Advanced topics

More fun with classification

July 22nd, 2021

NFL completion probability

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

```
library(tidyverse)
nfl_passing_plays <-
  read_csv("http://www.stat.cmu.edu/cmsac/sure/2021/materials/data/eda_projects/nfl_passing_plays
  # 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:

```
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:

```
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
                               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 0.297
## 2         2 0.298
## 3         3 0.293
## 4         4 0.277
## 5         5 0.296
```

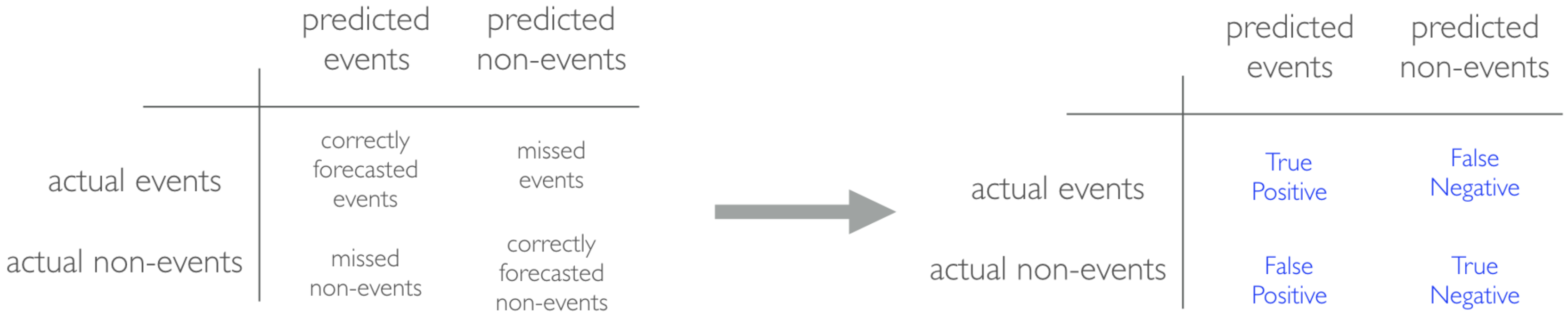
Let's think more carefully about what's going on here...

Evaluating the prediction threshold

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

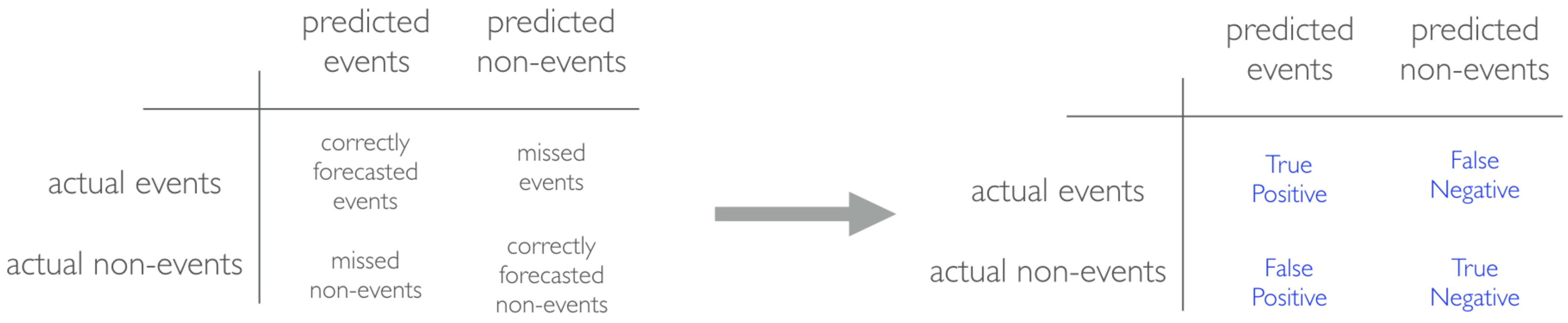
$$\hat{Y} = \hat{C}(x) = \begin{cases} 1 & \hat{p}(x) > c \\ 0 & \hat{p}(x) \leq c \end{cases}$$

