Classification via Logistic Regression, Part 2

Will Doyle

2022-07-12

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'

## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'

## Warning: package 'tibble' was built under R version 4.1.2

## Warning: package 'recipes' was built under R version 4.1.2
```

Logistic Regression

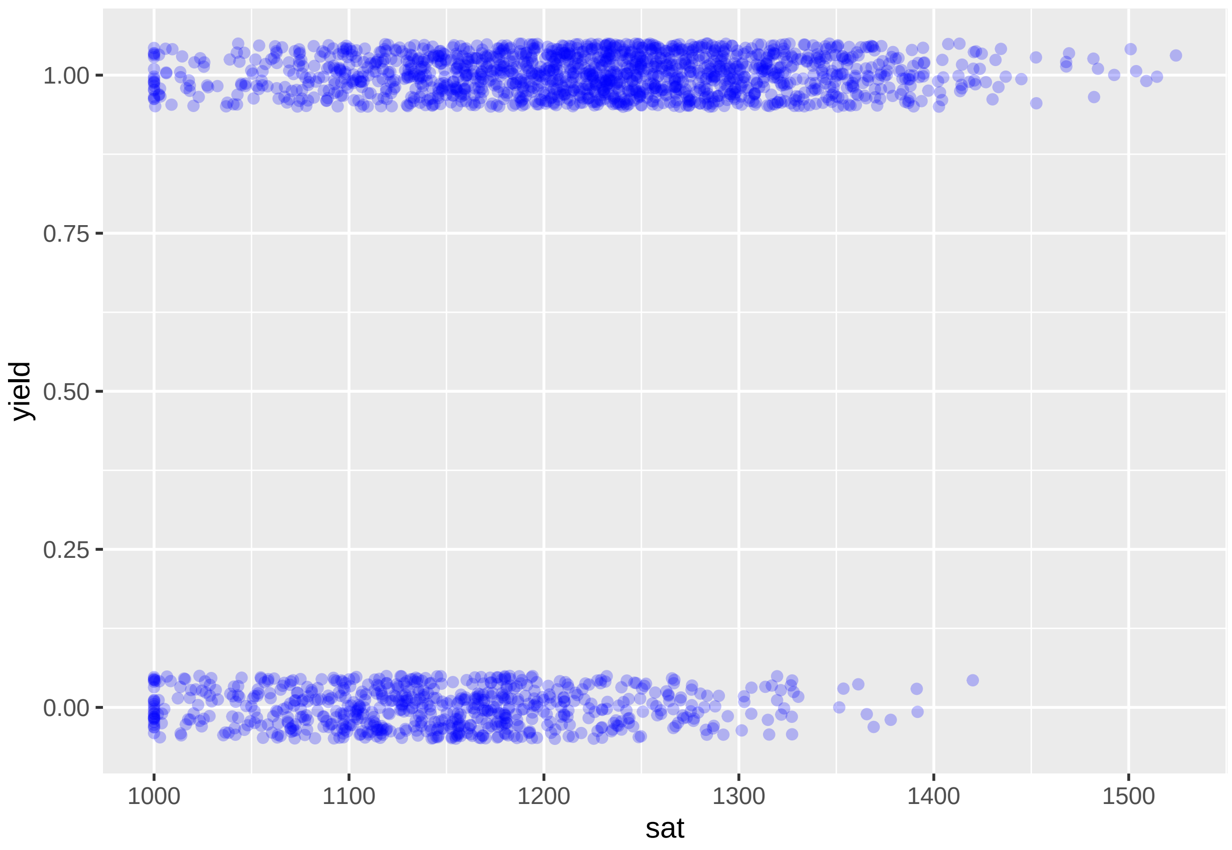
So far, we've been using tools we know for classification. While we *can* use conditional means or linear regression for classification, it's better to use a tool that was created specifically for binary outcomes.

Logistic regression is set up to handle binary outcomes as the dependent variable. The downside to logistic regression is that it is modeling the log odds of the outcome, which means all of the coefficients are expressed as log odds, which (almost) no one understands intuitively.

Let's take a look at a simple plot of our dependent variable as a function of one independent variable: SAT scores. I'm using `geom_jitter` to allow the points to "bounce" around a bit on the y axis so we can actually see them, but it's important to note that they can only be 0s or 1s.

Yield as a Function of SAT Scores

