Introduction to Machine Learning

Week 3: Applied ML - Classification

Spring 2025

Linear Regression (last week)

- Learn a set of model parameters $\boldsymbol{\beta}$ which minimize the error (SSE) between the true labels $\mathbf{y}$ and the predicted labels $\hat{\mathbf{y}}$. The predicted output is a linear combination of the learned coefficients and the observed data:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_D x_D$$

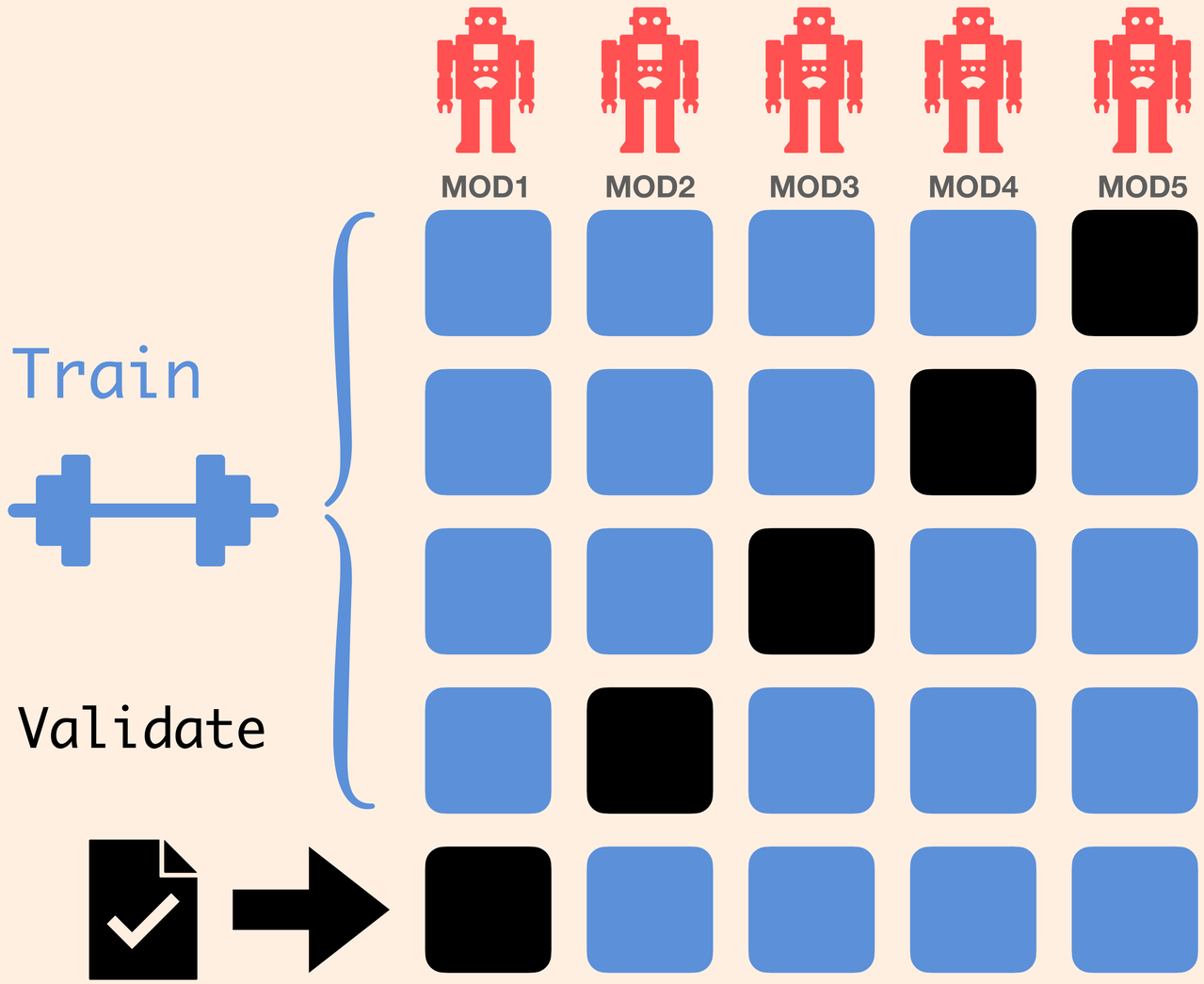
- We can learn non-linear trends by applying transformations to the inputs and fitting one coefficient for each resultant term: e.g. `lm(y ~ x + I(x^2))` would learn a quadratic trend with three parameters (including the intercept)

- More complex models (many parameters) have more flexibility to fit to the training data and, in general, will have better **training-set performance** than simpler models. We must be cautious, however, of such models **overfitting** to its specific training data.

Overfitting and Resampling (last week)

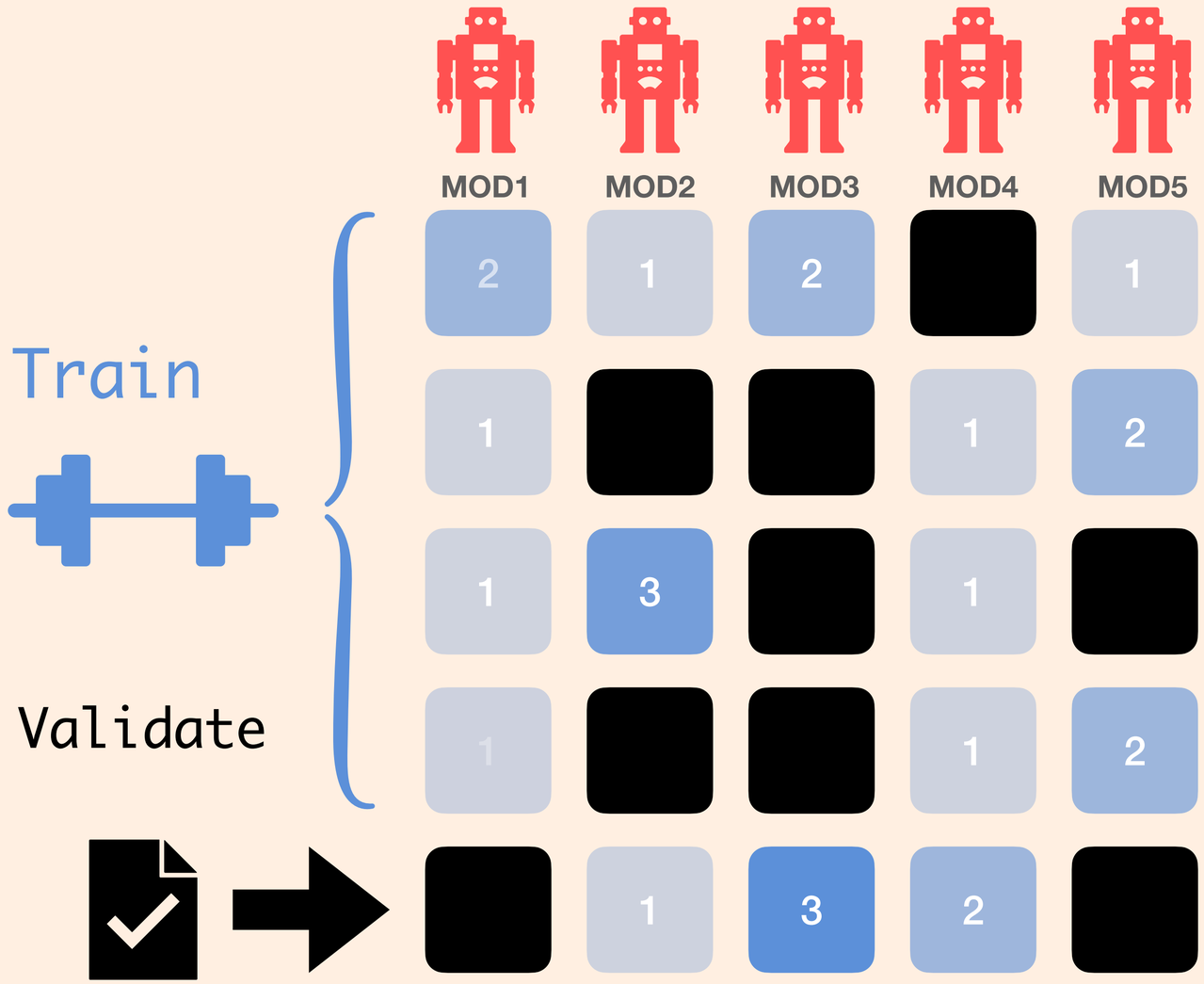
- Complex models risk overfitting to the training data - good performance on samples it has already seen, but performs poorly on new data and may not represent the “true” underlying relationship
- To mitigate overfitting, resampling splits data into training and validation/hold-out sets so that performance can be evaluated on samples left out of training
- K-fold cross-validation partitions data into k subsets. Train/test k different models, each having one unique subset as test data and the remaining k-1 as training. Average performance across folds to
- Bootstrapping is similar to cross-validation, except it does not require every sample to appear as a validation point exactly once.

5-fold Cross-Validation



- Randomly partition the data into k folds, such that each sample appears in a test set exactly once
- Start with model 1: train using Fold 1's training split, and calculate performance on the testing split
- Repeat for models 2-k
- Summarize performance on the held out data for all folds

Bootstrapping



- In each bootstrap, we sample N training points *with-replacement* from the set of all observations
- Samples can be repeated multiple times in a given bootstrap
- The validation set consists of any samples that were not selected for training (can vary)

Classification models

The regression learning problem was…

$$y = f(\mathbf{x}) + \epsilon$$

- $f(\mathbf{x})$ represents the mean trend, or the model's prediction for all inputs.
- The true response, y, always has some degree of uncertainty or noise.
- The learning problem is to find a set of parameters, $\boldsymbol{\beta}$, such that the error between the predicted response and the true label is minimized.
- For regression problems, we use the sum of squared-errors (SSE) to evaluate performance. For a given sample, the error is essentially the difference $y - f(\mathbf{x})$.
- **For classification problems where the response is not a continuous variable, does it seem appropriate to use SSE to assess the error?**

In regression the response, y, is continuous

- The response is a decimal point number
- The model predicts $f(\mathbf{x})$, a continuous value
- Therefore, the error is also continuous!

Instead, consider a situation where the response is categorical or discrete

- Let's specifically consider the case of **binary** response which can only take one of two possible states.
- Examples include:
  - If a basketball player makes a shot, or misses it
  - Will someone default on a loan, or not?
  - Are radio signals from space natural phenomena, or messages from an alien civilization?

All those binary outcome examples have non-numeric responses

- To be general, we usually say that there is an **EVENT** that we are interested in classifying.
- So the binary outcome is: **EVENT** vs **NON-EVENT**

We want to classify a non-numeric value!

- Can we use our regression formulation for classifying **EVENT** vs **NON-EVENT**?

$$y = f(\mathbf{x}) + \text{error}$$

$$?$$

We want to classify a non-numeric value!

- Can we use our regression formulation for classifying **EVENT** vs **NON-EVENT**?

$$y = f(\mathbf{x}) + \text{error}$$

- NO! We cannot calculate $\text{EVENT} - f(\mathbf{x}) = \text{error}$

However, we can change the **encoding**…

- Instead of using the class labels directly, we can ENCODE the binary outcome as:

- If the EVENT occurs: $y = 1$
- If the EVENT does NOT occur: $y = 0$

We can now calculate the error between the encoded binary response and a model prediction

- The error would be:
  - If the EVENT occurs: $1 - f(\mathbf{x}) = \text{error}$
  - If the NON-EVENT occurs: $0 - f(\mathbf{x}) = -f(\mathbf{x}) = \text{error}$

We can now calculate the error between the encoded binary response and a model prediction

- The error would be:
  - If the EVENT occurs: $1 - f(\mathbf{x}) = \text{error}$
  - If the NON-EVENT occurs: $0 - f(\mathbf{x}) = -f(\mathbf{x}) = \text{error}$

- We have not discussed what the error "looks" like.
  - We have not discussed how the **LOSS FUNCTION** is formulated.
  - We will step through the details of this loss function via its associated **probability distribution** later in the semester

However, even without going through the math this situation just seems different from regression

- The error would be:
  - If the EVENT occurs: $1 - f(\mathbf{x}) = \text{error}$
  - If the NON-EVENT occurs: $0 - f(\mathbf{x}) = -f(\mathbf{x}) = \text{error}$

- Even though the response is encoded to be numeric, 1 or 0, there are still only 2 unique values!

- The binary classification loss function should be **DIFFERENT** from the regression loss function

Why can't we use linear regression for categorical responses?

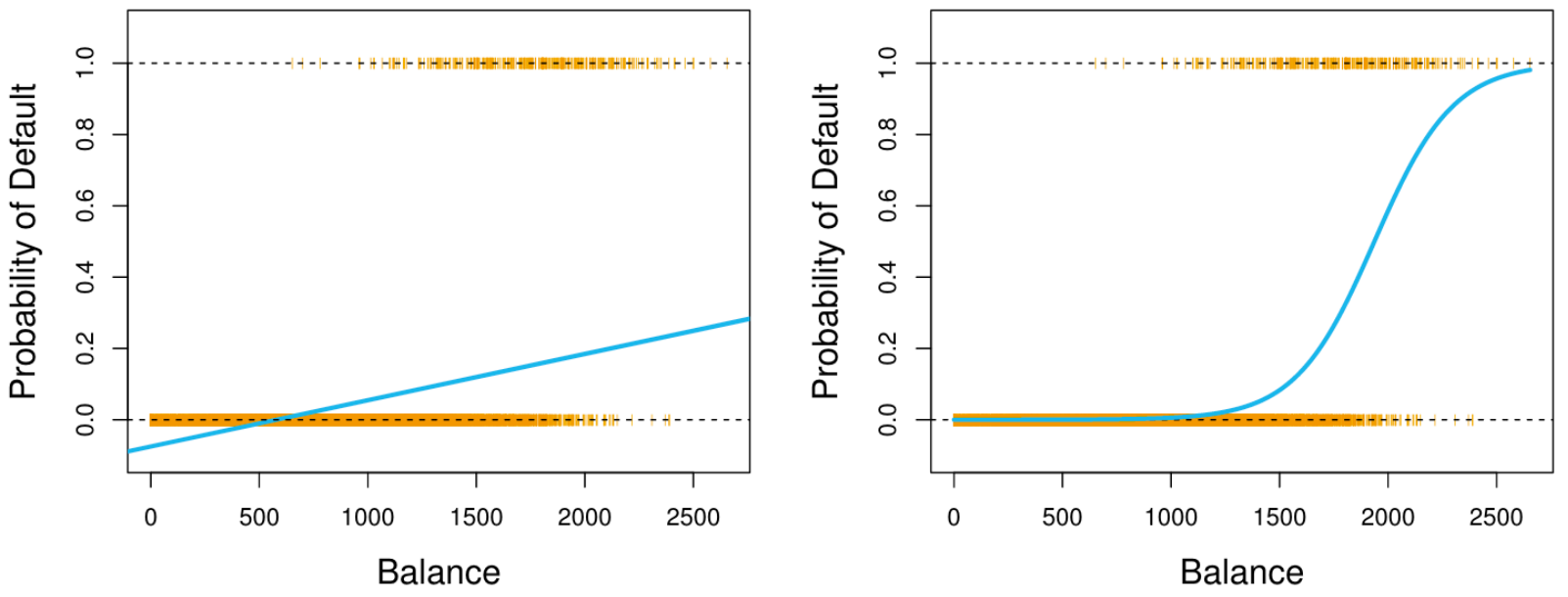


FIGURE 4.2. *Classification using the* `Default` *data.* Left: *Estimated probability of* `default` *using linear regression. Some estimated probabilities are negative! The orange ticks indicate the 0/1 values coded for* `default` *(*`No` *or* `Yes`*).* Right: *Predicted probabilities of* `default` *using logistic regression. All probabilities lie between* 0 *and* 1.

- Even in the special case of binary classification with $y \in \{0,1\}$, linear regression gives nonsensical results
- i.e. probabilities outside the range [0,1]
- Classification algorithms, such as logistic regression, define $f(\mathbf{x}, \boldsymbol{\beta})$ such that its output is a valid probability

The change in the loss function results in a change in what is modeled

- In most binary classification models, we are **NOT** actually predicting the CLASS – the binary outcome – directly.
  - Most models will **NOT** predict HIT or OUT, True or False, Stay or Leave, Pass or Fail,…

- **The change in the loss function results in most models predicting the EVENT PROBABILITY!!!!**

Most binary classification models have the following formulation

$$\text{EVENT PROBABILITY} = f(\mathbf{x}, \boldsymbol{\beta})$$

- The parameters, $\boldsymbol{\beta}$, are written explicitly within the relationship above.
- The LOSS FUNCTION considers the **LIKELIHOOD** of the observed response, $y = 1$ or $y = 0$, **GIVEN** the modeled EVENT PROBABILITY.
- The parameters are learned by MINIMIZING the LOSS.

Let's use another toy problem to gain intuition about modeling the probability

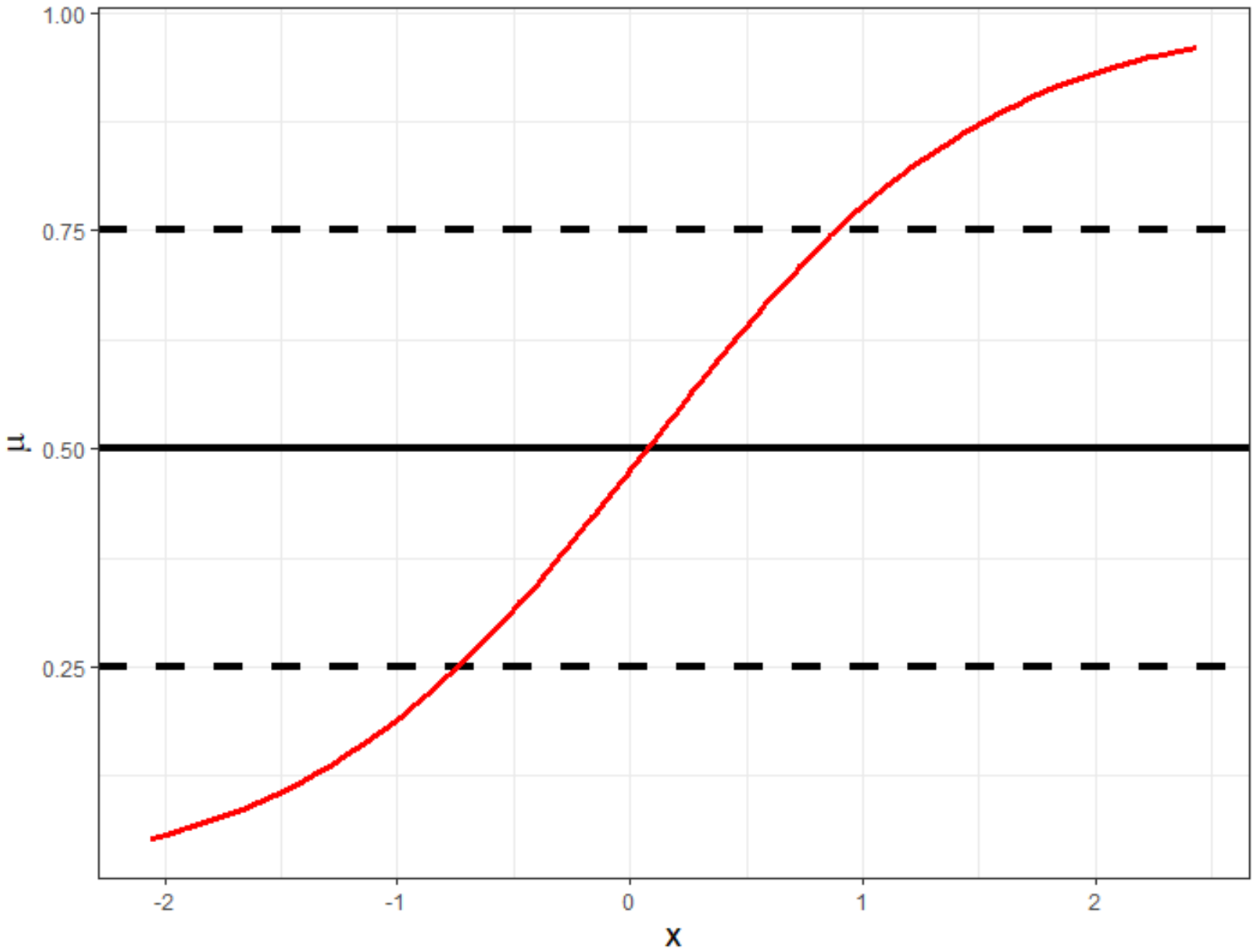
- We will use a single input, x, like we did in the regression toy problem.

- I specified the TRUE parameters, $\boldsymbol{\beta}_*$, that control the TRUE EVENT PROBABILITY.

$$\text{TRUE EVENT PROBABILITY} = f(\mathrm{x}, \boldsymbol{\beta}_*)$$

- I will denote the EVENT PROBABILITY as μ.

The TRUE EVENT PROBABILITY with respect to the input, x



Observations, y, were randomly generated based on the TRUE EVENT PROBABILITY

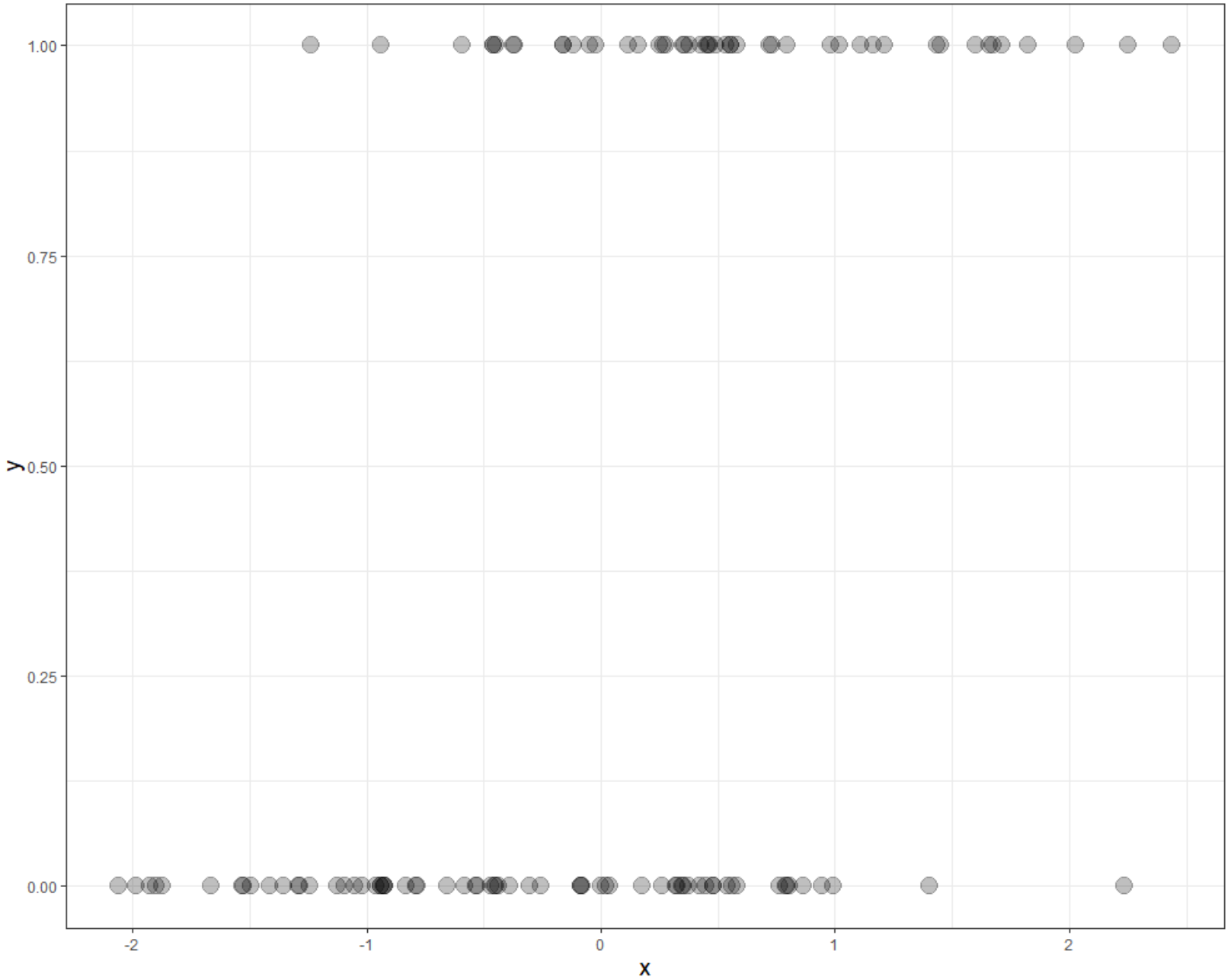
- The input values were randomly generated.
- The true event probability was calculated at each input value.
- A random binary outcome was generated based on the calculated true event probability.
- For this example, we will use 115 input-output pairs:

$$\{x_n, y_n\}_{n=1}^{N=115}$$

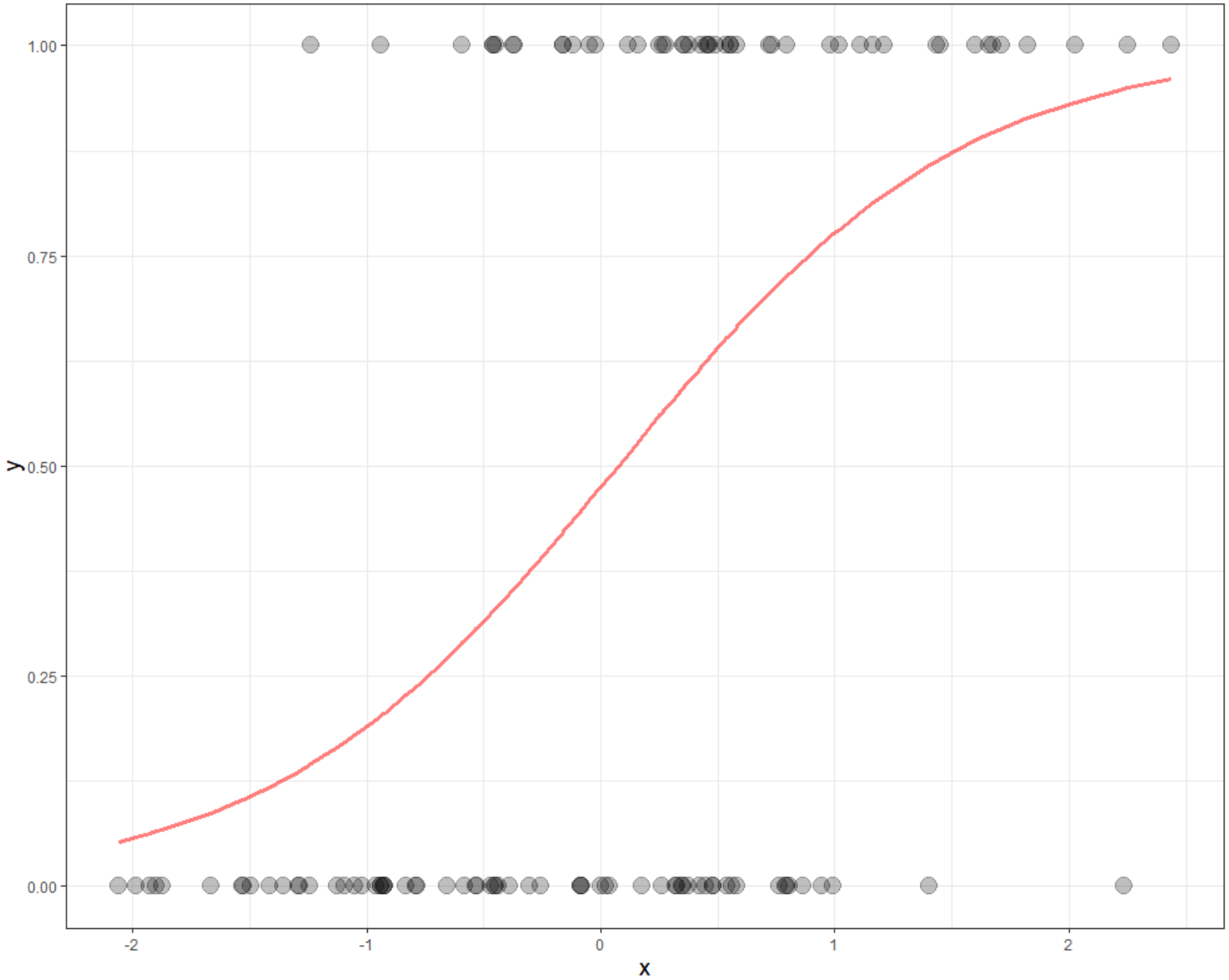
Store the generated data in the object `my_data`

```
> my_data
# A tibble: 115 x 2
        x     y
    <dbl> <int>
 1 -0.457     1
 2  0.426     1
 3 -0.785     0
 4 -1.93      0
 5  2.25      1
 6  1.60      1
 7 -1.87      0
 8 -0.161     1
 9  0.556     1
10  0.348     0
# ... with 105 more rows
```

