ROC Curves: An Overview

PSTAT 131/231

2022-10-19

Data For this, we'll use the Titanic dataset, as seen in Homework 3. We load in the data below and perform any necessary steps – converting variables to factors as needed, loading the appropriate packages, etc.:

```
library(tidyverse)
library(tidymodels)
library(discrim)
library(poissonreg)
library(corrr)
library(ggthemes)
tidymodels_prefer()

set.seed(3000) # can be any number

titanic <- read_csv(file = "homework-3/data/titanic.csv") %>%
  mutate(survived = factor(survived,
                           levels = c("Yes", "No")),
         pclass = factor(pclass),
         coin_flip = c(rep(0, 445), rep(1, 446)))

titanic_split <- titanic %>%
  initial_split(strata = survived, prop = 0.7)
titanic_train <- training(titanic_split)
titanic_test <- testing(titanic_split)
```

We split the data into training and testing sets, and choose to include 70% of the data in training and 30% in testing. Stratified sampling is important here because the outcome is imbalanced; there is a larger number of passengers that didn't survive than passengers that did.

Now we'll create recipes, and here we are going to specify two recipes – one that will do pretty well predicting the outcome, and one that won't.

The recipe described in Homework 3 will do a fairly good job, so we'll use that for what we'll call Recipe A:

Recipe A:

```
titanic_recipe_a <- recipe(survived ~ pclass + sex + age +
                           sib_sp + parch + fare, titanic_train) %>%
  step_impute_linear(age, impute_with = imp_vars(sib_sp)) %>%
  # choice of predictors to impute with is up to you
  step_dummy(all_nominal_predictors()) %>%
  step_interact(~ starts_with("sex"):age + age:fare)
```

For Recipe B, we will attempt to predict passenger survival with only `sib_sp`, the number of siblings or spouses on board:

Recipe B:

```
titanic_recipe_b <- recipe(survived ~ sib_sp, titanic_train)
```

Note that this recipe is much shorter; there's no age to impute, no categorical predictors, and no interactions (because there's only one predictor variable included).

Col1	Col2	Col3

We'll fit two logistic regressions, using each of these recipes, and call them Model A and Model B, respectively.

Model A:

```
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wkflow_a <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(titanic_recipe_a)

log_fit_a <- fit(log_wkflow_a, titanic_train)
```

Model B:

```
log_wkflow_b <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(titanic_recipe_b)

log_fit_b <- fit(log_wkflow_b, titanic_train)
```

We can obtain the accuracy for each model:

```
log_acc_a <- predict(log_fit_a, new_data = titanic_train, type = "class") %>%
  bind_cols(titanic_train %>% select(survived)) %>%
  accuracy(truth = survived, estimate = .pred_class)
log_acc_b <- predict(log_fit_b, new_data = titanic_train, type = "class") %>%
  bind_cols(titanic_train %>% select(survived)) %>%
  accuracy(truth = survived, estimate = .pred_class)

results <- bind_rows(log_acc_a, log_acc_b) %>%
  tibble() %>% mutate(model = c("Model A", "Model B")) %>%
  select(model, .estimate)
results
```

```
## # A tibble: 2 x 2
##   model   .estimate
##   <chr>       <dbl>
## 1 Model A     0.807
## 2 Model B     0.616
```

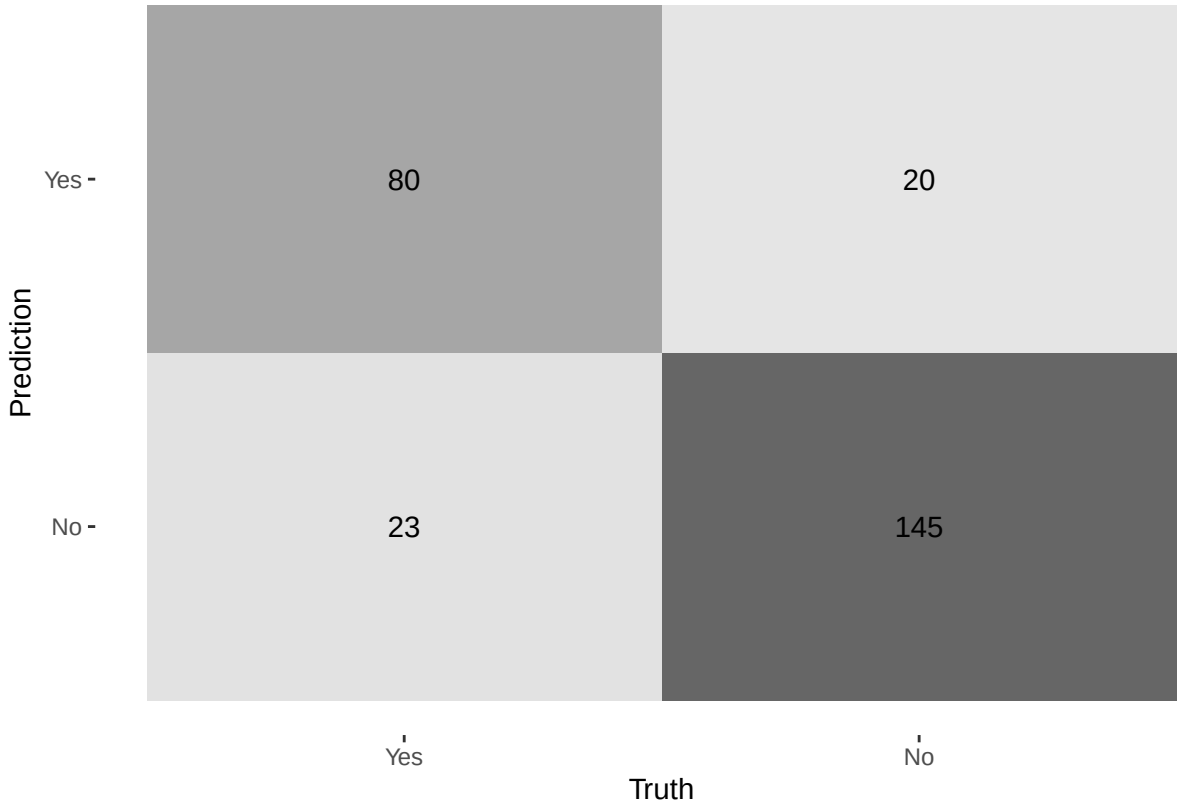
As expected, Model A has higher accuracy than Model B.

We'll fit both of these to the testing data, and then we'll explore their ROC curves.

