Advanced Topics in Classification

Multinomial logistic regression (++)

July 14th, 2023

Example: NFL Expected Points

What does football **play-by-play** data look like? Each row is a play with contextual information:

- **Possession team:** team with the ball, on offense (opposing team is on defense)
- **Down:** 4 downs to advance the ball 10 (or more) yards
    - New set of downs, else turnover to defense
- **Yards to go:** distance in yards to advance
- **Yard line:** distance in yards away from opponent's endzone (100 to 0) - the field position
- **Time remaining:** seconds remaining in game, each game is 3600 seconds long
    - 4 quarters, halftime in between, followed by a potential overtime (900 seconds)

Example: NFL Expected Points

Drive: a series of plays, changes with possession and the types of scoring events:

- **No Score:** 0 points - turnover the ball or half/game ends
- **Field Goal:** 3 points - kick through opponent's goal post
- **Touchdown:** 7 points - enter opponent's end zone
- **Safety:** 2 points for opponent - tackled in own endzone

Next Score: type of next score (current drive or future drives) with respect to possession team

- For: Touchdown (7), Field Goal (3), Safety (2)
- Against: -Touchdown (-7), -Field Goal (-3), -Safety (-2)
- No Score

Note: treating point-after-touchdown attempts (PATs) separately

Example: NFL Expected Points

Expected Points: Measure the value of play in terms of $\mathbb{E}[\text{points of next scoring play}]$

- i.e., historically, how many points have teams scored when in similar situations?

Explanatory variables: $\mathbf{X} =$ {down, yards to go, yard line, ...}

Want to **estimate the probabilities** of each scoring event to compute expected points:

- Outcome probabilities: $P(Y = y|\mathbf{X})$
- Expected Points $= E(Y|\mathbf{X}) = \sum_{y \in Y} y \cdot P(Y = y|\mathbf{X})$

How do we model more than two categories???

Review: logistic regression

Response variable Y has two possible values: 1 or 0, we estimate the probability

$$p(x) = P(Y = 1|X = x)$$

Assuming that we are dealing with two classes, the possible observed values for Y are 0 and 1,

$$Y|x \sim \text{Binomial}(n = 1, p = \mathbb{E}[Y|x]) = \text{Bernoulli}(p = \mathbb{E}[Y|x])$$

To limit the regression betweewn $[0, 1]$: use the **logit** function, aka the **log-odds ratio**

$$\text{logit}(p(x)) = \log\left[\frac{p(x)}{1 - p(x)}\right] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

meaning

$$p(x) = \frac{e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p}}$$

Multinomial logistic regression

We can extend this to K classes (via the softmax function):

$$P(Y = k^* \mid X = x) = \frac{e^{\beta_{0k^*} + \beta_{1k^*} x_1 + \cdots + \beta_{pk^*} x_p}}{\sum_{k=1}^{K} e^{\beta_{0k} + \beta_{1k} x_1 + \cdots + \beta_{pk} x_p}}$$

We only estimate coefficients for $K - 1$ classes **relative to reference class**

For example, let K be the reference then we use $K - 1$ logit transformations

- Use $\boldsymbol{\beta}$ for vector of coefficients and $\mathbf{X}$ for matrix of predictors

$$\log\left(\frac{P(Y=1|\mathbf{X})}{P(Y=K|\mathbf{X})}\right) = \boldsymbol{\beta}_1 \cdot \mathbf{X}$$
$$\log\left(\frac{P(Y=2|\mathbf{X})}{P(Y=K|\mathbf{X})}\right) = \boldsymbol{\beta}_2 \cdot \mathbf{X}$$
$$\log\left(\frac{P(Y=K-1|\mathbf{X})}{P(Y=K|\mathbf{X})}\right) = \boldsymbol{\beta}_{K-1} \cdot \mathbf{X}$$

Multinomial logistic regression for next score

$Y \in$ {Touchdown (7), Field Goal (3), Safety (2), No Score (0), -Safety (-2), -Field Goal (-3), -Touchdown (-7)}

$\mathbf{X} =$ {down, yards to go, yard line, ...}

Model is specified with **six logit transformations** relative to **No Score**:

$$\log\left(\frac{P(Y = \text{Touchdown } | \mathbf{X})}{P(Y = \text{No Score } | \mathbf{X})}\right) = \mathbf{X} \cdot \boldsymbol{\beta}_{\text{Touchdown}}$$

$$\log\left(\frac{P(Y = \text{Field Goal } | \mathbf{X})}{P(Y = \text{No Score } | \mathbf{X})}\right) = \mathbf{X} \cdot \boldsymbol{\beta}_{\text{Field Goal}},$$

$$\vdots$$

$$\log\left(\frac{P(Y = - \text{ Touchdown } | \mathbf{X})}{P(Y = \text{No Score } | \mathbf{X})}\right) = \mathbf{X} \cdot \boldsymbol{\beta}_{-\text{Touchdown}},$$

- Model is generating probabilities, agnostic of value associated with each next score type
- Fit multinomial logistic regression model in `R` with `nnet` package

NFL play-by-play data (2010 to 2020)

Initialized NFL play-by-play dataset with next score in half for each play

- Followed steps in script by Ben Baldwin (which copies Prof. Yurko's steps here)

```
library(tidyverse)
nfl_ep_model_data <- readRDS(url("https://shorturl.at/BTVZ1"))

nfl_ep_model_data <- nfl_ep_model_data %>%
  mutate(Next_Score_Half = fct_relevel(Next_Score_Half, "No_Score"),
         # log transform of yards to go and indicator for two minute warning:
         log_ydstogo = log(ydstogo),
         # Changing down into a factor variable:
         down = factor(down))
```

