Classification

Part 1

Prof. Bisbee

Vanderbilt University

Slides Updated: 2024-07-11

Agenda

1. Classification

2. Fortnite gaming (i.e., Prof's desperate attempt to be relevant)

```r
require(tidyverse)
fn <-
read_rds('https://github.com/jbisbee1/ISP_Data_Science_2024/raw/main/da
```

Definitions

- *Classification:* predicting the **class** of given data points via **predictive modeling**
    - *Class*: AKA targets, labels, or **categories**
    - *Predictive Modeling*: Approximate mapping function $f : X \to Y$
    - X: predictor variables
    - Y: outcome variable
    - f: ??

Mapping Functions

- We have already used a mapping functions!
- Linear Regression
  - $f\colon Y = \alpha + \beta X + \varepsilon$
- Underlying idea: X contain information about Y

It is in the Y

- If Y is continuous, we use OLS regression
- If Y is **binary**, we use "logistic" regression (AKA "logit")
  - As always, this is a **deep** area of study for those interested
- Today, using OLS for binary Y
  - Next few classes: replacing OLS regression with logit

Fortnite

Fortnite

- Goal is to win (i.e., be the last player alive)
- Professional e-sports teams want to maximize this probability
- RQ: How can we increase the number of victories?
- NB: we are moving out of the **Research** camp now, and into the **Prediction** world
    - We don't care so much about *why* a relationship exists, we just want to get accurate predictions
    - Theory can still help us, but want to start with the data to get our thinking started

The Data

```r
glimpse(fn)
```

```
## Rows: 957
## Columns: 24
## $ placed                <dbl> 17, 41, 36, 28, 3, 15, 9, 29,…
## $ mental_state          <chr> "sober", "sober", "high", "hi…
## $ eliminations          <dbl> 2, 0, 3, 1, 3, 0, 2, 3, 4, 1,…
## $ assists               <dbl> 0, 2, 0, 4, 2, 1, 2, 2, 0, 2,…
## $ revives               <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,…
## $ accuracy              <dbl> 0.19371429, 0.32400265, 0.336…
## $ hits                  <dbl> 10, 17, 38, 22, 49, 4, 43, 14…
## $ head_shots            <dbl> 1, 0, 0, 3, 18, 3, 2, 3, 13, …
## $ distance_traveled     <dbl> 226, 370, 725, 266, 938, 148,…
## $ materials_gathered    <dbl> 0, 0, 0, 358, 305, 0, 1286, 1…
## $ materials_used        <dbl> 0, 38, 0, 61, 234, 170, 195, …
## $ damage_taken          <dbl> 282, 203, 206, 262, 437, 151,…
## $ damage_to_players     <dbl> 372, 354, 206, 286, 823, 122,…
## $ damage_to_structures  <dbl> 538, 1403, 260, 3841, 1470, 4…
## $ won                   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,…
## $ player                <int> -5, -5, -5, -5, -5, -5, -5, -…
## $ gameId                <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10…
## $ startTime             <dttm> 2020-04-10 16:46:06, 2020-04…
```

The Data

- Start with the basics:

    1. What is the unit of analysis?
    2. Which variables are we interested in?

Prediction

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \varepsilon$$

- Y: victory (`won`)
- X: ??
  - In prediction, we don't care about **theory** or **research questions**
  - Just want to maximize **accuracy**...which X's are the "best"?
  - But theory can still help us make sensible choices about which X's to use
- Look at univariate & conditional relationships

The Data

- Outcome Y: `won`

```
require(scales)
fn %>%
  summarise(`Win %` = percent(mean(won)))
```

```
## # A tibble: 1 × 1
##   `Win %`
##   <chr>
## 1 30%
```

- Multivariate analysis?

Which X?

```r
fn %>%
  group_by(mental_state) %>%
  summarise(pr_win = mean(won))
```

```
## # A tibble: 2 × 2
##   mental_state pr_win
##   <chr>         <dbl>
## 1 high          0.234
## 2 sober         0.370
```

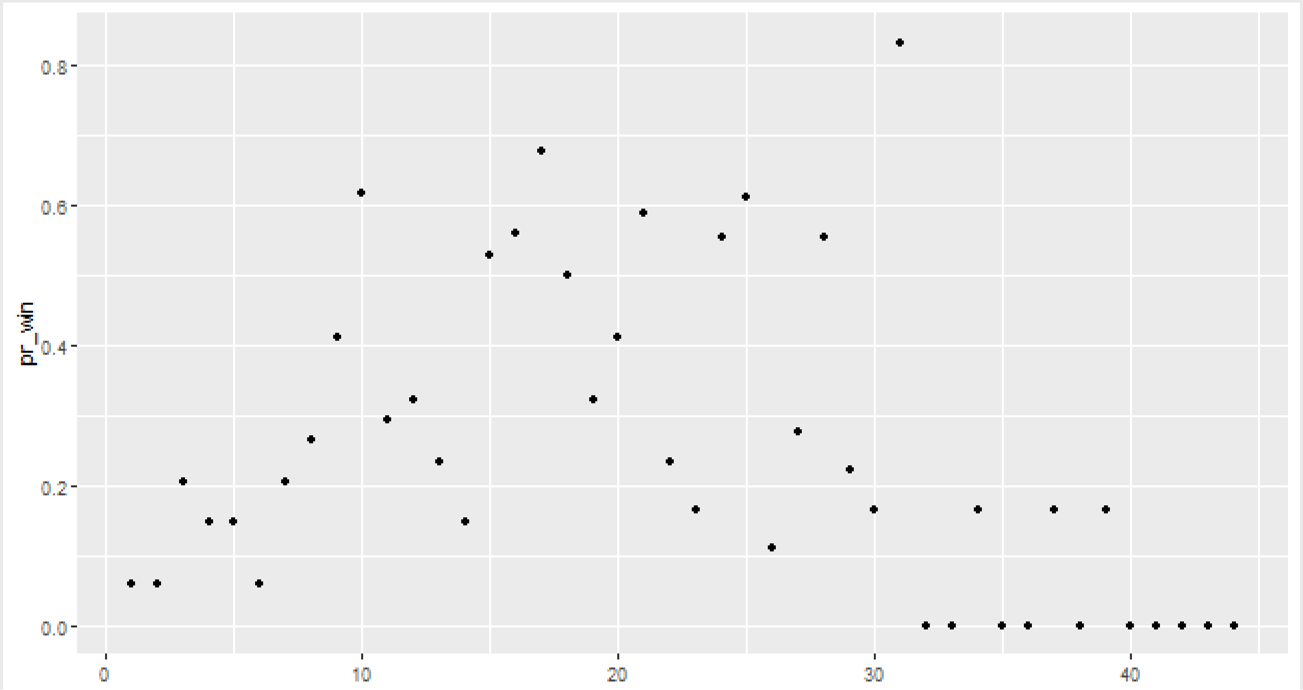
Which X?

```
fn %>%
  group_by(gameIdSession) %>%
  summarise(pr_win = mean(won))
```

```
## # A tibble: 44 × 2
##    gameIdSession pr_win
##            <int>  <dbl>
##  1             1 0.0588
##  2             2 0.0588
##  3             3 0.206
##  4             4 0.147
##  5             5 0.147
##  6             6 0.0588
##  7             7 0.206
##  8             8 0.265
##  9             9 0.412
## 10            10 0.618
## # ℹ 34 more rows
```

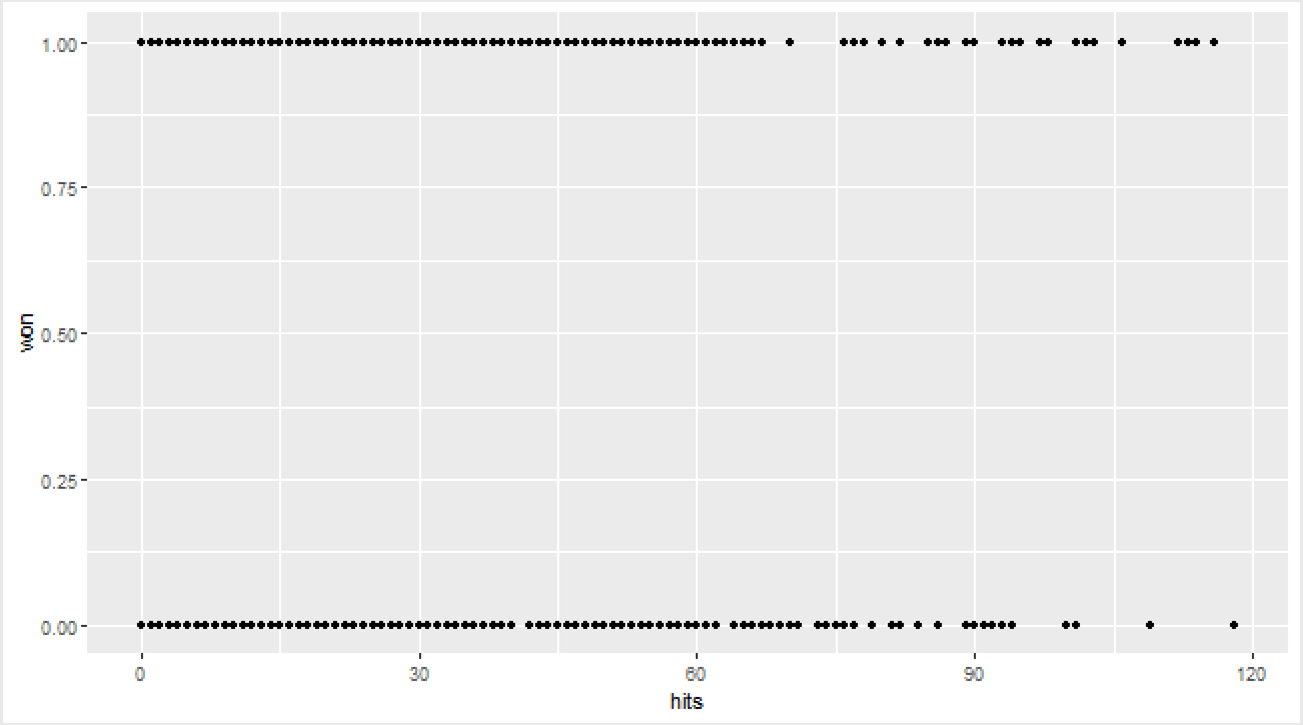
Which X?

```
fn %>%
  group_by(gameIdSession) %>%
  summarise(pr_win = mean(won)) %>%
  ggplot(aes(x = gameIdSession,
             y = pr_win)) +
  geom_point()
```



