College Football Playoff Predictor

Zayne Maughan

Introduction:

American Football dominates the culture and backbone of the American community. College football allows for closer connection and community to be present than professional teams often can because colleges are often in much closer proximity, and one can always be a fan of their alma mater wherever they are. In the recent past, there have been many changes to the college football landscape. Some of these include the transfer portal, NIL, and the playoffs. The playoffs have produced some of the most exciting games in college football history. The playoffs have also come to much scrutiny as they were limited to four teams. Because of this many good teams have been left out and felt as though they should have been in. The reason why this is intriguing is that the decision of which team should be in has been left to the hand of a playoff committee which has rotated who is on the committee over the last ten years. This committee has held firm that some of the factors they most consider when determining the playoff are strength of schedule, and conference champions. The main task of this analysis is to create a predictive classification model that predicts which four teams would make it to the playoffs.

*** Note: the playoff model changed in 2024. It now has 12 teams competing for the national title. I did not change this for the final year of data. I instead included the final four teams from the 2024 playoff.

DATA:

The data for this analysis comes from the cfbfastR package. This package allows a user with API keys to query their data and get college football statistics and play-by-play data. The four main functions of this package that I used to create a dataset for prediction are: *cfbd_rankings*, this query provides the rankings for a given year, other optional arguments are taken to shorten the desired data, *cfbd_game_team_stats*, this query goes the game by game statistics of a team in a given year, *cfbd_season_stats_team* and *cfbd_season_stats_advanced*, these give seasonal statistics for each team over the course of a given team.

These queries were used along with some other feature engineering as follows:

```
get_ranked_wins_all_teams <- function(year) {
  # Preload all game stats and rankings for the season
  all_weekly_games <- purrr::map_dfr(1:13, function(week_num) {
    week_data <- cfbd_game_team_stats(year, week = week_num, division = "fbs") |>
      dplyr::filter(conference %in% c(
        "American Athletic",
        "FBS Indepenents", "Big 12", "SEC", "ACC",
        "Conference USA", "Pac-12", "Southern",
        "Sun Belt", "Big Ten", "Mountain West",
        "Mid-American"
      ))
    week_data$week <- week_num # Add week for later filtering
    week_data
  })

  all_rankings <- purrr::map_dfr(1:13, function(week_num) {
    cfbd_rankings(year, week = week_num) |>
```

```
      dplyr::filter(poll == "AP Top 25") |>
      dplyr::mutate(week = week_num)
  })

  # Now compute ranked wins per team
  all_weekly_games |>
    dplyr::mutate(
      result = points > points_allowed,
      ranked_win = as.integer(result & opponent %in% all_rankings$school)
    ) |>
    dplyr::group_by(school) |>
    dplyr::summarise(ranked_wins = sum(ranked_win, na.rm = TRUE), .groups = "drop")
}


full_data <- purrr::map_dfr(c(2014:2019, 2021:2024), function(year) {
  message("Processing year: ", year)
  Sys.sleep(1.5)

  team_stats <- cfbd_stats_season_team(year)
  adv_stats <- cfbd_stats_season_advanced(year)
  ranked_wins_df <- get_ranked_wins_all_teams(year)

  dplyr::inner_join(adv_stats, team_stats, by = "team") |>
    dplyr::left_join(ranked_wins_df, by = c("team" = "school")) |>
    dplyr::mutate(ranked_wins = tidyr::replace_na(ranked_wins, 0))
})

playoffs_2014 <- c("Alabama", "Oregon", "Florida State", "Ohio State")
playoffs_2015 <- c("Clemson", "Alabama", "Michigan State", "Oklahoma")
playoffs_2016 <- c("Alabama", "Clemson", "Ohio State", "Washington")
playoffs_2017 <- c("Clemson", "Georgia", "Oklahoma", "Alabama")
playoffs_2018 <- c("Alabama", "Clemson", "Notre Dame", "Oklahoma")
playoffs_2019 <- c("LSU", "Ohio State", "Clemson", "Oklahoma")
playoffs_2021 <- c("Alabama", "Michigan", "Georgia", "Cincinnati")
playoffs_2022 <- c("Georgia", "Michigan", "TCU", "Ohio State")
playoffs_2023 <- c("Michigan", "Washington", "Texas", "Alabama")
playoffs_2024 <- c("Penn State", "Texas", "Notre Dame", "Ohio State")


full_data |>
  dplyr::mutate(
    playoff = dplyr::case_when(
      (year == 2014 & team %in% playoffs_2014) ~ 1,
      (year == 2015 & team %in% playoffs_2015) ~ 1,
      (year == 2016 & team %in% playoffs_2016) ~ 1,
      (year == 2017 & team %in% playoffs_2017) ~ 1,
      (year == 2018 & team %in% playoffs_2018) ~ 1,
      (year == 2019 & team %in% playoffs_2019) ~ 1,
      (year == 2021 & team %in% playoffs_2021) ~ 1,
      (year == 2024 & team %in% playoffs_2022) ~ 1,
      (year == 2023 & team %in% playoffs_2023) ~ 1,
      (year == 2024 & team %in% playoffs_2024) ~ 1,
```

```
      .default = 0
    )
  ) -> full_data

# There were some NAs that could be added with a zero padded statistic
modeling_data <- full_data %>%
  tidyr::replace_na(list(
    passes_intercepted = c(0),
    passes_intercepted_yds = c(0),
    passes_intercepted_TDs = 0
  )) |>
  dplyr::mutate(playoff = factor(ifelse(playoff == 1, "True", "False"),
    levels = c("False", "True")
  ))
```

