College Admissions, Part 1

Homework

Prof. Bisbee

Due Date: 2024-03-19 & 03-21

College Admissions: From the College's View

All of you have quite recently gone through the stressful process of figuring out which college to attend. You most likely selected colleges you thought might be a good fit, sent off applications, heard back from them, and then weighed your options. Those around you probably emphasized what an important decision this is for you and for your future.

Colleges see this process from an entirely different point of view. A college needs students to enroll first of all in order to collect enough tuition revenues in order to keep the lights on and the faculty paid. Almost all private colleges receive most of their revenues from tuition, and public colleges receive about equal amounts of funding from tuition and state funds, with state funds based on how many students they enroll. Second, colleges want to enroll certain types of students– colleges based their reputation based on which students enroll, with greater prestige associated with enrolling students with better demonstrated academic qualifications. The job of enrolling a class that provides enough revenue AND has certain characteristics falls to the Enrollment Management office on a campus. This office typically includes the admissions office as well as the financial aid office.

The College Admissions "Funnel"

The admissions funnel (https://www.curacubby.com/blog/admissions-funnel) is a well-established metaphor for understanding the enrollment process from the college's perspective. It begins with colleges identifying prospective students: those who might be interested in enrolling. This proceeds to "interested" students, who engage with the college via registering on the college website, sending test scores, visiting campus, or requesting other information. Some portion of these interested students will then apply. Applicants are then considered, and admissions decisions are made. From this group of admitted students a certain proportion will actually enroll. Here's live data from UC Santa Cruz (go Banana Slugs!) on their enrollment funnel (https://iraps.ucsc.edu/iraps-public-dashboards/student-demand/admissions-funnel.html).

Each stage in this process involves modeling and prediction: how can we predict which prospective students will end up being interested students? How many interested students will turn into applicants? And, most importantly, how many admitted students will actually show up in the fall?

Colleges aren't powerless in this process. Instead, they execute a careful strategy to intervene at each stage to get both the number and type of students they want to convert to the next stage. These are the core functions of enrollment management. Why did you receive so many emails, brochures and maybe even text messages? Some model somewhere said that the intervention could convert you from a prospect to an interest, or from an interest to an applicant.

We're going to focus on the very last step: from admitted students to what's called a yield: a student who actually shows up and sits down for classes in the fall.

The stakes are large: if too few students show up, then the institutions will not have enough revenues to operate. If too many show up the institution will not have capacity for all of them. On top of this, enrollment managers are also tasked with the combined goals of increasing academic prestige (usually through test scores and GPA) and increasing the socioeconomic diversity of the entering class. As we'll see, these are not easy tasks.

The Data

We're going to be using a dataset that was constructed to resemble a typical admissions dataset. To be clear: this is not real data, but instead is based on the relationships we see in actual datasets. Using real data in this case would be a violation of privacy.

```
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'purrr' was built under R version 4.3.3
```

```
# library(tidymodels)
library(scales)
```

```
## Warning: package 'scales' was built under R version 4.3.3
```

```
ad<-read_rds("https://github.com/jbisbee1/DS1000_S2024/raw/main/data/admit_data.rds")%>%ungroup
()
```

Codebook for `admit_data.rds` :

Variable Name	Description
ID	Student id
income	Family income (AGI)
sat	SAT/ACT score (ACT scores converted to SAT scale)
gpa	HS GPA, four point scale
visit	Did student visit campus?
legacy	Did student parent go to this college?
registered	Did student register on the website?
sent_scores	Did student send scores prior to applying?
distance	Distance from student home address to campus
tuition	Stated tuition: $45,000
need_aid	Need-based aid offered
merit_aid	Merit-based aid offered

net_price	Net Price: Tuition less aid received
yield	Student enrolled in classes in fall after admission