How to fit the model?

```
init_ep_model <- multinom(Next_Score_Half ~ half_seconds_remaining + yardline_100 + down +
                            log_ydstogo + log_ydstogo*down + yardline_100*down,
                          data = nfl_ep_model_data, maxit = 300)
```

What does the `summary()` *function return?*

Leave-one-season-out cross-validation

```
library(nnet)
init_loso_cv_preds <-
  map_dfr(unique(nfl_ep_model_data$season),
          function(x) {
            # Separate test and training data:
            test_data <- nfl_ep_model_data %>% filter(season == x)
            train_data <- nfl_ep_model_data %>% filter(season != x)

            # Fit multinomial logistic regression model:
            ep_model <-
              multinom(Next_Score_Half ~ half_seconds_remaining + yardline_100 + down +
                         log_ydstogo + log_ydstogo*down + yardline_100*down,
                       data = train_data, maxit = 300)

            # Return dataset of class probabilities:
            predict(ep_model, newdata = test_data, type = "probs") %>%
              as_tibble() %>%
              mutate(Next_Score_Half = test_data$Next_Score_Half,
                     season = x)
              })
```

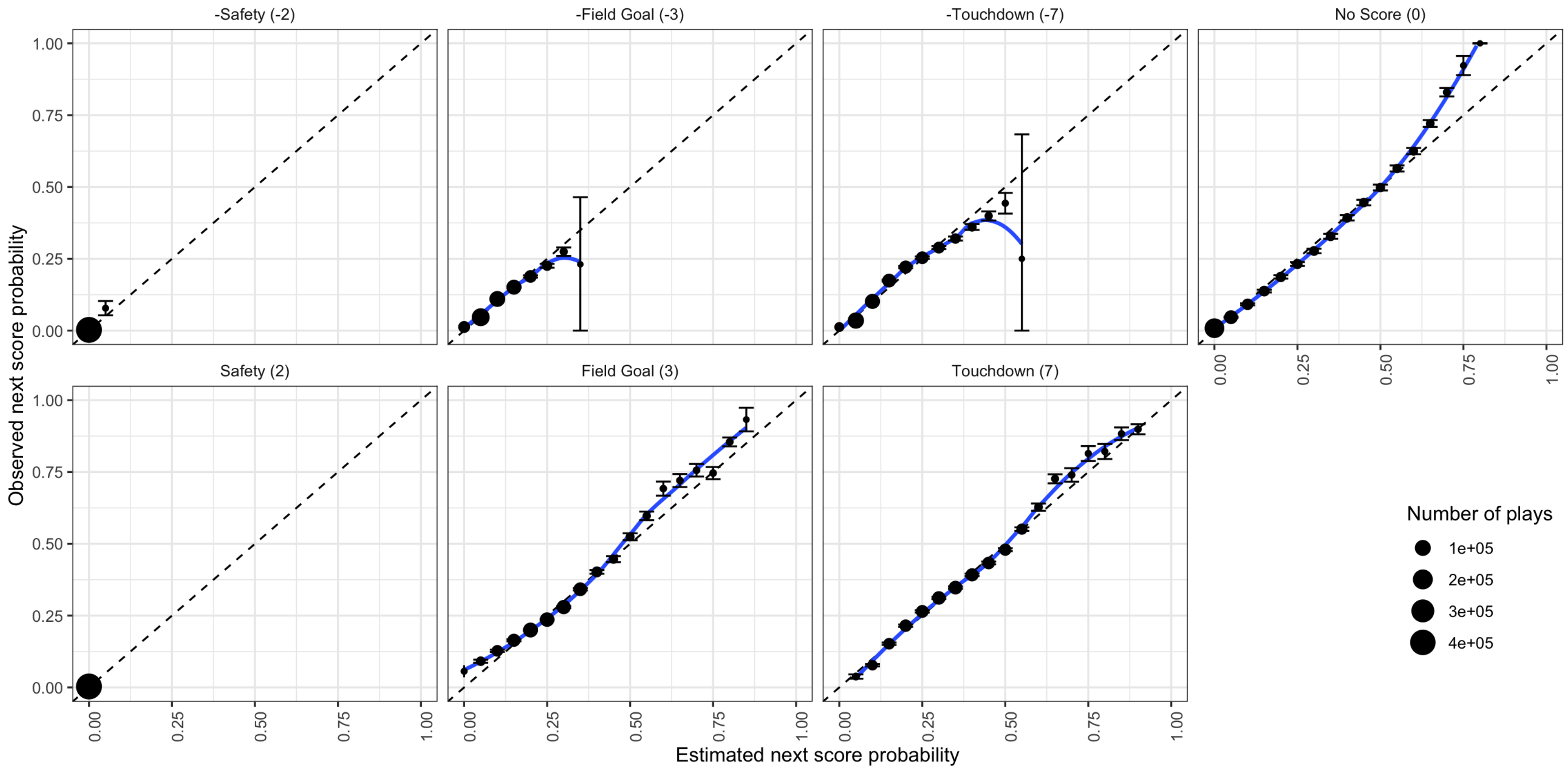
Calibration results for each scoring event

```
ep_cv_loso_calibration_results <- init_loso_cv_preds %>%
  pivot_longer(No_Score:Touchdown,
               names_to = "next_score_type",
               values_to = "pred_prob") %>%
  mutate(bin_pred_prob = round(pred_prob / 0.05) * .05) %>%
  group_by(next_score_type, bin_pred_prob) %>%
  summarize(n_plays = n(),
            n_scoring_event = length(which(Next_Score_Half == next_score_type)),
            bin_actual_prob = n_scoring_event / n_plays,
            bin_se = sqrt((bin_actual_prob * (1 - bin_actual_prob)) / n_plays)) %>%
  ungroup() %>%
  mutate(bin_upper = pmin(bin_actual_prob + 2 * bin_se, 1),
         bin_lower = pmax(bin_actual_prob - 2 * bin_se, 0))
```