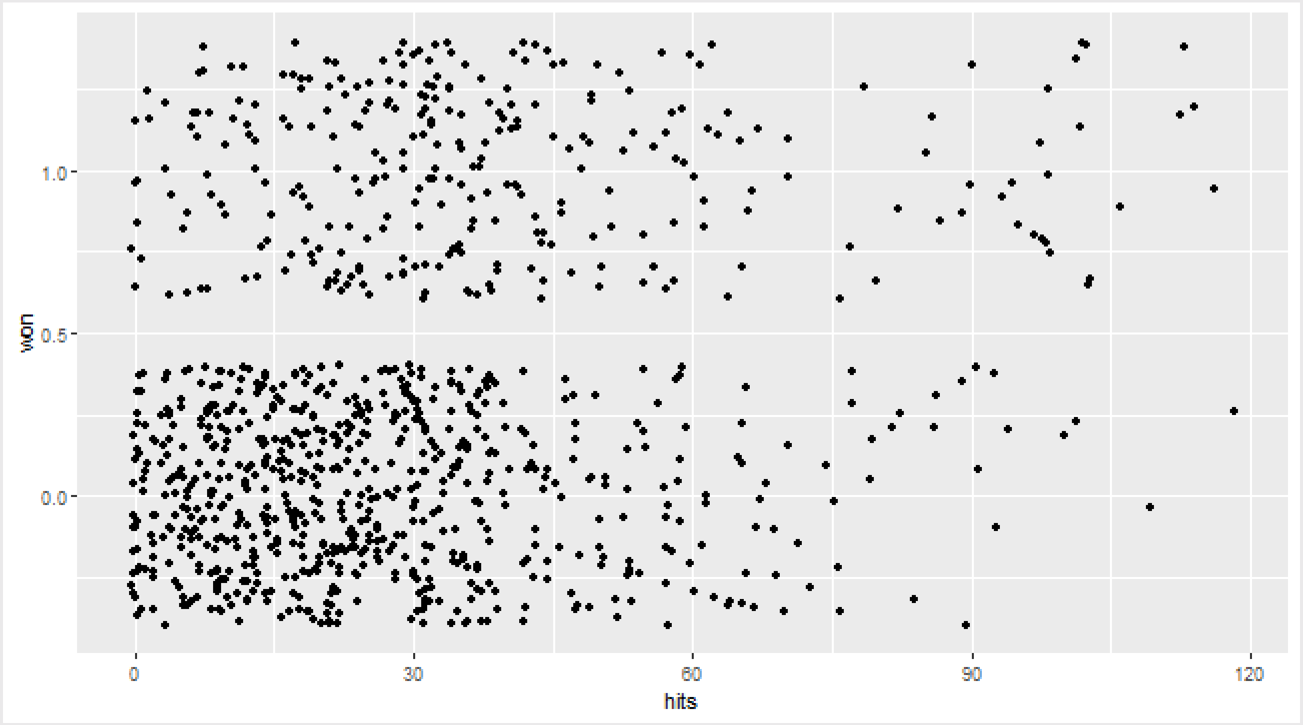
Which X?

```
fn %>%
  ggplot(aes(x = hits,y = won)) +
  geom_point()
```



Which X?

```
fn %>%
  ggplot(aes(x = hits,y = won)) +
  geom_jitter()
```



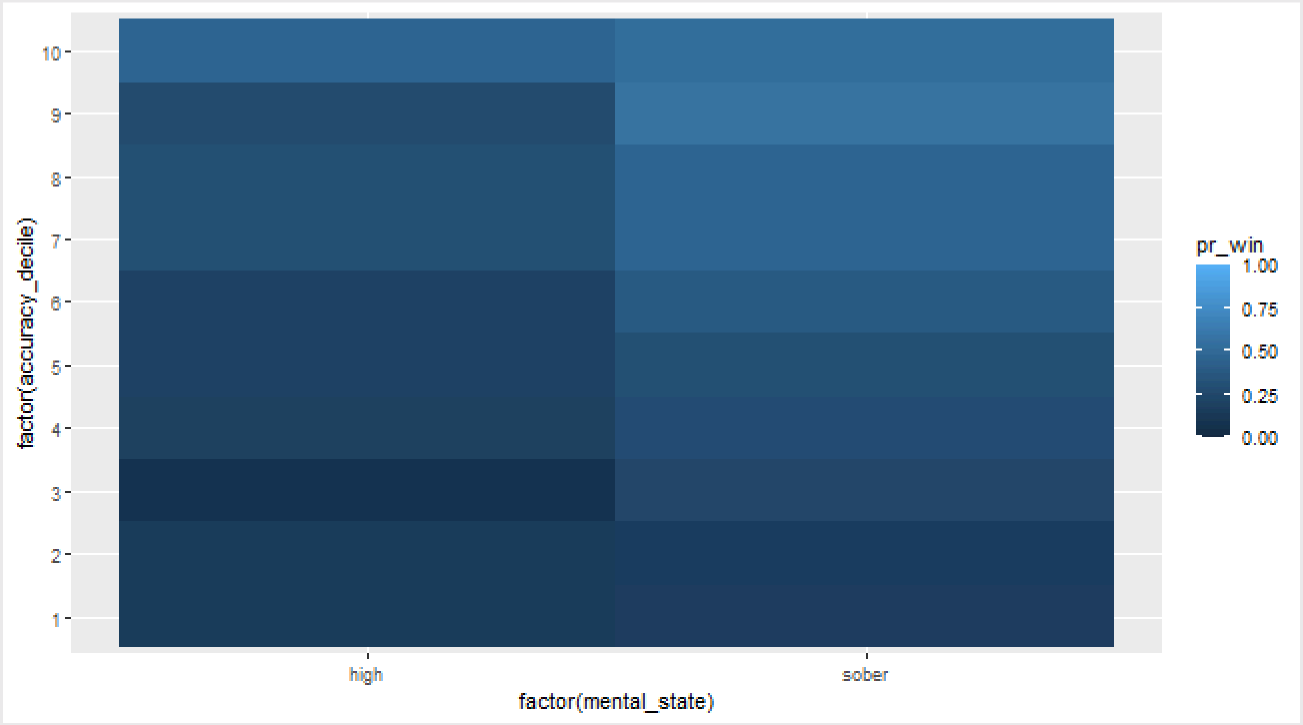
Heatmaps

- Look at 3-dimensions of data
  - Done this before by tweaking `fill`, `color`, or `size`
- `geom_tile()`: create a heatmap

```
p <- fn %>%
  mutate(accuracy_decile = ntile(hits,n=10)) %>% # Bin hits by decile
(10%)
  group_by(accuracy_decile,mental_state) %>% # Calculate average
winning by mental state and accuracy
  summarise(pr_win = mean(won),
            .groups = 'drop') %>%
  ggplot(aes(x = factor(mental_state),
             y = factor(accuracy_decile), # Both x and y-axes are
factors
             fill = pr_win)) + # Fill by third dimension
  geom_tile() + # Creates rectangles
  scale_fill_gradient(limits = c(0,1)) # Set fill color (can do much
more here)
```

Heatmaps

```
p
```



factor(accuracy_decile)
10
9
8
7
6
5
4
3
2
1
high
sober
factor(mental_state)
pr_win
1.00
0.75
0.50
0.25
0.00

Simplest Predictions

- Remember: regression is just fancier conditional means

```
fn <- fn %>%
  mutate(hits_decile = ntile(hits,n=10)) %>% # Bin hits by decile
(10%)
  group_by(hits_decile,mental_state) %>% # Calculate average winning
by mental state and accuracy
  mutate(prob_win = mean(won)) %>% # use mutate() instead of
summarise() to avoid collapsing the data
  mutate(pred_win = ifelse(prob_win > .5,1,0)) %>% # If the
probability is greater than 50-50, predict a win
  ungroup()
```

Simplest Predictions

- Conditional means

```
fn %>%
  group_by(won,pred_win) %>%
  summarise(nGames=n(),.groups = 'drop')
```

```
## # A tibble: 4 × 3
##       won pred_win nGames
##     <dbl>    <dbl>  <int>
## 1       0        0    625
## 2       0        1     41
## 3       1        0    241
## 4       1        1     50
```

- How good is this? Think about the underlying goal...we want a model that accurately predicts whether a game is won or not
- The `won` column is the **truth**...it tells us whether the game was won or not
- The `pred_win` column is our **prediction**

Accuracy

- What is "accuracy"?
  - Proportion "correct" predictions
- For a binary outcome, "accuracy" has two dimensions
  - Proportion of correct `1`s: **Sensitivity**
  - Proportion of correct `0`s: **Specificity**