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? $\frac{TN}{TN+FP}$

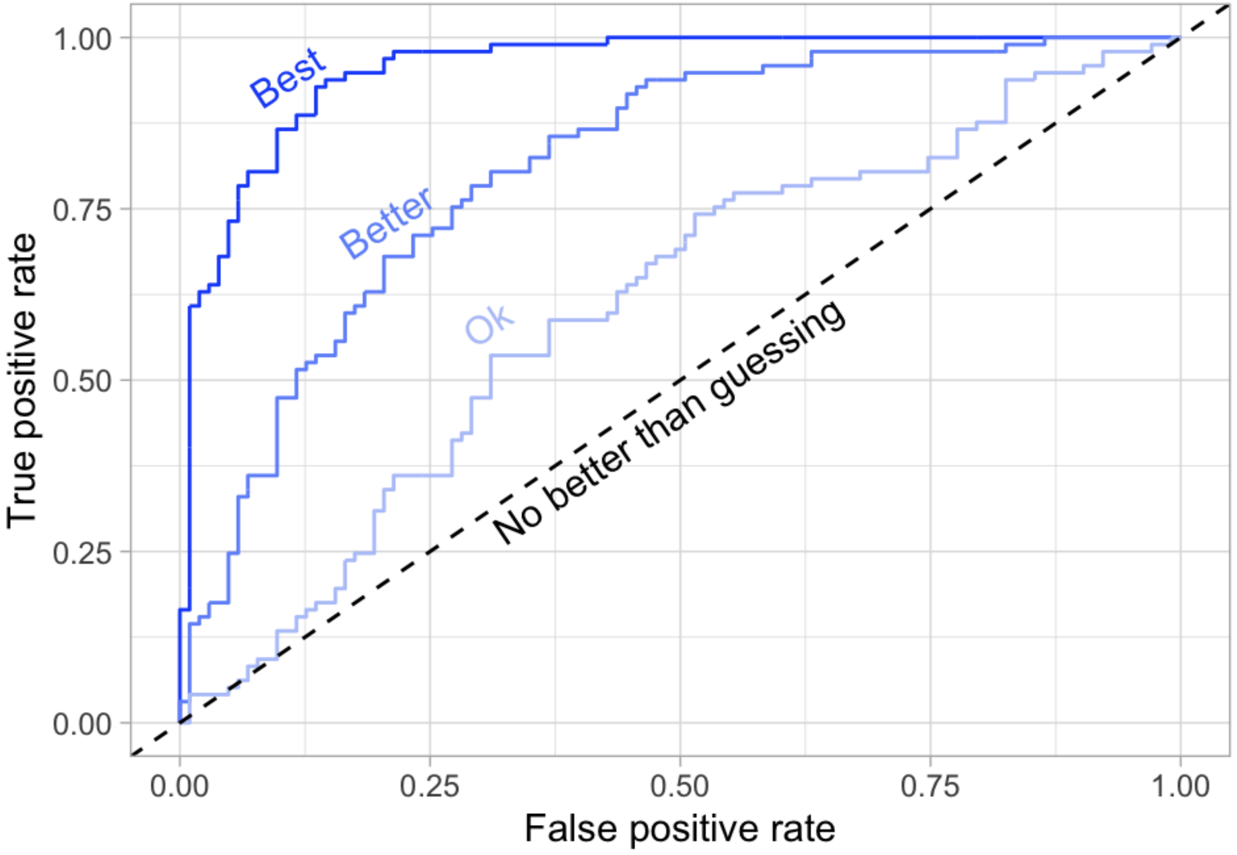
So how do we handle this?