Plot the binary outcomes with respect to x

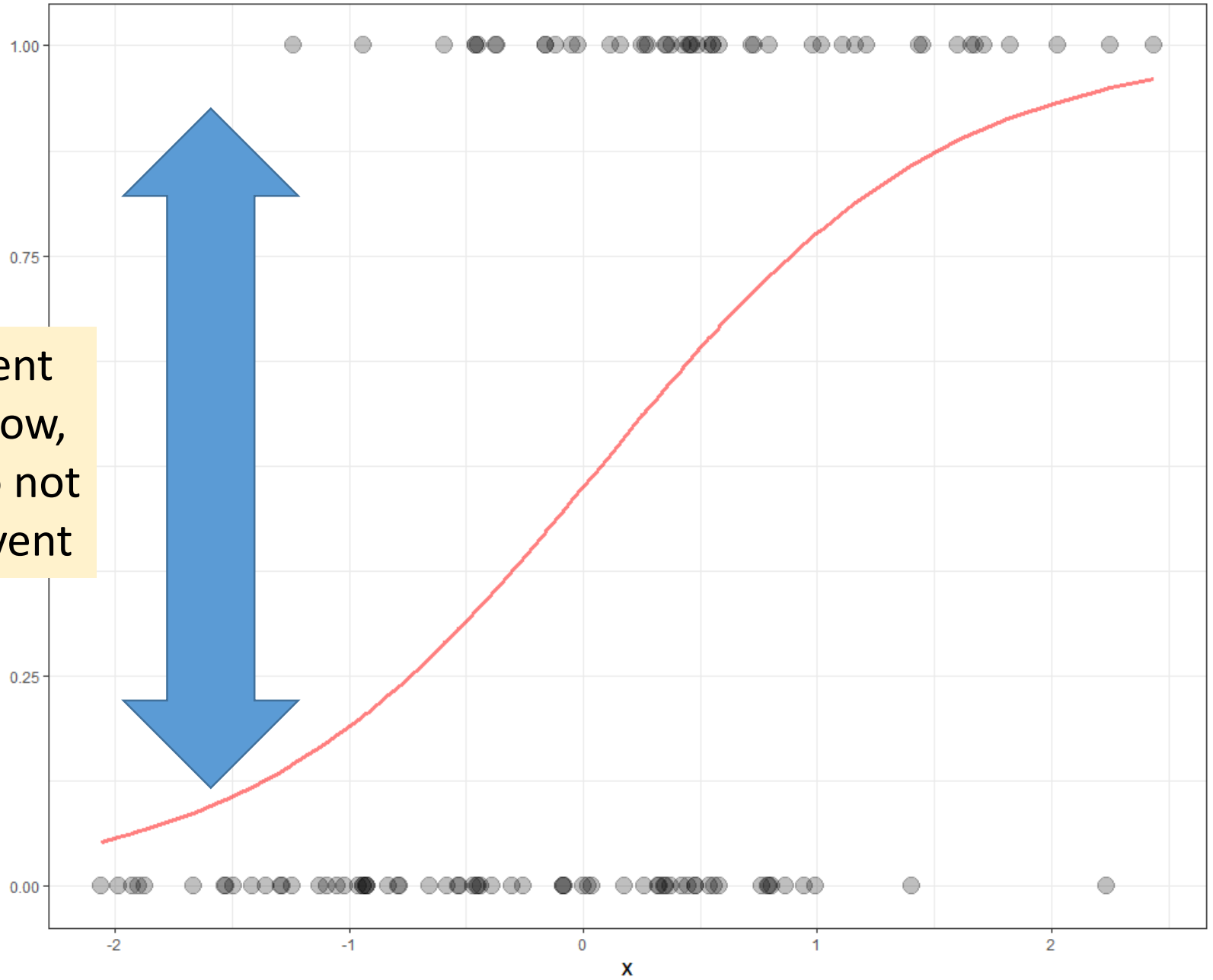


Include the TRUE EVENT PROBABILITY for reference

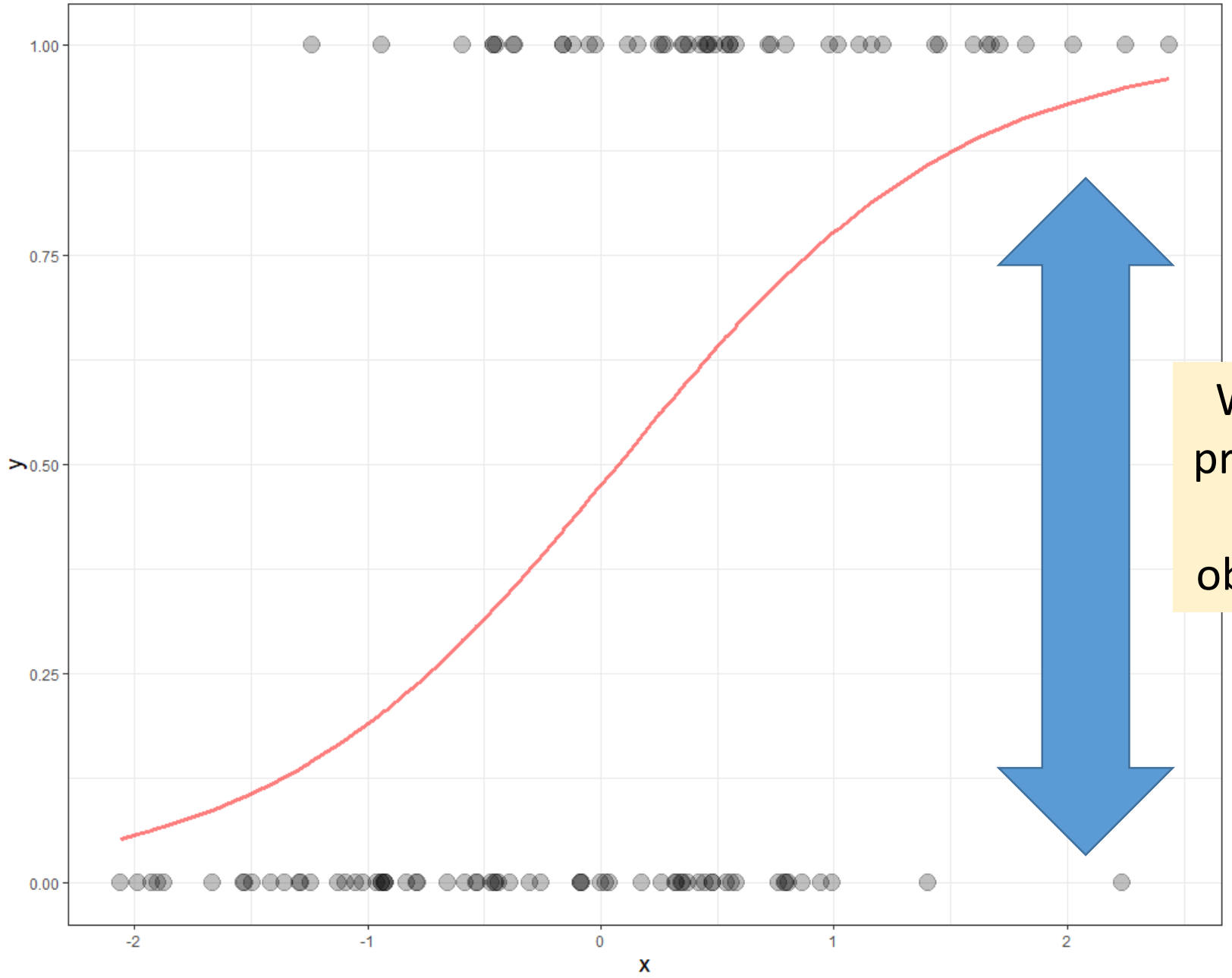


Include the TRUE EVENT PROBABILITY for reference

When the event probability is low, we typically do not observe the event

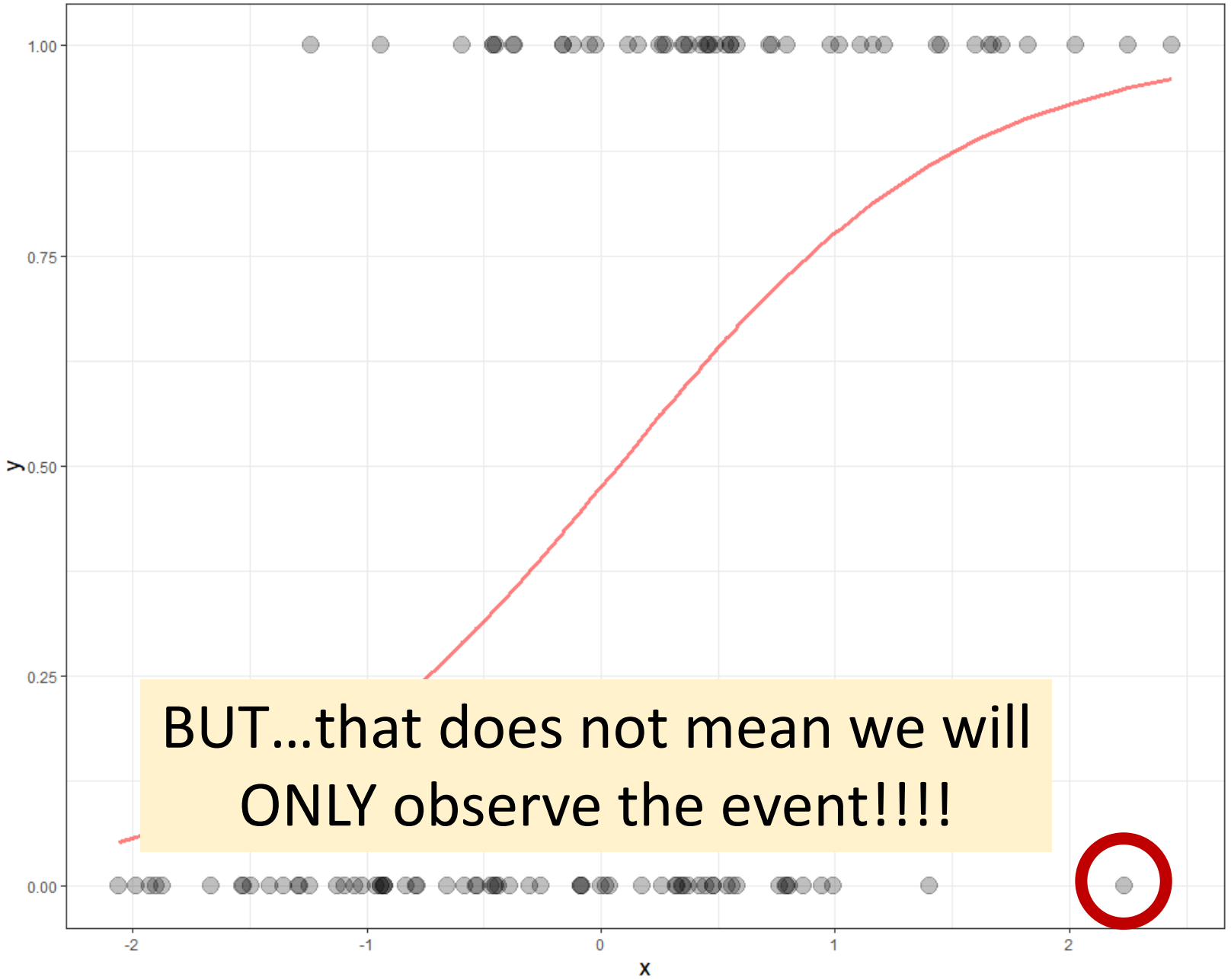


Include the TRUE EVENT PROBABILITY for reference

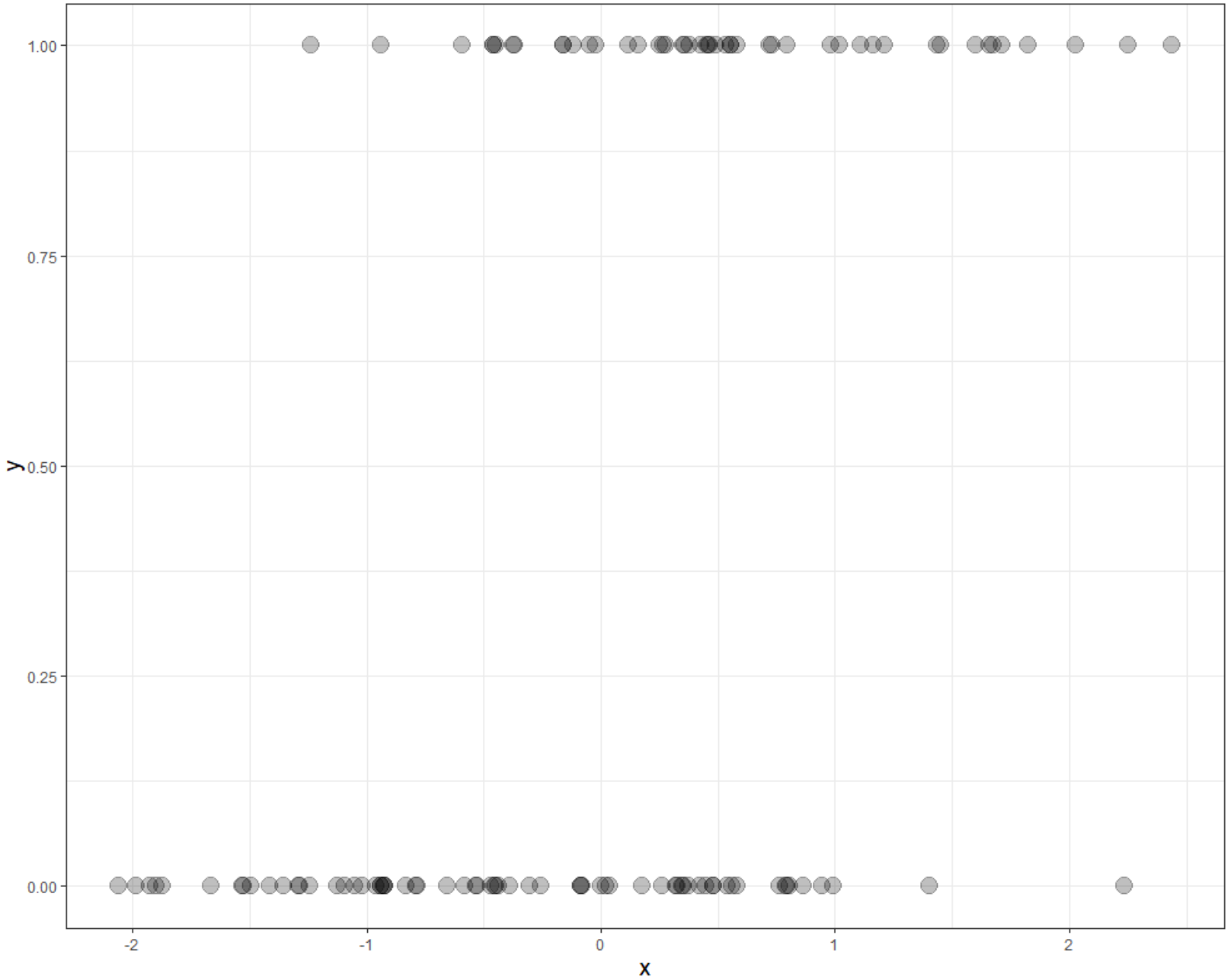


When the event probability is high, we typically observe the event

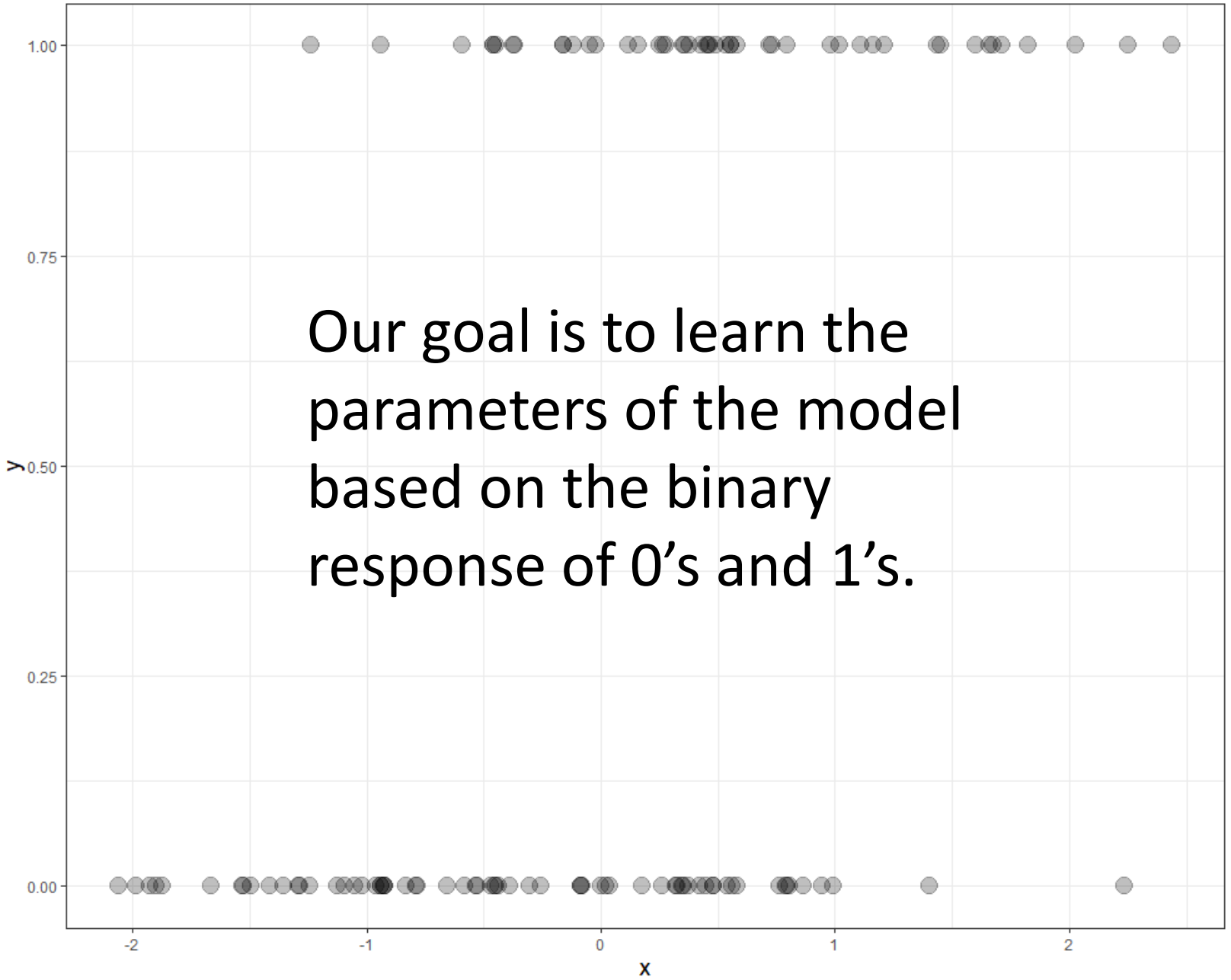
Include the TRUE EVENT PROBABILITY for reference



With real data…we will **NEVER** know the TRUE event probability



Plot the binary outcomes with respect to x



Our goal is to learn the parameters of the model based on the binary response of 0's and 1's.

We will again let `R` handle the heavy lifting for us and call an existing function to fit a model

- We can fit the model in `R` by using the `glm()` function and the formula interface.

- The function call is similar to `lm()`, except we must specify the **distribution** or **likelihood family**.

- We will discuss why that is later in the semester. For now, it's enough to say the **family** sets the **loss function**.

The model summary is displayed below

```
> mod1 <- glm(y ~ x, data = my_data, family = "binomial")
> mod1 %>% summary()

Call:
glm(formula = y ~ x, family = "binomial", data = my_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1457  -0.9159  -0.4746   1.0833   2.0255

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.4495     0.2180  -2.062   0.0392 *
x             1.1838     0.2671   4.432 9.36e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 156.27  on 114  degrees of freedom
Residual deviance: 128.82  on 113  degrees of freedom
AIC: 132.82

Number of Fisher Scoring iterations: 4
```

Rather than discussing the parameters, let's interrogate the model behavior via predictions

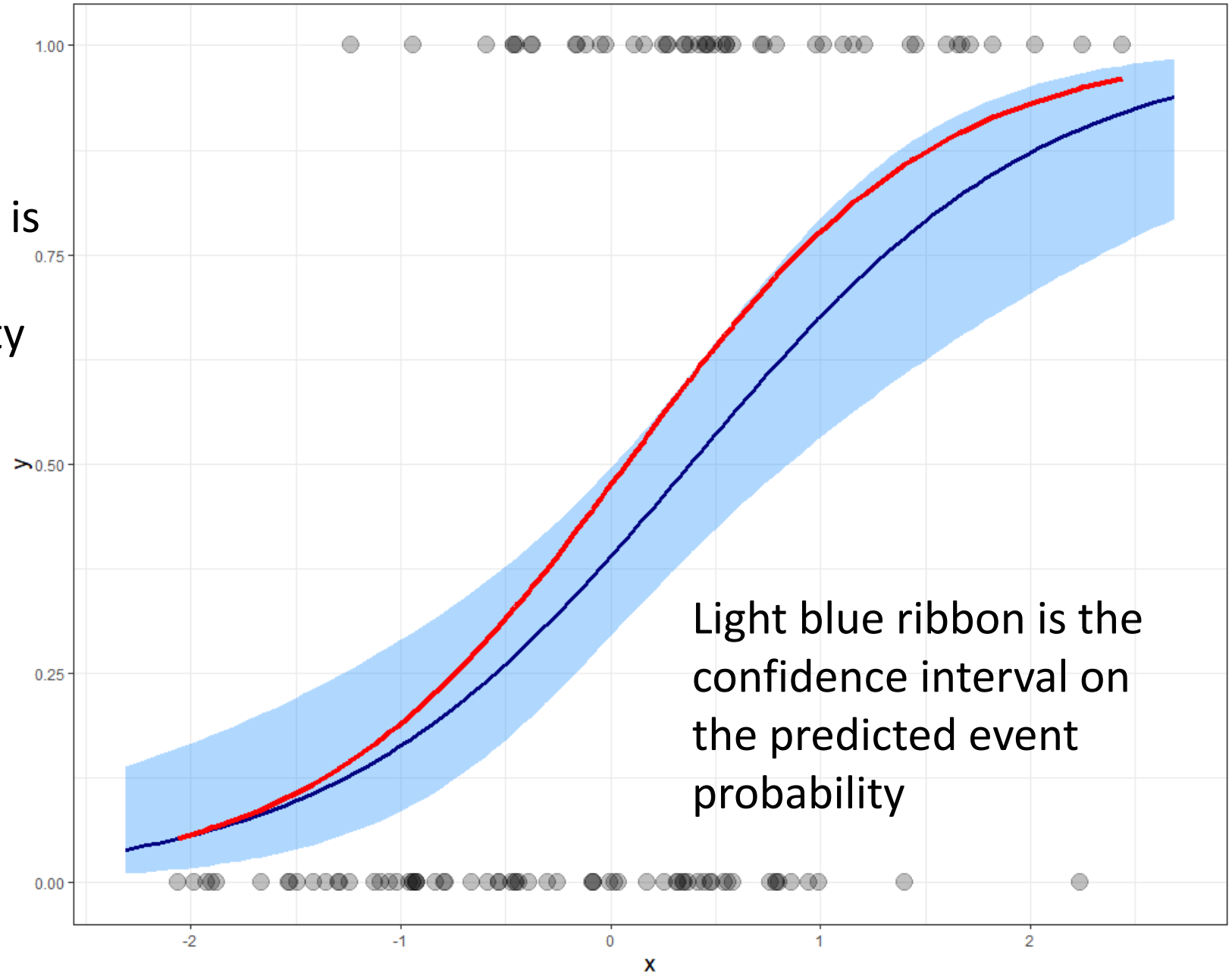
- A test or prediction grid was created using 101 evenly spaced points over the range of the input values in `my_data`.

- The code for how to make predictions will not be shown yet since it is introduces terminology that we will discuss in depth later in the semester.

- For now, just consider that the fit model was used to predict the EVENT PROBABILITY over a range of input values.

EVENT PROBABILITY predictions with respect to the x compared to the TRUE EVENT PROBABILITY

Dark blue curve is the predicted event probability



How can we assess model performance?

- We will NEVER know the TRUE event probability…
- We observe an EVENT…which we have encoded as 1 or 0…
- What can we even compare our prediction to?

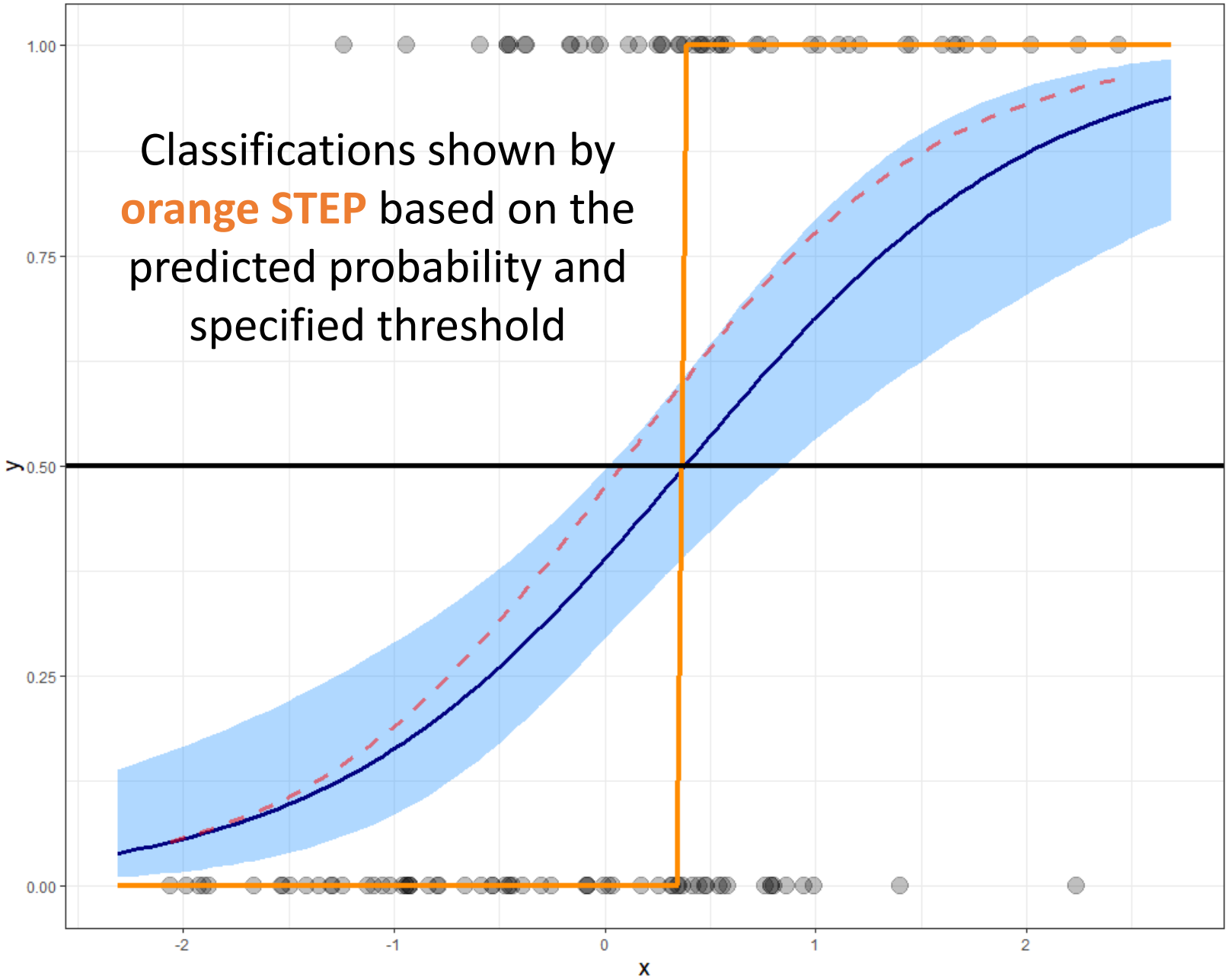
In order to classify which event occurs, we must make a **DECISION**

- Compare the predicted probability to a **THRESHOLD**.

- If the predicted probability is **GREATER** than the threshold, set the predicted class to $y = 1$ or the **EVENT**.

- If the predicted probability is **LESS** than the threshold, set the predicted class to $y = 0$ or the **NON-EVENT**.

The default THRESHOLD is 0.5…why do you think that is the case?



Threshold = 0.5

By classifying the prediction, we can directly compare the predicted class to the observed class

- Leads to a natural performance metric for binary classification…ACCURACY.

- Accuracy is the number of correct classifications divided by the total number of observations.

By classifying the prediction, we can directly compare the predicted class to the observed class

- HOWEVER, Accuracy does NOT consider which class was predicted correctly…

- Or, which class was INCORRECTLY predicted…

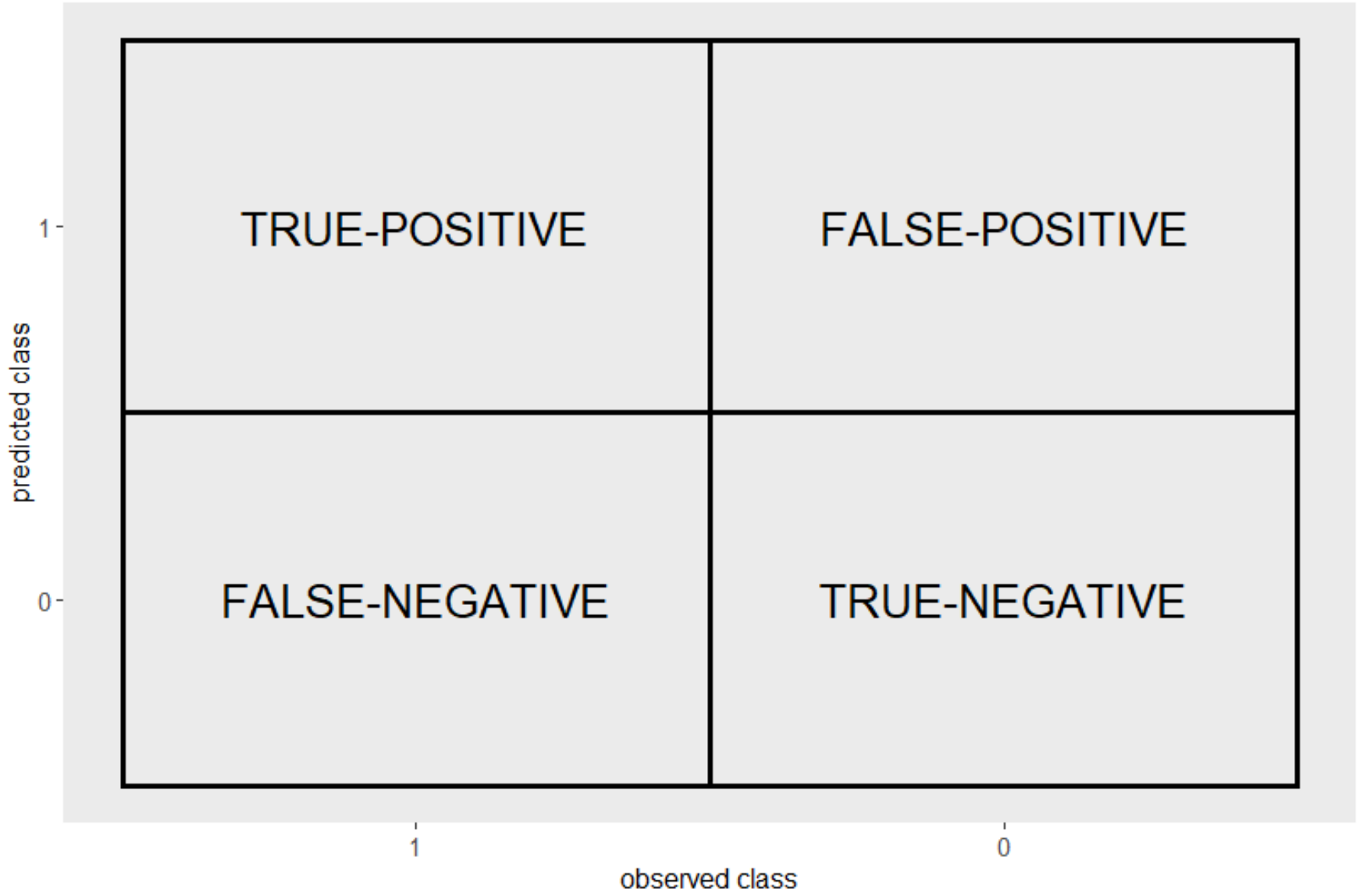
In binary classification problems there are 4 combinations of predicted and observed classes

- The 4 combinations are visualized with a **CONFUSION MATRIX**.

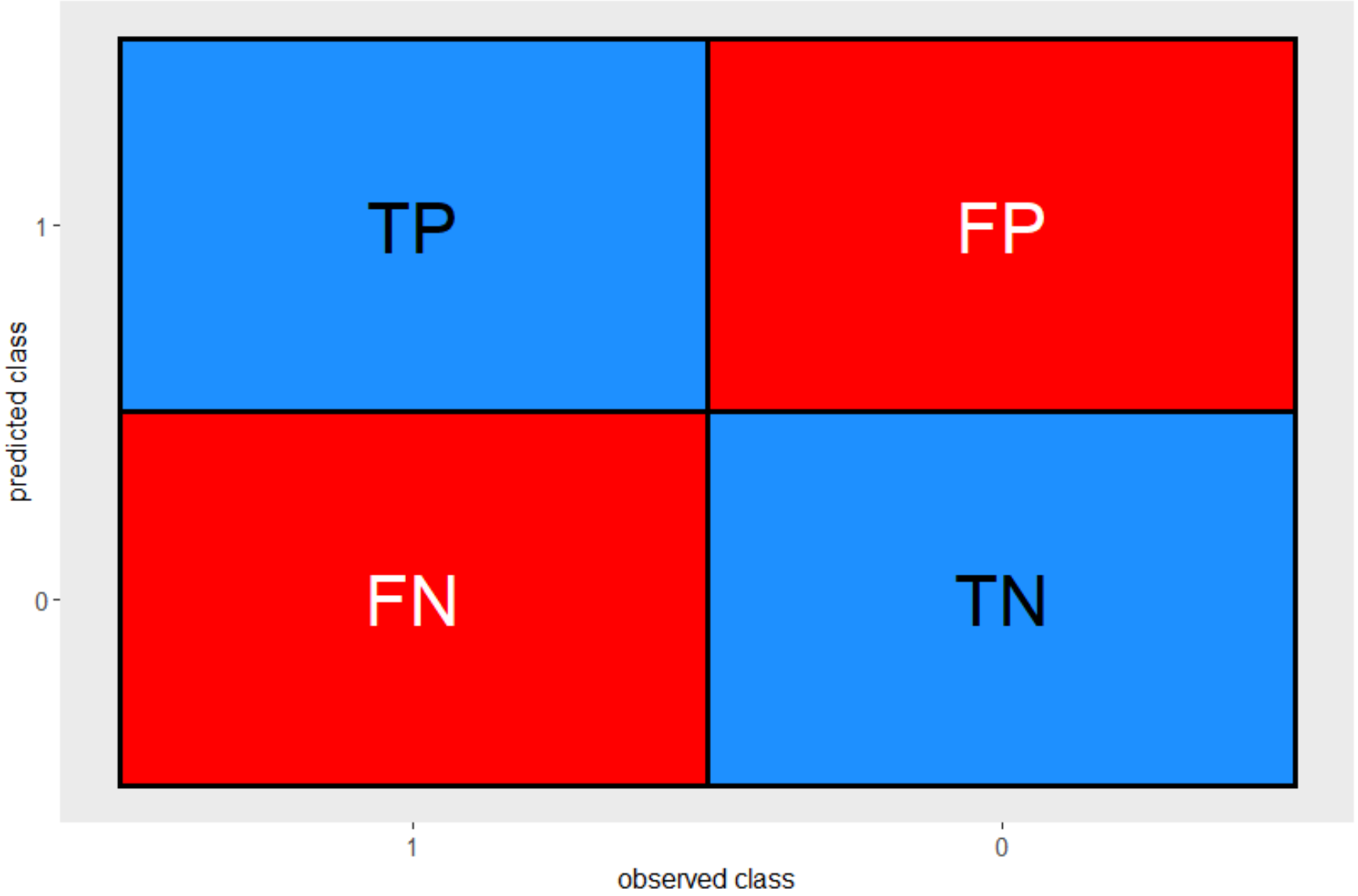
- Each cell in the matrix has a meaning.

- Each cell holds the number of times that particular combination occurred.

Confusion matrix



Confusion matrix terms typically written in shorthand



The confusion matrix allows us to calculate model accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Threshold controls the tradeoff between false-positives and false-negatives

- **False positive/Type I error:** *non-events* are incorrectly predicted to be events. These errors result in incorrectly assigning samples to the event class - false accusation
    - Important for: spam email (avoid hiding important messages), **criminal recidivism**, plagiarism detection
- **False negative/Type II error:** *events* are incorrectly predicted to be non-events. These errors result in missing cases that should be identified.
    - Important for: disease detection, failure/risk analysis,

We can calculate metrics which evaluate the TRADE-OFF between the different type of ERRORS.

- **Sensitivity** or **True Positive Rate (TPR)** is the rate that the EVENT is correctly predicted out of all observed EVENTS.

$$Sensitivity = \frac{TP}{TP + FN}$$

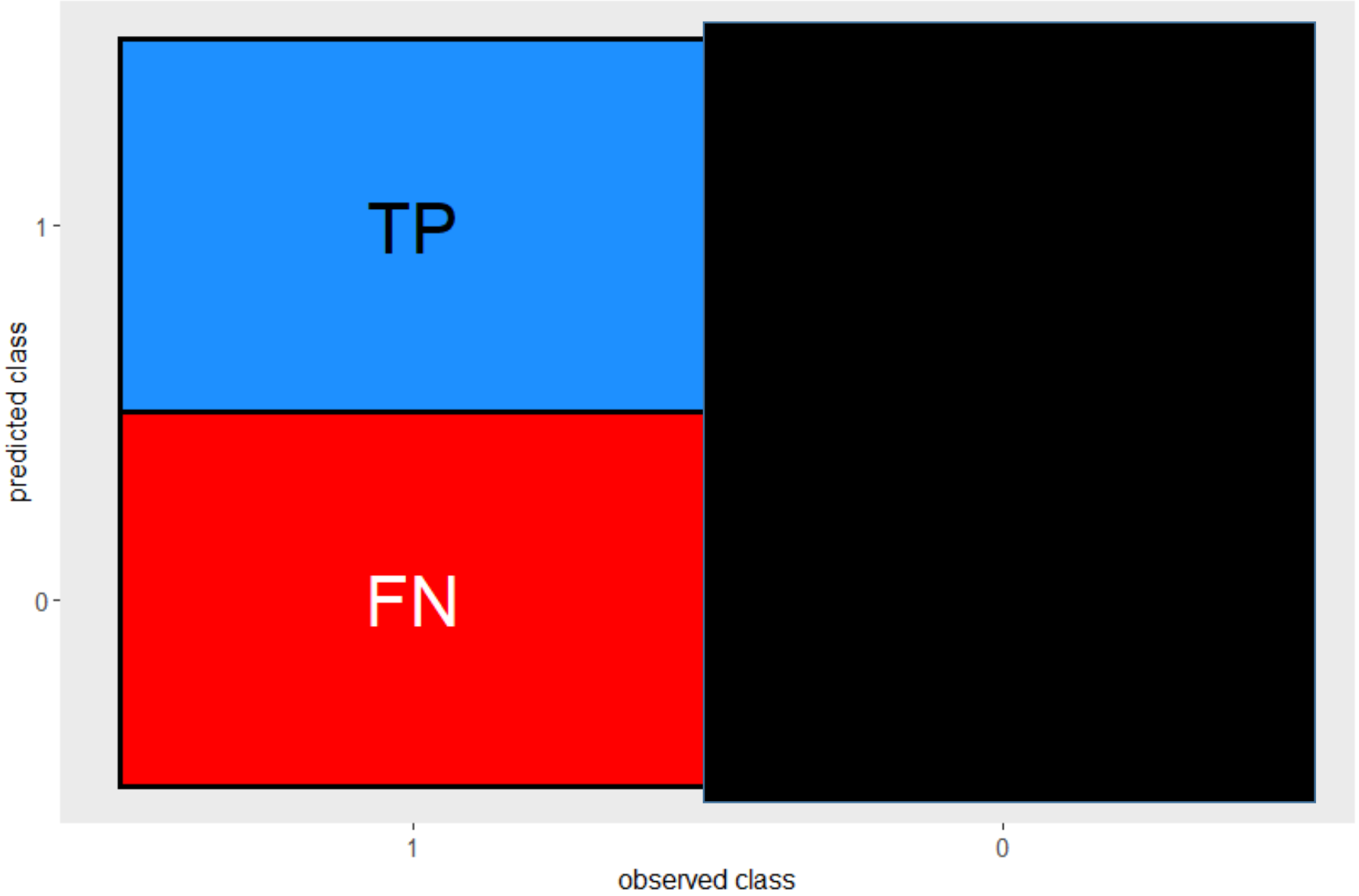
Sensitivity: how many positive samples were correctly identified?

- Is the model *sensitive* to events, does it notice when one of the samples belongs to the positive class? True positive predictions divided by *all* positives (*true positives* and *false negatives*)

$$\frac{TP}{TP + FN} = \frac{p(\hat{y} = 1 \,|\, y = 1)}{p(y = 1)}$$

$$p(y = 1) = p(\hat{y} = 1 \,|\, y = 1) + p(\hat{y} = 0 \,|\, y = 1)$$

Sensitivity **ONLY** considers OBSERVED EVENTS



We can calculate metrics which evaluate the TRADE-OFF between the different type of ERRORS.

- **Specificity** is the fraction of correctly predicted NON-EVENTS out of all observed NON-EVENTS.

$$Specificity = \frac{TN}{TN + FP}$$

Specificity: how many negative samples were correctly identified?

- **Specificity**: what proportion of negative samples are correctly predicted by the model? True positive predictions divided by *all* positives (*true positives* and *false negatives*)

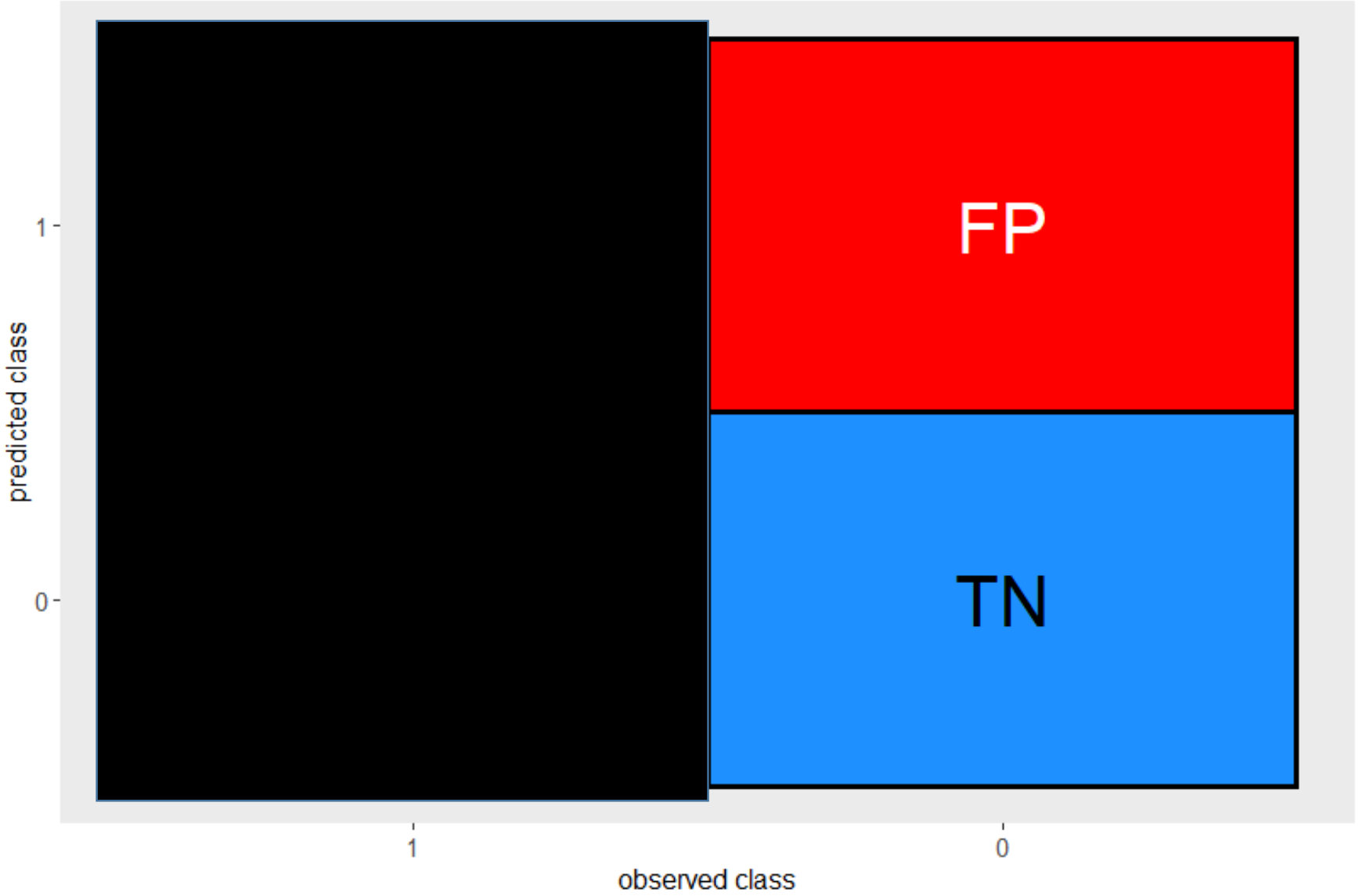
$$\frac{TN}{TN + FP} = \frac{p(\hat{y} = 0 \,|\, y = 0)}{p(y = 0)}$$

$$p(y = 0) = p(\hat{y} = 0 \,|\, y = 0) + p(\hat{y} = 1 \,|\, y = 0)$$

- **False positive rate (FPR):** what proportion of negative samples are *incorrectly* predicted by the model?

$$1 - \text{Spec} = \frac{p(\hat{y} = 1 \,|\, y = 0)}{p(y = 0)}$$

Specificity focuses on observations **without** the event



False Positive Rate (FPR)

$$FPR = 1 - Specificity$$

- FPR is the rate that the EVENT is **incorrectly** predicted out of all times we did NOT observe the event.