Calibration results for each scoring event

```r
ep_cv_loso_calibration_results %>%
  mutate(next_score_type = fct_relevel(next_score_type, "Opp_Safety", "Opp_Field_Goal",
                                       "Opp_Touchdown", "No_Score", "Safety", "Field_Goal",
                                       "Touchdown"),
  next_score_type = fct_recode(next_score_type, "-Field Goal (-3)" = "Opp_Field_Goal",
                               "-Safety (-2)" = "Opp_Safety", "-Touchdown (-7)" = "Opp_Touchdown'
                               "Field Goal (3)" = "Field_Goal", "No Score (0)" = "No_Score",
                               "Touchdown (7)" = "Touchdown", "Safety (2)" = "Safety")) %>%
  ggplot(aes(x = bin_pred_prob, y = bin_actual_prob)) +
  geom_abline(slope = 1, intercept = 0, color = "black", linetype = "dashed") +
  geom_smooth(se = FALSE) +
  geom_point(aes(size = n_plays)) +
  geom_errorbar(aes(ymin = bin_lower, ymax = bin_upper)) + #coord_equal() +
  scale_x_continuous(limits = c(0,1)) +
  scale_y_continuous(limits = c(0,1)) +
  labs(size = "Number of plays", x = "Estimated next score probability",
       y = "Observed next score probability") +
  theme_bw() +
  theme(strip.background = element_blank(),
        axis.text.x = element_text(angle = 90),
        legend.position = c(1, .05), legend.justification = c(1, 0)) +
  facet_wrap(~ next_score_type, ncol = 4)
```

Calibration results for each scoring event



Multinomial classification with XGBoost

Use same NFL play-by-play dataset as before but get ready for XGBoost...

```
nfl_ep_model_data <- nfl_ep_model_data %>%
  mutate(Next_Score_Half = fct_relevel(Next_Score_Half,
                                       "No_Score", "Safety", "Field_Goal", "Touchdown",
                                       "Opp_Safety", "Opp_Field_Goal", "Opp_Touchdown"),
         next_score_label = as.numeric(Next_Score_Half) - 1)
model_variables <- c("half_seconds_remaining", "yardline_100", "down", "ydstogo")
```

XGBoost requires the multinomial categories to be numeric starting at 0

Leave-one-season-out cross-validation

```
xgb_loso_cv_preds <-
  map_dfr(unique(nfl_ep_model_data$season), function(x) {

            # Separate test and training data - scale variables:
            test_data <- nfl_ep_model_data %>% filter(season == x)
            test_data_x <- as.matrix(dplyr::select(test_data, model_variables))
            train_data <- nfl_ep_model_data %>% filter(season != x)
            train_data_x <- as.matrix(dplyr::select(train_data, model_variables))
            train_data_y <- train_data$next_score_label

            xgb_model <- xgboost(data = train_data_x, label = train_data_y, nrounds = 100,
                                 max_depth = 3, eta = 0.3, gamma = 0, colsample_bytree = 1,
                                 min_child_weight = 1, subsample = 1, nthread = 1,
                                 objective = 'multi:softprob', num_class = 7,
                                 eval_metric = 'mlogloss', verbose = 0)

            xgb_preds <- matrix(predict(xgb_model, test_data_x), ncol = 7, byrow = TRUE) %>%
              as_tibble()
            colnames(xgb_preds) <- c("No_Score", "Safety", "Field_Goal", "Touchdown",
                                     "Opp_Safety", "Opp_Field_Goal", "Opp_Touchdown")
            xgb_preds %>%
              mutate(Next_Score_Half = test_data$Next_Score_Half,season = x)
            })
```