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

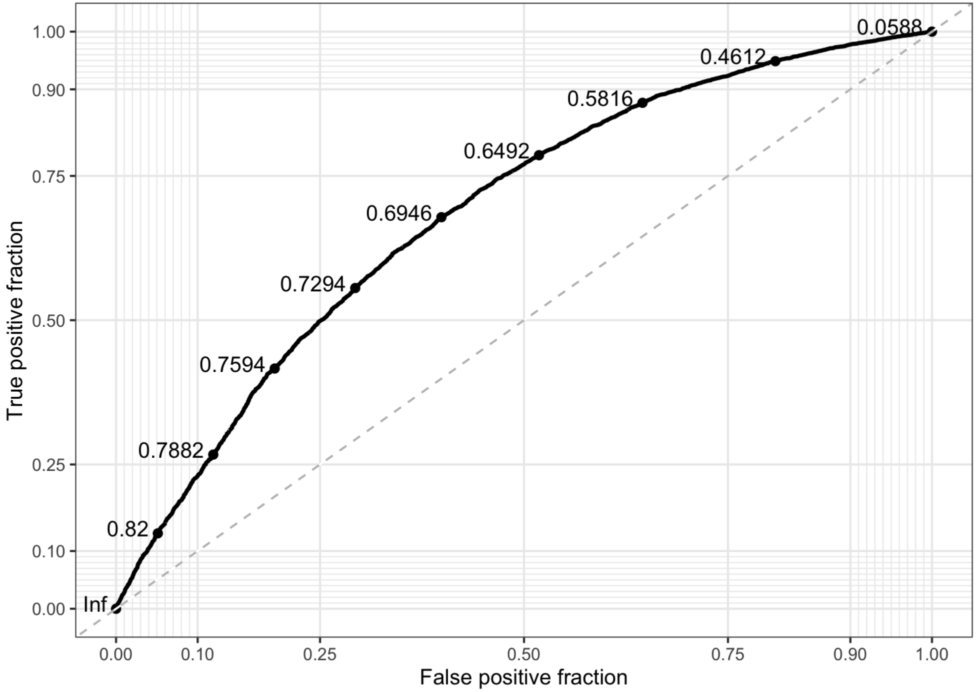
Want to maximize the **area under the curve (AUC)**



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, linet
  labs(color = "Test fold")
with(logit_cv_preds,
     MLmetrics::AUC(test_pred_probs, test_act
```

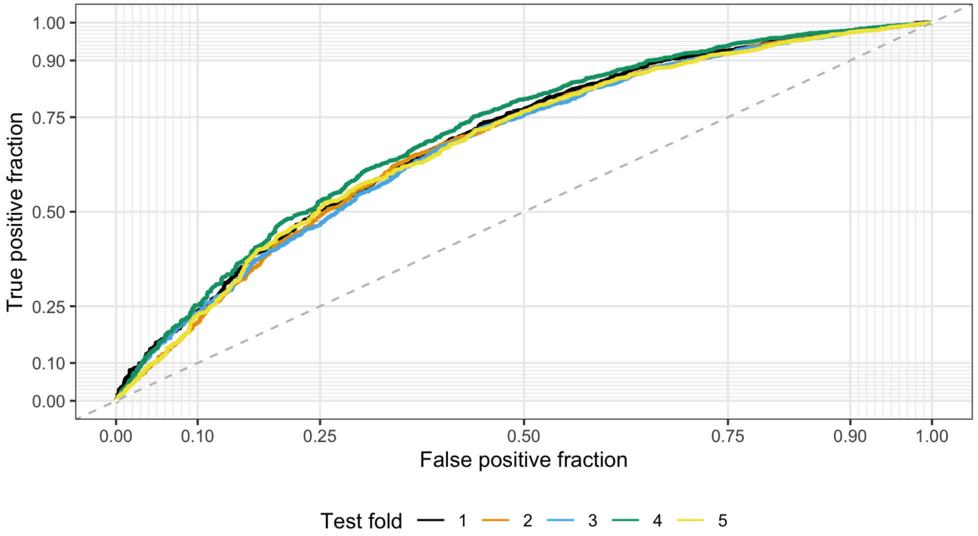


```
## [1] 0.6864571
```

plotROC and holdout AUC by test fold

```
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, linet
  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
```

There is definitely room for improvement...