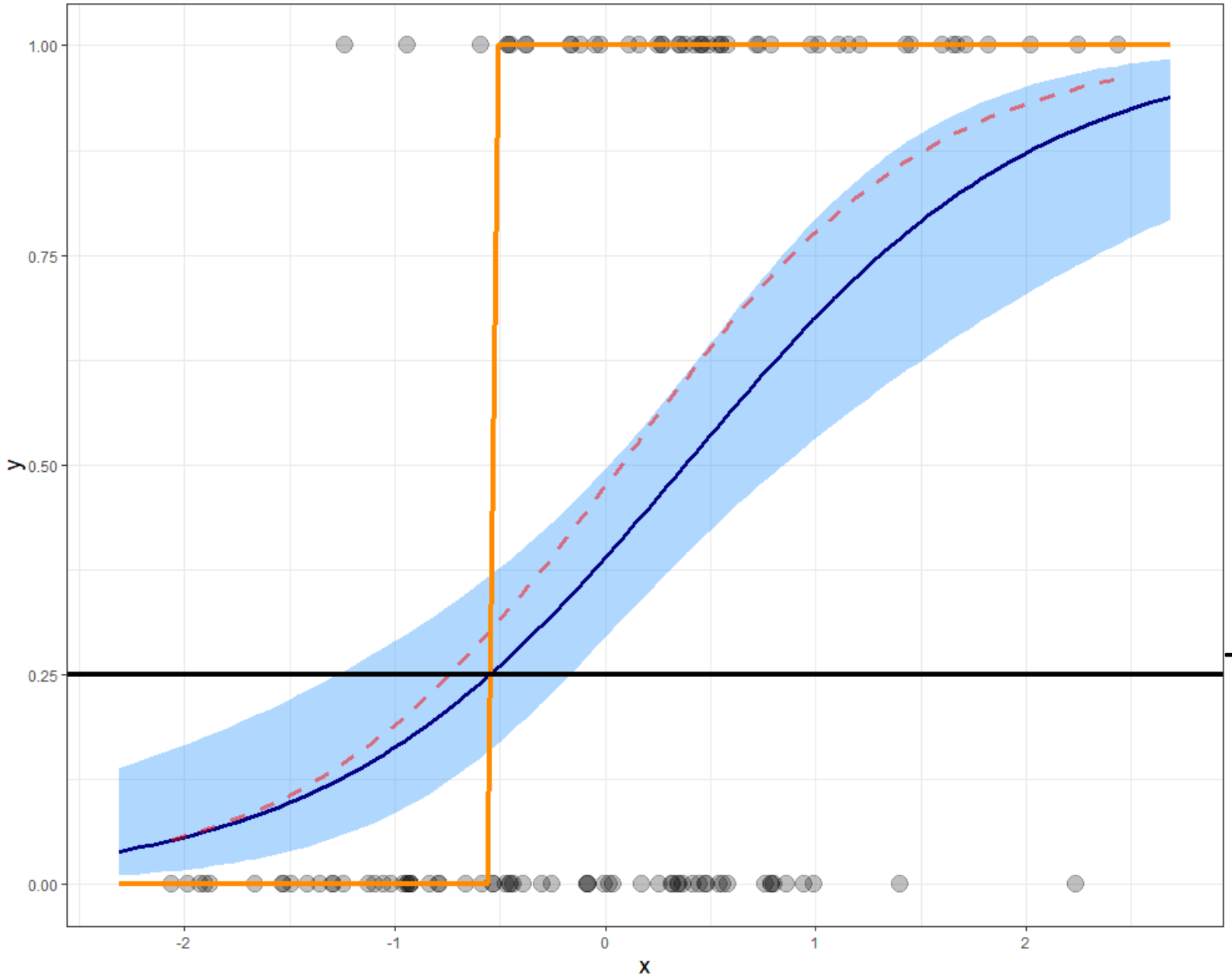
What does the TRADE-OFF represent?

- Classifications are based on comparing the predicted probability to a threshold value.

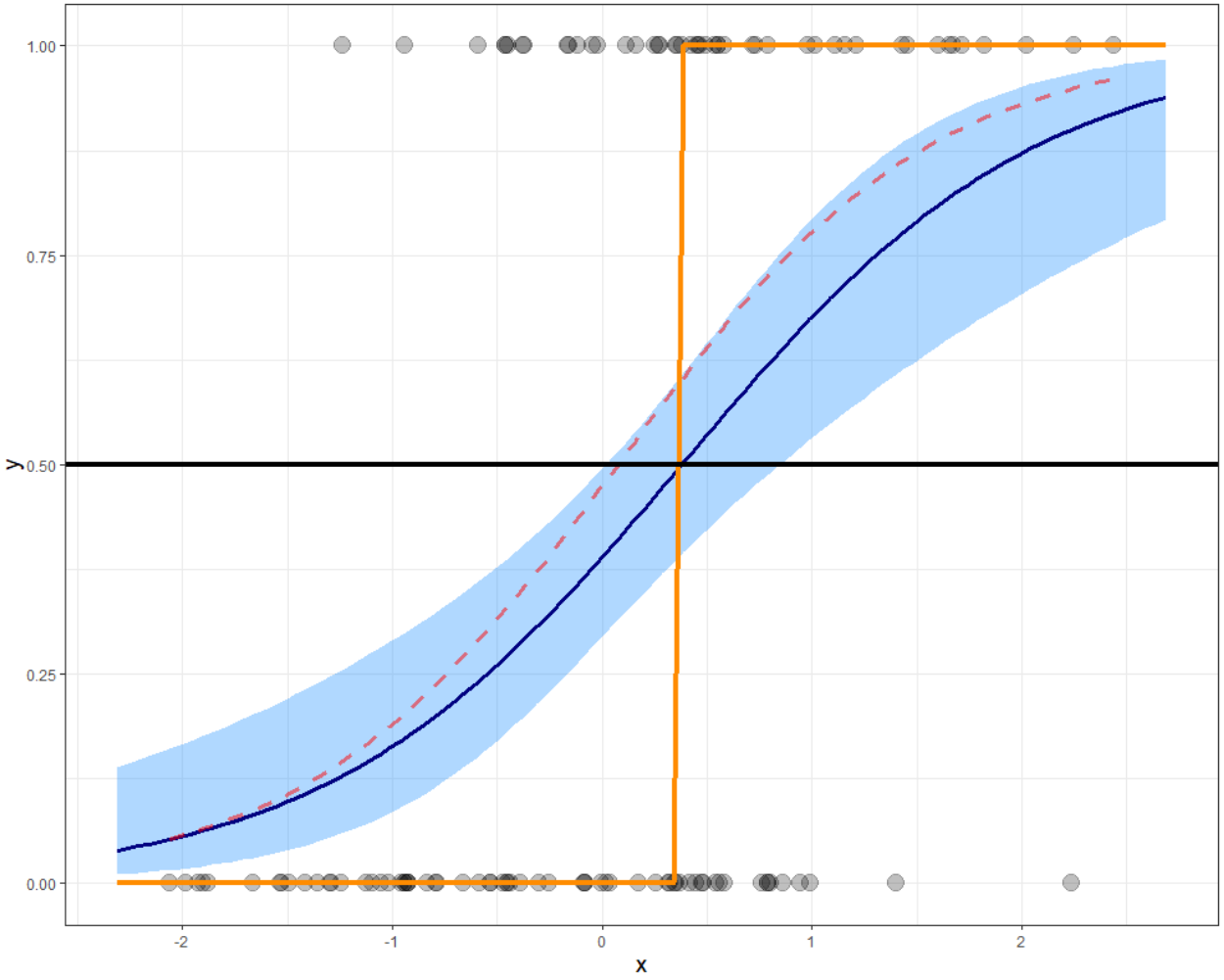
- Although 0.5 is a justifiable choice...**we can change the threshold**!

- Changing the threshold will change the predicted class, **even though the predicted probability remains the same**!

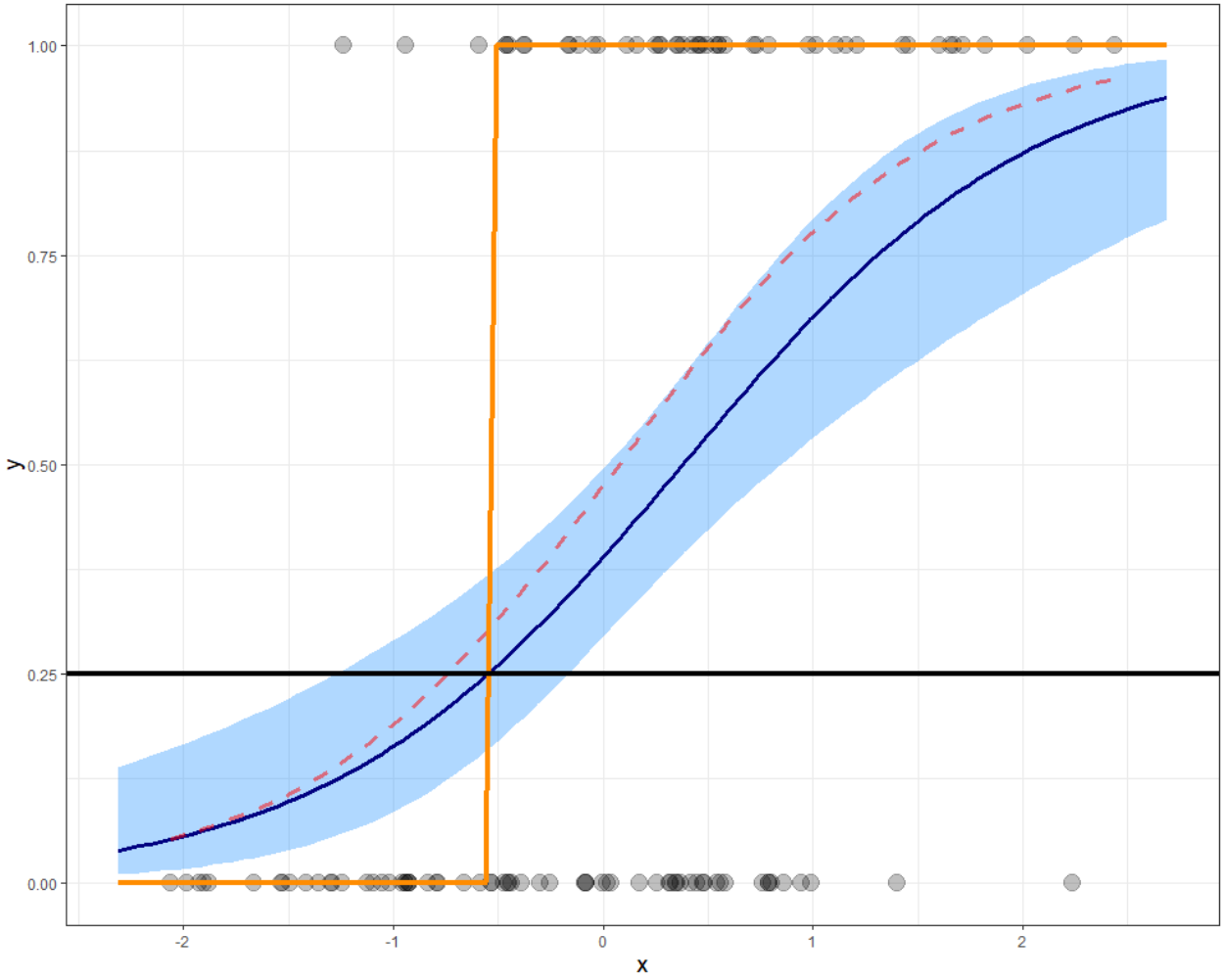
Reducing the threshold increases the number of classified events



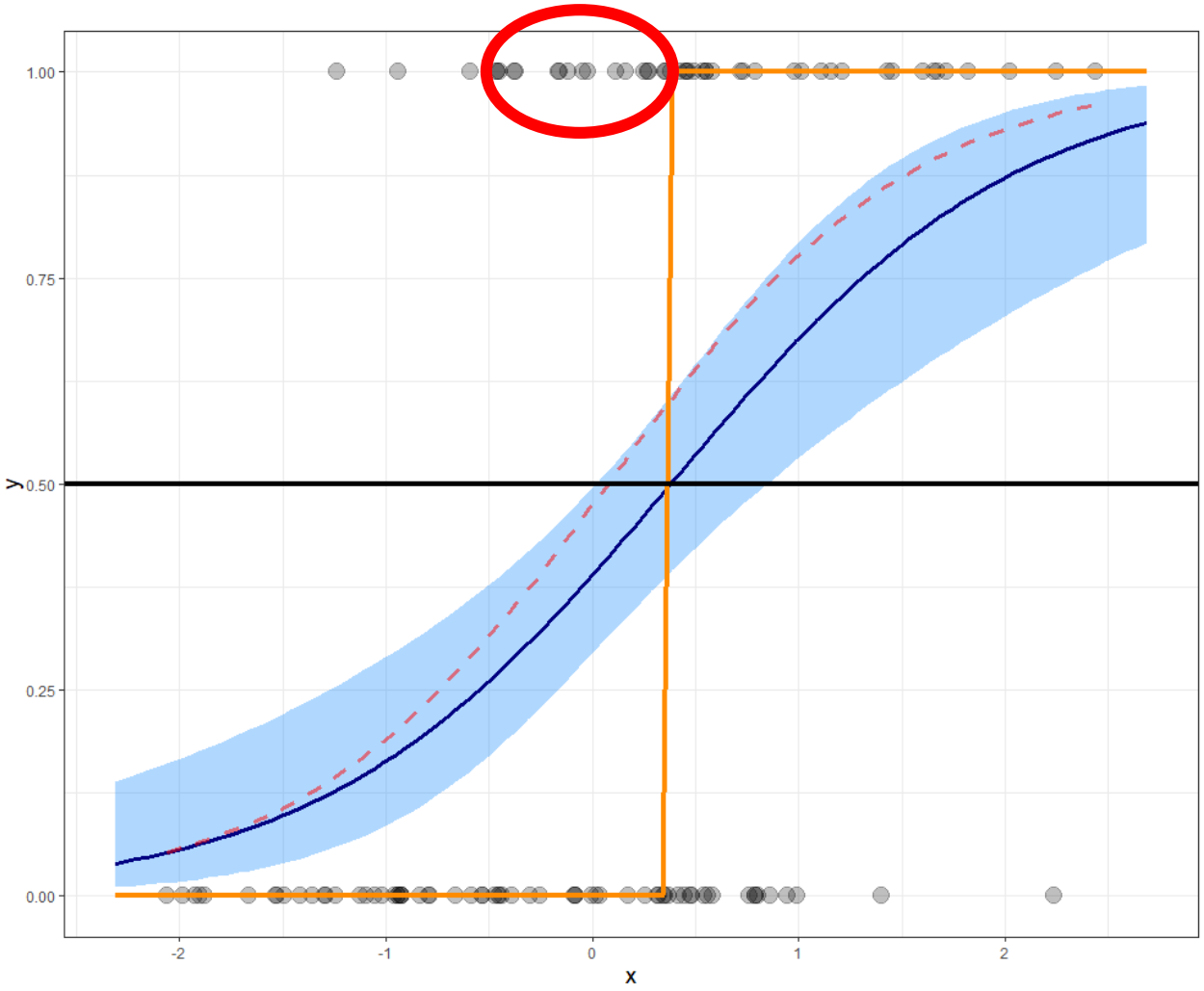
Threshold = 0.25



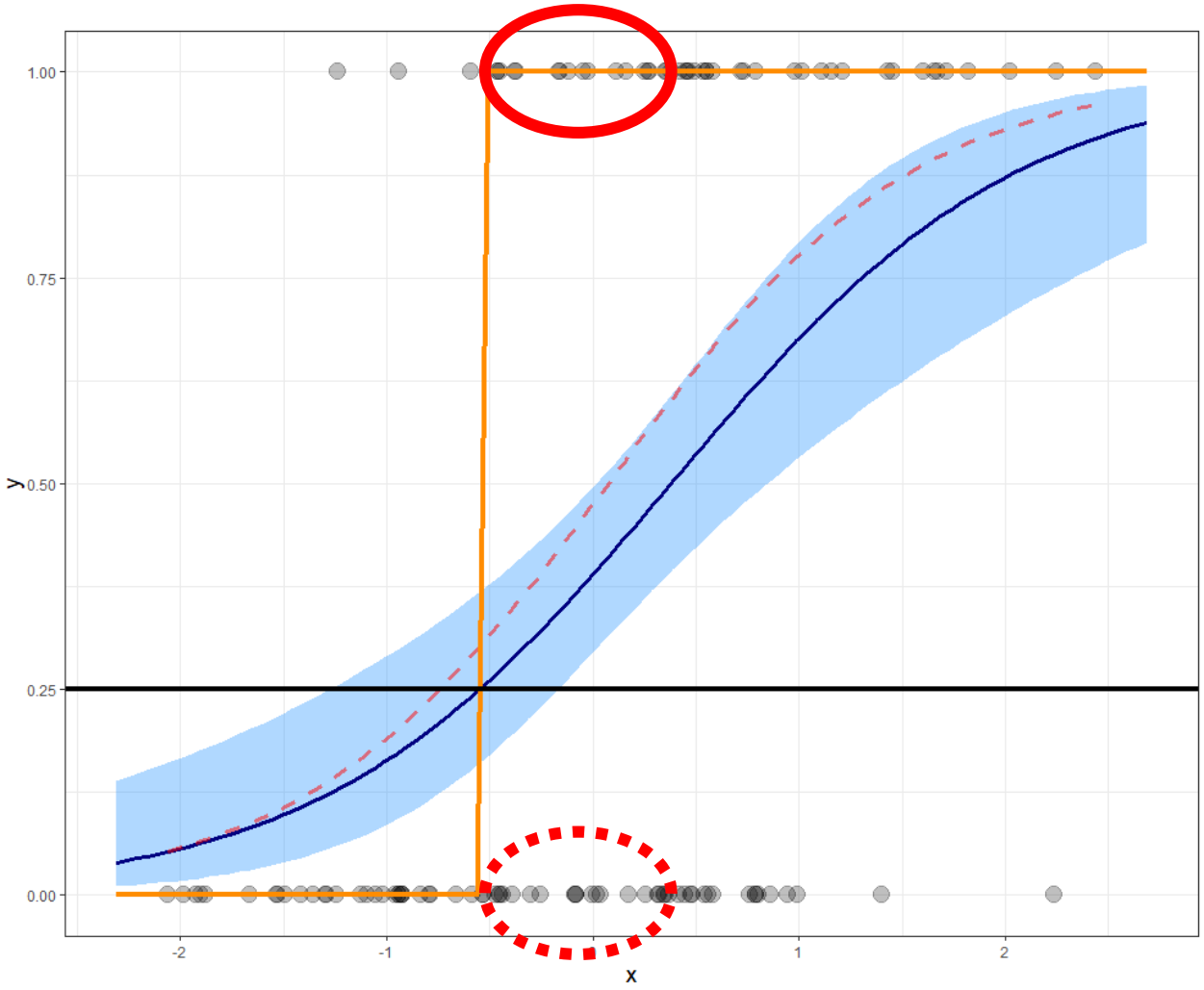
Threshold = 0.5



Threshold = 0.25

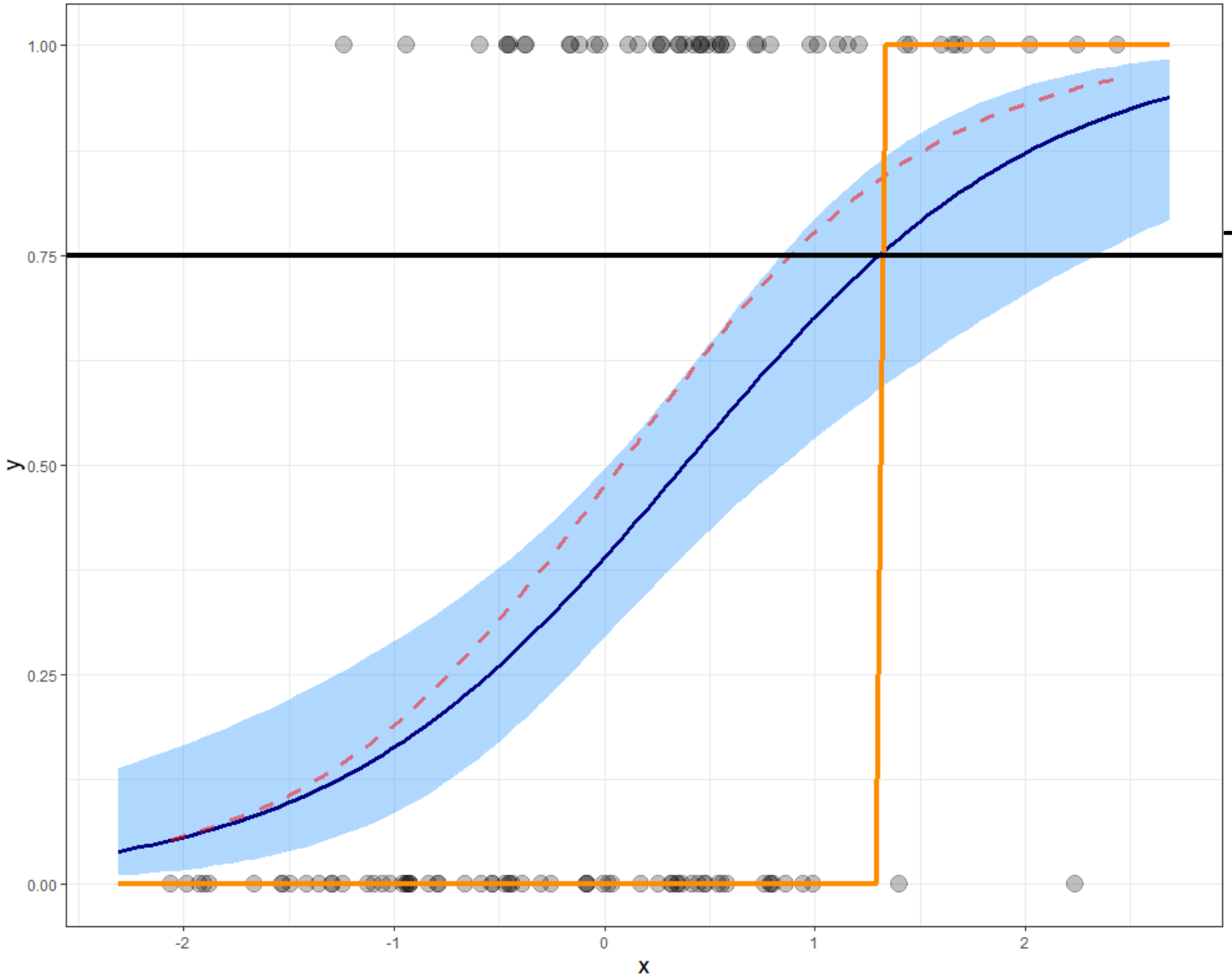


Threshold = 0.5

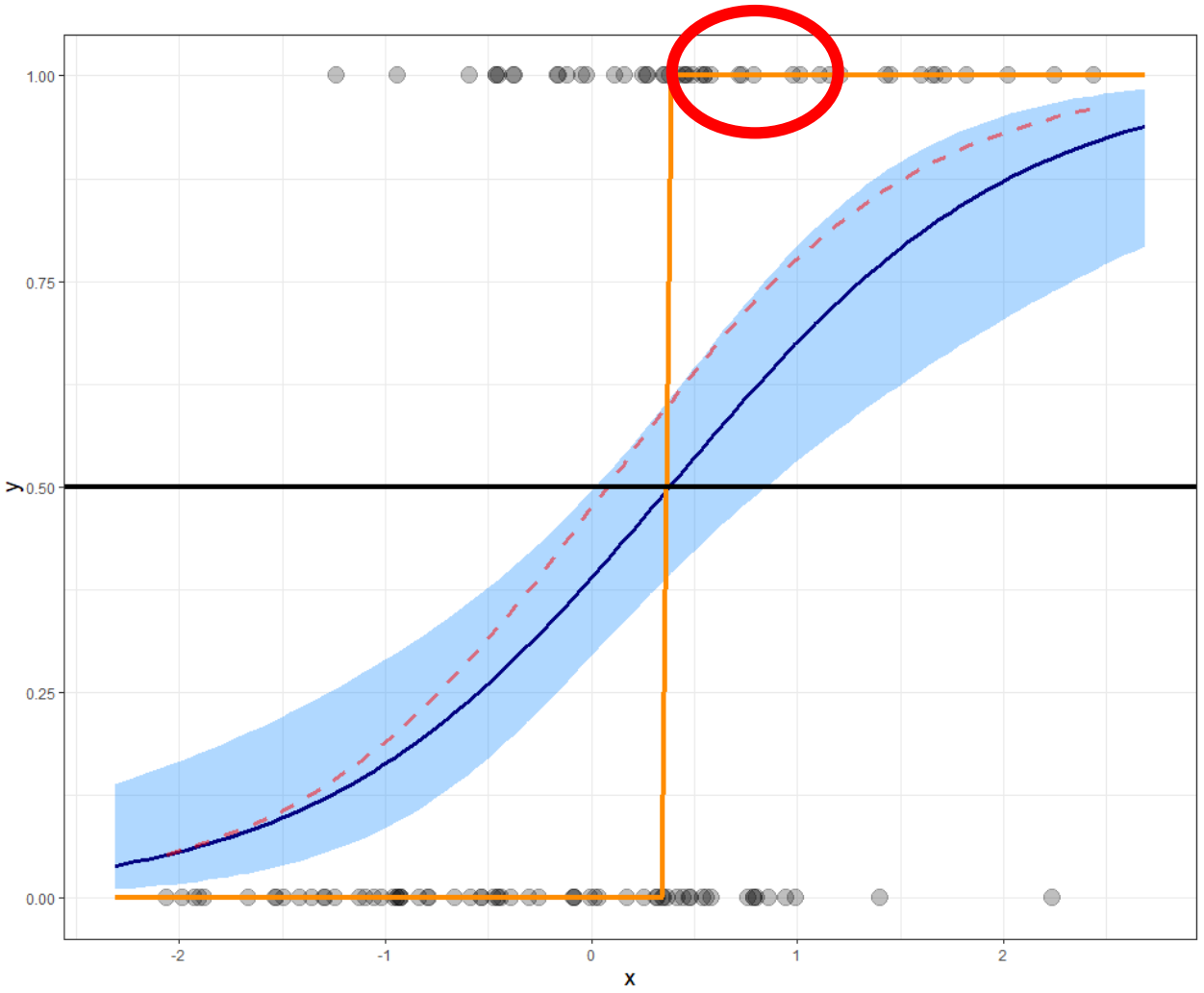


Threshold = 0.25

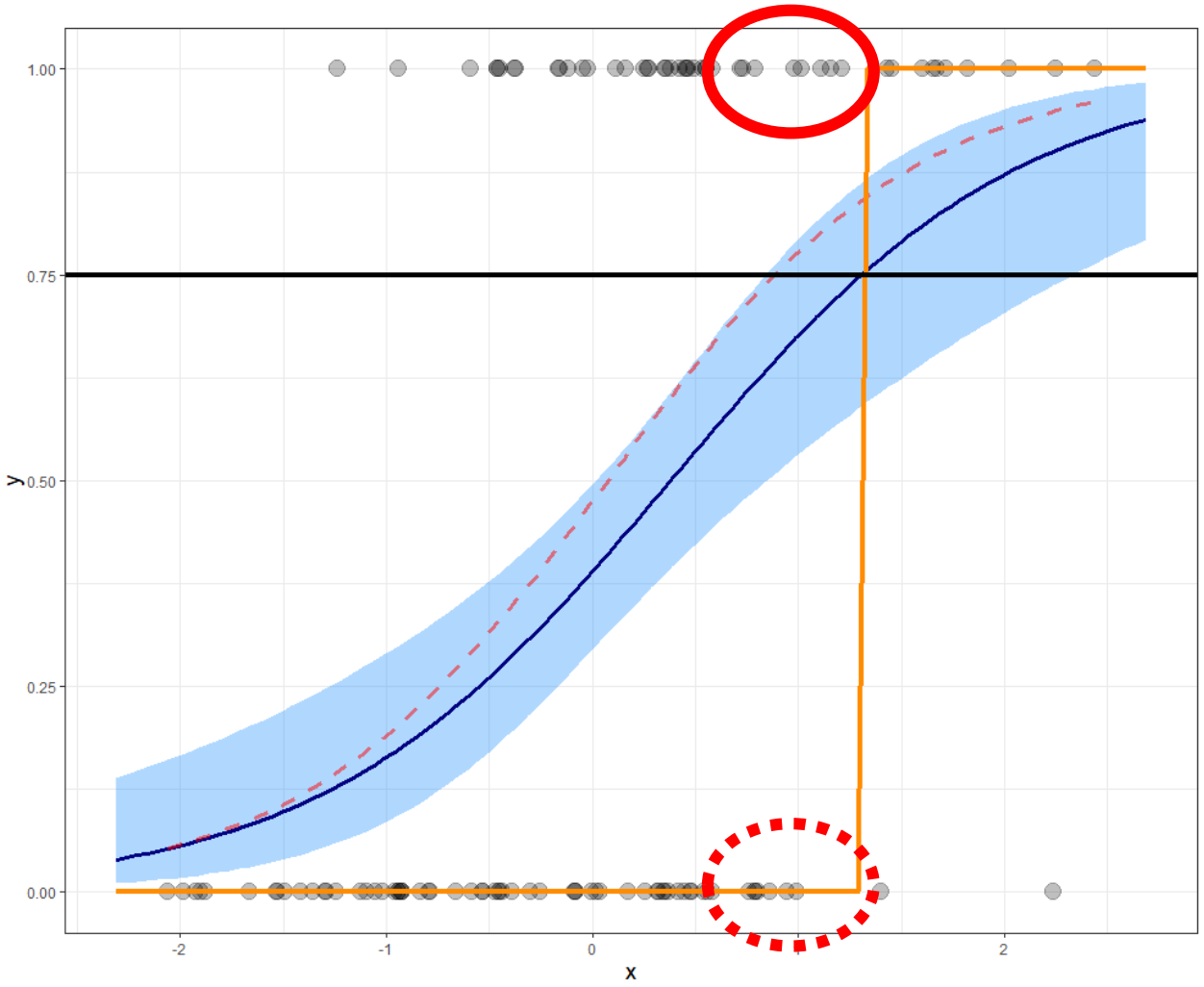
Increasing the threshold decreases the number of classified events



Threshold = 0.75



Threshold = 0.5



Threshold = 0.75

Each time we change the threshold, we need to evaluate a new confusion matrix!

- Decreasing the threshold makes it “easier” to predict the event.

- Increasing the number of predicted events increases the Sensitivity.
  - Since we will correctly predict the event more often, when the event occurs!

Each time we change the threshold, we need to evaluate a new confusion matrix!

- However, increasing the Sensitivity comes at a price!

- **Increasing the number of predicted events also increases the False Positive Rate**.
  - Since we will incorrectly classify non-events as events more often!

Why would we want to change the threshold?

- Perhaps one of the errors is WORSE than the other.
- Consider classifying cancer from a medical test.
- Incorrectly classifying a patient with cancer may result in more tests.
- However, incorrectly missing the cancer has serious consequences.

Changing the threshold many times allows to consider how TPR and FPR trade-off each other

- The trade-off is visualized with a **Receiver Operating Characteristic (ROC) curve**.

- The ROC curve can be summarized by integrating the area under the curve, leading to the performance metric appropriately named…the **Area Under the Curve (AUC)**.

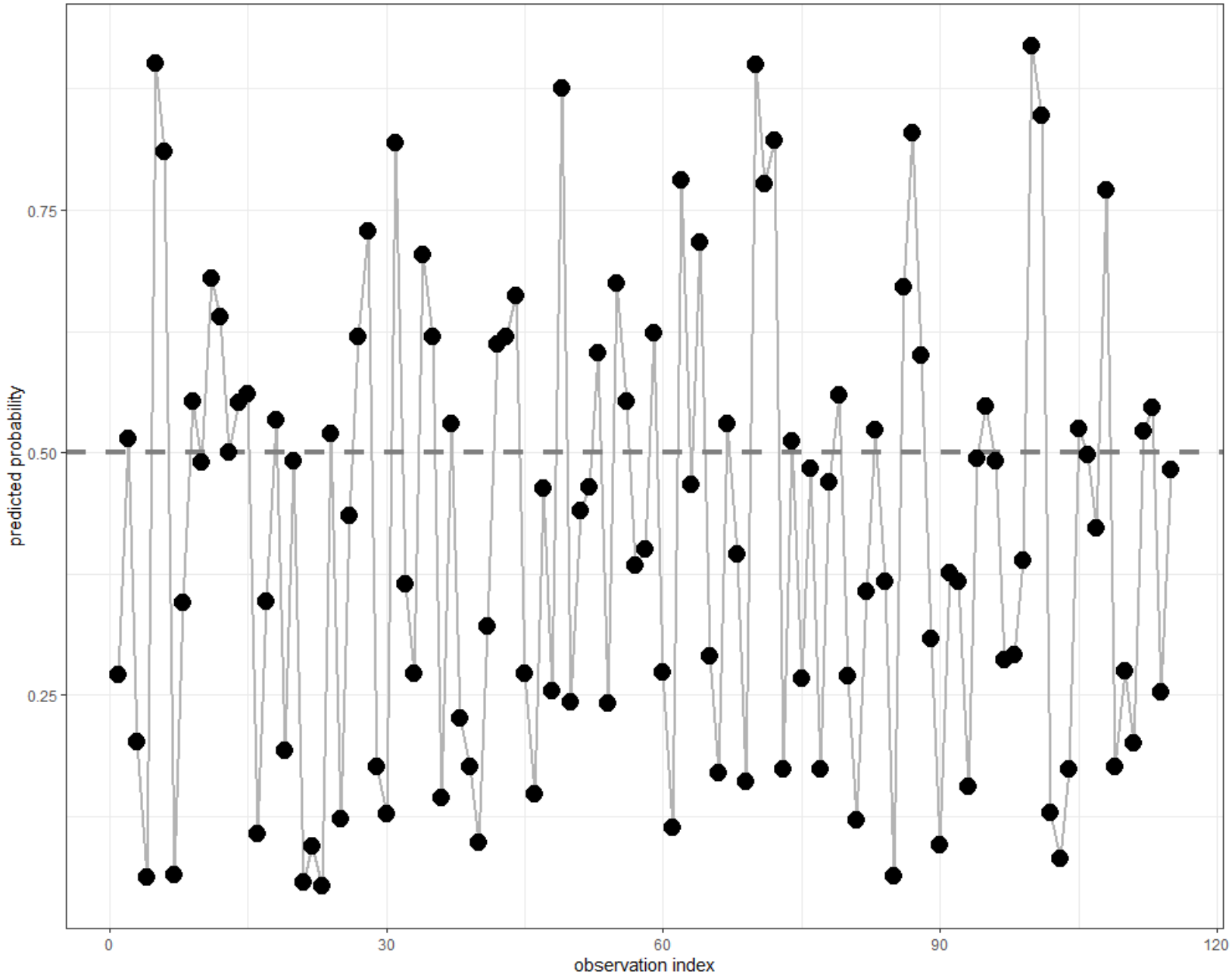
Let’s see how this all works with our simple demonstration problem

- First consider the threshold value of 0.5.

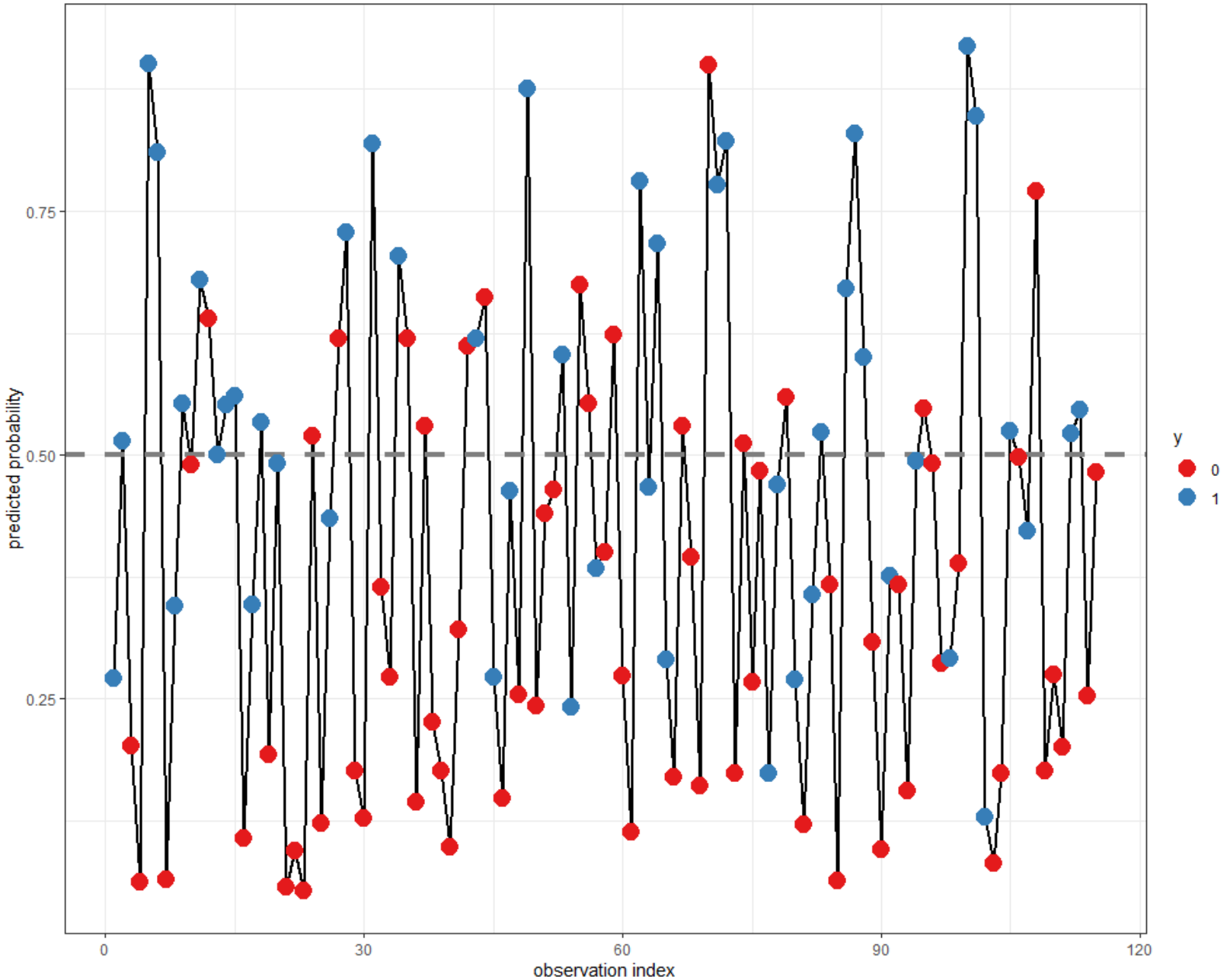
- We will look at predictions on the training set.