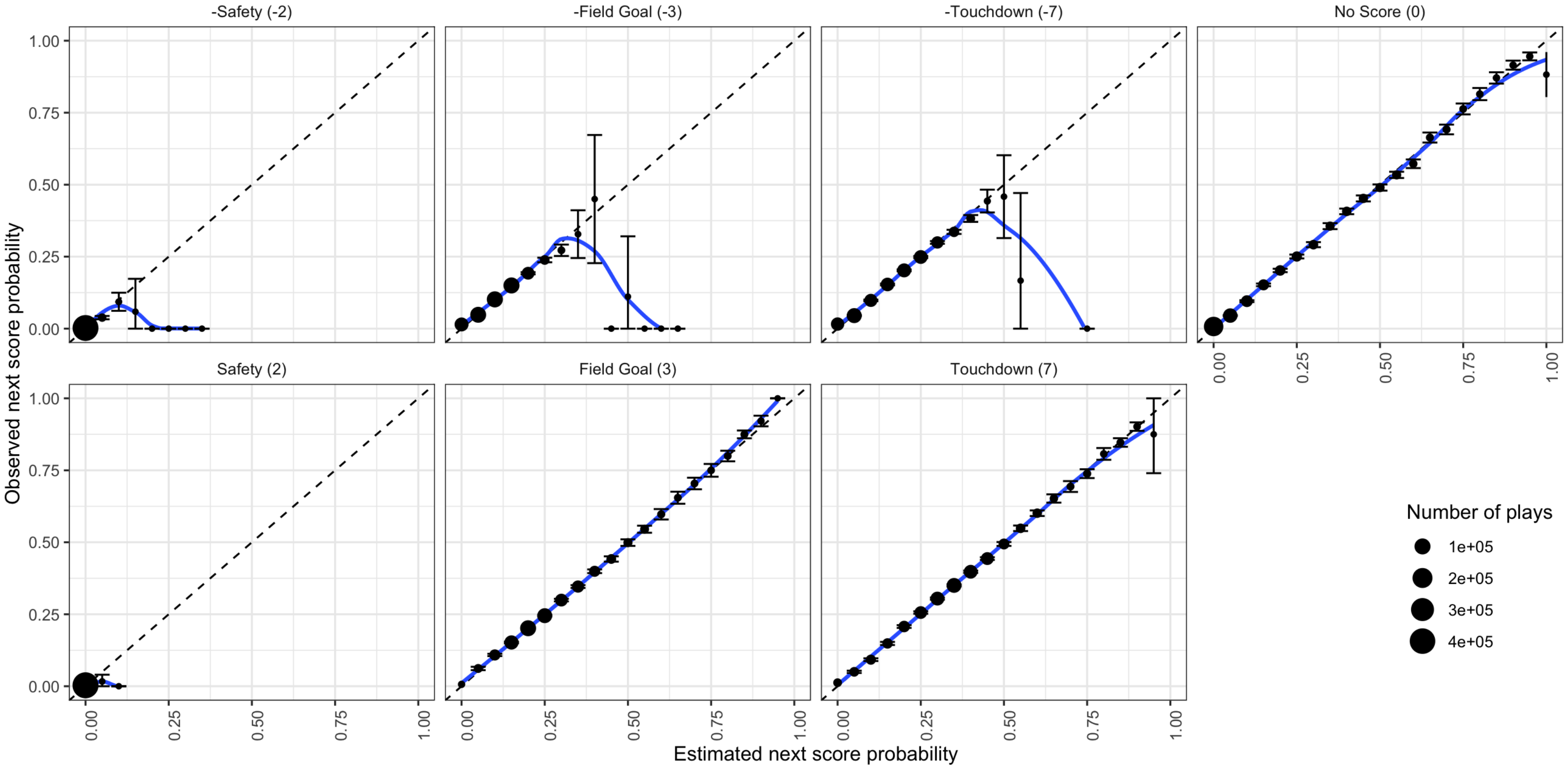
Calibration results for each scoring event

```
ep_cv_loso_calibration_results <- xgb_loso_cv_preds %>%
  pivot_longer(No_Score:Opp_Touchdown,
               names_to = "next_score_type",
               values_to = "pred_prob") %>%
  mutate(bin_pred_prob = round(pred_prob / 0.05) * .05) %>%
  group_by(next_score_type, bin_pred_prob) %>%
  summarize(n_plays = n(),
            n_scoring_event = length(which(Next_Score_Half == next_score_type)),
            bin_actual_prob = n_scoring_event / n_plays,
            bin_se = sqrt((bin_actual_prob * (1 - bin_actual_prob)) / n_plays)) %>%
  ungroup() %>%
  mutate(bin_upper = pmin(bin_actual_prob + 2 * bin_se, 1),
         bin_lower = pmax(bin_actual_prob - 2 * bin_se, 0))
```

Calibration results for each scoring event

```r
ep_cv_loso_calibration_results %>%
  mutate(next_score_type = fct_relevel(next_score_type, "Opp_Safety", "Opp_Field_Goal",
                                       "Opp_Touchdown", "No_Score", "Safety", "Field_Goal",
                                       "Touchdown"),
  next_score_type = fct_recode(next_score_type, "-Field Goal (-3)" = "Opp_Field_Goal",
                               "-Safety (-2)" = "Opp_Safety", "-Touchdown (-7)" = "Opp_Touchdown"
                               "Field Goal (3)" = "Field_Goal", "No Score (0)" = "No_Score",
                               "Touchdown (7)" = "Touchdown", "Safety (2)" = "Safety")) %>%
  ggplot(aes(x = bin_pred_prob, y = bin_actual_prob)) +
  geom_abline(slope = 1, intercept = 0, color = "black", linetype = "dashed") +
  geom_smooth(se = FALSE) +
  geom_point(aes(size = n_plays)) +
  geom_errorbar(aes(ymin = bin_lower, ymax = bin_upper)) + #coord_equal() +
  scale_x_continuous(limits = c(0,1)) +
  scale_y_continuous(limits = c(0,1)) +
  labs(size = "Number of plays", x = "Estimated next score probability",
       y = "Observed next score probability") +
  theme_bw() +
  theme(strip.background = element_blank(),
        axis.text.x = element_text(angle = 90),
        legend.position = c(1, .05), legend.justification = c(1, 0)) +
  facet_wrap(~ next_score_type, ncol = 4)
```

Calibration results for each scoring event



Model Evaluation for Classification

Back to a binary example: NFL completion probability

Binary outcome model: $Y \in \{\text{Incomplete } (0), \text{Complete } (1)\}$

```r
library(tidyverse)
nfl_passing_plays <-
  read_csv("https://shorturl.at/ADMWZ") %>%
  # Only keep rows with passer and receiver information known:
  filter(!is.na(passer_player_id), !is.na(receiver_player_id),
         !is.na(epa), !is.na(air_yards), !is.na(pass_location)) %>%
  # Combine passer and receiver unique IDs:
  mutate(passer_name_id = paste0(passer_player_name, ":", passer_player_id),
         receiver_name_id = paste0(receiver_player_name, ":", receiver_player_id))
```

Create train and test folds based on games:

```r
set.seed(1985)
game_fold_table <- tibble(game_id = unique(nfl_passing_plays$game_id)) %>%
  mutate(game_fold = sample(rep(1:5, length.out = n()), n()))
nfl_passing_plays <- nfl_passing_plays %>% dplyr::left_join(game_fold_table, by = "game_id")
```

