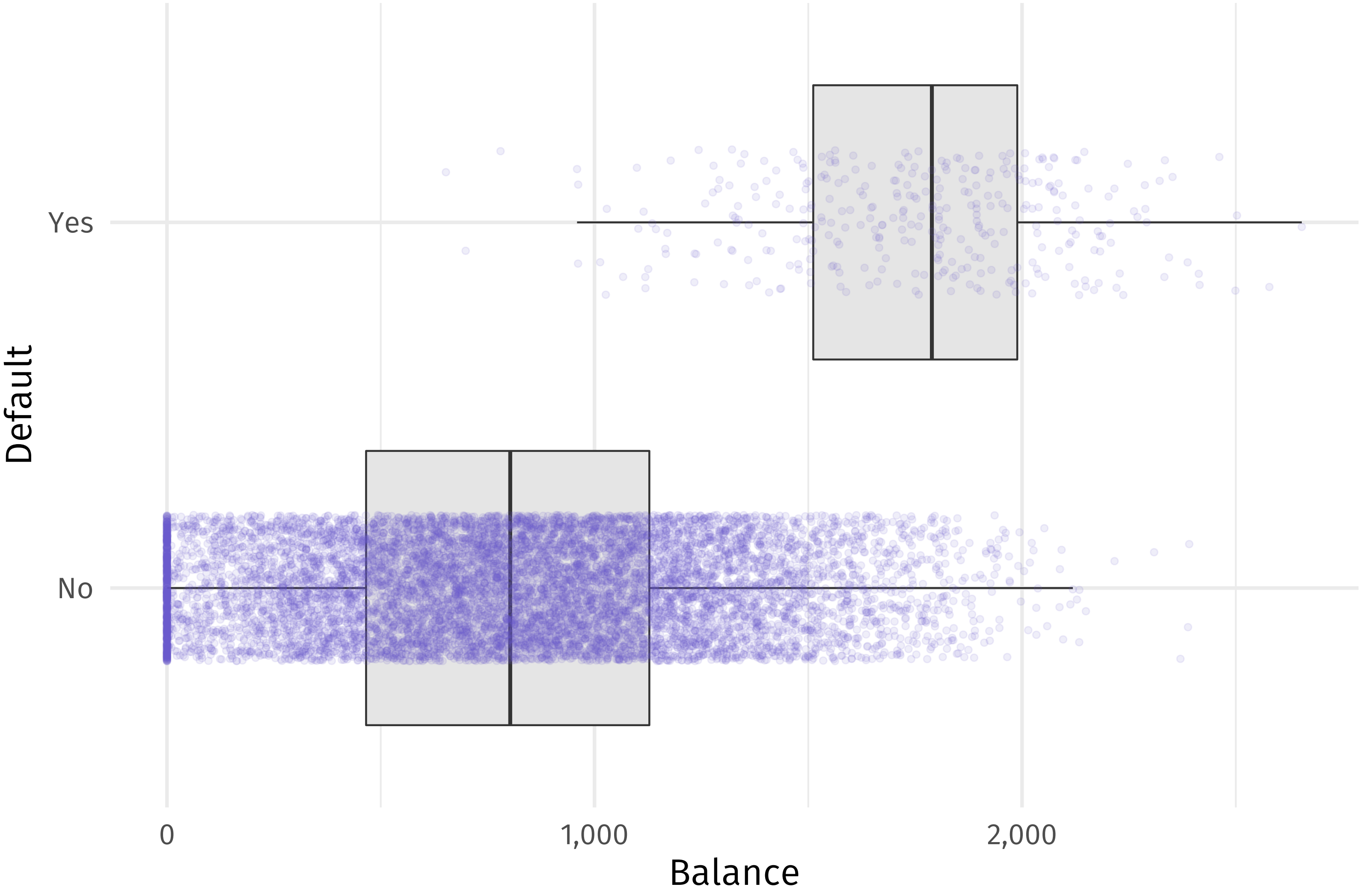
Other benefits: Coefficients are easily interpreted + we know how OLS works.

[†] Some people get very worked up about this point.

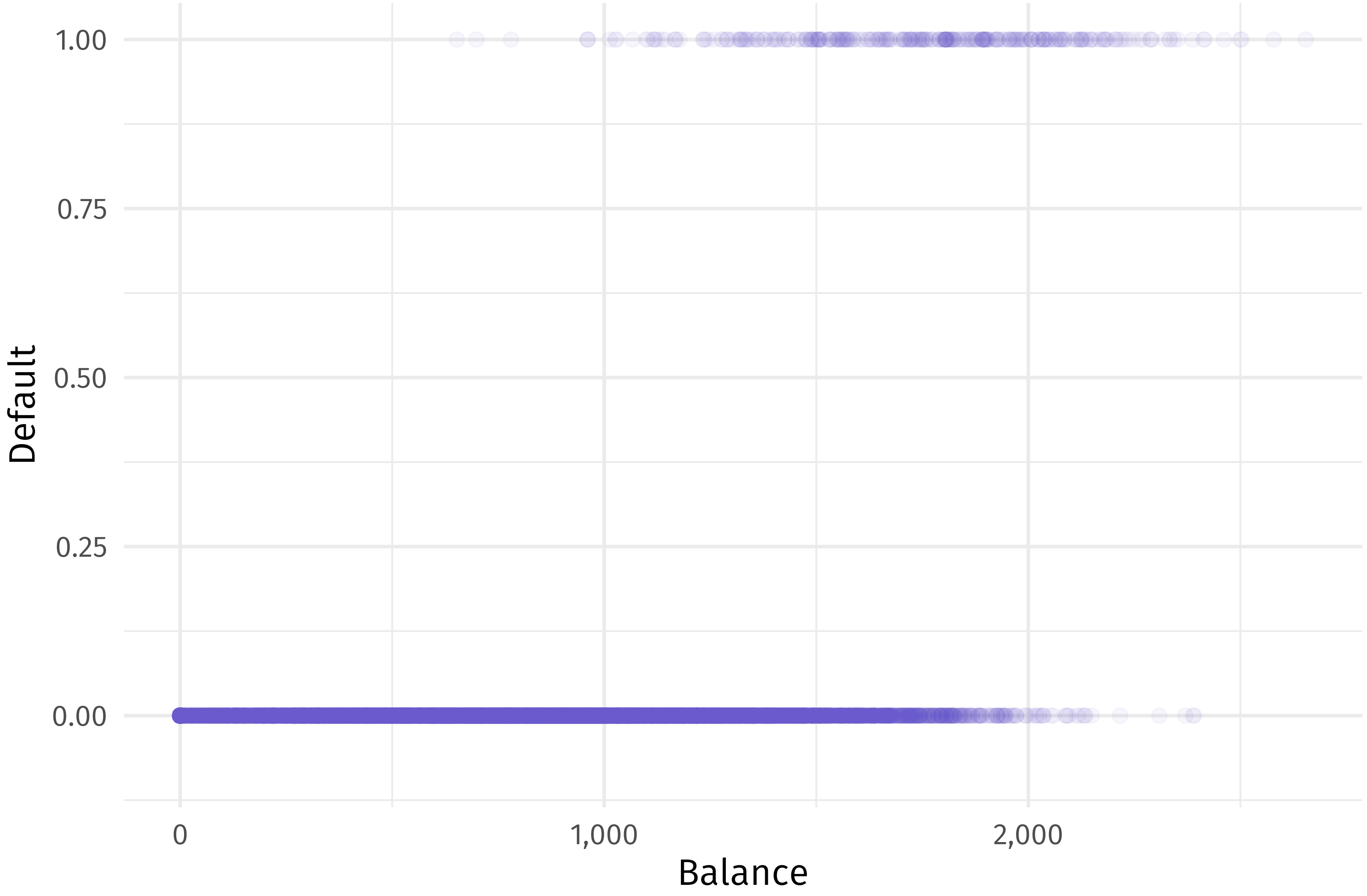
Let's consider an example: the `Default` dataset from `ISLR`

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
No	No	230.87	32,799

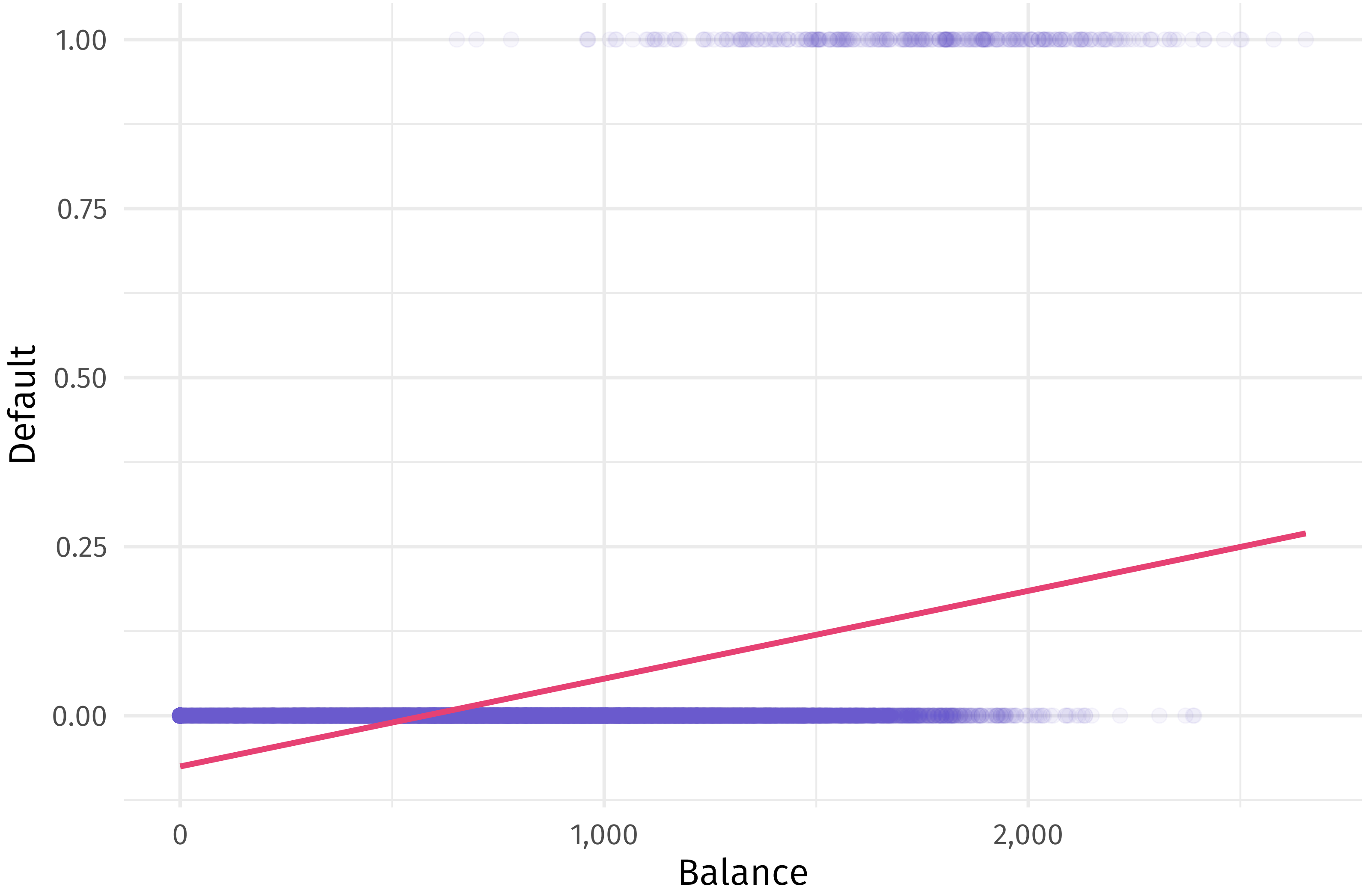
The data: The outcome, default, only takes two values (only 3.3% default).