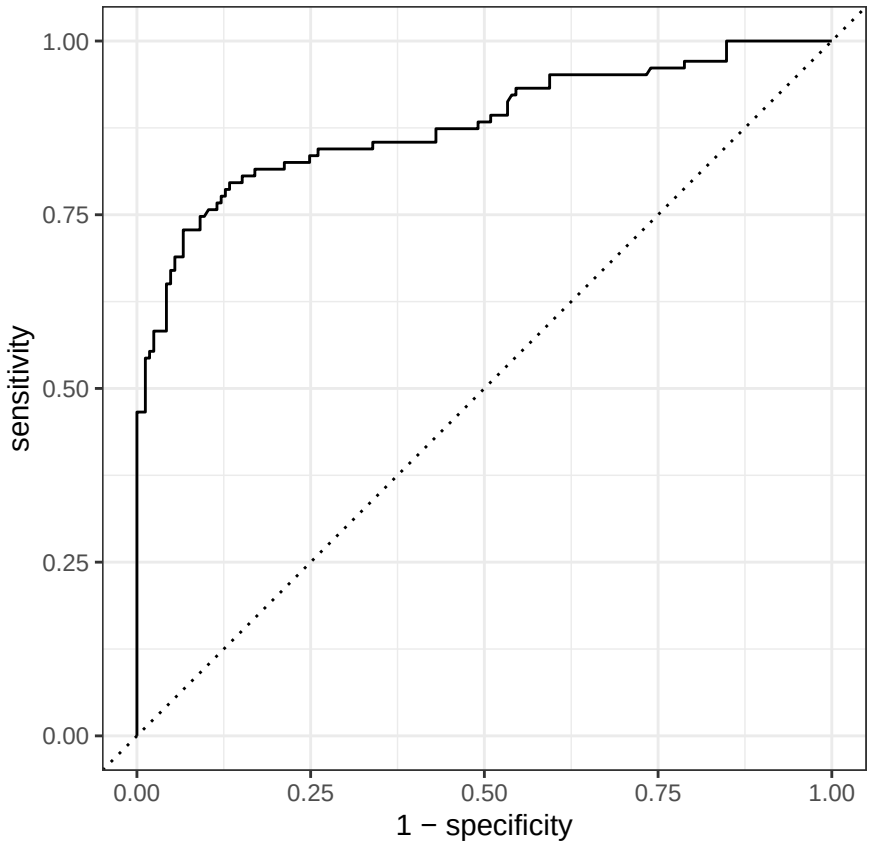
```
log_test_a <- fit(log_wkflow_a, titanic_test)
log_a_results <- augment(log_test_a, new_data = titanic_test)

log_test_b <- fit(log_wkflow_b, titanic_test)
log_b_results <- augment(log_test_b, new_data = titanic_test)
```

Model A: ROC Curve First, let's view a confusion matrix:

```
log_a_results %>%
  conf_mat(truth = survived, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```




```
log_a_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot()
```



We can look at the plot, and we can also ask R for the table of threshold values and their corresponding true positive and false positive rates. As a reminder, "sensitivity" is another term for **true positive** rate, and true positive rate is defined as $\frac{TP}{TP+FN}$, or the number of true positives divided by the total number of true positives and false negatives. The x-axis, 1 - specificity, is another name for the **false positive** rate, which is defined as $\frac{FP}{FP+TN}$. Here is a table of the threshold values, sensitivity, and specificity:

```
log_a_results %>%
  roc_curve(survived, .pred_Yes)
```

```
## # A tibble: 254 x 3
##    .threshold specificity sensitivity
##         <dbl>       <dbl>       <dbl>
##  1 -Inf            0                1
##  2    0.00131      0                1
##  3    0.00474      0.0121           1
##  4    0.00563      0.0182           1
##  5    0.00635      0.0242           1
##  6    0.0116       0.0303           1
##  7    0.0117       0.0364           1
##  8    0.0122       0.0424           1
##  9    0.0147       0.0485           1
## 10    0.0186       0.0545           1
## # ... with 244 more rows
## # i Use `print(n = ...)` to see more rows
```

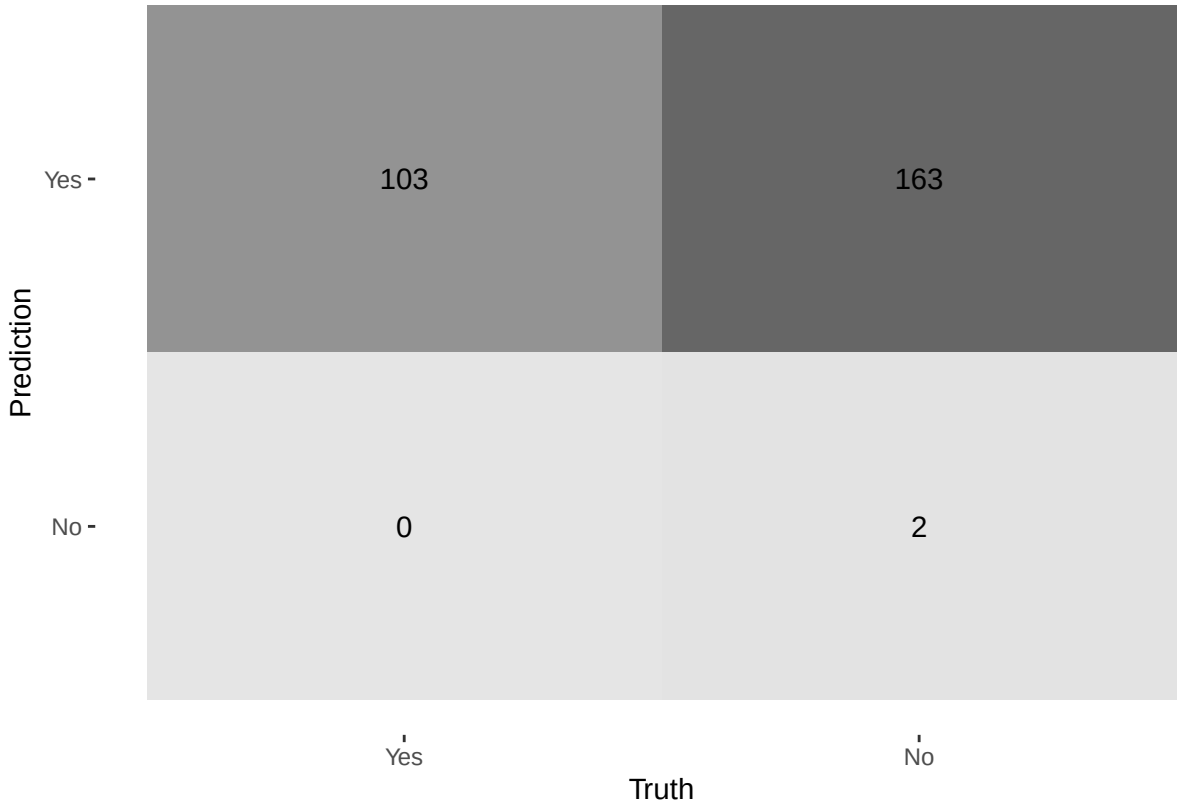
Notice that, by default, the `roc_curve()` function tries values for `threshold` in a grid from `-Inf`, or basically 0, to `Inf`, or basically 1. There are 254 rows, which means it has tried 254 possible values for the threshold. Let's look at the first few predicted class probabilities Model A has generated:

```
log_a_results %>%
  select(.pred_Yes, survived) %>%
  head()
```

```
## # A tibble: 6 x 2
##   .pred_Yes survived
##       <dbl> <fct>
## 1   0.0672  No
## 2   0.268   No
## 3   0.792   Yes
## 4   0.00474 No
## 5   0.247   Yes
## 6   0.693   Yes
```

If we try a threshold of 0.001310378, the first value in the previous table, we would be predicting `Yes` for any observations with $p \geq 0.001310378$. A confusion matrix of the **predicted class values** based on this threshold looks like:

```
augment(log_test_a, new_data = titanic_test) %>%
  mutate(.pred_class = if_else(.pred_Yes < 0.001310378, "No", "Yes")) %>%
  conf_mat(truth = survived, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



