

Single or Out?

James Musso

5/29/2020

Overview

Introduction

The game of baseball lends itself well to statistical analysis. Since its inception in the latter half of the 19th century, numerous observers have used statistics to better understand what contributes to a winning effort. Many but not all of the factors or features are apparent from mere observation. Statistical analysis, particularly within the last forty years, has unveiled some important contributors to winning that are not obvious to the naked eye, and are sometimes even counterintuitive.

At its core, baseball is about scoring more runs than the other team. A run is scored when a batter is able to reach first base, then second base, then third base, and finally home plate. Thus, the primary objective of a baseball team's offense is to first have a batter reach base and then eventually score a run by crossing home plate before his team makes three outs in the inning. Many events can cause a batter to reach home. One such event, the single, is the subject of this study.

A single occurs when: 1) a batter bats a ball into fair territory, 2) none of the nine players on the field (fielders) are able to catch the ball before it hits the ground, 3) none of the fielders are able to tag the batter with the ball before the batter contacts first base, and 4) none of the fielders are able to contact first base with the ball in their possession before the batter reaches first base.

A single is just one type of hit. Batted balls resulting in hits are the primary way that batters reach base. For players and fans alike, hits are among the most entertaining events in a baseball game. But singles tend to be less exciting than other types (doubles, triples, and home runs). A single only allows a batter to reach first base, so the batter, now a baserunner, is still a long way from home. Singles often aren't hit as hard and don't travel as far as other types of hits. But they are the most common way to reach base, and so they play an important role in scoring runs.

During the 2019 Major League Baseball (MLB) regular season, batters stepped to the plate with an opportunity to bat the ball about 186,000 times (<https://baseballsavant.mlb.com/league>). On about 126,000 of those occasions, they succeeded in hitting the ball somewhere onto the field. About 26,000 of those batted balls were singles, allowing the batter to reach first base. All the other types of hits combined amounted to only 16,000. That means about 84,000 batted balls, or two-thirds, resulted in outs. Baseball is a game of failure; most batters never make it back home.

What is it about the ubiquitous single that deserves the attention of a study? Perhaps more than any other batting event, a batted ball resulting in a single seems to be the event that's most difficult to distinguish from a field out (an out made by a fielder on a batter ball). Like field outs, singles come in a variety of shapes and sizes. Both a weakly hit ground ball tapped a few feet in front of home plate, and a screaming line drive hit off of the top of an outfield wall have been singles, and it's not that unusual to see either of these dramatically different examples.

So what is it that makes a batted ball a single rather than an out? Is it merely a matter of hitting the ball where there happens to be no fielder? Is it merely a matter of chance? Or are there identifiable features that distinguish a single from a field out, and if so, can we incorporate those features into a statistical model that reliably predicts when a batted ball will result in a single rather than an out? The goal of this effort is to answer these questions.¹

¹This paper was inspired by earlier research by Bill Pettit and Jim Albert that sought to distinguish outs from hits in general. Pettit, Bill, "Using Statcast Data to Predict Hits" (2016). <https://tht.fangraphs.com/using-statcast-data-to-predict-hits/> ; Albert, Jim, "Chance of Hit as Function of Launch Angle, Exit Velocity and Spray Angle" (2018). <https://baseballwithr.wordpress.com/2018/01/15/chance-of-hit-as-function-of-launch-angle-exit-velocity-and-spray-angle/>

One point of clarification. We are not determining the probability of a particular batter hitting a single in a particular game situation. If that was the case, we would be considering features such as the past performance of the batter, pitcher, and fielders, the altitude, dimensions and configuration of the park, the temperature and humidity, the wind direction and velocity, cloud cover and the position of the sun, the number of pitches the pitcher has already thrown in the game, the number of baserunners and outs, the inning, and so on and so on and so on. Instead, this study focuses on features the batter can conceivably control, namely, the direction he hits the ball, how hard he hits the ball, and how fast he runs to first base, as these are the most fundamental components of a single.

The Data

In 2015, MLB installed a state-of-the-art tracking technology, Statcast, in all 30 parks. It consists of a Trackman Doppler radar and high-definition Chyron Hego cameras. The radar, positioned above and behind home plate, captures just about everything that happens to the ball at 20,000 frames per second. The six stereoscopic cameras, installed down each foul line, track the movement of players on the field. In short, the technology generates data allowing the skills of players to be quantified in new ways.

Baseball Savant is MLB's public-facing repository for Statcast data. It includes, among other things, a powerful search tool that allows users to create their own custom queries. The datasets used in this study were created using Baseball Savant's various applications and search tools at BaseballSavant.MLB.com.

The primary dataset (*battedballs_2019_Raw*, or *battedballs* for short) obtained from Baseball Savant originally included 108,881 batted balls hit during the 2019 MLB season that resulted in either a single or an out (or an error, which, by definition, should have been an out). Excluded were batted balls that resulted in other types of hits (doubles, triples, and home runs), and batted balls officially ruled to be sacrifice bunts. Sacrifice bunts were excluded because, by definition, the batter is only attempting to advance a baserunner, not get a hit.

The key features or predictors in our *battedballs* dataset include Statcast measurements of how the ball comes off the bat in terms of direction and velocity. With regard to direction, a batted ball travels through three-dimensional space, but for our purposes, we're really only interested in two of those dimensions: vertical (up-down) and horizontal (left-right). The third dimension, depth, is called Hit Distance by Statcast, and it's really telling us where the ball ended up, which is where the ball landed or was touched by a fielder. And knowing where the ball landed or was touched by a fielder comes very close to directly telling us the answer to our question of whether the batted ball will be a single or an out. In other words, we're more interested in telling a batter how he has to hit the ball (the process) rather than where his batted ball needs to land on the field (the result), something that hopefully is already obvious to him.

Statcast's measure of a batted ball's vertical direction is called **Launch Angle** (*launch_angle* in the *battedballs* dataset), which represents the vertical angle at which the ball leaves the bat after being struck. A batted ball with a launch angle of 10 degrees or less is probably a ground ball, while a launch angle greater than 39 degrees is probably a pop up. But our primary purpose in using Launch Angle is not to deduce whether a batted ball is a ground ball, line drive, fly ball, or pop up. We're using it to help predict whether a batted ball will be a single or an out.

At present, Statcast surprisingly does not directly measure the horizontal direction of a batted ball (or at least doesn't publish such measurements through Baseball Savant). But it does provide a batted ball's final location coordinates (referred to as *hc_x* and *hc_y*), which reveal horizontal direction as well as distance. Like Hit Distance, *hc_x* and *hc_y* tell us where the ball ended up, so including them in our model would introduce bias. But we can use them to compute, with a little trigonometry, a very good estimate of horizontal direction (commonly referred to as **Spray Angle**). These calculations will accordingly appear in the *battedballs* dataset under the variable *spray_angle*.

Another feature, this one categorical, was added to our mix in the hope it would enhance the significance of *spray_angle*. Baseball Savant provides, for each batted ball, information regarding the locations of the fielders at the time the ball was batted into play. By itself, this information can tell us little if anything

about the likelihood of a batted ball resulting in a single or an out. Plus, it violates our guideline about using variables the batter can conceivably control. However, its interaction with *spray_angle* might improve a model's ability to make such single-out distinctions for predictive purposes. So we opted to include this information in our models for the sole purpose of making *spray_angle* more meaningful.

For infielders, the variable is named *if_fielding_alignment*, and its possible values represent three somewhat broad categories intended to describe where the infielders are located with respect to second base. According to MLB's Glossary (<http://m.mlb.com/glossary/statcast/shifts>), a value of "Standard" indicates all four infielders are standing in their traditional spots; that is, "where, under neutral conditions (first to eighth inning, no runners on), the league average fielder was positioned 70 percent to 90 percent of the time." In mathematical terms, MLB defines these "traditional spots" based on spray angle and distance from home plate.

A value of "Infield shift" indicates at least three of the four infielders are located on one side of second base (almost always the batter's pull side, when he contacts the ball early).

Lastly, a value of "Strategic" encompasses fielding alignments that are neither "Standard" nor "Infield shift". According to MLB's Glossary, an example would be a single infielder playing out of position, "like a second baseman being shifted to short right field... or a shortstop moving very close to...second base...but not quite moving to the other side of it."

Of course, knowing how the infielders are aligned can only be useful for batted balls with relatively low launch angles (Low Ground Balls and High Ground balls). Still, such batted balls comprise more than more than half of all the batted balls in our data, and for nearly one-third of those batted balls, infielders were in a non-standard alignment. So we thought it worthwhile to investigate whether the inclusion of *if_fielding_alignment* could improve our models.

Statcast's measurement of how hard the ball was hit is called **Exit Velocity** (*launch_speed* in the *battedballs* dataset). Specifically, it measures the speed of the ball (in miles per hour) immediately after the batter makes contact. Its potential relevance to our "single or out" question is apparent: a faster moving ball gives the fielders less time to react and catch the ball or stop the ball, making it less likely they will get the batter out.

A second dataset (*hometofirst_2019_Raw*, or *hometofirst* for short) from Baseball Savant includes Statcast's **Home to First** measurement. It's the length of time, in seconds, from the moment the bat hits the ball to the moment the batter touches first base. Basically, it's telling us how fast the batter can run to first base, and it's another one of our fundamental components of a single. However, it's only relevant for our study if the batted ball is a low ground ball (launch angle < 0 degrees). The batter's ability to reach first base quickly is far less likely to matter if the batted ball is a line drive, fly ball, or pop up. Still, low ground balls comprise about 37% of all the batted balls in our dataset, so the batter's ability to reach first base quickly is potentially a helpful predictor in our model.

Baseball Savant makes Home to First data available to us in the form of a season average rather than for each individual batted ball. And the season average wisely considers only batted balls that it characterizes as Topped or Weakly Hit (based on Launch Angle and Exit Velocity) because these plays are likely to require the batter to make a serious effort to run as fast as possible. Moreover, the average considers only the fastest 70 percent of these serious effort plays.

The Process

We followed a typical path toward the development of predictive models. The initial task, referred to as data wrangling, is to get the data into a form that can be used to develop algorithms. This involved importing the data files into R data frames, removing unneeded observations with missing data, removing irrelevant observations, combining data frames, adding new variables and computing their values from existing variables, converting categorical variables into numeric form, converting continuous variables into new categorical variables, and reordering the columns in the final data frame. All of these steps would facilitate our later analysis.

The next key task was to explore the data to deepen our understanding of which variables might be most helpful in a predictive model, and how they related to each other and to the outcome we were trying to predict. Both quantitative and visual tools were used for this purpose. In the context of this project, plots of batted balls would help reveal insights, as would various types of boxplots and density plots.

Density plots proved particularly helpful in clarifying relationships between variables, showing the distribution of data values as a continuous line while the Y axis shows proportions or frequency. The message they convey is not muddled by the disparate number of observations in the groups portrayed because it shows proportions rather than counts. This was important in our case because the classes (singles and outs) of our outcome variable had significantly different numbers of observations (outs occur much more frequently than singles).

At this point, we were almost ready to begin building models, but a couple of important data preparation tasks remained to be completed. The first involved rescaling the data to meet the particular needs of each type of algorithm we anticipated testing. Because we expected to be working with a variety of algorithms, we opted to create transformed versions of our key predictor variables using three methods: centering, standardization, and normalization. These methods are discussed in some detail later in this report.

The last task in this step involved randomly partitioning our final data frame into a train set and a test set. This allowed us to use one portion of our data solely to fit an algorithm, and a completely distinct and independent portion of our data to test the predictive ability of the newly fit algorithm. This, in turn, diminishes the problem of overfitting or overtraining the model to perform very well with our data, and not nearly as well once it's used in the real world with different data.

Finally, we could begin constructing models and testing them. We initially built a simple model based on a guessing algorithm. This would serve as a baseline for comparison once we began building other models. If our later models couldn't outperform the baseline model, we would have a serious problem. We next constructed a logistic regression model. Logistic regression is essentially a form of linear regression that has been adapted for predicting a binary outcome from a set of continuous and/or categorical predictor variables, which was exactly what we were trying to do.

A K-nearest-neighbors model was built next, followed by a classification tree model and finally, a random forest model. Building each of these models involved training or fitting the model to our train set, and then testing the model by allowing it to estimate the probabilities of each outcome based on the data in the test set. The model would then make its final prediction based on the probabilities. At that point we became like a teacher grading an exam: how many predictions did the model get right?

During this model-building stage, an important decision had to be made about the criteria we would apply when assessing each model's predictions. We eventually decided to use two somewhat different methods, neither of which appears to be in mainstream use at this time, but which appeared to be well-suited for our particular challenge. The first one we labeled **Maximum Truescore**, which is based on maximizing the harmonic mean of sensitivity and specificity. The second method, **Minimum Distance**, seeks to minimize the distance between the ROC curve and the coordinate (0, 1), which is where both sensitivity and specificity will be very high.

The results of both methods can be computed using only sensitivity and specificity. The two methods were published in the same paper, the specific purpose of which was to develop better methods to address the "decision threshold shifting issue" that occurs when attempting to classify imbalanced data.² That was our challenge: too many outs and not enough singles would make our classification models perform less well because they cause the threshold that best differentiates the outcomes (singles and outs) to shift in favor of the majority class (outs). This is discussed in some detail later in this paper.

We ended our study by examining the prediction mistakes made by our best predictive model. It was hoped such a review might reveal ways of improving our model's performance and suggest further areas for research.

²Song, B., Zhang, G., Zhu, W. et al. "ROC operating point selection for classification of imbalanced data with application to computer-aided polyp detection in CT colonography." Int J CARS 9, 79–89 (2014) doi:10.1007/s11548-013-0913-8 The authors' manuscript may be viewed at no cost at <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3835757/#idm140500764776720title>

Methodology and Analysis

Wrangling the Data

At the outset, the required R packages were installed and loaded.

```
# Install and Load Required Packages

if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                         repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",
                                         repos = "http://cran.us.r-project.org")
if(!require(ROCR)) install.packages("ROCR",
                                         repos = "http://cran.us.r-project.org")
if(!require(interplot)) install.packages("interplot",
                                         repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart",
                                         repos = "http://cran.us.r-project.org")
if(!require(rpart.plot)) install.packages("rpart.plot",
                                         repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
                                         repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos = "http://cran.us.r-project.org")
if(!require(gtable)) install.packages("gtable",
                                         repos = "http://cran.us.r-project.org")
if(!require(lemon)) install.packages("lemon",
                                         repos = "http://cran.us.r-project.org")
if(!require(kknn)) install.packages("kknn",
                                         repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr",
                                         repos = "http://cran.us.r-project.org")
if(!require(markdown)) install.packages("markdown",
                                         repos = "http://cran.us.r-project.org")
if(!require(rmarkdown)) install.packages("rmarkdown",
                                         repos = "http://cran.us.r-project.org")
if(!require(formatR)) install.packages("formatR",
                                         repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex",
                                         repos = "http://cran.us.r-project.org")
```

To reshape our data into a useful form for modeling, we first imported our two data files from the internet (<https://github.com/jfmusso/HarvardX/>) directly into R data frames known as tibbles.

```
# Data is contained in two files: battedballs_2019_Raw.csv and hometofirst_2019_Raw.csv
# located at https://github.com/jfmusso/HarvardX/

# Import the data files into R data frames.

battedballs <-
  read_csv(
    "https://raw.githubusercontent.com/jfmusso/HarvardX/master/battedballs_2019_Raw.csv",
```

```

col_types = cols(
  game_date = col_date(format = "%m/%d/%Y"),
  batter = col_character(),
  player_name = col_character(),
  stand = col_character(),
  bb_type = col_factor(levels = c("ground_ball", "line_drive", "fly_ball",
                                    "popup")),
  events = col_factor(),
  des = col_character(),
  launch_speed = col_double(),
  launch_angle = col_double(),
  hc_x = col_double(),
  hc_y = col_double(),
  hit_location = col_factor(),
  if_fielding_alignment = col_factor(levels = c("Standard", "Strategic",
                                                 "Infield shift")),
  of_fielding_alignment = col_factor(),
  hit_distance_sc = col_integer(),
  launch_speed_angle = col_factor(),
  game_pk = col_character(),
  game_year = col_character(),
  game_type = col_character(),
  home_team = col_character(),
  away_team = col_character(),
  pitcher = col_factor(),
  p_throws = col_character(),
  description = col_character(),
  type = col_character()
)
)

class(battedballs[])

hometofirst <-
  read_csv(
    "https://raw.githubusercontent.com/jfmusso/HarvardX/master/hometofirst_2019_Raw.csv",
    col_types = cols(
      last_name = col_character(),
      first_name = col_character(),
      player_id = col_character(),
      team_id = col_character(),
      team = col_character(),
      position = col_character(),
      age = col_number(),
      competitive_runs = col_integer(),
      hp_to_1b = col_double(),
      sprint_speed = col_double()
    )
  )

class(hometofirst[])

```

We then filtered the *firsttohome* data frame to remove 58 players whose First-to-Home readings were missing (presumably because they didn't have any "serious effort" runs to first base). The *battedballs* data frame was then filtered to remove batted balls events with missing values for any of our important predictors. This resulted in removing 889 batted balls due to missing coordinate and fielder alignment data. The *battedballs* data frame now stood at 107,992 observations.

```
# Remove players from *hometofirst* data frame if they have missing values for hp_to_1b.

hometofirst <- filter(hometofirst, hp_to_1b != "NA")

# Remove batted balls from battedballs data frame if they have missing values
# for any of the following variables: hc_x, hc_y, if_fielding_alignment, or
# of_fielding_alignment.

battedballs <- filter(battedballs, hc_x != "NA", hc_y != "NA",
                      if_fielding_alignment != "NA",
                      of_fielding_alignment != "NA")
```

We then combined our *battedballs* and *hometofirst* data frames, adding each batter's Home to First average to each of his batted ball events (rows) in the *battedballs* data frame. Now, we had one data frame, *battedballs*, containing all of our potentially relevant data in the form of 103,810 batted ball events (observations). So in the process of combining our two data frames, we lost 4,182 batted balls hit primarily by pitchers for whom there was insufficient Home to First data (pitchers don't bat nearly as often as position players).

```
# Combine battedballs and hometofirst data frames.

hometofirst <- rename(hometofirst, batter = player_id)
hometofirst <- hometofirst %>% dplyr::select(batter, team, position, age, hp_to_1b)
battedballs <- left_join(battedballs, hometofirst, by = "batter")

# Remove players from battedballs for whom there is no hp-to-1b data.

battedballs <- filter(battedballs, hp_to_1b != "NA")
```

The next step was to remove the batted balls that were hit into foul territory based on the description of the play. We were interested in distinguishing singles from outs on balls hit into fair territory. The foul balls in our data frame resulted only in outs because, by the rules of the game, fouls can't result in a hit of any kind, including singles. Excluding fouls reduced our data frame by 3226 to 100,584 batted balls.

```
# Remove batted balls hit into foul territory.

battedballs <- battedballs %>% mutate(foul = str_detect(des, "foul")) %>%
  filter(foul == FALSE)
```

It was now time to compute one of our key predictors, *spray_angle*, and add it to the *battedballs* data frame. This required using a formula to first transform Baseball Savant's coordinate data so that home plate would be located at coordinates (0, 0).³

³The formula used in this study was published in a Western Michigan University honors thesis by Tess Kolp (Kolp, Tess, "Home Run Probability Based on Hit Distance and Direction" (2018). Honors Theses. 2934. https://scholarworks.wmich.edu/honors_theses/2934). There are a number of formulas published on the internet that attempt to perform the same conversion, but after some testing, the formula published by Kolp was determined to be the most accurate. Kolp credits Tom Tango as the source for the formula. Tango works for MLB Advanced Media, helping to facilitate the development and deployment of Statcast data.

```

# Compute Spray Angle of each batted ball and add a new column containing results.

# Transform x coordinate so that home plate is at 0.
battedballs <- battedballs %>% mutate(hc_x_Kolp = round(2.33 * (hc_x - 126), 2))

# Transform y coordinate so that home plate is at 0.
battedballs <- battedballs %>% mutate(hc_y_Kolp = round(2.33 * (204.5 - hc_y), 2))

```

A trigonometric function was then applied to the transformed coordinates to derive a spray angle for each batted ball. Thus, the left field foul line was located at -45 degrees, and the right field foul line was located at 45 degrees. A batted ball hit directly up the middle of the field (toward the pitcher's mound) would have a spray angle of 0. This version of Spray Angle was named *spray_angle_Kolp*.

```

# Compute spray_angle
battedballs <- battedballs %>%
  mutate(spray_angle_Kolp = round((180 / pi) * atan(hc_x_Kolp / hc_y_Kolp), 1))

```

Lastly, we adjusted the spray angle measurements for the side of the plate the batter stands on, so that batted balls hit to the pull side (the side of the field the batter stands on, with respect to home plate) will always have a negative spray angle. The final form of our Spray Angle feature was named *spray_angle_adj*.

```

# Adjust for side of plate batter stands on, so that batted balls hit to the side of
# the field the batter stands on (with respect to home plate) will always have a
# negative spray angle.

battedballs <- battedballs %>%
  mutate(spray_angle_adj = ifelse(stand == "L", -spray_angle_Kolp, spray_angle_Kolp))

```

Having finalized our Spray Angle data, we turned our attention to its interactive partner, *if_fielding_alignment*. To facilitate the development of models, we needed to convert it from a categorical variable into a numeric variable. Numeric values of 1, 2, and 3 corresponded to the categorical values “Standard”, “Strategic”, and “Infield shift”, respectively.

```

# Convert our categorical predictor variable, if_fielding_alignment, into a numeric
# variable.

battedballs <- battedballs %>% mutate(num_if_alignment =
                                         as.numeric(if_fielding_alignment))

```

We also added a more granular classification scheme for batted ball types.⁴ This approach uses six categories, each with a unique set of features and good year-to-year correlations. The new batted ball types were added to *battedballs* under the variable name *adv_bb_type*.

```

# Add a more precise division of batted ball types (adv_bb_type) to more accurately
# categorize the values of our launch_angle predictor.

battedballs <- battedballs %>%
  mutate(adv_bb_type = cut(battedballs$launch_angle,
                           breaks = c(-90, -0.1, 10, 19, 26, 39, 90),
                           labels = c("low_ground_ball", "high_ground_ball",
                                     "low_line_drive", "high_line_drive",
                                     "fly_ball", "popup")))

```

⁴Perpetua, Andrew, “Launch Angle Derived Batted Ball Types” (2017). <https://fantasy.fangraphs.com/anglebbtypes/>.

To facilitate plotting, we added a categorical variable based on *launch_speed*.

```
# Add a categorical variable based on *launch_speed*.

battedballs <- battedballs %>%
  mutate(launch_speed_cat = cut(battedballs$launch_speed,
                                breaks = c(10, 69.9, 79.9, 89.9, 100, 125),
                                labels =
                                  c("<70", "70-79.9", "80-89.9", "90-100", ">100")))
```

We also added a categorical variable based on *spray_angle* to facilitate plotting.

```
# Add categorical variables based on *spray_angle_Kolp* and *spray_angle_adj*.

battedballs <- battedballs %>%
  mutate(spray_angle_Kolp_cat = cut(battedballs$spray_angle_Kolp,
                                    breaks = c(-90, -45.1, -40, -31, -22, -9, 9, 25, 35,
                                               43, 45, 90),
                                    labels = c("-90:-45.1", "LF Line (-45:-40)",
                                              "3rd Baseman (-39.9:-31)",
                                              "Hole 5-6 (-30.9:-22)",
                                              "Shortstop (-21.9:-9)",
                                              "Hole Middle (-8.9:9)",
                                              "2nd Baseman (9.1:25)",
                                              "Hole 3-4 (25.1:35)",
                                              "1st Baseman (35.1:43)",
                                              "RF Line (43.1:45)", "45.1:90")))

battedballs <- battedballs %>%
  mutate(spray_angle_adj_cat = cut(battedballs$spray_angle_adj,
                                   breaks = c(-90, -45.1, -40, -31, -22, -9, 9, 25, 35,
                                              43, 45, 90),
                                   labels = c("-90:-45.1", "Pulled Down Line (-45:-40)",
                                             "Pulled Corner Inf (-39.9:-31)",
                                             "Pulled Side Hole (-30.9:-22)",
                                             "Pulled Mid Inf (-21.9:-9)",
                                             "Middle Hole (-8.9:9)",
                                             "Oppo Mid Inf (9.1:25)",
                                             "Oppo Side Hole (25.1:35)",
                                             "Oppo Corner Inf (35.1:43)",
                                             "Oppo Down Line (43.1:45)", "45.1:90"))))
```

Finally, we added a categorical variable for *hp_to_1b*.

```
# Add a categorical variable based on *hp_to_1b*.

battedballs <- battedballs %>%
  mutate(hp_to_1b_cat = cut(battedballs$hp_to_1b,
                            breaks = c(3.9, 4.2, 4.5, 4.8, 5.1),
                            labels = c("3.9-4.2", "4.21-4.5", "4.51-4.8",
                                      "4.81-5.1")))
```

We then reordered the columns in *battedballs* into a more logical sequence.

```
# Reorder the columns in battedballs.

battedballs <- battedballs %>% dplyr::select(game_date, batter, player_name, age, stand,
  position, team, bb_type, adv_bb_type, events, des, hp_to_1b, hp_to_1b_cat,
  launch_speed, launch_speed_cat, launch_angle, hc_x, hc_y, hc_x_Kolp, hc_y_Kolp,
  spray_angle_Kolp, spray_angle_Kolp_cat, spray_angle_adj, spray_angle_adj_cat,
  hit_location, hit_distance_sc, if_fielding_alignment, num_if_alignment,
  of_fielding_alignment, launch_speed_angle, game_pk, game_year, game_type,
  home_team, away_team, pitcher, p_throws, description, type, foul)
```

We were ready to take a closer look at our data.

Exploring the Data

First, we wanted to get a general sense of the distribution of the predictors and outcomes.

```
# Create a tibble containing just the predictors and outcome we will use.

battedballs_var <- dplyr::select(battedballs, events, launch_angle, launch_speed,
  spray_angle_Kolp, spray_angle_adj, hp_to_1b,
  if_fielding_alignment)

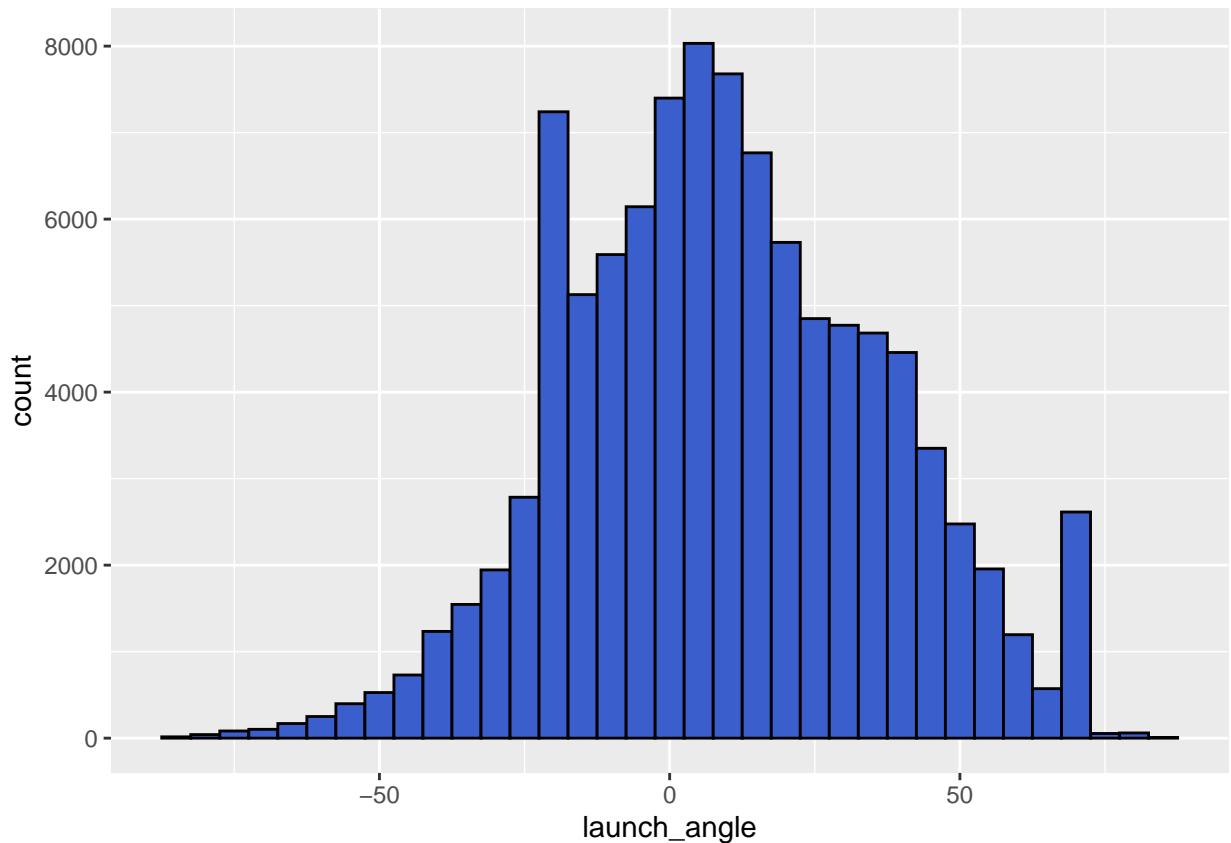
# Generate a statistical distribution of battedballs_var.

summary(battedballs_var)
```

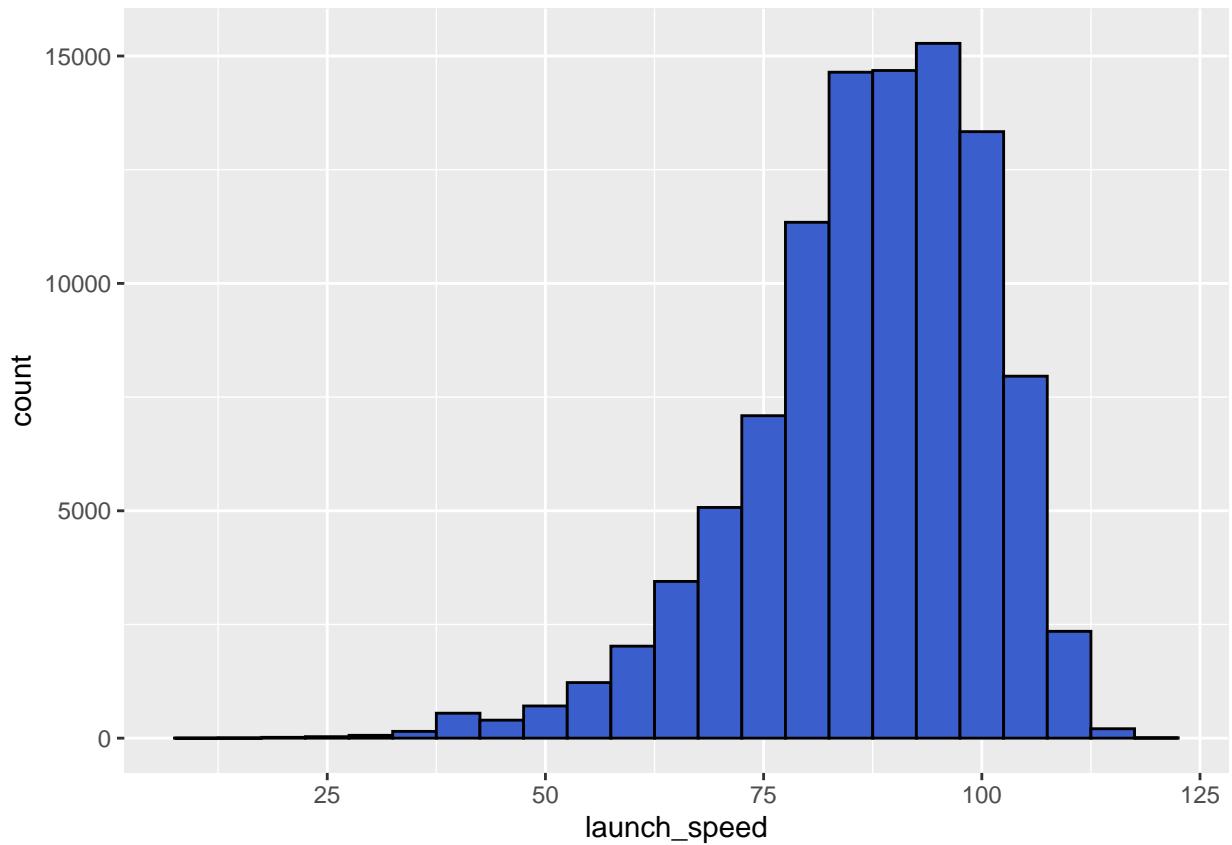
```
##      events    launch_angle    launch_speed    spray_angle_Kolp
##  single   :24901   Min.   :-87.200   Min.   : 11.50   Min.   :-83.1000
##  field_out:75683   1st Qu.: -9.700   1st Qu.: 80.00   1st Qu.:-20.5000
##                               Median :  8.100   Median : 89.00   Median : -0.3000
##                               Mean   :  9.505   Mean   : 87.07   Mean   :  0.4025
##                               3rd Qu.: 28.700   3rd Qu.: 97.10   3rd Qu.: 21.9000
##                               Max.   : 83.700   Max.   :120.60   Max.   : 88.2000
##      spray_angle_adj    hp_to_1b    if_fielding_alignment
##  Min.   :-77.300   Min.   :3.930   Standard   :66886
##  1st Qu.:-24.600   1st Qu.:4.250   Strategic   : 8474
##  Median : -7.400   Median :4.380   Infield shift:25224
##  Mean   : -4.398   Mean   :4.404
##  3rd Qu.: 15.500   3rd Qu.:4.530
##  Max.   : 88.200   Max.   :5.030
```

```
# Create a histogram showing the distribution of values for each
# of the predictors.
```

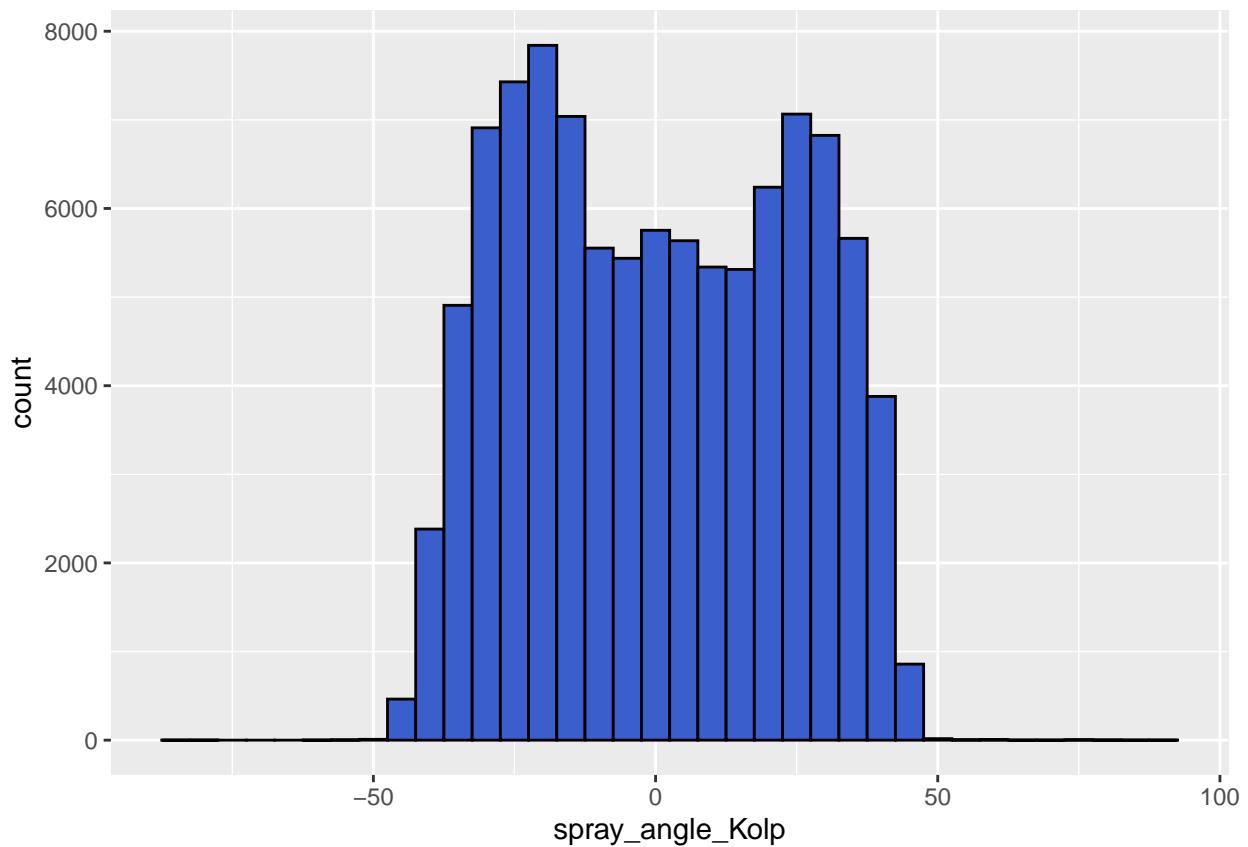
```
battedballs_var %>% ggplot(aes(x = launch_angle)) +
  geom_histogram(fill = "royalblue3", color = "black",
                 binwidth = 5)
```



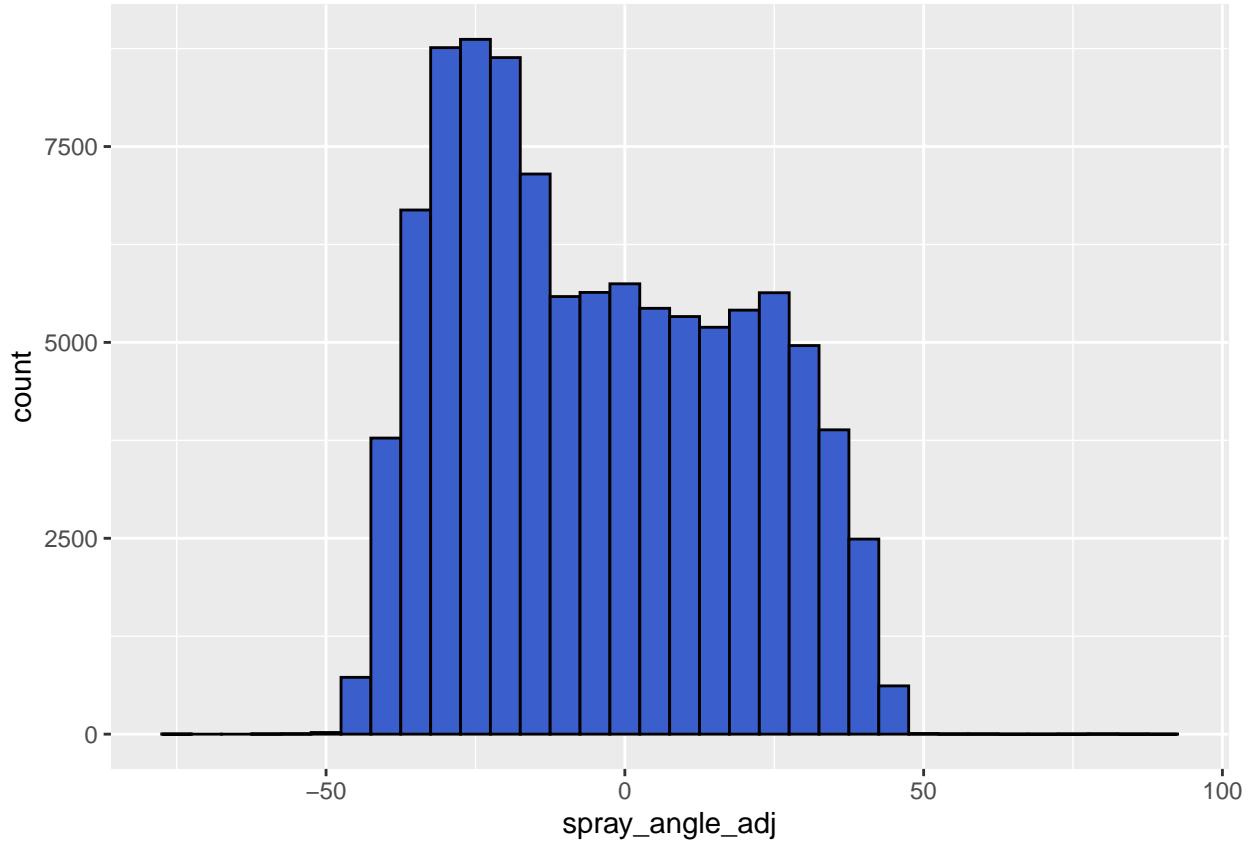
```
battedballs_var %>% ggplot(aes(x = launch_speed)) +  
  geom_histogram(fill = "royalblue3", color = "black",  
    binwidth = 5)
```



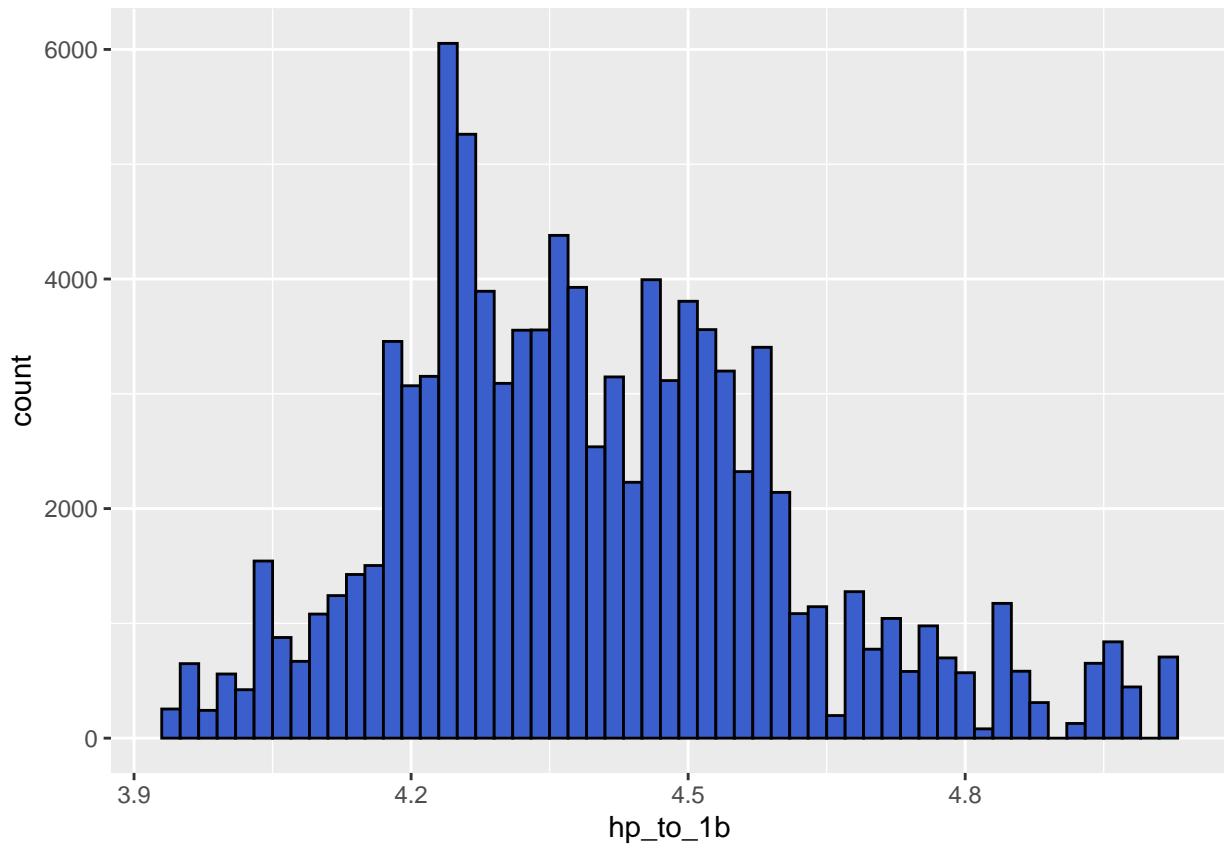
```
battedballs_var %>% ggplot(aes(x = spray_angle_Kolp)) +  
  geom_histogram(fill = "royalblue3", color = "black",  
    binwidth = 5)
```



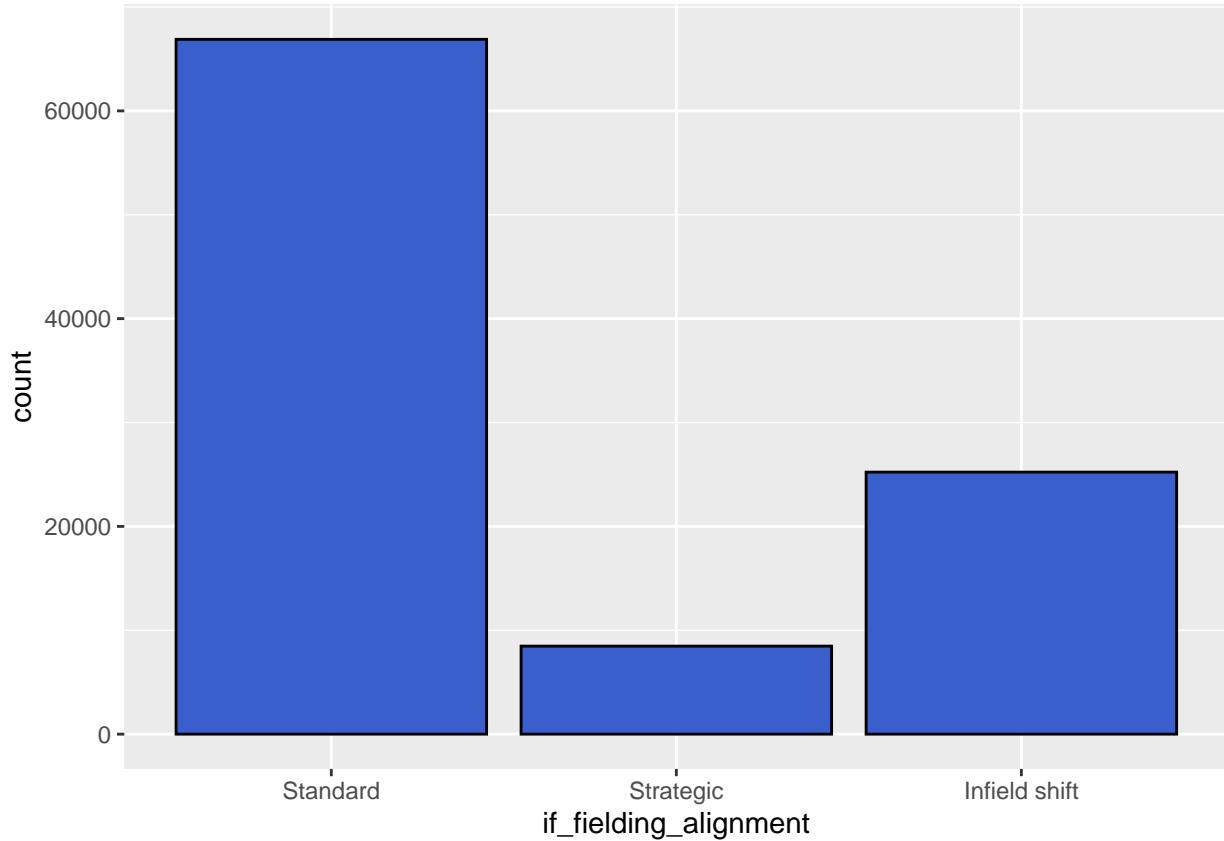
```
battedballs_var %>% ggplot(aes(x = spray_angle_adj)) +  
  geom_histogram(fill = "royalblue3", color = "black",  
    binwidth = 5)
```



```
battedballs_var %>% ggplot(aes(x = hp_to_1b)) +  
  geom_histogram(fill = "royalblue3", color = "black",  
    binwidth = .02)
```



```
battedballs_var %>% ggplot(aes(x = if_fielding_alignment)) +  
  geom_bar(fill = "royalblue3", color = "black")
```



To understand the relationship between the predictors, we created a correlation matrix. As expected, the predictors were largely independent of one another, with the exception of *launch_angle* and *spray_angle_adj*. Their modest correlation indicated that higher launch angles were associated with hitting the ball to the opposite field.

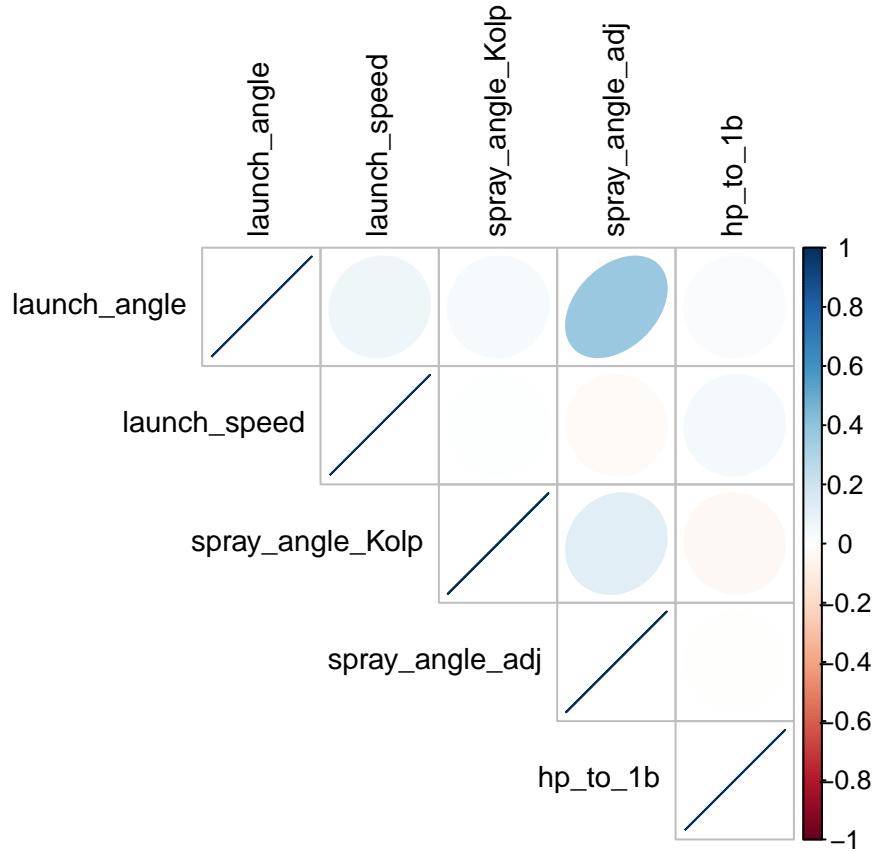
```
# Create a correlation matrix of the predictors to understand the
# relationship between them.

library(corrplot)

(cor_battedballs <- round(cor(battedballs_var[, 2:6], method = "spearman"), 2))

##          launch_angle launch_speed spray_angle_Kolp
## launch_angle      1.00       0.06      0.03
## launch_speed      0.06       1.00      0.00
## spray_angle_Kolp   0.03       0.00      1.00
## spray_angle_adj     0.37      -0.03      0.12
## hp_to_1b           0.02       0.04     -0.04
##          spray_angle_adj hp_to_1b
## launch_angle        0.37       0.02
## launch_speed        -0.03      0.04
## spray_angle_Kolp    0.12      -0.04
## spray_angle_adj      1.00     -0.01
## hp_to_1b            -0.01      1.00
```

```
corrplot(cor_battedballs, type = "upper", method = "ellipse", tl.cex = 0.9,
        tl.col = "black")
```



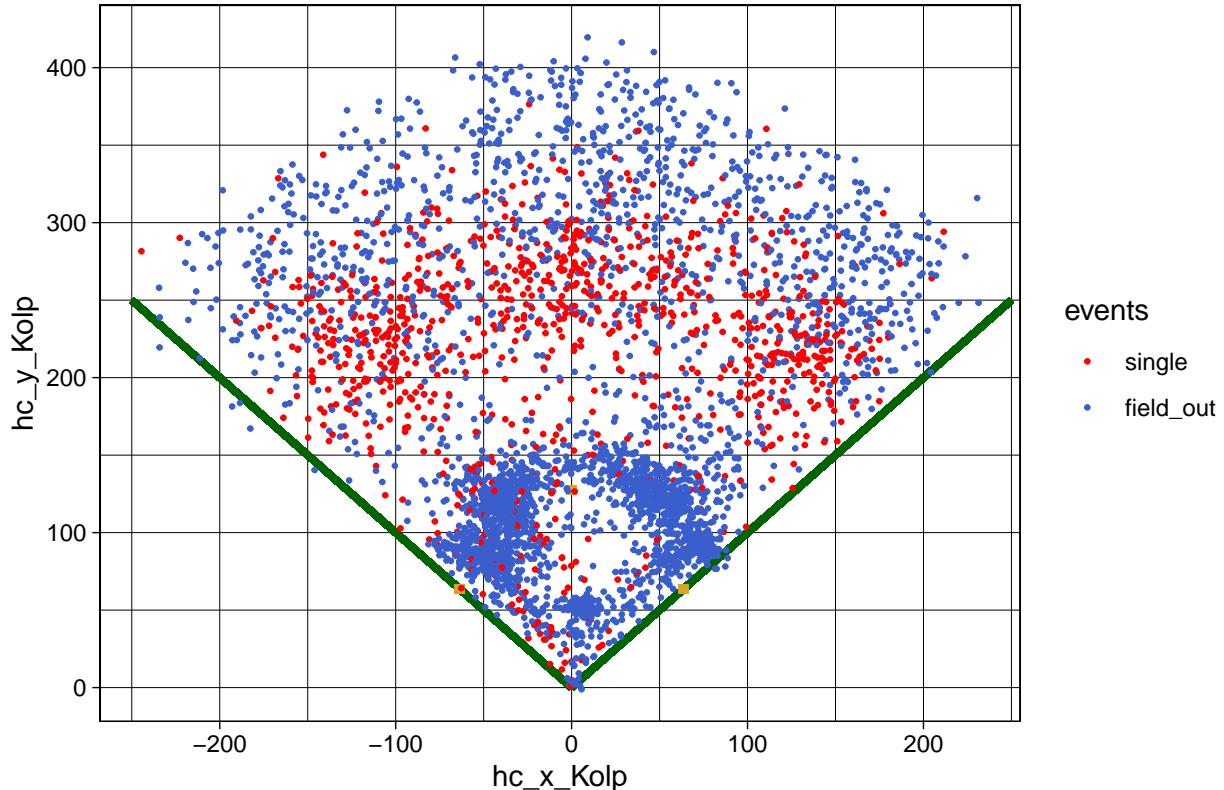
Visualizing a random sample of 4,000 of our 100,584 batted balls confirmed the reasonableness of our transformed coordinates. Only a handful of data points fall outside of our superimposed third base and first base foul lines. In addition, batted balls resulting in outs are clustered where fielders are normally positioned, while those resulting in singles tend to end up beyond the infielders and in front of where the outfielders are normally positioned.

```
# Visualize batted balls resulting in singles or outs.

set.seed(1)
battedballs_sample <- sample_n(battedballs, 4000, replace = FALSE)

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Fair Batted Balls Using Transformed Coordinates")
```

Fair Batted Balls Using Transformed Coordinates



The plots of batted ball coordinates revealed distinctive patterns distinguishing singles from outs. We could see that many of the singles ended up in shallow left field, center field, and right field, where they were first touched by an outfielder. But what were the characteristics of those batted balls? More precisely, what, if anything, did the batters do differently to get those singles? Would any, some, or all of our batter-centric features (Exit Velocity, Launch Angle, Spray Angle, and Home to First) be able to distinguish singles from outs well enough to be included in a model that could reliably predict whether a batted ball would result in a single or an out? We had to begin to understand the relationships between our features and our outcomes.

Launch Angle

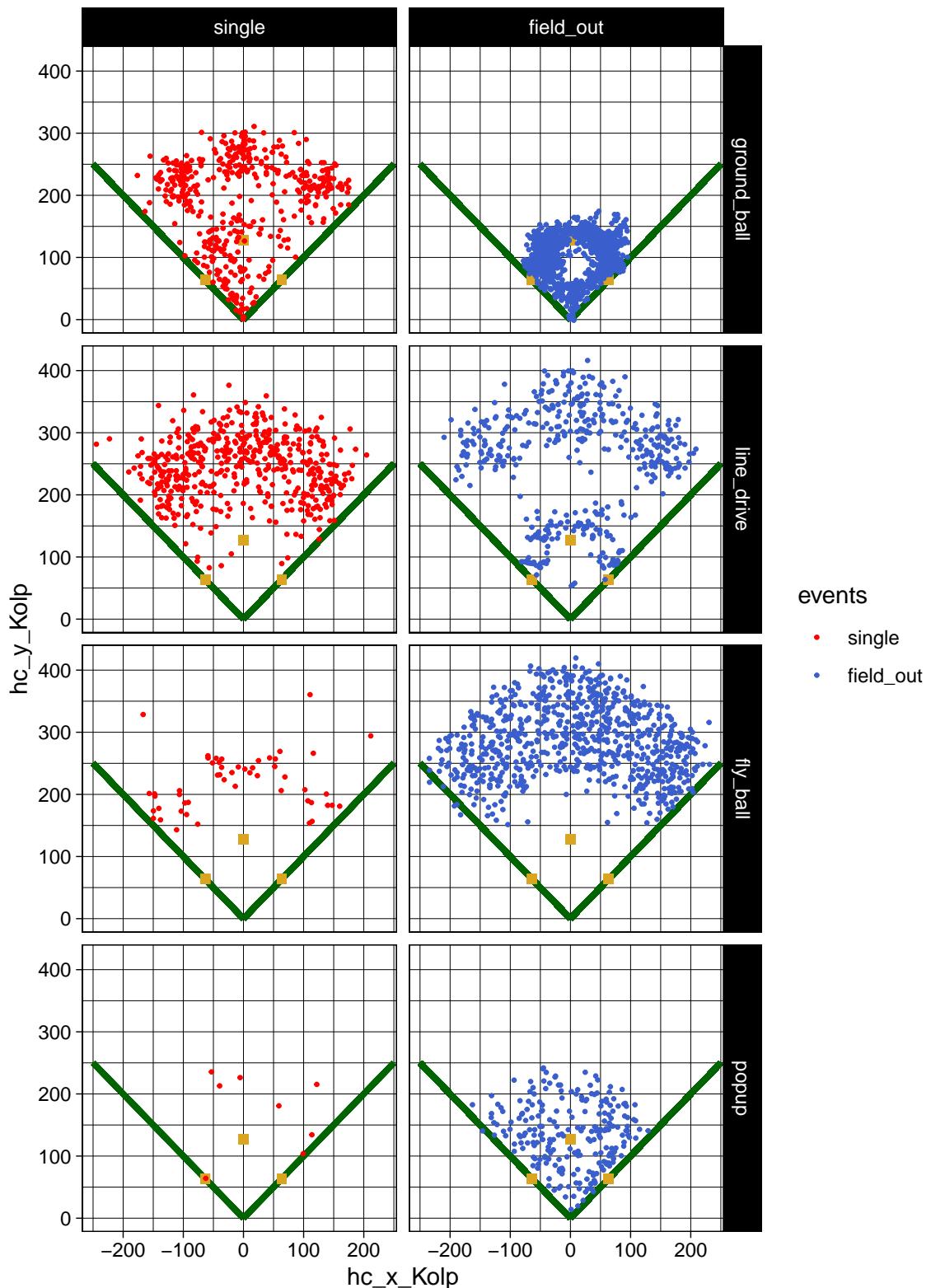
We first looked at singles and outs by *launch_angle*. You can think of batted ball type (*bb_type* in *battedballs*) as a categorical version of our *launch_angle* feature. The traditional batted ball types are “ground_ball” (launch angle less than 10 degrees), “line_drive” (10-25 degrees), “fly ball” (25-50 degrees), and “popup” (greater than 50 degrees). Ground balls comprise 50% of *battedballs*, followed by line drives (23%), fly balls (21%), and pop ups (6%).

```
# Visualize singles and outs, by type of batted ball.
```

```
ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
```

```
geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by Type of Batted Ball") +
  facet_grid(bb_type ~ events)
```

Singles vs Outs, by Type of Batted Ball



A close look at the plots reveals something that perhaps the casual baseball fan doesn't consciously notice. As expected, the ground ball plots confirm that ground ball outs are made all around the infield, though

fewer are made on balls hit up the middle past the pitcher. But the real revelation appears in the plot of ground ball singles, where we see that significantly more singles are hit to the left side than the right side of the infield. This makes sense because a fielder picking up a ground ball on the left side of the infield has a much longer throw to first base to get the batter out. Batters are able to beat the throws to first base on some of the dribblers and choppers hit down the third base line, and on grounders hit into the hole between the shortstop and third baseman. These are “infield hits”; they never make it to the outfield, yet the batter reaches first base safely. This suggest we should consider including *spray_angle_Kolp* as a predictor in our models, in addition to *spray_angle_adj*. *spray_angle_Kolp* is not adjusted for the side of the plate the batter bats on, so any ground ball hit to the left side of the infield ought to have a higher probability of resulting in a single.

As for the ground ball singles that reach the outfield, here we would expect *launch_speed* to make a difference in our predictive models. To get to the outfield, groundballs have to be hit between the infielders. Ground balls with higher exit velocities would give infielders less time to move toward the expected path of the ball before the ball rolls past them.

With regard to line drives, the singles are hit between or above the infielders before being stopped by outfielders in shallow left field, center field, and right field. The outs tend to be hit directly where fielders are traditionally positioned. Line drives are generally hit harder than other types of batted balls, giving fielders less time to move toward the ball. So whether a line drive results in a single would appear to depend more on the initial alignment of the fielders when the ball is hit. Popups and fly balls result in relatively few singles because fielders generally have enough time to move under the ball and catch it, regardless of where they are originally positioned when the ball is hit.

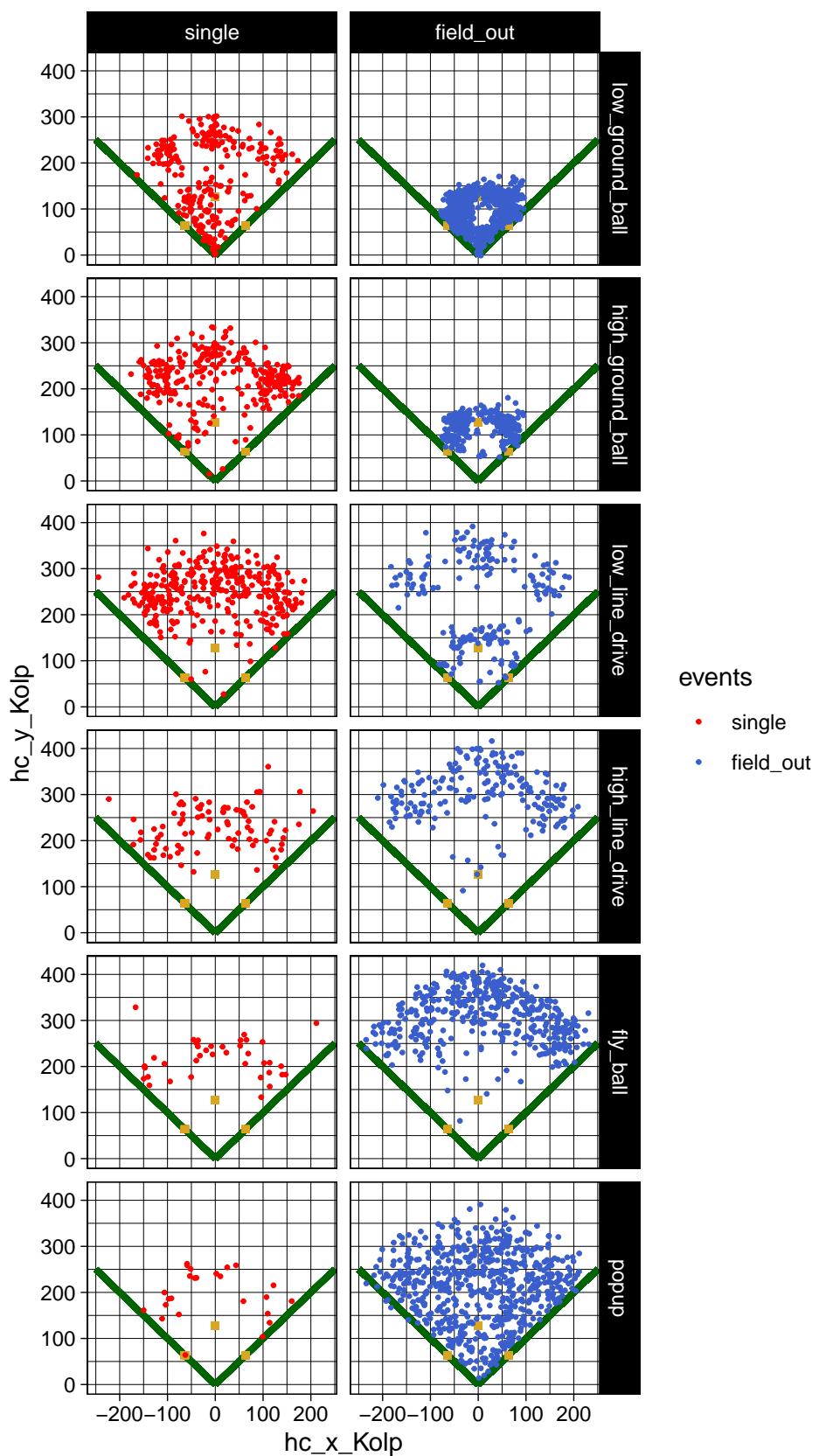
We also looked at *adv_bb_type*, the more granular classification scheme for batted ball types. The categories, again defined by Launch Angle, include: Low Ground Ball (less than 0 degrees), High Ground Ball (0-10 degrees), Low Line Drive (10.1-19 degrees), High Line Drive (19.1-26 degrees), Fly Ball (26.1-39 degrees) and Popup (greater than 39 degrees).⁵ Low Ground Balls comprise 37% of *battedballs*, followed by High Ground Balls (16%), Popups (15%), Low Line Drives (12%), Fly Balls (12%), and High Line Drives (7%).

```
# Visualize singles and outs, by adv_bb_type.

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by adv_bb_type") +
  facet_grid(adv_bb_type ~ events)
```

⁵Perpetua, Andrew, “Launch Angle Derived Batted Ball Types” (2017). <https://fantasy.fangraphs.com/anglebbtypes/>.

Singles vs Outs, by adv_bb_type



Visually, we can now see that it's Low Ground Balls with negative launch angles that can result in singles when hit to the left side of the infield. This makes sense, since balls chopped into the ground give the batter more time to reach first base safely. We can also see that the high versions of ground balls and line drives end up somewhat further from home plate than their low counterparts. This was expected.

To better quantify what was depicted in our plots of batted ball coordinates, we created box plots showing the distribution of *launch_angle* by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

```
# Create boxplots to visualize the distribution of launch_angle by singles and outs.

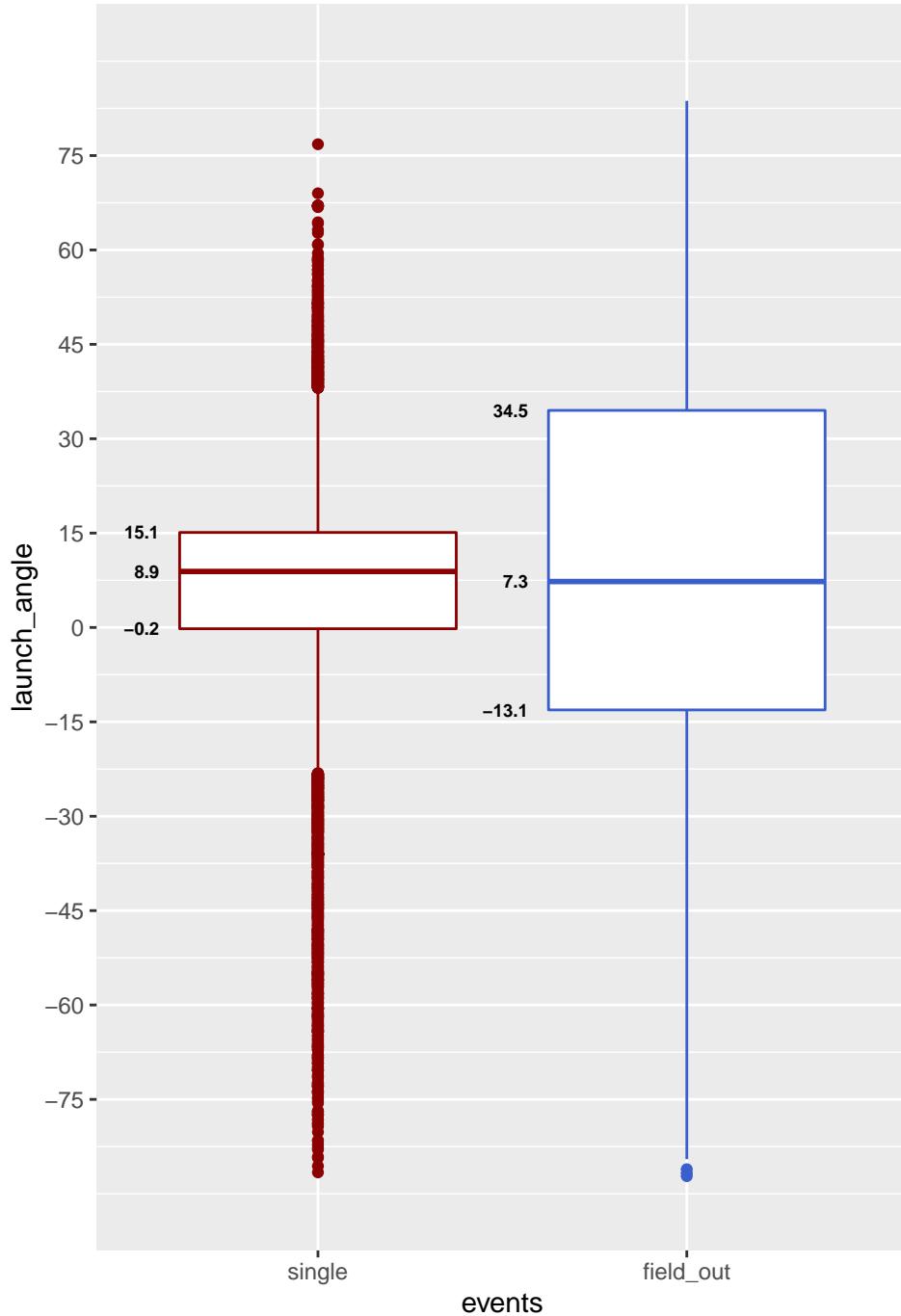
bb_outcome_by_launch_angle <- ggplot(battedballs,
                                         aes(x = events, y = launch_angle, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-75, 75, by = 15),
                     labels = c("-75", "-60", "-45", "-30", "-15", "0", "15", "30",
                               "45", "60", "75"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Launch Angle") +
  theme(legend.position = "none")

bb_outcome_by_launch_angle_data <- layer_data(bb_outcome_by_launch_angle)

launch_angle_single_1quartile <- bb_outcome_by_launch_angle_data[1, 3]
launch_angle_single_median <- bb_outcome_by_launch_angle_data[1, 4]
launch_angle_single_3quartile <- bb_outcome_by_launch_angle_data[1, 5]
launch_angle_out_1quartile <- bb_outcome_by_launch_angle_data[2, 3]
launch_angle_out_median <- bb_outcome_by_launch_angle_data[2, 4]
launch_angle_out_3quartile <- bb_outcome_by_launch_angle_data[2, 5]

ggplot(battedballs,
       aes(x = events, y = launch_angle, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-75, 75, by = 15),
                     labels = c("-75", "-60", "-45", "-30", "-15", "0", "15", "30",
                               "45", "60", "75"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Launch Angle") +
  theme(legend.position = "none") +
  annotate("text", x = c(.57, .57, .57), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_angle_single_1quartile, launch_angle_single_median,
                launch_angle_single_3quartile),
          label = c(launch_angle_single_1quartile, launch_angle_single_median,
                    launch_angle_single_3quartile)) +
  annotate("text", x = c(1.57, 1.57, 1.57), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_angle_out_1quartile, launch_angle_out_median,
                launch_angle_out_3quartile),
          label = c(launch_angle_out_1quartile, launch_angle_out_median,
                    launch_angle_out_3quartile))
```

Singles vs Outs, by Launch Angle



The boxplots revealed that singles tend to have a launch angle between 0 and 15 degrees. Many outs fell within this range as well, but outs had an interquartile range (-13.1 degrees to 34.5 degrees) more than three times as large as singles. Launch Angle was certainly going to help us distinguish singles from outs.

This was confirmed by quickly computing the proportion of each batted ball type resulting in a single. Singles comprised about 25% of our entire *battedballs* dataset, so this served as our baseline. Approximately 63% of Low Line Drives resulted in singles, followed by 45% of High Ground Balls, 29% of High Line Drives, 17% of Low Ground Balls, 9% of Fly Balls, and 2% of Popups (recall that the launch angles for High Ground Balls and Low Line Drives range from 0 to 19 degrees). The variability was significant; *launch_angle* would be a

very important predictor in our models.

```
# Compute the proportion of singles for each adv_bb_type.

(la_prop_lgb <- battedballs %>% filter(adv_bb_type == "low_ground_ball" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
                           filter(adv_bb_type == "low_ground_ball") %>%
                           summarize(n = n()))

##           n
## 1 0.1686908

(la_prop_hgb <- battedballs %>% filter(adv_bb_type == "high_ground_ball" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
                           filter(adv_bb_type == "high_ground_ball") %>%
                           summarize(n = n()))

##           n
## 1 0.4500062

(la_prop_lld <- battedballs %>% filter(adv_bb_type == "low_line_drive" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
                           filter(adv_bb_type == "low_line_drive") %>%
                           summarize(n = n()))

##           n
## 1 0.6309292

(la_prop_hld <- battedballs %>% filter(adv_bb_type == "high_line_drive" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
                           filter(adv_bb_type == "high_line_drive") %>%
                           summarize(n = n()))

##           n
## 1 0.2870396

(la_prop_fb <- battedballs %>% filter(adv_bb_type == "fly_ball" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
                           filter(adv_bb_type == "fly_ball") %>%
                           summarize(n = n()))

##           n
## 1 0.09390404
```

```
(la_prop_pu <- battedballs %>% filter(adv_bb_type == "popup" &
                                         battedballs$events == "single") %>%
  summarize(n = n()) / (battedballs %>%
    filter(adv_bb_type == "popup") %>%
    summarize(n = n()))
```

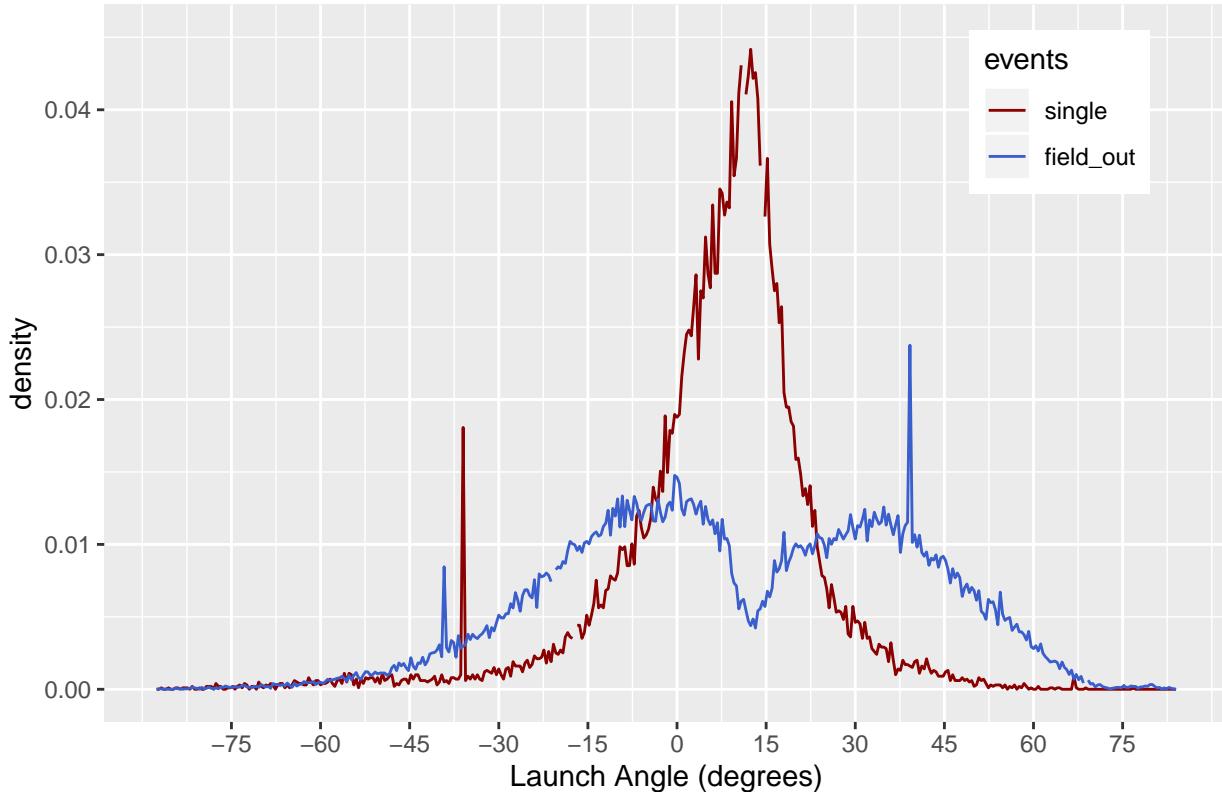
```
##          n
## 1 0.02485598
```

A comprehensive visualization of these proportions was achieved by creating a density plot. The density plot showed the proportion of singles and outs at each Launch Angle. By plotting proportions (densities) rather than counts, we avoided the distortion caused by one of our outcomes (out) having far more observations than our other outcome (single). The area under a density curve always totals 1.

```
# Plot the density of our Launch Angle predictor by result (single or out).
```

```
x_scale <- scale_x_continuous(breaks = seq(-75, 75, by = 15),
                               labels = c("-75", "-60", "-45", "-30", "-15", "0", "15",
                                         "30", "45", "60", "75"),
                               limits = NULL)
y_scale <- scale_y_continuous(limits = c(0, 0.045))
(la_density <- ggplot(battedballs, aes(x = launch_angle, y = stat(density),
                                         color = events)) +
  geom_freqpoly(binwidth = 0.4) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of Launch Angle by Result (Single or Out)",
       x = "Launch Angle (degrees)") +
  theme(legend.position = c(.85, .85)))
```

Density Plot of Launch Angle by Result (Single or Out)



```
la_density_data <- layer_data(la_density)
la_density_data <- filter(la_density_data, y != "NA")
```

Density rises for singles and falls for outs between launch angles of about 0 and 13 degrees. Density peaks for singles at a launch angle of 12.4. The density plot results appear to be consistent with those from our box plots and our proportions using *adv_bb_type*.⁶

Exit Velocity

MLB believes a hard hit ball likely to result in a hit has an exit velocity of at least 95 miles per hour. But is this true for singles?⁷ We again plotted the coordinates of singles and outs, this time by *launch_speed_cat*. The five categories of launch speeds were: less than 70 mph (comprised 11% of *battedballs*), 70-79.9 mph (14% of *battedballs*), 80-89.9 mph (28%), 90-100 (30%), and greater than 100 mph (17%).

```
# Visualize singles and outs, by Exit Velocity.

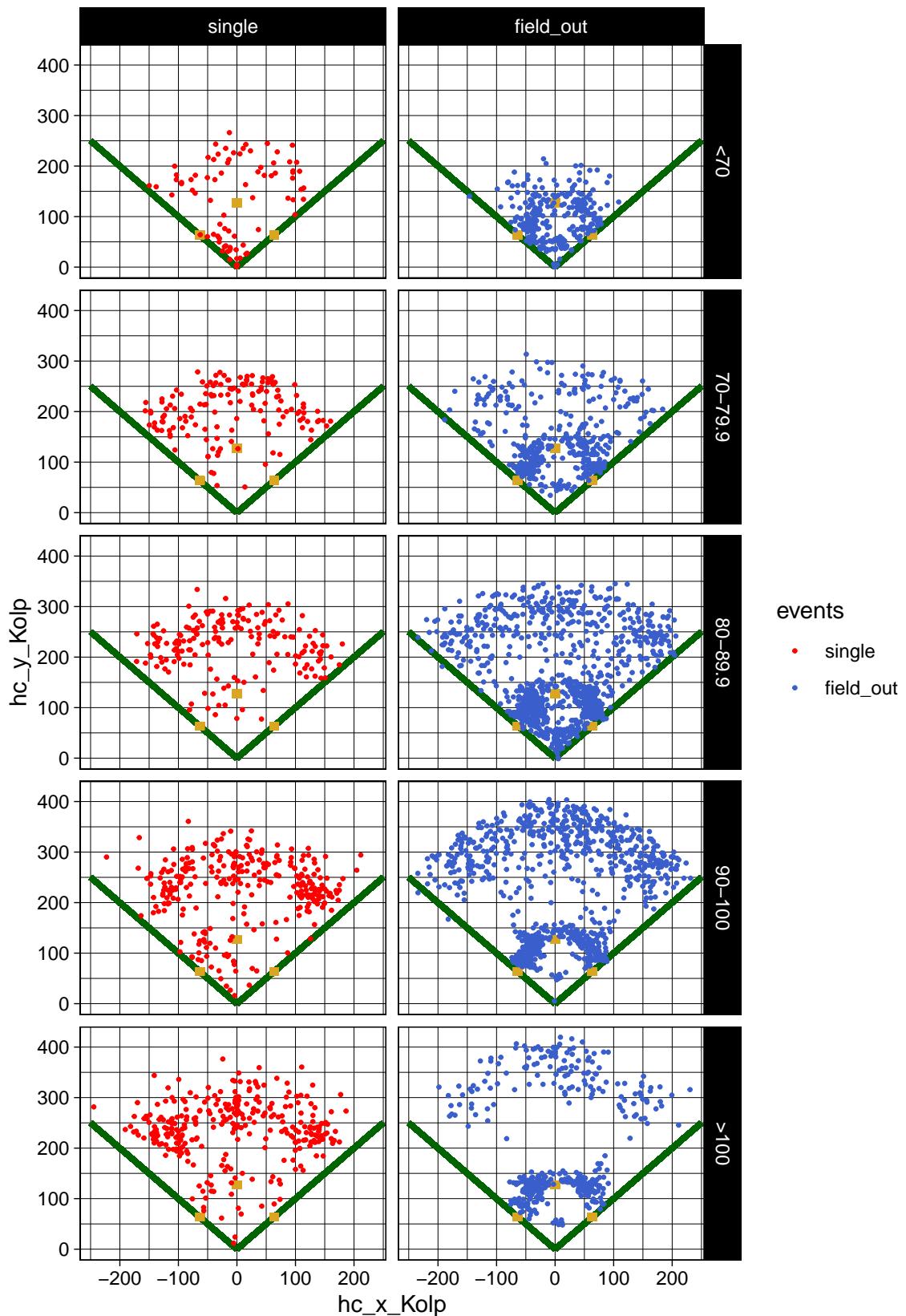
ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
```

⁶Note that the peripheral spikes appearing in the density plot are the result of Baseball Savant's methodology for imputing missing data caused by the Statcast technology failing to obtain measurements on certain batted balls. To provide clarity in this and other density plots, some of these spikes were removed by imposing scale limits on the axes. However, the imputed data was not removed from the *battedballs* dataset itself, as doing so would have introduced bias. Perpetua, Andrew, "Accounting for the 'No Nulls' Solution" (2017). <https://tbt.fangraphs.com/43416-2/>.

⁷<http://m.mlb.com/glossary/statcast/hard-hit-rate>

```
geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by launch_speed_cat") +
  facet_grid(launch_speed_cat ~ events)
```

Singles vs Outs, by launch_speed_cat



At exit velocities less than 80 mph, we see more outs being made in the infield than the outfield. The opposite appears true for balls hit at least 90 mph; these balls are reaching the outfielders on the fly, so they're being caught despite their high velocity.

The frequency of outs appears greatest in the middle velocity category, 80 to 89.9 mph. These batted balls are hit hard enough to reach the fielders, but not so hard that fielders don't have enough time to react. By comparison, singles seem to gradually increase in density as exit velocity increases. Again, we see that most of the singles end up in the shallow portions of the outfield, indicating they were hit well enough to get between or over the infielders. Exit velocity would certainly aid the batter in this respect. The singles in the lowest launch speed category are the exception, as a good portion of them result in infield singles on the left side. We saw these infield singles earlier when we analyzed launch angles.

Following the data exploration approach we used for Launch Angle, we next created box plots showing the distribution of *launch_speed* by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

```
# Create boxplots to visualize the distribution of launch_speed by singles and outs.

bb_outcome_by_launch_speed <- ggplot(battedballs,
                                         aes(x = events, y = launch_speed, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(10, 130, by = 15),
                     labels = c("10", "25", "40", "55", "70", "85", "100", "115", "130"),
                     limits = c(10, 130)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Exit Velocity") +
  theme(legend.position = "none")

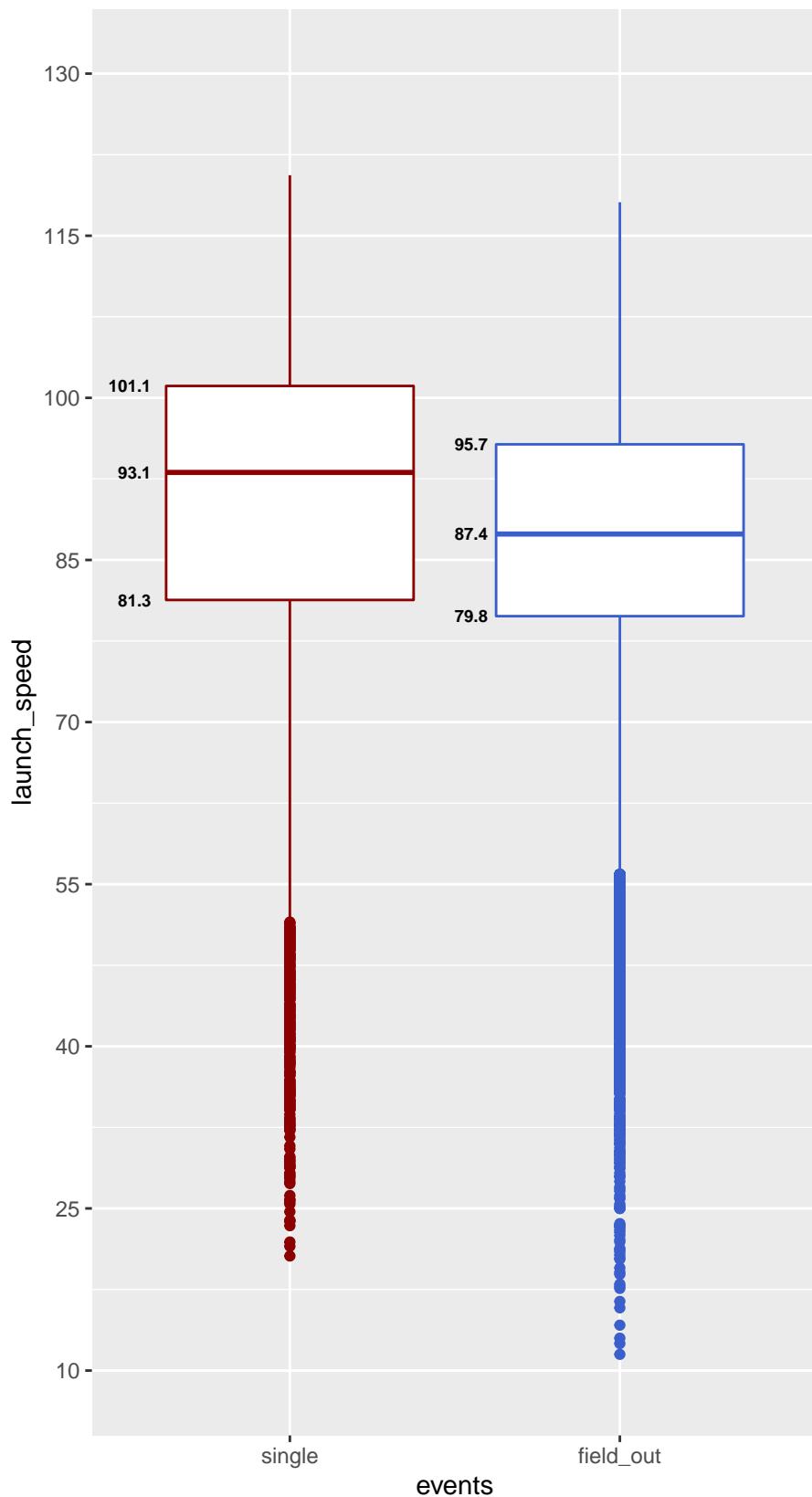
bb_outcome_by_launch_speed_data <- layer_data(bb_outcome_by_launch_speed)

launch_speed_single_1quartile <- bb_outcome_by_launch_speed_data[1, 3]
launch_speed_single_median <- bb_outcome_by_launch_speed_data[1, 4]
launch_speed_single_3quartile <- bb_outcome_by_launch_speed_data[1, 5]
launch_speed_out_1quartile <- bb_outcome_by_launch_speed_data[2, 3]
launch_speed_out_median <- bb_outcome_by_launch_speed_data[2, 4]
launch_speed_out_3quartile <- bb_outcome_by_launch_speed_data[2, 5]

ggplot(battedballs,
       aes(x = events, y = launch_speed, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(10, 130, by = 15),
                     labels = c("10", "25", "40", "55", "70", "85", "100", "115", "130"),
                     limits = c(10, 130)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Exit Velocity") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.58, .58, .58),
           y = c(launch_speed_single_1quartile, launch_speed_single_median,
                 launch_speed_single_3quartile),
           label = c(launch_speed_single_1quartile, launch_speed_single_median,
                     launch_speed_single_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6, 1.6, 1.6),
           y = c(launch_speed_out_1quartile, launch_speed_out_median,
                 launch_speed_out_3quartile),
           label = c(launch_speed_out_1quartile, launch_speed_out_median,
```

```
launch_speed_out_3quartile))
```

Singles vs Outs, by Exit Velocity



Here, the Exit Velocity boxplots for singles and outs were more similar to each other than the Launch Angle boxplots. Still, the median and third quartile were more than five miles per hour faster for singles than outs. MLB's 95 mile-per-hour threshold appeared to have some validity for singles. But it's also interesting to note that the second quartile begins at the relatively low exit velocity of 81.3 miles per hour, only 1.5 miles per hour faster than the second quartile for outs. It seemed that a fair number of singles were being hit at lower exit velocities. This perhaps distinguishes singles from other types of hits.

We next computed the proportion of batted balls resulting in a single for each launch speed category. Recall that singles comprise about 25% of our entire *battedballs* dataset, so that's our baseline. From the lowest to the highest velocity categories, the computed proportions (22%, 24%, 15%, 26%, and 43%) revealed an interesting pattern: the frequency of singles dropped in the mid-speed category (80 to 90 mph) and rose in the fastest category (above 100 mph), but were otherwise near the overall singles average of 25% of batted balls. That is, singles are hit with greatest frequency at the lower and higher ends of the exit velocity spectrum, while the 28% of batted balls hit between 80 and 90 miles per hour were more easily turned into outs!

That makes some sense because fielders need the ball to make a field out. If the ball is hit too slowly, the fielder can't get to the ball quickly enough to make an out. If the ball is hit too hard, the fielder isn't able to react fast enough to glove the ball. Still, to see this reflected in the numbers was a bit surprising. More importantly, the relative lack of variability in Exit Velocity across singles and outs suggested that our *launch_speed* predictor was not going to be as important as *launch_angle* in our models.

```
# Compute the proportion of singles for each launch_speed_cat.

(ls_prop_lt70 <- battedballs %>% filter(launch_speed_cat == "<70" &
                                             battedballs$events == "single") %>%
    summarize(n = n()) / (battedballs %>%
                           filter(launch_speed_cat == "<70") %>%
                           summarize(n = n()))

##               n
## 1 0.2242256

(ls_prop_to80 <- battedballs %>% filter(launch_speed_cat == "70-79.9" &
                                             battedballs$events == "single") %>%
    summarize(n = n()) / (battedballs %>%
                           filter(launch_speed_cat == "70-79.9") %>%
                           summarize(n = n()))

##               n
## 1 0.2366472

(ls_prop_to90 <- battedballs %>% filter(launch_speed_cat == "80-89.9" &
                                             battedballs$events == "single") %>%
    summarize(n = n()) / (battedballs %>%
                           filter(launch_speed_cat == "80-89.9") %>%
                           summarize(n = n()))

##               n
## 1 0.1453527
```

```
(ls_prop_100 <- battedballs %>% filter(launch_speed_cat == "90-100" &
                                         battedballs$events == "single") %>%
  summarize(n = n())) / (battedballs %>%
  filter(launch_speed_cat == "90-100") %>%
  summarize(n = n()))
```

```
##           n
## 1 0.2571764
```

```
(ls_prop_gt100 <- battedballs %>% filter(launch_speed_cat == ">100"
                                             & battedballs$events == "single") %>%
  summarize(n = n()) / (battedballs %>%
  filter(launch_speed_cat == ">100") %>%
  summarize(n = n()))
```

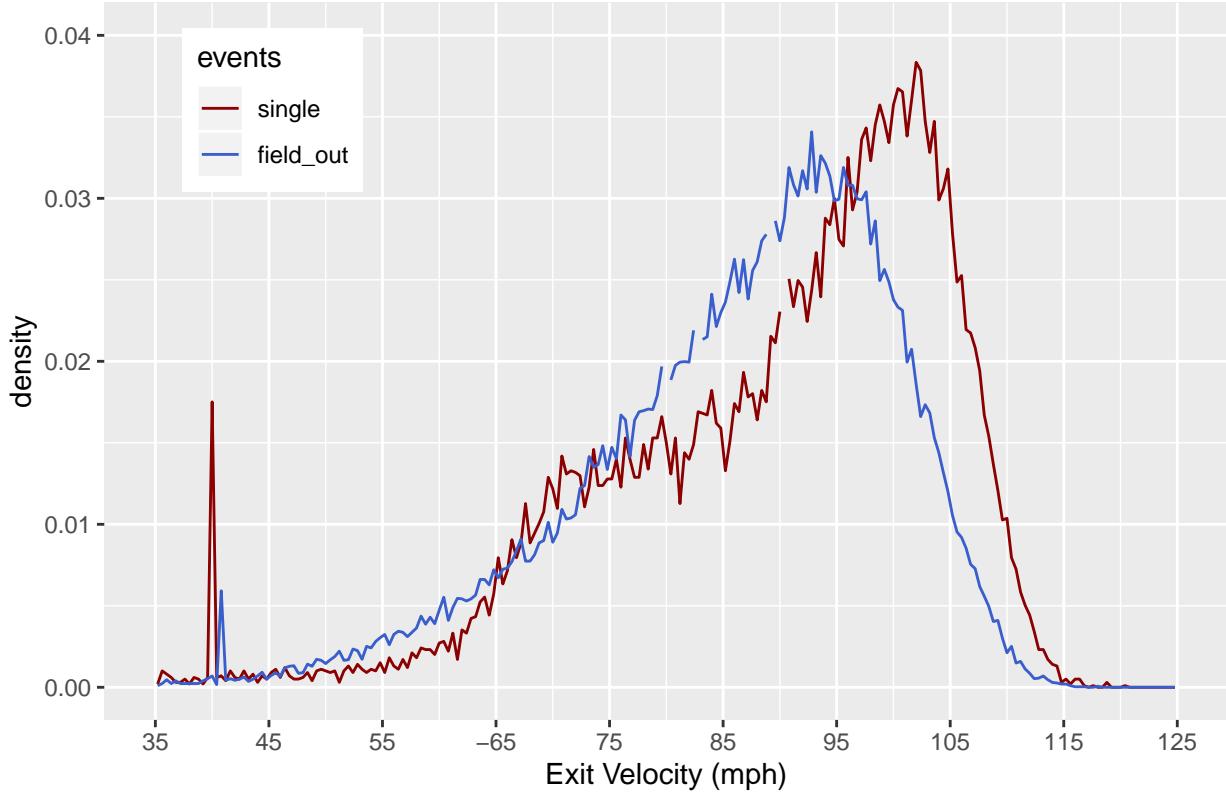
```
##           n
## 1 0.4278705
```

A density plot would provide a more thorough presentation of these proportions. The density plot we constructed shows the proportion of singles and outs at each Exit Velocity.

```
# Plot the density of our Exit Velocity predictor (*launch_speed*) by result (single or
# out).

x_scale <- scale_x_continuous(breaks = seq(35, 125, by = 10),
                               labels = c("35", "45", "55", "-65", "75", "85", 95, "105",
                                         "115", "125"),
                               limits = c(35, 125))
y_scale <- scale_y_continuous(limits = c(0, 0.04))
(ls_density <- ggplot(battedballs, aes(x = launch_speed, y = stat(density),
                                         color = events)) +
  geom_freqpoly(binwidth = 0.4) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of Exit Velocity by Result (Single or Out)",
       x = "Exit Velocity (mph)") +
  theme(legend.position = c(.15, .85)))
```

Density Plot of Exit Velocity by Result (Single or Out)



```
ls_density_data <- layer_data(ls_density)
ls_density_data <- filter(ls_density_data, y != "NA", x != "NA")
```

Our tentative conclusions above were largely corroborated by the density plots. The singles plot and the outs plot overlap for the most part, but the frequency of singles exceeds that of outs at higher (and even some lower) exit velocities. MLB's 95 miles per hour threshold is borne out with more precision here, though the threshold for singles appears to be closer to 97 or 98 based on the density plots. This actually makes sense because singles, being hit at lower launch angles than other types of hits, have to be hit harder to get through and past the infield.

Here, one wonders whether all the singles imputed around 40 miles per hour changed the lower tail of the singles plot in a significant way. In spite of the imputed data, the density plots reflect the same tendencies revealed throughout our exploration of the Exit Velocity data. Exit Velocity was going to be a helpful predictor in our models, but perhaps not to the same extent as Launch Angle.

Spray Angle

Spray Angle is the most complex of our key predictors. You'll recall that we expected (or at least hoped) our Spray Angle predictor would be aided by an interactive variable, Infield Fielding Alignment (*if_fielding_alignment* in *battedballs*). Thus, we restricted our exploration of Spray Angle to batted balls with a launch angle of 10 degrees or less (Low Ground Balls and High Ground Balls in our *adv_bb_type* variable). To facilitate plotting, we created a new random sample of just batted balls with launch angles of 10 degrees or less .

```
# Create a random sample of 2,000 ground balls.

set.seed(1)
battedballs_gb_sample <- sample_n(filter(battedballs, launch_angle <= 10), 2000,
                                    replace = FALSE)
```

This time, we would be plotting singles and outs by both a predictor (first, we'll use *spray_angle_Kolp*) and its interactive partner (*if_fielding_alignment*). Kolp's version of Spray Angle is not adjusted for the side of the plate on which the batter hits, so batted balls will appear on the true left and right sides of the field.

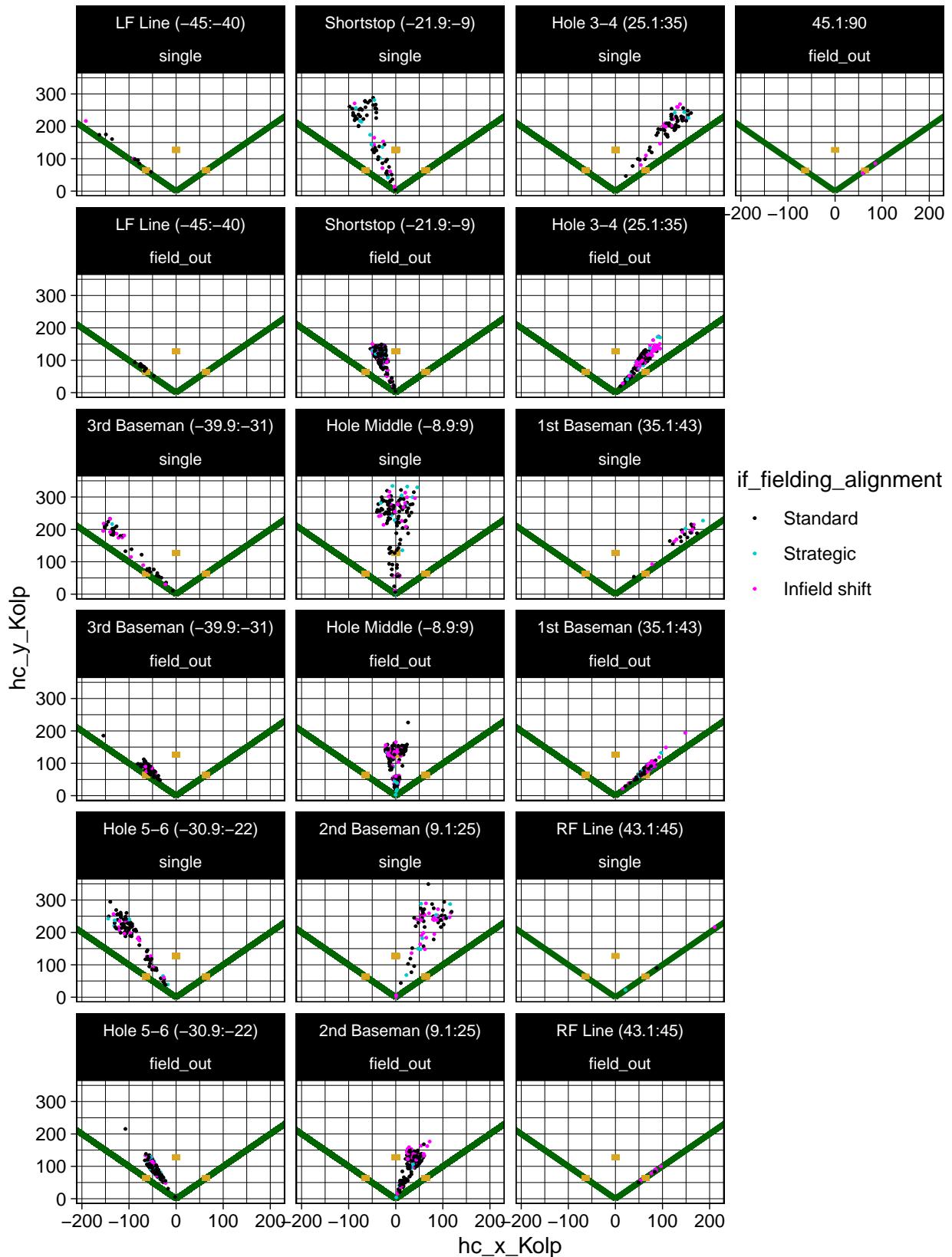
```
# Visualize singles and outs, by Spray Angle (Kolp's version).

spray_angle_Kolp_plot <- ggplot(battedballs_gb_sample,
                                 aes(hc_x_Kolp, hc_y_Kolp,
                                     color = if_fielding_alignment)) +
  scale_color_manual(values = c(Standard = "black", Strategic = "cyan3",
                                `Infield shift` = "magenta1")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.2) + theme_linedraw() +
  labs(title = "Singles vs Outs, by spray_angle_Kolp") +
  facet_wrap(~ spray_angle_Kolp_cat + events, dir = "v", nrow = 6) +
  theme(strip.text = element_text(size = 8))

# gtable_show_names(spray_angle_Kolp_plot)

reposition_legend(spray_angle_Kolp_plot, position = 'center', panel = 'panel-1-6')
```

Singles vs Outs, by spray_angle_Kolp



The first thing to note is that each panel displays batted balls within a certain range of spray angles. Together, the ranges cover the entire field from the left field foul line to the right field foul line. Instead of making each range the same size, an effort was made to define a range according to whether it extends into holes between infielders, or projects directly toward an infielder. For example, starting on the left side, the first panel displays a range of spray angles (-45 to -40 degrees) extending into a gap down the left field foul line. A third baseman would typically have to dive to his right to stop a ball hit in this range. The next panel to the right encompasses a range (-40 to -31 degrees) projecting directly toward the standard position of the third baseman. The next panel to the right (-31 to -22 degrees) displays balls heading toward the hole between the areas typically covered by the shortstop and third baseman. The panels progress in this manner from left to right until the right field foul line is reached.

The colors of the plotted batted balls indicate the alignment of the infielders at the time the pitch was delivered. Balls hit when the infielders were in their traditional locations (about 69% of batted balls) are shown in black, while those hit when the infielders were fully shifted (about 23% of batted balls) are shown in magenta. When the infielders were strategically positioned (about 8% of batted balls), the balls are plotted in cyan.

As would be expected, the plots show most of the singles reaching the outfield (except for the spattering of infield hits we've repeatedly seen on the left side of the infield). Outs, on the other hand, were stopped in the infield. No surprise there. More interesting is the observation that singles appear to be more dense in the panels showing spray angles that extend into gaps between the infielders, such as the Hole 5-6, Hole Middle, and Hole 3-4 panels. That suggests that *spray_angle_Kolp* ought to make a meaningful contribution to our predictive models.

Patterns from the infield alignment are more difficult to discern. There definitely seem to be more outs made up the middle and on the right side of the infield when the infielders are fully shifted. This didn't make sense until we realized that left-handed batters face non-standard infield alignments about half the time, while right-handed batters face non-standard infield alignments only 20% of the time.

But the key question remained as to whether infield alignment could help Spray Angle distinguish singles from outs. In theory, when infielders are in a non-standard alignment, we ought to see more singles being hit where infielders are traditionally located, and more outs being made where infield gaps are traditionally located. But this isn't readily apparent across the singles and outs panels.

It could be that the plots simply don't have the fine resolution to capture what's happening for the 4% of batted balls having launch angles of 10 degrees or less, and resulting in a single against a non-standard infield alignment. Perhaps infield shifts are so effective at preventing singles that we don't have enough singles in our sample to tell the story. It's also possible that the spray angle ranges simply aren't informative enough about the exact locations of the infielders. The picture may also be muddied by the fact that *spray_angle_Kolp* doesn't consider the side of the plate on which the batter stands. Perhaps we need to know whether the batter is pulling the ball into a shift, or hitting the ball away from the shift.

Fortunately, *spray_angle_adj* does contain this information.

```
# Visualize singles and outs, by *spray_angle_adj*

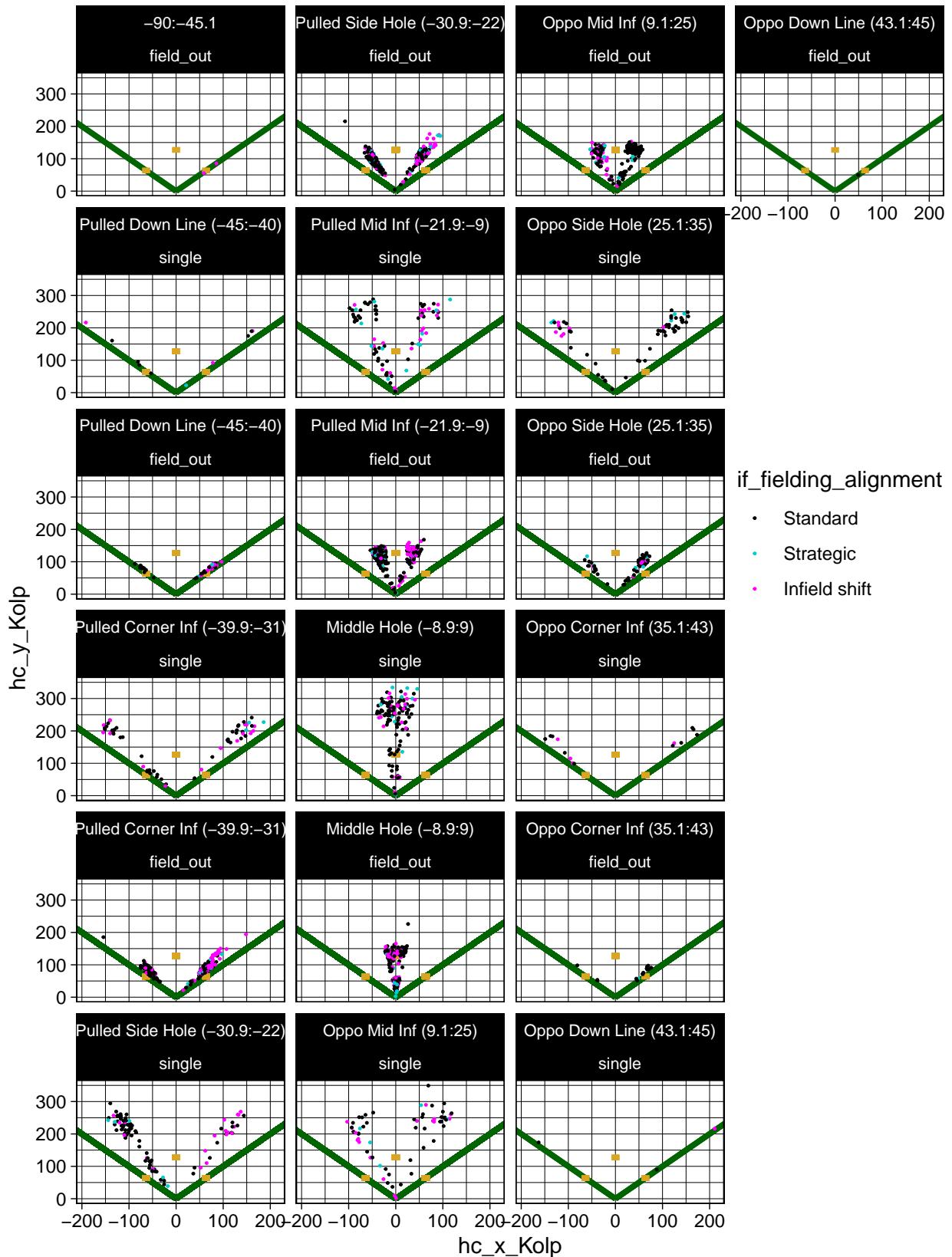
spray_angle_adj_plot <- ggplot(battedballs_gb_sample,
  aes(hc_x_Kolp, hc_y_Kolp, color = if_fielding_alignment)) +
  scale_color_manual(values = c(Standard = "black", Strategic = "cyan3",
    `Infield shift` = "magenta1")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
    fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
    fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
```

```
        fill = "goldenrod") +
geom_point(size = 0.2) + theme_linedraw() +
labs(title = "Singles vs Outs, by spray_angle_adj") +
facet_wrap(~ spray_angle_adj_cat + events, dir = "v", nrow = 6) +
theme(strip.text = element_text(size = 8))

# gtable_show_names(spray_angle_adj_plot)

reposition_legend(spray_angle_adj_plot, position = 'center', panel = 'panel-1-6')
```

Singles vs Outs, by spray_angle_adj



The plots of singles and outs by *spray_angle_adj* are organized quite differently than our *spray_angle_Kolp* plots. We still have the ranges of spray angles defined as they were earlier. But instead of progressing from the left side to the right side of the field, the panels progress from batted balls being pulled down either foul line to batted balls being hit toward the opposite field foul lines. Thus, you'll see two ranges of batted balls on each panel, one for a left-handed batter and one for a right-handed batter.

Recall that shifts deploy three infielders on the pull side of second base. The outs panels accordingly show (in magenta color) balls hit up the middle and into the pull side hole being turned into outs by shifted infielders. However, the singles panels still don't have enough data points to make any patterns visible. There's a nice collection of singles hit into the opposite side hole by left-handed hitters facing a shift. But that's about as much as we can see. At this point, the jury was still out on the usefulness of *if_fielding_alignment*.