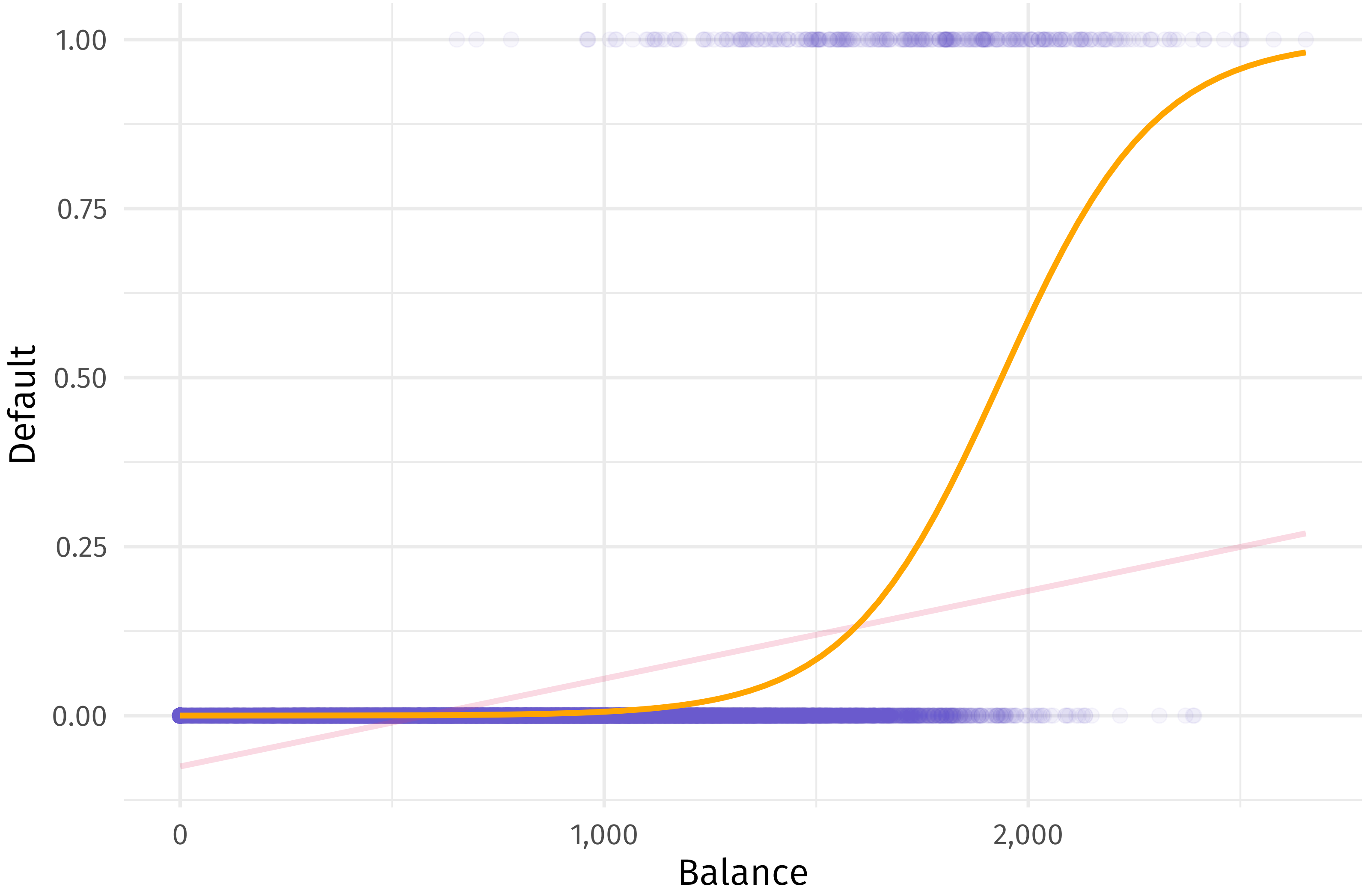
The data: The outcome, default, only takes two values (only 3.3% default).



The linear probability model struggles with prediction in this setting.



Logistic regression *appears* to offer an improvement.



So... what's logistic regression?

Logistic regression

Logistic regression

Intro

Logistic regression **models the probability** that our outcome Y belongs to a **specific category** (often whichever category we think of as `TRUE`).

For example, we just saw a graph where

$$\Pr(\text{Default} = \text{Yes}|\text{Balance}) = p(\text{Balance})$$

we are modeling the probability of `default` as a function of `balance`.

We use the **estimated probabilities** to **make predictions**, *e.g.*,

- if $p(\text{Balance}) \geq 0.5$, we could predict "Yes" for Default
- to be conservative, we could predict "Yes" if $p(\text{Balance}) \geq 0.1$

Logistic regression

What's *logistic*?

We want to model probability as a function of the predictors ($\beta_0 + \beta_1 X$).

Linear probability model

linear transform. of predictors

$$p(X) = \beta_0 + \beta_1 X$$

Logistic model

logistic transform. of predictors

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

What does this *logistic function* $\left(\frac{e^x}{1+e^x}\right)$ do?

1. ensures predictions are between 0 ($x \rightarrow -\infty$) and 1 ($x \rightarrow \infty$)

2. forces an S-shaped curved through the data (not linear)

Logistic regression

What's *logistic*?

With a little math, you can show

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \implies \log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

New definition: **log odds**[†] on the RHS and **linear predictors** on the LHS.

1. **interpretation** of β_j is about log odds—not probability
2. **changes in probability** due to X depend on level of X[†]

[†] The "log odds" is sometimes called "logit". [††] It's nonlinear!

Logistic regression

Estimation

Before we can start predicting, we need to estimate the β_js.

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \implies \log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X$$

We estimate logistic regression using **maximum likelihood estimation**.

Maximum likelihood estimation (MLE) searches for the β_js that make our data "most likely" given the model we've written.

Logistic regression

Maximum likelihood

MLE searches for the β_js that make our data "most likely" using our model.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

1. β_j tells us how x_j affects the log odds
2. odds $= \dfrac{p(X)}{1-p(X)}$. If $p(X) > 0.5$, then odds > 1 and log odds > 0.

So we want choose β_j such that

- log odds are above zero for observations where $y_i = 1$
- log odds even larger for areas of x_j where most is have $y_i = 1$

Logistic regression

Formally: The likelihood function

We estimate logistic regression by maximizing **the likelihood function**[†]

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

The likelihood function is maximized by

- making $p(x_i)$ large for individuals with $y_i = 1$
- making $p(x_i)$ small for individuals with $y_i = 0$

Put simply: Maximum likelihood maximizes a predictive performance, conditional on the model we have written down.

[†] Generally, we actually will maximize the *log* of the likelihood function.

Logistic regression

In R

In R, you can run logistic regression using the `glm()` function.

Aside: Related to `lm`, `glm` stands for *generalized* (linear model).

"Generalized" essentially means that we're applying some transformation to $\beta_0 + \beta_1 X$ like logistic regression applies the logistic function.

Logistic regression

In R

In R, you can run logistic regression using the `glm()` function.

Key arguments (very similar to `lm()`)

- specify a `formula`,[†] *e.g.*, `y ~ .` or `y ~ x + I(x^2)`
- define `family = "binomial"` (so R knows to run logistic regression)
- give the function some `data`

```r
est_logistic = glm(
  i_default ~ balance,
  family = "binomial",
  data = default_df
)
```

[†] Notice that we're back in the world of needing to select a model...

```r
est_logistic %>% summary()
```

```
#> 
#> Call:
#> glm(formula = i_default ~ balance, family = "binomial", data = default_df)
#> 
#> Deviance Residuals: 
#>     Min       1Q   Median       3Q      Max  
#> -2.2697  -0.1465  -0.0589  -0.0221   3.7589  
#> 
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)    
#> (Intercept) -1.065e+01  3.612e-01  -29.49   <2e-16 ***
#> balance      5.499e-03  2.204e-04   24.95   <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: 2920.6  on 9999  degrees of freedom
#> Residual deviance: 1596.5  on 9998  degrees of freedom
#> AIC: 1600.5
#> 
#> Number of Fisher Scoring iterations: 8
```

Logistic regression

Estimates and predictions

Thus, our estimates are $\hat{\beta}_0 \approx -10.65$ and $\hat{\beta}_1 \approx 0.0055$.

Remember: These coefficients are for the **log odds**.

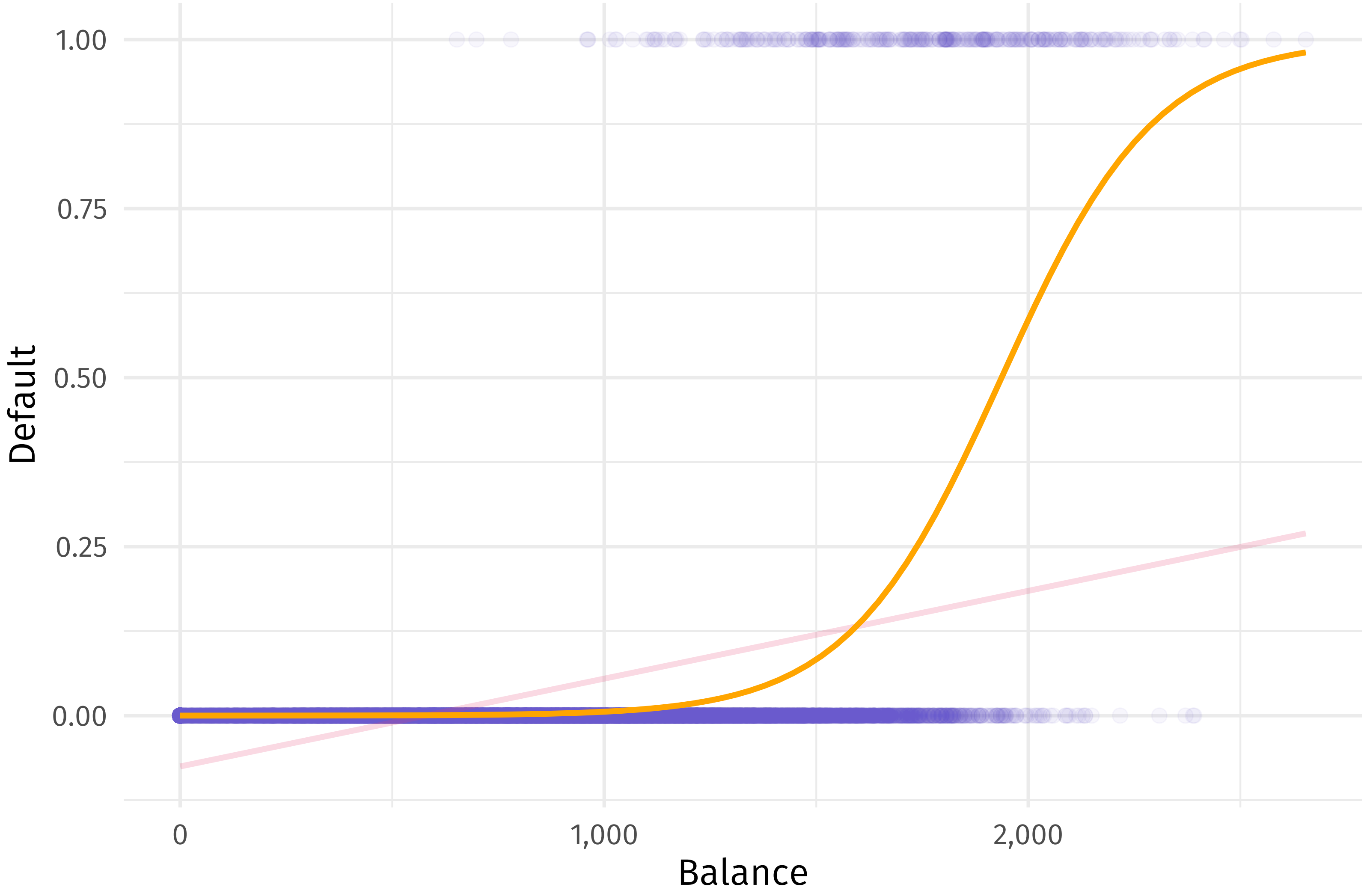
If we want **to make predictions** for y_i (whether or not i defaults), then we first must **estimate the probability** $p(\text{Balance})$

$$\hat{p}(\text{Balance}) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 \text{Balance}}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 \text{Balance}}} \approx \frac{e^{-10.65 + 0.0055 \cdot \text{Balance}}}{1 + e^{-10.65 + 0.0055 \cdot \text{Balance}}}$$

- If $\text{Balance} = 0$, we then estimate $\hat{p} \approx 0.000024$
- If $\text{Balance} = 2,000$, we then estimate $\hat{p} \approx 0.586$
- If $\text{Balance} = 3,000$, we then estimate $\hat{p} \approx 0.997$ [†]

[†] You get a sense of the nonlinearity of the predictors' effects.