Logistic Regression

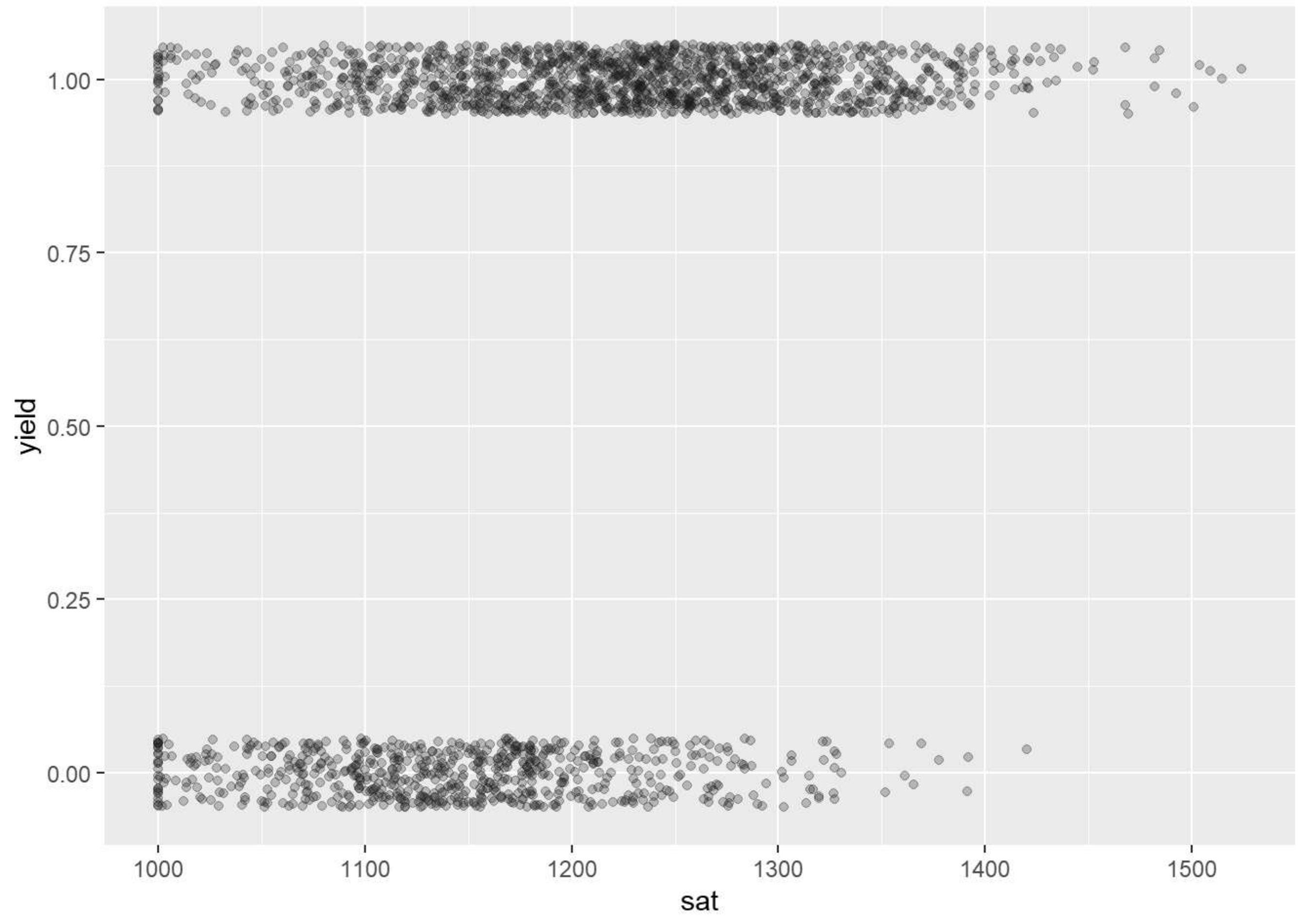
So far, we've been using tools we know for classification. While we *can* use conditional means or linear regression for classification, it's better to use a tool that was created specifically for binary outcomes.