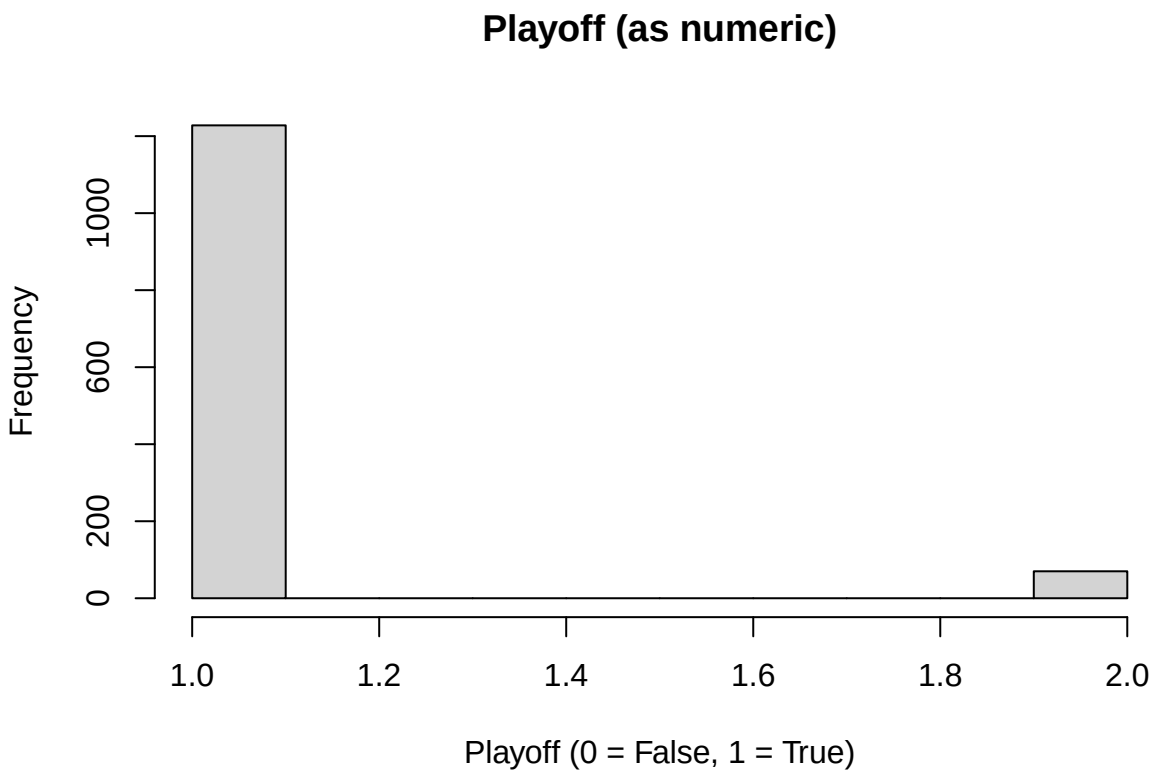
There was not a cohesive strength of schedule ranking or list. Playing and beating multiple ranked teams would significantly improve a teams opportunity to get into the playoffs and as such it was included and created to prove the strength of schedule factor.