```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.690
## 2         2 0.679
## 3         3 0.678
## 4         4 0.705
## 5         5 0.681
```

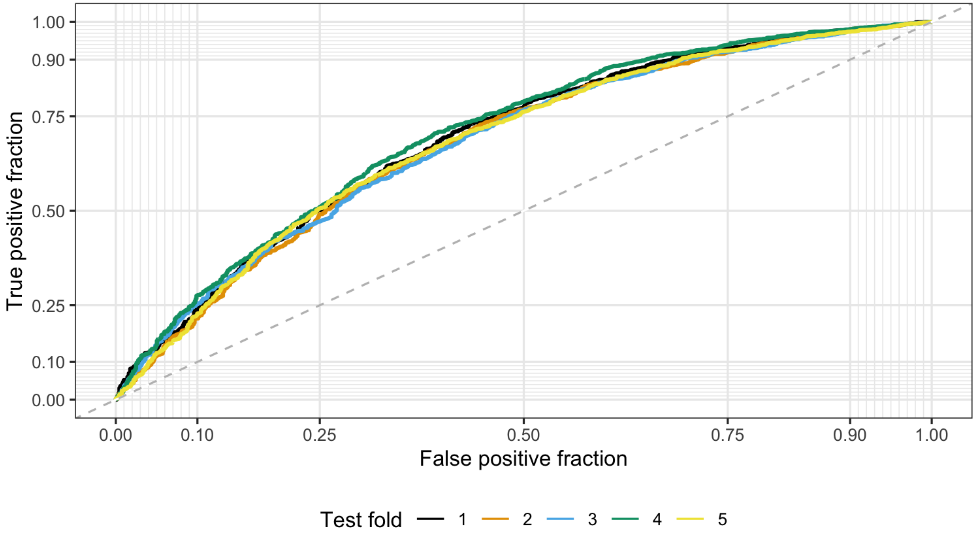
Add in varying intercepts with `glmer`?

```r
library(lme4)
glmer_cv_preds <-
  map_dfr(unique(nfl_passing_plays$game_fold),
          function(test_fold) {

            # Separate test and training data - scale variables:
            test_data <- nfl_passing_plays %>% filter(game_fold == test_fold) %>%
              mutate(yardline_100 = scale(yardline_100), air_yards = scale(air_yards))

            train_data <- nfl_passing_plays %>% filter(game_fold != test_fold)  %>%
              mutate(yardline_100 = scale(yardline_100), air_yards = scale(air_yards))

            # Train model:
            glmer_model <- glmer(complete_pass ~ yardline_100 + shotgun + air_yards +
                                  pass_location + qb_hit +
                                  (1|passer_name_id) + (1|receiver_name_id),
                                 data = train_data, family = "binomial")

            # Return tibble of holdout results:
            tibble(test_pred_probs = predict(glmer_model, newdata = test_data,
                                             type = "response",
                                             allow.new.levels = TRUE),
                   test_actual = test_data$complete_pass,
                   game_fold = test_fold)
```

Add in varying intercepts with `glmer`?

```
glmer_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, linet
  ggthemes::scale_color_colorblind() +
  labs(color = "Test fold") +
  theme(legend.position = "bottom")
glmer_cv_preds %>% group_by(game_fold) %>%
  summarize(auc = MLmetrics::AUC(test_pred_pr
```

Looks like player-level effects do not help!