Logistic regression's predictions of $p(\text{Balance})$



Note: Everything we've done so far extends to models with many predictors.

Old news: You can use `predict()` to get predictions out of `glm` objects.

New and important: `predict()` produces multiple `type`s of predictions

1. `type = "response"` predicts *on the scale of the response variable*
   for logistic regression, this means **predicted probabilities** (0 to 1)

2. `type = "link"` predicts *on the scale of the linear predictors*
   for logistic regression, this means **predicted log odds** (-∞ to ∞)

Beware: The default is `type = "link"`, which you may not want.

Logistic regression

Prediction

Putting it all together, we can get (estimated) probabilities $\hat{p}(X)$

```r
# Predictions on scale of response (outcome) variable
p_hat = predict(est_logistic, type = "response")
```

which we can use to make predictions on y

```r
# Predict '1' if p_hat is greater or equal to 0.5
y_hat = as.numeric(p_hat >= 0.5)
```

So how did we do?

Assessment

Assessment

How did we do?

We guessed 97.25% of the observations correctly.

Q 97.25% 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 30.03% of the defaults, which is clearer better than 0%.

Q How can we more formally assess our model's performance?

A All roads lead to the **confusion matrix**.

[†] This idea is called the *null classifier*.

Assessment

The confusion matrix

The confusion matrix is us a convenient way to display **correct** and **incorrect** predictions for each class of our outcome.

		Truth	
		No	**Yes**
Prediction	**No**	True Negative (TN)	False Negative (FN)
	Yes	False Positive (FP)	True Positive (TP)

The accuracy of a method is the share of correct predictions, *i.e.*,

Accuracy = (**TN** + **TP**) / (**TN** + **TP** + **FN** + **FP**)

This matrix also helps display many other measures of assessment.

Assessment

The confusion matrix

Sensitivity: the share of positive outcomes $Y = 1$ that we correctly predict.

Sensitivity = **TP** / (**TP** + **FN**)

		Truth	
		No	**Yes**
Prediction	**No**	True Negative (TN)	False Negative (FN)
	Yes	False Positive (FP)	True Positive (TP)

Sensitivity is also called recall and the true-positive rate.

One minus sensitivity is the type-II error rate.

Assessment

The confusion matrix

Specificity: the share of neg. outcomes $(Y = 0)$ that we correctly predict.

Specificity = **TN** / (**TN** + **FP**)

		Truth	
		No	**Yes**
Prediction	**No**	True Negative (TN)	False Negative (FN)
	Yes	False Positive (FP)	True Positive (TP)

One minus specificity is the false-positive rate or type-I error rate.

The confusion matrix

Precision: the share of predicted positives ($\hat{Y} = 1$) that are correct.

Precision = **TP** / (**TP** + **FP**)

		Truth	
		No	**Yes**
Prediction	**No**	True Negative (TN)	False Negative (FN)
	Yes	False Positive (FP)	True Positive (TP)

Assessment

Which assessment?

Q So *which* criterion should we use?

A You should use the *right* criterion for your context.

- Are true positives more valuable than true negatives?
  Sensitivity will be key.

- Do you want to have high confidence in predicted positives?
  Precision is your friend

- Are all errors equal?
  Accuracy is perfect.

There's a lot more, *e.g.*, the F_1 score combines precision and sensitivity.

Assessment

Confusion in R

`confusionMatrix()` from `caret` calculates the confusion matrix—and many other statistics.

- `data`: a `factor` vector of predictions (use `as.factor()` if needed)
- `reference`: a `factor` vector of true outcomes

```r
cm_logistic = confusionMatrix(
  # Our predictions
  data = y_hat %>% as.factor(),
  # Truth
  reference = default_df$i_default %>% as.factor()
)
```

```
#> Confusion Matrix and Statistics
#> 
#>            Reference
#>  Prediction    0    1
#>           0 9625  233
#>           1   42  100
#> 
#>                  Accuracy : 0.9725
#>                    95% CI : (0.9691, 0.9756)
#>       No Information Rate : 0.9667
#>       P-Value [Acc > NIR] : 0.0004973
#> 
#>                     Kappa : 0.4093
#> 
#>    Mcnemar's Test P-Value : < 2.2e-16
#> 
#>               Sensitivity : 0.9957
#>               Specificity : 0.3003
#>            Pos Pred Value : 0.9764
#>            Neg Pred Value : 0.7042
#>                Prevalence : 0.9667
#>            Detection Rate : 0.9625
#>      Detection Prevalence : 0.9858
#>         Balanced Accuracy : 0.6480
```

Assessment

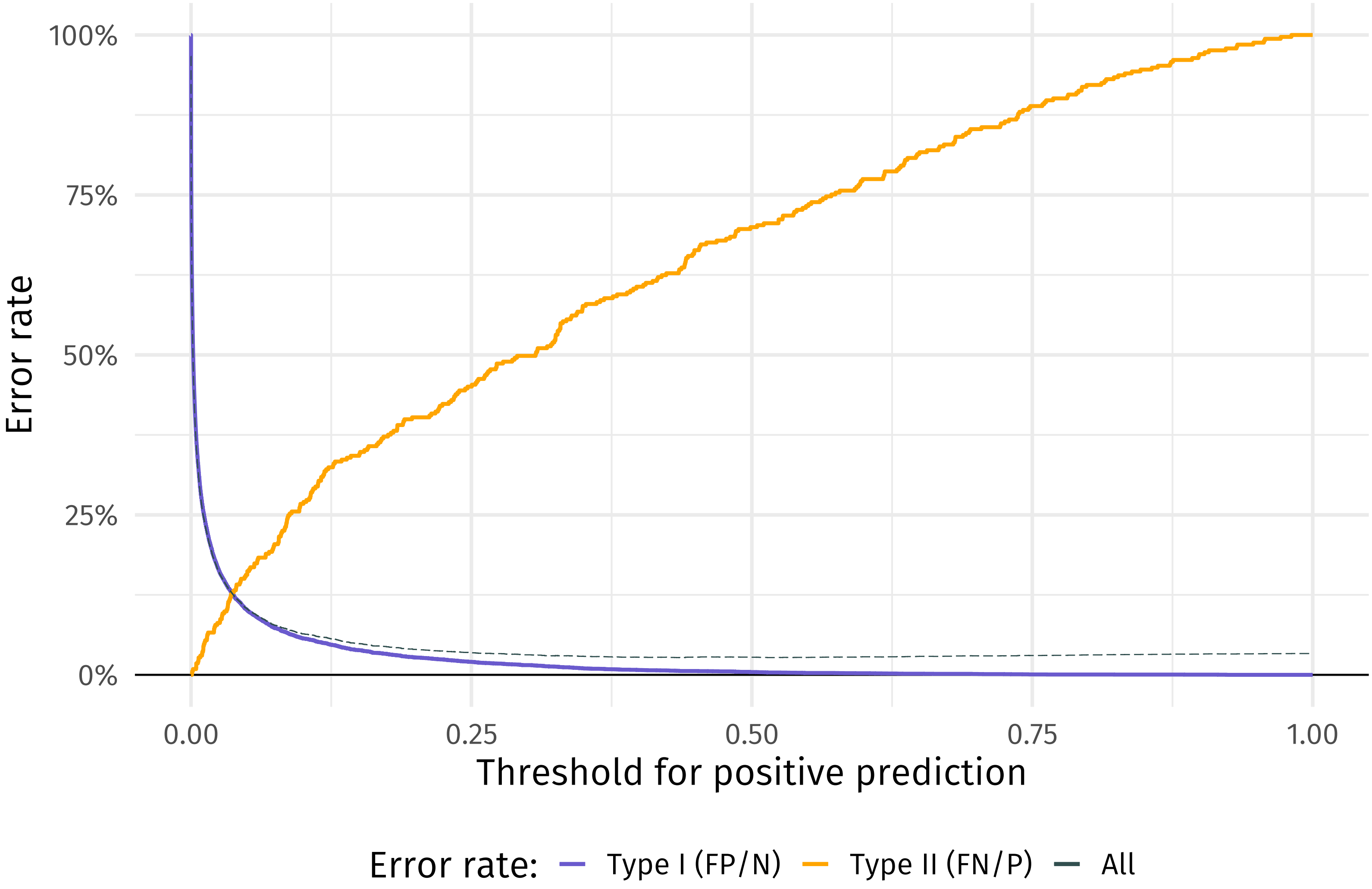
Thresholds

Your setting also dictates the "optimal" threshold that moves a prediction from one class (*e.g.*, Default = No) to another class (Default = Yes).