Logistic regression review

Generate data of test predictions with particular model:

```r
logit_cv_preds <-
  map_dfr(unique(nfl_passing_plays$game_fold),
          function(test_fold) {
            # Separate test and training data:
            test_data <- nfl_passing_plays %>%
              filter(game_fold == test_fold)
            train_data <- nfl_passing_plays %>%
              filter(game_fold != test_fold)

            # Train model:
            logit_model <- glm(complete_pass ~ yardline_100 + shotgun + air_yards +
                                 pass_location + qb_hit,
                               data = train_data,family = "binomial")

            # Return tibble of holdout results:
            tibble(test_pred_probs = predict(logit_model, newdata = test_data,
                                             type = "response"),
                   test_actual = test_data$complete_pass,
                   game_fold = test_fold)
          })
```

Holdout performance by fold

```
logit_cv_preds %>%
  mutate(test_pred = ifelse(test_pred_probs < .5, 0, 1)) %>%
  group_by(game_fold) %>%
  summarize(mcr = mean(test_pred != test_actual))
```

```
## # A tibble: 5 × 2
##   game_fold   mcr
##       <int> <dbl>
## 1         1 0.284
## 2         2 0.291
## 3         3 0.294
## 4         4 0.281
## 5         5 0.307
```

Let's think more carefully about what's going on here...

Evaluating the prediction threshold

We can really write our classification as a function of some cutoff c:

$$\hat{Y} = \hat{C}(x) = \begin{cases} 1 & \hat{p}(x) > c \\ 0 & \hat{p}(x) \leq c \end{cases}$$

Given the classifications, we can form a confusion matrix:

	predicted events	predicted non-events
actual events	correctly forecasted events	missed events
actual non-events	missed non-events	correctly forecasted non-events

	predicted events	predicted non-events
actual events	True Positive	False Negative
actual non-events	False Positive	True Negative

	predicted events	predicted non-events
actual events	correctly forecasted events	missed events
actual non-events	missed non-events	correctly forecasted non-events

	predicted events	predicted non-events
actual events	True Positive	False Negative
actual non-events	False Positive	True Negative

We want to **maximize** all of the following (positive means 1, negative means 0):

- **Accuracy**: How often is the classifier correct? $\frac{TP+TN}{total}$
- **Precision**: How often is it right for predicted positives? $\frac{TP}{TP+FP}$
- **Sensitivity**, aka true positive rate (TPR) or power: How often does it detect positives? $\frac{TP}{TP+FN}$
- **Specificity**, aka true negative rate (TNR), or 1 - false positive rate (FPR): How often does it detect negatives? $\frac{TN}{TN+FP}$

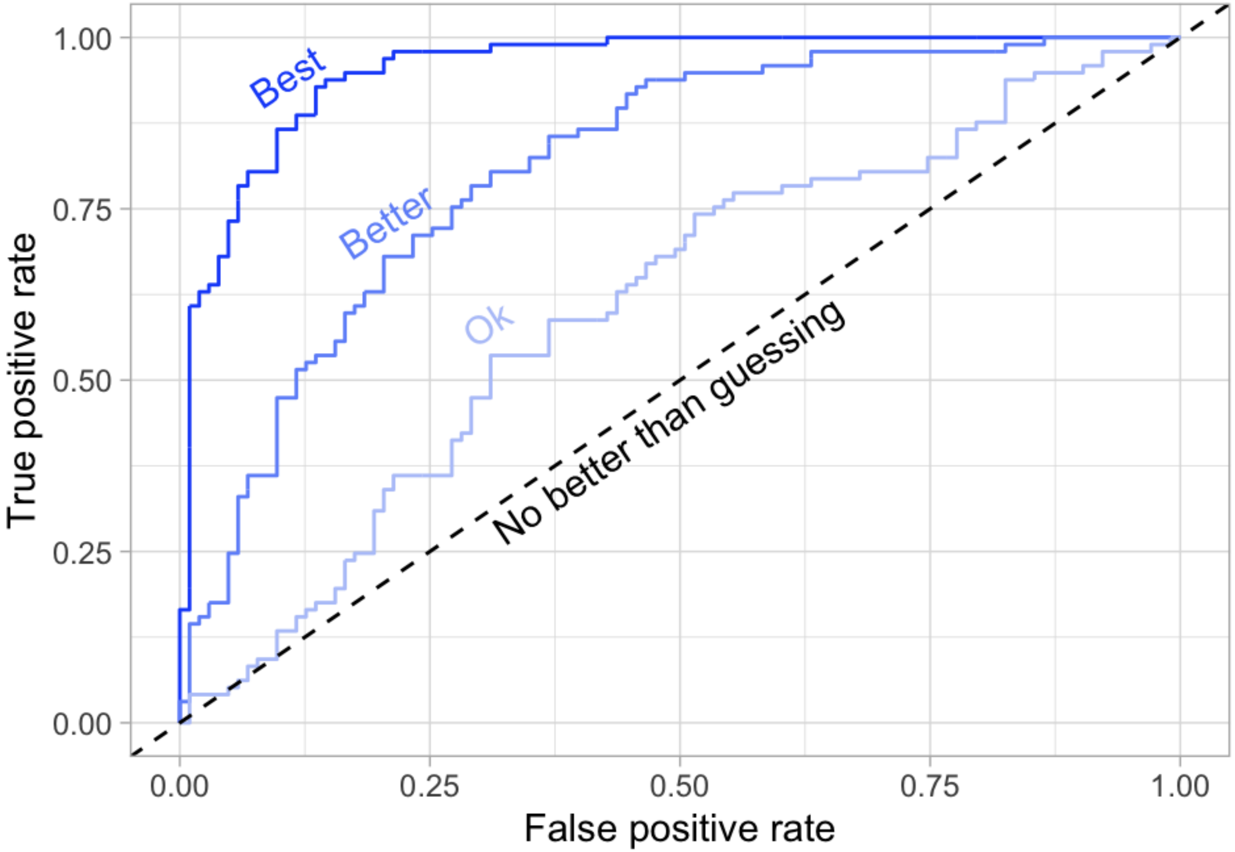
So how do we handle this?