```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.692
## 2         2 0.681
## 3         3 0.682
## 4         4 0.705
## 5         5 0.684
```

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

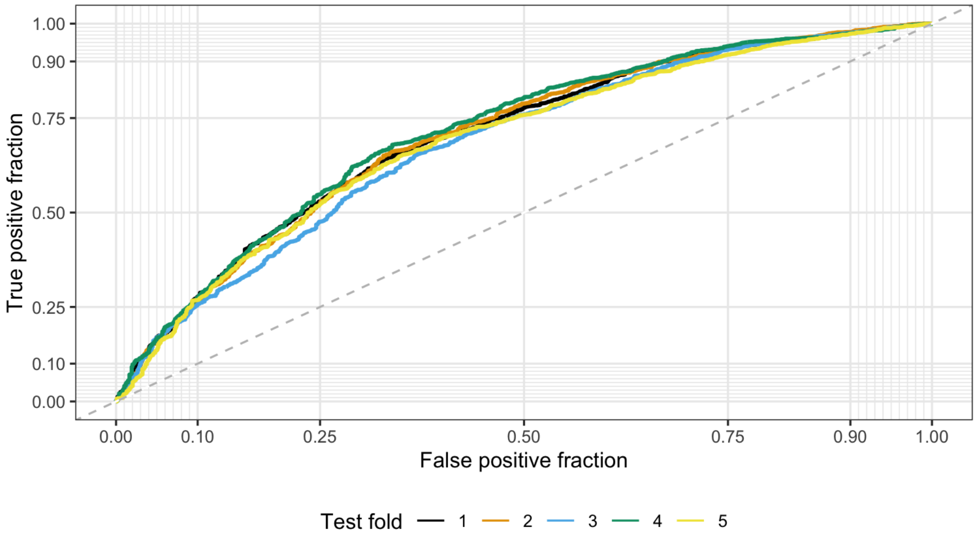
For each tree compute class proportion in terminal node, then take average across all trees

```r
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

```
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, linet
  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
```

Looks like just a modest improvement



```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.702
## 2         2 0.703
## 3         3 0.683
## 4         4 0.713
## 5         5 0.690
```

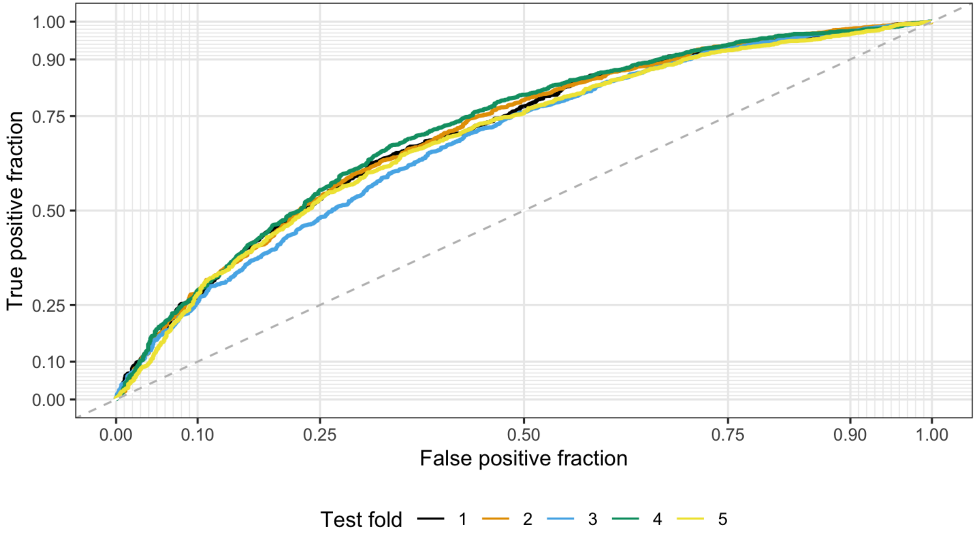
XGBoost!

```r
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, linet
  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
```