We next created box plots showing the distribution of Spray Angle and Infield Alignment by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

```
# Create boxplots to visualize the distribution of Spray Angle and Infield Alignment
# by singles and outs.

battedballs_gb <- battedballs %>% filter(launch_angle <= 10)
battedballs_gb_std <- battedballs %>% filter(launch_angle <= 10,
                                              if_fielding_alignment == "Standard")
battedballs_gb_strat <- battedballs %>% filter(launch_angle <= 10,
                                                if_fielding_alignment == "Strategic")
battedballs_gb_shift <- battedballs %>% filter(launch_angle <= 10,
                                               if_fielding_alignment == "Infield shift")

# Standard Infield Alignment
bb_outcome_by_spray_angle_Kolp_std <- ggplot(battedballs_gb_std,
                                                 aes(x = events, y = spray_angle_Kolp,
                                                      color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                               "40", "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Standard Infield Alignment") +
  theme(legend.position = "none")

bb_outcome_by_spray_angle_Kolp_std_data <- layer_data(bb_outcome_by_spray_angle_Kolp_std)

spray_angle_Kolp_single_std_1quartile <- bb_outcome_by_spray_angle_Kolp_std_data[1, 3]
spray_angle_Kolp_single_std_median <- bb_outcome_by_spray_angle_Kolp_std_data[1, 4]
spray_angle_Kolp_single_std_3quartile <- bb_outcome_by_spray_angle_Kolp_std_data[1, 5]
spray_angle_Kolp_out_std_1quartile <- bb_outcome_by_spray_angle_Kolp_std_data[2, 3]
spray_angle_Kolp_out_std_median <- bb_outcome_by_spray_angle_Kolp_std_data[2, 4]
spray_angle_Kolp_out_std_3quartile <- bb_outcome_by_spray_angle_Kolp_std_data[2, 5]

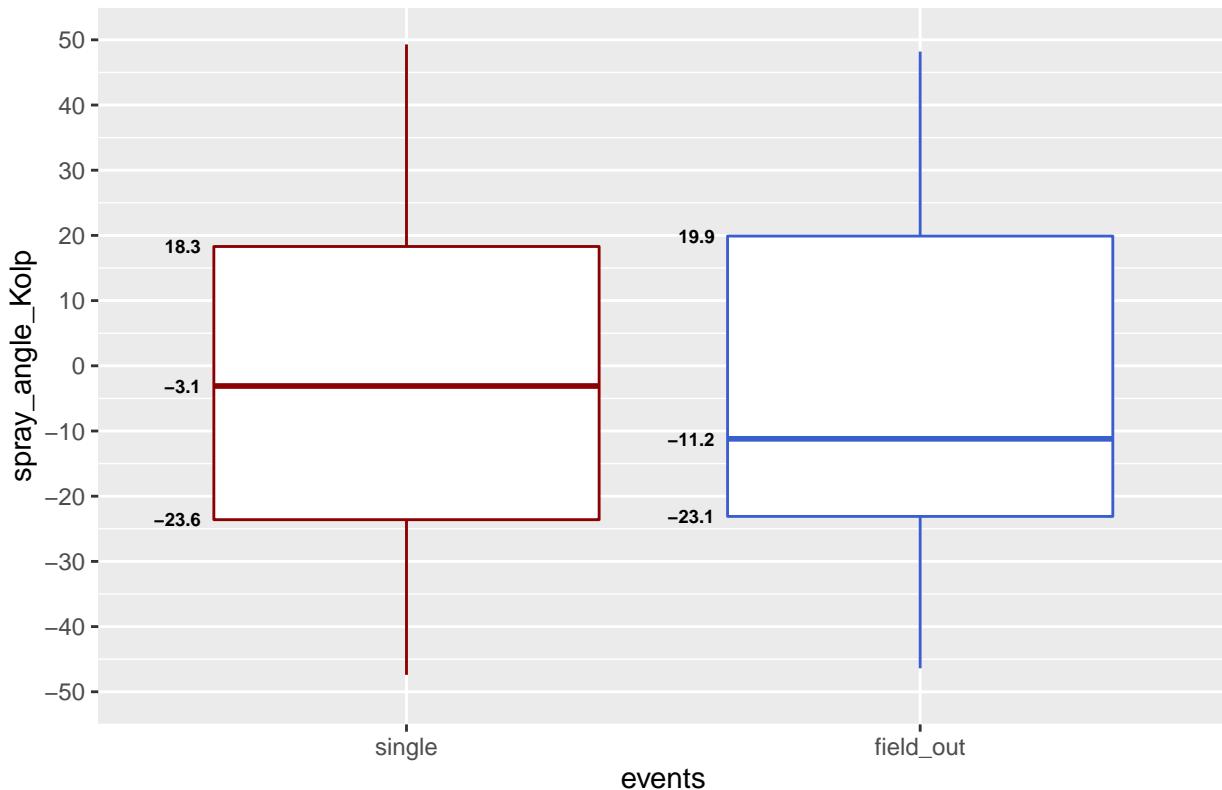
ggplot(battedballs_gb_std,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                               "40", "50"),
```

```

    limits = c(-50, 50)) +
scale_color_manual(values = c("red4", "royalblue3")) +
ggtitle("Singles vs Outs, by spray_angle_Kolp and Standard Infield Alignment") +
theme(legend.position = "none") +
annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
y = c(spray_angle_Kolp_single_std_1quartile,
spray_angle_Kolp_single_std_median,
spray_angle_Kolp_single_std_3quartile),
label = c(spray_angle_Kolp_single_std_1quartile,
spray_angle_Kolp_single_std_median,
spray_angle_Kolp_single_std_3quartile)) +
annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6, 1.6, 1.6),
y = c(spray_angle_Kolp_out_std_1quartile, spray_angle_Kolp_out_std_median,
spray_angle_Kolp_out_std_3quartile),
label = c(spray_angle_Kolp_out_std_1quartile, spray_angle_Kolp_out_std_median,
spray_angle_Kolp_out_std_3quartile))

```

Singles vs Outs, by spray_angle_Kolp and Standard Infield Alignment



```

# Strategic Infield Alignment
bb_outcome_by_spray_angle_Kolp_strat <- ggplot(battedballs_gb_strat,
                                                 aes(x = events, y = spray_angle_Kolp,
                                                     color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                               "40", "50")),

```

```

    limits = c(-50, 50)) +
scale_color_manual(values = c("red4", "royalblue3")) +
ggtitle("Singles vs Outs, by spray_angle_Kolp and Strategic Infield Alignment") +
theme(legend.position = "none")

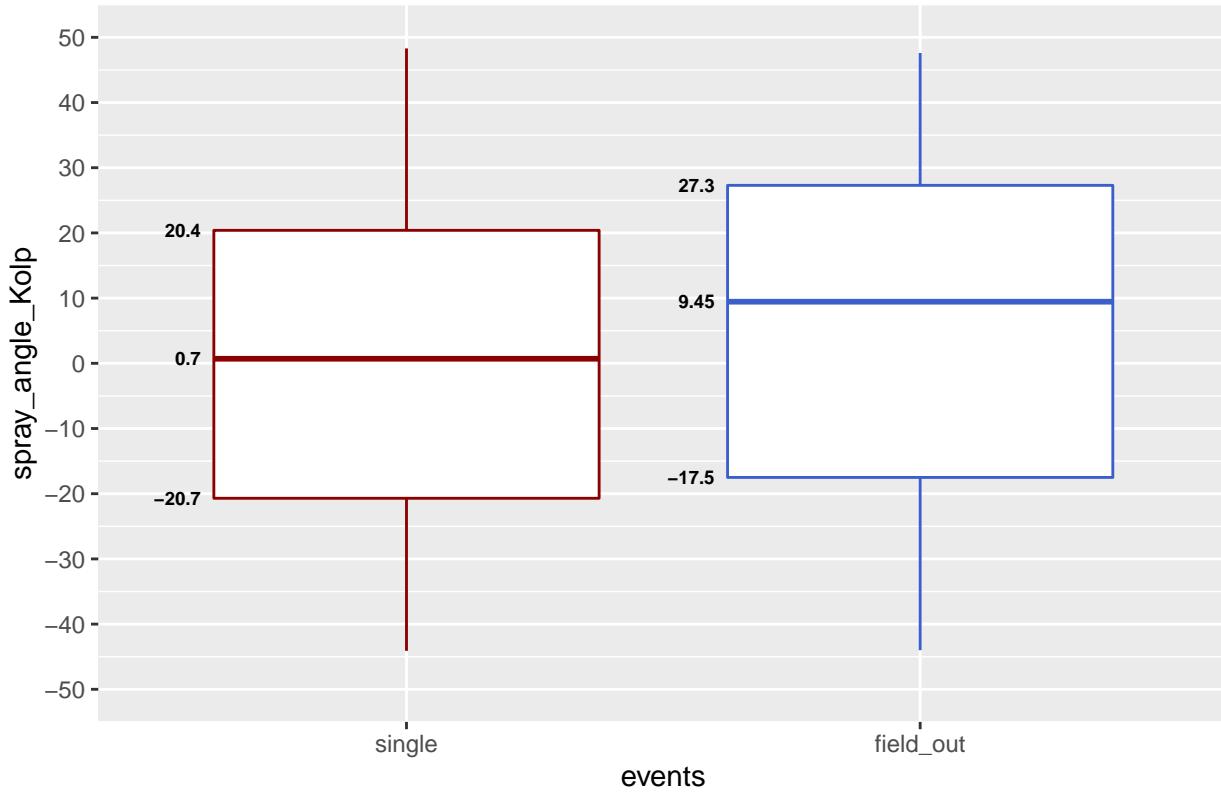
bb_outcome_by_spray_angle_Kolp_strat_data <-
layer_data(bb_outcome_by_spray_angle_Kolp_strat)

spray_angle_Kolp_single_strat_1quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 3]
spray_angle_Kolp_single_strat_median <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 4]
spray_angle_Kolp_single_strat_3quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 5]
spray_angle_Kolp_out_strat_1quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 3]
spray_angle_Kolp_out_strat_median <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 4]
spray_angle_Kolp_out_strat_3quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 5]

ggplot(battedballs_gb_strat,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
geom_boxplot() +
scale_y_continuous(breaks = seq(-50, 50, by = 10),
                   labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                             "40", "50")),
limits = c(-50, 50)) +
scale_color_manual(values = c("red4", "royalblue3")) +
ggtitle("Singles vs Outs, by spray_angle_Kolp and Strategic Infield Alignment") +
theme(legend.position = "none") +
annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
        y = c(spray_angle_Kolp_single_strat_1quartile,
               spray_angle_Kolp_single_strat_median,
               spray_angle_Kolp_single_strat_3quartile),
        label = c(spray_angle_Kolp_single_strat_1quartile,
                  spray_angle_Kolp_single_strat_median,
                  spray_angle_Kolp_single_strat_3quartile)) +
annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6, 1.6, 1.6),
        y = c(spray_angle_Kolp_out_strat_1quartile,
               spray_angle_Kolp_out_strat_median,
               spray_angle_Kolp_out_strat_3quartile),
        label = c(spray_angle_Kolp_out_strat_1quartile,
                  spray_angle_Kolp_out_strat_median,
                  spray_angle_Kolp_out_strat_3quartile))

```

Singles vs Outs, by spray_angle_Kolp and Strategic Infield Alignment



```
# Infield Shift Alignment

bb_outcome_by_spray_angle_Kolp_shift <- ggplot(battedballs_gb_shift,
                                                 aes(x = events, y = spray_angle_Kolp,
                                                      color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                               "40", "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Infield Shift Alignment") +
  theme(legend.position = "none")

bb_outcome_by_spray_angle_Kolp_shift_data <-
  layer_data(bb_outcome_by_spray_angle_Kolp_shift)

spray_angle_Kolp_single_shift_1quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 3]
spray_angle_Kolp_single_shift_median <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 4]
spray_angle_Kolp_single_shift_3quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 5]
spray_angle_Kolp_out_shift_1quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 3]
spray_angle_Kolp_out_shift_median <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 4]
spray_angle_Kolp_out_shift_3quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 5]

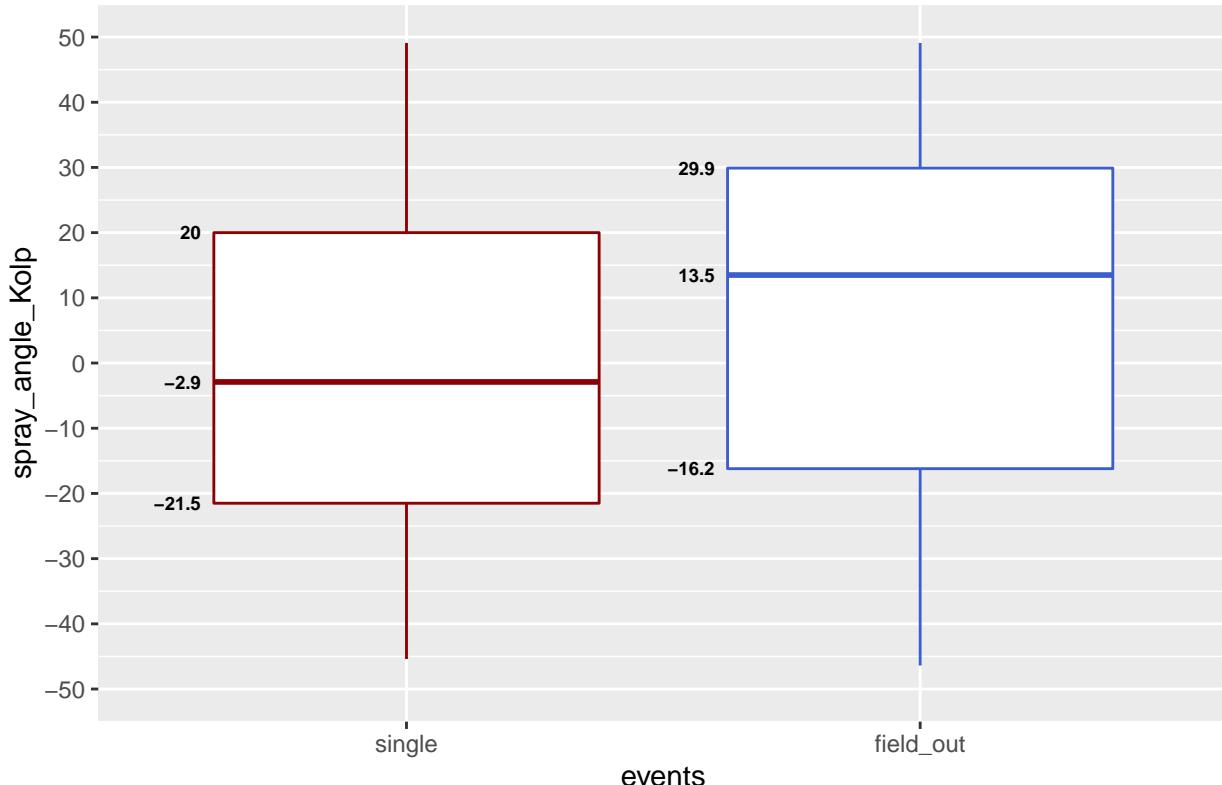
ggplot(battedballs_gb_shift,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
```

```

geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                               "40", "50")),
  limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Infield Shift Alignment") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(spray_angle_Kolp_single_shift_1quartile,
                  spray_angle_Kolp_single_shift_median,
                  spray_angle_Kolp_single_shift_3quartile),
           label = c(spray_angle_Kolp_single_shift_1quartile,
                     spray_angle_Kolp_single_shift_median,
                     spray_angle_Kolp_single_shift_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6, 1.6, 1.6),
           y = c(spray_angle_Kolp_out_shift_1quartile,
                  spray_angle_Kolp_out_shift_median,
                  spray_angle_Kolp_out_shift_3quartile),
           label = c(spray_angle_Kolp_out_shift_1quartile,
                     spray_angle_Kolp_out_shift_median,
                     spray_angle_Kolp_out_shift_3quartile))

```

Singles vs Outs, by spray_angle_Kolp and Infield Shift Alignment



Our *spray_angle_Kolp* box plots produced clearer results once *if_fielding_alignment* was taken into account. When infielders were in Standard alignment, the median spray angle was significantly lower for outs (-11.2 degrees) than singles (-3.1 degrees). That is, outs were more frequent when the ball was hit toward the

shortstop on the left side of the infield. Singles were more frequent when hit up the middle. So far, so good.

When infielders were in a non-standard alignment (Strategic or Shift), the entire interquartile range of spray angles was shifted up, encompassing much higher values for outs than for singles. The difference was particularly pronounced when the infield was fully shifted; the medians were 16.4 degrees apart (13.5 degrees as compared to -2.9)! So outs were much more frequent on balls hit to the right side of the infield, which would be into the shift for left-handed batters (we already know that the vast majority of shifts are deployed against left-handed batters). Singles, again, were more frequent when hit up the middle.

Note that the interquartile ranges for singles barely changed across infield alignments, but the interquartile ranges for outs changed markedly. This likely explains why our earlier plots of batted ball coordinates depicted recognizable spray angle and infield alignment patterns for outs, but not singles.

We next computed the proportion of batted balls resulting in a single for each *spray_angle_Kolp* category. Again, singles comprise about 25% of our entire *battedballs* dataset, so that's our baseline.

```
# Compute the proportion of singles for each *spray_angle_Kolp_cat* with a Standard
# Infield Fielding Alignment.

(sa_std_prop_LFline <- (battedballs_gb_std %>%
    filter(spray_angle_Kolp_cat == "LF Line (-45:-40)" &
           events == "single") %>% summarize(n = n())))
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "LF Line (-45:-40)") %>%
    summarize(n = n()))

##             n
## 1 0.2821918

(sa_std_prop_3b <- (battedballs_gb_std %>%
    filter(spray_angle_Kolp_cat == "3rd Baseman (-39.9:-31)" &
           events == "single") %>% summarize(n = n())))
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "3rd Baseman (-39.9:-31)") %>%
    summarize(n = n()))

##             n
## 1 0.1814528

(sa_std_prop_hole56 <- (battedballs_gb_std %>%
    filter(spray_angle_Kolp_cat == "Hole 5-6 (-30.9:-22)" &
           events == "single") %>% summarize(n = n())))
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole 5-6 (-30.9:-22)") %>%
    summarize(n = n()))

##             n
## 1 0.3228307

(sa_std_prop_ss <- (battedballs_gb_std %>%
    filter(spray_angle_Kolp_cat == "Shortstop (-21.9:-9)" &
           events == "single") %>% summarize(n = n())))
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Shortstop (-21.9:-9)") %>%
    summarize(n = n()))

##             n
## 1 0.1495143
```

```

(sa_std_prop_holeMid <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "Hole Middle (-8.9:9)" &
                                  events == "single") %>% summarize(n = n())))
/
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole Middle (-8.9:9)") %>%
  summarize(n = n()))

##           n
## 1 0.467631

(sa_std_prop_2b <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "2nd Baseman (9.1:25)" &
                                  events == "single") %>% summarize(n = n())))
/
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "2nd Baseman (9.1:25)") %>%
  summarize(n = n()))

##           n
## 1 0.1710062

(sa_std_prop_hole34 <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "Hole 3-4 (25.1:35)" &
                                  events == "single") %>% summarize(n = n())))
/
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole 3-4 (25.1:35)") %>%
  summarize(n = n()))

##           n
## 1 0.3420378

(sa_std_prop_1b <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "1st Baseman (35.1:43)" &
                                  events == "single") %>% summarize(n = n())))
/
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "1st Baseman (35.1:43)") %>%
  summarize(n = n()))

##           n
## 1 0.161244

(sa_std_prop_RFline <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "RF Line (43.1:45)" &
                                  events == "single") %>% summarize(n = n())))
/
(battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "RF Line (43.1:45)") %>%
  summarize(n = n()))

##           n
## 1 0.2105263

```

The computed proportions of singles given a Standard infield alignment largely follow the pattern we would expect to see if spray angles did indeed help distinguish singles from outs. Proportions are lowest when balls are hit at spray angles projecting toward the traditional location of an infielder; they're highest when balls are hit at spray angles extending toward traditionally located gaps between infielders (hole56, hole34, and especially up the middle). The variability in the proportions shows that **spray angle relative to**

infielders' locations has a role to play in our predictive models. However, it will likely be a less significant role than that of Launch Angle and Exit Velocity simply because we have defined the scope of our spray angle predictor such that it applies only to ground balls, which comprise a little more than half of our *battedballs* dataset.

We now wanted to see what the proportions of singles looked like across spray angles when the infield alignment was fully shifted. This time, we used *spray_angle_adj* because when the infielders are shifted, knowing whether the batter is left-handed or right-handed aids our understanding of the results.

```
# Compute the proportion of singles for each *spray_angle_adj_cat* with an Infield Shift
# Alignment.

(sa_shift_prop_LFline <- (battedballs_gb_shift %>%
  filter(spray_angle_adj_cat == "Pulled Down Line (-45:-40)" &
         events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Pulled Down Line (-45:-40)") %>%
    summarize(n = n())))

##           n
## 1 0.07430341

(sa_shift_prop_3b <- (battedballs_gb_shift %>%
  filter(spray_angle_adj_cat == "Pulled Corner Inf (-39.9:-31)" &
         events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Pulled Corner Inf (-39.9:-31)") %>%
    summarize(n = n())))

##           n
## 1 0.1343658

(sa_shift_prop_hole56 <- (battedballs_gb_shift %>%
  filter(spray_angle_adj_cat == "Pulled Side Hole (-30.9:-22)" &
         events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Pulled Side Hole (-30.9:-22)") %>%
    summarize(n = n())))

##           n
## 1 0.1321696

(sa_shift_prop_ss <- (battedballs_gb_shift %>%
  filter(spray_angle_adj_cat == "Pulled Mid Inf (-21.9:-9)" &
         events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Pulled Mid Inf (-21.9:-9)") %>%
    summarize(n = n())))

##           n
## 1 0.1646972
```

```

(sa_shift_prop_holeMid <- (battedballs_gb_shift %>%
                                filter(spray_angle_adj_cat == "Middle Hole (-8.9:9)" &
                                       events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Middle Hole (-8.9:9)") %>%
    summarize(n = n())))

```

```

##           n
## 1 0.3205989

```

```

(sa_shift_prop_2b <- (battedballs_gb_shift %>%
                           filter(spray_angle_adj_cat == "Oppo Mid Inf (9.1:25)" &
                                  events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Oppo Mid Inf (9.1:25)") %>%
    summarize(n = n())))

```

```

##           n
## 1 0.4870849

```

```

(sa_shift_prop_hole34 <- (battedballs_gb_shift %>%
                               filter(spray_angle_adj_cat == "Oppo Side Hole (25.1:35)" &
                                      events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Oppo Side Hole (25.1:35)") %>%
    summarize(n = n())))

```

```

##           n
## 1 0.5823755

```

```

(sa_shift_prop_1b <- (battedballs_gb_shift %>%
                           filter(spray_angle_adj_cat == "Oppo Corner Inf (35.1:43)" &
                                  events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Oppo Corner Inf (35.1:43)") %>%
    summarize(n = n())))

```

```

##           n
## 1 0.5228758

```

```

(sa_shift_prop_RFline <- (battedballs_gb_shift
                            %>% filter(spray_angle_adj_cat == "Oppo Down Line (43.1:45)" &
                                       events == "single") %>% summarize(n = n())) /
  (battedballs_gb_shift %>%
    filter(spray_angle_adj_cat == "Oppo Down Line (43.1:45)") %>%
    summarize(n = n())))

```

```

##           n
## 1 0.7142857

```

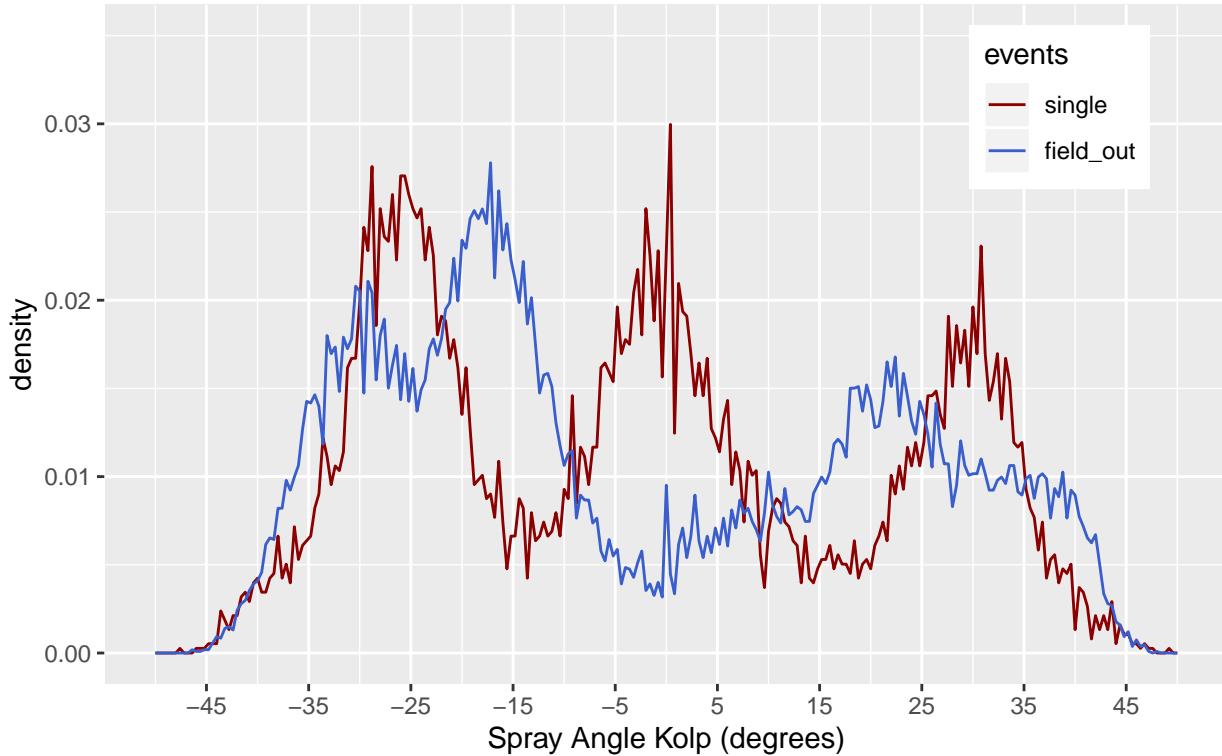
When a batter pulls the ball against the shift, the singles proportions never rise above 17% across the various pull spray angle ranges. Then we see a dramatic increase beginning with balls hit up the middle (32%) and progressively rising until we reach the opposite field foul line (64%). A shifted infield leaves few if any gaps on the pull side of the field, but huge gaps if the batter can manage to hit the ball to the opposite side. These numbers make credible our supposition that *if_fielding_alignment* adds useful information to our Spray Angle predictors.

Finally, we turned to the density plot to present a more comprehensive picture of singles and outs proportions without having to rely on arbitrarily constructed spray angle categories. The density plot we constructed showed the proportion of singles and outs at each Spray Angle and infield alignment.

```
# Plot the density of *spray_angle_Kolp* by result (single or out) and a Standard infield
# alignment.

x_scale <- scale_x_continuous(breaks = seq(-45, 45, by = 10),
                                labels = c("-45", "-35", "-25", "-15", "-5", "5", "15",
                                          "25", "35", "45"),
                                limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))
(sa_Kolp_density <- ggplot(battedballs_gb_std, aes(x = spray_angle_Kolp,
                                                       y = stat(density), color = events)) +
  geom_freqpoly(binwidth = 0.4) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of Spray Angle (Kolp) by Result (Single or Out)
        and a Standard Infield Alignment", x = "Spray Angle Kolp (degrees)") +
  theme(legend.position = c(.85, .85)))
```

Density Plot of Spray Angle (Kolp) by Result (Single or Out) and a Standard Infield Alignment



```
sa_Kolp_density_data <- layer_data(sa_Kolp_density)
sa_Kolp_density_data <- filter(sa_Kolp_density_data, y != "NA", x != "NA")
```

The density plot of *spray_angle_Kolp* with a Standard infield alignment quashes much of our uncertainty surrounding the locations of infield gaps, that is, spray angles at which ground balls are more likely to result in singles because they project toward locations between the infielders. We can state with more certainty now that the singles gaps (in degrees) are located at -30 to -23 (between the 3rd baseman and shortstop), -8 to 8 (up the middle, between the shortstop and second baseman), and 27 to 34 (between the second baseman and the first baseman).

The density plot reveals two other interesting aspects of spray angles. One is that there really are no singles gaps down the foul lines. When balls do get between the corner infielders and the foul lines, they are generally going to result in extra base hits, not singles. The other revelation is that some gaps are better than others for hitting singles. Specifically, the gap between the second baseman and the first baseman is not as good for singles as the gap between the 3rd baseman and shortstop. And the gap between the 3rd baseman and shortstop is not as good for singles as the gap up the middle. So we again see that the frequency of singles diminishes on the right side of the field because it's closer to first base. An infielder on the right side doesn't always have to field the ball cleanly to get the batter out at first base. If he just knocks the ball down, he's often able to recover and throw the batter out at first in time.

We now had to see if we could learn as much from a density plot of *spray_angle_adj* with a shifted infield.

```
# Plot the density of *spray_angle_adj* by result (single or out) and a shifted infield
# alignment.

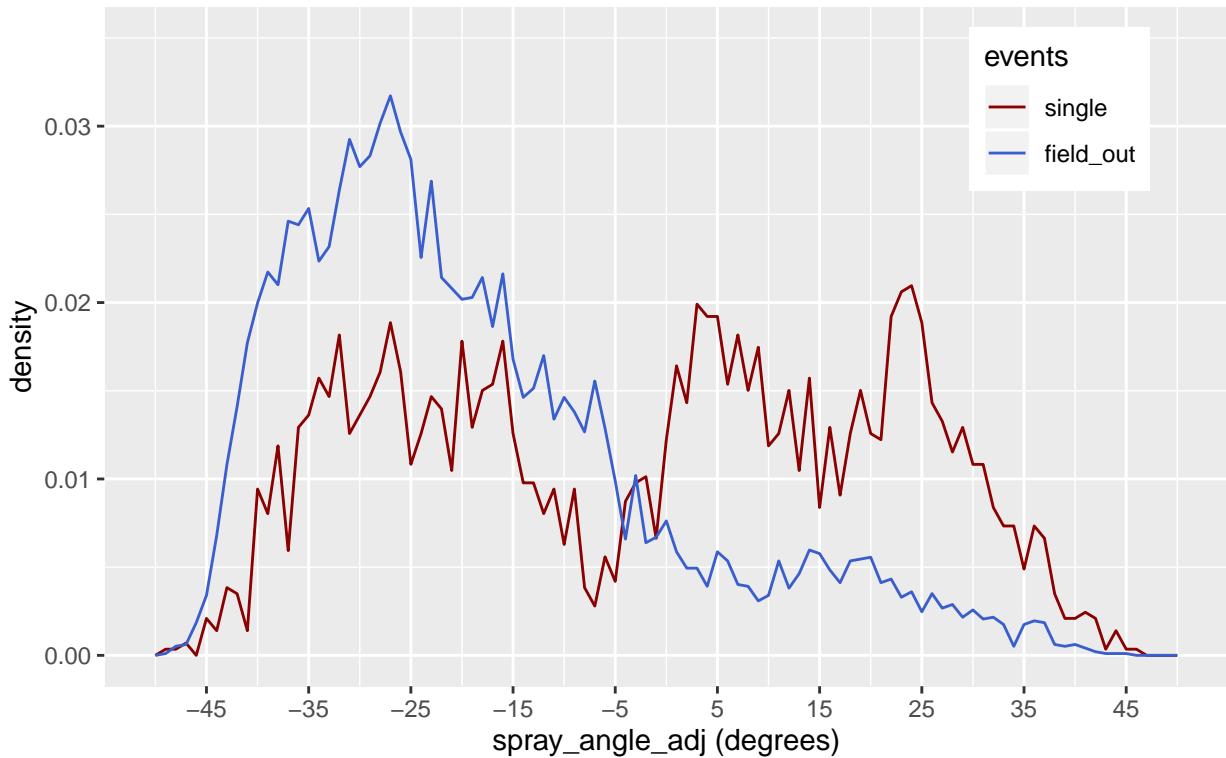
x_scale <- scale_x_continuous(breaks = seq(-45, 45, by = 10),
```

```

    labels = c("-45", "-35", "-25", "-15", "-5", "5", "15",
              "25", "35", "45"),
    limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))
(sa_adj_density <- ggplot(battedballs_gb_shift, aes(x = spray_angle_adj,
                                                       y = stat(density), color = events)) +
  geom_freqpoly(binwidth = 1) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of spray_angle_adj by Result (Single or Out)
        and a Shifted Infield Alignment", x = "spray_angle_adj (degrees)") +
  theme(legend.position = c(.85, .85)))

```

Density Plot of spray_angle_adj by Result (Single or Out)
and a Shifted Infield Alignment



```

sa_adj_density_data <- layer_data(sa_adj_density)
sa_adj_density_data <- filter(sa_adj_density_data, y != "NA", x != "NA")

```

Here we see once again that when the infield is shifted, there really is no good spray angle for singles on the pull side of the field. The best sliver of hope is perhaps between -20 and -15 degrees, where the middle infielder is normally positioned. The opposite side of the field is mostly undefended, except where the middle infielder on that side of the field normally plays.

Home to First

Like Spray Angle, our last key predictor to explore, `home_to_1b`, is only going to affect the likelihood of a single on certain batted ball events. In this case, our population of batted ball events is restricted to topped and weakly hit ground balls. Topped and weak hits are two levels of an MLB metric called `launch_speed_angle`, which attempts to characterize the quality of a batter's contact based on the combination of Launch Angle and Exit Velocity (https://baseballsavant.mlb.com/csv-docs#launch_speed_angle). MLB considers the batter's time to first base to matter most when balls are topped or weakly hit. So we opted to use MLB's metric, along with the requirement that the batted ball type be a ground ball.

```
# Create a subset of *battedballs* consisting only of Topped and Weakly Hit Ground Balls.

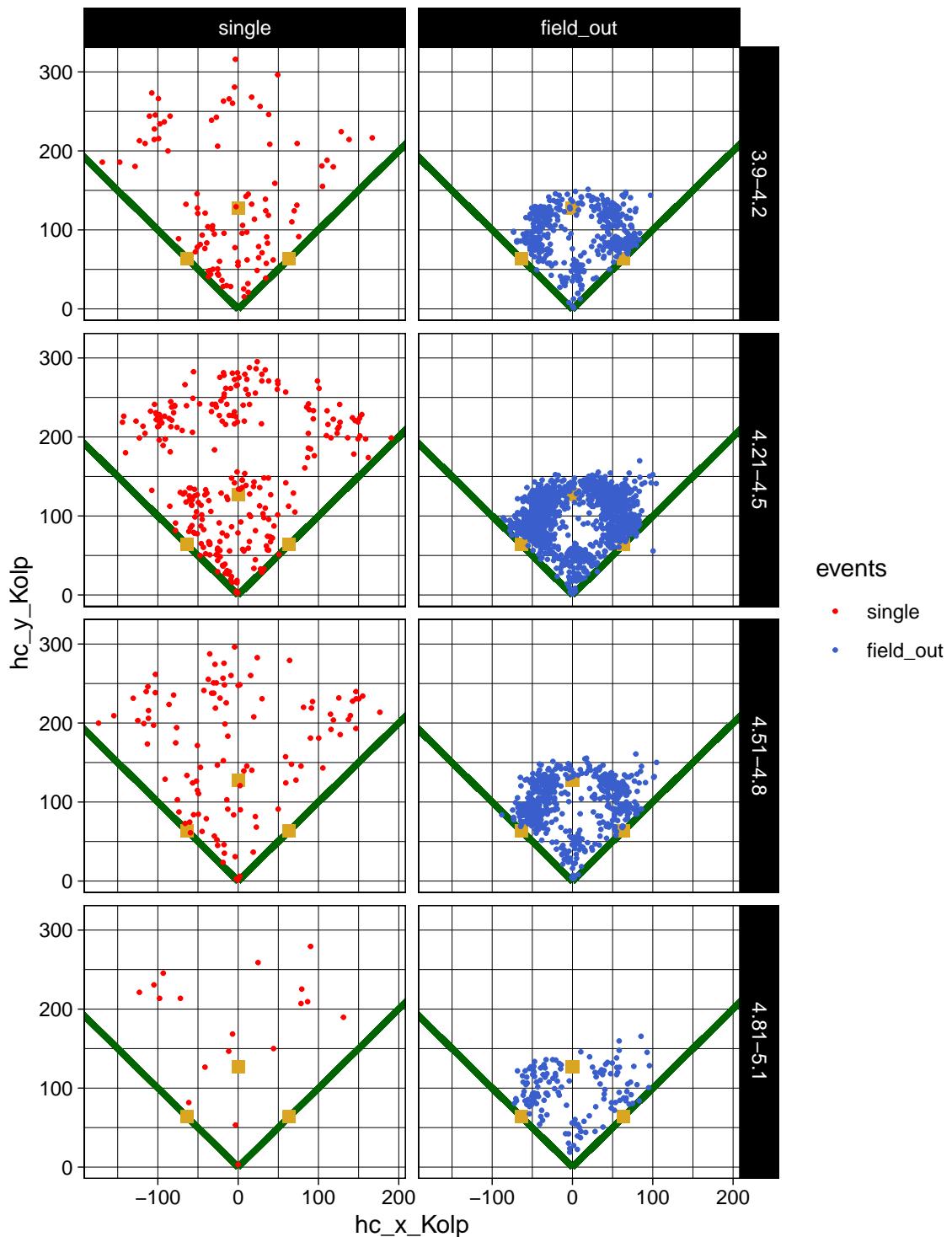
battedballs_lowgb_topweak <- battedballs %>% filter(bb_type == "ground_ball" &
                                                 adv_bb_type == "low_ground_ball")
battedballs_lowgb_topweak <- battedballs_lowgb_topweak %>%
  filter(launch_speed_angle == "1" | launch_speed_angle == "2")

set.seed(1)
battedballs_lowgb_topweak_sample <- sample_n(battedballs_lowgb_topweak, 3000,
                                               replace = FALSE)

# Visualize singles and outs, by Home to First predictor (topped and weakly hit ground
# balls only).
.

ggplot(battedballs_lowgb_topweak_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by hp_to_1b") +
  facet_grid(hp_to_1b_cat ~ events)
```

Singles vs Outs, by hp_to_1b



It's difficult to discern from these plots whether batters' Home to First measurements affect the likelihood of a single. The number of observations differs from row to row of the facet grid, and the proportions are hard to gauge. But in the second row (4.21-4.5), where the observations are most plentiful, it's worth noting a denser cluster of infield singles extending down the third base line, as compared to the first base line. So that theme continues here.

```

# Create boxplots to visualize the distribution of hp_to_1b by singles and outs.

bb_outcome_by_hp_to_1b <- ggplot(battedballs_lowgb_topweak,
                                    aes(x = events, y = hp_to_1b, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                               "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none")

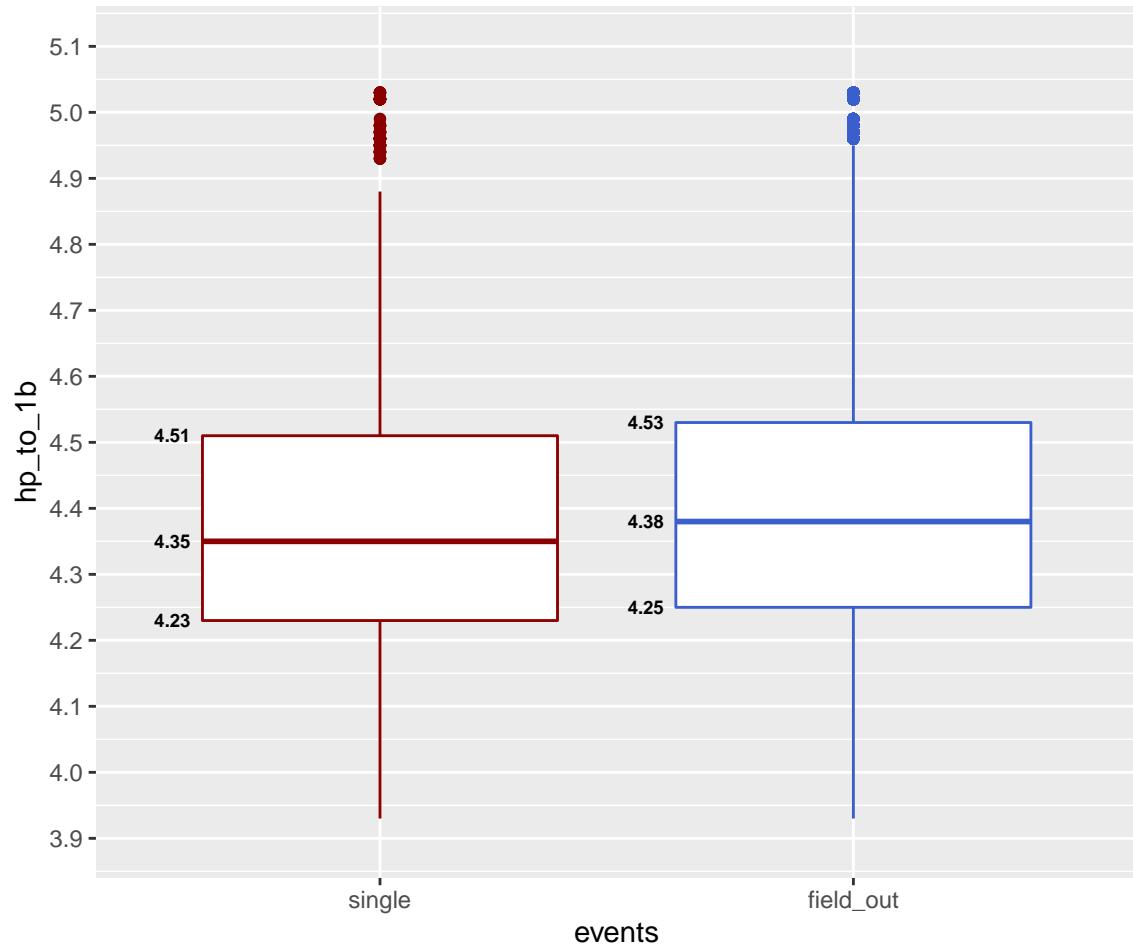
bb_outcome_by_hp_to_1b_data <- layer_data(bb_outcome_by_hp_to_1b)

hp_to_1b_single_1quartile <- bb_outcome_by_hp_to_1b_data[1, 3]
hp_to_1b_single_median <- bb_outcome_by_hp_to_1b_data[1, 4]
hp_to_1b_single_3quartile <- bb_outcome_by_hp_to_1b_data[1, 5]
hp_to_1b_out_1quartile <- bb_outcome_by_hp_to_1b_data[2, 3]
hp_to_1b_out_median <- bb_outcome_by_hp_to_1b_data[2, 4]
hp_to_1b_out_3quartile <- bb_outcome_by_hp_to_1b_data[2, 5]

ggplot(battedballs_lowgb_topweak,
       aes(x = events, y = hp_to_1b, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                               "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(hp_to_1b_single_1quartile, hp_to_1b_single_median,
                 hp_to_1b_single_3quartile),
           label = c(hp_to_1b_single_1quartile, hp_to_1b_single_median,
                     hp_to_1b_single_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6, 1.6, 1.6),
           y = c(hp_to_1b_out_1quartile, hp_to_1b_out_median, hp_to_1b_out_3quartile),
           label = c(hp_to_1b_out_1quartile, hp_to_1b_out_median, hp_to_1b_out_3quartile))

```

Singles vs Outs, by Home to First
 (topped and weakly hit ground balls only)

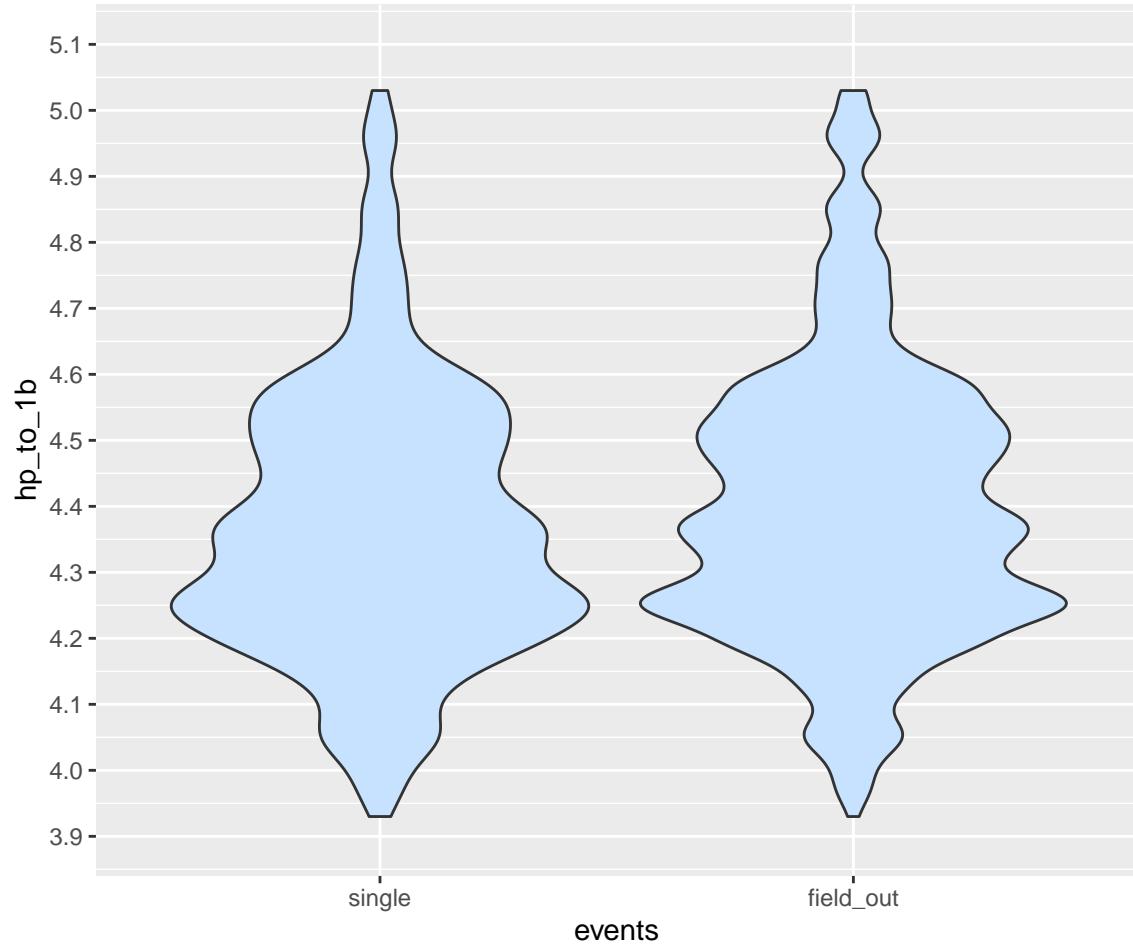


The boxplots aren't much more informative than our plots of batted ball coordinates. The Home to First stats barely differ between singles and outs. We decided to see whether a violin plot revealed more visual cues.

```
# Create violin plots to visualize the distribution of hp_to_1b by singles and outs.

ggplot(battedballs_lowgb_topweak,
       aes(x = events, y = hp_to_1b)) +
  geom_violin(fill = "slategray1") +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                               "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none")
```

Singles vs Outs, by Home to First (topped and weakly hit ground balls only)



Here, the differences in `hp_to_1b` for singles and outs were a little more apparent. There are more singles for times of 4.1 seconds or less, and there are fewer singles for times of 4.5 or more seconds. But the differences aren't dramatic, and we wondered whether the `hp_to_1b` feature was going to be a meaningful contributor to our models.

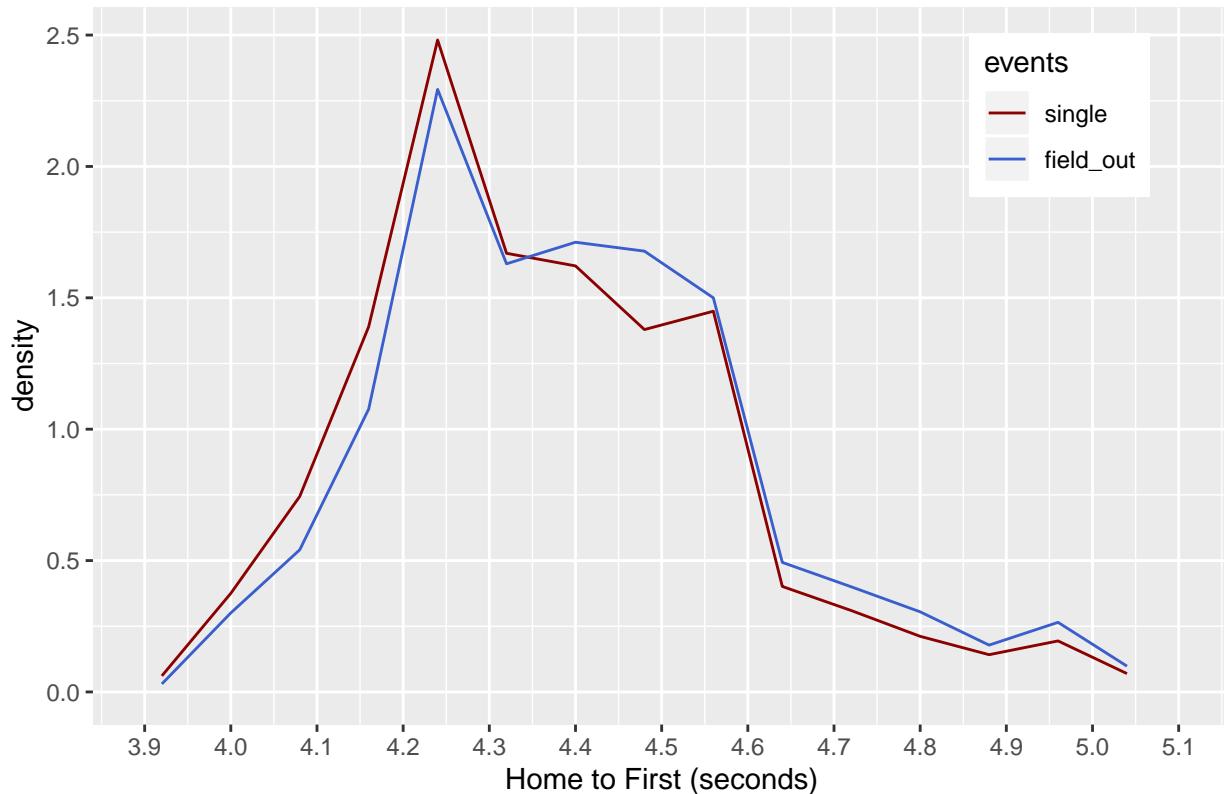
Our last look at `hp_to_1b` used density plots.

```
# Plot the density of our Home to First predictor (*hp_to_1b*) by result (single or out).

x_scale <- scale_x_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                               labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",
                                         "4.6", "4.7", "4.8", "4.9", "5.0", "5.1"),
                               limits = c(3.9, 5.1))
y_scale <- scale_y_continuous()
(h1b_density <- ggplot(battedballs_lowgb_topweak, aes(x = hp_to_1b, y = stat(density),
                                                       color = events)) +
  geom_freqpoly(binwidth = .08) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of Home to First by Result (Single or Out)",
```

```
x = "Home to First (seconds)" +
theme(legend.position = c(.85, .85)))
```

Density Plot of Home to First by Result (Single or Out)



```
h1b_density_data <- layer_data(h1b_density)
h1b_density_data <- filter(h1b_density_data, y != "NA", x != "NA")
```

Here we could identify with greater clarity the point at which the batter's time to first base affected the likelihood of getting a single. Singles were slightly more dense than outs for batters who could reach first base in 4.3 seconds or less (about 35% of the players for whom we have Home to First data). That makes sense. But the density plots for singles and outs were very close to each other, perhaps because there are so few batted balls for which *hp_to_1b* differences of tenths of a second are really going to turn an out into a single or vice versa. In other words, when it comes to hitting singles, Home to First speed just doesn't matter much except in relatively rare circumstances.

Prepare the Data for Use in Machine Learning Algorithms

Before we could begin to develop some models using different algorithms, we needed to rescale our data. Depending on the algorithm being used and the purpose for which its being used, if the distributions of the predictor variables are significantly different, rescaling mitigates these differences so that certain predictors don't dominate the algorithm simply by virtue of their large scale. Rescaling involves adding or subtracting a constant from each predictor value, and then multiplying or dividing by a constant.

Two primary methods are commonly used to rescale predictor values. The first, Normalization (sometimes referred to as Min-Max Scaling), involves subtracting the minimum value and then dividing by the range.

This has the effect of making the minimum value of the predictor zero and the maximum value one. Normalization is considered most useful when using a distance-based algorithm (such as K-Nearest Neighbors) for classification purposes.

The other primary method is Standardization (sometimes referred to as Z-Score Normalization, just to confuse things). It involves subtracting a measure of location such as the mean, and then dividing by the standard deviation. This technique transforms the predictor values so that their mean is zero with a standard deviation of one. Standardization is considered most useful when using linear regression, logistic regression, and linear discriminant analysis.

Here, we employed both Normalization and Standardization since we expected to be testing a variety of algorithms.

Also, since our outcome (single or out) is categorical rather than continuous, the *events* variable, which contained the outcome, was coded as a factor to facilitate the process of constructing a predictive machine learning algorithm or model.

```
# Make sure battedballs$events is coded as a factor with levels
# single, followed by field_out.

battedballs$events <- factor(battedballs$events, levels = c("single", "field_out"))

# Confirm which level of battedballs$events has been assigned to 0.

contrasts(as.factor(battedballs$events))

##           field_out
## single          0
## field_out       1

# Center the predictor variables to reduce non-essential multicollinearity among the
# predictors and improve the interpretation of their coefficients.

battedballs <- battedballs %>% mutate(c_launch_angle = launch_angle - mean(launch_angle))
battedballs <- battedballs %>% mutate(c_launch_speed = launch_speed - mean(launch_speed))
battedballs <- battedballs %>%
  mutate(c_spray_angle_Kolp = spray_angle_Kolp - mean(spray_angle_Kolp))
battedballs <- battedballs %>%
  mutate(c_spray_angle_adj = spray_angle_adj - mean(spray_angle_adj))
battedballs <- battedballs %>% mutate(c_hp_to_1b = hp_to_1b - mean(hp_to_1b))
battedballs <- battedballs %>%
  mutate(c_if_alignment = as.numeric(num_if_alignment) -
    mean(as.numeric(num_if_alignment)))

# Standardize the predictor variables.

battedballs <- battedballs %>% mutate(s_launch_angle = c_launch_angle / sd(launch_angle))
battedballs <- battedballs %>% mutate(s_launch_speed = c_launch_speed / sd(launch_speed))
battedballs <- battedballs %>%
  mutate(s_spray_angle_Kolp = c_spray_angle_Kolp / sd(spray_angle_Kolp))
battedballs <- battedballs %>%
  mutate(s_spray_angle_adj = c_spray_angle_adj / sd(spray_angle_adj))
battedballs <- battedballs %>% mutate(s_hp_to_1b = c_hp_to_1b / sd(hp_to_1b))
battedballs <- battedballs %>% mutate(s_if_alignment = c_if_alignment /
  sd(as.numeric(num_if_alignment)))
```

```
# Normalize the predictor variables.

battedballs <- battedballs %>%
  mutate(n_launch_angle = (launch_angle - min(launch_angle)) /
    (max(launch_angle) - min(launch_angle)))
battedballs <- battedballs %>%
  mutate(n_launch_speed = (launch_speed - min(launch_speed)) /
    (max(launch_speed) - min(launch_speed)))
battedballs <- battedballs %>%
  mutate(n_spray_angle_Kolp = (spray_angle_Kolp - min(spray_angle_Kolp)) /
    (max(spray_angle_Kolp) - min(spray_angle_Kolp)))
battedballs <- battedballs %>%
  mutate(n_spray_angle_adj = (spray_angle_adj - min(spray_angle_adj)) /
    (max(spray_angle_adj) - min(spray_angle_adj)))
battedballs <- battedballs %>%
  mutate(n_hp_to_1b = (hp_to_1b - min(hp_to_1b)) /
    (max(hp_to_1b) - min(hp_to_1b)))
battedballs <- battedballs %>%
  mutate(n_if_alignment = (num_if_alignment - min(num_if_alignment)) /
    (max(num_if_alignment) - min(num_if_alignment)))
```

We also needed to randomly partition our *battedballs* data frame into a training set and a test set. The training set would be used to train or construct the algorithm, and the test set would be used to determine how accurately the algorithm predicted the outcome (single or out) from an independent dataset. About 20 percent of batted balls was allocated to the newly created *test_set* tibble, while the remainder was used for the new *train_set* tibble.

```
# Partition battedballs.

set.seed(1)
test_index <- createDataPartition(y = battedballs$events, times = 1, p = 0.20,
                                   list = FALSE)
train_set <- battedballs[-test_index,]
test_set <- battedballs[test_index,]
```

Build and Assess a Baseline Model

Our first model used an algorithm that randomly guessed the outcomes in *test_set*. The predictive ability of this guessing algorithm could then be used as a baseline for comparison to subsequently developed models.

The crucial decision here was how we would measure the predictive ability of the guessing algorithm and others we would construct. Our project sought to determine whether a model could reliably distinguish between a single and an out. This presented a particular machine learning problem called prevalence. We had a classification problem with a dichotomous categorical response variable (*events*), and one of the response variable's two possible values (*field_out*) occurring far more frequently than the other. Specifically, 75% of the batted balls in our dataset were field outs, while only 25% were singles. Imbalanced outcome classes cause the threshold that best differentiates the outcomes to shift in favor of the majority class. Under these circumstances, methods for measuring predictive ability will often produce very misleading results, leading to the adoption of inferior models. Thus, our *training_set* could be considered biased.

We needed to use a method that took prevalence into account. Bowen Song, Guopeng Zhang, Wei Zhu & Zhengrong Liang have proposed three new assessment methods (shortest distance, harmonic mean and anti-

harmonic mean) that specifically address the challenge of classifying imbalanced data.⁸ The authors found the performance of their proposed methods to be “adaptive and robust with different levels of imbalanced data.” We opted to use two of these methods, harmonic mean and shortest distance, for our project.

The harmonic mean method computes a harmonic average of sensitivity and specificity, which we labeled **truescore** because we care about correctly identifying both singles (measured by sensitivity or the true positive rate (TPR)) and outs (measured by specificity or the true negative rate (TNR)). According to the authors, the manner in which a harmonic mean is calculated means that “the larger the difference of the elements in the pair, the smaller the harmonic mean is.” Thus, “it mitigate[s] the influence of the larger value and increase[s] the influence of the small value.... It pays more attention to the balance of the pair compared to the arithmetic mean.” To the extent a model’s decision point (i.e., selection point, cutoff or threshold) varied from the one that maximized the harmonic average of sensitivity and specificity, it was inferior.

When it made sense, we also applied a second method, shortest distance (or minimum distance), to evaluate the predictive ability of our models. A receiver operating characteristic (ROC) curve is a widely used plot for depicting the tradeoff between a model’s sensitivity (TPR) and its false positive rate (FPR, which is 1 - specificity) for a given decision point. The model’s possible decision points form the actual ROC curve. The ideal coordinate on the graph, where sensitivity and specificity are both maximized, is (0, 1). Hence, the point on the ROC curve closest to (0, 1) represents the model’s optimal decision point. To the extent a model’s decision point varies from the one closest to (0, 1), it is inferior. The authors provide an equation for computing minimum distance using sensitivity and specificity.

In the case of our baseline model, we computed a truescore even though it employed a guessing algorithm, and did not consider the data underlying our predictor variables.

```
# Develop a baseline model that guesses the outcomes in the test_set.

set.seed(1)
bl_preds <- sample(c("single", "field_out"), size = length(test_index),
                     replace = TRUE, prob = c(0.25, 0.75)) %>%
  factor(levels = levels(test_set$events))
(bl_confusionM <- confusionMatrix(data = bl_preds, reference = test_set$events))

## Confusion Matrix and Statistics
##
##             Reference
## Prediction   single field_out
##     single      1294      3775
##     field_out    3687     11362
##
##             Accuracy : 0.6291
##                 95% CI : (0.6224, 0.6358)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : 1.0000
##
##             Kappa : 0.0103
##
##     Mcnemar's Test P-Value : 0.3139
##
##             Sensitivity : 0.25979
##             Specificity  : 0.75061
##     Pos Pred Value : 0.25528
##     Neg Pred Value : 0.75500
```

⁸Song, B., Zhang, G., Zhu, W. et al.

```

##          Prevalence : 0.24759
##          Detection Rate : 0.06432
##  Detection Prevalence : 0.25196
##          Balanced Accuracy : 0.50520
##
##          'Positive' Class : single
##



```

Single is the positive outcome. With a TPR of only .26 and a TNR of .75, our guessing algorithm ended up with a truescore of .386. Other models we develop ought to do better.

So with a categorical outcome, five continuous numeric predictors, and a conditional variable (*num_if_alignment*, Spray Angle's interactive partner), we needed to select an appropriate type of model, that is, one capable of classifying each batted ball observation as either a single or an out, based on its corresponding Launch Angle, Exit Velocity, Spray Angle (two versions), Infield Alignment, and Home to First measurements.

Build and Assess a Logistic Regression Model

Generalized linear models adapt the linear model framework to cover non-normal dependent variables, such as a categorical outcome. They assume that the outcome or response variable follows an exponential rather than linear distribution. And they estimate the parameters in an algorithm using maximum likelihood rather than least squares. The end result is an estimated probability that an observation (batted ball) is in a particular class (single or out) given (conditioned on) the values of one or more predictors.

The generalized linear model often used to predict a binary (0 or 1) outcome variable from a set of numeric variables is logistic regression. It assumes the outcomes (in this context, a probability distribution) follow a

binomial distribution, and it transforms the conditional probabilities into log odds to ensure the probability estimates are always between 0 and 1. The log odds represent how much more likely something will happen than not happen. Thus, probabilities will be symmetric around 0. The accuracy of the model's predictions depend on how close its probability estimates are to the actual probabilities.

So we proceeded to fit a logistic regression model to the standardized data in *train_set*.

```
# Confirm which level of train_set$events has been assigned to 0.

contrasts(as.factor(train_set$events))

##           field_out
## single          0
## field_out       1

# Fit a model to train_set$events using logistic regression.

fit_logit <- train_set %>% mutate(y = as.numeric(events == "field_out")) %>%
  glm(y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp + s_spray_angle_adj +
    s_hp_to_1b + s_if_alignment, data = ., family = "binomial")

# Make sure each regression coefficient is statistically significant (p < .05).

summary(fit_logit)

## 
## Call:
## glm(formula = y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
##       s_spray_angle_adj + s_hp_to_1b + s_if_alignment, family = "binomial",
##       data = .)
## 
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max 
## -2.5845  0.3907  0.6779  0.7933  1.3146 
## 
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) 1.156180  0.008492 136.147 < 2e-16 ***
## s_launch_angle 0.273571  0.009474  28.875 < 2e-16 ***
## s_launch_speed -0.356507  0.009007 -39.580 < 2e-16 ***
## s_spray_angle_Kolp 0.095177  0.008485  11.217 < 2e-16 ***
## s_spray_angle_adj -0.173628  0.009065 -19.154 < 2e-16 ***
## s_hp_to_1b      0.054310  0.008409   6.458 1.06e-10 ***
## s_if_alignment   0.015170  0.008447   1.796   0.0725 .  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
## Null deviance: 90063  on 80465  degrees of freedom
## Residual deviance: 87625  on 80459  degrees of freedom
## AIC: 87639
## 
## Number of Fisher Scoring iterations: 4
```

```

exp(coef(fit_logit))

##          (Intercept)      s_launch_angle      s_launch_speed
## 3.1777707           1.3146512           0.7001173
## s_spray_angle_Kolp  s_spray_angle_adj    s_hp_to_1b
## 1.0998539           0.8406098           1.0558123
##   s_if_alignment
## 1.0152856

```

```

exp(confint(fit_logit))

##                  2.5 %    97.5 %
## (Intercept) 3.1254016 3.2311942
## s_launch_angle 1.2904876 1.3393161
## s_launch_speed 0.6878422 0.7125628
## s_spray_angle_Kolp 1.0817216 1.1183050
## s_spray_angle_adj 0.8258031 0.8556751
## s_hp_to_1b 1.0385685 1.0733751
## s_if_alignment 0.9986340 1.0322562

```

From the p-values for the regression coefficients, we can see that all of our predictors made a significant contribution to the model except for Infield Alignment. This is no surprise since we knew that Infield Alignment was useful only as a moderator of Spray Angle. By itself, it didn't have much to say about whether a batted ball would result in a single or an out.

The function for the exponential distribution (`exp()`) makes the regression coefficients easier to interpret by converting the log odds back to the odds scale. Now we can see, for example, that the odds of a single are increased by a factor of 1.315 for a one unit increase in *launch_angle*, assuming the other predictors remain constant. The confidence interval function (`confint()`) generates the confidence intervals of the coefficients.

We then used our logistic regression model to estimate the probability of a single for each batted ball in *test_set*.

```

# Using our logistic regression model (fit_logit), estimate the probability
# of a single for each observation (batted ball) in test_set.

p_hat_logit <- predict(fit_logit, newdata = test_set, type = "response")
test_set <- add_column(test_set, p_hat_logit)

```

We next needed to apply a decision rule to the probabilities so that the model could actually predict whether each batted ball in the *test_set* will result in a single. This first required determining the optimal cutoff (probability) that best differentiated between singles and outs. First, we defined “optimal cutoff” as the probability that maximizes the truescore.

```

# Determine the cutoff in the decision rule by identifying the cutoff (probability)
# that maximizes the truescore.

cutoffs <- test_set$p_hat_logit
truescores <- map_dbl(cutoffs, function(x){
  truescore_preds <- ifelse(test_set$p_hat_logit >= x, "single", "field_out") %>%
    factor(levels = levels(test_set$events))
  truescore_tprs <- round(sensitivity(truescore_preds,

```

```

                    reference = factor(test_set$events)), 6)
truescore_tnrs <- round(specificity(truescore_preds,
                                      reference = factor(test_set$events)), 6)
truescores <- round((2 * truescore_tprs * truescore_tnrs) /
                      (truescore_tprs + truescore_tnrs) ,6)
})
test_set <- mutate(test_set, truescores = truescores)
(max_truescore <- max(truescores))

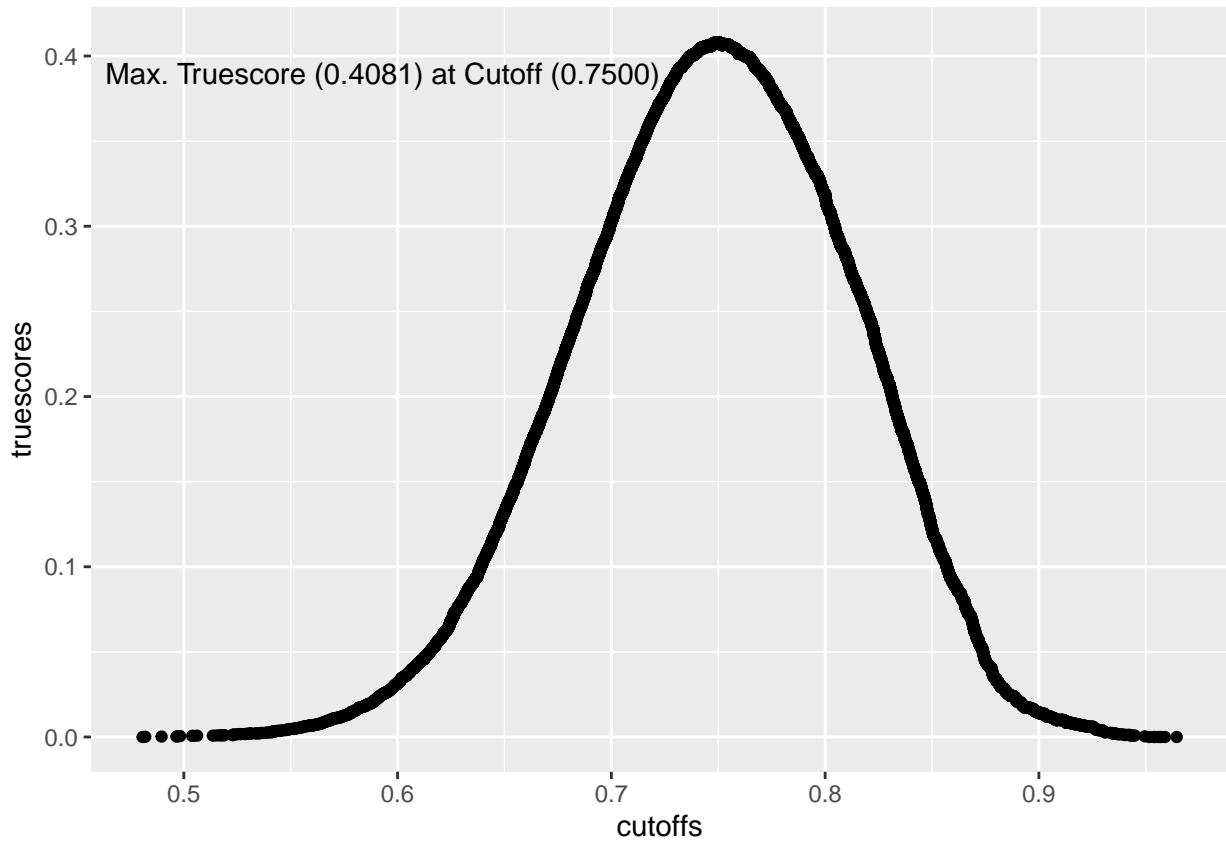
## [1] 0.408139

(max_truescore_cutoff <- round(cutoffs[which.max(truescores)], 6))

## [1] 0.749984

qplot(x = cutoffs, y = truescores) + annotate("text", x = 0.593, y = 0.39,
                                              label = "Max. Truescore (0.4081) at Cutoff (0.7500)")

```



Now we could apply the decision rule based on the cutoff that maximized the truescore. Once we obtained the model's predictions against the *test_set*, we could compute the model's TPR, TNR, and FPR.

```

# Apply a decision rule based on the cutoff that maximizes the truescore.

max_truescore_preds <- ifelse(p_hat_logit >= max_truescore_cutoff, "single",

```

```

    "field_out") %>%
factor(levels = levels(test_set$events))
test_set <- add_column(test_set, max_truescore_preds)
(max_truescore_tpr <- round(sensitivity(max_truescore_preds,
                                         reference = factor(test_set$events)), 6))

## [1] 0.383256

(max_truescore_tnr <- round(specificity(max_truescore_preds,
                                         reference = factor(test_set$events)), 6))

## [1] 0.436216

(max_truescore_fpr <- round(1 - max_truescore_tnr, 6))

## [1] 0.563784

```

Next, we used our minimum distance method to determine the optimal cutoff point, used that cutoff point to make predictions against *test_set*, and computed the model's TPR, TNR, and FPR.

```

# Determine the cutoff in the decision rule by identifying the probability that
# minimizes the distance from the model's ROC curve to the coordinate (0, 1).

pred <- prediction(test_set$p_hat_logit, test_set$events, label.ordering = c("field_out",
                                                                           "single"))

perf <- performance(pred, measure="tpr", x.measure="fpr")
rocr_perf_elements <- tibble(cutoffs = perf@alpha.values[[1]],
                             FPR = round(perf@x.values[[1]], 6),
                             TPR = round(perf@y.values[[1]], 6))
rocr_perf_elements <- rocr_perf_elements %>% mutate(distance =
                                                       sqrt((1 - TPR)^2 + (FPR)^2))

(min_distance <- round(min(rocr_perf_elements$distance), 6))

## [1] 0.831776

(min_distance_cutoff <-
  round(rocr_perf_elements$cutoffs[which.min(rocr_perf_elements$distance)], 6))

## [1] 0.764628

(min_distance_tpr <-
  round(rocr_perf_elements$TPR[which.min(rocr_perf_elements$distance)], 6))

## [1] 0.328448

(min_distance_fpr <-
  round(rocr_perf_elements$FPR[which.min(rocr_perf_elements$distance)], 6))

## [1] 0.490784

```

```

(min_distance_tnr <- round(1 - min_distance_fpr, 6))

## [1] 0.509216

# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(min_distance_truescore <- round((2 * min_distance_tpr * min_distance_tnr) /
                                    (min_distance_tpr + min_distance_tnr), 6))

## [1] 0.399327

# Compute the distance of the optimal cutoff point identified by maximizing the truescore.

(max_truescore_distance <-
  round(sqrt((1 - max_truescore_tpr)^2 + (max_truescore_fpr)^2), 6))

## [1] 0.835599

```

A plot of the ROC curve and the two decision points generated by our cutoff methods helped us visualize the difference between the two approaches. The other labeled decision points are for reference only.

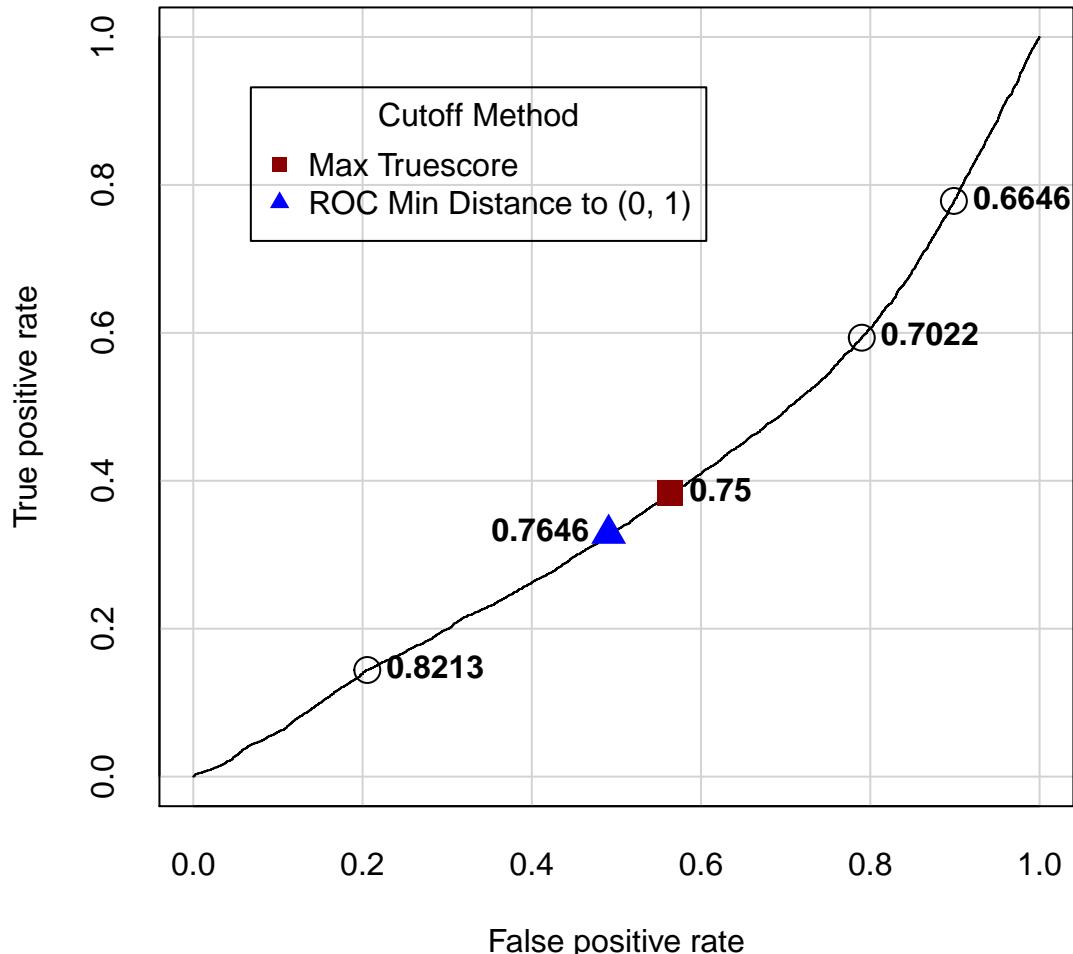
```

# Create a ROC curve visualizing the optimal cutoff points identified
# by our two methods: Max Truescore and Min Distance.

plot(perf, main = "ROC Curve Showing Single-Out TPR-FPR Trade-Offs
  at Max Truescore Cutoff and Min Distance Cutoff",
      print.cutoffs.at = c(0.749984, 0.764628, 0.8213, 0.7022, 0.6646),
      cutoff.label.function = function(x) { round(x, 4) },
      points.pch = c(15, 17, 21, 21, 21),
      points.col = c("dark red", "blue", "black", "black", "black"),
      points.cex = c(1.8, 1.8, 1.8, 1.8, 1.8), text.font = c(2, 2, 2, 2, 2),
      text.pos = c(4, 2, 4, 4, 4),
      cutoff.label.text = c("Max Truescore", "Min Distance", "ref", "ref", "ref"),
      xaxis.tck = 1, yaxis.tck = 1, xaxis.col.tick = "light gray",
      yaxis.col.tick = "light gray")
legend("topleft", inset = 0.1, title = "Cutoff Method",
       legend = c("Max Truescore", "ROC Min Distance to (0, 1)"),
       pch = c(15, 17), col = c("dark red", "blue"))

```

ROC Curve Showing Single-Out TPR–FPR Trade–Offs at Max Truescore Cutoff and Min Distance Cutoff



For purposes of comparing our two logistic regression models, we computed the truescore of the Minimum Distance Model, and the distance of the Maximum Truescore Model.

```
# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(min_distance_truescore <- round((2 * min_distance_tpr * min_distance_tnr) /
                                (min_distance_tpr + min_distance_tnr), 6))

## [1] 0.399327

# Compute the distance of the optimal cutoff point identified by maximizing the truescore.

(max_truescore_distance <-
  round(sqrt((1 - max_truescore_tpr)^2 + (max_truescore_fpr)^2), 6))

## [1] 0.835599
```

Finally, we created a table displaying the assessment results obtained for the models tested thus far.

```
# Create a table displaying the assessment results obtained thus far.

assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                         "Log. Regression"),
                               `Cutoff Method` =
                                   c(NA, "Max Truescore", "Min Distance"),
                               `Truescore` =
                                   c(bl_truescore, max_truescore, min_distance_truescore),
                               TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr),
                               TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr),
                               Distance =
                                   c(NA, max_truescore_distance, min_distance),
                               FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr),
                               `Best Cutoff` =
                                   c(NA, max_truescore_cutoff, min_distance_cutoff))
knitr::kable(assessment_results[1:3, ],
             caption = "Assessment Results - First Logistic Regression Model")
```

Table 1: Assessment Results - First Logistic Regression Model

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Max Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628

Shockingly, our two logistic regression models performed only marginally better than our guessing algorithm. What to do? The function for fitting generalized linear models allows a user to specify an interactive term by inserting a colon between two interactive variables in the formula argument. So here was an opportunity to see if including infield alignment in our model contributed to the model's predictive ability.

```
# Fit a refined logistic regression model (r_fit_logit) with interaction terms to
# train_set$events.

r_fit_logit <- train_set %>% mutate(y = as.numeric(events == "field_out")) %>%
  glm(y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
    s_spray_angle_adj + s_hp_to_1b + s_if_alignment +
    s_spray_angle_Kolp:s_if_alignment + s_spray_angle_adj:s_if_alignment,
    data = ., family = "binomial")

# Make sure each regression coefficient is statistically significant (p < .05).

summary(r_fit_logit)
```

```
## 
## Call:
## glm(formula = y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
##       s_spray_angle_adj + s_hp_to_1b + s_if_alignment + s_spray_angle_Kolp:s_if_alignment +
##       s_spray_angle_adj:s_if_alignment, family = "binomial", data = .)
## 
```

```

## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.6177   0.3812   0.6754   0.7955   1.3129
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)               1.152070  0.008505 135.452 < 2e-16
## s_launch_angle            0.274971  0.009485 28.990 < 2e-16
## s_launch_speed           -0.358486  0.009016 -39.762 < 2e-16
## s_spray_angle_Kolp        0.068066  0.008914  7.636 2.25e-14
## s_spray_angle_adj        -0.158948  0.009455 -16.811 < 2e-16
## s_hp_to_1b                 0.053108  0.008418  6.309 2.81e-10
## s_if_alignment              0.017145  0.008463  2.026  0.0428
## s_spray_angle_Kolp:s_if_alignment 0.040482  0.008953  4.522 6.13e-06
## s_spray_angle_adj:s_if_alignment -0.081680  0.008923 -9.154 < 2e-16
##
## (Intercept)               ***
## s_launch_angle             ***
## s_launch_speed             ***
## s_spray_angle_Kolp         ***
## s_spray_angle_adj          ***
## s_hp_to_1b                  ***
## s_if_alignment                *
## s_spray_angle_Kolp:s_if_alignment ***
## s_spray_angle_adj:s_if_alignment ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 90063  on 80465  degrees of freedom
## Residual deviance: 87508  on 80457  degrees of freedom
## AIC: 87526
##
## Number of Fisher Scoring iterations: 4

```

```
exp(coef(r_fit_logit))
```

##	(Intercept)	s_launch_angle
##	3.1647378	1.3164923
##	s_launch_speed	s_spray_angle_Kolp
##	0.6987334	1.0704363
##	s_spray_angle_adj	s_hp_to_1b
##	0.8530406	1.0545436
##	s_if_alignment	s_spray_angle_Kolp:s_if_alignment
##	1.0172931	1.0413125
##	s_spray_angle_adj:s_if_alignment	
##	0.9215669	

```
exp(confint(r_fit_logit))
```

##	2.5 %	97.5 %
## (Intercept)	3.1125027	3.2180253

```

## s_launch_angle           1.2922674 1.3412206
## s_launch_speed          0.6864713 0.7111658
## s_spray_angle_Kolp      1.0519014 1.0893089
## s_spray_angle_adj       0.8373737 0.8689921
## s_hp_to_1b               1.0373024 1.0721038
## s_if_alignment           1.0005776 1.0343301
## s_spray_angle_Kolp:s_if_alignment 1.0231970 1.0597435
## s_spray_angle_adj:s_if_alignment 0.9055892 0.9378245

```

As can be seen from the return of the summary() function, the p-values of all of our predictors (even infield alignment!) were statistically significant. The interactive terms' exponentiated coefficients appeared modest but not minuscule in impact. In any case, researchers appear to agree that the nonlinear nature of interaction means that "the statistical estimate of an interactive effect cannot be interpreted as straightforward as the coefficient of a regular regression parameter."⁹ In other words, the statistical significance of an interactive term's regression coefficient doesn't tell us much.

Researchers have turned to other techniques to assess the effect of a conditional variable (Infield Alignment) on the contribution of another variable (Spray Angle) to the dependent variable (Event (single or out)). Wondering whether either variety of our logistic regression model would perform better than previously as a result of the interactive term, we decided to use the interplot package to construct a conditional effect plot that would help us visualize the effect of infield alignment on the coefficient for Spray Angle.

```

# Create a conditional effect plot to visualize how infield alignment affects
# the coefficient for s_spray_angle_adj.

```

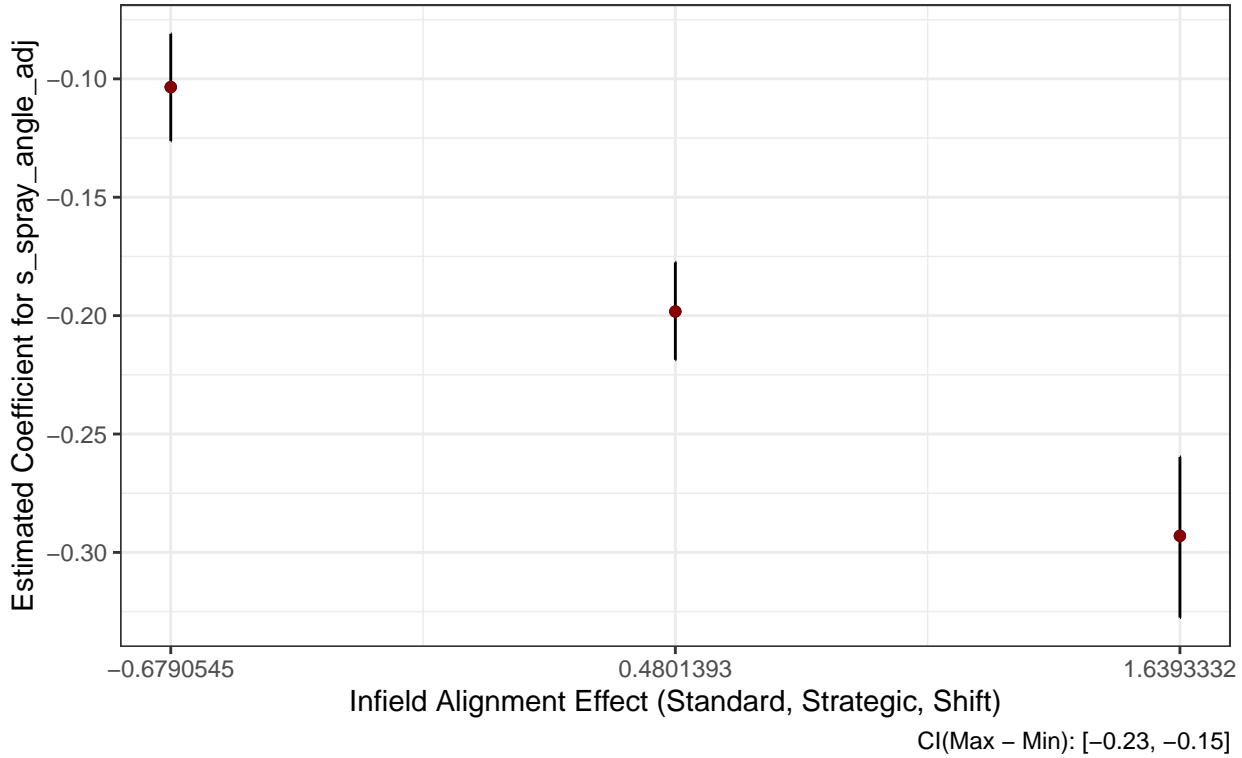
```

interplot(m = r_fit_logit, var1 = "s_spray_angle_adj", var2 = "s_if_alignment") +
  geom_point(color = "dark red") +
  ggtitle("Estimated Coefficient for Spray Angle Effect on Single,
           by Alignment of Infielders") +
  theme(plot.title = element_text(face = "bold")) +
  xlab("Infield Alignment Effect (Standard, Strategic, Shift)") +
  ylab("Estimated Coefficient for s_spray_angle_adj") +
  theme_bw()

```

⁹Solt, Frederick, Hu, Yue, "interplot: Plot the Effects of Variables in Interaction Terms" (2019). <https://cran.r-project.org/web/packages/interplot/vignettes/interplot-vignette.html>

Estimated Coefficient for Spray Angle Effect on Single, by Alignment of Infielders



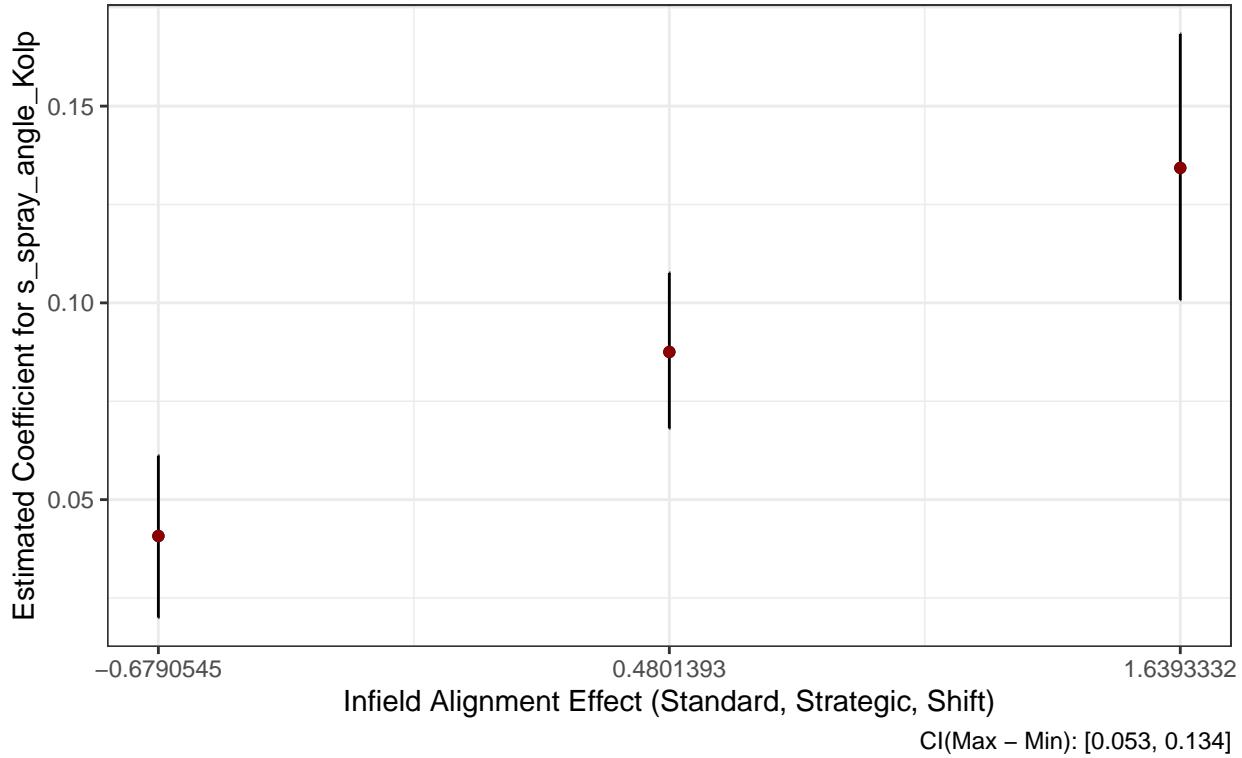
The above conditional effect plot shows how the estimated coefficient for $s_{spray_angle_adj}$ varies depending on the level of the conditional factor $s_{infield_alignment}$. The lines extending from each point represent the 95% confidence interval. In this case, the intervals do not overlap, indicating genuinely distinct coefficients for each level. Moreover, the nature of the effect makes sense. When the infielders are shifted, $s_{spray_angle_adj}$ has a greater impact on the response variable. We saw during data exploration the dramatic density plots of $spray_angle_adj$ differentiating singles from outs when a shift was deployed.

We expected $s_{if_alignment}$ to have a smaller effect on $s_{spray_angle_Kolp}$, but we created a separate conditional effect plot for this moderator as well.

```
# Create a conditional effect plot to visualize how infield alignment affects
# the coefficient for s_spray_angle_Kolp.
```

```
interplot(m = r_fit_logit, var1 = "s_spray_angle_Kolp", var2 = "s_if_alignment") +
  geom_point(color = "dark red") +
  ggtitle("Estimated Coefficient for Spray Angle Effect on Single,
           by Alignment of Infielders") +
  theme(plot.title = element_text(face = "bold")) +
  xlab("Infield Alignment Effect (Standard, Strategic, Shift)") +
  ylab("Estimated Coefficient for s_spray_angle_Kolp") +
  theme_bw()
```

Estimated Coefficient for Spray Angle Effect on Single, by Alignment of Infielders



The plot confirmed our expectations. This time, the estimated coefficients were closer to each other, with confidence intervals overlapping or nearly overlapping. Yet, the plot still depicted an interactive relationship between infield alignment and *spray_angle_Kolp*.

With our hopes buoyed, we proceeded to test the new logistic regression model using *test_set*.

```
# Using our refined logistic regression model (r_fit_logit), estimate the probability
# of a single for each observation (batted ball) in test_set.
```

```
r_p_hat_logit <- predict(r_fit_logit, newdata = test_set, type = "response")
test_set <- add_column(test_set, r_p_hat_logit)
```

Using the same steps we used earlier, we next determined the optimal cutoff (probability) using the Maximum Truescore method. The optimal cutoff was then used to apply a decision rule that predicted whether each batted ball in the *test_set* will result in a single.

```
# Determine the cutoff in the decision rule by identifying the cutoff (probability)
# that maximizes the truescore.
```

```
r_cutoffs <- test_set$r_p_hat_logit
r_truescores <- map_dbl(r_cutoffs, function(x){
  r_truescore_preds <- ifelse(test_set$r_p_hat_logit >= x, "single", "field_out") %>%
    factor(levels = levels(test_set$events))
  r_truescore_tprs <- round(sensitivity(r_truescore_preds,
                                         reference = factor(test_set$events)), 6)
  r_truescore_tnrs <- round(specificity(r_truescore_preds,
```

```

            reference = factor(test_set$events)), 6)
r_truescores <- round((2 * r_truescore_tprs * r_truescore_tnrs) /
(r_truescore_tprs + r_truescore_tnrs) ,6)
})
test_set <- mutate(test_set, r_truescores = r_truescores)
(r_max_truescore <- max(r_truescores))

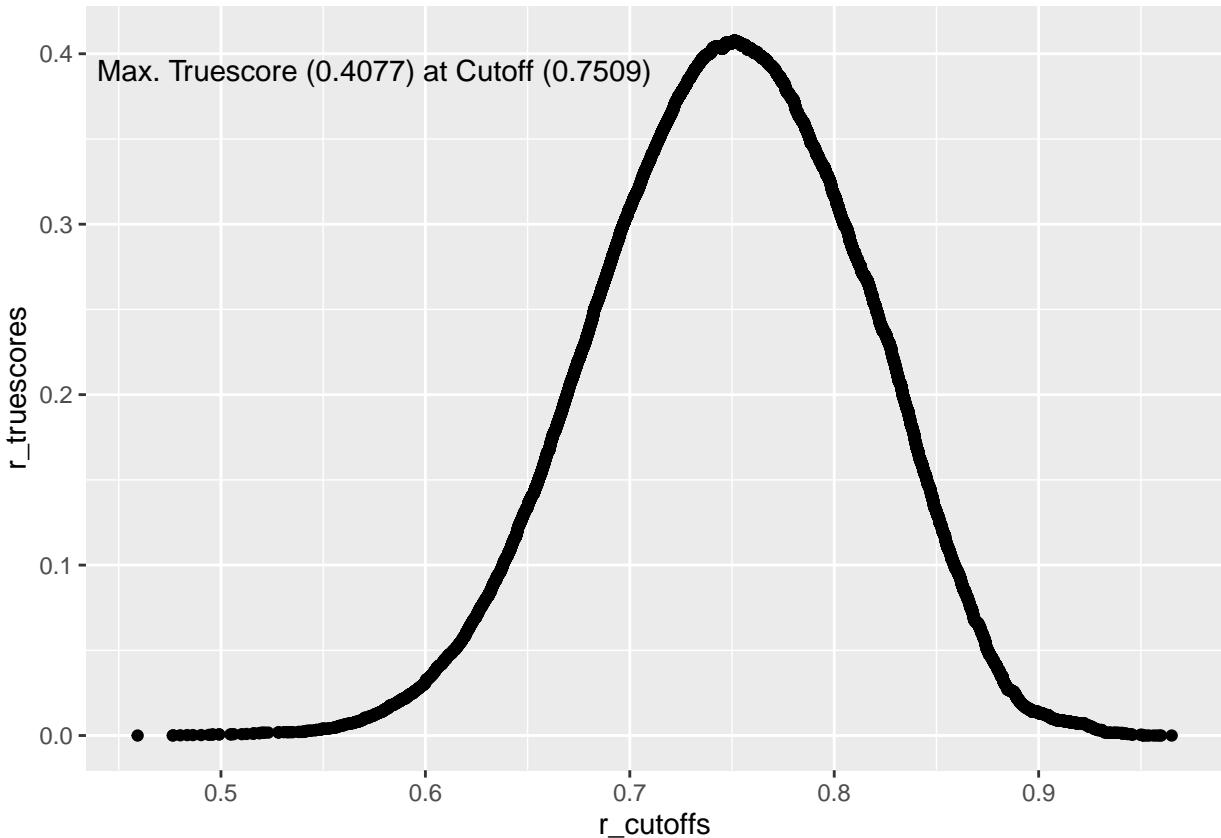
## [1] 0.407656

(r_max_truescore_cutoff <- round(r_cutoffs[which.max(r_truescores)], 6))

## [1] 0.750887

qplot(x = r_cutoffs, y = r_truescores) +
annotate("text", x = 0.575, y = 0.39,
label = "Max. Truescore (0.4077) at Cutoff (0.7509)")

```



```

# Apply a decision rule based on the cutoff that maximizes the truescore.

r_max_truescore_preds <- ifelse(r_p_hat_logit >= r_max_truescore_cutoff, "single",
"field_out") %>%
factor(levels = levels(test_set$events))
test_set <- add_column(test_set, r_max_truescore_preds)
(r_max_truescore_tpr <- round(sensitivity(r_max_truescore_preds,
reference = factor(test_set$events)), 6))

```

```

## [1] 0.376832

(r_max_truescore_tnr <- round(specificity(r_max_truescore_preds,
                                             reference = factor(test_set$events)), 6))

```

```

## [1] 0.443417

(r_max_truescore_fpr <- round(1 - r_max_truescore_tnr, 6))

```

```

## [1] 0.556583

```

We repeated the same steps again, this time using the Minimum Distance method.

```

# Determine the cutoff in the decision rule by identifying the probability that
# minimizes the distance from the model's ROC curve to the coordinate (0, 1). Then apply
# a decision rule using that probability (cutoff).

r_pred <- prediction(test_set$r_p_hat_logit, test_set$events,
                       label.ordering = c("field_out", "single"))
r_perf <- performance(r_pred, measure="tpr", x.measure="fpr")
r_rocr_perf_elements <- tibble(r_cutoffs = r_perf$alpha.values[[1]],
                                 r_FPR = round(r_perf$x.values[[1]], 6),
                                 r_TPR = round(r_perf$y.values[[1]], 6))
r_rocr_perf_elements <- r_rocr_perf_elements %>%
  mutate(r_distance = sqrt((1 - r_TPR)^2 + (r_FPR)^2))

(r_min_distance <- round(min(r_rocr_perf_elements$r_distance), 6))

```

```

## [1] 0.832194

```

```

(r_min_distance_cutoff <-
  round(r_rocr_perf_elements$r_cutoffs[which.min(r_rocr_perf_elements$r_distance)], 6))

```

```

## [1] 0.766355

```

```

(r_min_distance_tpr <-
  round(r_rocr_perf_elements$r_TPR[which.min(r_rocr_perf_elements$r_distance)], 6))

```

```

## [1] 0.322024

```

```

(r_min_distance_fpr <-
  round(r_rocr_perf_elements$r_FPR[which.min(r_rocr_perf_elements$r_distance)], 6))

```

```

## [1] 0.482592

```

```

(r_min_distance_tnr <- round(1 - r_min_distance_fpr, 6))

```

```

## [1] 0.517408

```

We plotted the new ROC curve and the two decision points generated by our cutoff methods.

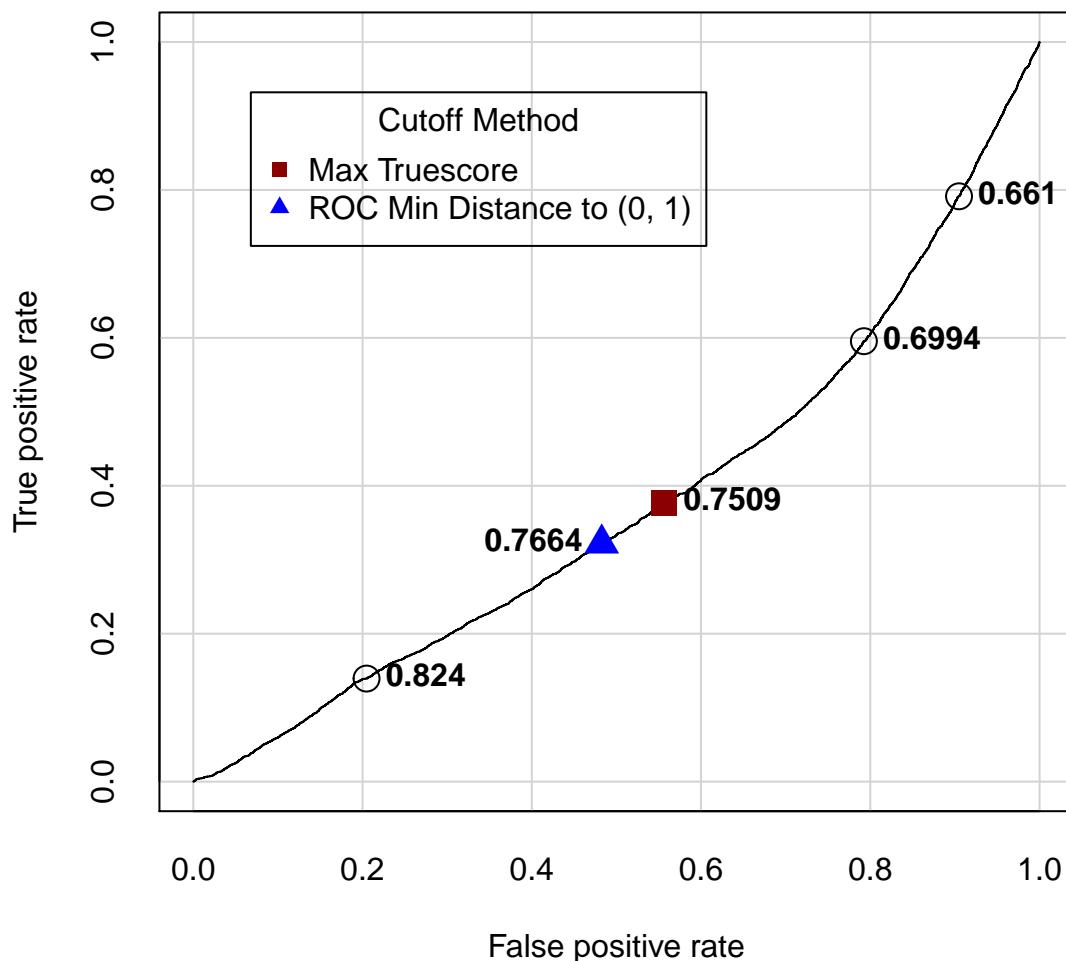
```

# Create a ROC curve allowing us to visualize the optimal cutoff points identified
# by our two methods: Max Truescore and Min Distance.

plot(r_perf, main = "ROC Curve (Refined Model):
Single-Out TPR-FPR Trade-Offs at Max Truescore Cutoff
and Min Distance Cutoff",
print.cutoffs.at = c(0.750887, 0.766355, 0.6610, 0.6994, 0.8240),
cutoff.label.function = function(x) { round(x, 4) },
points.pch = c(15, 17, 21, 21, 21),
points.col = c("dark red", "blue", "black", "black", "black"),
points.cex = c(1.8, 1.8, 1.8, 1.8, 1.8), text.font = c(2, 2, 2, 2, 2),
text.pos = c(4, 2, 4, 4, 4),
cutoff.label.text = c("Max Truescore", "Min Distance", "ref", "ref", "ref"),
xaxis.tck = 1, yaxis.tck = 1, xaxis.col.tick = "light gray",
yaxis.col.tick = "light gray")
legend("topleft", inset = 0.1, title = "Cutoff Method",
legend = c("Max Truescore", "ROC Min Distance to (0, 1)"),
pch = c(15, 17), col = c("dark red", "blue"))

```

ROC Curve (Refined Model): Single-Out TPR–FPR Trade–Offs at Max Truescore Cutoff and Min Distance Cutoff



We again computed the truescore of the Minimum Distance Model, and the distance of the Maximum Truescore Model to facilitate comparison of our two refined logistic regression models.

```
# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(r_min_distance_truescore <- round((2 * r_min_distance_tpr * r_min_distance_tnr) /
                                         (r_min_distance_tpr + r_min_distance_tnr), 6))

## [1] 0.396977

# Compute the distance of the optimal point identified by maximizing the weighted
# truescore.

(r_max_truescore_distance <-
  round(sqrt((1 - r_max_truescore_tpr)^2 + (r_max_truescore_fpr)^2), 6))

## [1] 0.835538
```

The updated table of results for all of our models followed.

```
# Create a table displaying all of the assessment results.

r_assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                         "Log. Regression", "Log. Regression",
                                         "Log. Regression"),
                                 `Cutoff Method` = c(NA, "Max Truescore",
                                         "Min Distance",
                                         "Max Truescore (ref.)",
                                         "Min Distance (ref.)"),
                                 `Truescore` = c(bl_truescore, max_truescore,
                                                 min_distance_truescore, r_max_truescore,
                                                 r_min_distance_truescore),
                                 TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                         r_max_truescore_tpr, r_min_distance_tpr),
                                 TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                         r_max_truescore_tnr, r_min_distance_tnr),
                                 `Distance` = c(NA, max_truescore_distance,
                                               min_distance,
                                               r_max_truescore_distance,
                                               r_min_distance),
                                 FPR = c((1 - bl_tnr), max_truescore_fpr,
                                         min_distance_fpr, r_max_truescore_fpr,
                                         r_min_distance_fpr),
                                 `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                   min_distance_cutoff,
                                                   r_max_truescore_cutoff,
                                                   r_min_distance_cutoff))

knitr::kable(r_assessment_results[1:5, ],
             caption = "Assessment Results - Refined Logistic Regression Model")
```

Table 2: Assessment Results - Refined Logistic Regression Model

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Max Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
Log. Regression	Max Truescore (ref.)	0.407656	0.376832	0.443417	0.835538	0.556583	0.750887
Log. Regression	Min Distance (ref.)	0.396977	0.322024	0.517408	0.832194	0.482592	0.766355

Our so called “refined” logistic regression model with an interactive term actually performed worse than our base logistic regression model without any interactive terms. This was true under both the Maximum Truescore method and the Minimum Distance method of measuring predictive ability. In actuality, the results were quite similar. What did stand out, however, was that Maximum Truescore seemed to strike a better balance between sensitivity and specificity than Minimum Distance.

In the end, the predictive ability of all of our logistic regression models was highly disappointing. It could be that the linear nature of logistic regression was simply not flexible enough to capture the complex relationships between our features and our dependent response variable. We turned to nonparametric algorithms to capture these relationships.

Build and Assess a Model with the K-Nearest Neighbors Algorithm (knn)

The knn algorithm estimates conditional probability by averaging the k nearest points (a neighborhood). The parameter k specifies the size of the neighborhood. A larger k results in a smoother estimate, while a smaller k results in a more flexible estimate.

To quickly see whether knn could potentially provide a better performing model than logistic regression, we first used the default value of k, which is 5. We also used normalized data for knn.

```
# Build a knn model with the default k=5. Prepare the train and test sets,
# define the response vector and matrix of predictors, and fit the model
# using the knn3 function.

train_set2 <- dplyr::select(train_set, -(hit_distance_sc))
test_set2 <- dplyr::select(test_set, -(hit_distance_sc), -(p_hat_logit), -(truescores),
                           -(max_truescore_preds), -(r_p_hat_logit), -(r_truescores),
                           -(r_max_truescore_preds))
x <- as.matrix(train_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                             "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
y <- factor(train_set2$events)
knn_fit_k5 <- knn3(x, y, k = 5)

# Test the knn_fit_k5 model. Define the matrix of new data (test_set) and use it to
# predict the outcomes.

z <- as.matrix(test_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                            "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
knn_y_hat <- predict(knn_fit_k5, z, type = "class")
knn_y_hat_prob <- predict(knn_fit_k5, z, type = "prob")

# Assess the predictions using both the Truescore method and the Minimum Distance method.
# Compare the performance of knn_fit_k5 to that of the logistic regression models.

confusionMatrix(data = knn_y_hat, reference = test_set2$events)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction  single field_out
##   single      3162      1437
##   field_out    1819     13700
##
##                 Accuracy : 0.8382
##                 95% CI : (0.833, 0.8432)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                 Kappa : 0.5541
##
##     Mcnemar's Test P-Value : 2.438e-11
##
##                 Sensitivity : 0.6348
##                 Specificity : 0.9051
##     Pos Pred Value : 0.6875
```

```

##           Neg Pred Value : 0.8828
##           Prevalence : 0.2476
##           Detection Rate : 0.1572
## Detection Prevalence : 0.2286
##           Balanced Accuracy : 0.7699
##
##           'Positive' Class : single
##

knn_k5_tpr <- round(sensitivity(knn_y_hat, reference = factor(test_set2$events)), 6)
knn_k5_tnr <- round(specificity(knn_y_hat, reference = factor(test_set2$events)), 6)
knn_k5_fpr <- round(1 - knn_k5_tnr, 6)
knn_k5_truescore <- round((2 * knn_k5_tpr * knn_k5_tnr) / (knn_k5_tpr + knn_k5_tnr), 6)
knn_k5_distance <- round(sqrt((1 - knn_k5_tpr)^2 + (knn_k5_fpr)^2), 6)

assessment_results <- tibble(Model = c("Baseline", "Log. Regression", "Log. Regression",
                                         "KNN_k5"),
                                `Cutoff Method` = c(NA, "Truescore", "Min Distance", NA),
                                `Truescore` = c(bl_truescore, max_truescore,
                                                min_distance_truescore, knn_k5_truescore),
                                TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                         knn_k5_tpr),
                                TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                         knn_k5_tnr),
                                Distance = c(NA, max_truescore_distance, min_distance,
                                             knn_k5_distance),
                                FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                         knn_k5_fpr),
                                `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                 min_distance_cutoff, NA))
knitr::kable(assessment_results[1:4, ], caption = "Assessment Results")

```

Table 3: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN_k5	NA	0.746224	0.634812	0.905067	0.377326	0.094933	NA

The knn model with k=5 performed well, especially in comparison to the logistic regression models. But could its performance be even better with an optimized value for k? We turned to the caret package because of its ability to perform cross-validation to identify optimal k. In this case, we used 10-fold cross-validation, with 20% of the *train_set* serving as a validation set for cross-validation purposes only. Nineteen different k values, ranging from 3 to 39, were subjected to cross-validation. With each of the 19 different k values being tested 10 times, there was a total of 190 tests, so this took some time to complete.

```

# Use caret::train() to develop a knn algorithm with an optimal value
# for k and the default decision threshold of 0.50. The optimal value
# of k should maximize the truescore of the model.

```

```

train_set2 <- dplyr::select(train_set, -(hit_distance_sc))

```

```

test_set2 <- dplyr::select(test_set, -(hit_distance_sc), -(p_hat_logit), -(truescores),
                           -(max_truescore_preds), -(r_p_hat_logit), -(r_truescores),
                           -(r_max_truescore_preds))

x <- as.matrix(train_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                             "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
y <- train_set2$events

# Use the train function and cross-validation to predict "single"
# or "field_out" for each value of k from 3 to 39, by twos.

set.seed(1)
fitControl <- trainControl(method = "cv", number = 10, p = 0.8, returnData = TRUE,
                            returnResamp = "all", savePredictions = "all",
                            summaryFunction = twoClassSummary, classProbs = TRUE)

knn_train <- train(x, y, method = "knn",
                    tuneGrid = data.frame(k = seq(3, 39, 2)),
                    trControl = fitControl)

```

The `train()` function in the caret package computes the sensitivity and specificity for each of the ten folds (resamples) for each value of k (190 tests in all). Because the folds are properly stratified for cross-validation, we can simply average the sensitivity and specificity scores across the ten folds for each value of k (as opposed to first summing up the true positives, false negatives, true negatives, and false positives across the ten folds and then computing sensitivity and specificity for each value of k). This will allow us to compute the truescore and distance for each value of k, and, in turn, identify the optimal value of k. We first defined optimal k as the value of k with the highest average truescore, and then as the value of k with the shortest average distance to (0, 1).

```

# Extract the average sensitivity and specificity for each value
# of k, compute the corresponding Truescore and Distance, and
# identify the value of k that maximizes the Truescore and
# minimizes the Distance to (0, 1).

knn_train cms <- as_tibble(knn_train$results)
knn_train cms <- knn_train cms %>% dplyr::select(k, Sens, Spec) %>%
  mutate(Truescore = round((2 * Sens * Spec) / (Sens + Spec), 6)) %>%
  mutate(Distance = round(sqrt((1 - Sens)^2 + (1 - Spec)^2), 6))

max(knn_train cms$Truescore)

```

```
## [1] 0.737093
```

```
knn_train cms$k[which.max(knn_train cms$Truescore)]
```

```
## [1] 7
```

```
min(knn_train cms$Distance)
```

```
## [1] 0.390797
```

```
knn_train_cms$k[which.min(knn_train_cms$Distance)]
```

```
## [1] 7
```

In this case, both Maximum Truescore and Minimum Distance agreed that the optimal value of k is 7. Armed with optimal k, we could now go back and fit a new model with k=7, and use that model to predict outcomes based on the data in test_set2.

```
# Fit an optimized knn model (k = 7) to the entire predictor matrix and outcome vector.  
  
knn_fit_k7 <- knn3(x, y, k = 7)  
  
# Apply our latest knn model (knn_fit_k7) to predict  
# outcomes based on the predictors in test_set2.  
  
knn_y_hat_class <- predict(knn_fit_k7, z, type = "class")  
knn_y_hat_class_tbl <- as_tibble(knn_y_hat_class)  
knn_y_hat_class_tbl <- knn_y_hat_class_tbl %>% mutate(preds = value) %>%  
  dplyr::select(preds)
```

We were finally in a position to assess the predictive ability of our k-optimized knn model.

```
# Assess the predictive ability of our latest knn model (knn_fit_k7).  
  
confusionMatrix(data = knn_y_hat_class_tbl$preds, reference = test_set2$events)  
  
## Confusion Matrix and Statistics  
##  
##          Reference  
## Prediction single field_out  
##   single      3166      1328  
##   field_out    1815     13809  
##  
##          Accuracy : 0.8438  
##                  95% CI : (0.8387, 0.8488)  
##      No Information Rate : 0.7524  
##      P-Value [Acc > NIR] : < 2.2e-16  
##  
##          Kappa : 0.5665  
##  
##  Mcnemar's Test P-Value : < 2.2e-16  
##  
##          Sensitivity : 0.6356  
##          Specificity : 0.9123  
##      Pos Pred Value : 0.7045  
##      Neg Pred Value : 0.8838  
##          Prevalence : 0.2476  
##          Detection Rate : 0.1574  
##      Detection Prevalence : 0.2234  
##          Balanced Accuracy : 0.7739  
##  
##          'Positive' Class : single  
##
```

```

knn_k7_tpr <- round(sensitivity(knn_y_hat_class_tbl$preds,
                                    reference = factor(test_set2$events)), 6)
knn_k7_tnr <- round(specificity(knn_y_hat_class_tbl$preds,
                                    reference = factor(test_set2$events)), 6)
knn_k7_fpr <- round(1 - knn_k7_tnr, 6)
knn_k7_truescore <- round((2 * knn_k7_tpr * knn_k7_tnr) / (knn_k7_tpr + knn_k7_tnr), 6)
knn_k7_distance <- round(sqrt((1 - knn_k7_tpr)^2 + (knn_k7_fpr)^2), 6)

assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                         "Log. Regression", "KNN_k5", "KNN_k7"),
                               `Cutoff Method` = c(NA, "Truescore", "Min Distance",
                                                 "default (0.50)", "default (0.50)"),
                               `Truescore` = c(bl_truescore, max_truescore,
                                              min_distance_truescore, knn_k5_truescore,
                                              knn_k7_truescore),
                               TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                       knn_k5_tpr, knn_k7_tpr),
                               TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                       knn_k5_tnr, knn_k7_tnr),
                               Distance = c(NA, max_truescore_distance, min_distance,
                                           knn_k5_distance, knn_k7_distance),
                               FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                       knn_k5_fpr, knn_k7_fpr),
                               `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                min_distance_cutoff, "default", "default"))
knitr::kable(assessment_results[1:5, ], caption = "Assessment Results")

```

Table 4: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN_k5	default (0.50)	0.746224	0.634812	0.905067	0.377326	0.094933	default
KNN_k7	default (0.50)	0.749218	0.635615	0.912268	0.374798	0.087732	default

Using an optimal value of k equal to 7, the predictive ability of our latest knn model was slightly better than that of our first knn model with k equal to 5. We now had a decent truescore of .7492 and a shorter distance to (0, 1) of .3748. But the true negative rate of 0.91 was still significantly better than the true positive rate of 0.64. In other words, our model was much better at predicting outs than singles. The majority class, *field_out*, was being favored by the model because the class sizes were so different. What could be done with the knn algorithm to counter this effect and achieve greater balance between the model's sensitivity and specificity?

Like many standard classification algorithms, knn attempts to maximize the total number of correct predictions. But this approach may not be appropriate when the class sizes are imbalanced. Adjusting the decision threshold is one approach that's been proposed to account for imbalanced class sizes.¹⁰ Indeed, we've already seen this approach applied to our logistic regression algorithms.