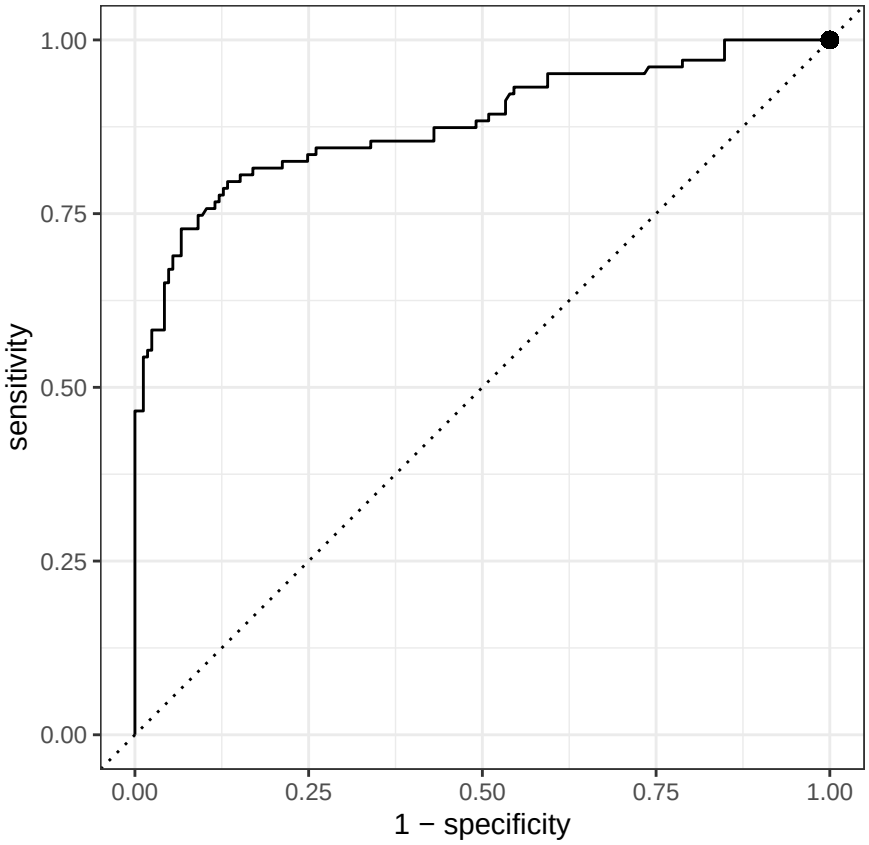
So the true positive rate corresponding to this threshold value is $\frac{103}{103}$, or 1, and the false positive rate, which is defined as 1 - specificity, is $\frac{163}{165} = 0.9878788$, which is basically 1. In other words, the sensitivity is 1 and the specificity is very close to 1, which matches what we see from the ROC table of threshold values:

```
log_a_results %>%
  roc_curve(survived, .pred_Yes) %>%
  head()
```

```
## # A tibble: 6 x 3
##   .threshold specificity sensitivity
##        <dbl>       <dbl>       <dbl>
## 1 -Inf            0                1
## 2    0.00131      0                1
## 3    0.00474      0.0121           1
## 4    0.00563      0.0182           1
## 5    0.00635      0.0242           1
## 6    0.0116       0.0303           1
```

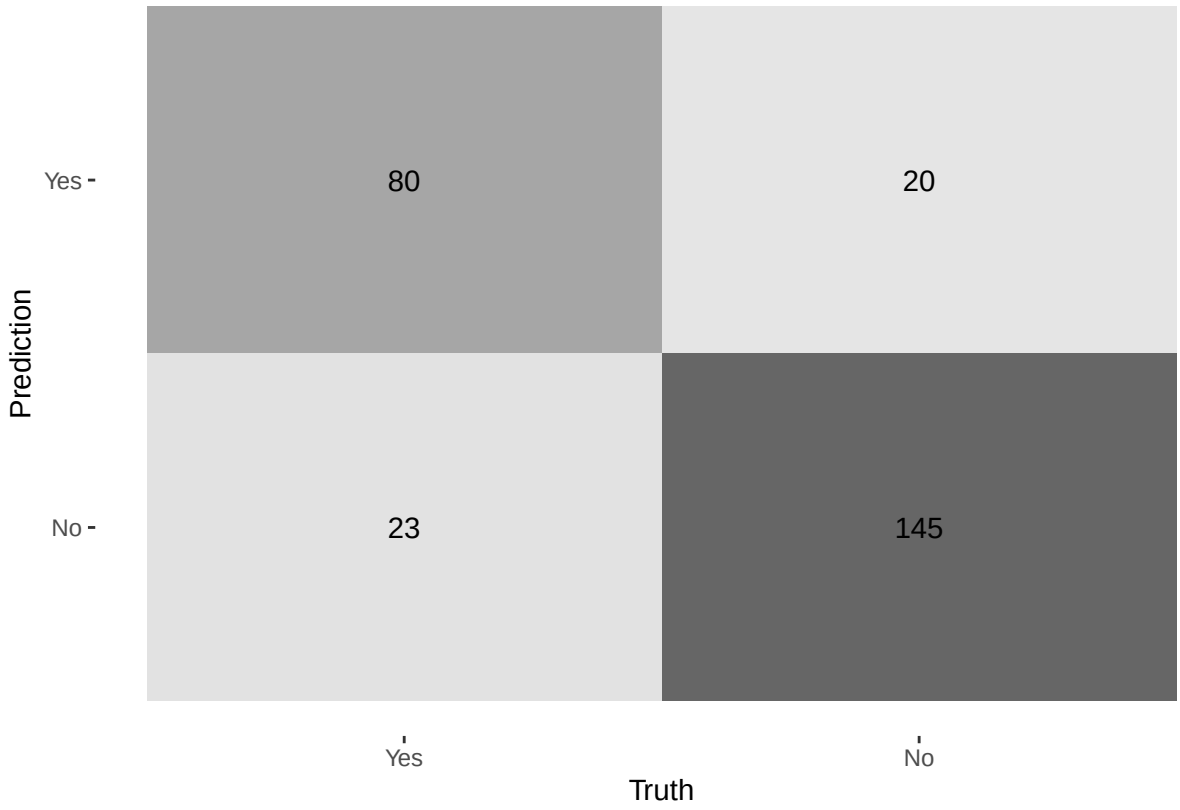
On the ROC curve plot, that is the following point:

```
log_a_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot() +
  geom_point(aes(x = 1, y = 1), colour = spec_colors[1], size = 2.5)
```