```
ad%>%
  ggplot(aes(x=sat,y=yield))+
  geom_jitter(width=.01,height=.05,alpha=.25,color="blue")
```

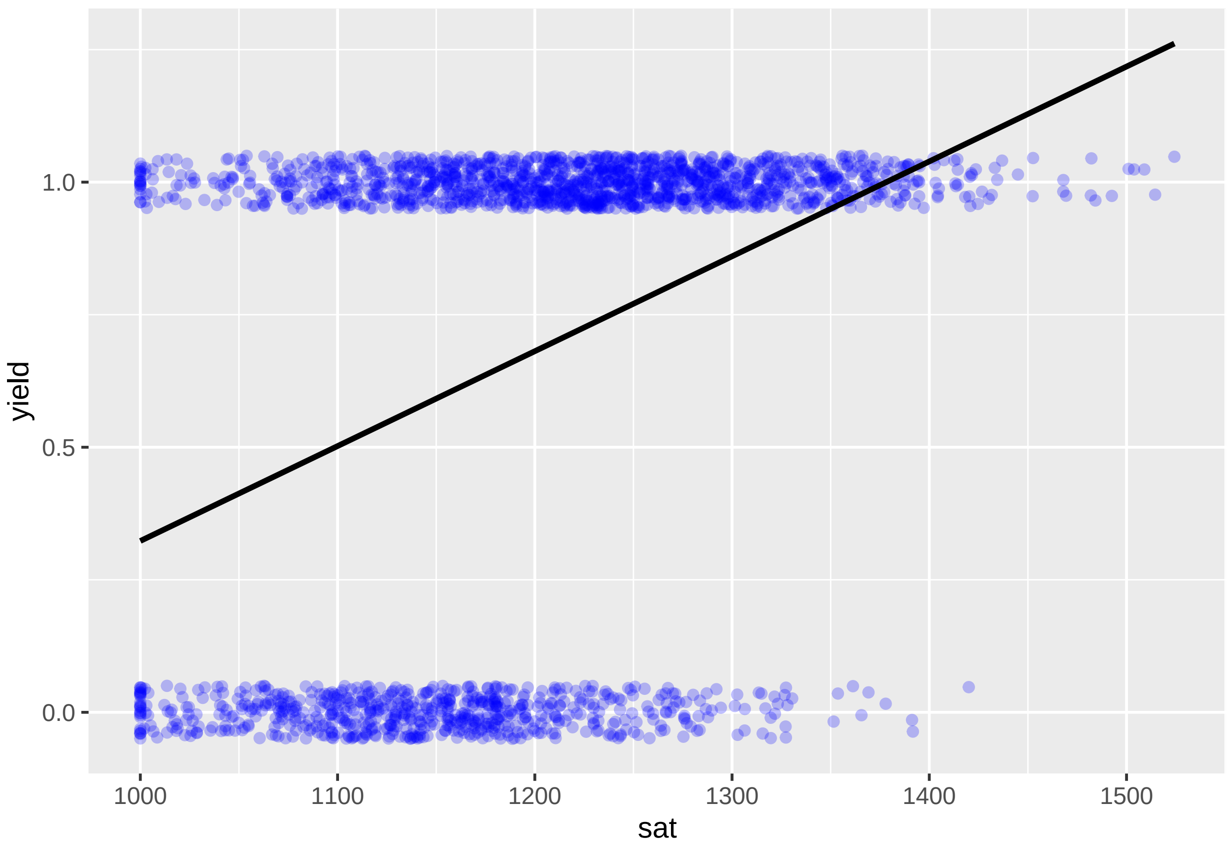


We can see there are more higher SAT students than lower SAT students that ended up enrolling. A linear model in this case would look like this:

Predicted "Probabilities" from a Linear Model

```
ad%>%
  ggplot(aes(x=sat,y=yield))+
  geom_jitter(width=.01,height=.05,alpha=.25,color="blue")+
  geom_smooth(method="lm",se = FALSE,color="black")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



We can see the issue we identified last time: we CAN fit a model, but it doesn't make a ton of sense. In particular, it doesn't follow the data very well and it ends up with probabilities outside 0,1.

Generalized Linear Models

What we need is a better function that connects y to x. The idea of connecting y to x with a function other than a simple line is called a generalized linear model.

A *Generalized Linear Model* posits that the probability that y is equal to some value is a function of the independent variables and the coefficients or other parameters via a *link function*:

$P(y|\mathbf{x}) = G(\beta_0 + \mathbf{x_i}\beta)$

In our case, we're interested in the probability that $y = 1$

$P(y = 1|\mathbf{x}) = G(\beta_0 + \mathbf{x_i}\beta)$

There are several functions that "map" onto a 0,1 continuum. The most commonly used is the logistic function, which gives us the *logit model*.