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")
```

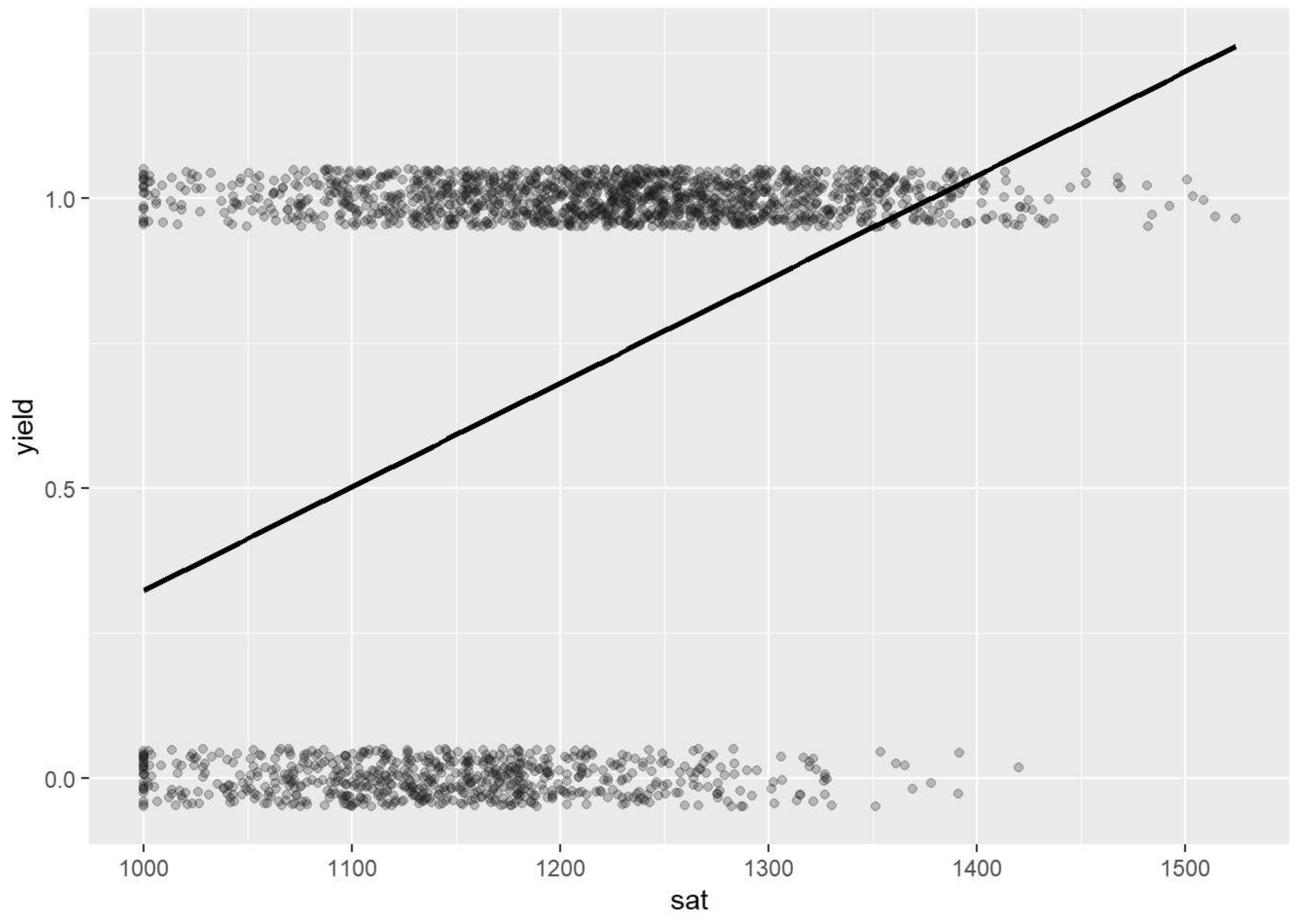


We can see there are more higher SAT students than lower SAT students that ended up enrolling. A linear model in this case would look like this:

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 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*.

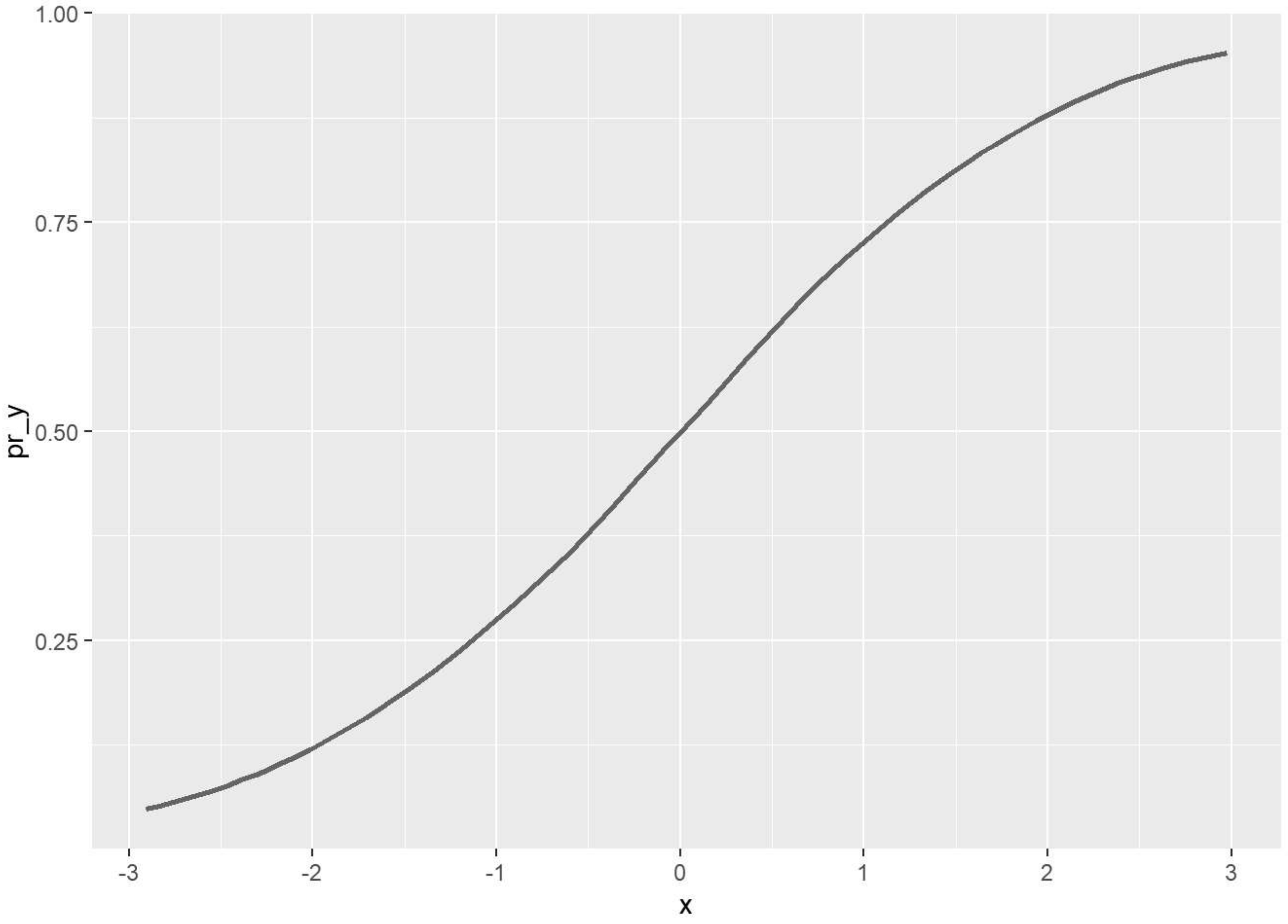
The logistic function is given by:

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

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