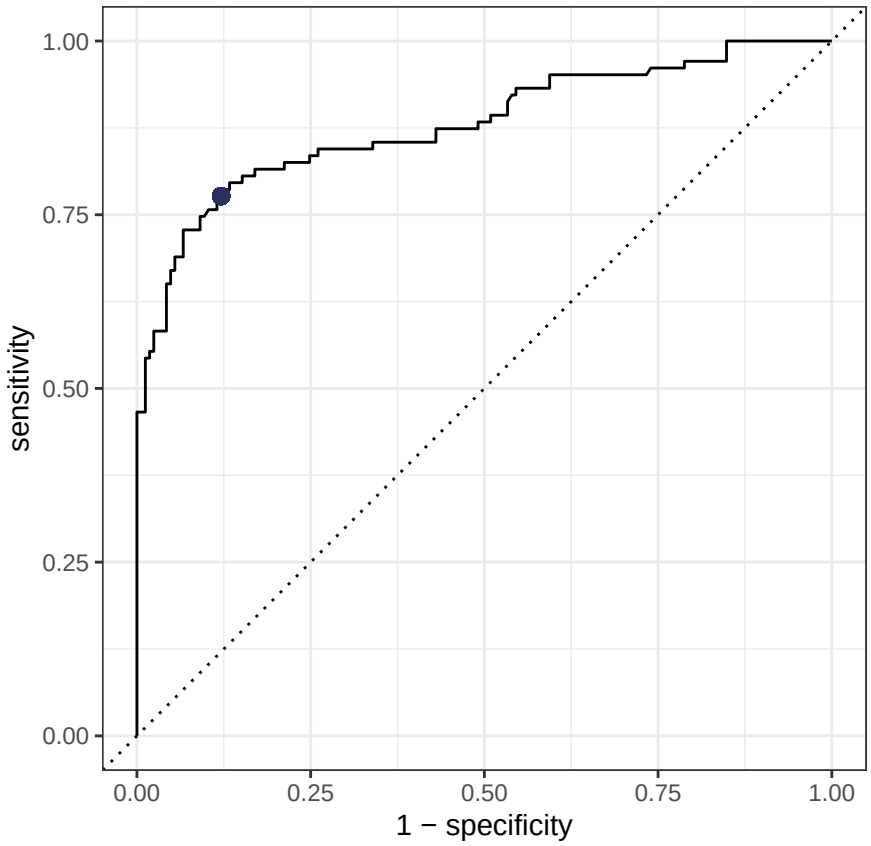
Let's look at the default threshold value for a binary classification problem, and see where that falls on the curve. Generally, the default threshold value is at 0.50. We'll specifically classify the predictions as No if $p < .50$ and `Yes` otherwise:

```
augment(log_test_a, new_data = titanic_test) %>%
  mutate(.pred_class = if_else(.pred_Yes < 0.5, "No", "Yes")) %>%
  conf_mat(truth = survived, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



The true positive rate for a threshold of 0.50 is then $\frac{80}{103} = 0.7767$, and the false positive rate is $\frac{20}{165} = 0.1212121$. On the ROC curve, that is this point:

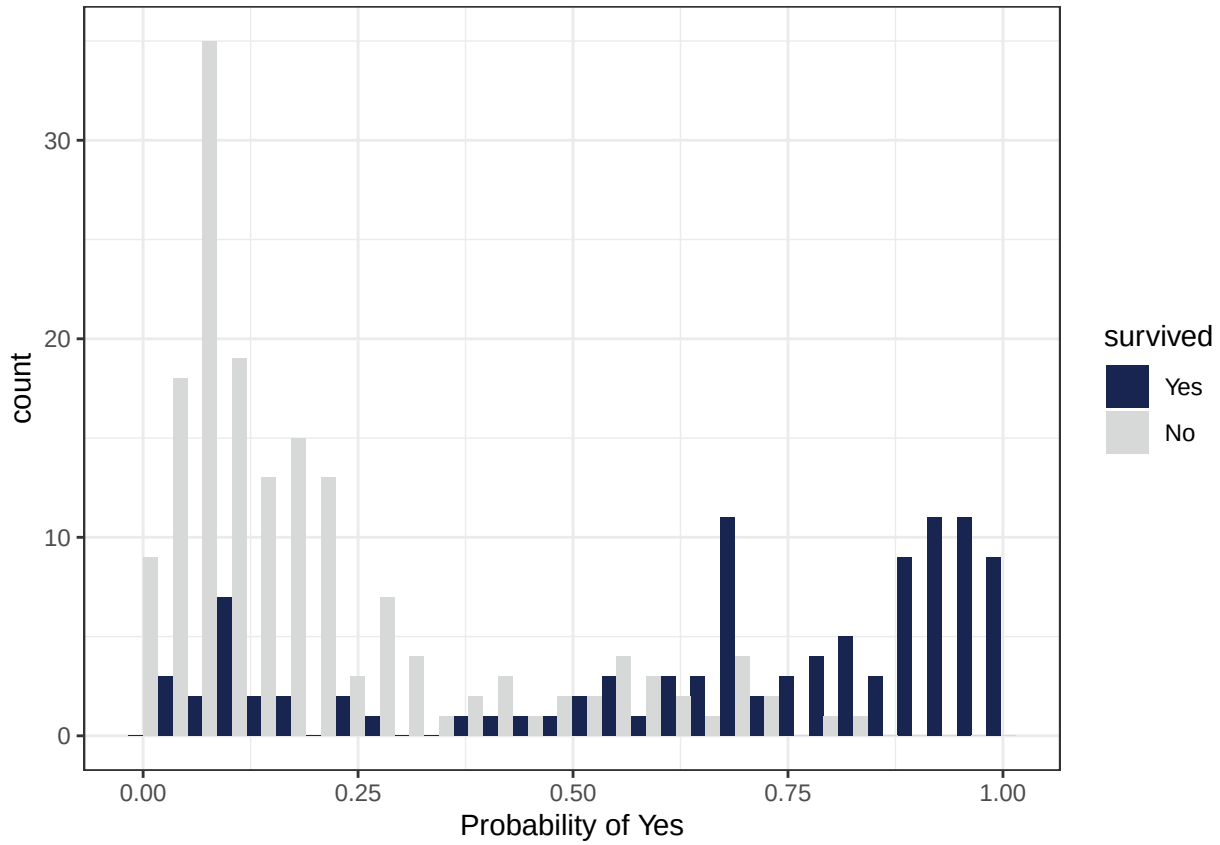
```
log_a_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot() +
  geom_point(aes(x = 0.1212121, y = 0.776699), colour = spec_colors[3], size = 2.5)
```



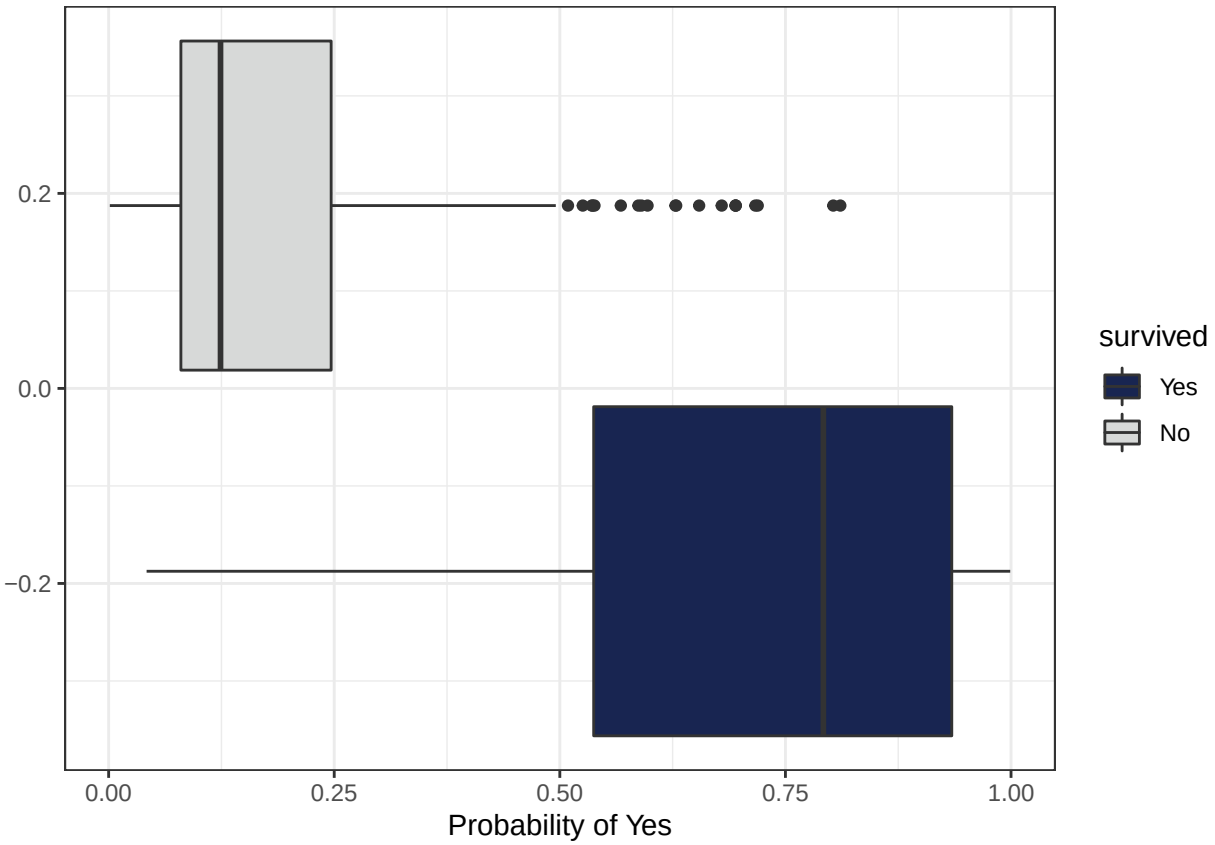
And we could do the same exact thing for threshold values of 0.25, 0.75, etc. – for basically any value between 0 and 1. Our goal is to develop a model that is able to maximize sensitivity while minimizing 1 - specificity, or that can maximize the true positive rate while minimizing the false positive rate.

Here is a distribution of the predicted probabilities:

```
log_a_results %>%
  ggplot(aes(x = .pred_Yes, fill = survived)) +
  geom_histogram(position = "dodge") +
  scale_fill_manual(values = c(spec_colors[2], spec_colors[4])) +
  theme_bw() +
  xlab("Probability of Yes")
```

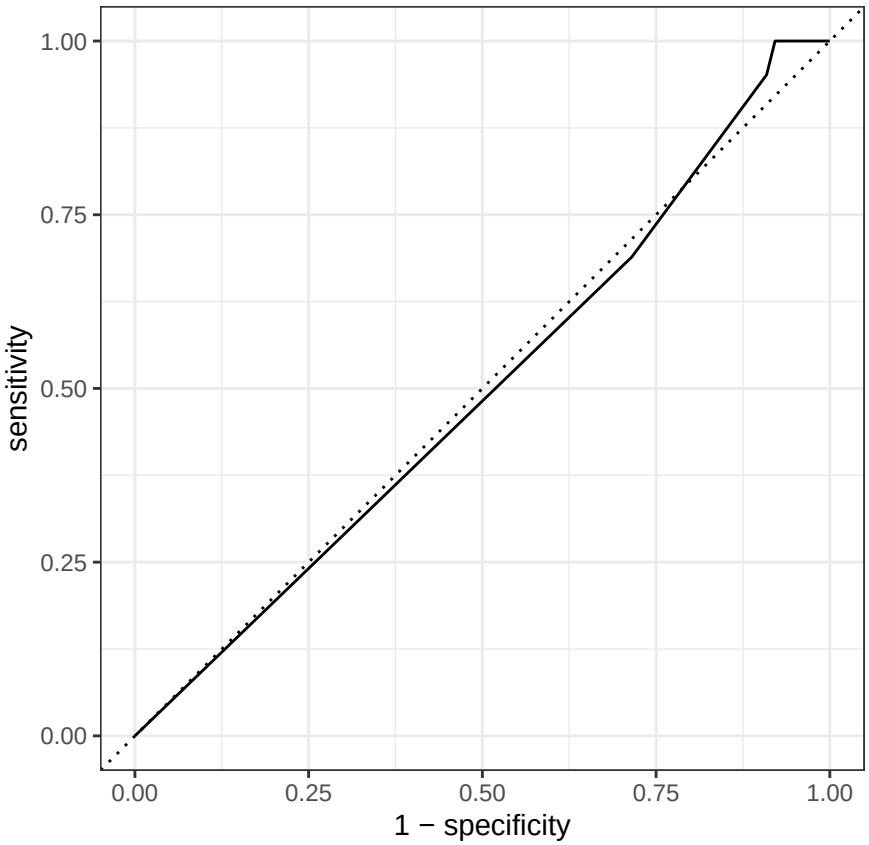



```
log_a_results %>%
  ggplot(aes(x = .pred_Yes, fill = survived)) +
  geom_boxplot() +
  scale_fill_manual(values = c(spec_colors[2], spec_colors[4])) + theme_bw() +
  xlab("Probability of Yes")
```



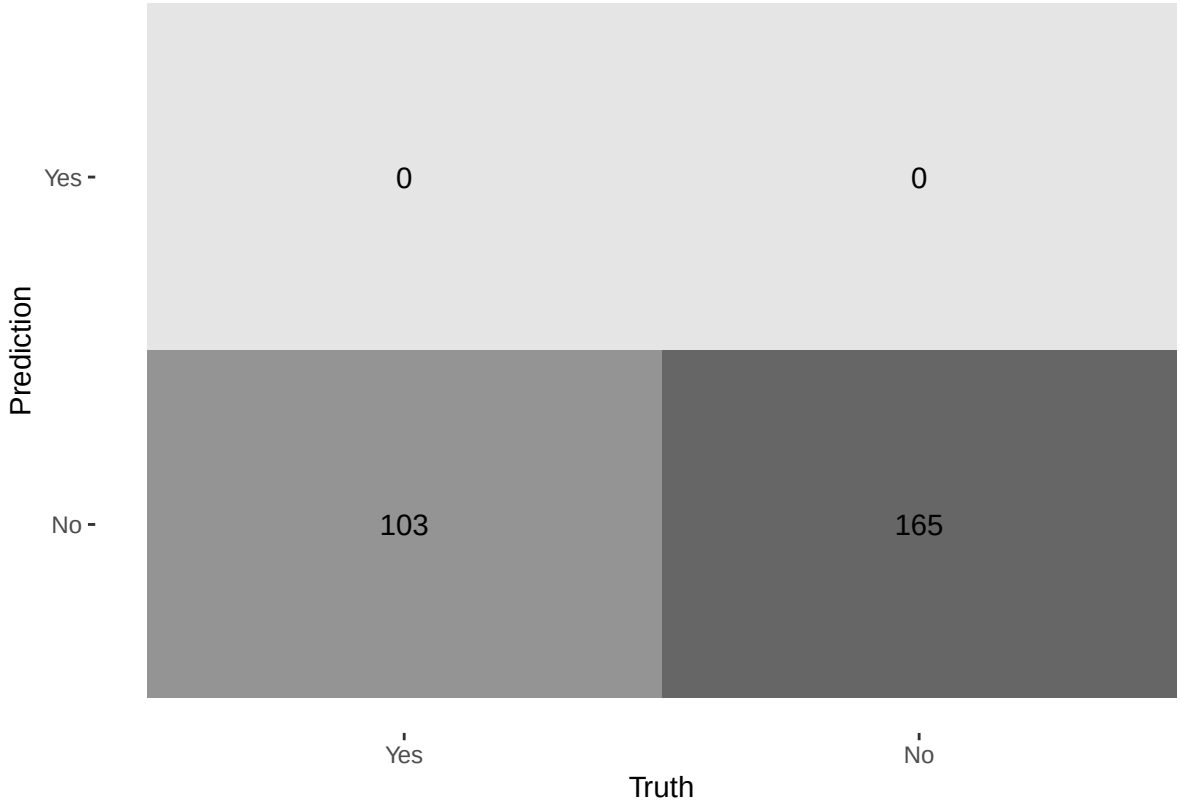
Model B: ROC Curve: For models that do not predict the outcome as well, it basically doesn't matter where you set the threshold value – that model will NEVER achieve a true positive rate of 1 and a false positive rate of 0 (which a "perfect" classifier can do). Let's take a look at the ROC curve of our intentionally bad model:

```
log_b_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot()
```



It's hard to get much worse than that! Predicting survival solely on the basis of number of siblings or spouses on board is just about as bad as predicting survival based on the result of a coin flip. Let's look at the confusion matrix.

```
log_b_results %>%
  conf_mat(truth = survived, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



As we may have suspected – this model is simply never predicting `Yes`.

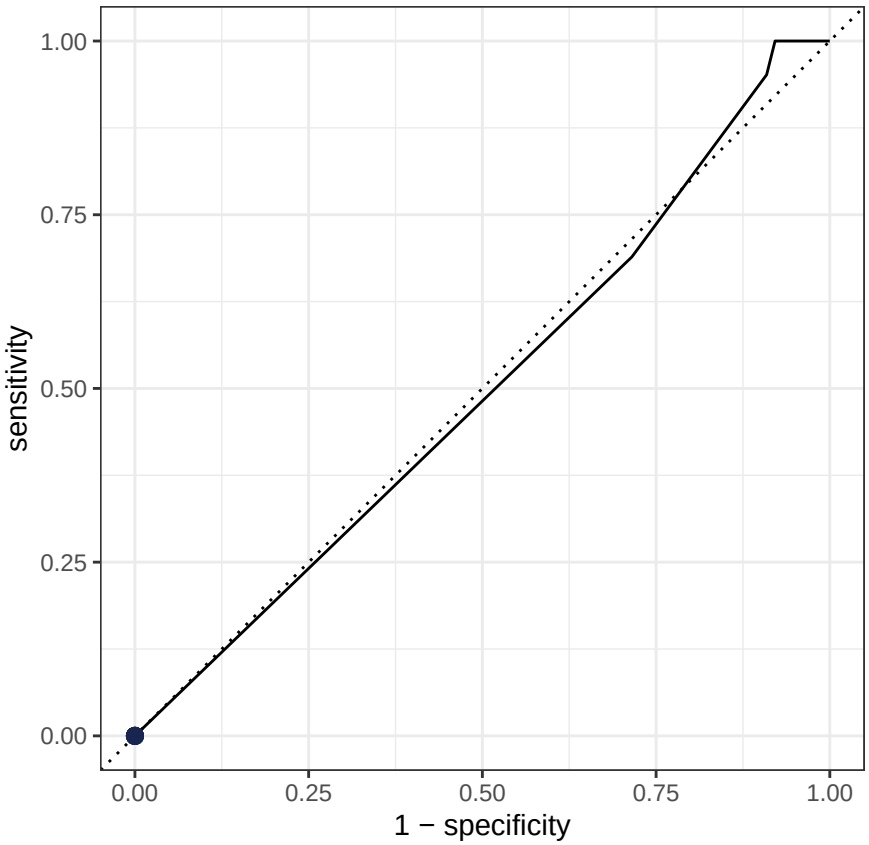
It's doing this because there are more `No` values in the data set, and therefore predicting No for every observation actually results in what appears to be a relatively high accuracy value; the accuracy of this model was about 0.61. But as you'll see, the area under its ROC curve is considerably lower. Area under the ROC curve is a metric for how well the model can successfully discriminate between classes, and this model is doing a pretty terrible job of that. In fact, the area under the ROC curve for this model is only about 0.50 – a coin flip:

```
log_b_results %>%
  roc_auc(survived, .pred_Yes)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.496
```

For the default threshold value of 0.50, we can get the true positive and false positive rates from the confusion matrix above; the true positive rate is $\frac{0}{103}$ and the false positive rate is $\frac{0}{165}$, so sensitivity is 0 and 1 - specificity is 0. The default threshold value of 0.50 is this point on the ROC curve:

```
log_b_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot() +
  geom_point(aes(x = 0, y = 0), colour = spec_colors[2], size = 2.5)
```



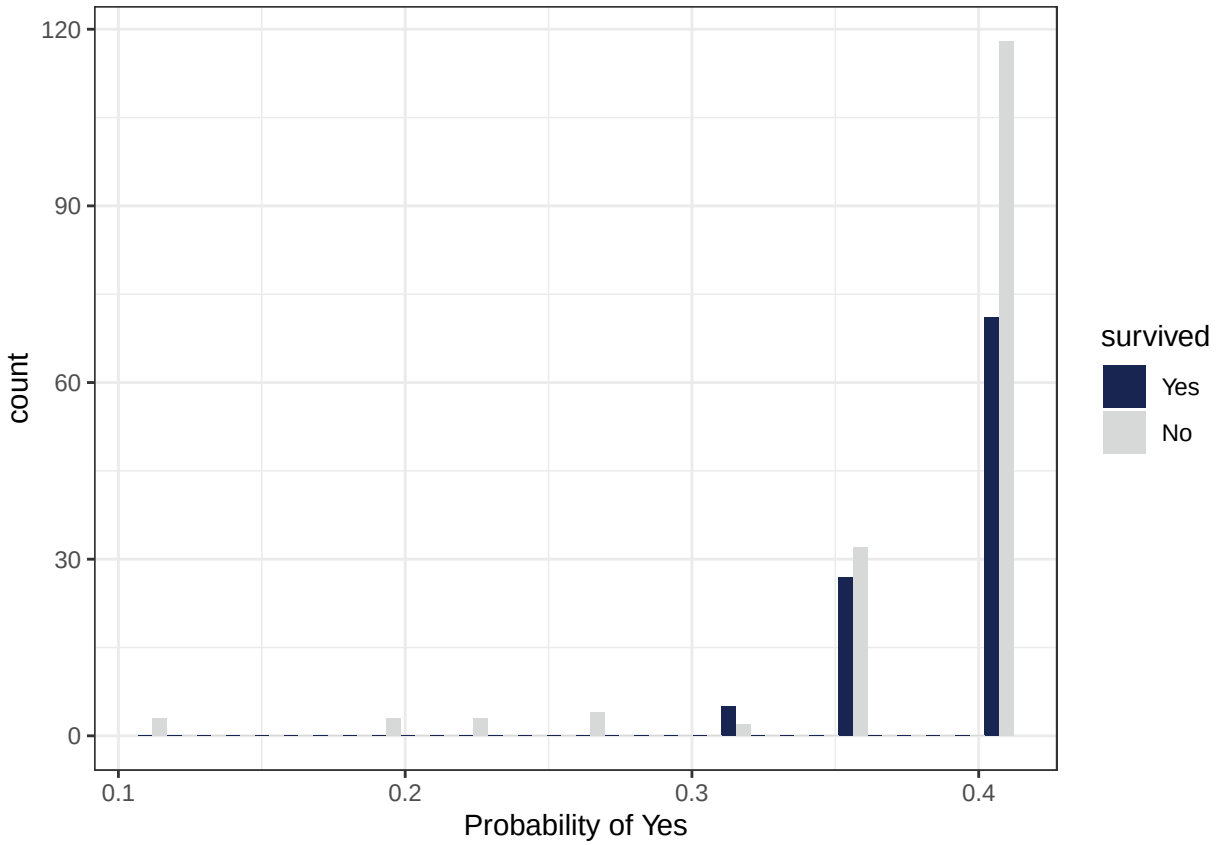
Let's look at the ROC table of threshold values:

```
log_b_results %>%
  roc_curve(survived, .pred_Yes)
```

```
## # A tibble: 9 x 3
##   .threshold specificity sensitivity
##        <dbl>       <dbl>       <dbl>
## 1   -Inf          0           1
## 2      0.113      0           1
## 3      0.193      0.0182      1
## 4      0.228      0.0364      1
## 5      0.267      0.0545      1
## 6      0.311      0.0788      1
## 7      0.358      0.0909      0.951
## 8      0.408      0.285       0.689
## 9    Inf          1           0
```