Accuracy

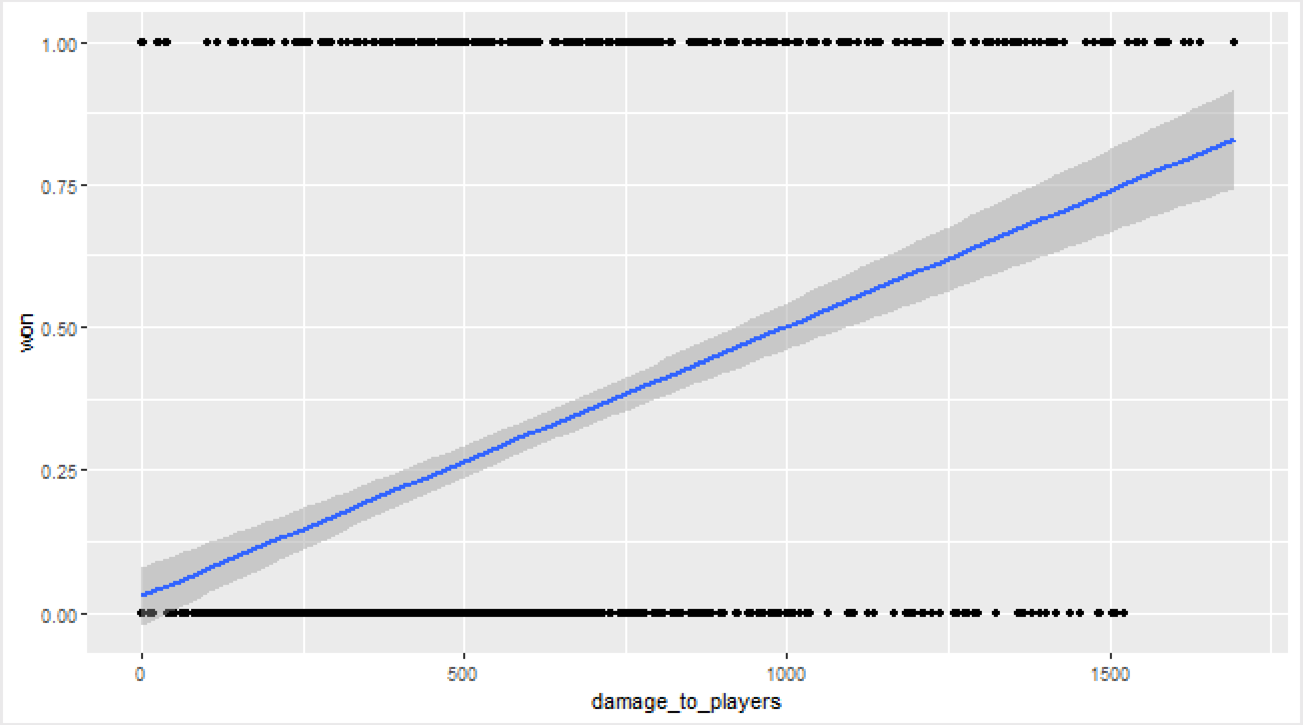
```r
(sumTab <- fn %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games))
```

```
## # A tibble: 4 × 5
##     won pred_win total_games nGames   prop
##   <dbl>    <dbl>       <int>  <int>  <dbl>
## 1     0        0         666    625 0.938
## 2     0        1         666     41 0.0616
## 3     1        0         291    241 0.828
## 4     1        1         291     50 0.172
```

- Overall accuracy: (625+50) / (666+291) = 71%
- But we are doing **great** at predicting losses (94%)...
- ...and **terribly** at predicting wins (17%)

Regression

```r
fn %>%
  ggplot(aes(x = damage_to_players,y = won)) +
  geom_point() +
  geom_smooth(method = 'lm')
```



Regression

- Binary outcome variable!
    - A linear regression is not the best solution
    - Predictions can exceed support of Y
- But it can still work! **linear probability model**

```
mLM <- lm(won ~ hits + accuracy + mental_state,fn)
```

Linear Regression

```
require(broom) # broom package makes it easy to read regression
output
```

```
## Loading required package: broom
```

```
tidy(mLM) %>% # This would be the same as summary(mLM)
  mutate_at(vars(-term),function(x) round(x,5))
```

```
## # A tibble: 4 × 5
##   term                estimate std.error statistic p.value
##   <chr>                  <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)          0.219     0.0336       6.52       0
## 2 hits                 0.00646   0.00065      9.91       0
## 3 accuracy            -0.725     0.108       -6.72       0
## 4 mental_statesober  0.155     0.0281       5.53       0
```

Linear Regression

```r
mLM <- lm(won ~ scale(hits) + scale(accuracy) + mental_state,fn)
tidy(mLM)
```

```
## # A tibble: 4 × 5
##   term              estimate std.error statistic  p.value
##   <chr>                <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)          0.224    0.0201     11.1  4.65e-27
## 2 scale(hits)          0.147    0.0148      9.91 4.14e-22
## 3 scale(accuracy)     -0.100    0.0149     -6.72 3.07e-11
## 4 mental_statesober    0.155    0.0281      5.53 4.25e- 8
```

```r
fn %>%
  summarise_at(vars(hits,accuracy),function(x) round(sd(x),1))
```

```
## # A tibble: 1 × 2
##    hits accuracy
##   <dbl>    <dbl>
## 1  22.7      0.1
```

Evaluating Predictions

```r
mLM <- lm(won ~ hits + accuracy + mental_state + damage_taken +
head_shots + gameIdSession,fn)
fn %>%
  mutate(preds = predict(mLM)) %>%
  mutate(predBinary = ifelse(preds > .5,1,0)) %>%
  select(won,predBinary,preds)
```

```
## # A tibble: 957 × 3
##       won predBinary  preds
##     <dbl>      <dbl>  <dbl>
##  1      0          0 0.320
##  2      0          0 0.239
##  3      0          0 0.193
##  4      0          0 0.285
##  5      0          0 0.148
##  6      0          0 0.175
##  7      0          0 0.258
##  8      0          0 0.115
##  9      0          0 0.239
## 10      1          0 0.0982
## # ℹ 947 more rows
```

Evaluating Predictions

```
(sumTab <- fn %>%
  mutate(pred_win = ifelse(predict(mLM) > .5,1,0)) %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = percent(nGames / total_games)) %>%
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) /
sum(nGames))))
```

```
## # A tibble: 4 × 6
##     won pred_win total_games nGames prop  accuracy
##   <dbl>    <dbl>       <int>  <int> <chr> <chr>
## 1     0        0         666    615 92%   71%
## 2     0        1         666     51 8%    71%
## 3     1        0         291    226 78%   71%
## 4     1        1         291     65 22%   71%
```

Evaluating Predictions

- Overall accuracy is just the number of correct predictions (either `0` or `1`) out of all possible
    - Is 71% good?
    - What would the dumbest guess be? Never win! 70%
- Might also want to care about just `1`s
    - **Sensitivity**: Predicted wins / actual wins = 22%
- Also might care about just `0`s
    - **Specificity**: Predicted losses / actual losses = 92%

Thresholds

- Shifting the threshold for `0` or `1` prediction can matter

```
fn %>%
  mutate(pred_win = ifelse(predict(mLM) > .4,1,0)) %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = percent(nGames / total_games)) %>%
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) /
sum(nGames)))
```

```
## # A tibble: 4 × 6
##     won pred_win total_games nGames prop  accuracy
##   <dbl>    <dbl>       <int>  <int> <chr> <chr>
## 1     0        0         666    542 81.4% 72%
## 2     0        1         666    124 18.6% 72%
## 3     1        0         291    144 49.5% 72%
## 4     1        1         291    147 50.5% 72%
```

Thresholds

- Shifting the threshold for `0` or `1` prediction can matter

```
fn %>%
  mutate(pred_win = ifelse(predict(mLM) > .7,1,0)) %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = percent(nGames / total_games)) %>%
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) /
sum(nGames)))
```

```
## # A tibble: 4 × 6
##     won pred_win total_games nGames prop  accuracy
##   <dbl>    <dbl>       <int>  <int> <chr> <chr>
## 1     0        0         666    663 99.5% 70%
## 2     0        1         666      3 0.5%  70%
## 3     1        0         291    280 96.2% 70%
## 4     1        1         291     11 3.8%  70%
```

- Restricting to above 70% means we don't think anyone wins!

Thresholds

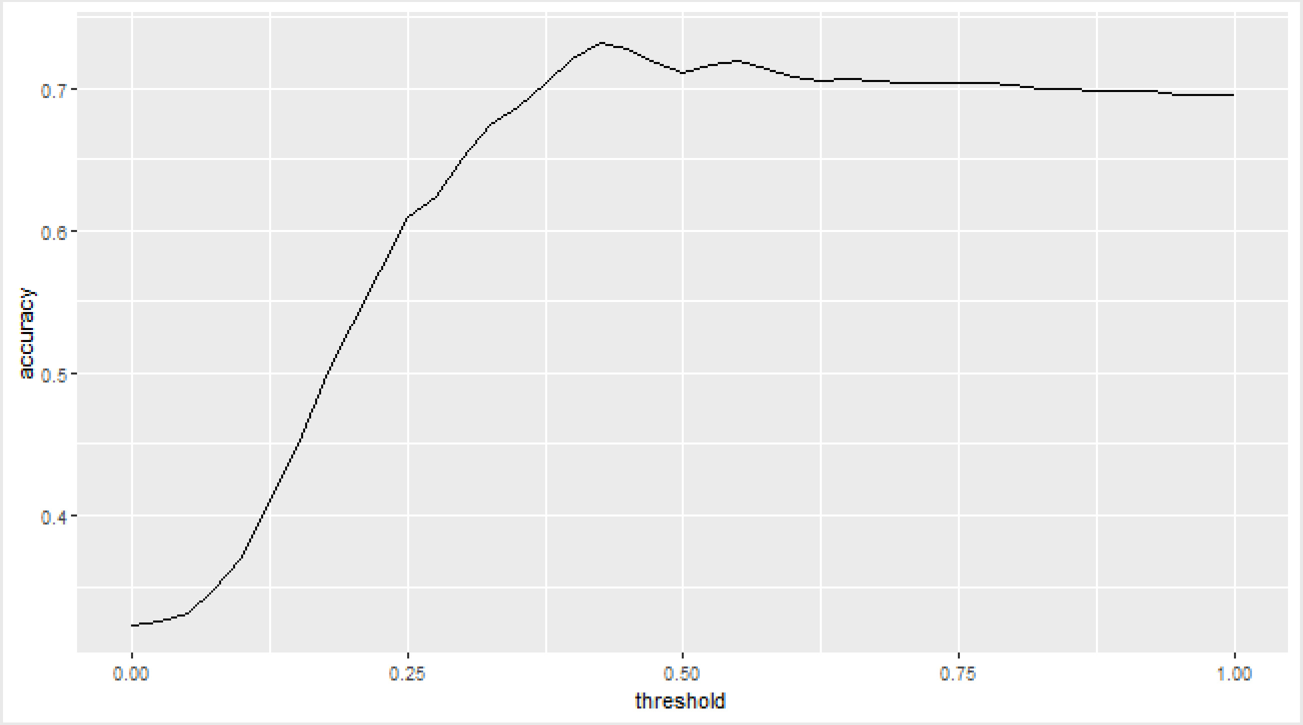
- We could keep trying different values until we hit on one that maximizes our accuracy
- But this is inefficient! Let's loop it instead!

```
toplot <- NULL
for(thresh in seq(0,1,by = .025)) {
  toplot <- fn %>%
  mutate(pred_win = ifelse(predict(mLM) > thresh,1,0)) %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>%
  ungroup() %>%
  mutate(accuracy = sum((won == pred_win)*nGames) / sum(nGames)) %>%
  mutate(threshold = thresh) %>%
    bind_rows(toplot)
}
```