- Next slide shows a figure where the x-axis is the training set observation index and the y-axis is the model predicted probability.

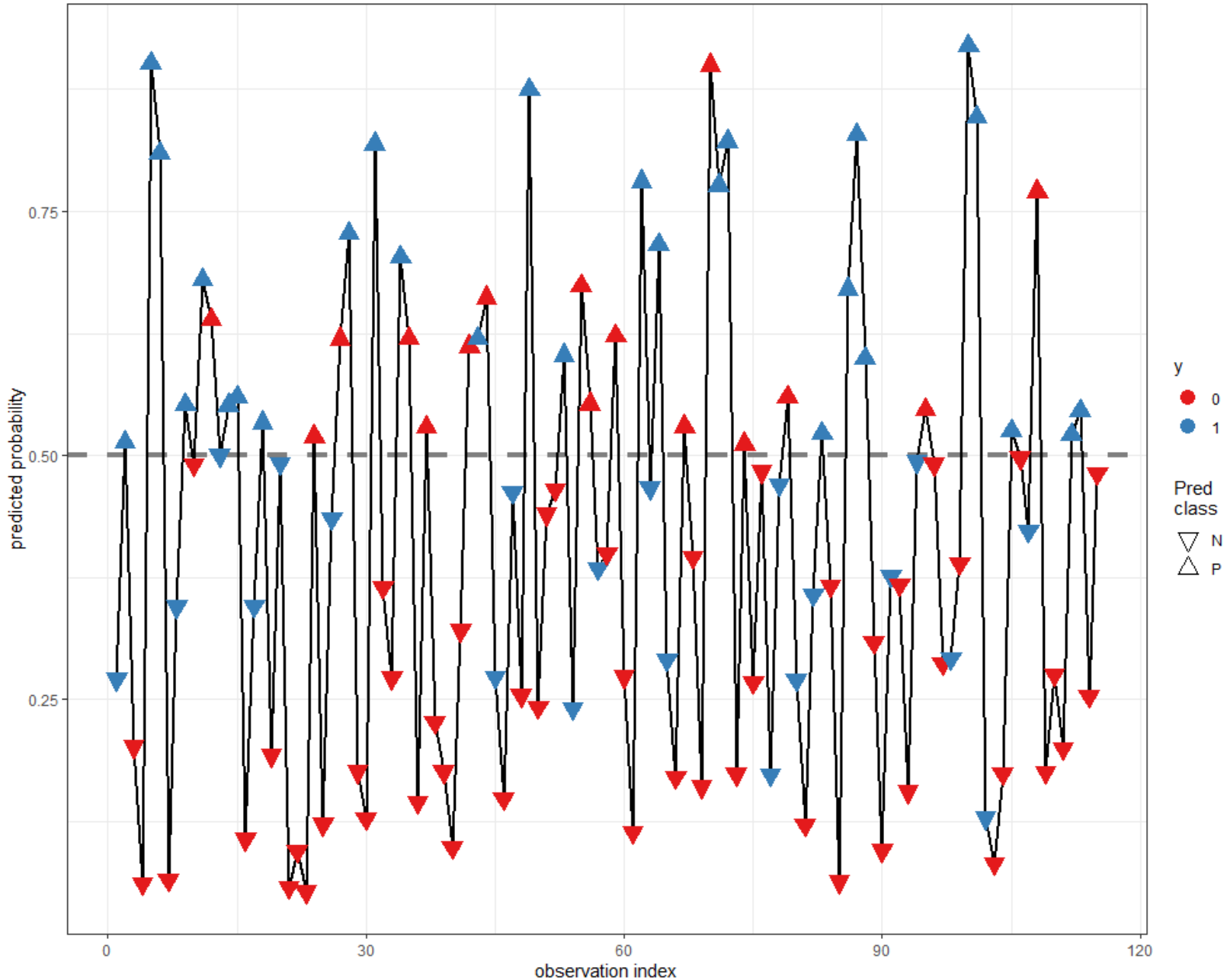
Predicted event probability for each training point



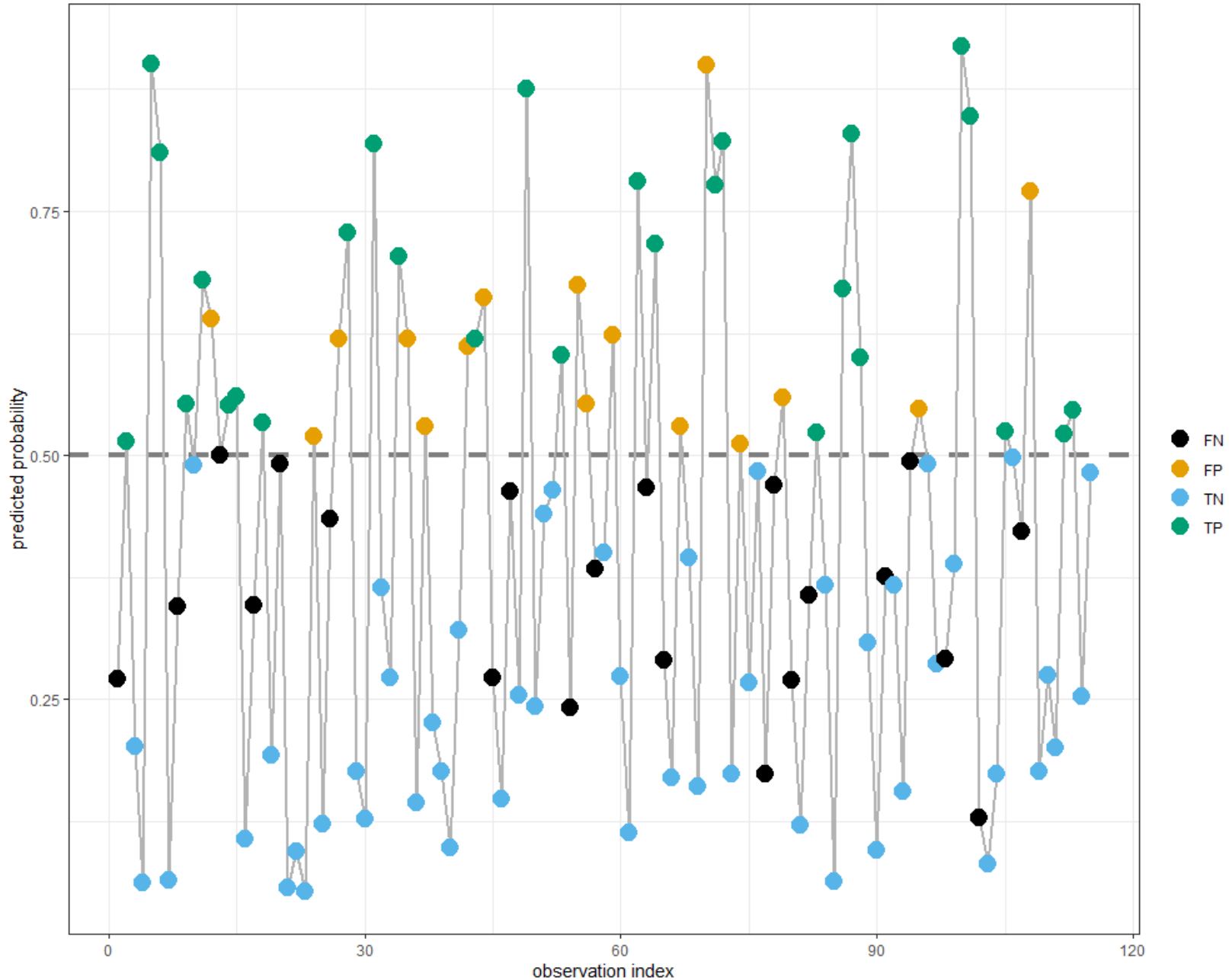
Denote the OBSERVED class for each training point



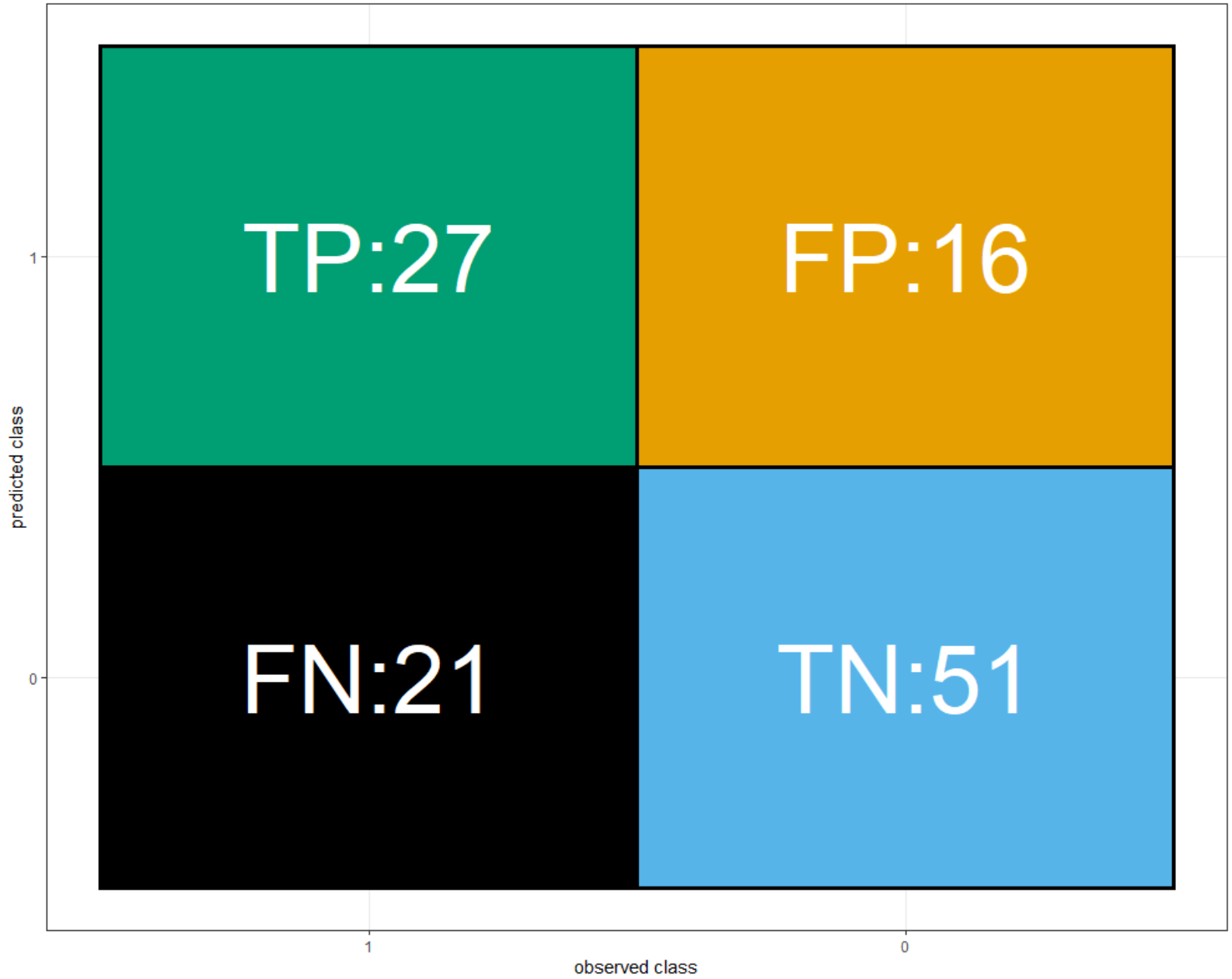
Denote the model CLASSIFICATION based on the threshold=0.5



Denote confusion matrix cell by the marker color



Count the number of observations per color to create the confusion matrix!



Calculate the confusion matrix metrics

```
# A tibble: 1 x 9
  threshold    FN    FP    TN    TP Accuracy Sensitivity Specificity   FPR
      <dbl> <int> <int> <int> <int>    <dbl>       <dbl>       <dbl> <dbl>
1       0.5    21    16    51    27    0.678       0.562       0.761 0.239
```

Compare Accuracy to the empirical fraction of events in the training set

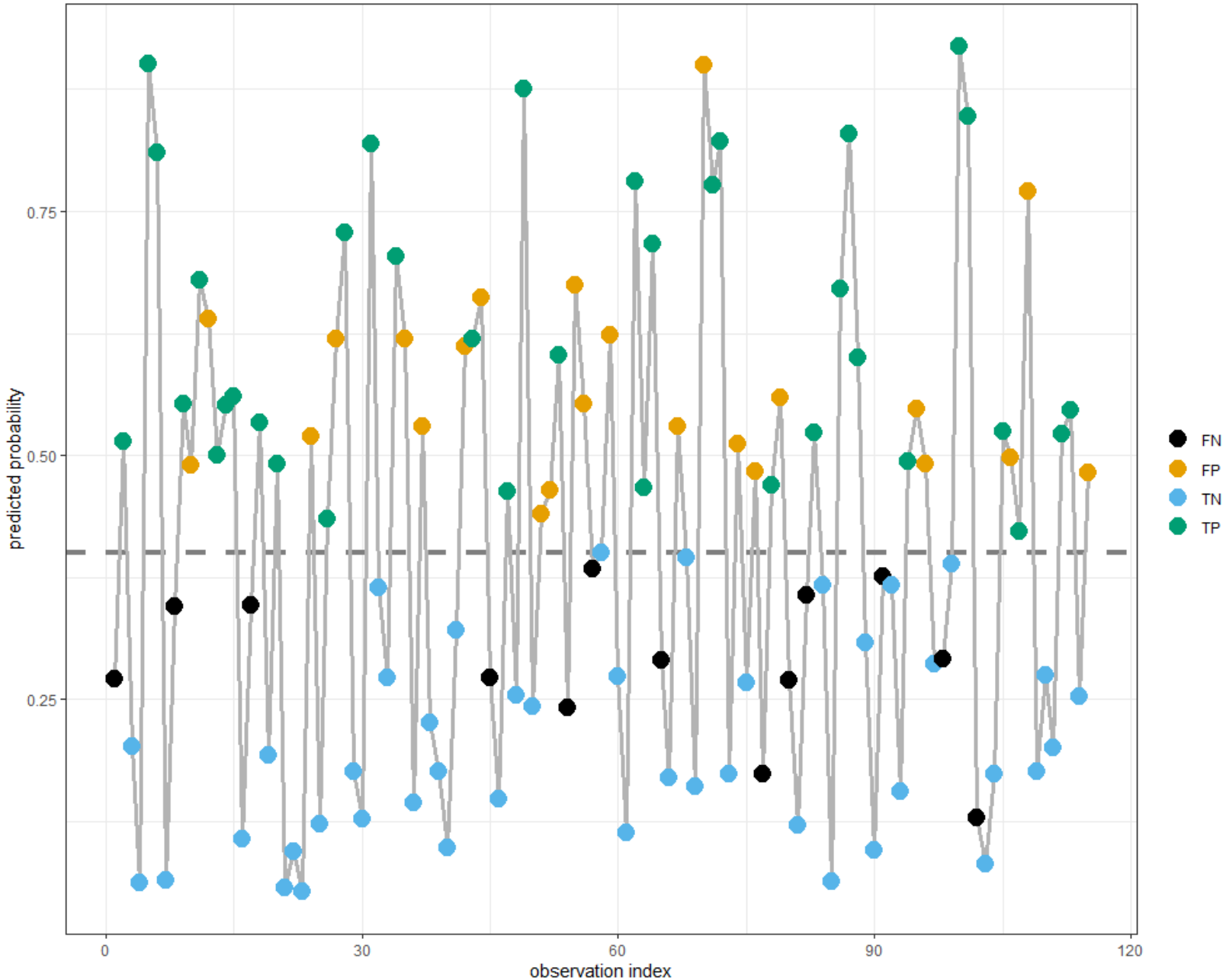
```
> mean(my_data$y == 1)
[1] 0.4173913
```

Roughly a balanced data set

Great! But this was just for ONE threshold!

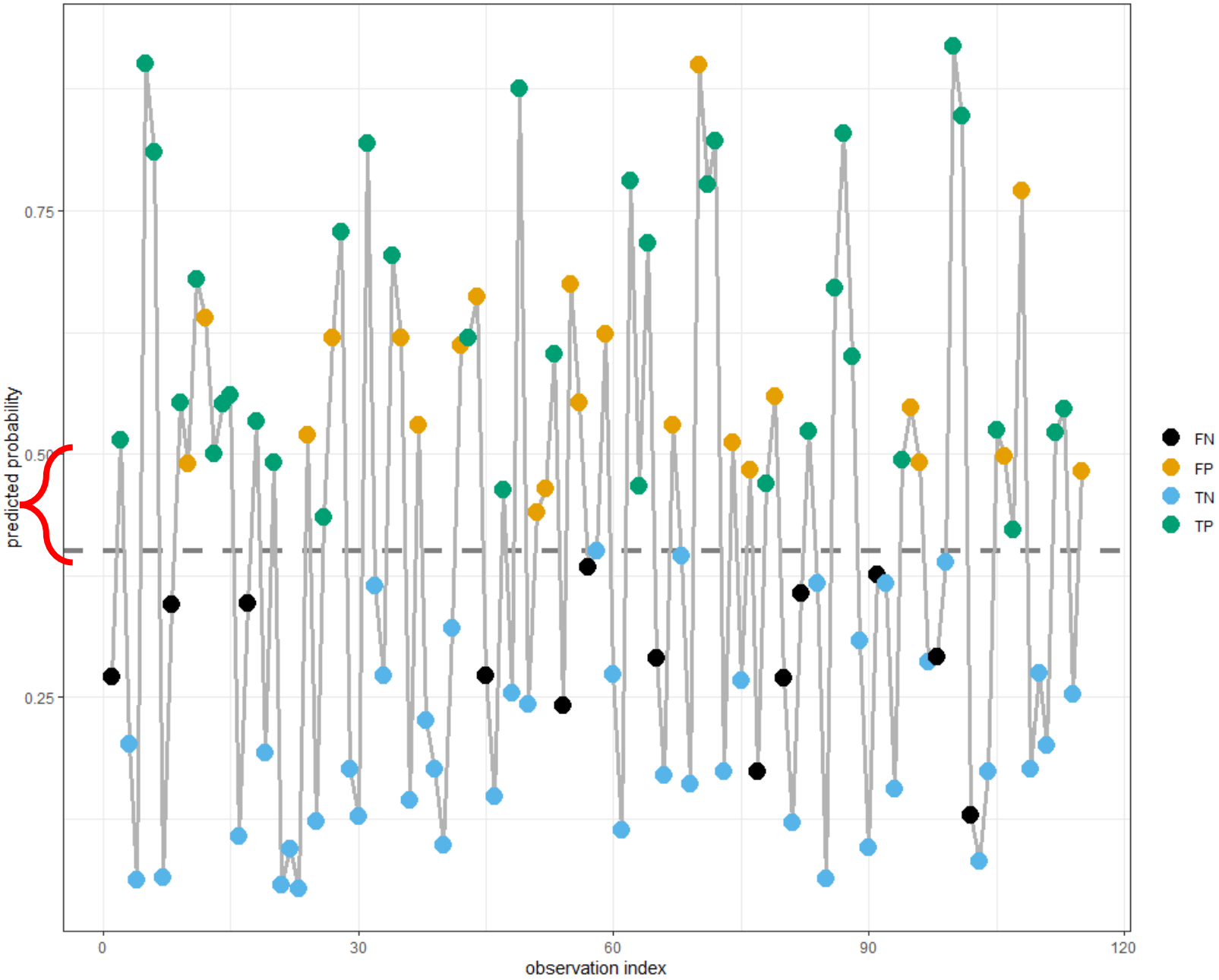
- Let’s change the threshold and repeat!

Decrease threshold from 0.5 to 0.4

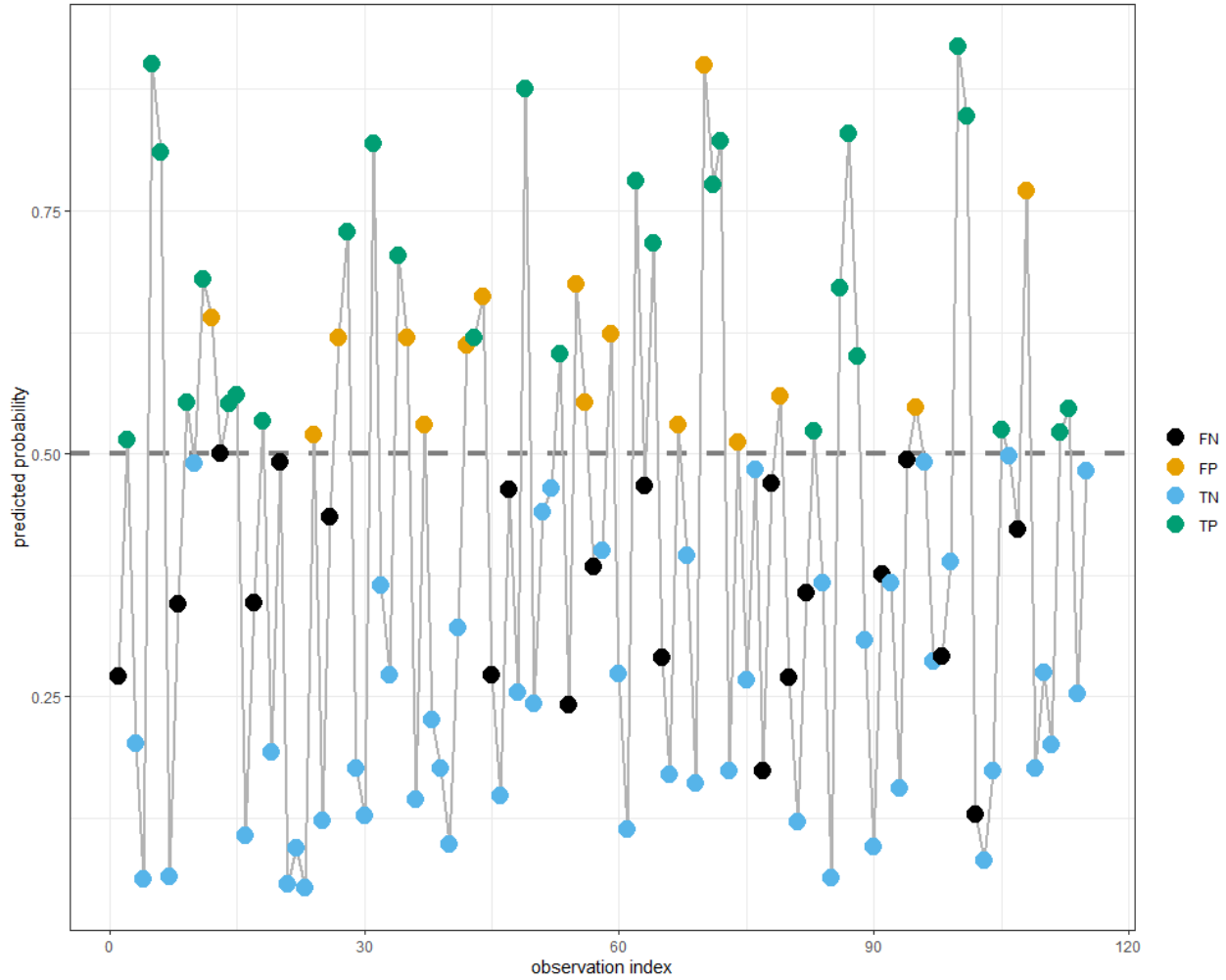


Decrease threshold from 0.5 to 0.4

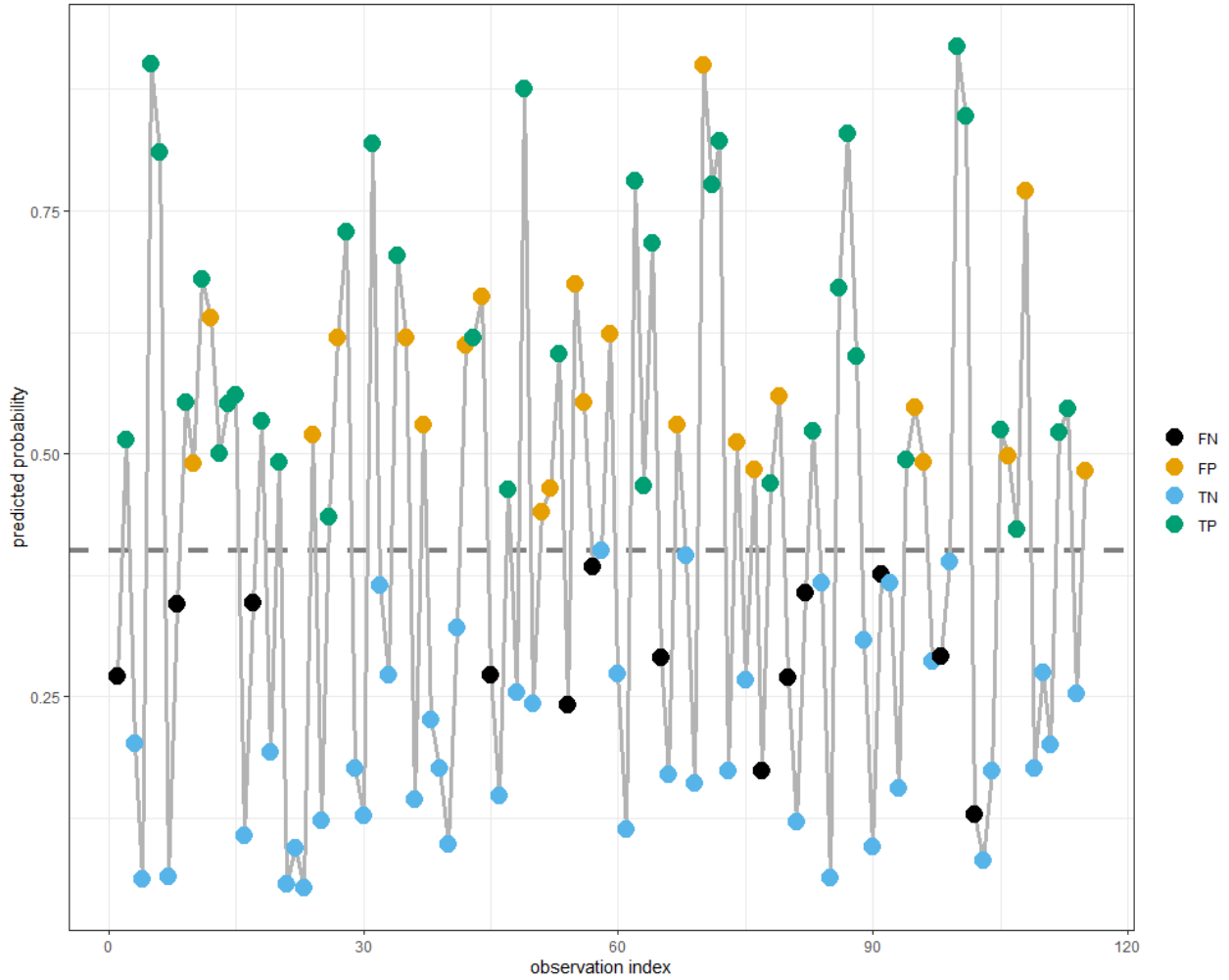
Observations between predicted probability of 0.4 and 0.5 changed elements in the confusion matrix!!



Show side-by-side to make it easier to see the changes



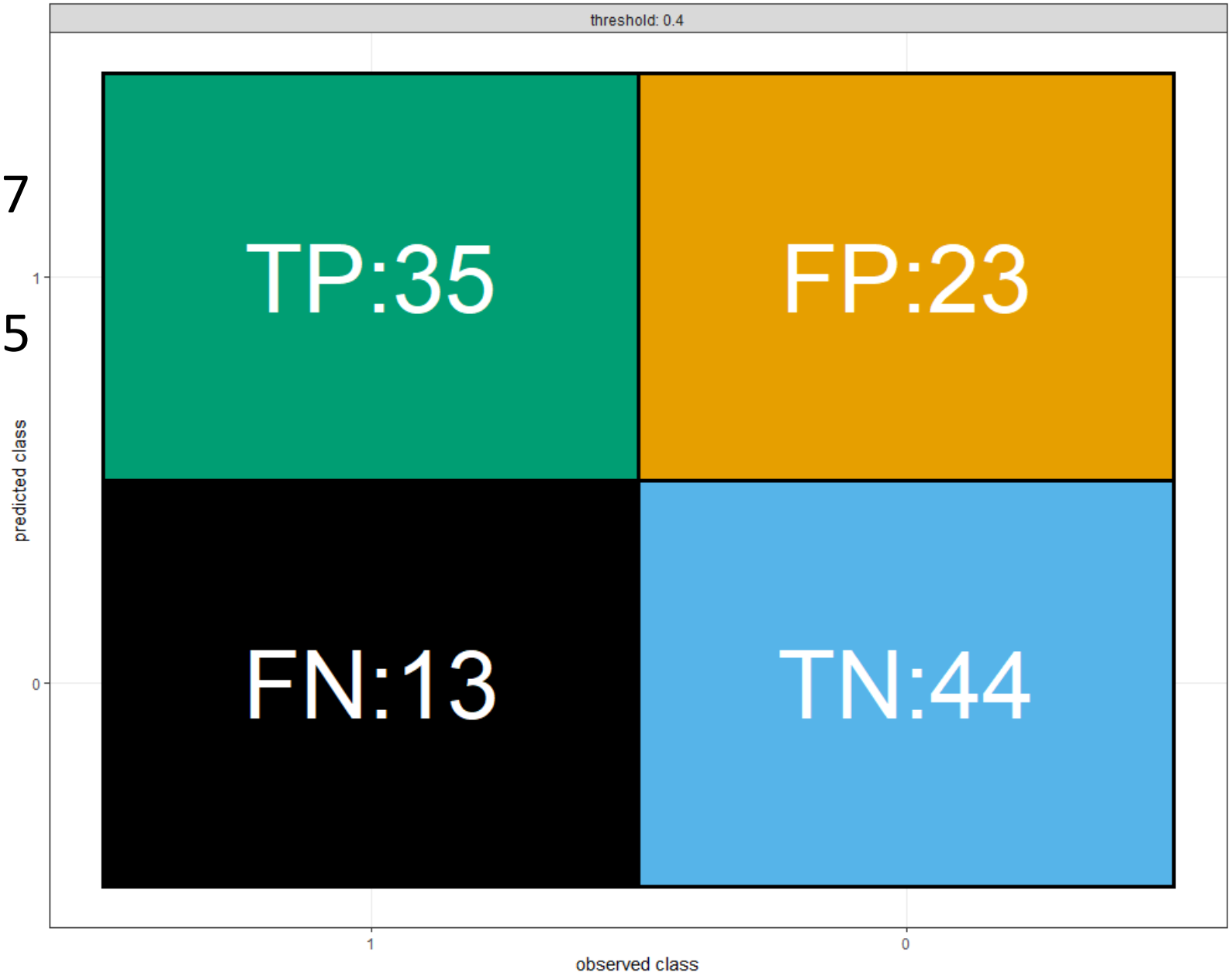
Threshold = 0.5



Threshold = 0.4

Fewer TRUE-NEGATIVES compared to the 0.5 threshold!

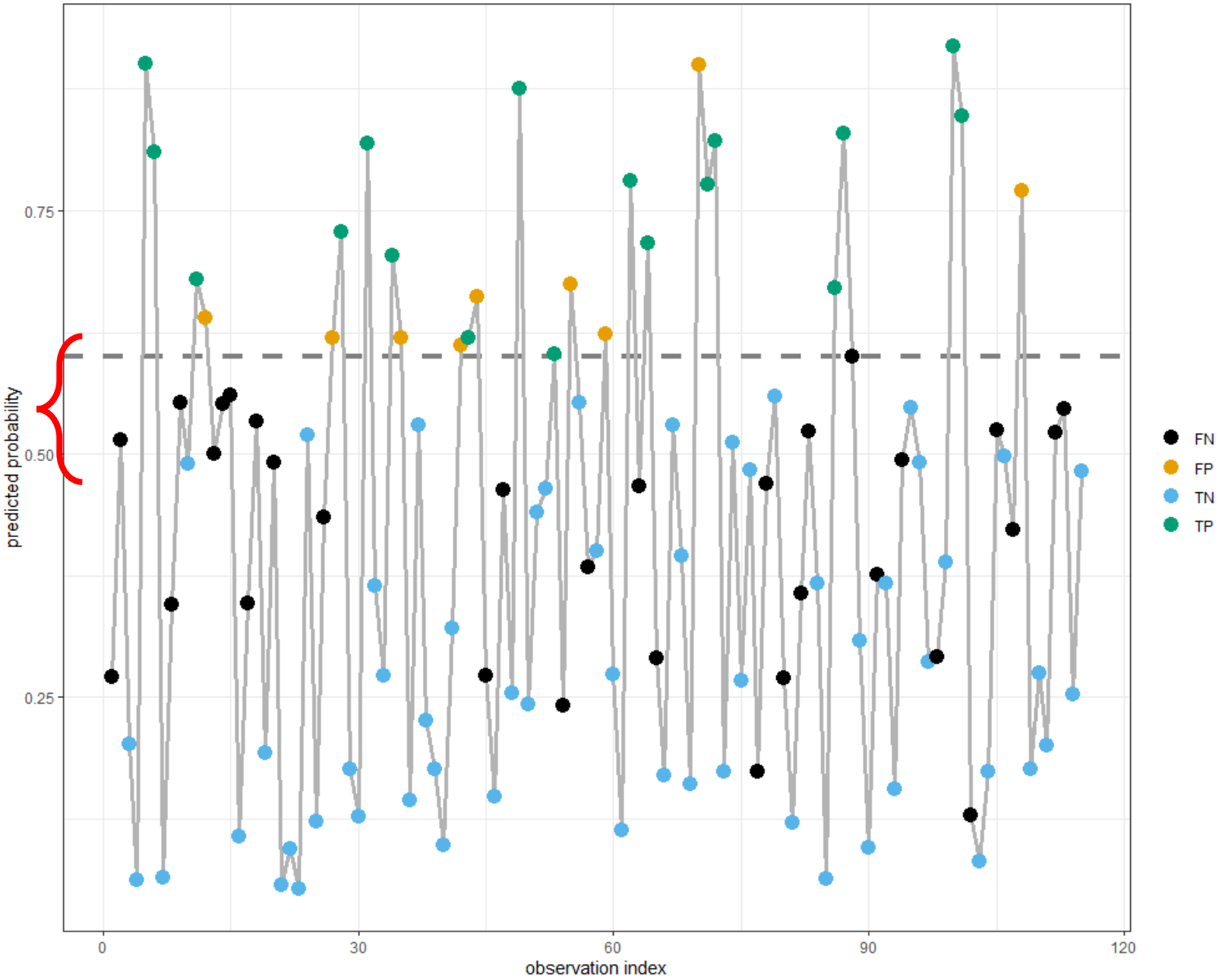
We had TP:27 with threshold=0.5



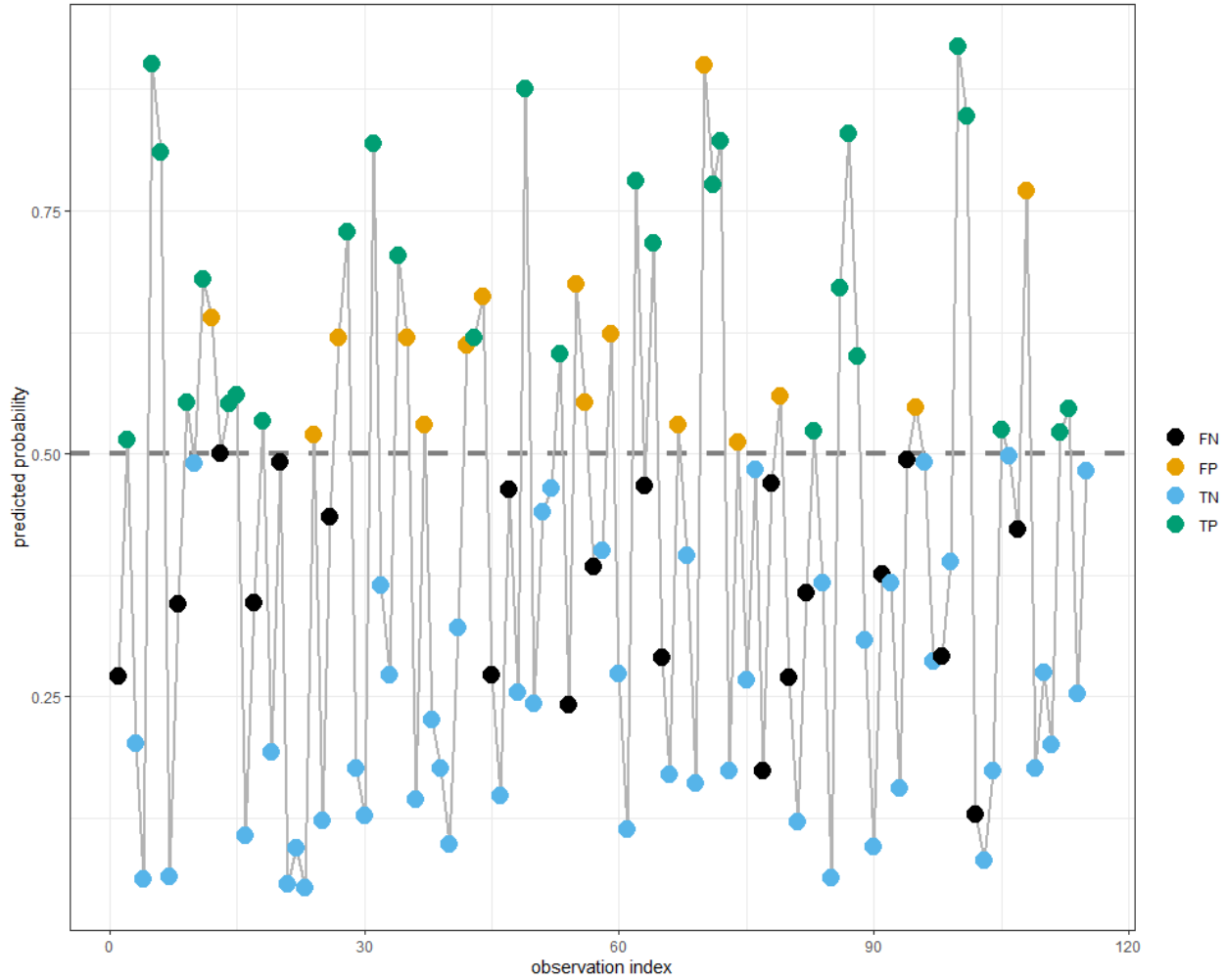
We had TN:51 with threshold=0.5

Increase threshold from 0.5 to 0.6

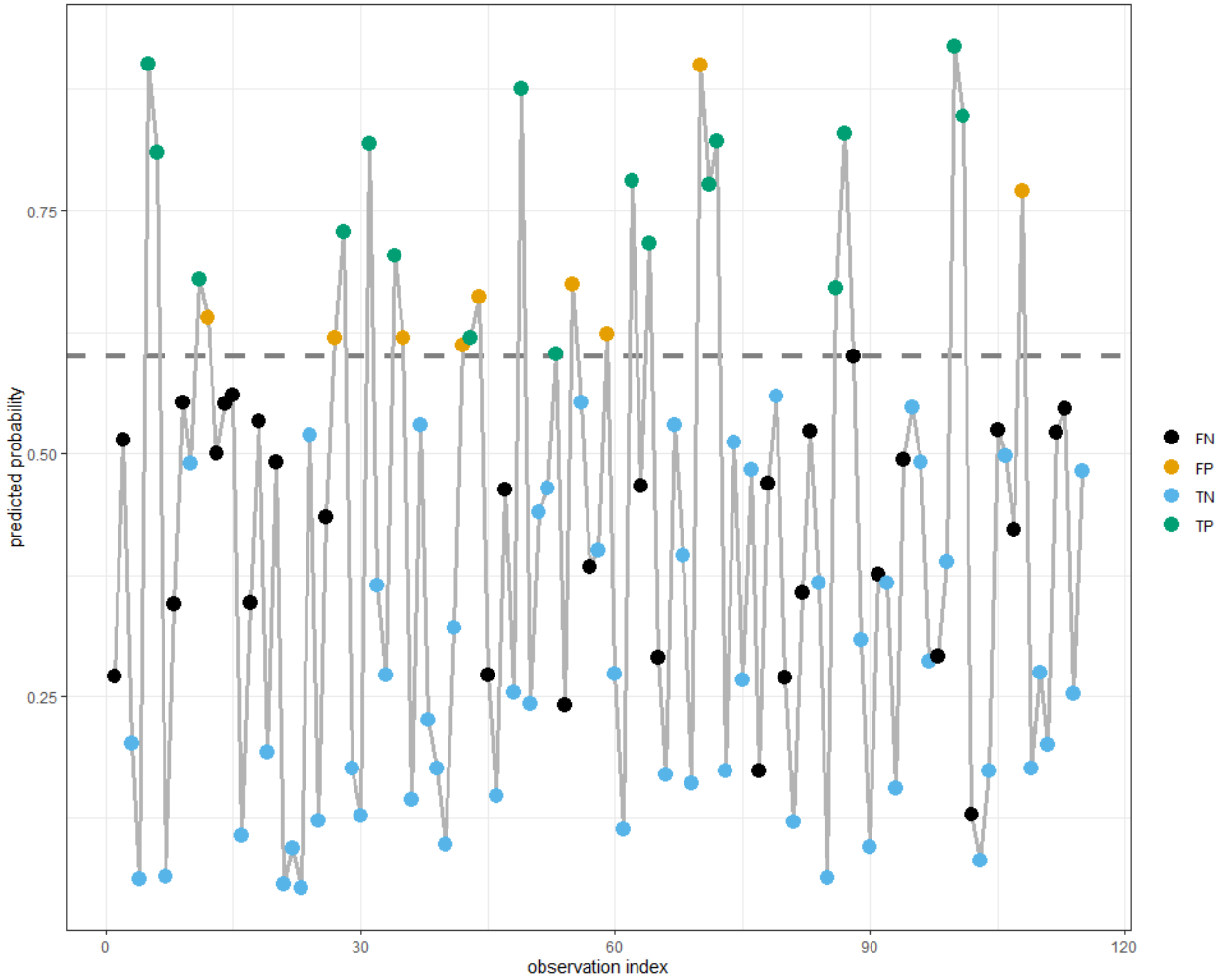
Observations between predicted probability of 0.5 and 0.6 changed elements in the confusion matrix!!