In the context of our project, with a neighborhood size of seven and “single” being the positive outcome, a prediction of single required four of the seven nearest observations to have resulted in a single. But what

¹⁰Chen, J. J., Tsai, C. A., Moon, H., Ahn, H. and Chen, C. H. (2006). The Use of Decision Threshold Adjustment in Class Prediction. *SAR & QSAR in Environmental Research*, 17, 337-351. DOI: 10.1080/10659360600787700

if we predicted a single when three or more of the seven closest observations resulted in a single? Would using this lower decision threshold generate predictions with a higher truescore or lower distance to $(0, 1)$? We tested probability cutoffs of 0.25 (corresponding to two votes) and 0.40 (three votes), and compared the results to those obtained from the standard knn algorithm, which used four votes (corresponding to our probability cutoff of 0.55).

```
# Determine the cutoff in the decision rule by identifying the decision
# threshold that maximizes the Truescore, given a value of k equal to 7.

knn_y_hat_prob <- predict(knn_fit_k7, z, type = "prob")
knn_y_hat_prob_tbl <- as_tibble(knn_y_hat_prob)
knn_cutoffs <- c(0.25, 0.40, 0.55)
knn_truescores <- map_dbl(knn_cutoffs, function(x){
  truescore_preds <- ifelse(knn_y_hat_prob_tbl$single > x, "single",
                             "field_out") %>%
    factor(levels = levels(test_set2$events))
  truescore_tprs <- round(sensitivity(truescore_preds,
                                         reference = factor(test_set2$events)), 6)
  truescore_tnrs <- round(specificity(truescore_preds,
                                         reference = factor(test_set2$events)), 6)
  truescores <- round((2 * truescore_tprs * truescore_tnrs) /
    (truescore_tprs + truescore_tnrs), 6)
})
(knn_max_truescore <- max(knn_truescores))

## [1] 0.805345

(knn_max_truescore_cutoff <- round(knn_cutoffs[which.max(knn_truescores)], 6))

## [1] 0.4

knn_decision_thresholds <- tibble(Cutoffs = knn_cutoffs,
                                    Truescore = knn_truescores)
knitr::kable(knn_decision_thresholds[1:3, ], caption = "Decision Thresholds")
```

Table 5: Decision Thresholds

Cutoffs	Truescore
0.25	0.803993
0.40	0.805345
0.55	0.749241

```
# Determine the cutoff in the decision rule by identifying the decision
# threshold that minimizes the distance to the coordinate  $(0, 1)$ .
```

```
knn_distances <- map_dbl(knn_cutoffs, function(x){
  distance_preds <- ifelse(knn_y_hat_prob_tbl$single > x, "single",
                            "field_out") %>%
    factor(levels = levels(test_set2$events))
```

```

distance_tprs <- round(sensitivity(distance_preds,
                                     reference = factor(test_set2$events)), 6)
distance_fprs <- round(1 - specificity(distance_preds,
                                         reference = factor(test_set2$events)), 6)
distances <- round(sqrt((1 - distance_tprs)^2 + (distance_fprs)^2), 6)
})

(knn_min_distance <- round(min(knn_distances), 6))

## [1] 0.277434

(knn_min_distance_cutoff <- round(knn_cutoffs[which.min(knn_distances)], 6))

## [1] 0.4

knn_decision_thresholds <- tibble(Cutoffs = knn_cutoffs,
                                    Truescore = knn_truescores,
                                    Distance = knn_distances)
knitr::kable(knn_decision_thresholds[1:3, ], caption = "Decision Thresholds")

```

Table 6: Decision Thresholds

Cutoffs	Truescore	Distance
0.25	0.803993	0.285748
0.40	0.805345	0.277434
0.55	0.749241	0.374782

The results indeed showed that changing the decision threshold to three votes (0.40) both maximized the truescore and minimized the distance when the value of k is 7. The improvement was significant. It was no surprise then when we observed much less disparity between the sensitivity and specificity of this improved knn model.

```

# Assess the predictive ability of our final knn model (knn_fit_k7 with a
# neighborhood of 7 and a decision threshold of 3 votes to predict a single).

knn_k7c40_cutoff <- knn_min_distance_cutoff
preds_k7c40 <- ifelse(knn_y_hat_prob_tbl$single > knn_k7c40_cutoff, "single", "field_out")
preds_k7c40 <- factor(preds_k7c40, levels = c("single", "field_out"))

confusionMatrix(data = preds_k7c40, reference = factor(test_set2$events))

## Confusion Matrix and Statistics
##
##          Reference
## Prediction  single field_out
##   single      3850      2413
##   field_out    1131     12724
##
##          Accuracy : 0.8238

```

```

##                               95% CI : (0.8185, 0.8291)
##      No Information Rate : 0.7524
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                          Kappa : 0.5648
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##      Sensitivity : 0.7729
##      Specificity : 0.8406
##      Pos Pred Value : 0.6147
##      Neg Pred Value : 0.9184
##      Prevalence : 0.2476
##      Detection Rate : 0.1914
##      Detection Prevalence : 0.3113
##      Balanced Accuracy : 0.8068
##
##      'Positive' Class : single
##


knn_k7c40_tpr <- round(sensitivity(preds_k7c40, reference = factor(test_set2$events)),
                         6)
knn_k7c40_tnr <- round(specificity(preds_k7c40, reference = factor(test_set2$events)),
                         6)
knn_k7c40_fpr <- round(1 - knn_k7c40_tnr, 6)
knn_k7c40_truescore <- round((2 * knn_k7c40_tpr * knn_k7c40_tnr) / (knn_k7c40_tpr +
                                                               knn_k7c40_tnr),
                               6)
knn_k7c40_distance <- round(sqrt((1 - knn_k7c40_tpr)^2 + (knn_k7c40_fpr)^2), 6)

assessment_results <- tibble(Model = c("Baseline", "Log. Regression", "Log. Regression",
                                         "KNN_k5", "KNN_k7", "KNN_k7c40"),
                               `Cutoff Method` = c(NA, "Truescore", "Min Distance",
                                                 "default (0.50)", "default (0.50)",
                                                 "both"),
                               `Truescore` = c(bl_truescore, max_truescore,
                                              min_distance_truescore, knn_k5_truescore,
                                              knn_k7_truescore, knn_k7c40_truescore),
                               TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                       knn_k5_tpr, knn_k7_tpr, knn_k7c40_tpr),
                               TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                       knn_k5_tnr, knn_k7_tnr, knn_k7c40_tnr),
                               Distance = c(NA, max_truescore_distance, min_distance,
                                            knn_k5_distance, knn_k7_distance,
                                            knn_k7c40_distance),
                               FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                       knn_k5_fpr, knn_k7_fpr, knn_k7c40_fpr),
                               `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                min_distance_cutoff, "default",
                                                "default", knn_k7c40_cutoff))
knitr::kable(assessment_results[1:6, ], caption = "Assessment Results")

```

Table 7: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN_k5	default (0.50)	0.746224	0.634812	0.905067	0.377326	0.094933	default
KNN_k7	default (0.50)	0.749218	0.635615	0.912268	0.374798	0.087732	default
KNN_k7c40	both	0.805345	0.772937	0.840589	0.277434	0.159411	0.4

Our project now had new targets: a truescore of 0.805 and a distance of 0.277.

The knn algorithm had performed very well, particularly after we lowered the cutoff in the decision rule to 0.4. But it seemed as if it could be improved even more. With an optimal k value of seven, the standard knn algorithm generated only 15 distinct probabilities: 0, 1, one-seventh (0.143), two-sevenths (0.286), three-sevenths (0.429), four-sevenths (0.571), five-sevenths (0.714), six-sevenths (0.857). Seven additional distinct probabilities were assigned to a handful of observations: 0.125, 0.250, 0.375, 0.500, 0.625, 0.750, and 0.875. Putting aside how the standard knn algorithm could assign fractions of a vote to an observation, we're still left with only 15 possible probabilities. That's not enough to identify an optimal cutoff with any precision. If the truescores peaked at a cutoff somewhere between 0.25 and 0.40 we wouldn't know it.

As it turned out, this problem was solved by the discovery of another problem. We identified the best cutoff, 0.40, based on the assumption that the optimal value of k was 7. We did, in fact, determine from the data that k=7 resulted in the highest truescore, but that determination was made while accepting the knn algorithm's default cutoff of 0.5. Once we became open to the idea that an alternate cutoff might result in a higher truescore, the cutoff became akin to a second parameter, in addition to k. Max Kuhn, author of the caret package, describes in his documentation the process of optimizing the decision cutoff when there are class imbalances. He treats the cutoff as if it's just another parameter to be optimized.¹¹ ¹²

If we think of the cutoff as a second parameter to be optimized, was the approach we took earlier appropriate? That is, was it proper to identify the optimal value of k using the default cutoff of 0.5, and then use only that "optimal" value of k to identify the optimal cutoff? Moreover, was it appropriate to use the test set to identify the optimal cutoff? The answers, of course, were no and no. The right way to tune a model with multiple parameters is to optimize the parameters simultaneously, not one at a time as we did above. That is, every combination of the parameters' values should be tested to identify the combination that best satisfies the assessment criteria.¹³ In our case, we were looking for the combination of k and cutoff that resulted in the highest truescore and the shortest distance to (0, 1).

Kuhn further clarifies the process of optimizing the cutoff along with other parameters.

One of the most common ways to deal with [performance bias against the minority class] is to determine an alternate > probability cut off. . . . But to do this well, another set of data (not the test set) is needed to set the cut off and > the test set is used to validate it. . . . [T]he model can be tuned, using resampling, to determine any model tuning > parameters as well as an appropriate cut off for the probabilities.

¹¹Kuhn, Max, "The caret Package: Optimizing probability thresholds for class imbalances" (2019). <https://topepo.github.io/caret/index.html>

¹²As an aside, some academics insist the process of selecting the cutoff is not a matter of data science, but instead a business decision. We wholeheartedly disagree. The business decision is the selection of criteria for assessing the predictive ability of models (in our case, the Maximum Truescore and the Minimum Distance to (0, 1)). It's a business decision about the relative value of correctly predicting various possible outcomes, and the relative cost of incorrectly predicting those same outcomes. But once those criteria are selected, identifying an optimal decision threshold when class sizes are imbalanced can be a critical step in developing the very best model based on the assessment criteria. And that makes it a matter of data science.

¹³Kuhn, Max, "The caret Package: Customizing the Tuning Process" (2019). <https://topepo.github.io/caret/index.html>

We were now faced with the task of testing every combination of 19 different values of k (ranging from 3 to 39, by twos) and 25 different decision cutoffs (ranging from 0.25 to 0.49) across 10 folds of data. That amounted to 475 different combinations of k and cutoff, each one of which would be tested 10 times using different subsets of our training set. It remained to be seen whether this approach would result in a better performing knn model.

```
# Use caret::train() to develop a knn algorithm with an optimal combination of values
# for k and the decision threshold. The optimal values of k and the decision threshold
# should maximize the truescore of the model.

train_set2 <- dplyr::select(train_set, -(hit_distance_sc))
test_set2 <- dplyr::select(test_set, -(hit_distance_sc), -(p_hat_logit), -(truescores),
                           -(max_truescore_preds), -(r_p_hat_logit), -(r_truescores),
                           -(r_max_truescore_preds))

x <- as.matrix(train_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                             "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
y <- train_set2$events

# Use the train function and cross-validation to compute the probabilities of "single"
# for a range of k's.

set.seed(1)
fitControl <- trainControl(method = "cv", number = 10, p = 0.8, returnData = TRUE,
                            returnResamp = "all", savePredictions = "all",
                            summaryFunction = twoClassSummary, classProbs = TRUE)

knn_train <- train(x, y, method = "knn",
                    tuneGrid = data.frame(k = seq(3, 39, 2)),
                    trControl = fitControl)
```

One nice feature of caret's train() function is that we could use arguments (returnResamp and classProbs) to return the computed probabilities for each value of k across 10 folds of data. This was invaluable because we needed to generate predictions for each of our 475 combinations of k and cutoff, again across 10 folds of data.

```
# Generate predictions by applying a range of thresholds (cutoffs) to the
# probabilities for each value of k.

knn_train_pred_tib <- as_tibble(knn_train$pred)
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred25 = ifelse(knn_train_pred_tib$single > 0.25, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred26 = ifelse(knn_train_pred_tib$single > 0.26, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred27 = ifelse(knn_train_pred_tib$single > 0.27, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred28 = ifelse(knn_train_pred_tib$single > 0.28, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred29 = ifelse(knn_train_pred_tib$single > 0.29, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
  mutate(pred30 = ifelse(knn_train_pred_tib$single > 0.30, "single", "field_out"))
```

```

    mutate(pred31 = ifelse(knn_train_pred_tib$single > 0.31, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred32 = ifelse(knn_train_pred_tib$single > 0.32, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred33 = ifelse(knn_train_pred_tib$single > 0.33, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred34 = ifelse(knn_train_pred_tib$single > 0.34, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred35 = ifelse(knn_train_pred_tib$single > 0.35, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred36 = ifelse(knn_train_pred_tib$single > 0.36, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred37 = ifelse(knn_train_pred_tib$single > 0.37, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred38 = ifelse(knn_train_pred_tib$single > 0.38, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred39 = ifelse(knn_train_pred_tib$single > 0.39, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred40 = ifelse(knn_train_pred_tib$single > 0.40, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred41 = ifelse(knn_train_pred_tib$single > 0.41, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred42 = ifelse(knn_train_pred_tib$single > 0.42, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred43 = ifelse(knn_train_pred_tib$single > 0.43, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred44 = ifelse(knn_train_pred_tib$single > 0.44, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred45 = ifelse(knn_train_pred_tib$single > 0.45, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred46 = ifelse(knn_train_pred_tib$single > 0.46, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred47 = ifelse(knn_train_pred_tib$single > 0.47, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred48 = ifelse(knn_train_pred_tib$single > 0.48, "single", "field_out"))
knn_train_pred_tib <- knn_train_pred_tib %>%
    mutate(pred49 = ifelse(knn_train_pred_tib$single > 0.49, "single", "field_out"))

# Convert all character columns to factors using dplyr package
# (https://gist.github.com/ramhiser/character2factor.r).

knn_train_pred_tib_f <- knn_train_pred_tib %>%
    mutate_if(sapply(knn_train_pred_tib, is.character), as.factor)
sapply(knn_train_pred_tib_f, class)

# Make sure all of the factor outcomes have the same levels in the same order.

knn_train_pred_tib_f$pred25 <- factor(knn_train_pred_tib_f$pred25,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred26 <- factor(knn_train_pred_tib_f$pred26,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred27 <- factor(knn_train_pred_tib_f$pred27,
                                         levels = c("single", "field_out"))

```

```

knn_train_pred_tib_f$pred28 <- factor(knn_train_pred_tib_f$pred28,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred29 <- factor(knn_train_pred_tib_f$pred29,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred30 <- factor(knn_train_pred_tib_f$pred30,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred31 <- factor(knn_train_pred_tib_f$pred31,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred32 <- factor(knn_train_pred_tib_f$pred32,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred33 <- factor(knn_train_pred_tib_f$pred33,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred34 <- factor(knn_train_pred_tib_f$pred34,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred35 <- factor(knn_train_pred_tib_f$pred35,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred36 <- factor(knn_train_pred_tib_f$pred36,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred37 <- factor(knn_train_pred_tib_f$pred37,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred38 <- factor(knn_train_pred_tib_f$pred38,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred39 <- factor(knn_train_pred_tib_f$pred39,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred40 <- factor(knn_train_pred_tib_f$pred40,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred41 <- factor(knn_train_pred_tib_f$pred41,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred42 <- factor(knn_train_pred_tib_f$pred42,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred43 <- factor(knn_train_pred_tib_f$pred43,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred44 <- factor(knn_train_pred_tib_f$pred44,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred45 <- factor(knn_train_pred_tib_f$pred45,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred46 <- factor(knn_train_pred_tib_f$pred46,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred47 <- factor(knn_train_pred_tib_f$pred47,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred48 <- factor(knn_train_pred_tib_f$pred48,
                                         levels = c("single", "field_out"))
knn_train_pred_tib_f$pred49 <- factor(knn_train_pred_tib_f$pred49,
                                         levels = c("single", "field_out"))
sapply(knn_train_pred_tib_f, levels)

# Convert knn_train_pred_tib_f$Resample into a numeric type.

knn_train_pred_tib_f$Resample <- as.numeric(knn_train_pred_tib_f$Resample)

```

To facilitate further analysis, we split all of our predictions in `knn_train_pred_tib_f` into 190 separate tibbles for each value of k and fold (Resample).

```

# Form separate tibbles for each combination of k and fold (Resample).

k3f1 <- knn_train_pred_tib_f %>% filter(Resample == 1, k == 3)
k3f2 <- knn_train_pred_tib_f %>% filter(Resample == 2, k == 3)
k3f3 <- knn_train_pred_tib_f %>% filter(Resample == 3, k == 3)
k3f4 <- knn_train_pred_tib_f %>% filter(Resample == 4, k == 3)
k3f5 <- knn_train_pred_tib_f %>% filter(Resample == 5, k == 3)
k3f6 <- knn_train_pred_tib_f %>% filter(Resample == 6, k == 3)
k3f7 <- knn_train_pred_tib_f %>% filter(Resample == 7, k == 3)
k3f8 <- knn_train_pred_tib_f %>% filter(Resample == 8, k == 3)
k3f9 <- knn_train_pred_tib_f %>% filter(Resample == 9, k == 3)
k3f10 <- knn_train_pred_tib_f %>% filter(Resample == 10, k == 3)

k5f1 <- knn_train_pred_tib_f %>% filter(Resample == 1, k == 5)
k5f2 <- knn_train_pred_tib_f %>% filter(Resample == 2, k == 5)
k5f3 <- knn_train_pred_tib_f %>% filter(Resample == 3, k == 5)
k5f4 <- knn_train_pred_tib_f %>% filter(Resample == 4, k == 5)
k5f5 <- knn_train_pred_tib_f %>% filter(Resample == 5, k == 5)
k5f6 <- knn_train_pred_tib_f %>% filter(Resample == 6, k == 5)
k5f7 <- knn_train_pred_tib_f %>% filter(Resample == 7, k == 5)
k5f8 <- knn_train_pred_tib_f %>% filter(Resample == 8, k == 5)
k5f9 <- knn_train_pred_tib_f %>% filter(Resample == 9, k == 5)
k5f10 <- knn_train_pred_tib_f %>% filter(Resample == 10, k == 5)

k7f1 <- knn_train_pred_tib_f %>% filter(Resample == 1, k == 7)
k7f2 <- knn_train_pred_tib_f %>% filter(Resample == 2, k == 7)
k7f3 <- knn_train_pred_tib_f %>% filter(Resample == 3, k == 7)
k7f4 <- knn_train_pred_tib_f %>% filter(Resample == 4, k == 7)
k7f5 <- knn_train_pred_tib_f %>% filter(Resample == 5, k == 7)
k7f6 <- knn_train_pred_tib_f %>% filter(Resample == 6, k == 7)
k7f7 <- knn_train_pred_tib_f %>% filter(Resample == 7, k == 7)
k7f8 <- knn_train_pred_tib_f %>% filter(Resample == 8, k == 7)
k7f9 <- knn_train_pred_tib_f %>% filter(Resample == 9, k == 7)
k7f10 <- knn_train_pred_tib_f %>% filter(Resample == 10, k == 7)

k9f1 <- knn_train_pred_tib_f %>% filter(Resample == 1, k == 9)
k9f2 <- knn_train_pred_tib_f %>% filter(Resample == 2, k == 9)
k9f3 <- knn_train_pred_tib_f %>% filter(Resample == 3, k == 9)
k9f4 <- knn_train_pred_tib_f %>% filter(Resample == 4, k == 9)
k9f5 <- knn_train_pred_tib_f %>% filter(Resample == 5, k == 9)
k9f6 <- knn_train_pred_tib_f %>% filter(Resample == 6, k == 9)
k9f7 <- knn_train_pred_tib_f %>% filter(Resample == 7, k == 9)
k9f8 <- knn_train_pred_tib_f %>% filter(Resample == 8, k == 9)
k9f9 <- knn_train_pred_tib_f %>% filter(Resample == 9, k == 9)
k9f10 <- knn_train_pred_tib_f %>% filter(Resample == 10, k == 9)

k11f1 <- knn_train_pred_tib_f %>% filter(Resample == 1, k == 11)
k11f2 <- knn_train_pred_tib_f %>% filter(Resample == 2, k == 11)
k11f3 <- knn_train_pred_tib_f %>% filter(Resample == 3, k == 11)
k11f4 <- knn_train_pred_tib_f %>% filter(Resample == 4, k == 11)
k11f5 <- knn_train_pred_tib_f %>% filter(Resample == 5, k == 11)
k11f6 <- knn_train_pred_tib_f %>% filter(Resample == 6, k == 11)
k11f7 <- knn_train_pred_tib_f %>% filter(Resample == 7, k == 11)

```


Form a list and vector of these tibbles.

```

fk_dfs_1 <- list(k3f1, k3f2, k3f3, k3f4, k3f5, k3f6, k3f7, k3f8, k3f9, k3f10,
                  k5f1, k5f2, k5f3, k5f4, k5f5, k5f6, k5f7, k5f8, k5f9, k5f10,
                  k7f1, k7f2, k7f3, k7f4, k7f5, k7f6, k7f7, k7f8, k7f9, k7f10,
                  k9f1, k9f2, k9f3, k9f4, k9f5, k9f6, k9f7, k9f8, k9f9, k9f10,
                  k11f1, k11f2, k11f3, k11f4, k11f5, k11f6, k11f7, k11f8, k11f9, k11f10,
                  k13f1, k13f2, k13f3, k13f4, k13f5, k13f6, k13f7, k13f8, k13f9, k13f10,
                  k15f1, k15f2, k15f3, k15f4, k15f5, k15f6, k15f7, k15f8, k15f9, k15f10,
                  k17f1, k17f2, k17f3, k17f4, k17f5, k17f6, k17f7, k17f8, k17f9, k17f10,
                  k19f1, k19f2, k19f3, k19f4, k19f5, k19f6, k19f7, k19f8, k19f9, k19f10,
                  k21f1, k21f2, k21f3, k21f4, k21f5, k21f6, k21f7, k21f8, k21f9, k21f10,
                  k23f1, k23f2, k23f3, k23f4, k23f5, k23f6, k23f7, k23f8, k23f9, k23f10,
                  k25f1, k25f2, k25f3, k25f4, k25f5, k25f6, k25f7, k25f8, k25f9, k25f10,
                  k27f1, k27f2, k27f3, k27f4, k27f5, k27f6, k27f7, k27f8, k27f9, k27f10,
                  k29f1, k29f2, k29f3, k29f4, k29f5, k29f6, k29f7, k29f8, k29f9, k29f10,
                  k31f1, k31f2, k31f3, k31f4, k31f5, k31f6, k31f7, k31f8, k31f9, k31f10,
                  k33f1, k33f2, k33f3, k33f4, k33f5, k33f6, k33f7, k33f8, k33f9, k33f10,
                  k35f1, k35f2, k35f3, k35f4, k35f5, k35f6, k35f7, k35f8, k35f9, k35f10,
                  k37f1, k37f2, k37f3, k37f4, k37f5, k37f6, k37f7, k37f8, k37f9, k37f10,
                  k39f1, k39f2, k39f3, k39f4, k39f5, k39f6, k39f7, k39f8, k39f9, k39f10)

fk_dfs_v <- c("k3f1", "k3f2", "k3f3", "k3f4", "k3f5", "k3f6", "k3f7", "k3f8", "k3f9",
              "k3f10", "k5f1", "k5f2", "k5f3", "k5f4", "k5f5", "k5f6", "k5f7", "k5f8",
              "k5f9", "k5f10", "k7f1", "k7f2", "k7f3", "k7f4", "k7f5", "k7f6", "k7f7",
              "k7f8", "k7f9", "k7f10", "k9f1", "k9f2", "k9f3", "k9f4", "k9f5", "k9f6",
              "k9f7", "k9f8", "k9f9", "k9f10", "k11f1", "k11f2", "k11f3", "k11f4",
              "k11f5", "k11f6", "k11f7", "k11f8", "k11f9", "k11f10", "k13f1", "k13f2",
              "k13f3", "k13f4", "k13f5", "k13f6", "k13f7", "k13f8", "k13f9", "k13f10",
              "k15f1", "k15f2", "k15f3", "k15f4", "k15f5", "k15f6", "k15f7", "k15f8",
              "k15f9", "k15f10", "k17f1", "k17f2", "k17f3", "k17f4", "k17f5", "k17f6",
              "k17f7", "k17f8", "k17f9", "k17f10", "k19f1", "k19f2", "k19f3", "k19f4",
              "k19f5", "k19f6", "k19f7", "k19f8", "k19f9", "k19f10", "k21f1", "k21f2",
              "k21f3", "k21f4", "k21f5", "k21f6", "k21f7", "k21f8", "k21f9", "k21f10",
              "k23f1", "k23f2", "k23f3", "k23f4", "k23f5", "k23f6", "k23f7", "k23f8",
              "k23f9", "k23f10", "k25f1", "k25f2", "k25f3", "k25f4", "k25f5", "k25f6",
              "k25f7", "k25f8", "k25f9", "k25f10", "k27f1", "k27f2", "k27f3", "k27f4",
              "k27f5", "k27f6", "k27f7", "k27f8", "k27f9", "k27f10", "k29f1", "k29f2",
              "k29f3", "k29f4", "k29f5", "k29f6", "k29f7", "k29f8", "k29f9", "k29f10",
              "k31f1", "k31f2", "k31f3", "k31f4", "k31f5", "k31f6", "k31f7", "k31f8",
              "k31f9", "k31f10", "k33f1", "k33f2", "k33f3", "k33f4", "k33f5", "k33f6",
              "k33f7", "k33f8", "k33f9", "k33f10", "k35f1", "k35f2", "k35f3", "k35f4",
              "k35f5", "k35f6", "k35f7", "k35f8", "k35f9", "k35f10", "k37f1", "k37f2",
              "k37f3", "k37f4", "k37f5", "k37f6", "k37f7", "k37f8", "k37f9", "k37f10",
              "k39f1", "k39f2", "k39f3", "k39f4", "k39f5", "k39f6", "k39f7", "k39f8",
              "k39f9", "k39f10")

```

For each combination of k and cutoff, we could now sum the true positives, false negatives, true negatives, and false positives across 10 folds, and compute the sensitivity (true positive rate or TPR), specificity (true negative rate or TNR), and truescore.

```

#####
# 0.25 Cutoff
#####

```

```

# For the decision cutoff of 0.25, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c25 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred25, obs))
  confusionMatrix(ss$pred25, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c25) <- fk_dfs_v

cm_c25_tables <- sapply(cm_c25, "[[", 2)
cm_c25_tables <- as_tibble(cm_c25_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.25.

k3c25_tpr <- round(sum(cm_c25_tables[1, 1:10]) /
  (sum(cm_c25_tables[1, 1:10]) +
   sum(cm_c25_tables[2, 1:10])), 6)
k3c25_tnr <- round(sum(cm_c25_tables[4, 1:10]) /
  (sum(cm_c25_tables[4, 1:10]) +
   sum(cm_c25_tables[3, 1:10])), 6)
k3c25_truescore <- round((2 * k3c25_tpr * k3c25_tnr) /
  (k3c25_tpr + k3c25_tnr), 6)

k5c25_tpr <- round(sum(cm_c25_tables[1, 11:20]) /
  (sum(cm_c25_tables[1, 11:20]) +
   sum(cm_c25_tables[2, 11:20])), 6)
k5c25_tnr <- round(sum(cm_c25_tables[4, 11:20]) /
  (sum(cm_c25_tables[4, 11:20]) +
   sum(cm_c25_tables[3, 11:20])), 6)
k5c25_truescore <- round((2 * k5c25_tpr * k5c25_tnr) /
  (k5c25_tpr + k5c25_tnr), 6)

k7c25_tpr <- round(sum(cm_c25_tables[1, 21:30]) /
  (sum(cm_c25_tables[1, 21:30]) +
   sum(cm_c25_tables[2, 21:30])), 6)
k7c25_tnr <- round(sum(cm_c25_tables[4, 21:30]) /
  (sum(cm_c25_tables[4, 21:30]) +
   sum(cm_c25_tables[3, 21:30])), 6)
k7c25_truescore <- round((2 * k7c25_tpr * k7c25_tnr) /
  (k7c25_tpr + k7c25_tnr), 6)

k9c25_tpr <- round(sum(cm_c25_tables[1, 31:40]) /
  (sum(cm_c25_tables[1, 31:40]) +
   sum(cm_c25_tables[2, 31:40])), 6)
k9c25_tnr <- round(sum(cm_c25_tables[4, 31:40]) /
  (sum(cm_c25_tables[4, 31:40]) +
   sum(cm_c25_tables[3, 31:40])), 6)
k9c25_truescore <- round((2 * k9c25_tpr * k9c25_tnr) /
  (k9c25_tpr + k9c25_tnr), 6)

```

```

k11c25_tpr <- round(sum(cm_c25_tables[1, 41:50]) /
                      (sum(cm_c25_tables[1, 41:50]) +
                       sum(cm_c25_tables[2, 41:50])), 6)
k11c25_tnr <- round(sum(cm_c25_tables[4, 41:50]) /
                      (sum(cm_c25_tables[4, 41:50]) +
                       sum(cm_c25_tables[3, 41:50])), 6)
k11c25_truescore <- round((2 * k11c25_tpr * k11c25_tnr) /
                             (k11c25_tpr + k11c25_tnr), 6)

k13c25_tpr <- round(sum(cm_c25_tables[1, 51:60]) /
                      (sum(cm_c25_tables[1, 51:60]) +
                       sum(cm_c25_tables[2, 51:60])), 6)
k13c25_tnr <- round(sum(cm_c25_tables[4, 51:60]) /
                      (sum(cm_c25_tables[4, 51:60]) +
                       sum(cm_c25_tables[3, 51:60])), 6)
k13c25_truescore <- round((2 * k13c25_tpr * k13c25_tnr) /
                             (k13c25_tpr + k13c25_tnr), 6)

k15c25_tpr <- round(sum(cm_c25_tables[1, 61:70]) /
                      (sum(cm_c25_tables[1, 61:70]) +
                       sum(cm_c25_tables[2, 61:70])), 6)
k15c25_tnr <- round(sum(cm_c25_tables[4, 61:70]) /
                      (sum(cm_c25_tables[4, 61:70]) +
                       sum(cm_c25_tables[3, 61:70])), 6)
k15c25_truescore <- round((2 * k15c25_tpr * k15c25_tnr) /
                             (k15c25_tpr + k15c25_tnr), 6)

k17c25_tpr <- round(sum(cm_c25_tables[1, 71:80]) /
                      (sum(cm_c25_tables[1, 71:80]) +
                       sum(cm_c25_tables[2, 71:80])), 6)
k17c25_tnr <- round(sum(cm_c25_tables[4, 71:80]) /
                      (sum(cm_c25_tables[4, 71:80]) +
                       sum(cm_c25_tables[3, 71:80])), 6)
k17c25_truescore <- round((2 * k17c25_tpr * k17c25_tnr) /
                             (k17c25_tpr + k17c25_tnr), 6)

k19c25_tpr <- round(sum(cm_c25_tables[1, 81:90]) /
                      (sum(cm_c25_tables[1, 81:90]) +
                       sum(cm_c25_tables[2, 81:90])), 6)
k19c25_tnr <- round(sum(cm_c25_tables[4, 81:90]) /
                      (sum(cm_c25_tables[4, 81:90]) +
                       sum(cm_c25_tables[3, 81:90])), 6)
k19c25_truescore <- round((2 * k19c25_tpr * k19c25_tnr) /
                             (k19c25_tpr + k19c25_tnr), 6)

k21c25_tpr <- round(sum(cm_c25_tables[1, 91:100]) /
                      (sum(cm_c25_tables[1, 91:100]) +
                       sum(cm_c25_tables[2, 91:100])), 6)
k21c25_tnr <- round(sum(cm_c25_tables[4, 91:100]) /
                      (sum(cm_c25_tables[4, 91:100]) +
                       sum(cm_c25_tables[3, 91:100])), 6)
k21c25_truescore <- round((2 * k21c25_tpr * k21c25_tnr) /
                             (k21c25_tpr + k21c25_tnr), 6)

```

```

k23c25_tpr <- round(sum(cm_c25_tables[1, 101:110]) /
                      (sum(cm_c25_tables[1, 101:110]) +
                       sum(cm_c25_tables[2, 101:110])), 6)
k23c25_tnr <- round(sum(cm_c25_tables[4, 101:110]) /
                      (sum(cm_c25_tables[4, 101:110]) +
                       sum(cm_c25_tables[3, 101:110])), 6)
k23c25_truescore <- round((2 * k23c25_tpr * k23c25_tnr) /
                             (k23c25_tpr + k23c25_tnr), 6)

k25c25_tpr <- round(sum(cm_c25_tables[1, 111:120]) /
                      (sum(cm_c25_tables[1, 111:120]) +
                       sum(cm_c25_tables[2, 111:120])), 6)
k25c25_tnr <- round(sum(cm_c25_tables[4, 111:120]) /
                      (sum(cm_c25_tables[4, 111:120]) +
                       sum(cm_c25_tables[3, 111:120])), 6)
k25c25_truescore <- round((2 * k25c25_tpr * k25c25_tnr) /
                             (k25c25_tpr + k25c25_tnr), 6)

k27c25_tpr <- round(sum(cm_c25_tables[1, 121:130]) /
                      (sum(cm_c25_tables[1, 121:130]) +
                       sum(cm_c25_tables[2, 121:130])), 6)
k27c25_tnr <- round(sum(cm_c25_tables[4, 121:130]) /
                      (sum(cm_c25_tables[4, 121:130]) +
                       sum(cm_c25_tables[3, 121:130])), 6)
k27c25_truescore <- round((2 * k27c25_tpr * k27c25_tnr) /
                             (k27c25_tpr + k27c25_tnr), 6)

k29c25_tpr <- round(sum(cm_c25_tables[1, 131:140]) /
                      (sum(cm_c25_tables[1, 131:140]) +
                       sum(cm_c25_tables[2, 131:140])), 6)
k29c25_tnr <- round(sum(cm_c25_tables[4, 131:140]) /
                      (sum(cm_c25_tables[4, 131:140]) +
                       sum(cm_c25_tables[3, 131:140])), 6)
k29c25_truescore <- round((2 * k29c25_tpr * k29c25_tnr) /
                             (k29c25_tpr + k29c25_tnr), 6)

k31c25_tpr <- round(sum(cm_c25_tables[1, 141:150]) /
                      (sum(cm_c25_tables[1, 141:150]) +
                       sum(cm_c25_tables[2, 141:150])), 6)
k31c25_tnr <- round(sum(cm_c25_tables[4, 141:150]) /
                      (sum(cm_c25_tables[4, 141:150]) +
                       sum(cm_c25_tables[3, 141:150])), 6)
k31c25_truescore <- round((2 * k31c25_tpr * k31c25_tnr) /
                             (k31c25_tpr + k31c25_tnr), 6)

k33c25_tpr <- round(sum(cm_c25_tables[1, 151:160]) /
                      (sum(cm_c25_tables[1, 151:160]) +
                       sum(cm_c25_tables[2, 151:160])), 6)
k33c25_tnr <- round(sum(cm_c25_tables[4, 151:160]) /
                      (sum(cm_c25_tables[4, 151:160]) +
                       sum(cm_c25_tables[3, 151:160])), 6)
k33c25_truescore <- round((2 * k33c25_tpr * k33c25_tnr) /
                             (k33c25_tpr + k33c25_tnr), 6)

```

```

k35c25_tpr <- round(sum(cm_c25_tables[1, 161:170]) /
  (sum(cm_c25_tables[1, 161:170]) +
   sum(cm_c25_tables[2, 161:170])), 6)
k35c25_tnr <- round(sum(cm_c25_tables[4, 161:170]) /
  (sum(cm_c25_tables[4, 161:170]) +
   sum(cm_c25_tables[3, 161:170])), 6)
k35c25_truescore <- round((2 * k35c25_tpr * k35c25_tnr) /
  (k35c25_tpr + k35c25_tnr), 6)

k37c25_tpr <- round(sum(cm_c25_tables[1, 171:180]) /
  (sum(cm_c25_tables[1, 171:180]) +
   sum(cm_c25_tables[2, 171:180])), 6)
k37c25_tnr <- round(sum(cm_c25_tables[4, 171:180]) /
  (sum(cm_c25_tables[4, 171:180]) +
   sum(cm_c25_tables[3, 171:180])), 6)
k37c25_truescore <- round((2 * k37c25_tpr * k37c25_tnr) /
  (k37c25_tpr + k37c25_tnr), 6)

k39c25_tpr <- round(sum(cm_c25_tables[1, 181:190]) /
  (sum(cm_c25_tables[1, 181:190]) +
   sum(cm_c25_tables[2, 181:190])), 6)
k39c25_tnr <- round(sum(cm_c25_tables[4, 181:190]) /
  (sum(cm_c25_tables[4, 181:190]) +
   sum(cm_c25_tables[3, 181:190])), 6)
k39c25_truescore <- round((2 * k39c25_tpr * k39c25_tnr) /
  (k39c25_tpr + k39c25_tnr), 6)

# Compile the 0.25 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c25_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.25,
  TPR = c(k3c25_tpr, k5c25_tpr, k7c25_tpr, k9c25_tpr, k11c25_tpr,
    k13c25_tpr, k15c25_tpr, k17c25_tpr, k19c25_tpr, k21c25_tpr,
    k23c25_tpr, k25c25_tpr, k27c25_tpr, k29c25_tpr, k31c25_tpr,
    k33c25_tpr, k35c25_tpr, k37c25_tpr, k39c25_tpr),
  TNR = c(k3c25_tnr, k5c25_tnr, k7c25_tnr, k9c25_tnr, k11c25_tnr,
    k13c25_tnr, k15c25_tnr, k17c25_tnr, k19c25_tnr, k21c25_tnr,
    k23c25_tnr, k25c25_tnr, k27c25_tnr, k29c25_tnr, k31c25_tnr,
    k33c25_tnr, k35c25_tnr, k37c25_tnr, k39c25_tnr),
  Truescore = c(k3c25_truescore, k5c25_truescore, k7c25_truescore,
    k9c25_truescore, k11c25_truescore, k13c25_truescore,
    k15c25_truescore, k17c25_truescore, k19c25_truescore,
    k21c25_truescore, k23c25_truescore, k25c25_truescore,
    k27c25_truescore, k29c25_truescore, k31c25_truescore,
    k33c25_truescore, k35c25_truescore, k37c25_truescore,
    k39c25_truescore))

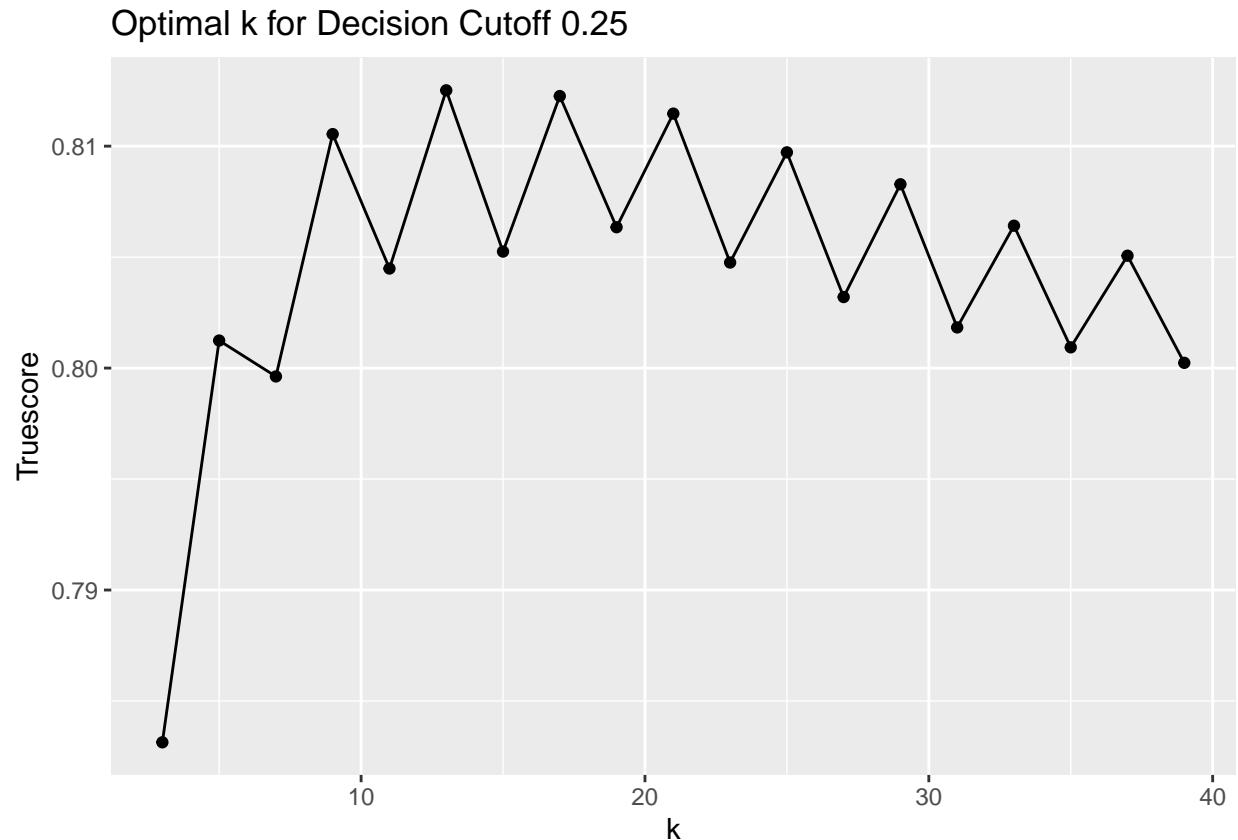
knitr::kable(c25_results[1:19, ], caption = "c25_results")

```

Table 8: c25_results

k	Cut	TPR	TNR	Truescore
3	0.25	0.853815	0.723268	0.783138
5	0.25	0.794629	0.807964	0.801241
7	0.25	0.869378	0.740231	0.799623
9	0.25	0.834438	0.787979	0.810543
11	0.25	0.876255	0.743583	0.804486
13	0.25	0.851556	0.776897	0.812515
15	0.25	0.879367	0.742658	0.805251
17	0.25	0.860793	0.768903	0.812257
19	0.25	0.883082	0.741882	0.806347
21	0.25	0.867520	0.762214	0.811465
23	0.25	0.882781	0.739405	0.804757
25	0.25	0.871084	0.756433	0.809720
27	0.25	0.883785	0.736085	0.803201
29	0.25	0.873193	0.752354	0.808282
31	0.25	0.884287	0.733442	0.801832
33	0.25	0.874398	0.748241	0.806415
35	0.25	0.885141	0.731361	0.800936
37	0.25	0.876707	0.744244	0.805063
39	0.25	0.886195	0.729478	0.800236

```
ggplot(c25_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.25")
```



```

#####
# 0.26 Cutoff
#####

# For the decision cutoff of 0.26, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c26 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred26, obs))
  confusionMatrix(ss$pred26, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c26) <- fk_dfs_v

cm_c26_tables <- sapply(cm_c26, "[[", 2)
cm_c26_tables <- as_tibble(cm_c26_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.26.

k3c26_tpr <- round(sum(cm_c26_tables[1, 1:10]) /
  (sum(cm_c26_tables[1, 1:10]) + sum(cm_c26_tables[2, 1:10])), 6)
k3c26_tnr <- round(sum(cm_c26_tables[4, 1:10]) /
  (sum(cm_c26_tables[4, 1:10]) + sum(cm_c26_tables[3, 1:10])), 6)
k3c26_truescore <- round((2 * k3c26_tpr * k3c26_tnr) /
  (k3c26_tpr + k3c26_tnr), 6)

k5c26_tpr <- round(sum(cm_c26_tables[1, 11:20]) /
  (sum(cm_c26_tables[1, 11:20]) + sum(cm_c26_tables[2, 11:20])), 6)
k5c26_tnr <- round(sum(cm_c26_tables[4, 11:20]) /
  (sum(cm_c26_tables[4, 11:20]) + sum(cm_c26_tables[3, 11:20])), 6)
k5c26_truescore <- round((2 * k5c26_tpr * k5c26_tnr) /
  (k5c26_tpr + k5c26_tnr), 6)

k7c26_tpr <- round(sum(cm_c26_tables[1, 21:30]) /
  (sum(cm_c26_tables[1, 21:30]) + sum(cm_c26_tables[2, 21:30])), 6)
k7c26_tnr <- round(sum(cm_c26_tables[4, 21:30]) /
  (sum(cm_c26_tables[4, 21:30]) + sum(cm_c26_tables[3, 21:30])), 6)
k7c26_truescore <- round((2 * k7c26_tpr * k7c26_tnr) /
  (k7c26_tpr + k7c26_tnr), 6)

k9c26_tpr <- round(sum(cm_c26_tables[1, 31:40]) /
  (sum(cm_c26_tables[1, 31:40]) + sum(cm_c26_tables[2, 31:40])), 6)
k9c26_tnr <- round(sum(cm_c26_tables[4, 31:40]) /
  (sum(cm_c26_tables[4, 31:40]) + sum(cm_c26_tables[3, 31:40])), 6)
k9c26_truescore <- round((2 * k9c26_tpr * k9c26_tnr) /
  (k9c26_tpr + k9c26_tnr), 6)

k11c26_tpr <- round(sum(cm_c26_tables[1, 41:50]) /
  (sum(cm_c26_tables[1, 41:50]) + sum(cm_c26_tables[2, 41:50])), 6)
k11c26_tnr <- round(sum(cm_c26_tables[4, 41:50]) /

```

```

(sum(cm_c26_tables[4, 41:50]) + sum(cm_c26_tables[3, 41:50])), 6)
k11c26_truescore <- round((2 * k11c26_tpr * k11c26_tnr) /
(k11c26_tpr + k11c26_tnr), 6)

k13c26_tpr <- round(sum(cm_c26_tables[1, 51:60]) /
(sum(cm_c26_tables[1, 51:60]) + sum(cm_c26_tables[2, 51:60])), 6)
k13c26_tnr <- round(sum(cm_c26_tables[4, 51:60]) /
(sum(cm_c26_tables[4, 51:60]) + sum(cm_c26_tables[3, 51:60])), 6)
k13c26_truescore <- round((2 * k13c26_tpr * k13c26_tnr) /
(k13c26_tpr + k13c26_tnr), 6)

k15c26_tpr <- round(sum(cm_c26_tables[1, 61:70]) /
(sum(cm_c26_tables[1, 61:70]) + sum(cm_c26_tables[2, 61:70])), 6)
k15c26_tnr <- round(sum(cm_c26_tables[4, 61:70]) /
(sum(cm_c26_tables[4, 61:70]) + sum(cm_c26_tables[3, 61:70])), 6)
k15c26_truescore <- round((2 * k15c26_tpr * k15c26_tnr) /
(k15c26_tpr + k15c26_tnr), 6)

k17c26_tpr <- round(sum(cm_c26_tables[1, 71:80]) /
(sum(cm_c26_tables[1, 71:80]) + sum(cm_c26_tables[2, 71:80])), 6)
k17c26_tnr <- round(sum(cm_c26_tables[4, 71:80]) /
(sum(cm_c26_tables[4, 71:80]) + sum(cm_c26_tables[3, 71:80])), 6)
k17c26_truescore <- round((2 * k17c26_tpr * k17c26_tnr) /
(k17c26_tpr + k17c26_tnr), 6)

k19c26_tpr <- round(sum(cm_c26_tables[1, 81:90]) /
(sum(cm_c26_tables[1, 81:90]) + sum(cm_c26_tables[2, 81:90])), 6)
k19c26_tnr <- round(sum(cm_c26_tables[4, 81:90]) /
(sum(cm_c26_tables[4, 81:90]) + sum(cm_c26_tables[3, 81:90])), 6)
k19c26_truescore <- round((2 * k19c26_tpr * k19c26_tnr) /
(k19c26_tpr + k19c26_tnr), 6)

k21c26_tpr <- round(sum(cm_c26_tables[1, 91:100]) /
(sum(cm_c26_tables[1, 91:100]) + sum(cm_c26_tables[2, 91:100])), 6)
k21c26_tnr <- round(sum(cm_c26_tables[4, 91:100]) /
(sum(cm_c26_tables[4, 91:100]) + sum(cm_c26_tables[3, 91:100])), 6)
k21c26_truescore <- round((2 * k21c26_tpr * k21c26_tnr) /
(k21c26_tpr + k21c26_tnr), 6)

k23c26_tpr <- round(sum(cm_c26_tables[1, 101:110]) /
(sum(cm_c26_tables[1, 101:110]) + sum(cm_c26_tables[2, 101:110])), 6)
k23c26_tnr <- round(sum(cm_c26_tables[4, 101:110]) /
(sum(cm_c26_tables[4, 101:110]) + sum(cm_c26_tables[3, 101:110])), 6)
k23c26_truescore <- round((2 * k23c26_tpr * k23c26_tnr) /
(k23c26_tpr + k23c26_tnr), 6)

k25c26_tpr <- round(sum(cm_c26_tables[1, 111:120]) /
(sum(cm_c26_tables[1, 111:120]) + sum(cm_c26_tables[2, 111:120])), 6)
k25c26_tnr <- round(sum(cm_c26_tables[4, 111:120]) /
(sum(cm_c26_tables[4, 111:120]) + sum(cm_c26_tables[3, 111:120])), 6)
k25c26_truescore <- round((2 * k25c26_tpr * k25c26_tnr) /
(k25c26_tpr + k25c26_tnr), 6)

```

```

k27c26_tpr <- round(sum(cm_c26_tables[1, 121:130]) /
  (sum(cm_c26_tables[1, 121:130]) + sum(cm_c26_tables[2, 121:130])), 6)
k27c26_tnr <- round(sum(cm_c26_tables[4, 121:130]) /
  (sum(cm_c26_tables[4, 121:130]) + sum(cm_c26_tables[3, 121:130])), 6)
k27c26_truescore <- round((2 * k27c26_tpr * k27c26_tnr) /
  (k27c26_tpr + k27c26_tnr), 6)

k29c26_tpr <- round(sum(cm_c26_tables[1, 131:140]) /
  (sum(cm_c26_tables[1, 131:140]) + sum(cm_c26_tables[2, 131:140])), 6)
k29c26_tnr <- round(sum(cm_c26_tables[4, 131:140]) /
  (sum(cm_c26_tables[4, 131:140]) + sum(cm_c26_tables[3, 131:140])), 6)
k29c26_truescore <- round((2 * k29c26_tpr * k29c26_tnr) /
  (k29c26_tpr + k29c26_tnr), 6)

k31c26_tpr <- round(sum(cm_c26_tables[1, 141:150]) /
  (sum(cm_c26_tables[1, 141:150]) + sum(cm_c26_tables[2, 141:150])), 6)
k31c26_tnr <- round(sum(cm_c26_tables[4, 141:150]) /
  (sum(cm_c26_tables[4, 141:150]) + sum(cm_c26_tables[3, 141:150])), 6)
k31c26_truescore <- round((2 * k31c26_tpr * k31c26_tnr) /
  (k31c26_tpr + k31c26_tnr), 6)

k33c26_tpr <- round(sum(cm_c26_tables[1, 151:160]) /
  (sum(cm_c26_tables[1, 151:160]) + sum(cm_c26_tables[2, 151:160])), 6)
k33c26_tnr <- round(sum(cm_c26_tables[4, 151:160]) /
  (sum(cm_c26_tables[4, 151:160]) + sum(cm_c26_tables[3, 151:160])), 6)
k33c26_truescore <- round((2 * k33c26_tpr * k33c26_tnr) /
  (k33c26_tpr + k33c26_tnr), 6)

k35c26_tpr <- round(sum(cm_c26_tables[1, 161:170]) /
  (sum(cm_c26_tables[1, 161:170]) + sum(cm_c26_tables[2, 161:170])), 6)
k35c26_tnr <- round(sum(cm_c26_tables[4, 161:170]) /
  (sum(cm_c26_tables[4, 161:170]) + sum(cm_c26_tables[3, 161:170])), 6)
k35c26_truescore <- round((2 * k35c26_tpr * k35c26_tnr) /
  (k35c26_tpr + k35c26_tnr), 6)

k37c26_tpr <- round(sum(cm_c26_tables[1, 171:180]) /
  (sum(cm_c26_tables[1, 171:180]) + sum(cm_c26_tables[2, 171:180])), 6)
k37c26_tnr <- round(sum(cm_c26_tables[4, 171:180]) /
  (sum(cm_c26_tables[4, 171:180]) + sum(cm_c26_tables[3, 171:180])), 6)
k37c26_truescore <- round((2 * k37c26_tpr * k37c26_tnr) /
  (k37c26_tpr + k37c26_tnr), 6)

k39c26_tpr <- round(sum(cm_c26_tables[1, 181:190]) /
  (sum(cm_c26_tables[1, 181:190]) + sum(cm_c26_tables[2, 181:190])), 6)
k39c26_tnr <- round(sum(cm_c26_tables[4, 181:190]) /
  (sum(cm_c26_tables[4, 181:190]) + sum(cm_c26_tables[3, 181:190])), 6)
k39c26_truescore <- round((2 * k39c26_tpr * k39c26_tnr) /
  (k39c26_tpr + k39c26_tnr), 6)

# Compile the 0.26 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c26_results <- tibble(k = seq(3, 39, 2),

```

```

Cut = 0.26,
TPR = c(k3c26_tpr, k5c26_tpr, k7c26_tpr, k9c26_tpr, k11c26_tpr,
         k13c26_tpr, k15c26_tpr, k17c26_tpr, k19c26_tpr, k21c26_tpr,
         k23c26_tpr, k25c26_tpr, k27c26_tpr, k29c26_tpr, k31c26_tpr,
         k33c26_tpr, k35c26_tpr, k37c26_tpr, k39c26_tpr),
TNR = c(k3c26_tnr, k5c26_tnr, k7c26_tnr, k9c26_tnr, k11c26_tnr,
         k13c26_tnr, k15c26_tnr, k17c26_tnr, k19c26_tnr, k21c26_tnr,
         k23c26_tnr, k25c26_tnr, k27c26_tnr, k29c26_tnr, k31c26_tnr,
         k33c26_tnr, k35c26_tnr, k37c26_tnr, k39c26_tnr),
Truescore = c(k3c26_truescore, k5c26_truescore, k7c26_truescore,
              k9c26_truescore, k11c26_truescore, k13c26_truescore,
              k15c26_truescore, k17c26_truescore, k19c26_truescore,
              k21c26_truescore, k23c26_truescore, k25c26_truescore,
              k27c26_truescore, k29c26_truescore, k31c26_truescore,
              k33c26_truescore, k35c26_truescore, k37c26_truescore,
              k39c26_truescore))

knitr::kable(c26_results[1:19, ], caption = "c26_results")

```

Table 9: c26_results

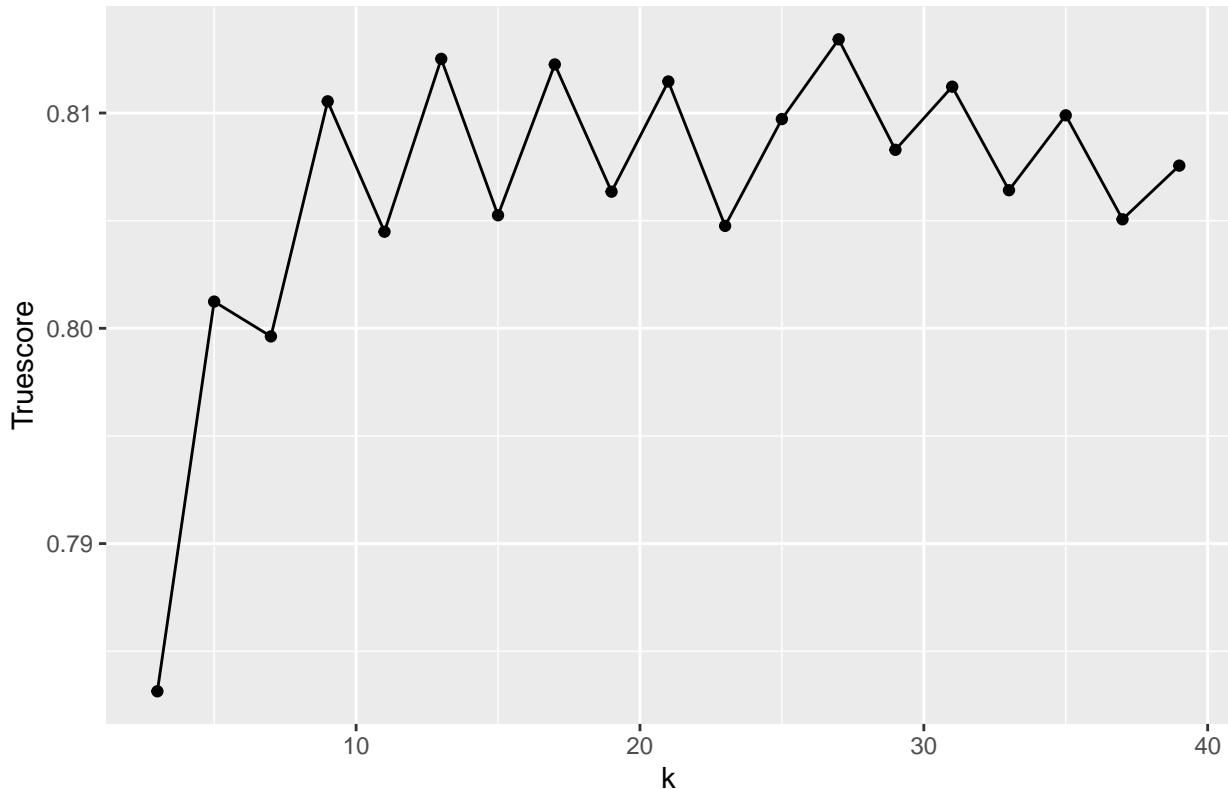
k	Cut	TPR	TNR	Truescore
3	0.26	0.853815	0.723268	0.783138
5	0.26	0.794629	0.807964	0.801241
7	0.26	0.869378	0.740231	0.799623
9	0.26	0.834438	0.787979	0.810543
11	0.26	0.876255	0.743583	0.804486
13	0.26	0.851556	0.776897	0.812515
15	0.26	0.879367	0.742658	0.805251
17	0.26	0.860793	0.768903	0.812257
19	0.26	0.883082	0.741882	0.806347
21	0.26	0.867520	0.762214	0.811465
23	0.26	0.882781	0.739405	0.804757
25	0.26	0.871084	0.756433	0.809720
27	0.26	0.859739	0.771843	0.813423
29	0.26	0.873193	0.752370	0.808291
31	0.26	0.862902	0.765385	0.811223
33	0.26	0.874398	0.748241	0.806415
35	0.26	0.866968	0.759869	0.809893
37	0.26	0.876707	0.744244	0.805063
39	0.26	0.868223	0.754815	0.807557

```

ggplot(c26_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.26")

```

Optimal k for Decision Cutoff 0.26



```
#####
# 0.27 Cutoff
#####

# For the decision cutoff of 0.27, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c27 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred27, obs))
  confusionMatrix(ss$pred27, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c27) <- fk_dfs_v

cm_c27_tables <- sapply(cm_c27, "[[", 2)
cm_c27_tables <- as_tibble(cm_c27_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.27.

k3c27_tpr <- round(sum(cm_c27_tables[1, 1:10]) /
  (sum(cm_c27_tables[1, 1:10]) + sum(cm_c27_tables[2, 1:10])), 6)
k3c27_tnr <- round(sum(cm_c27_tables[4, 1:10]) /
  (sum(cm_c27_tables[4, 1:10]) + sum(cm_c27_tables[3, 1:10])), 6)
```

```

k3c27_truescore <- round((2 * k3c27_tpr * k3c27_tnr) /
  (k3c27_tpr + k3c27_tnr), 6)

k5c27_tpr <- round(sum(cm_c27_tables[1, 11:20]) /
  (sum(cm_c27_tables[1, 11:20]) + sum(cm_c27_tables[2, 11:20])), 6)
k5c27_tnr <- round(sum(cm_c27_tables[4, 11:20]) /
  (sum(cm_c27_tables[4, 11:20]) + sum(cm_c27_tables[3, 11:20])), 6)
k5c27_truescore <- round((2 * k5c27_tpr * k5c27_tnr) /
  (k5c27_tpr + k5c27_tnr), 6)

k7c27_tpr <- round(sum(cm_c27_tables[1, 21:30]) /
  (sum(cm_c27_tables[1, 21:30]) + sum(cm_c27_tables[2, 21:30])), 6)
k7c27_tnr <- round(sum(cm_c27_tables[4, 21:30]) /
  (sum(cm_c27_tables[4, 21:30]) + sum(cm_c27_tables[3, 21:30])), 6)
k7c27_truescore <- round((2 * k7c27_tpr * k7c27_tnr) /
  (k7c27_tpr + k7c27_tnr), 6)

k9c27_tpr <- round(sum(cm_c27_tables[1, 31:40]) /
  (sum(cm_c27_tables[1, 31:40]) + sum(cm_c27_tables[2, 31:40])), 6)
k9c27_tnr <- round(sum(cm_c27_tables[4, 31:40]) /
  (sum(cm_c27_tables[4, 31:40]) + sum(cm_c27_tables[3, 31:40])), 6)
k9c27_truescore <- round((2 * k9c27_tpr * k9c27_tnr) /
  (k9c27_tpr + k9c27_tnr), 6)

k11c27_tpr <- round(sum(cm_c27_tables[1, 41:50]) /
  (sum(cm_c27_tables[1, 41:50]) + sum(cm_c27_tables[2, 41:50])), 6)
k11c27_tnr <- round(sum(cm_c27_tables[4, 41:50]) /
  (sum(cm_c27_tables[4, 41:50]) + sum(cm_c27_tables[3, 41:50])), 6)
k11c27_truescore <- round((2 * k11c27_tpr * k11c27_tnr) /
  (k11c27_tpr + k11c27_tnr), 6)

k13c27_tpr <- round(sum(cm_c27_tables[1, 51:60]) /
  (sum(cm_c27_tables[1, 51:60]) + sum(cm_c27_tables[2, 51:60])), 6)
k13c27_tnr <- round(sum(cm_c27_tables[4, 51:60]) /
  (sum(cm_c27_tables[4, 51:60]) + sum(cm_c27_tables[3, 51:60])), 6)
k13c27_truescore <- round((2 * k13c27_tpr * k13c27_tnr) /
  (k13c27_tpr + k13c27_tnr), 6)

k15c27_tpr <- round(sum(cm_c27_tables[1, 61:70]) /
  (sum(cm_c27_tables[1, 61:70]) + sum(cm_c27_tables[2, 61:70])), 6)
k15c27_tnr <- round(sum(cm_c27_tables[4, 61:70]) /
  (sum(cm_c27_tables[4, 61:70]) + sum(cm_c27_tables[3, 61:70])), 6)
k15c27_truescore <- round((2 * k15c27_tpr * k15c27_tnr) /
  (k15c27_tpr + k15c27_tnr), 6)

k17c27_tpr <- round(sum(cm_c27_tables[1, 71:80]) /
  (sum(cm_c27_tables[1, 71:80]) + sum(cm_c27_tables[2, 71:80])), 6)
k17c27_tnr <- round(sum(cm_c27_tables[4, 71:80]) /
  (sum(cm_c27_tables[4, 71:80]) + sum(cm_c27_tables[3, 71:80])), 6)
k17c27_truescore <- round((2 * k17c27_tpr * k17c27_tnr) /
  (k17c27_tpr + k17c27_tnr), 6)

k19c27_tpr <- round(sum(cm_c27_tables[1, 81:90]) /

```

```

(sum(cm_c27_tables[1, 81:90]) + sum(cm_c27_tables[2, 81:90])), 6)
k19c27_tnr <- round(sum(cm_c27_tables[4, 81:90]) /
  (sum(cm_c27_tables[4, 81:90]) + sum(cm_c27_tables[3, 81:90])), 6)
k19c27_truescore <- round((2 * k19c27_tpr * k19c27_tnr) /
  (k19c27_tpr + k19c27_tnr), 6)

k21c27_tpr <- round(sum(cm_c27_tables[1, 91:100]) /
  (sum(cm_c27_tables[1, 91:100]) + sum(cm_c27_tables[2, 91:100])), 6)
k21c27_tnr <- round(sum(cm_c27_tables[4, 91:100]) /
  (sum(cm_c27_tables[4, 91:100]) + sum(cm_c27_tables[3, 91:100])), 6)
k21c27_truescore <- round((2 * k21c27_tpr * k21c27_tnr) /
  (k21c27_tpr + k21c27_tnr), 6)

k23c27_tpr <- round(sum(cm_c27_tables[1, 101:110]) /
  (sum(cm_c27_tables[1, 101:110]) + sum(cm_c27_tables[2, 101:110])), 6)
k23c27_tnr <- round(sum(cm_c27_tables[4, 101:110]) /
  (sum(cm_c27_tables[4, 101:110]) + sum(cm_c27_tables[3, 101:110])), 6)
k23c27_truescore <- round((2 * k23c27_tpr * k23c27_tnr) /
  (k23c27_tpr + k23c27_tnr), 6)

k25c27_tpr <- round(sum(cm_c27_tables[1, 111:120]) /
  (sum(cm_c27_tables[1, 111:120]) + sum(cm_c27_tables[2, 111:120])), 6)
k25c27_tnr <- round(sum(cm_c27_tables[4, 111:120]) /
  (sum(cm_c27_tables[4, 111:120]) + sum(cm_c27_tables[3, 111:120])), 6)
k25c27_truescore <- round((2 * k25c27_tpr * k25c27_tnr) /
  (k25c27_tpr + k25c27_tnr), 6)

k27c27_tpr <- round(sum(cm_c27_tables[1, 121:130]) /
  (sum(cm_c27_tables[1, 121:130]) + sum(cm_c27_tables[2, 121:130])), 6)
k27c27_tnr <- round(sum(cm_c27_tables[4, 121:130]) /
  (sum(cm_c27_tables[4, 121:130]) + sum(cm_c27_tables[3, 121:130])), 6)
k27c27_truescore <- round((2 * k27c27_tpr * k27c27_tnr) /
  (k27c27_tpr + k27c27_tnr), 6)

k29c27_tpr <- round(sum(cm_c27_tables[1, 131:140]) /
  (sum(cm_c27_tables[1, 131:140]) + sum(cm_c27_tables[2, 131:140])), 6)
k29c27_tnr <- round(sum(cm_c27_tables[4, 131:140]) /
  (sum(cm_c27_tables[4, 131:140]) + sum(cm_c27_tables[3, 131:140])), 6)
k29c27_truescore <- round((2 * k29c27_tpr * k29c27_tnr) /
  (k29c27_tpr + k29c27_tnr), 6)

k31c27_tpr <- round(sum(cm_c27_tables[1, 141:150]) /
  (sum(cm_c27_tables[1, 141:150]) + sum(cm_c27_tables[2, 141:150])), 6)
k31c27_tnr <- round(sum(cm_c27_tables[4, 141:150]) /
  (sum(cm_c27_tables[4, 141:150]) + sum(cm_c27_tables[3, 141:150])), 6)
k31c27_truescore <- round((2 * k31c27_tpr * k31c27_tnr) /
  (k31c27_tpr + k31c27_tnr), 6)

k33c27_tpr <- round(sum(cm_c27_tables[1, 151:160]) /
  (sum(cm_c27_tables[1, 151:160]) + sum(cm_c27_tables[2, 151:160])), 6)
k33c27_tnr <- round(sum(cm_c27_tables[4, 151:160]) /
  (sum(cm_c27_tables[4, 151:160]) + sum(cm_c27_tables[3, 151:160])), 6)
k33c27_truescore <- round((2 * k33c27_tpr * k33c27_tnr) /

```

```

(k33c27_tpr + k33c27_tnr), 6)

k35c27_tpr <- round(sum(cm_c27_tables[1, 161:170]) /
  (sum(cm_c27_tables[1, 161:170]) + sum(cm_c27_tables[2, 161:170])), 6)
k35c27_tnr <- round(sum(cm_c27_tables[4, 161:170]) /
  (sum(cm_c27_tables[4, 161:170]) + sum(cm_c27_tables[3, 161:170])), 6)
k35c27_truescore <- round((2 * k35c27_tpr * k35c27_tnr) /
  (k35c27_tpr + k35c27_tnr), 6)

k37c27_tpr <- round(sum(cm_c27_tables[1, 171:180]) /
  (sum(cm_c27_tables[1, 171:180]) + sum(cm_c27_tables[2, 171:180])), 6)
k37c27_tnr <- round(sum(cm_c27_tables[4, 171:180]) /
  (sum(cm_c27_tables[4, 171:180]) + sum(cm_c27_tables[3, 171:180])), 6)
k37c27_truescore <- round((2 * k37c27_tpr * k37c27_tnr) /
  (k37c27_tpr + k37c27_tnr), 6)

k39c27_tpr <- round(sum(cm_c27_tables[1, 181:190]) /
  (sum(cm_c27_tables[1, 181:190]) + sum(cm_c27_tables[2, 181:190])), 6)
k39c27_tnr <- round(sum(cm_c27_tables[4, 181:190]) /
  (sum(cm_c27_tables[4, 181:190]) + sum(cm_c27_tables[3, 181:190])), 6)
k39c27_truescore <- round((2 * k39c27_tpr * k39c27_tnr) /
  (k39c27_tpr + k39c27_tnr), 6)

# Compile the 0.27 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c27_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.27,
  TPR = c(k3c27_tpr, k5c27_tpr, k7c27_tpr, k9c27_tpr, k11c27_tpr,
    k13c27_tpr, k15c27_tpr, k17c27_tpr, k19c27_tpr, k21c27_tpr,
    k23c27_tpr, k25c27_tpr, k27c27_tpr, k29c27_tpr, k31c27_tpr,
    k33c27_tpr, k35c27_tpr, k37c27_tpr, k39c27_tpr),
  TNR = c(k3c27_tnr, k5c27_tnr, k7c27_tnr, k9c27_tnr, k11c27_tnr,
    k13c27_tnr, k15c27_tnr, k17c27_tnr, k19c27_tnr, k21c27_tnr,
    k23c27_tnr, k25c27_tnr, k27c27_tnr, k29c27_tnr, k31c27_tnr,
    k33c27_tnr, k35c27_tnr, k37c27_tnr, k39c27_tnr),
  Truescore = c(k3c27_truescore, k5c27_truescore, k7c27_truescore,
    k9c27_truescore, k11c27_truescore, k13c27_truescore,
    k15c27_truescore, k17c27_truescore, k19c27_truescore,
    k21c27_truescore, k23c27_truescore, k25c27_truescore,
    k27c27_truescore, k29c27_truescore, k31c27_truescore,
    k33c27_truescore, k35c27_truescore, k37c27_truescore,
    k39c27_truescore))

knitr::kable(c27_results[1:19, ], caption = "c27_results")

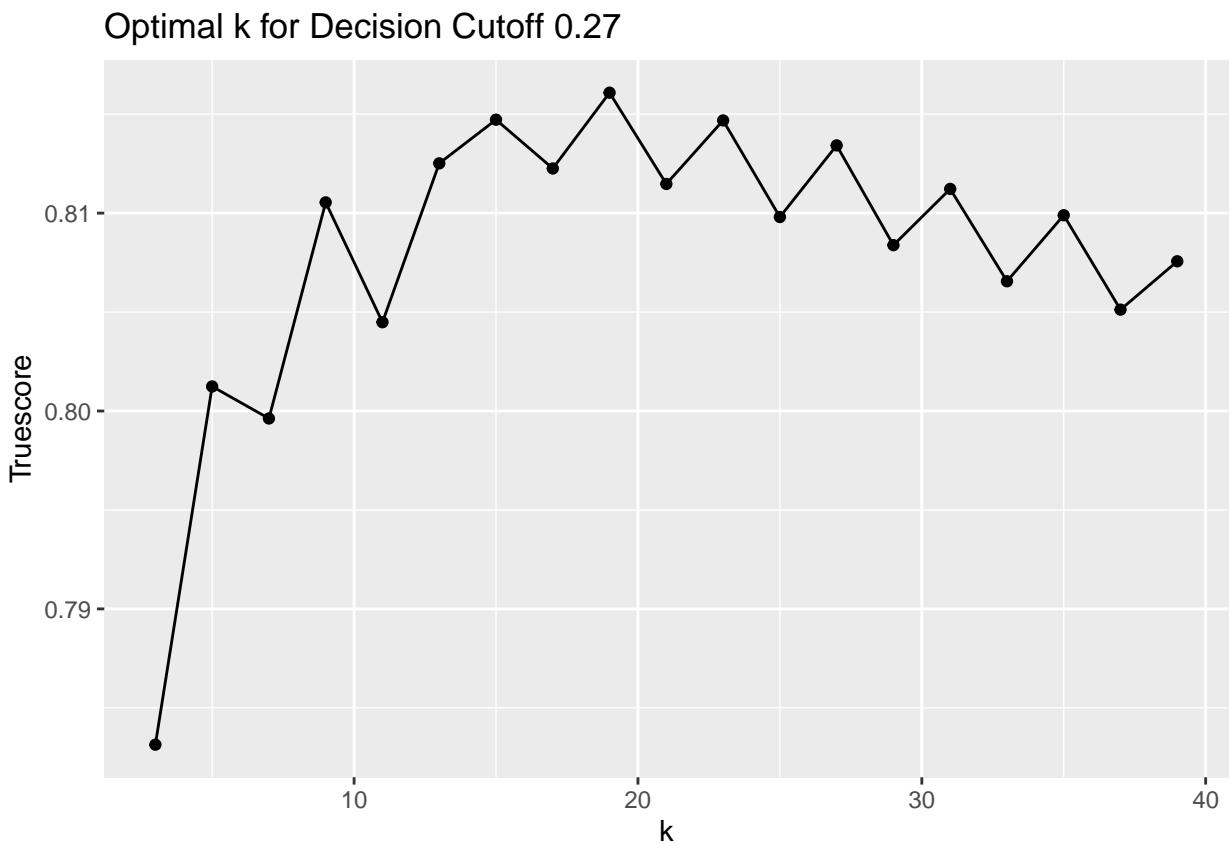
```

Table 10: c27_results

k	Cut	TPR	TNR	Truescore
3	0.27	0.853815	0.723268	0.783138
5	0.27	0.794629	0.807964	0.801241
7	0.27	0.869378	0.740231	0.799623
9	0.27	0.834438	0.787979	0.810543

k	Cut	TPR	TNR	Truescore
11	0.27	0.876255	0.743583	0.804486
13	0.27	0.851556	0.776897	0.812515
15	0.27	0.830773	0.799277	0.814721
17	0.27	0.860793	0.768903	0.812257
19	0.27	0.845833	0.788359	0.816085
21	0.27	0.867520	0.762230	0.811474
23	0.27	0.854418	0.778482	0.814684
25	0.27	0.871034	0.756615	0.809803
27	0.27	0.859739	0.771843	0.813423
29	0.27	0.873092	0.752601	0.808381
31	0.27	0.862902	0.765385	0.811223
33	0.27	0.874347	0.748522	0.806557
35	0.27	0.866968	0.759869	0.809893
37	0.27	0.876456	0.744525	0.805122
39	0.27	0.868223	0.754831	0.807566

```
ggplot(c27_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.27")
```



```
#####
# 0.28 Cutoff
#####
```

```

# For the decision cutoff of 0.28, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c28 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred28, obs))
  confusionMatrix(ss$pred28, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c28) <- fk_dfs_v

cm_c28_tables <- sapply(cm_c28, "[[", 2)
cm_c28_tables <- as_tibble(cm_c28_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.28.

k3c28_tpr <- round(sum(cm_c28_tables[1, 1:10]) /
  (sum(cm_c28_tables[1, 1:10]) + sum(cm_c28_tables[2, 1:10])), 6)
k3c28_tnr <- round(sum(cm_c28_tables[4, 1:10]) /
  (sum(cm_c28_tables[4, 1:10]) + sum(cm_c28_tables[3, 1:10])), 6)
k3c28_truescore <- round((2 * k3c28_tpr * k3c28_tnr) /
  (k3c28_tpr + k3c28_tnr), 6)

k5c28_tpr <- round(sum(cm_c28_tables[1, 11:20]) /
  (sum(cm_c28_tables[1, 11:20]) + sum(cm_c28_tables[2, 11:20])), 6)
k5c28_tnr <- round(sum(cm_c28_tables[4, 11:20]) /
  (sum(cm_c28_tables[4, 11:20]) + sum(cm_c28_tables[3, 11:20])), 6)
k5c28_truescore <- round((2 * k5c28_tpr * k5c28_tnr) /
  (k5c28_tpr + k5c28_tnr), 6)

k7c28_tpr <- round(sum(cm_c28_tables[1, 21:30]) /
  (sum(cm_c28_tables[1, 21:30]) + sum(cm_c28_tables[2, 21:30])), 6)
k7c28_tnr <- round(sum(cm_c28_tables[4, 21:30]) /
  (sum(cm_c28_tables[4, 21:30]) + sum(cm_c28_tables[3, 21:30])), 6)
k7c28_truescore <- round((2 * k7c28_tpr * k7c28_tnr) /
  (k7c28_tpr + k7c28_tnr), 6)

k9c28_tpr <- round(sum(cm_c28_tables[1, 31:40]) /
  (sum(cm_c28_tables[1, 31:40]) + sum(cm_c28_tables[2, 31:40])), 6)
k9c28_tnr <- round(sum(cm_c28_tables[4, 31:40]) /
  (sum(cm_c28_tables[4, 31:40]) + sum(cm_c28_tables[3, 31:40])), 6)
k9c28_truescore <- round((2 * k9c28_tpr * k9c28_tnr) /
  (k9c28_tpr + k9c28_tnr), 6)

k11c28_tpr <- round(sum(cm_c28_tables[1, 41:50]) /
  (sum(cm_c28_tables[1, 41:50]) + sum(cm_c28_tables[2, 41:50])), 6)
k11c28_tnr <- round(sum(cm_c28_tables[4, 41:50]) /
  (sum(cm_c28_tables[4, 41:50]) + sum(cm_c28_tables[3, 41:50])), 6)
k11c28_truescore <- round((2 * k11c28_tpr * k11c28_tnr) /
  (k11c28_tpr + k11c28_tnr), 6)

```

```

k13c28_tpr <- round(sum(cm_c28_tables[1, 51:60]) /
  (sum(cm_c28_tables[1, 51:60]) + sum(cm_c28_tables[2, 51:60])), 6)
k13c28_tnr <- round(sum(cm_c28_tables[4, 51:60]) /
  (sum(cm_c28_tables[4, 51:60]) + sum(cm_c28_tables[3, 51:60])), 6)
k13c28_truescore <- round((2 * k13c28_tpr * k13c28_tnr) /
  (k13c28_tpr + k13c28_tnr), 6)

k15c28_tpr <- round(sum(cm_c28_tables[1, 61:70]) /
  (sum(cm_c28_tables[1, 61:70]) + sum(cm_c28_tables[2, 61:70])), 6)
k15c28_tnr <- round(sum(cm_c28_tables[4, 61:70]) /
  (sum(cm_c28_tables[4, 61:70]) + sum(cm_c28_tables[3, 61:70])), 6)
k15c28_truescore <- round((2 * k15c28_tpr * k15c28_tnr) /
  (k15c28_tpr + k15c28_tnr), 6)

k17c28_tpr <- round(sum(cm_c28_tables[1, 71:80]) /
  (sum(cm_c28_tables[1, 71:80]) + sum(cm_c28_tables[2, 71:80])), 6)
k17c28_tnr <- round(sum(cm_c28_tables[4, 71:80]) /
  (sum(cm_c28_tables[4, 71:80]) + sum(cm_c28_tables[3, 71:80])), 6)
k17c28_truescore <- round((2 * k17c28_tpr * k17c28_tnr) /
  (k17c28_tpr + k17c28_tnr), 6)

k19c28_tpr <- round(sum(cm_c28_tables[1, 81:90]) /
  (sum(cm_c28_tables[1, 81:90]) + sum(cm_c28_tables[2, 81:90])), 6)
k19c28_tnr <- round(sum(cm_c28_tables[4, 81:90]) /
  (sum(cm_c28_tables[4, 81:90]) + sum(cm_c28_tables[3, 81:90])), 6)
k19c28_truescore <- round((2 * k19c28_tpr * k19c28_tnr) /
  (k19c28_tpr + k19c28_tnr), 6)

k21c28_tpr <- round(sum(cm_c28_tables[1, 91:100]) /
  (sum(cm_c28_tables[1, 91:100]) + sum(cm_c28_tables[2, 91:100])), 6)
k21c28_tnr <- round(sum(cm_c28_tables[4, 91:100]) /
  (sum(cm_c28_tables[4, 91:100]) + sum(cm_c28_tables[3, 91:100])), 6)
k21c28_truescore <- round((2 * k21c28_tpr * k21c28_tnr) /
  (k21c28_tpr + k21c28_tnr), 6)

k23c28_tpr <- round(sum(cm_c28_tables[1, 101:110]) /
  (sum(cm_c28_tables[1, 101:110]) + sum(cm_c28_tables[2, 101:110])), 6)
k23c28_tnr <- round(sum(cm_c28_tables[4, 101:110]) /
  (sum(cm_c28_tables[4, 101:110]) + sum(cm_c28_tables[3, 101:110])), 6)
k23c28_truescore <- round((2 * k23c28_tpr * k23c28_tnr) /
  (k23c28_tpr + k23c28_tnr), 6)

k25c28_tpr <- round(sum(cm_c28_tables[1, 111:120]) /
  (sum(cm_c28_tables[1, 111:120]) + sum(cm_c28_tables[2, 111:120])), 6)
k25c28_tnr <- round(sum(cm_c28_tables[4, 111:120]) /
  (sum(cm_c28_tables[4, 111:120]) + sum(cm_c28_tables[3, 111:120])), 6)
k25c28_truescore <- round((2 * k25c28_tpr * k25c28_tnr) /
  (k25c28_tpr + k25c28_tnr), 6)

k27c28_tpr <- round(sum(cm_c28_tables[1, 121:130]) /
  (sum(cm_c28_tables[1, 121:130]) + sum(cm_c28_tables[2, 121:130])), 6)
k27c28_tnr <- round(sum(cm_c28_tables[4, 121:130]) /
  (sum(cm_c28_tables[4, 121:130]) + sum(cm_c28_tables[3, 121:130])), 6)

```

```

k27c28_truescore <- round((2 * k27c28_tpr * k27c28_tnr) /
  (k27c28_tpr + k27c28_tnr), 6)

k29c28_tpr <- round(sum(cm_c28_tables[1, 131:140]) /
  (sum(cm_c28_tables[1, 131:140]) + sum(cm_c28_tables[2, 131:140])), 6)
k29c28_tnr <- round(sum(cm_c28_tables[4, 131:140]) /
  (sum(cm_c28_tables[4, 131:140]) + sum(cm_c28_tables[3, 131:140])), 6)
k29c28_truescore <- round((2 * k29c28_tpr * k29c28_tnr) /
  (k29c28_tpr + k29c28_tnr), 6)

k31c28_tpr <- round(sum(cm_c28_tables[1, 141:150]) /
  (sum(cm_c28_tables[1, 141:150]) + sum(cm_c28_tables[2, 141:150])), 6)
k31c28_tnr <- round(sum(cm_c28_tables[4, 141:150]) /
  (sum(cm_c28_tables[4, 141:150]) + sum(cm_c28_tables[3, 141:150])), 6)
k31c28_truescore <- round((2 * k31c28_tpr * k31c28_tnr) /
  (k31c28_tpr + k31c28_tnr), 6)

k33c28_tpr <- round(sum(cm_c28_tables[1, 151:160]) /
  (sum(cm_c28_tables[1, 151:160]) + sum(cm_c28_tables[2, 151:160])), 6)
k33c28_tnr <- round(sum(cm_c28_tables[4, 151:160]) /
  (sum(cm_c28_tables[4, 151:160]) + sum(cm_c28_tables[3, 151:160])), 6)
k33c28_truescore <- round((2 * k33c28_tpr * k33c28_tnr) /
  (k33c28_tpr + k33c28_tnr), 6)

k35c28_tpr <- round(sum(cm_c28_tables[1, 161:170]) /
  (sum(cm_c28_tables[1, 161:170]) + sum(cm_c28_tables[2, 161:170])), 6)
k35c28_tnr <- round(sum(cm_c28_tables[4, 161:170]) /
  (sum(cm_c28_tables[4, 161:170]) + sum(cm_c28_tables[3, 161:170])), 6)
k35c28_truescore <- round((2 * k35c28_tpr * k35c28_tnr) /
  (k35c28_tpr + k35c28_tnr), 6)

k37c28_tpr <- round(sum(cm_c28_tables[1, 171:180]) /
  (sum(cm_c28_tables[1, 171:180]) + sum(cm_c28_tables[2, 171:180])), 6)
k37c28_tnr <- round(sum(cm_c28_tables[4, 171:180]) /
  (sum(cm_c28_tables[4, 171:180]) + sum(cm_c28_tables[3, 171:180])), 6)
k37c28_truescore <- round((2 * k37c28_tpr * k37c28_tnr) /
  (k37c28_tpr + k37c28_tnr), 6)

k39c28_tpr <- round(sum(cm_c28_tables[1, 181:190]) /
  (sum(cm_c28_tables[1, 181:190]) + sum(cm_c28_tables[2, 181:190])), 6)
k39c28_tnr <- round(sum(cm_c28_tables[4, 181:190]) /
  (sum(cm_c28_tables[4, 181:190]) + sum(cm_c28_tables[3, 181:190])), 6)
k39c28_truescore <- round((2 * k39c28_tpr * k39c28_tnr) /
  (k39c28_tpr + k39c28_tnr), 6)

# Compile the 0.28 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c28_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.28,
  TPR = c(k3c28_tpr, k5c28_tpr, k7c28_tpr, k9c28_tpr, k11c28_tpr,
    k13c28_tpr, k15c28_tpr, k17c28_tpr, k19c28_tpr, k21c28_tpr,
    k23c28_tpr, k25c28_tpr, k27c28_tpr, k29c28_tpr, k31c28_tpr,
    k33c28_tpr, k35c28_tpr, k37c28_tpr, k39c28_tpr))

```

```

        k33c28_tpr, k35c28_tpr, k37c28_tpr, k39c28_tpr),
TNR = c(k3c28_tnr, k5c28_tnr, k7c28_tnr, k9c28_tnr, k11c28_tnr,
         k13c28_tnr, k15c28_tnr, k17c28_tnr, k19c28_tnr, k21c28_tnr,
         k23c28_tnr, k25c28_tnr, k27c28_tnr, k29c28_tnr, k31c28_tnr,
         k33c28_tnr, k35c28_tnr, k37c28_tnr, k39c28_tnr),
Truescore = c(k3c28_truescore, k5c28_truescore, k7c28_truescore,
              k9c28_truescore, k11c28_truescore, k13c28_truescore,
              k15c28_truescore, k17c28_truescore, k19c28_truescore,
              k21c28_truescore, k23c28_truescore, k25c28_truescore,
              k27c28_truescore, k29c28_truescore, k31c28_truescore,
              k33c28_truescore, k35c28_truescore, k37c28_truescore,
              k39c28_truescore))

knitr::kable(c28_results[1:19, ], caption = "c28_results")

```

Table 11: c28_results

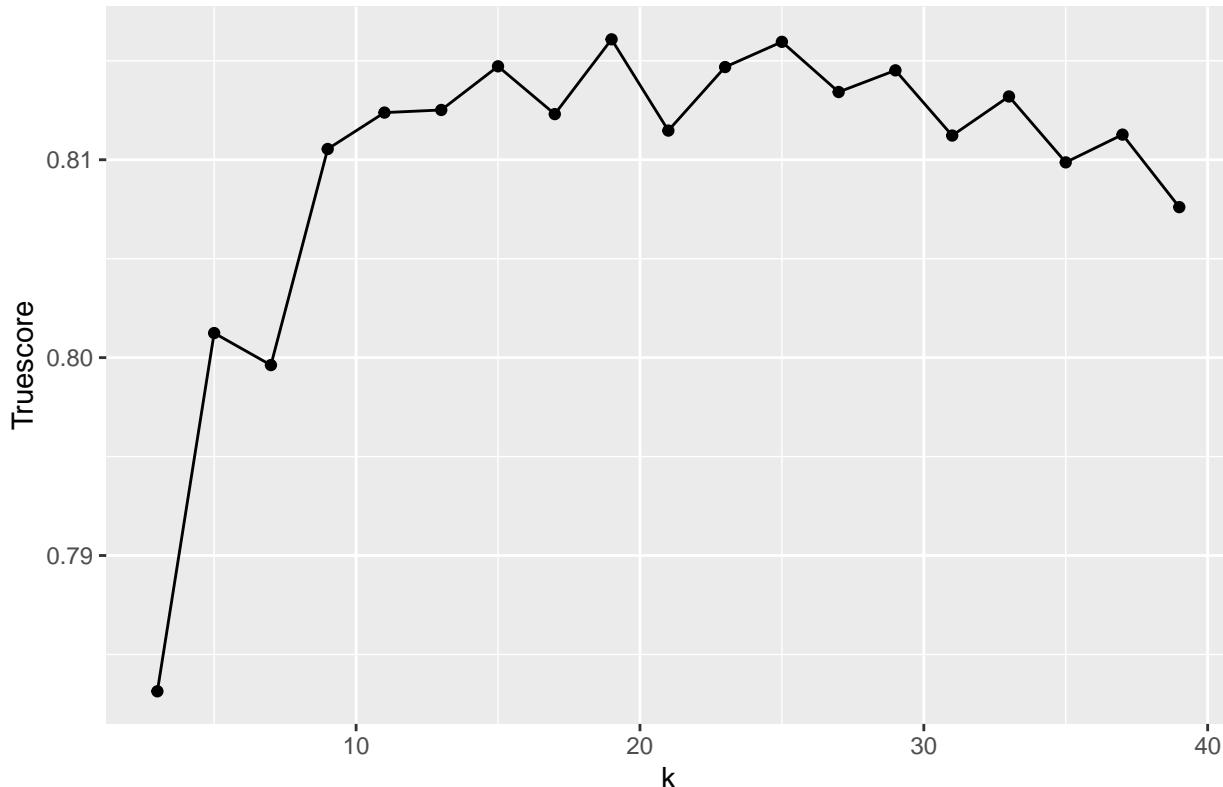
k	Cut	TPR	TNR	Truescore
3	0.28	0.853815	0.723268	0.783138
5	0.28	0.794629	0.807964	0.801241
7	0.28	0.869378	0.740231	0.799623
9	0.28	0.834438	0.787979	0.810543
11	0.28	0.809588	0.815198	0.812383
13	0.28	0.851556	0.776897	0.812515
15	0.28	0.830773	0.799277	0.814721
17	0.28	0.860592	0.769151	0.812306
19	0.28	0.845833	0.788359	0.816085
21	0.28	0.867369	0.762346	0.811474
23	0.28	0.854418	0.778482	0.814684
25	0.28	0.841717	0.791729	0.815958
27	0.28	0.859739	0.771843	0.813423
29	0.28	0.847992	0.783586	0.814518
31	0.28	0.862902	0.765385	0.811223
33	0.28	0.854317	0.775856	0.813198
35	0.28	0.866717	0.760017	0.809868
37	0.28	0.857530	0.769745	0.811270
39	0.28	0.868072	0.755013	0.807605

```

ggplot(c28_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.28")

```

Optimal k for Decision Cutoff 0.28



```
#####
# 0.29 Cutoff
#####

# For the decision cutoff of 0.29, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c29 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred29, obs))
  confusionMatrix(ss$pred29, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c29) <- fk_dfs_v

cm_c29_tables <- sapply(cm_c29, "[[", 2)
cm_c29_tables <- as_tibble(cm_c29_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.29.

k3c29_tpr <- round(sum(cm_c29_tables[1, 1:10]) /
  (sum(cm_c29_tables[1, 1:10]) + sum(cm_c29_tables[2, 1:10])), 6)
k3c29_tnr <- round(sum(cm_c29_tables[4, 1:10]) /
  (sum(cm_c29_tables[4, 1:10]) + sum(cm_c29_tables[3, 1:10])), 6)
```

```

k3c29_truescore <- round((2 * k3c29_tpr * k3c29_tnr) /
  (k3c29_tpr + k3c29_tnr), 6)

k5c29_tpr <- round(sum(cm_c29_tables[1, 11:20]) /
  (sum(cm_c29_tables[1, 11:20]) + sum(cm_c29_tables[2, 11:20])), 6)
k5c29_tnr <- round(sum(cm_c29_tables[4, 11:20]) /
  (sum(cm_c29_tables[4, 11:20]) + sum(cm_c29_tables[3, 11:20])), 6)
k5c29_truescore <- round((2 * k5c29_tpr * k5c29_tnr) /
  (k5c29_tpr + k5c29_tnr), 6)

k7c29_tpr <- round(sum(cm_c29_tables[1, 21:30]) /
  (sum(cm_c29_tables[1, 21:30]) + sum(cm_c29_tables[2, 21:30])), 6)
k7c29_tnr <- round(sum(cm_c29_tables[4, 21:30]) /
  (sum(cm_c29_tables[4, 21:30]) + sum(cm_c29_tables[3, 21:30])), 6)
k7c29_truescore <- round((2 * k7c29_tpr * k7c29_tnr) /
  (k7c29_tpr + k7c29_tnr), 6)

k9c29_tpr <- round(sum(cm_c29_tables[1, 31:40]) /
  (sum(cm_c29_tables[1, 31:40]) + sum(cm_c29_tables[2, 31:40])), 6)
k9c29_tnr <- round(sum(cm_c29_tables[4, 31:40]) /
  (sum(cm_c29_tables[4, 31:40]) + sum(cm_c29_tables[3, 31:40])), 6)
k9c29_truescore <- round((2 * k9c29_tpr * k9c29_tnr) /
  (k9c29_tpr + k9c29_tnr), 6)

k11c29_tpr <- round(sum(cm_c29_tables[1, 41:50]) /
  (sum(cm_c29_tables[1, 41:50]) + sum(cm_c29_tables[2, 41:50])), 6)
k11c29_tnr <- round(sum(cm_c29_tables[4, 41:50]) /
  (sum(cm_c29_tables[4, 41:50]) + sum(cm_c29_tables[3, 41:50])), 6)
k11c29_truescore <- round((2 * k11c29_tpr * k11c29_tnr) /
  (k11c29_tpr + k11c29_tnr), 6)

k13c29_tpr <- round(sum(cm_c29_tables[1, 51:60]) /
  (sum(cm_c29_tables[1, 51:60]) + sum(cm_c29_tables[2, 51:60])), 6)
k13c29_tnr <- round(sum(cm_c29_tables[4, 51:60]) /
  (sum(cm_c29_tables[4, 51:60]) + sum(cm_c29_tables[3, 51:60])), 6)
k13c29_truescore <- round((2 * k13c29_tpr * k13c29_tnr) /
  (k13c29_tpr + k13c29_tnr), 6)

k15c29_tpr <- round(sum(cm_c29_tables[1, 61:70]) /
  (sum(cm_c29_tables[1, 61:70]) + sum(cm_c29_tables[2, 61:70])), 6)
k15c29_tnr <- round(sum(cm_c29_tables[4, 61:70]) /
  (sum(cm_c29_tables[4, 61:70]) + sum(cm_c29_tables[3, 61:70])), 6)
k15c29_truescore <- round((2 * k15c29_tpr * k15c29_tnr) /
  (k15c29_tpr + k15c29_tnr), 6)

k17c29_tpr <- round(sum(cm_c29_tables[1, 71:80]) /
  (sum(cm_c29_tables[1, 71:80]) + sum(cm_c29_tables[2, 71:80])), 6)
k17c29_tnr <- round(sum(cm_c29_tables[4, 71:80]) /
  (sum(cm_c29_tables[4, 71:80]) + sum(cm_c29_tables[3, 71:80])), 6)
k17c29_truescore <- round((2 * k17c29_tpr * k17c29_tnr) /
  (k17c29_tpr + k17c29_tnr), 6)

k19c29_tpr <- round(sum(cm_c29_tables[1, 81:90]) /

```

```

(sum(cm_c29_tables[1, 81:90]) + sum(cm_c29_tables[2, 81:90])), 6)
k19c29_tnr <- round(sum(cm_c29_tables[4, 81:90]) /
  (sum(cm_c29_tables[4, 81:90]) + sum(cm_c29_tables[3, 81:90])), 6)
k19c29_truescore <- round((2 * k19c29_tpr * k19c29_tnr) /
  (k19c29_tpr + k19c29_tnr), 6)

k21c29_tpr <- round(sum(cm_c29_tables[1, 91:100]) /
  (sum(cm_c29_tables[1, 91:100]) + sum(cm_c29_tables[2, 91:100])), 6)
k21c29_tnr <- round(sum(cm_c29_tables[4, 91:100]) /
  (sum(cm_c29_tables[4, 91:100]) + sum(cm_c29_tables[3, 91:100])), 6)
k21c29_truescore <- round((2 * k21c29_tpr * k21c29_tnr) /
  (k21c29_tpr + k21c29_tnr), 6)

k23c29_tpr <- round(sum(cm_c29_tables[1, 101:110]) /
  (sum(cm_c29_tables[1, 101:110]) + sum(cm_c29_tables[2, 101:110])), 6)
k23c29_tnr <- round(sum(cm_c29_tables[4, 101:110]) /
  (sum(cm_c29_tables[4, 101:110]) + sum(cm_c29_tables[3, 101:110])), 6)
k23c29_truescore <- round((2 * k23c29_tpr * k23c29_tnr) /
  (k23c29_tpr + k23c29_tnr), 6)

k25c29_tpr <- round(sum(cm_c29_tables[1, 111:120]) /
  (sum(cm_c29_tables[1, 111:120]) + sum(cm_c29_tables[2, 111:120])), 6)
k25c29_tnr <- round(sum(cm_c29_tables[4, 111:120]) /
  (sum(cm_c29_tables[4, 111:120]) + sum(cm_c29_tables[3, 111:120])), 6)
k25c29_truescore <- round((2 * k25c29_tpr * k25c29_tnr) /
  (k25c29_tpr + k25c29_tnr), 6)

k27c29_tpr <- round(sum(cm_c29_tables[1, 121:130]) /
  (sum(cm_c29_tables[1, 121:130]) + sum(cm_c29_tables[2, 121:130])), 6)
k27c29_tnr <- round(sum(cm_c29_tables[4, 121:130]) /
  (sum(cm_c29_tables[4, 121:130]) + sum(cm_c29_tables[3, 121:130])), 6)
k27c29_truescore <- round((2 * k27c29_tpr * k27c29_tnr) /
  (k27c29_tpr + k27c29_tnr), 6)

k29c29_tpr <- round(sum(cm_c29_tables[1, 131:140]) /
  (sum(cm_c29_tables[1, 131:140]) + sum(cm_c29_tables[2, 131:140])), 6)
k29c29_tnr <- round(sum(cm_c29_tables[4, 131:140]) /
  (sum(cm_c29_tables[4, 131:140]) + sum(cm_c29_tables[3, 131:140])), 6)
k29c29_truescore <- round((2 * k29c29_tpr * k29c29_tnr) /
  (k29c29_tpr + k29c29_tnr), 6)

k31c29_tpr <- round(sum(cm_c29_tables[1, 141:150]) /
  (sum(cm_c29_tables[1, 141:150]) + sum(cm_c29_tables[2, 141:150])), 6)
k31c29_tnr <- round(sum(cm_c29_tables[4, 141:150]) /
  (sum(cm_c29_tables[4, 141:150]) + sum(cm_c29_tables[3, 141:150])), 6)
k31c29_truescore <- round((2 * k31c29_tpr * k31c29_tnr) /
  (k31c29_tpr + k31c29_tnr), 6)

k33c29_tpr <- round(sum(cm_c29_tables[1, 151:160]) /
  (sum(cm_c29_tables[1, 151:160]) + sum(cm_c29_tables[2, 151:160])), 6)
k33c29_tnr <- round(sum(cm_c29_tables[4, 151:160]) /
  (sum(cm_c29_tables[4, 151:160]) + sum(cm_c29_tables[3, 151:160])), 6)
k33c29_truescore <- round((2 * k33c29_tpr * k33c29_tnr) /

```

```

(k33c29_tpr + k33c29_tnr), 6)

k35c29_tpr <- round(sum(cm_c29_tables[1, 161:170]) /
  (sum(cm_c29_tables[1, 161:170]) + sum(cm_c29_tables[2, 161:170])), 6)
k35c29_tnr <- round(sum(cm_c29_tables[4, 161:170]) /
  (sum(cm_c29_tables[4, 161:170]) + sum(cm_c29_tables[3, 161:170])), 6)
k35c29_truescore <- round((2 * k35c29_tpr * k35c29_tnr) /
  (k35c29_tpr + k35c29_tnr), 6)

k37c29_tpr <- round(sum(cm_c29_tables[1, 171:180]) /
  (sum(cm_c29_tables[1, 171:180]) + sum(cm_c29_tables[2, 171:180])), 6)
k37c29_tnr <- round(sum(cm_c29_tables[4, 171:180]) /
  (sum(cm_c29_tables[4, 171:180]) + sum(cm_c29_tables[3, 171:180])), 6)
k37c29_truescore <- round((2 * k37c29_tpr * k37c29_tnr) /
  (k37c29_tpr + k37c29_tnr), 6)

k39c29_tpr <- round(sum(cm_c29_tables[1, 181:190]) /
  (sum(cm_c29_tables[1, 181:190]) + sum(cm_c29_tables[2, 181:190])), 6)
k39c29_tnr <- round(sum(cm_c29_tables[4, 181:190]) /
  (sum(cm_c29_tables[4, 181:190]) + sum(cm_c29_tables[3, 181:190])), 6)
k39c29_truescore <- round((2 * k39c29_tpr * k39c29_tnr) /
  (k39c29_tpr + k39c29_tnr), 6)

# Compile the 0.29 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c29_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.29,
  TPR = c(k3c29_tpr, k5c29_tpr, k7c29_tpr, k9c29_tpr, k11c29_tpr,
    k13c29_tpr, k15c29_tpr, k17c29_tpr, k19c29_tpr, k21c29_tpr,
    k23c29_tpr, k25c29_tpr, k27c29_tpr, k29c29_tpr, k31c29_tpr,
    k33c29_tpr, k35c29_tpr, k37c29_tpr, k39c29_tpr),
  TNR = c(k3c29_tnr, k5c29_tnr, k7c29_tnr, k9c29_tnr, k11c29_tnr,
    k13c29_tnr, k15c29_tnr, k17c29_tnr, k19c29_tnr, k21c29_tnr,
    k23c29_tnr, k25c29_tnr, k27c29_tnr, k29c29_tnr, k31c29_tnr,
    k33c29_tnr, k35c29_tnr, k37c29_tnr, k39c29_tnr),
  Truescore = c(k3c29_truescore, k5c29_truescore, k7c29_truescore,
    k9c29_truescore, k11c29_truescore, k13c29_truescore,
    k15c29_truescore, k17c29_truescore, k19c29_truescore,
    k21c29_truescore, k23c29_truescore, k25c29_truescore,
    k27c29_truescore, k29c29_truescore, k31c29_truescore,
    k33c29_truescore, k35c29_truescore, k37c29_truescore,
    k39c29_truescore))

knitr::kable(c29_results[1:19, ], caption = "c29_results")

```

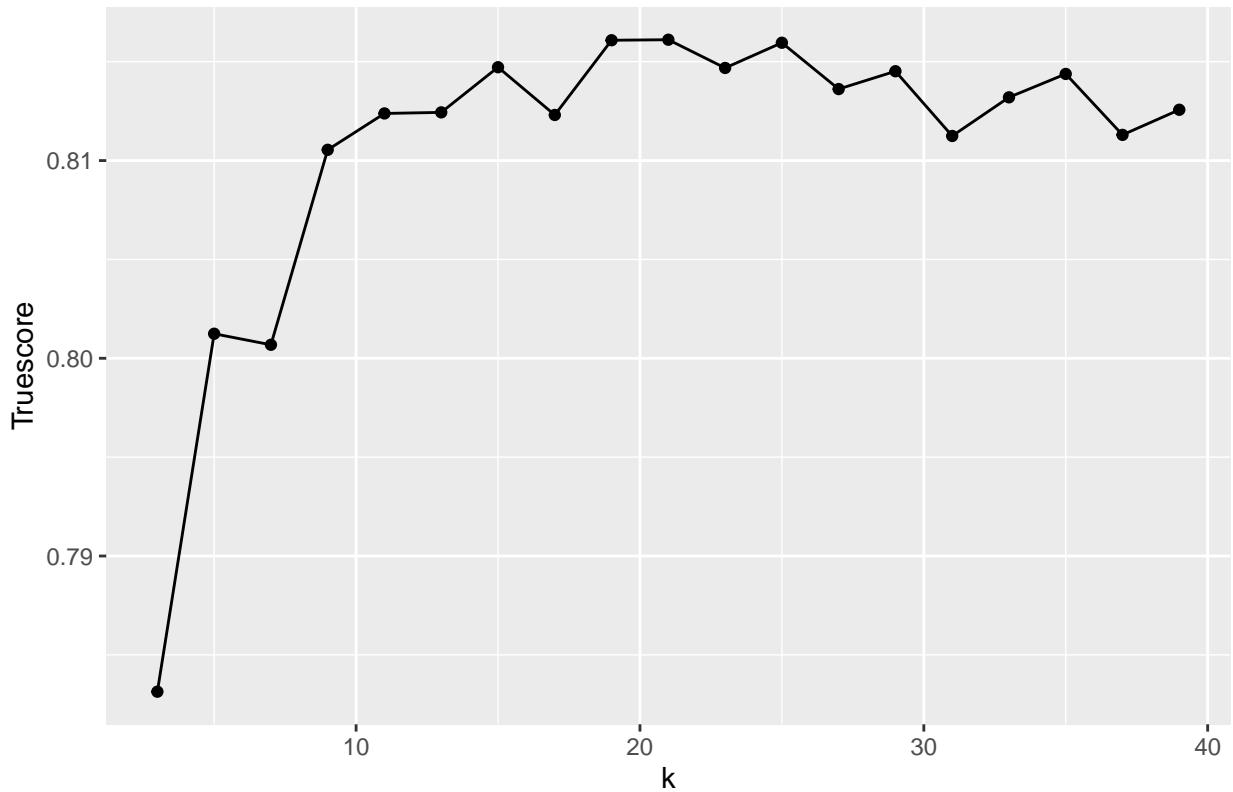
Table 12: c29_results

k	Cut	TPR	TNR	Truescore
3	0.29	0.853815	0.723268	0.783138
5	0.29	0.794629	0.807964	0.801241
7	0.29	0.764508	0.840452	0.800683
9	0.29	0.834438	0.787979	0.810543

k	Cut	TPR	TNR	Truescore
11	0.29	0.809588	0.815198	0.812383
13	0.29	0.851205	0.777046	0.812437
15	0.29	0.830773	0.799277	0.814721
17	0.29	0.860592	0.769151	0.812306
19	0.29	0.845833	0.788359	0.816085
21	0.29	0.831225	0.801539	0.816112
23	0.29	0.854418	0.778482	0.814684
25	0.29	0.841717	0.791729	0.815958
27	0.29	0.859739	0.772190	0.813616
29	0.29	0.847992	0.783586	0.814518
31	0.29	0.862801	0.765501	0.811244
33	0.29	0.854317	0.775856	0.813198
35	0.29	0.845733	0.785271	0.814381
37	0.29	0.857329	0.769960	0.811299
39	0.29	0.850251	0.778086	0.812569

```
ggplot(c29_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.29")
```

Optimal k for Decision Cutoff 0.29



```
#####
# 0.30 Cutoff
#####
```

```

# For the decision cutoff of 0.30, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c30 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred30, obs))
  confusionMatrix(ss$pred30, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c30) <- fk_dfs_v

cm_c30_tables <- sapply(cm_c30, "[[", 2)
cm_c30_tables <- as_tibble(cm_c30_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.30.

k3c30_tpr <- round(sum(cm_c30_tables[1, 1:10]) /
  (sum(cm_c30_tables[1, 1:10]) + sum(cm_c30_tables[2, 1:10])), 6)
k3c30_tnr <- round(sum(cm_c30_tables[4, 1:10]) /
  (sum(cm_c30_tables[4, 1:10]) + sum(cm_c30_tables[3, 1:10])), 6)
k3c30_truescore <- round((2 * k3c30_tpr * k3c30_tnr) /
  (k3c30_tpr + k3c30_tnr), 6)

k5c30_tpr <- round(sum(cm_c30_tables[1, 11:20]) /
  (sum(cm_c30_tables[1, 11:20]) + sum(cm_c30_tables[2, 11:20])), 6)
k5c30_tnr <- round(sum(cm_c30_tables[4, 11:20]) /
  (sum(cm_c30_tables[4, 11:20]) + sum(cm_c30_tables[3, 11:20])), 6)
k5c30_truescore <- round((2 * k5c30_tpr * k5c30_tnr) /
  (k5c30_tpr + k5c30_tnr), 6)

k7c30_tpr <- round(sum(cm_c30_tables[1, 21:30]) /
  (sum(cm_c30_tables[1, 21:30]) + sum(cm_c30_tables[2, 21:30])), 6)
k7c30_tnr <- round(sum(cm_c30_tables[4, 21:30]) /
  (sum(cm_c30_tables[4, 21:30]) + sum(cm_c30_tables[3, 21:30])), 6)
k7c30_truescore <- round((2 * k7c30_tpr * k7c30_tnr) /
  (k7c30_tpr + k7c30_tnr), 6)

k9c30_tpr <- round(sum(cm_c30_tables[1, 31:40]) /
  (sum(cm_c30_tables[1, 31:40]) + sum(cm_c30_tables[2, 31:40])), 6)
k9c30_tnr <- round(sum(cm_c30_tables[4, 31:40]) /
  (sum(cm_c30_tables[4, 31:40]) + sum(cm_c30_tables[3, 31:40])), 6)
k9c30_truescore <- round((2 * k9c30_tpr * k9c30_tnr) /
  (k9c30_tpr + k9c30_tnr), 6)

k11c30_tpr <- round(sum(cm_c30_tables[1, 41:50]) /
  (sum(cm_c30_tables[1, 41:50]) + sum(cm_c30_tables[2, 41:50])), 6)
k11c30_tnr <- round(sum(cm_c30_tables[4, 41:50]) /
  (sum(cm_c30_tables[4, 41:50]) + sum(cm_c30_tables[3, 41:50])), 6)
k11c30_truescore <- round((2 * k11c30_tpr * k11c30_tnr) /
  (k11c30_tpr + k11c30_tnr), 6)

```

```

k13c30_tpr <- round(sum(cm_c30_tables[1, 51:60]) /
  (sum(cm_c30_tables[1, 51:60]) + sum(cm_c30_tables[2, 51:60])), 6)
k13c30_tnr <- round(sum(cm_c30_tables[4, 51:60]) /
  (sum(cm_c30_tables[4, 51:60]) + sum(cm_c30_tables[3, 51:60])), 6)
k13c30_truescore <- round((2 * k13c30_tpr * k13c30_tnr) /
  (k13c30_tpr + k13c30_tnr), 6)

k15c30_tpr <- round(sum(cm_c30_tables[1, 61:70]) /
  (sum(cm_c30_tables[1, 61:70]) + sum(cm_c30_tables[2, 61:70])), 6)
k15c30_tnr <- round(sum(cm_c30_tables[4, 61:70]) /
  (sum(cm_c30_tables[4, 61:70]) + sum(cm_c30_tables[3, 61:70])), 6)
k15c30_truescore <- round((2 * k15c30_tpr * k15c30_tnr) /
  (k15c30_tpr + k15c30_tnr), 6)

k17c30_tpr <- round(sum(cm_c30_tables[1, 71:80]) /
  (sum(cm_c30_tables[1, 71:80]) + sum(cm_c30_tables[2, 71:80])), 6)
k17c30_tnr <- round(sum(cm_c30_tables[4, 71:80]) /
  (sum(cm_c30_tables[4, 71:80]) + sum(cm_c30_tables[3, 71:80])), 6)
k17c30_truescore <- round((2 * k17c30_tpr * k17c30_tnr) /
  (k17c30_tpr + k17c30_tnr), 6)

k19c30_tpr <- round(sum(cm_c30_tables[1, 81:90]) /
  (sum(cm_c30_tables[1, 81:90]) + sum(cm_c30_tables[2, 81:90])), 6)
k19c30_tnr <- round(sum(cm_c30_tables[4, 81:90]) /
  (sum(cm_c30_tables[4, 81:90]) + sum(cm_c30_tables[3, 81:90])), 6)
k19c30_truescore <- round((2 * k19c30_tpr * k19c30_tnr) /
  (k19c30_tpr + k19c30_tnr), 6)

k21c30_tpr <- round(sum(cm_c30_tables[1, 91:100]) /
  (sum(cm_c30_tables[1, 91:100]) + sum(cm_c30_tables[2, 91:100])), 6)
k21c30_tnr <- round(sum(cm_c30_tables[4, 91:100]) /
  (sum(cm_c30_tables[4, 91:100]) + sum(cm_c30_tables[3, 91:100])), 6)
k21c30_truescore <- round((2 * k21c30_tpr * k21c30_tnr) /
  (k21c30_tpr + k21c30_tnr), 6)

k23c30_tpr <- round(sum(cm_c30_tables[1, 101:110]) /
  (sum(cm_c30_tables[1, 101:110]) + sum(cm_c30_tables[2, 101:110])), 6)
k23c30_tnr <- round(sum(cm_c30_tables[4, 101:110]) /
  (sum(cm_c30_tables[4, 101:110]) + sum(cm_c30_tables[3, 101:110])), 6)
k23c30_truescore <- round((2 * k23c30_tpr * k23c30_tnr) /
  (k23c30_tpr + k23c30_tnr), 6)

k25c30_tpr <- round(sum(cm_c30_tables[1, 111:120]) /
  (sum(cm_c30_tables[1, 111:120]) + sum(cm_c30_tables[2, 111:120])), 6)
k25c30_tnr <- round(sum(cm_c30_tables[4, 111:120]) /
  (sum(cm_c30_tables[4, 111:120]) + sum(cm_c30_tables[3, 111:120])), 6)
k25c30_truescore <- round((2 * k25c30_tpr * k25c30_tnr) /
  (k25c30_tpr + k25c30_tnr), 6)

k27c30_tpr <- round(sum(cm_c30_tables[1, 121:130]) /
  (sum(cm_c30_tables[1, 121:130]) + sum(cm_c30_tables[2, 121:130])), 6)
k27c30_tnr <- round(sum(cm_c30_tables[4, 121:130]) /
  (sum(cm_c30_tables[4, 121:130]) + sum(cm_c30_tables[3, 121:130])), 6)

```

```

k27c30_truescore <- round((2 * k27c30_tpr * k27c30_tnr) /
  (k27c30_tpr + k27c30_tnr), 6)

k29c30_tpr <- round(sum(cm_c30_tables[1, 131:140]) /
  (sum(cm_c30_tables[1, 131:140]) + sum(cm_c30_tables[2, 131:140])), 6)
k29c30_tnr <- round(sum(cm_c30_tables[4, 131:140]) /
  (sum(cm_c30_tables[4, 131:140]) + sum(cm_c30_tables[3, 131:140])), 6)
k29c30_truescore <- round((2 * k29c30_tpr * k29c30_tnr) /
  (k29c30_tpr + k29c30_tnr), 6)

k31c30_tpr <- round(sum(cm_c30_tables[1, 141:150]) /
  (sum(cm_c30_tables[1, 141:150]) + sum(cm_c30_tables[2, 141:150])), 6)
k31c30_tnr <- round(sum(cm_c30_tables[4, 141:150]) /
  (sum(cm_c30_tables[4, 141:150]) + sum(cm_c30_tables[3, 141:150])), 6)
k31c30_truescore <- round((2 * k31c30_tpr * k31c30_tnr) /
  (k31c30_tpr + k31c30_tnr), 6)

k33c30_tpr <- round(sum(cm_c30_tables[1, 151:160]) /
  (sum(cm_c30_tables[1, 151:160]) + sum(cm_c30_tables[2, 151:160])), 6)
k33c30_tnr <- round(sum(cm_c30_tables[4, 151:160]) /
  (sum(cm_c30_tables[4, 151:160]) + sum(cm_c30_tables[3, 151:160])), 6)
k33c30_truescore <- round((2 * k33c30_tpr * k33c30_tnr) /
  (k33c30_tpr + k33c30_tnr), 6)

k35c30_tpr <- round(sum(cm_c30_tables[1, 161:170]) /
  (sum(cm_c30_tables[1, 161:170]) + sum(cm_c30_tables[2, 161:170])), 6)
k35c30_tnr <- round(sum(cm_c30_tables[4, 161:170]) /
  (sum(cm_c30_tables[4, 161:170]) + sum(cm_c30_tables[3, 161:170])), 6)
k35c30_truescore <- round((2 * k35c30_tpr * k35c30_tnr) /
  (k35c30_tpr + k35c30_tnr), 6)

k37c30_tpr <- round(sum(cm_c30_tables[1, 171:180]) /
  (sum(cm_c30_tables[1, 171:180]) + sum(cm_c30_tables[2, 171:180])), 6)
k37c30_tnr <- round(sum(cm_c30_tables[4, 171:180]) /
  (sum(cm_c30_tables[4, 171:180]) + sum(cm_c30_tables[3, 171:180])), 6)
k37c30_truescore <- round((2 * k37c30_tpr * k37c30_tnr) /
  (k37c30_tpr + k37c30_tnr), 6)

k39c30_tpr <- round(sum(cm_c30_tables[1, 181:190]) /
  (sum(cm_c30_tables[1, 181:190]) + sum(cm_c30_tables[2, 181:190])), 6)
k39c30_tnr <- round(sum(cm_c30_tables[4, 181:190]) /
  (sum(cm_c30_tables[4, 181:190]) + sum(cm_c30_tables[3, 181:190])), 6)
k39c30_truescore <- round((2 * k39c30_tpr * k39c30_tnr) /
  (k39c30_tpr + k39c30_tnr), 6)

# Compile the 0.30 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c30_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.30,
  TPR = c(k3c30_tpr, k5c30_tpr, k7c30_tpr, k9c30_tpr, k11c30_tpr,
    k13c30_tpr, k15c30_tpr, k17c30_tpr, k19c30_tpr, k21c30_tpr,
    k23c30_tpr, k25c30_tpr, k27c30_tpr, k29c30_tpr, k31c30_tpr,
    k33c30_tpr, k35c30_tpr, k37c30_tpr, k39c30_tpr))

```

```

        k33c30_tpr, k35c30_tpr, k37c30_tpr, k39c30_tpr),
TNR = c(k3c30_tnr, k5c30_tnr, k7c30_tnr, k9c30_tnr, k11c30_tnr,
         k13c30_tnr, k15c30_tnr, k17c30_tnr, k19c30_tnr, k21c30_tnr,
         k23c30_tnr, k25c30_tnr, k27c30_tnr, k29c30_tnr, k31c30_tnr,
         k33c30_tnr, k35c30_tnr, k37c30_tnr, k39c30_tnr),
Truescore = c(k3c30_truescore, k5c30_truescore, k7c30_truescore,
              k9c30_truescore, k11c30_truescore, k13c30_truescore,
              k15c30_truescore, k17c30_truescore, k19c30_truescore,
              k21c30_truescore, k23c30_truescore, k25c30_truescore,
              k27c30_truescore, k29c30_truescore, k31c30_truescore,
              k33c30_truescore, k35c30_truescore, k37c30_truescore,
              k39c30_truescore))

knitr::kable(c30_results[1:19, ], caption = "c30_results")