We want to balance with high power and low false positive rate

Receiver Operating Characteristic (ROC) curve

Check all possible values for the cutoff c, plot the power against false positive rate

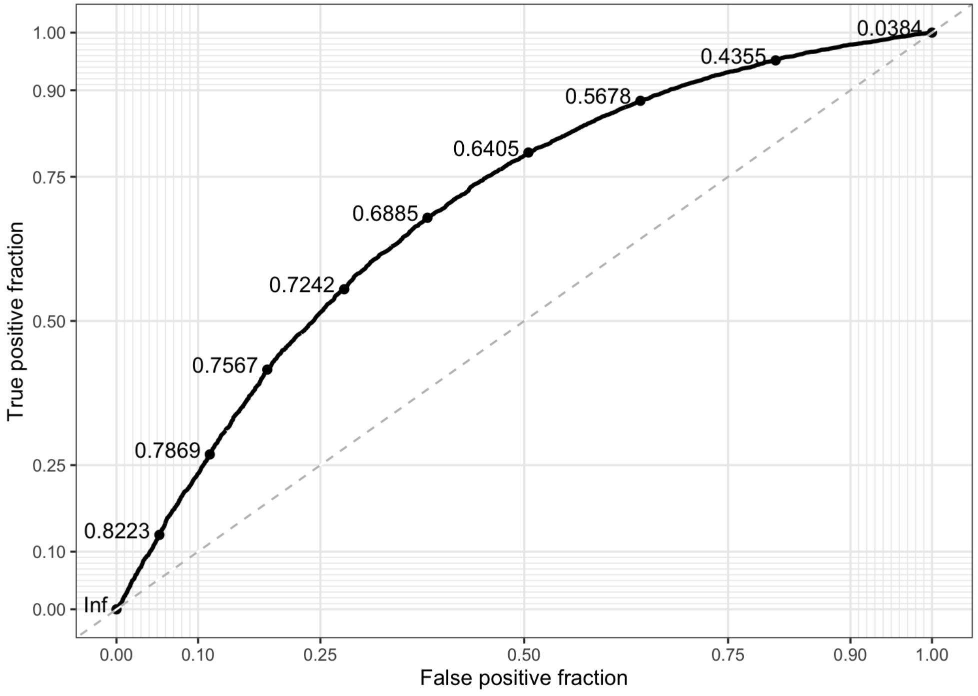
Want to maximize the **area under the curve (AUC)**



`plotROC` and holdout AUC

- `d` stands for disease status (the outcome)
- `m` stands for marker (the prediction)

```
library(plotROC)
logit_cv_preds %>%
  ggplot() +
  geom_roc(aes(d = test_actual,
               m = test_pred_probs),
           labelround = 4) +
  style_roc() +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "gray") +
  labs(color = "Test fold")
with(logit_cv_preds,
     MLmetrics::AUC(test_pred_probs, test_act
```

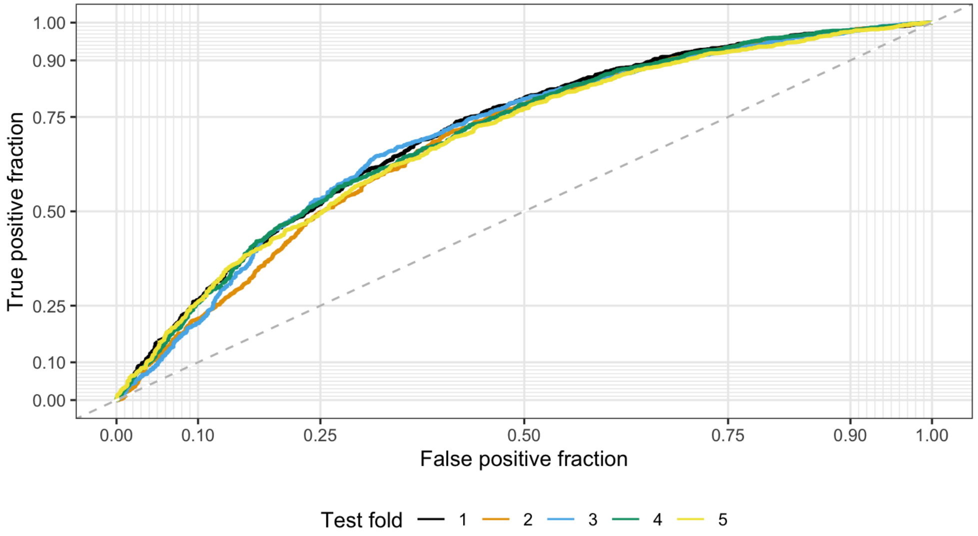



```
## [1] 0.6957778
```

`plotROC` and holdout AUC by test fold

```r
logit_cv_preds %>%
  ggplot() +
  geom_roc(aes(d = test_actual,
               m = test_pred_probs,
               color = as.factor(game_fold)),
           n.cuts = 0) +
  style_roc() +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "gray") +
  ggthemes::scale_color_colorblind() +
  labs(color = "Test fold") +
  theme(legend.position = "bottom")
logit_cv_preds %>% group_by(game_fold) %>%
  summarize(auc = MLmetrics::AUC(test_pred_pr
                                 test_actual)
```

There is definitely room for improvement...




```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.707
## 2         2 0.686
## 3         3 0.699
## 4         4 0.700
## 5         5 0.690
```

Tree-based approach?