Show side-by-side to make it easier to see the changes



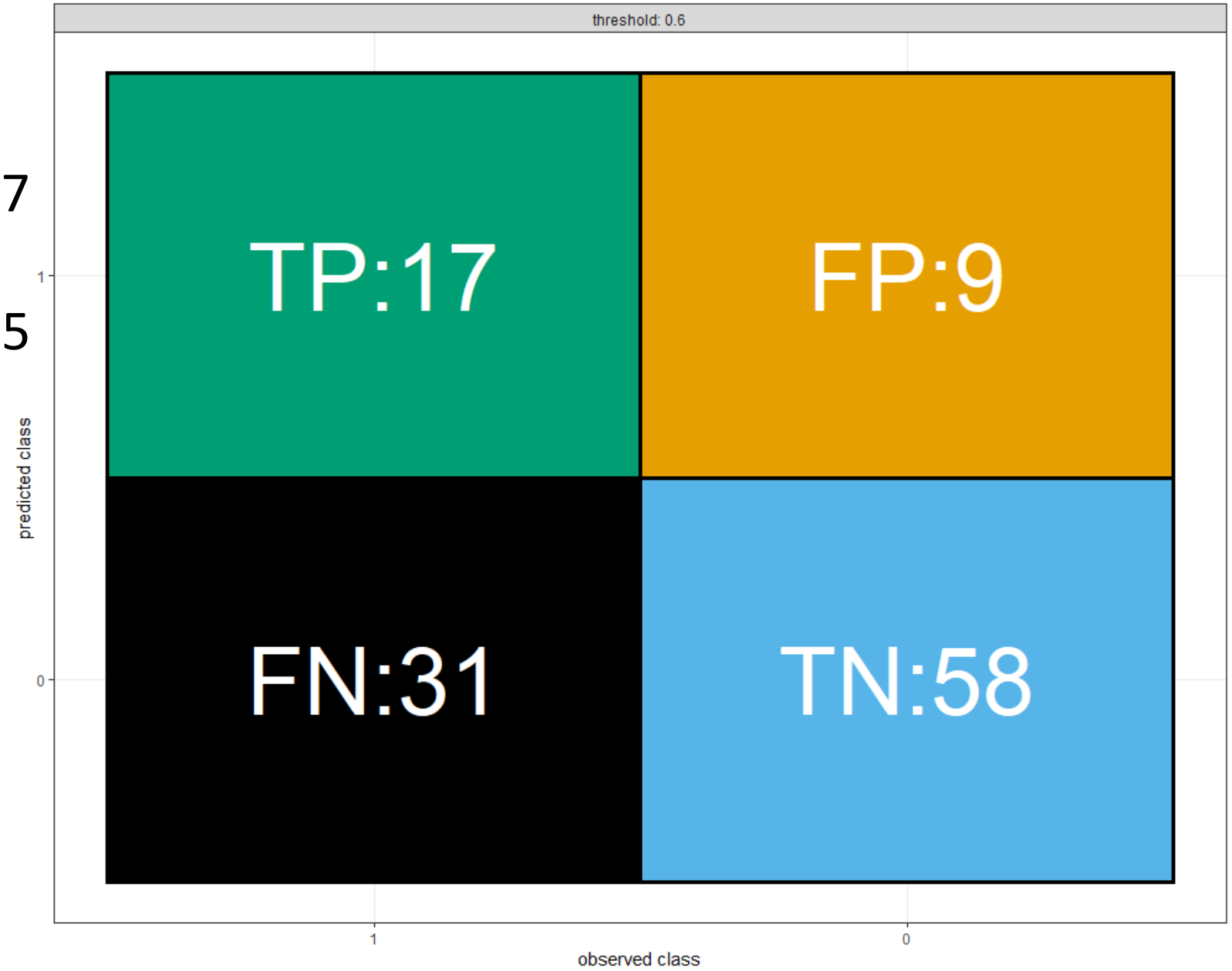
Threshold = 0.5



Threshold = 0.6

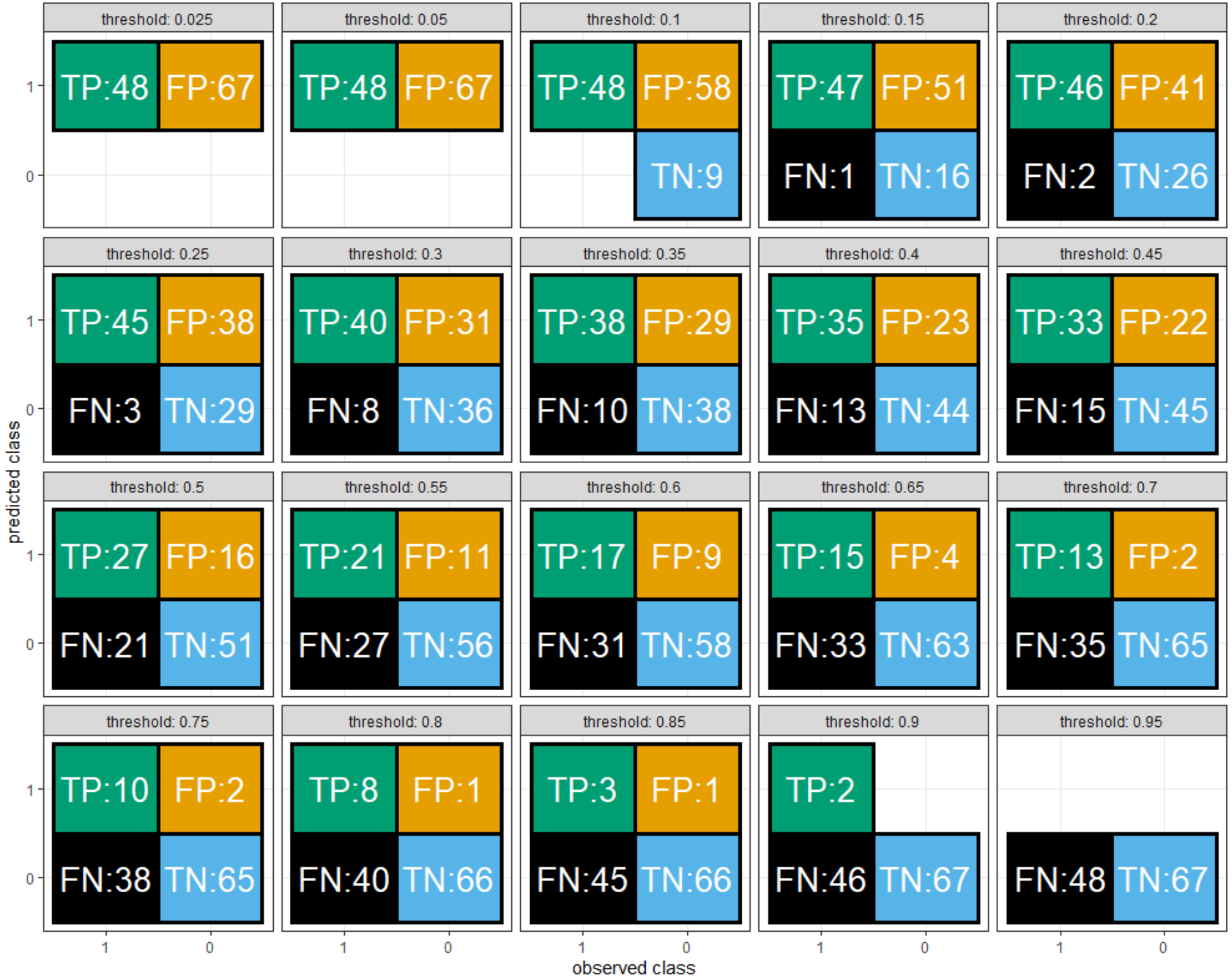
MORE TRUE-NEGATIVES compared to the 0.5 threshold!

We had TP:27 with threshold=0.5

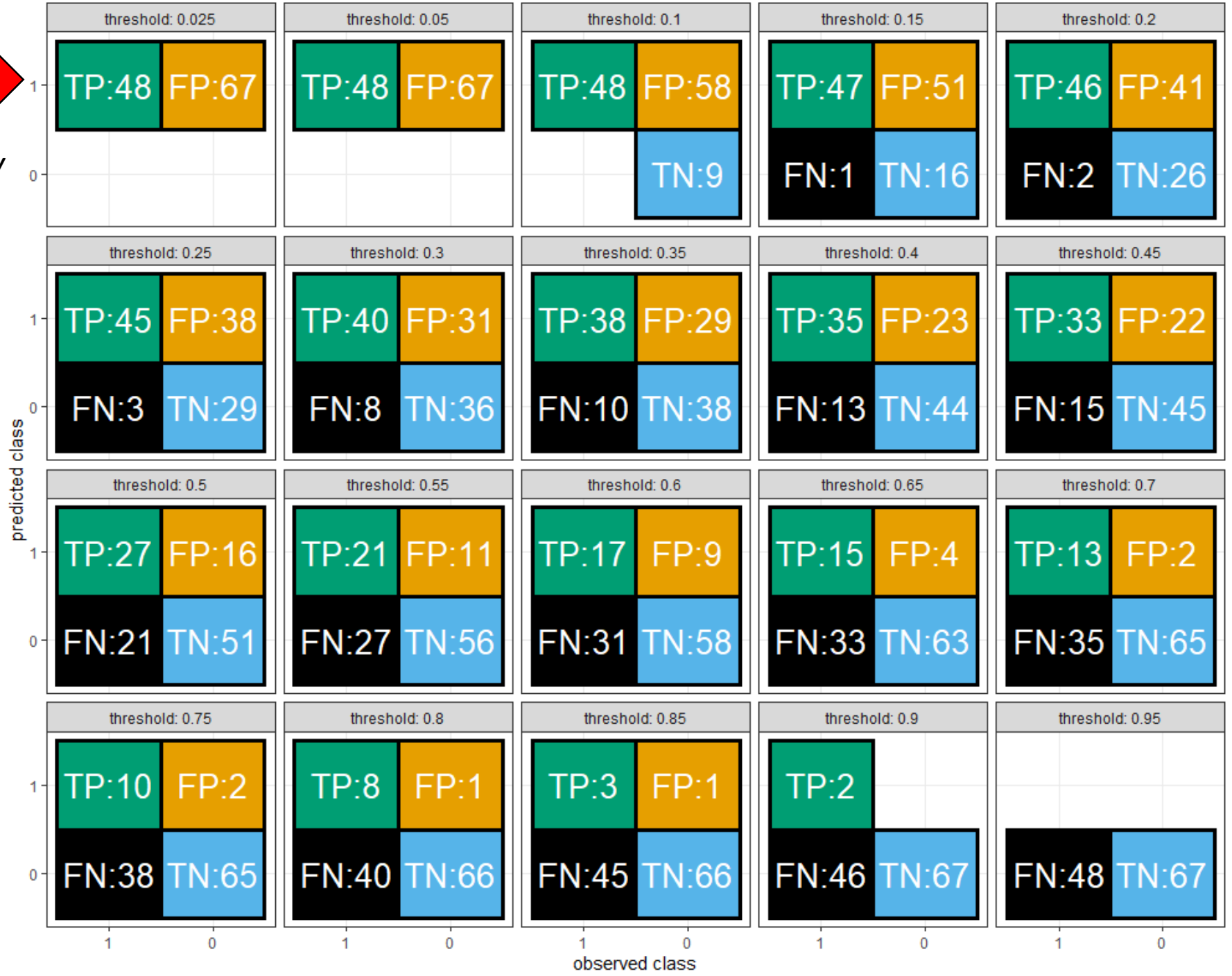


We had TN:51 with threshold=0.5

Repeat these steps for threshold values from near 0 to near 1.


threshold: 0.025
TP:48 FP:67
threshold: 0.05
TP:48 FP:67
threshold: 0.1
TP:48 FP:58
TN:9
threshold: 0.15
TP:47 FP:51
FN:1 TN:16
threshold: 0.2
TP:46 FP:41
FN:2 TN:26
threshold: 0.25
TP:45 FP:38
FN:3 TN:29
threshold: 0.3
TP:40 FP:31
FN:8 TN:36
threshold: 0.35
TP:38 FP:29
FN:10 TN:38
threshold: 0.4
TP:35 FP:23
FN:13 TN:44
threshold: 0.45
TP:33 FP:22
FN:15 TN:45
threshold: 0.5
TP:27 FP:16
FN:21 TN:51
threshold: 0.55
TP:21 FP:11
FN:27 TN:56
threshold: 0.6
TP:17 FP:9
FN:31 TN:58
threshold: 0.65
TP:15 FP:4
FN:33 TN:63
threshold: 0.7
TP:13 FP:2
FN:35 TN:65
threshold: 0.75
TP:10 FP:2
FN:38 TN:65
threshold: 0.8
TP:8 FP:1
FN:40 TN:66
threshold: 0.85
TP:3 FP:1
FN:45 TN:66
threshold: 0.9
TP:2
FN:46 TN:67
threshold: 0.95
FN:48 TN:67
predicted class
1
0
observed class
1
0

Very low thresholds, ONLY classifying the EVENT

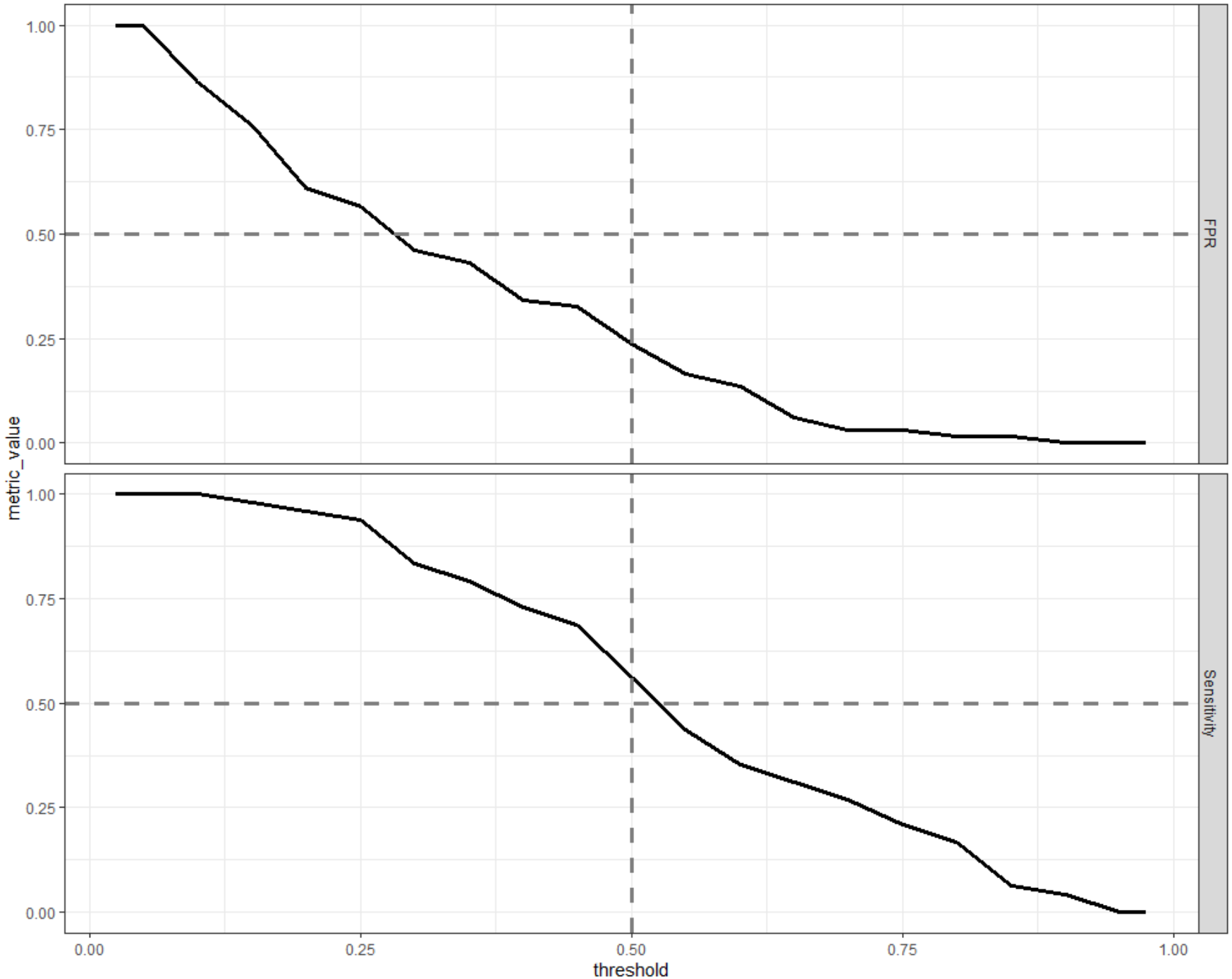


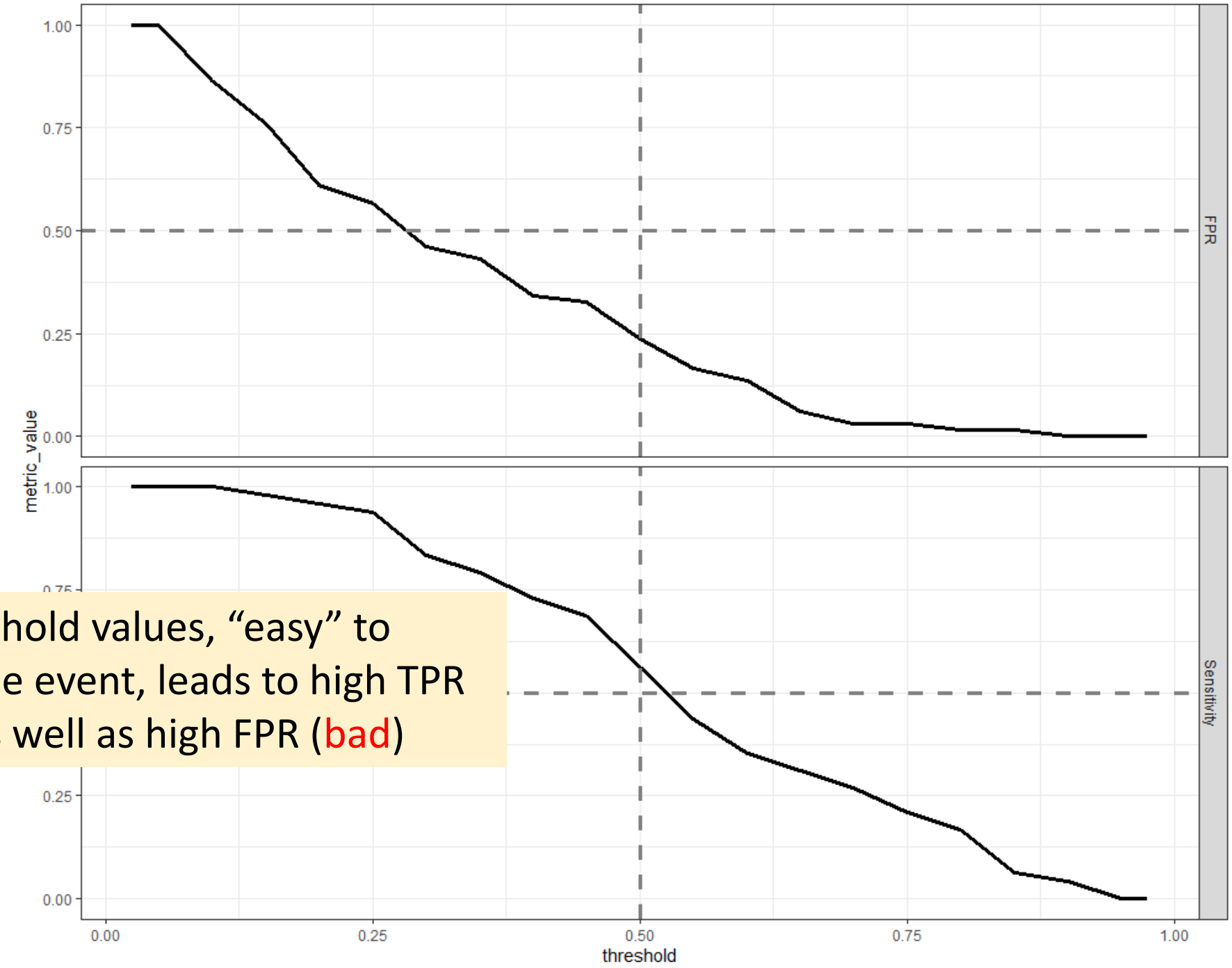
Very high thresholds, ONLY classifying the NON-EVENT

That’s nice…but that’s a lot of figures to look at!

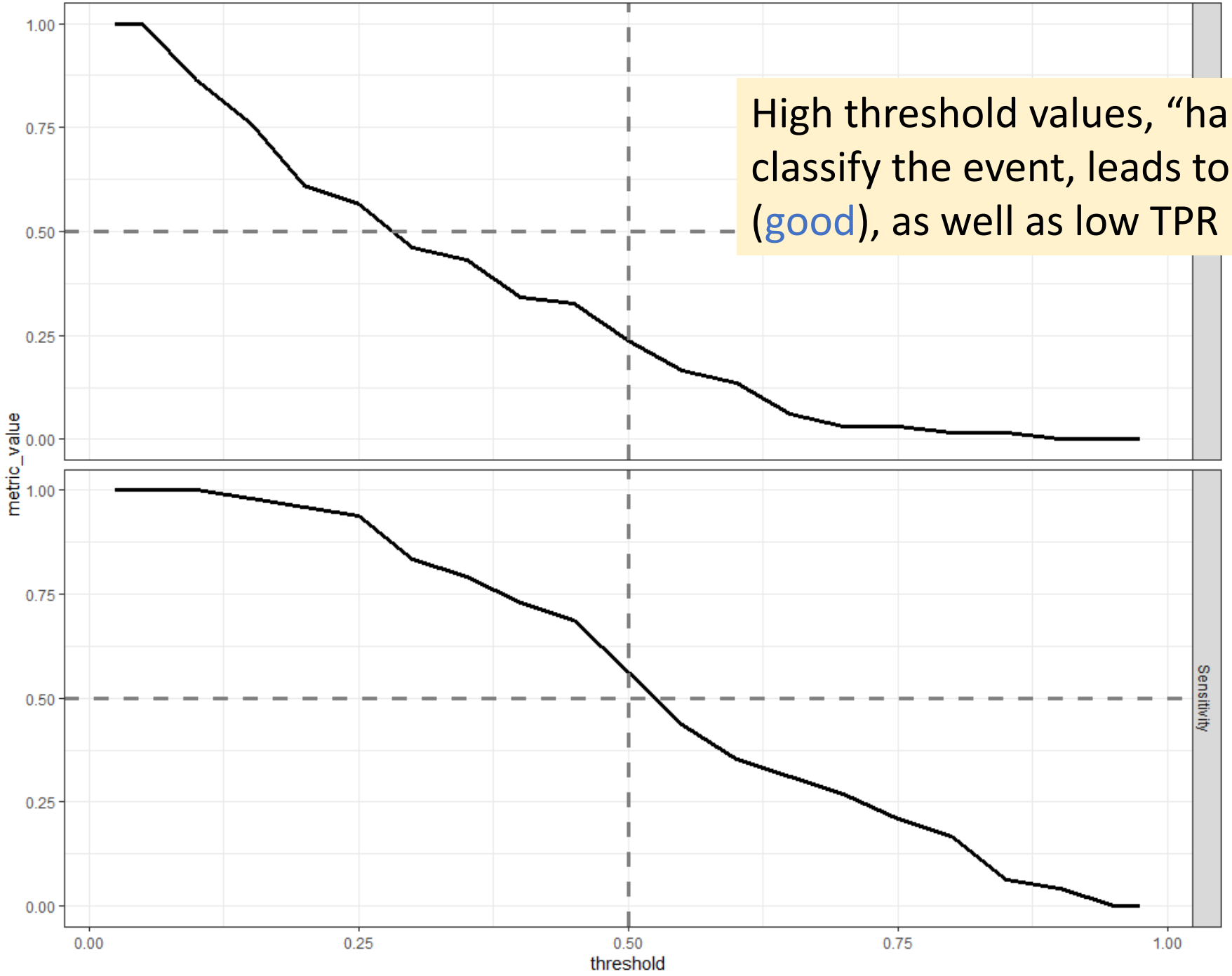
- Let’s consider how the Sensitivity (or True Positive Rate, TPR) and the False Positive Rate (FPR) metrics vary as the threshold changes.

- Instead of looking at counts, we will look at fractions (or rates).

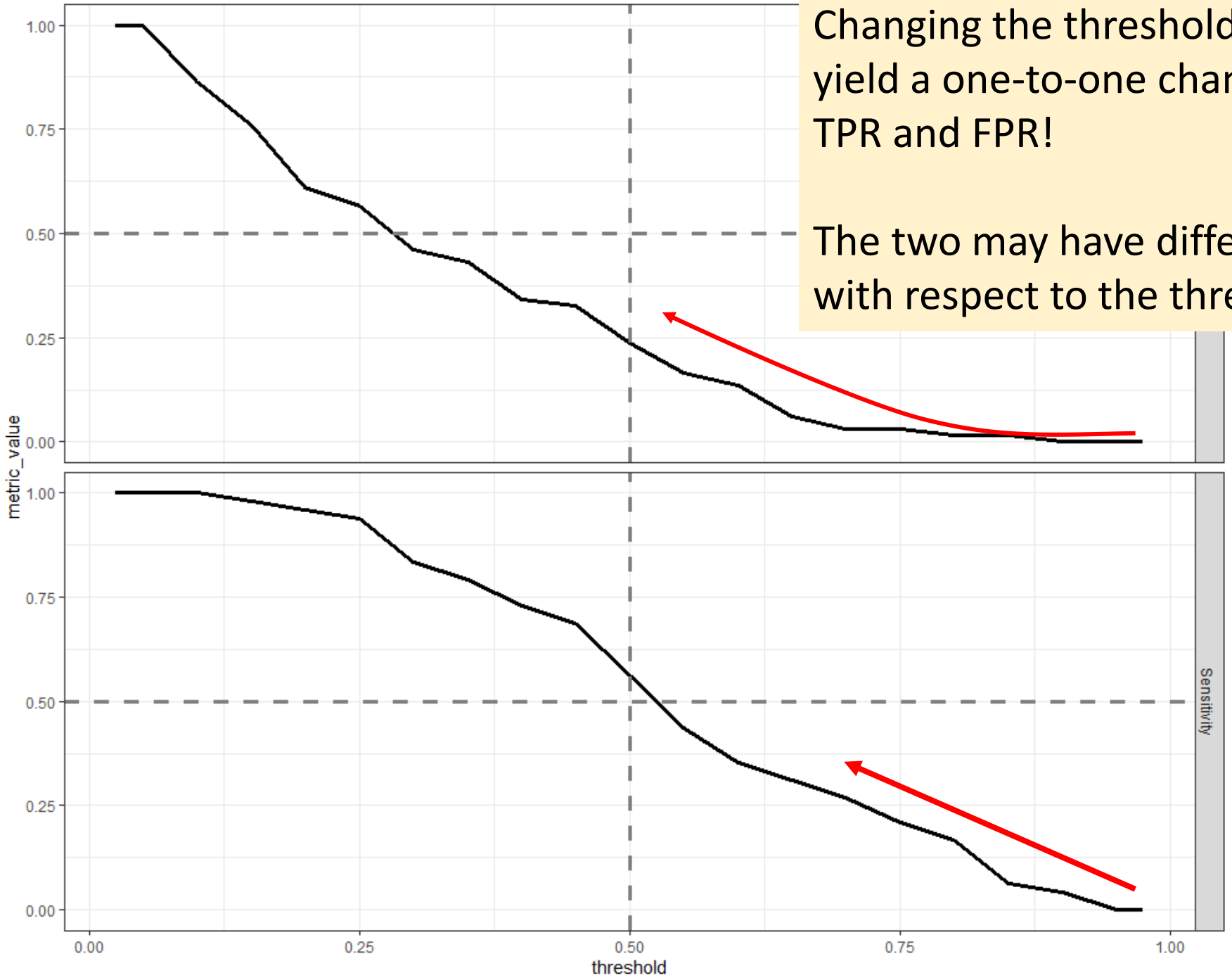

1.00
0.75
0.50
0.25
0.00
FPR
metric_value
1.00
0.75
0.50
0.25
0.00
Sensitivity
0.00
0.25
0.50
0.75
1.00
threshold



Low threshold values, “easy” to classify the event, leads to high TPR (good), as well as high FPR (bad)

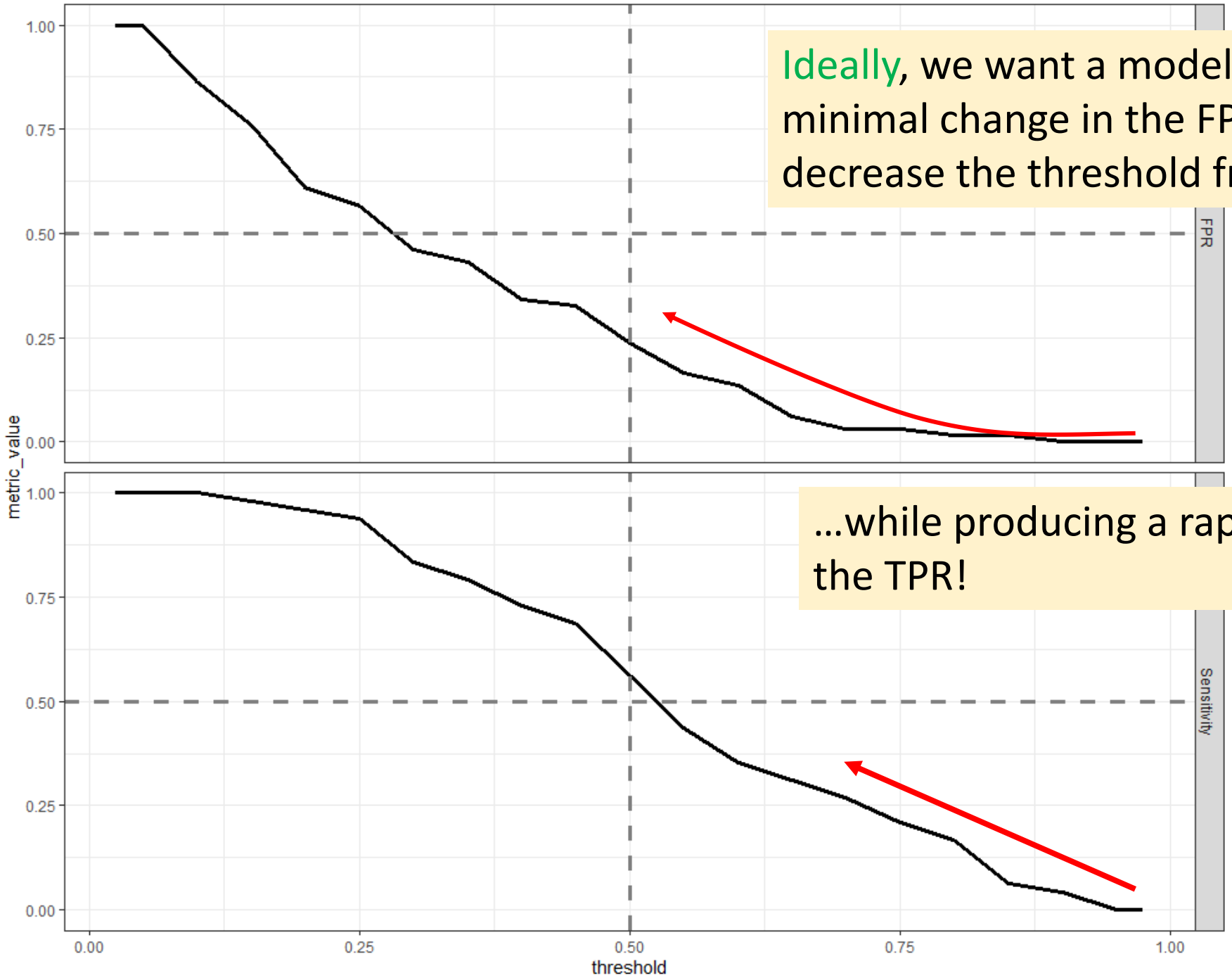


High threshold values, “hard” to classify the event, leads to low FPR (good), as well as low TPR (bad)



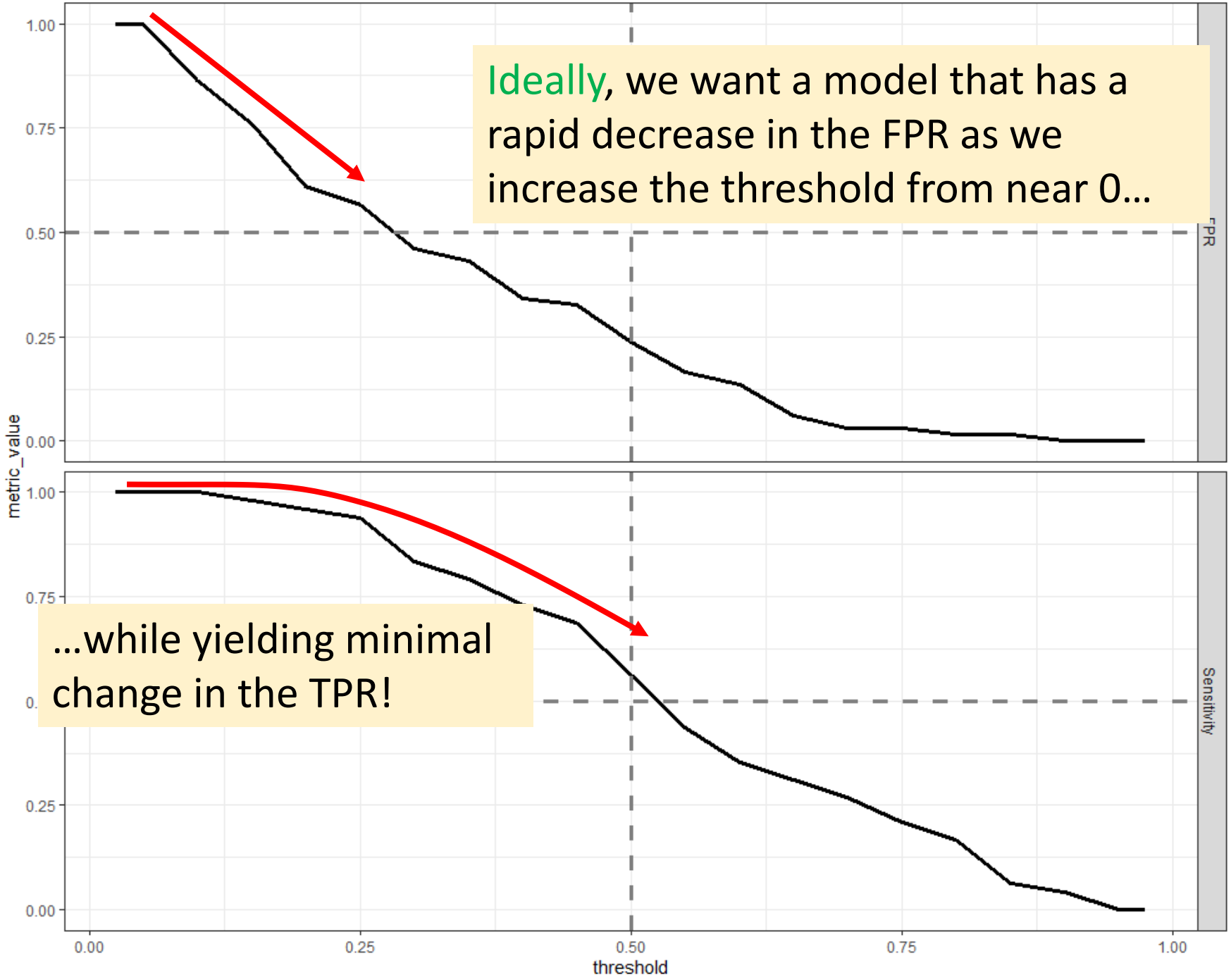
Changing the threshold may NOT yield a one-to-one change between TPR and FPR!