Thresholds

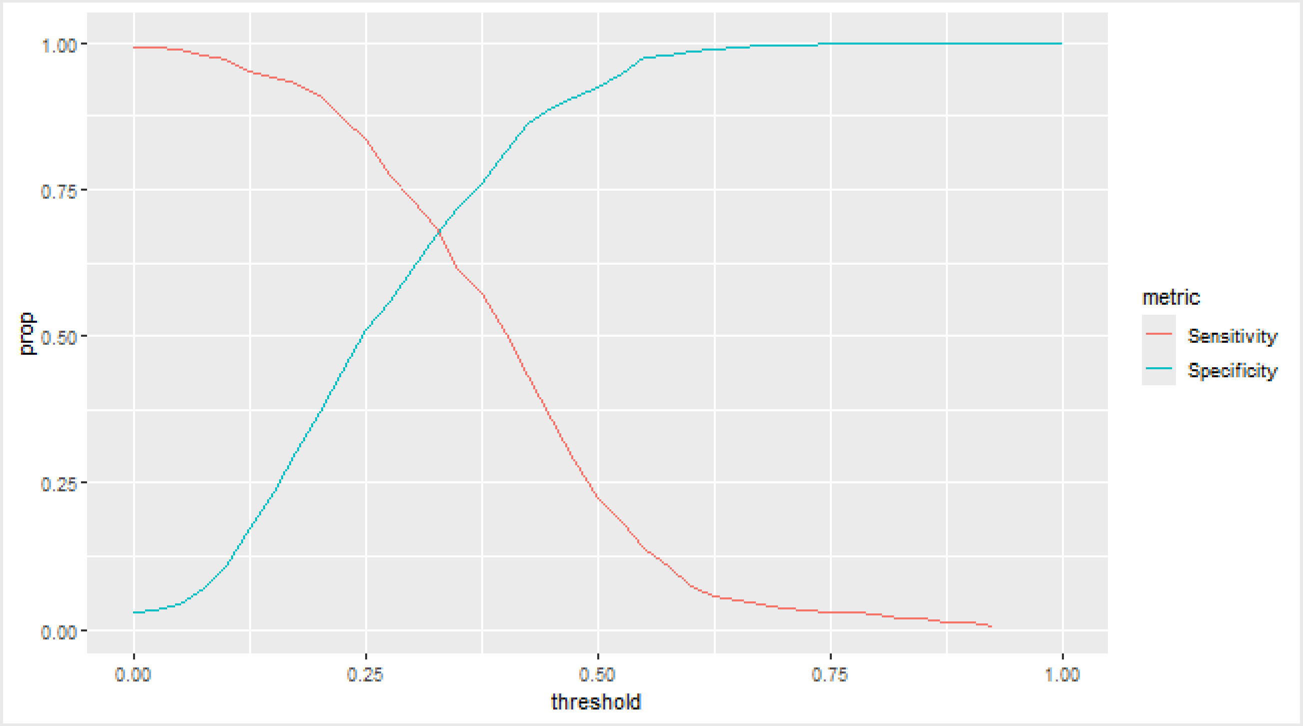
- We might only care about accuracy by itself (although this is a bit naive)

```
toplot %>%
  select(accuracy,threshold) %>%
  distinct() %>%
  ggplot(aes(x = threshold,y = accuracy)) +
  geom_line()
```



Thresholds

```r
toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                         ifelse(won == 0 & pred_win == 0,'Specificity',NA))) %>%
  drop_na(metric) %>%
  ggplot(aes(x = threshold,y = prop,color = metric)) +
  geom_line()
```



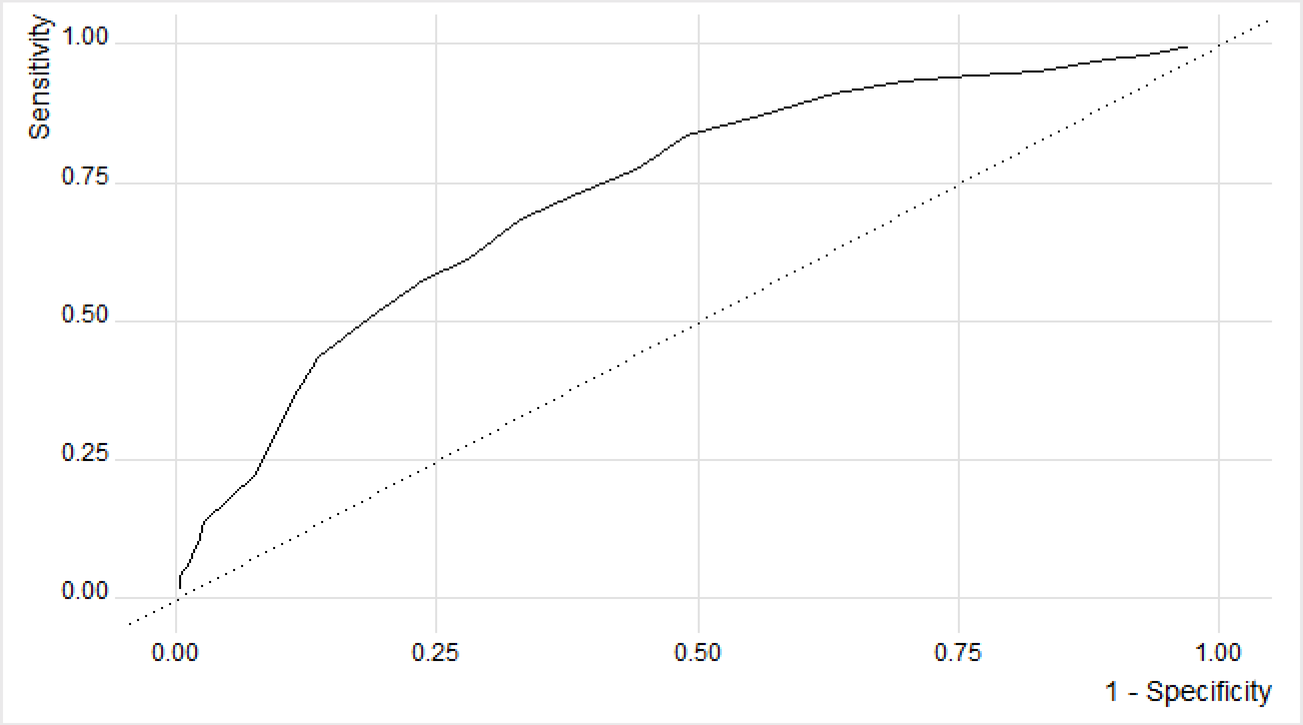
ROC Curve

- Receiver-Operator Characteristic (ROC) Curve
- Commonly used to evaluate classification methods
    - X-axis: 1-specificity
    - Y-axis: sensitivity

```
p <- toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                              ifelse(won == 0 & pred_win ==
0,'Specificity',NA))) %>%
  drop_na(metric) %>%
  select(prop,metric,threshold) %>%
  spread(metric,prop) %>%
  arrange(desc(Specificity),Sensitivity) %>%
  ggplot(aes(x = 1-Specificity,y = Sensitivity)) +
  geom_line() +
  xlim(c(0,1)) + ylim(c(0,1)) +
  geom_abline(slope = 1,intercept = 0,linetype = 'dotted') +
  ggridges::theme_ridges()
```

ROC Curve

```
p
```



- Better models have high levels of sensitivity **and** specificity at every threshold

AUC Measure

- Area Under the Curve (AUC)
    - A single number summarizing classification performance

```
require(tidymodels)
roc_auc(data = fn %>%
  mutate(pred_win = predict(mLM),
         truth = factor(won,levels = c('1','0'))) %>%
  select(truth,pred_win),truth,pred_win)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.736
```

AUC

- What is a "good" AUC?
    - We know it is bounded between 0 (i.e., it predicts everything **perfectly wrong**) and 1 (i.e., it predicts everything **perfectly correct**)
    - But typically we don't see AUC values less than 0.5 (why is this?)
- AUC can be interpreted like numeric grades at Vandy (and for this class)
    - 0.95+ is amazing
    - 0.9 - 0.95 is very good
    - 0.8-range is B-tier
    - 0.7-range is C-tier
    - 0.6-range is really bad
    - AUC values less than 0.6 are failing

Party time!

- Adding more variables / trying different combinations
- **Workflow**
    1. Train models
    2. Predict models
    3. Evaluate models

Train models

```
m1 <- lm(won ~ hits,fn)
m2 <- lm(won ~ hits + head_shots,fn)
m3 <- lm(won ~ hits + accuracy + head_shots,fn)
m4 <- lm(won ~ hits + accuracy + head_shots + mental_state,fn)
m5 <- lm(won ~ hits + accuracy + head_shots + mental_state +
distance_traveled,fn)
m6 <- lm(won ~ hits + accuracy + mental_state + head_shots +
distance_traveled + gameIdSession,fn)
```