An additional step was taken to ensure greater balances between classes. This can be seen in the modeling section.

EDA The combined seasonal statistics produced a data table with 115 variables and 1298 observations. Since the size is so large it would be hard to visualize all variables. Here is a summary of the first 5 predictors:

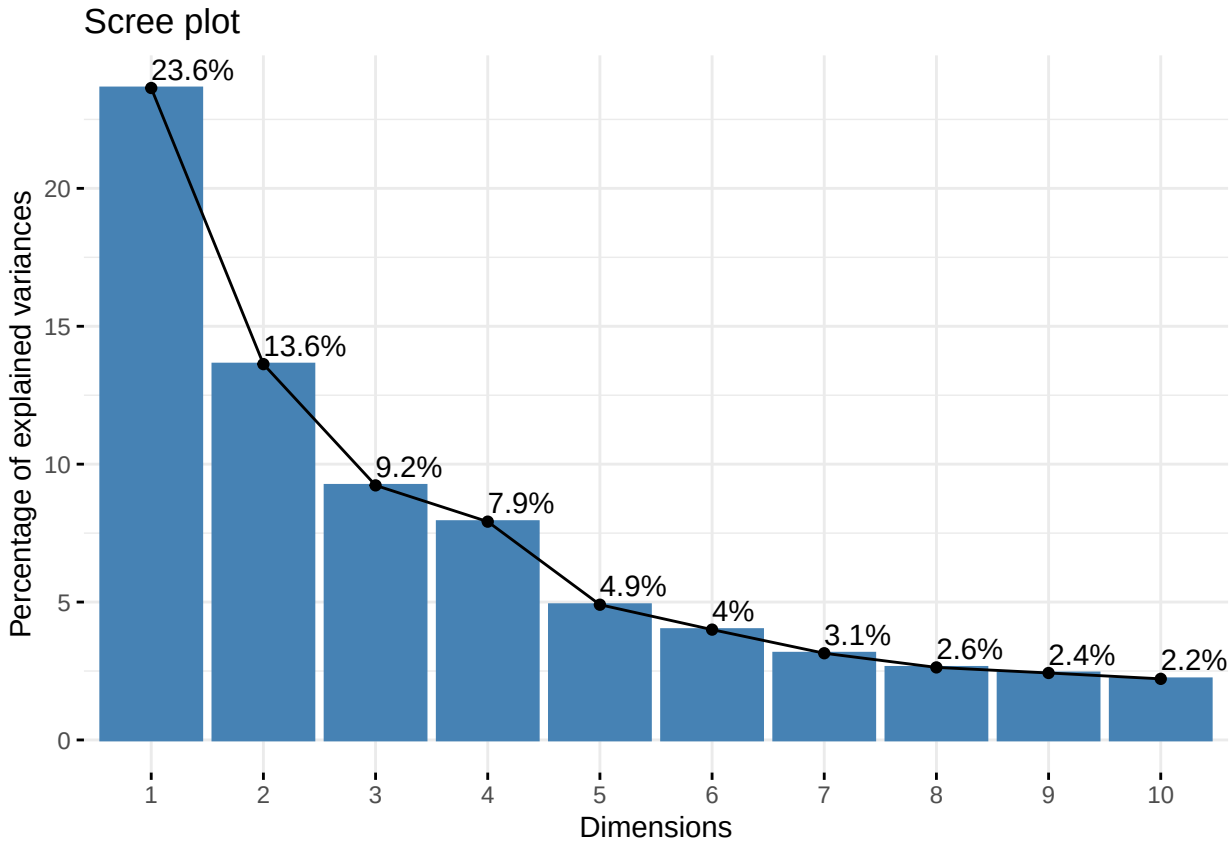
```
#>  conference.x         off_plays        off_drives
#>  Length:1298        Min.   : 613.0   Min.   :112
#>  Class :character   1st Qu.: 821.2   1st Qu.:149
#>  Mode  :character   Median : 881.0   Median :158
#>                     Mean   : 884.7   Mean   :159
#>                     3rd Qu.: 947.0   3rd Qu.:168
#>                     Max.   :1231.0   Max.   :212
```