The two may have different slopes with respect to the threshold!



Ideally, we want a model that has minimal change in the FPR as we decrease the threshold from near 1…

…while producing a rapid increase in the TPR!


Ideally, we want a model that has a rapid decrease in the FPR as we increase the threshold from near 0…
…while yielding minimal change in the TPR!
metric_value
1.00
0.75
0.50
0.25
0.00
FPR
Sensitivity
0.00
0.25
0.50
0.75
1.00
threshold

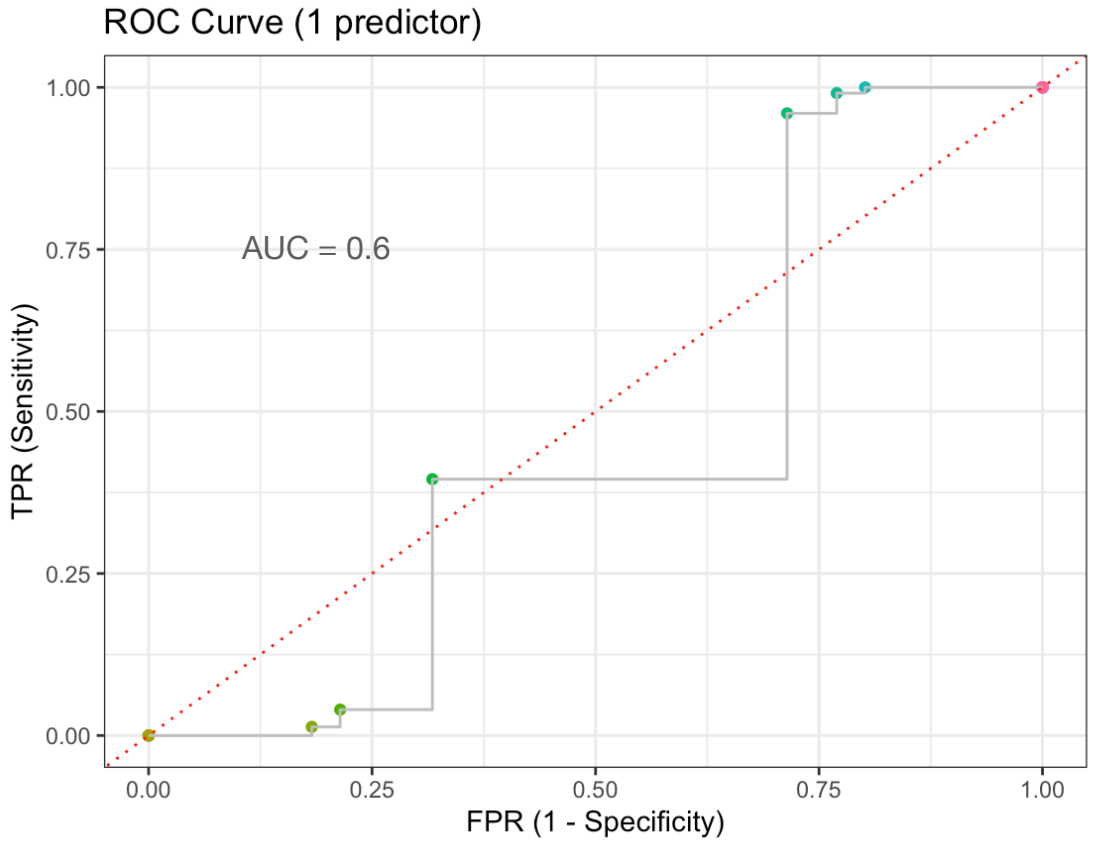
In practice, we combine the two previous graphs into a single visualization

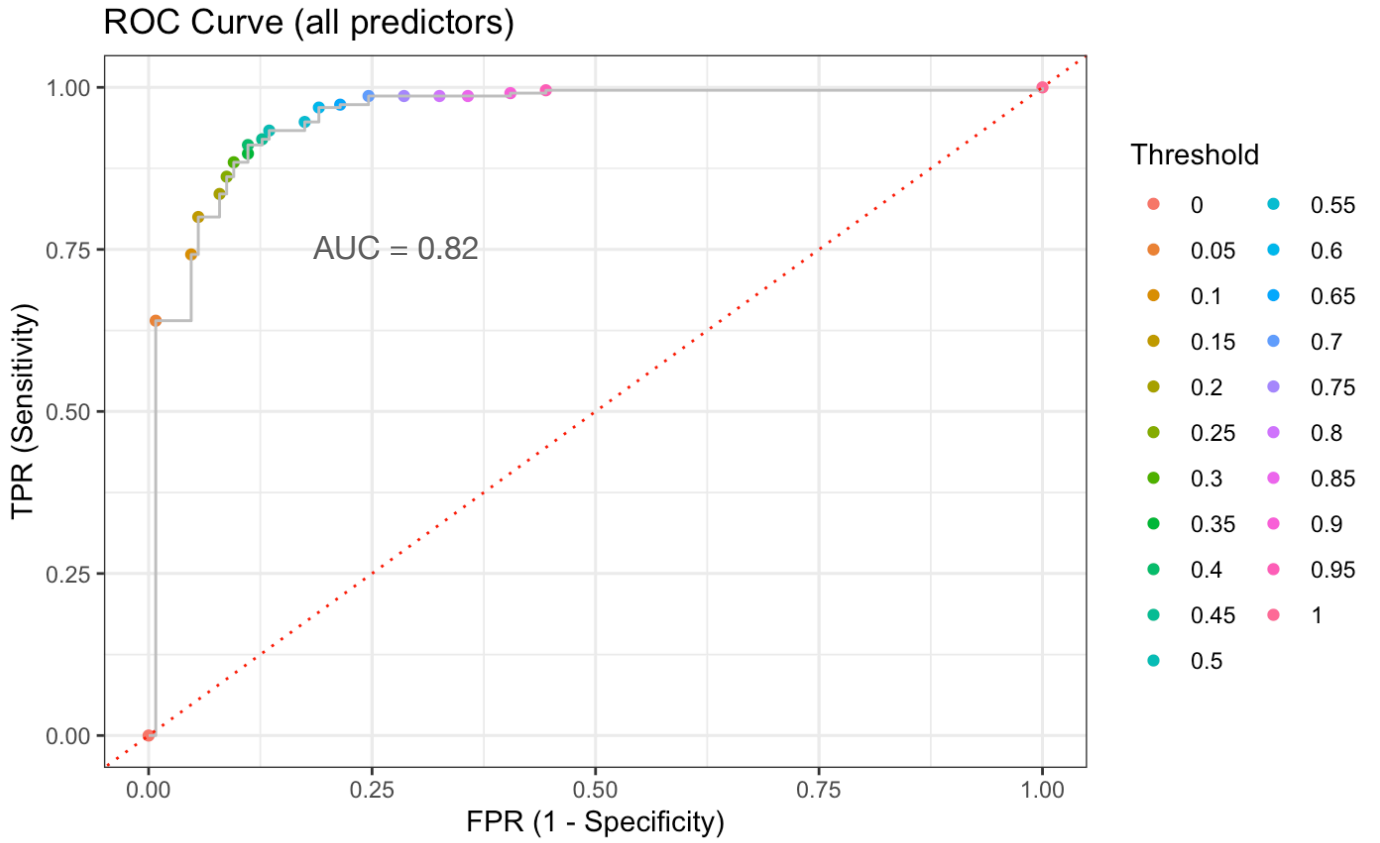
- The **ROC curve** is visualized as the **Sensitivity** (TPR) with respect to **1 – Specificity** (FPR).

- The **hidden third dimension** is the value of the threshold which controls the TPR and FPR.

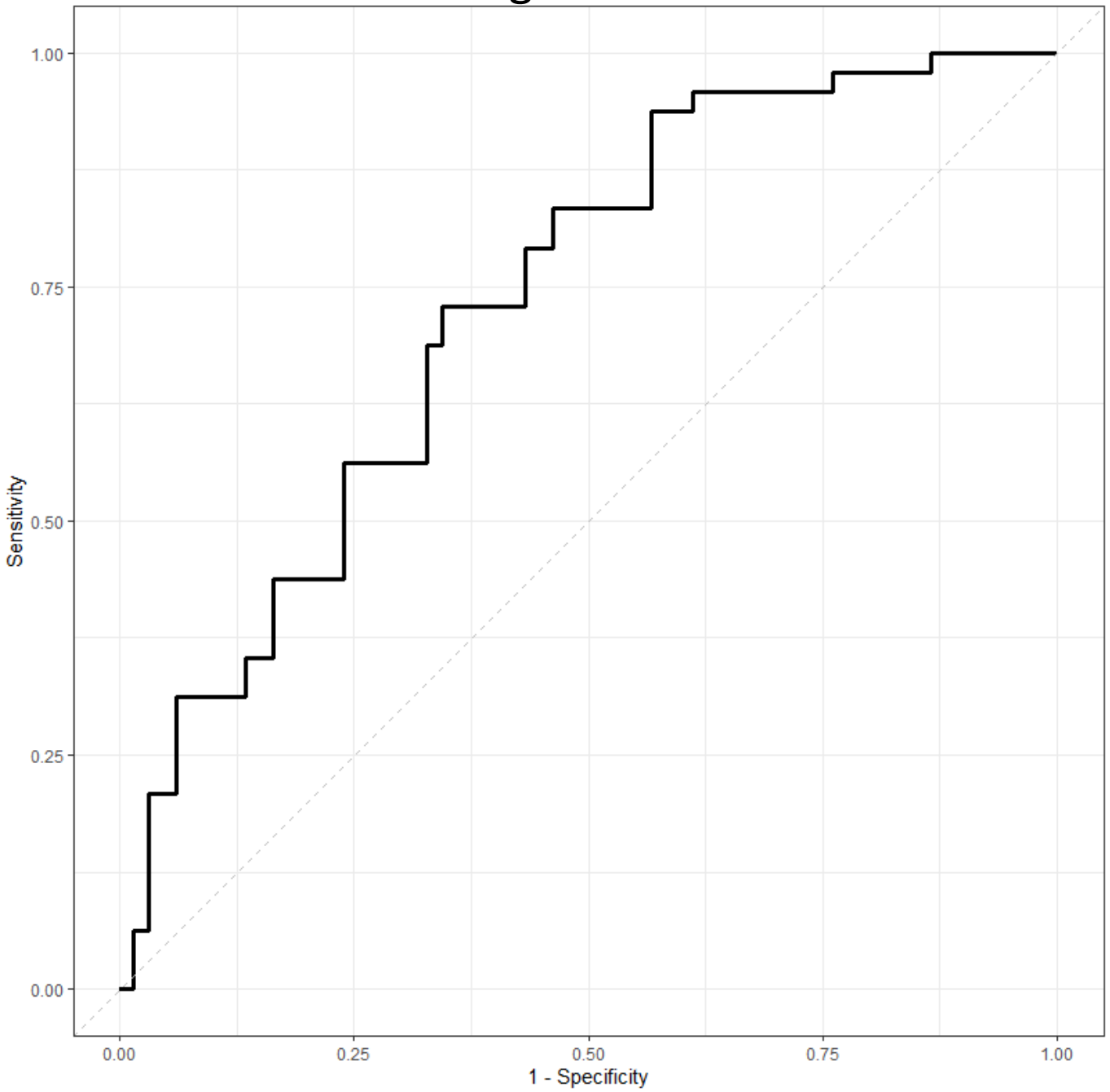
- The ROC curve tells how the model performs as we change the threshold. **The model cannot deviate from the plotted trajectory**!

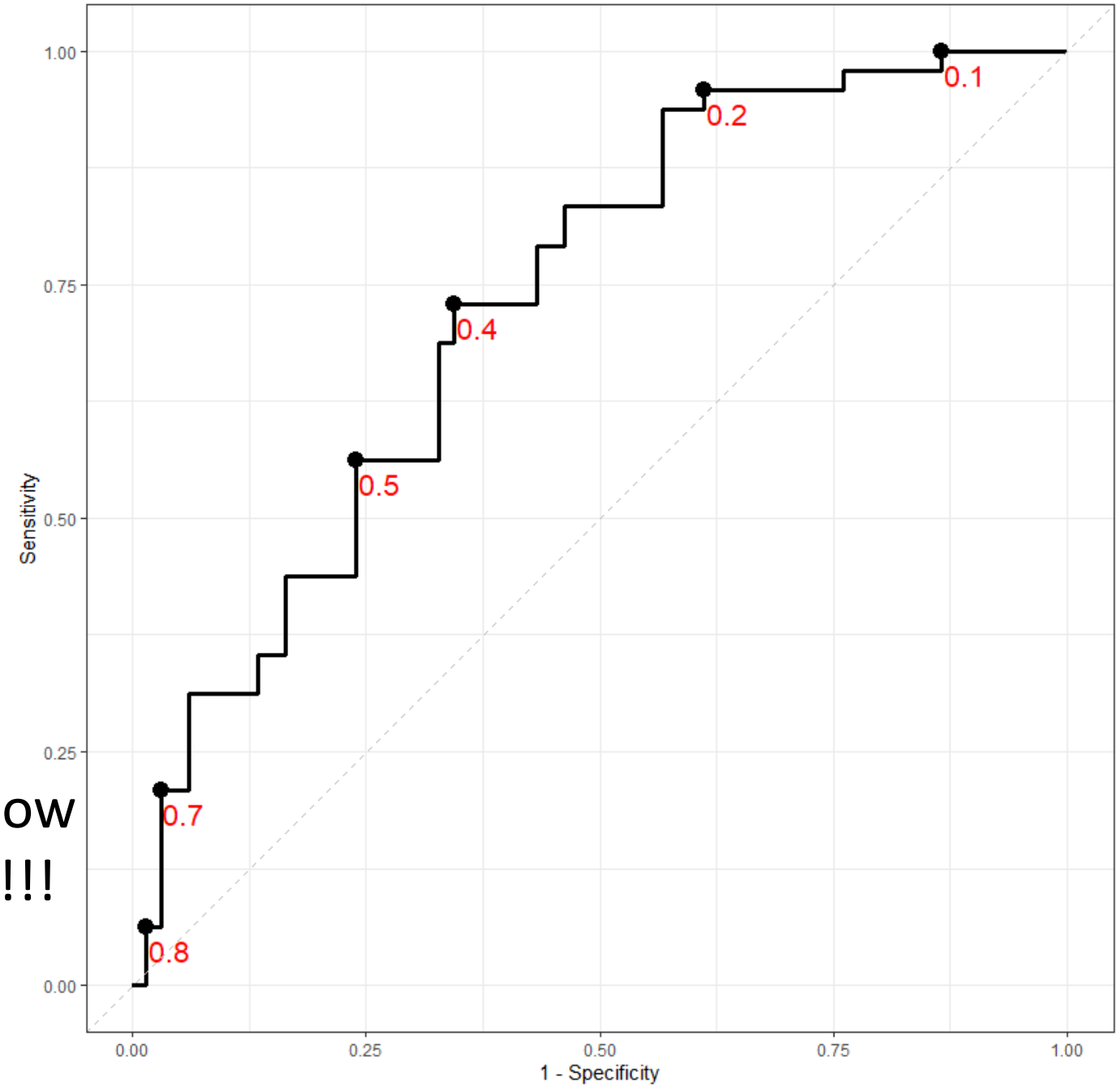
How can we tell if an ROC curve is good?





Training set ROC curve

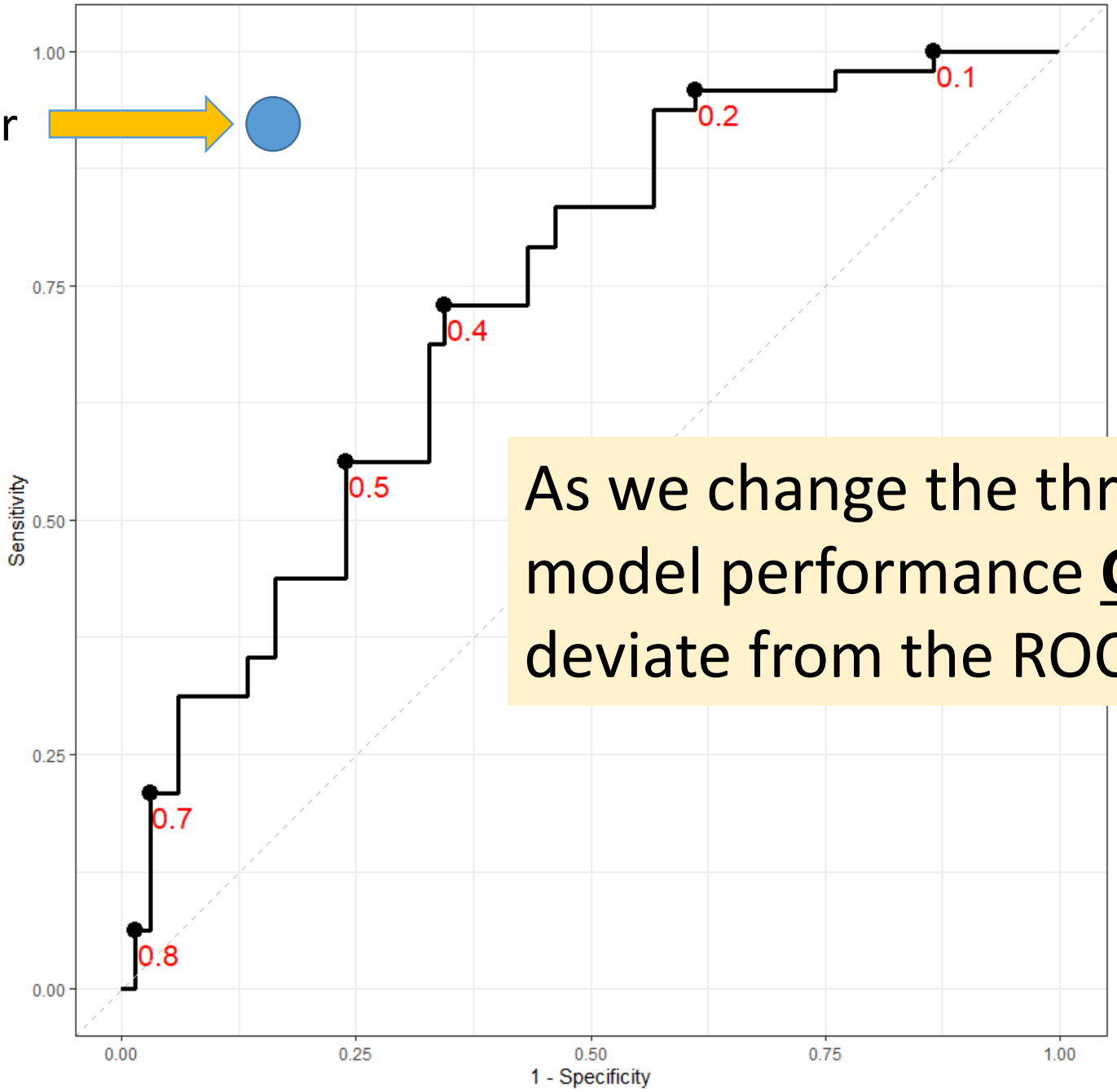




Low thresholds high TPR and high FPR!!!

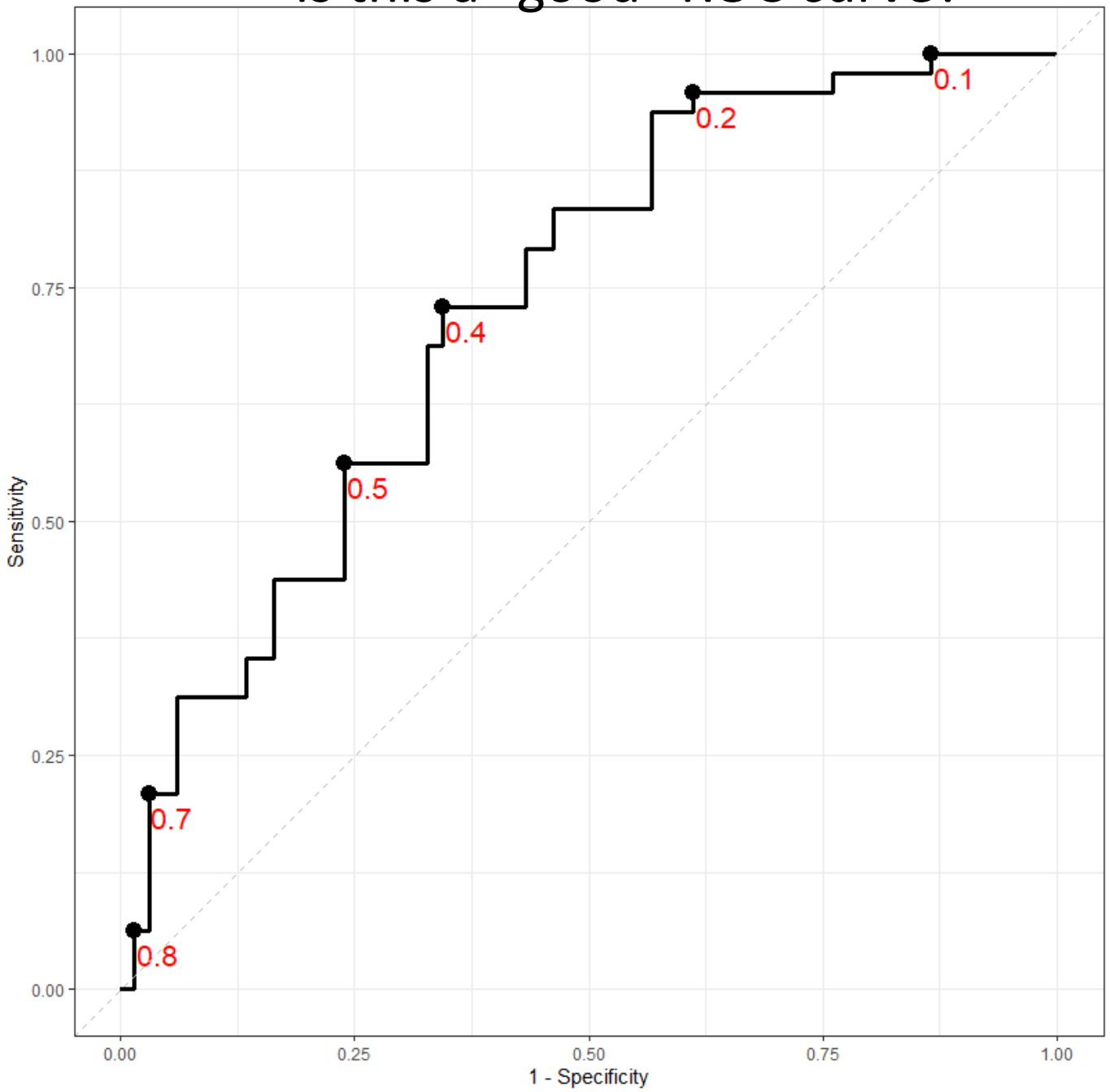
High thresholds low TPR and low FPR!!!

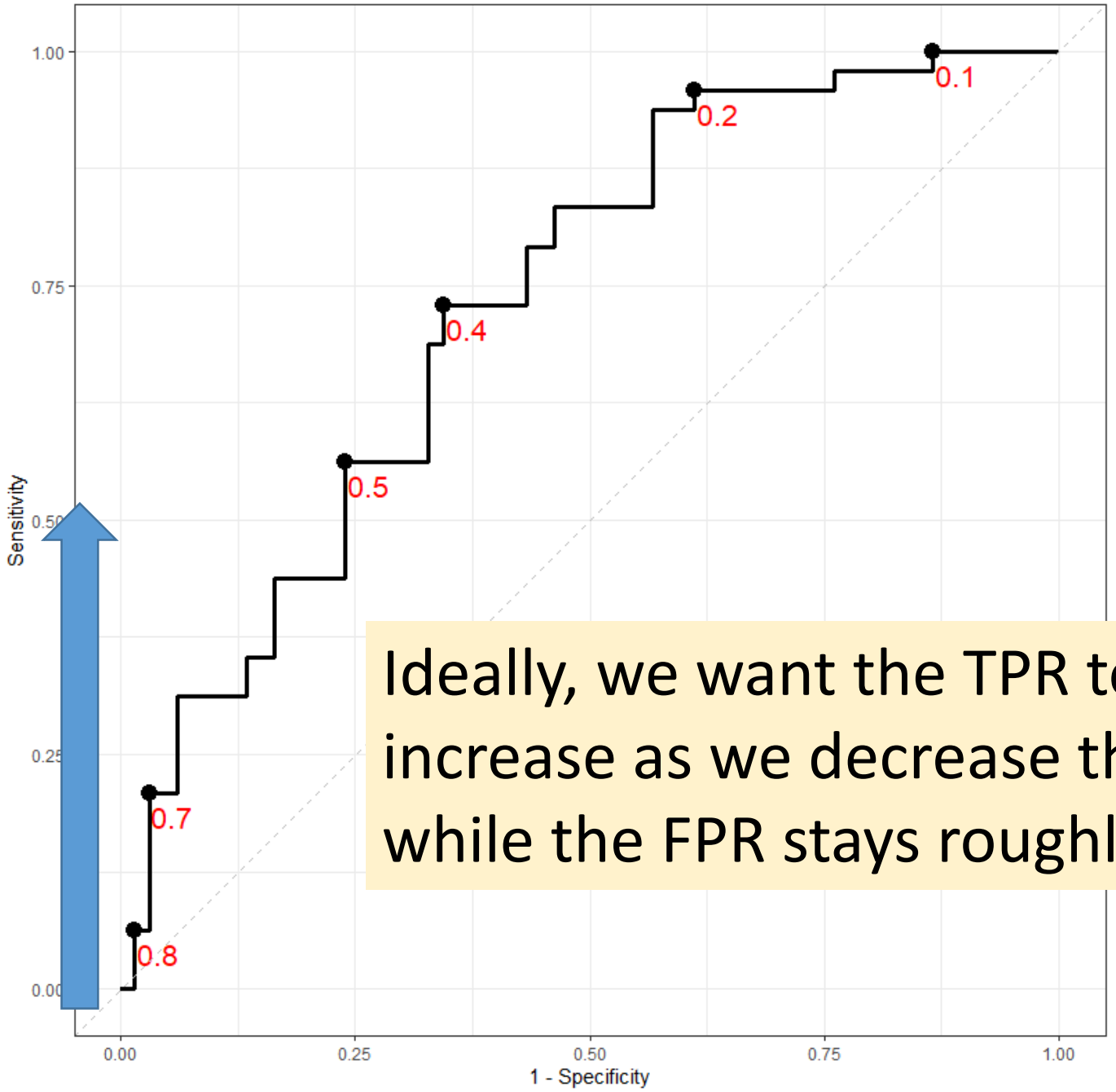
This condition can never occur for this model!



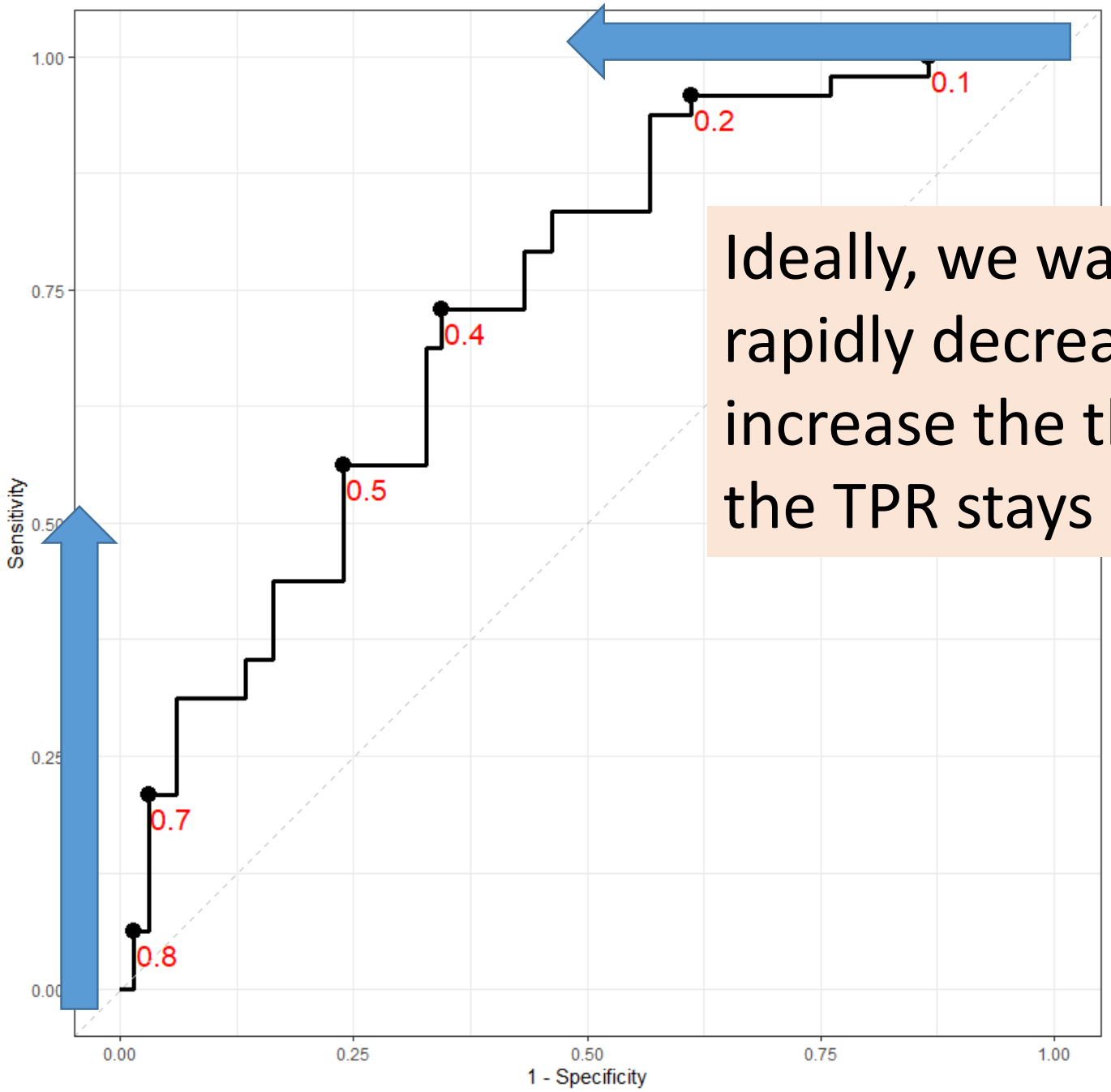
As we change the threshold, the model performance **CANNOT** deviate from the ROC curve!

Is this a “good” ROC curve?

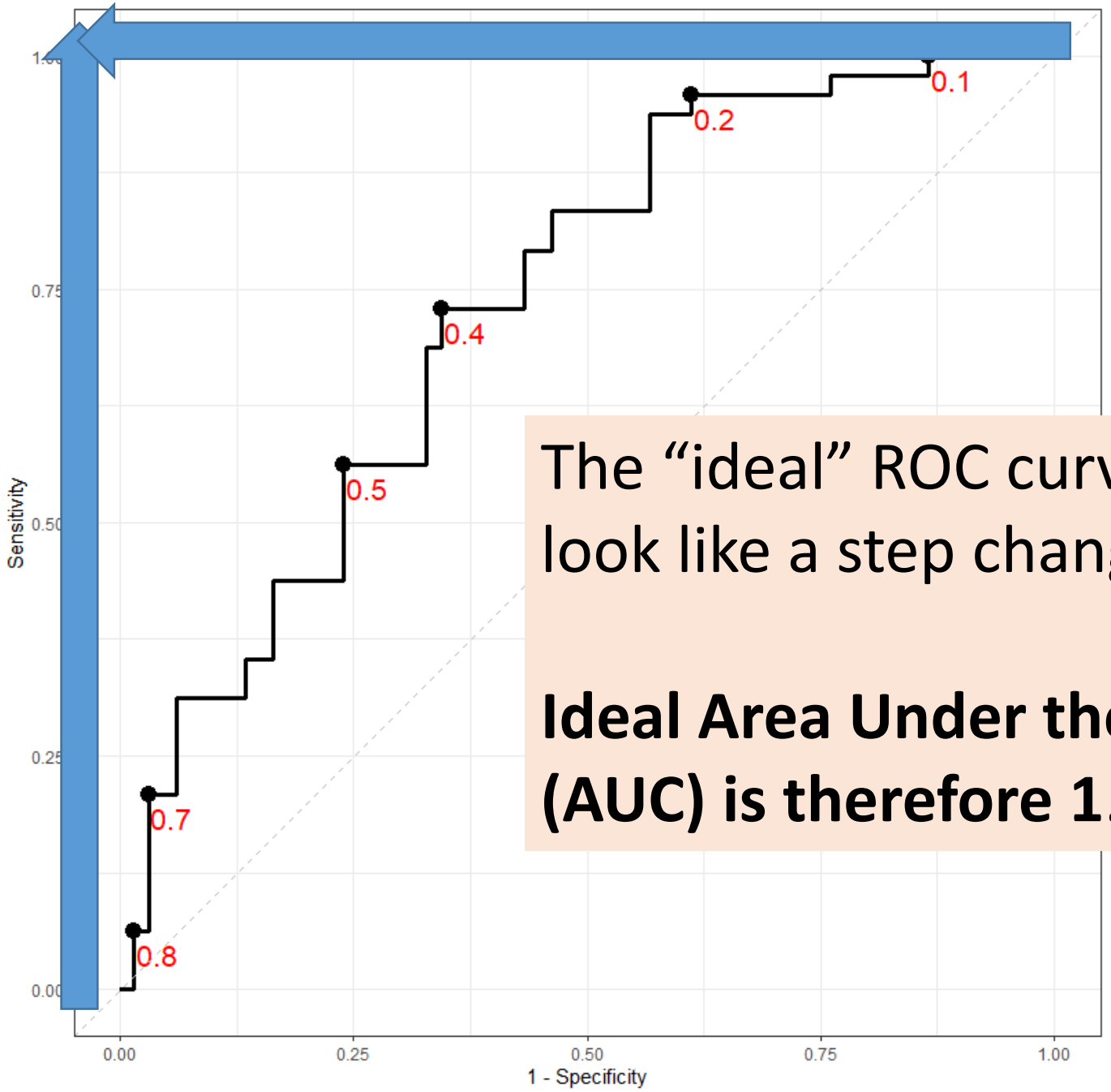




Ideally, we want the TPR to rapidly increase as we decrease the threshold, while the FPR stays roughly constant



Ideally, we want the FPR to rapidly decrease as we increase the threshold, while the TPR stays roughly constant

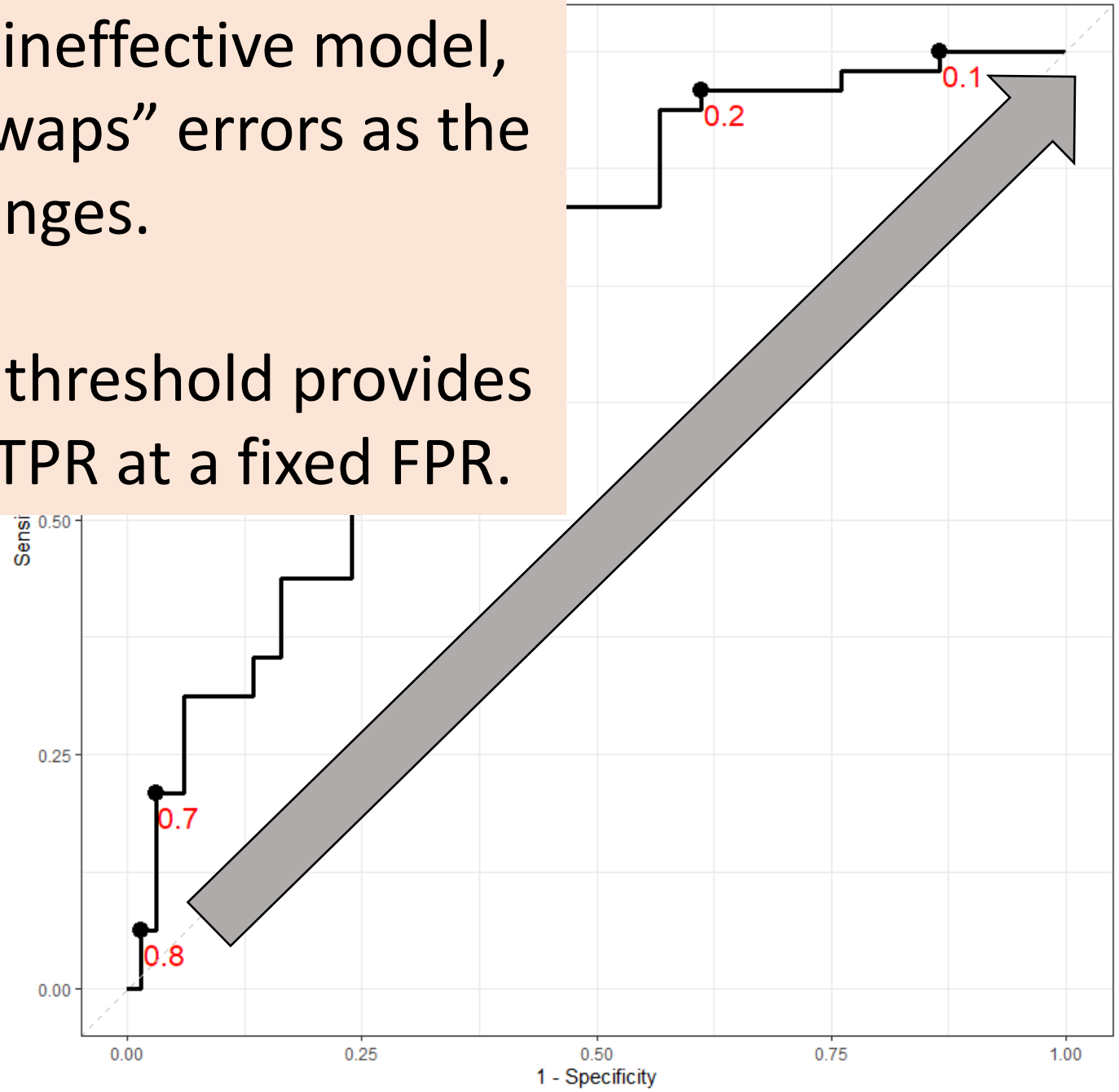


The “ideal” ROC curve would look like a step change!