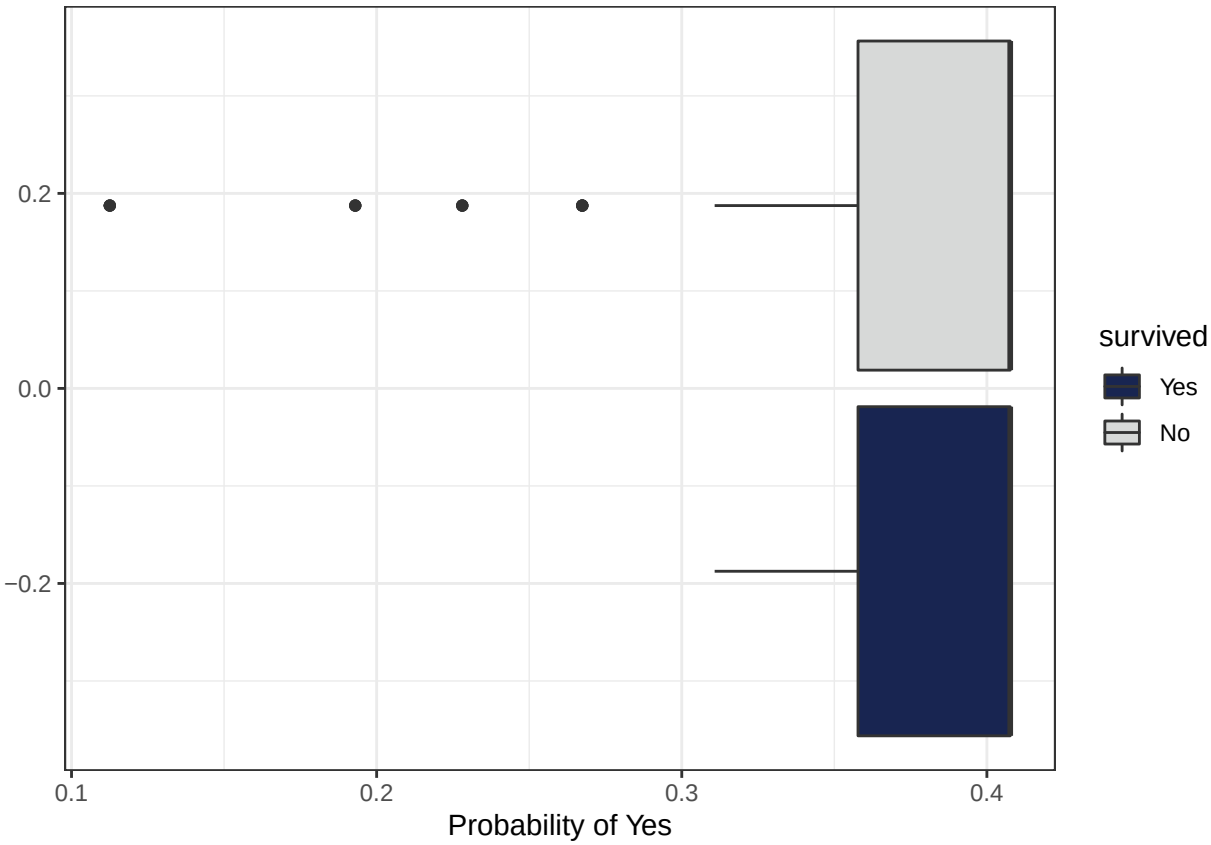
Here, the model has only tried nine potential thresholds. Why is that?

```
log_b_results %>%
  ggplot(aes(x = .pred_Yes, fill = survived)) +
  geom_histogram(position = "dodge") +
  scale_fill_manual(values = c(spec_colors[2], spec_colors[4])) + theme_bw() +
  xlab("Probability of Yes")
```