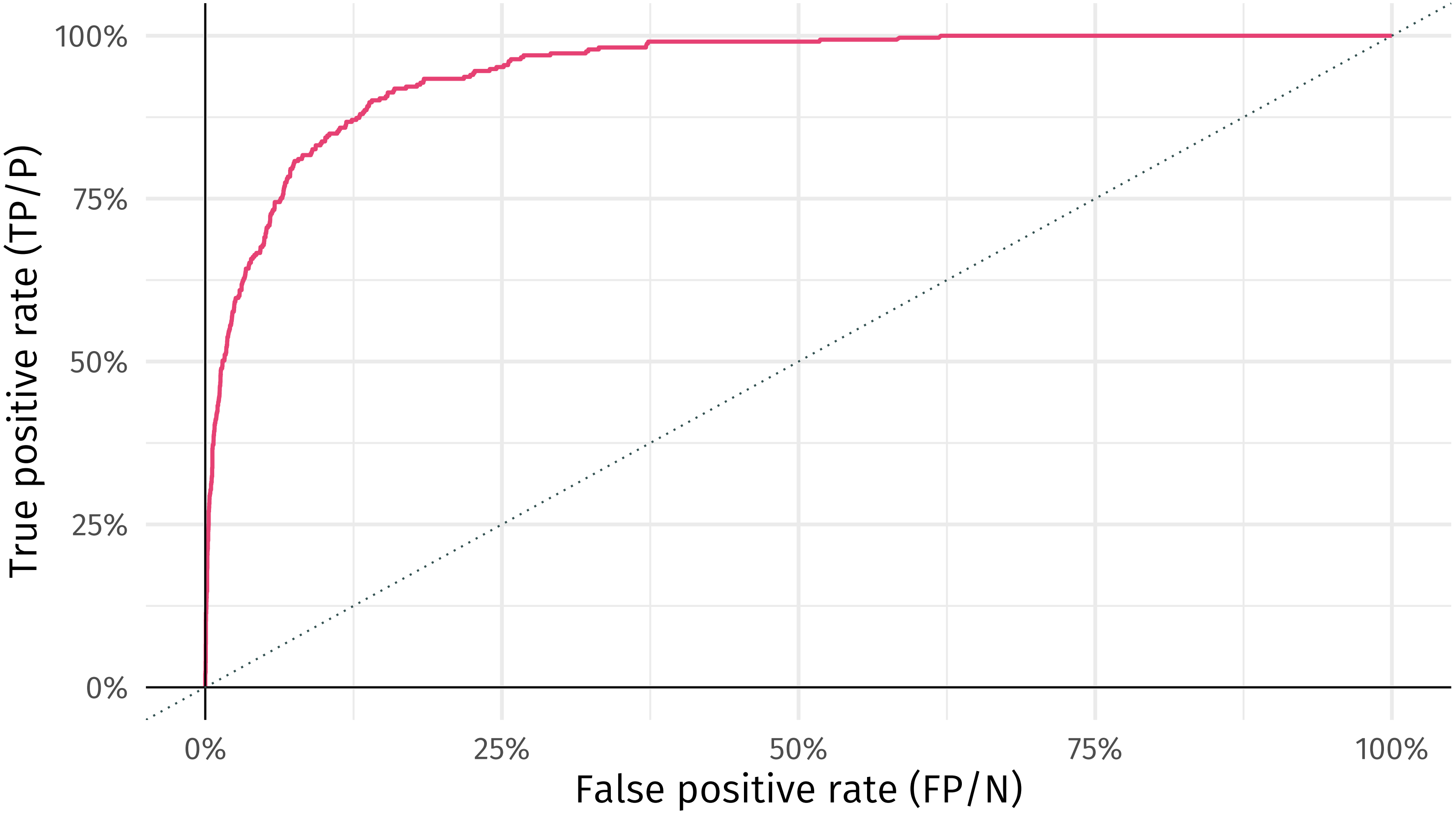
The Bayes classifier suggests a probability threshold of 0.5.

The Bayes classifier can't be beat in terms of *accuracy*, but if you have goals other than accuracy, you should consider other thresholds.

As we vary the threshold, our error rates (types **I**, **II**, and **overall**) change.

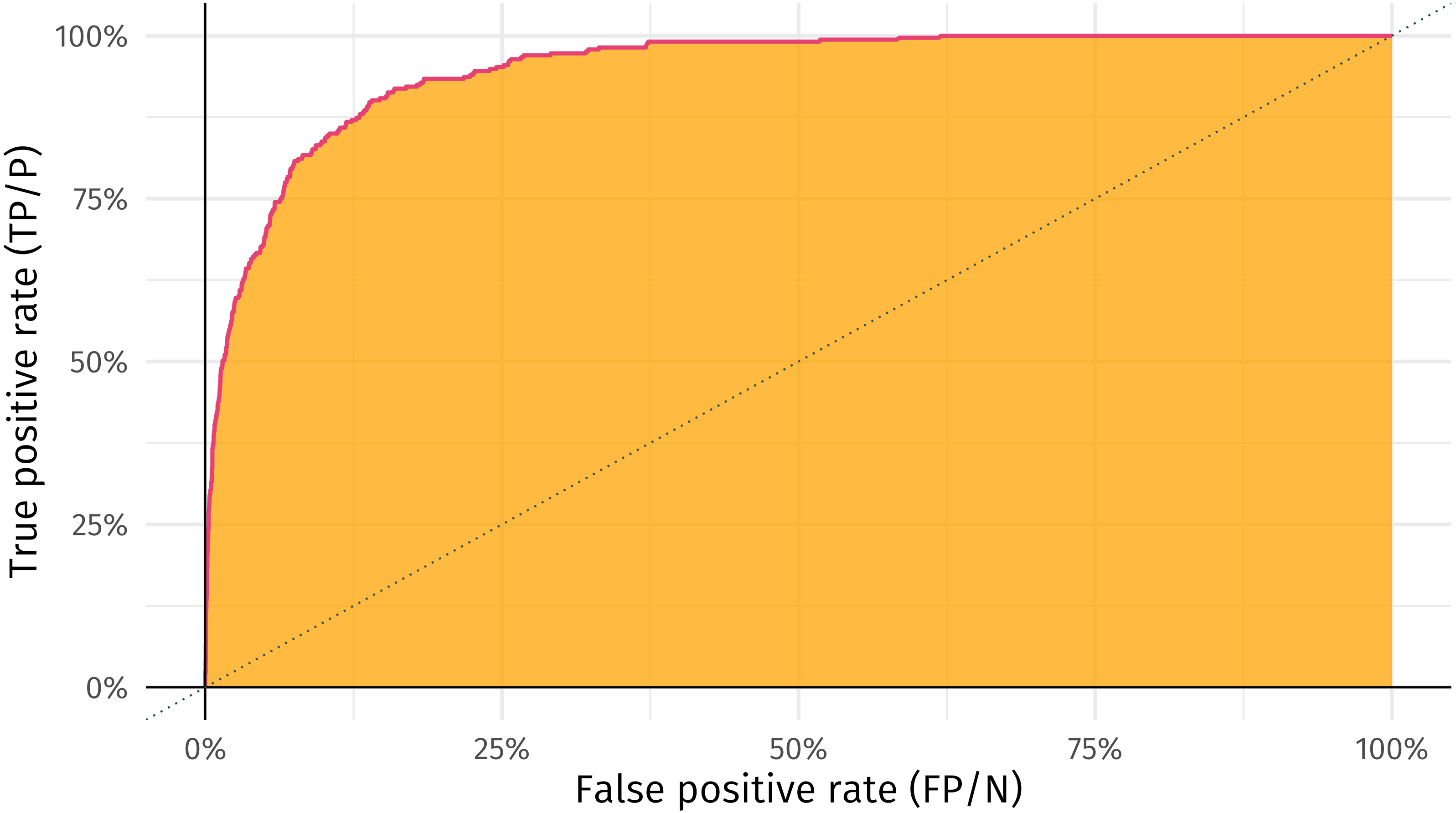


The ROC curve plots the true- (TP/P) and the false-positive rates (FP/N).



"Best performance" means the ROC curve hugs the top-left corner.

The **AUC** gives the area under the (ROC) curve.



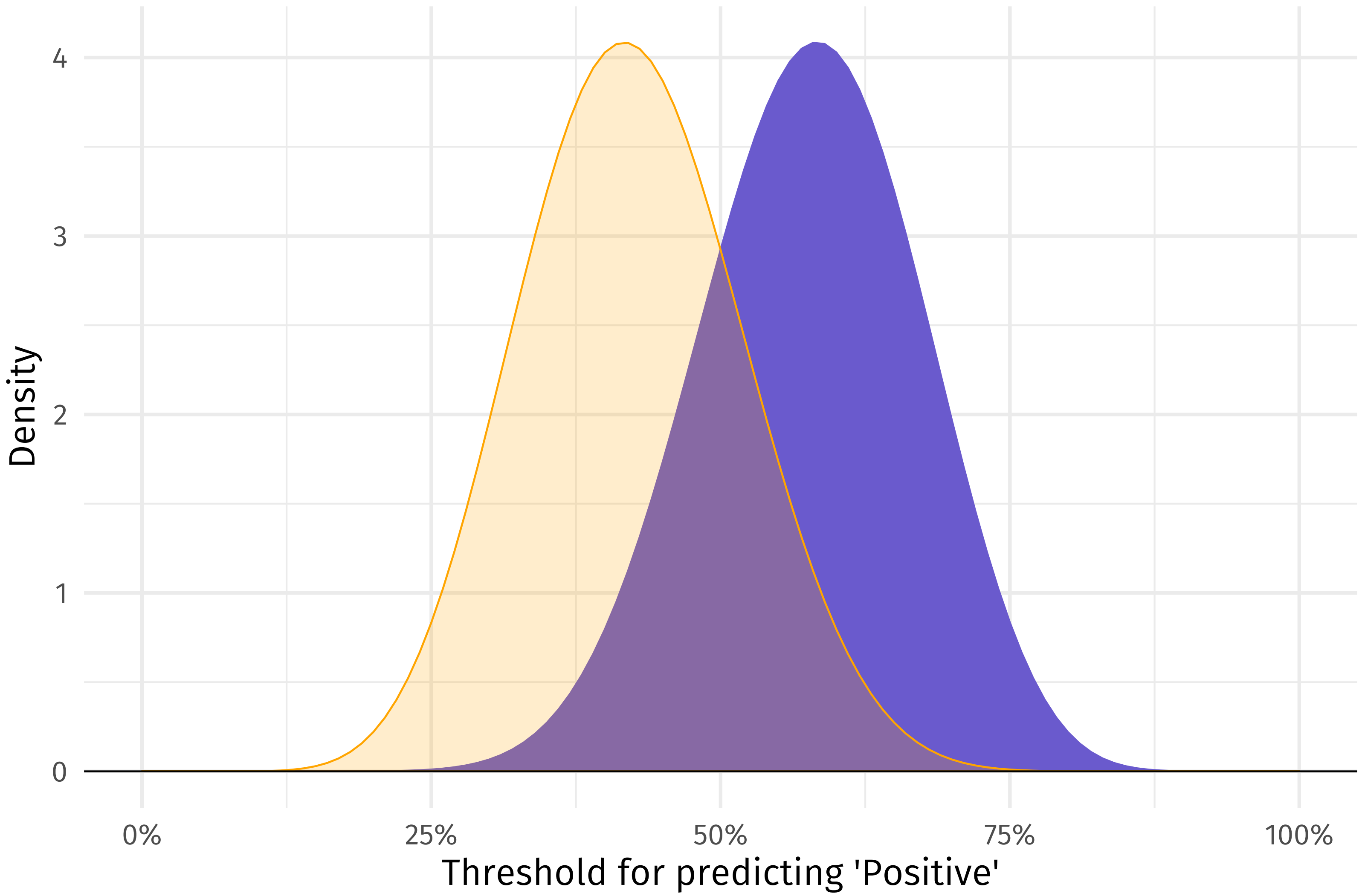
"Best performance" means the AUC is near 1. Random chance: 0.5

Q So what information is AUC telling us?