```
x<-runif(100,-3,3)
pr_y=1/(1+exp(-x))
as_tibble(pr_y = pr_y,x = 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, 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: `glm(formula,family,data)` . Notice that it looks very similar to the linear regression function: `lm(formula,data)` . The only difference is the **name** of the function (`glm()` versus `lm()`) and the additional input `family` . This input can take on many different values, but for this class, we only want the **logit**, which requires `family = binomial(link = "logit")` .

Putting it all together:

Plotting Predictions from Logistic Regression

```
ad_analysis <- ad %>%
  ungroup() %>%
  select(yield,sat,net_price,legacy) %>%
  drop_na()

m <- glm(yield ~ sat, family = binomial(link = "logit"), data = ad_analysis)# %>% ## Run a glm
summary(m)
```

```
##
## Call:
## glm(formula = yield ~ sat, family = binomial(link = "logit"),
##     data = ad_analysis)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.077e+01  6.965e-01  -15.46   <2e-16 ***
## sat          9.730e-03  5.926e-04   16.42   <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: 2689.5  on 2149  degrees of freedom
## Residual deviance: 2353.7  on 2148  degrees of freedom
## AIC: 2357.7
##
## Number of Fisher Scoring iterations: 4
```

How to interpret? It is more complicated than a linear regression model, and beyond what you are expected to know in an intro to data science class. We **cannot** say that each additional SAT score point corresponds to a 0.000973 increase in `yield` . However, we can conclude that there is a (1) positive and (2) statistically significant association between SAT scores and attending.

In this class…just focus on the **sign** of the coefficient and the **p-value**.

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

```
## INSERT CODE HERE
```

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.

Evaluating

Since the outcome is binary, we want to evaluate our model on the basis of **sensitivity**, **specificity**, and **accuracy**. To get started, let's generate predictions again.