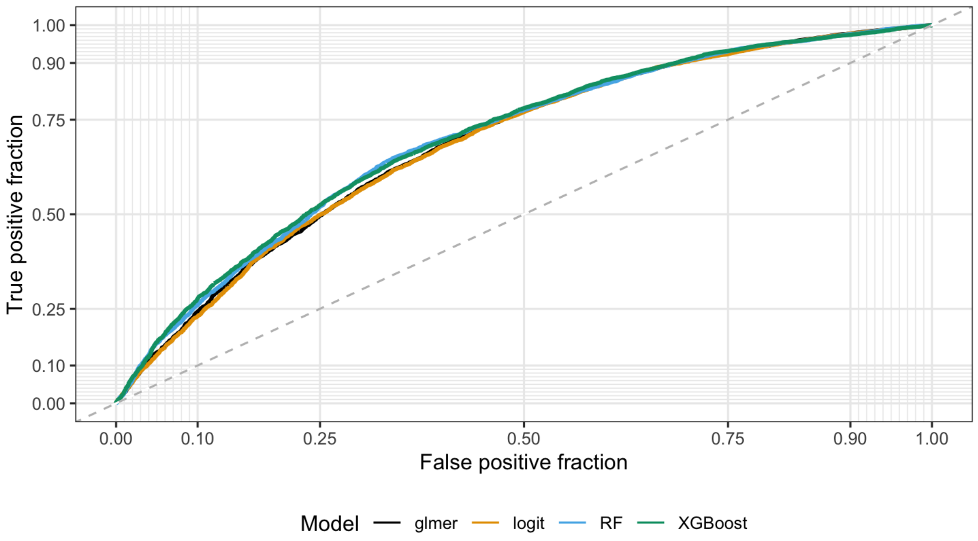
Should actually tune this more...



```
## # A tibble: 5 × 2
##   game_fold   auc
##       <int> <dbl>
## 1         1 0.704
## 2         2 0.706
## 3         3 0.683
## 4         4 0.715
## 5         5 0.692
```

All together now...

```
bind_rows(
  mutate(logit_cv_preds, type = "logit"),
  mutate(glmer_cv_preds, type = "glmer"),
  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, linet
  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

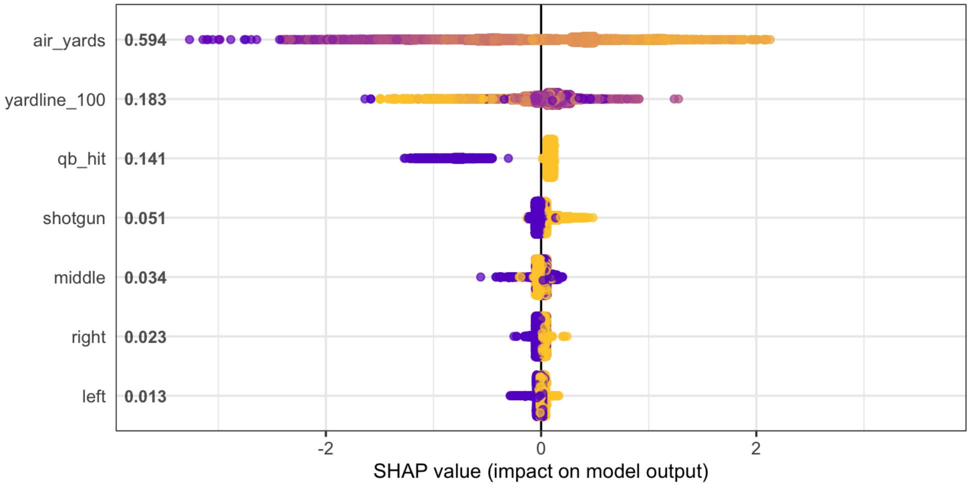
Can use them in multiple ways:

- 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

```
library(xgboost)
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
                     gamma = 0, colsample_bytree = 1, min_child_weight = 1, subsample = 1, nthrea
                     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)
```



Multinomial classification with XGBoost

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

```
nfl_ep_model_data <- read_rds("http://www.stat.cmu.edu/cmsac/sure/2021/materials/data/model_pbp_
  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

```
library(xgboost)
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_de
                                 eta = 0.3, gamma = 0, colsample_bytree = 1, min_child_weight =
                                 subsample = 1, nthread = 1, objective = 'multi:softprob', num_c
                                 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)
            })
```