Visualizing the response variable:



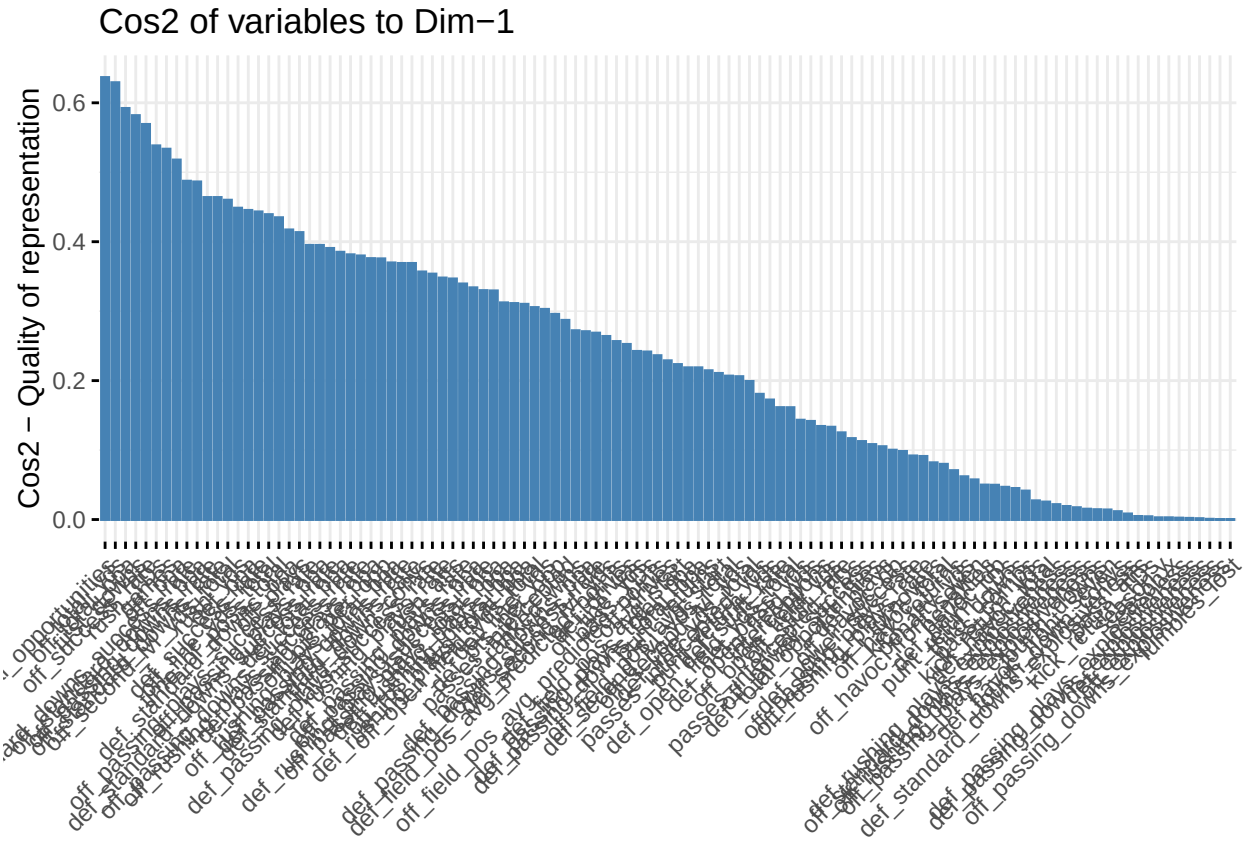
Correlation Due to the size of the data, dimensionality reduction deemed important to try. I decided to try and do PCA on the numeric predictors. This would have been everything besides the team name, the conference, and the response variable.

```
pca_data <- modeling_data %>%
  select(-c("playoff", "conference.x", "conference.y", "team"))

pca <- princomp(scale(pca_data))

fviz_eig(pca, addlabels = TRUE)
```