NOTE: when predicting a `glm()` model, set `type = "response"` !

```
m <- glm(yield ~ sat, family = binomial(link = "logit"), data = ad_analysis)# %>% ## Run a glm
summary(m)
```

```
##
## Call:
## glm(formula = yield ~ sat, family = binomial(link = "logit"),
##     data = ad_analysis)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.077e+01  6.965e-01  -15.46   <2e-16 ***
## sat          9.730e-03  5.926e-04   16.42   <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: 2689.5  on 2149  degrees of freedom
## Residual deviance: 2353.7  on 2148  degrees of freedom
## AIC: 2357.7
##
## Number of Fisher Scoring iterations: 4
```

```
ad_analysis%>%
  mutate(preds = predict(m,type = 'response')) %>% # Predicting our new model
  mutate(pred_attend = ifelse(preds > .5,1,0)) %>% # Converting predicted probabilities into 1s
and 0s
  group_by(yield)%>%
  mutate(total_attend=n())%>%
  group_by(yield,pred_attend)%>%
  summarize(n(),`Actual Group`=mean(total_attend))%>%
  mutate(Proportion=`n()`/`Actual Group`)%>%
  rename(`Actually Attended`=yield,
         `Predicted to Attend`=pred_attend,
         `Number of Students`=`n()`)
```

```
## `summarise()` has grouped output by 'yield'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 × 5
## # Groups:   Actually Attended [2]
##   `Actually Attended` `Predicted to Attend` `Number of Students` `Actual Group`
##                 <int>                 <dbl>                <int>          <dbl>
## 1                   0                     0                  220            684
## 2                   0                     1                  464            684
## 3                   1                     0                  173           1466
## 4                   1                     1                 1293           1466
## # ℹ 1 more variable: Proportion <dbl>
```

Our **sensitivity** is 0.88 or 88%, our **specificity** is 0.32 or 32%, and our overall **accuracy** is `(220 + 1293) / 2150` or 0.70 (70%).

The Thresholds

Note that we required a “threshold” to come up with these measures of sensitivity, specificity, and accuracy. Specifically, we relied on a coin toss. If the predicted probability of attending was greater than 50%, we assumed that the student would attend, otherwise they wouldn’t. But this choice doesn’t always have to be 50%. We can choose a number of different thresholds.

```
ad_analysis%>%
  mutate(preds = predict(m,type = 'response')) %>% # Predicting our new model
  mutate(pred_attend = ifelse(preds > .35,1,0)) %>% # A lower threshold of 0.35 means that more
students are predicted to attend
  group_by(yield)%>%
  mutate(total_attend=n())%>%
  group_by(yield,pred_attend)%>%
  summarize(n(),`Actual Group`=mean(total_attend))%>%
  mutate(Proportion=`n()`/`Actual Group`)%>%
  rename(`Actually Attended`=yield,
         `Predicted to Attend`=pred_attend,
         `Number of Students`=`n()`)
```

```
## `summarise()` has grouped output by 'yield'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 × 5
## # Groups:   Actually Attended [2]
##   `Actually Attended` `Predicted to Attend` `Number of Students` `Actual Group`
##                 <int>                 <dbl>                <int>          <dbl>
## 1                   0                     0                   73            684
## 2                   0                     1                  611            684
## 3                   1                     0                   51           1466
## 4                   1                     1                 1415           1466
## # ℹ 1 more variable: Proportion <dbl>
```

```
ad_analysis%>%
  mutate(preds = predict(m,type = 'response')) %>% # Predicting our new model
  mutate(pred_attend = ifelse(preds > .75,1,0)) %>% # A higher threshold of 0.75 means that fewe
r students are predicted to attend
  group_by(yield)%>%
  mutate(total_attend=n())%>%
  group_by(yield,pred_attend)%>%
  summarize(n(),`Actual Group`=mean(total_attend))%>%
  mutate(Proportion=`n()`/`Actual Group`)%>%
  rename(`Actually Attended`=yield,
         `Predicted to Attend`=pred_attend,
         `Number of Students`=`n()`)
```

```
## `summarise()` has grouped output by 'yield'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 × 5
## # Groups:   Actually Attended [2]
##   `Actually Attended` `Predicted to Attend` `Number of Students` `Actual Group`
##                 <int>                 <dbl>                <int>          <dbl>
## 1                   0                     0                  566            684
## 2                   0                     1                  118            684
## 3                   1                     0                  661           1466
## 4                   1                     1                  805           1466
## # ℹ 1 more variable: Proportion <dbl>
```