```

Table 13: c30_results

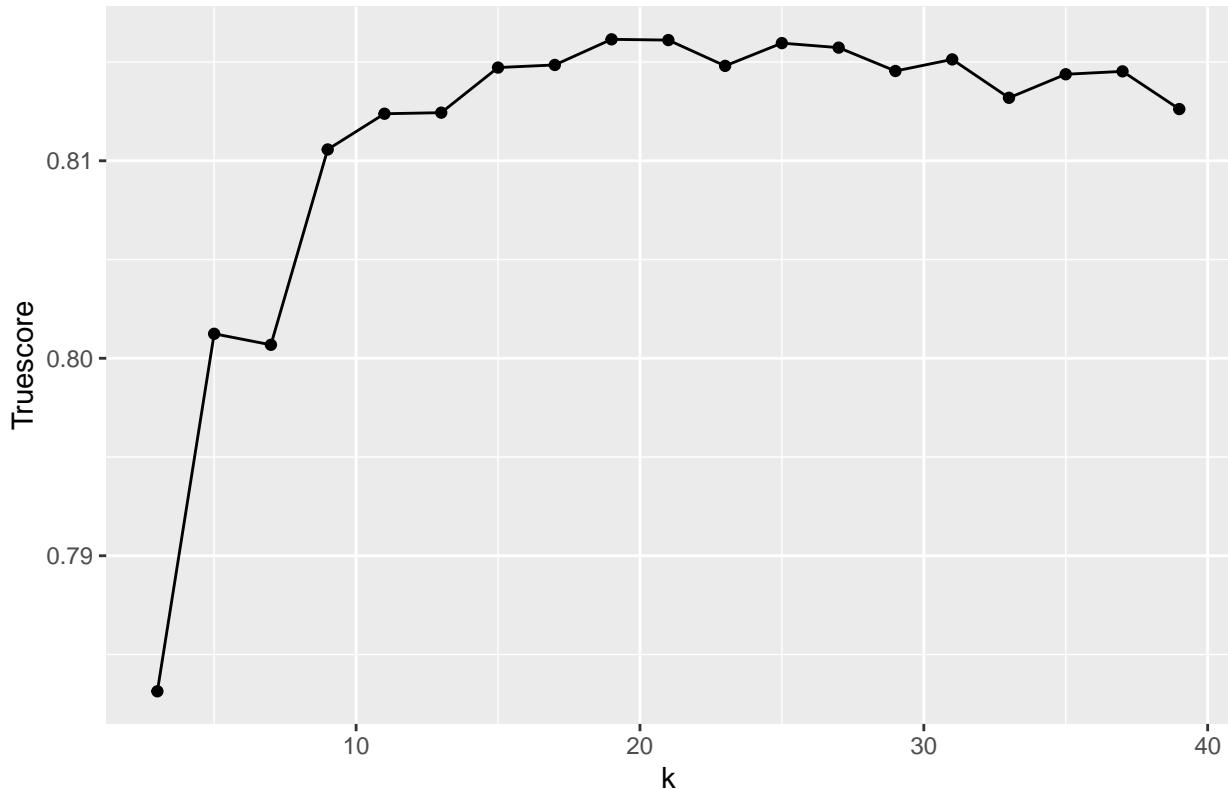
k	Cut	TPR	TNR	Truescore
3	0.3	0.853815	0.723268	0.783138
5	0.3	0.794629	0.807964	0.801241
7	0.3	0.764508	0.840452	0.800683
9	0.3	0.834337	0.788128	0.810574
11	0.3	0.809588	0.815198	0.812383
13	0.3	0.851205	0.777046	0.812437
15	0.3	0.830773	0.799277	0.814721
17	0.3	0.814458	0.815248	0.814853
19	0.3	0.845783	0.788524	0.816150
21	0.3	0.831225	0.801539	0.816112
23	0.3	0.854367	0.778747	0.814806
25	0.3	0.841717	0.791729	0.815958
27	0.3	0.829618	0.802299	0.815730
29	0.3	0.847841	0.783768	0.814546
31	0.3	0.838203	0.793298	0.815133
33	0.3	0.854016	0.776088	0.813189
35	0.3	0.845733	0.785271	0.814381
37	0.3	0.836797	0.793413	0.814528
39	0.3	0.850201	0.778218	0.812619

```

ggplot(c30_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.30")

```

Optimal k for Decision Cutoff 0.30



```
#####
# 0.31 Cutoff
#####

# For the decision cutoff of 0.31, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c31 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred31, obs))
  confusionMatrix(ss$pred31, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c31) <- fk_dfs_v

cm_c31_tables <- sapply(cm_c31, "[[", 2)
cm_c31_tables <- as_tibble(cm_c31_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.31.

k3c31_tpr <- round(sum(cm_c31_tables[1, 1:10]) /
  (sum(cm_c31_tables[1, 1:10]) + sum(cm_c31_tables[2, 1:10])), 6)
k3c31_tnr <- round(sum(cm_c31_tables[4, 1:10]) /
  (sum(cm_c31_tables[4, 1:10]) + sum(cm_c31_tables[3, 1:10])), 6)
```

```

k3c31_truescore <- round((2 * k3c31_tpr * k3c31_tnr) /
  (k3c31_tpr + k3c31_tnr), 6)

k5c31_tpr <- round(sum(cm_c31_tables[1, 11:20]) /
  (sum(cm_c31_tables[1, 11:20]) + sum(cm_c31_tables[2, 11:20])), 6)
k5c31_tnr <- round(sum(cm_c31_tables[4, 11:20]) /
  (sum(cm_c31_tables[4, 11:20]) + sum(cm_c31_tables[3, 11:20])), 6)
k5c31_truescore <- round((2 * k5c31_tpr * k5c31_tnr) /
  (k5c31_tpr + k5c31_tnr), 6)

k7c31_tpr <- round(sum(cm_c31_tables[1, 21:30]) /
  (sum(cm_c31_tables[1, 21:30]) + sum(cm_c31_tables[2, 21:30])), 6)
k7c31_tnr <- round(sum(cm_c31_tables[4, 21:30]) /
  (sum(cm_c31_tables[4, 21:30]) + sum(cm_c31_tables[3, 21:30])), 6)
k7c31_truescore <- round((2 * k7c31_tpr * k7c31_tnr) /
  (k7c31_tpr + k7c31_tnr), 6)

k9c31_tpr <- round(sum(cm_c31_tables[1, 31:40]) /
  (sum(cm_c31_tables[1, 31:40]) + sum(cm_c31_tables[2, 31:40])), 6)
k9c31_tnr <- round(sum(cm_c31_tables[4, 31:40]) /
  (sum(cm_c31_tables[4, 31:40]) + sum(cm_c31_tables[3, 31:40])), 6)
k9c31_truescore <- round((2 * k9c31_tpr * k9c31_tnr) /
  (k9c31_tpr + k9c31_tnr), 6)

k11c31_tpr <- round(sum(cm_c31_tables[1, 41:50]) /
  (sum(cm_c31_tables[1, 41:50]) + sum(cm_c31_tables[2, 41:50])), 6)
k11c31_tnr <- round(sum(cm_c31_tables[4, 41:50]) /
  (sum(cm_c31_tables[4, 41:50]) + sum(cm_c31_tables[3, 41:50])), 6)
k11c31_truescore <- round((2 * k11c31_tpr * k11c31_tnr) /
  (k11c31_tpr + k11c31_tnr), 6)

k13c31_tpr <- round(sum(cm_c31_tables[1, 51:60]) /
  (sum(cm_c31_tables[1, 51:60]) + sum(cm_c31_tables[2, 51:60])), 6)
k13c31_tnr <- round(sum(cm_c31_tables[4, 51:60]) /
  (sum(cm_c31_tables[4, 51:60]) + sum(cm_c31_tables[3, 51:60])), 6)
k13c31_truescore <- round((2 * k13c31_tpr * k13c31_tnr) /
  (k13c31_tpr + k13c31_tnr), 6)

k15c31_tpr <- round(sum(cm_c31_tables[1, 61:70]) /
  (sum(cm_c31_tables[1, 61:70]) + sum(cm_c31_tables[2, 61:70])), 6)
k15c31_tnr <- round(sum(cm_c31_tables[4, 61:70]) /
  (sum(cm_c31_tables[4, 61:70]) + sum(cm_c31_tables[3, 61:70])), 6)
k15c31_truescore <- round((2 * k15c31_tpr * k15c31_tnr) /
  (k15c31_tpr + k15c31_tnr), 6)

k17c31_tpr <- round(sum(cm_c31_tables[1, 71:80]) /
  (sum(cm_c31_tables[1, 71:80]) + sum(cm_c31_tables[2, 71:80])), 6)
k17c31_tnr <- round(sum(cm_c31_tables[4, 71:80]) /
  (sum(cm_c31_tables[4, 71:80]) + sum(cm_c31_tables[3, 71:80])), 6)
k17c31_truescore <- round((2 * k17c31_tpr * k17c31_tnr) /
  (k17c31_tpr + k17c31_tnr), 6)

k19c31_tpr <- round(sum(cm_c31_tables[1, 81:90]) /

```

```

(sum(cm_c31_tables[1, 81:90]) + sum(cm_c31_tables[2, 81:90])), 6)
k19c31_tnr <- round(sum(cm_c31_tables[4, 81:90]) /
  (sum(cm_c31_tables[4, 81:90]) + sum(cm_c31_tables[3, 81:90])), 6)
k19c31_truescore <- round((2 * k19c31_tpr * k19c31_tnr) /
  (k19c31_tpr + k19c31_tnr), 6)

k21c31_tpr <- round(sum(cm_c31_tables[1, 91:100]) /
  (sum(cm_c31_tables[1, 91:100]) + sum(cm_c31_tables[2, 91:100])), 6)
k21c31_tnr <- round(sum(cm_c31_tables[4, 91:100]) /
  (sum(cm_c31_tables[4, 91:100]) + sum(cm_c31_tables[3, 91:100])), 6)
k21c31_truescore <- round((2 * k21c31_tpr * k21c31_tnr) /
  (k21c31_tpr + k21c31_tnr), 6)

k23c31_tpr <- round(sum(cm_c31_tables[1, 101:110]) /
  (sum(cm_c31_tables[1, 101:110]) + sum(cm_c31_tables[2, 101:110])), 6)
k23c31_tnr <- round(sum(cm_c31_tables[4, 101:110]) /
  (sum(cm_c31_tables[4, 101:110]) + sum(cm_c31_tables[3, 101:110])), 6)
k23c31_truescore <- round((2 * k23c31_tpr * k23c31_tnr) /
  (k23c31_tpr + k23c31_tnr), 6)

k25c31_tpr <- round(sum(cm_c31_tables[1, 111:120]) /
  (sum(cm_c31_tables[1, 111:120]) + sum(cm_c31_tables[2, 111:120])), 6)
k25c31_tnr <- round(sum(cm_c31_tables[4, 111:120]) /
  (sum(cm_c31_tables[4, 111:120]) + sum(cm_c31_tables[3, 111:120])), 6)
k25c31_truescore <- round((2 * k25c31_tpr * k25c31_tnr) /
  (k25c31_tpr + k25c31_tnr), 6)

k27c31_tpr <- round(sum(cm_c31_tables[1, 121:130]) /
  (sum(cm_c31_tables[1, 121:130]) + sum(cm_c31_tables[2, 121:130])), 6)
k27c31_tnr <- round(sum(cm_c31_tables[4, 121:130]) /
  (sum(cm_c31_tables[4, 121:130]) + sum(cm_c31_tables[3, 121:130])), 6)
k27c31_truescore <- round((2 * k27c31_tpr * k27c31_tnr) /
  (k27c31_tpr + k27c31_tnr), 6)

k29c31_tpr <- round(sum(cm_c31_tables[1, 131:140]) /
  (sum(cm_c31_tables[1, 131:140]) + sum(cm_c31_tables[2, 131:140])), 6)
k29c31_tnr <- round(sum(cm_c31_tables[4, 131:140]) /
  (sum(cm_c31_tables[4, 131:140]) + sum(cm_c31_tables[3, 131:140])), 6)
k29c31_truescore <- round((2 * k29c31_tpr * k29c31_tnr) /
  (k29c31_tpr + k29c31_tnr), 6)

k31c31_tpr <- round(sum(cm_c31_tables[1, 141:150]) /
  (sum(cm_c31_tables[1, 141:150]) + sum(cm_c31_tables[2, 141:150])), 6)
k31c31_tnr <- round(sum(cm_c31_tables[4, 141:150]) /
  (sum(cm_c31_tables[4, 141:150]) + sum(cm_c31_tables[3, 141:150])), 6)
k31c31_truescore <- round((2 * k31c31_tpr * k31c31_tnr) /
  (k31c31_tpr + k31c31_tnr), 6)

k33c31_tpr <- round(sum(cm_c31_tables[1, 151:160]) /
  (sum(cm_c31_tables[1, 151:160]) + sum(cm_c31_tables[2, 151:160])), 6)
k33c31_tnr <- round(sum(cm_c31_tables[4, 151:160]) /
  (sum(cm_c31_tables[4, 151:160]) + sum(cm_c31_tables[3, 151:160])), 6)
k33c31_truescore <- round((2 * k33c31_tpr * k33c31_tnr) /

```

```

(k33c31_tpr + k33c31_tnr), 6)

k35c31_tpr <- round(sum(cm_c31_tables[1, 161:170]) /
  (sum(cm_c31_tables[1, 161:170]) + sum(cm_c31_tables[2, 161:170])), 6)
k35c31_tnr <- round(sum(cm_c31_tables[4, 161:170]) /
  (sum(cm_c31_tables[4, 161:170]) + sum(cm_c31_tables[3, 161:170])), 6)
k35c31_truescore <- round((2 * k35c31_tpr * k35c31_tnr) /
  (k35c31_tpr + k35c31_tnr), 6)

k37c31_tpr <- round(sum(cm_c31_tables[1, 171:180]) /
  (sum(cm_c31_tables[1, 171:180]) + sum(cm_c31_tables[2, 171:180])), 6)
k37c31_tnr <- round(sum(cm_c31_tables[4, 171:180]) /
  (sum(cm_c31_tables[4, 171:180]) + sum(cm_c31_tables[3, 171:180])), 6)
k37c31_truescore <- round((2 * k37c31_tpr * k37c31_tnr) /
  (k37c31_tpr + k37c31_tnr), 6)

k39c31_tpr <- round(sum(cm_c31_tables[1, 181:190]) /
  (sum(cm_c31_tables[1, 181:190]) + sum(cm_c31_tables[2, 181:190])), 6)
k39c31_tnr <- round(sum(cm_c31_tables[4, 181:190]) /
  (sum(cm_c31_tables[4, 181:190]) + sum(cm_c31_tables[3, 181:190])), 6)
k39c31_truescore <- round((2 * k39c31_tpr * k39c31_tnr) /
  (k39c31_tpr + k39c31_tnr), 6)

# Compile the 0.31 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c31_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.31,
  TPR = c(k3c31_tpr, k5c31_tpr, k7c31_tpr, k9c31_tpr, k11c31_tpr,
    k13c31_tpr, k15c31_tpr, k17c31_tpr, k19c31_tpr, k21c31_tpr,
    k23c31_tpr, k25c31_tpr, k27c31_tpr, k29c31_tpr, k31c31_tpr,
    k33c31_tpr, k35c31_tpr, k37c31_tpr, k39c31_tpr),
  TNR = c(k3c31_tnr, k5c31_tnr, k7c31_tnr, k9c31_tnr, k11c31_tnr,
    k13c31_tnr, k15c31_tnr, k17c31_tnr, k19c31_tnr, k21c31_tnr,
    k23c31_tnr, k25c31_tnr, k27c31_tnr, k29c31_tnr, k31c31_tnr,
    k33c31_tnr, k35c31_tnr, k37c31_tnr, k39c31_tnr),
  Truescore = c(k3c31_truescore, k5c31_truescore, k7c31_truescore,
    k9c31_truescore, k11c31_truescore, k13c31_truescore,
    k15c31_truescore, k17c31_truescore, k19c31_truescore,
    k21c31_truescore, k23c31_truescore, k25c31_truescore,
    k27c31_truescore, k29c31_truescore, k31c31_truescore,
    k33c31_truescore, k35c31_truescore, k37c31_truescore,
    k39c31_truescore))

knitr::kable(c31_results[1:19, ], caption = "c31_results")

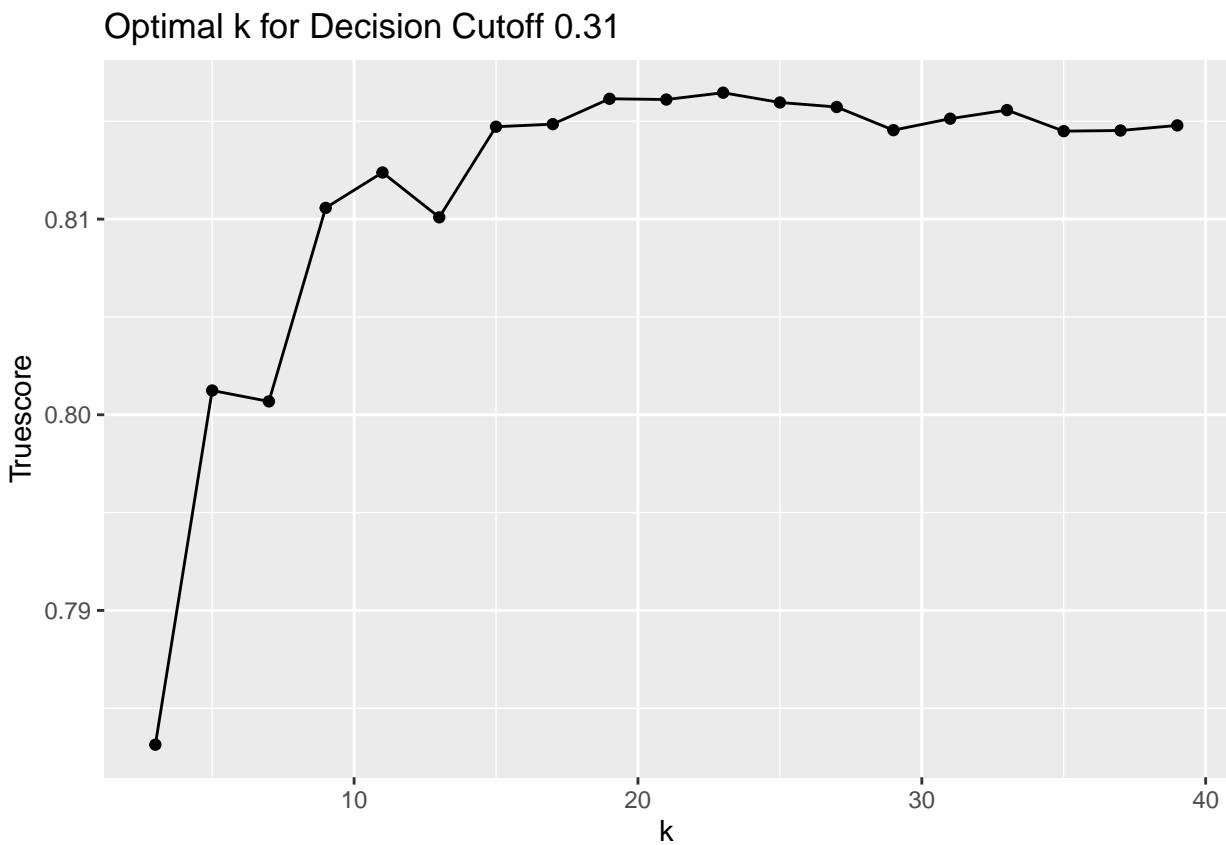
```

Table 14: c31_results

k	Cut	TPR	TNR	Truescore
3	0.31	0.853815	0.723268	0.783138
5	0.31	0.794629	0.807964	0.801241
7	0.31	0.764508	0.840452	0.800683
9	0.31	0.834337	0.788128	0.810574

k	Cut	TPR	TNR	Truescore
11	0.31	0.809588	0.815198	0.812383
13	0.31	0.788655	0.832722	0.810090
15	0.31	0.830773	0.799277	0.814721
17	0.31	0.814458	0.815248	0.814853
19	0.31	0.845783	0.788524	0.816150
21	0.31	0.831225	0.801539	0.816112
23	0.31	0.819478	0.813464	0.816460
25	0.31	0.841667	0.791778	0.815961
27	0.31	0.829618	0.802299	0.815730
29	0.31	0.847841	0.783768	0.814546
31	0.31	0.838203	0.793298	0.815133
33	0.31	0.829468	0.802134	0.815572
35	0.31	0.845633	0.785568	0.814495
37	0.31	0.836797	0.793413	0.814528
39	0.31	0.829367	0.800714	0.814789

```
ggplot(c31_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.31")
```



```
#####
# 0.32 Cutoff
#####
```

```

# For the decision cutoff of 0.32, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c32 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred32, obs))
  confusionMatrix(ss$pred32, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c32) <- fk_dfs_v

cm_c32_tables <- sapply(cm_c32, "[[", 2)
cm_c32_tables <- as_tibble(cm_c32_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.32.

k3c32_tpr <- round(sum(cm_c32_tables[1, 1:10]) /
  (sum(cm_c32_tables[1, 1:10]) + sum(cm_c32_tables[2, 1:10])), 6)
k3c32_tnr <- round(sum(cm_c32_tables[4, 1:10]) /
  (sum(cm_c32_tables[4, 1:10]) + sum(cm_c32_tables[3, 1:10])), 6)
k3c32_truescore <- round((2 * k3c32_tpr * k3c32_tnr) /
  (k3c32_tpr + k3c32_tnr), 6)

k5c32_tpr <- round(sum(cm_c32_tables[1, 11:20]) /
  (sum(cm_c32_tables[1, 11:20]) + sum(cm_c32_tables[2, 11:20])), 6)
k5c32_tnr <- round(sum(cm_c32_tables[4, 11:20]) /
  (sum(cm_c32_tables[4, 11:20]) + sum(cm_c32_tables[3, 11:20])), 6)
k5c32_truescore <- round((2 * k5c32_tpr * k5c32_tnr) /
  (k5c32_tpr + k5c32_tnr), 6)

k7c32_tpr <- round(sum(cm_c32_tables[1, 21:30]) /
  (sum(cm_c32_tables[1, 21:30]) + sum(cm_c32_tables[2, 21:30])), 6)
k7c32_tnr <- round(sum(cm_c32_tables[4, 21:30]) /
  (sum(cm_c32_tables[4, 21:30]) + sum(cm_c32_tables[3, 21:30])), 6)
k7c32_truescore <- round((2 * k7c32_tpr * k7c32_tnr) /
  (k7c32_tpr + k7c32_tnr), 6)

k9c32_tpr <- round(sum(cm_c32_tables[1, 31:40]) /
  (sum(cm_c32_tables[1, 31:40]) + sum(cm_c32_tables[2, 31:40])), 6)
k9c32_tnr <- round(sum(cm_c32_tables[4, 31:40]) /
  (sum(cm_c32_tables[4, 31:40]) + sum(cm_c32_tables[3, 31:40])), 6)
k9c32_truescore <- round((2 * k9c32_tpr * k9c32_tnr) /
  (k9c32_tpr + k9c32_tnr), 6)

k11c32_tpr <- round(sum(cm_c32_tables[1, 41:50]) /
  (sum(cm_c32_tables[1, 41:50]) + sum(cm_c32_tables[2, 41:50])), 6)
k11c32_tnr <- round(sum(cm_c32_tables[4, 41:50]) /
  (sum(cm_c32_tables[4, 41:50]) + sum(cm_c32_tables[3, 41:50])), 6)
k11c32_truescore <- round((2 * k11c32_tpr * k11c32_tnr) /
  (k11c32_tpr + k11c32_tnr), 6)

```

```

k13c32_tpr <- round(sum(cm_c32_tables[1, 51:60]) /
  (sum(cm_c32_tables[1, 51:60]) + sum(cm_c32_tables[2, 51:60])), 6)
k13c32_tnr <- round(sum(cm_c32_tables[4, 51:60]) /
  (sum(cm_c32_tables[4, 51:60]) + sum(cm_c32_tables[3, 51:60])), 6)
k13c32_truescore <- round((2 * k13c32_tpr * k13c32_tnr) /
  (k13c32_tpr + k13c32_tnr), 6)

k15c32_tpr <- round(sum(cm_c32_tables[1, 61:70]) /
  (sum(cm_c32_tables[1, 61:70]) + sum(cm_c32_tables[2, 61:70])), 6)
k15c32_tnr <- round(sum(cm_c32_tables[4, 61:70]) /
  (sum(cm_c32_tables[4, 61:70]) + sum(cm_c32_tables[3, 61:70])), 6)
k15c32_truescore <- round((2 * k15c32_tpr * k15c32_tnr) /
  (k15c32_tpr + k15c32_tnr), 6)

k17c32_tpr <- round(sum(cm_c32_tables[1, 71:80]) /
  (sum(cm_c32_tables[1, 71:80]) + sum(cm_c32_tables[2, 71:80])), 6)
k17c32_tnr <- round(sum(cm_c32_tables[4, 71:80]) /
  (sum(cm_c32_tables[4, 71:80]) + sum(cm_c32_tables[3, 71:80])), 6)
k17c32_truescore <- round((2 * k17c32_tpr * k17c32_tnr) /
  (k17c32_tpr + k17c32_tnr), 6)

k19c32_tpr <- round(sum(cm_c32_tables[1, 81:90]) /
  (sum(cm_c32_tables[1, 81:90]) + sum(cm_c32_tables[2, 81:90])), 6)
k19c32_tnr <- round(sum(cm_c32_tables[4, 81:90]) /
  (sum(cm_c32_tables[4, 81:90]) + sum(cm_c32_tables[3, 81:90])), 6)
k19c32_truescore <- round((2 * k19c32_tpr * k19c32_tnr) /
  (k19c32_tpr + k19c32_tnr), 6)

k21c32_tpr <- round(sum(cm_c32_tables[1, 91:100]) /
  (sum(cm_c32_tables[1, 91:100]) + sum(cm_c32_tables[2, 91:100])), 6)
k21c32_tnr <- round(sum(cm_c32_tables[4, 91:100]) /
  (sum(cm_c32_tables[4, 91:100]) + sum(cm_c32_tables[3, 91:100])), 6)
k21c32_truescore <- round((2 * k21c32_tpr * k21c32_tnr) /
  (k21c32_tpr + k21c32_tnr), 6)

k23c32_tpr <- round(sum(cm_c32_tables[1, 101:110]) /
  (sum(cm_c32_tables[1, 101:110]) + sum(cm_c32_tables[2, 101:110])), 6)
k23c32_tnr <- round(sum(cm_c32_tables[4, 101:110]) /
  (sum(cm_c32_tables[4, 101:110]) + sum(cm_c32_tables[3, 101:110])), 6)
k23c32_truescore <- round((2 * k23c32_tpr * k23c32_tnr) /
  (k23c32_tpr + k23c32_tnr), 6)

k25c32_tpr <- round(sum(cm_c32_tables[1, 111:120]) /
  (sum(cm_c32_tables[1, 111:120]) + sum(cm_c32_tables[2, 111:120])), 6)
k25c32_tnr <- round(sum(cm_c32_tables[4, 111:120]) /
  (sum(cm_c32_tables[4, 111:120]) + sum(cm_c32_tables[3, 111:120])), 6)
k25c32_truescore <- round((2 * k25c32_tpr * k25c32_tnr) /
  (k25c32_tpr + k25c32_tnr), 6)

k27c32_tpr <- round(sum(cm_c32_tables[1, 121:130]) /
  (sum(cm_c32_tables[1, 121:130]) + sum(cm_c32_tables[2, 121:130])), 6)
k27c32_tnr <- round(sum(cm_c32_tables[4, 121:130]) /
  (sum(cm_c32_tables[4, 121:130]) + sum(cm_c32_tables[3, 121:130])), 6)

```

```

k27c32_truescore <- round((2 * k27c32_tpr * k27c32_tnr) /
  (k27c32_tpr + k27c32_tnr), 6)

k29c32_tpr <- round(sum(cm_c32_tables[1, 131:140]) /
  (sum(cm_c32_tables[1, 131:140]) + sum(cm_c32_tables[2, 131:140])), 6)
k29c32_tnr <- round(sum(cm_c32_tables[4, 131:140]) /
  (sum(cm_c32_tables[4, 131:140]) + sum(cm_c32_tables[3, 131:140])), 6)
k29c32_truescore <- round((2 * k29c32_tpr * k29c32_tnr) /
  (k29c32_tpr + k29c32_tnr), 6)

k31c32_tpr <- round(sum(cm_c32_tables[1, 141:150]) /
  (sum(cm_c32_tables[1, 141:150]) + sum(cm_c32_tables[2, 141:150])), 6)
k31c32_tnr <- round(sum(cm_c32_tables[4, 141:150]) /
  (sum(cm_c32_tables[4, 141:150]) + sum(cm_c32_tables[3, 141:150])), 6)
k31c32_truescore <- round((2 * k31c32_tpr * k31c32_tnr) /
  (k31c32_tpr + k31c32_tnr), 6)

k33c32_tpr <- round(sum(cm_c32_tables[1, 151:160]) /
  (sum(cm_c32_tables[1, 151:160]) + sum(cm_c32_tables[2, 151:160])), 6)
k33c32_tnr <- round(sum(cm_c32_tables[4, 151:160]) /
  (sum(cm_c32_tables[4, 151:160]) + sum(cm_c32_tables[3, 151:160])), 6)
k33c32_truescore <- round((2 * k33c32_tpr * k33c32_tnr) /
  (k33c32_tpr + k33c32_tnr), 6)

k35c32_tpr <- round(sum(cm_c32_tables[1, 161:170]) /
  (sum(cm_c32_tables[1, 161:170]) + sum(cm_c32_tables[2, 161:170])), 6)
k35c32_tnr <- round(sum(cm_c32_tables[4, 161:170]) /
  (sum(cm_c32_tables[4, 161:170]) + sum(cm_c32_tables[3, 161:170])), 6)
k35c32_truescore <- round((2 * k35c32_tpr * k35c32_tnr) /
  (k35c32_tpr + k35c32_tnr), 6)

k37c32_tpr <- round(sum(cm_c32_tables[1, 171:180]) /
  (sum(cm_c32_tables[1, 171:180]) + sum(cm_c32_tables[2, 171:180])), 6)
k37c32_tnr <- round(sum(cm_c32_tables[4, 171:180]) /
  (sum(cm_c32_tables[4, 171:180]) + sum(cm_c32_tables[3, 171:180])), 6)
k37c32_truescore <- round((2 * k37c32_tpr * k37c32_tnr) /
  (k37c32_tpr + k37c32_tnr), 6)

k39c32_tpr <- round(sum(cm_c32_tables[1, 181:190]) /
  (sum(cm_c32_tables[1, 181:190]) + sum(cm_c32_tables[2, 181:190])), 6)
k39c32_tnr <- round(sum(cm_c32_tables[4, 181:190]) /
  (sum(cm_c32_tables[4, 181:190]) + sum(cm_c32_tables[3, 181:190])), 6)
k39c32_truescore <- round((2 * k39c32_tpr * k39c32_tnr) /
  (k39c32_tpr + k39c32_tnr), 6)

# Compile the 0.32 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c32_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.32,
  TPR = c(k3c32_tpr, k5c32_tpr, k7c32_tpr, k9c32_tpr, k11c32_tpr,
    k13c32_tpr, k15c32_tpr, k17c32_tpr, k19c32_tpr, k21c32_tpr,
    k23c32_tpr, k25c32_tpr, k27c32_tpr, k29c32_tpr, k31c32_tpr,
    k33c32_tpr, k35c32_tpr, k37c32_tpr, k39c32_tpr))

```

```

        k33c32_tpr, k35c32_tpr, k37c32_tpr, k39c32_tpr),
TNR = c(k3c32_tnr, k5c32_tnr, k7c32_tnr, k9c32_tnr, k11c32_tnr,
         k13c32_tnr, k15c32_tnr, k17c32_tnr, k19c32_tnr, k21c32_tnr,
         k23c32_tnr, k25c32_tnr, k27c32_tnr, k29c32_tnr, k31c32_tnr,
         k33c32_tnr, k35c32_tnr, k37c32_tnr, k39c32_tnr),
Truescore = c(k3c32_truescore, k5c32_truescore, k7c32_truescore,
              k9c32_truescore, k11c32_truescore, k13c32_truescore,
              k15c32_truescore, k17c32_truescore, k19c32_truescore,
              k21c32_truescore, k23c32_truescore, k25c32_truescore,
              k27c32_truescore, k29c32_truescore, k31c32_truescore,
              k33c32_truescore, k35c32_truescore, k37c32_truescore,
              k39c32_truescore))

knitr::kable(c32_results[1:19, ], caption = "c32_results")

```

Table 15: c32_results

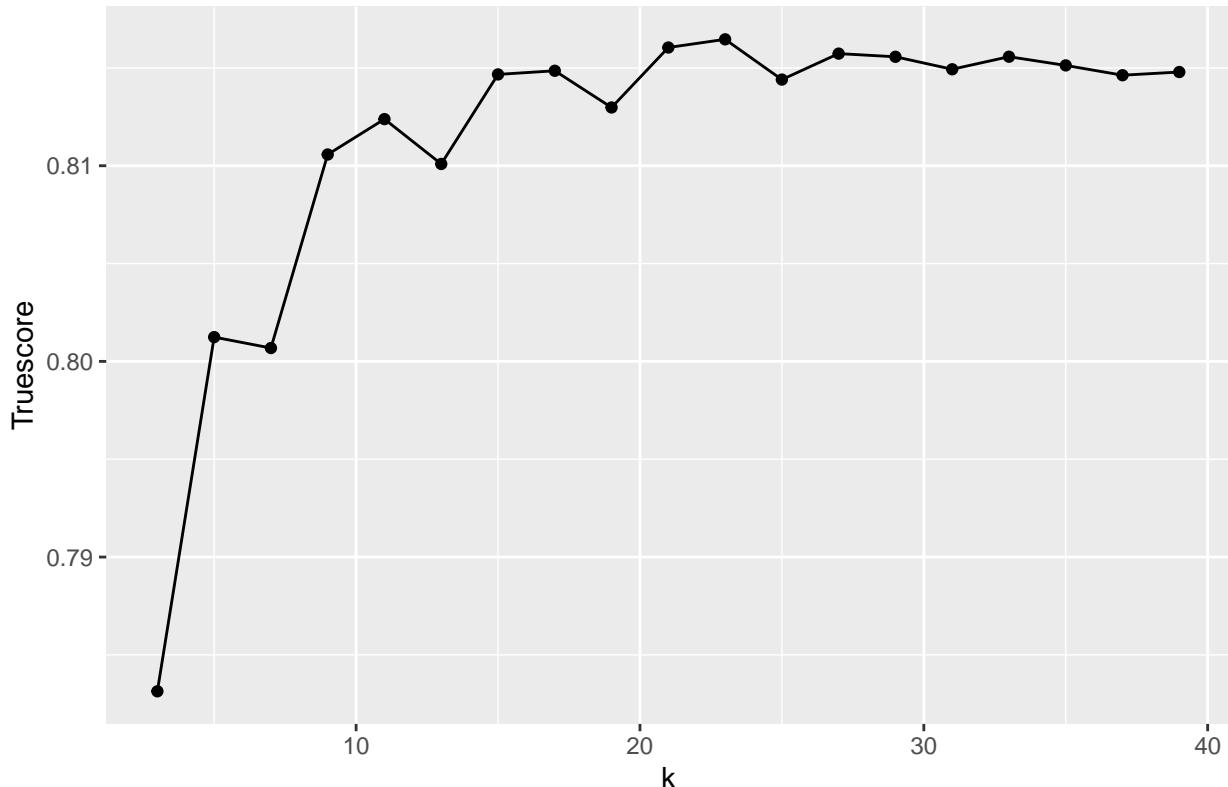
k	Cut	TPR	TNR	Truescore
3	0.32	0.853815	0.723268	0.783138
5	0.32	0.794629	0.807964	0.801241
7	0.32	0.764508	0.840452	0.800683
9	0.32	0.834337	0.788128	0.810574
11	0.32	0.809588	0.815198	0.812383
13	0.32	0.788655	0.832722	0.810090
15	0.32	0.830572	0.799359	0.814667
17	0.32	0.814458	0.815248	0.814853
19	0.32	0.799849	0.826545	0.812978
21	0.32	0.830974	0.801638	0.816042
23	0.32	0.819478	0.813464	0.816460
25	0.32	0.806275	0.822697	0.814403
27	0.32	0.829618	0.802299	0.815730
29	0.32	0.819679	0.811499	0.815568
31	0.32	0.837701	0.793364	0.814930
33	0.32	0.829468	0.802134	0.815572
35	0.32	0.820984	0.809352	0.815127
37	0.32	0.836747	0.793645	0.814626
39	0.32	0.829367	0.800714	0.814789

```

ggplot(c32_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.32")

```

Optimal k for Decision Cutoff 0.32



```
#####
# 0.33 Cutoff
#####

# For the decision cutoff of 0.33, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c33 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred33, obs))
  confusionMatrix(ss$pred33, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c33) <- fk_dfs_v

cm_c33_tables <- sapply(cm_c33, "[[", 2)
cm_c33_tables <- as_tibble(cm_c33_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.33.

k3c33_tpr <- round(sum(cm_c33_tables[1, 1:10]) /
  (sum(cm_c33_tables[1, 1:10]) + sum(cm_c33_tables[2, 1:10])), 6)
k3c33_tnr <- round(sum(cm_c33_tables[4, 1:10]) /
  (sum(cm_c33_tables[4, 1:10]) + sum(cm_c33_tables[3, 1:10])), 6)
```

```

k3c33_truescore <- round((2 * k3c33_tpr * k3c33_tnr) /
  (k3c33_tpr + k3c33_tnr), 6)

k5c33_tpr <- round(sum(cm_c33_tables[1, 11:20]) /
  (sum(cm_c33_tables[1, 11:20]) + sum(cm_c33_tables[2, 11:20])), 6)
k5c33_tnr <- round(sum(cm_c33_tables[4, 11:20]) /
  (sum(cm_c33_tables[4, 11:20]) + sum(cm_c33_tables[3, 11:20])), 6)
k5c33_truescore <- round((2 * k5c33_tpr * k5c33_tnr) /
  (k5c33_tpr + k5c33_tnr), 6)

k7c33_tpr <- round(sum(cm_c33_tables[1, 21:30]) /
  (sum(cm_c33_tables[1, 21:30]) + sum(cm_c33_tables[2, 21:30])), 6)
k7c33_tnr <- round(sum(cm_c33_tables[4, 21:30]) /
  (sum(cm_c33_tables[4, 21:30]) + sum(cm_c33_tables[3, 21:30])), 6)
k7c33_truescore <- round((2 * k7c33_tpr * k7c33_tnr) /
  (k7c33_tpr + k7c33_tnr), 6)

k9c33_tpr <- round(sum(cm_c33_tables[1, 31:40]) /
  (sum(cm_c33_tables[1, 31:40]) + sum(cm_c33_tables[2, 31:40])), 6)
k9c33_tnr <- round(sum(cm_c33_tables[4, 31:40]) /
  (sum(cm_c33_tables[4, 31:40]) + sum(cm_c33_tables[3, 31:40])), 6)
k9c33_truescore <- round((2 * k9c33_tpr * k9c33_tnr) /
  (k9c33_tpr + k9c33_tnr), 6)

k11c33_tpr <- round(sum(cm_c33_tables[1, 41:50]) /
  (sum(cm_c33_tables[1, 41:50]) + sum(cm_c33_tables[2, 41:50])), 6)
k11c33_tnr <- round(sum(cm_c33_tables[4, 41:50]) /
  (sum(cm_c33_tables[4, 41:50]) + sum(cm_c33_tables[3, 41:50])), 6)
k11c33_truescore <- round((2 * k11c33_tpr * k11c33_tnr) /
  (k11c33_tpr + k11c33_tnr), 6)

k13c33_tpr <- round(sum(cm_c33_tables[1, 51:60]) /
  (sum(cm_c33_tables[1, 51:60]) + sum(cm_c33_tables[2, 51:60])), 6)
k13c33_tnr <- round(sum(cm_c33_tables[4, 51:60]) /
  (sum(cm_c33_tables[4, 51:60]) + sum(cm_c33_tables[3, 51:60])), 6)
k13c33_truescore <- round((2 * k13c33_tpr * k13c33_tnr) /
  (k13c33_tpr + k13c33_tnr), 6)

k15c33_tpr <- round(sum(cm_c33_tables[1, 61:70]) /
  (sum(cm_c33_tables[1, 61:70]) + sum(cm_c33_tables[2, 61:70])), 6)
k15c33_tnr <- round(sum(cm_c33_tables[4, 61:70]) /
  (sum(cm_c33_tables[4, 61:70]) + sum(cm_c33_tables[3, 61:70])), 6)
k15c33_truescore <- round((2 * k15c33_tpr * k15c33_tnr) /
  (k15c33_tpr + k15c33_tnr), 6)

k17c33_tpr <- round(sum(cm_c33_tables[1, 71:80]) /
  (sum(cm_c33_tables[1, 71:80]) + sum(cm_c33_tables[2, 71:80])), 6)
k17c33_tnr <- round(sum(cm_c33_tables[4, 71:80]) /
  (sum(cm_c33_tables[4, 71:80]) + sum(cm_c33_tables[3, 71:80])), 6)
k17c33_truescore <- round((2 * k17c33_tpr * k17c33_tnr) /
  (k17c33_tpr + k17c33_tnr), 6)

k19c33_tpr <- round(sum(cm_c33_tables[1, 81:90]) /

```

```

(sum(cm_c33_tables[1, 81:90]) + sum(cm_c33_tables[2, 81:90])), 6)
k19c33_tnr <- round(sum(cm_c33_tables[4, 81:90]) /
  (sum(cm_c33_tables[4, 81:90]) + sum(cm_c33_tables[3, 81:90])), 6)
k19c33_truescore <- round((2 * k19c33_tpr * k19c33_tnr) /
  (k19c33_tpr + k19c33_tnr), 6)

k21c33_tpr <- round(sum(cm_c33_tables[1, 91:100]) /
  (sum(cm_c33_tables[1, 91:100]) + sum(cm_c33_tables[2, 91:100])), 6)
k21c33_tnr <- round(sum(cm_c33_tables[4, 91:100]) /
  (sum(cm_c33_tables[4, 91:100]) + sum(cm_c33_tables[3, 91:100])), 6)
k21c33_truescore <- round((2 * k21c33_tpr * k21c33_tnr) /
  (k21c33_tpr + k21c33_tnr), 6)

k23c33_tpr <- round(sum(cm_c33_tables[1, 101:110]) /
  (sum(cm_c33_tables[1, 101:110]) + sum(cm_c33_tables[2, 101:110])), 6)
k23c33_tnr <- round(sum(cm_c33_tables[4, 101:110]) /
  (sum(cm_c33_tables[4, 101:110]) + sum(cm_c33_tables[3, 101:110])), 6)
k23c33_truescore <- round((2 * k23c33_tpr * k23c33_tnr) /
  (k23c33_tpr + k23c33_tnr), 6)

k25c33_tpr <- round(sum(cm_c33_tables[1, 111:120]) /
  (sum(cm_c33_tables[1, 111:120]) + sum(cm_c33_tables[2, 111:120])), 6)
k25c33_tnr <- round(sum(cm_c33_tables[4, 111:120]) /
  (sum(cm_c33_tables[4, 111:120]) + sum(cm_c33_tables[3, 111:120])), 6)
k25c33_truescore <- round((2 * k25c33_tpr * k25c33_tnr) /
  (k25c33_tpr + k25c33_tnr), 6)

k27c33_tpr <- round(sum(cm_c33_tables[1, 121:130]) /
  (sum(cm_c33_tables[1, 121:130]) + sum(cm_c33_tables[2, 121:130])), 6)
k27c33_tnr <- round(sum(cm_c33_tables[4, 121:130]) /
  (sum(cm_c33_tables[4, 121:130]) + sum(cm_c33_tables[3, 121:130])), 6)
k27c33_truescore <- round((2 * k27c33_tpr * k27c33_tnr) /
  (k27c33_tpr + k27c33_tnr), 6)

k29c33_tpr <- round(sum(cm_c33_tables[1, 131:140]) /
  (sum(cm_c33_tables[1, 131:140]) + sum(cm_c33_tables[2, 131:140])), 6)
k29c33_tnr <- round(sum(cm_c33_tables[4, 131:140]) /
  (sum(cm_c33_tables[4, 131:140]) + sum(cm_c33_tables[3, 131:140])), 6)
k29c33_truescore <- round((2 * k29c33_tpr * k29c33_tnr) /
  (k29c33_tpr + k29c33_tnr), 6)

k31c33_tpr <- round(sum(cm_c33_tables[1, 141:150]) /
  (sum(cm_c33_tables[1, 141:150]) + sum(cm_c33_tables[2, 141:150])), 6)
k31c33_tnr <- round(sum(cm_c33_tables[4, 141:150]) /
  (sum(cm_c33_tables[4, 141:150]) + sum(cm_c33_tables[3, 141:150])), 6)
k31c33_truescore <- round((2 * k31c33_tpr * k31c33_tnr) /
  (k31c33_tpr + k31c33_tnr), 6)

k33c33_tpr <- round(sum(cm_c33_tables[1, 151:160]) /
  (sum(cm_c33_tables[1, 151:160]) + sum(cm_c33_tables[2, 151:160])), 6)
k33c33_tnr <- round(sum(cm_c33_tables[4, 151:160]) /
  (sum(cm_c33_tables[4, 151:160]) + sum(cm_c33_tables[3, 151:160])), 6)
k33c33_truescore <- round((2 * k33c33_tpr * k33c33_tnr) /

```

```

(k33c33_tpr + k33c33_tnr), 6)

k35c33_tpr <- round(sum(cm_c33_tables[1, 161:170]) /
  (sum(cm_c33_tables[1, 161:170]) + sum(cm_c33_tables[2, 161:170])), 6)
k35c33_tnr <- round(sum(cm_c33_tables[4, 161:170]) /
  (sum(cm_c33_tables[4, 161:170]) + sum(cm_c33_tables[3, 161:170])), 6)
k35c33_truescore <- round((2 * k35c33_tpr * k35c33_tnr) /
  (k35c33_tpr + k35c33_tnr), 6)

k37c33_tpr <- round(sum(cm_c33_tables[1, 171:180]) /
  (sum(cm_c33_tables[1, 171:180]) + sum(cm_c33_tables[2, 171:180])), 6)
k37c33_tnr <- round(sum(cm_c33_tables[4, 171:180]) /
  (sum(cm_c33_tables[4, 171:180]) + sum(cm_c33_tables[3, 171:180])), 6)
k37c33_truescore <- round((2 * k37c33_tpr * k37c33_tnr) /
  (k37c33_tpr + k37c33_tnr), 6)

k39c33_tpr <- round(sum(cm_c33_tables[1, 181:190]) /
  (sum(cm_c33_tables[1, 181:190]) + sum(cm_c33_tables[2, 181:190])), 6)
k39c33_tnr <- round(sum(cm_c33_tables[4, 181:190]) /
  (sum(cm_c33_tables[4, 181:190]) + sum(cm_c33_tables[3, 181:190])), 6)
k39c33_truescore <- round((2 * k39c33_tpr * k39c33_tnr) /
  (k39c33_tpr + k39c33_tnr), 6)

# Compile the 0.33 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c33_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.33,
  TPR = c(k3c33_tpr, k5c33_tpr, k7c33_tpr, k9c33_tpr, k11c33_tpr,
    k13c33_tpr, k15c33_tpr, k17c33_tpr, k19c33_tpr, k21c33_tpr,
    k23c33_tpr, k25c33_tpr, k27c33_tpr, k29c33_tpr, k31c33_tpr,
    k33c33_tpr, k35c33_tpr, k37c33_tpr, k39c33_tpr),
  TNR = c(k3c33_tnr, k5c33_tnr, k7c33_tnr, k9c33_tnr, k11c33_tnr,
    k13c33_tnr, k15c33_tnr, k17c33_tnr, k19c33_tnr, k21c33_tnr,
    k23c33_tnr, k25c33_tnr, k27c33_tnr, k29c33_tnr, k31c33_tnr,
    k33c33_tnr, k35c33_tnr, k37c33_tnr, k39c33_tnr),
  Truescore = c(k3c33_truescore, k5c33_truescore, k7c33_truescore,
    k9c33_truescore, k11c33_truescore, k13c33_truescore,
    k15c33_truescore, k17c33_truescore, k19c33_truescore,
    k21c33_truescore, k23c33_truescore, k25c33_truescore,
    k27c33_truescore, k29c33_truescore, k31c33_truescore,
    k33c33_truescore, k35c33_truescore, k37c33_truescore,
    k39c33_truescore))

knitr::kable(c33_results[1:19, ], caption = "c33_results")

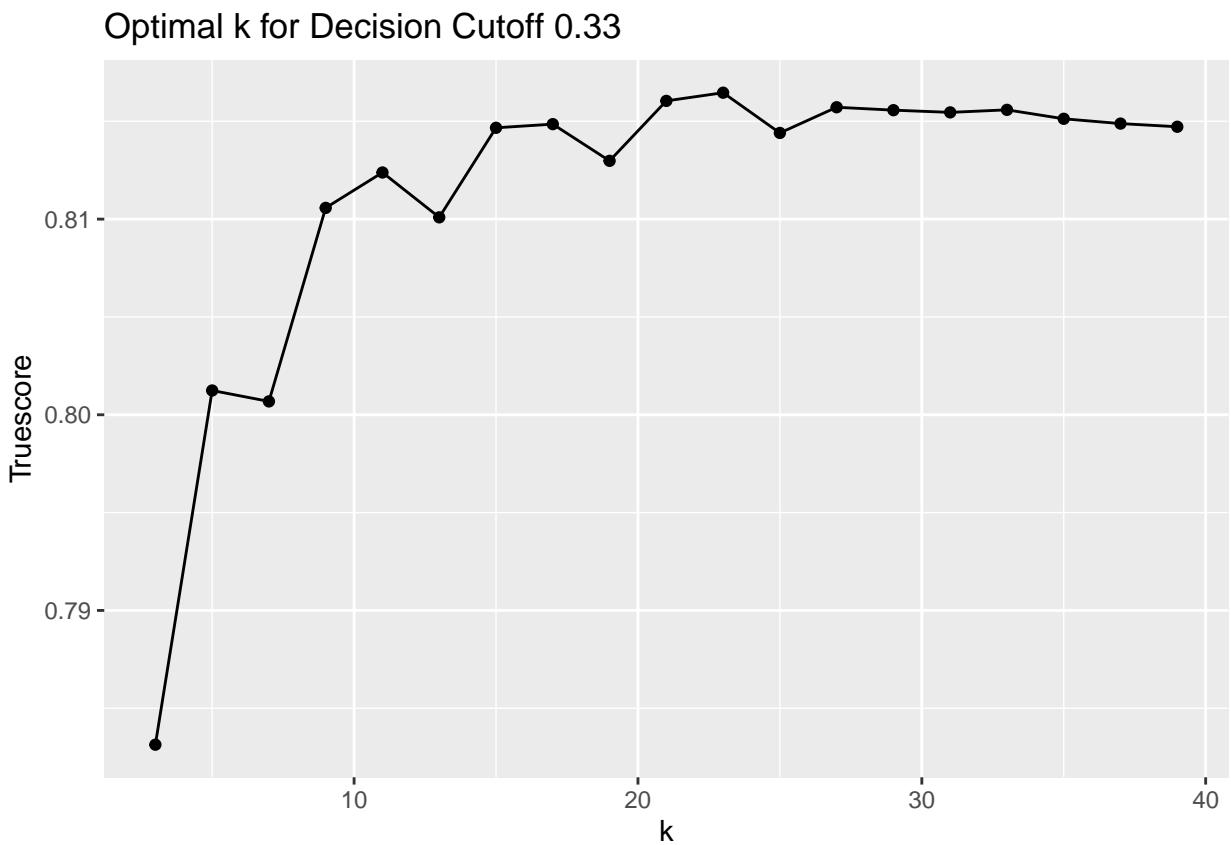
```

Table 16: c33_results

k	Cut	TPR	TNR	Truescore
3	0.33	0.853815	0.723268	0.783138
5	0.33	0.794629	0.807964	0.801241
7	0.33	0.764508	0.840452	0.800683
9	0.33	0.834337	0.788128	0.810574

k	Cut	TPR	TNR	Truescore
11	0.33	0.809588	0.815198	0.812383
13	0.33	0.788655	0.832722	0.810090
15	0.33	0.830572	0.799359	0.814667
17	0.33	0.814458	0.815248	0.814853
19	0.33	0.799849	0.826545	0.812978
21	0.33	0.830974	0.801638	0.816042
23	0.33	0.819478	0.813464	0.816460
25	0.33	0.806275	0.822697	0.814403
27	0.33	0.829468	0.802415	0.815717
29	0.33	0.819679	0.811499	0.815568
31	0.33	0.810843	0.820120	0.815455
33	0.33	0.829317	0.802299	0.815584
35	0.33	0.820984	0.809352	0.815127
37	0.33	0.813153	0.816619	0.814882
39	0.33	0.829066	0.800862	0.814720

```
ggplot(c33_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.33")
```



```
#####
# 0.34 Cutoff
#####
```

```

# For the decision cutoff of 0.34, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c34 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred34, obs))
  confusionMatrix(ss$pred34, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c34) <- fk_dfs_v

cm_c34_tables <- sapply(cm_c34, "[[", 2)
cm_c34_tables <- as_tibble(cm_c34_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.34.

k3c34_tpr <- round(sum(cm_c34_tables[1, 1:10]) /
  (sum(cm_c34_tables[1, 1:10]) + sum(cm_c34_tables[2, 1:10])), 6)
k3c34_tnr <- round(sum(cm_c34_tables[4, 1:10]) /
  (sum(cm_c34_tables[4, 1:10]) + sum(cm_c34_tables[3, 1:10])), 6)
k3c34_truescore <- round((2 * k3c34_tpr * k3c34_tnr) /
  (k3c34_tpr + k3c34_tnr), 6)

k5c34_tpr <- round(sum(cm_c34_tables[1, 11:20]) /
  (sum(cm_c34_tables[1, 11:20]) + sum(cm_c34_tables[2, 11:20])), 6)
k5c34_tnr <- round(sum(cm_c34_tables[4, 11:20]) /
  (sum(cm_c34_tables[4, 11:20]) + sum(cm_c34_tables[3, 11:20])), 6)
k5c34_truescore <- round((2 * k5c34_tpr * k5c34_tnr) /
  (k5c34_tpr + k5c34_tnr), 6)

k7c34_tpr <- round(sum(cm_c34_tables[1, 21:30]) /
  (sum(cm_c34_tables[1, 21:30]) + sum(cm_c34_tables[2, 21:30])), 6)
k7c34_tnr <- round(sum(cm_c34_tables[4, 21:30]) /
  (sum(cm_c34_tables[4, 21:30]) + sum(cm_c34_tables[3, 21:30])), 6)
k7c34_truescore <- round((2 * k7c34_tpr * k7c34_tnr) /
  (k7c34_tpr + k7c34_tnr), 6)

k9c34_tpr <- round(sum(cm_c34_tables[1, 31:40]) /
  (sum(cm_c34_tables[1, 31:40]) + sum(cm_c34_tables[2, 31:40])), 6)
k9c34_tnr <- round(sum(cm_c34_tables[4, 31:40]) /
  (sum(cm_c34_tables[4, 31:40]) + sum(cm_c34_tables[3, 31:40])), 6)
k9c34_truescore <- round((2 * k9c34_tpr * k9c34_tnr) /
  (k9c34_tpr + k9c34_tnr), 6)

k11c34_tpr <- round(sum(cm_c34_tables[1, 41:50]) /
  (sum(cm_c34_tables[1, 41:50]) + sum(cm_c34_tables[2, 41:50])), 6)
k11c34_tnr <- round(sum(cm_c34_tables[4, 41:50]) /
  (sum(cm_c34_tables[4, 41:50]) + sum(cm_c34_tables[3, 41:50])), 6)
k11c34_truescore <- round((2 * k11c34_tpr * k11c34_tnr) /
  (k11c34_tpr + k11c34_tnr), 6)

```

```

k13c34_tpr <- round(sum(cm_c34_tables[1, 51:60]) /
  (sum(cm_c34_tables[1, 51:60]) + sum(cm_c34_tables[2, 51:60])), 6)
k13c34_tnr <- round(sum(cm_c34_tables[4, 51:60]) /
  (sum(cm_c34_tables[4, 51:60]) + sum(cm_c34_tables[3, 51:60])), 6)
k13c34_truescore <- round((2 * k13c34_tpr * k13c34_tnr) /
  (k13c34_tpr + k13c34_tnr), 6)

k15c34_tpr <- round(sum(cm_c34_tables[1, 61:70]) /
  (sum(cm_c34_tables[1, 61:70]) + sum(cm_c34_tables[2, 61:70])), 6)
k15c34_tnr <- round(sum(cm_c34_tables[4, 61:70]) /
  (sum(cm_c34_tables[4, 61:70]) + sum(cm_c34_tables[3, 61:70])), 6)
k15c34_truescore <- round((2 * k15c34_tpr * k15c34_tnr) /
  (k15c34_tpr + k15c34_tnr), 6)

k17c34_tpr <- round(sum(cm_c34_tables[1, 71:80]) /
  (sum(cm_c34_tables[1, 71:80]) + sum(cm_c34_tables[2, 71:80])), 6)
k17c34_tnr <- round(sum(cm_c34_tables[4, 71:80]) /
  (sum(cm_c34_tables[4, 71:80]) + sum(cm_c34_tables[3, 71:80])), 6)
k17c34_truescore <- round((2 * k17c34_tpr * k17c34_tnr) /
  (k17c34_tpr + k17c34_tnr), 6)

k19c34_tpr <- round(sum(cm_c34_tables[1, 81:90]) /
  (sum(cm_c34_tables[1, 81:90]) + sum(cm_c34_tables[2, 81:90])), 6)
k19c34_tnr <- round(sum(cm_c34_tables[4, 81:90]) /
  (sum(cm_c34_tables[4, 81:90]) + sum(cm_c34_tables[3, 81:90])), 6)
k19c34_truescore <- round((2 * k19c34_tpr * k19c34_tnr) /
  (k19c34_tpr + k19c34_tnr), 6)

k21c34_tpr <- round(sum(cm_c34_tables[1, 91:100]) /
  (sum(cm_c34_tables[1, 91:100]) + sum(cm_c34_tables[2, 91:100])), 6)
k21c34_tnr <- round(sum(cm_c34_tables[4, 91:100]) /
  (sum(cm_c34_tables[4, 91:100]) + sum(cm_c34_tables[3, 91:100])), 6)
k21c34_truescore <- round((2 * k21c34_tpr * k21c34_tnr) /
  (k21c34_tpr + k21c34_tnr), 6)

k23c34_tpr <- round(sum(cm_c34_tables[1, 101:110]) /
  (sum(cm_c34_tables[1, 101:110]) + sum(cm_c34_tables[2, 101:110])), 6)
k23c34_tnr <- round(sum(cm_c34_tables[4, 101:110]) /
  (sum(cm_c34_tables[4, 101:110]) + sum(cm_c34_tables[3, 101:110])), 6)
k23c34_truescore <- round((2 * k23c34_tpr * k23c34_tnr) /
  (k23c34_tpr + k23c34_tnr), 6)

k25c34_tpr <- round(sum(cm_c34_tables[1, 111:120]) /
  (sum(cm_c34_tables[1, 111:120]) + sum(cm_c34_tables[2, 111:120])), 6)
k25c34_tnr <- round(sum(cm_c34_tables[4, 111:120]) /
  (sum(cm_c34_tables[4, 111:120]) + sum(cm_c34_tables[3, 111:120])), 6)
k25c34_truescore <- round((2 * k25c34_tpr * k25c34_tnr) /
  (k25c34_tpr + k25c34_tnr), 6)

k27c34_tpr <- round(sum(cm_c34_tables[1, 121:130]) /
  (sum(cm_c34_tables[1, 121:130]) + sum(cm_c34_tables[2, 121:130])), 6)
k27c34_tnr <- round(sum(cm_c34_tables[4, 121:130]) /
  (sum(cm_c34_tables[4, 121:130]) + sum(cm_c34_tables[3, 121:130])), 6)

```

```

k27c34_truescore <- round((2 * k27c34_tpr * k27c34_tnr) /
  (k27c34_tpr + k27c34_tnr), 6)

k29c34_tpr <- round(sum(cm_c34_tables[1, 131:140]) /
  (sum(cm_c34_tables[1, 131:140]) + sum(cm_c34_tables[2, 131:140])), 6)
k29c34_tnr <- round(sum(cm_c34_tables[4, 131:140]) /
  (sum(cm_c34_tables[4, 131:140]) + sum(cm_c34_tables[3, 131:140])), 6)
k29c34_truescore <- round((2 * k29c34_tpr * k29c34_tnr) /
  (k29c34_tpr + k29c34_tnr), 6)

k31c34_tpr <- round(sum(cm_c34_tables[1, 141:150]) /
  (sum(cm_c34_tables[1, 141:150]) + sum(cm_c34_tables[2, 141:150])), 6)
k31c34_tnr <- round(sum(cm_c34_tables[4, 141:150]) /
  (sum(cm_c34_tables[4, 141:150]) + sum(cm_c34_tables[3, 141:150])), 6)
k31c34_truescore <- round((2 * k31c34_tpr * k31c34_tnr) /
  (k31c34_tpr + k31c34_tnr), 6)

k33c34_tpr <- round(sum(cm_c34_tables[1, 151:160]) /
  (sum(cm_c34_tables[1, 151:160]) + sum(cm_c34_tables[2, 151:160])), 6)
k33c34_tnr <- round(sum(cm_c34_tables[4, 151:160]) /
  (sum(cm_c34_tables[4, 151:160]) + sum(cm_c34_tables[3, 151:160])), 6)
k33c34_truescore <- round((2 * k33c34_tpr * k33c34_tnr) /
  (k33c34_tpr + k33c34_tnr), 6)

k35c34_tpr <- round(sum(cm_c34_tables[1, 161:170]) /
  (sum(cm_c34_tables[1, 161:170]) + sum(cm_c34_tables[2, 161:170])), 6)
k35c34_tnr <- round(sum(cm_c34_tables[4, 161:170]) /
  (sum(cm_c34_tables[4, 161:170]) + sum(cm_c34_tables[3, 161:170])), 6)
k35c34_truescore <- round((2 * k35c34_tpr * k35c34_tnr) /
  (k35c34_tpr + k35c34_tnr), 6)

k37c34_tpr <- round(sum(cm_c34_tables[1, 171:180]) /
  (sum(cm_c34_tables[1, 171:180]) + sum(cm_c34_tables[2, 171:180])), 6)
k37c34_tnr <- round(sum(cm_c34_tables[4, 171:180]) /
  (sum(cm_c34_tables[4, 171:180]) + sum(cm_c34_tables[3, 171:180])), 6)
k37c34_truescore <- round((2 * k37c34_tpr * k37c34_tnr) /
  (k37c34_tpr + k37c34_tnr), 6)

k39c34_tpr <- round(sum(cm_c34_tables[1, 181:190]) /
  (sum(cm_c34_tables[1, 181:190]) + sum(cm_c34_tables[2, 181:190])), 6)
k39c34_tnr <- round(sum(cm_c34_tables[4, 181:190]) /
  (sum(cm_c34_tables[4, 181:190]) + sum(cm_c34_tables[3, 181:190])), 6)
k39c34_truescore <- round((2 * k39c34_tpr * k39c34_tnr) /
  (k39c34_tpr + k39c34_tnr), 6)

# Compile the 0.34 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c34_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.34,
  TPR = c(k3c34_tpr, k5c34_tpr, k7c34_tpr, k9c34_tpr, k11c34_tpr,
    k13c34_tpr, k15c34_tpr, k17c34_tpr, k19c34_tpr, k21c34_tpr,
    k23c34_tpr, k25c34_tpr, k27c34_tpr, k29c34_tpr, k31c34_tpr,
    k33c34_tpr, k35c34_tpr, k37c34_tpr, k39c34_tpr))

```

```

            k33c34_tpr, k35c34_tpr, k37c34_tpr, k39c34_tpr),
TNR = c(k3c34_tnr, k5c34_tnr, k7c34_tnr, k9c34_tnr, k11c34_tnr,
         k13c34_tnr, k15c34_tnr, k17c34_tnr, k19c34_tnr, k21c34_tnr,
         k23c34_tnr, k25c34_tnr, k27c34_tnr, k29c34_tnr, k31c34_tnr,
         k33c34_tnr, k35c34_tnr, k37c34_tnr, k39c34_tnr),
Truescore = c(k3c34_truescore, k5c34_truescore, k7c34_truescore,
              k9c34_truescore, k11c34_truescore, k13c34_truescore,
              k15c34_truescore, k17c34_truescore, k19c34_truescore,
              k21c34_truescore, k23c34_truescore, k25c34_truescore,
              k27c34_truescore, k29c34_truescore, k31c34_truescore,
              k33c34_truescore, k35c34_truescore, k37c34_truescore,
              k39c34_truescore))

knitr::kable(c34_results[1:19, ], caption = "c34_results")

```

Table 17: c34_results

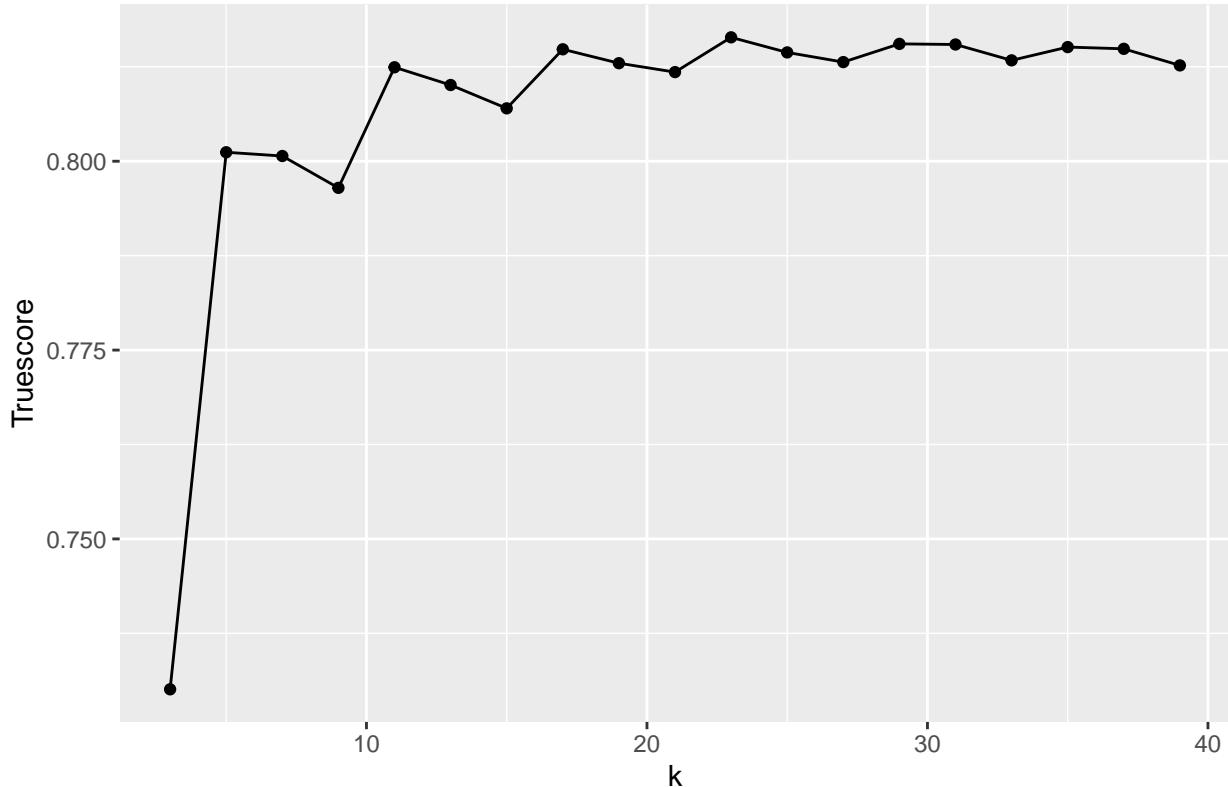
k	Cut	TPR	TNR	Truescore
3	0.34	0.616215	0.895600	0.730092
5	0.34	0.794428	0.808063	0.801187
7	0.34	0.764508	0.840468	0.800690
9	0.34	0.742520	0.858884	0.796474
11	0.34	0.809488	0.815413	0.812440
13	0.34	0.788655	0.832722	0.810090
15	0.34	0.772289	0.844977	0.807000
17	0.34	0.814257	0.815380	0.814818
19	0.34	0.799849	0.826545	0.812978
21	0.34	0.787851	0.837248	0.811799
23	0.34	0.819227	0.813596	0.816402
25	0.34	0.806275	0.822697	0.814403
27	0.34	0.796135	0.830856	0.813125
29	0.34	0.819428	0.811680	0.815536
31	0.34	0.810843	0.820120	0.815455
33	0.34	0.801104	0.825984	0.813354
35	0.34	0.820733	0.809566	0.815111
37	0.34	0.813153	0.816619	0.814882
39	0.34	0.803564	0.822020	0.812687

```

ggplot(c34_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.34")

```

Optimal k for Decision Cutoff 0.34



```
#####
# 0.35 Cutoff
#####

# For the decision cutoff of 0.35, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c35 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred35, obs))
  confusionMatrix(ss$pred35, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c35) <- fk_dfs_v

cm_c35_tables <- sapply(cm_c35, "[[", 2)
cm_c35_tables <- as_tibble(cm_c35_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.35.

k3c35_tpr <- round(sum(cm_c35_tables[1, 1:10]) /
  (sum(cm_c35_tables[1, 1:10]) + sum(cm_c35_tables[2, 1:10])), 6)
k3c35_tnr <- round(sum(cm_c35_tables[4, 1:10]) /
  (sum(cm_c35_tables[4, 1:10]) + sum(cm_c35_tables[3, 1:10])), 6)
```

```

k3c35_truescore <- round((2 * k3c35_tpr * k3c35_tnr) /
  (k3c35_tpr + k3c35_tnr), 6)

k5c35_tpr <- round(sum(cm_c35_tables[1, 11:20]) /
  (sum(cm_c35_tables[1, 11:20]) + sum(cm_c35_tables[2, 11:20])), 6)
k5c35_tnr <- round(sum(cm_c35_tables[4, 11:20]) /
  (sum(cm_c35_tables[4, 11:20]) + sum(cm_c35_tables[3, 11:20])), 6)
k5c35_truescore <- round((2 * k5c35_tpr * k5c35_tnr) /
  (k5c35_tpr + k5c35_tnr), 6)

k7c35_tpr <- round(sum(cm_c35_tables[1, 21:30]) /
  (sum(cm_c35_tables[1, 21:30]) + sum(cm_c35_tables[2, 21:30])), 6)
k7c35_tnr <- round(sum(cm_c35_tables[4, 21:30]) /
  (sum(cm_c35_tables[4, 21:30]) + sum(cm_c35_tables[3, 21:30])), 6)
k7c35_truescore <- round((2 * k7c35_tpr * k7c35_tnr) /
  (k7c35_tpr + k7c35_tnr), 6)

k9c35_tpr <- round(sum(cm_c35_tables[1, 31:40]) /
  (sum(cm_c35_tables[1, 31:40]) + sum(cm_c35_tables[2, 31:40])), 6)
k9c35_tnr <- round(sum(cm_c35_tables[4, 31:40]) /
  (sum(cm_c35_tables[4, 31:40]) + sum(cm_c35_tables[3, 31:40])), 6)
k9c35_truescore <- round((2 * k9c35_tpr * k9c35_tnr) /
  (k9c35_tpr + k9c35_tnr), 6)

k11c35_tpr <- round(sum(cm_c35_tables[1, 41:50]) /
  (sum(cm_c35_tables[1, 41:50]) + sum(cm_c35_tables[2, 41:50])), 6)
k11c35_tnr <- round(sum(cm_c35_tables[4, 41:50]) /
  (sum(cm_c35_tables[4, 41:50]) + sum(cm_c35_tables[3, 41:50])), 6)
k11c35_truescore <- round((2 * k11c35_tpr * k11c35_tnr) /
  (k11c35_tpr + k11c35_tnr), 6)

k13c35_tpr <- round(sum(cm_c35_tables[1, 51:60]) /
  (sum(cm_c35_tables[1, 51:60]) + sum(cm_c35_tables[2, 51:60])), 6)
k13c35_tnr <- round(sum(cm_c35_tables[4, 51:60]) /
  (sum(cm_c35_tables[4, 51:60]) + sum(cm_c35_tables[3, 51:60])), 6)
k13c35_truescore <- round((2 * k13c35_tpr * k13c35_tnr) /
  (k13c35_tpr + k13c35_tnr), 6)

k15c35_tpr <- round(sum(cm_c35_tables[1, 61:70]) /
  (sum(cm_c35_tables[1, 61:70]) + sum(cm_c35_tables[2, 61:70])), 6)
k15c35_tnr <- round(sum(cm_c35_tables[4, 61:70]) /
  (sum(cm_c35_tables[4, 61:70]) + sum(cm_c35_tables[3, 61:70])), 6)
k15c35_truescore <- round((2 * k15c35_tpr * k15c35_tnr) /
  (k15c35_tpr + k15c35_tnr), 6)

k17c35_tpr <- round(sum(cm_c35_tables[1, 71:80]) /
  (sum(cm_c35_tables[1, 71:80]) + sum(cm_c35_tables[2, 71:80])), 6)
k17c35_tnr <- round(sum(cm_c35_tables[4, 71:80]) /
  (sum(cm_c35_tables[4, 71:80]) + sum(cm_c35_tables[3, 71:80])), 6)
k17c35_truescore <- round((2 * k17c35_tpr * k17c35_tnr) /
  (k17c35_tpr + k17c35_tnr), 6)

k19c35_tpr <- round(sum(cm_c35_tables[1, 81:90]) /

```

```

(sum(cm_c35_tables[1, 81:90]) + sum(cm_c35_tables[2, 81:90])), 6)
k19c35_tnr <- round(sum(cm_c35_tables[4, 81:90]) /
  (sum(cm_c35_tables[4, 81:90]) + sum(cm_c35_tables[3, 81:90])), 6)
k19c35_truescore <- round((2 * k19c35_tpr * k19c35_tnr) /
  (k19c35_tpr + k19c35_tnr), 6)

k21c35_tpr <- round(sum(cm_c35_tables[1, 91:100]) /
  (sum(cm_c35_tables[1, 91:100]) + sum(cm_c35_tables[2, 91:100])), 6)
k21c35_tnr <- round(sum(cm_c35_tables[4, 91:100]) /
  (sum(cm_c35_tables[4, 91:100]) + sum(cm_c35_tables[3, 91:100])), 6)
k21c35_truescore <- round((2 * k21c35_tpr * k21c35_tnr) /
  (k21c35_tpr + k21c35_tnr), 6)

k23c35_tpr <- round(sum(cm_c35_tables[1, 101:110]) /
  (sum(cm_c35_tables[1, 101:110]) + sum(cm_c35_tables[2, 101:110])), 6)
k23c35_tnr <- round(sum(cm_c35_tables[4, 101:110]) /
  (sum(cm_c35_tables[4, 101:110]) + sum(cm_c35_tables[3, 101:110])), 6)
k23c35_truescore <- round((2 * k23c35_tpr * k23c35_tnr) /
  (k23c35_tpr + k23c35_tnr), 6)

k25c35_tpr <- round(sum(cm_c35_tables[1, 111:120]) /
  (sum(cm_c35_tables[1, 111:120]) + sum(cm_c35_tables[2, 111:120])), 6)
k25c35_tnr <- round(sum(cm_c35_tables[4, 111:120]) /
  (sum(cm_c35_tables[4, 111:120]) + sum(cm_c35_tables[3, 111:120])), 6)
k25c35_truescore <- round((2 * k25c35_tpr * k25c35_tnr) /
  (k25c35_tpr + k25c35_tnr), 6)

k27c35_tpr <- round(sum(cm_c35_tables[1, 121:130]) /
  (sum(cm_c35_tables[1, 121:130]) + sum(cm_c35_tables[2, 121:130])), 6)
k27c35_tnr <- round(sum(cm_c35_tables[4, 121:130]) /
  (sum(cm_c35_tables[4, 121:130]) + sum(cm_c35_tables[3, 121:130])), 6)
k27c35_truescore <- round((2 * k27c35_tpr * k27c35_tnr) /
  (k27c35_tpr + k27c35_tnr), 6)

k29c35_tpr <- round(sum(cm_c35_tables[1, 131:140]) /
  (sum(cm_c35_tables[1, 131:140]) + sum(cm_c35_tables[2, 131:140])), 6)
k29c35_tnr <- round(sum(cm_c35_tables[4, 131:140]) /
  (sum(cm_c35_tables[4, 131:140]) + sum(cm_c35_tables[3, 131:140])), 6)
k29c35_truescore <- round((2 * k29c35_tpr * k29c35_tnr) /
  (k29c35_tpr + k29c35_tnr), 6)

k31c35_tpr <- round(sum(cm_c35_tables[1, 141:150]) /
  (sum(cm_c35_tables[1, 141:150]) + sum(cm_c35_tables[2, 141:150])), 6)
k31c35_tnr <- round(sum(cm_c35_tables[4, 141:150]) /
  (sum(cm_c35_tables[4, 141:150]) + sum(cm_c35_tables[3, 141:150])), 6)
k31c35_truescore <- round((2 * k31c35_tpr * k31c35_tnr) /
  (k31c35_tpr + k31c35_tnr), 6)

k33c35_tpr <- round(sum(cm_c35_tables[1, 151:160]) /
  (sum(cm_c35_tables[1, 151:160]) + sum(cm_c35_tables[2, 151:160])), 6)
k33c35_tnr <- round(sum(cm_c35_tables[4, 151:160]) /
  (sum(cm_c35_tables[4, 151:160]) + sum(cm_c35_tables[3, 151:160])), 6)
k33c35_truescore <- round((2 * k33c35_tpr * k33c35_tnr) /

```

```

(k33c35_tpr + k33c35_tnr), 6)

k35c35_tpr <- round(sum(cm_c35_tables[1, 161:170]) /
  (sum(cm_c35_tables[1, 161:170]) + sum(cm_c35_tables[2, 161:170])), 6)
k35c35_tnr <- round(sum(cm_c35_tables[4, 161:170]) /
  (sum(cm_c35_tables[4, 161:170]) + sum(cm_c35_tables[3, 161:170])), 6)
k35c35_truescore <- round((2 * k35c35_tpr * k35c35_tnr) /
  (k35c35_tpr + k35c35_tnr), 6)

k37c35_tpr <- round(sum(cm_c35_tables[1, 171:180]) /
  (sum(cm_c35_tables[1, 171:180]) + sum(cm_c35_tables[2, 171:180])), 6)
k37c35_tnr <- round(sum(cm_c35_tables[4, 171:180]) /
  (sum(cm_c35_tables[4, 171:180]) + sum(cm_c35_tables[3, 171:180])), 6)
k37c35_truescore <- round((2 * k37c35_tpr * k37c35_tnr) /
  (k37c35_tpr + k37c35_tnr), 6)

k39c35_tpr <- round(sum(cm_c35_tables[1, 181:190]) /
  (sum(cm_c35_tables[1, 181:190]) + sum(cm_c35_tables[2, 181:190])), 6)
k39c35_tnr <- round(sum(cm_c35_tables[4, 181:190]) /
  (sum(cm_c35_tables[4, 181:190]) + sum(cm_c35_tables[3, 181:190])), 6)
k39c35_truescore <- round((2 * k39c35_tpr * k39c35_tnr) /
  (k39c35_tpr + k39c35_tnr), 6)

# Compile the 0.35 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c35_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.35,
  TPR = c(k3c35_tpr, k5c35_tpr, k7c35_tpr, k9c35_tpr, k11c35_tpr,
    k13c35_tpr, k15c35_tpr, k17c35_tpr, k19c35_tpr, k21c35_tpr,
    k23c35_tpr, k25c35_tpr, k27c35_tpr, k29c35_tpr, k31c35_tpr,
    k33c35_tpr, k35c35_tpr, k37c35_tpr, k39c35_tpr),
  TNR = c(k3c35_tnr, k5c35_tnr, k7c35_tnr, k9c35_tnr, k11c35_tnr,
    k13c35_tnr, k15c35_tnr, k17c35_tnr, k19c35_tnr, k21c35_tnr,
    k23c35_tnr, k25c35_tnr, k27c35_tnr, k29c35_tnr, k31c35_tnr,
    k33c35_tnr, k35c35_tnr, k37c35_tnr, k39c35_tnr),
  Truescore = c(k3c35_truescore, k5c35_truescore, k7c35_truescore,
    k9c35_truescore, k11c35_truescore, k13c35_truescore,
    k15c35_truescore, k17c35_truescore, k19c35_truescore,
    k21c35_truescore, k23c35_truescore, k25c35_truescore,
    k27c35_truescore, k29c35_truescore, k31c35_truescore,
    k33c35_truescore, k35c35_truescore, k37c35_truescore,
    k39c35_truescore))

knitr::kable(c35_results[1:19, ], caption = "c35_results")

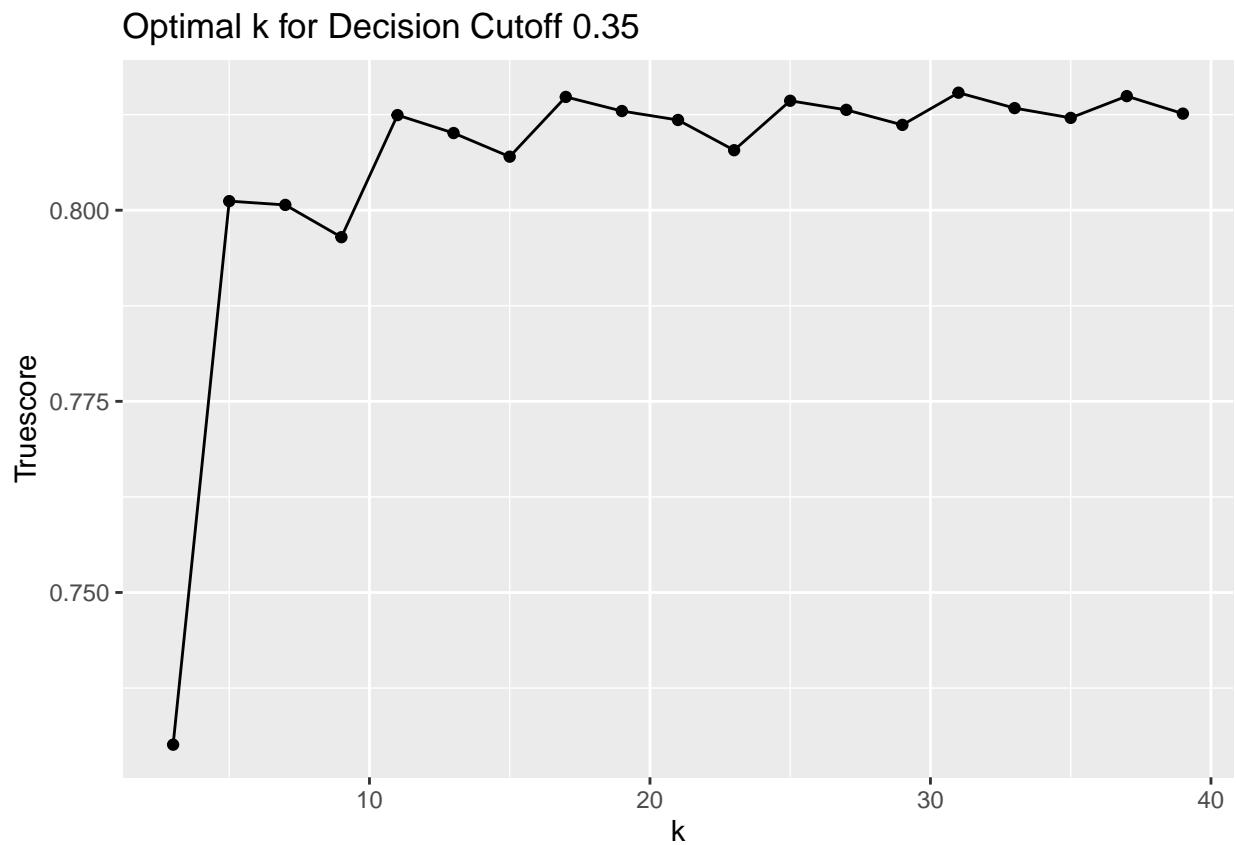
```

Table 18: c35_results

k	Cut	TPR	TNR	Truescore
3	0.35	0.616215	0.895600	0.730092
5	0.35	0.794428	0.808063	0.801187
7	0.35	0.764508	0.840468	0.800690
9	0.35	0.742520	0.858884	0.796474

k	Cut	TPR	TNR	Truescore
11	0.35	0.809488	0.815413	0.812440
13	0.35	0.788655	0.832722	0.810090
15	0.35	0.772289	0.844977	0.807000
17	0.35	0.814257	0.815380	0.814818
19	0.35	0.799699	0.826694	0.812972
21	0.35	0.787851	0.837248	0.811799
23	0.35	0.774096	0.844680	0.807849
25	0.35	0.805974	0.822829	0.814314
27	0.35	0.796135	0.830856	0.813125
29	0.35	0.786295	0.837644	0.811158
31	0.35	0.810542	0.820252	0.815368
33	0.35	0.801104	0.825984	0.813354
35	0.35	0.792520	0.832623	0.812077
37	0.35	0.813102	0.816751	0.814922
39	0.35	0.803313	0.822168	0.812631

```
ggplot(c35_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.35")
```



```
#####
# 0.36 Cutoff
#####
```

```

# For the decision cutoff of 0.36, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c36 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred36, obs))
  confusionMatrix(ss$pred36, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c36) <- fk_dfs_v

cm_c36_tables <- sapply(cm_c36, "[[", 2)
cm_c36_tables <- as_tibble(cm_c36_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.36.

k3c36_tpr <- round(sum(cm_c36_tables[1, 1:10]) /
  (sum(cm_c36_tables[1, 1:10]) + sum(cm_c36_tables[2, 1:10])), 6)
k3c36_tnr <- round(sum(cm_c36_tables[4, 1:10]) /
  (sum(cm_c36_tables[4, 1:10]) + sum(cm_c36_tables[3, 1:10])), 6)
k3c36_truescore <- round((2 * k3c36_tpr * k3c36_tnr) /
  (k3c36_tpr + k3c36_tnr), 6)

k5c36_tpr <- round(sum(cm_c36_tables[1, 11:20]) /
  (sum(cm_c36_tables[1, 11:20]) + sum(cm_c36_tables[2, 11:20])), 6)
k5c36_tnr <- round(sum(cm_c36_tables[4, 11:20]) /
  (sum(cm_c36_tables[4, 11:20]) + sum(cm_c36_tables[3, 11:20])), 6)
k5c36_truescore <- round((2 * k5c36_tpr * k5c36_tnr) /
  (k5c36_tpr + k5c36_tnr), 6)

k7c36_tpr <- round(sum(cm_c36_tables[1, 21:30]) /
  (sum(cm_c36_tables[1, 21:30]) + sum(cm_c36_tables[2, 21:30])), 6)
k7c36_tnr <- round(sum(cm_c36_tables[4, 21:30]) /
  (sum(cm_c36_tables[4, 21:30]) + sum(cm_c36_tables[3, 21:30])), 6)
k7c36_truescore <- round((2 * k7c36_tpr * k7c36_tnr) /
  (k7c36_tpr + k7c36_tnr), 6)

k9c36_tpr <- round(sum(cm_c36_tables[1, 31:40]) /
  (sum(cm_c36_tables[1, 31:40]) + sum(cm_c36_tables[2, 31:40])), 6)
k9c36_tnr <- round(sum(cm_c36_tables[4, 31:40]) /
  (sum(cm_c36_tables[4, 31:40]) + sum(cm_c36_tables[3, 31:40])), 6)
k9c36_truescore <- round((2 * k9c36_tpr * k9c36_tnr) /
  (k9c36_tpr + k9c36_tnr), 6)

k11c36_tpr <- round(sum(cm_c36_tables[1, 41:50]) /
  (sum(cm_c36_tables[1, 41:50]) + sum(cm_c36_tables[2, 41:50])), 6)
k11c36_tnr <- round(sum(cm_c36_tables[4, 41:50]) /
  (sum(cm_c36_tables[4, 41:50]) + sum(cm_c36_tables[3, 41:50])), 6)
k11c36_truescore <- round((2 * k11c36_tpr * k11c36_tnr) /
  (k11c36_tpr + k11c36_tnr), 6)

```

```

k13c36_tpr <- round(sum(cm_c36_tables[1, 51:60]) /
  (sum(cm_c36_tables[1, 51:60]) + sum(cm_c36_tables[2, 51:60])), 6)
k13c36_tnr <- round(sum(cm_c36_tables[4, 51:60]) /
  (sum(cm_c36_tables[4, 51:60]) + sum(cm_c36_tables[3, 51:60])), 6)
k13c36_truescore <- round((2 * k13c36_tpr * k13c36_tnr) /
  (k13c36_tpr + k13c36_tnr), 6)

k15c36_tpr <- round(sum(cm_c36_tables[1, 61:70]) /
  (sum(cm_c36_tables[1, 61:70]) + sum(cm_c36_tables[2, 61:70])), 6)
k15c36_tnr <- round(sum(cm_c36_tables[4, 61:70]) /
  (sum(cm_c36_tables[4, 61:70]) + sum(cm_c36_tables[3, 61:70])), 6)
k15c36_truescore <- round((2 * k15c36_tpr * k15c36_tnr) /
  (k15c36_tpr + k15c36_tnr), 6)

k17c36_tpr <- round(sum(cm_c36_tables[1, 71:80]) /
  (sum(cm_c36_tables[1, 71:80]) + sum(cm_c36_tables[2, 71:80])), 6)
k17c36_tnr <- round(sum(cm_c36_tables[4, 71:80]) /
  (sum(cm_c36_tables[4, 71:80]) + sum(cm_c36_tables[3, 71:80])), 6)
k17c36_truescore <- round((2 * k17c36_tpr * k17c36_tnr) /
  (k17c36_tpr + k17c36_tnr), 6)

k19c36_tpr <- round(sum(cm_c36_tables[1, 81:90]) /
  (sum(cm_c36_tables[1, 81:90]) + sum(cm_c36_tables[2, 81:90])), 6)
k19c36_tnr <- round(sum(cm_c36_tables[4, 81:90]) /
  (sum(cm_c36_tables[4, 81:90]) + sum(cm_c36_tables[3, 81:90])), 6)
k19c36_truescore <- round((2 * k19c36_tpr * k19c36_tnr) /
  (k19c36_tpr + k19c36_tnr), 6)

k21c36_tpr <- round(sum(cm_c36_tables[1, 91:100]) /
  (sum(cm_c36_tables[1, 91:100]) + sum(cm_c36_tables[2, 91:100])), 6)
k21c36_tnr <- round(sum(cm_c36_tables[4, 91:100]) /
  (sum(cm_c36_tables[4, 91:100]) + sum(cm_c36_tables[3, 91:100])), 6)
k21c36_truescore <- round((2 * k21c36_tpr * k21c36_tnr) /
  (k21c36_tpr + k21c36_tnr), 6)

k23c36_tpr <- round(sum(cm_c36_tables[1, 101:110]) /
  (sum(cm_c36_tables[1, 101:110]) + sum(cm_c36_tables[2, 101:110])), 6)
k23c36_tnr <- round(sum(cm_c36_tables[4, 101:110]) /
  (sum(cm_c36_tables[4, 101:110]) + sum(cm_c36_tables[3, 101:110])), 6)
k23c36_truescore <- round((2 * k23c36_tpr * k23c36_tnr) /
  (k23c36_tpr + k23c36_tnr), 6)

k25c36_tpr <- round(sum(cm_c36_tables[1, 111:120]) /
  (sum(cm_c36_tables[1, 111:120]) + sum(cm_c36_tables[2, 111:120])), 6)
k25c36_tnr <- round(sum(cm_c36_tables[4, 111:120]) /
  (sum(cm_c36_tables[4, 111:120]) + sum(cm_c36_tables[3, 111:120])), 6)
k25c36_truescore <- round((2 * k25c36_tpr * k25c36_tnr) /
  (k25c36_tpr + k25c36_tnr), 6)

k27c36_tpr <- round(sum(cm_c36_tables[1, 121:130]) /
  (sum(cm_c36_tables[1, 121:130]) + sum(cm_c36_tables[2, 121:130])), 6)
k27c36_tnr <- round(sum(cm_c36_tables[4, 121:130]) /
  (sum(cm_c36_tables[4, 121:130]) + sum(cm_c36_tables[3, 121:130])), 6)

```

```

k27c36_truescore <- round((2 * k27c36_tpr * k27c36_tnr) /
  (k27c36_tpr + k27c36_tnr), 6)

k29c36_tpr <- round(sum(cm_c36_tables[1, 131:140]) /
  (sum(cm_c36_tables[1, 131:140]) + sum(cm_c36_tables[2, 131:140])), 6)
k29c36_tnr <- round(sum(cm_c36_tables[4, 131:140]) /
  (sum(cm_c36_tables[4, 131:140]) + sum(cm_c36_tables[3, 131:140])), 6)
k29c36_truescore <- round((2 * k29c36_tpr * k29c36_tnr) /
  (k29c36_tpr + k29c36_tnr), 6)

k31c36_tpr <- round(sum(cm_c36_tables[1, 141:150]) /
  (sum(cm_c36_tables[1, 141:150]) + sum(cm_c36_tables[2, 141:150])), 6)
k31c36_tnr <- round(sum(cm_c36_tables[4, 141:150]) /
  (sum(cm_c36_tables[4, 141:150]) + sum(cm_c36_tables[3, 141:150])), 6)
k31c36_truescore <- round((2 * k31c36_tpr * k31c36_tnr) /
  (k31c36_tpr + k31c36_tnr), 6)

k33c36_tpr <- round(sum(cm_c36_tables[1, 151:160]) /
  (sum(cm_c36_tables[1, 151:160]) + sum(cm_c36_tables[2, 151:160])), 6)
k33c36_tnr <- round(sum(cm_c36_tables[4, 151:160]) /
  (sum(cm_c36_tables[4, 151:160]) + sum(cm_c36_tables[3, 151:160])), 6)
k33c36_truescore <- round((2 * k33c36_tpr * k33c36_tnr) /
  (k33c36_tpr + k33c36_tnr), 6)

k35c36_tpr <- round(sum(cm_c36_tables[1, 161:170]) /
  (sum(cm_c36_tables[1, 161:170]) + sum(cm_c36_tables[2, 161:170])), 6)
k35c36_tnr <- round(sum(cm_c36_tables[4, 161:170]) /
  (sum(cm_c36_tables[4, 161:170]) + sum(cm_c36_tables[3, 161:170])), 6)
k35c36_truescore <- round((2 * k35c36_tpr * k35c36_tnr) /
  (k35c36_tpr + k35c36_tnr), 6)

k37c36_tpr <- round(sum(cm_c36_tables[1, 171:180]) /
  (sum(cm_c36_tables[1, 171:180]) + sum(cm_c36_tables[2, 171:180])), 6)
k37c36_tnr <- round(sum(cm_c36_tables[4, 171:180]) /
  (sum(cm_c36_tables[4, 171:180]) + sum(cm_c36_tables[3, 171:180])), 6)
k37c36_truescore <- round((2 * k37c36_tpr * k37c36_tnr) /
  (k37c36_tpr + k37c36_tnr), 6)

k39c36_tpr <- round(sum(cm_c36_tables[1, 181:190]) /
  (sum(cm_c36_tables[1, 181:190]) + sum(cm_c36_tables[2, 181:190])), 6)
k39c36_tnr <- round(sum(cm_c36_tables[4, 181:190]) /
  (sum(cm_c36_tables[4, 181:190]) + sum(cm_c36_tables[3, 181:190])), 6)
k39c36_truescore <- round((2 * k39c36_tpr * k39c36_tnr) /
  (k39c36_tpr + k39c36_tnr), 6)

# Compile the 0.36 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c36_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.36,
  TPR = c(k3c36_tpr, k5c36_tpr, k7c36_tpr, k9c36_tpr, k11c36_tpr,
    k13c36_tpr, k15c36_tpr, k17c36_tpr, k19c36_tpr, k21c36_tpr,
    k23c36_tpr, k25c36_tpr, k27c36_tpr, k29c36_tpr, k31c36_tpr,
    k33c36_tpr, k35c36_tpr, k37c36_tpr, k39c36_tpr))

```

```

            k33c36_tpr, k35c36_tpr, k37c36_tpr, k39c36_tpr),
TNR = c(k3c36_tnr, k5c36_tnr, k7c36_tnr, k9c36_tnr, k11c36_tnr,
         k13c36_tnr, k15c36_tnr, k17c36_tnr, k19c36_tnr, k21c36_tnr,
         k23c36_tnr, k25c36_tnr, k27c36_tnr, k29c36_tnr, k31c36_tnr,
         k33c36_tnr, k35c36_tnr, k37c36_tnr, k39c36_tnr),
Truescore = c(k3c36_truescore, k5c36_truescore, k7c36_truescore,
              k9c36_truescore, k11c36_truescore, k13c36_truescore,
              k15c36_truescore, k17c36_truescore, k19c36_truescore,
              k21c36_truescore, k23c36_truescore, k25c36_truescore,
              k27c36_truescore, k29c36_truescore, k31c36_truescore,
              k33c36_truescore, k35c36_truescore, k37c36_truescore,
              k39c36_truescore))

knitr::kable(c36_results[1:19, ], caption = "c36_results")

```

Table 19: c36_results

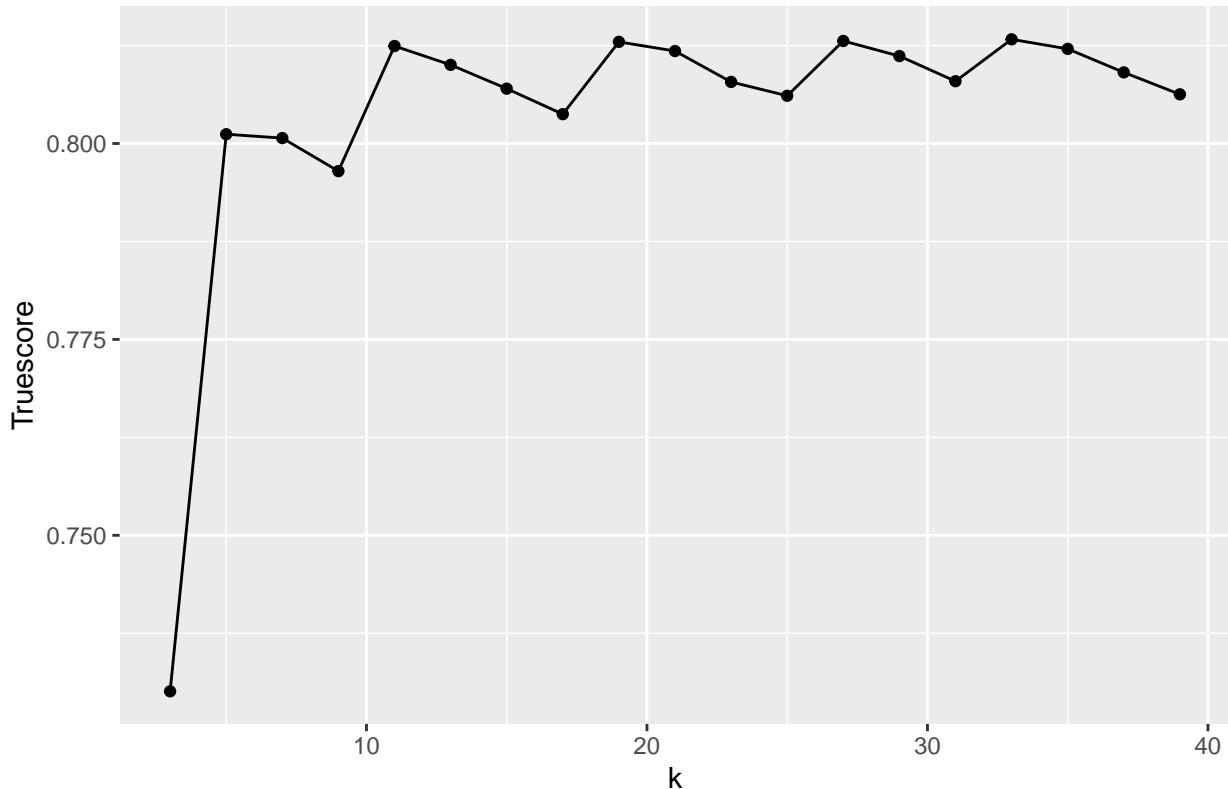
k	Cut	TPR	TNR	Truescore
3	0.36	0.616215	0.895600	0.730092
5	0.36	0.794428	0.808063	0.801187
7	0.36	0.764508	0.840468	0.800690
9	0.36	0.742520	0.858884	0.796474
11	0.36	0.809488	0.815413	0.812440
13	0.36	0.788454	0.832838	0.810038
15	0.36	0.772289	0.844977	0.807000
17	0.36	0.758484	0.854755	0.803747
19	0.36	0.799699	0.826694	0.812972
21	0.36	0.787851	0.837248	0.811799
23	0.36	0.774096	0.844697	0.807857
25	0.36	0.764960	0.851898	0.806092
27	0.36	0.795884	0.831054	0.813089
29	0.36	0.786295	0.837644	0.811158
31	0.36	0.775251	0.843524	0.807948
33	0.36	0.800803	0.826182	0.813295
35	0.36	0.792520	0.832623	0.812077
37	0.36	0.782781	0.837198	0.809076
39	0.36	0.773444	0.842021	0.806277

```

ggplot(c36_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.36")

```

Optimal k for Decision Cutoff 0.36



```
#####
# 0.37 Cutoff
#####

# For the decision cutoff of 0.37, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c37 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred37, obs))
  confusionMatrix(ss$pred37, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c37) <- fk_dfs_v

cm_c37_tables <- sapply(cm_c37, "[[", 2)
cm_c37_tables <- as_tibble(cm_c37_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.37.

k3c37_tpr <- round(sum(cm_c37_tables[1, 1:10]) /
  (sum(cm_c37_tables[1, 1:10]) + sum(cm_c37_tables[2, 1:10])), 6)
k3c37_tnr <- round(sum(cm_c37_tables[4, 1:10]) /
  (sum(cm_c37_tables[4, 1:10]) + sum(cm_c37_tables[3, 1:10])), 6)
```

```

k3c37_truescore <- round((2 * k3c37_tpr * k3c37_tnr) /
  (k3c37_tpr + k3c37_tnr), 6)

k5c37_tpr <- round(sum(cm_c37_tables[1, 11:20]) /
  (sum(cm_c37_tables[1, 11:20]) + sum(cm_c37_tables[2, 11:20])), 6)
k5c37_tnr <- round(sum(cm_c37_tables[4, 11:20]) /
  (sum(cm_c37_tables[4, 11:20]) + sum(cm_c37_tables[3, 11:20])), 6)
k5c37_truescore <- round((2 * k5c37_tpr * k5c37_tnr) /
  (k5c37_tpr + k5c37_tnr), 6)

k7c37_tpr <- round(sum(cm_c37_tables[1, 21:30]) /
  (sum(cm_c37_tables[1, 21:30]) + sum(cm_c37_tables[2, 21:30])), 6)
k7c37_tnr <- round(sum(cm_c37_tables[4, 21:30]) /
  (sum(cm_c37_tables[4, 21:30]) + sum(cm_c37_tables[3, 21:30])), 6)
k7c37_truescore <- round((2 * k7c37_tpr * k7c37_tnr) /
  (k7c37_tpr + k7c37_tnr), 6)

k9c37_tpr <- round(sum(cm_c37_tables[1, 31:40]) /
  (sum(cm_c37_tables[1, 31:40]) + sum(cm_c37_tables[2, 31:40])), 6)
k9c37_tnr <- round(sum(cm_c37_tables[4, 31:40]) /
  (sum(cm_c37_tables[4, 31:40]) + sum(cm_c37_tables[3, 31:40])), 6)
k9c37_truescore <- round((2 * k9c37_tpr * k9c37_tnr) /
  (k9c37_tpr + k9c37_tnr), 6)

k11c37_tpr <- round(sum(cm_c37_tables[1, 41:50]) /
  (sum(cm_c37_tables[1, 41:50]) + sum(cm_c37_tables[2, 41:50])), 6)
k11c37_tnr <- round(sum(cm_c37_tables[4, 41:50]) /
  (sum(cm_c37_tables[4, 41:50]) + sum(cm_c37_tables[3, 41:50])), 6)
k11c37_truescore <- round((2 * k11c37_tpr * k11c37_tnr) /
  (k11c37_tpr + k11c37_tnr), 6)

k13c37_tpr <- round(sum(cm_c37_tables[1, 51:60]) /
  (sum(cm_c37_tables[1, 51:60]) + sum(cm_c37_tables[2, 51:60])), 6)
k13c37_tnr <- round(sum(cm_c37_tables[4, 51:60]) /
  (sum(cm_c37_tables[4, 51:60]) + sum(cm_c37_tables[3, 51:60])), 6)
k13c37_truescore <- round((2 * k13c37_tpr * k13c37_tnr) /
  (k13c37_tpr + k13c37_tnr), 6)

k15c37_tpr <- round(sum(cm_c37_tables[1, 61:70]) /
  (sum(cm_c37_tables[1, 61:70]) + sum(cm_c37_tables[2, 61:70])), 6)
k15c37_tnr <- round(sum(cm_c37_tables[4, 61:70]) /
  (sum(cm_c37_tables[4, 61:70]) + sum(cm_c37_tables[3, 61:70])), 6)
k15c37_truescore <- round((2 * k15c37_tpr * k15c37_tnr) /
  (k15c37_tpr + k15c37_tnr), 6)

k17c37_tpr <- round(sum(cm_c37_tables[1, 71:80]) /
  (sum(cm_c37_tables[1, 71:80]) + sum(cm_c37_tables[2, 71:80])), 6)
k17c37_tnr <- round(sum(cm_c37_tables[4, 71:80]) /
  (sum(cm_c37_tables[4, 71:80]) + sum(cm_c37_tables[3, 71:80])), 6)
k17c37_truescore <- round((2 * k17c37_tpr * k17c37_tnr) /
  (k17c37_tpr + k17c37_tnr), 6)

k19c37_tpr <- round(sum(cm_c37_tables[1, 81:90]) /

```

```

(sum(cm_c37_tables[1, 81:90]) + sum(cm_c37_tables[2, 81:90])), 6)
k19c37_tnr <- round(sum(cm_c37_tables[4, 81:90]) /
  (sum(cm_c37_tables[4, 81:90]) + sum(cm_c37_tables[3, 81:90])), 6)
k19c37_truescore <- round((2 * k19c37_tpr * k19c37_tnr) /
  (k19c37_tpr + k19c37_tnr), 6)

k21c37_tpr <- round(sum(cm_c37_tables[1, 91:100]) /
  (sum(cm_c37_tables[1, 91:100]) + sum(cm_c37_tables[2, 91:100])), 6)
k21c37_tnr <- round(sum(cm_c37_tables[4, 91:100]) /
  (sum(cm_c37_tables[4, 91:100]) + sum(cm_c37_tables[3, 91:100])), 6)
k21c37_truescore <- round((2 * k21c37_tpr * k21c37_tnr) /
  (k21c37_tpr + k21c37_tnr), 6)

k23c37_tpr <- round(sum(cm_c37_tables[1, 101:110]) /
  (sum(cm_c37_tables[1, 101:110]) + sum(cm_c37_tables[2, 101:110])), 6)
k23c37_tnr <- round(sum(cm_c37_tables[4, 101:110]) /
  (sum(cm_c37_tables[4, 101:110]) + sum(cm_c37_tables[3, 101:110])), 6)
k23c37_truescore <- round((2 * k23c37_tpr * k23c37_tnr) /
  (k23c37_tpr + k23c37_tnr), 6)

k25c37_tpr <- round(sum(cm_c37_tables[1, 111:120]) /
  (sum(cm_c37_tables[1, 111:120]) + sum(cm_c37_tables[2, 111:120])), 6)
k25c37_tnr <- round(sum(cm_c37_tables[4, 111:120]) /
  (sum(cm_c37_tables[4, 111:120]) + sum(cm_c37_tables[3, 111:120])), 6)
k25c37_truescore <- round((2 * k25c37_tpr * k25c37_tnr) /
  (k25c37_tpr + k25c37_tnr), 6)

k27c37_tpr <- round(sum(cm_c37_tables[1, 121:130]) /
  (sum(cm_c37_tables[1, 121:130]) + sum(cm_c37_tables[2, 121:130])), 6)
k27c37_tnr <- round(sum(cm_c37_tables[4, 121:130]) /
  (sum(cm_c37_tables[4, 121:130]) + sum(cm_c37_tables[3, 121:130])), 6)
k27c37_truescore <- round((2 * k27c37_tpr * k27c37_tnr) /
  (k27c37_tpr + k27c37_tnr), 6)

k29c37_tpr <- round(sum(cm_c37_tables[1, 131:140]) /
  (sum(cm_c37_tables[1, 131:140]) + sum(cm_c37_tables[2, 131:140])), 6)
k29c37_tnr <- round(sum(cm_c37_tables[4, 131:140]) /
  (sum(cm_c37_tables[4, 131:140]) + sum(cm_c37_tables[3, 131:140])), 6)
k29c37_truescore <- round((2 * k29c37_tpr * k29c37_tnr) /
  (k29c37_tpr + k29c37_tnr), 6)

k31c37_tpr <- round(sum(cm_c37_tables[1, 141:150]) /
  (sum(cm_c37_tables[1, 141:150]) + sum(cm_c37_tables[2, 141:150])), 6)
k31c37_tnr <- round(sum(cm_c37_tables[4, 141:150]) /
  (sum(cm_c37_tables[4, 141:150]) + sum(cm_c37_tables[3, 141:150])), 6)
k31c37_truescore <- round((2 * k31c37_tpr * k31c37_tnr) /
  (k31c37_tpr + k31c37_tnr), 6)

k33c37_tpr <- round(sum(cm_c37_tables[1, 151:160]) /
  (sum(cm_c37_tables[1, 151:160]) + sum(cm_c37_tables[2, 151:160])), 6)
k33c37_tnr <- round(sum(cm_c37_tables[4, 151:160]) /
  (sum(cm_c37_tables[4, 151:160]) + sum(cm_c37_tables[3, 151:160])), 6)
k33c37_truescore <- round((2 * k33c37_tpr * k33c37_tnr) /

```

```

(k33c37_tpr + k33c37_tnr), 6)

k35c37_tpr <- round(sum(cm_c37_tables[1, 161:170]) /
  (sum(cm_c37_tables[1, 161:170]) + sum(cm_c37_tables[2, 161:170])), 6)
k35c37_tnr <- round(sum(cm_c37_tables[4, 161:170]) /
  (sum(cm_c37_tables[4, 161:170]) + sum(cm_c37_tables[3, 161:170])), 6)
k35c37_truescore <- round((2 * k35c37_tpr * k35c37_tnr) /
  (k35c37_tpr + k35c37_tnr), 6)

k37c37_tpr <- round(sum(cm_c37_tables[1, 171:180]) /
  (sum(cm_c37_tables[1, 171:180]) + sum(cm_c37_tables[2, 171:180])), 6)
k37c37_tnr <- round(sum(cm_c37_tables[4, 171:180]) /
  (sum(cm_c37_tables[4, 171:180]) + sum(cm_c37_tables[3, 171:180])), 6)
k37c37_truescore <- round((2 * k37c37_tpr * k37c37_tnr) /
  (k37c37_tpr + k37c37_tnr), 6)

k39c37_tpr <- round(sum(cm_c37_tables[1, 181:190]) /
  (sum(cm_c37_tables[1, 181:190]) + sum(cm_c37_tables[2, 181:190])), 6)
k39c37_tnr <- round(sum(cm_c37_tables[4, 181:190]) /
  (sum(cm_c37_tables[4, 181:190]) + sum(cm_c37_tables[3, 181:190])), 6)
k39c37_truescore <- round((2 * k39c37_tpr * k39c37_tnr) /
  (k39c37_tpr + k39c37_tnr), 6)

# Compile the 0.37 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c37_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.37,
  TPR = c(k3c37_tpr, k5c37_tpr, k7c37_tpr, k9c37_tpr, k11c37_tpr,
    k13c37_tpr, k15c37_tpr, k17c37_tpr, k19c37_tpr, k21c37_tpr,
    k23c37_tpr, k25c37_tpr, k27c37_tpr, k29c37_tpr, k31c37_tpr,
    k33c37_tpr, k35c37_tpr, k37c37_tpr, k39c37_tpr),
  TNR = c(k3c37_tnr, k5c37_tnr, k7c37_tnr, k9c37_tnr, k11c37_tnr,
    k13c37_tnr, k15c37_tnr, k17c37_tnr, k19c37_tnr, k21c37_tnr,
    k23c37_tnr, k25c37_tnr, k27c37_tnr, k29c37_tnr, k31c37_tnr,
    k33c37_tnr, k35c37_tnr, k37c37_tnr, k39c37_tnr),
  Truescore = c(k3c37_truescore, k5c37_truescore, k7c37_truescore,
    k9c37_truescore, k11c37_truescore, k13c37_truescore,
    k15c37_truescore, k17c37_truescore, k19c37_truescore,
    k21c37_truescore, k23c37_truescore, k25c37_truescore,
    k27c37_truescore, k29c37_truescore, k31c37_truescore,
    k33c37_truescore, k35c37_truescore, k37c37_truescore,
    k39c37_truescore))

knitr::kable(c37_results[1:19, ], caption = "c37_results")

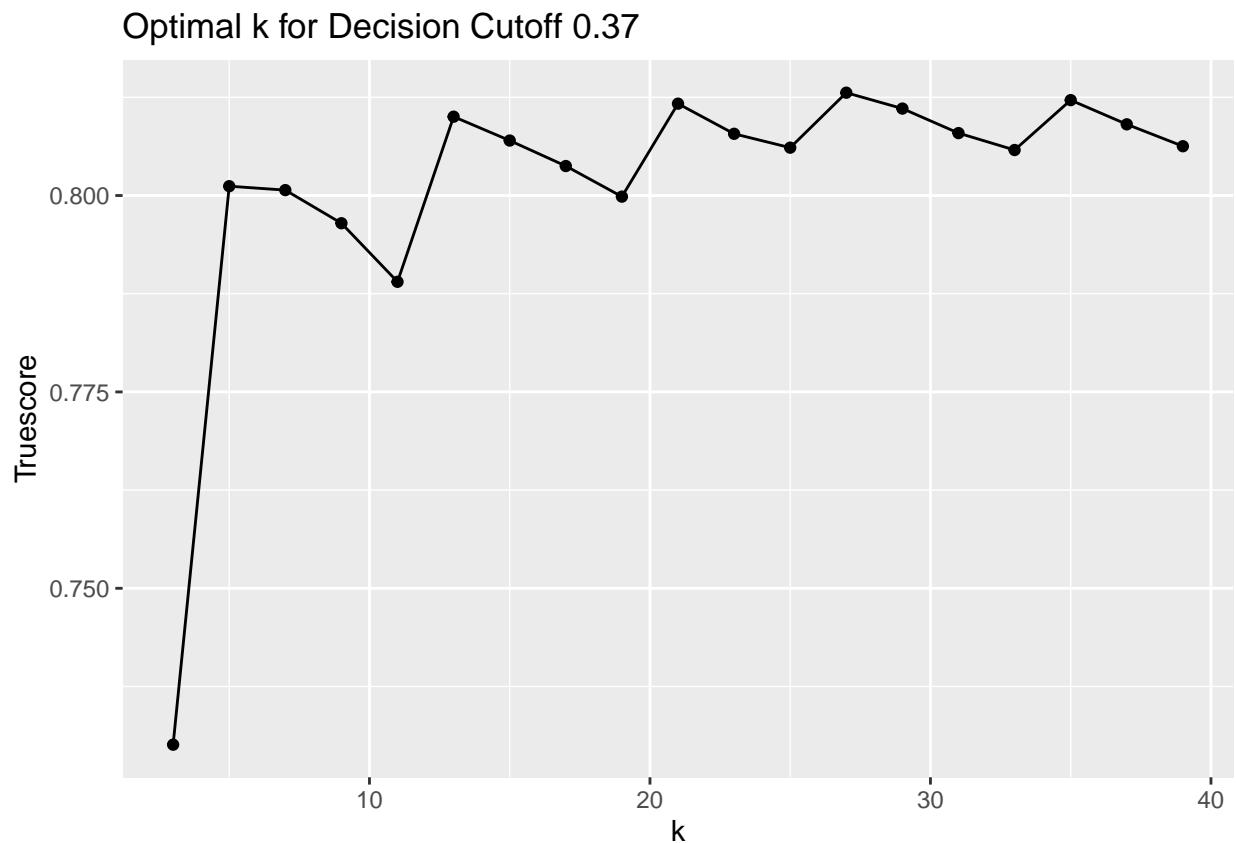
```