The logistic function is given by:

$f(x) = \frac{1}{1+exp^{-k(x-x_0)}}$

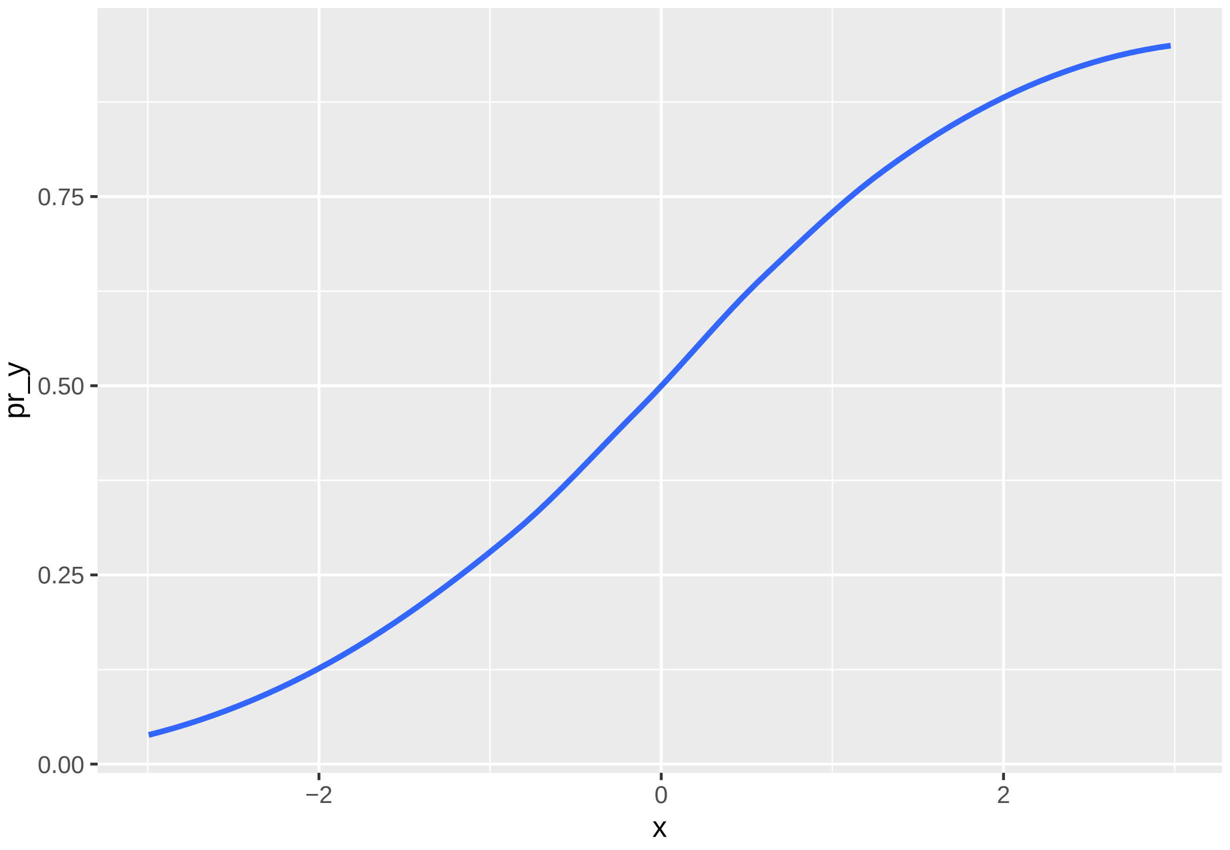
The Logistic Function: Pr(Y) as a Function of X

```
x<-runif(100,-3,3)
```

```r
pr_y=1/(1+exp(-x))

as_tibble(pr_y,x)%>%
  ggplot(aes(x=x,y=pr_y))+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Mapped onto our GLM, this gives us:

$P(y = 1|\mathbf{x}) = \frac{exp(\beta_0 + \mathbf{x_i}\beta)}{1+exp(\beta_0+\mathbf{x_i}\beta)}$

The critical thing to note about the above is that the link function maps the entire result of estimation $(\beta_0 + \mathbf{x_i}\beta)$ onto the 0,1 continuum. Thus, the change in the $P(y = 1|\mathbf{x})$ is a function of *all* of the independent variables and coefficients together, *not* one at a time.