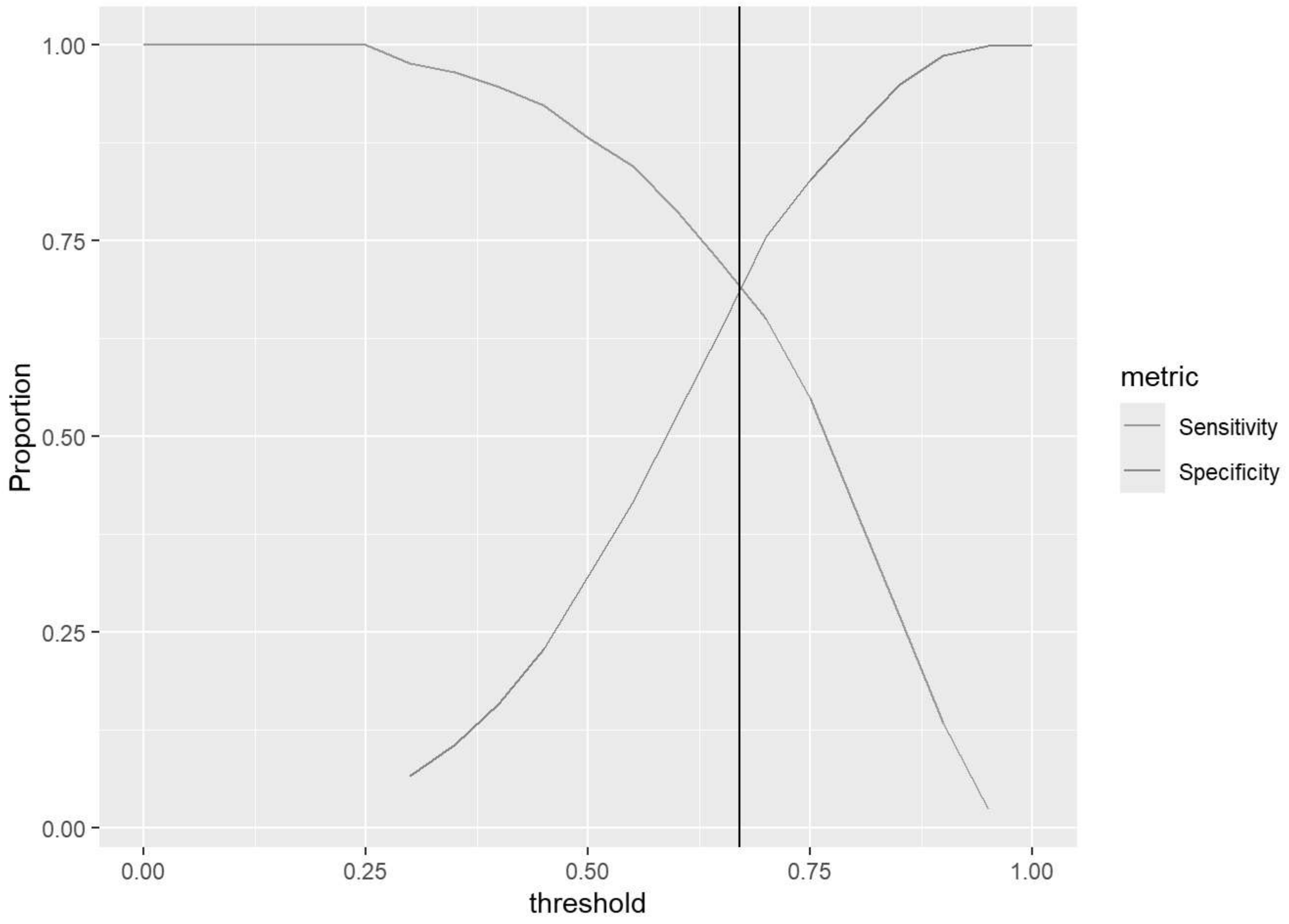
So how do we determine the optimal threshold? Loop over different choices!

```r
threshRes <- NULL

for(thresh in seq(0,1,by = .05)) { # Loop over values between zero and one, increasing by 0.05
  tmp <- ad_analysis%>%
  mutate(preds = predict(m,type = 'response')) %>% # Predicting our new model
  mutate(pred_attend = ifelse(preds > thresh,1,0)) %>% # Plug in our threshold value
  group_by(yield)%>%
  mutate(total_attend=n())%>%
  group_by(yield,pred_attend)%>%
  summarize(n(),`Actual Group`=mean(total_attend),.groups = 'drop')%>%
  mutate(Proportion=`n()`/`Actual Group`)%>%
  rename(`Actually Attended`=yield,
         `Predicted to Attend`=pred_attend,
         `Number of Students`=`n()`) %>%
    mutate(threshold = thresh)

  threshRes <- threshRes %>% bind_rows(tmp)
}

threshRes %>%
  mutate(metric = ifelse(`Actually Attended` == 1 & `Predicted to Attend` == 1,'Sensitivity',
                         ifelse(`Actually Attended` == 0 & `Predicted to Attend` == 0,'Specifici
ty',NA))) %>%
  drop_na() %>%
  ggplot(aes(x = threshold,y = Proportion,color = metric)) +
  geom_line() +
  geom_vline(xintercept = .67)
```