Table 20: c37_results

k	Cut	TPR	TNR	Truescore
3	0.37	0.616215	0.895600	0.730092
5	0.37	0.794428	0.808063	0.801187
7	0.37	0.764508	0.840468	0.800690
9	0.37	0.742520	0.858884	0.796474

k	Cut	TPR	TNR	Truescore
11	0.37	0.721285	0.870792	0.789019
13	0.37	0.788454	0.832838	0.810038
15	0.37	0.772289	0.844977	0.807000
17	0.37	0.758484	0.854772	0.803755
19	0.37	0.745532	0.862732	0.799862
21	0.37	0.787550	0.837363	0.811693
23	0.37	0.774096	0.844697	0.807857
25	0.37	0.764960	0.851898	0.806092
27	0.37	0.795884	0.831054	0.813089
29	0.37	0.786044	0.837743	0.811070
31	0.37	0.775251	0.843524	0.807948
33	0.37	0.766918	0.848826	0.805796
35	0.37	0.792470	0.832821	0.812145
37	0.37	0.782631	0.837347	0.809065
39	0.37	0.773444	0.842021	0.806277

```
ggplot(c37_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.37")
```



```
#####
# 0.38 Cutoff
#####
```

```

# For the decision cutoff of 0.38, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c38 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred38, obs))
  confusionMatrix(ss$pred38, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c38) <- fk_dfs_v

cm_c38_tables <- sapply(cm_c38, "[[", 2)
cm_c38_tables <- as_tibble(cm_c38_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.38.

k3c38_tpr <- round(sum(cm_c38_tables[1, 1:10]) /
  (sum(cm_c38_tables[1, 1:10]) + sum(cm_c38_tables[2, 1:10])), 6)
k3c38_tnr <- round(sum(cm_c38_tables[4, 1:10]) /
  (sum(cm_c38_tables[4, 1:10]) + sum(cm_c38_tables[3, 1:10])), 6)
k3c38_truescore <- round((2 * k3c38_tpr * k3c38_tnr) /
  (k3c38_tpr + k3c38_tnr), 6)

k5c38_tpr <- round(sum(cm_c38_tables[1, 11:20]) /
  (sum(cm_c38_tables[1, 11:20]) + sum(cm_c38_tables[2, 11:20])), 6)
k5c38_tnr <- round(sum(cm_c38_tables[4, 11:20]) /
  (sum(cm_c38_tables[4, 11:20]) + sum(cm_c38_tables[3, 11:20])), 6)
k5c38_truescore <- round((2 * k5c38_tpr * k5c38_tnr) /
  (k5c38_tpr + k5c38_tnr), 6)

k7c38_tpr <- round(sum(cm_c38_tables[1, 21:30]) /
  (sum(cm_c38_tables[1, 21:30]) + sum(cm_c38_tables[2, 21:30])), 6)
k7c38_tnr <- round(sum(cm_c38_tables[4, 21:30]) /
  (sum(cm_c38_tables[4, 21:30]) + sum(cm_c38_tables[3, 21:30])), 6)
k7c38_truescore <- round((2 * k7c38_tpr * k7c38_tnr) /
  (k7c38_tpr + k7c38_tnr), 6)

k9c38_tpr <- round(sum(cm_c38_tables[1, 31:40]) /
  (sum(cm_c38_tables[1, 31:40]) + sum(cm_c38_tables[2, 31:40])), 6)
k9c38_tnr <- round(sum(cm_c38_tables[4, 31:40]) /
  (sum(cm_c38_tables[4, 31:40]) + sum(cm_c38_tables[3, 31:40])), 6)
k9c38_truescore <- round((2 * k9c38_tpr * k9c38_tnr) /
  (k9c38_tpr + k9c38_tnr), 6)

k11c38_tpr <- round(sum(cm_c38_tables[1, 41:50]) /
  (sum(cm_c38_tables[1, 41:50]) + sum(cm_c38_tables[2, 41:50])), 6)
k11c38_tnr <- round(sum(cm_c38_tables[4, 41:50]) /
  (sum(cm_c38_tables[4, 41:50]) + sum(cm_c38_tables[3, 41:50])), 6)
k11c38_truescore <- round((2 * k11c38_tpr * k11c38_tnr) /
  (k11c38_tpr + k11c38_tnr), 6)

```

```

k13c38_tpr <- round(sum(cm_c38_tables[1, 51:60]) /
  (sum(cm_c38_tables[1, 51:60]) + sum(cm_c38_tables[2, 51:60])), 6)
k13c38_tnr <- round(sum(cm_c38_tables[4, 51:60]) /
  (sum(cm_c38_tables[4, 51:60]) + sum(cm_c38_tables[3, 51:60])), 6)
k13c38_truescore <- round((2 * k13c38_tpr * k13c38_tnr) /
  (k13c38_tpr + k13c38_tnr), 6)

k15c38_tpr <- round(sum(cm_c38_tables[1, 61:70]) /
  (sum(cm_c38_tables[1, 61:70]) + sum(cm_c38_tables[2, 61:70])), 6)
k15c38_tnr <- round(sum(cm_c38_tables[4, 61:70]) /
  (sum(cm_c38_tables[4, 61:70]) + sum(cm_c38_tables[3, 61:70])), 6)
k15c38_truescore <- round((2 * k15c38_tpr * k15c38_tnr) /
  (k15c38_tpr + k15c38_tnr), 6)

k17c38_tpr <- round(sum(cm_c38_tables[1, 71:80]) /
  (sum(cm_c38_tables[1, 71:80]) + sum(cm_c38_tables[2, 71:80])), 6)
k17c38_tnr <- round(sum(cm_c38_tables[4, 71:80]) /
  (sum(cm_c38_tables[4, 71:80]) + sum(cm_c38_tables[3, 71:80])), 6)
k17c38_truescore <- round((2 * k17c38_tpr * k17c38_tnr) /
  (k17c38_tpr + k17c38_tnr), 6)

k19c38_tpr <- round(sum(cm_c38_tables[1, 81:90]) /
  (sum(cm_c38_tables[1, 81:90]) + sum(cm_c38_tables[2, 81:90])), 6)
k19c38_tnr <- round(sum(cm_c38_tables[4, 81:90]) /
  (sum(cm_c38_tables[4, 81:90]) + sum(cm_c38_tables[3, 81:90])), 6)
k19c38_truescore <- round((2 * k19c38_tpr * k19c38_tnr) /
  (k19c38_tpr + k19c38_tnr), 6)

k21c38_tpr <- round(sum(cm_c38_tables[1, 91:100]) /
  (sum(cm_c38_tables[1, 91:100]) + sum(cm_c38_tables[2, 91:100])), 6)
k21c38_tnr <- round(sum(cm_c38_tables[4, 91:100]) /
  (sum(cm_c38_tables[4, 91:100]) + sum(cm_c38_tables[3, 91:100])), 6)
k21c38_truescore <- round((2 * k21c38_tpr * k21c38_tnr) /
  (k21c38_tpr + k21c38_tnr), 6)

k23c38_tpr <- round(sum(cm_c38_tables[1, 101:110]) /
  (sum(cm_c38_tables[1, 101:110]) + sum(cm_c38_tables[2, 101:110])), 6)
k23c38_tnr <- round(sum(cm_c38_tables[4, 101:110]) /
  (sum(cm_c38_tables[4, 101:110]) + sum(cm_c38_tables[3, 101:110])), 6)
k23c38_truescore <- round((2 * k23c38_tpr * k23c38_tnr) /
  (k23c38_tpr + k23c38_tnr), 6)

k25c38_tpr <- round(sum(cm_c38_tables[1, 111:120]) /
  (sum(cm_c38_tables[1, 111:120]) + sum(cm_c38_tables[2, 111:120])), 6)
k25c38_tnr <- round(sum(cm_c38_tables[4, 111:120]) /
  (sum(cm_c38_tables[4, 111:120]) + sum(cm_c38_tables[3, 111:120])), 6)
k25c38_truescore <- round((2 * k25c38_tpr * k25c38_tnr) /
  (k25c38_tpr + k25c38_tnr), 6)

k27c38_tpr <- round(sum(cm_c38_tables[1, 121:130]) /
  (sum(cm_c38_tables[1, 121:130]) + sum(cm_c38_tables[2, 121:130])), 6)
k27c38_tnr <- round(sum(cm_c38_tables[4, 121:130]) /
  (sum(cm_c38_tables[4, 121:130]) + sum(cm_c38_tables[3, 121:130])), 6)

```

```

k27c38_truescore <- round((2 * k27c38_tpr * k27c38_tnr) /
  (k27c38_tpr + k27c38_tnr), 6)

k29c38_tpr <- round(sum(cm_c38_tables[1, 131:140]) /
  (sum(cm_c38_tables[1, 131:140]) + sum(cm_c38_tables[2, 131:140])), 6)
k29c38_tnr <- round(sum(cm_c38_tables[4, 131:140]) /
  (sum(cm_c38_tables[4, 131:140]) + sum(cm_c38_tables[3, 131:140])), 6)
k29c38_truescore <- round((2 * k29c38_tpr * k29c38_tnr) /
  (k29c38_tpr + k29c38_tnr), 6)

k31c38_tpr <- round(sum(cm_c38_tables[1, 141:150]) /
  (sum(cm_c38_tables[1, 141:150]) + sum(cm_c38_tables[2, 141:150])), 6)
k31c38_tnr <- round(sum(cm_c38_tables[4, 141:150]) /
  (sum(cm_c38_tables[4, 141:150]) + sum(cm_c38_tables[3, 141:150])), 6)
k31c38_truescore <- round((2 * k31c38_tpr * k31c38_tnr) /
  (k31c38_tpr + k31c38_tnr), 6)

k33c38_tpr <- round(sum(cm_c38_tables[1, 151:160]) /
  (sum(cm_c38_tables[1, 151:160]) + sum(cm_c38_tables[2, 151:160])), 6)
k33c38_tnr <- round(sum(cm_c38_tables[4, 151:160]) /
  (sum(cm_c38_tables[4, 151:160]) + sum(cm_c38_tables[3, 151:160])), 6)
k33c38_truescore <- round((2 * k33c38_tpr * k33c38_tnr) /
  (k33c38_tpr + k33c38_tnr), 6)

k35c38_tpr <- round(sum(cm_c38_tables[1, 161:170]) /
  (sum(cm_c38_tables[1, 161:170]) + sum(cm_c38_tables[2, 161:170])), 6)
k35c38_tnr <- round(sum(cm_c38_tables[4, 161:170]) /
  (sum(cm_c38_tables[4, 161:170]) + sum(cm_c38_tables[3, 161:170])), 6)
k35c38_truescore <- round((2 * k35c38_tpr * k35c38_tnr) /
  (k35c38_tpr + k35c38_tnr), 6)

k37c38_tpr <- round(sum(cm_c38_tables[1, 171:180]) /
  (sum(cm_c38_tables[1, 171:180]) + sum(cm_c38_tables[2, 171:180])), 6)
k37c38_tnr <- round(sum(cm_c38_tables[4, 171:180]) /
  (sum(cm_c38_tables[4, 171:180]) + sum(cm_c38_tables[3, 171:180])), 6)
k37c38_truescore <- round((2 * k37c38_tpr * k37c38_tnr) /
  (k37c38_tpr + k37c38_tnr), 6)

k39c38_tpr <- round(sum(cm_c38_tables[1, 181:190]) /
  (sum(cm_c38_tables[1, 181:190]) + sum(cm_c38_tables[2, 181:190])), 6)
k39c38_tnr <- round(sum(cm_c38_tables[4, 181:190]) /
  (sum(cm_c38_tables[4, 181:190]) + sum(cm_c38_tables[3, 181:190])), 6)
k39c38_truescore <- round((2 * k39c38_tpr * k39c38_tnr) /
  (k39c38_tpr + k39c38_tnr), 6)

# Compile the 0.38 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c38_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.38,
  TPR = c(k3c38_tpr, k5c38_tpr, k7c38_tpr, k9c38_tpr, k11c38_tpr,
    k13c38_tpr, k15c38_tpr, k17c38_tpr, k19c38_tpr, k21c38_tpr,
    k23c38_tpr, k25c38_tpr, k27c38_tpr, k29c38_tpr, k31c38_tpr,
    k33c38_tpr, k35c38_tpr, k37c38_tpr, k39c38_tpr))

```

```

            k33c38_tpr, k35c38_tpr, k37c38_tpr, k39c38_tpr),
TNR = c(k3c38_tnr, k5c38_tnr, k7c38_tnr, k9c38_tnr, k11c38_tnr,
         k13c38_tnr, k15c38_tnr, k17c38_tnr, k19c38_tnr, k21c38_tnr,
         k23c38_tnr, k25c38_tnr, k27c38_tnr, k29c38_tnr, k31c38_tnr,
         k33c38_tnr, k35c38_tnr, k37c38_tnr, k39c38_tnr),
Truescore = c(k3c38_truescore, k5c38_truescore, k7c38_truescore,
              k9c38_truescore, k11c38_truescore, k13c38_truescore,
              k15c38_truescore, k17c38_truescore, k19c38_truescore,
              k21c38_truescore, k23c38_truescore, k25c38_truescore,
              k27c38_truescore, k29c38_truescore, k31c38_truescore,
              k33c38_truescore, k35c38_truescore, k37c38_truescore,
              k39c38_truescore))

knitr::kable(c38_results[1:19, ], caption = "c38_results")

```

Table 21: c38_results

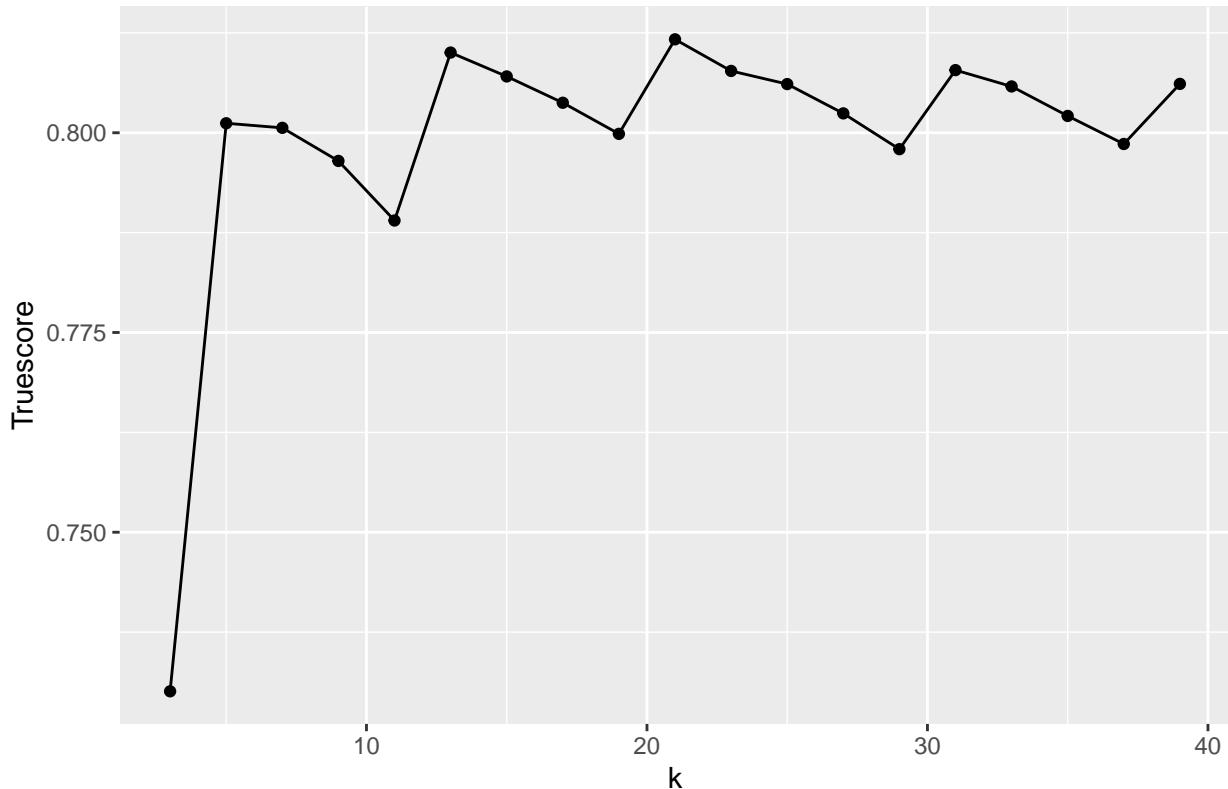
k	Cut	TPR	TNR	Truescore
3	0.38	0.616215	0.895600	0.730092
5	0.38	0.794428	0.808063	0.801187
7	0.38	0.764207	0.840667	0.800616
9	0.38	0.742520	0.858884	0.796474
11	0.38	0.721285	0.870792	0.789019
13	0.38	0.788454	0.832838	0.810038
15	0.38	0.772239	0.845143	0.807048
17	0.38	0.758484	0.854772	0.803755
19	0.38	0.745532	0.862732	0.799862
21	0.38	0.787550	0.837363	0.811693
23	0.38	0.773795	0.844796	0.807738
25	0.38	0.764960	0.851898	0.806092
27	0.38	0.754217	0.857216	0.802425
29	0.38	0.742972	0.861708	0.797947
31	0.38	0.774900	0.843706	0.807841
33	0.38	0.766918	0.848826	0.805796
35	0.38	0.757329	0.852509	0.802105
37	0.38	0.747741	0.856869	0.798594
39	0.38	0.773092	0.842087	0.806116

```

ggplot(c38_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.38")

```

Optimal k for Decision Cutoff 0.38



```
#####
# 0.39 Cutoff
#####

# For the decision cutoff of 0.39, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c39 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred39, obs))
  confusionMatrix(ss$pred39, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c39) <- fk_dfs_v

cm_c39_tables <- sapply(cm_c39, "[[", 2)
cm_c39_tables <- as_tibble(cm_c39_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.39.

k3c39_tpr <- round(sum(cm_c39_tables[1, 1:10]) /
  (sum(cm_c39_tables[1, 1:10]) + sum(cm_c39_tables[2, 1:10])), 6)
k3c39_tnr <- round(sum(cm_c39_tables[4, 1:10]) /
  (sum(cm_c39_tables[4, 1:10]) + sum(cm_c39_tables[3, 1:10])), 6)
```

```

k3c39_truescore <- round((2 * k3c39_tpr * k3c39_tnr) /
  (k3c39_tpr + k3c39_tnr), 6)

k5c39_tpr <- round(sum(cm_c39_tables[1, 11:20]) /
  (sum(cm_c39_tables[1, 11:20]) + sum(cm_c39_tables[2, 11:20])), 6)
k5c39_tnr <- round(sum(cm_c39_tables[4, 11:20]) /
  (sum(cm_c39_tables[4, 11:20]) + sum(cm_c39_tables[3, 11:20])), 6)
k5c39_truescore <- round((2 * k5c39_tpr * k5c39_tnr) /
  (k5c39_tpr + k5c39_tnr), 6)

k7c39_tpr <- round(sum(cm_c39_tables[1, 21:30]) /
  (sum(cm_c39_tables[1, 21:30]) + sum(cm_c39_tables[2, 21:30])), 6)
k7c39_tnr <- round(sum(cm_c39_tables[4, 21:30]) /
  (sum(cm_c39_tables[4, 21:30]) + sum(cm_c39_tables[3, 21:30])), 6)
k7c39_truescore <- round((2 * k7c39_tpr * k7c39_tnr) /
  (k7c39_tpr + k7c39_tnr), 6)

k9c39_tpr <- round(sum(cm_c39_tables[1, 31:40]) /
  (sum(cm_c39_tables[1, 31:40]) + sum(cm_c39_tables[2, 31:40])), 6)
k9c39_tnr <- round(sum(cm_c39_tables[4, 31:40]) /
  (sum(cm_c39_tables[4, 31:40]) + sum(cm_c39_tables[3, 31:40])), 6)
k9c39_truescore <- round((2 * k9c39_tpr * k9c39_tnr) /
  (k9c39_tpr + k9c39_tnr), 6)

k11c39_tpr <- round(sum(cm_c39_tables[1, 41:50]) /
  (sum(cm_c39_tables[1, 41:50]) + sum(cm_c39_tables[2, 41:50])), 6)
k11c39_tnr <- round(sum(cm_c39_tables[4, 41:50]) /
  (sum(cm_c39_tables[4, 41:50]) + sum(cm_c39_tables[3, 41:50])), 6)
k11c39_truescore <- round((2 * k11c39_tpr * k11c39_tnr) /
  (k11c39_tpr + k11c39_tnr), 6)

k13c39_tpr <- round(sum(cm_c39_tables[1, 51:60]) /
  (sum(cm_c39_tables[1, 51:60]) + sum(cm_c39_tables[2, 51:60])), 6)
k13c39_tnr <- round(sum(cm_c39_tables[4, 51:60]) /
  (sum(cm_c39_tables[4, 51:60]) + sum(cm_c39_tables[3, 51:60])), 6)
k13c39_truescore <- round((2 * k13c39_tpr * k13c39_tnr) /
  (k13c39_tpr + k13c39_tnr), 6)

k15c39_tpr <- round(sum(cm_c39_tables[1, 61:70]) /
  (sum(cm_c39_tables[1, 61:70]) + sum(cm_c39_tables[2, 61:70])), 6)
k15c39_tnr <- round(sum(cm_c39_tables[4, 61:70]) /
  (sum(cm_c39_tables[4, 61:70]) + sum(cm_c39_tables[3, 61:70])), 6)
k15c39_truescore <- round((2 * k15c39_tpr * k15c39_tnr) /
  (k15c39_tpr + k15c39_tnr), 6)

k17c39_tpr <- round(sum(cm_c39_tables[1, 71:80]) /
  (sum(cm_c39_tables[1, 71:80]) + sum(cm_c39_tables[2, 71:80])), 6)
k17c39_tnr <- round(sum(cm_c39_tables[4, 71:80]) /
  (sum(cm_c39_tables[4, 71:80]) + sum(cm_c39_tables[3, 71:80])), 6)
k17c39_truescore <- round((2 * k17c39_tpr * k17c39_tnr) /
  (k17c39_tpr + k17c39_tnr), 6)

k19c39_tpr <- round(sum(cm_c39_tables[1, 81:90]) /

```

```

(sum(cm_c39_tables[1, 81:90]) + sum(cm_c39_tables[2, 81:90])), 6)
k19c39_tnr <- round(sum(cm_c39_tables[4, 81:90]) /
  (sum(cm_c39_tables[4, 81:90]) + sum(cm_c39_tables[3, 81:90])), 6)
k19c39_truescore <- round((2 * k19c39_tpr * k19c39_tnr) /
  (k19c39_tpr + k19c39_tnr), 6)

k21c39_tpr <- round(sum(cm_c39_tables[1, 91:100]) /
  (sum(cm_c39_tables[1, 91:100]) + sum(cm_c39_tables[2, 91:100])), 6)
k21c39_tnr <- round(sum(cm_c39_tables[4, 91:100]) /
  (sum(cm_c39_tables[4, 91:100]) + sum(cm_c39_tables[3, 91:100])), 6)
k21c39_truescore <- round((2 * k21c39_tpr * k21c39_tnr) /
  (k21c39_tpr + k21c39_tnr), 6)

k23c39_tpr <- round(sum(cm_c39_tables[1, 101:110]) /
  (sum(cm_c39_tables[1, 101:110]) + sum(cm_c39_tables[2, 101:110])), 6)
k23c39_tnr <- round(sum(cm_c39_tables[4, 101:110]) /
  (sum(cm_c39_tables[4, 101:110]) + sum(cm_c39_tables[3, 101:110])), 6)
k23c39_truescore <- round((2 * k23c39_tpr * k23c39_tnr) /
  (k23c39_tpr + k23c39_tnr), 6)

k25c39_tpr <- round(sum(cm_c39_tables[1, 111:120]) /
  (sum(cm_c39_tables[1, 111:120]) + sum(cm_c39_tables[2, 111:120])), 6)
k25c39_tnr <- round(sum(cm_c39_tables[4, 111:120]) /
  (sum(cm_c39_tables[4, 111:120]) + sum(cm_c39_tables[3, 111:120])), 6)
k25c39_truescore <- round((2 * k25c39_tpr * k25c39_tnr) /
  (k25c39_tpr + k25c39_tnr), 6)

k27c39_tpr <- round(sum(cm_c39_tables[1, 121:130]) /
  (sum(cm_c39_tables[1, 121:130]) + sum(cm_c39_tables[2, 121:130])), 6)
k27c39_tnr <- round(sum(cm_c39_tables[4, 121:130]) /
  (sum(cm_c39_tables[4, 121:130]) + sum(cm_c39_tables[3, 121:130])), 6)
k27c39_truescore <- round((2 * k27c39_tpr * k27c39_tnr) /
  (k27c39_tpr + k27c39_tnr), 6)

k29c39_tpr <- round(sum(cm_c39_tables[1, 131:140]) /
  (sum(cm_c39_tables[1, 131:140]) + sum(cm_c39_tables[2, 131:140])), 6)
k29c39_tnr <- round(sum(cm_c39_tables[4, 131:140]) /
  (sum(cm_c39_tables[4, 131:140]) + sum(cm_c39_tables[3, 131:140])), 6)
k29c39_truescore <- round((2 * k29c39_tpr * k29c39_tnr) /
  (k29c39_tpr + k29c39_tnr), 6)

k31c39_tpr <- round(sum(cm_c39_tables[1, 141:150]) /
  (sum(cm_c39_tables[1, 141:150]) + sum(cm_c39_tables[2, 141:150])), 6)
k31c39_tnr <- round(sum(cm_c39_tables[4, 141:150]) /
  (sum(cm_c39_tables[4, 141:150]) + sum(cm_c39_tables[3, 141:150])), 6)
k31c39_truescore <- round((2 * k31c39_tpr * k31c39_tnr) /
  (k31c39_tpr + k31c39_tnr), 6)

k33c39_tpr <- round(sum(cm_c39_tables[1, 151:160]) /
  (sum(cm_c39_tables[1, 151:160]) + sum(cm_c39_tables[2, 151:160])), 6)
k33c39_tnr <- round(sum(cm_c39_tables[4, 151:160]) /
  (sum(cm_c39_tables[4, 151:160]) + sum(cm_c39_tables[3, 151:160])), 6)
k33c39_truescore <- round((2 * k33c39_tpr * k33c39_tnr) /

```

```

(k33c39_tpr + k33c39_tnr), 6)

k35c39_tpr <- round(sum(cm_c39_tables[1, 161:170]) /
  (sum(cm_c39_tables[1, 161:170]) + sum(cm_c39_tables[2, 161:170])), 6)
k35c39_tnr <- round(sum(cm_c39_tables[4, 161:170]) /
  (sum(cm_c39_tables[4, 161:170]) + sum(cm_c39_tables[3, 161:170])), 6)
k35c39_truescore <- round((2 * k35c39_tpr * k35c39_tnr) /
  (k35c39_tpr + k35c39_tnr), 6)

k37c39_tpr <- round(sum(cm_c39_tables[1, 171:180]) /
  (sum(cm_c39_tables[1, 171:180]) + sum(cm_c39_tables[2, 171:180])), 6)
k37c39_tnr <- round(sum(cm_c39_tables[4, 171:180]) /
  (sum(cm_c39_tables[4, 171:180]) + sum(cm_c39_tables[3, 171:180])), 6)
k37c39_truescore <- round((2 * k37c39_tpr * k37c39_tnr) /
  (k37c39_tpr + k37c39_tnr), 6)

k39c39_tpr <- round(sum(cm_c39_tables[1, 181:190]) /
  (sum(cm_c39_tables[1, 181:190]) + sum(cm_c39_tables[2, 181:190])), 6)
k39c39_tnr <- round(sum(cm_c39_tables[4, 181:190]) /
  (sum(cm_c39_tables[4, 181:190]) + sum(cm_c39_tables[3, 181:190])), 6)
k39c39_truescore <- round((2 * k39c39_tpr * k39c39_tnr) /
  (k39c39_tpr + k39c39_tnr), 6)

# Compile the 0.39 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c39_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.39,
  TPR = c(k3c39_tpr, k5c39_tpr, k7c39_tpr, k9c39_tpr, k11c39_tpr,
    k13c39_tpr, k15c39_tpr, k17c39_tpr, k19c39_tpr, k21c39_tpr,
    k23c39_tpr, k25c39_tpr, k27c39_tpr, k29c39_tpr, k31c39_tpr,
    k33c39_tpr, k35c39_tpr, k37c39_tpr, k39c39_tpr),
  TNR = c(k3c39_tnr, k5c39_tnr, k7c39_tnr, k9c39_tnr, k11c39_tnr,
    k13c39_tnr, k15c39_tnr, k17c39_tnr, k19c39_tnr, k21c39_tnr,
    k23c39_tnr, k25c39_tnr, k27c39_tnr, k29c39_tnr, k31c39_tnr,
    k33c39_tnr, k35c39_tnr, k37c39_tnr, k39c39_tnr),
  Truescore = c(k3c39_truescore, k5c39_truescore, k7c39_truescore,
    k9c39_truescore, k11c39_truescore, k13c39_truescore,
    k15c39_truescore, k17c39_truescore, k19c39_truescore,
    k21c39_truescore, k23c39_truescore, k25c39_truescore,
    k27c39_truescore, k29c39_truescore, k31c39_truescore,
    k33c39_truescore, k35c39_truescore, k37c39_truescore,
    k39c39_truescore))

knitr::kable(c39_results[1:19, ], caption = "c39_results")

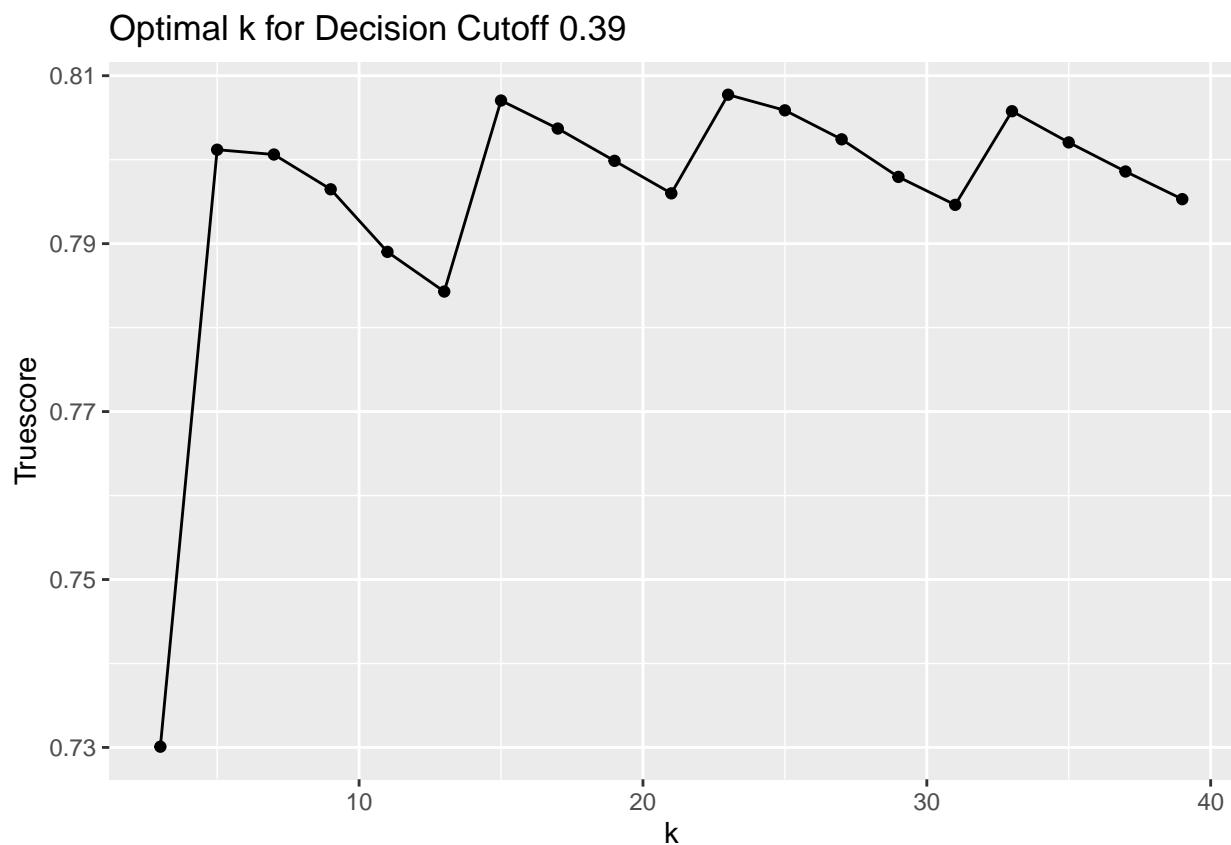
```

Table 22: c39_results

k	Cut	TPR	TNR	Truescore
3	0.39	0.616215	0.895600	0.730092
5	0.39	0.794428	0.808063	0.801187
7	0.39	0.764207	0.840667	0.800616
9	0.39	0.742520	0.858884	0.796474

k	Cut	TPR	TNR	Truescore
11	0.39	0.721285	0.870792	0.789019
13	0.39	0.708785	0.877812	0.784295
15	0.39	0.772239	0.845143	0.807048
17	0.39	0.758283	0.854920	0.803707
19	0.39	0.745532	0.862732	0.799862
21	0.39	0.734538	0.868662	0.795990
23	0.39	0.773795	0.844796	0.807738
25	0.39	0.764458	0.852046	0.805879
27	0.39	0.754217	0.857216	0.802425
29	0.39	0.742972	0.861708	0.797947
31	0.39	0.734388	0.865623	0.794623
33	0.39	0.766717	0.849024	0.805774
35	0.39	0.757179	0.852591	0.802057
37	0.39	0.747741	0.856869	0.798594
39	0.39	0.739257	0.860536	0.795299

```
ggplot(c39_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.39")
```



```
#####
# 0.40 Cutoff
#####
```

```

# For the decision cutoff of 0.40, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c40 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred40, obs))
  confusionMatrix(ss$pred40, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c40) <- fk_dfs_v

cm_c40_tables <- sapply(cm_c40, "[[", 2)
cm_c40_tables <- as_tibble(cm_c40_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.40.

k3c40_tpr <- round(sum(cm_c40_tables[1, 1:10]) /
  (sum(cm_c40_tables[1, 1:10]) + sum(cm_c40_tables[2, 1:10])), 6)
k3c40_tnr <- round(sum(cm_c40_tables[4, 1:10]) /
  (sum(cm_c40_tables[4, 1:10]) + sum(cm_c40_tables[3, 1:10])), 6)
k3c40_truescore <- round((2 * k3c40_tpr * k3c40_tnr) /
  (k3c40_tpr + k3c40_tnr), 6)

k5c40_tpr <- round(sum(cm_c40_tables[1, 11:20]) /
  (sum(cm_c40_tables[1, 11:20]) + sum(cm_c40_tables[2, 11:20])), 6)
k5c40_tnr <- round(sum(cm_c40_tables[4, 11:20]) /
  (sum(cm_c40_tables[4, 11:20]) + sum(cm_c40_tables[3, 11:20])), 6)
k5c40_truescore <- round((2 * k5c40_tpr * k5c40_tnr) /
  (k5c40_tpr + k5c40_tnr), 6)

k7c40_tpr <- round(sum(cm_c40_tables[1, 21:30]) /
  (sum(cm_c40_tables[1, 21:30]) + sum(cm_c40_tables[2, 21:30])), 6)
k7c40_tnr <- round(sum(cm_c40_tables[4, 21:30]) /
  (sum(cm_c40_tables[4, 21:30]) + sum(cm_c40_tables[3, 21:30])), 6)
k7c40_truescore <- round((2 * k7c40_tpr * k7c40_tnr) /
  (k7c40_tpr + k7c40_tnr), 6)

k9c40_tpr <- round(sum(cm_c40_tables[1, 31:40]) /
  (sum(cm_c40_tables[1, 31:40]) + sum(cm_c40_tables[2, 31:40])), 6)
k9c40_tnr <- round(sum(cm_c40_tables[4, 31:40]) /
  (sum(cm_c40_tables[4, 31:40]) + sum(cm_c40_tables[3, 31:40])), 6)
k9c40_truescore <- round((2 * k9c40_tpr * k9c40_tnr) /
  (k9c40_tpr + k9c40_tnr), 6)

k11c40_tpr <- round(sum(cm_c40_tables[1, 41:50]) /
  (sum(cm_c40_tables[1, 41:50]) + sum(cm_c40_tables[2, 41:50])), 6)
k11c40_tnr <- round(sum(cm_c40_tables[4, 41:50]) /
  (sum(cm_c40_tables[4, 41:50]) + sum(cm_c40_tables[3, 41:50])), 6)
k11c40_truescore <- round((2 * k11c40_tpr * k11c40_tnr) /
  (k11c40_tpr + k11c40_tnr), 6)

```

```

k13c40_tpr <- round(sum(cm_c40_tables[1, 51:60]) /
  (sum(cm_c40_tables[1, 51:60]) + sum(cm_c40_tables[2, 51:60])), 6)
k13c40_tnr <- round(sum(cm_c40_tables[4, 51:60]) /
  (sum(cm_c40_tables[4, 51:60]) + sum(cm_c40_tables[3, 51:60])), 6)
k13c40_truescore <- round((2 * k13c40_tpr * k13c40_tnr) /
  (k13c40_tpr + k13c40_tnr), 6)

k15c40_tpr <- round(sum(cm_c40_tables[1, 61:70]) /
  (sum(cm_c40_tables[1, 61:70]) + sum(cm_c40_tables[2, 61:70])), 6)
k15c40_tnr <- round(sum(cm_c40_tables[4, 61:70]) /
  (sum(cm_c40_tables[4, 61:70]) + sum(cm_c40_tables[3, 61:70])), 6)
k15c40_truescore <- round((2 * k15c40_tpr * k15c40_tnr) /
  (k15c40_tpr + k15c40_tnr), 6)

k17c40_tpr <- round(sum(cm_c40_tables[1, 71:80]) /
  (sum(cm_c40_tables[1, 71:80]) + sum(cm_c40_tables[2, 71:80])), 6)
k17c40_tnr <- round(sum(cm_c40_tables[4, 71:80]) /
  (sum(cm_c40_tables[4, 71:80]) + sum(cm_c40_tables[3, 71:80])), 6)
k17c40_truescore <- round((2 * k17c40_tpr * k17c40_tnr) /
  (k17c40_tpr + k17c40_tnr), 6)

k19c40_tpr <- round(sum(cm_c40_tables[1, 81:90]) /
  (sum(cm_c40_tables[1, 81:90]) + sum(cm_c40_tables[2, 81:90])), 6)
k19c40_tnr <- round(sum(cm_c40_tables[4, 81:90]) /
  (sum(cm_c40_tables[4, 81:90]) + sum(cm_c40_tables[3, 81:90])), 6)
k19c40_truescore <- round((2 * k19c40_tpr * k19c40_tnr) /
  (k19c40_tpr + k19c40_tnr), 6)

k21c40_tpr <- round(sum(cm_c40_tables[1, 91:100]) /
  (sum(cm_c40_tables[1, 91:100]) + sum(cm_c40_tables[2, 91:100])), 6)
k21c40_tnr <- round(sum(cm_c40_tables[4, 91:100]) /
  (sum(cm_c40_tables[4, 91:100]) + sum(cm_c40_tables[3, 91:100])), 6)
k21c40_truescore <- round((2 * k21c40_tpr * k21c40_tnr) /
  (k21c40_tpr + k21c40_tnr), 6)

k23c40_tpr <- round(sum(cm_c40_tables[1, 101:110]) /
  (sum(cm_c40_tables[1, 101:110]) + sum(cm_c40_tables[2, 101:110])), 6)
k23c40_tnr <- round(sum(cm_c40_tables[4, 101:110]) /
  (sum(cm_c40_tables[4, 101:110]) + sum(cm_c40_tables[3, 101:110])), 6)
k23c40_truescore <- round((2 * k23c40_tpr * k23c40_tnr) /
  (k23c40_tpr + k23c40_tnr), 6)

k25c40_tpr <- round(sum(cm_c40_tables[1, 111:120]) /
  (sum(cm_c40_tables[1, 111:120]) + sum(cm_c40_tables[2, 111:120])), 6)
k25c40_tnr <- round(sum(cm_c40_tables[4, 111:120]) /
  (sum(cm_c40_tables[4, 111:120]) + sum(cm_c40_tables[3, 111:120])), 6)
k25c40_truescore <- round((2 * k25c40_tpr * k25c40_tnr) /
  (k25c40_tpr + k25c40_tnr), 6)

k27c40_tpr <- round(sum(cm_c40_tables[1, 121:130]) /
  (sum(cm_c40_tables[1, 121:130]) + sum(cm_c40_tables[2, 121:130])), 6)
k27c40_tnr <- round(sum(cm_c40_tables[4, 121:130]) /
  (sum(cm_c40_tables[4, 121:130]) + sum(cm_c40_tables[3, 121:130])), 6)

```

```

k27c40_truescore <- round((2 * k27c40_tpr * k27c40_tnr) /
  (k27c40_tpr + k27c40_tnr), 6)

k29c40_tpr <- round(sum(cm_c40_tables[1, 131:140]) /
  (sum(cm_c40_tables[1, 131:140]) + sum(cm_c40_tables[2, 131:140])), 6)
k29c40_tnr <- round(sum(cm_c40_tables[4, 131:140]) /
  (sum(cm_c40_tables[4, 131:140]) + sum(cm_c40_tables[3, 131:140])), 6)
k29c40_truescore <- round((2 * k29c40_tpr * k29c40_tnr) /
  (k29c40_tpr + k29c40_tnr), 6)

k31c40_tpr <- round(sum(cm_c40_tables[1, 141:150]) /
  (sum(cm_c40_tables[1, 141:150]) + sum(cm_c40_tables[2, 141:150])), 6)
k31c40_tnr <- round(sum(cm_c40_tables[4, 141:150]) /
  (sum(cm_c40_tables[4, 141:150]) + sum(cm_c40_tables[3, 141:150])), 6)
k31c40_truescore <- round((2 * k31c40_tpr * k31c40_tnr) /
  (k31c40_tpr + k31c40_tnr), 6)

k33c40_tpr <- round(sum(cm_c40_tables[1, 151:160]) /
  (sum(cm_c40_tables[1, 151:160]) + sum(cm_c40_tables[2, 151:160])), 6)
k33c40_tnr <- round(sum(cm_c40_tables[4, 151:160]) /
  (sum(cm_c40_tables[4, 151:160]) + sum(cm_c40_tables[3, 151:160])), 6)
k33c40_truescore <- round((2 * k33c40_tpr * k33c40_tnr) /
  (k33c40_tpr + k33c40_tnr), 6)

k35c40_tpr <- round(sum(cm_c40_tables[1, 161:170]) /
  (sum(cm_c40_tables[1, 161:170]) + sum(cm_c40_tables[2, 161:170])), 6)
k35c40_tnr <- round(sum(cm_c40_tables[4, 161:170]) /
  (sum(cm_c40_tables[4, 161:170]) + sum(cm_c40_tables[3, 161:170])), 6)
k35c40_truescore <- round((2 * k35c40_tpr * k35c40_tnr) /
  (k35c40_tpr + k35c40_tnr), 6)

k37c40_tpr <- round(sum(cm_c40_tables[1, 171:180]) /
  (sum(cm_c40_tables[1, 171:180]) + sum(cm_c40_tables[2, 171:180])), 6)
k37c40_tnr <- round(sum(cm_c40_tables[4, 171:180]) /
  (sum(cm_c40_tables[4, 171:180]) + sum(cm_c40_tables[3, 171:180])), 6)
k37c40_truescore <- round((2 * k37c40_tpr * k37c40_tnr) /
  (k37c40_tpr + k37c40_tnr), 6)

k39c40_tpr <- round(sum(cm_c40_tables[1, 181:190]) /
  (sum(cm_c40_tables[1, 181:190]) + sum(cm_c40_tables[2, 181:190])), 6)
k39c40_tnr <- round(sum(cm_c40_tables[4, 181:190]) /
  (sum(cm_c40_tables[4, 181:190]) + sum(cm_c40_tables[3, 181:190])), 6)
k39c40_truescore <- round((2 * k39c40_tpr * k39c40_tnr) /
  (k39c40_tpr + k39c40_tnr), 6)

# Compile the 0.40 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c40_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.40,
  TPR = c(k3c40_tpr, k5c40_tpr, k7c40_tpr, k9c40_tpr, k11c40_tpr,
    k13c40_tpr, k15c40_tpr, k17c40_tpr, k19c40_tpr, k21c40_tpr,
    k23c40_tpr, k25c40_tpr, k27c40_tpr, k29c40_tpr, k31c40_tpr,
    k33c40_tpr, k35c40_tpr, k37c40_tpr, k39c40_tpr))

```

```

        k33c40_tpr, k35c40_tpr, k37c40_tpr, k39c40_tpr),
TNR = c(k3c40_tnr, k5c40_tnr, k7c40_tnr, k9c40_tnr, k11c40_tnr,
         k13c40_tnr, k15c40_tnr, k17c40_tnr, k19c40_tnr, k21c40_tnr,
         k23c40_tnr, k25c40_tnr, k27c40_tnr, k29c40_tnr, k31c40_tnr,
         k33c40_tnr, k35c40_tnr, k37c40_tnr, k39c40_tnr),
Truescore = c(k3c40_truescore, k5c40_truescore, k7c40_truescore,
              k9c40_truescore, k11c40_truescore, k13c40_truescore,
              k15c40_truescore, k17c40_truescore, k19c40_truescore,
              k21c40_truescore, k23c40_truescore, k25c40_truescore,
              k27c40_truescore, k29c40_truescore, k31c40_truescore,
              k33c40_truescore, k35c40_truescore, k37c40_truescore,
              k39c40_truescore))

knitr::kable(c40_results[1:19, ], caption = "c40_results")

```

Table 23: c40_results

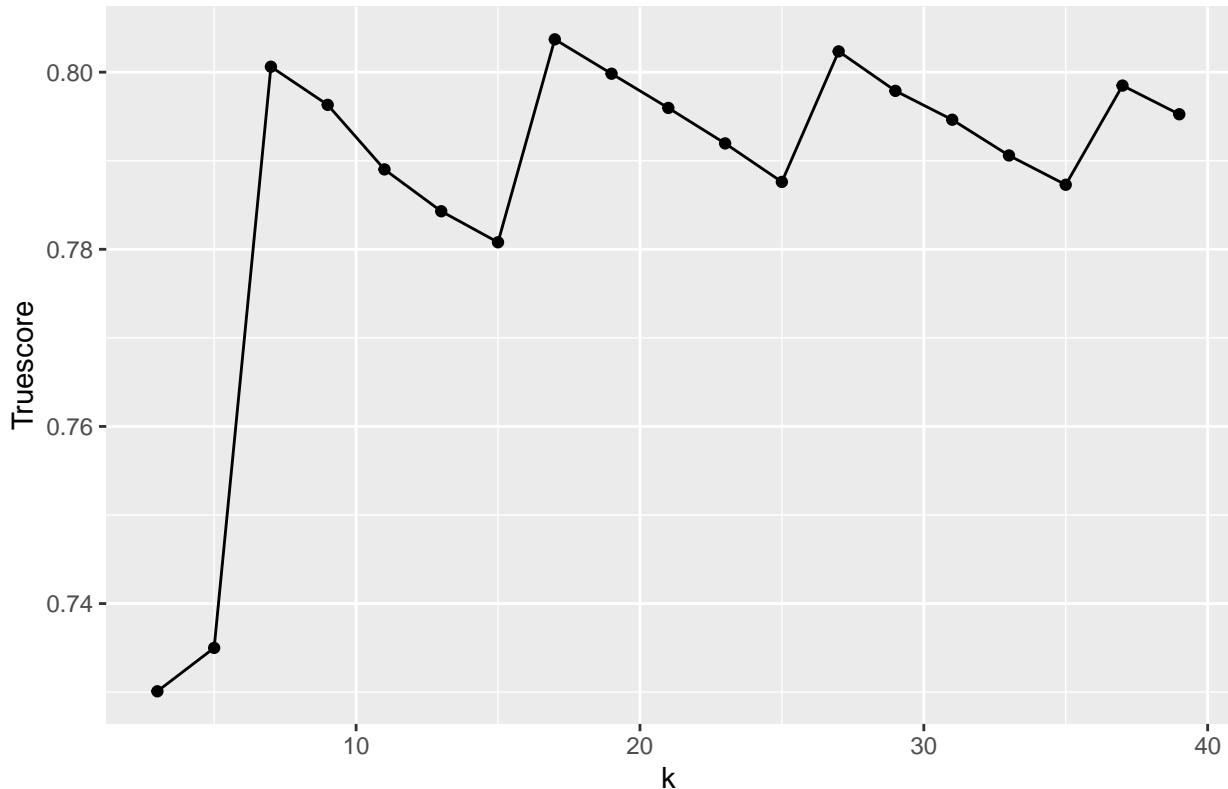
k	Cut	TPR	TNR	Truescore
3	0.4	0.616215	0.895600	0.730092
5	0.4	0.619076	0.904321	0.734993
7	0.4	0.764207	0.840667	0.800616
9	0.4	0.742169	0.858967	0.796308
11	0.4	0.721285	0.870792	0.789019
13	0.4	0.708785	0.877812	0.784295
15	0.4	0.698695	0.884749	0.780791
17	0.4	0.758283	0.854920	0.803707
19	0.4	0.745331	0.862914	0.799824
21	0.4	0.734488	0.868662	0.795960
23	0.4	0.723996	0.873997	0.791956
25	0.4	0.714458	0.877449	0.787609
27	0.4	0.754016	0.857299	0.802347
29	0.4	0.742771	0.861841	0.797888
31	0.4	0.734388	0.865639	0.794630
33	0.4	0.724849	0.869455	0.790594
35	0.4	0.717420	0.872229	0.787286
37	0.4	0.747490	0.856952	0.798487
39	0.4	0.739056	0.860684	0.795246

```

ggplot(c40_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.40")

```

Optimal k for Decision Cutoff 0.40



```
#####
# 0.41 Cutoff
#####

# For the decision cutoff of 0.41, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c41 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred41, obs))
  confusionMatrix(ss$pred41, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c41) <- fk_dfs_v

cm_c41_tables <- sapply(cm_c41, "[[", 2)
cm_c41_tables <- as_tibble(cm_c41_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.41.

k3c41_tpr <- round(sum(cm_c41_tables[1, 1:10]) /
  (sum(cm_c41_tables[1, 1:10]) + sum(cm_c41_tables[2, 1:10])), 6)
k3c41_tnr <- round(sum(cm_c41_tables[4, 1:10]) /
  (sum(cm_c41_tables[4, 1:10]) + sum(cm_c41_tables[3, 1:10])), 6)
```

```

k3c41_truescore <- round((2 * k3c41_tpr * k3c41_tnr) /
  (k3c41_tpr + k3c41_tnr), 6)

k5c41_tpr <- round(sum(cm_c41_tables[1, 11:20]) /
  (sum(cm_c41_tables[1, 11:20]) + sum(cm_c41_tables[2, 11:20])), 6)
k5c41_tnr <- round(sum(cm_c41_tables[4, 11:20]) /
  (sum(cm_c41_tables[4, 11:20]) + sum(cm_c41_tables[3, 11:20])), 6)
k5c41_truescore <- round((2 * k5c41_tpr * k5c41_tnr) /
  (k5c41_tpr + k5c41_tnr), 6)

k7c41_tpr <- round(sum(cm_c41_tables[1, 21:30]) /
  (sum(cm_c41_tables[1, 21:30]) + sum(cm_c41_tables[2, 21:30])), 6)
k7c41_tnr <- round(sum(cm_c41_tables[4, 21:30]) /
  (sum(cm_c41_tables[4, 21:30]) + sum(cm_c41_tables[3, 21:30])), 6)
k7c41_truescore <- round((2 * k7c41_tpr * k7c41_tnr) /
  (k7c41_tpr + k7c41_tnr), 6)

k9c41_tpr <- round(sum(cm_c41_tables[1, 31:40]) /
  (sum(cm_c41_tables[1, 31:40]) + sum(cm_c41_tables[2, 31:40])), 6)
k9c41_tnr <- round(sum(cm_c41_tables[4, 31:40]) /
  (sum(cm_c41_tables[4, 31:40]) + sum(cm_c41_tables[3, 31:40])), 6)
k9c41_truescore <- round((2 * k9c41_tpr * k9c41_tnr) /
  (k9c41_tpr + k9c41_tnr), 6)

k11c41_tpr <- round(sum(cm_c41_tables[1, 41:50]) /
  (sum(cm_c41_tables[1, 41:50]) + sum(cm_c41_tables[2, 41:50])), 6)
k11c41_tnr <- round(sum(cm_c41_tables[4, 41:50]) /
  (sum(cm_c41_tables[4, 41:50]) + sum(cm_c41_tables[3, 41:50])), 6)
k11c41_truescore <- round((2 * k11c41_tpr * k11c41_tnr) /
  (k11c41_tpr + k11c41_tnr), 6)

k13c41_tpr <- round(sum(cm_c41_tables[1, 51:60]) /
  (sum(cm_c41_tables[1, 51:60]) + sum(cm_c41_tables[2, 51:60])), 6)
k13c41_tnr <- round(sum(cm_c41_tables[4, 51:60]) /
  (sum(cm_c41_tables[4, 51:60]) + sum(cm_c41_tables[3, 51:60])), 6)
k13c41_truescore <- round((2 * k13c41_tpr * k13c41_tnr) /
  (k13c41_tpr + k13c41_tnr), 6)

k15c41_tpr <- round(sum(cm_c41_tables[1, 61:70]) /
  (sum(cm_c41_tables[1, 61:70]) + sum(cm_c41_tables[2, 61:70])), 6)
k15c41_tnr <- round(sum(cm_c41_tables[4, 61:70]) /
  (sum(cm_c41_tables[4, 61:70]) + sum(cm_c41_tables[3, 61:70])), 6)
k15c41_truescore <- round((2 * k15c41_tpr * k15c41_tnr) /
  (k15c41_tpr + k15c41_tnr), 6)

k17c41_tpr <- round(sum(cm_c41_tables[1, 71:80]) /
  (sum(cm_c41_tables[1, 71:80]) + sum(cm_c41_tables[2, 71:80])), 6)
k17c41_tnr <- round(sum(cm_c41_tables[4, 71:80]) /
  (sum(cm_c41_tables[4, 71:80]) + sum(cm_c41_tables[3, 71:80])), 6)
k17c41_truescore <- round((2 * k17c41_tpr * k17c41_tnr) /
  (k17c41_tpr + k17c41_tnr), 6)

k19c41_tpr <- round(sum(cm_c41_tables[1, 81:90]) /

```

```

(sum(cm_c41_tables[1, 81:90]) + sum(cm_c41_tables[2, 81:90])), 6)
k19c41_tnr <- round(sum(cm_c41_tables[4, 81:90]) /
  (sum(cm_c41_tables[4, 81:90]) + sum(cm_c41_tables[3, 81:90])), 6)
k19c41_truescore <- round((2 * k19c41_tpr * k19c41_tnr) /
  (k19c41_tpr + k19c41_tnr), 6)

k21c41_tpr <- round(sum(cm_c41_tables[1, 91:100]) /
  (sum(cm_c41_tables[1, 91:100]) + sum(cm_c41_tables[2, 91:100])), 6)
k21c41_tnr <- round(sum(cm_c41_tables[4, 91:100]) /
  (sum(cm_c41_tables[4, 91:100]) + sum(cm_c41_tables[3, 91:100])), 6)
k21c41_truescore <- round((2 * k21c41_tpr * k21c41_tnr) /
  (k21c41_tpr + k21c41_tnr), 6)

k23c41_tpr <- round(sum(cm_c41_tables[1, 101:110]) /
  (sum(cm_c41_tables[1, 101:110]) + sum(cm_c41_tables[2, 101:110])), 6)
k23c41_tnr <- round(sum(cm_c41_tables[4, 101:110]) /
  (sum(cm_c41_tables[4, 101:110]) + sum(cm_c41_tables[3, 101:110])), 6)
k23c41_truescore <- round((2 * k23c41_tpr * k23c41_tnr) /
  (k23c41_tpr + k23c41_tnr), 6)

k25c41_tpr <- round(sum(cm_c41_tables[1, 111:120]) /
  (sum(cm_c41_tables[1, 111:120]) + sum(cm_c41_tables[2, 111:120])), 6)
k25c41_tnr <- round(sum(cm_c41_tables[4, 111:120]) /
  (sum(cm_c41_tables[4, 111:120]) + sum(cm_c41_tables[3, 111:120])), 6)
k25c41_truescore <- round((2 * k25c41_tpr * k25c41_tnr) /
  (k25c41_tpr + k25c41_tnr), 6)

k27c41_tpr <- round(sum(cm_c41_tables[1, 121:130]) /
  (sum(cm_c41_tables[1, 121:130]) + sum(cm_c41_tables[2, 121:130])), 6)
k27c41_tnr <- round(sum(cm_c41_tables[4, 121:130]) /
  (sum(cm_c41_tables[4, 121:130]) + sum(cm_c41_tables[3, 121:130])), 6)
k27c41_truescore <- round((2 * k27c41_tpr * k27c41_tnr) /
  (k27c41_tpr + k27c41_tnr), 6)

k29c41_tpr <- round(sum(cm_c41_tables[1, 131:140]) /
  (sum(cm_c41_tables[1, 131:140]) + sum(cm_c41_tables[2, 131:140])), 6)
k29c41_tnr <- round(sum(cm_c41_tables[4, 131:140]) /
  (sum(cm_c41_tables[4, 131:140]) + sum(cm_c41_tables[3, 131:140])), 6)
k29c41_truescore <- round((2 * k29c41_tpr * k29c41_tnr) /
  (k29c41_tpr + k29c41_tnr), 6)

k31c41_tpr <- round(sum(cm_c41_tables[1, 141:150]) /
  (sum(cm_c41_tables[1, 141:150]) + sum(cm_c41_tables[2, 141:150])), 6)
k31c41_tnr <- round(sum(cm_c41_tables[4, 141:150]) /
  (sum(cm_c41_tables[4, 141:150]) + sum(cm_c41_tables[3, 141:150])), 6)
k31c41_truescore <- round((2 * k31c41_tpr * k31c41_tnr) /
  (k31c41_tpr + k31c41_tnr), 6)

k33c41_tpr <- round(sum(cm_c41_tables[1, 151:160]) /
  (sum(cm_c41_tables[1, 151:160]) + sum(cm_c41_tables[2, 151:160])), 6)
k33c41_tnr <- round(sum(cm_c41_tables[4, 151:160]) /
  (sum(cm_c41_tables[4, 151:160]) + sum(cm_c41_tables[3, 151:160])), 6)
k33c41_truescore <- round((2 * k33c41_tpr * k33c41_tnr) /

```

```

(k33c41_tpr + k33c41_tnr), 6)

k35c41_tpr <- round(sum(cm_c41_tables[1, 161:170]) /
  (sum(cm_c41_tables[1, 161:170]) + sum(cm_c41_tables[2, 161:170])), 6)
k35c41_tnr <- round(sum(cm_c41_tables[4, 161:170]) /
  (sum(cm_c41_tables[4, 161:170]) + sum(cm_c41_tables[3, 161:170])), 6)
k35c41_truescore <- round((2 * k35c41_tpr * k35c41_tnr) /
  (k35c41_tpr + k35c41_tnr), 6)

k37c41_tpr <- round(sum(cm_c41_tables[1, 171:180]) /
  (sum(cm_c41_tables[1, 171:180]) + sum(cm_c41_tables[2, 171:180])), 6)
k37c41_tnr <- round(sum(cm_c41_tables[4, 171:180]) /
  (sum(cm_c41_tables[4, 171:180]) + sum(cm_c41_tables[3, 171:180])), 6)
k37c41_truescore <- round((2 * k37c41_tpr * k37c41_tnr) /
  (k37c41_tpr + k37c41_tnr), 6)

k39c41_tpr <- round(sum(cm_c41_tables[1, 181:190]) /
  (sum(cm_c41_tables[1, 181:190]) + sum(cm_c41_tables[2, 181:190])), 6)
k39c41_tnr <- round(sum(cm_c41_tables[4, 181:190]) /
  (sum(cm_c41_tables[4, 181:190]) + sum(cm_c41_tables[3, 181:190])), 6)
k39c41_truescore <- round((2 * k39c41_tpr * k39c41_tnr) /
  (k39c41_tpr + k39c41_tnr), 6)

# Compile the 0.41 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c41_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.41,
  TPR = c(k3c41_tpr, k5c41_tpr, k7c41_tpr, k9c41_tpr, k11c41_tpr,
    k13c41_tpr, k15c41_tpr, k17c41_tpr, k19c41_tpr, k21c41_tpr,
    k23c41_tpr, k25c41_tpr, k27c41_tpr, k29c41_tpr, k31c41_tpr,
    k33c41_tpr, k35c41_tpr, k37c41_tpr, k39c41_tpr),
  TNR = c(k3c41_tnr, k5c41_tnr, k7c41_tnr, k9c41_tnr, k11c41_tnr,
    k13c41_tnr, k15c41_tnr, k17c41_tnr, k19c41_tnr, k21c41_tnr,
    k23c41_tnr, k25c41_tnr, k27c41_tnr, k29c41_tnr, k31c41_tnr,
    k33c41_tnr, k35c41_tnr, k37c41_tnr, k39c41_tnr),
  Truescore = c(k3c41_truescore, k5c41_truescore, k7c41_truescore,
    k9c41_truescore, k11c41_truescore, k13c41_truescore,
    k15c41_truescore, k17c41_truescore, k19c41_truescore,
    k21c41_truescore, k23c41_truescore, k25c41_truescore,
    k27c41_truescore, k29c41_truescore, k31c41_truescore,
    k33c41_truescore, k35c41_truescore, k37c41_truescore,
    k39c41_truescore))

knitr::kable(c41_results[1:19, ], caption = "c41_results")

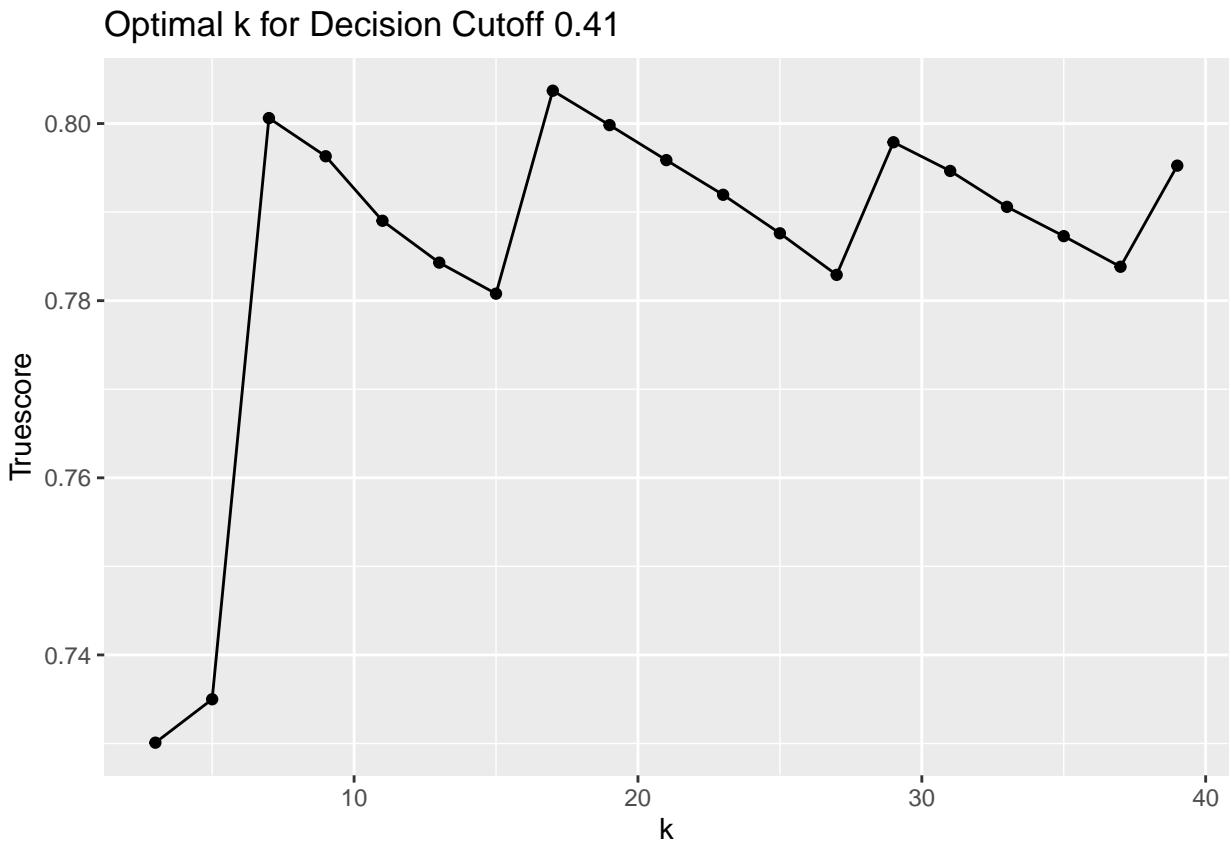
```

Table 24: c41_results

k	Cut	TPR	TNR	Truescore
3	0.41	0.616215	0.895600	0.730092
5	0.41	0.619076	0.904321	0.734993
7	0.41	0.764207	0.840667	0.800616
9	0.41	0.742169	0.858967	0.796308

k	Cut	TPR	TNR	Truescore
11	0.41	0.721285	0.870792	0.789019
13	0.41	0.708785	0.877812	0.784295
15	0.41	0.698695	0.884749	0.780791
17	0.41	0.758283	0.854920	0.803707
19	0.41	0.745331	0.862914	0.799824
21	0.41	0.734287	0.868728	0.795870
23	0.41	0.723996	0.873997	0.791956
25	0.41	0.714458	0.877449	0.787609
27	0.41	0.704669	0.880702	0.782913
29	0.41	0.742771	0.861841	0.797888
31	0.41	0.734287	0.865821	0.794648
33	0.41	0.724849	0.869455	0.790594
35	0.41	0.717420	0.872229	0.787286
37	0.41	0.709438	0.875648	0.783829
39	0.41	0.739056	0.860684	0.795246

```
ggplot(c41_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.41")
```



```
#####
# 0.42 Cutoff
#####
```

```

# For the decision cutoff of 0.42, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c42 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred42, obs))
  confusionMatrix(ss$pred42, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c42) <- fk_dfs_v

cm_c42_tables <- sapply(cm_c42, "[[", 2)
cm_c42_tables <- as_tibble(cm_c42_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.42.

k3c42_tpr <- round(sum(cm_c42_tables[1, 1:10]) /
  (sum(cm_c42_tables[1, 1:10]) + sum(cm_c42_tables[2, 1:10])), 6)
k3c42_tnr <- round(sum(cm_c42_tables[4, 1:10]) /
  (sum(cm_c42_tables[4, 1:10]) + sum(cm_c42_tables[3, 1:10])), 6)
k3c42_truescore <- round((2 * k3c42_tpr * k3c42_tnr) /
  (k3c42_tpr + k3c42_tnr), 6)

k5c42_tpr <- round(sum(cm_c42_tables[1, 11:20]) /
  (sum(cm_c42_tables[1, 11:20]) + sum(cm_c42_tables[2, 11:20])), 6)
k5c42_tnr <- round(sum(cm_c42_tables[4, 11:20]) /
  (sum(cm_c42_tables[4, 11:20]) + sum(cm_c42_tables[3, 11:20])), 6)
k5c42_truescore <- round((2 * k5c42_tpr * k5c42_tnr) /
  (k5c42_tpr + k5c42_tnr), 6)

k7c42_tpr <- round(sum(cm_c42_tables[1, 21:30]) /
  (sum(cm_c42_tables[1, 21:30]) + sum(cm_c42_tables[2, 21:30])), 6)
k7c42_tnr <- round(sum(cm_c42_tables[4, 21:30]) /
  (sum(cm_c42_tables[4, 21:30]) + sum(cm_c42_tables[3, 21:30])), 6)
k7c42_truescore <- round((2 * k7c42_tpr * k7c42_tnr) /
  (k7c42_tpr + k7c42_tnr), 6)

k9c42_tpr <- round(sum(cm_c42_tables[1, 31:40]) /
  (sum(cm_c42_tables[1, 31:40]) + sum(cm_c42_tables[2, 31:40])), 6)
k9c42_tnr <- round(sum(cm_c42_tables[4, 31:40]) /
  (sum(cm_c42_tables[4, 31:40]) + sum(cm_c42_tables[3, 31:40])), 6)
k9c42_truescore <- round((2 * k9c42_tpr * k9c42_tnr) /
  (k9c42_tpr + k9c42_tnr), 6)

k11c42_tpr <- round(sum(cm_c42_tables[1, 41:50]) /
  (sum(cm_c42_tables[1, 41:50]) + sum(cm_c42_tables[2, 41:50])), 6)
k11c42_tnr <- round(sum(cm_c42_tables[4, 41:50]) /
  (sum(cm_c42_tables[4, 41:50]) + sum(cm_c42_tables[3, 41:50])), 6)
k11c42_truescore <- round((2 * k11c42_tpr * k11c42_tnr) /
  (k11c42_tpr + k11c42_tnr), 6)

```

```

k13c42_tpr <- round(sum(cm_c42_tables[1, 51:60]) /
  (sum(cm_c42_tables[1, 51:60]) + sum(cm_c42_tables[2, 51:60])), 6)
k13c42_tnr <- round(sum(cm_c42_tables[4, 51:60]) /
  (sum(cm_c42_tables[4, 51:60]) + sum(cm_c42_tables[3, 51:60])), 6)
k13c42_truescore <- round((2 * k13c42_tpr * k13c42_tnr) /
  (k13c42_tpr + k13c42_tnr), 6)

k15c42_tpr <- round(sum(cm_c42_tables[1, 61:70]) /
  (sum(cm_c42_tables[1, 61:70]) + sum(cm_c42_tables[2, 61:70])), 6)
k15c42_tnr <- round(sum(cm_c42_tables[4, 61:70]) /
  (sum(cm_c42_tables[4, 61:70]) + sum(cm_c42_tables[3, 61:70])), 6)
k15c42_truescore <- round((2 * k15c42_tpr * k15c42_tnr) /
  (k15c42_tpr + k15c42_tnr), 6)

k17c42_tpr <- round(sum(cm_c42_tables[1, 71:80]) /
  (sum(cm_c42_tables[1, 71:80]) + sum(cm_c42_tables[2, 71:80])), 6)
k17c42_tnr <- round(sum(cm_c42_tables[4, 71:80]) /
  (sum(cm_c42_tables[4, 71:80]) + sum(cm_c42_tables[3, 71:80])), 6)
k17c42_truescore <- round((2 * k17c42_tpr * k17c42_tnr) /
  (k17c42_tpr + k17c42_tnr), 6)

k19c42_tpr <- round(sum(cm_c42_tables[1, 81:90]) /
  (sum(cm_c42_tables[1, 81:90]) + sum(cm_c42_tables[2, 81:90])), 6)
k19c42_tnr <- round(sum(cm_c42_tables[4, 81:90]) /
  (sum(cm_c42_tables[4, 81:90]) + sum(cm_c42_tables[3, 81:90])), 6)
k19c42_truescore <- round((2 * k19c42_tpr * k19c42_tnr) /
  (k19c42_tpr + k19c42_tnr), 6)

k21c42_tpr <- round(sum(cm_c42_tables[1, 91:100]) /
  (sum(cm_c42_tables[1, 91:100]) + sum(cm_c42_tables[2, 91:100])), 6)
k21c42_tnr <- round(sum(cm_c42_tables[4, 91:100]) /
  (sum(cm_c42_tables[4, 91:100]) + sum(cm_c42_tables[3, 91:100])), 6)
k21c42_truescore <- round((2 * k21c42_tpr * k21c42_tnr) /
  (k21c42_tpr + k21c42_tnr), 6)

k23c42_tpr <- round(sum(cm_c42_tables[1, 101:110]) /
  (sum(cm_c42_tables[1, 101:110]) + sum(cm_c42_tables[2, 101:110])), 6)
k23c42_tnr <- round(sum(cm_c42_tables[4, 101:110]) /
  (sum(cm_c42_tables[4, 101:110]) + sum(cm_c42_tables[3, 101:110])), 6)
k23c42_truescore <- round((2 * k23c42_tpr * k23c42_tnr) /
  (k23c42_tpr + k23c42_tnr), 6)

k25c42_tpr <- round(sum(cm_c42_tables[1, 111:120]) /
  (sum(cm_c42_tables[1, 111:120]) + sum(cm_c42_tables[2, 111:120])), 6)
k25c42_tnr <- round(sum(cm_c42_tables[4, 111:120]) /
  (sum(cm_c42_tables[4, 111:120]) + sum(cm_c42_tables[3, 111:120])), 6)
k25c42_truescore <- round((2 * k25c42_tpr * k25c42_tnr) /
  (k25c42_tpr + k25c42_tnr), 6)

k27c42_tpr <- round(sum(cm_c42_tables[1, 121:130]) /
  (sum(cm_c42_tables[1, 121:130]) + sum(cm_c42_tables[2, 121:130])), 6)
k27c42_tnr <- round(sum(cm_c42_tables[4, 121:130]) /
  (sum(cm_c42_tables[4, 121:130]) + sum(cm_c42_tables[3, 121:130])), 6)

```

```

k27c42_truescore <- round((2 * k27c42_tpr * k27c42_tnr) /
  (k27c42_tpr + k27c42_tnr), 6)

k29c42_tpr <- round(sum(cm_c42_tables[1, 131:140]) /
  (sum(cm_c42_tables[1, 131:140]) + sum(cm_c42_tables[2, 131:140])), 6)
k29c42_tnr <- round(sum(cm_c42_tables[4, 131:140]) /
  (sum(cm_c42_tables[4, 131:140]) + sum(cm_c42_tables[3, 131:140])), 6)
k29c42_truescore <- round((2 * k29c42_tpr * k29c42_tnr) /
  (k29c42_tpr + k29c42_tnr), 6)

k31c42_tpr <- round(sum(cm_c42_tables[1, 141:150]) /
  (sum(cm_c42_tables[1, 141:150]) + sum(cm_c42_tables[2, 141:150])), 6)
k31c42_tnr <- round(sum(cm_c42_tables[4, 141:150]) /
  (sum(cm_c42_tables[4, 141:150]) + sum(cm_c42_tables[3, 141:150])), 6)
k31c42_truescore <- round((2 * k31c42_tpr * k31c42_tnr) /
  (k31c42_tpr + k31c42_tnr), 6)

k33c42_tpr <- round(sum(cm_c42_tables[1, 151:160]) /
  (sum(cm_c42_tables[1, 151:160]) + sum(cm_c42_tables[2, 151:160])), 6)
k33c42_tnr <- round(sum(cm_c42_tables[4, 151:160]) /
  (sum(cm_c42_tables[4, 151:160]) + sum(cm_c42_tables[3, 151:160])), 6)
k33c42_truescore <- round((2 * k33c42_tpr * k33c42_tnr) /
  (k33c42_tpr + k33c42_tnr), 6)

k35c42_tpr <- round(sum(cm_c42_tables[1, 161:170]) /
  (sum(cm_c42_tables[1, 161:170]) + sum(cm_c42_tables[2, 161:170])), 6)
k35c42_tnr <- round(sum(cm_c42_tables[4, 161:170]) /
  (sum(cm_c42_tables[4, 161:170]) + sum(cm_c42_tables[3, 161:170])), 6)
k35c42_truescore <- round((2 * k35c42_tpr * k35c42_tnr) /
  (k35c42_tpr + k35c42_tnr), 6)

k37c42_tpr <- round(sum(cm_c42_tables[1, 171:180]) /
  (sum(cm_c42_tables[1, 171:180]) + sum(cm_c42_tables[2, 171:180])), 6)
k37c42_tnr <- round(sum(cm_c42_tables[4, 171:180]) /
  (sum(cm_c42_tables[4, 171:180]) + sum(cm_c42_tables[3, 171:180])), 6)
k37c42_truescore <- round((2 * k37c42_tpr * k37c42_tnr) /
  (k37c42_tpr + k37c42_tnr), 6)

k39c42_tpr <- round(sum(cm_c42_tables[1, 181:190]) /
  (sum(cm_c42_tables[1, 181:190]) + sum(cm_c42_tables[2, 181:190])), 6)
k39c42_tnr <- round(sum(cm_c42_tables[4, 181:190]) /
  (sum(cm_c42_tables[4, 181:190]) + sum(cm_c42_tables[3, 181:190])), 6)
k39c42_truescore <- round((2 * k39c42_tpr * k39c42_tnr) /
  (k39c42_tpr + k39c42_tnr), 6)

# Compile the 0.42 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c42_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.42,
  TPR = c(k3c42_tpr, k5c42_tpr, k7c42_tpr, k9c42_tpr, k11c42_tpr,
    k13c42_tpr, k15c42_tpr, k17c42_tpr, k19c42_tpr, k21c42_tpr,
    k23c42_tpr, k25c42_tpr, k27c42_tpr, k29c42_tpr, k31c42_tpr,
    k33c42_tpr, k35c42_tpr, k37c42_tpr, k39c42_tpr))

```

```

            k33c42_tpr, k35c42_tpr, k37c42_tpr, k39c42_tpr),
TNR = c(k3c42_tnr, k5c42_tnr, k7c42_tnr, k9c42_tnr, k11c42_tnr,
         k13c42_tnr, k15c42_tnr, k17c42_tnr, k19c42_tnr, k21c42_tnr,
         k23c42_tnr, k25c42_tnr, k27c42_tnr, k29c42_tnr, k31c42_tnr,
         k33c42_tnr, k35c42_tnr, k37c42_tnr, k39c42_tnr),
Truescore = c(k3c42_truescore, k5c42_truescore, k7c42_truescore,
              k9c42_truescore, k11c42_truescore, k13c42_truescore,
              k15c42_truescore, k17c42_truescore, k19c42_truescore,
              k21c42_truescore, k23c42_truescore, k25c42_truescore,
              k27c42_truescore, k29c42_truescore, k31c42_truescore,
              k33c42_truescore, k35c42_truescore, k37c42_truescore,
              k39c42_truescore))

knitr::kable(c42_results[1:19, ], caption = "c42_results")

```

Table 25: c42_results

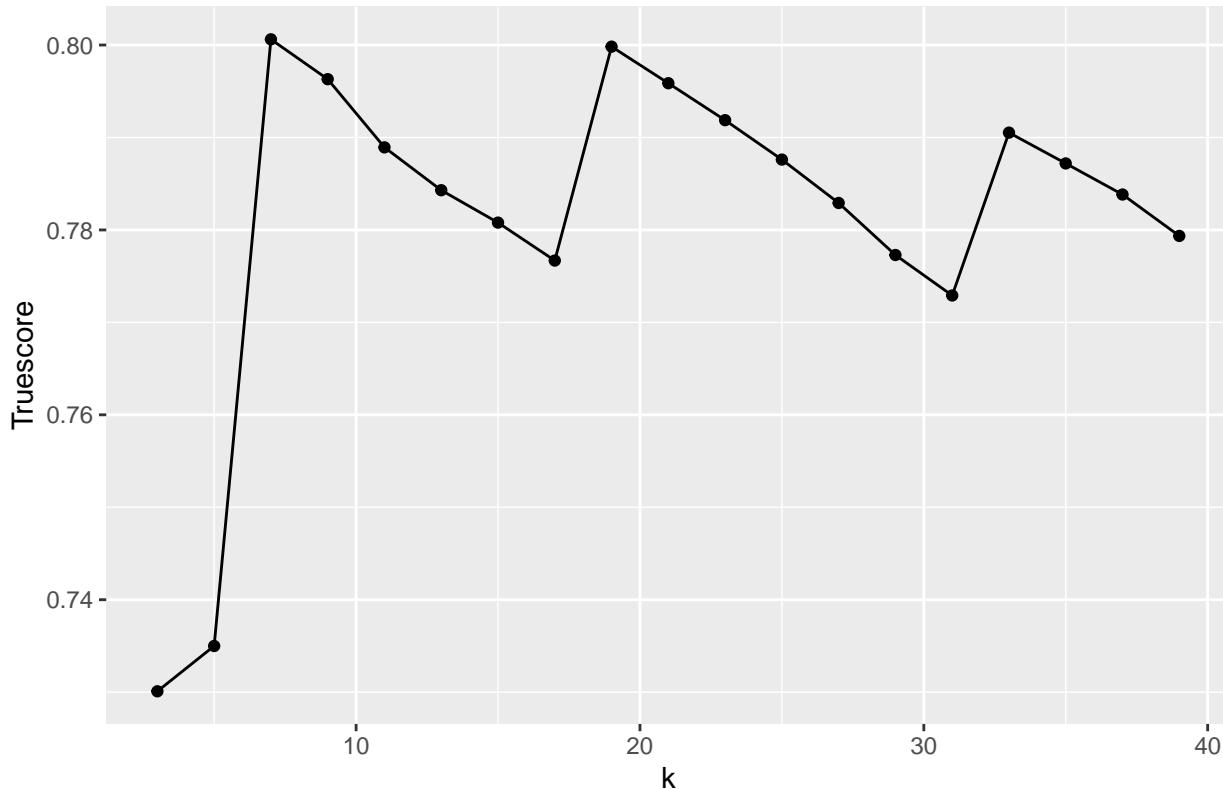
k	Cut	TPR	TNR	Truescore
3	0.42	0.616215	0.895600	0.730092
5	0.42	0.619076	0.904321	0.734993
7	0.42	0.764207	0.840667	0.800616
9	0.42	0.742169	0.858967	0.796308
11	0.42	0.721084	0.870875	0.788932
13	0.42	0.708785	0.877812	0.784295
15	0.42	0.698695	0.884749	0.780791
17	0.42	0.689157	0.889687	0.776687
19	0.42	0.745331	0.862914	0.799824
21	0.42	0.734287	0.868728	0.795870
23	0.42	0.723795	0.874079	0.791870
25	0.42	0.714458	0.877449	0.787609
27	0.42	0.704669	0.880702	0.782913
29	0.42	0.694076	0.883163	0.777285
31	0.42	0.685392	0.886054	0.772911
33	0.42	0.724649	0.869570	0.790523
35	0.42	0.717118	0.872444	0.787192
37	0.42	0.709438	0.875648	0.783829
39	0.42	0.700301	0.878522	0.779353

```

ggplot(c42_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.42")

```

Optimal k for Decision Cutoff 0.42



```
#####
# 0.43 Cutoff
#####

# For the decision cutoff of 0.43, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c43 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred43, obs))
  confusionMatrix(ss$pred43, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c43) <- fk_dfs_v

cm_c43_tables <- sapply(cm_c43, "[[", 2)
cm_c43_tables <- as_tibble(cm_c43_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.43.

k3c43_tpr <- round(sum(cm_c43_tables[1, 1:10]) /
  (sum(cm_c43_tables[1, 1:10]) + sum(cm_c43_tables[2, 1:10])), 6)
k3c43_tnr <- round(sum(cm_c43_tables[4, 1:10]) /
  (sum(cm_c43_tables[4, 1:10]) + sum(cm_c43_tables[3, 1:10])), 6)
```

```

k3c43_truescore <- round((2 * k3c43_tpr * k3c43_tnr) /
  (k3c43_tpr + k3c43_tnr), 6)

k5c43_tpr <- round(sum(cm_c43_tables[1, 11:20]) /
  (sum(cm_c43_tables[1, 11:20]) + sum(cm_c43_tables[2, 11:20])), 6)
k5c43_tnr <- round(sum(cm_c43_tables[4, 11:20]) /
  (sum(cm_c43_tables[4, 11:20]) + sum(cm_c43_tables[3, 11:20])), 6)
k5c43_truescore <- round((2 * k5c43_tpr * k5c43_tnr) /
  (k5c43_tpr + k5c43_tnr), 6)

k7c43_tpr <- round(sum(cm_c43_tables[1, 21:30]) /
  (sum(cm_c43_tables[1, 21:30]) + sum(cm_c43_tables[2, 21:30])), 6)
k7c43_tnr <- round(sum(cm_c43_tables[4, 21:30]) /
  (sum(cm_c43_tables[4, 21:30]) + sum(cm_c43_tables[3, 21:30])), 6)
k7c43_truescore <- round((2 * k7c43_tpr * k7c43_tnr) /
  (k7c43_tpr + k7c43_tnr), 6)

k9c43_tpr <- round(sum(cm_c43_tables[1, 31:40]) /
  (sum(cm_c43_tables[1, 31:40]) + sum(cm_c43_tables[2, 31:40])), 6)
k9c43_tnr <- round(sum(cm_c43_tables[4, 31:40]) /
  (sum(cm_c43_tables[4, 31:40]) + sum(cm_c43_tables[3, 31:40])), 6)
k9c43_truescore <- round((2 * k9c43_tpr * k9c43_tnr) /
  (k9c43_tpr + k9c43_tnr), 6)

k11c43_tpr <- round(sum(cm_c43_tables[1, 41:50]) /
  (sum(cm_c43_tables[1, 41:50]) + sum(cm_c43_tables[2, 41:50])), 6)
k11c43_tnr <- round(sum(cm_c43_tables[4, 41:50]) /
  (sum(cm_c43_tables[4, 41:50]) + sum(cm_c43_tables[3, 41:50])), 6)
k11c43_truescore <- round((2 * k11c43_tpr * k11c43_tnr) /
  (k11c43_tpr + k11c43_tnr), 6)

k13c43_tpr <- round(sum(cm_c43_tables[1, 51:60]) /
  (sum(cm_c43_tables[1, 51:60]) + sum(cm_c43_tables[2, 51:60])), 6)
k13c43_tnr <- round(sum(cm_c43_tables[4, 51:60]) /
  (sum(cm_c43_tables[4, 51:60]) + sum(cm_c43_tables[3, 51:60])), 6)
k13c43_truescore <- round((2 * k13c43_tpr * k13c43_tnr) /
  (k13c43_tpr + k13c43_tnr), 6)

k15c43_tpr <- round(sum(cm_c43_tables[1, 61:70]) /
  (sum(cm_c43_tables[1, 61:70]) + sum(cm_c43_tables[2, 61:70])), 6)
k15c43_tnr <- round(sum(cm_c43_tables[4, 61:70]) /
  (sum(cm_c43_tables[4, 61:70]) + sum(cm_c43_tables[3, 61:70])), 6)
k15c43_truescore <- round((2 * k15c43_tpr * k15c43_tnr) /
  (k15c43_tpr + k15c43_tnr), 6)

k17c43_tpr <- round(sum(cm_c43_tables[1, 71:80]) /
  (sum(cm_c43_tables[1, 71:80]) + sum(cm_c43_tables[2, 71:80])), 6)
k17c43_tnr <- round(sum(cm_c43_tables[4, 71:80]) /
  (sum(cm_c43_tables[4, 71:80]) + sum(cm_c43_tables[3, 71:80])), 6)
k17c43_truescore <- round((2 * k17c43_tpr * k17c43_tnr) /
  (k17c43_tpr + k17c43_tnr), 6)

k19c43_tpr <- round(sum(cm_c43_tables[1, 81:90]) /

```

```

(sum(cm_c43_tables[1, 81:90]) + sum(cm_c43_tables[2, 81:90])), 6)
k19c43_tnr <- round(sum(cm_c43_tables[4, 81:90]) /
  (sum(cm_c43_tables[4, 81:90]) + sum(cm_c43_tables[3, 81:90])), 6)
k19c43_truescore <- round((2 * k19c43_tpr * k19c43_tnr) /
  (k19c43_tpr + k19c43_tnr), 6)

k21c43_tpr <- round(sum(cm_c43_tables[1, 91:100]) /
  (sum(cm_c43_tables[1, 91:100]) + sum(cm_c43_tables[2, 91:100])), 6)
k21c43_tnr <- round(sum(cm_c43_tables[4, 91:100]) /
  (sum(cm_c43_tables[4, 91:100]) + sum(cm_c43_tables[3, 91:100])), 6)
k21c43_truescore <- round((2 * k21c43_tpr * k21c43_tnr) /
  (k21c43_tpr + k21c43_tnr), 6)

k23c43_tpr <- round(sum(cm_c43_tables[1, 101:110]) /
  (sum(cm_c43_tables[1, 101:110]) + sum(cm_c43_tables[2, 101:110])), 6)
k23c43_tnr <- round(sum(cm_c43_tables[4, 101:110]) /
  (sum(cm_c43_tables[4, 101:110]) + sum(cm_c43_tables[3, 101:110])), 6)
k23c43_truescore <- round((2 * k23c43_tpr * k23c43_tnr) /
  (k23c43_tpr + k23c43_tnr), 6)

k25c43_tpr <- round(sum(cm_c43_tables[1, 111:120]) /
  (sum(cm_c43_tables[1, 111:120]) + sum(cm_c43_tables[2, 111:120])), 6)
k25c43_tnr <- round(sum(cm_c43_tables[4, 111:120]) /
  (sum(cm_c43_tables[4, 111:120]) + sum(cm_c43_tables[3, 111:120])), 6)
k25c43_truescore <- round((2 * k25c43_tpr * k25c43_tnr) /
  (k25c43_tpr + k25c43_tnr), 6)

k27c43_tpr <- round(sum(cm_c43_tables[1, 121:130]) /
  (sum(cm_c43_tables[1, 121:130]) + sum(cm_c43_tables[2, 121:130])), 6)
k27c43_tnr <- round(sum(cm_c43_tables[4, 121:130]) /
  (sum(cm_c43_tables[4, 121:130]) + sum(cm_c43_tables[3, 121:130])), 6)
k27c43_truescore <- round((2 * k27c43_tpr * k27c43_tnr) /
  (k27c43_tpr + k27c43_tnr), 6)

k29c43_tpr <- round(sum(cm_c43_tables[1, 131:140]) /
  (sum(cm_c43_tables[1, 131:140]) + sum(cm_c43_tables[2, 131:140])), 6)
k29c43_tnr <- round(sum(cm_c43_tables[4, 131:140]) /
  (sum(cm_c43_tables[4, 131:140]) + sum(cm_c43_tables[3, 131:140])), 6)
k29c43_truescore <- round((2 * k29c43_tpr * k29c43_tnr) /
  (k29c43_tpr + k29c43_tnr), 6)

k31c43_tpr <- round(sum(cm_c43_tables[1, 141:150]) /
  (sum(cm_c43_tables[1, 141:150]) + sum(cm_c43_tables[2, 141:150])), 6)
k31c43_tnr <- round(sum(cm_c43_tables[4, 141:150]) /
  (sum(cm_c43_tables[4, 141:150]) + sum(cm_c43_tables[3, 141:150])), 6)
k31c43_truescore <- round((2 * k31c43_tpr * k31c43_tnr) /
  (k31c43_tpr + k31c43_tnr), 6)

k33c43_tpr <- round(sum(cm_c43_tables[1, 151:160]) /
  (sum(cm_c43_tables[1, 151:160]) + sum(cm_c43_tables[2, 151:160])), 6)
k33c43_tnr <- round(sum(cm_c43_tables[4, 151:160]) /
  (sum(cm_c43_tables[4, 151:160]) + sum(cm_c43_tables[3, 151:160])), 6)
k33c43_truescore <- round((2 * k33c43_tpr * k33c43_tnr) /

```

```

(k33c43_tpr + k33c43_tnr), 6)

k35c43_tpr <- round(sum(cm_c43_tables[1, 161:170]) /
  (sum(cm_c43_tables[1, 161:170]) + sum(cm_c43_tables[2, 161:170])), 6)
k35c43_tnr <- round(sum(cm_c43_tables[4, 161:170]) /
  (sum(cm_c43_tables[4, 161:170]) + sum(cm_c43_tables[3, 161:170])), 6)
k35c43_truescore <- round((2 * k35c43_tpr * k35c43_tnr) /
  (k35c43_tpr + k35c43_tnr), 6)

k37c43_tpr <- round(sum(cm_c43_tables[1, 171:180]) /
  (sum(cm_c43_tables[1, 171:180]) + sum(cm_c43_tables[2, 171:180])), 6)
k37c43_tnr <- round(sum(cm_c43_tables[4, 171:180]) /
  (sum(cm_c43_tables[4, 171:180]) + sum(cm_c43_tables[3, 171:180])), 6)
k37c43_truescore <- round((2 * k37c43_tpr * k37c43_tnr) /
  (k37c43_tpr + k37c43_tnr), 6)

k39c43_tpr <- round(sum(cm_c43_tables[1, 181:190]) /
  (sum(cm_c43_tables[1, 181:190]) + sum(cm_c43_tables[2, 181:190])), 6)
k39c43_tnr <- round(sum(cm_c43_tables[4, 181:190]) /
  (sum(cm_c43_tables[4, 181:190]) + sum(cm_c43_tables[3, 181:190])), 6)
k39c43_truescore <- round((2 * k39c43_tpr * k39c43_tnr) /
  (k39c43_tpr + k39c43_tnr), 6)

# Compile the 0.43 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c43_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.43,
  TPR = c(k3c43_tpr, k5c43_tpr, k7c43_tpr, k9c43_tpr, k11c43_tpr,
    k13c43_tpr, k15c43_tpr, k17c43_tpr, k19c43_tpr, k21c43_tpr,
    k23c43_tpr, k25c43_tpr, k27c43_tpr, k29c43_tpr, k31c43_tpr,
    k33c43_tpr, k35c43_tpr, k37c43_tpr, k39c43_tpr),
  TNR = c(k3c43_tnr, k5c43_tnr, k7c43_tnr, k9c43_tnr, k11c43_tnr,
    k13c43_tnr, k15c43_tnr, k17c43_tnr, k19c43_tnr, k21c43_tnr,
    k23c43_tnr, k25c43_tnr, k27c43_tnr, k29c43_tnr, k31c43_tnr,
    k33c43_tnr, k35c43_tnr, k37c43_tnr, k39c43_tnr),
  Truescore = c(k3c43_truescore, k5c43_truescore, k7c43_truescore,
    k9c43_truescore, k11c43_truescore, k13c43_truescore,
    k15c43_truescore, k17c43_truescore, k19c43_truescore,
    k21c43_truescore, k23c43_truescore, k25c43_truescore,
    k27c43_truescore, k29c43_truescore, k31c43_truescore,
    k33c43_truescore, k35c43_truescore, k37c43_truescore,
    k39c43_truescore))

knitr::kable(c43_results[1:19, ], caption = "c43_results")

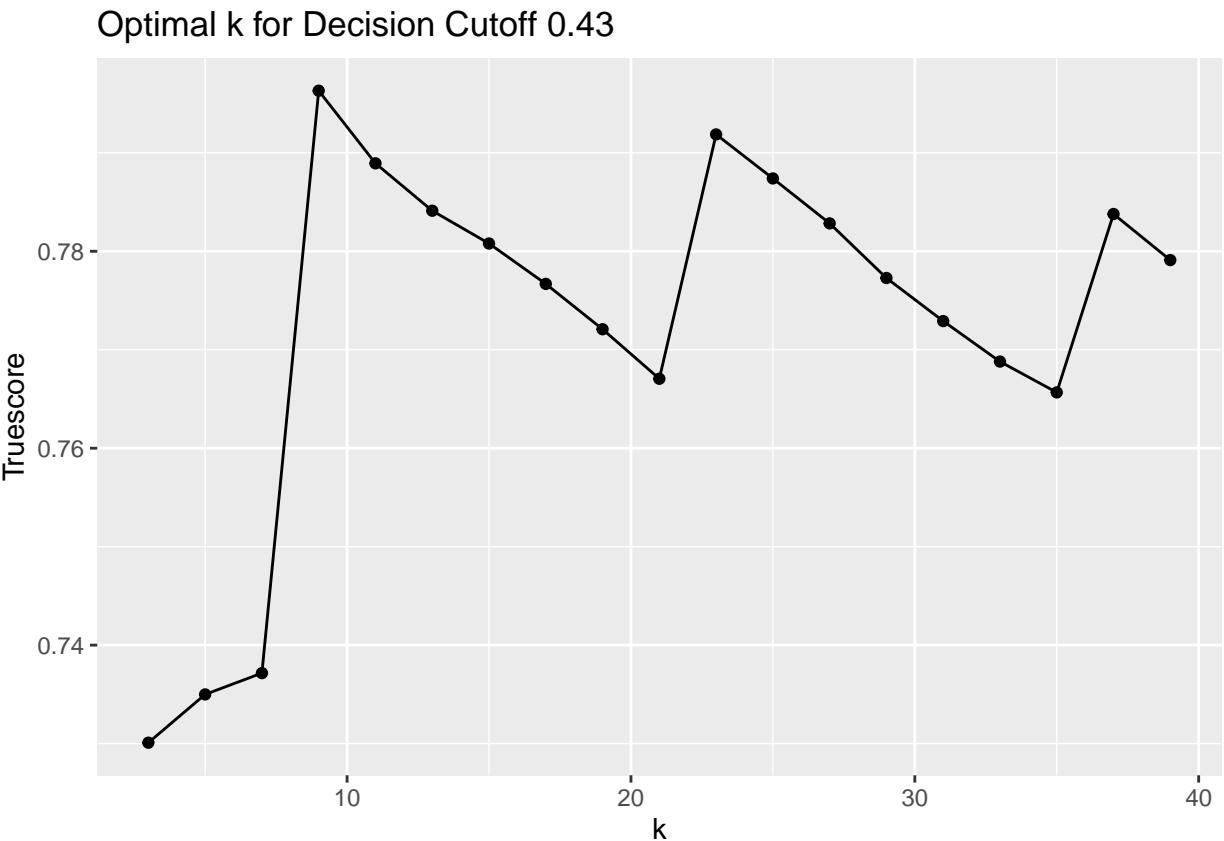
```

Table 26: c43_results

k	Cut	TPR	TNR	Truescore
3	0.43	0.616215	0.895600	0.730092
5	0.43	0.619076	0.904321	0.734993
7	0.43	0.620131	0.908631	0.737159
9	0.43	0.742169	0.858967	0.796308

k	Cut	TPR	TNR	Truescore
11	0.43	0.721084	0.870875	0.788932
13	0.43	0.708434	0.877911	0.784119
15	0.43	0.698695	0.884749	0.780791
17	0.43	0.689157	0.889687	0.776687
19	0.43	0.679869	0.893238	0.772083
21	0.43	0.670331	0.896393	0.767053
23	0.43	0.723795	0.874079	0.791870
25	0.43	0.714056	0.877531	0.787398
27	0.43	0.704468	0.880818	0.782834
29	0.43	0.694076	0.883163	0.777285
31	0.43	0.685392	0.886054	0.772911
33	0.43	0.677460	0.888597	0.768796
35	0.43	0.671185	0.891108	0.765667
37	0.43	0.709237	0.875846	0.783785
39	0.43	0.699799	0.878687	0.779106

```
ggplot(c43_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.43")
```



```
#####
# 0.44 Cutoff
#####
```

```

# For the decision cutoff of 0.44, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c44 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred44, obs))
  confusionMatrix(ss$pred44, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c44) <- fk_dfs_v

cm_c44_tables <- sapply(cm_c44, "[[", 2)
cm_c44_tables <- as_tibble(cm_c44_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.44.

k3c44_tpr <- round(sum(cm_c44_tables[1, 1:10]) /
  (sum(cm_c44_tables[1, 1:10]) + sum(cm_c44_tables[2, 1:10])), 6)
k3c44_tnr <- round(sum(cm_c44_tables[4, 1:10]) /
  (sum(cm_c44_tables[4, 1:10]) + sum(cm_c44_tables[3, 1:10])), 6)
k3c44_truescore <- round((2 * k3c44_tpr * k3c44_tnr) /
  (k3c44_tpr + k3c44_tnr), 6)

k5c44_tpr <- round(sum(cm_c44_tables[1, 11:20]) /
  (sum(cm_c44_tables[1, 11:20]) + sum(cm_c44_tables[2, 11:20])), 6)
k5c44_tnr <- round(sum(cm_c44_tables[4, 11:20]) /
  (sum(cm_c44_tables[4, 11:20]) + sum(cm_c44_tables[3, 11:20])), 6)
k5c44_truescore <- round((2 * k5c44_tpr * k5c44_tnr) /
  (k5c44_tpr + k5c44_tnr), 6)

k7c44_tpr <- round(sum(cm_c44_tables[1, 21:30]) /
  (sum(cm_c44_tables[1, 21:30]) + sum(cm_c44_tables[2, 21:30])), 6)
k7c44_tnr <- round(sum(cm_c44_tables[4, 21:30]) /
  (sum(cm_c44_tables[4, 21:30]) + sum(cm_c44_tables[3, 21:30])), 6)
k7c44_truescore <- round((2 * k7c44_tpr * k7c44_tnr) /
  (k7c44_tpr + k7c44_tnr), 6)

k9c44_tpr <- round(sum(cm_c44_tables[1, 31:40]) /
  (sum(cm_c44_tables[1, 31:40]) + sum(cm_c44_tables[2, 31:40])), 6)
k9c44_tnr <- round(sum(cm_c44_tables[4, 31:40]) /
  (sum(cm_c44_tables[4, 31:40]) + sum(cm_c44_tables[3, 31:40])), 6)
k9c44_truescore <- round((2 * k9c44_tpr * k9c44_tnr) /
  (k9c44_tpr + k9c44_tnr), 6)

k11c44_tpr <- round(sum(cm_c44_tables[1, 41:50]) /
  (sum(cm_c44_tables[1, 41:50]) + sum(cm_c44_tables[2, 41:50])), 6)
k11c44_tnr <- round(sum(cm_c44_tables[4, 41:50]) /
  (sum(cm_c44_tables[4, 41:50]) + sum(cm_c44_tables[3, 41:50])), 6)
k11c44_truescore <- round((2 * k11c44_tpr * k11c44_tnr) /
  (k11c44_tpr + k11c44_tnr), 6)

```

```

k13c44_tpr <- round(sum(cm_c44_tables[1, 51:60]) /
  (sum(cm_c44_tables[1, 51:60]) + sum(cm_c44_tables[2, 51:60])), 6)
k13c44_tnr <- round(sum(cm_c44_tables[4, 51:60]) /
  (sum(cm_c44_tables[4, 51:60]) + sum(cm_c44_tables[3, 51:60])), 6)
k13c44_truescore <- round((2 * k13c44_tpr * k13c44_tnr) /
  (k13c44_tpr + k13c44_tnr), 6)

k15c44_tpr <- round(sum(cm_c44_tables[1, 61:70]) /
  (sum(cm_c44_tables[1, 61:70]) + sum(cm_c44_tables[2, 61:70])), 6)
k15c44_tnr <- round(sum(cm_c44_tables[4, 61:70]) /
  (sum(cm_c44_tables[4, 61:70]) + sum(cm_c44_tables[3, 61:70])), 6)
k15c44_truescore <- round((2 * k15c44_tpr * k15c44_tnr) /
  (k15c44_tpr + k15c44_tnr), 6)

k17c44_tpr <- round(sum(cm_c44_tables[1, 71:80]) /
  (sum(cm_c44_tables[1, 71:80]) + sum(cm_c44_tables[2, 71:80])), 6)
k17c44_tnr <- round(sum(cm_c44_tables[4, 71:80]) /
  (sum(cm_c44_tables[4, 71:80]) + sum(cm_c44_tables[3, 71:80])), 6)
k17c44_truescore <- round((2 * k17c44_tpr * k17c44_tnr) /
  (k17c44_tpr + k17c44_tnr), 6)

k19c44_tpr <- round(sum(cm_c44_tables[1, 81:90]) /
  (sum(cm_c44_tables[1, 81:90]) + sum(cm_c44_tables[2, 81:90])), 6)
k19c44_tnr <- round(sum(cm_c44_tables[4, 81:90]) /
  (sum(cm_c44_tables[4, 81:90]) + sum(cm_c44_tables[3, 81:90])), 6)
k19c44_truescore <- round((2 * k19c44_tpr * k19c44_tnr) /
  (k19c44_tpr + k19c44_tnr), 6)

k21c44_tpr <- round(sum(cm_c44_tables[1, 91:100]) /
  (sum(cm_c44_tables[1, 91:100]) + sum(cm_c44_tables[2, 91:100])), 6)
k21c44_tnr <- round(sum(cm_c44_tables[4, 91:100]) /
  (sum(cm_c44_tables[4, 91:100]) + sum(cm_c44_tables[3, 91:100])), 6)
k21c44_truescore <- round((2 * k21c44_tpr * k21c44_tnr) /
  (k21c44_tpr + k21c44_tnr), 6)

k23c44_tpr <- round(sum(cm_c44_tables[1, 101:110]) /
  (sum(cm_c44_tables[1, 101:110]) + sum(cm_c44_tables[2, 101:110])), 6)
k23c44_tnr <- round(sum(cm_c44_tables[4, 101:110]) /
  (sum(cm_c44_tables[4, 101:110]) + sum(cm_c44_tables[3, 101:110])), 6)
k23c44_truescore <- round((2 * k23c44_tpr * k23c44_tnr) /
  (k23c44_tpr + k23c44_tnr), 6)

k25c44_tpr <- round(sum(cm_c44_tables[1, 111:120]) /
  (sum(cm_c44_tables[1, 111:120]) + sum(cm_c44_tables[2, 111:120])), 6)
k25c44_tnr <- round(sum(cm_c44_tables[4, 111:120]) /
  (sum(cm_c44_tables[4, 111:120]) + sum(cm_c44_tables[3, 111:120])), 6)
k25c44_truescore <- round((2 * k25c44_tpr * k25c44_tnr) /
  (k25c44_tpr + k25c44_tnr), 6)

k27c44_tpr <- round(sum(cm_c44_tables[1, 121:130]) /
  (sum(cm_c44_tables[1, 121:130]) + sum(cm_c44_tables[2, 121:130])), 6)
k27c44_tnr <- round(sum(cm_c44_tables[4, 121:130]) /
  (sum(cm_c44_tables[4, 121:130]) + sum(cm_c44_tables[3, 121:130])), 6)

```

```

k27c44_truescore <- round((2 * k27c44_tpr * k27c44_tnr) /
  (k27c44_tpr + k27c44_tnr), 6)

k29c44_tpr <- round(sum(cm_c44_tables[1, 131:140]) /
  (sum(cm_c44_tables[1, 131:140]) + sum(cm_c44_tables[2, 131:140])), 6)
k29c44_tnr <- round(sum(cm_c44_tables[4, 131:140]) /
  (sum(cm_c44_tables[4, 131:140]) + sum(cm_c44_tables[3, 131:140])), 6)
k29c44_truescore <- round((2 * k29c44_tpr * k29c44_tnr) /
  (k29c44_tpr + k29c44_tnr), 6)

k31c44_tpr <- round(sum(cm_c44_tables[1, 141:150]) /
  (sum(cm_c44_tables[1, 141:150]) + sum(cm_c44_tables[2, 141:150])), 6)
k31c44_tnr <- round(sum(cm_c44_tables[4, 141:150]) /
  (sum(cm_c44_tables[4, 141:150]) + sum(cm_c44_tables[3, 141:150])), 6)
k31c44_truescore <- round((2 * k31c44_tpr * k31c44_tnr) /
  (k31c44_tpr + k31c44_tnr), 6)

k33c44_tpr <- round(sum(cm_c44_tables[1, 151:160]) /
  (sum(cm_c44_tables[1, 151:160]) + sum(cm_c44_tables[2, 151:160])), 6)
k33c44_tnr <- round(sum(cm_c44_tables[4, 151:160]) /
  (sum(cm_c44_tables[4, 151:160]) + sum(cm_c44_tables[3, 151:160])), 6)
k33c44_truescore <- round((2 * k33c44_tpr * k33c44_tnr) /
  (k33c44_tpr + k33c44_tnr), 6)

k35c44_tpr <- round(sum(cm_c44_tables[1, 161:170]) /
  (sum(cm_c44_tables[1, 161:170]) + sum(cm_c44_tables[2, 161:170])), 6)
k35c44_tnr <- round(sum(cm_c44_tables[4, 161:170]) /
  (sum(cm_c44_tables[4, 161:170]) + sum(cm_c44_tables[3, 161:170])), 6)
k35c44_truescore <- round((2 * k35c44_tpr * k35c44_tnr) /
  (k35c44_tpr + k35c44_tnr), 6)

k37c44_tpr <- round(sum(cm_c44_tables[1, 171:180]) /
  (sum(cm_c44_tables[1, 171:180]) + sum(cm_c44_tables[2, 171:180])), 6)
k37c44_tnr <- round(sum(cm_c44_tables[4, 171:180]) /
  (sum(cm_c44_tables[4, 171:180]) + sum(cm_c44_tables[3, 171:180])), 6)
k37c44_truescore <- round((2 * k37c44_tpr * k37c44_tnr) /
  (k37c44_tpr + k37c44_tnr), 6)

k39c44_tpr <- round(sum(cm_c44_tables[1, 181:190]) /
  (sum(cm_c44_tables[1, 181:190]) + sum(cm_c44_tables[2, 181:190])), 6)
k39c44_tnr <- round(sum(cm_c44_tables[4, 181:190]) /
  (sum(cm_c44_tables[4, 181:190]) + sum(cm_c44_tables[3, 181:190])), 6)
k39c44_truescore <- round((2 * k39c44_tpr * k39c44_tnr) /
  (k39c44_tpr + k39c44_tnr), 6)

# Compile the 0.44 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c44_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.44,
  TPR = c(k3c44_tpr, k5c44_tpr, k7c44_tpr, k9c44_tpr, k11c44_tpr,
    k13c44_tpr, k15c44_tpr, k17c44_tpr, k19c44_tpr, k21c44_tpr,
    k23c44_tpr, k25c44_tpr, k27c44_tpr, k29c44_tpr, k31c44_tpr,
    k33c44_tpr, k35c44_tpr, k37c44_tpr, k39c44_tpr))

```

```

            k33c44_tpr, k35c44_tpr, k37c44_tpr, k39c44_tpr),
TNR = c(k3c44_tnr, k5c44_tnr, k7c44_tnr, k9c44_tnr, k11c44_tnr,
         k13c44_tnr, k15c44_tnr, k17c44_tnr, k19c44_tnr, k21c44_tnr,
         k23c44_tnr, k25c44_tnr, k27c44_tnr, k29c44_tnr, k31c44_tnr,
         k33c44_tnr, k35c44_tnr, k37c44_tnr, k39c44_tnr),
Truescore = c(k3c44_truescore, k5c44_truescore, k7c44_truescore,
              k9c44_truescore, k11c44_truescore, k13c44_truescore,
              k15c44_truescore, k17c44_truescore, k19c44_truescore,
              k21c44_truescore, k23c44_truescore, k25c44_truescore,
              k27c44_truescore, k29c44_truescore, k31c44_truescore,
              k33c44_truescore, k35c44_truescore, k37c44_truescore,
              k39c44_truescore))

knitr::kable(c44_results[1:19, ], caption = "c44_results")

```

Table 27: c44_results

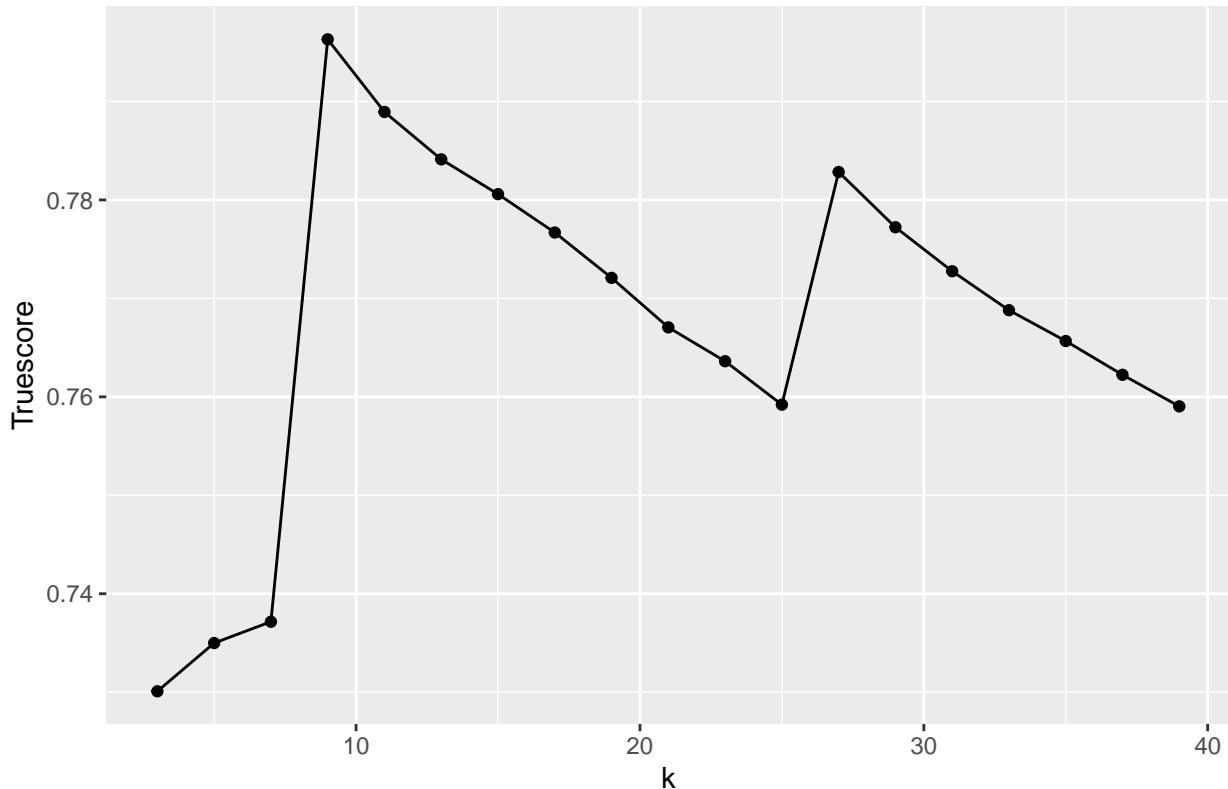
k	Cut	TPR	TNR	Truescore
3	0.44	0.616215	0.895600	0.730092
5	0.44	0.619076	0.904321	0.734993
7	0.44	0.620131	0.908631	0.737159
9	0.44	0.742169	0.858967	0.796308
11	0.44	0.721084	0.870875	0.788932
13	0.44	0.708434	0.877911	0.784119
15	0.44	0.698293	0.884864	0.780585
17	0.44	0.689157	0.889687	0.776687
19	0.44	0.679869	0.893238	0.772083
21	0.44	0.670331	0.896393	0.767053
23	0.44	0.663805	0.898755	0.763616
25	0.44	0.656074	0.900803	0.759204
27	0.44	0.704468	0.880818	0.782834
29	0.44	0.693926	0.883246	0.777223
31	0.44	0.685040	0.886235	0.772756
33	0.44	0.677460	0.888597	0.768796
35	0.44	0.671185	0.891108	0.765667
37	0.44	0.664659	0.893387	0.762234
39	0.44	0.658886	0.895072	0.759030

```

ggplot(c44_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.44")