I wanted to see if the conducting PCA on the data showed there was a few dimensions that described over 90% of the variance. However, looking at this graph one can see that the variance still carried over many dimensions and could be reduced but not by much. Still, I wanted to see if there were any important variables that I could keep.

```
fviz_cos2(pca, choice = "var")
```



There are a few that do not seem to add much importance to the first dimension in terms of their cosine squared value. I started to remove some of those that did not add much benefit.

I also checked certain variables that are highly correlated and removed some of those from the training data.

```
############################

# Example correlation matrix
cor_matrix <- cor(pca_data)

# Set a threshold
threshold <- 0.8

# Zero out self-correlations (optional)
diag(cor_matrix) <- NA

# Get the upper triangle only (to avoid duplicates)
upper_tri <- upper.tri(cor_matrix)

# Apply threshold filter
high_cor_indices <- which(abs(cor_matrix) > threshold & upper_tri, arr.ind = TRUE)

# Create a named vector
high_cor_values <- cor_matrix[high_cor_indices]
names(high_cor_values) <- apply(high_cor_indices, 1, function(idx) {
  paste(rownames(cor_matrix)[idx[1]], colnames(cor_matrix)[idx[2]], sep = " - ")
})
```

```
# View the named vector
print(head(high_cor_values))
#>                             off_ppa - off_total_ppa
#>                                            0.9814672
#>                          off_ppa - off_success_rate
#>                                            0.8781833
#>                    off_total_ppa - off_success_rate
#>                                            0.8884393
#>                       off_stuff_rate - off_line_yds
#>                                           -0.8337215
#>       off_line_yds_total - off_second_lvl_yds_total
#>                                            0.9151386
#> off_second_lvl_yds_total - off_open_field_yds_total
#>                                            0.8309204
```

Modeling One large issue that arose when trying to predict the final four teams or the four teams that would make it to the playoffs is that there are 134 teams and only four make it. This naturally causes imbalanced classes for prediction. One method that has been discussed has been SMOTE. I decided to implement this methodology to create greater balance between the classes. This implementation was made smooth by the **recipes** package.

```
# Split the data into training and test sets.
train_data <- modeling_data |>
  dplyr::filter(!season.x %in% c(2023, 2024)) |>
  select(-c(conference.x, conference.y))

test_data <- modeling_data |>
  dplyr::filter(season.x %in% c(2023, 2024)) |>
  dplyr::select(-c(conference.x, conference.y))
```