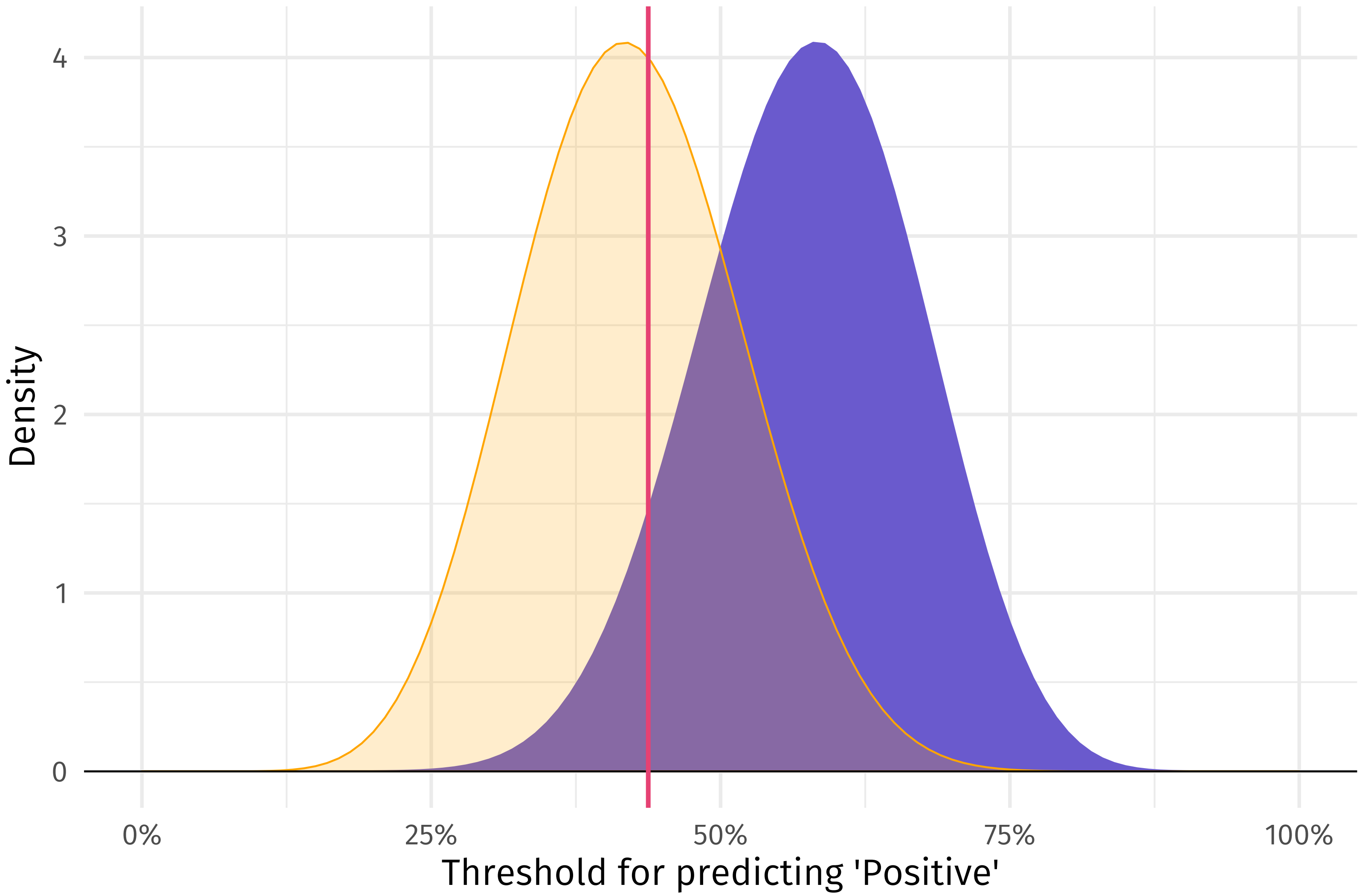
Q So what information is AUC telling us?

A AUC tells us how much we've **separated** the *positive* and *negative* labels.

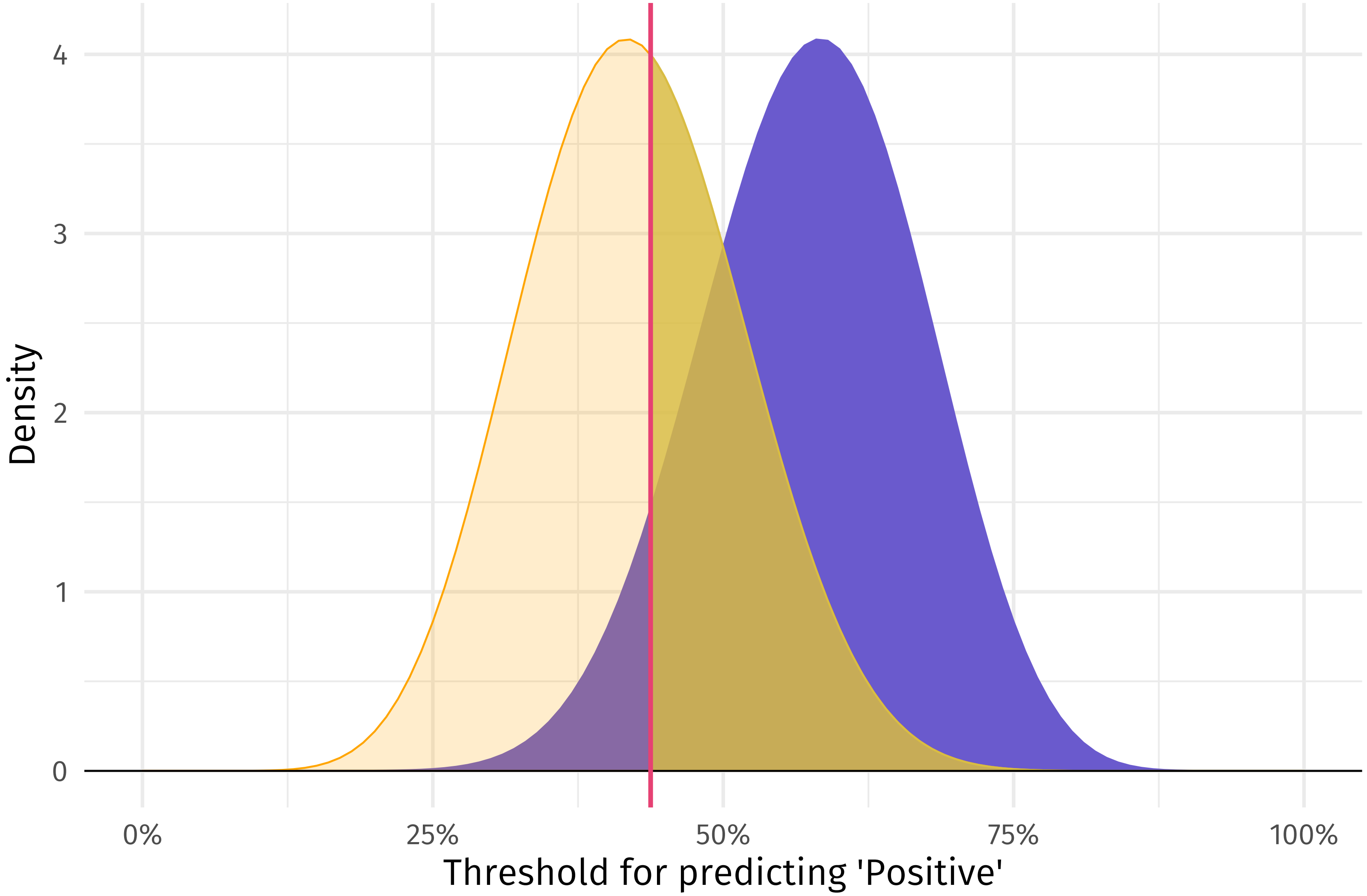
Example: Distributions of probabilities for **negative** and **positive** outcomes.