```

Optimal k for Decision Cutoff 0.44



```
#####
# 0.45 Cutoff
#####

# For the decision cutoff of 0.45, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c45 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred45, obs))
  confusionMatrix(ss$pred45, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c45) <- fk_dfs_v

cm_c45_tables <- sapply(cm_c45, "[[", 2)
cm_c45_tables <- as_tibble(cm_c45_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.45.

k3c45_tpr <- round(sum(cm_c45_tables[1, 1:10]) /
  (sum(cm_c45_tables[1, 1:10]) + sum(cm_c45_tables[2, 1:10])), 6)
k3c45_tnr <- round(sum(cm_c45_tables[4, 1:10]) /
  (sum(cm_c45_tables[4, 1:10]) + sum(cm_c45_tables[3, 1:10])), 6)
```

```

k3c45_truescore <- round((2 * k3c45_tpr * k3c45_tnr) /
  (k3c45_tpr + k3c45_tnr), 6)

k5c45_tpr <- round(sum(cm_c45_tables[1, 11:20]) /
  (sum(cm_c45_tables[1, 11:20]) + sum(cm_c45_tables[2, 11:20])), 6)
k5c45_tnr <- round(sum(cm_c45_tables[4, 11:20]) /
  (sum(cm_c45_tables[4, 11:20]) + sum(cm_c45_tables[3, 11:20])), 6)
k5c45_truescore <- round((2 * k5c45_tpr * k5c45_tnr) /
  (k5c45_tpr + k5c45_tnr), 6)

k7c45_tpr <- round(sum(cm_c45_tables[1, 21:30]) /
  (sum(cm_c45_tables[1, 21:30]) + sum(cm_c45_tables[2, 21:30])), 6)
k7c45_tnr <- round(sum(cm_c45_tables[4, 21:30]) /
  (sum(cm_c45_tables[4, 21:30]) + sum(cm_c45_tables[3, 21:30])), 6)
k7c45_truescore <- round((2 * k7c45_tpr * k7c45_tnr) /
  (k7c45_tpr + k7c45_tnr), 6)

k9c45_tpr <- round(sum(cm_c45_tables[1, 31:40]) /
  (sum(cm_c45_tables[1, 31:40]) + sum(cm_c45_tables[2, 31:40])), 6)
k9c45_tnr <- round(sum(cm_c45_tables[4, 31:40]) /
  (sum(cm_c45_tables[4, 31:40]) + sum(cm_c45_tables[3, 31:40])), 6)
k9c45_truescore <- round((2 * k9c45_tpr * k9c45_tnr) /
  (k9c45_tpr + k9c45_tnr), 6)

k11c45_tpr <- round(sum(cm_c45_tables[1, 41:50]) /
  (sum(cm_c45_tables[1, 41:50]) + sum(cm_c45_tables[2, 41:50])), 6)
k11c45_tnr <- round(sum(cm_c45_tables[4, 41:50]) /
  (sum(cm_c45_tables[4, 41:50]) + sum(cm_c45_tables[3, 41:50])), 6)
k11c45_truescore <- round((2 * k11c45_tpr * k11c45_tnr) /
  (k11c45_tpr + k11c45_tnr), 6)

k13c45_tpr <- round(sum(cm_c45_tables[1, 51:60]) /
  (sum(cm_c45_tables[1, 51:60]) + sum(cm_c45_tables[2, 51:60])), 6)
k13c45_tnr <- round(sum(cm_c45_tables[4, 51:60]) /
  (sum(cm_c45_tables[4, 51:60]) + sum(cm_c45_tables[3, 51:60])), 6)
k13c45_truescore <- round((2 * k13c45_tpr * k13c45_tnr) /
  (k13c45_tpr + k13c45_tnr), 6)

k15c45_tpr <- round(sum(cm_c45_tables[1, 61:70]) /
  (sum(cm_c45_tables[1, 61:70]) + sum(cm_c45_tables[2, 61:70])), 6)
k15c45_tnr <- round(sum(cm_c45_tables[4, 61:70]) /
  (sum(cm_c45_tables[4, 61:70]) + sum(cm_c45_tables[3, 61:70])), 6)
k15c45_truescore <- round((2 * k15c45_tpr * k15c45_tnr) /
  (k15c45_tpr + k15c45_tnr), 6)

k17c45_tpr <- round(sum(cm_c45_tables[1, 71:80]) /
  (sum(cm_c45_tables[1, 71:80]) + sum(cm_c45_tables[2, 71:80])), 6)
k17c45_tnr <- round(sum(cm_c45_tables[4, 71:80]) /
  (sum(cm_c45_tables[4, 71:80]) + sum(cm_c45_tables[3, 71:80])), 6)
k17c45_truescore <- round((2 * k17c45_tpr * k17c45_tnr) /
  (k17c45_tpr + k17c45_tnr), 6)

k19c45_tpr <- round(sum(cm_c45_tables[1, 81:90]) /

```

```

(sum(cm_c45_tables[1, 81:90]) + sum(cm_c45_tables[2, 81:90])), 6)
k19c45_tnr <- round(sum(cm_c45_tables[4, 81:90]) /
  (sum(cm_c45_tables[4, 81:90]) + sum(cm_c45_tables[3, 81:90])), 6)
k19c45_truescore <- round((2 * k19c45_tpr * k19c45_tnr) /
  (k19c45_tpr + k19c45_tnr), 6)

k21c45_tpr <- round(sum(cm_c45_tables[1, 91:100]) /
  (sum(cm_c45_tables[1, 91:100]) + sum(cm_c45_tables[2, 91:100])), 6)
k21c45_tnr <- round(sum(cm_c45_tables[4, 91:100]) /
  (sum(cm_c45_tables[4, 91:100]) + sum(cm_c45_tables[3, 91:100])), 6)
k21c45_truescore <- round((2 * k21c45_tpr * k21c45_tnr) /
  (k21c45_tpr + k21c45_tnr), 6)

k23c45_tpr <- round(sum(cm_c45_tables[1, 101:110]) /
  (sum(cm_c45_tables[1, 101:110]) + sum(cm_c45_tables[2, 101:110])), 6)
k23c45_tnr <- round(sum(cm_c45_tables[4, 101:110]) /
  (sum(cm_c45_tables[4, 101:110]) + sum(cm_c45_tables[3, 101:110])), 6)
k23c45_truescore <- round((2 * k23c45_tpr * k23c45_tnr) /
  (k23c45_tpr + k23c45_tnr), 6)

k25c45_tpr <- round(sum(cm_c45_tables[1, 111:120]) /
  (sum(cm_c45_tables[1, 111:120]) + sum(cm_c45_tables[2, 111:120])), 6)
k25c45_tnr <- round(sum(cm_c45_tables[4, 111:120]) /
  (sum(cm_c45_tables[4, 111:120]) + sum(cm_c45_tables[3, 111:120])), 6)
k25c45_truescore <- round((2 * k25c45_tpr * k25c45_tnr) /
  (k25c45_tpr + k25c45_tnr), 6)

k27c45_tpr <- round(sum(cm_c45_tables[1, 121:130]) /
  (sum(cm_c45_tables[1, 121:130]) + sum(cm_c45_tables[2, 121:130])), 6)
k27c45_tnr <- round(sum(cm_c45_tables[4, 121:130]) /
  (sum(cm_c45_tables[4, 121:130]) + sum(cm_c45_tables[3, 121:130])), 6)
k27c45_truescore <- round((2 * k27c45_tpr * k27c45_tnr) /
  (k27c45_tpr + k27c45_tnr), 6)

k29c45_tpr <- round(sum(cm_c45_tables[1, 131:140]) /
  (sum(cm_c45_tables[1, 131:140]) + sum(cm_c45_tables[2, 131:140])), 6)
k29c45_tnr <- round(sum(cm_c45_tables[4, 131:140]) /
  (sum(cm_c45_tables[4, 131:140]) + sum(cm_c45_tables[3, 131:140])), 6)
k29c45_truescore <- round((2 * k29c45_tpr * k29c45_tnr) /
  (k29c45_tpr + k29c45_tnr), 6)

k31c45_tpr <- round(sum(cm_c45_tables[1, 141:150]) /
  (sum(cm_c45_tables[1, 141:150]) + sum(cm_c45_tables[2, 141:150])), 6)
k31c45_tnr <- round(sum(cm_c45_tables[4, 141:150]) /
  (sum(cm_c45_tables[4, 141:150]) + sum(cm_c45_tables[3, 141:150])), 6)
k31c45_truescore <- round((2 * k31c45_tpr * k31c45_tnr) /
  (k31c45_tpr + k31c45_tnr), 6)

k33c45_tpr <- round(sum(cm_c45_tables[1, 151:160]) /
  (sum(cm_c45_tables[1, 151:160]) + sum(cm_c45_tables[2, 151:160])), 6)
k33c45_tnr <- round(sum(cm_c45_tables[4, 151:160]) /
  (sum(cm_c45_tables[4, 151:160]) + sum(cm_c45_tables[3, 151:160])), 6)
k33c45_truescore <- round((2 * k33c45_tpr * k33c45_tnr) /

```

```

(k33c45_tpr + k33c45_tnr), 6)

k35c45_tpr <- round(sum(cm_c45_tables[1, 161:170]) /
  (sum(cm_c45_tables[1, 161:170]) + sum(cm_c45_tables[2, 161:170])), 6)
k35c45_tnr <- round(sum(cm_c45_tables[4, 161:170]) /
  (sum(cm_c45_tables[4, 161:170]) + sum(cm_c45_tables[3, 161:170])), 6)
k35c45_truescore <- round((2 * k35c45_tpr * k35c45_tnr) /
  (k35c45_tpr + k35c45_tnr), 6)

k37c45_tpr <- round(sum(cm_c45_tables[1, 171:180]) /
  (sum(cm_c45_tables[1, 171:180]) + sum(cm_c45_tables[2, 171:180])), 6)
k37c45_tnr <- round(sum(cm_c45_tables[4, 171:180]) /
  (sum(cm_c45_tables[4, 171:180]) + sum(cm_c45_tables[3, 171:180])), 6)
k37c45_truescore <- round((2 * k37c45_tpr * k37c45_tnr) /
  (k37c45_tpr + k37c45_tnr), 6)

k39c45_tpr <- round(sum(cm_c45_tables[1, 181:190]) /
  (sum(cm_c45_tables[1, 181:190]) + sum(cm_c45_tables[2, 181:190])), 6)
k39c45_tnr <- round(sum(cm_c45_tables[4, 181:190]) /
  (sum(cm_c45_tables[4, 181:190]) + sum(cm_c45_tables[3, 181:190])), 6)
k39c45_truescore <- round((2 * k39c45_tpr * k39c45_tnr) /
  (k39c45_tpr + k39c45_tnr), 6)

# Compile the 0.45 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c45_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.45,
  TPR = c(k3c45_tpr, k5c45_tpr, k7c45_tpr, k9c45_tpr, k11c45_tpr,
    k13c45_tpr, k15c45_tpr, k17c45_tpr, k19c45_tpr, k21c45_tpr,
    k23c45_tpr, k25c45_tpr, k27c45_tpr, k29c45_tpr, k31c45_tpr,
    k33c45_tpr, k35c45_tpr, k37c45_tpr, k39c45_tpr),
  TNR = c(k3c45_tnr, k5c45_tnr, k7c45_tnr, k9c45_tnr, k11c45_tnr,
    k13c45_tnr, k15c45_tnr, k17c45_tnr, k19c45_tnr, k21c45_tnr,
    k23c45_tnr, k25c45_tnr, k27c45_tnr, k29c45_tnr, k31c45_tnr,
    k33c45_tnr, k35c45_tnr, k37c45_tnr, k39c45_tnr),
  Truescore = c(k3c45_truescore, k5c45_truescore, k7c45_truescore,
    k9c45_truescore, k11c45_truescore, k13c45_truescore,
    k15c45_truescore, k17c45_truescore, k19c45_truescore,
    k21c45_truescore, k23c45_truescore, k25c45_truescore,
    k27c45_truescore, k29c45_truescore, k31c45_truescore,
    k33c45_truescore, k35c45_truescore, k37c45_truescore,
    k39c45_truescore))

knitr::kable(c45_results[1:19, ], caption = "c45_results")

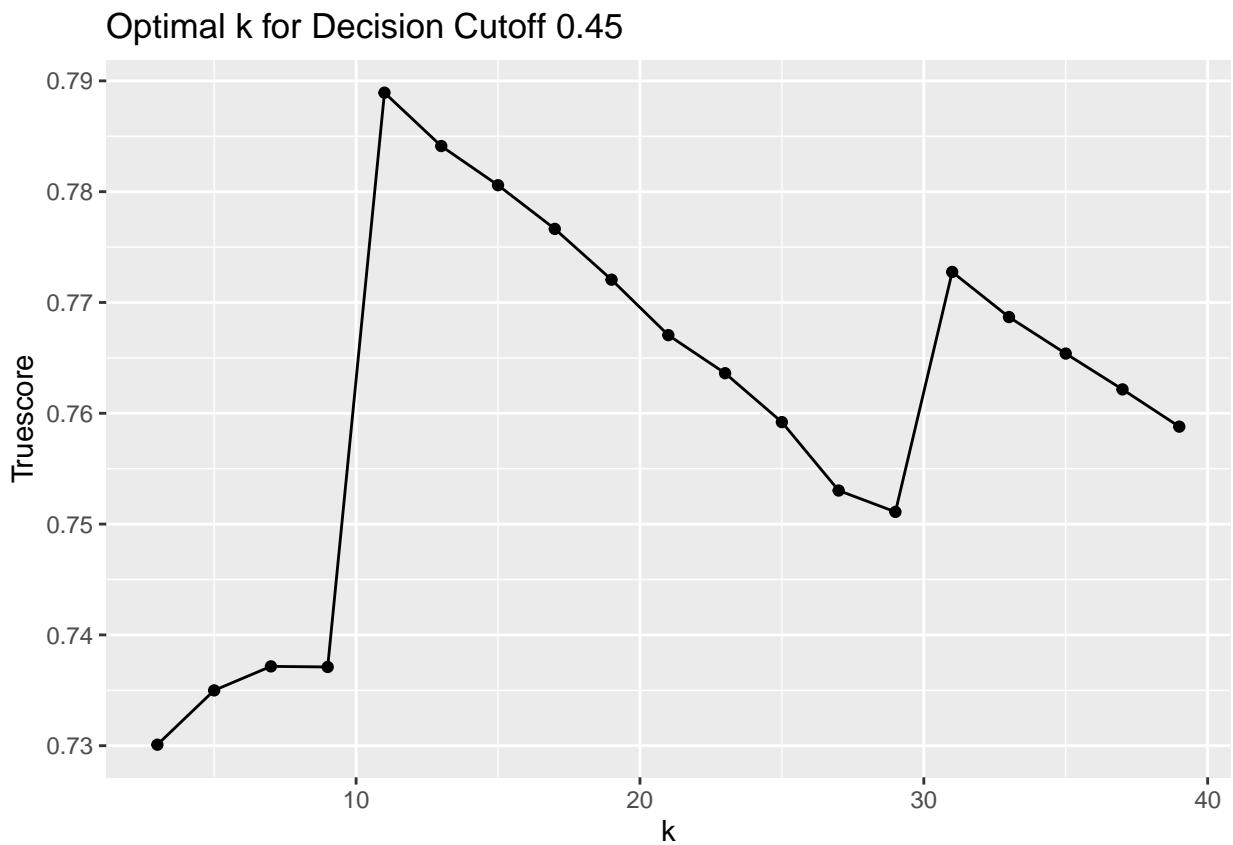
```

Table 28: c45_results

k	Cut	TPR	TNR	Truescore
3	0.45	0.616215	0.895600	0.730092
5	0.45	0.619076	0.904321	0.734993
7	0.45	0.620131	0.908631	0.737159
9	0.45	0.618474	0.912050	0.737106

k	Cut	TPR	TNR	Truescore
11	0.45	0.721084	0.870875	0.788932
13	0.45	0.708434	0.877911	0.784119
15	0.45	0.698293	0.884864	0.780585
17	0.45	0.689006	0.889819	0.776642
19	0.45	0.679769	0.893337	0.772056
21	0.45	0.670331	0.896393	0.767053
23	0.45	0.663805	0.898755	0.763616
25	0.45	0.656074	0.900803	0.759204
27	0.45	0.646386	0.901810	0.753028
29	0.45	0.642721	0.903445	0.751101
31	0.45	0.685040	0.886235	0.772756
33	0.45	0.677259	0.888647	0.768685
35	0.45	0.670683	0.891240	0.765389
37	0.45	0.664458	0.893535	0.762156
39	0.45	0.658484	0.895154	0.758793

```
ggplot(c45_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.45")
```



```
#####
# 0.46 Cutoff
#####
```

```

# For the decision cutoff of 0.46, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c46 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred46, obs))
  confusionMatrix(ss$pred46, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c46) <- fk_dfs_v

cm_c46_tables <- sapply(cm_c46, "[[", 2)
cm_c46_tables <- as_tibble(cm_c46_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.46.

k3c46_tpr <- round(sum(cm_c46_tables[1, 1:10]) /
  (sum(cm_c46_tables[1, 1:10]) + sum(cm_c46_tables[2, 1:10])), 6)
k3c46_tnr <- round(sum(cm_c46_tables[4, 1:10]) /
  (sum(cm_c46_tables[4, 1:10]) + sum(cm_c46_tables[3, 1:10])), 6)
k3c46_truescore <- round((2 * k3c46_tpr * k3c46_tnr) /
  (k3c46_tpr + k3c46_tnr), 6)

k5c46_tpr <- round(sum(cm_c46_tables[1, 11:20]) /
  (sum(cm_c46_tables[1, 11:20]) + sum(cm_c46_tables[2, 11:20])), 6)
k5c46_tnr <- round(sum(cm_c46_tables[4, 11:20]) /
  (sum(cm_c46_tables[4, 11:20]) + sum(cm_c46_tables[3, 11:20])), 6)
k5c46_truescore <- round((2 * k5c46_tpr * k5c46_tnr) /
  (k5c46_tpr + k5c46_tnr), 6)

k7c46_tpr <- round(sum(cm_c46_tables[1, 21:30]) /
  (sum(cm_c46_tables[1, 21:30]) + sum(cm_c46_tables[2, 21:30])), 6)
k7c46_tnr <- round(sum(cm_c46_tables[4, 21:30]) /
  (sum(cm_c46_tables[4, 21:30]) + sum(cm_c46_tables[3, 21:30])), 6)
k7c46_truescore <- round((2 * k7c46_tpr * k7c46_tnr) /
  (k7c46_tpr + k7c46_tnr), 6)

k9c46_tpr <- round(sum(cm_c46_tables[1, 31:40]) /
  (sum(cm_c46_tables[1, 31:40]) + sum(cm_c46_tables[2, 31:40])), 6)
k9c46_tnr <- round(sum(cm_c46_tables[4, 31:40]) /
  (sum(cm_c46_tables[4, 31:40]) + sum(cm_c46_tables[3, 31:40])), 6)
k9c46_truescore <- round((2 * k9c46_tpr * k9c46_tnr) /
  (k9c46_tpr + k9c46_tnr), 6)

k11c46_tpr <- round(sum(cm_c46_tables[1, 41:50]) /
  (sum(cm_c46_tables[1, 41:50]) + sum(cm_c46_tables[2, 41:50])), 6)
k11c46_tnr <- round(sum(cm_c46_tables[4, 41:50]) /
  (sum(cm_c46_tables[4, 41:50]) + sum(cm_c46_tables[3, 41:50])), 6)
k11c46_truescore <- round((2 * k11c46_tpr * k11c46_tnr) /
  (k11c46_tpr + k11c46_tnr), 6)

```

```

k13c46_tpr <- round(sum(cm_c46_tables[1, 51:60]) /
  (sum(cm_c46_tables[1, 51:60]) + sum(cm_c46_tables[2, 51:60])), 6)
k13c46_tnr <- round(sum(cm_c46_tables[4, 51:60]) /
  (sum(cm_c46_tables[4, 51:60]) + sum(cm_c46_tables[3, 51:60])), 6)
k13c46_truescore <- round((2 * k13c46_tpr * k13c46_tnr) /
  (k13c46_tpr + k13c46_tnr), 6)

k15c46_tpr <- round(sum(cm_c46_tables[1, 61:70]) /
  (sum(cm_c46_tables[1, 61:70]) + sum(cm_c46_tables[2, 61:70])), 6)
k15c46_tnr <- round(sum(cm_c46_tables[4, 61:70]) /
  (sum(cm_c46_tables[4, 61:70]) + sum(cm_c46_tables[3, 61:70])), 6)
k15c46_truescore <- round((2 * k15c46_tpr * k15c46_tnr) /
  (k15c46_tpr + k15c46_tnr), 6)

k17c46_tpr <- round(sum(cm_c46_tables[1, 71:80]) /
  (sum(cm_c46_tables[1, 71:80]) + sum(cm_c46_tables[2, 71:80])), 6)
k17c46_tnr <- round(sum(cm_c46_tables[4, 71:80]) /
  (sum(cm_c46_tables[4, 71:80]) + sum(cm_c46_tables[3, 71:80])), 6)
k17c46_truescore <- round((2 * k17c46_tpr * k17c46_tnr) /
  (k17c46_tpr + k17c46_tnr), 6)

k19c46_tpr <- round(sum(cm_c46_tables[1, 81:90]) /
  (sum(cm_c46_tables[1, 81:90]) + sum(cm_c46_tables[2, 81:90])), 6)
k19c46_tnr <- round(sum(cm_c46_tables[4, 81:90]) /
  (sum(cm_c46_tables[4, 81:90]) + sum(cm_c46_tables[3, 81:90])), 6)
k19c46_truescore <- round((2 * k19c46_tpr * k19c46_tnr) /
  (k19c46_tpr + k19c46_tnr), 6)

k21c46_tpr <- round(sum(cm_c46_tables[1, 91:100]) /
  (sum(cm_c46_tables[1, 91:100]) + sum(cm_c46_tables[2, 91:100])), 6)
k21c46_tnr <- round(sum(cm_c46_tables[4, 91:100]) /
  (sum(cm_c46_tables[4, 91:100]) + sum(cm_c46_tables[3, 91:100])), 6)
k21c46_truescore <- round((2 * k21c46_tpr * k21c46_tnr) /
  (k21c46_tpr + k21c46_tnr), 6)

k23c46_tpr <- round(sum(cm_c46_tables[1, 101:110]) /
  (sum(cm_c46_tables[1, 101:110]) + sum(cm_c46_tables[2, 101:110])), 6)
k23c46_tnr <- round(sum(cm_c46_tables[4, 101:110]) /
  (sum(cm_c46_tables[4, 101:110]) + sum(cm_c46_tables[3, 101:110])), 6)
k23c46_truescore <- round((2 * k23c46_tpr * k23c46_tnr) /
  (k23c46_tpr + k23c46_tnr), 6)

k25c46_tpr <- round(sum(cm_c46_tables[1, 111:120]) /
  (sum(cm_c46_tables[1, 111:120]) + sum(cm_c46_tables[2, 111:120])), 6)
k25c46_tnr <- round(sum(cm_c46_tables[4, 111:120]) /
  (sum(cm_c46_tables[4, 111:120]) + sum(cm_c46_tables[3, 111:120])), 6)
k25c46_truescore <- round((2 * k25c46_tpr * k25c46_tnr) /
  (k25c46_tpr + k25c46_tnr), 6)

k27c46_tpr <- round(sum(cm_c46_tables[1, 121:130]) /
  (sum(cm_c46_tables[1, 121:130]) + sum(cm_c46_tables[2, 121:130])), 6)
k27c46_tnr <- round(sum(cm_c46_tables[4, 121:130]) /
  (sum(cm_c46_tables[4, 121:130]) + sum(cm_c46_tables[3, 121:130])), 6)

```

```

k27c46_truescore <- round((2 * k27c46_tpr * k27c46_tnr) /
  (k27c46_tpr + k27c46_tnr), 6)

k29c46_tpr <- round(sum(cm_c46_tables[1, 131:140]) /
  (sum(cm_c46_tables[1, 131:140]) + sum(cm_c46_tables[2, 131:140])), 6)
k29c46_tnr <- round(sum(cm_c46_tables[4, 131:140]) /
  (sum(cm_c46_tables[4, 131:140]) + sum(cm_c46_tables[3, 131:140])), 6)
k29c46_truescore <- round((2 * k29c46_tpr * k29c46_tnr) /
  (k29c46_tpr + k29c46_tnr), 6)

k31c46_tpr <- round(sum(cm_c46_tables[1, 141:150]) /
  (sum(cm_c46_tables[1, 141:150]) + sum(cm_c46_tables[2, 141:150])), 6)
k31c46_tnr <- round(sum(cm_c46_tables[4, 141:150]) /
  (sum(cm_c46_tables[4, 141:150]) + sum(cm_c46_tables[3, 141:150])), 6)
k31c46_truescore <- round((2 * k31c46_tpr * k31c46_tnr) /
  (k31c46_tpr + k31c46_tnr), 6)

k33c46_tpr <- round(sum(cm_c46_tables[1, 151:160]) /
  (sum(cm_c46_tables[1, 151:160]) + sum(cm_c46_tables[2, 151:160])), 6)
k33c46_tnr <- round(sum(cm_c46_tables[4, 151:160]) /
  (sum(cm_c46_tables[4, 151:160]) + sum(cm_c46_tables[3, 151:160])), 6)
k33c46_truescore <- round((2 * k33c46_tpr * k33c46_tnr) /
  (k33c46_tpr + k33c46_tnr), 6)

k35c46_tpr <- round(sum(cm_c46_tables[1, 161:170]) /
  (sum(cm_c46_tables[1, 161:170]) + sum(cm_c46_tables[2, 161:170])), 6)
k35c46_tnr <- round(sum(cm_c46_tables[4, 161:170]) /
  (sum(cm_c46_tables[4, 161:170]) + sum(cm_c46_tables[3, 161:170])), 6)
k35c46_truescore <- round((2 * k35c46_tpr * k35c46_tnr) /
  (k35c46_tpr + k35c46_tnr), 6)

k37c46_tpr <- round(sum(cm_c46_tables[1, 171:180]) /
  (sum(cm_c46_tables[1, 171:180]) + sum(cm_c46_tables[2, 171:180])), 6)
k37c46_tnr <- round(sum(cm_c46_tables[4, 171:180]) /
  (sum(cm_c46_tables[4, 171:180]) + sum(cm_c46_tables[3, 171:180])), 6)
k37c46_truescore <- round((2 * k37c46_tpr * k37c46_tnr) /
  (k37c46_tpr + k37c46_tnr), 6)

k39c46_tpr <- round(sum(cm_c46_tables[1, 181:190]) /
  (sum(cm_c46_tables[1, 181:190]) + sum(cm_c46_tables[2, 181:190])), 6)
k39c46_tnr <- round(sum(cm_c46_tables[4, 181:190]) /
  (sum(cm_c46_tables[4, 181:190]) + sum(cm_c46_tables[3, 181:190])), 6)
k39c46_truescore <- round((2 * k39c46_tpr * k39c46_tnr) /
  (k39c46_tpr + k39c46_tnr), 6)

# Compile the 0.46 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c46_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.46,
  TPR = c(k3c46_tpr, k5c46_tpr, k7c46_tpr, k9c46_tpr, k11c46_tpr,
    k13c46_tpr, k15c46_tpr, k17c46_tpr, k19c46_tpr, k21c46_tpr,
    k23c46_tpr, k25c46_tpr, k27c46_tpr, k29c46_tpr, k31c46_tpr,
    k33c46_tpr, k35c46_tpr, k37c46_tpr, k39c46_tpr))

```

```

        k33c46_tpr, k35c46_tpr, k37c46_tpr, k39c46_tpr),
TNR = c(k3c46_tnr, k5c46_tnr, k7c46_tnr, k9c46_tnr, k11c46_tnr,
         k13c46_tnr, k15c46_tnr, k17c46_tnr, k19c46_tnr, k21c46_tnr,
         k23c46_tnr, k25c46_tnr, k27c46_tnr, k29c46_tnr, k31c46_tnr,
         k33c46_tnr, k35c46_tnr, k37c46_tnr, k39c46_tnr),
Truescore = c(k3c46_truescore, k5c46_truescore, k7c46_truescore,
              k9c46_truescore, k11c46_truescore, k13c46_truescore,
              k15c46_truescore, k17c46_truescore, k19c46_truescore,
              k21c46_truescore, k23c46_truescore, k25c46_truescore,
              k27c46_truescore, k29c46_truescore, k31c46_truescore,
              k33c46_truescore, k35c46_truescore, k37c46_truescore,
              k39c46_truescore))

knitr::kable(c46_results[1:19, ], caption = "c46_results")

```

Table 29: c46_results

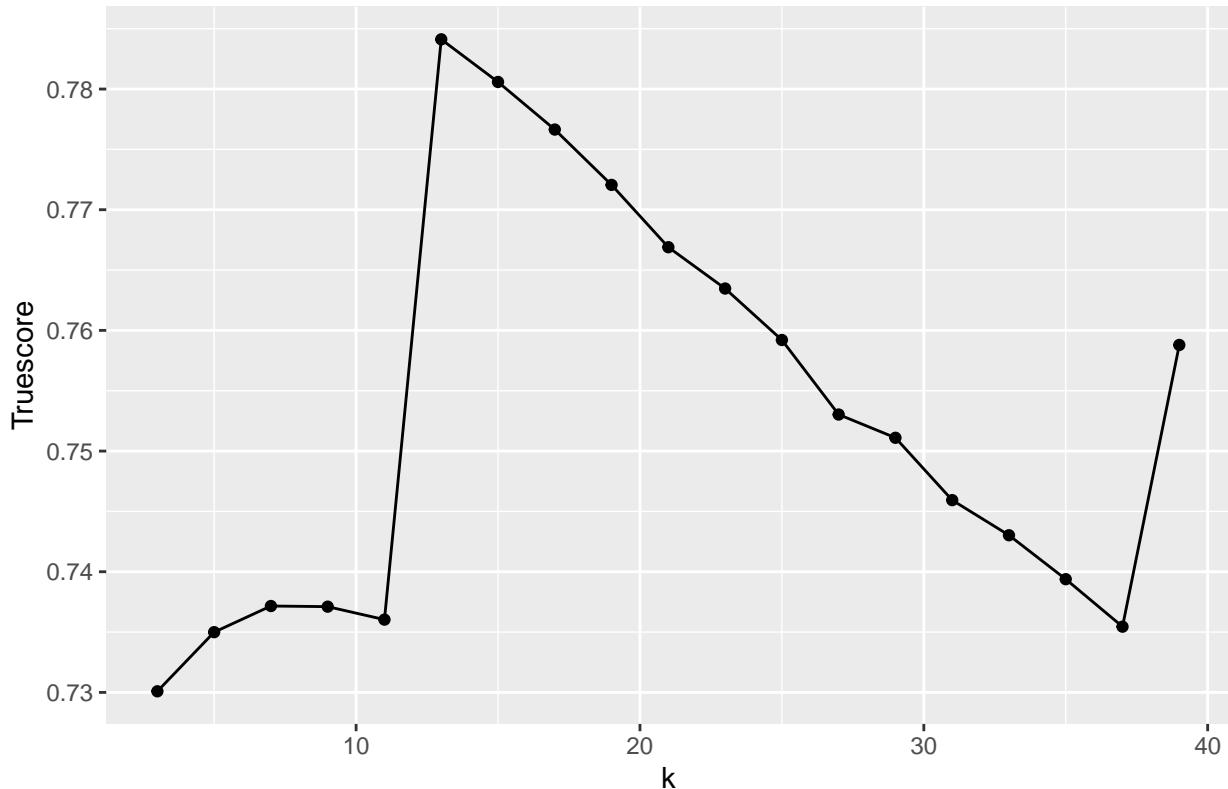
k	Cut	TPR	TNR	Truescore
3	0.46	0.616215	0.895600	0.730092
5	0.46	0.619076	0.904321	0.734993
7	0.46	0.620131	0.908631	0.737159
9	0.46	0.618474	0.912050	0.737106
11	0.46	0.616165	0.913801	0.736032
13	0.46	0.708434	0.877911	0.784119
15	0.46	0.698293	0.884864	0.780585
17	0.46	0.689006	0.889819	0.776642
19	0.46	0.679769	0.893337	0.772056
21	0.46	0.670030	0.896492	0.766892
23	0.46	0.663554	0.898804	0.763468
25	0.46	0.656074	0.900803	0.759204
27	0.46	0.646386	0.901810	0.753028
29	0.46	0.642721	0.903445	0.751101
31	0.46	0.634337	0.905180	0.745934
33	0.46	0.629317	0.906881	0.743024
35	0.46	0.623293	0.908615	0.739383
37	0.46	0.617319	0.909490	0.735449
39	0.46	0.658484	0.895154	0.758793

```

ggplot(c46_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.46")

```

Optimal k for Decision Cutoff 0.46



```
#####
# 0.47 Cutoff
#####

# For the decision cutoff of 0.47, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c47 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred47, obs))
  confusionMatrix(ss$pred47, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c47) <- fk_dfs_v

cm_c47_tables <- sapply(cm_c47, "[[", 2)
cm_c47_tables <- as_tibble(cm_c47_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.47.

k3c47_tpr <- round(sum(cm_c47_tables[1, 1:10]) /
  (sum(cm_c47_tables[1, 1:10]) + sum(cm_c47_tables[2, 1:10])), 6)
k3c47_tnr <- round(sum(cm_c47_tables[4, 1:10]) /
  (sum(cm_c47_tables[4, 1:10]) + sum(cm_c47_tables[3, 1:10])), 6)
```

```

k3c47_truescore <- round((2 * k3c47_tpr * k3c47_tnr) /
  (k3c47_tpr + k3c47_tnr), 6)

k5c47_tpr <- round(sum(cm_c47_tables[1, 11:20]) /
  (sum(cm_c47_tables[1, 11:20]) + sum(cm_c47_tables[2, 11:20])), 6)
k5c47_tnr <- round(sum(cm_c47_tables[4, 11:20]) /
  (sum(cm_c47_tables[4, 11:20]) + sum(cm_c47_tables[3, 11:20])), 6)
k5c47_truescore <- round((2 * k5c47_tpr * k5c47_tnr) /
  (k5c47_tpr + k5c47_tnr), 6)

k7c47_tpr <- round(sum(cm_c47_tables[1, 21:30]) /
  (sum(cm_c47_tables[1, 21:30]) + sum(cm_c47_tables[2, 21:30])), 6)
k7c47_tnr <- round(sum(cm_c47_tables[4, 21:30]) /
  (sum(cm_c47_tables[4, 21:30]) + sum(cm_c47_tables[3, 21:30])), 6)
k7c47_truescore <- round((2 * k7c47_tpr * k7c47_tnr) /
  (k7c47_tpr + k7c47_tnr), 6)

k9c47_tpr <- round(sum(cm_c47_tables[1, 31:40]) /
  (sum(cm_c47_tables[1, 31:40]) + sum(cm_c47_tables[2, 31:40])), 6)
k9c47_tnr <- round(sum(cm_c47_tables[4, 31:40]) /
  (sum(cm_c47_tables[4, 31:40]) + sum(cm_c47_tables[3, 31:40])), 6)
k9c47_truescore <- round((2 * k9c47_tpr * k9c47_tnr) /
  (k9c47_tpr + k9c47_tnr), 6)

k11c47_tpr <- round(sum(cm_c47_tables[1, 41:50]) /
  (sum(cm_c47_tables[1, 41:50]) + sum(cm_c47_tables[2, 41:50])), 6)
k11c47_tnr <- round(sum(cm_c47_tables[4, 41:50]) /
  (sum(cm_c47_tables[4, 41:50]) + sum(cm_c47_tables[3, 41:50])), 6)
k11c47_truescore <- round((2 * k11c47_tpr * k11c47_tnr) /
  (k11c47_tpr + k11c47_tnr), 6)

k13c47_tpr <- round(sum(cm_c47_tables[1, 51:60]) /
  (sum(cm_c47_tables[1, 51:60]) + sum(cm_c47_tables[2, 51:60])), 6)
k13c47_tnr <- round(sum(cm_c47_tables[4, 51:60]) /
  (sum(cm_c47_tables[4, 51:60]) + sum(cm_c47_tables[3, 51:60])), 6)
k13c47_truescore <- round((2 * k13c47_tpr * k13c47_tnr) /
  (k13c47_tpr + k13c47_tnr), 6)

k15c47_tpr <- round(sum(cm_c47_tables[1, 61:70]) /
  (sum(cm_c47_tables[1, 61:70]) + sum(cm_c47_tables[2, 61:70])), 6)
k15c47_tnr <- round(sum(cm_c47_tables[4, 61:70]) /
  (sum(cm_c47_tables[4, 61:70]) + sum(cm_c47_tables[3, 61:70])), 6)
k15c47_truescore <- round((2 * k15c47_tpr * k15c47_tnr) /
  (k15c47_tpr + k15c47_tnr), 6)

k17c47_tpr <- round(sum(cm_c47_tables[1, 71:80]) /
  (sum(cm_c47_tables[1, 71:80]) + sum(cm_c47_tables[2, 71:80])), 6)
k17c47_tnr <- round(sum(cm_c47_tables[4, 71:80]) /
  (sum(cm_c47_tables[4, 71:80]) + sum(cm_c47_tables[3, 71:80])), 6)
k17c47_truescore <- round((2 * k17c47_tpr * k17c47_tnr) /
  (k17c47_tpr + k17c47_tnr), 6)

k19c47_tpr <- round(sum(cm_c47_tables[1, 81:90]) /

```

```

(sum(cm_c47_tables[1, 81:90]) + sum(cm_c47_tables[2, 81:90])), 6)
k19c47_tnr <- round(sum(cm_c47_tables[4, 81:90]) /
  (sum(cm_c47_tables[4, 81:90]) + sum(cm_c47_tables[3, 81:90])), 6)
k19c47_truescore <- round((2 * k19c47_tpr * k19c47_tnr) /
  (k19c47_tpr + k19c47_tnr), 6)

k21c47_tpr <- round(sum(cm_c47_tables[1, 91:100]) /
  (sum(cm_c47_tables[1, 91:100]) + sum(cm_c47_tables[2, 91:100])), 6)
k21c47_tnr <- round(sum(cm_c47_tables[4, 91:100]) /
  (sum(cm_c47_tables[4, 91:100]) + sum(cm_c47_tables[3, 91:100])), 6)
k21c47_truescore <- round((2 * k21c47_tpr * k21c47_tnr) /
  (k21c47_tpr + k21c47_tnr), 6)

k23c47_tpr <- round(sum(cm_c47_tables[1, 101:110]) /
  (sum(cm_c47_tables[1, 101:110]) + sum(cm_c47_tables[2, 101:110])), 6)
k23c47_tnr <- round(sum(cm_c47_tables[4, 101:110]) /
  (sum(cm_c47_tables[4, 101:110]) + sum(cm_c47_tables[3, 101:110])), 6)
k23c47_truescore <- round((2 * k23c47_tpr * k23c47_tnr) /
  (k23c47_tpr + k23c47_tnr), 6)

k25c47_tpr <- round(sum(cm_c47_tables[1, 111:120]) /
  (sum(cm_c47_tables[1, 111:120]) + sum(cm_c47_tables[2, 111:120])), 6)
k25c47_tnr <- round(sum(cm_c47_tables[4, 111:120]) /
  (sum(cm_c47_tables[4, 111:120]) + sum(cm_c47_tables[3, 111:120])), 6)
k25c47_truescore <- round((2 * k25c47_tpr * k25c47_tnr) /
  (k25c47_tpr + k25c47_tnr), 6)

k27c47_tpr <- round(sum(cm_c47_tables[1, 121:130]) /
  (sum(cm_c47_tables[1, 121:130]) + sum(cm_c47_tables[2, 121:130])), 6)
k27c47_tnr <- round(sum(cm_c47_tables[4, 121:130]) /
  (sum(cm_c47_tables[4, 121:130]) + sum(cm_c47_tables[3, 121:130])), 6)
k27c47_truescore <- round((2 * k27c47_tpr * k27c47_tnr) /
  (k27c47_tpr + k27c47_tnr), 6)

k29c47_tpr <- round(sum(cm_c47_tables[1, 131:140]) /
  (sum(cm_c47_tables[1, 131:140]) + sum(cm_c47_tables[2, 131:140])), 6)
k29c47_tnr <- round(sum(cm_c47_tables[4, 131:140]) /
  (sum(cm_c47_tables[4, 131:140]) + sum(cm_c47_tables[3, 131:140])), 6)
k29c47_truescore <- round((2 * k29c47_tpr * k29c47_tnr) /
  (k29c47_tpr + k29c47_tnr), 6)

k31c47_tpr <- round(sum(cm_c47_tables[1, 141:150]) /
  (sum(cm_c47_tables[1, 141:150]) + sum(cm_c47_tables[2, 141:150])), 6)
k31c47_tnr <- round(sum(cm_c47_tables[4, 141:150]) /
  (sum(cm_c47_tables[4, 141:150]) + sum(cm_c47_tables[3, 141:150])), 6)
k31c47_truescore <- round((2 * k31c47_tpr * k31c47_tnr) /
  (k31c47_tpr + k31c47_tnr), 6)

k33c47_tpr <- round(sum(cm_c47_tables[1, 151:160]) /
  (sum(cm_c47_tables[1, 151:160]) + sum(cm_c47_tables[2, 151:160])), 6)
k33c47_tnr <- round(sum(cm_c47_tables[4, 151:160]) /
  (sum(cm_c47_tables[4, 151:160]) + sum(cm_c47_tables[3, 151:160])), 6)
k33c47_truescore <- round((2 * k33c47_tpr * k33c47_tnr) /

```

```

(k33c47_tpr + k33c47_tnr), 6)

k35c47_tpr <- round(sum(cm_c47_tables[1, 161:170]) /
  (sum(cm_c47_tables[1, 161:170]) + sum(cm_c47_tables[2, 161:170])), 6)
k35c47_tnr <- round(sum(cm_c47_tables[4, 161:170]) /
  (sum(cm_c47_tables[4, 161:170]) + sum(cm_c47_tables[3, 161:170])), 6)
k35c47_truescore <- round((2 * k35c47_tpr * k35c47_tnr) /
  (k35c47_tpr + k35c47_tnr), 6)

k37c47_tpr <- round(sum(cm_c47_tables[1, 171:180]) /
  (sum(cm_c47_tables[1, 171:180]) + sum(cm_c47_tables[2, 171:180])), 6)
k37c47_tnr <- round(sum(cm_c47_tables[4, 171:180]) /
  (sum(cm_c47_tables[4, 171:180]) + sum(cm_c47_tables[3, 171:180])), 6)
k37c47_truescore <- round((2 * k37c47_tpr * k37c47_tnr) /
  (k37c47_tpr + k37c47_tnr), 6)

k39c47_tpr <- round(sum(cm_c47_tables[1, 181:190]) /
  (sum(cm_c47_tables[1, 181:190]) + sum(cm_c47_tables[2, 181:190])), 6)
k39c47_tnr <- round(sum(cm_c47_tables[4, 181:190]) /
  (sum(cm_c47_tables[4, 181:190]) + sum(cm_c47_tables[3, 181:190])), 6)
k39c47_truescore <- round((2 * k39c47_tpr * k39c47_tnr) /
  (k39c47_tpr + k39c47_tnr), 6)

# Compile the 0.47 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c47_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.47,
  TPR = c(k3c47_tpr, k5c47_tpr, k7c47_tpr, k9c47_tpr, k11c47_tpr,
    k13c47_tpr, k15c47_tpr, k17c47_tpr, k19c47_tpr, k21c47_tpr,
    k23c47_tpr, k25c47_tpr, k27c47_tpr, k29c47_tpr, k31c47_tpr,
    k33c47_tpr, k35c47_tpr, k37c47_tpr, k39c47_tpr),
  TNR = c(k3c47_tnr, k5c47_tnr, k7c47_tnr, k9c47_tnr, k11c47_tnr,
    k13c47_tnr, k15c47_tnr, k17c47_tnr, k19c47_tnr, k21c47_tnr,
    k23c47_tnr, k25c47_tnr, k27c47_tnr, k29c47_tnr, k31c47_tnr,
    k33c47_tnr, k35c47_tnr, k37c47_tnr, k39c47_tnr),
  Truescore = c(k3c47_truescore, k5c47_truescore, k7c47_truescore,
    k9c47_truescore, k11c47_truescore, k13c47_truescore,
    k15c47_truescore, k17c47_truescore, k19c47_truescore,
    k21c47_truescore, k23c47_truescore, k25c47_truescore,
    k27c47_truescore, k29c47_truescore, k31c47_truescore,
    k33c47_truescore, k35c47_truescore, k37c47_truescore,
    k39c47_truescore))

knitr::kable(c47_results[1:19, ], caption = "c47_results")

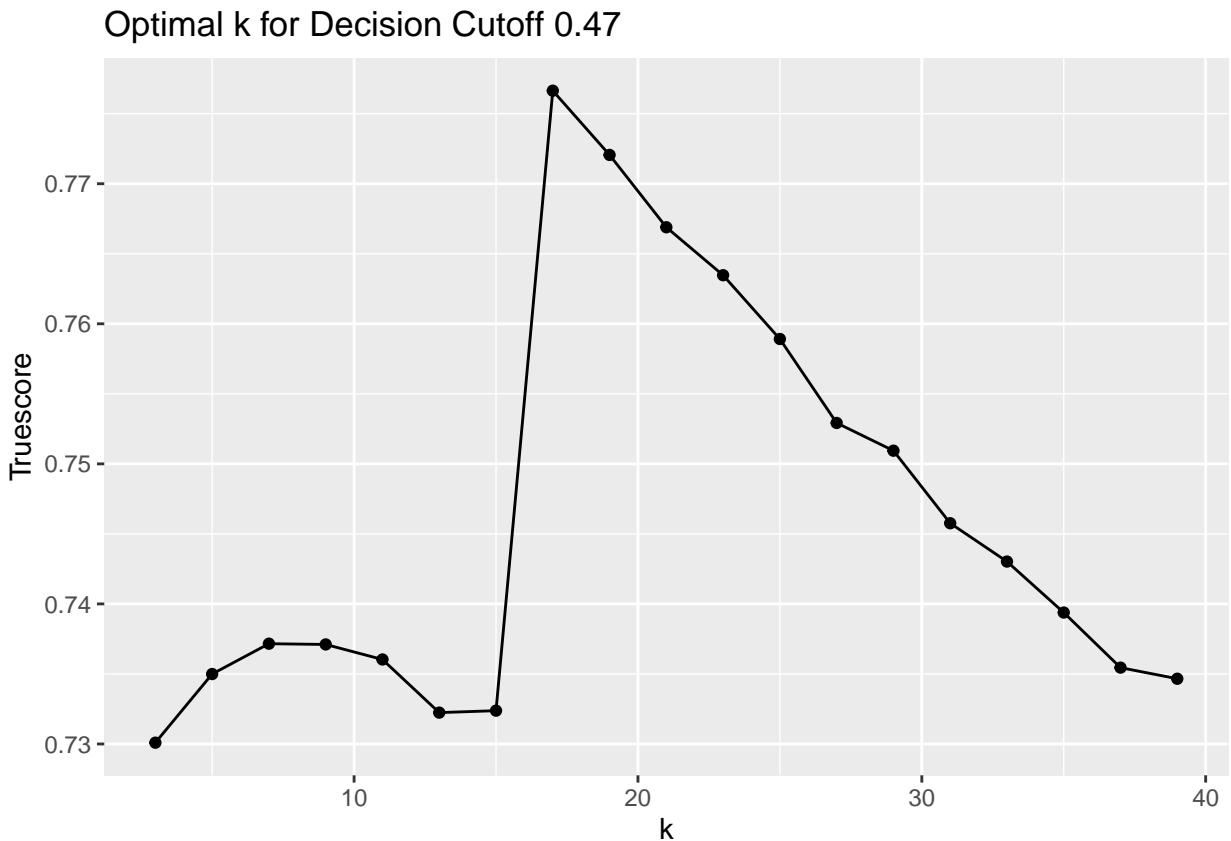
```

Table 30: c47_results

k	Cut	TPR	TNR	Truescore
3	0.47	0.616215	0.895600	0.730092
5	0.47	0.619076	0.904321	0.734993
7	0.47	0.620131	0.908631	0.737159
9	0.47	0.618474	0.912050	0.737106

k	Cut	TPR	TNR	Truescore
11	0.47	0.616165	0.913801	0.736032
13	0.47	0.609990	0.915767	0.732238
15	0.47	0.609337	0.917682	0.732378
17	0.47	0.689006	0.889819	0.776642
19	0.47	0.679769	0.893337	0.772056
21	0.47	0.670030	0.896492	0.766892
23	0.47	0.663554	0.898804	0.763468
25	0.47	0.655572	0.900935	0.758914
27	0.47	0.646135	0.901992	0.752921
29	0.47	0.642420	0.903577	0.750941
31	0.47	0.634036	0.905279	0.745760
33	0.47	0.629317	0.906881	0.743024
35	0.47	0.623293	0.908615	0.739383
37	0.47	0.617319	0.909490	0.735449
39	0.47	0.615713	0.910564	0.734658

```
ggplot(c47_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.47")
```



```
#####
# 0.48 Cutoff
#####
```

```

# For the decision cutoff of 0.48, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c48 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred48, obs))
  confusionMatrix(ss$pred48, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c48) <- fk_dfs_v

cm_c48_tables <- sapply(cm_c48, "[[", 2)
cm_c48_tables <- as_tibble(cm_c48_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.48.

k3c48_tpr <- round(sum(cm_c48_tables[1, 1:10]) /
  (sum(cm_c48_tables[1, 1:10]) + sum(cm_c48_tables[2, 1:10])), 6)
k3c48_tnr <- round(sum(cm_c48_tables[4, 1:10]) /
  (sum(cm_c48_tables[4, 1:10]) + sum(cm_c48_tables[3, 1:10])), 6)
k3c48_truescore <- round((2 * k3c48_tpr * k3c48_tnr) /
  (k3c48_tpr + k3c48_tnr), 6)

k5c48_tpr <- round(sum(cm_c48_tables[1, 11:20]) /
  (sum(cm_c48_tables[1, 11:20]) + sum(cm_c48_tables[2, 11:20])), 6)
k5c48_tnr <- round(sum(cm_c48_tables[4, 11:20]) /
  (sum(cm_c48_tables[4, 11:20]) + sum(cm_c48_tables[3, 11:20])), 6)
k5c48_truescore <- round((2 * k5c48_tpr * k5c48_tnr) /
  (k5c48_tpr + k5c48_tnr), 6)

k7c48_tpr <- round(sum(cm_c48_tables[1, 21:30]) /
  (sum(cm_c48_tables[1, 21:30]) + sum(cm_c48_tables[2, 21:30])), 6)
k7c48_tnr <- round(sum(cm_c48_tables[4, 21:30]) /
  (sum(cm_c48_tables[4, 21:30]) + sum(cm_c48_tables[3, 21:30])), 6)
k7c48_truescore <- round((2 * k7c48_tpr * k7c48_tnr) /
  (k7c48_tpr + k7c48_tnr), 6)

k9c48_tpr <- round(sum(cm_c48_tables[1, 31:40]) /
  (sum(cm_c48_tables[1, 31:40]) + sum(cm_c48_tables[2, 31:40])), 6)
k9c48_tnr <- round(sum(cm_c48_tables[4, 31:40]) /
  (sum(cm_c48_tables[4, 31:40]) + sum(cm_c48_tables[3, 31:40])), 6)
k9c48_truescore <- round((2 * k9c48_tpr * k9c48_tnr) /
  (k9c48_tpr + k9c48_tnr), 6)

k11c48_tpr <- round(sum(cm_c48_tables[1, 41:50]) /
  (sum(cm_c48_tables[1, 41:50]) + sum(cm_c48_tables[2, 41:50])), 6)
k11c48_tnr <- round(sum(cm_c48_tables[4, 41:50]) /
  (sum(cm_c48_tables[4, 41:50]) + sum(cm_c48_tables[3, 41:50])), 6)
k11c48_truescore <- round((2 * k11c48_tpr * k11c48_tnr) /
  (k11c48_tpr + k11c48_tnr), 6)

```

```

k13c48_tpr <- round(sum(cm_c48_tables[1, 51:60]) /
  (sum(cm_c48_tables[1, 51:60]) + sum(cm_c48_tables[2, 51:60])), 6)
k13c48_tnr <- round(sum(cm_c48_tables[4, 51:60]) /
  (sum(cm_c48_tables[4, 51:60]) + sum(cm_c48_tables[3, 51:60])), 6)
k13c48_truescore <- round((2 * k13c48_tpr * k13c48_tnr) /
  (k13c48_tpr + k13c48_tnr), 6)

k15c48_tpr <- round(sum(cm_c48_tables[1, 61:70]) /
  (sum(cm_c48_tables[1, 61:70]) + sum(cm_c48_tables[2, 61:70])), 6)
k15c48_tnr <- round(sum(cm_c48_tables[4, 61:70]) /
  (sum(cm_c48_tables[4, 61:70]) + sum(cm_c48_tables[3, 61:70])), 6)
k15c48_truescore <- round((2 * k15c48_tpr * k15c48_tnr) /
  (k15c48_tpr + k15c48_tnr), 6)

k17c48_tpr <- round(sum(cm_c48_tables[1, 71:80]) /
  (sum(cm_c48_tables[1, 71:80]) + sum(cm_c48_tables[2, 71:80])), 6)
k17c48_tnr <- round(sum(cm_c48_tables[4, 71:80]) /
  (sum(cm_c48_tables[4, 71:80]) + sum(cm_c48_tables[3, 71:80])), 6)
k17c48_truescore <- round((2 * k17c48_tpr * k17c48_tnr) /
  (k17c48_tpr + k17c48_tnr), 6)

k19c48_tpr <- round(sum(cm_c48_tables[1, 81:90]) /
  (sum(cm_c48_tables[1, 81:90]) + sum(cm_c48_tables[2, 81:90])), 6)
k19c48_tnr <- round(sum(cm_c48_tables[4, 81:90]) /
  (sum(cm_c48_tables[4, 81:90]) + sum(cm_c48_tables[3, 81:90])), 6)
k19c48_truescore <- round((2 * k19c48_tpr * k19c48_tnr) /
  (k19c48_tpr + k19c48_tnr), 6)

k21c48_tpr <- round(sum(cm_c48_tables[1, 91:100]) /
  (sum(cm_c48_tables[1, 91:100]) + sum(cm_c48_tables[2, 91:100])), 6)
k21c48_tnr <- round(sum(cm_c48_tables[4, 91:100]) /
  (sum(cm_c48_tables[4, 91:100]) + sum(cm_c48_tables[3, 91:100])), 6)
k21c48_truescore <- round((2 * k21c48_tpr * k21c48_tnr) /
  (k21c48_tpr + k21c48_tnr), 6)

k23c48_tpr <- round(sum(cm_c48_tables[1, 101:110]) /
  (sum(cm_c48_tables[1, 101:110]) + sum(cm_c48_tables[2, 101:110])), 6)
k23c48_tnr <- round(sum(cm_c48_tables[4, 101:110]) /
  (sum(cm_c48_tables[4, 101:110]) + sum(cm_c48_tables[3, 101:110])), 6)
k23c48_truescore <- round((2 * k23c48_tpr * k23c48_tnr) /
  (k23c48_tpr + k23c48_tnr), 6)

k25c48_tpr <- round(sum(cm_c48_tables[1, 111:120]) /
  (sum(cm_c48_tables[1, 111:120]) + sum(cm_c48_tables[2, 111:120])), 6)
k25c48_tnr <- round(sum(cm_c48_tables[4, 111:120]) /
  (sum(cm_c48_tables[4, 111:120]) + sum(cm_c48_tables[3, 111:120])), 6)
k25c48_truescore <- round((2 * k25c48_tpr * k25c48_tnr) /
  (k25c48_tpr + k25c48_tnr), 6)

k27c48_tpr <- round(sum(cm_c48_tables[1, 121:130]) /
  (sum(cm_c48_tables[1, 121:130]) + sum(cm_c48_tables[2, 121:130])), 6)
k27c48_tnr <- round(sum(cm_c48_tables[4, 121:130]) /
  (sum(cm_c48_tables[4, 121:130]) + sum(cm_c48_tables[3, 121:130])), 6)

```

```

k27c48_truescore <- round((2 * k27c48_tpr * k27c48_tnr) /
  (k27c48_tpr + k27c48_tnr), 6)

k29c48_tpr <- round(sum(cm_c48_tables[1, 131:140]) /
  (sum(cm_c48_tables[1, 131:140]) + sum(cm_c48_tables[2, 131:140])), 6)
k29c48_tnr <- round(sum(cm_c48_tables[4, 131:140]) /
  (sum(cm_c48_tables[4, 131:140]) + sum(cm_c48_tables[3, 131:140])), 6)
k29c48_truescore <- round((2 * k29c48_tpr * k29c48_tnr) /
  (k29c48_tpr + k29c48_tnr), 6)

k31c48_tpr <- round(sum(cm_c48_tables[1, 141:150]) /
  (sum(cm_c48_tables[1, 141:150]) + sum(cm_c48_tables[2, 141:150])), 6)
k31c48_tnr <- round(sum(cm_c48_tables[4, 141:150]) /
  (sum(cm_c48_tables[4, 141:150]) + sum(cm_c48_tables[3, 141:150])), 6)
k31c48_truescore <- round((2 * k31c48_tpr * k31c48_tnr) /
  (k31c48_tpr + k31c48_tnr), 6)

k33c48_tpr <- round(sum(cm_c48_tables[1, 151:160]) /
  (sum(cm_c48_tables[1, 151:160]) + sum(cm_c48_tables[2, 151:160])), 6)
k33c48_tnr <- round(sum(cm_c48_tables[4, 151:160]) /
  (sum(cm_c48_tables[4, 151:160]) + sum(cm_c48_tables[3, 151:160])), 6)
k33c48_truescore <- round((2 * k33c48_tpr * k33c48_tnr) /
  (k33c48_tpr + k33c48_tnr), 6)

k35c48_tpr <- round(sum(cm_c48_tables[1, 161:170]) /
  (sum(cm_c48_tables[1, 161:170]) + sum(cm_c48_tables[2, 161:170])), 6)
k35c48_tnr <- round(sum(cm_c48_tables[4, 161:170]) /
  (sum(cm_c48_tables[4, 161:170]) + sum(cm_c48_tables[3, 161:170])), 6)
k35c48_truescore <- round((2 * k35c48_tpr * k35c48_tnr) /
  (k35c48_tpr + k35c48_tnr), 6)

k37c48_tpr <- round(sum(cm_c48_tables[1, 171:180]) /
  (sum(cm_c48_tables[1, 171:180]) + sum(cm_c48_tables[2, 171:180])), 6)
k37c48_tnr <- round(sum(cm_c48_tables[4, 171:180]) /
  (sum(cm_c48_tables[4, 171:180]) + sum(cm_c48_tables[3, 171:180])), 6)
k37c48_truescore <- round((2 * k37c48_tpr * k37c48_tnr) /
  (k37c48_tpr + k37c48_tnr), 6)

k39c48_tpr <- round(sum(cm_c48_tables[1, 181:190]) /
  (sum(cm_c48_tables[1, 181:190]) + sum(cm_c48_tables[2, 181:190])), 6)
k39c48_tnr <- round(sum(cm_c48_tables[4, 181:190]) /
  (sum(cm_c48_tables[4, 181:190]) + sum(cm_c48_tables[3, 181:190])), 6)
k39c48_truescore <- round((2 * k39c48_tpr * k39c48_tnr) /
  (k39c48_tpr + k39c48_tnr), 6)

# Compile the 0.48 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c48_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.48,
  TPR = c(k3c48_tpr, k5c48_tpr, k7c48_tpr, k9c48_tpr, k11c48_tpr,
    k13c48_tpr, k15c48_tpr, k17c48_tpr, k19c48_tpr, k21c48_tpr,
    k23c48_tpr, k25c48_tpr, k27c48_tpr, k29c48_tpr, k31c48_tpr,
    k33c48_tpr, k35c48_tpr, k37c48_tpr, k39c48_tpr))

```

```

        k33c48_tpr, k35c48_tpr, k37c48_tpr, k39c48_tpr),
TNR = c(k3c48_tnr, k5c48_tnr, k7c48_tnr, k9c48_tnr, k11c48_tnr,
         k13c48_tnr, k15c48_tnr, k17c48_tnr, k19c48_tnr, k21c48_tnr,
         k23c48_tnr, k25c48_tnr, k27c48_tnr, k29c48_tnr, k31c48_tnr,
         k33c48_tnr, k35c48_tnr, k37c48_tnr, k39c48_tnr),
Truescore = c(k3c48_truescore, k5c48_truescore, k7c48_truescore,
              k9c48_truescore, k11c48_truescore, k13c48_truescore,
              k15c48_truescore, k17c48_truescore, k19c48_truescore,
              k21c48_truescore, k23c48_truescore, k25c48_truescore,
              k27c48_truescore, k29c48_truescore, k31c48_truescore,
              k33c48_truescore, k35c48_truescore, k37c48_truescore,
              k39c48_truescore))

knitr::kable(c48_results[1:19, ], caption = "c48_results")

```

Table 31: c48_results

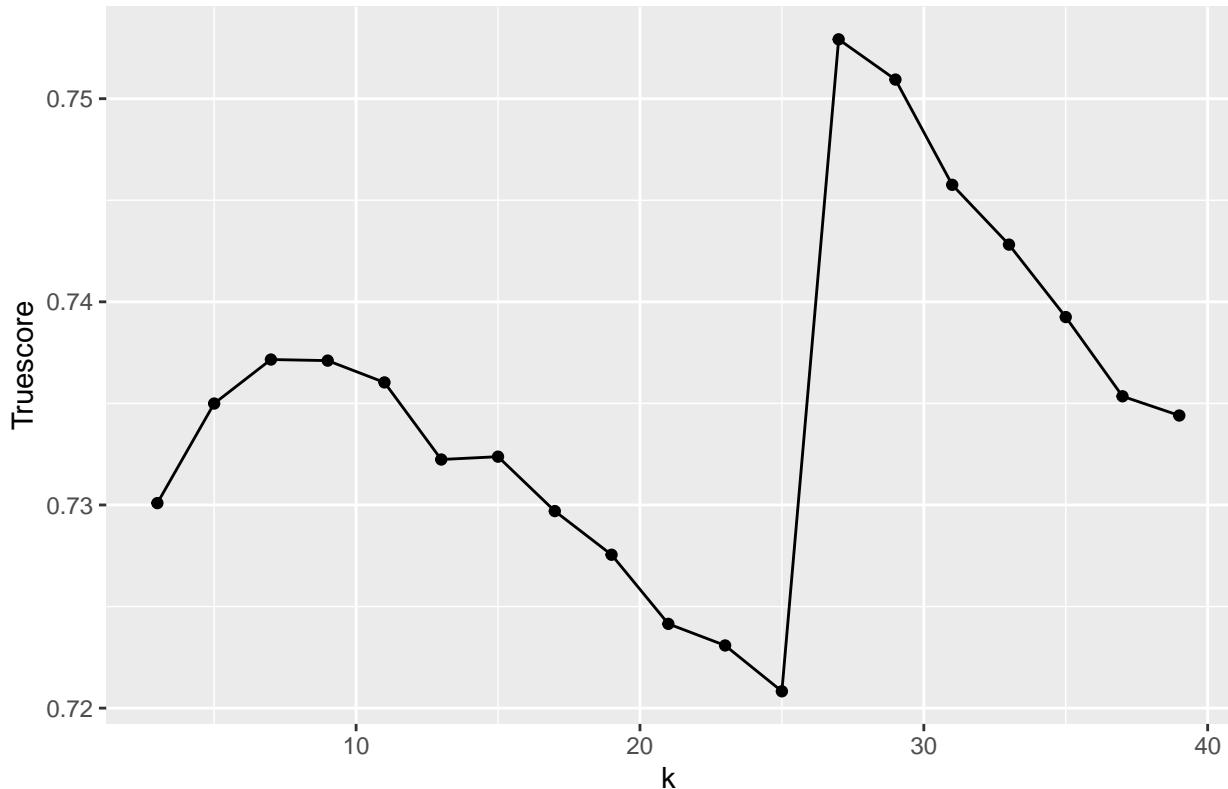
k	Cut	TPR	TNR	Truescore
3	0.48	0.616215	0.895600	0.730092
5	0.48	0.619076	0.904321	0.734993
7	0.48	0.620131	0.908631	0.737159
9	0.48	0.618474	0.912050	0.737106
11	0.48	0.616165	0.913801	0.736032
13	0.48	0.609990	0.915767	0.732238
15	0.48	0.609337	0.917682	0.732378
17	0.48	0.605371	0.918294	0.729699
19	0.48	0.601908	0.919483	0.727550
21	0.48	0.597189	0.919664	0.724148
23	0.48	0.595281	0.920771	0.723085
25	0.48	0.591968	0.921399	0.720828
27	0.48	0.646135	0.901992	0.752921
29	0.48	0.642420	0.903577	0.750941
31	0.48	0.634036	0.905279	0.745760
33	0.48	0.628966	0.906996	0.742817
35	0.48	0.623042	0.908747	0.739250
37	0.48	0.617118	0.909622	0.735350
39	0.48	0.615311	0.910663	0.734404

```

ggplot(c48_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.48")

```

Optimal k for Decision Cutoff 0.48



```
#####
# 0.49 Cutoff
#####

# For the decision cutoff of 0.49, generate a confusion matrix for
# every combination of k (3:39 by two) and fold (1 to 10).

cm_c49 <- sapply(fk_dfs_l, function(x) {
  ss <- subset(x, select = c(pred49, obs))
  confusionMatrix(ss$pred49, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c49) <- fk_dfs_v

cm_c49_tables <- sapply(cm_c49, "[[", 2)
cm_c49_tables <- as_tibble(cm_c49_tables)

# For each value of k, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each value of k when the decision cutoff is 0.49.

k3c49_tpr <- round(sum(cm_c49_tables[1, 1:10]) /
  (sum(cm_c49_tables[1, 1:10]) + sum(cm_c49_tables[2, 1:10])), 6)
k3c49_tnr <- round(sum(cm_c49_tables[4, 1:10]) /
  (sum(cm_c49_tables[4, 1:10]) + sum(cm_c49_tables[3, 1:10])), 6)
```

```

k3c49_truescore <- round((2 * k3c49_tpr * k3c49_tnr) /
  (k3c49_tpr + k3c49_tnr), 6)

k5c49_tpr <- round(sum(cm_c49_tables[1, 11:20]) /
  (sum(cm_c49_tables[1, 11:20]) + sum(cm_c49_tables[2, 11:20])), 6)
k5c49_tnr <- round(sum(cm_c49_tables[4, 11:20]) /
  (sum(cm_c49_tables[4, 11:20]) + sum(cm_c49_tables[3, 11:20])), 6)
k5c49_truescore <- round((2 * k5c49_tpr * k5c49_tnr) /
  (k5c49_tpr + k5c49_tnr), 6)

k7c49_tpr <- round(sum(cm_c49_tables[1, 21:30]) /
  (sum(cm_c49_tables[1, 21:30]) + sum(cm_c49_tables[2, 21:30])), 6)
k7c49_tnr <- round(sum(cm_c49_tables[4, 21:30]) /
  (sum(cm_c49_tables[4, 21:30]) + sum(cm_c49_tables[3, 21:30])), 6)
k7c49_truescore <- round((2 * k7c49_tpr * k7c49_tnr) /
  (k7c49_tpr + k7c49_tnr), 6)

k9c49_tpr <- round(sum(cm_c49_tables[1, 31:40]) /
  (sum(cm_c49_tables[1, 31:40]) + sum(cm_c49_tables[2, 31:40])), 6)
k9c49_tnr <- round(sum(cm_c49_tables[4, 31:40]) /
  (sum(cm_c49_tables[4, 31:40]) + sum(cm_c49_tables[3, 31:40])), 6)
k9c49_truescore <- round((2 * k9c49_tpr * k9c49_tnr) /
  (k9c49_tpr + k9c49_tnr), 6)

k11c49_tpr <- round(sum(cm_c49_tables[1, 41:50]) /
  (sum(cm_c49_tables[1, 41:50]) + sum(cm_c49_tables[2, 41:50])), 6)
k11c49_tnr <- round(sum(cm_c49_tables[4, 41:50]) /
  (sum(cm_c49_tables[4, 41:50]) + sum(cm_c49_tables[3, 41:50])), 6)
k11c49_truescore <- round((2 * k11c49_tpr * k11c49_tnr) /
  (k11c49_tpr + k11c49_tnr), 6)

k13c49_tpr <- round(sum(cm_c49_tables[1, 51:60]) /
  (sum(cm_c49_tables[1, 51:60]) + sum(cm_c49_tables[2, 51:60])), 6)
k13c49_tnr <- round(sum(cm_c49_tables[4, 51:60]) /
  (sum(cm_c49_tables[4, 51:60]) + sum(cm_c49_tables[3, 51:60])), 6)
k13c49_truescore <- round((2 * k13c49_tpr * k13c49_tnr) /
  (k13c49_tpr + k13c49_tnr), 6)

k15c49_tpr <- round(sum(cm_c49_tables[1, 61:70]) /
  (sum(cm_c49_tables[1, 61:70]) + sum(cm_c49_tables[2, 61:70])), 6)
k15c49_tnr <- round(sum(cm_c49_tables[4, 61:70]) /
  (sum(cm_c49_tables[4, 61:70]) + sum(cm_c49_tables[3, 61:70])), 6)
k15c49_truescore <- round((2 * k15c49_tpr * k15c49_tnr) /
  (k15c49_tpr + k15c49_tnr), 6)

k17c49_tpr <- round(sum(cm_c49_tables[1, 71:80]) /
  (sum(cm_c49_tables[1, 71:80]) + sum(cm_c49_tables[2, 71:80])), 6)
k17c49_tnr <- round(sum(cm_c49_tables[4, 71:80]) /
  (sum(cm_c49_tables[4, 71:80]) + sum(cm_c49_tables[3, 71:80])), 6)
k17c49_truescore <- round((2 * k17c49_tpr * k17c49_tnr) /
  (k17c49_tpr + k17c49_tnr), 6)

k19c49_tpr <- round(sum(cm_c49_tables[1, 81:90]) /

```

```

(sum(cm_c49_tables[1, 81:90]) + sum(cm_c49_tables[2, 81:90])), 6)
k19c49_tnr <- round(sum(cm_c49_tables[4, 81:90]) /
  (sum(cm_c49_tables[4, 81:90]) + sum(cm_c49_tables[3, 81:90])), 6)
k19c49_truescore <- round((2 * k19c49_tpr * k19c49_tnr) /
  (k19c49_tpr + k19c49_tnr), 6)

k21c49_tpr <- round(sum(cm_c49_tables[1, 91:100]) /
  (sum(cm_c49_tables[1, 91:100]) + sum(cm_c49_tables[2, 91:100])), 6)
k21c49_tnr <- round(sum(cm_c49_tables[4, 91:100]) /
  (sum(cm_c49_tables[4, 91:100]) + sum(cm_c49_tables[3, 91:100])), 6)
k21c49_truescore <- round((2 * k21c49_tpr * k21c49_tnr) /
  (k21c49_tpr + k21c49_tnr), 6)

k23c49_tpr <- round(sum(cm_c49_tables[1, 101:110]) /
  (sum(cm_c49_tables[1, 101:110]) + sum(cm_c49_tables[2, 101:110])), 6)
k23c49_tnr <- round(sum(cm_c49_tables[4, 101:110]) /
  (sum(cm_c49_tables[4, 101:110]) + sum(cm_c49_tables[3, 101:110])), 6)
k23c49_truescore <- round((2 * k23c49_tpr * k23c49_tnr) /
  (k23c49_tpr + k23c49_tnr), 6)

k25c49_tpr <- round(sum(cm_c49_tables[1, 111:120]) /
  (sum(cm_c49_tables[1, 111:120]) + sum(cm_c49_tables[2, 111:120])), 6)
k25c49_tnr <- round(sum(cm_c49_tables[4, 111:120]) /
  (sum(cm_c49_tables[4, 111:120]) + sum(cm_c49_tables[3, 111:120])), 6)
k25c49_truescore <- round((2 * k25c49_tpr * k25c49_tnr) /
  (k25c49_tpr + k25c49_tnr), 6)

k27c49_tpr <- round(sum(cm_c49_tables[1, 121:130]) /
  (sum(cm_c49_tables[1, 121:130]) + sum(cm_c49_tables[2, 121:130])), 6)
k27c49_tnr <- round(sum(cm_c49_tables[4, 121:130]) /
  (sum(cm_c49_tables[4, 121:130]) + sum(cm_c49_tables[3, 121:130])), 6)
k27c49_truescore <- round((2 * k27c49_tpr * k27c49_tnr) /
  (k27c49_tpr + k27c49_tnr), 6)

k29c49_tpr <- round(sum(cm_c49_tables[1, 131:140]) /
  (sum(cm_c49_tables[1, 131:140]) + sum(cm_c49_tables[2, 131:140])), 6)
k29c49_tnr <- round(sum(cm_c49_tables[4, 131:140]) /
  (sum(cm_c49_tables[4, 131:140]) + sum(cm_c49_tables[3, 131:140])), 6)
k29c49_truescore <- round((2 * k29c49_tpr * k29c49_tnr) /
  (k29c49_tpr + k29c49_tnr), 6)

k31c49_tpr <- round(sum(cm_c49_tables[1, 141:150]) /
  (sum(cm_c49_tables[1, 141:150]) + sum(cm_c49_tables[2, 141:150])), 6)
k31c49_tnr <- round(sum(cm_c49_tables[4, 141:150]) /
  (sum(cm_c49_tables[4, 141:150]) + sum(cm_c49_tables[3, 141:150])), 6)
k31c49_truescore <- round((2 * k31c49_tpr * k31c49_tnr) /
  (k31c49_tpr + k31c49_tnr), 6)

k33c49_tpr <- round(sum(cm_c49_tables[1, 151:160]) /
  (sum(cm_c49_tables[1, 151:160]) + sum(cm_c49_tables[2, 151:160])), 6)
k33c49_tnr <- round(sum(cm_c49_tables[4, 151:160]) /
  (sum(cm_c49_tables[4, 151:160]) + sum(cm_c49_tables[3, 151:160])), 6)
k33c49_truescore <- round((2 * k33c49_tpr * k33c49_tnr) /

```

```

(k33c49_tpr + k33c49_tnr), 6)

k35c49_tpr <- round(sum(cm_c49_tables[1, 161:170]) /
  (sum(cm_c49_tables[1, 161:170]) + sum(cm_c49_tables[2, 161:170])), 6)
k35c49_tnr <- round(sum(cm_c49_tables[4, 161:170]) /
  (sum(cm_c49_tables[4, 161:170]) + sum(cm_c49_tables[3, 161:170])), 6)
k35c49_truescore <- round((2 * k35c49_tpr * k35c49_tnr) /
  (k35c49_tpr + k35c49_tnr), 6)

k37c49_tpr <- round(sum(cm_c49_tables[1, 171:180]) /
  (sum(cm_c49_tables[1, 171:180]) + sum(cm_c49_tables[2, 171:180])), 6)
k37c49_tnr <- round(sum(cm_c49_tables[4, 171:180]) /
  (sum(cm_c49_tables[4, 171:180]) + sum(cm_c49_tables[3, 171:180])), 6)
k37c49_truescore <- round((2 * k37c49_tpr * k37c49_tnr) /
  (k37c49_tpr + k37c49_tnr), 6)

k39c49_tpr <- round(sum(cm_c49_tables[1, 181:190]) /
  (sum(cm_c49_tables[1, 181:190]) + sum(cm_c49_tables[2, 181:190])), 6)
k39c49_tnr <- round(sum(cm_c49_tables[4, 181:190]) /
  (sum(cm_c49_tables[4, 181:190]) + sum(cm_c49_tables[3, 181:190])), 6)
k39c49_truescore <- round((2 * k39c49_tpr * k39c49_tnr) /
  (k39c49_tpr + k39c49_tnr), 6)

# Compile the 0.49 cutoff results in a table, and identify the value
# of k that maximizes the truescore.

c49_results <- tibble(k = seq(3, 39, 2),
  Cut = 0.49,
  TPR = c(k3c49_tpr, k5c49_tpr, k7c49_tpr, k9c49_tpr, k11c49_tpr,
    k13c49_tpr, k15c49_tpr, k17c49_tpr, k19c49_tpr, k21c49_tpr,
    k23c49_tpr, k25c49_tpr, k27c49_tpr, k29c49_tpr, k31c49_tpr,
    k33c49_tpr, k35c49_tpr, k37c49_tpr, k39c49_tpr),
  TNR = c(k3c49_tnr, k5c49_tnr, k7c49_tnr, k9c49_tnr, k11c49_tnr,
    k13c49_tnr, k15c49_tnr, k17c49_tnr, k19c49_tnr, k21c49_tnr,
    k23c49_tnr, k25c49_tnr, k27c49_tnr, k29c49_tnr, k31c49_tnr,
    k33c49_tnr, k35c49_tnr, k37c49_tnr, k39c49_tnr),
  Truescore = c(k3c49_truescore, k5c49_truescore, k7c49_truescore,
    k9c49_truescore, k11c49_truescore, k13c49_truescore,
    k15c49_truescore, k17c49_truescore, k19c49_truescore,
    k21c49_truescore, k23c49_truescore, k25c49_truescore,
    k27c49_truescore, k29c49_truescore, k31c49_truescore,
    k33c49_truescore, k35c49_truescore, k37c49_truescore,
    k39c49_truescore))

knitr::kable(c49_results[1:19, ], caption = "c49_results")

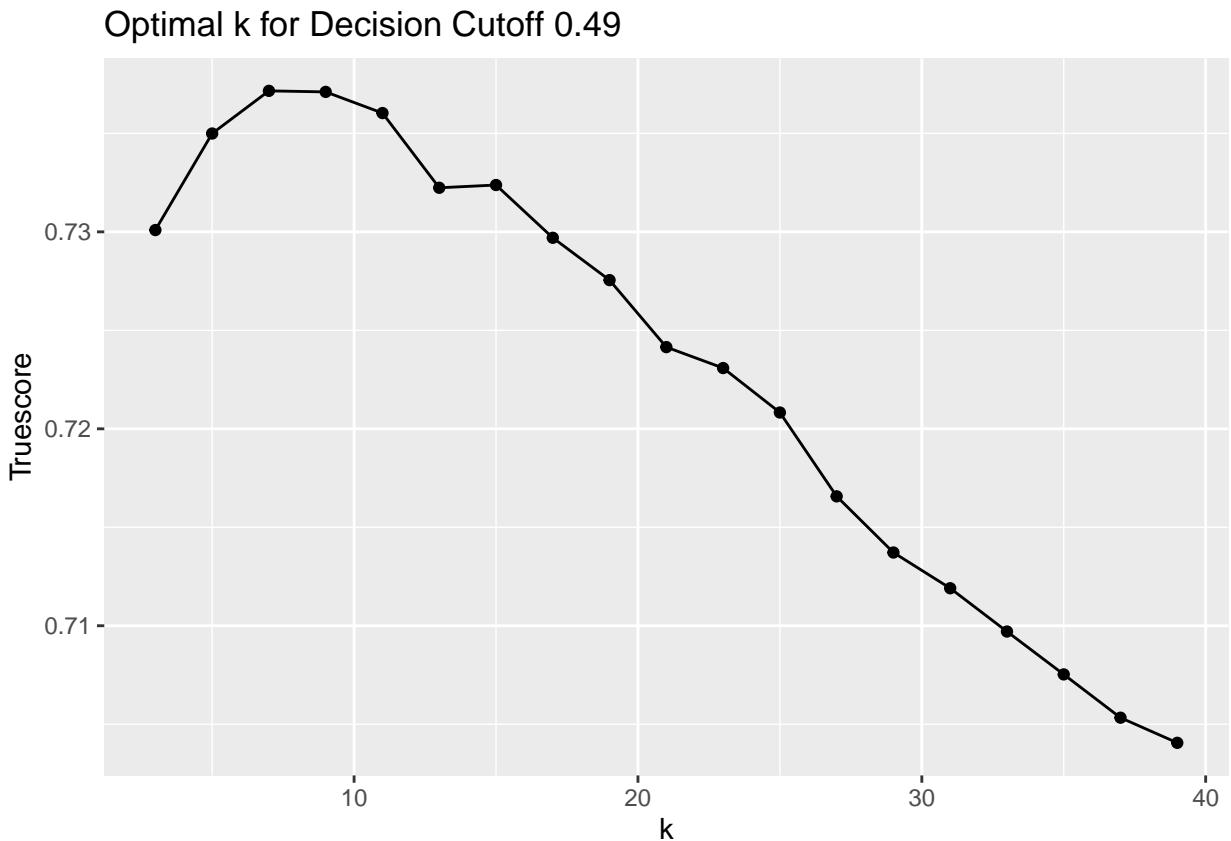
```

Table 32: c49_results

k	Cut	TPR	TNR	Truescore
3	0.49	0.616215	0.895600	0.730092
5	0.49	0.619076	0.904321	0.734993
7	0.49	0.620131	0.908631	0.737159
9	0.49	0.618474	0.912050	0.737106

k	Cut	TPR	TNR	Truescore
11	0.49	0.616165	0.913801	0.736032
13	0.49	0.609990	0.915767	0.732238
15	0.49	0.609337	0.917682	0.732378
17	0.49	0.605371	0.918294	0.729699
19	0.49	0.601908	0.919483	0.727550
21	0.49	0.597189	0.919664	0.724148
23	0.49	0.595281	0.920771	0.723085
25	0.49	0.591968	0.921399	0.720828
27	0.49	0.586094	0.921778	0.716571
29	0.49	0.581978	0.922555	0.713719
31	0.49	0.579518	0.922687	0.711905
33	0.49	0.576456	0.923083	0.709707
35	0.49	0.573544	0.923199	0.707530
37	0.49	0.570181	0.924454	0.705331
39	0.49	0.568323	0.924966	0.704056

```
ggplot(c49_results, aes(k, Truescore)) + geom_point() + geom_line() +
  labs(title = "Optimal k for Decision Cutoff 0.49")
```



We next combined the truescores for every combination of k and cutoff, computed the Distance to (0, 1) for every combination of k and cutoff, and identified the values of k and cutoff that resulted in the highest truescore and the shortest distance to (0, 1).

```

# Combine the results for every combination of k and cutoff that was tested.

br1 <- bind_rows(c25_results, c26_results, c27_results, c28_results, c29_results)
br2 <- bind_rows(c30_results, c31_results, c32_results, c33_results, c34_results)
br3 <- bind_rows(c35_results, c36_results, c37_results, c38_results, c39_results)
br4 <- bind_rows(c40_results, c41_results, c42_results, c43_results, c44_results)
br5 <- bind_rows(c45_results, c46_results, c47_results, c48_results, c49_results)
results_optkcut <- bind_rows(br1, br2, br3, br4, br5)

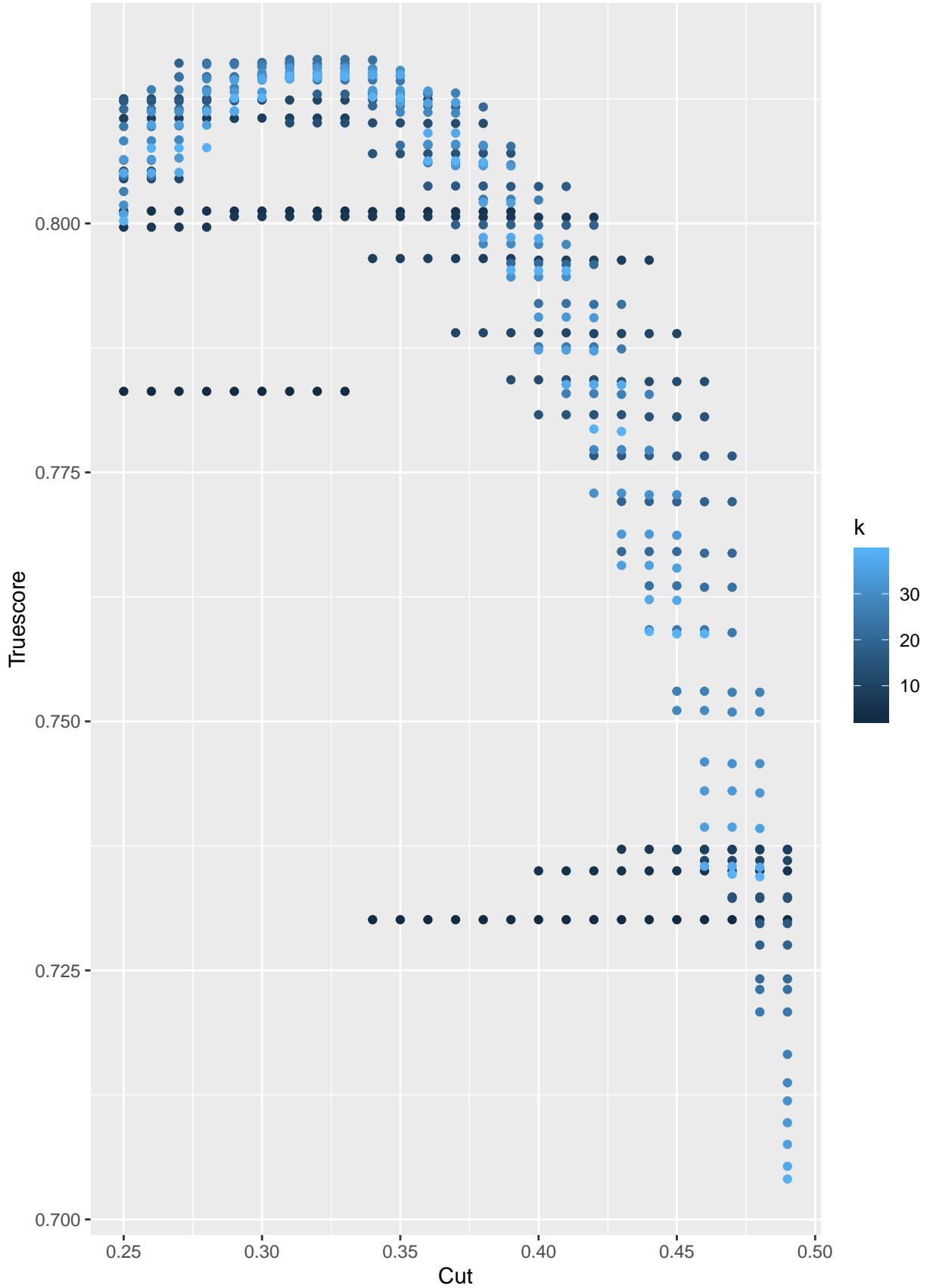
# Compute the Minimum Distance to (0, 1) for each combination of k and cutoff
# that was tested.

results_optkcut <- results_optkcut %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

results_optkcut_byTruescore <- results_optkcut %>% arrange(desc(Truescore))
results_optkcut_byK <- results_optkcut %>% arrange(k)
results_optkcut_byCut <- results_optkcut %>% arrange(Cut)

results_optkcut %>% ggplot(aes(x = Cut, y = Truescore, color = k)) + geom_point()

```



```

# Identify the optimal combination of values for k and the cutoff based on each of our
# assessment methods, Truescore and Minimum Distance to (0, 1).

max(results_optkcut$Truescore)

## [1] 0.81646

(knn_opt_k <- results_optkcut$k[which.max(results_optkcut$Truescore)])
```

```

## [1] 23

(knn_opt_cut <- results_optkcut$Cut[which.max(results_optkcut$Truescore)])
```

```

## [1] 0.31

min(results_optkcut$Distance)
```

```

## [1] 0.259584

results_optkcut$k[which.min(results_optkcut$Distance)]
```

```

## [1] 23

results_optkcut$Cut[which.min(results_optkcut$Distance)]
```

```

## [1] 0.31

```

We finally had our optimal combination of k and cutoff. A neighborhood size of 23 and a probability cutoff of 0.31 resulted in both the highest truescore, 0.81646, and the shortest distance to $(0, 1)$, 0.259584, based on 10-fold cross-validation. But how would we fare with the $test_set$? To find out, we first needed to fit a model with our new optimal k , and then use the model to predict outcomes based on the predictors in the $test_set$.

```

# Fit an optimized knn model (k = 23) to the entire predictor matrix and outcome vector.

knn_fit_k23 <- knn3(x, y, k = 23)

# Apply our latest knn model (knn_fit_k23) and optimal cutoff of 0.31 to predict
# outcomes based on the predictors in test_set2.

knn_y_hat_prob <- predict(knn_fit_k23, z, type = "prob")
knn_y_hat_prob_tbl <- as_tibble(knn_y_hat_prob)
knn_y_hat_prob_tbl <- knn_y_hat_prob_tbl %>%
  mutate(preds = ifelse(knn_y_hat_prob_tbl$single > 0.31, "single", "field_out"))
knn_y_hat_prob_tbl$preds <- factor(knn_y_hat_prob_tbl$preds,
                                     levels = c("single", "field_out"))

# Assess the predictive ability of our latest knn model (knn_fit_k23).

confusionMatrix(data = knn_y_hat_prob_tbl$preds, reference = test_set2$events)
```

```

## Confusion Matrix and Statistics
##
##             Reference
## Prediction  single field_out
##   single      4109      2793
##   field_out    872     12344
##
##                   Accuracy : 0.8178
##                   95% CI : (0.8124, 0.8231)
##   No Information Rate : 0.7524
##   P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5671
##
## McNemar's Test P-Value : < 2.2e-16
##
##                   Sensitivity : 0.8249
##                   Specificity : 0.8155
##   Pos Pred Value : 0.5953
##   Neg Pred Value : 0.9340
##   Prevalence : 0.2476
##   Detection Rate : 0.2042
##   Detection Prevalence : 0.3431
##   Balanced Accuracy : 0.8202
##
##   'Positive' Class : single
## 

(knn_k23c31_tpr <- round(sensitivity(knn_y_hat_prob_tbl$preds,
                                         reference = factor(test_set2$events)), 6))

## [1] 0.824935

(knn_k23c31_tnr <- round(specificity(knn_y_hat_prob_tbl$preds,
                                         reference = factor(test_set2$events)), 6))

## [1] 0.815485

(knn_k23c31_fpr <- round(1 - knn_k23c31_tnr, 6))

## [1] 0.184515

(knn_k23c31_truescore <- round((2 * knn_k23c31_tpr * knn_k23c31_tnr) /
                                         (knn_k23c31_tpr + knn_k23c31_tnr) , 6))

## [1] 0.820183

(knn_k23c31_distance <- round(sqrt((1 - knn_k23c31_tpr)^2 + (knn_k23c31_fpr)^2), 6))

## [1] 0.254349

```

```

assessment_results <- tibble(Model = c("Baseline", "Log Regression", "Log Regression",
                                      "KNN_k5", "KNN_k7", "KNN_k23c31"),
                               `Cutoff Method` = c(NA, "Truescore", "Distance",
                                                  "default(.50)", "default(.50)", "both"),
                               `Truescore` = c(bl_truescore, max_truescore,
                                              min_distance_truescore, knn_k5_truescore,
                                              knn_k7_truescore, knn_k23c31_truescore),
                               TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                       knn_k5_tpr, knn_k7_tpr, knn_k23c31_tpr),
                               TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                       knn_k5_tnr, knn_k7_tnr, knn_k23c31_tnr),
                               Distance = c(NA, max_truescore_distance, min_distance,
                                            knn_k5_distance, knn_k7_distance,
                                            knn_k23c31_distance),
                               FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                       knn_k5_fpr, knn_k7_fpr, knn_k23c31_fpr),
                               `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                min_distance_cutoff, "default",
                                                "default", knn_opt_cut))
knitr::kable(assessment_results[1:6, ], caption = "Assessment Results")

```

Table 33: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log Regression	Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN_k5	default(.50)	0.746224	0.634812	0.905067	0.377326	0.094933	default
KNN_k7	default(.50)	0.749218	0.635615	0.912268	0.374798	0.087732	default
KNN_k23c31	both	0.820183	0.824935	0.815485	0.254349	0.184515	0.31

Our KNN_k23c31 Model was now our best performer. It actually achieved a higher truescore (and shorter distance) on the *test_set* than on the *train_set*, a sign that it was not overfit and would perform well in the real world. With a truescore of 0.820183 and distance of 0.254349, it clearly bested the KNN_k7c40 Model. It also achieved the most balance between sensitivity (TPR), 0.824935, and specificity (TNR), 0.815485.

The k-Nearest-Neighbors algorithm had shown itself to be well suited to our challenge of distinguishing singles from outs. So we weren't ready to get off the KNN train quite yet.

Build and Assess a Model with the Weighted K-Nearest Neighbors Algorithm (kknn)

The weighted k-nearest neighbors algorithm in the kknn package takes into account the various distances between the nearest neighbors and the observation we're trying to classify.¹⁴ It does so by according different weights to each of the k nearest neighbors based on their distances. That is, each nearest neighbor no longer has simply one vote. The nearest neighbor who is closer will, in effect, have more votes than a more distant nearest neighbor. The kknn function allows you to choose any one of a number of different schemes or kernels for transforming distances into similarity scores that can be used as weights. The specific kernel chosen has not been found to be crucial.

¹⁴Hechenbichler, K., and Schliep, K.P. (2004). Weighted k-nearest-neighbor techniques and ordinal classification. Discussion paper399, SFB386, Ludwig-Maximilians University, Munich. <http://www.stat.uni-muenchen.de/sfb386/papers/dsp/paper399.ps>.

By applying the kknn method within the caret::train() function, we were able to determine the optimal values of k, kernel, and the decision cutoff. We tested 15 values of k ranging from 11 to 39. We also tested three kernels (triangular, gaussian, and optimal) for weighting each of the k nearest neighbors. Ten-fold cross-validation was used to generate the class probabilities. Generating probabilities would allow us to test 15 different decision cutoffs along with the parameters k and kernel. The optimal combination of k, kernel, and cutoff would be that which resulted in the highest truescore and/or the shortest distance to (0, 1). Warning: the train function in the following code chunk took many hours to complete, due, in part, to the 10-fold cross-validation as well as the large number of combinations (675) that had to be tested.

```
# Use caret::train() to develop a weighted knn algorithm with an optimal
# combination of values for k, kernel, and the decision threshold.
# The optimal combination of k, kernel, and the decision threshold should
# maximize the Truescore of the model.

train_set2 <- dplyr::select(train_set, -(hit_distance_sc))
test_set2 <- dplyr::select(test_set, -(hit_distance_sc))

x <- as.matrix(train_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                             "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
y <- train_set2$events

# Use the kknn method with the train function and cross-validation to
# compute the probabilities of "single" for a range of k's and kernels.

set.seed(1)
fitControl <- trainControl(method = "cv", number = 10, p = 0.8, returnData = TRUE,
                           returnResamp = "all", savePredictions = "all",
                           summaryFunction = twoClassSummary, classProbs = TRUE)
tuneGrid <- expand.grid(kmax = seq(11, 39, 2),
                        distance = 2,
                        kernel = c("triangular", "gaussian", "optimal"))

kknn_train <- train(x, y, method = "kknn",
                     tuneGrid = tuneGrid, trControl = fitControl)
```

We then applied 15 different cutoffs, ranging from 0.25 to 0.39, to generate predictions for every combination of k and kernel. The resulting tibble was further wrangled to facilitate analysis.

```
# Generate predictions by applying a range of thresholds (cutoffs) to the
# probabilities for each combination of k and kernel.

kknn_train_pred_tib <- as_tibble(kknn_train$pred)
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred25 = ifelse(kknn_train_pred_tib$single > 0.25, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred26 = ifelse(kknn_train_pred_tib$single > 0.26, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred27 = ifelse(kknn_train_pred_tib$single > 0.27, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred28 = ifelse(kknn_train_pred_tib$single > 0.28, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred29 = ifelse(kknn_train_pred_tib$single > 0.29, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred30 = ifelse(kknn_train_pred_tib$single > 0.30, "single", "field_out"))
```

```

kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred31 = ifelse(kknn_train_pred_tib$single > 0.31, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred32 = ifelse(kknn_train_pred_tib$single > 0.32, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred33 = ifelse(kknn_train_pred_tib$single > 0.33, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred34 = ifelse(kknn_train_pred_tib$single > 0.34, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred35 = ifelse(kknn_train_pred_tib$single > 0.35, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred36 = ifelse(kknn_train_pred_tib$single > 0.36, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred37 = ifelse(kknn_train_pred_tib$single > 0.37, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred38 = ifelse(kknn_train_pred_tib$single > 0.38, "single", "field_out"))
kknn_train_pred_tib <- kknn_train_pred_tib %>%
  mutate(pred39 = ifelse(kknn_train_pred_tib$single > 0.39, "single", "field_out"))

# Convert all character columns to factors using dplyr package
# (https://gist.github.com/ramhiser/character2factor.r).

kknn_train_pred_tib_f <- kknn_train_pred_tib %>%
  mutate_if(sapply(kknn_train_pred_tib, is.character), as.factor)

rm(kknn_train_pred_tib)
gc(reset = TRUE)

sapply(kknn_train_pred_tib_f, class)

# Make sure all of the factor outcomes have the same levels in the same order.

kknn_train_pred_tib_f$pred25 <- factor(kknn_train_pred_tib_f$pred25,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred26 <- factor(kknn_train_pred_tib_f$pred26,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred27 <- factor(kknn_train_pred_tib_f$pred27,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred28 <- factor(kknn_train_pred_tib_f$pred28,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred29 <- factor(kknn_train_pred_tib_f$pred29,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred30 <- factor(kknn_train_pred_tib_f$pred30,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred31 <- factor(kknn_train_pred_tib_f$pred31,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred32 <- factor(kknn_train_pred_tib_f$pred32,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred33 <- factor(kknn_train_pred_tib_f$pred33,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred34 <- factor(kknn_train_pred_tib_f$pred34,
                                         levels = c("single", "field_out"))

```

```

kknn_train_pred_tib_f$pred35 <- factor(kknn_train_pred_tib_f$pred35,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred36 <- factor(kknn_train_pred_tib_f$pred36,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred37 <- factor(kknn_train_pred_tib_f$pred37,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred38 <- factor(kknn_train_pred_tib_f$pred38,
                                         levels = c("single", "field_out"))
kknn_train_pred_tib_f$pred39 <- factor(kknn_train_pred_tib_f$pred39,
                                         levels = c("single", "field_out"))
sapply(kknn_train_pred_tib_f, levels)

# Convert kknn_train_pred_tib_f$Resample into a numeric type.

kknn_train_pred_tib_f$Resample <- as.numeric(kknn_train_pred_tib_f$Resample)

# Add a column for a numeric version of kknn_train_pred_tib_f$wt_kernel.

kknn_train_pred_tib_f <- kknn_train_pred_tib_f %>%
  mutate(wt_kernel_n = as.numeric(kknn_train_pred_tib_f$kernel))

# Rename "kernel" column to "wt_kernel".

kknn_train_pred_tib_f <- rename(kknn_train_pred_tib_f, wt_kernel = kernel)

# Reorder columns in kknn_train_pred_tib_f.

kknn_train_pred_tib_f <- dplyr::select(kknn_train_pred_tib_f, pred, obs,
                                         single, field_out, rowIndex, distance,
                                         kmax, wt_kernel, wt_kernel_n,
                                         everything())

```

Next, we created a separate tibble for each combination of k, kernel, and fold.

```

# Form separate tibbles for each combination of k (kmax), kernel (wt_kernel_n),
# and fold (Resample).

k11w1f1  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 1)
k11w1f2  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 2)
k11w1f3  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 3)
k11w1f4  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 4)
k11w1f5  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 5)
k11w1f6  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 6)
k11w1f7  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 7)
k11w1f8  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 8)
k11w1f9  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1, Resample == 9)
k11w1f10 <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 1,
                                                Resample == 10)

k11w2f1  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 2, Resample == 1)
k11w2f2  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 2, Resample == 2)
k11w2f3  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 2, Resample == 3)
k11w2f4  <- kknn_train_pred_tib_f %>% filter(kmax == 11, wt_kernel_n == 2, Resample == 4)

```



```

k39w3f8 <- kknn_train_pred_tib_f %>% filter(kmax == 39, wt_kernel_n == 3, Resample == 8)
k39w3f9 <- kknn_train_pred_tib_f %>% filter(kmax == 39, wt_kernel_n == 3, Resample == 9)
k39w3f10 <- kknn_train_pred_tib_f %>% filter(kmax == 39, wt_kernel_n == 3,
Resample == 10)

rm(kknn_train_pred_tib_f)
gc(reset = TRUE)

# Form a list and vector of these tibbles.

kwf_dfs_1 <- list(k11w1f1, k11w1f2, k11w1f3, k11w1f4, k11w1f5,
k11w1f6, k11w1f7, k11w1f8, k11w1f9, k11w1f10,
k11w2f1, k11w2f2, k11w2f3, k11w2f4, k11w2f5,
k11w2f6, k11w2f7, k11w2f8, k11w2f9, k11w2f10,
k11w3f1, k11w3f2, k11w3f3, k11w3f4, k11w3f5,
k11w3f6, k11w3f7, k11w3f8, k11w3f9, k11w3f10,
k13w1f1, k13w1f2, k13w1f3, k13w1f4, k13w1f5,
k13w1f6, k13w1f7, k13w1f8, k13w1f9, k13w1f10,
k13w2f1, k13w2f2, k13w2f3, k13w2f4, k13w2f5,
k13w2f6, k13w2f7, k13w2f8, k13w2f9, k13w2f10,
k13w3f1, k13w3f2, k13w3f3, k13w3f4, k13w3f5,
k13w3f6, k13w3f7, k13w3f8, k13w3f9, k13w3f10,
k15w1f1, k15w1f2, k15w1f3, k15w1f4, k15w1f5,
k15w1f6, k15w1f7, k15w1f8, k15w1f9, k15w1f10,
k15w2f1, k15w2f2, k15w2f3, k15w2f4, k15w2f5,
k15w2f6, k15w2f7, k15w2f8, k15w2f9, k15w2f10,
k15w3f1, k15w3f2, k15w3f3, k15w3f4, k15w3f5,
k15w3f6, k15w3f7, k15w3f8, k15w3f9, k15w3f10,
k17w1f1, k17w1f2, k17w1f3, k17w1f4, k17w1f5,
k17w1f6, k17w1f7, k17w1f8, k17w1f9, k17w1f10,
k17w2f1, k17w2f2, k17w2f3, k17w2f4, k17w2f5,
k17w2f6, k17w2f7, k17w2f8, k17w2f9, k17w2f10,
k17w3f1, k17w3f2, k17w3f3, k17w3f4, k17w3f5,
k17w3f6, k17w3f7, k17w3f8, k17w3f9, k17w3f10,
k19w1f1, k19w1f2, k19w1f3, k19w1f4, k19w1f5,
k19w1f6, k19w1f7, k19w1f8, k19w1f9, k19w1f10,
k19w2f1, k19w2f2, k19w2f3, k19w2f4, k19w2f5,
k19w2f6, k19w2f7, k19w2f8, k19w2f9, k19w2f10,
k19w3f1, k19w3f2, k19w3f3, k19w3f4, k19w3f5,
k19w3f6, k19w3f7, k19w3f8, k19w3f9, k19w3f10,
k21w1f1, k21w1f2, k21w1f3, k21w1f4, k21w1f5,
k21w1f6, k21w1f7, k21w1f8, k21w1f9, k21w1f10,
k21w2f1, k21w2f2, k21w2f3, k21w2f4, k21w2f5,
k21w2f6, k21w2f7, k21w2f8, k21w2f9, k21w2f10,
k21w3f1, k21w3f2, k21w3f3, k21w3f4, k21w3f5,
k21w3f6, k21w3f7, k21w3f8, k21w3f9, k21w3f10,
k23w1f1, k23w1f2, k23w1f3, k23w1f4, k23w1f5,
k23w1f6, k23w1f7, k23w1f8, k23w1f9, k23w1f10,
k23w2f1, k23w2f2, k23w2f3, k23w2f4, k23w2f5,
k23w2f6, k23w2f7, k23w2f8, k23w2f9, k23w2f10,
k23w3f1, k23w3f2, k23w3f3, k23w3f4, k23w3f5,
k23w3f6, k23w3f7, k23w3f8, k23w3f9, k23w3f10,

```

```

k25w1f1, k25w1f2, k25w1f3, k25w1f4, k25w1f5,
k25w1f6, k25w1f7, k25w1f8, k25w1f9, k25w1f10,
k25w2f1, k25w2f2, k25w2f3, k25w2f4, k25w2f5,
k25w2f6, k25w2f7, k25w2f8, k25w2f9, k25w2f10,
k25w3f1, k25w3f2, k25w3f3, k25w3f4, k25w3f5,
k25w3f6, k25w3f7, k25w3f8, k25w3f9, k25w3f10,
k27w1f1, k27w1f2, k27w1f3, k27w1f4, k27w1f5,
k27w1f6, k27w1f7, k27w1f8, k27w1f9, k27w1f10,
k27w2f1, k27w2f2, k27w2f3, k27w2f4, k27w2f5,
k27w2f6, k27w2f7, k27w2f8, k27w2f9, k27w2f10,
k27w3f1, k27w3f2, k27w3f3, k27w3f4, k27w3f5,
k27w3f6, k27w3f7, k27w3f8, k27w3f9, k27w3f10,
k29w1f1, k29w1f2, k29w1f3, k29w1f4, k29w1f5,
k29w1f6, k29w1f7, k29w1f8, k29w1f9, k29w1f10,
k29w2f1, k29w2f2, k29w2f3, k29w2f4, k29w2f5,
k29w2f6, k29w2f7, k29w2f8, k29w2f9, k29w2f10,
k29w3f1, k29w3f2, k29w3f3, k29w3f4, k29w3f5,
k29w3f6, k29w3f7, k29w3f8, k29w3f9, k29w3f10,
k31w1f1, k31w1f2, k31w1f3, k31w1f4, k31w1f5,
k31w1f6, k31w1f7, k31w1f8, k31w1f9, k31w1f10,
k31w2f1, k31w2f2, k31w2f3, k31w2f4, k31w2f5,
k31w2f6, k31w2f7, k31w2f8, k31w2f9, k31w2f10,
k31w3f1, k31w3f2, k31w3f3, k31w3f4, k31w3f5,
k31w3f6, k31w3f7, k31w3f8, k31w3f9, k31w3f10,
k33w1f1, k33w1f2, k33w1f3, k33w1f4, k33w1f5,
k33w1f6, k33w1f7, k33w1f8, k33w1f9, k33w1f10,
k33w2f1, k33w2f2, k33w2f3, k33w2f4, k33w2f5,
k33w2f6, k33w2f7, k33w2f8, k33w2f9, k33w2f10,
k33w3f1, k33w3f2, k33w3f3, k33w3f4, k33w3f5,
k33w3f6, k33w3f7, k33w3f8, k33w3f9, k33w3f10,
k35w1f1, k35w1f2, k35w1f3, k35w1f4, k35w1f5,
k35w1f6, k35w1f7, k35w1f8, k35w1f9, k35w1f10,
k35w2f1, k35w2f2, k35w2f3, k35w2f4, k35w2f5,
k35w2f6, k35w2f7, k35w2f8, k35w2f9, k35w2f10,
k35w3f1, k35w3f2, k35w3f3, k35w3f4, k35w3f5,
k35w3f6, k35w3f7, k35w3f8, k35w3f9, k35w3f10,
k37w1f1, k37w1f2, k37w1f3, k37w1f4, k37w1f5,
k37w1f6, k37w1f7, k37w1f8, k37w1f9, k37w1f10,
k37w2f1, k37w2f2, k37w2f3, k37w2f4, k37w2f5,
k37w2f6, k37w2f7, k37w2f8, k37w2f9, k37w2f10,
k37w3f1, k37w3f2, k37w3f3, k37w3f4, k37w3f5,
k37w3f6, k37w3f7, k37w3f8, k37w3f9, k37w3f10,
k39w1f1, k39w1f2, k39w1f3, k39w1f4, k39w1f5,
k39w1f6, k39w1f7, k39w1f8, k39w1f9, k39w1f10,
k39w2f1, k39w2f2, k39w2f3, k39w2f4, k39w2f5,
k39w2f6, k39w2f7, k39w2f8, k39w2f9, k39w2f10,
k39w3f1, k39w3f2, k39w3f3, k39w3f4, k39w3f5,
k39w3f6, k39w3f7, k39w3f8, k39w3f9, k39w3f10)

kwf_dfs_v <- c("k11w1f1", "k11w1f2", "k11w1f3", "k11w1f4", "k11w1f5",
              "k11w1f6", "k11w1f7", "k11w1f8", "k11w1f9", "k11w1f10",
              "k11w2f1", "k11w2f2", "k11w2f3", "k11w2f4", "k11w2f5",
              "k11w2f6", "k11w2f7", "k11w2f8", "k11w2f9", "k11w2f10",

```

"k11w3f1", "k11w3f2", "k11w3f3", "k11w3f4", "k11w3f5",
"k11w3f6", "k11w3f7", "k11w3f8", "k11w3f9", "k11w3f10",
"k13w1f1", "k13w1f2", "k13w1f3", "k13w1f4", "k13w1f5",
"k13w1f6", "k13w1f7", "k13w1f8", "k13w1f9", "k13w1f10",
"k13w2f1", "k13w2f2", "k13w2f3", "k13w2f4", "k13w2f5",
"k13w2f6", "k13w2f7", "k13w2f8", "k13w2f9", "k13w2f10",
"k13w3f1", "k13w3f2", "k13w3f3", "k13w3f4", "k13w3f5",
"k13w3f6", "k13w3f7", "k13w3f8", "k13w3f9", "k13w3f10",
"k15w1f1", "k15w1f2", "k15w1f3", "k15w1f4", "k15w1f5",
"k15w1f6", "k15w1f7", "k15w1f8", "k15w1f9", "k15w1f10",
"k15w2f1", "k15w2f2", "k15w2f3", "k15w2f4", "k15w2f5",
"k15w2f6", "k15w2f7", "k15w2f8", "k15w2f9", "k15w2f10",
"k15w3f1", "k15w3f2", "k15w3f3", "k15w3f4", "k15w3f5",
"k15w3f6", "k15w3f7", "k15w3f8", "k15w3f9", "k15w3f10",
"k17w1f1", "k17w1f2", "k17w1f3", "k17w1f4", "k17w1f5",
"k17w1f6", "k17w1f7", "k17w1f8", "k17w1f9", "k17w1f10",
"k17w2f1", "k17w2f2", "k17w2f3", "k17w2f4", "k17w2f5",
"k17w2f6", "k17w2f7", "k17w2f8", "k17w2f9", "k17w2f10",
"k17w3f1", "k17w3f2", "k17w3f3", "k17w3f4", "k17w3f5",
"k17w3f6", "k17w3f7", "k17w3f8", "k17w3f9", "k17w3f10",
"k19w1f1", "k19w1f2", "k19w1f3", "k19w1f4", "k19w1f5",
"k19w1f6", "k19w1f7", "k19w1f8", "k19w1f9", "k19w1f10",
"k19w2f1", "k19w2f2", "k19w2f3", "k19w2f4", "k19w2f5",
"k19w2f6", "k19w2f7", "k19w2f8", "k19w2f9", "k19w2f10",
"k19w3f1", "k19w3f2", "k19w3f3", "k19w3f4", "k19w3f5",
"k19w3f6", "k19w3f7", "k19w3f8", "k19w3f9", "k19w3f10",
"k21w1f1", "k21w1f2", "k21w1f3", "k21w1f4", "k21w1f5",
"k21w1f6", "k21w1f7", "k21w1f8", "k21w1f9", "k21w1f10",
"k21w2f1", "k21w2f2", "k21w2f3", "k21w2f4", "k21w2f5",
"k21w2f6", "k21w2f7", "k21w2f8", "k21w2f9", "k21w2f10",
"k21w3f1", "k21w3f2", "k21w3f3", "k21w3f4", "k21w3f5",
"k21w3f6", "k21w3f7", "k21w3f8", "k21w3f9", "k21w3f10",
"k23w1f1", "k23w1f2", "k23w1f3", "k23w1f4", "k23w1f5",
"k23w1f6", "k23w1f7", "k23w1f8", "k23w1f9", "k23w1f10",
"k23w2f1", "k23w2f2", "k23w2f3", "k23w2f4", "k23w2f5",
"k23w2f6", "k23w2f7", "k23w2f8", "k23w2f9", "k23w2f10",
"k23w3f1", "k23w3f2", "k23w3f3", "k23w3f4", "k23w3f5",
"k23w3f6", "k23w3f7", "k23w3f8", "k23w3f9", "k23w3f10",
"k25w1f1", "k25w1f2", "k25w1f3", "k25w1f4", "k25w1f5",
"k25w1f6", "k25w1f7", "k25w1f8", "k25w1f9", "k25w1f10",
"k25w2f1", "k25w2f2", "k25w2f3", "k25w2f4", "k25w2f5",
"k25w2f6", "k25w2f7", "k25w2f8", "k25w2f9", "k25w2f10",
"k25w3f1", "k25w3f2", "k25w3f3", "k25w3f4", "k25w3f5",
"k25w3f6", "k25w3f7", "k25w3f8", "k25w3f9", "k25w3f10",
"k27w1f1", "k27w1f2", "k27w1f3", "k27w1f4", "k27w1f5",
"k27w1f6", "k27w1f7", "k27w1f8", "k27w1f9", "k27w1f10",
"k27w2f1", "k27w2f2", "k27w2f3", "k27w2f4", "k27w2f5",
"k27w2f6", "k27w2f7", "k27w2f8", "k27w2f9", "k27w2f10",
"k27w3f1", "k27w3f2", "k27w3f3", "k27w3f4", "k27w3f5",
"k27w3f6", "k27w3f7", "k27w3f8", "k27w3f9", "k27w3f10",
"k29w1f1", "k29w1f2", "k29w1f3", "k29w1f4", "k29w1f5",
"k29w1f6", "k29w1f7", "k29w1f8", "k29w1f9", "k29w1f10",
"k29w2f1", "k29w2f2", "k29w2f3", "k29w2f4", "k29w2f5",

```

    "k29w2f6", "k29w2f7", "k29w2f8", "k29w2f9", "k29w2f10",
    "k29w3f1", "k29w3f2", "k29w3f3", "k29w3f4", "k29w3f5",
    "k29w3f6", "k29w3f7", "k29w3f8", "k29w3f9", "k29w3f10",
    "k31w1f1", "k31w1f2", "k31w1f3", "k31w1f4", "k31w1f5",
    "k31w1f6", "k31w1f7", "k31w1f8", "k31w1f9", "k31w1f10",
    "k31w2f1", "k31w2f2", "k31w2f3", "k31w2f4", "k31w2f5",
    "k31w2f6", "k31w2f7", "k31w2f8", "k31w2f9", "k31w2f10",
    "k31w3f1", "k31w3f2", "k31w3f3", "k31w3f4", "k31w3f5",
    "k31w3f6", "k31w3f7", "k31w3f8", "k31w3f9", "k31w3f10",
    "k33w1f1", "k33w1f2", "k33w1f3", "k33w1f4", "k33w1f5",
    "k33w1f6", "k33w1f7", "k33w1f8", "k33w1f9", "k33w1f10",
    "k33w2f1", "k33w2f2", "k33w2f3", "k33w2f4", "k33w2f5",
    "k33w2f6", "k33w2f7", "k33w2f8", "k33w2f9", "k33w2f10",
    "k33w3f1", "k33w3f2", "k33w3f3", "k33w3f4", "k33w3f5",
    "k33w3f6", "k33w3f7", "k33w3f8", "k33w3f9", "k33w3f10",
    "k35w1f1", "k35w1f2", "k35w1f3", "k35w1f4", "k35w1f5",
    "k35w1f6", "k35w1f7", "k35w1f8", "k35w1f9", "k35w1f10",
    "k35w2f1", "k35w2f2", "k35w2f3", "k35w2f4", "k35w2f5",
    "k35w2f6", "k35w2f7", "k35w2f8", "k35w2f9", "k35w2f10",
    "k35w3f1", "k35w3f2", "k35w3f3", "k35w3f4", "k35w3f5",
    "k35w3f6", "k35w3f7", "k35w3f8", "k35w3f9", "k35w3f10",
    "k37w1f1", "k37w1f2", "k37w1f3", "k37w1f4", "k37w1f5",
    "k37w1f6", "k37w1f7", "k37w1f8", "k37w1f9", "k37w1f10",
    "k37w2f1", "k37w2f2", "k37w2f3", "k37w2f4", "k37w2f5",
    "k37w2f6", "k37w2f7", "k37w2f8", "k37w2f9", "k37w2f10",
    "k37w3f1", "k37w3f2", "k37w3f3", "k37w3f4", "k37w3f5",
    "k37w3f6", "k37w3f7", "k37w3f8", "k37w3f9", "k37w3f10",
    "k39w1f1", "k39w1f2", "k39w1f3", "k39w1f4", "k39w1f5",
    "k39w1f6", "k39w1f7", "k39w1f8", "k39w1f9", "k39w1f10",
    "k39w2f1", "k39w2f2", "k39w2f3", "k39w2f4", "k39w2f5",
    "k39w2f6", "k39w2f7", "k39w2f8", "k39w2f9", "k39w2f10",
    "k39w3f1", "k39w3f2", "k39w3f3", "k39w3f4", "k39w3f5",
    "k39w3f6", "k39w3f7", "k39w3f8", "k39w3f9", "k39w3f10")