Predict models

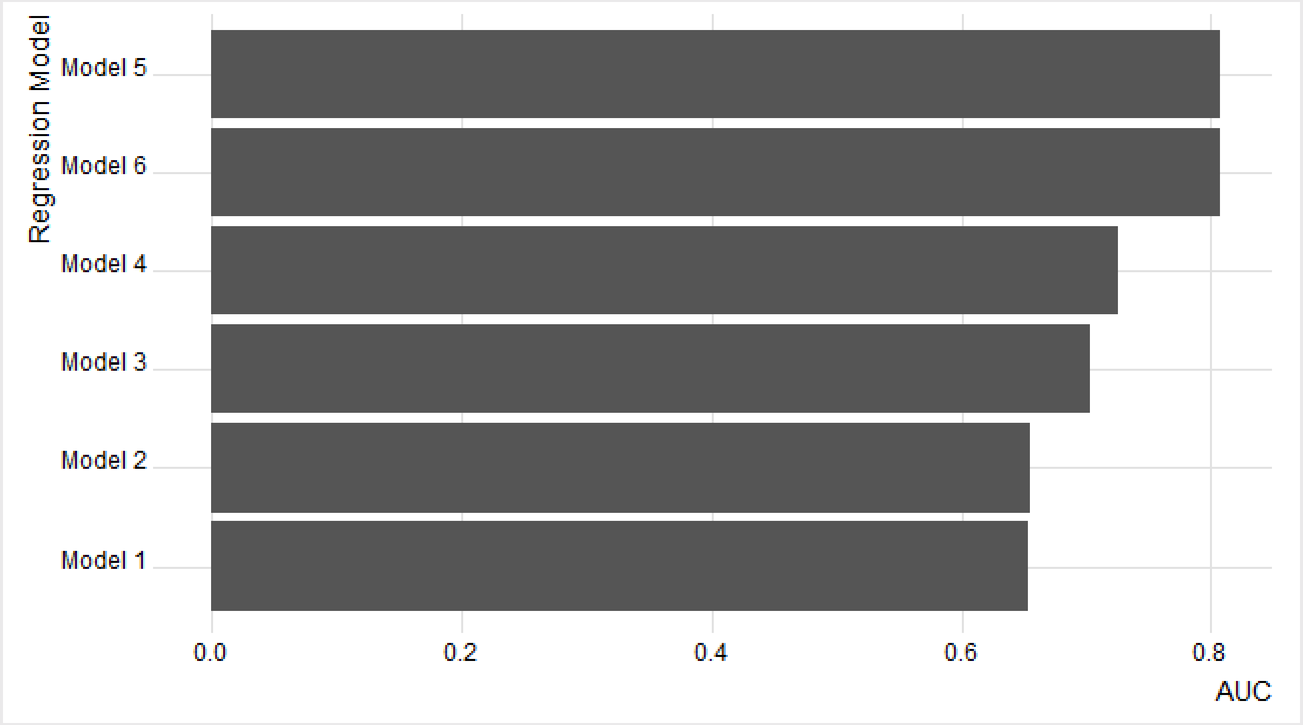
```
toEval <- fn %>%
  mutate(m1Preds = predict(m1),
         m2Preds = predict(m2),
         m3Preds = predict(m3),
         m4Preds = predict(m4),
         m5Preds = predict(m5),
         m6Preds = predict(m6),
         truth = factor(won,levels = c('1','0')))
```

Evaluate models

```r
rocRes <- NULL
for(model in 1:6) {
  rocRes <- roc_auc(toEval,truth,paste0('m',model,'Preds')) %>%
    mutate(model = paste0('Model ',model)) %>%
    bind_rows(rocRes)
}
```

Evaluate models

```r
rocRes %>%
  ggplot(aes(x = .estimate,y = reorder(model,.estimate))) +
  geom_bar(stat = 'identity') +
  ggridges::theme_ridges() + labs(x = 'AUC',y = 'Regression Model')
```



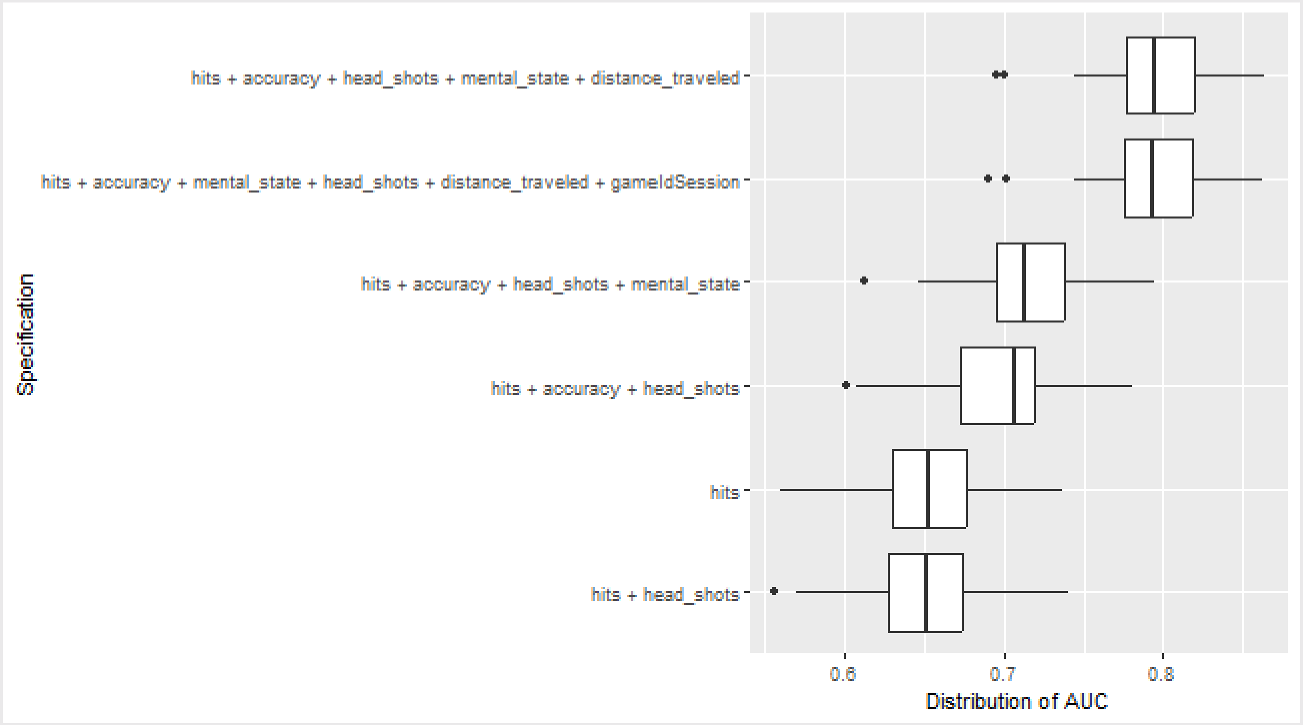
OVERFITTING

- Cross validation to the rescue!

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

  # Training models
  m1 <- lm(won ~ hits,train)
  m2 <- lm(won ~ hits + head_shots,train)
  m3 <- lm(won ~ hits + accuracy + head_shots,train)
  m4 <- lm(won ~ hits + accuracy + head_shots + mental_state,train)
  m5 <- lm(won ~ hits + accuracy + head_shots + mental_state + distance_traveled,train)
  m6 <- lm(won ~ hits + accuracy + mental_state + head_shots + distance_traveled + gameIdSession,train)

  # Predicting models
  toEval <- test %>%
    mutate(m1Preds = predict(m1,newdata = test),
           m2Preds = predict(m2,newdata = test),
           m3Preds = predict(m3,newdata = test),
           m4Preds = predict(m4,newdata = test),
           m5Preds = predict(m5,newdata = test),
           m6Preds = predict(m6,newdata = test),
           truth = factor(won,levels = c('1','0')))

  # Evaluating models
  rocResBS <- NULL
  for(model in 1:6) {
    rocResBS <- roc_auc(toEval,truth,paste0('m',model,'Preds')) %>%
      mutate(model = as.character(get(paste0('m',model))$call$formula)[3]) %>%
      bind_rows(rocResBS)
  }
  cvRes <- rocResBS %>%
    mutate(bsInd = i) %>%
    bind_rows(cvRes)
}
```

Cross Validation AUC