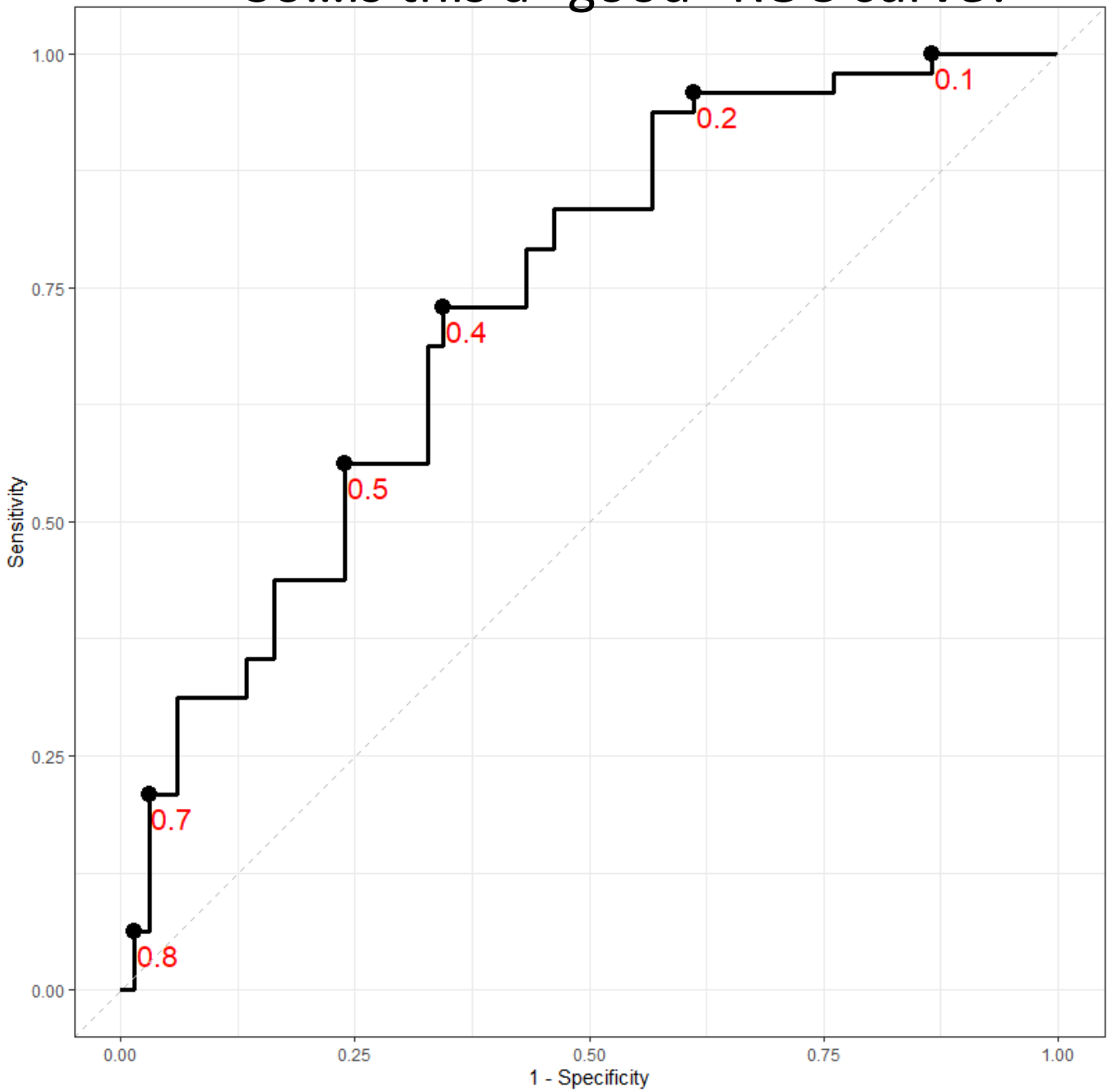
Ideal Area Under the Curve (AUC) is therefore 1.

A completely ineffective model, is one that “swaps” errors as the threshold changes.

Dropping the threshold provides no benefit in TPR at a fixed FPR.

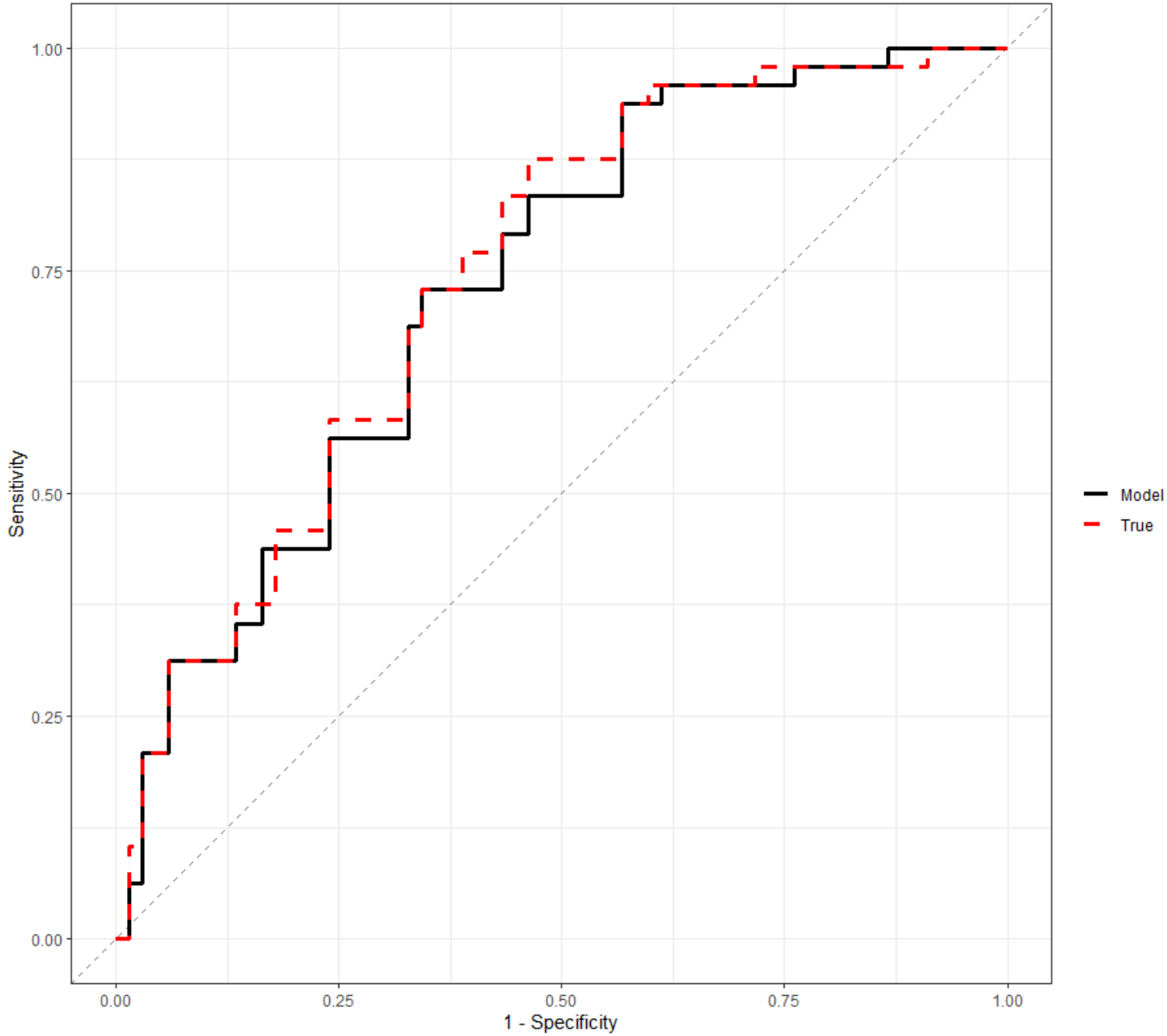


So…is this a “good” ROC curve?



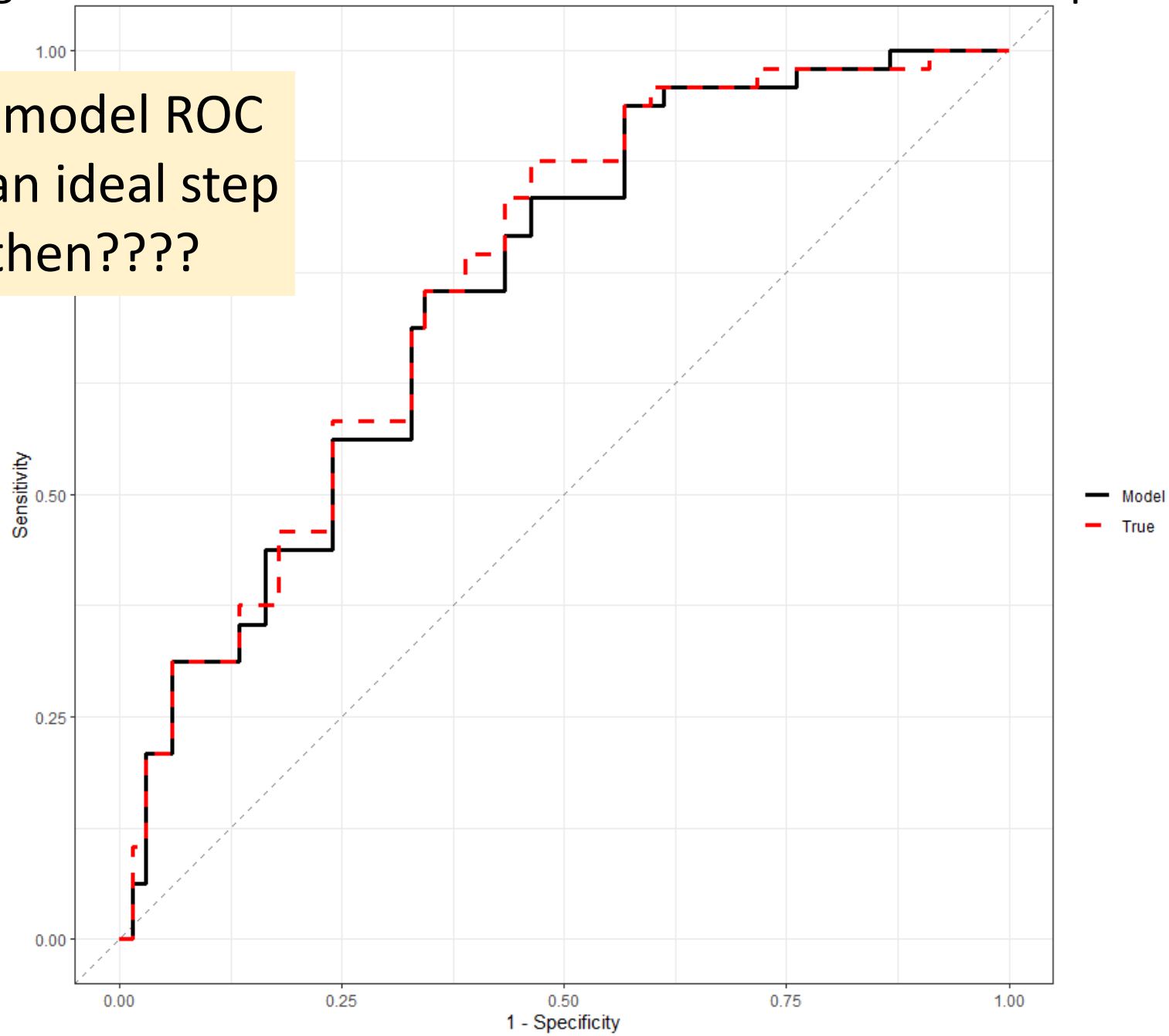
Since this is a toy demo with synthetic data, we know the TRUE event probability

- Allows us to calculate the TRUE confusion matrix for a given threshold.
- Which means we can calculate the TRUE ROC curve.
- Next slide overlays the TRUE ROC curve with the model ROC curve.


1.00
0.75
0.50
0.25
0.00
Sensitivity
0.00
0.25
0.50
0.75
1.00
1 - Specificity
Model
True

The agreement between the TRUTH and the MODEL looks quite good…

Why is the model ROC curve NOT an ideal step change then????



What's going on?

- The ROC curve, as with Accuracy, is interested in a **point-wise** comparison.

- We compare the predicted class to the reference observed class.

AUC Interpretation: which one of these is the “true” crosswalk?

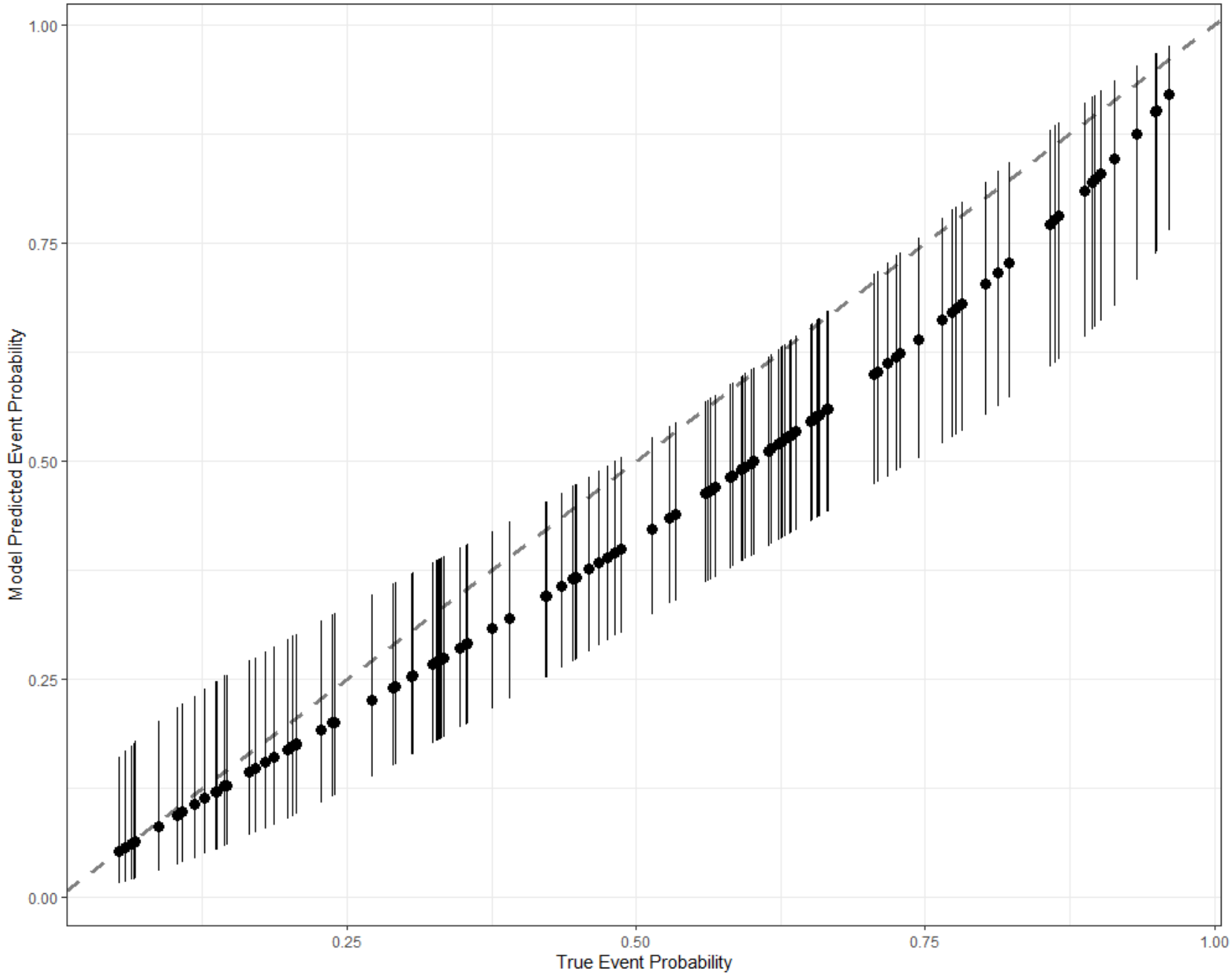
- Given 1 positive and 1 negative sample, what is the probability that the model correctly selects the positive sample?
- AUC = 0.9: 90% chance of selecting the correct image
- AUC = 0.5: 50% chance (random luck for a binary variable)
- Worst we can do is random guessing, thus AUC has a floor of 0.5

Is this wrong to do?

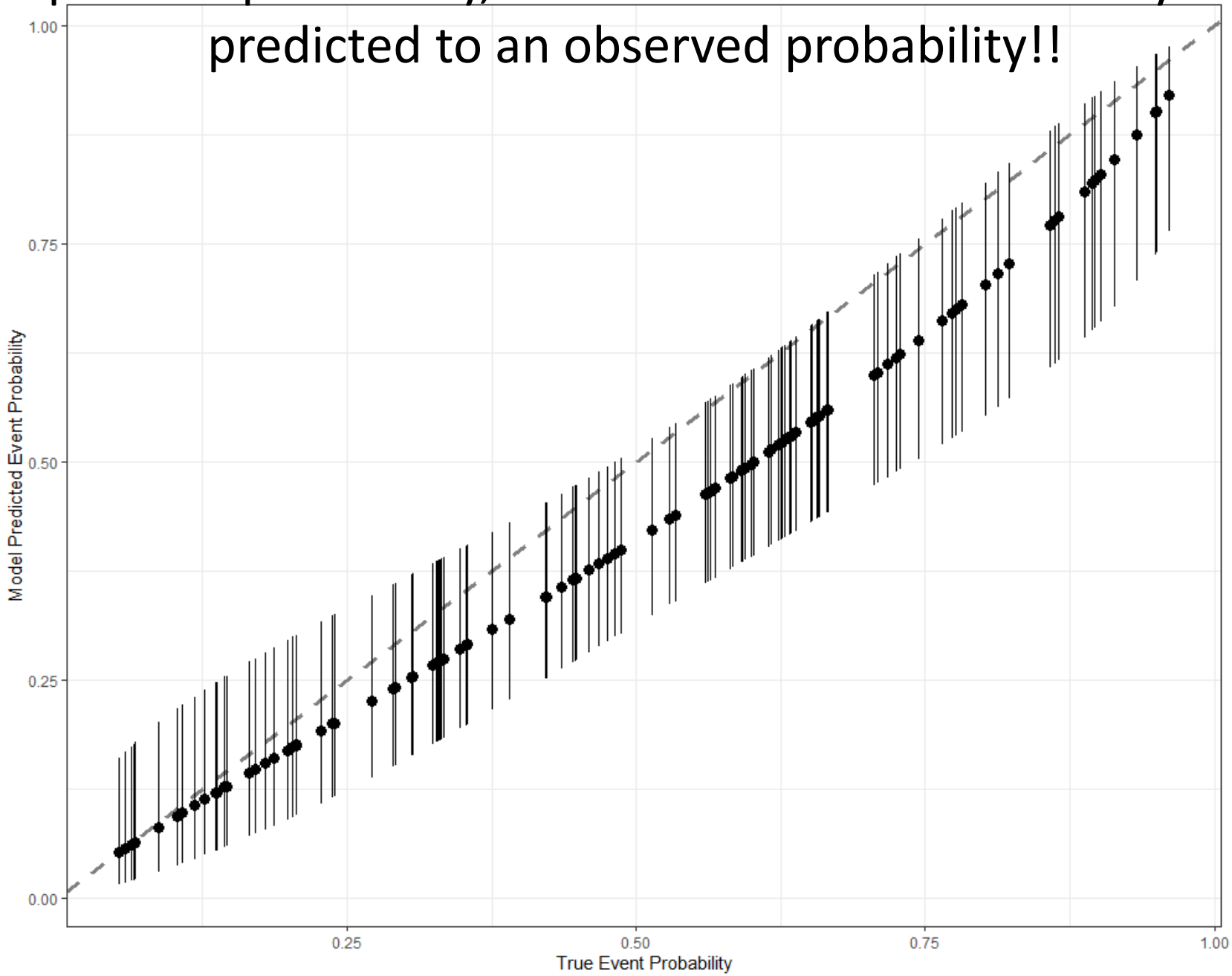
- No…Accuracy, ROC curves, and AUC are important performance metrics to consider when comparing models.

- But the model does not actually predict the class…

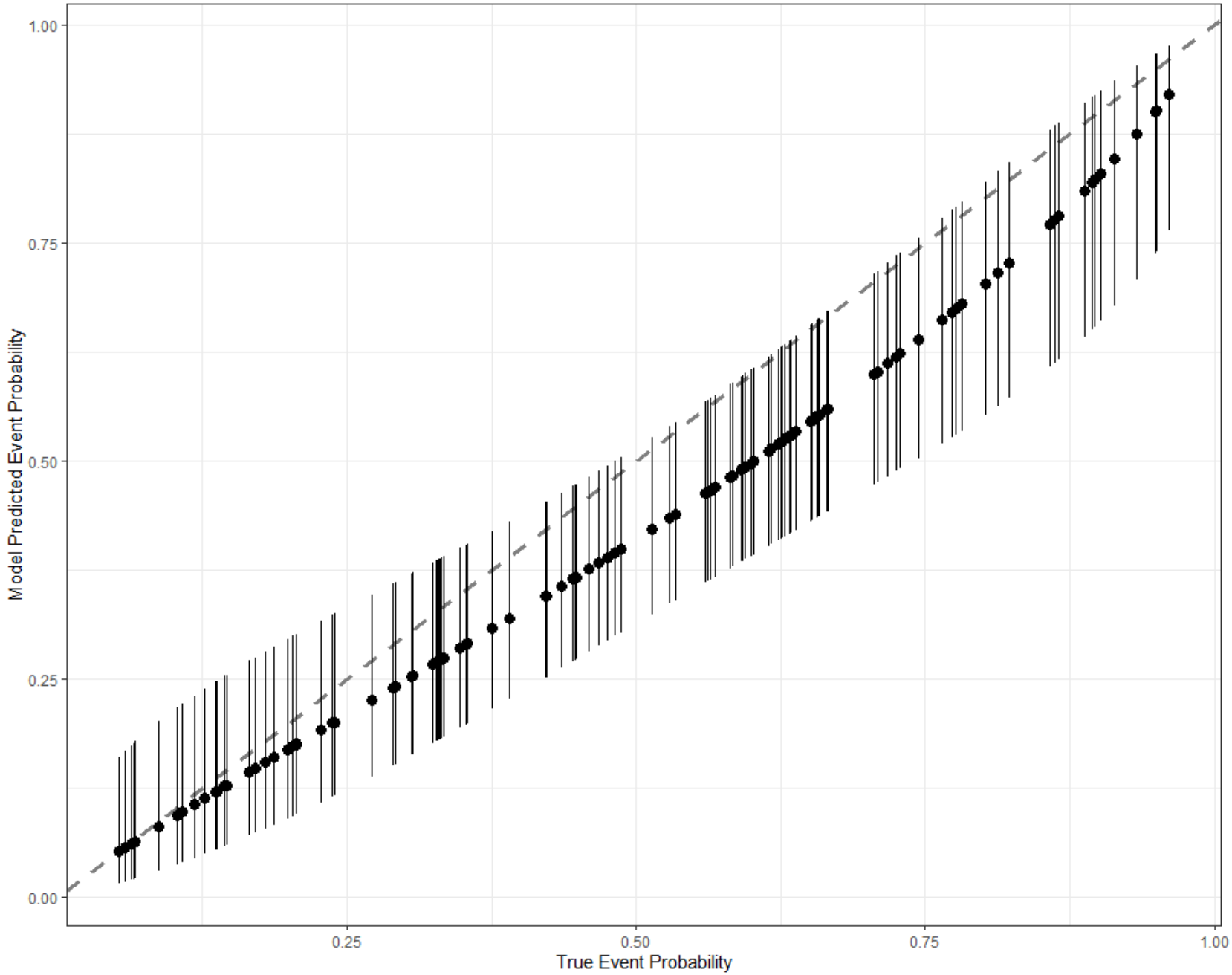
It would be great if we could make a predicted vs observed style figure…in terms of **probability**



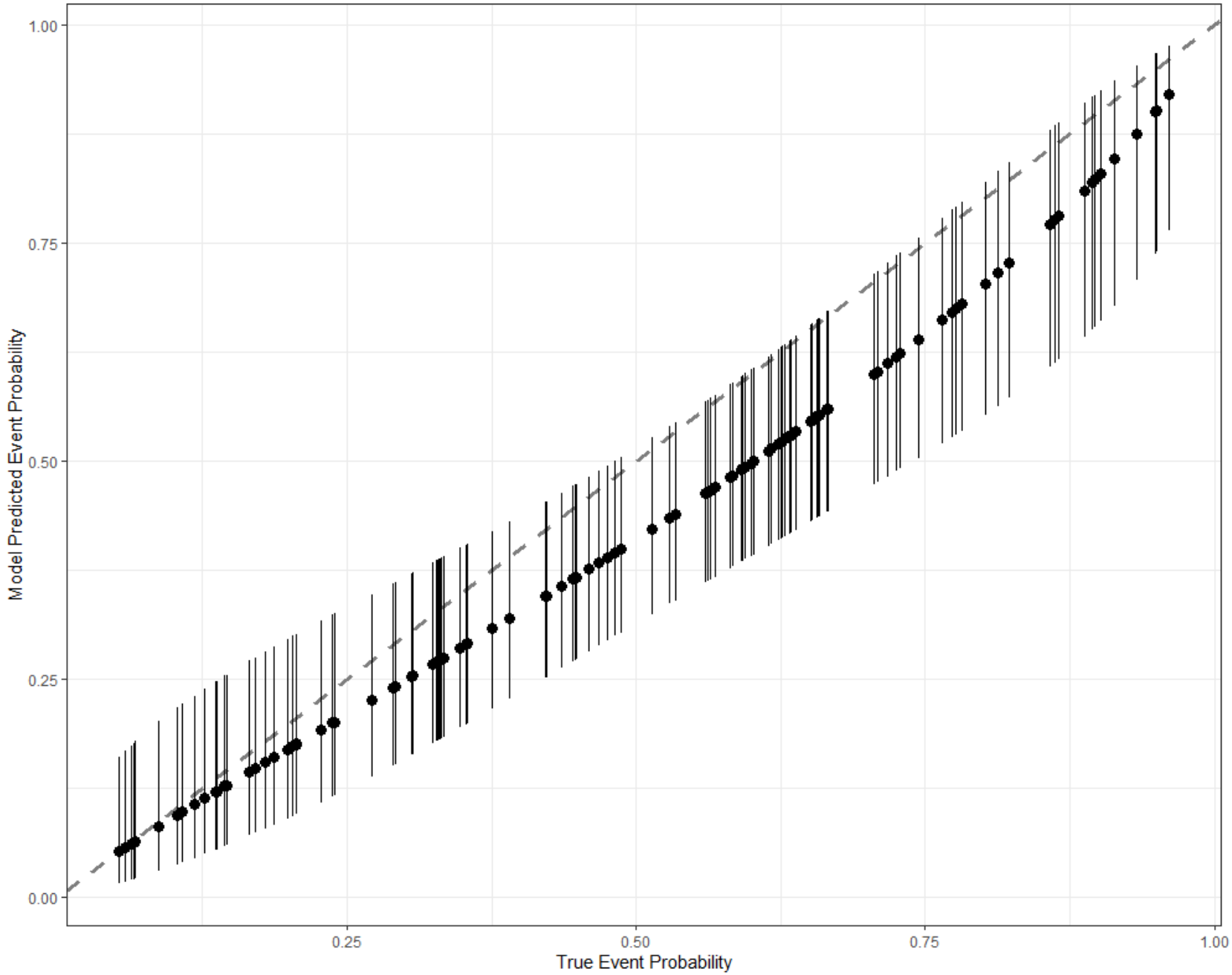
The model predicts probability, however we **CANNOT** directly compare the predicted to an observed probability!!



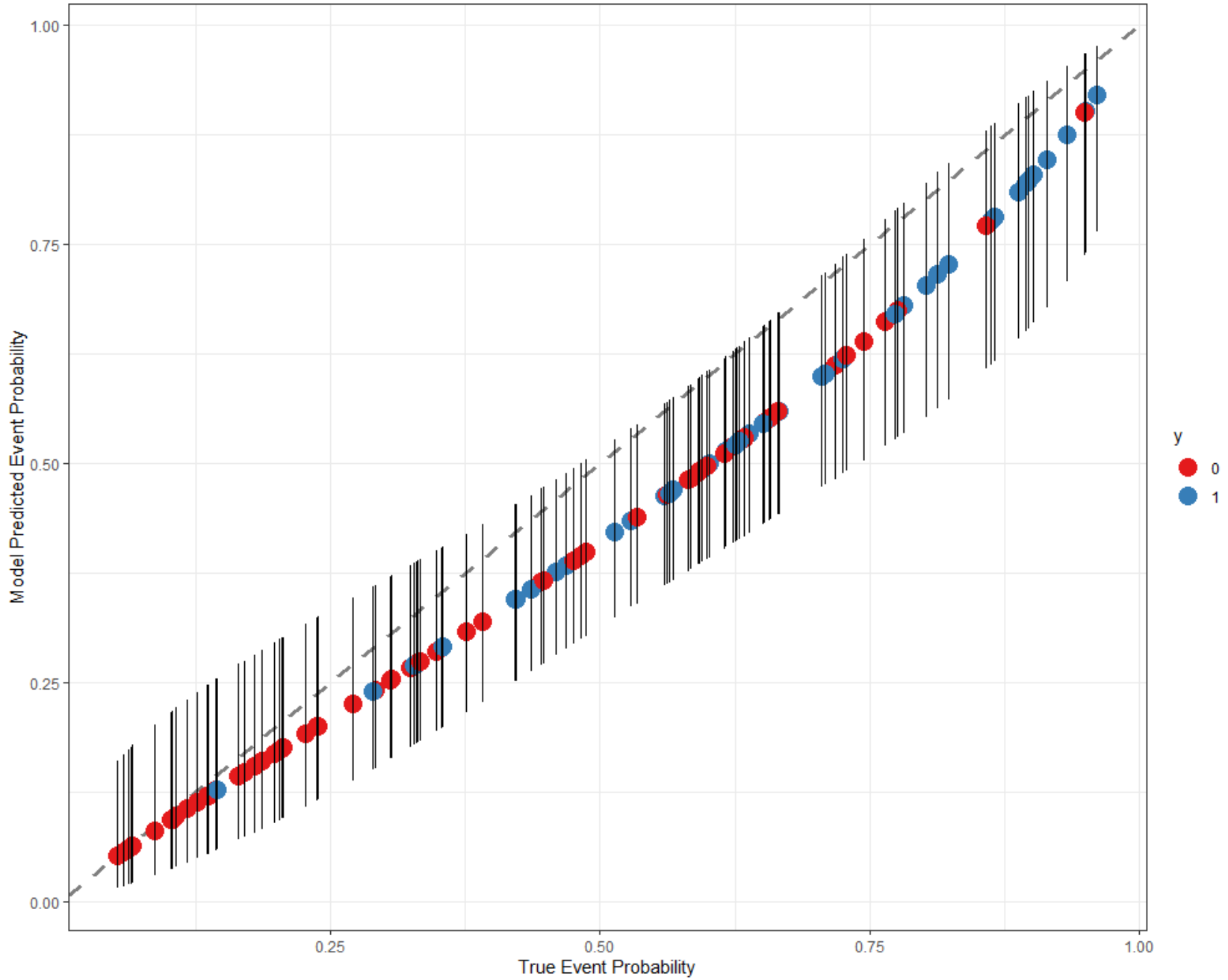
This figure can only be made because this is a TOY PROBLEM!



So what can we learn from this figure?

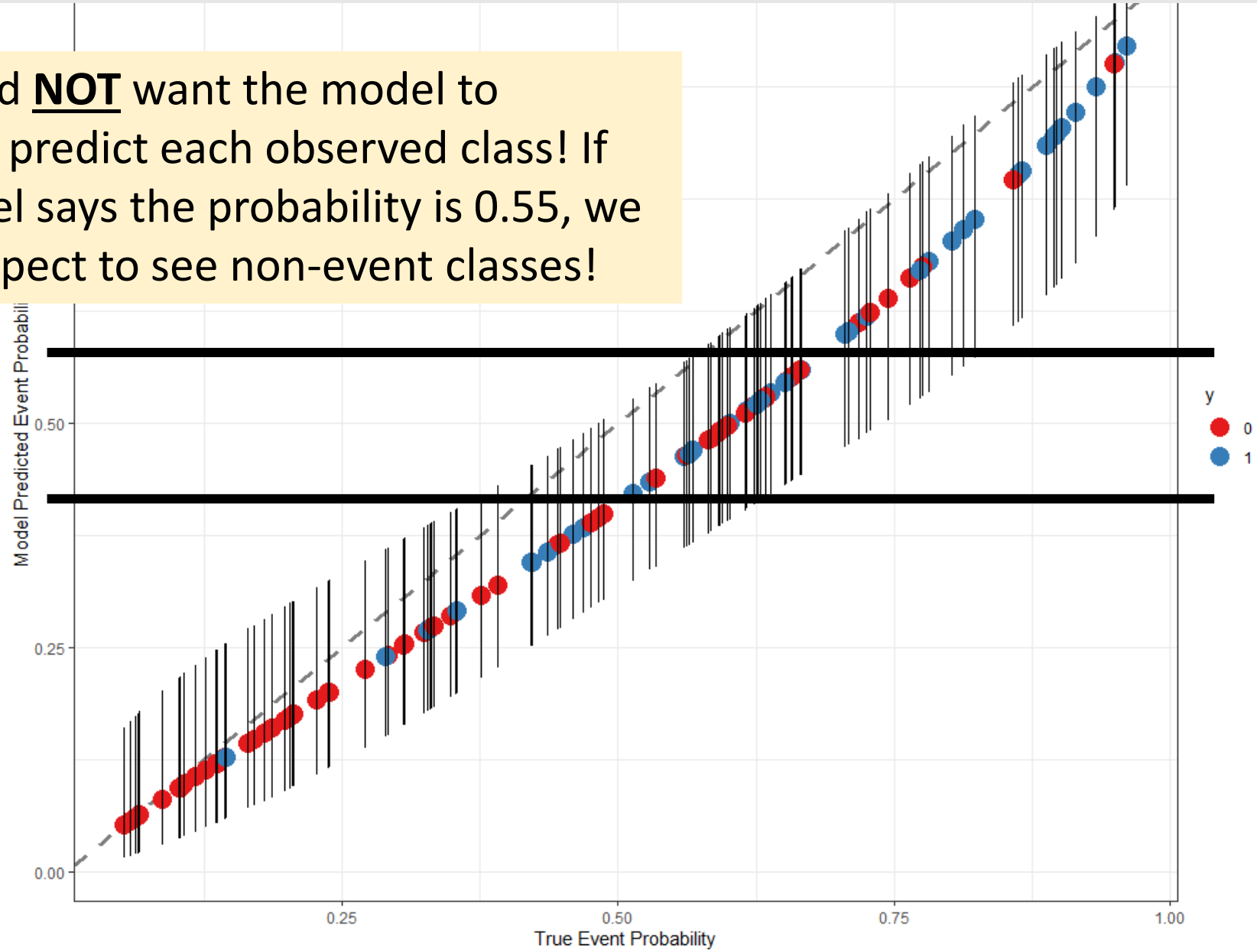


Denote the observed response by the marker color

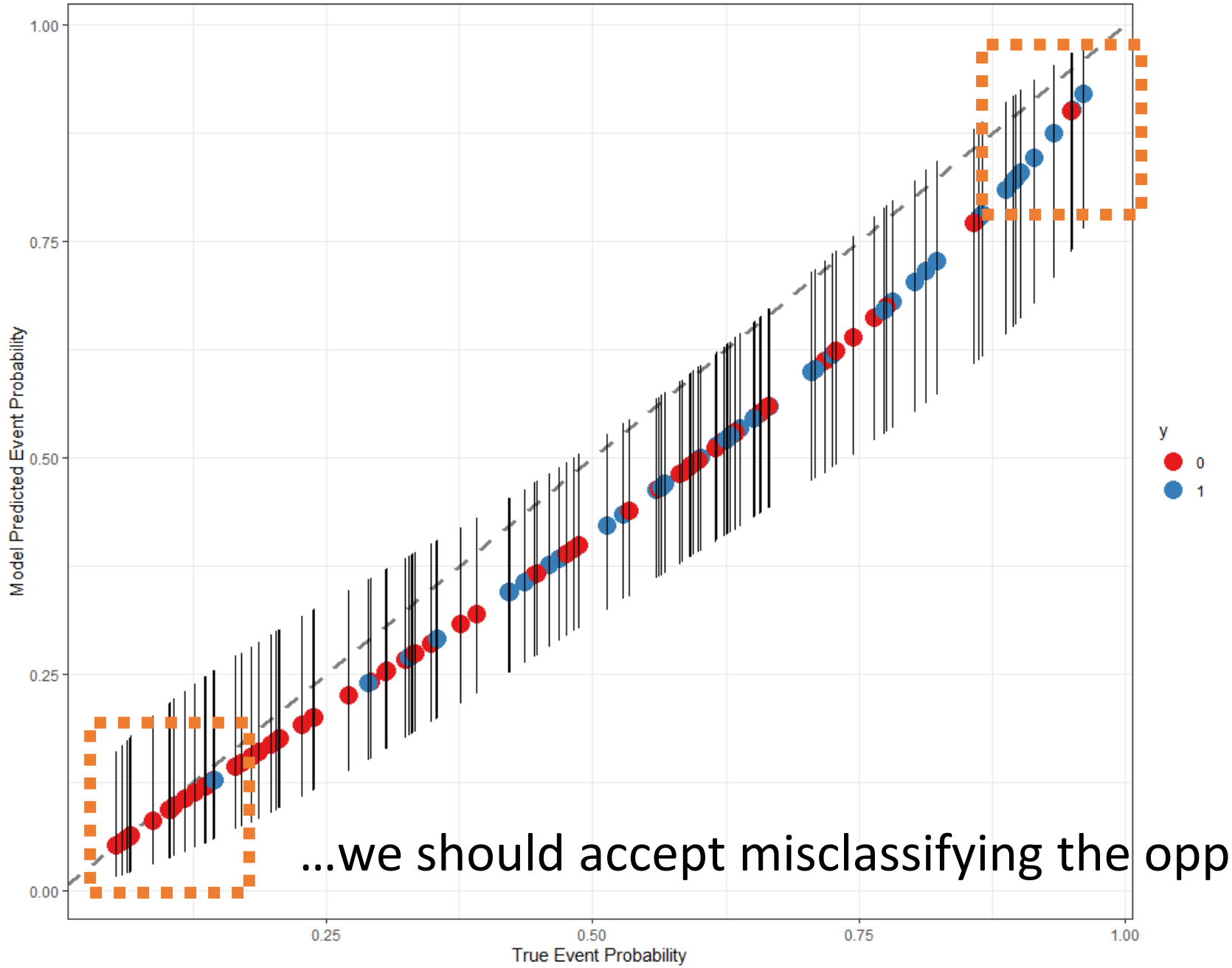


When the model predicts the probability to be around 0.5, we should expect to see a near even mix between the event and non-event.