```

For each combination of k, kernel and cutoff, we could now sum the true positives, false negatives, true negatives, and false positives across 10 folds, and compute the sensitivity (true positive rate or TPR), specificity (true negative rate or TNR), truescore, and distance to (0, 1).

```

#####
# 0.25 Cutoff
#####

# For the decision cutoff of 0.25, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c25 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred25, obs))
  confusionMatrix(ss$pred25, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c25) <- kwf_dfs_v

cm_c25_tables <- sapply(cm_c25, "[[", 2)

```

```

cm_c25_tables <- as_tibble(cm_c25_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.25.

k11w1c25_tpr <- round(sum(cm_c25_tables[1, 1:10]) /
  (sum(cm_c25_tables[1, 1:10]) + sum(cm_c25_tables[2, 1:10])), 6)
k11w1c25_tnr <- round(sum(cm_c25_tables[4, 1:10]) /
  (sum(cm_c25_tables[4, 1:10]) + sum(cm_c25_tables[3, 1:10])), 6)
k11w1c25_truescore <- round((2 * k11w1c25_tpr * k11w1c25_tnr) /
  (k11w1c25_tpr + k11w1c25_tnr), 6)

k11w2c25_tpr <- round(sum(cm_c25_tables[1, 11:20]) /
  (sum(cm_c25_tables[1, 11:20]) + sum(cm_c25_tables[2, 11:20])), 6)
k11w2c25_tnr <- round(sum(cm_c25_tables[4, 11:20]) /
  (sum(cm_c25_tables[4, 11:20]) + sum(cm_c25_tables[3, 11:20])), 6)
k11w2c25_truescore <- round((2 * k11w2c25_tpr * k11w2c25_tnr) /
  (k11w2c25_tpr + k11w2c25_tnr), 6)

k11w3c25_tpr <- round(sum(cm_c25_tables[1, 21:30]) /
  (sum(cm_c25_tables[1, 21:30]) + sum(cm_c25_tables[2, 21:30])), 6)
k11w3c25_tnr <- round(sum(cm_c25_tables[4, 21:30]) /
  (sum(cm_c25_tables[4, 21:30]) + sum(cm_c25_tables[3, 21:30])), 6)
k11w3c25_truescore <- round((2 * k11w3c25_tpr * k11w3c25_tnr) /
  (k11w3c25_tpr + k11w3c25_tnr), 6)

k13w1c25_tpr <- round(sum(cm_c25_tables[1, 31:40]) /
  (sum(cm_c25_tables[1, 31:40]) + sum(cm_c25_tables[2, 31:40])), 6)
k13w1c25_tnr <- round(sum(cm_c25_tables[4, 31:40]) /
  (sum(cm_c25_tables[4, 31:40]) + sum(cm_c25_tables[3, 31:40])), 6)
k13w1c25_truescore <- round((2 * k13w1c25_tpr * k13w1c25_tnr) /
  (k13w1c25_tpr + k13w1c25_tnr), 6)

k13w2c25_tpr <- round(sum(cm_c25_tables[1, 41:50]) /
  (sum(cm_c25_tables[1, 41:50]) + sum(cm_c25_tables[2, 41:50])), 6)
k13w2c25_tnr <- round(sum(cm_c25_tables[4, 41:50]) /
  (sum(cm_c25_tables[4, 41:50]) + sum(cm_c25_tables[3, 41:50])), 6)
k13w2c25_truescore <- round((2 * k13w2c25_tpr * k13w2c25_tnr) /
  (k13w2c25_tpr + k13w2c25_tnr), 6)

k13w3c25_tpr <- round(sum(cm_c25_tables[1, 51:60]) /
  (sum(cm_c25_tables[1, 51:60]) + sum(cm_c25_tables[2, 51:60])), 6)
k13w3c25_tnr <- round(sum(cm_c25_tables[4, 51:60]) /
  (sum(cm_c25_tables[4, 51:60]) + sum(cm_c25_tables[3, 51:60])), 6)
k13w3c25_truescore <- round((2 * k13w3c25_tpr * k13w3c25_tnr) /
  (k13w3c25_tpr + k13w3c25_tnr), 6)

k15w1c25_tpr <- round(sum(cm_c25_tables[1, 61:70]) /
  (sum(cm_c25_tables[1, 61:70]) + sum(cm_c25_tables[2, 61:70])), 6)

```

```

k15w1c25_tnr <- round(sum(cm_c25_tables[4, 61:70]) /
  (sum(cm_c25_tables[4, 61:70]) + sum(cm_c25_tables[3, 61:70])), 6)
k15w1c25_truescore <- round((2 * k15w1c25_tpr * k15w1c25_tnr) /
  (k15w1c25_tpr + k15w1c25_tnr), 6)

k15w2c25_tpr <- round(sum(cm_c25_tables[1, 71:80]) /
  (sum(cm_c25_tables[1, 71:80]) + sum(cm_c25_tables[2, 71:80])), 6)
k15w2c25_tnr <- round(sum(cm_c25_tables[4, 71:80]) /
  (sum(cm_c25_tables[4, 71:80]) + sum(cm_c25_tables[3, 71:80])), 6)
k15w2c25_truescore <- round((2 * k15w2c25_tpr * k15w2c25_tnr) /
  (k15w2c25_tpr + k15w2c25_tnr), 6)

k15w3c25_tpr <- round(sum(cm_c25_tables[1, 81:90]) /
  (sum(cm_c25_tables[1, 81:90]) + sum(cm_c25_tables[2, 81:90])), 6)
k15w3c25_tnr <- round(sum(cm_c25_tables[4, 81:90]) /
  (sum(cm_c25_tables[4, 81:90]) + sum(cm_c25_tables[3, 81:90])), 6)
k15w3c25_truescore <- round((2 * k15w3c25_tpr * k15w3c25_tnr) /
  (k15w3c25_tpr + k15w3c25_tnr), 6)

k17w1c25_tpr <- round(sum(cm_c25_tables[1, 91:100]) /
  (sum(cm_c25_tables[1, 91:100]) + sum(cm_c25_tables[2, 91:100])), 6)
k17w1c25_tnr <- round(sum(cm_c25_tables[4, 91:100]) /
  (sum(cm_c25_tables[4, 91:100]) + sum(cm_c25_tables[3, 91:100])), 6)
k17w1c25_truescore <- round((2 * k17w1c25_tpr * k17w1c25_tnr) /
  (k17w1c25_tpr + k17w1c25_tnr), 6)

k17w2c25_tpr <- round(sum(cm_c25_tables[1, 101:110]) /
  (sum(cm_c25_tables[1, 101:110]) + sum(cm_c25_tables[2, 101:110])), 6)
k17w2c25_tnr <- round(sum(cm_c25_tables[4, 101:110]) /
  (sum(cm_c25_tables[4, 101:110]) + sum(cm_c25_tables[3, 101:110])), 6)
k17w2c25_truescore <- round((2 * k17w2c25_tpr * k17w2c25_tnr) /
  (k17w2c25_tpr + k17w2c25_tnr), 6)

k17w3c25_tpr <- round(sum(cm_c25_tables[1, 111:120]) /
  (sum(cm_c25_tables[1, 111:120]) + sum(cm_c25_tables[2, 111:120])), 6)
k17w3c25_tnr <- round(sum(cm_c25_tables[4, 111:120]) /
  (sum(cm_c25_tables[4, 111:120]) + sum(cm_c25_tables[3, 111:120])), 6)
k17w3c25_truescore <- round((2 * k17w3c25_tpr * k17w3c25_tnr) /
  (k17w3c25_tpr + k17w3c25_tnr), 6)

k19w1c25_tpr <- round(sum(cm_c25_tables[1, 121:130]) /
  (sum(cm_c25_tables[1, 121:130]) + sum(cm_c25_tables[2, 121:130])), 6)
k19w1c25_tnr <- round(sum(cm_c25_tables[4, 121:130]) /
  (sum(cm_c25_tables[4, 121:130]) + sum(cm_c25_tables[3, 121:130])), 6)
k19w1c25_truescore <- round((2 * k19w1c25_tpr * k19w1c25_tnr) /
  (k19w1c25_tpr + k19w1c25_tnr), 6)

k19w2c25_tpr <- round(sum(cm_c25_tables[1, 131:140]) /
  (sum(cm_c25_tables[1, 131:140]) + sum(cm_c25_tables[2, 131:140])), 6)
k19w2c25_tnr <- round(sum(cm_c25_tables[4, 131:140]) /
  (sum(cm_c25_tables[4, 131:140]) + sum(cm_c25_tables[3, 131:140])), 6)

```

```

k19w2c25_truescore <- round((2 * k19w2c25_tpr * k19w2c25_tnr) /
  (k19w2c25_tpr + k19w2c25_tnr), 6)

k19w3c25_tpr <- round(sum(cm_c25_tables[1, 141:150]) /
  (sum(cm_c25_tables[1, 141:150]) + sum(cm_c25_tables[2, 141:150])), 6)
k19w3c25_tnr <- round(sum(cm_c25_tables[4, 141:150]) /
  (sum(cm_c25_tables[4, 141:150]) + sum(cm_c25_tables[3, 141:150])), 6)
k19w3c25_truescore <- round((2 * k19w3c25_tpr * k19w3c25_tnr) /
  (k19w3c25_tpr + k19w3c25_tnr), 6)

k21w1c25_tpr <- round(sum(cm_c25_tables[1, 151:160]) /
  (sum(cm_c25_tables[1, 151:160]) + sum(cm_c25_tables[2, 151:160])), 6)
k21w1c25_tnr <- round(sum(cm_c25_tables[4, 151:160]) /
  (sum(cm_c25_tables[4, 151:160]) + sum(cm_c25_tables[3, 151:160])), 6)
k21w1c25_truescore <- round((2 * k21w1c25_tpr * k21w1c25_tnr) /
  (k21w1c25_tpr + k21w1c25_tnr), 6)

k21w2c25_tpr <- round(sum(cm_c25_tables[1, 161:170]) /
  (sum(cm_c25_tables[1, 161:170]) + sum(cm_c25_tables[2, 161:170])), 6)
k21w2c25_tnr <- round(sum(cm_c25_tables[4, 161:170]) /
  (sum(cm_c25_tables[4, 161:170]) + sum(cm_c25_tables[3, 161:170])), 6)
k21w2c25_truescore <- round((2 * k21w2c25_tpr * k21w2c25_tnr) /
  (k21w2c25_tpr + k21w2c25_tnr), 6)

k21w3c25_tpr <- round(sum(cm_c25_tables[1, 171:180]) /
  (sum(cm_c25_tables[1, 171:180]) + sum(cm_c25_tables[2, 171:180])), 6)
k21w3c25_tnr <- round(sum(cm_c25_tables[4, 171:180]) /
  (sum(cm_c25_tables[4, 171:180]) + sum(cm_c25_tables[3, 171:180])), 6)
k21w3c25_truescore <- round((2 * k21w3c25_tpr * k21w3c25_tnr) /
  (k21w3c25_tpr + k21w3c25_tnr), 6)

k23w1c25_tpr <- round(sum(cm_c25_tables[1, 181:190]) /
  (sum(cm_c25_tables[1, 181:190]) + sum(cm_c25_tables[2, 181:190])), 6)
k23w1c25_tnr <- round(sum(cm_c25_tables[4, 181:190]) /
  (sum(cm_c25_tables[4, 181:190]) + sum(cm_c25_tables[3, 181:190])), 6)
k23w1c25_truescore <- round((2 * k23w1c25_tpr * k23w1c25_tnr) /
  (k23w1c25_tpr + k23w1c25_tnr), 6)

k23w2c25_tpr <- round(sum(cm_c25_tables[1, 191:200]) /
  (sum(cm_c25_tables[1, 191:200]) + sum(cm_c25_tables[2, 191:200])), 6)
k23w2c25_tnr <- round(sum(cm_c25_tables[4, 191:200]) /
  (sum(cm_c25_tables[4, 191:200]) + sum(cm_c25_tables[3, 191:200])), 6)
k23w2c25_truescore <- round((2 * k23w2c25_tpr * k23w2c25_tnr) /
  (k23w2c25_tpr + k23w2c25_tnr), 6)

k23w3c25_tpr <- round(sum(cm_c25_tables[1, 201:210]) /
  (sum(cm_c25_tables[1, 201:210]) + sum(cm_c25_tables[2, 201:210])), 6)
k23w3c25_tnr <- round(sum(cm_c25_tables[4, 201:210]) /
  (sum(cm_c25_tables[4, 201:210]) + sum(cm_c25_tables[3, 201:210])), 6)
k23w3c25_truescore <- round((2 * k23w3c25_tpr * k23w3c25_tnr) /
  (k23w3c25_tpr + k23w3c25_tnr), 6)

```

```

k25w1c25_tpr <- round(sum(cm_c25_tables[1, 211:220]) /
  (sum(cm_c25_tables[1, 211:220]) + sum(cm_c25_tables[2, 211:220])), 6)
k25w1c25_tnr <- round(sum(cm_c25_tables[4, 211:220]) /
  (sum(cm_c25_tables[4, 211:220]) + sum(cm_c25_tables[3, 211:220])), 6)
k25w1c25_truescore <- round((2 * k25w1c25_tpr * k25w1c25_tnr) /
  (k25w1c25_tpr + k25w1c25_tnr), 6)

k25w2c25_tpr <- round(sum(cm_c25_tables[1, 221:230]) /
  (sum(cm_c25_tables[1, 221:230]) + sum(cm_c25_tables[2, 221:230])), 6)
k25w2c25_tnr <- round(sum(cm_c25_tables[4, 221:230]) /
  (sum(cm_c25_tables[4, 221:230]) + sum(cm_c25_tables[3, 221:230])), 6)
k25w2c25_truescore <- round((2 * k25w2c25_tpr * k25w2c25_tnr) /
  (k25w2c25_tpr + k25w2c25_tnr), 6)

k25w3c25_tpr <- round(sum(cm_c25_tables[1, 231:240]) /
  (sum(cm_c25_tables[1, 231:240]) + sum(cm_c25_tables[2, 231:240])), 6)
k25w3c25_tnr <- round(sum(cm_c25_tables[4, 231:240]) /
  (sum(cm_c25_tables[4, 231:240]) + sum(cm_c25_tables[3, 231:240])), 6)
k25w3c25_truescore <- round((2 * k25w3c25_tpr * k25w3c25_tnr) /
  (k25w3c25_tpr + k25w3c25_tnr), 6)

k27w1c25_tpr <- round(sum(cm_c25_tables[1, 241:250]) /
  (sum(cm_c25_tables[1, 241:250]) + sum(cm_c25_tables[2, 241:250])), 6)
k27w1c25_tnr <- round(sum(cm_c25_tables[4, 241:250]) /
  (sum(cm_c25_tables[4, 241:250]) + sum(cm_c25_tables[3, 241:250])), 6)
k27w1c25_truescore <- round((2 * k27w1c25_tpr * k27w1c25_tnr) /
  (k27w1c25_tpr + k27w1c25_tnr), 6)

k27w2c25_tpr <- round(sum(cm_c25_tables[1, 251:260]) /
  (sum(cm_c25_tables[1, 251:260]) + sum(cm_c25_tables[2, 251:260])), 6)
k27w2c25_tnr <- round(sum(cm_c25_tables[4, 251:260]) /
  (sum(cm_c25_tables[4, 251:260]) + sum(cm_c25_tables[3, 251:260])), 6)
k27w2c25_truescore <- round((2 * k27w2c25_tpr * k27w2c25_tnr) /
  (k27w2c25_tpr + k27w2c25_tnr), 6)

k27w3c25_tpr <- round(sum(cm_c25_tables[1, 261:270]) /
  (sum(cm_c25_tables[1, 261:270]) + sum(cm_c25_tables[2, 261:270])), 6)
k27w3c25_tnr <- round(sum(cm_c25_tables[4, 261:270]) /
  (sum(cm_c25_tables[4, 261:270]) + sum(cm_c25_tables[3, 261:270])), 6)
k27w3c25_truescore <- round((2 * k27w3c25_tpr * k27w3c25_tnr) /
  (k27w3c25_tpr + k27w3c25_tnr), 6)

k29w1c25_tpr <- round(sum(cm_c25_tables[1, 271:280]) /
  (sum(cm_c25_tables[1, 271:280]) + sum(cm_c25_tables[2, 271:280])), 6)
k29w1c25_tnr <- round(sum(cm_c25_tables[4, 271:280]) /
  (sum(cm_c25_tables[4, 271:280]) + sum(cm_c25_tables[3, 271:280])), 6)
k29w1c25_truescore <- round((2 * k29w1c25_tpr * k29w1c25_tnr) /
  (k29w1c25_tpr + k29w1c25_tnr), 6)

k29w2c25_tpr <- round(sum(cm_c25_tables[1, 281:290]) /

```

```

(sum(cm_c25_tables[1, 281:290]) + sum(cm_c25_tables[2, 281:290])), 6)
k29w2c25_tnr <- round(sum(cm_c25_tables[4, 281:290]) /
  (sum(cm_c25_tables[4, 281:290]) + sum(cm_c25_tables[3, 281:290])), 6)
k29w2c25_truescore <- round((2 * k29w2c25_tpr * k29w2c25_tnr) /
  (k29w2c25_tpr + k29w2c25_tnr), 6)

k29w3c25_tpr <- round(sum(cm_c25_tables[1, 291:300]) /
  (sum(cm_c25_tables[1, 291:300]) + sum(cm_c25_tables[2, 291:300])), 6)
k29w3c25_tnr <- round(sum(cm_c25_tables[4, 291:300]) /
  (sum(cm_c25_tables[4, 291:300]) + sum(cm_c25_tables[3, 291:300])), 6)
k29w3c25_truescore <- round((2 * k29w3c25_tpr * k29w3c25_tnr) /
  (k29w3c25_tpr + k29w3c25_tnr), 6)

k31w1c25_tpr <- round(sum(cm_c25_tables[1, 301:310]) /
  (sum(cm_c25_tables[1, 301:310]) + sum(cm_c25_tables[2, 301:310])), 6)
k31w1c25_tnr <- round(sum(cm_c25_tables[4, 301:310]) /
  (sum(cm_c25_tables[4, 301:310]) + sum(cm_c25_tables[3, 301:310])), 6)
k31w1c25_truescore <- round((2 * k31w1c25_tpr * k31w1c25_tnr) /
  (k31w1c25_tpr + k31w1c25_tnr), 6)

k31w2c25_tpr <- round(sum(cm_c25_tables[1, 311:320]) /
  (sum(cm_c25_tables[1, 311:320]) + sum(cm_c25_tables[2, 311:320])), 6)
k31w2c25_tnr <- round(sum(cm_c25_tables[4, 311:320]) /
  (sum(cm_c25_tables[4, 311:320]) + sum(cm_c25_tables[3, 311:320])), 6)
k31w2c25_truescore <- round((2 * k31w2c25_tpr * k31w2c25_tnr) /
  (k31w2c25_tpr + k31w2c25_tnr), 6)

k31w3c25_tpr <- round(sum(cm_c25_tables[1, 321:330]) /
  (sum(cm_c25_tables[1, 321:330]) + sum(cm_c25_tables[2, 321:330])), 6)
k31w3c25_tnr <- round(sum(cm_c25_tables[4, 321:330]) /
  (sum(cm_c25_tables[4, 321:330]) + sum(cm_c25_tables[3, 321:330])), 6)
k31w3c25_truescore <- round((2 * k31w3c25_tpr * k31w3c25_tnr) /
  (k31w3c25_tpr + k31w3c25_tnr), 6)

k33w1c25_tpr <- round(sum(cm_c25_tables[1, 331:340]) /
  (sum(cm_c25_tables[1, 331:340]) + sum(cm_c25_tables[2, 331:340])), 6)
k33w1c25_tnr <- round(sum(cm_c25_tables[4, 331:340]) /
  (sum(cm_c25_tables[4, 331:340]) + sum(cm_c25_tables[3, 331:340])), 6)
k33w1c25_truescore <- round((2 * k33w1c25_tpr * k33w1c25_tnr) /
  (k33w1c25_tpr + k33w1c25_tnr), 6)

k33w2c25_tpr <- round(sum(cm_c25_tables[1, 341:350]) /
  (sum(cm_c25_tables[1, 341:350]) + sum(cm_c25_tables[2, 341:350])), 6)
k33w2c25_tnr <- round(sum(cm_c25_tables[4, 341:350]) /
  (sum(cm_c25_tables[4, 341:350]) + sum(cm_c25_tables[3, 341:350])), 6)
k33w2c25_truescore <- round((2 * k33w2c25_tpr * k33w2c25_tnr) /
  (k33w2c25_tpr + k33w2c25_tnr), 6)

k33w3c25_tpr <- round(sum(cm_c25_tables[1, 351:360]) /
  (sum(cm_c25_tables[1, 351:360]) + sum(cm_c25_tables[2, 351:360])), 6)
k33w3c25_tnr <- round(sum(cm_c25_tables[4, 351:360]) /

```

```

(sum(cm_c25_tables[4, 351:360]) + sum(cm_c25_tables[3, 351:360])), 6)
k33w3c25_truescore <- round((2 * k33w3c25_tpr * k33w3c25_tnr) /
(k33w3c25_tpr + k33w3c25_tnr), 6)

k35w1c25_tpr <- round(sum(cm_c25_tables[1, 361:370]) /
(sum(cm_c25_tables[1, 361:370]) + sum(cm_c25_tables[2, 361:370])), 6)
k35w1c25_tnr <- round(sum(cm_c25_tables[4, 361:370]) /
(sum(cm_c25_tables[4, 361:370]) + sum(cm_c25_tables[3, 361:370])), 6)
k35w1c25_truescore <- round((2 * k35w1c25_tpr * k35w1c25_tnr) /
(k35w1c25_tpr + k35w1c25_tnr), 6)

k35w2c25_tpr <- round(sum(cm_c25_tables[1, 371:380]) /
(sum(cm_c25_tables[1, 371:380]) + sum(cm_c25_tables[2, 371:380])), 6)
k35w2c25_tnr <- round(sum(cm_c25_tables[4, 371:380]) /
(sum(cm_c25_tables[4, 371:380]) + sum(cm_c25_tables[3, 371:380])), 6)
k35w2c25_truescore <- round((2 * k35w2c25_tpr * k35w2c25_tnr) /
(k35w2c25_tpr + k35w2c25_tnr), 6)

k35w3c25_tpr <- round(sum(cm_c25_tables[1, 381:390]) /
(sum(cm_c25_tables[1, 381:390]) + sum(cm_c25_tables[2, 381:390])), 6)
k35w3c25_tnr <- round(sum(cm_c25_tables[4, 381:390]) /
(sum(cm_c25_tables[4, 381:390]) + sum(cm_c25_tables[3, 381:390])), 6)
k35w3c25_truescore <- round((2 * k35w3c25_tpr * k35w3c25_tnr) /
(k35w3c25_tpr + k35w3c25_tnr), 6)

k37w1c25_tpr <- round(sum(cm_c25_tables[1, 391:400]) /
(sum(cm_c25_tables[1, 391:400]) + sum(cm_c25_tables[2, 391:400])), 6)
k37w1c25_tnr <- round(sum(cm_c25_tables[4, 391:400]) /
(sum(cm_c25_tables[4, 391:400]) + sum(cm_c25_tables[3, 391:400])), 6)
k37w1c25_truescore <- round((2 * k37w1c25_tpr * k37w1c25_tnr) /
(k37w1c25_tpr + k37w1c25_tnr), 6)

k37w2c25_tpr <- round(sum(cm_c25_tables[1, 401:410]) /
(sum(cm_c25_tables[1, 401:410]) + sum(cm_c25_tables[2, 401:410])), 6)
k37w2c25_tnr <- round(sum(cm_c25_tables[4, 401:410]) /
(sum(cm_c25_tables[4, 401:410]) + sum(cm_c25_tables[3, 401:410])), 6)
k37w2c25_truescore <- round((2 * k37w2c25_tpr * k37w2c25_tnr) /
(k37w2c25_tpr + k37w2c25_tnr), 6)

k37w3c25_tpr <- round(sum(cm_c25_tables[1, 411:420]) /
(sum(cm_c25_tables[1, 411:420]) + sum(cm_c25_tables[2, 411:420])), 6)
k37w3c25_tnr <- round(sum(cm_c25_tables[4, 411:420]) /
(sum(cm_c25_tables[4, 411:420]) + sum(cm_c25_tables[3, 411:420])), 6)
k37w3c25_truescore <- round((2 * k37w3c25_tpr * k37w3c25_tnr) /
(k37w3c25_tpr + k37w3c25_tnr), 6)

k39w1c25_tpr <- round(sum(cm_c25_tables[1, 421:430]) /
(sum(cm_c25_tables[1, 421:430]) + sum(cm_c25_tables[2, 421:430])), 6)
k39w1c25_tnr <- round(sum(cm_c25_tables[4, 421:430]) /
(sum(cm_c25_tables[4, 421:430]) + sum(cm_c25_tables[3, 421:430])), 6)

```

```

k39w1c25_truescore <- round((2 * k39w1c25_tpr * k39w1c25_tnr) /
  (k39w1c25_tpr + k39w1c25_tnr), 6)

k39w2c25_tpr <- round(sum(cm_c25_tables[1, 431:440]) /
  (sum(cm_c25_tables[1, 431:440]) + sum(cm_c25_tables[2, 431:440])), 6)
k39w2c25_tnr <- round(sum(cm_c25_tables[4, 431:440]) /
  (sum(cm_c25_tables[4, 431:440]) + sum(cm_c25_tables[3, 431:440])), 6)
k39w2c25_truescore <- round((2 * k39w2c25_tpr * k39w2c25_tnr) /
  (k39w2c25_tpr + k39w2c25_tnr), 6)

k39w3c25_tpr <- round(sum(cm_c25_tables[1, 441:450]) /
  (sum(cm_c25_tables[1, 441:450]) + sum(cm_c25_tables[2, 441:450])), 6)
k39w3c25_tnr <- round(sum(cm_c25_tables[4, 441:450]) /
  (sum(cm_c25_tables[4, 441:450]) + sum(cm_c25_tables[3, 441:450])), 6)
k39w3c25_truescore <- round((2 * k39w3c25_tpr * k39w3c25_tnr) /
  (k39w3c25_tpr + k39w3c25_tnr), 6)

# Compile the 0.25 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c25_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.25,
  TPR = c(k11w1c25_tpr, k11w2c25_tpr, k11w3c25_tpr, k13w1c25_tpr,
  k13w2c25_tpr, k13w3c25_tpr, k15w1c25_tpr, k15w2c25_tpr,
  k15w3c25_tpr, k17w1c25_tpr, k17w2c25_tpr, k17w3c25_tpr,
  k19w1c25_tpr, k19w2c25_tpr, k19w3c25_tpr, k21w1c25_tpr,
  k21w2c25_tpr, k21w3c25_tpr, k23w1c25_tpr, k23w2c25_tpr,
  k23w3c25_tpr, k25w1c25_tpr, k25w2c25_tpr, k25w3c25_tpr,
  k27w1c25_tpr, k27w2c25_tpr, k27w3c25_tpr, k29w1c25_tpr,
  k29w2c25_tpr, k29w3c25_tpr, k31w1c25_tpr, k31w2c25_tpr,
  k31w3c25_tpr, k33w1c25_tpr, k33w2c25_tpr, k33w3c25_tpr,
  k35w1c25_tpr, k35w2c25_tpr, k35w3c25_tpr, k37w1c25_tpr,
  k37w2c25_tpr, k37w3c25_tpr, k39w1c25_tpr, k39w2c25_tpr,
  k39w3c25_tpr),
  TNR = c(k11w1c25_tnr, k11w2c25_tnr, k11w3c25_tnr, k13w1c25_tnr,
  k13w2c25_tnr, k13w3c25_tnr, k15w1c25_tnr, k15w2c25_tnr,
  k15w3c25_tnr, k17w1c25_tnr, k17w2c25_tnr, k17w3c25_tnr,
  k19w1c25_tnr, k19w2c25_tnr, k19w3c25_tnr, k21w1c25_tnr,
  k21w2c25_tnr, k21w3c25_tnr, k23w1c25_tnr, k23w2c25_tnr,
  k23w3c25_tnr, k25w1c25_tnr, k25w2c25_tnr, k25w3c25_tnr,
  k27w1c25_tnr, k27w2c25_tnr, k27w3c25_tnr, k29w1c25_tnr,
  k29w2c25_tnr, k29w3c25_tnr, k31w1c25_tnr, k31w2c25_tnr,
  k31w3c25_tnr, k33w1c25_tnr, k33w2c25_tnr, k33w3c25_tnr,
  k35w1c25_tnr, k35w2c25_tnr, k35w3c25_tnr, k37w1c25_tnr,
  k37w2c25_tnr, k37w3c25_tnr, k39w1c25_tnr, k39w2c25_tnr,
  k39w3c25_tnr),
  Truescore = c(k11w1c25_truescore, k11w2c25_truescore,

```

```

k11w3c25_truescore, k13w1c25_truescore,
k13w2c25_truescore, k13w3c25_truescore,
k15w1c25_truescore, k15w2c25_truescore,
k15w3c25_truescore, k17w1c25_truescore,
k17w2c25_truescore, k17w3c25_truescore,
k19w1c25_truescore, k19w2c25_truescore,
k19w3c25_truescore, k21w1c25_truescore,
k21w2c25_truescore, k21w3c25_truescore,
k23w1c25_truescore, k23w2c25_truescore,
k23w3c25_truescore, k25w1c25_truescore,
k25w2c25_truescore, k25w3c25_truescore,
k27w1c25_truescore, k27w2c25_truescore,
k27w3c25_truescore, k29w1c25_truescore,
k29w2c25_truescore, k29w3c25_truescore,
k31w1c25_truescore, k31w2c25_truescore,
k31w3c25_truescore, k33w1c25_truescore,
k33w2c25_truescore, k33w3c25_truescore,
k35w1c25_truescore, k35w2c25_truescore,
k35w3c25_truescore, k37w1c25_truescore,
k37w2c25_truescore, k37w3c25_truescore,
k39w1c25_truescore, k39w2c25_truescore,
k39w3c25_truescore))

knitr::kable(c25_results[1:45, ], caption = "c25_results")

```

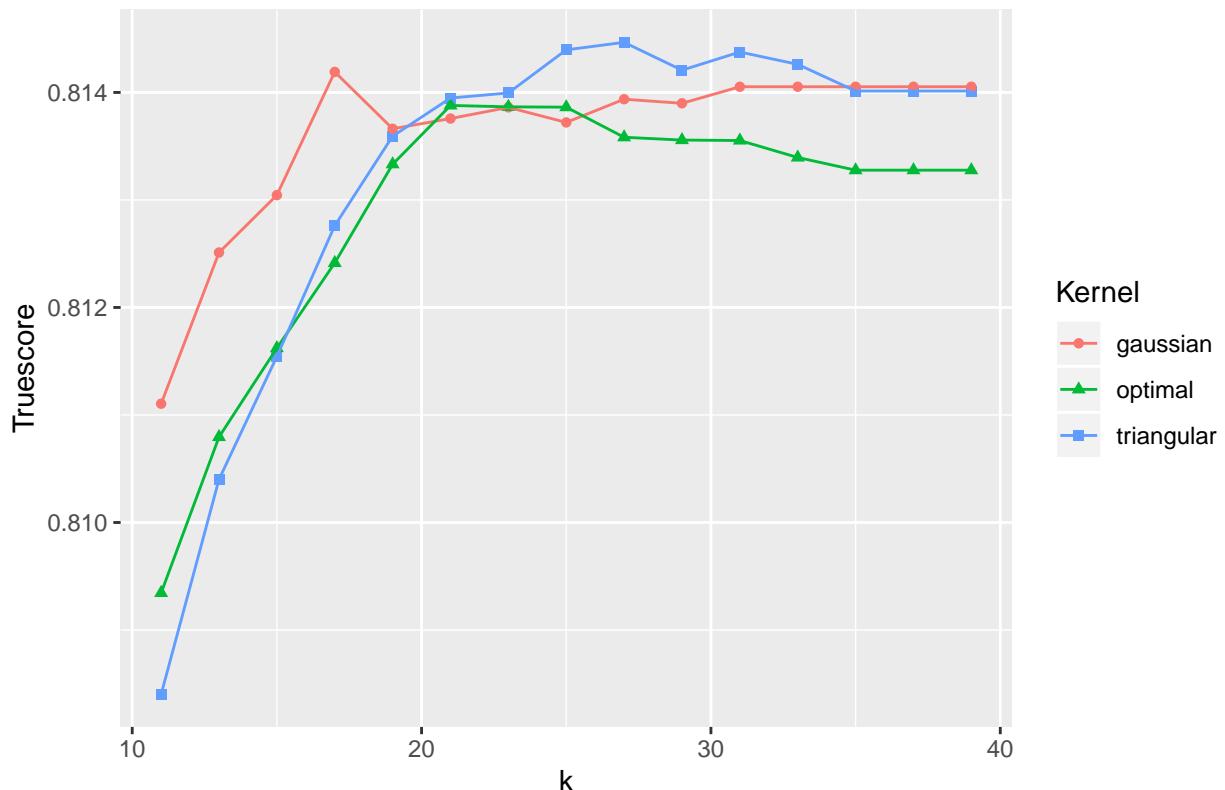
Table 34: c25_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.25	0.844679	0.775113	0.808402
11	gaussian	0.25	0.858785	0.768441	0.811105
11	optimal	0.25	0.846386	0.775410	0.809345
13	triangular	0.25	0.851707	0.772916	0.810401
13	gaussian	0.25	0.863404	0.767284	0.812511
13	optimal	0.25	0.853665	0.772025	0.810795
15	triangular	0.25	0.857078	0.770604	0.811544
15	gaussian	0.25	0.867269	0.765203	0.813045
15	optimal	0.25	0.858133	0.769894	0.811622
17	triangular	0.25	0.861697	0.769085	0.812761
17	gaussian	0.25	0.871185	0.764196	0.814191
17	optimal	0.25	0.861747	0.768424	0.812414
19	triangular	0.25	0.864508	0.768341	0.813593
19	gaussian	0.25	0.872189	0.762495	0.813662
19	optimal	0.25	0.865512	0.767086	0.813332
21	triangular	0.25	0.867520	0.766607	0.813948
21	gaussian	0.25	0.874297	0.761058	0.813757
21	optimal	0.25	0.868976	0.765352	0.813879
23	triangular	0.25	0.870030	0.764741	0.813995
23	gaussian	0.25	0.875301	0.760480	0.813861
23	optimal	0.25	0.870934	0.763816	0.813866
25	triangular	0.25	0.872841	0.763287	0.814396
25	gaussian	0.25	0.876205	0.759555	0.813721
25	optimal	0.25	0.873193	0.762082	0.813863
27	triangular	0.25	0.873996	0.762528	0.814466

k	Kernel	Cut	TPR	TNR	Truescore
27	gaussian	0.25	0.876707	0.759555	0.813937
27	optimal	0.25	0.873394	0.761438	0.813583
29	triangular	0.25	0.875402	0.761008	0.814207
29	gaussian	0.25	0.876707	0.759489	0.813899
29	optimal	0.25	0.874950	0.760215	0.813557
31	triangular	0.25	0.876406	0.760546	0.814376
31	gaussian	0.25	0.878213	0.758630	0.814053
31	optimal	0.25	0.875904	0.759489	0.813553
33	triangular	0.25	0.876908	0.759968	0.814261
33	gaussian	0.25	0.878213	0.758630	0.814053
33	optimal	0.25	0.876506	0.758762	0.813395
35	triangular	0.25	0.877259	0.759274	0.814013
35	gaussian	0.25	0.878213	0.758630	0.814053
35	optimal	0.25	0.876606	0.758481	0.813277
37	triangular	0.25	0.877259	0.759274	0.814013
37	gaussian	0.25	0.878213	0.758630	0.814053
37	optimal	0.25	0.876606	0.758481	0.813277
39	triangular	0.25	0.877259	0.759274	0.814013
39	gaussian	0.25	0.878213	0.758630	0.814053
39	optimal	0.25	0.876606	0.758481	0.813277

```
ggplot(c25_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.25 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.25 (Truescore)

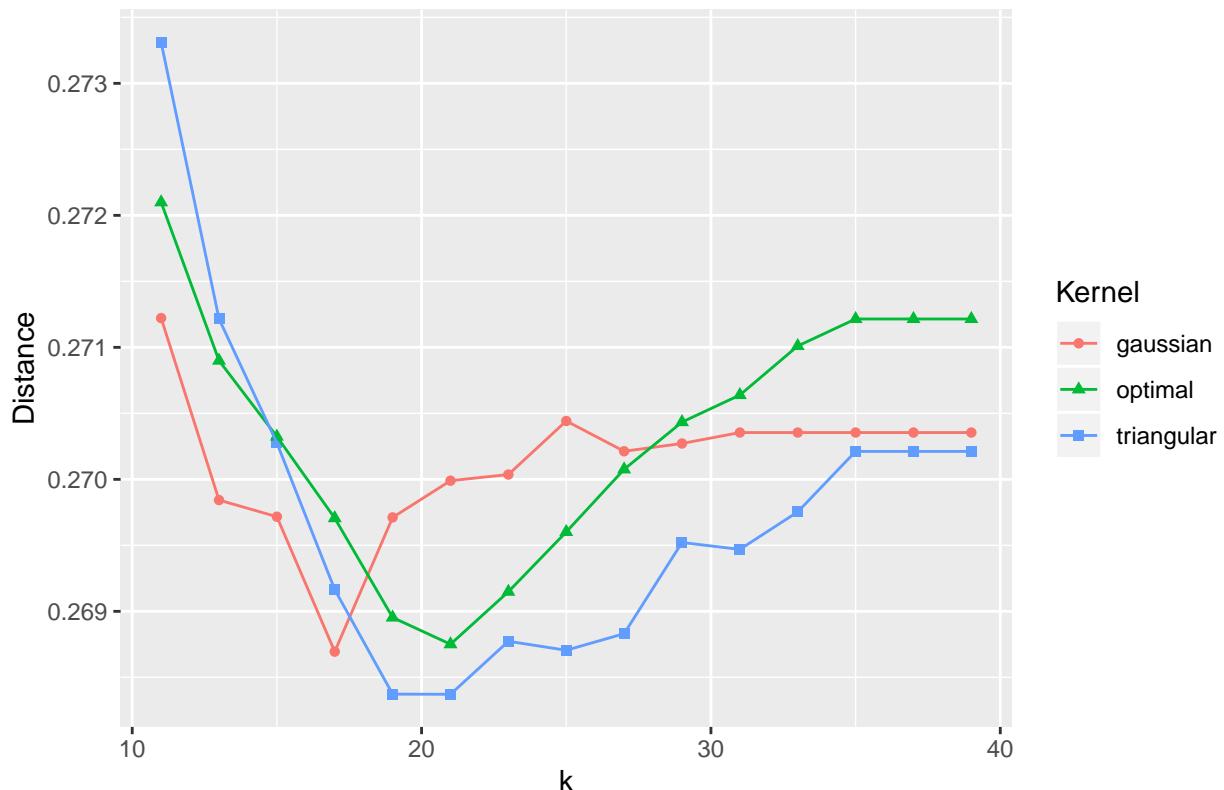


```
# For the Cutoff of 0.25, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c25_results <- c25_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c25_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.25 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.25 (Distance)



```
# For the Cutoff of 0.25, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c25_results$Truescore)
```

```
## [1] 0.814466
```

```
(c25_opt_k_ts <- c25_results$k[which.max(c25_results$Truescore)])
```

```
## [1] 27
```

```
(c25_opt_kernel_ts <- c25_results$Kernel[which.max(c25_results$Truescore)])
```

```
## [1] "triangular"
```

```
(c25_opt_cut_ts <- c25_results$Cut[which.max(c25_results$Truescore)])
```

```
## [1] 0.25
```

```
(c25_opt_tpr_ts <- c25_results$TPR[which.max(c25_results$Truescore)])
```

```
## [1] 0.873996
```

```

(c25_opt_tnr_ts <- c25_results$TNR[which.max(c25_results$Truescore)])

## [1] 0.762528

(c25_opt_d_ts <- c25_results$Distance[which.max(c25_results$Truescore)])

## [1] 0.2688307

min(c25_results$Distance)

## [1] 0.2683715

(c25_opt_k_dist <- c25_results$k[which.min(c25_results$Distance)])

## [1] 21

(c25_opt_kernel_dist <- c25_results$Kernel[which.min(c25_results$Distance)])

## [1] "triangular"

(c25_opt_cut_dist <- c25_results$Cut[which.min(c25_results$Distance)])

## [1] 0.25

(c25_opt_tpr_dist <- c25_results$TPR[which.min(c25_results$Distance)])

## [1] 0.86752

(c25_opt_tnr_dist <- c25_results$TNR[which.min(c25_results$Distance)])

## [1] 0.766607

(c25_opt_t_dist <- c25_results$Truescore[which.min(c25_results$Distance)])

## [1] 0.813948

#####
# 0.26 Cutoff
#####

# For the decision cutoff of 0.26, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c26 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred26, obs))
  confusionMatrix(ss$pred26, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c26) <- kwf_dfs_v

cm_c26_tables <- sapply(cm_c26, "[[", 2)
cm_c26_tables <- as_tibble(cm_c26_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.26.

k11w1c26_tpr <- round(sum(cm_c26_tables[1, 1:10]) /
  (sum(cm_c26_tables[1, 1:10]) + sum(cm_c26_tables[2, 1:10])), 6)
k11w1c26_tnr <- round(sum(cm_c26_tables[4, 1:10]) /
  (sum(cm_c26_tables[4, 1:10]) + sum(cm_c26_tables[3, 1:10])), 6)
k11w1c26_truescore <- round((2 * k11w1c26_tpr * k11w1c26_tnr) /
  (k11w1c26_tpr + k11w1c26_tnr), 6)

k11w2c26_tpr <- round(sum(cm_c26_tables[1, 11:20]) /
  (sum(cm_c26_tables[1, 11:20]) + sum(cm_c26_tables[2, 11:20])), 6)
k11w2c26_tnr <- round(sum(cm_c26_tables[4, 11:20]) /
  (sum(cm_c26_tables[4, 11:20]) + sum(cm_c26_tables[3, 11:20])), 6)
k11w2c26_truescore <- round((2 * k11w2c26_tpr * k11w2c26_tnr) /
  (k11w2c26_tpr + k11w2c26_tnr), 6)

k11w3c26_tpr <- round(sum(cm_c26_tables[1, 21:30]) /
  (sum(cm_c26_tables[1, 21:30]) + sum(cm_c26_tables[2, 21:30])), 6)
k11w3c26_tnr <- round(sum(cm_c26_tables[4, 21:30]) /
  (sum(cm_c26_tables[4, 21:30]) + sum(cm_c26_tables[3, 21:30])), 6)
k11w3c26_truescore <- round((2 * k11w3c26_tpr * k11w3c26_tnr) /
  (k11w3c26_tpr + k11w3c26_tnr), 6)

k13w1c26_tpr <- round(sum(cm_c26_tables[1, 31:40]) /
  (sum(cm_c26_tables[1, 31:40]) + sum(cm_c26_tables[2, 31:40])), 6)
k13w1c26_tnr <- round(sum(cm_c26_tables[4, 31:40]) /
  (sum(cm_c26_tables[4, 31:40]) + sum(cm_c26_tables[3, 31:40])), 6)
k13w1c26_truescore <- round((2 * k13w1c26_tpr * k13w1c26_tnr) /
  (k13w1c26_tpr + k13w1c26_tnr), 6)

k13w2c26_tpr <- round(sum(cm_c26_tables[1, 41:50]) /
  (sum(cm_c26_tables[1, 41:50]) + sum(cm_c26_tables[2, 41:50])), 6)
k13w2c26_tnr <- round(sum(cm_c26_tables[4, 41:50]) /
  (sum(cm_c26_tables[4, 41:50]) + sum(cm_c26_tables[3, 41:50])), 6)
k13w2c26_truescore <- round((2 * k13w2c26_tpr * k13w2c26_tnr) /
  (k13w2c26_tpr + k13w2c26_tnr), 6)

k13w3c26_tpr <- round(sum(cm_c26_tables[1, 51:60]) /
  (sum(cm_c26_tables[1, 51:60]) + sum(cm_c26_tables[2, 51:60])), 6)
k13w3c26_tnr <- round(sum(cm_c26_tables[4, 51:60]) /
  (sum(cm_c26_tables[4, 51:60]) + sum(cm_c26_tables[3, 51:60])), 6)
k13w3c26_truescore <- round((2 * k13w3c26_tpr * k13w3c26_tnr) /

```

```

(k13w3c26_tpr + k13w3c26_tnr), 6)

k15w1c26_tpr <- round(sum(cm_c26_tables[1, 61:70]) /
  (sum(cm_c26_tables[1, 61:70]) + sum(cm_c26_tables[2, 61:70])), 6)
k15w1c26_tnr <- round(sum(cm_c26_tables[4, 61:70]) /
  (sum(cm_c26_tables[4, 61:70]) + sum(cm_c26_tables[3, 61:70])), 6)
k15w1c26_truescore <- round((2 * k15w1c26_tpr * k15w1c26_tnr) /
  (k15w1c26_tpr + k15w1c26_tnr), 6)

k15w2c26_tpr <- round(sum(cm_c26_tables[1, 71:80]) /
  (sum(cm_c26_tables[1, 71:80]) + sum(cm_c26_tables[2, 71:80])), 6)
k15w2c26_tnr <- round(sum(cm_c26_tables[4, 71:80]) /
  (sum(cm_c26_tables[4, 71:80]) + sum(cm_c26_tables[3, 71:80])), 6)
k15w2c26_truescore <- round((2 * k15w2c26_tpr * k15w2c26_tnr) /
  (k15w2c26_tpr + k15w2c26_tnr), 6)

k15w3c26_tpr <- round(sum(cm_c26_tables[1, 81:90]) /
  (sum(cm_c26_tables[1, 81:90]) + sum(cm_c26_tables[2, 81:90])), 6)
k15w3c26_tnr <- round(sum(cm_c26_tables[4, 81:90]) /
  (sum(cm_c26_tables[4, 81:90]) + sum(cm_c26_tables[3, 81:90])), 6)
k15w3c26_truescore <- round((2 * k15w3c26_tpr * k15w3c26_tnr) /
  (k15w3c26_tpr + k15w3c26_tnr), 6)

k17w1c26_tpr <- round(sum(cm_c26_tables[1, 91:100]) /
  (sum(cm_c26_tables[1, 91:100]) + sum(cm_c26_tables[2, 91:100])), 6)
k17w1c26_tnr <- round(sum(cm_c26_tables[4, 91:100]) /
  (sum(cm_c26_tables[4, 91:100]) + sum(cm_c26_tables[3, 91:100])), 6)
k17w1c26_truescore <- round((2 * k17w1c26_tpr * k17w1c26_tnr) /
  (k17w1c26_tpr + k17w1c26_tnr), 6)

k17w2c26_tpr <- round(sum(cm_c26_tables[1, 101:110]) /
  (sum(cm_c26_tables[1, 101:110]) + sum(cm_c26_tables[2, 101:110])), 6)
k17w2c26_tnr <- round(sum(cm_c26_tables[4, 101:110]) /
  (sum(cm_c26_tables[4, 101:110]) + sum(cm_c26_tables[3, 101:110])), 6)
k17w2c26_truescore <- round((2 * k17w2c26_tpr * k17w2c26_tnr) /
  (k17w2c26_tpr + k17w2c26_tnr), 6)

k17w3c26_tpr <- round(sum(cm_c26_tables[1, 111:120]) /
  (sum(cm_c26_tables[1, 111:120]) + sum(cm_c26_tables[2, 111:120])), 6)
k17w3c26_tnr <- round(sum(cm_c26_tables[4, 111:120]) /
  (sum(cm_c26_tables[4, 111:120]) + sum(cm_c26_tables[3, 111:120])), 6)
k17w3c26_truescore <- round((2 * k17w3c26_tpr * k17w3c26_tnr) /
  (k17w3c26_tpr + k17w3c26_tnr), 6)

k19w1c26_tpr <- round(sum(cm_c26_tables[1, 121:130]) /
  (sum(cm_c26_tables[1, 121:130]) + sum(cm_c26_tables[2, 121:130])), 6)
k19w1c26_tnr <- round(sum(cm_c26_tables[4, 121:130]) /
  (sum(cm_c26_tables[4, 121:130]) + sum(cm_c26_tables[3, 121:130])), 6)
k19w1c26_truescore <- round((2 * k19w1c26_tpr * k19w1c26_tnr) /
  (k19w1c26_tpr + k19w1c26_tnr), 6)

```

```

k19w2c26_tpr <- round(sum(cm_c26_tables[1, 131:140]) /
  (sum(cm_c26_tables[1, 131:140]) + sum(cm_c26_tables[2, 131:140])), 6)
k19w2c26_tnr <- round(sum(cm_c26_tables[4, 131:140]) /
  (sum(cm_c26_tables[4, 131:140]) + sum(cm_c26_tables[3, 131:140])), 6)
k19w2c26_truescore <- round((2 * k19w2c26_tpr * k19w2c26_tnr) /
  (k19w2c26_tpr + k19w2c26_tnr), 6)

k19w3c26_tpr <- round(sum(cm_c26_tables[1, 141:150]) /
  (sum(cm_c26_tables[1, 141:150]) + sum(cm_c26_tables[2, 141:150])), 6)
k19w3c26_tnr <- round(sum(cm_c26_tables[4, 141:150]) /
  (sum(cm_c26_tables[4, 141:150]) + sum(cm_c26_tables[3, 141:150])), 6)
k19w3c26_truescore <- round((2 * k19w3c26_tpr * k19w3c26_tnr) /
  (k19w3c26_tpr + k19w3c26_tnr), 6)

k21w1c26_tpr <- round(sum(cm_c26_tables[1, 151:160]) /
  (sum(cm_c26_tables[1, 151:160]) + sum(cm_c26_tables[2, 151:160])), 6)
k21w1c26_tnr <- round(sum(cm_c26_tables[4, 151:160]) /
  (sum(cm_c26_tables[4, 151:160]) + sum(cm_c26_tables[3, 151:160])), 6)
k21w1c26_truescore <- round((2 * k21w1c26_tpr * k21w1c26_tnr) /
  (k21w1c26_tpr + k21w1c26_tnr), 6)

k21w2c26_tpr <- round(sum(cm_c26_tables[1, 161:170]) /
  (sum(cm_c26_tables[1, 161:170]) + sum(cm_c26_tables[2, 161:170])), 6)
k21w2c26_tnr <- round(sum(cm_c26_tables[4, 161:170]) /
  (sum(cm_c26_tables[4, 161:170]) + sum(cm_c26_tables[3, 161:170])), 6)
k21w2c26_truescore <- round((2 * k21w2c26_tpr * k21w2c26_tnr) /
  (k21w2c26_tpr + k21w2c26_tnr), 6)

k21w3c26_tpr <- round(sum(cm_c26_tables[1, 171:180]) /
  (sum(cm_c26_tables[1, 171:180]) + sum(cm_c26_tables[2, 171:180])), 6)
k21w3c26_tnr <- round(sum(cm_c26_tables[4, 171:180]) /
  (sum(cm_c26_tables[4, 171:180]) + sum(cm_c26_tables[3, 171:180])), 6)
k21w3c26_truescore <- round((2 * k21w3c26_tpr * k21w3c26_tnr) /
  (k21w3c26_tpr + k21w3c26_tnr), 6)

k23w1c26_tpr <- round(sum(cm_c26_tables[1, 181:190]) /
  (sum(cm_c26_tables[1, 181:190]) + sum(cm_c26_tables[2, 181:190])), 6)
k23w1c26_tnr <- round(sum(cm_c26_tables[4, 181:190]) /
  (sum(cm_c26_tables[4, 181:190]) + sum(cm_c26_tables[3, 181:190])), 6)
k23w1c26_truescore <- round((2 * k23w1c26_tpr * k23w1c26_tnr) /
  (k23w1c26_tpr + k23w1c26_tnr), 6)

k23w2c26_tpr <- round(sum(cm_c26_tables[1, 191:200]) /
  (sum(cm_c26_tables[1, 191:200]) + sum(cm_c26_tables[2, 191:200])), 6)
k23w2c26_tnr <- round(sum(cm_c26_tables[4, 191:200]) /
  (sum(cm_c26_tables[4, 191:200]) + sum(cm_c26_tables[3, 191:200])), 6)
k23w2c26_truescore <- round((2 * k23w2c26_tpr * k23w2c26_tnr) /
  (k23w2c26_tpr + k23w2c26_tnr), 6)

k23w3c26_tpr <- round(sum(cm_c26_tables[1, 201:210]) /
  (sum(cm_c26_tables[1, 201:210]) + sum(cm_c26_tables[2, 201:210])), 6)

```

```

k23w3c26_tnr <- round(sum(cm_c26_tables[4, 201:210]) /
  (sum(cm_c26_tables[4, 201:210]) + sum(cm_c26_tables[3, 201:210])), 6)
k23w3c26_truescore <- round((2 * k23w3c26_tpr * k23w3c26_tnr) /
  (k23w3c26_tpr + k23w3c26_tnr), 6)

k25w1c26_tpr <- round(sum(cm_c26_tables[1, 211:220]) /
  (sum(cm_c26_tables[1, 211:220]) + sum(cm_c26_tables[2, 211:220])), 6)
k25w1c26_tnr <- round(sum(cm_c26_tables[4, 211:220]) /
  (sum(cm_c26_tables[4, 211:220]) + sum(cm_c26_tables[3, 211:220])), 6)
k25w1c26_truescore <- round((2 * k25w1c26_tpr * k25w1c26_tnr) /
  (k25w1c26_tpr + k25w1c26_tnr), 6)

k25w2c26_tpr <- round(sum(cm_c26_tables[1, 221:230]) /
  (sum(cm_c26_tables[1, 221:230]) + sum(cm_c26_tables[2, 221:230])), 6)
k25w2c26_tnr <- round(sum(cm_c26_tables[4, 221:230]) /
  (sum(cm_c26_tables[4, 221:230]) + sum(cm_c26_tables[3, 221:230])), 6)
k25w2c26_truescore <- round((2 * k25w2c26_tpr * k25w2c26_tnr) /
  (k25w2c26_tpr + k25w2c26_tnr), 6)

k25w3c26_tpr <- round(sum(cm_c26_tables[1, 231:240]) /
  (sum(cm_c26_tables[1, 231:240]) + sum(cm_c26_tables[2, 231:240])), 6)
k25w3c26_tnr <- round(sum(cm_c26_tables[4, 231:240]) /
  (sum(cm_c26_tables[4, 231:240]) + sum(cm_c26_tables[3, 231:240])), 6)
k25w3c26_truescore <- round((2 * k25w3c26_tpr * k25w3c26_tnr) /
  (k25w3c26_tpr + k25w3c26_tnr), 6)

k27w1c26_tpr <- round(sum(cm_c26_tables[1, 241:250]) /
  (sum(cm_c26_tables[1, 241:250]) + sum(cm_c26_tables[2, 241:250])), 6)
k27w1c26_tnr <- round(sum(cm_c26_tables[4, 241:250]) /
  (sum(cm_c26_tables[4, 241:250]) + sum(cm_c26_tables[3, 241:250])), 6)
k27w1c26_truescore <- round((2 * k27w1c26_tpr * k27w1c26_tnr) /
  (k27w1c26_tpr + k27w1c26_tnr), 6)

k27w2c26_tpr <- round(sum(cm_c26_tables[1, 251:260]) /
  (sum(cm_c26_tables[1, 251:260]) + sum(cm_c26_tables[2, 251:260])), 6)
k27w2c26_tnr <- round(sum(cm_c26_tables[4, 251:260]) /
  (sum(cm_c26_tables[4, 251:260]) + sum(cm_c26_tables[3, 251:260])), 6)
k27w2c26_truescore <- round((2 * k27w2c26_tpr * k27w2c26_tnr) /
  (k27w2c26_tpr + k27w2c26_tnr), 6)

k27w3c26_tpr <- round(sum(cm_c26_tables[1, 261:270]) /
  (sum(cm_c26_tables[1, 261:270]) + sum(cm_c26_tables[2, 261:270])), 6)
k27w3c26_tnr <- round(sum(cm_c26_tables[4, 261:270]) /
  (sum(cm_c26_tables[4, 261:270]) + sum(cm_c26_tables[3, 261:270])), 6)
k27w3c26_truescore <- round((2 * k27w3c26_tpr * k27w3c26_tnr) /
  (k27w3c26_tpr + k27w3c26_tnr), 6)

k29w1c26_tpr <- round(sum(cm_c26_tables[1, 271:280]) /
  (sum(cm_c26_tables[1, 271:280]) + sum(cm_c26_tables[2, 271:280])), 6)
k29w1c26_tnr <- round(sum(cm_c26_tables[4, 271:280]) /

```

```

(sum(cm_c26_tables[4, 271:280]) + sum(cm_c26_tables[3, 271:280])), 6)
k29w1c26_truescore <- round((2 * k29w1c26_tpr * k29w1c26_tnr) /
(k29w1c26_tpr + k29w1c26_tnr), 6)

k29w2c26_tpr <- round(sum(cm_c26_tables[1, 281:290]) /
(sum(cm_c26_tables[1, 281:290]) + sum(cm_c26_tables[2, 281:290])), 6)
k29w2c26_tnr <- round(sum(cm_c26_tables[4, 281:290]) /
(sum(cm_c26_tables[4, 281:290]) + sum(cm_c26_tables[3, 281:290])), 6)
k29w2c26_truescore <- round((2 * k29w2c26_tpr * k29w2c26_tnr) /
(k29w2c26_tpr + k29w2c26_tnr), 6)

k29w3c26_tpr <- round(sum(cm_c26_tables[1, 291:300]) /
(sum(cm_c26_tables[1, 291:300]) + sum(cm_c26_tables[2, 291:300])), 6)
k29w3c26_tnr <- round(sum(cm_c26_tables[4, 291:300]) /
(sum(cm_c26_tables[4, 291:300]) + sum(cm_c26_tables[3, 291:300])), 6)
k29w3c26_truescore <- round((2 * k29w3c26_tpr * k29w3c26_tnr) /
(k29w3c26_tpr + k29w3c26_tnr), 6)

k31w1c26_tpr <- round(sum(cm_c26_tables[1, 301:310]) /
(sum(cm_c26_tables[1, 301:310]) + sum(cm_c26_tables[2, 301:310])), 6)
k31w1c26_tnr <- round(sum(cm_c26_tables[4, 301:310]) /
(sum(cm_c26_tables[4, 301:310]) + sum(cm_c26_tables[3, 301:310])), 6)
k31w1c26_truescore <- round((2 * k31w1c26_tpr * k31w1c26_tnr) /
(k31w1c26_tpr + k31w1c26_tnr), 6)

k31w2c26_tpr <- round(sum(cm_c26_tables[1, 311:320]) /
(sum(cm_c26_tables[1, 311:320]) + sum(cm_c26_tables[2, 311:320])), 6)
k31w2c26_tnr <- round(sum(cm_c26_tables[4, 311:320]) /
(sum(cm_c26_tables[4, 311:320]) + sum(cm_c26_tables[3, 311:320])), 6)
k31w2c26_truescore <- round((2 * k31w2c26_tpr * k31w2c26_tnr) /
(k31w2c26_tpr + k31w2c26_tnr), 6)

k31w3c26_tpr <- round(sum(cm_c26_tables[1, 321:330]) /
(sum(cm_c26_tables[1, 321:330]) + sum(cm_c26_tables[2, 321:330])), 6)
k31w3c26_tnr <- round(sum(cm_c26_tables[4, 321:330]) /
(sum(cm_c26_tables[4, 321:330]) + sum(cm_c26_tables[3, 321:330])), 6)
k31w3c26_truescore <- round((2 * k31w3c26_tpr * k31w3c26_tnr) /
(k31w3c26_tpr + k31w3c26_tnr), 6)

k33w1c26_tpr <- round(sum(cm_c26_tables[1, 331:340]) /
(sum(cm_c26_tables[1, 331:340]) + sum(cm_c26_tables[2, 331:340])), 6)
k33w1c26_tnr <- round(sum(cm_c26_tables[4, 331:340]) /
(sum(cm_c26_tables[4, 331:340]) + sum(cm_c26_tables[3, 331:340])), 6)
k33w1c26_truescore <- round((2 * k33w1c26_tpr * k33w1c26_tnr) /
(k33w1c26_tpr + k33w1c26_tnr), 6)

k33w2c26_tpr <- round(sum(cm_c26_tables[1, 341:350]) /
(sum(cm_c26_tables[1, 341:350]) + sum(cm_c26_tables[2, 341:350])), 6)
k33w2c26_tnr <- round(sum(cm_c26_tables[4, 341:350]) /
(sum(cm_c26_tables[4, 341:350]) + sum(cm_c26_tables[3, 341:350])), 6)
k33w2c26_truescore <- round((2 * k33w2c26_tpr * k33w2c26_tnr) /

```

```

(k33w2c26_tpr + k33w2c26_tnr), 6)

k33w3c26_tpr <- round(sum(cm_c26_tables[1, 351:360]) /
  (sum(cm_c26_tables[1, 351:360]) + sum(cm_c26_tables[2, 351:360])), 6)
k33w3c26_tnr <- round(sum(cm_c26_tables[4, 351:360]) /
  (sum(cm_c26_tables[4, 351:360]) + sum(cm_c26_tables[3, 351:360])), 6)
k33w3c26_truescore <- round((2 * k33w3c26_tpr * k33w3c26_tnr) /
  (k33w3c26_tpr + k33w3c26_tnr), 6)

k35w1c26_tpr <- round(sum(cm_c26_tables[1, 361:370]) /
  (sum(cm_c26_tables[1, 361:370]) + sum(cm_c26_tables[2, 361:370])), 6)
k35w1c26_tnr <- round(sum(cm_c26_tables[4, 361:370]) /
  (sum(cm_c26_tables[4, 361:370]) + sum(cm_c26_tables[3, 361:370])), 6)
k35w1c26_truescore <- round((2 * k35w1c26_tpr * k35w1c26_tnr) /
  (k35w1c26_tpr + k35w1c26_tnr), 6)

k35w2c26_tpr <- round(sum(cm_c26_tables[1, 371:380]) /
  (sum(cm_c26_tables[1, 371:380]) + sum(cm_c26_tables[2, 371:380])), 6)
k35w2c26_tnr <- round(sum(cm_c26_tables[4, 371:380]) /
  (sum(cm_c26_tables[4, 371:380]) + sum(cm_c26_tables[3, 371:380])), 6)
k35w2c26_truescore <- round((2 * k35w2c26_tpr * k35w2c26_tnr) /
  (k35w2c26_tpr + k35w2c26_tnr), 6)

k35w3c26_tpr <- round(sum(cm_c26_tables[1, 381:390]) /
  (sum(cm_c26_tables[1, 381:390]) + sum(cm_c26_tables[2, 381:390])), 6)
k35w3c26_tnr <- round(sum(cm_c26_tables[4, 381:390]) /
  (sum(cm_c26_tables[4, 381:390]) + sum(cm_c26_tables[3, 381:390])), 6)
k35w3c26_truescore <- round((2 * k35w3c26_tpr * k35w3c26_tnr) /
  (k35w3c26_tpr + k35w3c26_tnr), 6)

k37w1c26_tpr <- round(sum(cm_c26_tables[1, 391:400]) /
  (sum(cm_c26_tables[1, 391:400]) + sum(cm_c26_tables[2, 391:400])), 6)
k37w1c26_tnr <- round(sum(cm_c26_tables[4, 391:400]) /
  (sum(cm_c26_tables[4, 391:400]) + sum(cm_c26_tables[3, 391:400])), 6)
k37w1c26_truescore <- round((2 * k37w1c26_tpr * k37w1c26_tnr) /
  (k37w1c26_tpr + k37w1c26_tnr), 6)

k37w2c26_tpr <- round(sum(cm_c26_tables[1, 401:410]) /
  (sum(cm_c26_tables[1, 401:410]) + sum(cm_c26_tables[2, 401:410])), 6)
k37w2c26_tnr <- round(sum(cm_c26_tables[4, 401:410]) /
  (sum(cm_c26_tables[4, 401:410]) + sum(cm_c26_tables[3, 401:410])), 6)
k37w2c26_truescore <- round((2 * k37w2c26_tpr * k37w2c26_tnr) /
  (k37w2c26_tpr + k37w2c26_tnr), 6)

k37w3c26_tpr <- round(sum(cm_c26_tables[1, 411:420]) /
  (sum(cm_c26_tables[1, 411:420]) + sum(cm_c26_tables[2, 411:420])), 6)
k37w3c26_tnr <- round(sum(cm_c26_tables[4, 411:420]) /
  (sum(cm_c26_tables[4, 411:420]) + sum(cm_c26_tables[3, 411:420])), 6)
k37w3c26_truescore <- round((2 * k37w3c26_tpr * k37w3c26_tnr) /
  (k37w3c26_tpr + k37w3c26_tnr), 6)

```

```

k39w1c26_tpr <- round(sum(cm_c26_tables[1, 421:430]) /
  (sum(cm_c26_tables[1, 421:430]) + sum(cm_c26_tables[2, 421:430])), 6)
k39w1c26_tnr <- round(sum(cm_c26_tables[4, 421:430]) /
  (sum(cm_c26_tables[4, 421:430]) + sum(cm_c26_tables[3, 421:430])), 6)
k39w1c26_truescore <- round((2 * k39w1c26_tpr * k39w1c26_tnr) /
  (k39w1c26_tpr + k39w1c26_tnr), 6)

k39w2c26_tpr <- round(sum(cm_c26_tables[1, 431:440]) /
  (sum(cm_c26_tables[1, 431:440]) + sum(cm_c26_tables[2, 431:440])), 6)
k39w2c26_tnr <- round(sum(cm_c26_tables[4, 431:440]) /
  (sum(cm_c26_tables[4, 431:440]) + sum(cm_c26_tables[3, 431:440])), 6)
k39w2c26_truescore <- round((2 * k39w2c26_tpr * k39w2c26_tnr) /
  (k39w2c26_tpr + k39w2c26_tnr), 6)

k39w3c26_tpr <- round(sum(cm_c26_tables[1, 441:450]) /
  (sum(cm_c26_tables[1, 441:450]) + sum(cm_c26_tables[2, 441:450])), 6)
k39w3c26_tnr <- round(sum(cm_c26_tables[4, 441:450]) /
  (sum(cm_c26_tables[4, 441:450]) + sum(cm_c26_tables[3, 441:450])), 6)
k39w3c26_truescore <- round((2 * k39w3c26_tpr * k39w3c26_tnr) /
  (k39w3c26_tpr + k39w3c26_tnr), 6)

# Compile the 0.26 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c26_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.26,
  TPR = c(k11w1c26_tpr, k11w2c26_tpr, k11w3c26_tpr, k13w1c26_tpr,
    k13w2c26_tpr, k13w3c26_tpr, k15w1c26_tpr, k15w2c26_tpr,
    k15w3c26_tpr, k17w1c26_tpr, k17w2c26_tpr, k17w3c26_tpr,
    k19w1c26_tpr, k19w2c26_tpr, k19w3c26_tpr, k21w1c26_tpr,
    k21w2c26_tpr, k21w3c26_tpr, k23w1c26_tpr, k23w2c26_tpr,
    k23w3c26_tpr, k25w1c26_tpr, k25w2c26_tpr, k25w3c26_tpr,
    k27w1c26_tpr, k27w2c26_tpr, k27w3c26_tpr, k29w1c26_tpr,
    k29w2c26_tpr, k29w3c26_tpr, k31w1c26_tpr, k31w2c26_tpr,
    k31w3c26_tpr, k33w1c26_tpr, k33w2c26_tpr, k33w3c26_tpr,
    k35w1c26_tpr, k35w2c26_tpr, k35w3c26_tpr, k37w1c26_tpr,
    k37w2c26_tpr, k37w3c26_tpr, k39w1c26_tpr, k39w2c26_tpr,
    k39w3c26_tpr),
  TNR = c(k11w1c26_tnr, k11w2c26_tnr, k11w3c26_tnr, k13w1c26_tnr,
    k13w2c26_tnr, k13w3c26_tnr, k15w1c26_tnr, k15w2c26_tnr,
    k15w3c26_tnr, k17w1c26_tnr, k17w2c26_tnr, k17w3c26_tnr,
    k19w1c26_tnr, k19w2c26_tnr, k19w3c26_tnr, k21w1c26_tnr,
    k21w2c26_tnr, k21w3c26_tnr, k23w1c26_tnr, k23w2c26_tnr,
    k23w3c26_tnr, k25w1c26_tnr, k25w2c26_tnr, k25w3c26_tnr,
    k27w1c26_tnr, k27w2c26_tnr, k27w3c26_tnr, k29w1c26_tnr,
    k29w2c26_tnr, k29w3c26_tnr, k31w1c26_tnr, k31w2c26_tnr,
    k31w3c26_tnr, k33w1c26_tnr, k33w2c26_tnr, k33w3c26_tnr)

```

```

k35w1c26_tnr, k35w2c26_tnr, k35w3c26_tnr, k37w1c26_tnr,
k37w2c26_tnr, k37w3c26_tnr, k39w1c26_tnr, k39w2c26_tnr,
k39w3c26_tnr),
Truescore = c(k11w1c26_truescore, k11w2c26_truescore,
k11w3c26_truescore, k13w1c26_truescore,
k13w2c26_truescore, k13w3c26_truescore,
k15w1c26_truescore, k15w2c26_truescore,
k15w3c26_truescore, k17w1c26_truescore,
k17w2c26_truescore, k17w3c26_truescore,
k19w1c26_truescore, k19w2c26_truescore,
k19w3c26_truescore, k21w1c26_truescore,
k21w2c26_truescore, k21w3c26_truescore,
k23w1c26_truescore, k23w2c26_truescore,
k23w3c26_truescore, k25w1c26_truescore,
k25w2c26_truescore, k25w3c26_truescore,
k27w1c26_truescore, k27w2c26_truescore,
k27w3c26_truescore, k29w1c26_truescore,
k29w2c26_truescore, k29w3c26_truescore,
k31w1c26_truescore, k31w2c26_truescore,
k31w3c26_truescore, k33w1c26_truescore,
k33w2c26_truescore, k33w3c26_truescore,
k35w1c26_truescore, k35w2c26_truescore,
k35w3c26_truescore, k37w1c26_truescore,
k37w2c26_truescore, k37w3c26_truescore,
k39w1c26_truescore, k39w2c26_truescore,
k39w3c26_truescore))

```

`knitr::kable(c26_results[1:45,], caption = "c26_results")`

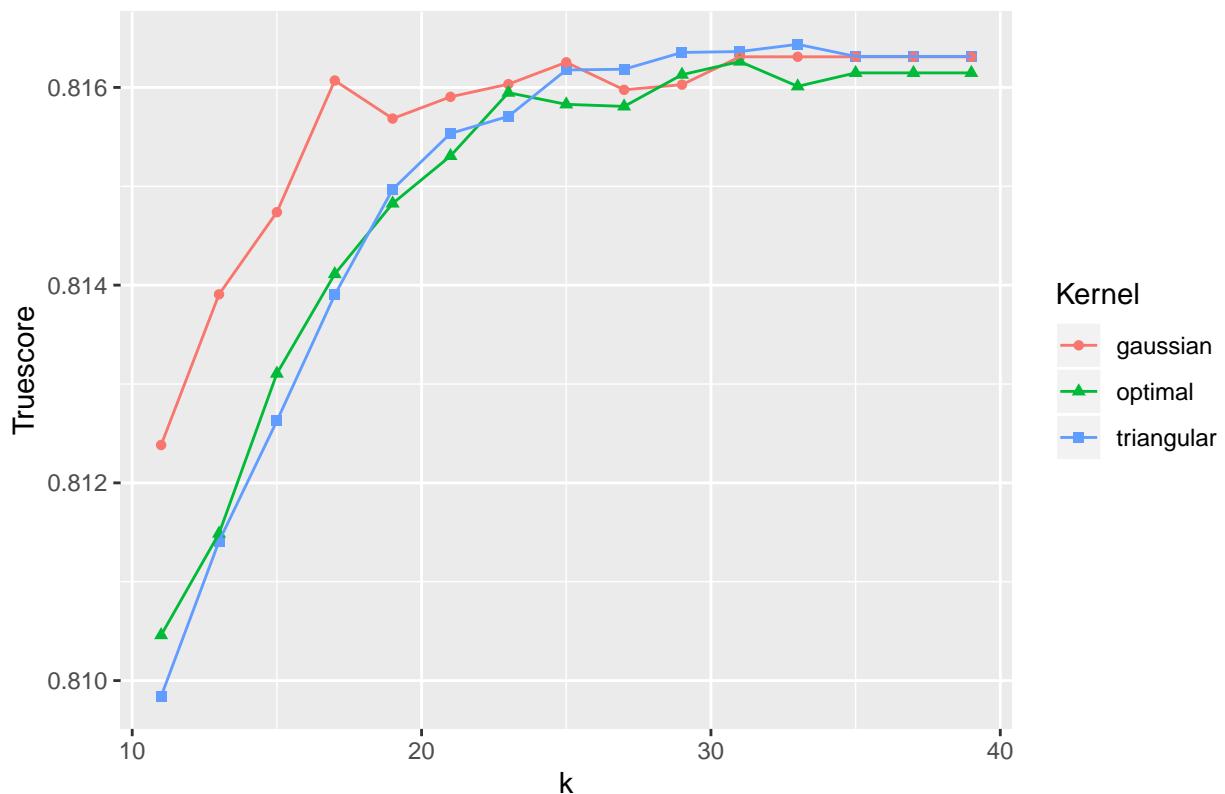
Table 35: c26_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.26	0.839508	0.782199	0.809841
11	gaussian	0.26	0.852259	0.776071	0.812383
11	optimal	0.26	0.838052	0.784627	0.810460
13	triangular	0.26	0.845080	0.780316	0.811408
13	gaussian	0.26	0.856124	0.775658	0.813907
13	optimal	0.26	0.846185	0.779523	0.811487
15	triangular	0.26	0.849548	0.778780	0.812626
15	gaussian	0.26	0.860341	0.773726	0.814738
15	optimal	0.26	0.852058	0.777558	0.813105
17	triangular	0.26	0.854267	0.777178	0.813901
17	gaussian	0.26	0.864307	0.772933	0.816070
17	optimal	0.26	0.855472	0.776567	0.814112
19	triangular	0.26	0.857831	0.776187	0.814969
19	gaussian	0.26	0.865161	0.771562	0.815685
19	optimal	0.26	0.858685	0.775229	0.814826
21	triangular	0.26	0.860793	0.774799	0.815535
21	gaussian	0.26	0.866717	0.770720	0.815905
21	optimal	0.26	0.861345	0.773940	0.815307
23	triangular	0.26	0.862299	0.773891	0.815707
23	gaussian	0.26	0.868223	0.769762	0.816033
23	optimal	0.26	0.864608	0.772471	0.815947

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.26	0.864960	0.772603	0.816177
25	gaussian	0.26	0.869629	0.769052	0.816254
25	optimal	0.26	0.865713	0.771380	0.815829
27	triangular	0.26	0.866014	0.771777	0.816184
27	gaussian	0.26	0.869378	0.768754	0.815976
27	optimal	0.26	0.866918	0.770389	0.815808
29	triangular	0.26	0.867922	0.770571	0.816354
29	gaussian	0.26	0.869729	0.768573	0.816028
29	optimal	0.26	0.868524	0.769696	0.816129
31	triangular	0.26	0.869076	0.769679	0.816363
31	gaussian	0.26	0.870984	0.768094	0.816310
31	optimal	0.26	0.869478	0.769184	0.816262
33	triangular	0.26	0.869980	0.769101	0.816436
33	gaussian	0.26	0.870984	0.768094	0.816310
33	optimal	0.26	0.870030	0.768308	0.816011
35	triangular	0.26	0.870231	0.768688	0.816314
35	gaussian	0.26	0.870984	0.768094	0.816310
35	optimal	0.26	0.870783	0.767962	0.816147
37	triangular	0.26	0.870231	0.768688	0.816314
37	gaussian	0.26	0.870984	0.768094	0.816310
37	optimal	0.26	0.870783	0.767962	0.816147
39	triangular	0.26	0.870231	0.768688	0.816314
39	gaussian	0.26	0.870984	0.768094	0.816310
39	optimal	0.26	0.870783	0.767962	0.816147

```
ggplot(c26_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.26 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.26 (Truescore)

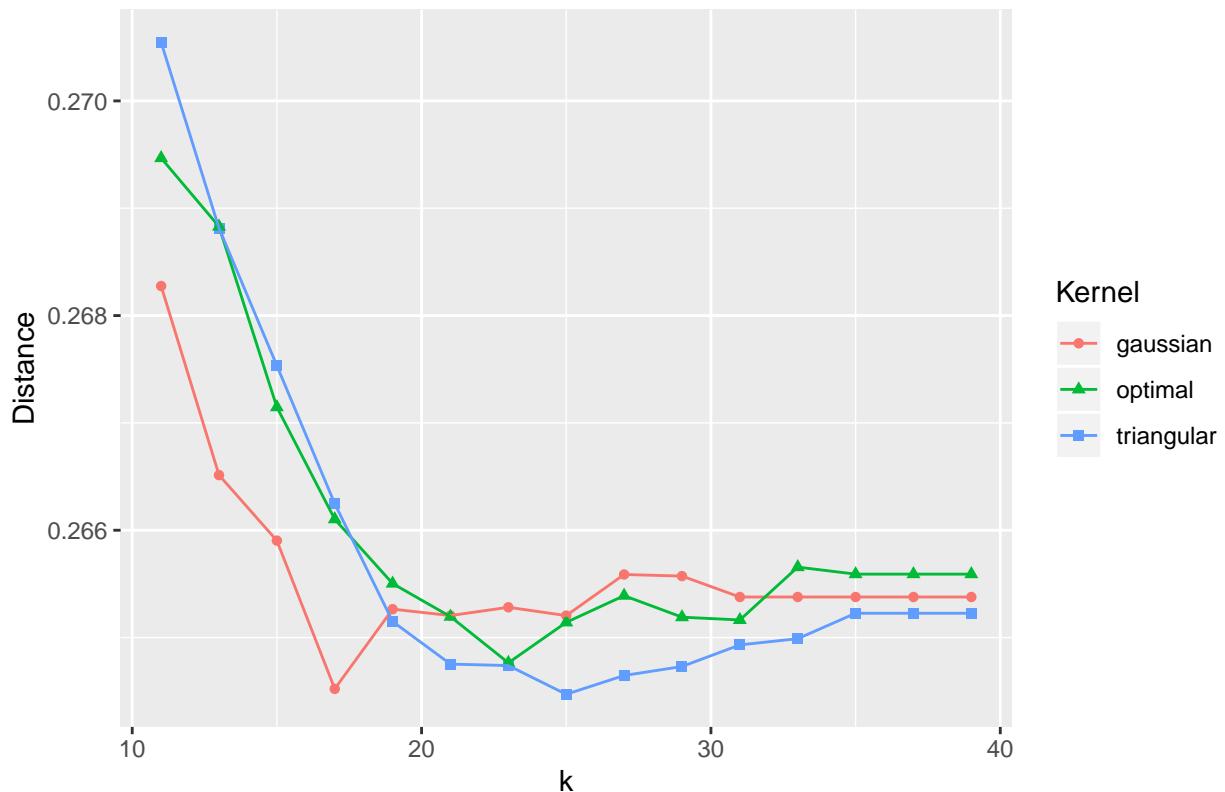


```
# For the Cutoff of 0.26, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c26_results <- c26_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c26_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.26 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.26 (Distance)



```
# For the Cutoff of 0.26, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c26_results$Truescore)
```

```
## [1] 0.816436
```

```
(c26_opt_k_ts <- c26_results$k[which.max(c26_results$Truescore)])
```

```
## [1] 33
```

```
(c26_opt_kernel_ts <- c26_results$Kernel[which.max(c26_results$Truescore)])
```

```
## [1] "triangular"
```

```
(c26_opt_cut_ts <- c26_results$Cut[which.max(c26_results$Truescore)])
```

```
## [1] 0.26
```

```
(c26_opt_tpr_ts <- c26_results$TPR[which.max(c26_results$Truescore)])
```

```
## [1] 0.86998
```

```

(c26_opt_tnr_ts <- c26_results$TNR[which.max(c26_results$Truescore)])

## [1] 0.769101

(c26_opt_d_ts <- c26_results$Distance[which.max(c26_results$Truescore)])

## [1] 0.2649897

min(c26_results$Distance)

## [1] 0.2644715

(c26_opt_k_dist <- c26_results$k[which.min(c26_results$Distance)])

## [1] 25

(c26_opt_kernel_dist <- c26_results$Kernel[which.min(c26_results$Distance)])

## [1] "triangular"

(c26_opt_cut_dist <- c26_results$Cut[which.min(c26_results$Distance)])

## [1] 0.26

(c26_opt_tpr_dist <- c26_results$TPR[which.min(c26_results$Distance)])

## [1] 0.86496

(c26_opt_tnr_dist <- c26_results$TNR[which.min(c26_results$Distance)])

## [1] 0.772603

(c26_opt_t_dist <- c26_results$Truescore[which.min(c26_results$Distance)])

## [1] 0.816177

#####
# 0.27 Cutoff
#####

# For the decision cutoff of 0.27, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c27 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred27, obs))
  confusionMatrix(ss$pred27, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c27) <- kwf_dfs_v

cm_c27_tables <- sapply(cm_c27, "[[", 2)
cm_c27_tables <- as_tibble(cm_c27_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.27.

k11w1c27_tpr <- round(sum(cm_c27_tables[1, 1:10]) /
  (sum(cm_c27_tables[1, 1:10]) + sum(cm_c27_tables[2, 1:10])), 6)
k11w1c27_tnr <- round(sum(cm_c27_tables[4, 1:10]) /
  (sum(cm_c27_tables[4, 1:10]) + sum(cm_c27_tables[3, 1:10])), 6)
k11w1c27_truescore <- round((2 * k11w1c27_tpr * k11w1c27_tnr) /
  (k11w1c27_tpr + k11w1c27_tnr), 6)

k11w2c27_tpr <- round(sum(cm_c27_tables[1, 11:20]) /
  (sum(cm_c27_tables[1, 11:20]) + sum(cm_c27_tables[2, 11:20])), 6)
k11w2c27_tnr <- round(sum(cm_c27_tables[4, 11:20]) /
  (sum(cm_c27_tables[4, 11:20]) + sum(cm_c27_tables[3, 11:20])), 6)
k11w2c27_truescore <- round((2 * k11w2c27_tpr * k11w2c27_tnr) /
  (k11w2c27_tpr + k11w2c27_tnr), 6)

k11w3c27_tpr <- round(sum(cm_c27_tables[1, 21:30]) /
  (sum(cm_c27_tables[1, 21:30]) + sum(cm_c27_tables[2, 21:30])), 6)
k11w3c27_tnr <- round(sum(cm_c27_tables[4, 21:30]) /
  (sum(cm_c27_tables[4, 21:30]) + sum(cm_c27_tables[3, 21:30])), 6)
k11w3c27_truescore <- round((2 * k11w3c27_tpr * k11w3c27_tnr) /
  (k11w3c27_tpr + k11w3c27_tnr), 6)

k13w1c27_tpr <- round(sum(cm_c27_tables[1, 31:40]) /
  (sum(cm_c27_tables[1, 31:40]) + sum(cm_c27_tables[2, 31:40])), 6)
k13w1c27_tnr <- round(sum(cm_c27_tables[4, 31:40]) /
  (sum(cm_c27_tables[4, 31:40]) + sum(cm_c27_tables[3, 31:40])), 6)
k13w1c27_truescore <- round((2 * k13w1c27_tpr * k13w1c27_tnr) /
  (k13w1c27_tpr + k13w1c27_tnr), 6)

k13w2c27_tpr <- round(sum(cm_c27_tables[1, 41:50]) /
  (sum(cm_c27_tables[1, 41:50]) + sum(cm_c27_tables[2, 41:50])), 6)
k13w2c27_tnr <- round(sum(cm_c27_tables[4, 41:50]) /
  (sum(cm_c27_tables[4, 41:50]) + sum(cm_c27_tables[3, 41:50])), 6)
k13w2c27_truescore <- round((2 * k13w2c27_tpr * k13w2c27_tnr) /
  (k13w2c27_tpr + k13w2c27_tnr), 6)

k13w3c27_tpr <- round(sum(cm_c27_tables[1, 51:60]) /
  (sum(cm_c27_tables[1, 51:60]) + sum(cm_c27_tables[2, 51:60])), 6)
k13w3c27_tnr <- round(sum(cm_c27_tables[4, 51:60]) /
  (sum(cm_c27_tables[4, 51:60]) + sum(cm_c27_tables[3, 51:60])), 6)
k13w3c27_truescore <- round((2 * k13w3c27_tpr * k13w3c27_tnr) /

```

```

(k13w3c27_tpr + k13w3c27_tnr), 6)

k15w1c27_tpr <- round(sum(cm_c27_tables[1, 61:70]) /
  (sum(cm_c27_tables[1, 61:70]) + sum(cm_c27_tables[2, 61:70])), 6)
k15w1c27_tnr <- round(sum(cm_c27_tables[4, 61:70]) /
  (sum(cm_c27_tables[4, 61:70]) + sum(cm_c27_tables[3, 61:70])), 6)
k15w1c27_truescore <- round((2 * k15w1c27_tpr * k15w1c27_tnr) /
  (k15w1c27_tpr + k15w1c27_tnr), 6)

k15w2c27_tpr <- round(sum(cm_c27_tables[1, 71:80]) /
  (sum(cm_c27_tables[1, 71:80]) + sum(cm_c27_tables[2, 71:80])), 6)
k15w2c27_tnr <- round(sum(cm_c27_tables[4, 71:80]) /
  (sum(cm_c27_tables[4, 71:80]) + sum(cm_c27_tables[3, 71:80])), 6)
k15w2c27_truescore <- round((2 * k15w2c27_tpr * k15w2c27_tnr) /
  (k15w2c27_tpr + k15w2c27_tnr), 6)

k15w3c27_tpr <- round(sum(cm_c27_tables[1, 81:90]) /
  (sum(cm_c27_tables[1, 81:90]) + sum(cm_c27_tables[2, 81:90])), 6)
k15w3c27_tnr <- round(sum(cm_c27_tables[4, 81:90]) /
  (sum(cm_c27_tables[4, 81:90]) + sum(cm_c27_tables[3, 81:90])), 6)
k15w3c27_truescore <- round((2 * k15w3c27_tpr * k15w3c27_tnr) /
  (k15w3c27_tpr + k15w3c27_tnr), 6)

k17w1c27_tpr <- round(sum(cm_c27_tables[1, 91:100]) /
  (sum(cm_c27_tables[1, 91:100]) + sum(cm_c27_tables[2, 91:100])), 6)
k17w1c27_tnr <- round(sum(cm_c27_tables[4, 91:100]) /
  (sum(cm_c27_tables[4, 91:100]) + sum(cm_c27_tables[3, 91:100])), 6)
k17w1c27_truescore <- round((2 * k17w1c27_tpr * k17w1c27_tnr) /
  (k17w1c27_tpr + k17w1c27_tnr), 6)

k17w2c27_tpr <- round(sum(cm_c27_tables[1, 101:110]) /
  (sum(cm_c27_tables[1, 101:110]) + sum(cm_c27_tables[2, 101:110])), 6)
k17w2c27_tnr <- round(sum(cm_c27_tables[4, 101:110]) /
  (sum(cm_c27_tables[4, 101:110]) + sum(cm_c27_tables[3, 101:110])), 6)
k17w2c27_truescore <- round((2 * k17w2c27_tpr * k17w2c27_tnr) /
  (k17w2c27_tpr + k17w2c27_tnr), 6)

k17w3c27_tpr <- round(sum(cm_c27_tables[1, 111:120]) /
  (sum(cm_c27_tables[1, 111:120]) + sum(cm_c27_tables[2, 111:120])), 6)
k17w3c27_tnr <- round(sum(cm_c27_tables[4, 111:120]) /
  (sum(cm_c27_tables[4, 111:120]) + sum(cm_c27_tables[3, 111:120])), 6)
k17w3c27_truescore <- round((2 * k17w3c27_tpr * k17w3c27_tnr) /
  (k17w3c27_tpr + k17w3c27_tnr), 6)

k19w1c27_tpr <- round(sum(cm_c27_tables[1, 121:130]) /
  (sum(cm_c27_tables[1, 121:130]) + sum(cm_c27_tables[2, 121:130])), 6)
k19w1c27_tnr <- round(sum(cm_c27_tables[4, 121:130]) /
  (sum(cm_c27_tables[4, 121:130]) + sum(cm_c27_tables[3, 121:130])), 6)
k19w1c27_truescore <- round((2 * k19w1c27_tpr * k19w1c27_tnr) /
  (k19w1c27_tpr + k19w1c27_tnr), 6)

```

```

k19w2c27_tpr <- round(sum(cm_c27_tables[1, 131:140]) /
  (sum(cm_c27_tables[1, 131:140]) + sum(cm_c27_tables[2, 131:140])), 6)
k19w2c27_tnr <- round(sum(cm_c27_tables[4, 131:140]) /
  (sum(cm_c27_tables[4, 131:140]) + sum(cm_c27_tables[3, 131:140])), 6)
k19w2c27_truescore <- round((2 * k19w2c27_tpr * k19w2c27_tnr) /
  (k19w2c27_tpr + k19w2c27_tnr), 6)

k19w3c27_tpr <- round(sum(cm_c27_tables[1, 141:150]) /
  (sum(cm_c27_tables[1, 141:150]) + sum(cm_c27_tables[2, 141:150])), 6)
k19w3c27_tnr <- round(sum(cm_c27_tables[4, 141:150]) /
  (sum(cm_c27_tables[4, 141:150]) + sum(cm_c27_tables[3, 141:150])), 6)
k19w3c27_truescore <- round((2 * k19w3c27_tpr * k19w3c27_tnr) /
  (k19w3c27_tpr + k19w3c27_tnr), 6)

k21w1c27_tpr <- round(sum(cm_c27_tables[1, 151:160]) /
  (sum(cm_c27_tables[1, 151:160]) + sum(cm_c27_tables[2, 151:160])), 6)
k21w1c27_tnr <- round(sum(cm_c27_tables[4, 151:160]) /
  (sum(cm_c27_tables[4, 151:160]) + sum(cm_c27_tables[3, 151:160])), 6)
k21w1c27_truescore <- round((2 * k21w1c27_tpr * k21w1c27_tnr) /
  (k21w1c27_tpr + k21w1c27_tnr), 6)

k21w2c27_tpr <- round(sum(cm_c27_tables[1, 161:170]) /
  (sum(cm_c27_tables[1, 161:170]) + sum(cm_c27_tables[2, 161:170])), 6)
k21w2c27_tnr <- round(sum(cm_c27_tables[4, 161:170]) /
  (sum(cm_c27_tables[4, 161:170]) + sum(cm_c27_tables[3, 161:170])), 6)
k21w2c27_truescore <- round((2 * k21w2c27_tpr * k21w2c27_tnr) /
  (k21w2c27_tpr + k21w2c27_tnr), 6)

k21w3c27_tpr <- round(sum(cm_c27_tables[1, 171:180]) /
  (sum(cm_c27_tables[1, 171:180]) + sum(cm_c27_tables[2, 171:180])), 6)
k21w3c27_tnr <- round(sum(cm_c27_tables[4, 171:180]) /
  (sum(cm_c27_tables[4, 171:180]) + sum(cm_c27_tables[3, 171:180])), 6)
k21w3c27_truescore <- round((2 * k21w3c27_tpr * k21w3c27_tnr) /
  (k21w3c27_tpr + k21w3c27_tnr), 6)

k23w1c27_tpr <- round(sum(cm_c27_tables[1, 181:190]) /
  (sum(cm_c27_tables[1, 181:190]) + sum(cm_c27_tables[2, 181:190])), 6)
k23w1c27_tnr <- round(sum(cm_c27_tables[4, 181:190]) /
  (sum(cm_c27_tables[4, 181:190]) + sum(cm_c27_tables[3, 181:190])), 6)
k23w1c27_truescore <- round((2 * k23w1c27_tpr * k23w1c27_tnr) /
  (k23w1c27_tpr + k23w1c27_tnr), 6)

k23w2c27_tpr <- round(sum(cm_c27_tables[1, 191:200]) /
  (sum(cm_c27_tables[1, 191:200]) + sum(cm_c27_tables[2, 191:200])), 6)
k23w2c27_tnr <- round(sum(cm_c27_tables[4, 191:200]) /
  (sum(cm_c27_tables[4, 191:200]) + sum(cm_c27_tables[3, 191:200])), 6)
k23w2c27_truescore <- round((2 * k23w2c27_tpr * k23w2c27_tnr) /
  (k23w2c27_tpr + k23w2c27_tnr), 6)

k23w3c27_tpr <- round(sum(cm_c27_tables[1, 201:210]) /
  (sum(cm_c27_tables[1, 201:210]) + sum(cm_c27_tables[2, 201:210])), 6)

```

```

k23w3c27_tnr <- round(sum(cm_c27_tables[4, 201:210]) /
  (sum(cm_c27_tables[4, 201:210]) + sum(cm_c27_tables[3, 201:210])), 6)
k23w3c27_truescore <- round((2 * k23w3c27_tpr * k23w3c27_tnr) /
  (k23w3c27_tpr + k23w3c27_tnr), 6)

k25w1c27_tpr <- round(sum(cm_c27_tables[1, 211:220]) /
  (sum(cm_c27_tables[1, 211:220]) + sum(cm_c27_tables[2, 211:220])), 6)
k25w1c27_tnr <- round(sum(cm_c27_tables[4, 211:220]) /
  (sum(cm_c27_tables[4, 211:220]) + sum(cm_c27_tables[3, 211:220])), 6)
k25w1c27_truescore <- round((2 * k25w1c27_tpr * k25w1c27_tnr) /
  (k25w1c27_tpr + k25w1c27_tnr), 6)

k25w2c27_tpr <- round(sum(cm_c27_tables[1, 221:230]) /
  (sum(cm_c27_tables[1, 221:230]) + sum(cm_c27_tables[2, 221:230])), 6)
k25w2c27_tnr <- round(sum(cm_c27_tables[4, 221:230]) /
  (sum(cm_c27_tables[4, 221:230]) + sum(cm_c27_tables[3, 221:230])), 6)
k25w2c27_truescore <- round((2 * k25w2c27_tpr * k25w2c27_tnr) /
  (k25w2c27_tpr + k25w2c27_tnr), 6)

k25w3c27_tpr <- round(sum(cm_c27_tables[1, 231:240]) /
  (sum(cm_c27_tables[1, 231:240]) + sum(cm_c27_tables[2, 231:240])), 6)
k25w3c27_tnr <- round(sum(cm_c27_tables[4, 231:240]) /
  (sum(cm_c27_tables[4, 231:240]) + sum(cm_c27_tables[3, 231:240])), 6)
k25w3c27_truescore <- round((2 * k25w3c27_tpr * k25w3c27_tnr) /
  (k25w3c27_tpr + k25w3c27_tnr), 6)

k27w1c27_tpr <- round(sum(cm_c27_tables[1, 241:250]) /
  (sum(cm_c27_tables[1, 241:250]) + sum(cm_c27_tables[2, 241:250])), 6)
k27w1c27_tnr <- round(sum(cm_c27_tables[4, 241:250]) /
  (sum(cm_c27_tables[4, 241:250]) + sum(cm_c27_tables[3, 241:250])), 6)
k27w1c27_truescore <- round((2 * k27w1c27_tpr * k27w1c27_tnr) /
  (k27w1c27_tpr + k27w1c27_tnr), 6)

k27w2c27_tpr <- round(sum(cm_c27_tables[1, 251:260]) /
  (sum(cm_c27_tables[1, 251:260]) + sum(cm_c27_tables[2, 251:260])), 6)
k27w2c27_tnr <- round(sum(cm_c27_tables[4, 251:260]) /
  (sum(cm_c27_tables[4, 251:260]) + sum(cm_c27_tables[3, 251:260])), 6)
k27w2c27_truescore <- round((2 * k27w2c27_tpr * k27w2c27_tnr) /
  (k27w2c27_tpr + k27w2c27_tnr), 6)

k27w3c27_tpr <- round(sum(cm_c27_tables[1, 261:270]) /
  (sum(cm_c27_tables[1, 261:270]) + sum(cm_c27_tables[2, 261:270])), 6)
k27w3c27_tnr <- round(sum(cm_c27_tables[4, 261:270]) /
  (sum(cm_c27_tables[4, 261:270]) + sum(cm_c27_tables[3, 261:270])), 6)
k27w3c27_truescore <- round((2 * k27w3c27_tpr * k27w3c27_tnr) /
  (k27w3c27_tpr + k27w3c27_tnr), 6)

k29w1c27_tpr <- round(sum(cm_c27_tables[1, 271:280]) /
  (sum(cm_c27_tables[1, 271:280]) + sum(cm_c27_tables[2, 271:280])), 6)
k29w1c27_tnr <- round(sum(cm_c27_tables[4, 271:280]) /

```

```

(sum(cm_c27_tables[4, 271:280]) + sum(cm_c27_tables[3, 271:280])), 6)
k29w1c27_truescore <- round((2 * k29w1c27_tpr * k29w1c27_tnr) /
(k29w1c27_tpr + k29w1c27_tnr), 6)

k29w2c27_tpr <- round(sum(cm_c27_tables[1, 281:290]) /
(sum(cm_c27_tables[1, 281:290]) + sum(cm_c27_tables[2, 281:290])), 6)
k29w2c27_tnr <- round(sum(cm_c27_tables[4, 281:290]) /
(sum(cm_c27_tables[4, 281:290]) + sum(cm_c27_tables[3, 281:290])), 6)
k29w2c27_truescore <- round((2 * k29w2c27_tpr * k29w2c27_tnr) /
(k29w2c27_tpr + k29w2c27_tnr), 6)

k29w3c27_tpr <- round(sum(cm_c27_tables[1, 291:300]) /
(sum(cm_c27_tables[1, 291:300]) + sum(cm_c27_tables[2, 291:300])), 6)
k29w3c27_tnr <- round(sum(cm_c27_tables[4, 291:300]) /
(sum(cm_c27_tables[4, 291:300]) + sum(cm_c27_tables[3, 291:300])), 6)
k29w3c27_truescore <- round((2 * k29w3c27_tpr * k29w3c27_tnr) /
(k29w3c27_tpr + k29w3c27_tnr), 6)

k31w1c27_tpr <- round(sum(cm_c27_tables[1, 301:310]) /
(sum(cm_c27_tables[1, 301:310]) + sum(cm_c27_tables[2, 301:310])), 6)
k31w1c27_tnr <- round(sum(cm_c27_tables[4, 301:310]) /
(sum(cm_c27_tables[4, 301:310]) + sum(cm_c27_tables[3, 301:310])), 6)
k31w1c27_truescore <- round((2 * k31w1c27_tpr * k31w1c27_tnr) /
(k31w1c27_tpr + k31w1c27_tnr), 6)

k31w2c27_tpr <- round(sum(cm_c27_tables[1, 311:320]) /
(sum(cm_c27_tables[1, 311:320]) + sum(cm_c27_tables[2, 311:320])), 6)
k31w2c27_tnr <- round(sum(cm_c27_tables[4, 311:320]) /
(sum(cm_c27_tables[4, 311:320]) + sum(cm_c27_tables[3, 311:320])), 6)
k31w2c27_truescore <- round((2 * k31w2c27_tpr * k31w2c27_tnr) /
(k31w2c27_tpr + k31w2c27_tnr), 6)

k31w3c27_tpr <- round(sum(cm_c27_tables[1, 321:330]) /
(sum(cm_c27_tables[1, 321:330]) + sum(cm_c27_tables[2, 321:330])), 6)
k31w3c27_tnr <- round(sum(cm_c27_tables[4, 321:330]) /
(sum(cm_c27_tables[4, 321:330]) + sum(cm_c27_tables[3, 321:330])), 6)
k31w3c27_truescore <- round((2 * k31w3c27_tpr * k31w3c27_tnr) /
(k31w3c27_tpr + k31w3c27_tnr), 6)

k33w1c27_tpr <- round(sum(cm_c27_tables[1, 331:340]) /
(sum(cm_c27_tables[1, 331:340]) + sum(cm_c27_tables[2, 331:340])), 6)
k33w1c27_tnr <- round(sum(cm_c27_tables[4, 331:340]) /
(sum(cm_c27_tables[4, 331:340]) + sum(cm_c27_tables[3, 331:340])), 6)
k33w1c27_truescore <- round((2 * k33w1c27_tpr * k33w1c27_tnr) /
(k33w1c27_tpr + k33w1c27_tnr), 6)

k33w2c27_tpr <- round(sum(cm_c27_tables[1, 341:350]) /
(sum(cm_c27_tables[1, 341:350]) + sum(cm_c27_tables[2, 341:350])), 6)
k33w2c27_tnr <- round(sum(cm_c27_tables[4, 341:350]) /
(sum(cm_c27_tables[4, 341:350]) + sum(cm_c27_tables[3, 341:350])), 6)
k33w2c27_truescore <- round((2 * k33w2c27_tpr * k33w2c27_tnr) /

```

```

(k33w2c27_tpr + k33w2c27_tnr), 6)

k33w3c27_tpr <- round(sum(cm_c27_tables[1, 351:360]) /
  (sum(cm_c27_tables[1, 351:360]) + sum(cm_c27_tables[2, 351:360])), 6)
k33w3c27_tnr <- round(sum(cm_c27_tables[4, 351:360]) /
  (sum(cm_c27_tables[4, 351:360]) + sum(cm_c27_tables[3, 351:360])), 6)
k33w3c27_truescore <- round((2 * k33w3c27_tpr * k33w3c27_tnr) /
  (k33w3c27_tpr + k33w3c27_tnr), 6)

k35w1c27_tpr <- round(sum(cm_c27_tables[1, 361:370]) /
  (sum(cm_c27_tables[1, 361:370]) + sum(cm_c27_tables[2, 361:370])), 6)
k35w1c27_tnr <- round(sum(cm_c27_tables[4, 361:370]) /
  (sum(cm_c27_tables[4, 361:370]) + sum(cm_c27_tables[3, 361:370])), 6)
k35w1c27_truescore <- round((2 * k35w1c27_tpr * k35w1c27_tnr) /
  (k35w1c27_tpr + k35w1c27_tnr), 6)

k35w2c27_tpr <- round(sum(cm_c27_tables[1, 371:380]) /
  (sum(cm_c27_tables[1, 371:380]) + sum(cm_c27_tables[2, 371:380])), 6)
k35w2c27_tnr <- round(sum(cm_c27_tables[4, 371:380]) /
  (sum(cm_c27_tables[4, 371:380]) + sum(cm_c27_tables[3, 371:380])), 6)
k35w2c27_truescore <- round((2 * k35w2c27_tpr * k35w2c27_tnr) /
  (k35w2c27_tpr + k35w2c27_tnr), 6)

k35w3c27_tpr <- round(sum(cm_c27_tables[1, 381:390]) /
  (sum(cm_c27_tables[1, 381:390]) + sum(cm_c27_tables[2, 381:390])), 6)
k35w3c27_tnr <- round(sum(cm_c27_tables[4, 381:390]) /
  (sum(cm_c27_tables[4, 381:390]) + sum(cm_c27_tables[3, 381:390])), 6)
k35w3c27_truescore <- round((2 * k35w3c27_tpr * k35w3c27_tnr) /
  (k35w3c27_tpr + k35w3c27_tnr), 6)

k37w1c27_tpr <- round(sum(cm_c27_tables[1, 391:400]) /
  (sum(cm_c27_tables[1, 391:400]) + sum(cm_c27_tables[2, 391:400])), 6)
k37w1c27_tnr <- round(sum(cm_c27_tables[4, 391:400]) /
  (sum(cm_c27_tables[4, 391:400]) + sum(cm_c27_tables[3, 391:400])), 6)
k37w1c27_truescore <- round((2 * k37w1c27_tpr * k37w1c27_tnr) /
  (k37w1c27_tpr + k37w1c27_tnr), 6)

k37w2c27_tpr <- round(sum(cm_c27_tables[1, 401:410]) /
  (sum(cm_c27_tables[1, 401:410]) + sum(cm_c27_tables[2, 401:410])), 6)
k37w2c27_tnr <- round(sum(cm_c27_tables[4, 401:410]) /
  (sum(cm_c27_tables[4, 401:410]) + sum(cm_c27_tables[3, 401:410])), 6)
k37w2c27_truescore <- round((2 * k37w2c27_tpr * k37w2c27_tnr) /
  (k37w2c27_tpr + k37w2c27_tnr), 6)

k37w3c27_tpr <- round(sum(cm_c27_tables[1, 411:420]) /
  (sum(cm_c27_tables[1, 411:420]) + sum(cm_c27_tables[2, 411:420])), 6)
k37w3c27_tnr <- round(sum(cm_c27_tables[4, 411:420]) /
  (sum(cm_c27_tables[4, 411:420]) + sum(cm_c27_tables[3, 411:420])), 6)
k37w3c27_truescore <- round((2 * k37w3c27_tpr * k37w3c27_tnr) /
  (k37w3c27_tpr + k37w3c27_tnr), 6)

```

```

k39w1c27_tpr <- round(sum(cm_c27_tables[1, 421:430]) /
  (sum(cm_c27_tables[1, 421:430]) + sum(cm_c27_tables[2, 421:430])), 6)
k39w1c27_tnr <- round(sum(cm_c27_tables[4, 421:430]) /
  (sum(cm_c27_tables[4, 421:430]) + sum(cm_c27_tables[3, 421:430])), 6)
k39w1c27_truescore <- round((2 * k39w1c27_tpr * k39w1c27_tnr) /
  (k39w1c27_tpr + k39w1c27_tnr), 6)

k39w2c27_tpr <- round(sum(cm_c27_tables[1, 431:440]) /
  (sum(cm_c27_tables[1, 431:440]) + sum(cm_c27_tables[2, 431:440])), 6)
k39w2c27_tnr <- round(sum(cm_c27_tables[4, 431:440]) /
  (sum(cm_c27_tables[4, 431:440]) + sum(cm_c27_tables[3, 431:440])), 6)
k39w2c27_truescore <- round((2 * k39w2c27_tpr * k39w2c27_tnr) /
  (k39w2c27_tpr + k39w2c27_tnr), 6)

k39w3c27_tpr <- round(sum(cm_c27_tables[1, 441:450]) /
  (sum(cm_c27_tables[1, 441:450]) + sum(cm_c27_tables[2, 441:450])), 6)
k39w3c27_tnr <- round(sum(cm_c27_tables[4, 441:450]) /
  (sum(cm_c27_tables[4, 441:450]) + sum(cm_c27_tables[3, 441:450])), 6)
k39w3c27_truescore <- round((2 * k39w3c27_tpr * k39w3c27_tnr) /
  (k39w3c27_tpr + k39w3c27_tnr), 6)

# Compile the 0.27 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c27_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.27,
  TPR = c(k11w1c27_tpr, k11w2c27_tpr, k11w3c27_tpr, k13w1c27_tpr,
    k13w2c27_tpr, k13w3c27_tpr, k15w1c27_tpr, k15w2c27_tpr,
    k15w3c27_tpr, k17w1c27_tpr, k17w2c27_tpr, k17w3c27_tpr,
    k19w1c27_tpr, k19w2c27_tpr, k19w3c27_tpr, k21w1c27_tpr,
    k21w2c27_tpr, k21w3c27_tpr, k23w1c27_tpr, k23w2c27_tpr,
    k23w3c27_tpr, k25w1c27_tpr, k25w2c27_tpr, k25w3c27_tpr,
    k27w1c27_tpr, k27w2c27_tpr, k27w3c27_tpr, k29w1c27_tpr,
    k29w2c27_tpr, k29w3c27_tpr, k31w1c27_tpr, k31w2c27_tpr,
    k31w3c27_tpr, k33w1c27_tpr, k33w2c27_tpr, k33w3c27_tpr,
    k35w1c27_tpr, k35w2c27_tpr, k35w3c27_tpr, k37w1c27_tpr,
    k37w2c27_tpr, k37w3c27_tpr, k39w1c27_tpr, k39w2c27_tpr,
    k39w3c27_tpr),
  TNR = c(k11w1c27_tnr, k11w2c27_tnr, k11w3c27_tnr, k13w1c27_tnr,
    k13w2c27_tnr, k13w3c27_tnr, k15w1c27_tnr, k15w2c27_tnr,
    k15w3c27_tnr, k17w1c27_tnr, k17w2c27_tnr, k17w3c27_tnr,
    k19w1c27_tnr, k19w2c27_tnr, k19w3c27_tnr, k21w1c27_tnr,
    k21w2c27_tnr, k21w3c27_tnr, k23w1c27_tnr, k23w2c27_tnr,
    k23w3c27_tnr, k25w1c27_tnr, k25w2c27_tnr, k25w3c27_tnr,
    k27w1c27_tnr, k27w2c27_tnr, k27w3c27_tnr, k29w1c27_tnr,
    k29w2c27_tnr, k29w3c27_tnr, k31w1c27_tnr, k31w2c27_tnr,
    k31w3c27_tnr, k33w1c27_tnr, k33w2c27_tnr, k33w3c27_tnr)

```

```

k35w1c27_tnr, k35w2c27_tnr, k35w3c27_tnr, k37w1c27_tnr,
k37w2c27_tnr, k37w3c27_tnr, k39w1c27_tnr, k39w2c27_tnr,
k39w3c27_tnr),
Truescore = c(k11w1c27_truescore, k11w2c27_truescore,
k11w3c27_truescore, k13w1c27_truescore,
k13w2c27_truescore, k13w3c27_truescore,
k15w1c27_truescore, k15w2c27_truescore,
k15w3c27_truescore, k17w1c27_truescore,
k17w2c27_truescore, k17w3c27_truescore,
k19w1c27_truescore, k19w2c27_truescore,
k19w3c27_truescore, k21w1c27_truescore,
k21w2c27_truescore, k21w3c27_truescore,
k23w1c27_truescore, k23w2c27_truescore,
k23w3c27_truescore, k25w1c27_truescore,
k25w2c27_truescore, k25w3c27_truescore,
k27w1c27_truescore, k27w2c27_truescore,
k27w3c27_truescore, k29w1c27_truescore,
k29w2c27_truescore, k29w3c27_truescore,
k31w1c27_truescore, k31w2c27_truescore,
k31w3c27_truescore, k33w1c27_truescore,
k33w2c27_truescore, k33w3c27_truescore,
k35w1c27_truescore, k35w2c27_truescore,
k35w3c27_truescore, k37w1c27_truescore,
k37w2c27_truescore, k37w3c27_truescore,
k39w1c27_truescore, k39w2c27_truescore,
k39w3c27_truescore))

```

```

knitr::kable(c27_results[1:45, ], caption = "c27_results")

```

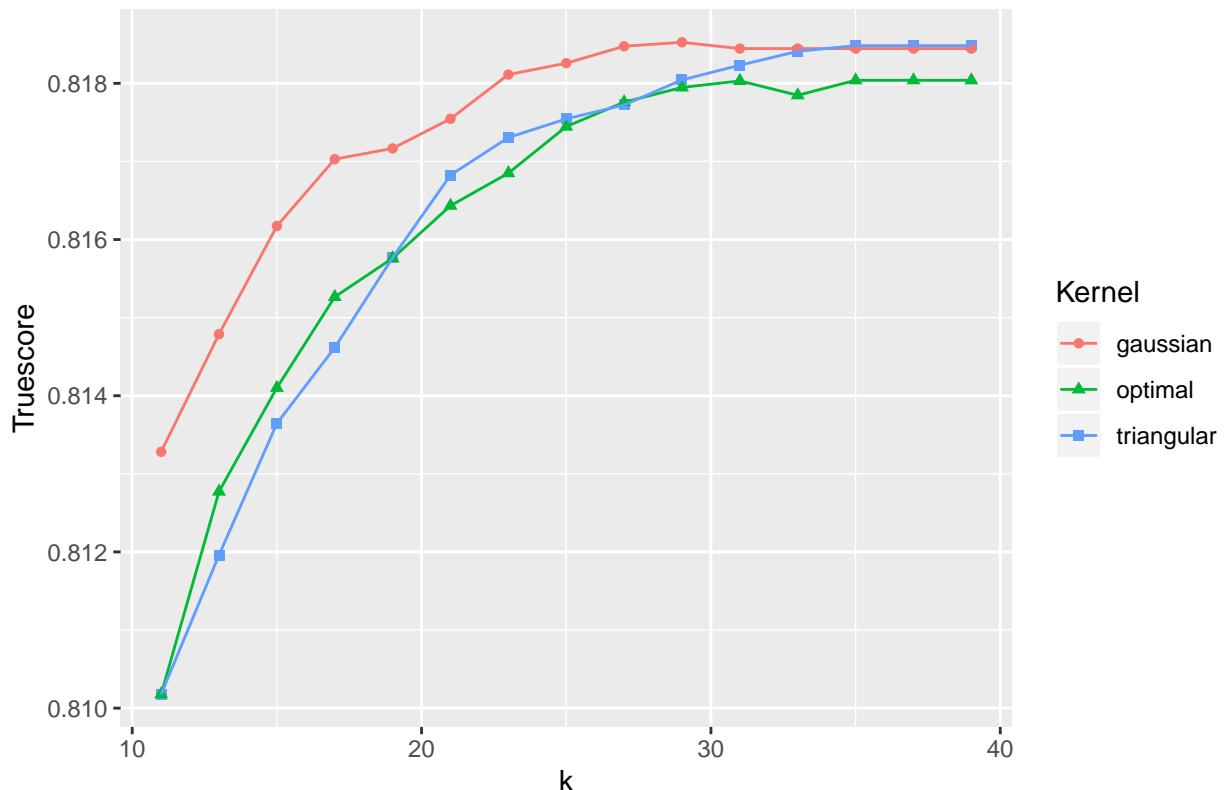
Table 36: c27_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.27	0.832279	0.789218	0.810177
11	gaussian	0.27	0.845030	0.783834	0.813282
11	optimal	0.27	0.832028	0.789449	0.810179
13	triangular	0.27	0.837651	0.787798	0.811960
13	gaussian	0.27	0.848946	0.783272	0.814788
13	optimal	0.27	0.839608	0.787600	0.812773
15	triangular	0.27	0.842871	0.786377	0.813645
15	gaussian	0.27	0.853665	0.781835	0.816173
15	optimal	0.27	0.844327	0.785964	0.814101
17	triangular	0.27	0.846988	0.784627	0.814616
17	gaussian	0.27	0.857129	0.780514	0.817029
17	optimal	0.27	0.848645	0.784412	0.815265
19	triangular	0.27	0.850552	0.783718	0.815768
19	gaussian	0.27	0.858032	0.780018	0.817167
19	optimal	0.27	0.851355	0.783024	0.815761
21	triangular	0.27	0.854167	0.782612	0.816825
21	gaussian	0.27	0.859689	0.779341	0.817546
21	optimal	0.27	0.854116	0.781934	0.816433
23	triangular	0.27	0.856426	0.781604	0.817306
23	gaussian	0.27	0.861345	0.779011	0.818112
23	optimal	0.27	0.856376	0.780811	0.816850