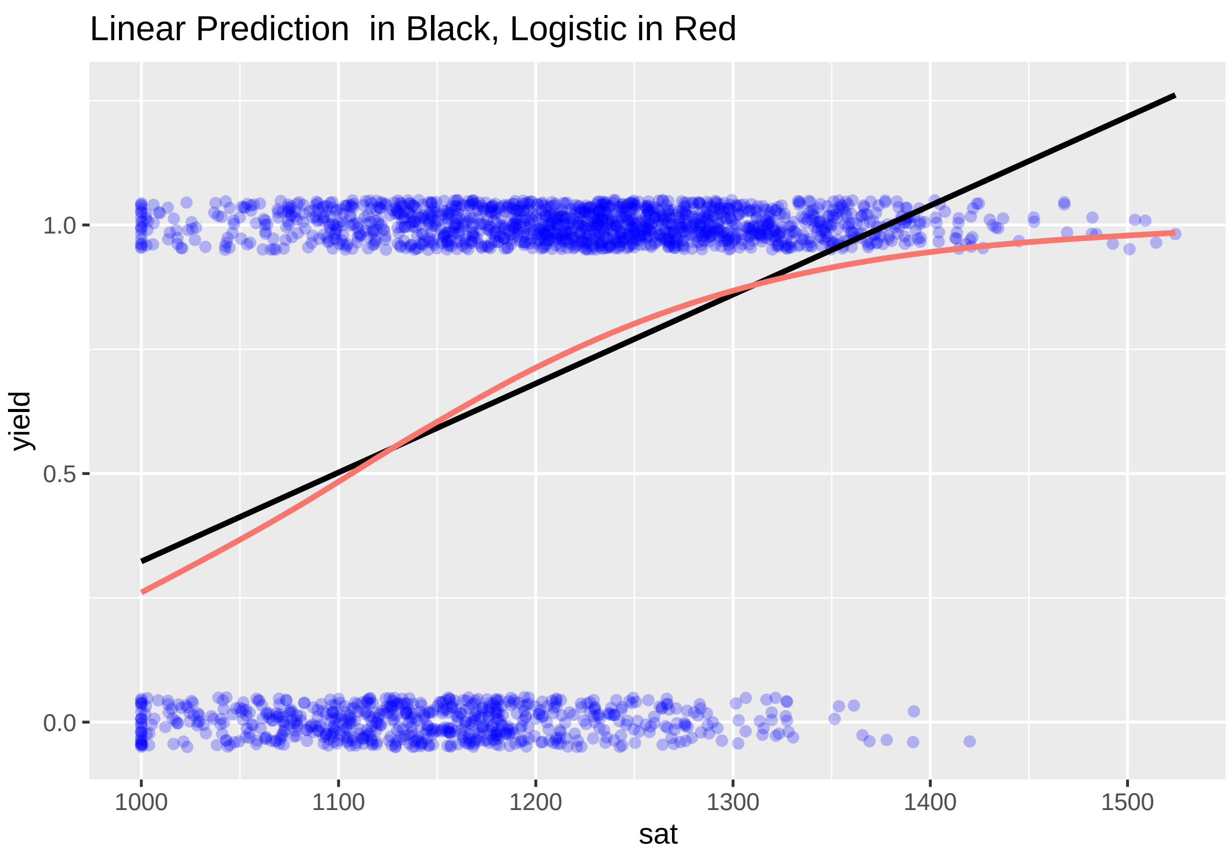
What does this mean? It means that the coefficients can only be interpreted on the *logit* scale, and don't have the normal interpretation we would use for OLS regression. Instead, to understand what the logistic regression coefficients mean, you're going to have to convert the entire term $(\beta_0 + \mathbf{x_i}\beta)$ to the probability scale, using the inverse of the function. Luckily we have computers to do this for us . . .

If we use this link function on our data, it would look like this:

Plotting Predictions from Logistic Regression