```
cvRes %>%
  ggplot(aes(x = .estimate,y = factor(reorder(model,.estimate)))) +
  geom_boxplot() + labs(x = 'Distribution of AUC',y =
'Specification')
```



Conclusion

- Classification is just a type of prediction
  - We used linear regression
  - But there are **much** fancier algorithms out there
- After the break:
  - A *slightly* fancier algorithm: logistic regression
  - How to use the models to achieve the team's goals

BREAK

Agenda

1. Introducing **logit**
2. Running logit
3. Evaluating logit

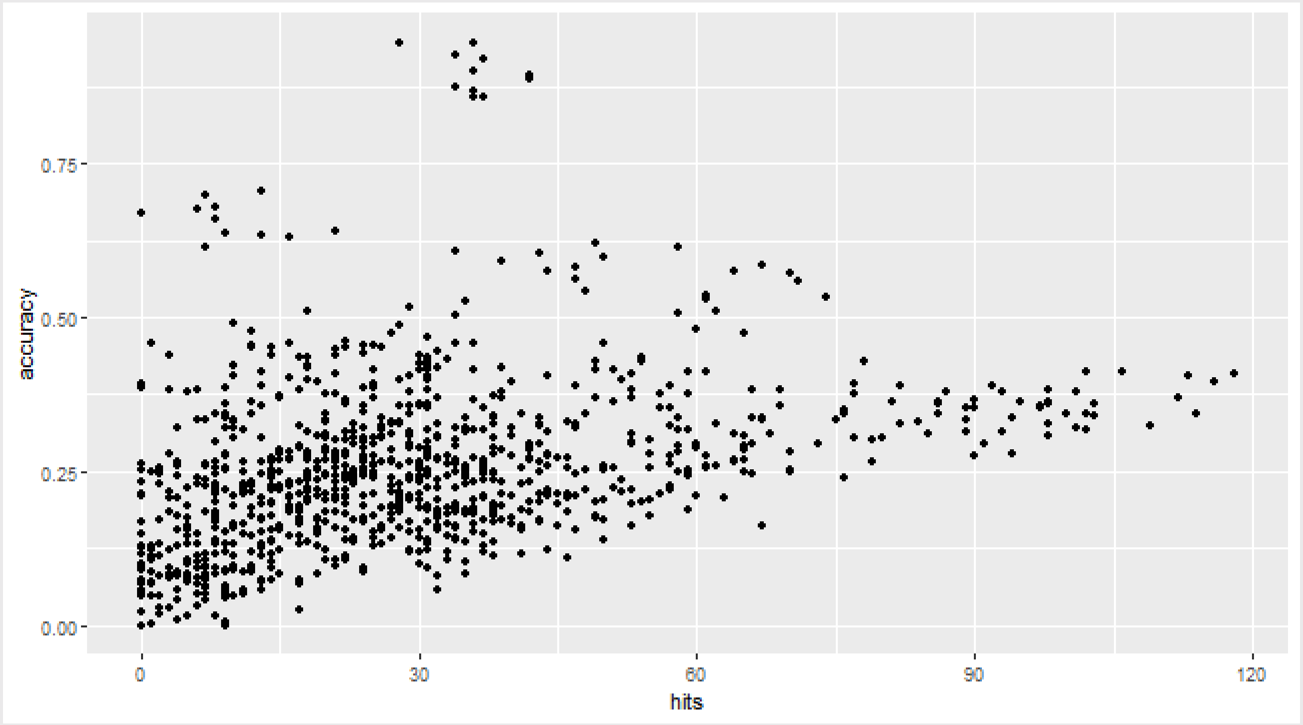
Logit Regression

- A different **type** of **regression**
  - What do we mean by **type**?
- Let's take a step back

Regression Types

- "Linear" regression...why is it "linear"?

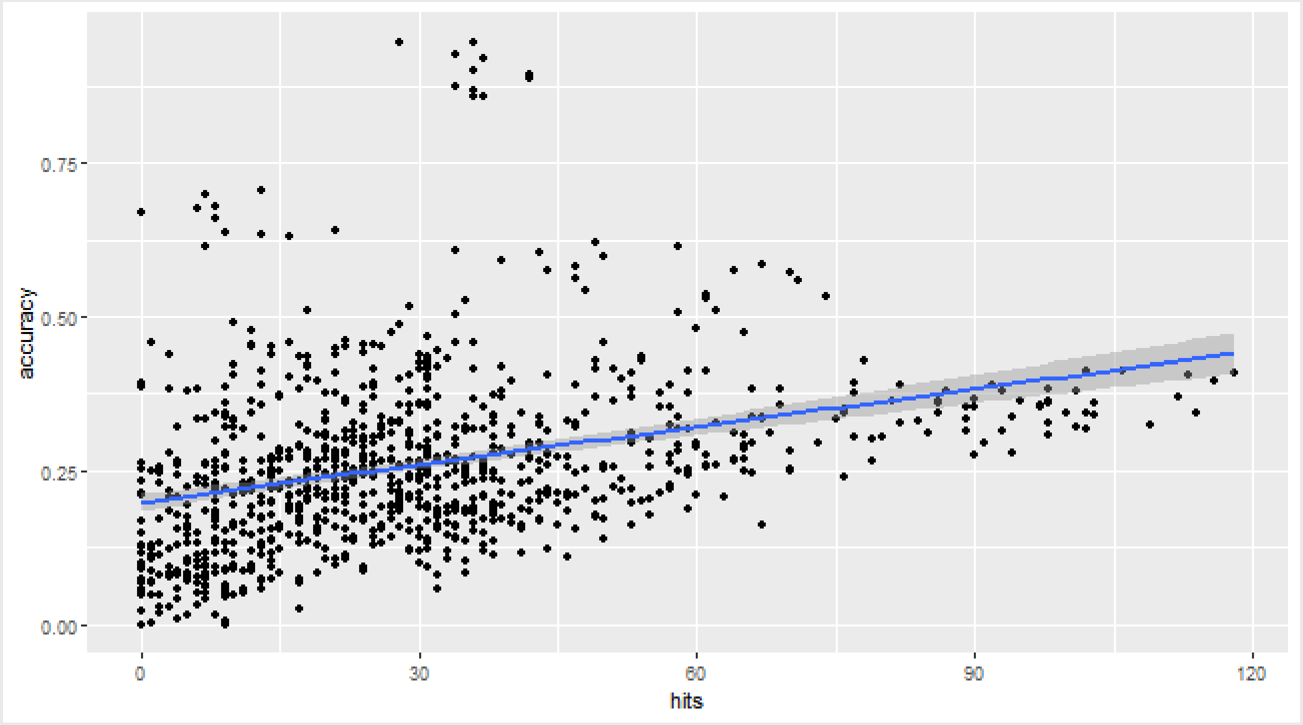
```
(p <- fn %>%
  ggplot(aes(x = hits,y = accuracy)) +
  geom_point())
```



Regression Types

- "Linear" regression...why is it "linear"?
- Because you can summarize it with a line!

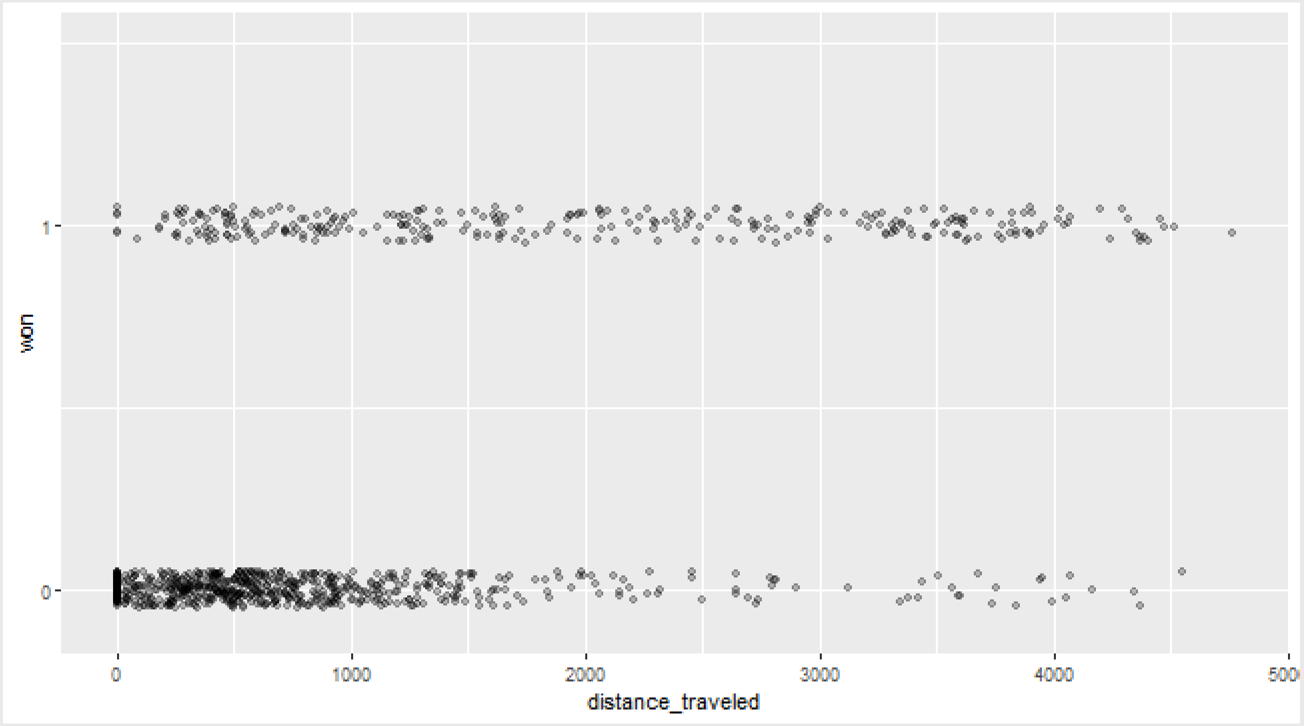
```
p + geom_smooth(method = 'lm')
```



Regression Types

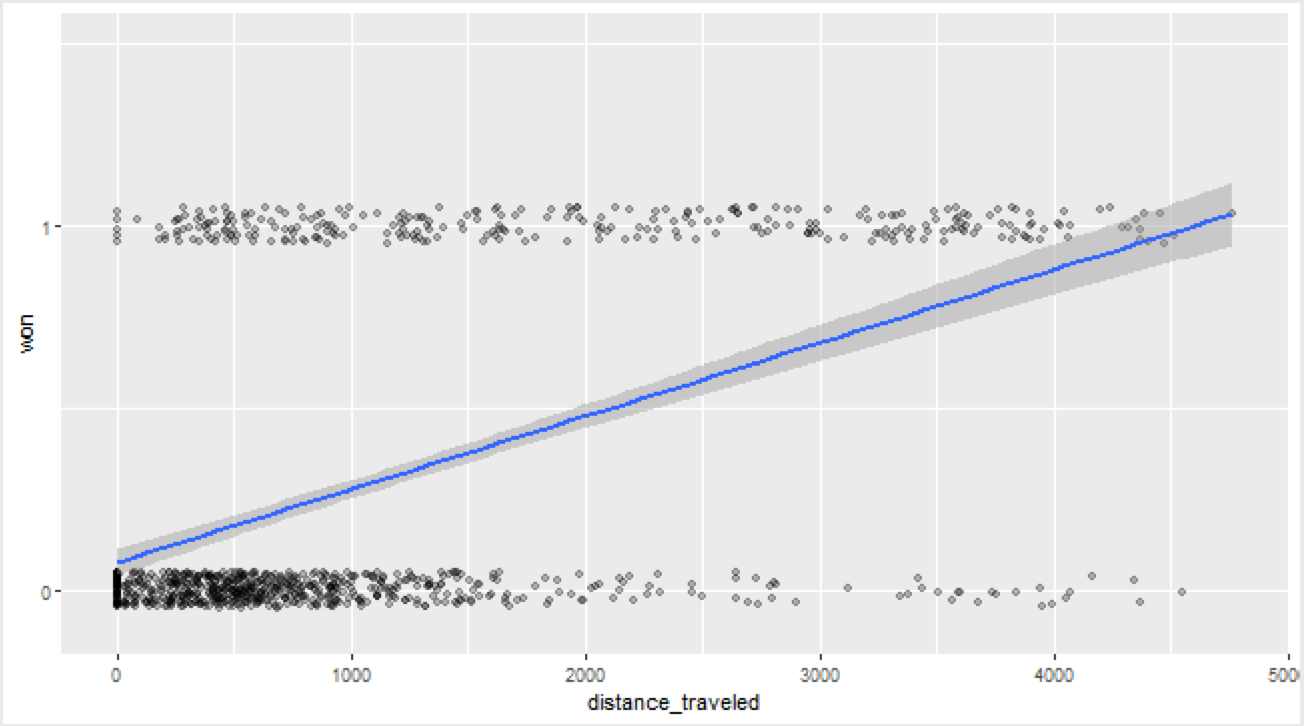
- But what if the outcome is binary?