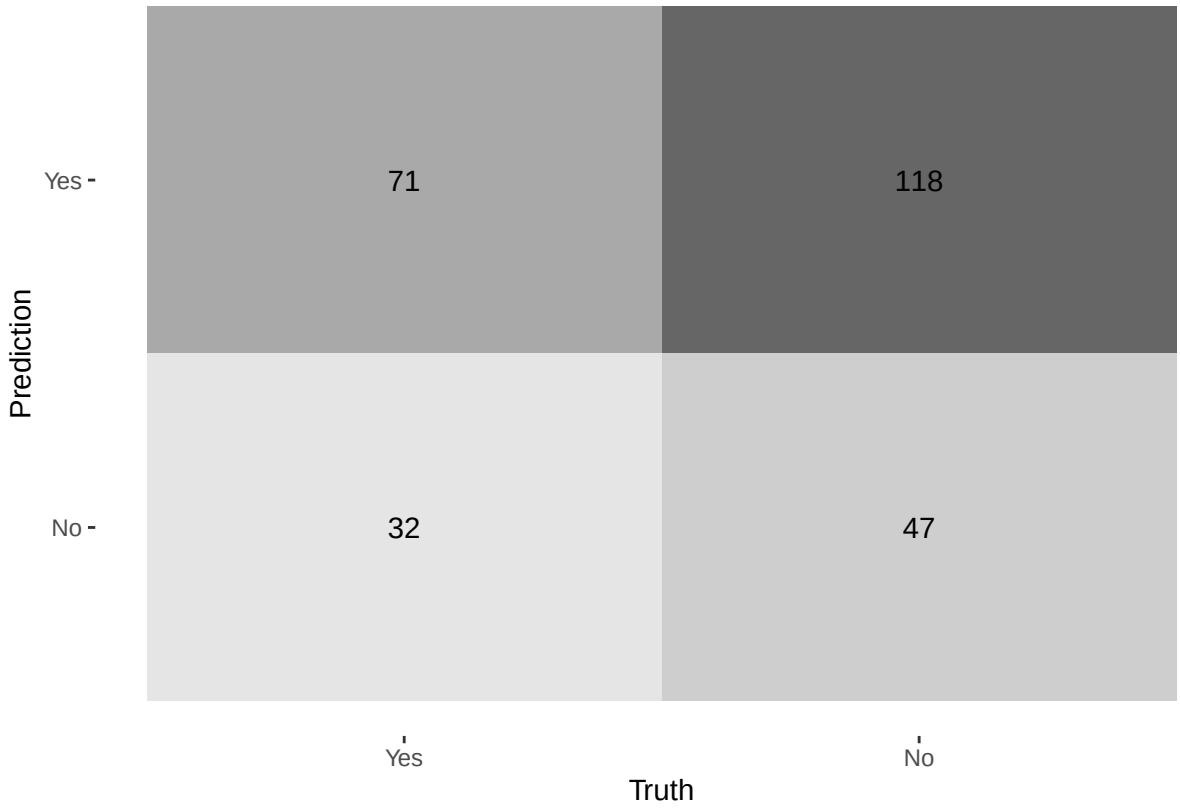
The threshold values are selected as a grid over the range of observed predicted probabilities. This model has not predicted any observation's probability of Yes as greater than 0.4076525 ... which, coincidentally, is the largest threshold value considered in its ROC curve. That's because considering any larger values for the threshold would give the same results as `Inf`. We can also view this information as a boxplot, which really serves to illustrate the overlap:

```
log_b_results %>%
  ggplot(aes(x = .pred_Yes, fill = survived)) +
  geom_boxplot() +
  scale_fill_manual(values = c(spec_colors[2], spec_colors[4])) + theme_bw() +
  xlab("Probability of Yes")
```



With that in mind, let's focus on a threshold value of, say, 0.4076525. Remember that we want to maximize sensitivity and minimize the false positive rate, or minimize 1 - specificity. At a threshold of 0.4076525, the confusion matrix looks like:

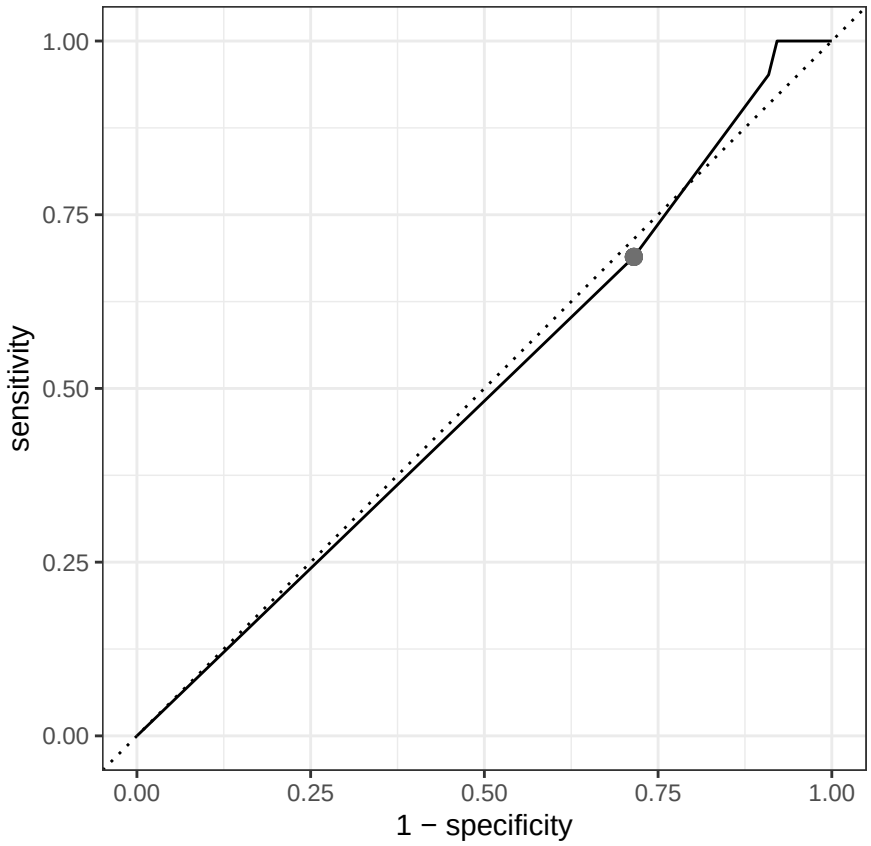
```
log_b_results %>%
  mutate(pred_class = if_else(.pred_Yes <= 0.4076524, "No", "Yes")) %>%
  select(survived, pred_class) %>%
  conf_mat(truth = survived, estimate = pred_class) %>%
  autoplot(type = "heatmap")
```



Now we're seeing some `Yes` predictions, because we adjusted our threshold for predicting `Yes` downward. The true positive rate is $\frac{71}{103} = 0.6893204$, same as we saw in the output earlier for this threshold, and the false positive rate, or 1 - specificity, is $\frac{118}{165} = 0.7151515$.

So a threshold value of 0.4076525 represents this point on the curve:

```
log_b_results %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot() +
  geom_point(aes(x = 0.7151515, y = 0.6893204), colour = spec_colors[5], size = 2.5)
```



Conclusions So what do you take away from here? First, each of these curves is unique to the model's predicted probabilities. That's why the function `roc_curve()` will throw an error if you try to give it predicted class values rather than probabilities; it needs the actual probability values in order to calculate sensitivity and specificity based on a range of thresholds.

The area under an ROC curve is, fundamentally, a measure of a model's ability to **discriminate between classes**. Model A can discriminate between 0 and 1 fairly well; higher p-values, or basically p-values above 0.50, generally correspond to observations that are positive, and lower p-values generally correspond to negative observations, with a few exceptions. Model B **cannot** discriminate between 0 and 1. The distribution of p-values is about the same for both 0s and 1s; the predicted p-value actually doesn't provide much information about whether an observation is 0 or 1, because the model doesn't know.

ROC curves have two primary uses in machine learning:

1. Evaluating and comparing model performance;
2. Selecting the optimal threshold value for a specific model.

To compare model performance, one usually compares the area under the curve for competing models, preferring the model with an AUC value closer to 1. It's also often useful to plot competing ROC curves (you can even do this on the same graph) to get a visual idea of their discriminative abilities. Remember, though, if you have a model whose AUC is **equal** to 1, that's usually a sign that you've included the outcome, or some linear combination of the outcome, as a predictor.

Once you've settled on a model, you can also use the ROC curve to determine the optimal threshold(s). This is arguably a less common use, though, because the default threshold(s) typically works well and provides a good balance between sensitivity and specificity. But if you are working on a project where you want to be extra careful to accurately predict **every** 0 or, vice versa, every 1, it can be advisable to adjust the threshold in the desired direction.

If you are working with a multiclass outcome, where there are more than two possible values, the ROC curve idea is easily extended! You can generate a curve to represent how well the model distinguishes between **each** class (we'll see examples of this in the homework and labs).