We need to first convert categorical variables into dummy indicators:

```
model_data <- nfl_passing_plays %>%
  mutate(play_id = 1:n(),
         complete_pass = as.factor(complete_pass)) %>%
  dplyr::select(play_id, complete_pass, yardline_100, shotgun, air_yards, qb_hit,
                game_fold, pass_location) %>%
  mutate(pass_location_val = 1) %>%
  pivot_wider(id_cols = play_id:game_fold,
              names_from = pass_location, values_from = pass_location_val,
              values_fill = 0) %>%
  dplyr::select(-play_id)
```

Random forests using probability forest

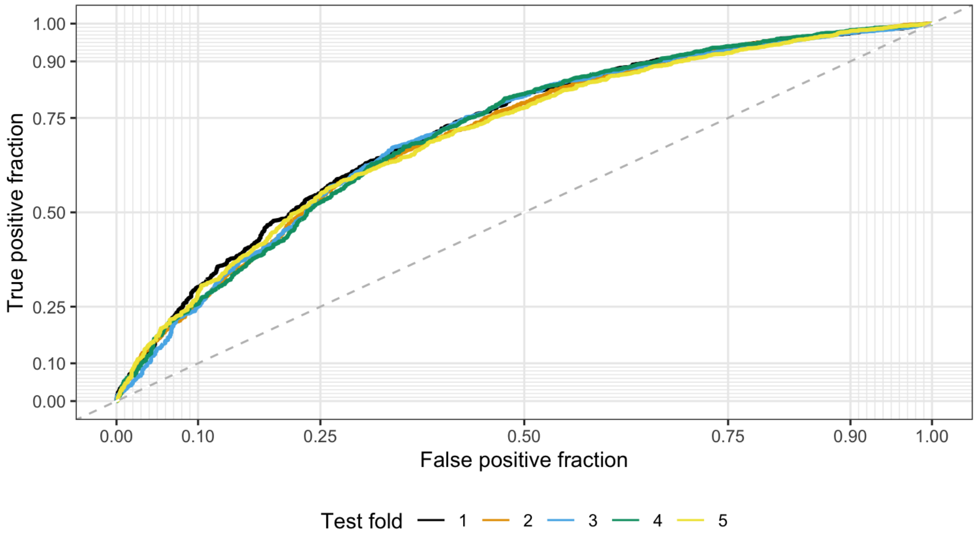
For each tree compute class proportion in terminal node, then take average across all trees

```
library(ranger)
rf_prob_cv_preds <-
  map_dfr(unique(model_data$game_fold),
          function(test_fold) {

            # Separate test and training data - scale variables:
            test_data <- model_data %>% filter(game_fold == test_fold)

            train_data <- model_data %>% filter(game_fold != test_fold)

            rf_prob_model <-
              ranger(complete_pass ~ ., data = dplyr::select(train_data, -game_fold),
                     probability = TRUE)

            # Return tibble of holdout results:
            tibble(test_pred_probs =
                     as.numeric(predict(rf_prob_model, data = test_data,
                                        type = "response")$predictions[,2]),
                   test_actual = as.numeric(test_data$complete_pass) - 1,
                   game_fold = test_fold)
          })
```

Random forests using probability forest

```r
rf_prob_cv_preds %>%
  ggplot() +
  geom_roc(aes(d = test_actual,
               m = test_pred_probs,
               color = as.factor(game_fold)),
           n.cuts = 0) +
  style_roc() +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "gray") +
  ggthemes::scale_color_colorblind() +
  labs(color = "Test fold") +
  theme(legend.position = "bottom")
rf_prob_cv_preds %>% group_by(game_fold) %>%
  summarize(auc = MLmetrics::AUC(test_pred_pr
                                 test_actual)
```

Looks like just a modest improvement




```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.717
## 2         2 0.705
## 3         3 0.707
## 4         4 0.708
## 5         5 0.704
```

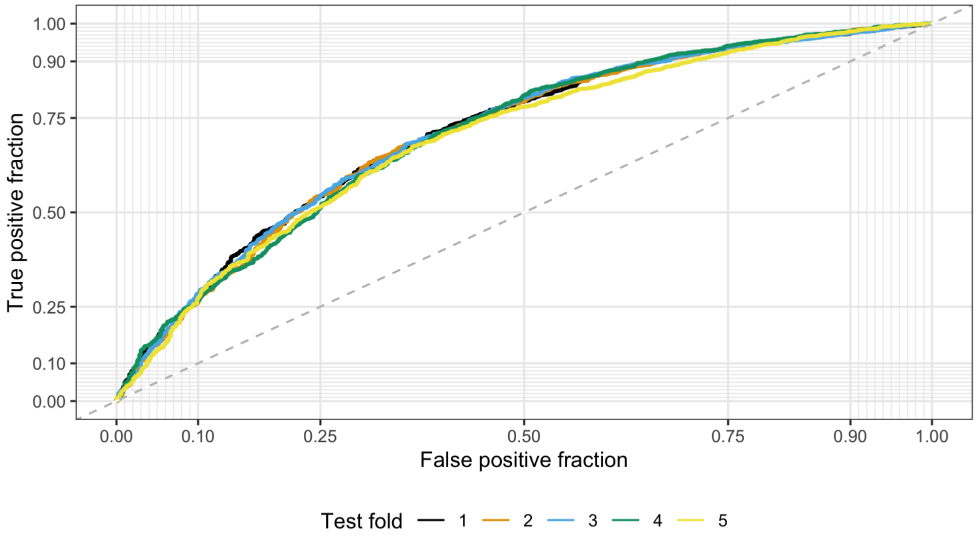
XGBoost!

```
library(xgboost)
xgb_cv_preds <-
  map_dfr(unique(model_data$game_fold),
          function(test_fold) {
            # Separate test and training data - scale variables:
            test_data <- model_data %>% filter(game_fold == test_fold)
            test_data_x <- as.matrix(dplyr::select(test_data, -complete_pass, -game_fold))
            train_data <- model_data %>% filter(game_fold != test_fold)
            train_data_x <- as.matrix(dplyr::select(train_data, -complete_pass, -game_fold))
            train_data_y <- as.numeric(train_data$complete_pass) - 1

            xgb_model <- xgboost(data = train_data_x, label = train_data_y,
                                 nrounds = 100, max_depth = 3, eta = 0.3,
                                 gamma = 0, colsample_bytree = 1, min_child_weight = 1,
                                 subsample = 1, nthread = 1,
                                 objective = 'binary:logistic', eval_metric = 'auc',
                                 verbose = 0)

            # Return tibble of holdout results:
            tibble(test_pred_probs =
                     as.numeric(predict(xgb_model, newdata = test_data_x, type = "response")),
                   test_actual = as.numeric(test_data$complete_pass) - 1,
                   game_fold = test_fold)
          })
```