```
(p <- fn %>% ggplot(aes(x = distance_traveled,y = won)) +
   scale_y_continuous(breaks = c(0,1),limits = c(-.1,1.5)) +
  geom_jitter(width = .01,height = .05,alpha = .25))
```



Regression Types

- But what if the outcome is binary?
- Lines seem too clumsy
  - If `1` = won, how can you go higher?



Logit

- Theory: binary outcomes are **proxies** for some **latent** measure
  - Binary outcome `won`: either placed first or did not
  - Latent outcome `placed`: continuous measure
  - Might also imagine **ability**: continuous measure
- The higher your `ability`, the more likely you are to win
- Logit regression: model the `ability`
  - What is `ability` actually?
  - Probability of winning: $Pr(won)$
- Part of a broader class of models called "generalized linear model" (GLM)

$$Pr(y = 1|x) = G(\alpha + \beta X)$$

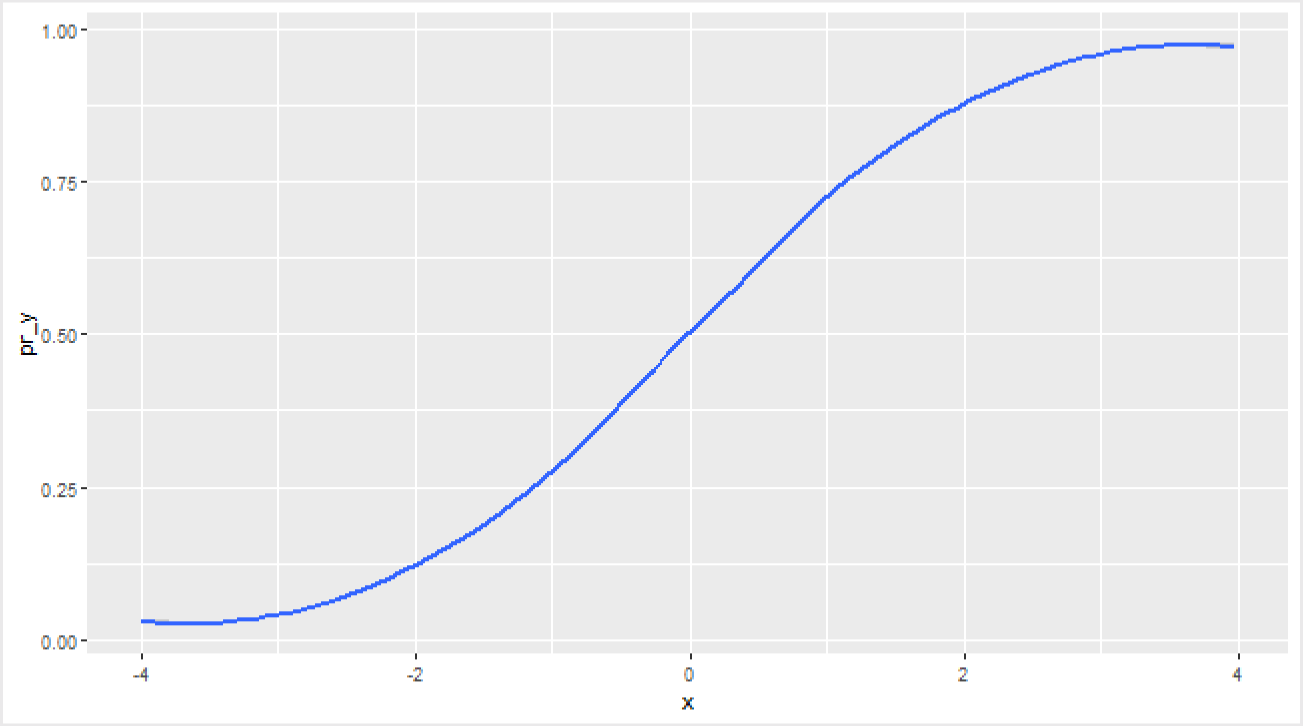
GLMs

- $Pr(y = 1|x) = G(\alpha + \beta X)$
- Does this look familiar?
- Linear regression: $Y = \alpha + \beta X$
  - Outcome: $Y \rightarrow Pr(y = 1|x)$
  - Mapping: $\alpha + \beta X \rightarrow G(\alpha + \beta X)$
- G is the "link function"
  - Transforms values of $\alpha + \beta X$ into **probabilities**
- Logistic function: specific type of link function

$$G(x) = \frac{1}{1 + exp(-x)}$$

Logistic Function

```r
x <- runif(100,-4,4)
pr_y <- 1/(1 + exp(-x))
as_tibble(pr_y = pr_y,x = x) %>%
  ggplot(aes(x = x,y = pr_y)) +
  geom_smooth()
```

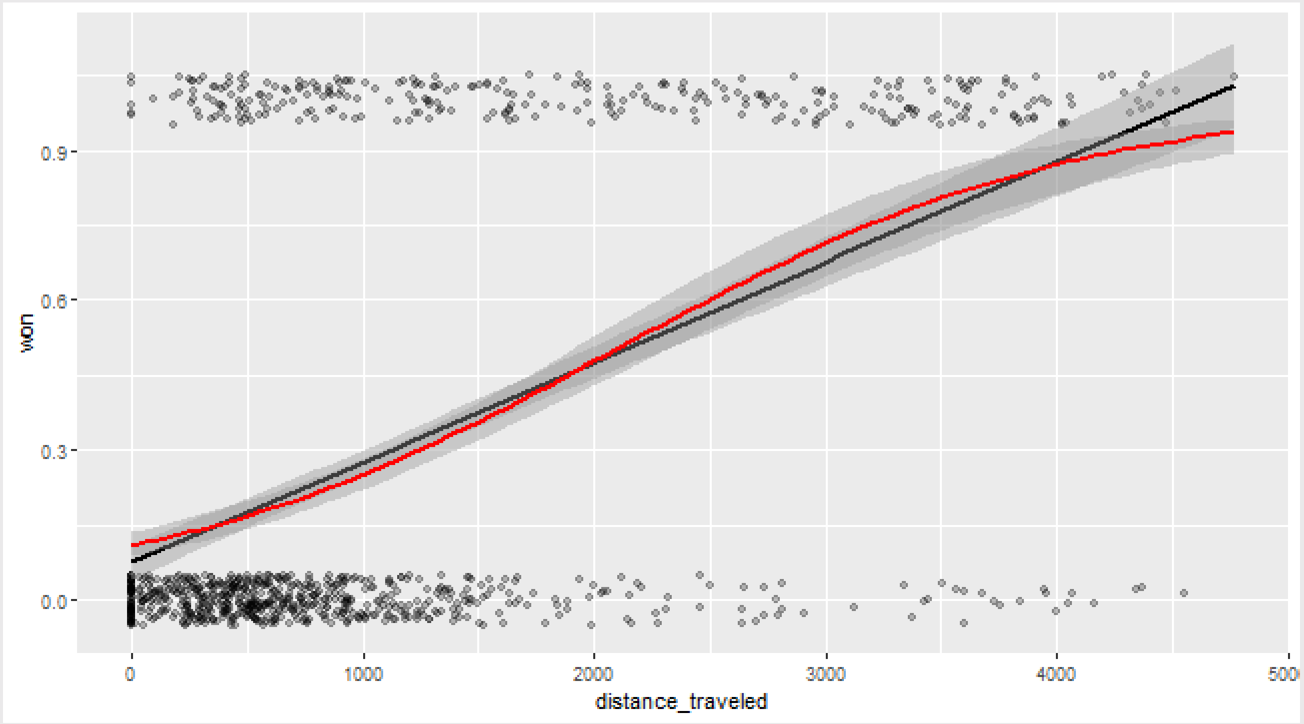


Logistic Function

- But what about real data like $\alpha + \beta X$?
- $G(X) = \frac{exp(\alpha+\beta X)}{1+exp(\alpha+\beta X)}$
- We estimate this with `glm(formula,data,family)`
  - Note similarity to `lm(formula,data)`
- `family = binomial(link = "logit")`

Logistic Regression (logit)

```r
fn %>% ggplot(aes(x = distance_traveled,y = won)) +
  geom_jitter(width = .01,height = .05,alpha = .25) +
  geom_smooth(method = 'lm',color = 'black') +
  geom_smooth(method = 'glm',color = 'red',
              method.args = list(family = binomial(link = 'logit')))
```



Logistic Regression (logit)

```
# Train model
mLogit <- glm(formula = won ~ distance_traveled,data = fn,family =
binomial(link = 'logit'))

# Predict model
fn <- fn %>%
  mutate(prob_won = predict(mLogit,type = 'response')) %>%
  mutate(pred_won = ifelse(prob_won > .5,1,0))

# Evaluate model
eval <- fn %>%
  group_by(won) %>%
  mutate(total_games = n()) %>%
  group_by(won,pred_won,total_games) %>%
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>%
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_won)*nGames) /
sum(nGames)))
```

Logistic Regression (logit)