```r
glm(yield ~ sat, family = binomial(link = "logit"), data = ad) %>% ## Run a glm
  augment(type.predict = "response") %>% ## generate predictions on the prob scale
  ggplot(aes(x = sat, y = yield)) +
  geom_jitter(
    width = .01,
    height = .05,
    alpha = .25,
    color = "blue"
  ) +
  geom_smooth(method = "lm", se = FALSE, color = "black")+
  geom_smooth(aes(x=sat,y=.fitted,color="red"))+ ## Show pred probs
  theme(legend.position = "none")+
  ggtitle("Linear Prediction  in Black, Logistic in Red")
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

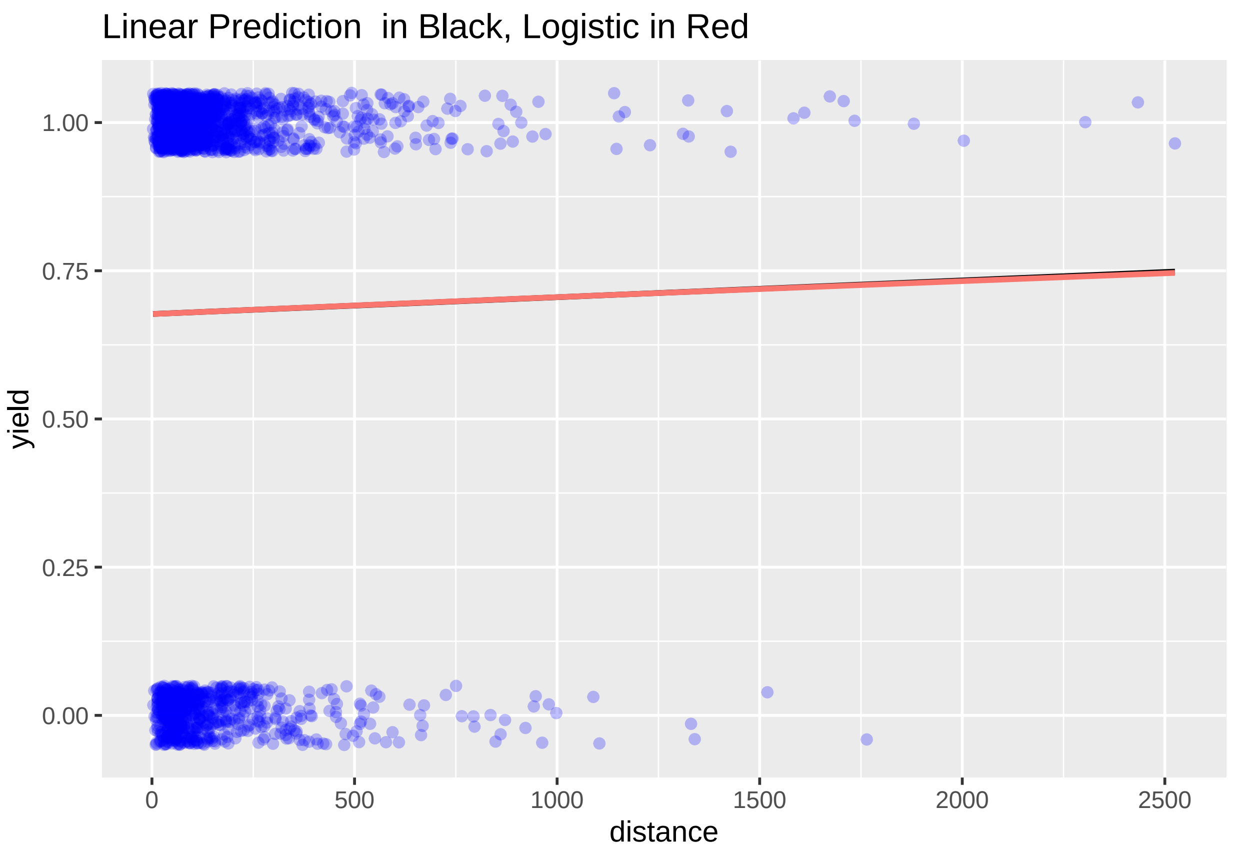


Quick Exercise: Replicate the above model using distance as a predictor and comment on what it tells you

```
glm(yield ~ distance, family = binomial(link = "logit"), data = ad) %>% ## Run a glm
  augment(type.predict = "response") %>% ## generate predictions on the prob scale
  ggplot(aes(x = distance, y = yield)) +
  geom_jitter(
    width = .01,
    height = .05,
    alpha = .25,
    color = "blue"
  ) +
  geom_smooth(method = "lm", se = FALSE, color = "black")+
  geom_smooth(aes(x=distance,y=.fitted,color="red"))+ ## Show pred probs
  theme(legend.position = "none")+
  ggtitle("Linear Prediction  in Black, Logistic in Red")
```

```
## `geom_smooth()` using formula 'y ~ x'

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



As you're getting started, this is what we recommend with these models:

- Use coefficient estimates for sign and significance only–don't try and come up with a substantive interpretation.
- Generate probability estimates based on characteristics for substantive interpretations.

Using Tidymodels for Classification

For us to run a logistic regression in the tidymodels framework, the dependent variable has to be a factor, so we need to do some . . .

Data Wrangling