After splitting the data, I created a "recipe" to be used for SMOTE and the train and test sets for the modeling.

```
######################## SMOTE FOR THE DATA AND THE MODEL ######################
smote_subset_recipe <- recipe(playoff ~ ., data = train_data) |>
  update_role(team, new_role = "id") |>
  step_rm(
    off_ppa, off_total_ppa, off_field_pos_avg_predicted_points,
    off_havoc_total, off_success_rate, off_standard_downs_rate,
    off_rushing_plays_ppa, off_rushing_plays_rate, off_drives,
    def_drives, def_ppa, def_field_pos_avg_predicted_points,
    def_havoc_total, def_success_rate, def_standard_downs_rate,
    def_rushing_plays_ppa, def_rushing_plays_rate,
    def_passing_downs_total_ppa, def_passing_plays_ppa, season.x, season.y,
    pass_atts, rush_yds, first_downs, penalty_yds, kick_returns,
    def_standard_downs_ppa, def_open_field_yds, off_line_yds_total,
    net_pass_yds, off_open_field_yds, off_passing_plays_ppa,
    def_standard_downs_success_rate, off_passing_downs_explosiveness,
    fumbles_lost,
    off_explosiveness, def_passing_downs_explosiveness,
    def_explosiveness, def_passing_downs_explosiveness, fourth_downs,
    def_havoc_front_seven, fourth_down_convs, off_passing_plays_explosiveness,
    off_rushing_plays_explosiveness, off_standard_downs_explosiveness,
    def_rushing_plays_explosiveness, kick_return_yds,
```

```
    def_standard_downs_explosiveness, pass_comps, third_downs, penalties,
    punt_return_TDs, def_havoc_db, off_havoc_front_seven
  ) |>
  step_smote(playoff)

# Prep only once on training data
smote_sub_prep <- prep(smote_subset_recipe, retain = TRUE)

# Apply to training data
smote_sub_train <- juice(smote_sub_prep)

# Apply the same transformations to test data
smote_sub_test <- bake(smote_sub_prep, new_data = test_data)
```

After SMOTE, I trained 6 different models. They are an XGBoost model, a random forest, a support vector machine, a K-Nearest Neighbors classifier, a logistic regression model regularized with lasso , and a decision tree. They followed a similar structure given the ease of use of the **caret** package in R. XGBoost, Random Forest, and support vector machines were chosen due to machine learning opporunities and usual performances as better predictors. K-Nearest Neighbors, logistic regression, and decision trees were also chosen due to interpretability and as a baseline for the predictions.

XGBoost:

```
# Detect number of cores and create cluster
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

xgb_grid <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(4, 6),
  eta = c(0.1, 1),
  gamma = c(0, 1),
  colsample_bytree = c(0.7, 1),
  min_child_weight = c(1, 2),
  subsample = c(0.7, 0.9)
)

set.seed(24)
repeated_oob <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final"
)


xgb_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "xgbTree",
  metric = "ROC",
  tuneGrid = xgb_grid
)
```

```
stopCluster(cl)
registerDoSEQ()
```

Support Vector Machine

```
# Detect number of cores and create cluster
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

svm_grid <- expand.grid(
  C = c(0.25, 0.5, 1),
  sigma = c(0.01, 0.015, 0.02)
)

set.seed(24)

repeated_oob <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final"
)

svm_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "svmRadial",
  metric = "ROC",
  tuneGrid = svm_grid
)

stopCluster(cl)
registerDoSEQ()
```

Random Forest:

```
# Detect number of cores and create cluster
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

repeated_oob <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final"
)

rf_grid <- expand.grid(
  mtry = c(2, 4, 6, 8),
  splitrule = "gini",
```

```
  min.node.size = 1
)

set.seed(24)
rf_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "ranger", # <-- important change here
  metric = "ROC",
  tuneGrid = rf_grid,
  importance = "impurity"
)


stopCluster(cl)
registerDoSEQ()
```

K-Nearest Neighbors:

```
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

repeated_oob <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final"
)

knn_grid <- expand.grid(
  k = c(3, 5, 7, 9, 11)
)

set.seed(24)
knn_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "knn",
  metric = "ROC",
  tuneGrid = knn_grid
)


stopCluster(cl)
registerDoSEQ()
```

Logistic Regression Regularized with LASSO:

```
# Detect number of cores and create cluster
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

repeated_oob <- trainControl(
```

```
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final"
)

lasso_grid <- expand.grid(
  alpha = 1, # 1 = LASSO penalty
  lambda = c(0.0001, 0.001, 0.01, 0.1, 1) # different levels of shrinkage
)

set.seed(24)
lasso_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "glmnet",
  metric = "ROC",
  tuneGrid = lasso_grid
)

stopCluster(cl)
registerDoSEQ()
```

Decision Tree:

```
cl <- makeCluster(parallel::detectCores() - 1) # Leave 1 core free
registerDoParallel(cl)

cart_grid <- expand.grid(
  cp = c(0.001, 0.01, 0.05, 0.1)
)

set.seed(24)
cart_sub_model <- train(playoff ~ .,
  data = smote_sub_train |> select(-team),
  trControl = repeated_oob,
  method = "rpart",
  metric = "ROC",
  tuneGrid = cart_grid
)


stopCluster(cl)
registerDoSEQ()
```

All of these models were trained using cross-validation and by doing a grid search over their individual hyperparameters. Performance of these models can be seen in the following section.