```
eval
```

```
## # A tibble: 4 × 6
##     won pred_won total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl> <chr>
## 1     0        0         666    620 0.931  78%
## 2     0        1         666     46 0.0691 78%
## 3     1        0         291    163 0.560  78%
## 4     1        1         291    128 0.440  78%
```

Logistic Regression (logit)

- Can also calculate ROC Curve and AUC

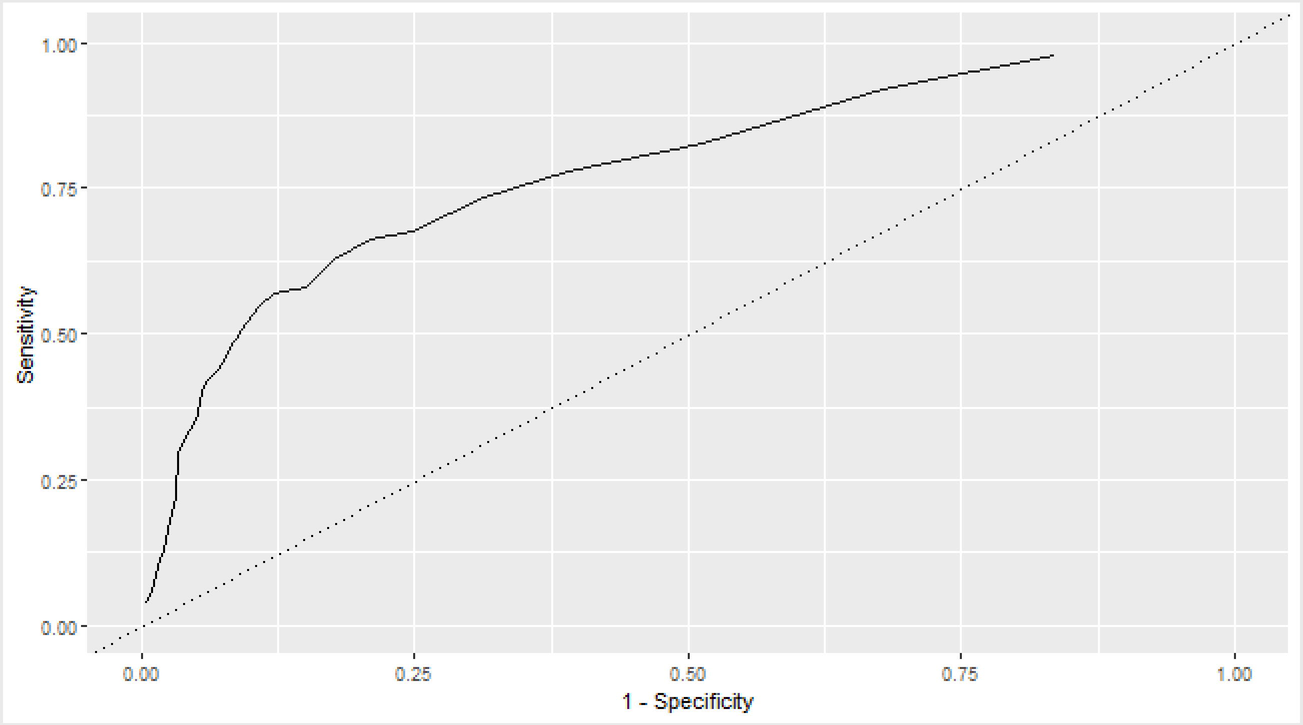
```r
toplot <- NULL
for(thresh in seq(0,1,by = .025)) {
  toplot <- fn %>%
    mutate(pred_won = ifelse(predict(mLogit,type = 'response') >
thresh,1,0)) %>%
    group_by(won) %>%
    mutate(total_games = n()) %>%
    group_by(won,pred_won,total_games) %>%
    summarise(nGames=n(),.groups = 'drop') %>%
    mutate(prop = nGames / total_games) %>%
    ungroup() %>%
    mutate(threshold = thresh) %>%
    bind_rows(toplot)
}
```

Logistic Regression (logit)

```r
p <- toplot %>%
  mutate(metric = ifelse(won == 1 & pred_won == 1,'Sensitivity',
                              ifelse(won == 0 & pred_won ==
0,'Specificity',NA))) %>%
  drop_na(metric) %>%
  select(prop,metric,threshold) %>%
  spread(metric,prop) %>%
  arrange(desc(Specificity),Sensitivity) %>%
  ggplot(aes(x = 1-Specificity,y = Sensitivity)) +
  geom_line() +
  xlim(c(0,1)) + ylim(c(0,1)) +
  geom_abline(slope = 1,intercept = 0,linetype = 'dotted')
```

Logistic Regression (logit)

```
p
```



Logistic Regression (logit)

```r
require(tidymodels)
roc_auc(data = fn %>%
  mutate(prob_won = predict(mLogit,type = 'response'),
         truth = factor(won,levels = c('1','0'))) %>%
  select(truth,prob_won),truth,prob_won)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.782
```

Comparing Models

- Two big questions in prediction:
    1. Do I have the correct predictors X?
    2. Do I have the best model?
- Two types of outcomes (thus far)
    1. Continuous Y: use **RMSE**
    2. Binary Y: use **AUC**
- Let's determine the best model from the following:
    - X: (1) `distance_traveled + mental_state` vs. (2) `distance_traveled + mental_state + hits`
    - Model: (1) conditional means vs. (2) `lm` vs. (3) `glm`

Comparing Models

- Conditional means - simple X

```r
results <- NULL

# Train & Predict
toEval <- fn %>%
  mutate(distDec = ntile(distance_traveled,n = 10)) %>%
  group_by(distDec,mental_state) %>%
  mutate(prob_won = mean(won),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'CM',
         predictors = 'Simple') %>%
  bind_rows(results)
```

Comparing Models

- Conditional means - complex X

```r
# Train & Predict
toEval <- fn %>%
  mutate(distDec = ntile(distance_traveled,n = 10),
         hitsDec = ntile(hits,n = 10)) %>%
  group_by(distDec,hitsDec,mental_state) %>%
  mutate(prob_won = mean(won),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'CM',
         predictors = 'Complex') %>%
  bind_rows(results)
```

Comparing Models

- Linear regression (`lm`) - simple X

```r
# Train
m <- lm(won ~ distance_traveled + mental_state,fn)

# Predict
toEval <- fn %>%
  mutate(prob_won = predict(m),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'LM',
         predictors = 'Simple') %>%
  bind_rows(results)
```

Comparing Models

- Linear regression (`lm`) - complex X

```
# Train
m <- lm(won ~ distance_traveled + mental_state + hits,fn)

# Predict
toEval <- fn %>%
  mutate(prob_won = predict(m),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'LM',
         predictors = 'Complex') %>%
  bind_rows(results)
```

Comparing Models

- Logit regression (`glm`) - simple X

```
# Train
m <- glm(won ~ distance_traveled + mental_state,fn,family =
binomial(link = 'logit'))

# Predict
toEval <- fn %>%
  mutate(prob_won = predict(m,type = 'response'),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'GLM',
         predictors = 'Simple') %>%
  bind_rows(results)
```

Comparing Models

- Logit regression (`glm`) - complex X

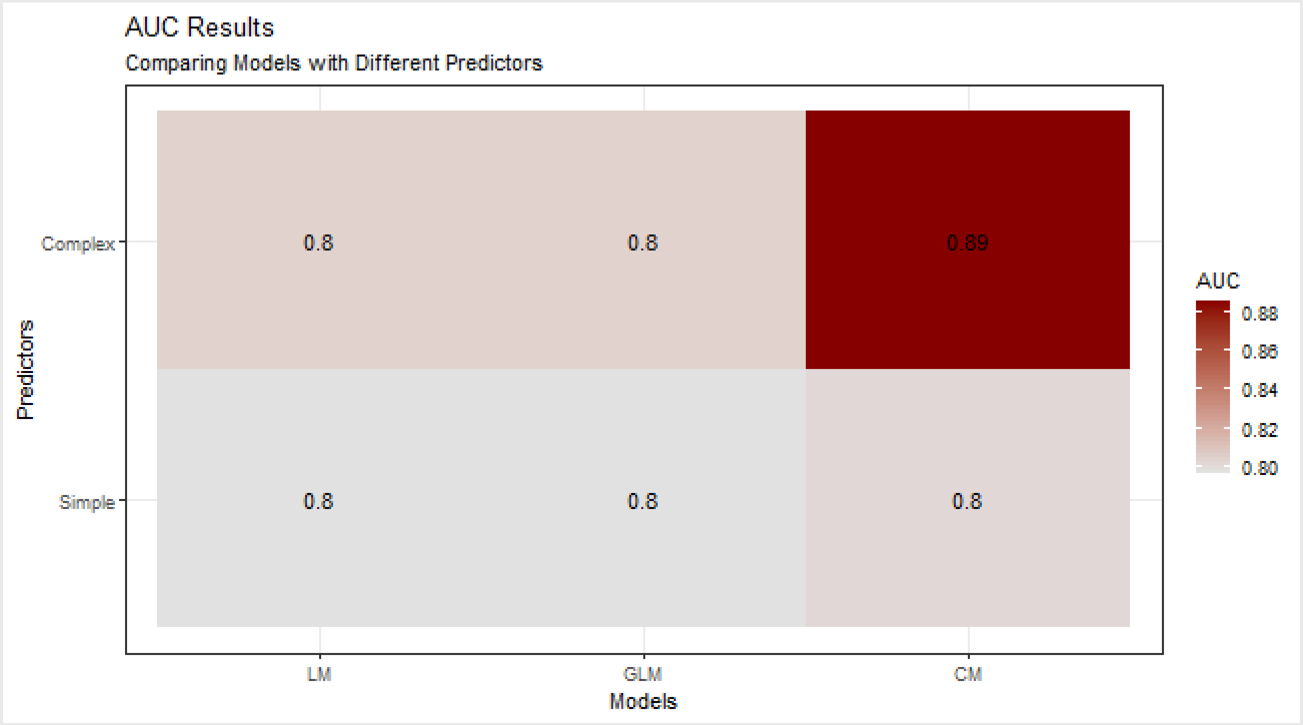
```
# Train
m <- glm(won ~ distance_traveled + mental_state + hits,fn,family =
binomial(link = 'logit'))

# Predict
toEval <- fn %>%
  mutate(prob_won = predict(m,type = 'response'),
         truth = factor(won,levels = c('1','0'))) %>%
    ungroup() %>%
    select(truth,prob_won)

# Evaluate
results <- roc_auc(data = toEval,truth,prob_won) %>%
  mutate(model = 'GLM',
         predictors = 'Complex') %>%
  bind_rows(results)
```

Comparing Models

```r
p <- results %>%
  ggplot(aes(x = reorder(model,.estimate),
             y = reorder(predictors,.estimate),
             fill = .estimate,label = round(.estimate,2))) +
  geom_tile() +
  scale_fill_continuous(low = 'grey90',high = 'darkred') +
  geom_text() +
  labs(title = 'AUC Results',
       subtitle = 'Comparing Models with Different Predictors',
       x = 'Models',y = 'Predictors',
       fill = 'AUC') +
  theme_bw()
```

Comparing Models

```
p
```



Conclusion

- Conditional means outperform regression models?
    - Yes: conditional means allow for cell-specific predictions
    - No: conditional means are more susceptible to **overfitting**
- How would you re-evaluate these models-X-predictors to account for overfitting?