```
ad<-ad%>%
  mutate(yield_f=as_factor(ifelse(yield==1,"Yes","No")))%>%
  mutate(yield_f=fct_relevel(yield_f,ref="No"))
```

The code above converts the binary variable of yield to a factor variable `yield_f`, with "Yes" as the outcome and "No" as the reference (or excluded) category.

As usual, we're going to use the "tidymodels" approach to running this model, which works much better for a standard data science workflow. It begins with splitting the data into testing and training datasets using the `initial_split` function.

Training and Testing

```
split_data<-initial_split(ad)

ad_train<-training(split_data)

ad_test<-testing(split_data)
```

Settting the Model

Now, instead of `linear_reg` we need to specify `logistic_reg` as our regression. The engine will be `glm` for generalized linear model and we'll go ahead and specify that we want to do classification: predicting group membership.

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

Formula and Recipe

The formula for a logit model looks just like the formula for a linear model. We'll add that formula to the recipe in the standard way.

```
admit_formula<-as.formula("yield_f~sat+net_price+legacy+visit")

admit_recipe<-recipe(admit_formula,ad_train)
```

Workflow

Now we can add our model and our recipe to a workflow.

```
ad_wf<-workflow()%>%
  add_model(logit_mod)%>%
  add_recipe(admit_recipe)
```

Train Model

And we're ready to fit the model and add the results to our workflow.

```
ad_wf<-ad_wf%>%
  fit(ad_train)
```

We can print out the coefficients using `tidy`

```
ad_wf%>%tidy()
```

```
## # A tibble: 5 x 5
##   term         estimate  std.error statistic  p.value
##   <chr>           <dbl>      <dbl>     <dbl>    <dbl>
## 1 (Intercept) -20.8      1.31          -15.9  6.65e-57
## 2 sat           0.0169    0.00103        16.3  8.38e-60
## 3 net_price     0.0000586 0.00000547     10.7  8.98e-27
## 4 legacy        0.516     0.145           3.57 3.60e- 4
## 5 visit         0.344     0.128           2.68 7.26e- 3
```

What this is telling us is that controlling for net price, SAT scores have a negative relationship with yield, that net price also has a negative relationship with yield, and that controlling for SAT and net price, legacy has a negative relationship with yield.

Quick Exercise: add a campus visit to the model and see what the results tell you.

Understanding Model fit for Classification

We previously explored the concepts of accuracy, sensitivity and specificity.

- Accuracy represents the proportion of cases correctly classified.
- Sensitivity represents the proportion of "1s" or positive outcomes correctly classified
- Specificity represents the proportion of "0s" or negative outcomes correctly classified

All of these are dependent on our choice of threshold: at what probability do we say that something should be classified as a 1 or a 0? We many times use .5 as a "default" threshold, but there's nothing mathematically proven that says that .5 is a correct threshold.

Accuracy

Remember that accuracy must always be evaluated relative to the baseline rate. Let's check our baseline:

```
ad_test%>%summarize(mean(yield))
```

```
##   mean(yield)
## 1   0.6821561
```

So, by simply guessing "Yes" for everyone, we could get a an accuracy of .7 or so. Let's check and see how our model is doing:

```
ad_wf%>%
  predict(ad_test)%>%
  bind_cols(ad_test)%>%
  accuracy(truth=yield_f,estimate=.pred_class,event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.764
```

Okay, so baseline is about .7 while our model has an overall accuracy of .78: better!

Sensitivity

Sensitivity is the proportion of positive outcomes correctly predicted. For us it's the proportion of enrolled students correctly predicted.

```
ad_wf%>%
  predict(ad_test)%>%
  bind_cols(ad_test)%>%
 sens(truth=yield_f,estimate=.pred_class,event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.918
```

So, out of all of the enrolled students, our model correctly predicted 93 percent.

Question: What would happen to our sensitivity if we moved the threshold closer to 1?

Specificity

Specificity is the probability of correctly classifying negative responses, in our case the probability that the model correctly classifies those who do NOT enroll.