XGBoost

```
xgb_cv_preds %>%
  ggplot() +
  geom_roc(aes(d = test_actual,
               m = test_pred_probs,
               color = as.factor(game_fold)),
           n.cuts = 0) +
  style_roc() +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "gray") +
  ggthemes::scale_color_colorblind() +
  labs(color = "Test fold") +
  theme(legend.position = "bottom")
xgb_cv_preds %>% group_by(game_fold) %>%
  summarize(auc = MLmetrics::AUC(test_pred_pr
                                  test_actual)
```

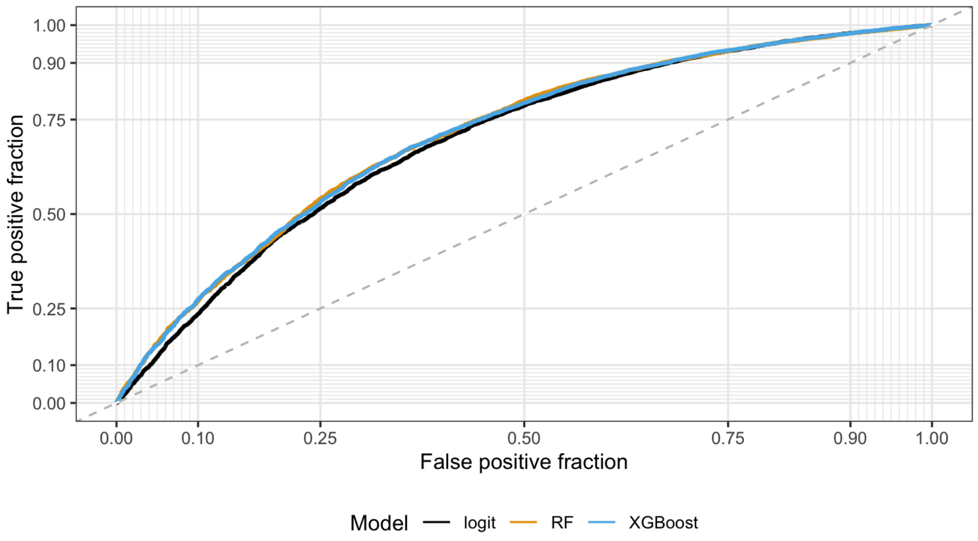
Should actually tune this more...




```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.712
## 2         2 0.708
## 3         3 0.711
## 4         4 0.708
## 5         5 0.697
```

All together now...

```r
bind_rows(
  mutate(logit_cv_preds, type = "logit"),
  mutate(rf_prob_cv_preds, type = "RF"),
  mutate(xgb_cv_preds, type = "XGBoost")) %>%
  ggplot() +
  geom_roc(aes(d = test_actual,
               m = test_pred_probs,
               color = type),
           n.cuts = 0) +
  style_roc() +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dashed",
              color = "gray") +
  ggthemes::scale_color_colorblind() +
  labs(color = "Model") +
  theme(legend.position = "bottom")
```



Pretty similar performance across all models...

Explaining predictions with SHAP-values

SHAP-values are based on Shapley values (an idea from game theory) and are used to measure the contributions from each feature in the model to the prediction for an individual observation

Shapley value ϕ_i^j for feature value j for observation i can be interpreted as:

- the value of feature j contributed ϕ_i^j to the prediction of observation i compared to the average prediction for the dataset
- linear regression coefficients function in the same way

Can use them in multiple ways:

- View total importance: $\frac{1}{n}\sum|\phi_i^j|$
- View distribution of ϕ_i^j for each feature
- Plot ϕ_i^j against feature value for partial dependence

SHAPforxgboost

Fit model on full data then extract SHAP-values with `SHAPforxgboost`

```
train_data_x <- as.matrix(dplyr::select(model_data, -complete_pass, -game_fold))
train_data_y <- as.numeric(model_data$complete_pass) - 1
xgb_model <- xgboost(data = train_data_x, label = train_data_y, nrounds = 100, max_depth = 3,
                     eta = 0.3, gamma = 0, colsample_bytree = 1, min_child_weight = 1,
                     subsample = 1, nthread = 1, objective = 'binary:logistic',
                     eval_metric = 'auc', verbose = 0)
library(SHAPforxgboost)
shap_value_list <- shap.values(xgb_model, X_train = train_data_x)
shap.plot.summary.wrap1(xgb_model, X = train_data_x)
```