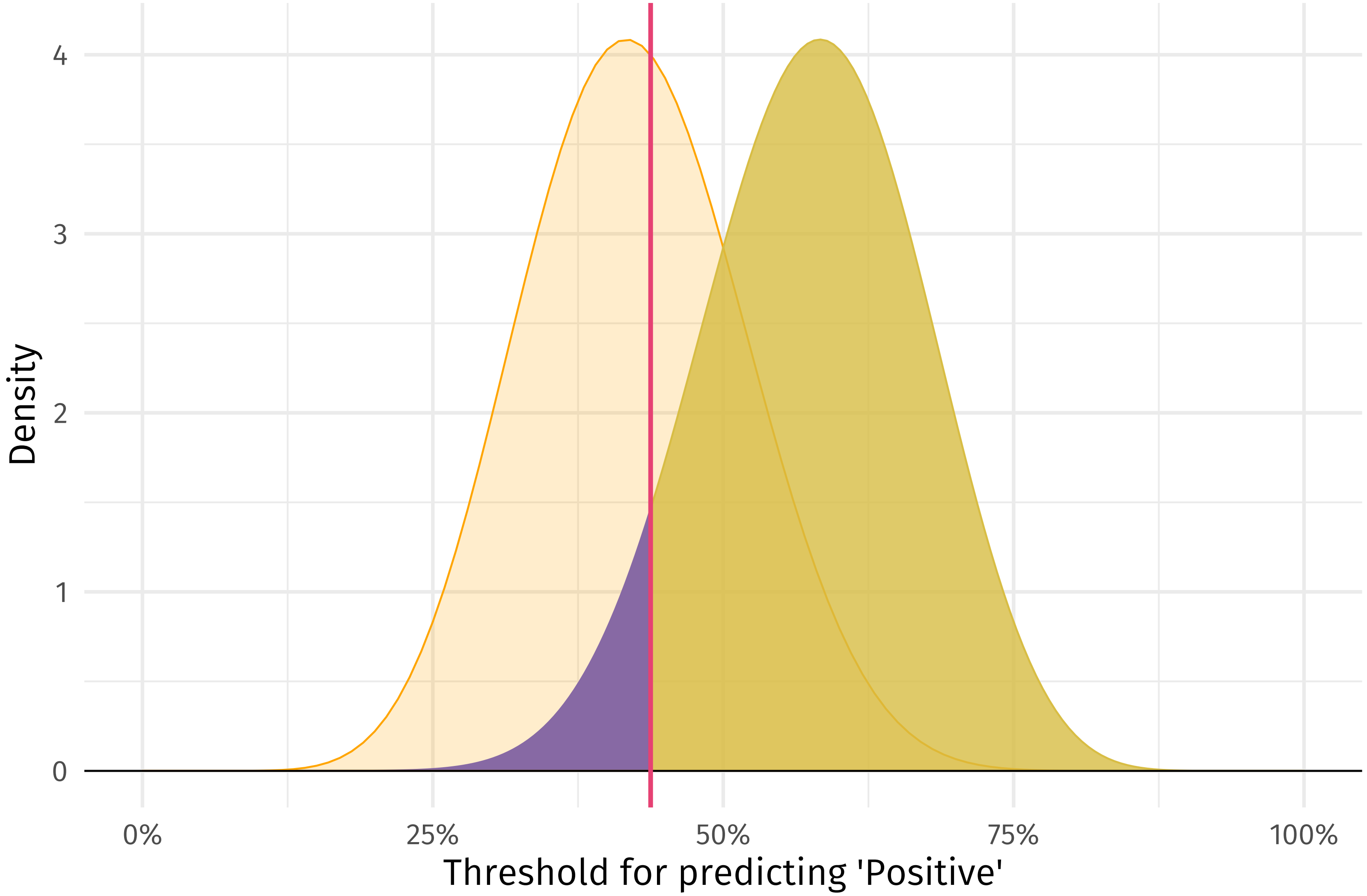
For any given **threshold**



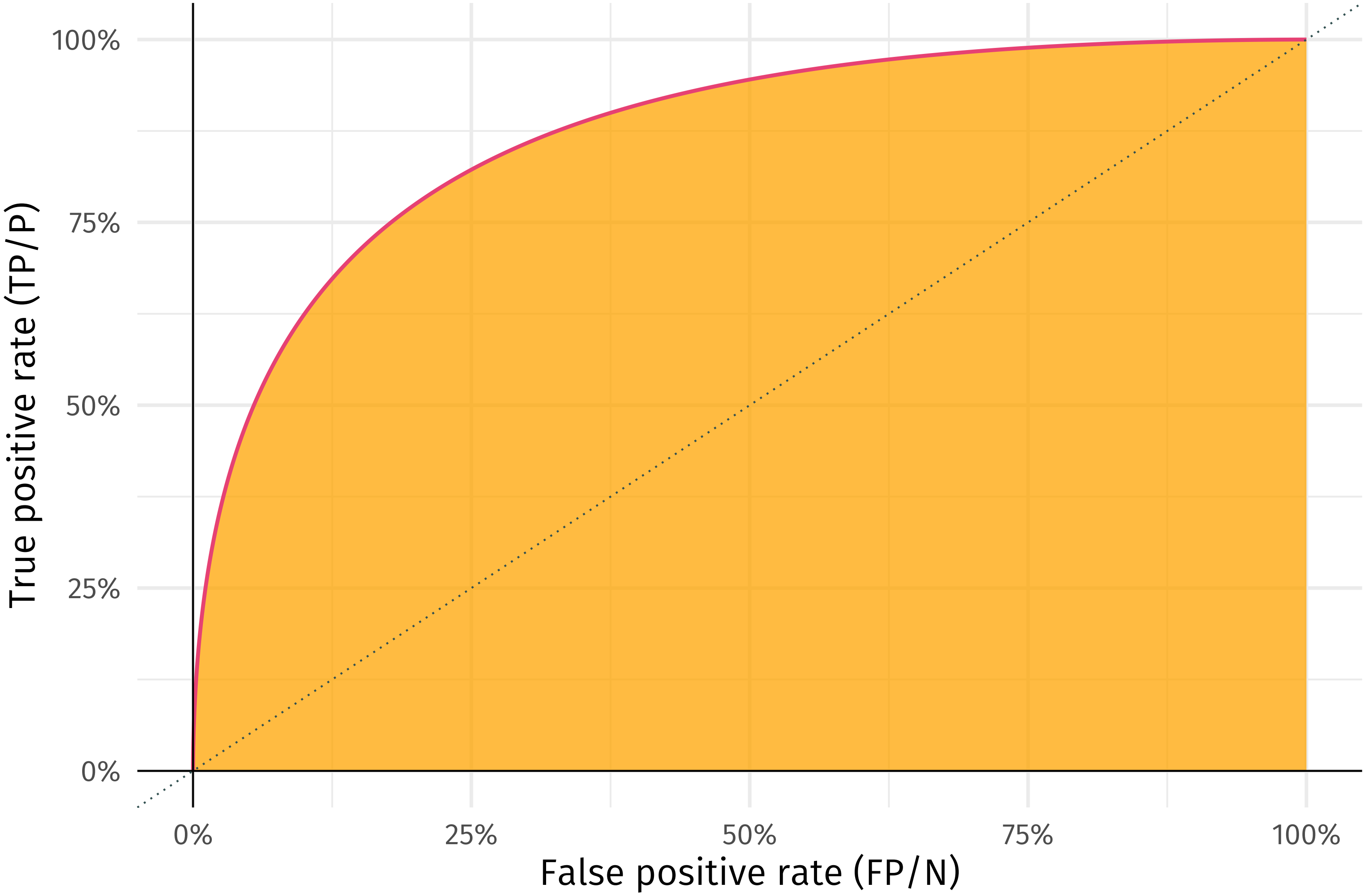
For any given **threshold**, we get **false positives**



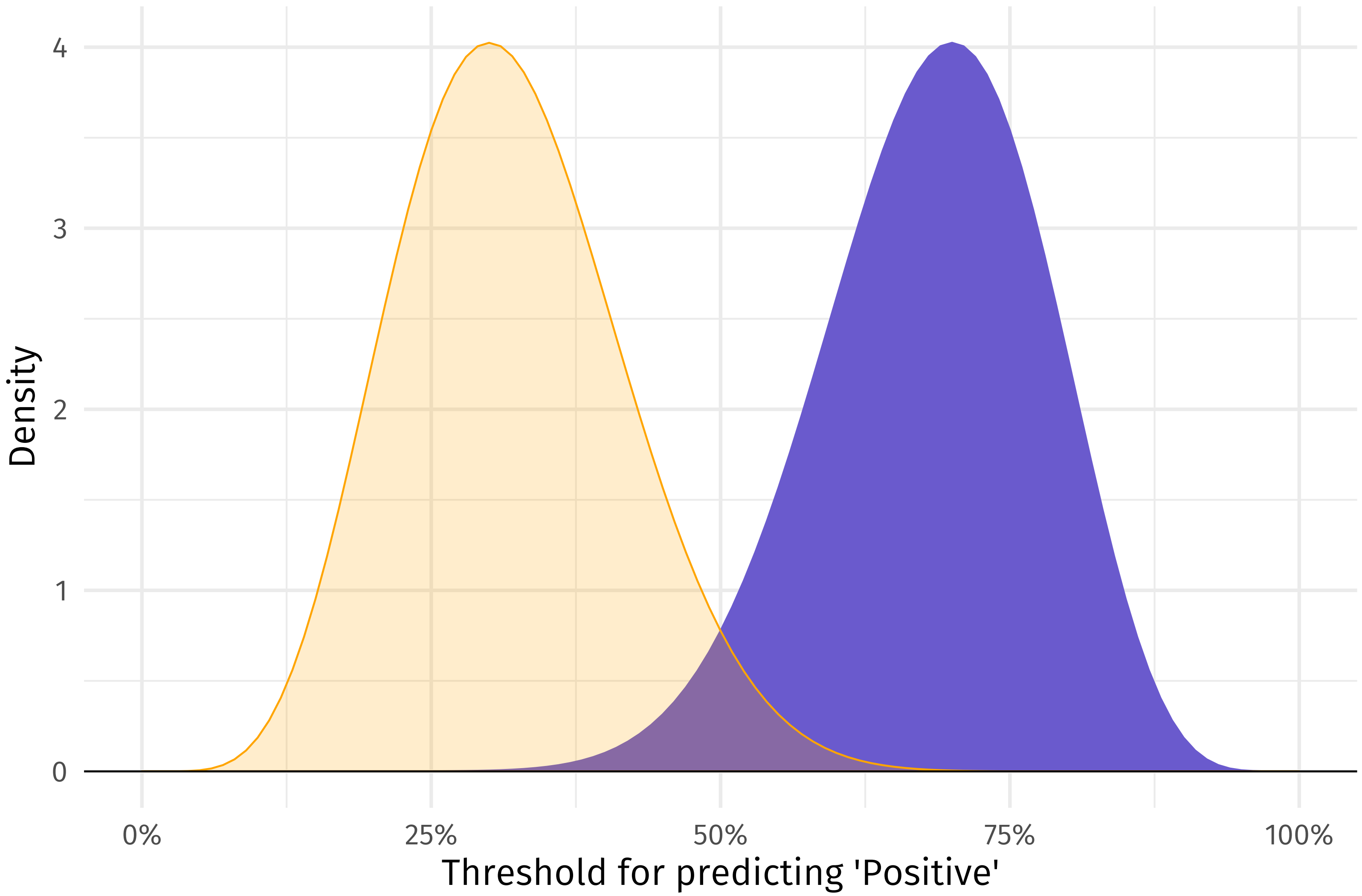
For any given **threshold**, we get false positives and **true positives**.



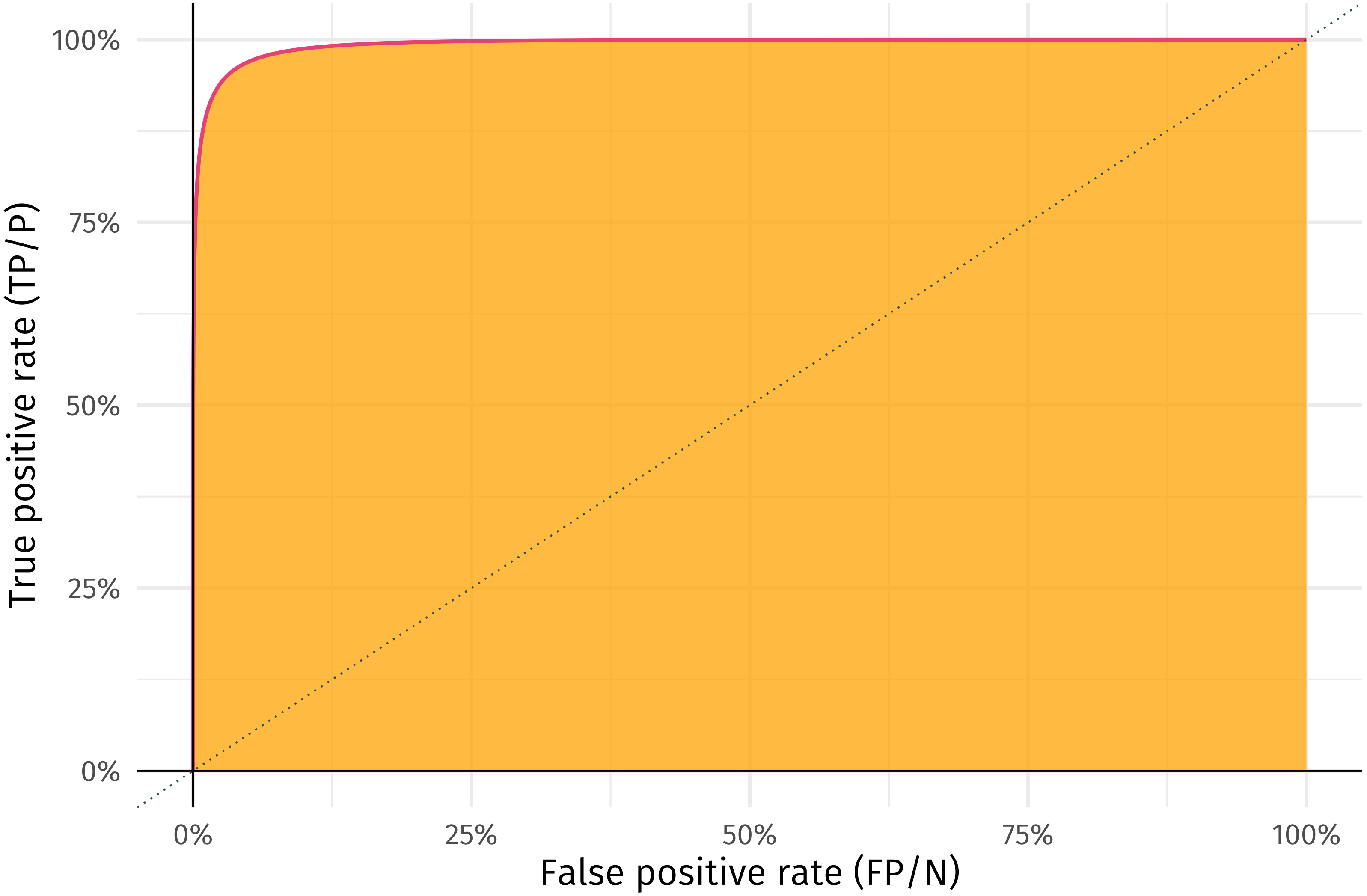
… moving through all possible thresholds generates the **ROC** (**AUC** ≈ 0.872).