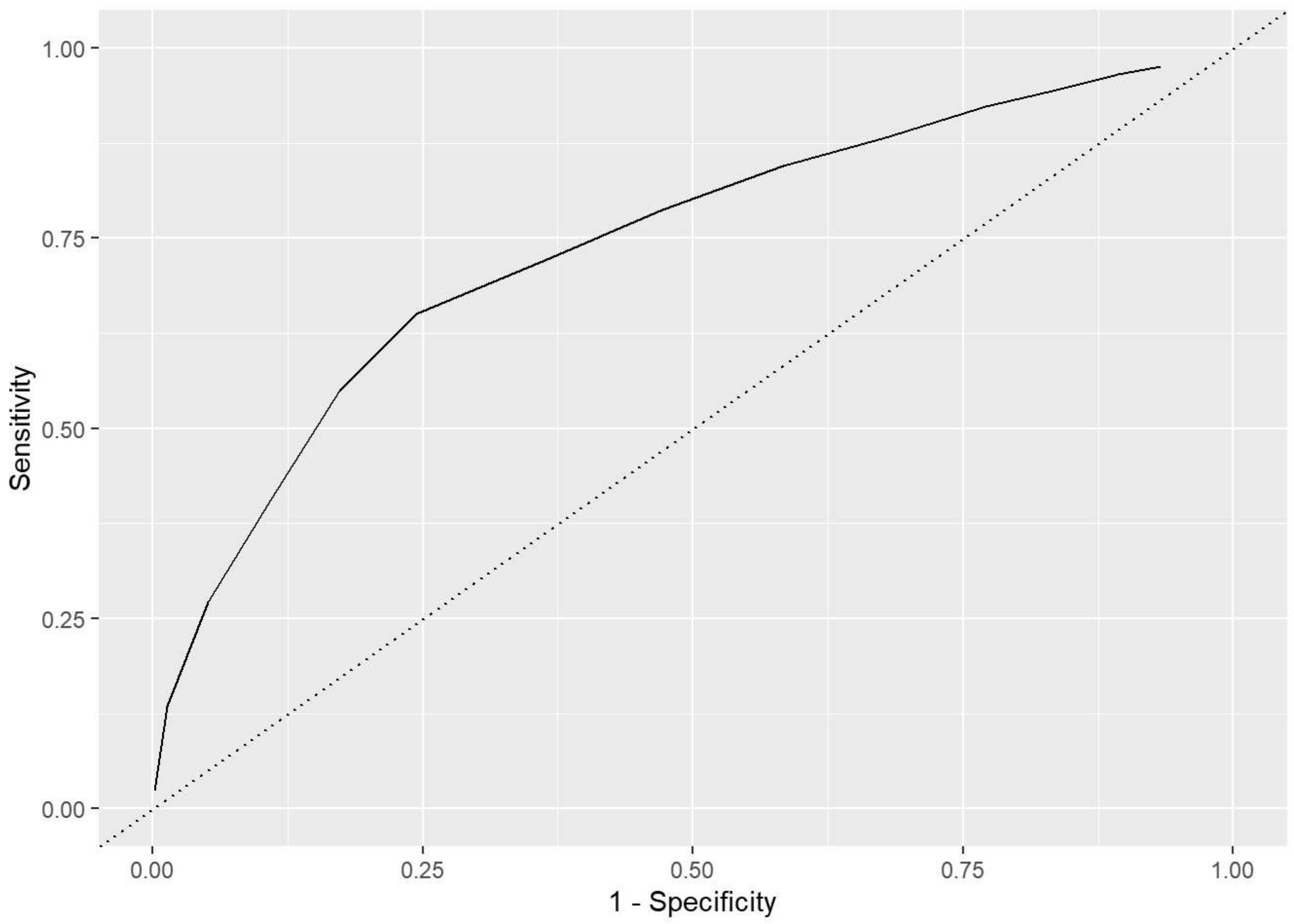
The optimal threshold is the one that maximizes Sensitivity and Specificity! (Although this is use-case dependent. You might prefer to do better on accurately predicting those who do attend than you do about predicting those who don't.) In this case it is about 0.67.