```
ad_wf%>%
  predict(ad_test)%>%
  bind_cols(ad_test)%>%
 spec(truth=yield_f,estimate=.pred_class,event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.433
```

This means that our model only correctly classified 42 percent of those who did not enroll.

Question: What would happen to our specificity if we moved the threshold closer to 1?

At this point, our overall judgment on the predictions from this model is that it's pretty good at classifying those who enroll, and not so good at classifying those who do not enroll. It's overall more accurate than the baseline, but we could probably do better.

Varying Thresholds

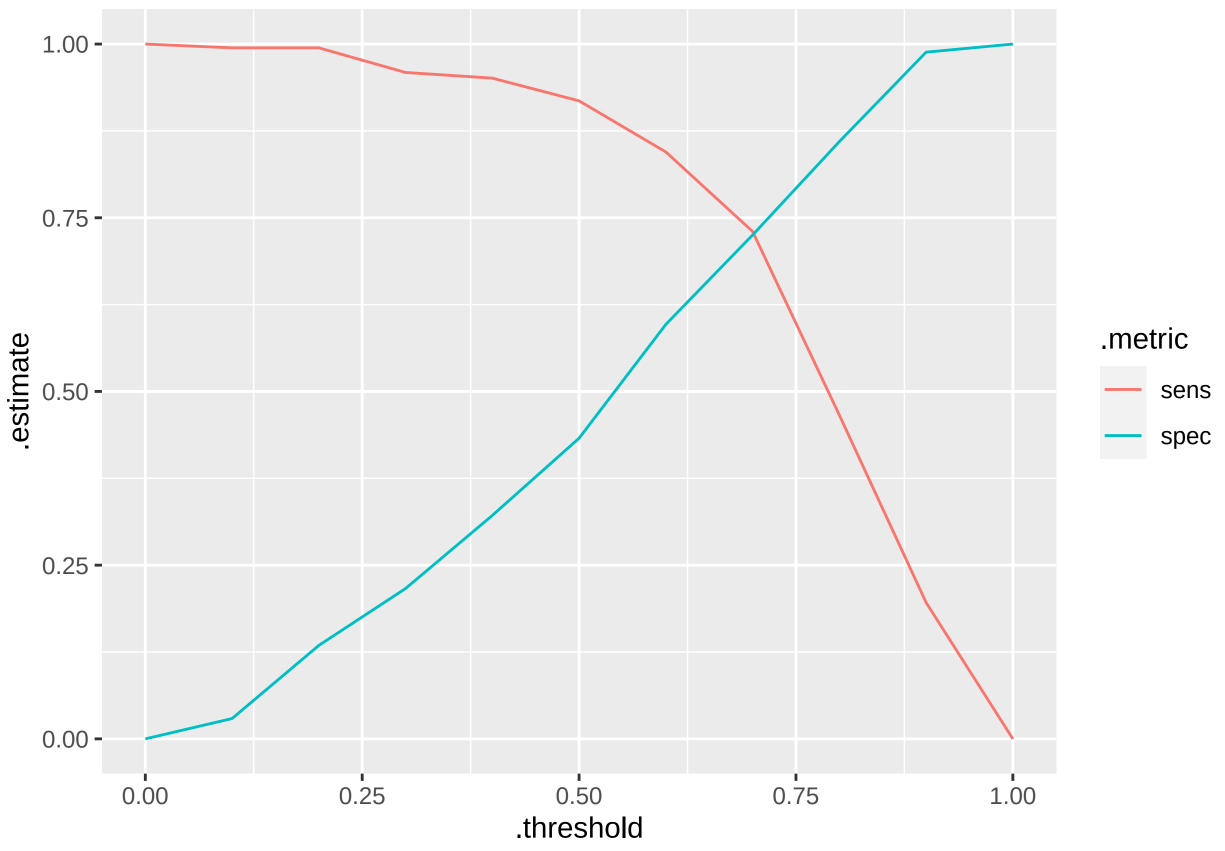
As mentioned before, the calculated sensitivity and specificity depend on the threshold we use. We can vary this threshold to get a sense of what difference it might make. Below. I use the `threshold_perf` command to vary the threshold from 0 to 1, calculating both sensitivity and specificity at each threshold.

```
options(yardstick.event_first = FALSE)

threshold_example<-ad_wf%>%
  predict(ad_test,type="prob")%>%
  bind_cols(ad_test)%>%
   threshold_perf(truth=yield_f,
                 estimate=.pred_Yes,
               thresholds=seq(0,1,by=.1),
               metrics=c("sens","spec"))
```

```
## Warning: The `yardstick.event_first` option has been deprecated as of yardstick 0.0.7 and will be co
## Instead, set the following argument directly in the metric function:
## `options(yardstick.event_first = TRUE)`  -> `event_level = 'first'` (the default)
## `options(yardstick.event_first = FALSE)` -> `event_level = 'second'`
## This warning is displayed once per session.
```