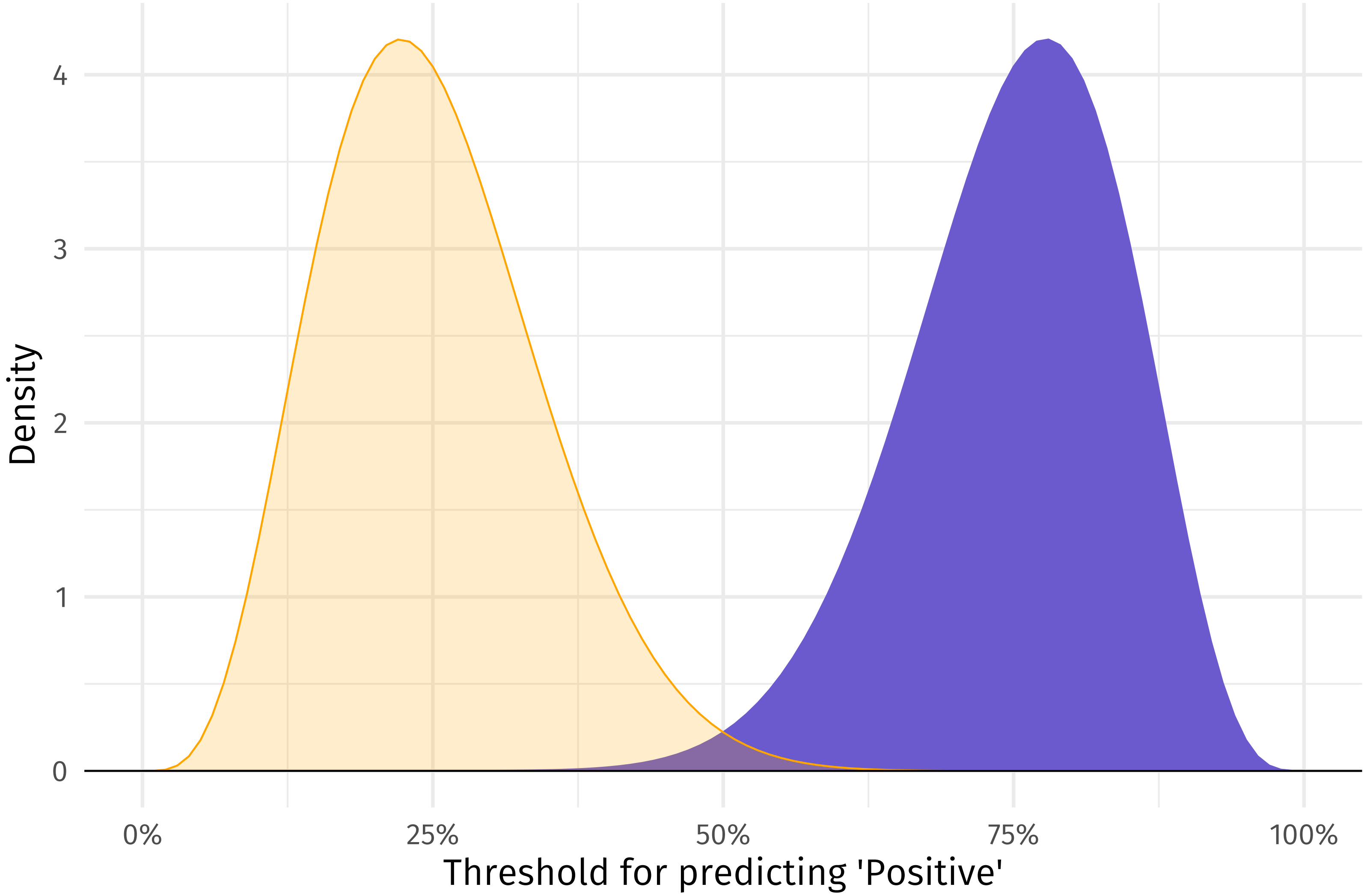
Increasing separation between **negative** and **positive** outcomes…



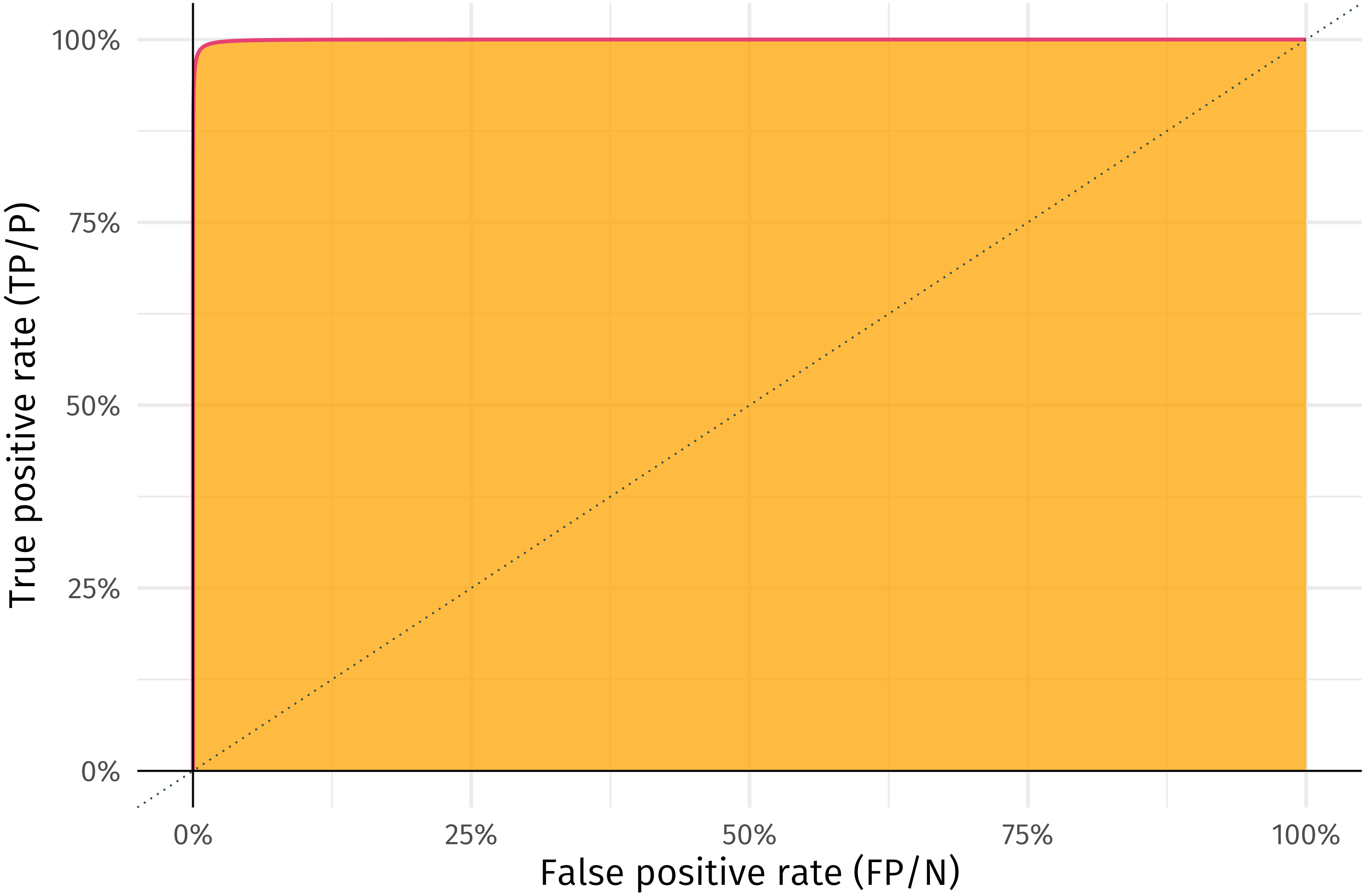
… reduces error (shifts **ROC**) and increases **AUC** (≈ 0.994).



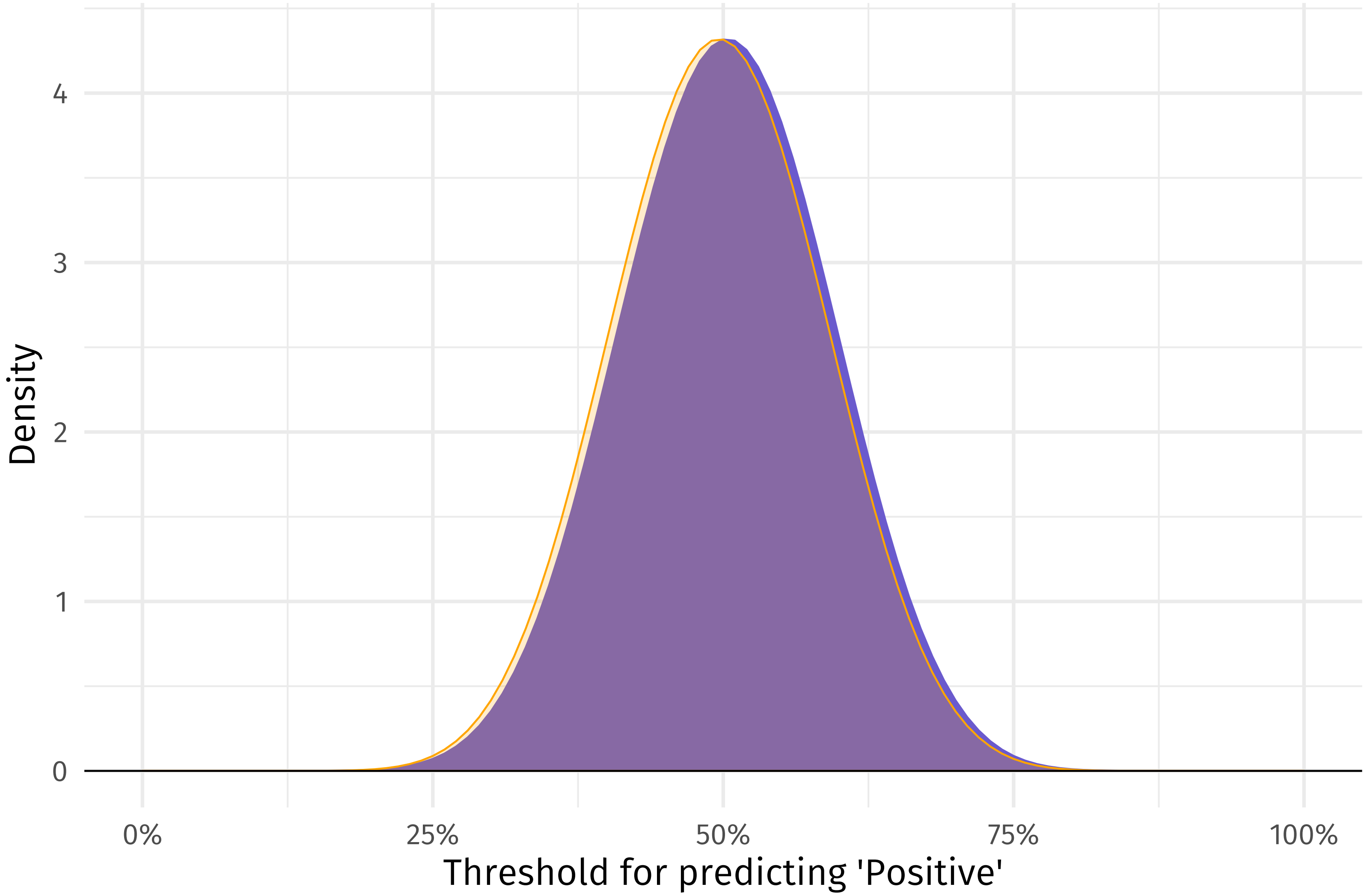
Further increasing separation between **negative** and **positive** outcomes…