The ROC curve

As the preceding plot makes clear, there is a **trade-off** between sensitivity and specificity. We can visualize this trade-off by putting `1-specificity` on the x-axis, and `sensitivity` on the y-axis, to create the "Receiver-Operator Characteristic (ROC) Curve".

```
threshRes %>%
  mutate(metric = ifelse(`Actually Attended` == 1 & `Predicted to Attend` == 1,'Sensitivity',
                         ifelse(`Actually Attended` == 0 & `Predicted to Attend` == 0,'Specifici
ty',NA))) %>%
  drop_na() %>%
  select(Proportion,metric,threshold) %>%
  spread(metric,Proportion) %>% # Create two columns, one for spec, the other for sens
  ggplot(aes(x = 1-Specificity,y = Sensitivity)) + # X-axis is 1-Specificity
  geom_line() +
  xlim(c(0,1)) + ylim(c(0,1)) +
  geom_abline(slope = 1,intercept = 0,linetype = 'dotted') # The curve is always evaluated in re
ference to the diagonal line.
```

```
## Warning: Removed 7 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



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.

As such, we can extract a single number that captures the quality of our model from this plot: the “Area Under the Curve” (AUC). The further away from the diagonal line is our ROC curve, the better our model performs, and the higher is the AUC. But how to calculate the AUC? Thankfully, there is a helpful `R` package that will do this for us: `tidymodels`.

```
require(tidymodels)
```

```
## Loading required package: tidymodels
```

```
## Warning: package 'tidymodels' was built under R version 4.3.3
```

```
## ── Attaching packages ──────────────────────────────────────── tidymodels 1.1.1 ──
## ✓ broom        1.0.5      ✓ rsample      1.2.0
## ✓ dials        1.2.1      ✓ tune         1.1.2
## ✓ infer        1.0.6      ✓ workflows    1.1.4
## ✓ modeldata    1.3.0      ✓ workflowsets 1.0.1
## ✓ parsnip      1.2.0      ✓ yardstick    1.3.0
## ✓ recipes      1.0.10
```

```
## Warning: package 'dials' was built under R version 4.3.3
```

```
## Warning: package 'infer' was built under R version 4.3.3
```

```
## Warning: package 'modeldata' was built under R version 4.3.3
```

```
## Warning: package 'parsnip' was built under R version 4.3.3
```

```
## Warning: package 'recipes' was built under R version 4.3.3
```

```
## Warning: package 'rsample' was built under R version 4.3.3
```

```
## Warning: package 'tune' was built under R version 4.3.3
```

```
## Warning: package 'workflows' was built under R version 4.3.3
```

```
## Warning: package 'workflowsets' was built under R version 4.3.3
```

```
## Warning: package 'yardstick' was built under R version 4.3.3
```

```
## ── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```
roc_auc(data = ad_analysis %>%
  mutate(pred_attend = predict(m,type = 'response'),
         truth = factor(yield,levels = c('1','0'))) %>% # Make sure the outcome is converted to factors with '1' first and '0' second!
  select(truth,pred_attend),truth,pred_attend)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.742
```

Our curve covers almost 75% of the total area above the diagonal line. Is this good?

Just like with RMSE, you are primarily interested in the AUC measure to compare different models against each other.

But if you HAVE to, it turns out that – in general – 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?

Cross Validation

Just like RMSE calculated on the full data risks overfitting, AUC does also.

How to overcome? Cross validation!

```
set.seed(123)
cvRes <- NULL
for(i in 1:100) {
  # Cross validation prep
  inds <- sample(1:nrow(ad_analysis),size = round(nrow(ad_analysis)*.8),replace = F)
  train <- ad_analysis %>% slice(inds)
  test <- ad_analysis %>% slice(-inds)

  # Training models
  m1 <- glm(yield ~ sat,family = binomial(link = "logit"),train)

  # Predicting models
  toEval <- test %>%
    mutate(m1Preds = predict(m1,newdata = test,type = 'response'),
           truth = factor(yield,levels = c('1','0')))

  # Evaluating models
  rocRes <- roc_auc(data = toEval,truth = truth,m1Preds)
  cvRes <- rocRes %>%
    mutate(bsInd = i) %>%
    bind_rows(cvRes)
}

mean(cvRes$.estimate)
```

```
## [1] 0.7404506
```