```
ggplot(filter(threshold_example,.metric%in%c("sens","spec")),
       aes(x=.threshold,y=.estimate,color=.metric))+
  geom_line()
```

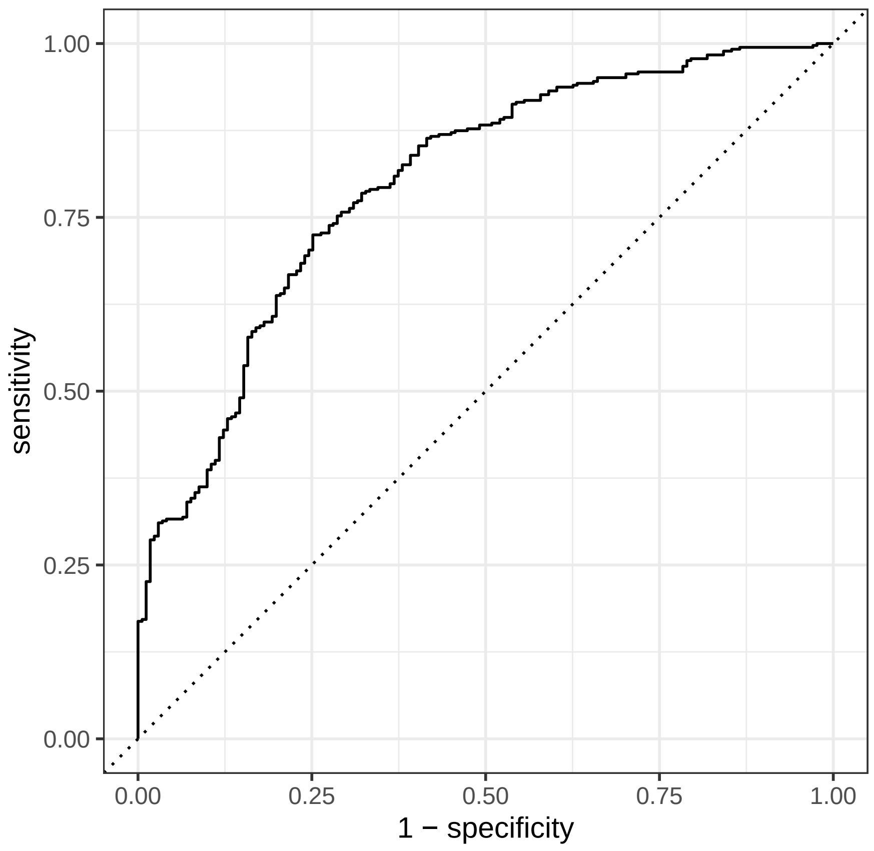


What this shows is that the specificity increases fairly steadily with higher thresholds, while the sensitivity decreases rapidly with higher thresholds above .5. A threshold of about .7 would likely give the best balance between specificity and sensitivity.

Receiver-Operator Characteristic (ROC) Curve

The Receiver-Operator Characteristic (ROC) curve shows the combination of sensitivity and specificity at a large number (typically 100) different thresholds. It's plotted with the sensitivity on the y axis and 1-specificity on the x axis. The idea is that a model that is very predictive will have high levels of sensitivity AND high levels of specificity at EVERY threshold. Such a model will cover most of the available area above the baseline of .5. A model with low levels of sensitivity and low levels of specificity at every threshold will cover almost none of the available area above the baseline of .5. Here's a drawing of the ROC curve for our predictions:

```
ad_wf%>%
  predict(ad_test,type="prob")%>%
  bind_cols(ad_test)%>%
  roc_curve(truth=yield_f,.estimate=.pred_Yes,event_level="second")%>%
  autoplot()
```



And here's the proportion of the area in the graph (AUC) covered by the curve drawn above:

```r
ad_wf%>%
  predict(ad_test,type="prob")%>%
  bind_cols(ad_test)%>%
  roc_auc(truth=yield_f,estimate=.pred_Yes,event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.798
```

The nice thing is that interpreting AUC is just like interpreting academic grades:

Below .6= bad (F)

.6-.7= still not great (D)

.7-.8= Ok . .. (C)

.8-.9= Pretty good (B)

.9-1= Very good fit (A)

Quick Exercise: Rerun the model with sent_scores added: does it improve model fit?

```r
admit_formula<-as.formula("yield_f~sat+net_price+legacy+visit+sent_scores")

admit_recipe<-recipe(admit_formula,ad_train)
```

```
ad_wf<-workflow()%>%
  add_model(logit_mod)%>%
  add_recipe(admit_recipe)
```

Train Model

And we're ready to fit the model and add the results to our workflow.

```
ad_wf<-ad_wf%>%
  fit(ad_train)
```

```
ad_wf%>%
  predict(ad_test,type="prob")%>%
  bind_cols(ad_test)%>%
  roc_auc(truth=yield_f,estimate=.pred_Yes,event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.798
```