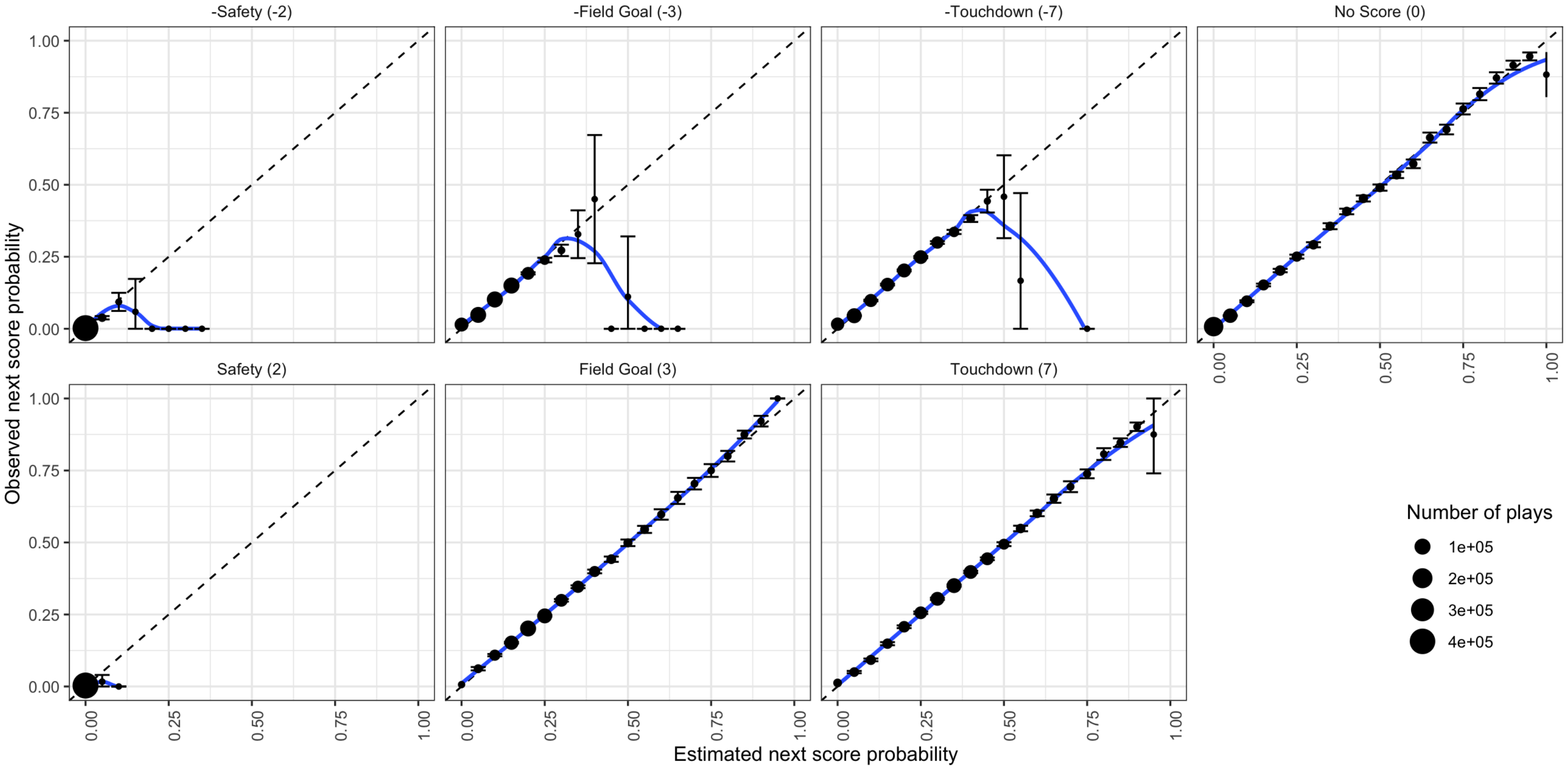
Calibration results for each scoring event

```
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", "Touc
  next_score_type = fct_recode(next_score_type, "-Field Goal (-3)" = "Opp_Field_Goal", "-Safety 
                               "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


-Safety (-2)
-Field Goal (-3)
-Touchdown (-7)
No Score (0)
Safety (2)
Field Goal (3)
Touchdown (7)
Observed next score probability
Estimated next score probability
Number of plays
1e+05
2e+05
3e+05
4e+05