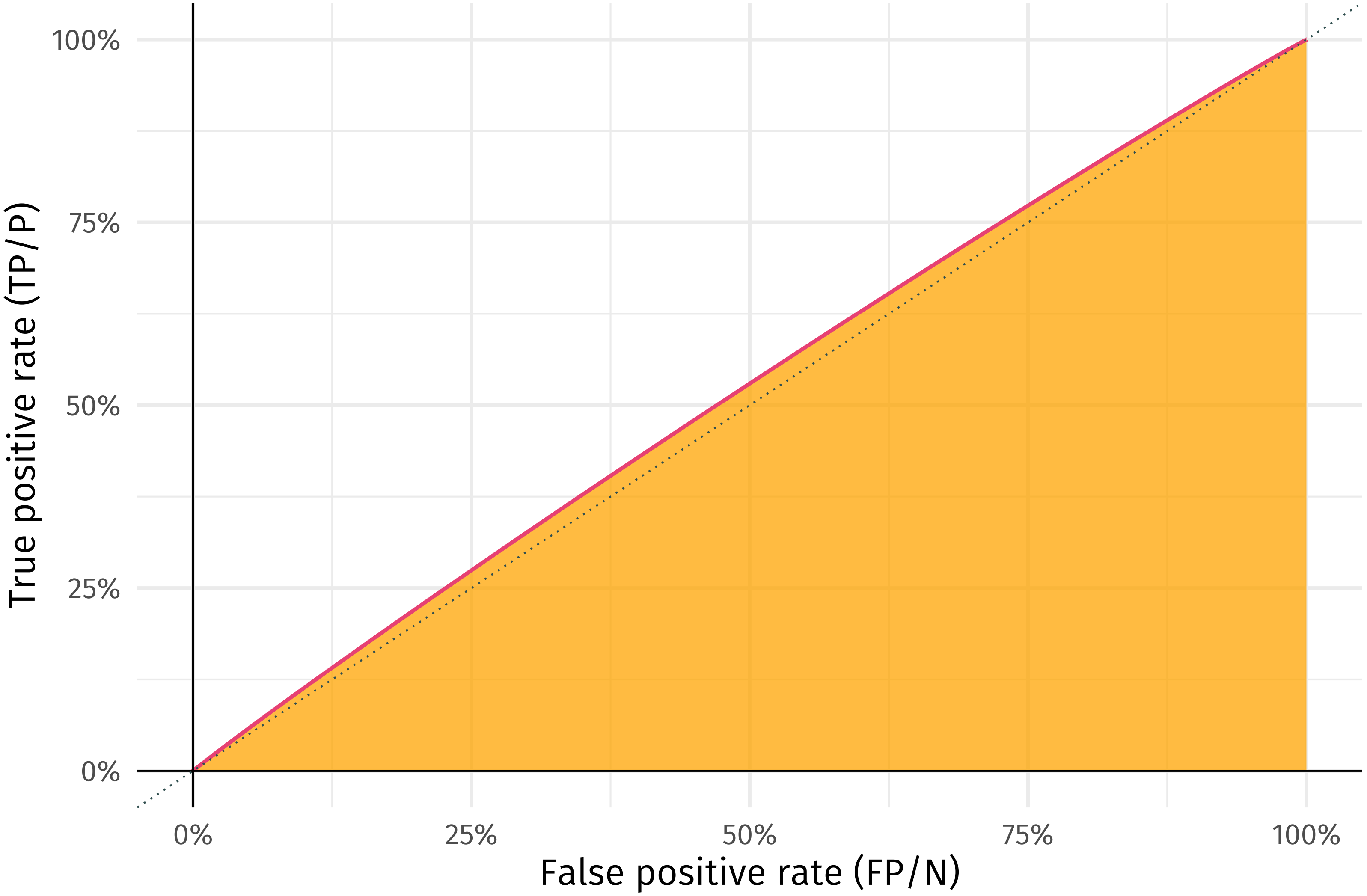
… reduces error (shifts **ROC**) and increases **AUC** (≈ 1).



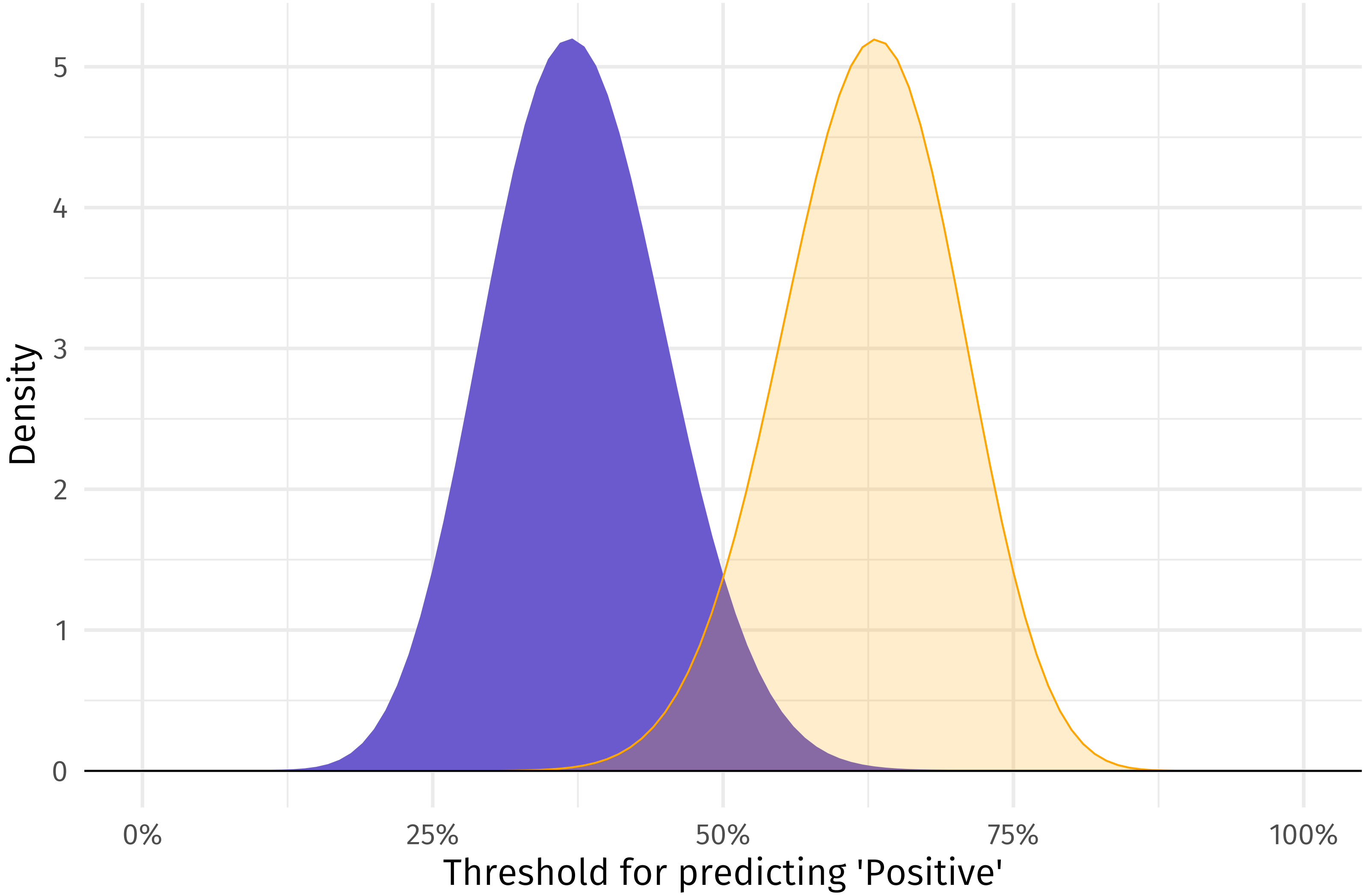
Tiny separation ("guessing") between **negative** and **positive** outcomes...



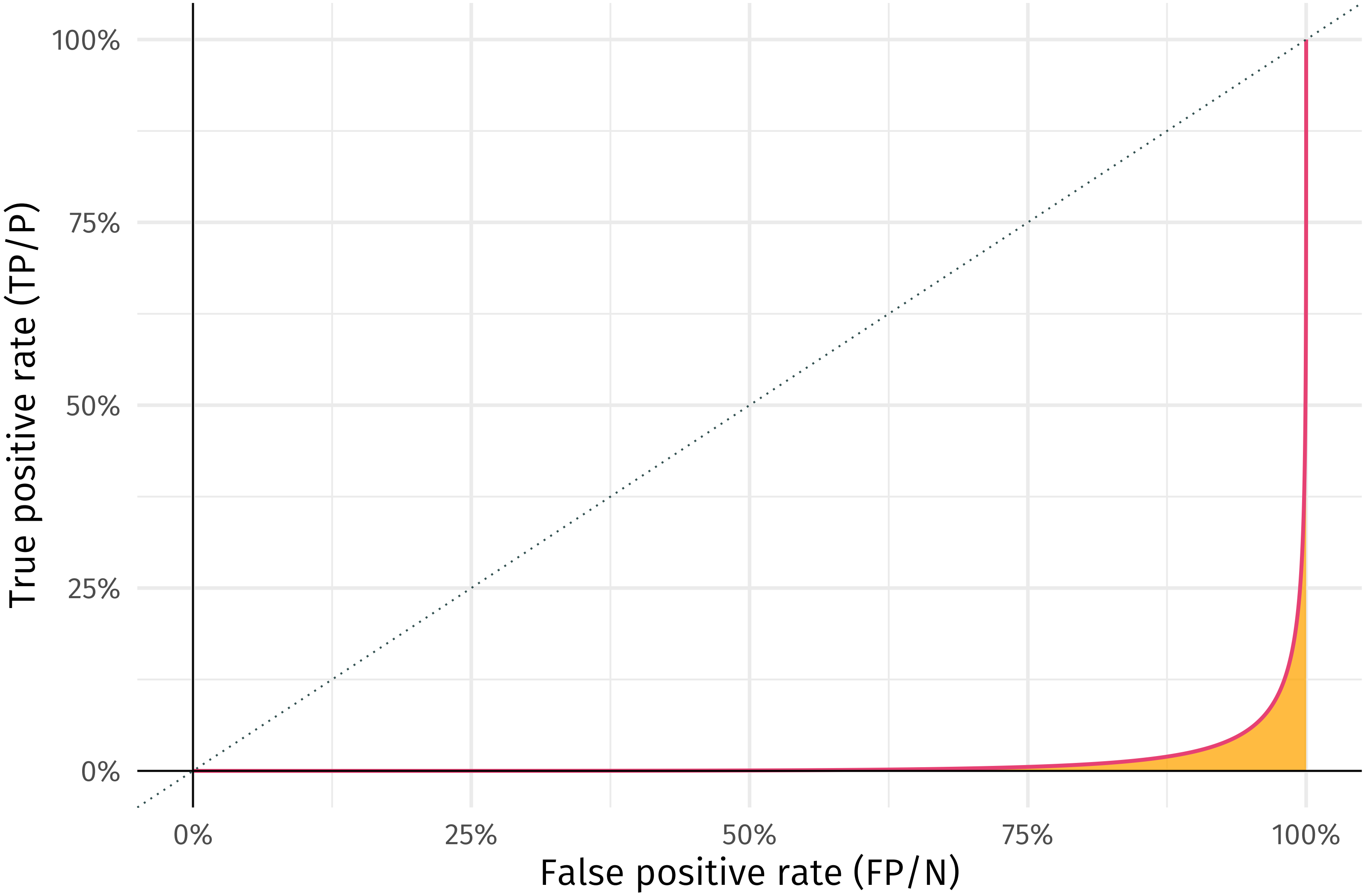
… increases error (shifts **ROC**) and pushes **AUC** toward 0.5 (here ≈ 0.523).



Getting **negative** and **positive** outcomes backwards...



… increases error (shifts **ROC**) and pushes **AUC** toward 0 (here ≈ 0.012).



Sources

These notes draw upon

- An Introduction to Statistical Learning (*ISL*)
  James, Witten, Hastie, and Tibshirani
- *Receiver Operating Characteristic Curves Demystified (in Python)*

Table of contents

Admin

Classification

Logistic regression

Assessment

Other