Interpretation The performance of the models is as follows:

```
#> # A tibble: 6 x 5
#>   Model         Accuracy Precision Recall   AUC
#>   <chr>            <dbl>     <dbl>  <dbl> <dbl>
```

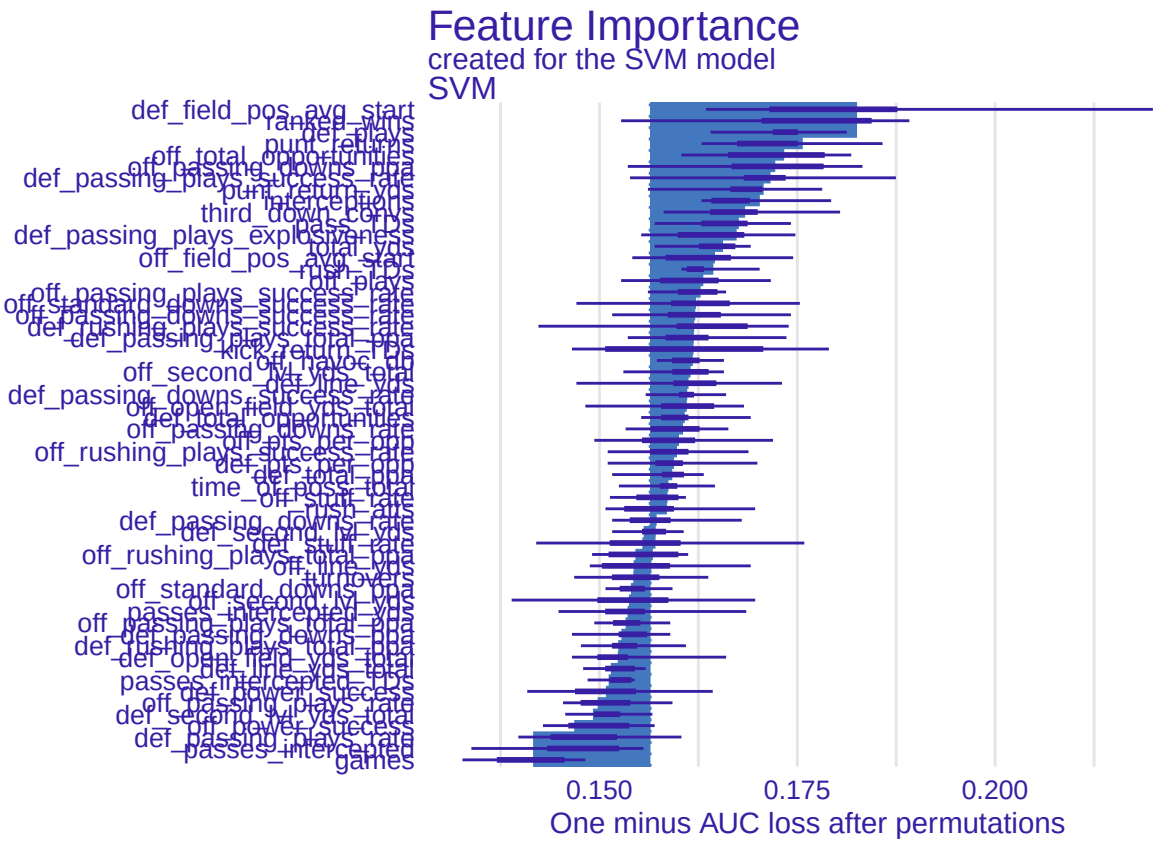
```
#> 1 XGboost          0.933     0.954  0.976 0.872
#> 2 SVM              0.948     0.954  0.992 0.844
#> 3 Random Forest    0.944     0.965  0.976 0.876
#> 4 KNN              0.697     0.973  0.700 0.736
#> 5 LASSO            0.869     0.982  0.877 0.857
#> 6 Decision Tree    0.910     0.956  0.949 0.623
```

The three best performing models are the XGBoost, Random Forest, and SVM model. Out of these three, I would say that the Forest performed the best. They have the best accuracy as well as the best area under the ROC curve.