We would **NOT** want the model to perfectly predict each observed class! If the model says the probability is 0.55, we would expect to see non-event classes!



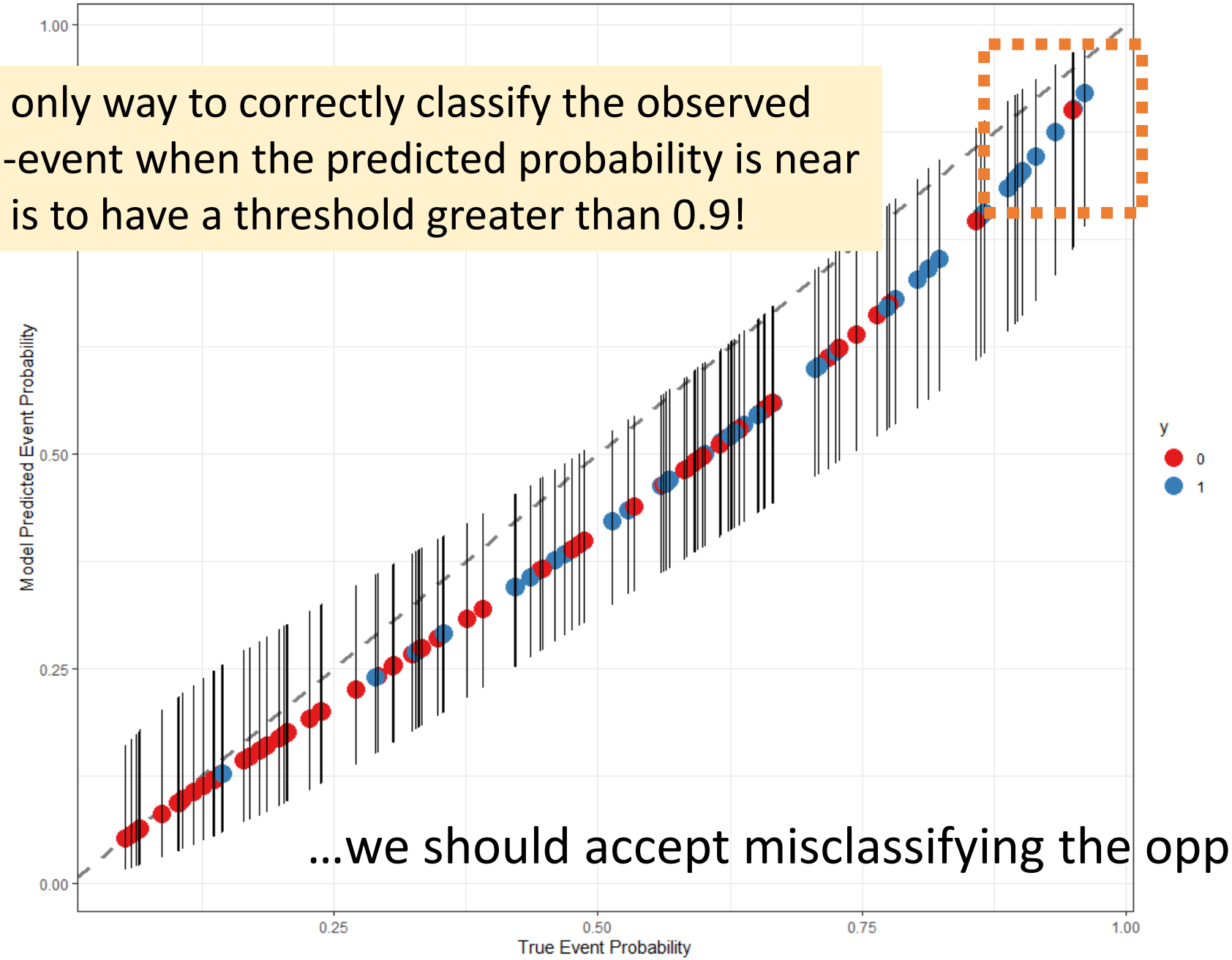
If the model is representative of behavior when predicting high or low event probabilities…



…we should accept misclassifying the opposite class!!!!!

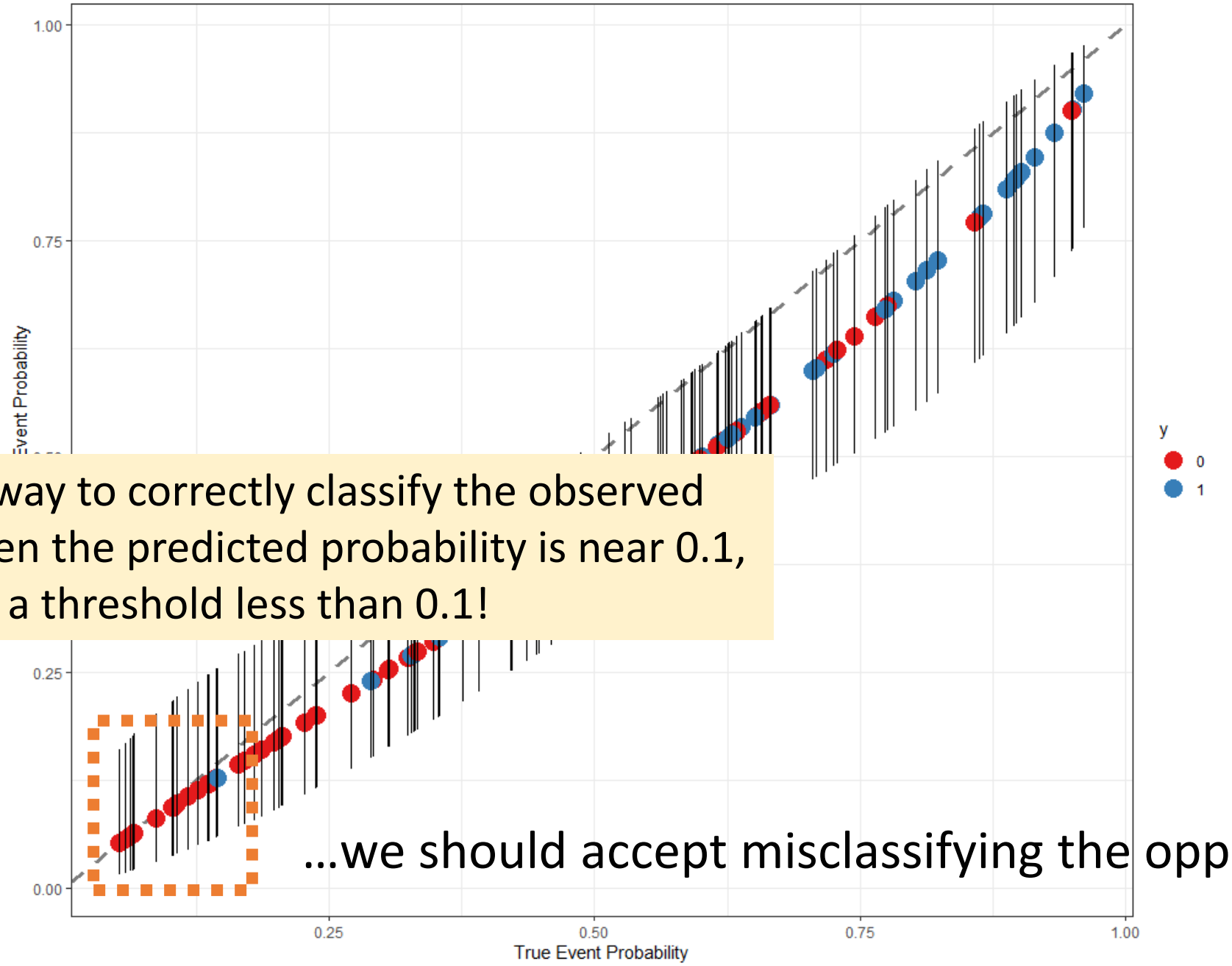
If the model is representative of behavior when predicting high or low event probabilities…

The only way to correctly classify the observed non-event when the predicted probability is near 0.9, is to have a threshold greater than 0.9!



…we should accept misclassifying the opposite class!!!!!

If the model is representative of behavior when predicting high or low event probabilities…



The only way to correctly classify the observed event when the predicted probability is near 0.1, is to have a threshold less than 0.1!

…we should accept misclassifying the opposite class!!!!!

Well calibrated models are those with agreement between predicted probabilities and empirical frequencies

- If the model predicts a probability of 0.33, we expect, on average, to observe the event roughly one third of the time.

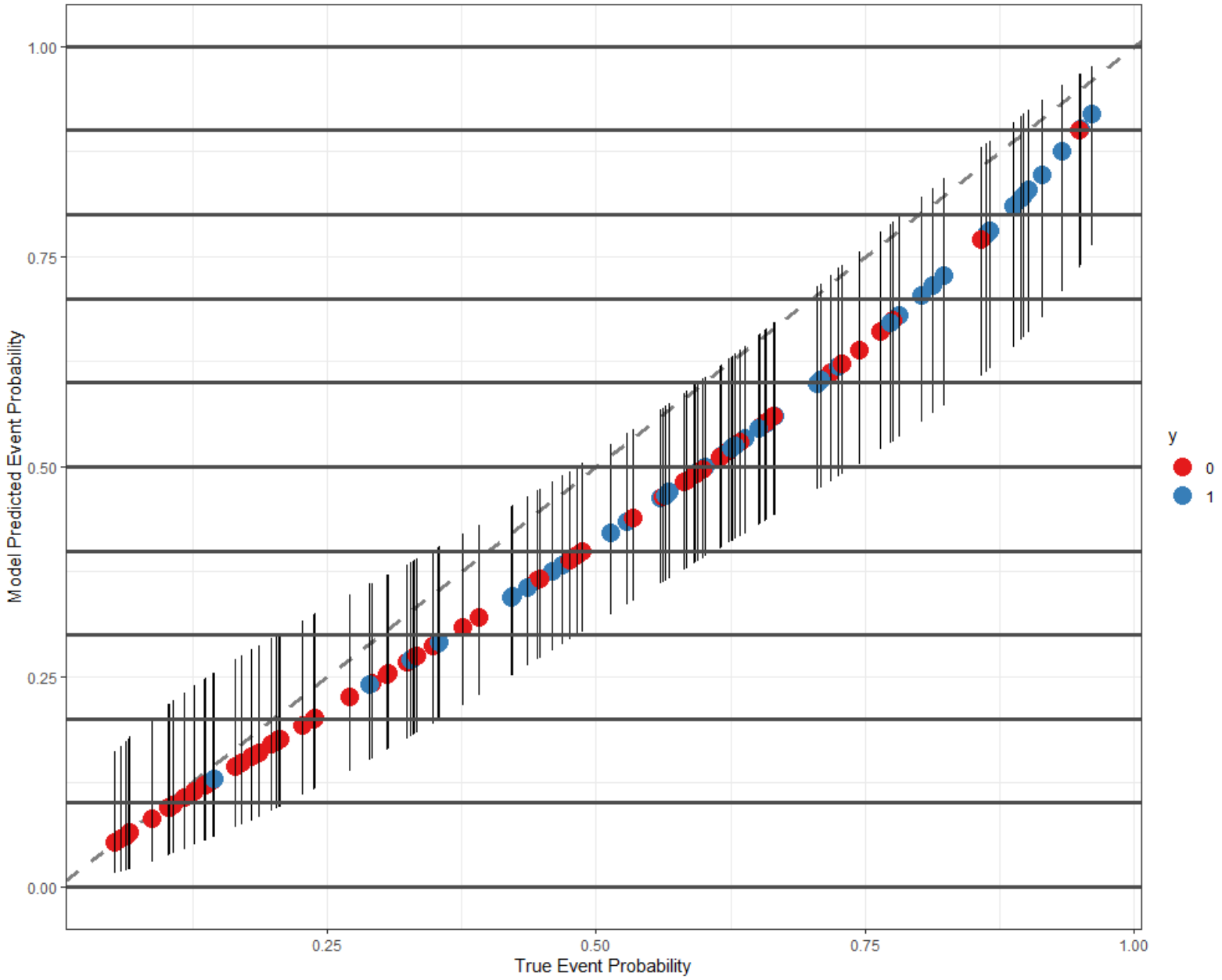
- If we do in fact observe the event about one third of time, for conditions associated with the model predicting 0.33, the model appears to representative of the process.

- We would call such a model a **well calibrated model**.

Calibration curves are graphical tools to accomplish this goal

- Compare predicted probabilities to empirical event frequencies.
- The event frequencies are estimated by binning the observations together.
- Bins are defined based on the predicted probability.

Bins defined based on the predicted event probability, 10 bins shown below

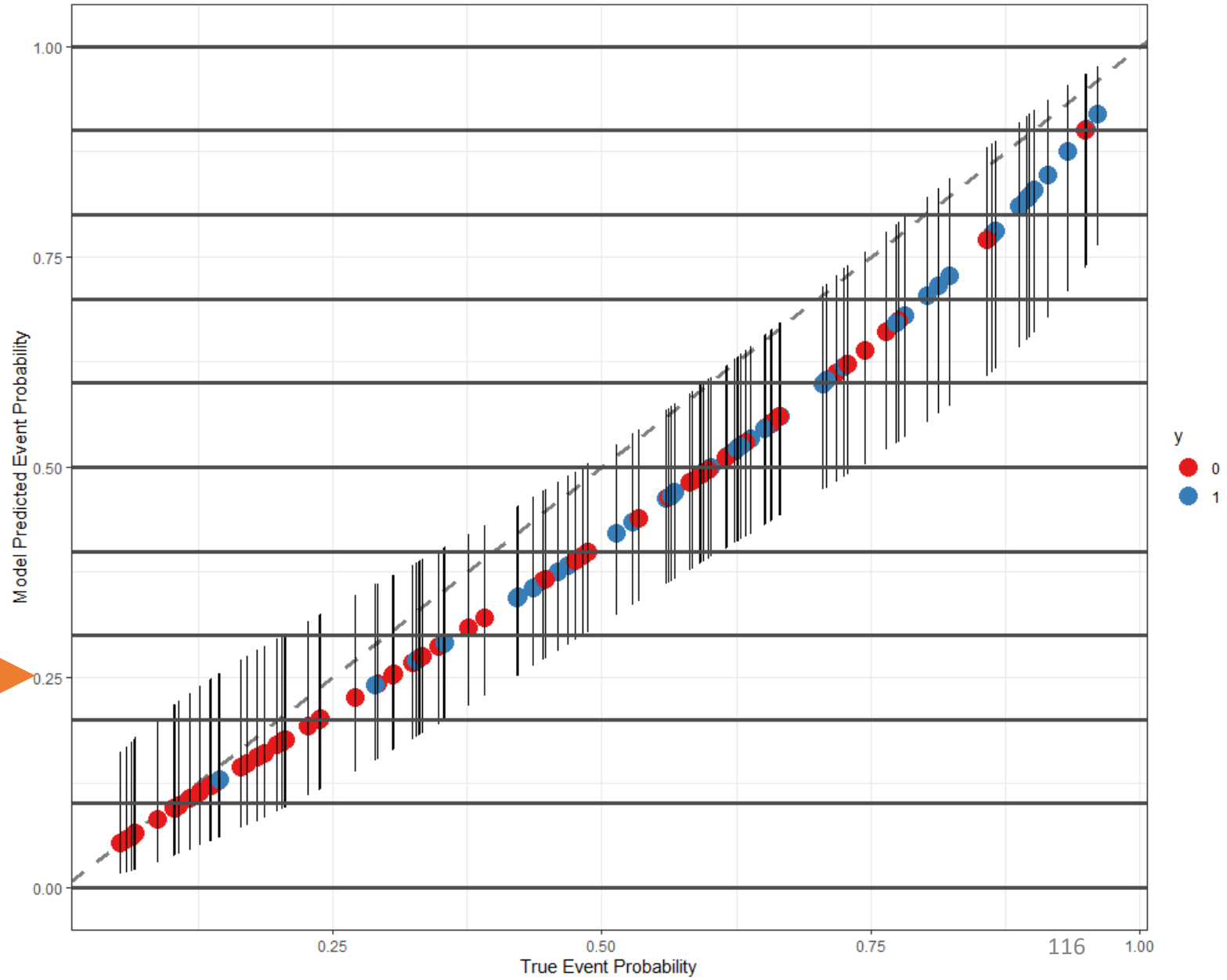


Bins defined based on the predicted event probability, 10 bins shown below

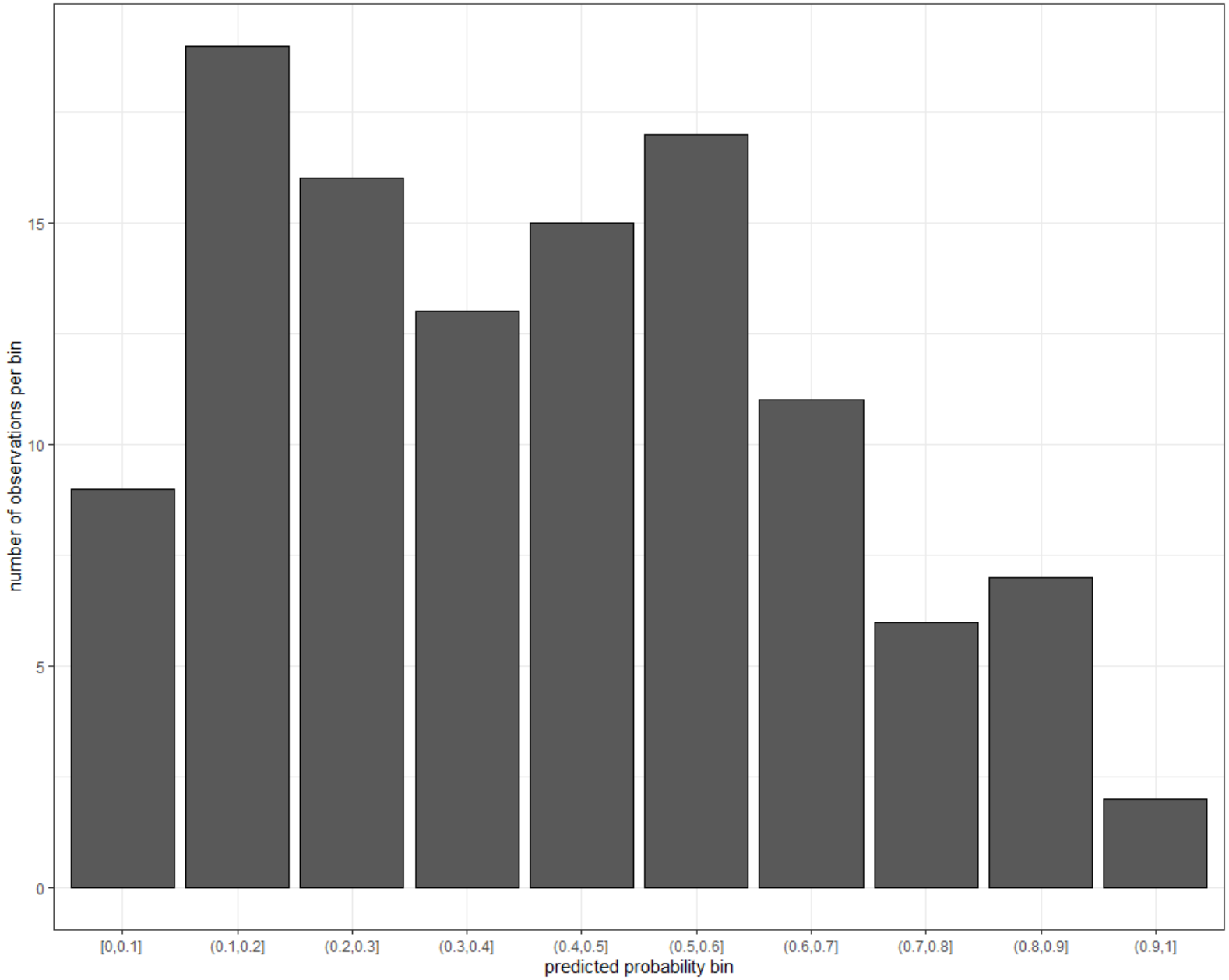
Predicted probabilities will be represented by the mid-point of each bin.

"Lumps" separate predictions into representative "forecasts".

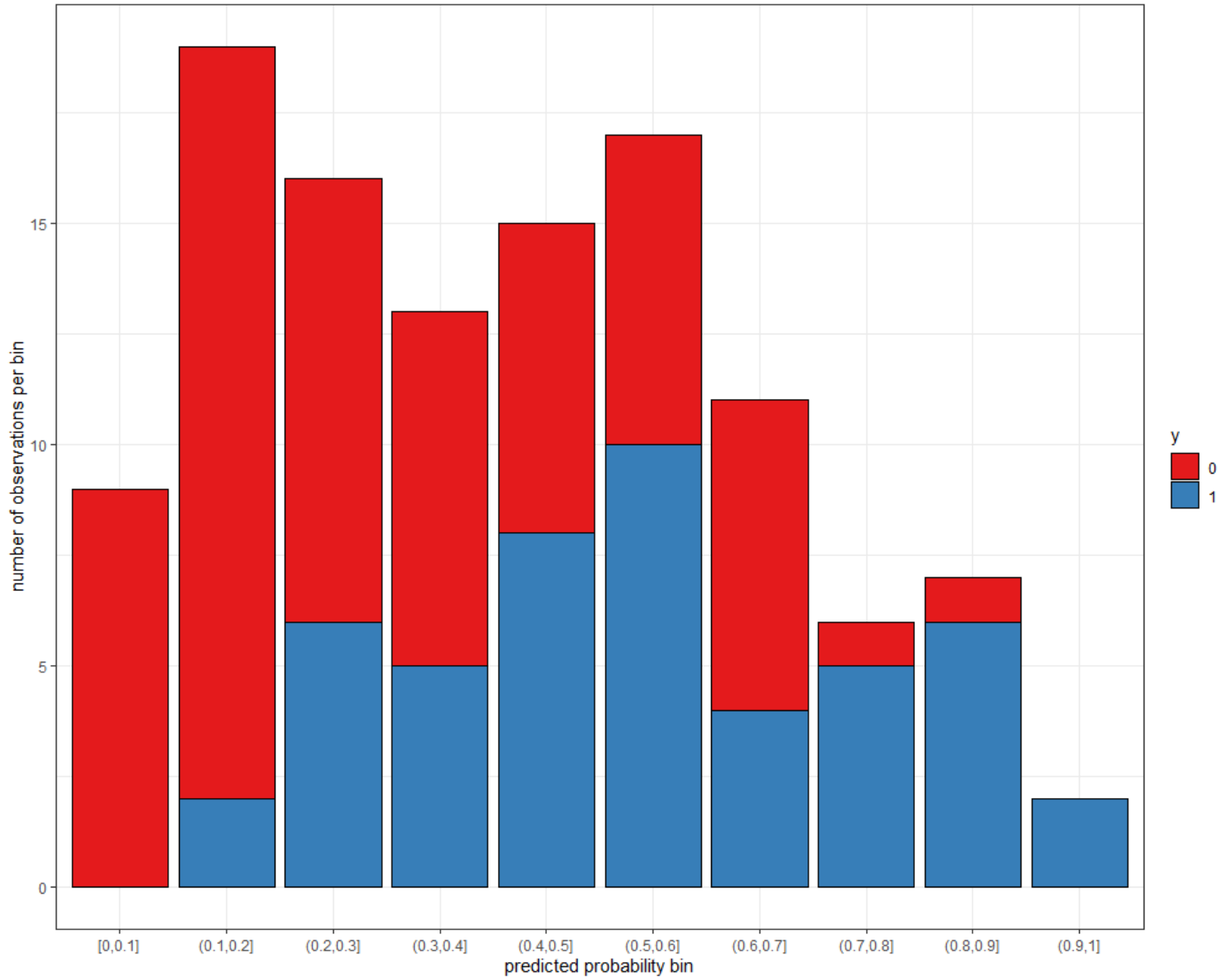
The 25% bin is between 20% and 30%



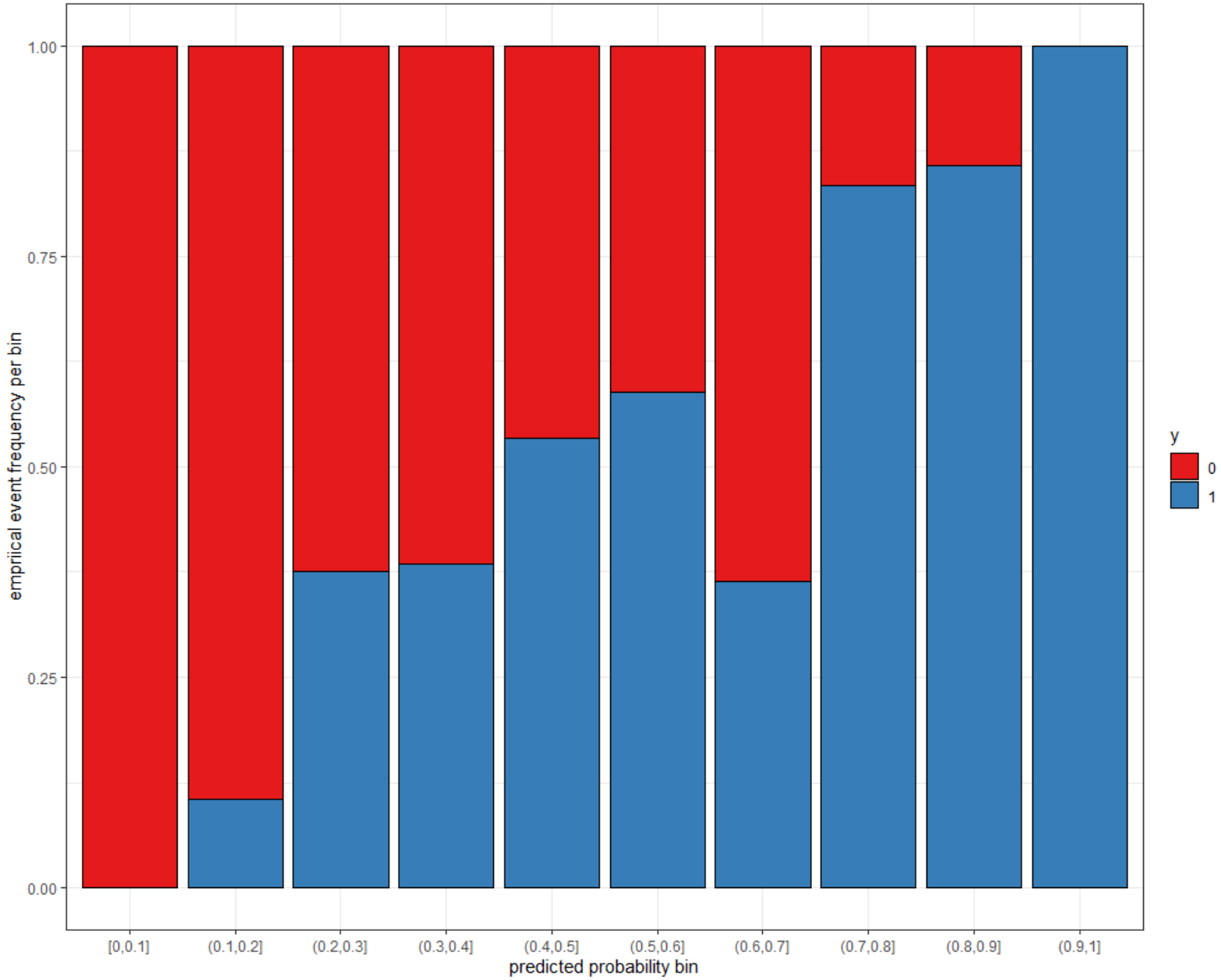
Count the number of observations within each predicted probability bin



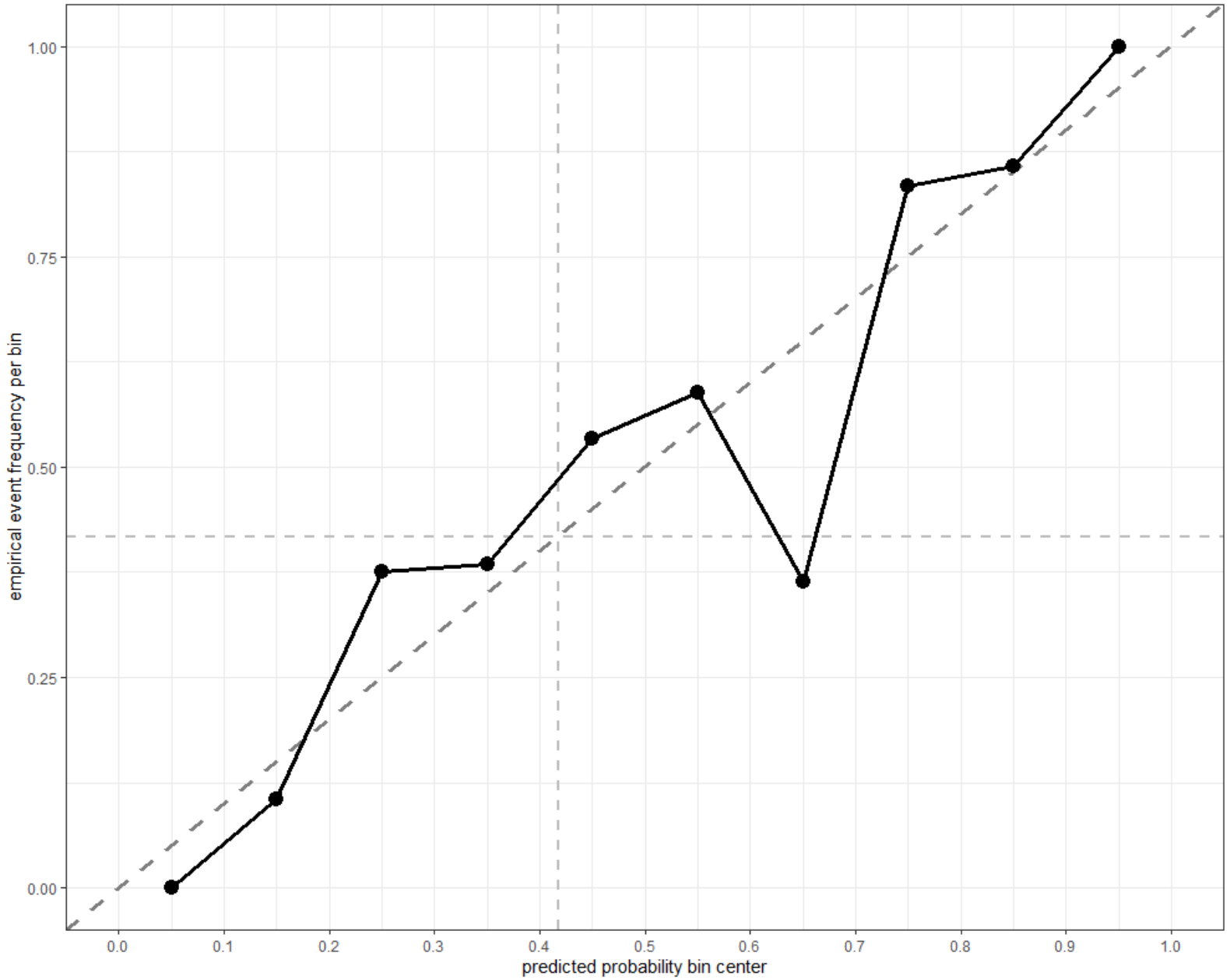
Count the number of events and non-events within each bin



Calculate the empirical event frequency/proportion per bin



Calibration curve based on bin width of 0.1

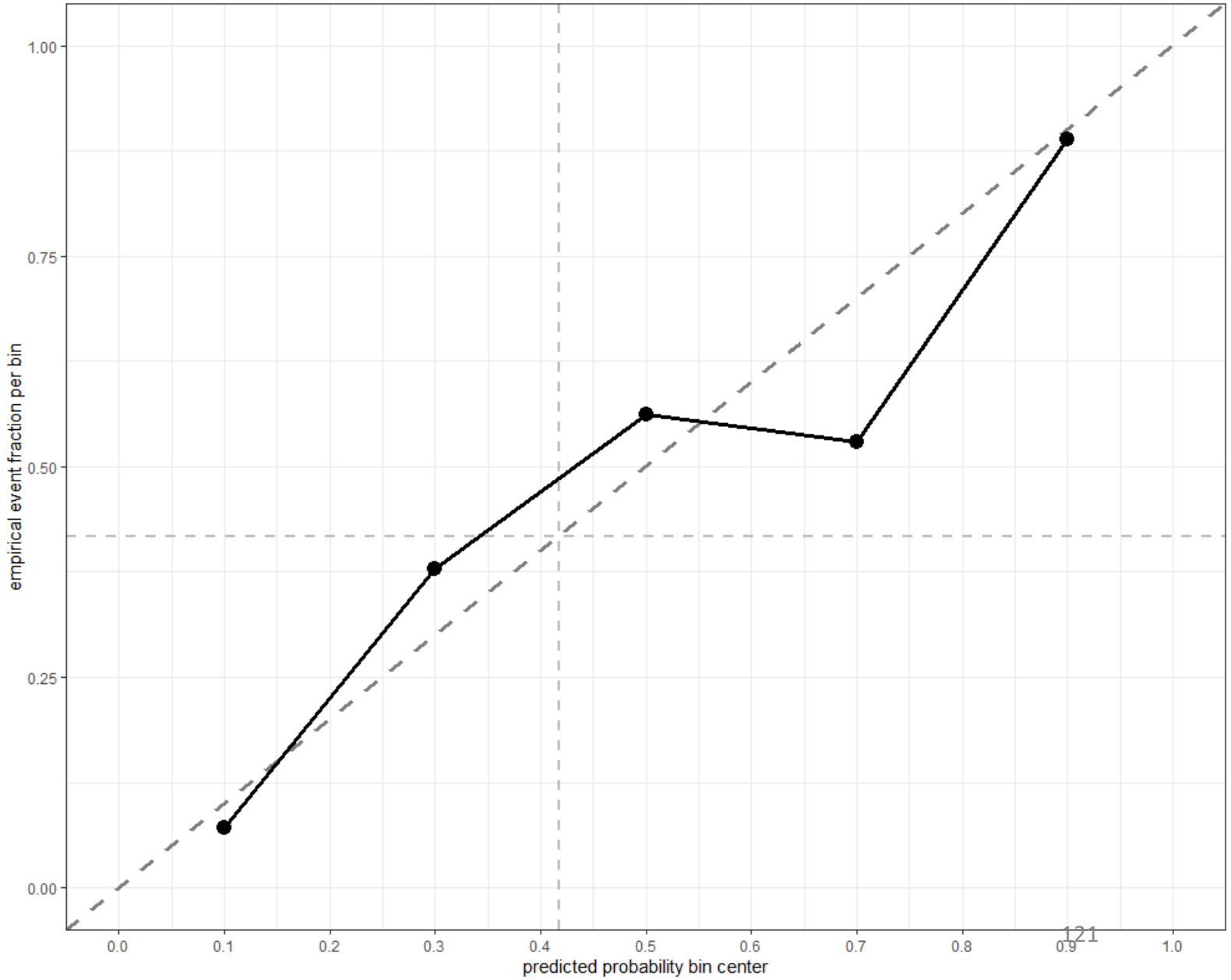


Challenge with calibration curves…specifying the bin width…bin width of 0.2

Fewer bins means there are more samples per bin.

Less susceptible to low sample size estimates to the event frequency per bin.

Too few bins however limits the ability to check model performance over wide range of conditions.



Calibration curves ignore point-wise accuracy

- Think of calibration curves as studying the long-term behavior of the model.

- If the model is well-calibrated, the model predictions will represent the long-run frequencies of the classes.