The variable and permutation importance for those three models are as follows:

```
varImp(xgb_sub_model)
#> xgbTree variable importance
#>
#>   only 20 most important variables shown (out of 58)
#>
#>                                Overall
#> ranked_wins                    100.000
#> punt_returns                    27.164
#> punt_return_yds                 27.025
#> kick_return_TDs                 22.051
#> def_field_pos_avg_start         11.945
#> def_passing_plays_success_rate  11.018
#> passes_intercepted              10.570
#> passes_intercepted_yds           8.512
#> off_rushing_plays_total_ppa      8.049
#> def_line_yds                     7.316
#> passes_intercepted_TDs           6.539
#> interceptions                    6.363
#> off_field_pos_avg_start          6.130
#> off_open_field_yds_total         5.075
#> def_open_field_yds_total         4.909
#> off_plays                        4.445
#> def_second_lvl_yds               4.417
#> off_havoc_db                     4.297
#> def_passing_plays_rate           4.287
#> def_total_ppa                    4.246

varImp(rf_sub_model)
#> ranger variable importance
#>
#>   only 20 most important variables shown (out of 58)
#>
#>                                Overall
#> punt_return_yds                 100.00
#> ranked_wins                      97.41
#> punt_returns                     76.58
#> def_passing_plays_success_rate   59.58
#> def_total_ppa                    58.46
#> def_field_pos_avg_start          50.62
#> def_passing_plays_total_ppa      42.54
#> def_line_yds                     41.06
#> off_rushing_plays_total_ppa      38.19
```

```
#> rush_atts                          36.62
#> def_passing_downs_success_rate     34.87
#> def_passing_downs_ppa              32.59
#> def_pts_per_opp                    28.73
#> passes_intercepted_yds             28.18
#> off_field_pos_avg_start            27.94
#> interceptions                      27.06
#> off_plays                          26.04
#> off_total_opportunities            25.67
#> def_second_lvl_yds                 24.80
#> off_standard_downs_success_rate    24.63

explainer <- explain(
  model = svm_sub_model,
  data = smote_sub_test |> select(-playoff, -team),
  y = as.numeric(smote_sub_test$playoff == "True"),  # <- FIX here
  label = "SVM"
)
#> Preparation of a new explainer is initiated
#>   -> model label       :  SVM
#>   -> data              :  267  rows  58  cols
#>   -> data              :  tibble converted into a data.frame
#>   -> target variable   :  267  values
#>   -> predict function  :  yhat.train  will be used (  default  )
#>   -> predicted values  :  No value for predict function target column. (  default  )
#>   -> model_info        :  package caret , ver. 7.0.1 , task classification (  default  )
#>   -> predicted values  :  numerical, min =  3.251555e-11 , mean =  0.01567699 , max =  0.8662533
#>   -> residual function :  difference between y and yhat (  default  )
#>   -> residuals         :  numerical, min =  -0.8467605 , mean =  0.03675746 , max =  1
#>   A new explainer has been created!

# Now you can run model_parts
vi <- model_parts(explainer)
plot(vi)
```



Conclusion It seems as though the Random Forest was the best model at classifying which four teams made it to the playoffs. Through PCA and correlation analysis, seasonal data can be shrunk and give a good explanation to who would make it to the playoff. It does appear that strength of schedule is a powerful factor into the decision making of the playoff committee. However, this may be biased towards who the committee already thinks is good and the rankings may show that. Overall, this analysis shows that winning games against good teams will get a team into the playoffs.