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.27	0.858384	0.780415	0.817545
25	gaussian	0.27	0.862500	0.778334	0.818258
25	optimal	0.27	0.858986	0.779738	0.817446
27	triangular	0.27	0.859639	0.779705	0.817723
27	gaussian	0.27	0.863002	0.778317	0.818475
27	optimal	0.27	0.859940	0.779523	0.817759
29	triangular	0.27	0.861195	0.779011	0.818044
29	gaussian	0.27	0.863052	0.778367	0.818525
29	optimal	0.27	0.861345	0.778714	0.817948
31	triangular	0.27	0.862400	0.778367	0.818232
31	gaussian	0.27	0.864157	0.777326	0.818445
31	optimal	0.27	0.862299	0.778086	0.818031
33	triangular	0.27	0.863102	0.778119	0.818410
33	gaussian	0.27	0.864157	0.777326	0.818445
33	optimal	0.27	0.862600	0.777508	0.817847
35	triangular	0.27	0.863855	0.777640	0.818483
35	gaussian	0.27	0.864157	0.777326	0.818445
35	optimal	0.27	0.863353	0.777244	0.818039
37	triangular	0.27	0.863855	0.777640	0.818483
37	gaussian	0.27	0.864157	0.777326	0.818445
37	optimal	0.27	0.863353	0.777244	0.818039
39	triangular	0.27	0.863855	0.777640	0.818483
39	gaussian	0.27	0.864157	0.777326	0.818445
39	optimal	0.27	0.863353	0.777244	0.818039

```
ggplot(c27_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.27 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.27 (Truescore)

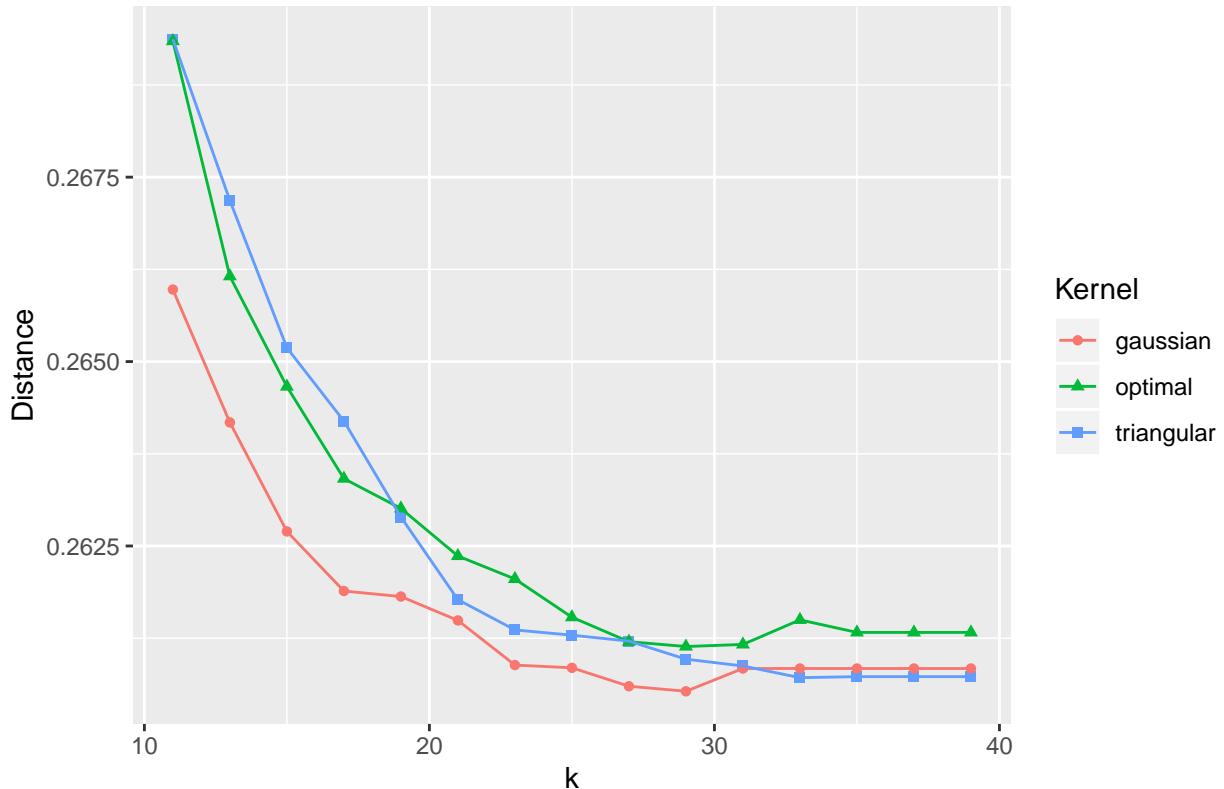


```
# For the Cutoff of 0.27, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c27_results <- c27_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c27_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.27 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.27 (Distance)



```
# For the Cutoff of 0.27, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c27_results$Truescore)
```

```
## [1] 0.818525
```

```
(c27_opt_k_ts <- c27_results$k[which.max(c27_results$Truescore)])
```

```
## [1] 29
```

```
(c27_opt_kernel_ts <- c27_results$Kernel[which.max(c27_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c27_opt_cut_ts <- c27_results$Cut[which.max(c27_results$Truescore)])
```

```
## [1] 0.27
```

```
(c27_opt_tpr_ts <- c27_results$TPR[which.max(c27_results$Truescore)])
```

```
## [1] 0.863052
```

```

(c27_opt_tnr_ts <- c27_results$TNR[which.max(c27_results$Truescore)])

## [1] 0.778367

(c27_opt_d_ts <- c27_results$Distance[which.max(c27_results$Truescore)])

## [1] 0.2605301

min(c27_results$Distance)

## [1] 0.2605301

(c27_opt_k_dist <- c27_results$k[which.min(c27_results$Distance)])

## [1] 29

(c27_opt_kernel_dist <- c27_results$Kernel[which.min(c27_results$Distance)])

## [1] "gaussian"

(c27_opt_cut_dist <- c27_results$Cut[which.min(c27_results$Distance)])

## [1] 0.27

(c27_opt_tpr_dist <- c27_results$TPR[which.min(c27_results$Distance)])

## [1] 0.863052

(c27_opt_tnr_dist <- c27_results$TNR[which.min(c27_results$Distance)])

## [1] 0.778367

(c27_opt_t_dist <- c27_results$Truescore[which.min(c27_results$Distance)])

## [1] 0.818525

#####
# 0.28 Cutoff
#####

# For the decision cutoff of 0.28, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c28 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred28, obs))
  confusionMatrix(ss$pred28, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c28) <- kwf_dfs_v

cm_c28_tables <- sapply(cm_c28, "[[", 2)
cm_c28_tables <- as_tibble(cm_c28_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.28.

k11w1c28_tpr <- round(sum(cm_c28_tables[1, 1:10]) /
  (sum(cm_c28_tables[1, 1:10]) + sum(cm_c28_tables[2, 1:10])), 6)
k11w1c28_tnr <- round(sum(cm_c28_tables[4, 1:10]) /
  (sum(cm_c28_tables[4, 1:10]) + sum(cm_c28_tables[3, 1:10])), 6)
k11w1c28_truescore <- round((2 * k11w1c28_tpr * k11w1c28_tnr) /
  (k11w1c28_tpr + k11w1c28_tnr), 6)

k11w2c28_tpr <- round(sum(cm_c28_tables[1, 11:20]) /
  (sum(cm_c28_tables[1, 11:20]) + sum(cm_c28_tables[2, 11:20])), 6)
k11w2c28_tnr <- round(sum(cm_c28_tables[4, 11:20]) /
  (sum(cm_c28_tables[4, 11:20]) + sum(cm_c28_tables[3, 11:20])), 6)
k11w2c28_truescore <- round((2 * k11w2c28_tpr * k11w2c28_tnr) /
  (k11w2c28_tpr + k11w2c28_tnr), 6)

k11w3c28_tpr <- round(sum(cm_c28_tables[1, 21:30]) /
  (sum(cm_c28_tables[1, 21:30]) + sum(cm_c28_tables[2, 21:30])), 6)
k11w3c28_tnr <- round(sum(cm_c28_tables[4, 21:30]) /
  (sum(cm_c28_tables[4, 21:30]) + sum(cm_c28_tables[3, 21:30])), 6)
k11w3c28_truescore <- round((2 * k11w3c28_tpr * k11w3c28_tnr) /
  (k11w3c28_tpr + k11w3c28_tnr), 6)

k13w1c28_tpr <- round(sum(cm_c28_tables[1, 31:40]) /
  (sum(cm_c28_tables[1, 31:40]) + sum(cm_c28_tables[2, 31:40])), 6)
k13w1c28_tnr <- round(sum(cm_c28_tables[4, 31:40]) /
  (sum(cm_c28_tables[4, 31:40]) + sum(cm_c28_tables[3, 31:40])), 6)
k13w1c28_truescore <- round((2 * k13w1c28_tpr * k13w1c28_tnr) /
  (k13w1c28_tpr + k13w1c28_tnr), 6)

k13w2c28_tpr <- round(sum(cm_c28_tables[1, 41:50]) /
  (sum(cm_c28_tables[1, 41:50]) + sum(cm_c28_tables[2, 41:50])), 6)
k13w2c28_tnr <- round(sum(cm_c28_tables[4, 41:50]) /
  (sum(cm_c28_tables[4, 41:50]) + sum(cm_c28_tables[3, 41:50])), 6)
k13w2c28_truescore <- round((2 * k13w2c28_tpr * k13w2c28_tnr) /
  (k13w2c28_tpr + k13w2c28_tnr), 6)

k13w3c28_tpr <- round(sum(cm_c28_tables[1, 51:60]) /
  (sum(cm_c28_tables[1, 51:60]) + sum(cm_c28_tables[2, 51:60])), 6)
k13w3c28_tnr <- round(sum(cm_c28_tables[4, 51:60]) /
  (sum(cm_c28_tables[4, 51:60]) + sum(cm_c28_tables[3, 51:60])), 6)
k13w3c28_truescore <- round((2 * k13w3c28_tpr * k13w3c28_tnr) /

```

```

(k13w3c28_tpr + k13w3c28_tnr), 6)

k15w1c28_tpr <- round(sum(cm_c28_tables[1, 61:70]) /
  (sum(cm_c28_tables[1, 61:70]) + sum(cm_c28_tables[2, 61:70])), 6)
k15w1c28_tnr <- round(sum(cm_c28_tables[4, 61:70]) /
  (sum(cm_c28_tables[4, 61:70]) + sum(cm_c28_tables[3, 61:70])), 6)
k15w1c28_truescore <- round((2 * k15w1c28_tpr * k15w1c28_tnr) /
  (k15w1c28_tpr + k15w1c28_tnr), 6)

k15w2c28_tpr <- round(sum(cm_c28_tables[1, 71:80]) /
  (sum(cm_c28_tables[1, 71:80]) + sum(cm_c28_tables[2, 71:80])), 6)
k15w2c28_tnr <- round(sum(cm_c28_tables[4, 71:80]) /
  (sum(cm_c28_tables[4, 71:80]) + sum(cm_c28_tables[3, 71:80])), 6)
k15w2c28_truescore <- round((2 * k15w2c28_tpr * k15w2c28_tnr) /
  (k15w2c28_tpr + k15w2c28_tnr), 6)

k15w3c28_tpr <- round(sum(cm_c28_tables[1, 81:90]) /
  (sum(cm_c28_tables[1, 81:90]) + sum(cm_c28_tables[2, 81:90])), 6)
k15w3c28_tnr <- round(sum(cm_c28_tables[4, 81:90]) /
  (sum(cm_c28_tables[4, 81:90]) + sum(cm_c28_tables[3, 81:90])), 6)
k15w3c28_truescore <- round((2 * k15w3c28_tpr * k15w3c28_tnr) /
  (k15w3c28_tpr + k15w3c28_tnr), 6)

k17w1c28_tpr <- round(sum(cm_c28_tables[1, 91:100]) /
  (sum(cm_c28_tables[1, 91:100]) + sum(cm_c28_tables[2, 91:100])), 6)
k17w1c28_tnr <- round(sum(cm_c28_tables[4, 91:100]) /
  (sum(cm_c28_tables[4, 91:100]) + sum(cm_c28_tables[3, 91:100])), 6)
k17w1c28_truescore <- round((2 * k17w1c28_tpr * k17w1c28_tnr) /
  (k17w1c28_tpr + k17w1c28_tnr), 6)

k17w2c28_tpr <- round(sum(cm_c28_tables[1, 101:110]) /
  (sum(cm_c28_tables[1, 101:110]) + sum(cm_c28_tables[2, 101:110])), 6)
k17w2c28_tnr <- round(sum(cm_c28_tables[4, 101:110]) /
  (sum(cm_c28_tables[4, 101:110]) + sum(cm_c28_tables[3, 101:110])), 6)
k17w2c28_truescore <- round((2 * k17w2c28_tpr * k17w2c28_tnr) /
  (k17w2c28_tpr + k17w2c28_tnr), 6)

k17w3c28_tpr <- round(sum(cm_c28_tables[1, 111:120]) /
  (sum(cm_c28_tables[1, 111:120]) + sum(cm_c28_tables[2, 111:120])), 6)
k17w3c28_tnr <- round(sum(cm_c28_tables[4, 111:120]) /
  (sum(cm_c28_tables[4, 111:120]) + sum(cm_c28_tables[3, 111:120])), 6)
k17w3c28_truescore <- round((2 * k17w3c28_tpr * k17w3c28_tnr) /
  (k17w3c28_tpr + k17w3c28_tnr), 6)

k19w1c28_tpr <- round(sum(cm_c28_tables[1, 121:130]) /
  (sum(cm_c28_tables[1, 121:130]) + sum(cm_c28_tables[2, 121:130])), 6)
k19w1c28_tnr <- round(sum(cm_c28_tables[4, 121:130]) /
  (sum(cm_c28_tables[4, 121:130]) + sum(cm_c28_tables[3, 121:130])), 6)
k19w1c28_truescore <- round((2 * k19w1c28_tpr * k19w1c28_tnr) /
  (k19w1c28_tpr + k19w1c28_tnr), 6)

```

```

k19w2c28_tpr <- round(sum(cm_c28_tables[1, 131:140]) /
  (sum(cm_c28_tables[1, 131:140]) + sum(cm_c28_tables[2, 131:140])), 6)
k19w2c28_tnr <- round(sum(cm_c28_tables[4, 131:140]) /
  (sum(cm_c28_tables[4, 131:140]) + sum(cm_c28_tables[3, 131:140])), 6)
k19w2c28_truescore <- round((2 * k19w2c28_tpr * k19w2c28_tnr) /
  (k19w2c28_tpr + k19w2c28_tnr), 6)

k19w3c28_tpr <- round(sum(cm_c28_tables[1, 141:150]) /
  (sum(cm_c28_tables[1, 141:150]) + sum(cm_c28_tables[2, 141:150])), 6)
k19w3c28_tnr <- round(sum(cm_c28_tables[4, 141:150]) /
  (sum(cm_c28_tables[4, 141:150]) + sum(cm_c28_tables[3, 141:150])), 6)
k19w3c28_truescore <- round((2 * k19w3c28_tpr * k19w3c28_tnr) /
  (k19w3c28_tpr + k19w3c28_tnr), 6)

k21w1c28_tpr <- round(sum(cm_c28_tables[1, 151:160]) /
  (sum(cm_c28_tables[1, 151:160]) + sum(cm_c28_tables[2, 151:160])), 6)
k21w1c28_tnr <- round(sum(cm_c28_tables[4, 151:160]) /
  (sum(cm_c28_tables[4, 151:160]) + sum(cm_c28_tables[3, 151:160])), 6)
k21w1c28_truescore <- round((2 * k21w1c28_tpr * k21w1c28_tnr) /
  (k21w1c28_tpr + k21w1c28_tnr), 6)

k21w2c28_tpr <- round(sum(cm_c28_tables[1, 161:170]) /
  (sum(cm_c28_tables[1, 161:170]) + sum(cm_c28_tables[2, 161:170])), 6)
k21w2c28_tnr <- round(sum(cm_c28_tables[4, 161:170]) /
  (sum(cm_c28_tables[4, 161:170]) + sum(cm_c28_tables[3, 161:170])), 6)
k21w2c28_truescore <- round((2 * k21w2c28_tpr * k21w2c28_tnr) /
  (k21w2c28_tpr + k21w2c28_tnr), 6)

k21w3c28_tpr <- round(sum(cm_c28_tables[1, 171:180]) /
  (sum(cm_c28_tables[1, 171:180]) + sum(cm_c28_tables[2, 171:180])), 6)
k21w3c28_tnr <- round(sum(cm_c28_tables[4, 171:180]) /
  (sum(cm_c28_tables[4, 171:180]) + sum(cm_c28_tables[3, 171:180])), 6)
k21w3c28_truescore <- round((2 * k21w3c28_tpr * k21w3c28_tnr) /
  (k21w3c28_tpr + k21w3c28_tnr), 6)

k23w1c28_tpr <- round(sum(cm_c28_tables[1, 181:190]) /
  (sum(cm_c28_tables[1, 181:190]) + sum(cm_c28_tables[2, 181:190])), 6)
k23w1c28_tnr <- round(sum(cm_c28_tables[4, 181:190]) /
  (sum(cm_c28_tables[4, 181:190]) + sum(cm_c28_tables[3, 181:190])), 6)
k23w1c28_truescore <- round((2 * k23w1c28_tpr * k23w1c28_tnr) /
  (k23w1c28_tpr + k23w1c28_tnr), 6)

k23w2c28_tpr <- round(sum(cm_c28_tables[1, 191:200]) /
  (sum(cm_c28_tables[1, 191:200]) + sum(cm_c28_tables[2, 191:200])), 6)
k23w2c28_tnr <- round(sum(cm_c28_tables[4, 191:200]) /
  (sum(cm_c28_tables[4, 191:200]) + sum(cm_c28_tables[3, 191:200])), 6)
k23w2c28_truescore <- round((2 * k23w2c28_tpr * k23w2c28_tnr) /
  (k23w2c28_tpr + k23w2c28_tnr), 6)

k23w3c28_tpr <- round(sum(cm_c28_tables[1, 201:210]) /
  (sum(cm_c28_tables[1, 201:210]) + sum(cm_c28_tables[2, 201:210])), 6)

```

```

k23w3c28_tnr <- round(sum(cm_c28_tables[4, 201:210]) /
  (sum(cm_c28_tables[4, 201:210]) + sum(cm_c28_tables[3, 201:210])), 6)
k23w3c28_truescore <- round((2 * k23w3c28_tpr * k23w3c28_tnr) /
  (k23w3c28_tpr + k23w3c28_tnr), 6)

k25w1c28_tpr <- round(sum(cm_c28_tables[1, 211:220]) /
  (sum(cm_c28_tables[1, 211:220]) + sum(cm_c28_tables[2, 211:220])), 6)
k25w1c28_tnr <- round(sum(cm_c28_tables[4, 211:220]) /
  (sum(cm_c28_tables[4, 211:220]) + sum(cm_c28_tables[3, 211:220])), 6)
k25w1c28_truescore <- round((2 * k25w1c28_tpr * k25w1c28_tnr) /
  (k25w1c28_tpr + k25w1c28_tnr), 6)

k25w2c28_tpr <- round(sum(cm_c28_tables[1, 221:230]) /
  (sum(cm_c28_tables[1, 221:230]) + sum(cm_c28_tables[2, 221:230])), 6)
k25w2c28_tnr <- round(sum(cm_c28_tables[4, 221:230]) /
  (sum(cm_c28_tables[4, 221:230]) + sum(cm_c28_tables[3, 221:230])), 6)
k25w2c28_truescore <- round((2 * k25w2c28_tpr * k25w2c28_tnr) /
  (k25w2c28_tpr + k25w2c28_tnr), 6)

k25w3c28_tpr <- round(sum(cm_c28_tables[1, 231:240]) /
  (sum(cm_c28_tables[1, 231:240]) + sum(cm_c28_tables[2, 231:240])), 6)
k25w3c28_tnr <- round(sum(cm_c28_tables[4, 231:240]) /
  (sum(cm_c28_tables[4, 231:240]) + sum(cm_c28_tables[3, 231:240])), 6)
k25w3c28_truescore <- round((2 * k25w3c28_tpr * k25w3c28_tnr) /
  (k25w3c28_tpr + k25w3c28_tnr), 6)

k27w1c28_tpr <- round(sum(cm_c28_tables[1, 241:250]) /
  (sum(cm_c28_tables[1, 241:250]) + sum(cm_c28_tables[2, 241:250])), 6)
k27w1c28_tnr <- round(sum(cm_c28_tables[4, 241:250]) /
  (sum(cm_c28_tables[4, 241:250]) + sum(cm_c28_tables[3, 241:250])), 6)
k27w1c28_truescore <- round((2 * k27w1c28_tpr * k27w1c28_tnr) /
  (k27w1c28_tpr + k27w1c28_tnr), 6)

k27w2c28_tpr <- round(sum(cm_c28_tables[1, 251:260]) /
  (sum(cm_c28_tables[1, 251:260]) + sum(cm_c28_tables[2, 251:260])), 6)
k27w2c28_tnr <- round(sum(cm_c28_tables[4, 251:260]) /
  (sum(cm_c28_tables[4, 251:260]) + sum(cm_c28_tables[3, 251:260])), 6)
k27w2c28_truescore <- round((2 * k27w2c28_tpr * k27w2c28_tnr) /
  (k27w2c28_tpr + k27w2c28_tnr), 6)

k27w3c28_tpr <- round(sum(cm_c28_tables[1, 261:270]) /
  (sum(cm_c28_tables[1, 261:270]) + sum(cm_c28_tables[2, 261:270])), 6)
k27w3c28_tnr <- round(sum(cm_c28_tables[4, 261:270]) /
  (sum(cm_c28_tables[4, 261:270]) + sum(cm_c28_tables[3, 261:270])), 6)
k27w3c28_truescore <- round((2 * k27w3c28_tpr * k27w3c28_tnr) /
  (k27w3c28_tpr + k27w3c28_tnr), 6)

k29w1c28_tpr <- round(sum(cm_c28_tables[1, 271:280]) /
  (sum(cm_c28_tables[1, 271:280]) + sum(cm_c28_tables[2, 271:280])), 6)
k29w1c28_tnr <- round(sum(cm_c28_tables[4, 271:280]) /

```

```

(sum(cm_c28_tables[4, 271:280]) + sum(cm_c28_tables[3, 271:280])), 6)
k29w1c28_truescore <- round((2 * k29w1c28_tpr * k29w1c28_tnr) /
(k29w1c28_tpr + k29w1c28_tnr), 6)

k29w2c28_tpr <- round(sum(cm_c28_tables[1, 281:290]) /
(sum(cm_c28_tables[1, 281:290]) + sum(cm_c28_tables[2, 281:290])), 6)
k29w2c28_tnr <- round(sum(cm_c28_tables[4, 281:290]) /
(sum(cm_c28_tables[4, 281:290]) + sum(cm_c28_tables[3, 281:290])), 6)
k29w2c28_truescore <- round((2 * k29w2c28_tpr * k29w2c28_tnr) /
(k29w2c28_tpr + k29w2c28_tnr), 6)

k29w3c28_tpr <- round(sum(cm_c28_tables[1, 291:300]) /
(sum(cm_c28_tables[1, 291:300]) + sum(cm_c28_tables[2, 291:300])), 6)
k29w3c28_tnr <- round(sum(cm_c28_tables[4, 291:300]) /
(sum(cm_c28_tables[4, 291:300]) + sum(cm_c28_tables[3, 291:300])), 6)
k29w3c28_truescore <- round((2 * k29w3c28_tpr * k29w3c28_tnr) /
(k29w3c28_tpr + k29w3c28_tnr), 6)

k31w1c28_tpr <- round(sum(cm_c28_tables[1, 301:310]) /
(sum(cm_c28_tables[1, 301:310]) + sum(cm_c28_tables[2, 301:310])), 6)
k31w1c28_tnr <- round(sum(cm_c28_tables[4, 301:310]) /
(sum(cm_c28_tables[4, 301:310]) + sum(cm_c28_tables[3, 301:310])), 6)
k31w1c28_truescore <- round((2 * k31w1c28_tpr * k31w1c28_tnr) /
(k31w1c28_tpr + k31w1c28_tnr), 6)

k31w2c28_tpr <- round(sum(cm_c28_tables[1, 311:320]) /
(sum(cm_c28_tables[1, 311:320]) + sum(cm_c28_tables[2, 311:320])), 6)
k31w2c28_tnr <- round(sum(cm_c28_tables[4, 311:320]) /
(sum(cm_c28_tables[4, 311:320]) + sum(cm_c28_tables[3, 311:320])), 6)
k31w2c28_truescore <- round((2 * k31w2c28_tpr * k31w2c28_tnr) /
(k31w2c28_tpr + k31w2c28_tnr), 6)

k31w3c28_tpr <- round(sum(cm_c28_tables[1, 321:330]) /
(sum(cm_c28_tables[1, 321:330]) + sum(cm_c28_tables[2, 321:330])), 6)
k31w3c28_tnr <- round(sum(cm_c28_tables[4, 321:330]) /
(sum(cm_c28_tables[4, 321:330]) + sum(cm_c28_tables[3, 321:330])), 6)
k31w3c28_truescore <- round((2 * k31w3c28_tpr * k31w3c28_tnr) /
(k31w3c28_tpr + k31w3c28_tnr), 6)

k33w1c28_tpr <- round(sum(cm_c28_tables[1, 331:340]) /
(sum(cm_c28_tables[1, 331:340]) + sum(cm_c28_tables[2, 331:340])), 6)
k33w1c28_tnr <- round(sum(cm_c28_tables[4, 331:340]) /
(sum(cm_c28_tables[4, 331:340]) + sum(cm_c28_tables[3, 331:340])), 6)
k33w1c28_truescore <- round((2 * k33w1c28_tpr * k33w1c28_tnr) /
(k33w1c28_tpr + k33w1c28_tnr), 6)

k33w2c28_tpr <- round(sum(cm_c28_tables[1, 341:350]) /
(sum(cm_c28_tables[1, 341:350]) + sum(cm_c28_tables[2, 341:350])), 6)
k33w2c28_tnr <- round(sum(cm_c28_tables[4, 341:350]) /
(sum(cm_c28_tables[4, 341:350]) + sum(cm_c28_tables[3, 341:350])), 6)
k33w2c28_truescore <- round((2 * k33w2c28_tpr * k33w2c28_tnr) /

```

```

(k33w2c28_tpr + k33w2c28_tnr), 6)

k33w3c28_tpr <- round(sum(cm_c28_tables[1, 351:360]) /
  (sum(cm_c28_tables[1, 351:360]) + sum(cm_c28_tables[2, 351:360])), 6)
k33w3c28_tnr <- round(sum(cm_c28_tables[4, 351:360]) /
  (sum(cm_c28_tables[4, 351:360]) + sum(cm_c28_tables[3, 351:360])), 6)
k33w3c28_truescore <- round((2 * k33w3c28_tpr * k33w3c28_tnr) /
  (k33w3c28_tpr + k33w3c28_tnr), 6)

k35w1c28_tpr <- round(sum(cm_c28_tables[1, 361:370]) /
  (sum(cm_c28_tables[1, 361:370]) + sum(cm_c28_tables[2, 361:370])), 6)
k35w1c28_tnr <- round(sum(cm_c28_tables[4, 361:370]) /
  (sum(cm_c28_tables[4, 361:370]) + sum(cm_c28_tables[3, 361:370])), 6)
k35w1c28_truescore <- round((2 * k35w1c28_tpr * k35w1c28_tnr) /
  (k35w1c28_tpr + k35w1c28_tnr), 6)

k35w2c28_tpr <- round(sum(cm_c28_tables[1, 371:380]) /
  (sum(cm_c28_tables[1, 371:380]) + sum(cm_c28_tables[2, 371:380])), 6)
k35w2c28_tnr <- round(sum(cm_c28_tables[4, 371:380]) /
  (sum(cm_c28_tables[4, 371:380]) + sum(cm_c28_tables[3, 371:380])), 6)
k35w2c28_truescore <- round((2 * k35w2c28_tpr * k35w2c28_tnr) /
  (k35w2c28_tpr + k35w2c28_tnr), 6)

k35w3c28_tpr <- round(sum(cm_c28_tables[1, 381:390]) /
  (sum(cm_c28_tables[1, 381:390]) + sum(cm_c28_tables[2, 381:390])), 6)
k35w3c28_tnr <- round(sum(cm_c28_tables[4, 381:390]) /
  (sum(cm_c28_tables[4, 381:390]) + sum(cm_c28_tables[3, 381:390])), 6)
k35w3c28_truescore <- round((2 * k35w3c28_tpr * k35w3c28_tnr) /
  (k35w3c28_tpr + k35w3c28_tnr), 6)

k37w1c28_tpr <- round(sum(cm_c28_tables[1, 391:400]) /
  (sum(cm_c28_tables[1, 391:400]) + sum(cm_c28_tables[2, 391:400])), 6)
k37w1c28_tnr <- round(sum(cm_c28_tables[4, 391:400]) /
  (sum(cm_c28_tables[4, 391:400]) + sum(cm_c28_tables[3, 391:400])), 6)
k37w1c28_truescore <- round((2 * k37w1c28_tpr * k37w1c28_tnr) /
  (k37w1c28_tpr + k37w1c28_tnr), 6)

k37w2c28_tpr <- round(sum(cm_c28_tables[1, 401:410]) /
  (sum(cm_c28_tables[1, 401:410]) + sum(cm_c28_tables[2, 401:410])), 6)
k37w2c28_tnr <- round(sum(cm_c28_tables[4, 401:410]) /
  (sum(cm_c28_tables[4, 401:410]) + sum(cm_c28_tables[3, 401:410])), 6)
k37w2c28_truescore <- round((2 * k37w2c28_tpr * k37w2c28_tnr) /
  (k37w2c28_tpr + k37w2c28_tnr), 6)

k37w3c28_tpr <- round(sum(cm_c28_tables[1, 411:420]) /
  (sum(cm_c28_tables[1, 411:420]) + sum(cm_c28_tables[2, 411:420])), 6)
k37w3c28_tnr <- round(sum(cm_c28_tables[4, 411:420]) /
  (sum(cm_c28_tables[4, 411:420]) + sum(cm_c28_tables[3, 411:420])), 6)
k37w3c28_truescore <- round((2 * k37w3c28_tpr * k37w3c28_tnr) /
  (k37w3c28_tpr + k37w3c28_tnr), 6)

```

```

k39w1c28_tpr <- round(sum(cm_c28_tables[1, 421:430]) /
  (sum(cm_c28_tables[1, 421:430]) + sum(cm_c28_tables[2, 421:430])), 6)
k39w1c28_tnr <- round(sum(cm_c28_tables[4, 421:430]) /
  (sum(cm_c28_tables[4, 421:430]) + sum(cm_c28_tables[3, 421:430])), 6)
k39w1c28_truescore <- round((2 * k39w1c28_tpr * k39w1c28_tnr) /
  (k39w1c28_tpr + k39w1c28_tnr), 6)

k39w2c28_tpr <- round(sum(cm_c28_tables[1, 431:440]) /
  (sum(cm_c28_tables[1, 431:440]) + sum(cm_c28_tables[2, 431:440])), 6)
k39w2c28_tnr <- round(sum(cm_c28_tables[4, 431:440]) /
  (sum(cm_c28_tables[4, 431:440]) + sum(cm_c28_tables[3, 431:440])), 6)
k39w2c28_truescore <- round((2 * k39w2c28_tpr * k39w2c28_tnr) /
  (k39w2c28_tpr + k39w2c28_tnr), 6)

k39w3c28_tpr <- round(sum(cm_c28_tables[1, 441:450]) /
  (sum(cm_c28_tables[1, 441:450]) + sum(cm_c28_tables[2, 441:450])), 6)
k39w3c28_tnr <- round(sum(cm_c28_tables[4, 441:450]) /
  (sum(cm_c28_tables[4, 441:450]) + sum(cm_c28_tables[3, 441:450])), 6)
k39w3c28_truescore <- round((2 * k39w3c28_tpr * k39w3c28_tnr) /
  (k39w3c28_tpr + k39w3c28_tnr), 6)

# Compile the 0.28 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c28_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.28,
  TPR = c(k11w1c28_tpr, k11w2c28_tpr, k11w3c28_tpr, k13w1c28_tpr,
    k13w2c28_tpr, k13w3c28_tpr, k15w1c28_tpr, k15w2c28_tpr,
    k15w3c28_tpr, k17w1c28_tpr, k17w2c28_tpr, k17w3c28_tpr,
    k19w1c28_tpr, k19w2c28_tpr, k19w3c28_tpr, k21w1c28_tpr,
    k21w2c28_tpr, k21w3c28_tpr, k23w1c28_tpr, k23w2c28_tpr,
    k23w3c28_tpr, k25w1c28_tpr, k25w2c28_tpr, k25w3c28_tpr,
    k27w1c28_tpr, k27w2c28_tpr, k27w3c28_tpr, k29w1c28_tpr,
    k29w2c28_tpr, k29w3c28_tpr, k31w1c28_tpr, k31w2c28_tpr,
    k31w3c28_tpr, k33w1c28_tpr, k33w2c28_tpr, k33w3c28_tpr,
    k35w1c28_tpr, k35w2c28_tpr, k35w3c28_tpr, k37w1c28_tpr,
    k37w2c28_tpr, k37w3c28_tpr, k39w1c28_tpr, k39w2c28_tpr,
    k39w3c28_tpr),
  TNR = c(k11w1c28_tnr, k11w2c28_tnr, k11w3c28_tnr, k13w1c28_tnr,
    k13w2c28_tnr, k13w3c28_tnr, k15w1c28_tnr, k15w2c28_tnr,
    k15w3c28_tnr, k17w1c28_tnr, k17w2c28_tnr, k17w3c28_tnr,
    k19w1c28_tnr, k19w2c28_tnr, k19w3c28_tnr, k21w1c28_tnr,
    k21w2c28_tnr, k21w3c28_tnr, k23w1c28_tnr, k23w2c28_tnr,
    k23w3c28_tnr, k25w1c28_tnr, k25w2c28_tnr, k25w3c28_tnr,
    k27w1c28_tnr, k27w2c28_tnr, k27w3c28_tnr, k29w1c28_tnr,
    k29w2c28_tnr, k29w3c28_tnr, k31w1c28_tnr, k31w2c28_tnr,
    k31w3c28_tnr, k33w1c28_tnr, k33w2c28_tnr, k33w3c28_tnr)

```

```

k35w1c28_tnr, k35w2c28_tnr, k35w3c28_tnr, k37w1c28_tnr,
k37w2c28_tnr, k37w3c28_tnr, k39w1c28_tnr, k39w2c28_tnr,
k39w3c28_tnr),
Truescore = c(k11w1c28_truescore, k11w2c28_truescore,
k11w3c28_truescore, k13w1c28_truescore,
k13w2c28_truescore, k13w3c28_truescore,
k15w1c28_truescore, k15w2c28_truescore,
k15w3c28_truescore, k17w1c28_truescore,
k17w2c28_truescore, k17w3c28_truescore,
k19w1c28_truescore, k19w2c28_truescore,
k19w3c28_truescore, k21w1c28_truescore,
k21w2c28_truescore, k21w3c28_truescore,
k23w1c28_truescore, k23w2c28_truescore,
k23w3c28_truescore, k25w1c28_truescore,
k25w2c28_truescore, k25w3c28_truescore,
k27w1c28_truescore, k27w2c28_truescore,
k27w3c28_truescore, k29w1c28_truescore,
k29w2c28_truescore, k29w3c28_truescore,
k31w1c28_truescore, k31w2c28_truescore,
k31w3c28_truescore, k33w1c28_truescore,
k33w2c28_truescore, k33w3c28_truescore,
k35w1c28_truescore, k35w2c28_truescore,
k35w3c28_truescore, k37w1c28_truescore,
k37w2c28_truescore, k37w3c28_truescore,
k39w1c28_truescore, k39w2c28_truescore,
k39w3c28_truescore))

```

`knitr::kable(c28_results[1:45,], caption = "c28_results")`

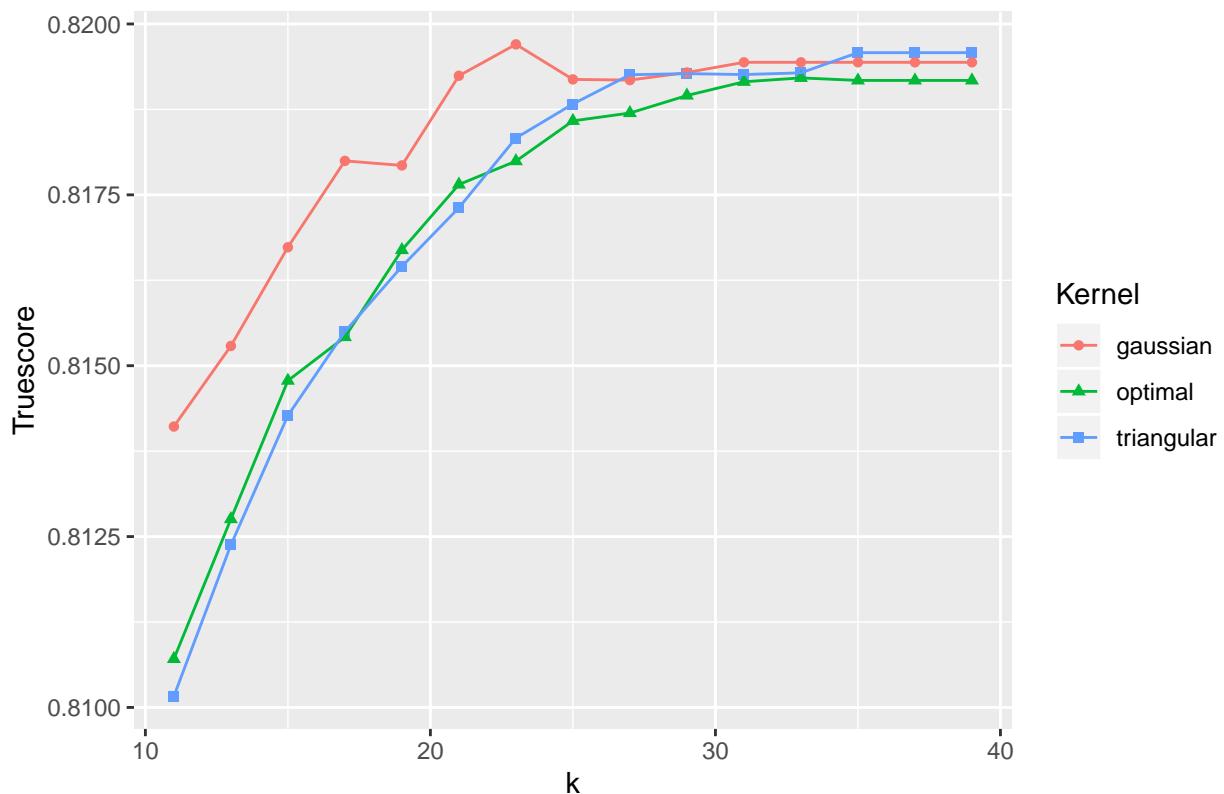
Table 37: c28_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.28	0.824548	0.796271	0.810163
11	gaussian	0.28	0.837701	0.791811	0.814110
11	optimal	0.28	0.824849	0.797047	0.810710
13	triangular	0.28	0.830622	0.794916	0.812377
13	gaussian	0.28	0.841265	0.790870	0.815289
13	optimal	0.28	0.832229	0.794173	0.812756
15	triangular	0.28	0.836145	0.793512	0.814271
15	gaussian	0.28	0.846084	0.789350	0.816733
15	optimal	0.28	0.837149	0.793578	0.814781
17	triangular	0.28	0.840161	0.792241	0.815498
17	gaussian	0.28	0.849548	0.788706	0.817997
17	optimal	0.28	0.840663	0.791646	0.815419
19	triangular	0.28	0.843424	0.791151	0.816452
19	gaussian	0.28	0.850402	0.787847	0.817930
19	optimal	0.28	0.844277	0.790853	0.816692
21	triangular	0.28	0.846737	0.789862	0.817311
21	gaussian	0.28	0.853514	0.787616	0.819242
21	optimal	0.28	0.847390	0.789928	0.817651
23	triangular	0.28	0.850000	0.788937	0.818331
23	gaussian	0.28	0.855371	0.786889	0.819702
23	optimal	0.28	0.849347	0.788871	0.817993

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.28	0.851556	0.788524	0.818829
25	gaussian	0.28	0.855171	0.786113	0.819189
25	optimal	0.28	0.851908	0.787765	0.818582
27	triangular	0.28	0.852811	0.788244	0.819257
27	gaussian	0.28	0.855422	0.785882	0.819179
27	optimal	0.28	0.852912	0.787121	0.818697
29	triangular	0.28	0.854066	0.787203	0.819273
29	gaussian	0.28	0.855723	0.785832	0.819290
29	optimal	0.28	0.854267	0.786443	0.818953
31	triangular	0.28	0.854819	0.786542	0.819260
31	gaussian	0.28	0.857229	0.784841	0.819439
31	optimal	0.28	0.855271	0.785964	0.819154
33	triangular	0.28	0.854970	0.786460	0.819285
33	gaussian	0.28	0.857229	0.784841	0.819439
33	optimal	0.28	0.855924	0.785518	0.819211
35	triangular	0.28	0.856024	0.786113	0.819580
35	gaussian	0.28	0.857229	0.784841	0.819439
35	optimal	0.28	0.856175	0.785238	0.819174
37	triangular	0.28	0.856024	0.786113	0.819580
37	gaussian	0.28	0.857229	0.784841	0.819439
37	optimal	0.28	0.856175	0.785238	0.819174
39	triangular	0.28	0.856024	0.786113	0.819580
39	gaussian	0.28	0.857229	0.784841	0.819439
39	optimal	0.28	0.856175	0.785238	0.819174

```
ggplot(c28_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.28 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.28 (Truescore)

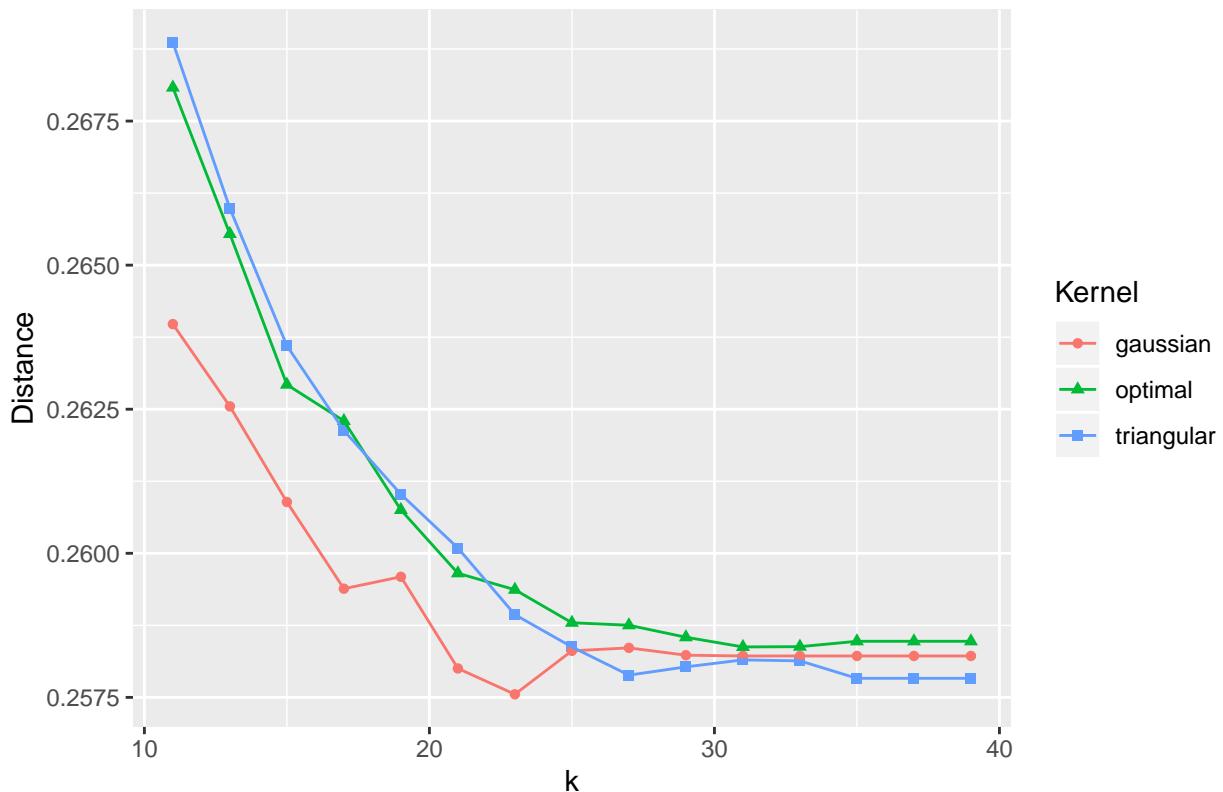


```
# For the Cutoff of 0.28, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c28_results <- c28_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c28_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.28 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.28 (Distance)



```
# For the Cutoff of 0.28, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c28_results$Truescore)
```

```
## [1] 0.819702
```

```
(c28_opt_k_ts <- c28_results$k[which.max(c28_results$Truescore)])
```

```
## [1] 23
```

```
(c28_opt_kernel_ts <- c28_results$Kernel[which.max(c28_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c28_opt_cut_ts <- c28_results$Cut[which.max(c28_results$Truescore)])
```

```
## [1] 0.28
```

```
(c28_opt_tpr_ts <- c28_results$TPR[which.max(c28_results$Truescore)])
```

```
## [1] 0.855371
```

```

(c28_opt_tnr_ts <- c28_results$TNR[which.max(c28_results$Truescore)])

## [1] 0.786889

(c28_opt_d_ts <- c28_results$Distance[which.max(c28_results$Truescore)])

## [1] 0.2575536

min(c28_results$Distance)

## [1] 0.2575536

(c28_opt_k_dist <- c28_results$k[which.min(c28_results$Distance)])

## [1] 23

(c28_opt_kernel_dist <- c28_results$Kernel[which.min(c28_results$Distance)])

## [1] "gaussian"

(c28_opt_cut_dist <- c28_results$Cut[which.min(c28_results$Distance)])

## [1] 0.28

(c28_opt_tpr_dist <- c28_results$TPR[which.min(c28_results$Distance)])

## [1] 0.855371

(c28_opt_tnr_dist <- c28_results$TNR[which.min(c28_results$Distance)])

## [1] 0.786889

(c28_opt_t_dist <- c28_results$Truescore[which.min(c28_results$Distance)])

## [1] 0.819702

#####
# 0.29 Cutoff
#####

# For the decision cutoff of 0.29, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c29 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred29, obs))
  confusionMatrix(ss$pred29, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c29) <- kwf_dfs_v

cm_c29_tables <- sapply(cm_c29, "[[", 2)
cm_c29_tables <- as_tibble(cm_c29_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.29.

k11w1c29_tpr <- round(sum(cm_c29_tables[1, 1:10]) /
  (sum(cm_c29_tables[1, 1:10]) + sum(cm_c29_tables[2, 1:10])), 6)
k11w1c29_tnr <- round(sum(cm_c29_tables[4, 1:10]) /
  (sum(cm_c29_tables[4, 1:10]) + sum(cm_c29_tables[3, 1:10])), 6)
k11w1c29_truescore <- round((2 * k11w1c29_tpr * k11w1c29_tnr) /
  (k11w1c29_tpr + k11w1c29_tnr), 6)

k11w2c29_tpr <- round(sum(cm_c29_tables[1, 11:20]) /
  (sum(cm_c29_tables[1, 11:20]) + sum(cm_c29_tables[2, 11:20])), 6)
k11w2c29_tnr <- round(sum(cm_c29_tables[4, 11:20]) /
  (sum(cm_c29_tables[4, 11:20]) + sum(cm_c29_tables[3, 11:20])), 6)
k11w2c29_truescore <- round((2 * k11w2c29_tpr * k11w2c29_tnr) /
  (k11w2c29_tpr + k11w2c29_tnr), 6)

k11w3c29_tpr <- round(sum(cm_c29_tables[1, 21:30]) /
  (sum(cm_c29_tables[1, 21:30]) + sum(cm_c29_tables[2, 21:30])), 6)
k11w3c29_tnr <- round(sum(cm_c29_tables[4, 21:30]) /
  (sum(cm_c29_tables[4, 21:30]) + sum(cm_c29_tables[3, 21:30])), 6)
k11w3c29_truescore <- round((2 * k11w3c29_tpr * k11w3c29_tnr) /
  (k11w3c29_tpr + k11w3c29_tnr), 6)

k13w1c29_tpr <- round(sum(cm_c29_tables[1, 31:40]) /
  (sum(cm_c29_tables[1, 31:40]) + sum(cm_c29_tables[2, 31:40])), 6)
k13w1c29_tnr <- round(sum(cm_c29_tables[4, 31:40]) /
  (sum(cm_c29_tables[4, 31:40]) + sum(cm_c29_tables[3, 31:40])), 6)
k13w1c29_truescore <- round((2 * k13w1c29_tpr * k13w1c29_tnr) /
  (k13w1c29_tpr + k13w1c29_tnr), 6)

k13w2c29_tpr <- round(sum(cm_c29_tables[1, 41:50]) /
  (sum(cm_c29_tables[1, 41:50]) + sum(cm_c29_tables[2, 41:50])), 6)
k13w2c29_tnr <- round(sum(cm_c29_tables[4, 41:50]) /
  (sum(cm_c29_tables[4, 41:50]) + sum(cm_c29_tables[3, 41:50])), 6)
k13w2c29_truescore <- round((2 * k13w2c29_tpr * k13w2c29_tnr) /
  (k13w2c29_tpr + k13w2c29_tnr), 6)

k13w3c29_tpr <- round(sum(cm_c29_tables[1, 51:60]) /
  (sum(cm_c29_tables[1, 51:60]) + sum(cm_c29_tables[2, 51:60])), 6)
k13w3c29_tnr <- round(sum(cm_c29_tables[4, 51:60]) /
  (sum(cm_c29_tables[4, 51:60]) + sum(cm_c29_tables[3, 51:60])), 6)
k13w3c29_truescore <- round((2 * k13w3c29_tpr * k13w3c29_tnr) /

```

```

(k13w3c29_tpr + k13w3c29_tnr), 6)

k15w1c29_tpr <- round(sum(cm_c29_tables[1, 61:70]) /
  (sum(cm_c29_tables[1, 61:70]) + sum(cm_c29_tables[2, 61:70])), 6)
k15w1c29_tnr <- round(sum(cm_c29_tables[4, 61:70]) /
  (sum(cm_c29_tables[4, 61:70]) + sum(cm_c29_tables[3, 61:70])), 6)
k15w1c29_truescore <- round((2 * k15w1c29_tpr * k15w1c29_tnr) /
  (k15w1c29_tpr + k15w1c29_tnr), 6)

k15w2c29_tpr <- round(sum(cm_c29_tables[1, 71:80]) /
  (sum(cm_c29_tables[1, 71:80]) + sum(cm_c29_tables[2, 71:80])), 6)
k15w2c29_tnr <- round(sum(cm_c29_tables[4, 71:80]) /
  (sum(cm_c29_tables[4, 71:80]) + sum(cm_c29_tables[3, 71:80])), 6)
k15w2c29_truescore <- round((2 * k15w2c29_tpr * k15w2c29_tnr) /
  (k15w2c29_tpr + k15w2c29_tnr), 6)

k15w3c29_tpr <- round(sum(cm_c29_tables[1, 81:90]) /
  (sum(cm_c29_tables[1, 81:90]) + sum(cm_c29_tables[2, 81:90])), 6)
k15w3c29_tnr <- round(sum(cm_c29_tables[4, 81:90]) /
  (sum(cm_c29_tables[4, 81:90]) + sum(cm_c29_tables[3, 81:90])), 6)
k15w3c29_truescore <- round((2 * k15w3c29_tpr * k15w3c29_tnr) /
  (k15w3c29_tpr + k15w3c29_tnr), 6)

k17w1c29_tpr <- round(sum(cm_c29_tables[1, 91:100]) /
  (sum(cm_c29_tables[1, 91:100]) + sum(cm_c29_tables[2, 91:100])), 6)
k17w1c29_tnr <- round(sum(cm_c29_tables[4, 91:100]) /
  (sum(cm_c29_tables[4, 91:100]) + sum(cm_c29_tables[3, 91:100])), 6)
k17w1c29_truescore <- round((2 * k17w1c29_tpr * k17w1c29_tnr) /
  (k17w1c29_tpr + k17w1c29_tnr), 6)

k17w2c29_tpr <- round(sum(cm_c29_tables[1, 101:110]) /
  (sum(cm_c29_tables[1, 101:110]) + sum(cm_c29_tables[2, 101:110])), 6)
k17w2c29_tnr <- round(sum(cm_c29_tables[4, 101:110]) /
  (sum(cm_c29_tables[4, 101:110]) + sum(cm_c29_tables[3, 101:110])), 6)
k17w2c29_truescore <- round((2 * k17w2c29_tpr * k17w2c29_tnr) /
  (k17w2c29_tpr + k17w2c29_tnr), 6)

k17w3c29_tpr <- round(sum(cm_c29_tables[1, 111:120]) /
  (sum(cm_c29_tables[1, 111:120]) + sum(cm_c29_tables[2, 111:120])), 6)
k17w3c29_tnr <- round(sum(cm_c29_tables[4, 111:120]) /
  (sum(cm_c29_tables[4, 111:120]) + sum(cm_c29_tables[3, 111:120])), 6)
k17w3c29_truescore <- round((2 * k17w3c29_tpr * k17w3c29_tnr) /
  (k17w3c29_tpr + k17w3c29_tnr), 6)

k19w1c29_tpr <- round(sum(cm_c29_tables[1, 121:130]) /
  (sum(cm_c29_tables[1, 121:130]) + sum(cm_c29_tables[2, 121:130])), 6)
k19w1c29_tnr <- round(sum(cm_c29_tables[4, 121:130]) /
  (sum(cm_c29_tables[4, 121:130]) + sum(cm_c29_tables[3, 121:130])), 6)
k19w1c29_truescore <- round((2 * k19w1c29_tpr * k19w1c29_tnr) /
  (k19w1c29_tpr + k19w1c29_tnr), 6)

```

```

k19w2c29_tpr <- round(sum(cm_c29_tables[1, 131:140]) /
  (sum(cm_c29_tables[1, 131:140]) + sum(cm_c29_tables[2, 131:140])), 6)
k19w2c29_tnr <- round(sum(cm_c29_tables[4, 131:140]) /
  (sum(cm_c29_tables[4, 131:140]) + sum(cm_c29_tables[3, 131:140])), 6)
k19w2c29_truescore <- round((2 * k19w2c29_tpr * k19w2c29_tnr) /
  (k19w2c29_tpr + k19w2c29_tnr), 6)

k19w3c29_tpr <- round(sum(cm_c29_tables[1, 141:150]) /
  (sum(cm_c29_tables[1, 141:150]) + sum(cm_c29_tables[2, 141:150])), 6)
k19w3c29_tnr <- round(sum(cm_c29_tables[4, 141:150]) /
  (sum(cm_c29_tables[4, 141:150]) + sum(cm_c29_tables[3, 141:150])), 6)
k19w3c29_truescore <- round((2 * k19w3c29_tpr * k19w3c29_tnr) /
  (k19w3c29_tpr + k19w3c29_tnr), 6)

k21w1c29_tpr <- round(sum(cm_c29_tables[1, 151:160]) /
  (sum(cm_c29_tables[1, 151:160]) + sum(cm_c29_tables[2, 151:160])), 6)
k21w1c29_tnr <- round(sum(cm_c29_tables[4, 151:160]) /
  (sum(cm_c29_tables[4, 151:160]) + sum(cm_c29_tables[3, 151:160])), 6)
k21w1c29_truescore <- round((2 * k21w1c29_tpr * k21w1c29_tnr) /
  (k21w1c29_tpr + k21w1c29_tnr), 6)

k21w2c29_tpr <- round(sum(cm_c29_tables[1, 161:170]) /
  (sum(cm_c29_tables[1, 161:170]) + sum(cm_c29_tables[2, 161:170])), 6)
k21w2c29_tnr <- round(sum(cm_c29_tables[4, 161:170]) /
  (sum(cm_c29_tables[4, 161:170]) + sum(cm_c29_tables[3, 161:170])), 6)
k21w2c29_truescore <- round((2 * k21w2c29_tpr * k21w2c29_tnr) /
  (k21w2c29_tpr + k21w2c29_tnr), 6)

k21w3c29_tpr <- round(sum(cm_c29_tables[1, 171:180]) /
  (sum(cm_c29_tables[1, 171:180]) + sum(cm_c29_tables[2, 171:180])), 6)
k21w3c29_tnr <- round(sum(cm_c29_tables[4, 171:180]) /
  (sum(cm_c29_tables[4, 171:180]) + sum(cm_c29_tables[3, 171:180])), 6)
k21w3c29_truescore <- round((2 * k21w3c29_tpr * k21w3c29_tnr) /
  (k21w3c29_tpr + k21w3c29_tnr), 6)

k23w1c29_tpr <- round(sum(cm_c29_tables[1, 181:190]) /
  (sum(cm_c29_tables[1, 181:190]) + sum(cm_c29_tables[2, 181:190])), 6)
k23w1c29_tnr <- round(sum(cm_c29_tables[4, 181:190]) /
  (sum(cm_c29_tables[4, 181:190]) + sum(cm_c29_tables[3, 181:190])), 6)
k23w1c29_truescore <- round((2 * k23w1c29_tpr * k23w1c29_tnr) /
  (k23w1c29_tpr + k23w1c29_tnr), 6)

k23w2c29_tpr <- round(sum(cm_c29_tables[1, 191:200]) /
  (sum(cm_c29_tables[1, 191:200]) + sum(cm_c29_tables[2, 191:200])), 6)
k23w2c29_tnr <- round(sum(cm_c29_tables[4, 191:200]) /
  (sum(cm_c29_tables[4, 191:200]) + sum(cm_c29_tables[3, 191:200])), 6)
k23w2c29_truescore <- round((2 * k23w2c29_tpr * k23w2c29_tnr) /
  (k23w2c29_tpr + k23w2c29_tnr), 6)

k23w3c29_tpr <- round(sum(cm_c29_tables[1, 201:210]) /
  (sum(cm_c29_tables[1, 201:210]) + sum(cm_c29_tables[2, 201:210])), 6)

```

```

k23w3c29_tnr <- round(sum(cm_c29_tables[4, 201:210]) /
  (sum(cm_c29_tables[4, 201:210]) + sum(cm_c29_tables[3, 201:210])), 6)
k23w3c29_truescore <- round((2 * k23w3c29_tpr * k23w3c29_tnr) /
  (k23w3c29_tpr + k23w3c29_tnr), 6)

k25w1c29_tpr <- round(sum(cm_c29_tables[1, 211:220]) /
  (sum(cm_c29_tables[1, 211:220]) + sum(cm_c29_tables[2, 211:220])), 6)
k25w1c29_tnr <- round(sum(cm_c29_tables[4, 211:220]) /
  (sum(cm_c29_tables[4, 211:220]) + sum(cm_c29_tables[3, 211:220])), 6)
k25w1c29_truescore <- round((2 * k25w1c29_tpr * k25w1c29_tnr) /
  (k25w1c29_tpr + k25w1c29_tnr), 6)

k25w2c29_tpr <- round(sum(cm_c29_tables[1, 221:230]) /
  (sum(cm_c29_tables[1, 221:230]) + sum(cm_c29_tables[2, 221:230])), 6)
k25w2c29_tnr <- round(sum(cm_c29_tables[4, 221:230]) /
  (sum(cm_c29_tables[4, 221:230]) + sum(cm_c29_tables[3, 221:230])), 6)
k25w2c29_truescore <- round((2 * k25w2c29_tpr * k25w2c29_tnr) /
  (k25w2c29_tpr + k25w2c29_tnr), 6)

k25w3c29_tpr <- round(sum(cm_c29_tables[1, 231:240]) /
  (sum(cm_c29_tables[1, 231:240]) + sum(cm_c29_tables[2, 231:240])), 6)
k25w3c29_tnr <- round(sum(cm_c29_tables[4, 231:240]) /
  (sum(cm_c29_tables[4, 231:240]) + sum(cm_c29_tables[3, 231:240])), 6)
k25w3c29_truescore <- round((2 * k25w3c29_tpr * k25w3c29_tnr) /
  (k25w3c29_tpr + k25w3c29_tnr), 6)

k27w1c29_tpr <- round(sum(cm_c29_tables[1, 241:250]) /
  (sum(cm_c29_tables[1, 241:250]) + sum(cm_c29_tables[2, 241:250])), 6)
k27w1c29_tnr <- round(sum(cm_c29_tables[4, 241:250]) /
  (sum(cm_c29_tables[4, 241:250]) + sum(cm_c29_tables[3, 241:250])), 6)
k27w1c29_truescore <- round((2 * k27w1c29_tpr * k27w1c29_tnr) /
  (k27w1c29_tpr + k27w1c29_tnr), 6)

k27w2c29_tpr <- round(sum(cm_c29_tables[1, 251:260]) /
  (sum(cm_c29_tables[1, 251:260]) + sum(cm_c29_tables[2, 251:260])), 6)
k27w2c29_tnr <- round(sum(cm_c29_tables[4, 251:260]) /
  (sum(cm_c29_tables[4, 251:260]) + sum(cm_c29_tables[3, 251:260])), 6)
k27w2c29_truescore <- round((2 * k27w2c29_tpr * k27w2c29_tnr) /
  (k27w2c29_tpr + k27w2c29_tnr), 6)

k27w3c29_tpr <- round(sum(cm_c29_tables[1, 261:270]) /
  (sum(cm_c29_tables[1, 261:270]) + sum(cm_c29_tables[2, 261:270])), 6)
k27w3c29_tnr <- round(sum(cm_c29_tables[4, 261:270]) /
  (sum(cm_c29_tables[4, 261:270]) + sum(cm_c29_tables[3, 261:270])), 6)
k27w3c29_truescore <- round((2 * k27w3c29_tpr * k27w3c29_tnr) /
  (k27w3c29_tpr + k27w3c29_tnr), 6)

k29w1c29_tpr <- round(sum(cm_c29_tables[1, 271:280]) /
  (sum(cm_c29_tables[1, 271:280]) + sum(cm_c29_tables[2, 271:280])), 6)
k29w1c29_tnr <- round(sum(cm_c29_tables[4, 271:280]) /

```

```

(sum(cm_c29_tables[4, 271:280]) + sum(cm_c29_tables[3, 271:280])), 6)
k29w1c29_truescore <- round((2 * k29w1c29_tpr * k29w1c29_tnr) /
(k29w1c29_tpr + k29w1c29_tnr), 6)

k29w2c29_tpr <- round(sum(cm_c29_tables[1, 281:290]) /
(sum(cm_c29_tables[1, 281:290]) + sum(cm_c29_tables[2, 281:290])), 6)
k29w2c29_tnr <- round(sum(cm_c29_tables[4, 281:290]) /
(sum(cm_c29_tables[4, 281:290]) + sum(cm_c29_tables[3, 281:290])), 6)
k29w2c29_truescore <- round((2 * k29w2c29_tpr * k29w2c29_tnr) /
(k29w2c29_tpr + k29w2c29_tnr), 6)

k29w3c29_tpr <- round(sum(cm_c29_tables[1, 291:300]) /
(sum(cm_c29_tables[1, 291:300]) + sum(cm_c29_tables[2, 291:300])), 6)
k29w3c29_tnr <- round(sum(cm_c29_tables[4, 291:300]) /
(sum(cm_c29_tables[4, 291:300]) + sum(cm_c29_tables[3, 291:300])), 6)
k29w3c29_truescore <- round((2 * k29w3c29_tpr * k29w3c29_tnr) /
(k29w3c29_tpr + k29w3c29_tnr), 6)

k31w1c29_tpr <- round(sum(cm_c29_tables[1, 301:310]) /
(sum(cm_c29_tables[1, 301:310]) + sum(cm_c29_tables[2, 301:310])), 6)
k31w1c29_tnr <- round(sum(cm_c29_tables[4, 301:310]) /
(sum(cm_c29_tables[4, 301:310]) + sum(cm_c29_tables[3, 301:310])), 6)
k31w1c29_truescore <- round((2 * k31w1c29_tpr * k31w1c29_tnr) /
(k31w1c29_tpr + k31w1c29_tnr), 6)

k31w2c29_tpr <- round(sum(cm_c29_tables[1, 311:320]) /
(sum(cm_c29_tables[1, 311:320]) + sum(cm_c29_tables[2, 311:320])), 6)
k31w2c29_tnr <- round(sum(cm_c29_tables[4, 311:320]) /
(sum(cm_c29_tables[4, 311:320]) + sum(cm_c29_tables[3, 311:320])), 6)
k31w2c29_truescore <- round((2 * k31w2c29_tpr * k31w2c29_tnr) /
(k31w2c29_tpr + k31w2c29_tnr), 6)

k31w3c29_tpr <- round(sum(cm_c29_tables[1, 321:330]) /
(sum(cm_c29_tables[1, 321:330]) + sum(cm_c29_tables[2, 321:330])), 6)
k31w3c29_tnr <- round(sum(cm_c29_tables[4, 321:330]) /
(sum(cm_c29_tables[4, 321:330]) + sum(cm_c29_tables[3, 321:330])), 6)
k31w3c29_truescore <- round((2 * k31w3c29_tpr * k31w3c29_tnr) /
(k31w3c29_tpr + k31w3c29_tnr), 6)

k33w1c29_tpr <- round(sum(cm_c29_tables[1, 331:340]) /
(sum(cm_c29_tables[1, 331:340]) + sum(cm_c29_tables[2, 331:340])), 6)
k33w1c29_tnr <- round(sum(cm_c29_tables[4, 331:340]) /
(sum(cm_c29_tables[4, 331:340]) + sum(cm_c29_tables[3, 331:340])), 6)
k33w1c29_truescore <- round((2 * k33w1c29_tpr * k33w1c29_tnr) /
(k33w1c29_tpr + k33w1c29_tnr), 6)

k33w2c29_tpr <- round(sum(cm_c29_tables[1, 341:350]) /
(sum(cm_c29_tables[1, 341:350]) + sum(cm_c29_tables[2, 341:350])), 6)
k33w2c29_tnr <- round(sum(cm_c29_tables[4, 341:350]) /
(sum(cm_c29_tables[4, 341:350]) + sum(cm_c29_tables[3, 341:350])), 6)
k33w2c29_truescore <- round((2 * k33w2c29_tpr * k33w2c29_tnr) /

```

```

(k33w2c29_tpr + k33w2c29_tnr), 6)

k33w3c29_tpr <- round(sum(cm_c29_tables[1, 351:360]) /
  (sum(cm_c29_tables[1, 351:360]) + sum(cm_c29_tables[2, 351:360])), 6)
k33w3c29_tnr <- round(sum(cm_c29_tables[4, 351:360]) /
  (sum(cm_c29_tables[4, 351:360]) + sum(cm_c29_tables[3, 351:360])), 6)
k33w3c29_truescore <- round((2 * k33w3c29_tpr * k33w3c29_tnr) /
  (k33w3c29_tpr + k33w3c29_tnr), 6)

k35w1c29_tpr <- round(sum(cm_c29_tables[1, 361:370]) /
  (sum(cm_c29_tables[1, 361:370]) + sum(cm_c29_tables[2, 361:370])), 6)
k35w1c29_tnr <- round(sum(cm_c29_tables[4, 361:370]) /
  (sum(cm_c29_tables[4, 361:370]) + sum(cm_c29_tables[3, 361:370])), 6)
k35w1c29_truescore <- round((2 * k35w1c29_tpr * k35w1c29_tnr) /
  (k35w1c29_tpr + k35w1c29_tnr), 6)

k35w2c29_tpr <- round(sum(cm_c29_tables[1, 371:380]) /
  (sum(cm_c29_tables[1, 371:380]) + sum(cm_c29_tables[2, 371:380])), 6)
k35w2c29_tnr <- round(sum(cm_c29_tables[4, 371:380]) /
  (sum(cm_c29_tables[4, 371:380]) + sum(cm_c29_tables[3, 371:380])), 6)
k35w2c29_truescore <- round((2 * k35w2c29_tpr * k35w2c29_tnr) /
  (k35w2c29_tpr + k35w2c29_tnr), 6)

k35w3c29_tpr <- round(sum(cm_c29_tables[1, 381:390]) /
  (sum(cm_c29_tables[1, 381:390]) + sum(cm_c29_tables[2, 381:390])), 6)
k35w3c29_tnr <- round(sum(cm_c29_tables[4, 381:390]) /
  (sum(cm_c29_tables[4, 381:390]) + sum(cm_c29_tables[3, 381:390])), 6)
k35w3c29_truescore <- round((2 * k35w3c29_tpr * k35w3c29_tnr) /
  (k35w3c29_tpr + k35w3c29_tnr), 6)

k37w1c29_tpr <- round(sum(cm_c29_tables[1, 391:400]) /
  (sum(cm_c29_tables[1, 391:400]) + sum(cm_c29_tables[2, 391:400])), 6)
k37w1c29_tnr <- round(sum(cm_c29_tables[4, 391:400]) /
  (sum(cm_c29_tables[4, 391:400]) + sum(cm_c29_tables[3, 391:400])), 6)
k37w1c29_truescore <- round((2 * k37w1c29_tpr * k37w1c29_tnr) /
  (k37w1c29_tpr + k37w1c29_tnr), 6)

k37w2c29_tpr <- round(sum(cm_c29_tables[1, 401:410]) /
  (sum(cm_c29_tables[1, 401:410]) + sum(cm_c29_tables[2, 401:410])), 6)
k37w2c29_tnr <- round(sum(cm_c29_tables[4, 401:410]) /
  (sum(cm_c29_tables[4, 401:410]) + sum(cm_c29_tables[3, 401:410])), 6)
k37w2c29_truescore <- round((2 * k37w2c29_tpr * k37w2c29_tnr) /
  (k37w2c29_tpr + k37w2c29_tnr), 6)

k37w3c29_tpr <- round(sum(cm_c29_tables[1, 411:420]) /
  (sum(cm_c29_tables[1, 411:420]) + sum(cm_c29_tables[2, 411:420])), 6)
k37w3c29_tnr <- round(sum(cm_c29_tables[4, 411:420]) /
  (sum(cm_c29_tables[4, 411:420]) + sum(cm_c29_tables[3, 411:420])), 6)
k37w3c29_truescore <- round((2 * k37w3c29_tpr * k37w3c29_tnr) /
  (k37w3c29_tpr + k37w3c29_tnr), 6)

```

```

k39w1c29_tpr <- round(sum(cm_c29_tables[1, 421:430]) /
  (sum(cm_c29_tables[1, 421:430]) + sum(cm_c29_tables[2, 421:430])), 6)
k39w1c29_tnr <- round(sum(cm_c29_tables[4, 421:430]) /
  (sum(cm_c29_tables[4, 421:430]) + sum(cm_c29_tables[3, 421:430])), 6)
k39w1c29_truescore <- round((2 * k39w1c29_tpr * k39w1c29_tnr) /
  (k39w1c29_tpr + k39w1c29_tnr), 6)

k39w2c29_tpr <- round(sum(cm_c29_tables[1, 431:440]) /
  (sum(cm_c29_tables[1, 431:440]) + sum(cm_c29_tables[2, 431:440])), 6)
k39w2c29_tnr <- round(sum(cm_c29_tables[4, 431:440]) /
  (sum(cm_c29_tables[4, 431:440]) + sum(cm_c29_tables[3, 431:440])), 6)
k39w2c29_truescore <- round((2 * k39w2c29_tpr * k39w2c29_tnr) /
  (k39w2c29_tpr + k39w2c29_tnr), 6)

k39w3c29_tpr <- round(sum(cm_c29_tables[1, 441:450]) /
  (sum(cm_c29_tables[1, 441:450]) + sum(cm_c29_tables[2, 441:450])), 6)
k39w3c29_tnr <- round(sum(cm_c29_tables[4, 441:450]) /
  (sum(cm_c29_tables[4, 441:450]) + sum(cm_c29_tables[3, 441:450])), 6)
k39w3c29_truescore <- round((2 * k39w3c29_tpr * k39w3c29_tnr) /
  (k39w3c29_tpr + k39w3c29_tnr), 6)

# Compile the 0.29 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c29_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.29,
  TPR = c(k11w1c29_tpr, k11w2c29_tpr, k11w3c29_tpr, k13w1c29_tpr,
    k13w2c29_tpr, k13w3c29_tpr, k15w1c29_tpr, k15w2c29_tpr,
    k15w3c29_tpr, k17w1c29_tpr, k17w2c29_tpr, k17w3c29_tpr,
    k19w1c29_tpr, k19w2c29_tpr, k19w3c29_tpr, k21w1c29_tpr,
    k21w2c29_tpr, k21w3c29_tpr, k23w1c29_tpr, k23w2c29_tpr,
    k23w3c29_tpr, k25w1c29_tpr, k25w2c29_tpr, k25w3c29_tpr,
    k27w1c29_tpr, k27w2c29_tpr, k27w3c29_tpr, k29w1c29_tpr,
    k29w2c29_tpr, k29w3c29_tpr, k31w1c29_tpr, k31w2c29_tpr,
    k31w3c29_tpr, k33w1c29_tpr, k33w2c29_tpr, k33w3c29_tpr,
    k35w1c29_tpr, k35w2c29_tpr, k35w3c29_tpr, k37w1c29_tpr,
    k37w2c29_tpr, k37w3c29_tpr, k39w1c29_tpr, k39w2c29_tpr,
    k39w3c29_tpr),
  TNR = c(k11w1c29_tnr, k11w2c29_tnr, k11w3c29_tnr, k13w1c29_tnr,
    k13w2c29_tnr, k13w3c29_tnr, k15w1c29_tnr, k15w2c29_tnr,
    k15w3c29_tnr, k17w1c29_tnr, k17w2c29_tnr, k17w3c29_tnr,
    k19w1c29_tnr, k19w2c29_tnr, k19w3c29_tnr, k21w1c29_tnr,
    k21w2c29_tnr, k21w3c29_tnr, k23w1c29_tnr, k23w2c29_tnr,
    k23w3c29_tnr, k25w1c29_tnr, k25w2c29_tnr, k25w3c29_tnr,
    k27w1c29_tnr, k27w2c29_tnr, k27w3c29_tnr, k29w1c29_tnr,
    k29w2c29_tnr, k29w3c29_tnr, k31w1c29_tnr, k31w2c29_tnr,
    k31w3c29_tnr, k33w1c29_tnr, k33w2c29_tnr, k33w3c29_tnr),
  Truescore = k39w1c29_truescore)

```

```

k35w1c29_tnr, k35w2c29_tnr, k35w3c29_tnr, k37w1c29_tnr,
k37w2c29_tnr, k37w3c29_tnr, k39w1c29_tnr, k39w2c29_tnr,
k39w3c29_tnr),
Truescore = c(k11w1c29_truescore, k11w2c29_truescore,
k11w3c29_truescore, k13w1c29_truescore,
k13w2c29_truescore, k13w3c29_truescore,
k15w1c29_truescore, k15w2c29_truescore,
k15w3c29_truescore, k17w1c29_truescore,
k17w2c29_truescore, k17w3c29_truescore,
k19w1c29_truescore, k19w2c29_truescore,
k19w3c29_truescore, k21w1c29_truescore,
k21w2c29_truescore, k21w3c29_truescore,
k23w1c29_truescore, k23w2c29_truescore,
k23w3c29_truescore, k25w1c29_truescore,
k25w2c29_truescore, k25w3c29_truescore,
k27w1c29_truescore, k27w2c29_truescore,
k27w3c29_truescore, k29w1c29_truescore,
k29w2c29_truescore, k29w3c29_truescore,
k31w1c29_truescore, k31w2c29_truescore,
k31w3c29_truescore, k33w1c29_truescore,
k33w2c29_truescore, k33w3c29_truescore,
k35w1c29_truescore, k35w2c29_truescore,
k35w3c29_truescore, k37w1c29_truescore,
k37w2c29_truescore, k37w3c29_truescore,
k39w1c29_truescore, k39w2c29_truescore,
k39w3c29_truescore))

```

`knitr::kable(c29_results[1:45,], caption = "c29_results")`

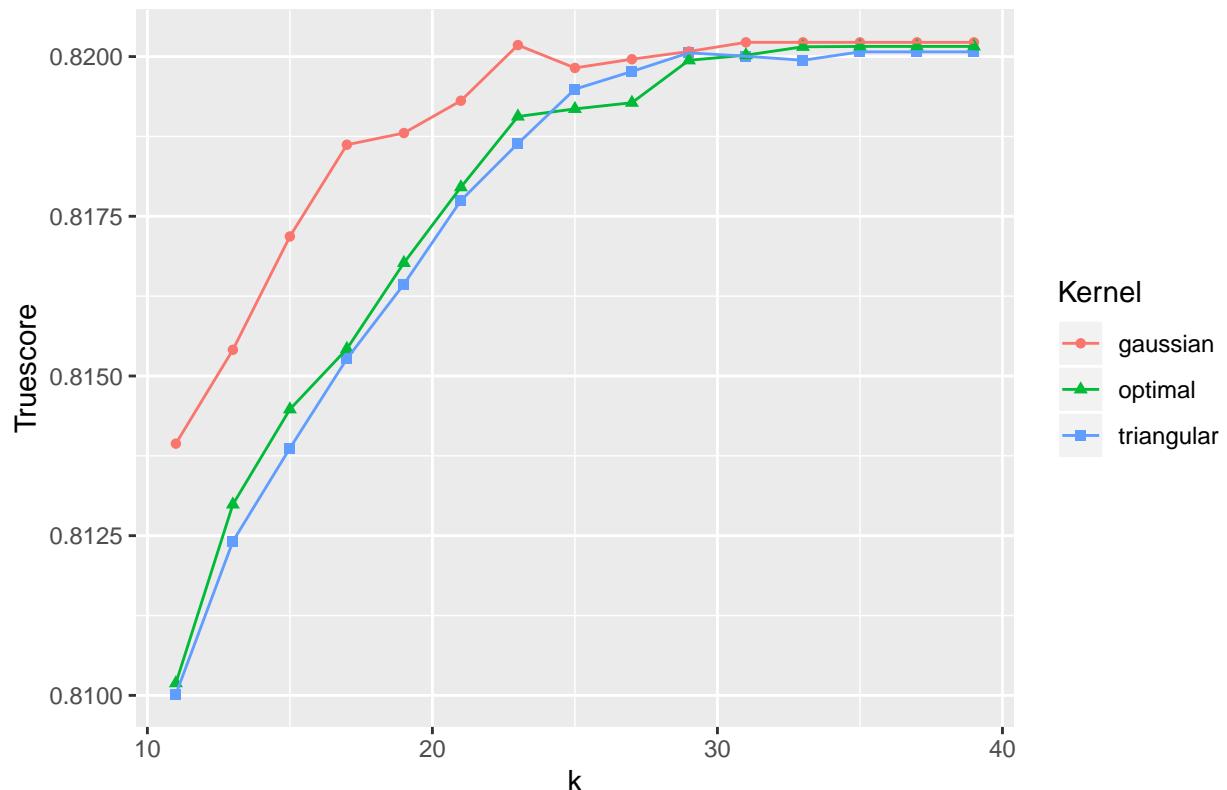
Table 38: c29_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.29	0.817319	0.802844	0.810017
11	gaussian	0.29	0.829920	0.798566	0.813941
11	optimal	0.29	0.817520	0.802993	0.810191
13	triangular	0.29	0.823494	0.801622	0.812411
13	gaussian	0.29	0.833735	0.797873	0.815410
13	optimal	0.29	0.824649	0.801655	0.812989
15	triangular	0.29	0.828012	0.800201	0.813869
15	gaussian	0.29	0.839006	0.796469	0.817184
15	optimal	0.29	0.828564	0.800862	0.814478
17	triangular	0.29	0.831878	0.799310	0.815269
17	gaussian	0.29	0.842520	0.796039	0.818620
17	optimal	0.29	0.832631	0.798913	0.815424
19	triangular	0.29	0.835141	0.798550	0.816436
19	gaussian	0.29	0.843574	0.795445	0.818803
19	optimal	0.29	0.836747	0.797724	0.816770
21	triangular	0.29	0.839056	0.797493	0.817747
21	gaussian	0.29	0.845582	0.794619	0.819309
21	optimal	0.29	0.839458	0.797526	0.817955
23	triangular	0.29	0.841968	0.796568	0.818639
23	gaussian	0.29	0.847440	0.794619	0.820180
23	optimal	0.29	0.842771	0.796650	0.819062

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.29	0.844177	0.796205	0.819490
25	gaussian	0.29	0.847691	0.793727	0.819822
25	optimal	0.29	0.844378	0.795445	0.819181
27	triangular	0.29	0.845231	0.795792	0.819767
27	gaussian	0.29	0.848193	0.793545	0.819959
27	optimal	0.29	0.845181	0.794916	0.819278
29	triangular	0.29	0.846787	0.794966	0.820059
29	gaussian	0.29	0.848394	0.793595	0.820080
29	optimal	0.29	0.846988	0.794569	0.819942
31	triangular	0.29	0.847239	0.794470	0.820006
31	gaussian	0.29	0.849799	0.792637	0.820223
31	optimal	0.29	0.847892	0.793925	0.820022
33	triangular	0.29	0.847590	0.794041	0.819942
33	gaussian	0.29	0.849799	0.792637	0.820223
33	optimal	0.29	0.848343	0.793777	0.820153
35	triangular	0.29	0.848193	0.793760	0.820074
35	gaussian	0.29	0.849799	0.792637	0.820223
35	optimal	0.29	0.848695	0.793479	0.820159
37	triangular	0.29	0.848193	0.793760	0.820074
37	gaussian	0.29	0.849799	0.792637	0.820223
37	optimal	0.29	0.848695	0.793479	0.820159
39	triangular	0.29	0.848193	0.793760	0.820074
39	gaussian	0.29	0.849799	0.792637	0.820223
39	optimal	0.29	0.848695	0.793479	0.820159

```
ggplot(c29_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.29 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.29 (Truescore)

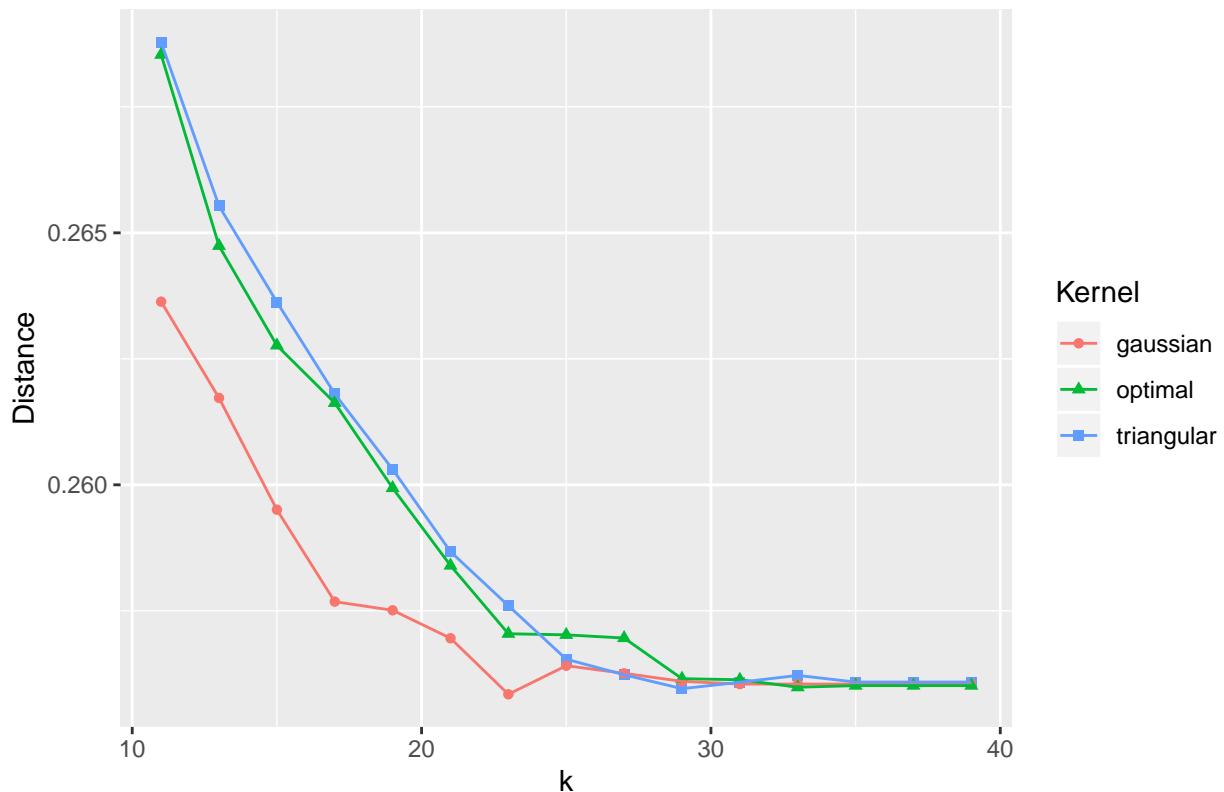


```
# For the Cutoff of 0.29, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c29_results <- c29_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c29_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.29 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.29 (Distance)



```
# For the Cutoff of 0.29, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c29_results$Truescore)
```

```
## [1] 0.820223
```

```
(c29_opt_k_ts <- c29_results$k[which.max(c29_results$Truescore)])
```

```
## [1] 31
```

```
(c29_opt_kernel_ts <- c29_results$Kernel[which.max(c29_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c29_opt_cut_ts <- c29_results$Cut[which.max(c29_results$Truescore)])
```

```
## [1] 0.29
```

```
(c29_opt_tpr_ts <- c29_results$TPR[which.max(c29_results$Truescore)])
```

```
## [1] 0.849799
```

```

(c29_opt_tnr_ts <- c29_results$TNR[which.max(c29_results$Truescore)])

## [1] 0.792637

(c29_opt_d_ts <- c29_results$Distance[which.max(c29_results$Truescore)])

## [1] 0.2560464

min(c29_results$Distance)

## [1] 0.2558435

(c29_opt_k_dist <- c29_results$k[which.min(c29_results$Distance)])

## [1] 23

(c29_opt_kernel_dist <- c29_results$Kernel[which.min(c29_results$Distance)])

## [1] "gaussian"

(c29_opt_cut_dist <- c29_results$Cut[which.min(c29_results$Distance)])

## [1] 0.29

(c29_opt_tpr_dist <- c29_results$TPR[which.min(c29_results$Distance)])

## [1] 0.84744

(c29_opt_tnr_dist <- c29_results$TNR[which.min(c29_results$Distance)])

## [1] 0.794619

(c29_opt_t_dist <- c29_results$Truescore[which.min(c29_results$Distance)])

## [1] 0.82018

#####
# 0.30 Cutoff
#####

# For the decision cutoff of 0.30, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c30 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred30, obs))
  confusionMatrix(ss$pred30, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c30) <- kwf_dfs_v

cm_c30_tables <- sapply(cm_c30, "[[", 2)
cm_c30_tables <- as_tibble(cm_c30_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.30.

k11w1c30_tpr <- round(sum(cm_c30_tables[1, 1:10]) /
  (sum(cm_c30_tables[1, 1:10]) + sum(cm_c30_tables[2, 1:10])), 6)
k11w1c30_tnr <- round(sum(cm_c30_tables[4, 1:10]) /
  (sum(cm_c30_tables[4, 1:10]) + sum(cm_c30_tables[3, 1:10])), 6)
k11w1c30_truescore <- round((2 * k11w1c30_tpr * k11w1c30_tnr) /
  (k11w1c30_tpr + k11w1c30_tnr), 6)

k11w2c30_tpr <- round(sum(cm_c30_tables[1, 11:20]) /
  (sum(cm_c30_tables[1, 11:20]) + sum(cm_c30_tables[2, 11:20])), 6)
k11w2c30_tnr <- round(sum(cm_c30_tables[4, 11:20]) /
  (sum(cm_c30_tables[4, 11:20]) + sum(cm_c30_tables[3, 11:20])), 6)
k11w2c30_truescore <- round((2 * k11w2c30_tpr * k11w2c30_tnr) /
  (k11w2c30_tpr + k11w2c30_tnr), 6)

k11w3c30_tpr <- round(sum(cm_c30_tables[1, 21:30]) /
  (sum(cm_c30_tables[1, 21:30]) + sum(cm_c30_tables[2, 21:30])), 6)
k11w3c30_tnr <- round(sum(cm_c30_tables[4, 21:30]) /
  (sum(cm_c30_tables[4, 21:30]) + sum(cm_c30_tables[3, 21:30])), 6)
k11w3c30_truescore <- round((2 * k11w3c30_tpr * k11w3c30_tnr) /
  (k11w3c30_tpr + k11w3c30_tnr), 6)

k13w1c30_tpr <- round(sum(cm_c30_tables[1, 31:40]) /
  (sum(cm_c30_tables[1, 31:40]) + sum(cm_c30_tables[2, 31:40])), 6)
k13w1c30_tnr <- round(sum(cm_c30_tables[4, 31:40]) /
  (sum(cm_c30_tables[4, 31:40]) + sum(cm_c30_tables[3, 31:40])), 6)
k13w1c30_truescore <- round((2 * k13w1c30_tpr * k13w1c30_tnr) /
  (k13w1c30_tpr + k13w1c30_tnr), 6)

k13w2c30_tpr <- round(sum(cm_c30_tables[1, 41:50]) /
  (sum(cm_c30_tables[1, 41:50]) + sum(cm_c30_tables[2, 41:50])), 6)
k13w2c30_tnr <- round(sum(cm_c30_tables[4, 41:50]) /
  (sum(cm_c30_tables[4, 41:50]) + sum(cm_c30_tables[3, 41:50])), 6)
k13w2c30_truescore <- round((2 * k13w2c30_tpr * k13w2c30_tnr) /
  (k13w2c30_tpr + k13w2c30_tnr), 6)

k13w3c30_tpr <- round(sum(cm_c30_tables[1, 51:60]) /
  (sum(cm_c30_tables[1, 51:60]) + sum(cm_c30_tables[2, 51:60])), 6)
k13w3c30_tnr <- round(sum(cm_c30_tables[4, 51:60]) /
  (sum(cm_c30_tables[4, 51:60]) + sum(cm_c30_tables[3, 51:60])), 6)
k13w3c30_truescore <- round((2 * k13w3c30_tpr * k13w3c30_tnr) /

```

```

(k13w3c30_tpr + k13w3c30_tnr), 6)

k15w1c30_tpr <- round(sum(cm_c30_tables[1, 61:70]) /
  (sum(cm_c30_tables[1, 61:70]) + sum(cm_c30_tables[2, 61:70])), 6)
k15w1c30_tnr <- round(sum(cm_c30_tables[4, 61:70]) /
  (sum(cm_c30_tables[4, 61:70]) + sum(cm_c30_tables[3, 61:70])), 6)
k15w1c30_truescore <- round((2 * k15w1c30_tpr * k15w1c30_tnr) /
  (k15w1c30_tpr + k15w1c30_tnr), 6)

k15w2c30_tpr <- round(sum(cm_c30_tables[1, 71:80]) /
  (sum(cm_c30_tables[1, 71:80]) + sum(cm_c30_tables[2, 71:80])), 6)
k15w2c30_tnr <- round(sum(cm_c30_tables[4, 71:80]) /
  (sum(cm_c30_tables[4, 71:80]) + sum(cm_c30_tables[3, 71:80])), 6)
k15w2c30_truescore <- round((2 * k15w2c30_tpr * k15w2c30_tnr) /
  (k15w2c30_tpr + k15w2c30_tnr), 6)

k15w3c30_tpr <- round(sum(cm_c30_tables[1, 81:90]) /
  (sum(cm_c30_tables[1, 81:90]) + sum(cm_c30_tables[2, 81:90])), 6)
k15w3c30_tnr <- round(sum(cm_c30_tables[4, 81:90]) /
  (sum(cm_c30_tables[4, 81:90]) + sum(cm_c30_tables[3, 81:90])), 6)
k15w3c30_truescore <- round((2 * k15w3c30_tpr * k15w3c30_tnr) /
  (k15w3c30_tpr + k15w3c30_tnr), 6)

k17w1c30_tpr <- round(sum(cm_c30_tables[1, 91:100]) /
  (sum(cm_c30_tables[1, 91:100]) + sum(cm_c30_tables[2, 91:100])), 6)
k17w1c30_tnr <- round(sum(cm_c30_tables[4, 91:100]) /
  (sum(cm_c30_tables[4, 91:100]) + sum(cm_c30_tables[3, 91:100])), 6)
k17w1c30_truescore <- round((2 * k17w1c30_tpr * k17w1c30_tnr) /
  (k17w1c30_tpr + k17w1c30_tnr), 6)

k17w2c30_tpr <- round(sum(cm_c30_tables[1, 101:110]) /
  (sum(cm_c30_tables[1, 101:110]) + sum(cm_c30_tables[2, 101:110])), 6)
k17w2c30_tnr <- round(sum(cm_c30_tables[4, 101:110]) /
  (sum(cm_c30_tables[4, 101:110]) + sum(cm_c30_tables[3, 101:110])), 6)
k17w2c30_truescore <- round((2 * k17w2c30_tpr * k17w2c30_tnr) /
  (k17w2c30_tpr + k17w2c30_tnr), 6)

k17w3c30_tpr <- round(sum(cm_c30_tables[1, 111:120]) /
  (sum(cm_c30_tables[1, 111:120]) + sum(cm_c30_tables[2, 111:120])), 6)
k17w3c30_tnr <- round(sum(cm_c30_tables[4, 111:120]) /
  (sum(cm_c30_tables[4, 111:120]) + sum(cm_c30_tables[3, 111:120])), 6)
k17w3c30_truescore <- round((2 * k17w3c30_tpr * k17w3c30_tnr) /
  (k17w3c30_tpr + k17w3c30_tnr), 6)

k19w1c30_tpr <- round(sum(cm_c30_tables[1, 121:130]) /
  (sum(cm_c30_tables[1, 121:130]) + sum(cm_c30_tables[2, 121:130])), 6)
k19w1c30_tnr <- round(sum(cm_c30_tables[4, 121:130]) /
  (sum(cm_c30_tables[4, 121:130]) + sum(cm_c30_tables[3, 121:130])), 6)
k19w1c30_truescore <- round((2 * k19w1c30_tpr * k19w1c30_tnr) /
  (k19w1c30_tpr + k19w1c30_tnr), 6)

```

```

k19w2c30_tpr <- round(sum(cm_c30_tables[1, 131:140]) /
  (sum(cm_c30_tables[1, 131:140]) + sum(cm_c30_tables[2, 131:140])), 6)
k19w2c30_tnr <- round(sum(cm_c30_tables[4, 131:140]) /
  (sum(cm_c30_tables[4, 131:140]) + sum(cm_c30_tables[3, 131:140])), 6)
k19w2c30_truescore <- round((2 * k19w2c30_tpr * k19w2c30_tnr) /
  (k19w2c30_tpr + k19w2c30_tnr), 6)

k19w3c30_tpr <- round(sum(cm_c30_tables[1, 141:150]) /
  (sum(cm_c30_tables[1, 141:150]) + sum(cm_c30_tables[2, 141:150])), 6)
k19w3c30_tnr <- round(sum(cm_c30_tables[4, 141:150]) /
  (sum(cm_c30_tables[4, 141:150]) + sum(cm_c30_tables[3, 141:150])), 6)
k19w3c30_truescore <- round((2 * k19w3c30_tpr * k19w3c30_tnr) /
  (k19w3c30_tpr + k19w3c30_tnr), 6)

k21w1c30_tpr <- round(sum(cm_c30_tables[1, 151:160]) /
  (sum(cm_c30_tables[1, 151:160]) + sum(cm_c30_tables[2, 151:160])), 6)
k21w1c30_tnr <- round(sum(cm_c30_tables[4, 151:160]) /
  (sum(cm_c30_tables[4, 151:160]) + sum(cm_c30_tables[3, 151:160])), 6)
k21w1c30_truescore <- round((2 * k21w1c30_tpr * k21w1c30_tnr) /
  (k21w1c30_tpr + k21w1c30_tnr), 6)

k21w2c30_tpr <- round(sum(cm_c30_tables[1, 161:170]) /
  (sum(cm_c30_tables[1, 161:170]) + sum(cm_c30_tables[2, 161:170])), 6)
k21w2c30_tnr <- round(sum(cm_c30_tables[4, 161:170]) /
  (sum(cm_c30_tables[4, 161:170]) + sum(cm_c30_tables[3, 161:170])), 6)
k21w2c30_truescore <- round((2 * k21w2c30_tpr * k21w2c30_tnr) /
  (k21w2c30_tpr + k21w2c30_tnr), 6)

k21w3c30_tpr <- round(sum(cm_c30_tables[1, 171:180]) /
  (sum(cm_c30_tables[1, 171:180]) + sum(cm_c30_tables[2, 171:180])), 6)
k21w3c30_tnr <- round(sum(cm_c30_tables[4, 171:180]) /
  (sum(cm_c30_tables[4, 171:180]) + sum(cm_c30_tables[3, 171:180])), 6)
k21w3c30_truescore <- round((2 * k21w3c30_tpr * k21w3c30_tnr) /
  (k21w3c30_tpr + k21w3c30_tnr), 6)

k23w1c30_tpr <- round(sum(cm_c30_tables[1, 181:190]) /
  (sum(cm_c30_tables[1, 181:190]) + sum(cm_c30_tables[2, 181:190])), 6)
k23w1c30_tnr <- round(sum(cm_c30_tables[4, 181:190]) /
  (sum(cm_c30_tables[4, 181:190]) + sum(cm_c30_tables[3, 181:190])), 6)
k23w1c30_truescore <- round((2 * k23w1c30_tpr * k23w1c30_tnr) /
  (k23w1c30_tpr + k23w1c30_tnr), 6)

k23w2c30_tpr <- round(sum(cm_c30_tables[1, 191:200]) /
  (sum(cm_c30_tables[1, 191:200]) + sum(cm_c30_tables[2, 191:200])), 6)
k23w2c30_tnr <- round(sum(cm_c30_tables[4, 191:200]) /
  (sum(cm_c30_tables[4, 191:200]) + sum(cm_c30_tables[3, 191:200])), 6)
k23w2c30_truescore <- round((2 * k23w2c30_tpr * k23w2c30_tnr) /
  (k23w2c30_tpr + k23w2c30_tnr), 6)

k23w3c30_tpr <- round(sum(cm_c30_tables[1, 201:210]) /
  (sum(cm_c30_tables[1, 201:210]) + sum(cm_c30_tables[2, 201:210])), 6)

```

```

k23w3c30_tnr <- round(sum(cm_c30_tables[4, 201:210]) /
  (sum(cm_c30_tables[4, 201:210]) + sum(cm_c30_tables[3, 201:210])), 6)
k23w3c30_truescore <- round((2 * k23w3c30_tpr * k23w3c30_tnr) /
  (k23w3c30_tpr + k23w3c30_tnr), 6)

k25w1c30_tpr <- round(sum(cm_c30_tables[1, 211:220]) /
  (sum(cm_c30_tables[1, 211:220]) + sum(cm_c30_tables[2, 211:220])), 6)
k25w1c30_tnr <- round(sum(cm_c30_tables[4, 211:220]) /
  (sum(cm_c30_tables[4, 211:220]) + sum(cm_c30_tables[3, 211:220])), 6)
k25w1c30_truescore <- round((2 * k25w1c30_tpr * k25w1c30_tnr) /
  (k25w1c30_tpr + k25w1c30_tnr), 6)

k25w2c30_tpr <- round(sum(cm_c30_tables[1, 221:230]) /
  (sum(cm_c30_tables[1, 221:230]) + sum(cm_c30_tables[2, 221:230])), 6)
k25w2c30_tnr <- round(sum(cm_c30_tables[4, 221:230]) /
  (sum(cm_c30_tables[4, 221:230]) + sum(cm_c30_tables[3, 221:230])), 6)
k25w2c30_truescore <- round((2 * k25w2c30_tpr * k25w2c30_tnr) /
  (k25w2c30_tpr + k25w2c30_tnr), 6)

k25w3c30_tpr <- round(sum(cm_c30_tables[1, 231:240]) /
  (sum(cm_c30_tables[1, 231:240]) + sum(cm_c30_tables[2, 231:240])), 6)
k25w3c30_tnr <- round(sum(cm_c30_tables[4, 231:240]) /
  (sum(cm_c30_tables[4, 231:240]) + sum(cm_c30_tables[3, 231:240])), 6)
k25w3c30_truescore <- round((2 * k25w3c30_tpr * k25w3c30_tnr) /
  (k25w3c30_tpr + k25w3c30_tnr), 6)

k27w1c30_tpr <- round(sum(cm_c30_tables[1, 241:250]) /
  (sum(cm_c30_tables[1, 241:250]) + sum(cm_c30_tables[2, 241:250])), 6)
k27w1c30_tnr <- round(sum(cm_c30_tables[4, 241:250]) /
  (sum(cm_c30_tables[4, 241:250]) + sum(cm_c30_tables[3, 241:250])), 6)
k27w1c30_truescore <- round((2 * k27w1c30_tpr * k27w1c30_tnr) /
  (k27w1c30_tpr + k27w1c30_tnr), 6)

k27w2c30_tpr <- round(sum(cm_c30_tables[1, 251:260]) /
  (sum(cm_c30_tables[1, 251:260]) + sum(cm_c30_tables[2, 251:260])), 6)
k27w2c30_tnr <- round(sum(cm_c30_tables[4, 251:260]) /
  (sum(cm_c30_tables[4, 251:260]) + sum(cm_c30_tables[3, 251:260])), 6)
k27w2c30_truescore <- round((2 * k27w2c30_tpr * k27w2c30_tnr) /
  (k27w2c30_tpr + k27w2c30_tnr), 6)

k27w3c30_tpr <- round(sum(cm_c30_tables[1, 261:270]) /
  (sum(cm_c30_tables[1, 261:270]) + sum(cm_c30_tables[2, 261:270])), 6)
k27w3c30_tnr <- round(sum(cm_c30_tables[4, 261:270]) /
  (sum(cm_c30_tables[4, 261:270]) + sum(cm_c30_tables[3, 261:270])), 6)
k27w3c30_truescore <- round((2 * k27w3c30_tpr * k27w3c30_tnr) /
  (k27w3c30_tpr + k27w3c30_tnr), 6)

k29w1c30_tpr <- round(sum(cm_c30_tables[1, 271:280]) /
  (sum(cm_c30_tables[1, 271:280]) + sum(cm_c30_tables[2, 271:280])), 6)
k29w1c30_tnr <- round(sum(cm_c30_tables[4, 271:280]) /

```

```

(sum(cm_c30_tables[4, 271:280]) + sum(cm_c30_tables[3, 271:280])), 6)
k29w1c30_truescore <- round((2 * k29w1c30_tpr * k29w1c30_tnr) /
(k29w1c30_tpr + k29w1c30_tnr), 6)

k29w2c30_tpr <- round(sum(cm_c30_tables[1, 281:290]) /
(sum(cm_c30_tables[1, 281:290]) + sum(cm_c30_tables[2, 281:290])), 6)
k29w2c30_tnr <- round(sum(cm_c30_tables[4, 281:290]) /
(sum(cm_c30_tables[4, 281:290]) + sum(cm_c30_tables[3, 281:290])), 6)
k29w2c30_truescore <- round((2 * k29w2c30_tpr * k29w2c30_tnr) /
(k29w2c30_tpr + k29w2c30_tnr), 6)

k29w3c30_tpr <- round(sum(cm_c30_tables[1, 291:300]) /
(sum(cm_c30_tables[1, 291:300]) + sum(cm_c30_tables[2, 291:300])), 6)
k29w3c30_tnr <- round(sum(cm_c30_tables[4, 291:300]) /
(sum(cm_c30_tables[4, 291:300]) + sum(cm_c30_tables[3, 291:300])), 6)
k29w3c30_truescore <- round((2 * k29w3c30_tpr * k29w3c30_tnr) /
(k29w3c30_tpr + k29w3c30_tnr), 6)

k31w1c30_tpr <- round(sum(cm_c30_tables[1, 301:310]) /
(sum(cm_c30_tables[1, 301:310]) + sum(cm_c30_tables[2, 301:310])), 6)
k31w1c30_tnr <- round(sum(cm_c30_tables[4, 301:310]) /
(sum(cm_c30_tables[4, 301:310]) + sum(cm_c30_tables[3, 301:310])), 6)
k31w1c30_truescore <- round((2 * k31w1c30_tpr * k31w1c30_tnr) /
(k31w1c30_tpr + k31w1c30_tnr), 6)

k31w2c30_tpr <- round(sum(cm_c30_tables[1, 311:320]) /
(sum(cm_c30_tables[1, 311:320]) + sum(cm_c30_tables[2, 311:320])), 6)
k31w2c30_tnr <- round(sum(cm_c30_tables[4, 311:320]) /
(sum(cm_c30_tables[4, 311:320]) + sum(cm_c30_tables[3, 311:320])), 6)
k31w2c30_truescore <- round((2 * k31w2c30_tpr * k31w2c30_tnr) /
(k31w2c30_tpr + k31w2c30_tnr), 6)

k31w3c30_tpr <- round(sum(cm_c30_tables[1, 321:330]) /
(sum(cm_c30_tables[1, 321:330]) + sum(cm_c30_tables[2, 321:330])), 6)
k31w3c30_tnr <- round(sum(cm_c30_tables[4, 321:330]) /
(sum(cm_c30_tables[4, 321:330]) + sum(cm_c30_tables[3, 321:330])), 6)
k31w3c30_truescore <- round((2 * k31w3c30_tpr * k31w3c30_tnr) /
(k31w3c30_tpr + k31w3c30_tnr), 6)

k33w1c30_tpr <- round(sum(cm_c30_tables[1, 331:340]) /
(sum(cm_c30_tables[1, 331:340]) + sum(cm_c30_tables[2, 331:340])), 6)
k33w1c30_tnr <- round(sum(cm_c30_tables[4, 331:340]) /
(sum(cm_c30_tables[4, 331:340]) + sum(cm_c30_tables[3, 331:340])), 6)
k33w1c30_truescore <- round((2 * k33w1c30_tpr * k33w1c30_tnr) /
(k33w1c30_tpr + k33w1c30_tnr), 6)

k33w2c30_tpr <- round(sum(cm_c30_tables[1, 341:350]) /
(sum(cm_c30_tables[1, 341:350]) + sum(cm_c30_tables[2, 341:350])), 6)
k33w2c30_tnr <- round(sum(cm_c30_tables[4, 341:350]) /
(sum(cm_c30_tables[4, 341:350]) + sum(cm_c30_tables[3, 341:350])), 6)
k33w2c30_truescore <- round((2 * k33w2c30_tpr * k33w2c30_tnr) /

```

```

(k33w2c30_tpr + k33w2c30_tnr), 6)

k33w3c30_tpr <- round(sum(cm_c30_tables[1, 351:360]) /
  (sum(cm_c30_tables[1, 351:360]) + sum(cm_c30_tables[2, 351:360])), 6)
k33w3c30_tnr <- round(sum(cm_c30_tables[4, 351:360]) /
  (sum(cm_c30_tables[4, 351:360]) + sum(cm_c30_tables[3, 351:360])), 6)
k33w3c30_truescore <- round((2 * k33w3c30_tpr * k33w3c30_tnr) /
  (k33w3c30_tpr + k33w3c30_tnr), 6)

k35w1c30_tpr <- round(sum(cm_c30_tables[1, 361:370]) /
  (sum(cm_c30_tables[1, 361:370]) + sum(cm_c30_tables[2, 361:370])), 6)
k35w1c30_tnr <- round(sum(cm_c30_tables[4, 361:370]) /
  (sum(cm_c30_tables[4, 361:370]) + sum(cm_c30_tables[3, 361:370])), 6)
k35w1c30_truescore <- round((2 * k35w1c30_tpr * k35w1c30_tnr) /
  (k35w1c30_tpr + k35w1c30_tnr), 6)

k35w2c30_tpr <- round(sum(cm_c30_tables[1, 371:380]) /
  (sum(cm_c30_tables[1, 371:380]) + sum(cm_c30_tables[2, 371:380])), 6)
k35w2c30_tnr <- round(sum(cm_c30_tables[4, 371:380]) /
  (sum(cm_c30_tables[4, 371:380]) + sum(cm_c30_tables[3, 371:380])), 6)
k35w2c30_truescore <- round((2 * k35w2c30_tpr * k35w2c30_tnr) /
  (k35w2c30_tpr + k35w2c30_tnr), 6)

k35w3c30_tpr <- round(sum(cm_c30_tables[1, 381:390]) /
  (sum(cm_c30_tables[1, 381:390]) + sum(cm_c30_tables[2, 381:390])), 6)
k35w3c30_tnr <- round(sum(cm_c30_tables[4, 381:390]) /
  (sum(cm_c30_tables[4, 381:390]) + sum(cm_c30_tables[3, 381:390])), 6)
k35w3c30_truescore <- round((2 * k35w3c30_tpr * k35w3c30_tnr) /
  (k35w3c30_tpr + k35w3c30_tnr), 6)

k37w1c30_tpr <- round(sum(cm_c30_tables[1, 391:400]) /
  (sum(cm_c30_tables[1, 391:400]) + sum(cm_c30_tables[2, 391:400])), 6)
k37w1c30_tnr <- round(sum(cm_c30_tables[4, 391:400]) /
  (sum(cm_c30_tables[4, 391:400]) + sum(cm_c30_tables[3, 391:400])), 6)
k37w1c30_truescore <- round((2 * k37w1c30_tpr * k37w1c30_tnr) /
  (k37w1c30_tpr + k37w1c30_tnr), 6)

k37w2c30_tpr <- round(sum(cm_c30_tables[1, 401:410]) /
  (sum(cm_c30_tables[1, 401:410]) + sum(cm_c30_tables[2, 401:410])), 6)
k37w2c30_tnr <- round(sum(cm_c30_tables[4, 401:410]) /
  (sum(cm_c30_tables[4, 401:410]) + sum(cm_c30_tables[3, 401:410])), 6)
k37w2c30_truescore <- round((2 * k37w2c30_tpr * k37w2c30_tnr) /
  (k37w2c30_tpr + k37w2c30_tnr), 6)

k37w3c30_tpr <- round(sum(cm_c30_tables[1, 411:420]) /
  (sum(cm_c30_tables[1, 411:420]) + sum(cm_c30_tables[2, 411:420])), 6)
k37w3c30_tnr <- round(sum(cm_c30_tables[4, 411:420]) /
  (sum(cm_c30_tables[4, 411:420]) + sum(cm_c30_tables[3, 411:420])), 6)
k37w3c30_truescore <- round((2 * k37w3c30_tpr * k37w3c30_tnr) /
  (k37w3c30_tpr + k37w3c30_tnr), 6)

```

```

k39w1c30_tpr <- round(sum(cm_c30_tables[1, 421:430]) /
  (sum(cm_c30_tables[1, 421:430]) + sum(cm_c30_tables[2, 421:430])), 6)
k39w1c30_tnr <- round(sum(cm_c30_tables[4, 421:430]) /
  (sum(cm_c30_tables[4, 421:430]) + sum(cm_c30_tables[3, 421:430])), 6)
k39w1c30_truescore <- round((2 * k39w1c30_tpr * k39w1c30_tnr) /
  (k39w1c30_tpr + k39w1c30_tnr), 6)

k39w2c30_tpr <- round(sum(cm_c30_tables[1, 431:440]) /
  (sum(cm_c30_tables[1, 431:440]) + sum(cm_c30_tables[2, 431:440])), 6)
k39w2c30_tnr <- round(sum(cm_c30_tables[4, 431:440]) /
  (sum(cm_c30_tables[4, 431:440]) + sum(cm_c30_tables[3, 431:440])), 6)
k39w2c30_truescore <- round((2 * k39w2c30_tpr * k39w2c30_tnr) /
  (k39w2c30_tpr + k39w2c30_tnr), 6)

k39w3c30_tpr <- round(sum(cm_c30_tables[1, 441:450]) /
  (sum(cm_c30_tables[1, 441:450]) + sum(cm_c30_tables[2, 441:450])), 6)
k39w3c30_tnr <- round(sum(cm_c30_tables[4, 441:450]) /
  (sum(cm_c30_tables[4, 441:450]) + sum(cm_c30_tables[3, 441:450])), 6)
k39w3c30_truescore <- round((2 * k39w3c30_tpr * k39w3c30_tnr) /
  (k39w3c30_tpr + k39w3c30_tnr), 6)

# Compile the 0.30 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c30_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.30,
  TPR = c(k11w1c30_tpr, k11w2c30_tpr, k11w3c30_tpr, k13w1c30_tpr,
    k13w2c30_tpr, k13w3c30_tpr, k15w1c30_tpr, k15w2c30_tpr,
    k15w3c30_tpr, k17w1c30_tpr, k17w2c30_tpr, k17w3c30_tpr,
    k19w1c30_tpr, k19w2c30_tpr, k19w3c30_tpr, k21w1c30_tpr,
    k21w2c30_tpr, k21w3c30_tpr, k23w1c30_tpr, k23w2c30_tpr,
    k23w3c30_tpr, k25w1c30_tpr, k25w2c30_tpr, k25w3c30_tpr,
    k27w1c30_tpr, k27w2c30_tpr, k27w3c30_tpr, k29w1c30_tpr,
    k29w2c30_tpr, k29w3c30_tpr, k31w1c30_tpr, k31w2c30_tpr,
    k31w3c30_tpr, k33w1c30_tpr, k33w2c30_tpr, k33w3c30_tpr,
    k35w1c30_tpr, k35w2c30_tpr, k35w3c30_tpr, k37w1c30_tpr,
    k37w2c30_tpr, k37w3c30_tpr, k39w1c30_tpr, k39w2c30_tpr,
    k39w3c30_tpr),
  TNR = c(k11w1c30_tnr, k11w2c30_tnr, k11w3c30_tnr, k13w1c30_tnr,
    k13w2c30_tnr, k13w3c30_tnr, k15w1c30_tnr, k15w2c30_tnr,
    k15w3c30_tnr, k17w1c30_tnr, k17w2c30_tnr, k17w3c30_tnr,
    k19w1c30_tnr, k19w2c30_tnr, k19w3c30_tnr, k21w1c30_tnr,
    k21w2c30_tnr, k21w3c30_tnr, k23w1c30_tnr, k23w2c30_tnr,
    k23w3c30_tnr, k25w1c30_tnr, k25w2c30_tnr, k25w3c30_tnr,
    k27w1c30_tnr, k27w2c30_tnr, k27w3c30_tnr, k29w1c30_tnr,
    k29w2c30_tnr, k29w3c30_tnr, k31w1c30_tnr, k31w2c30_tnr,
    k31w3c30_tnr, k33w1c30_tnr, k33w2c30_tnr, k33w3c30_tnr)

```

```

k35w1c30_tnr, k35w2c30_tnr, k35w3c30_tnr, k37w1c30_tnr,
k37w2c30_tnr, k37w3c30_tnr, k39w1c30_tnr, k39w2c30_tnr,
k39w3c30_tnr),
Truescore = c(k11w1c30_truescore, k11w2c30_truescore,
k11w3c30_truescore, k13w1c30_truescore,
k13w2c30_truescore, k13w3c30_truescore,
k15w1c30_truescore, k15w2c30_truescore,
k15w3c30_truescore, k17w1c30_truescore,
k17w2c30_truescore, k17w3c30_truescore,
k19w1c30_truescore, k19w2c30_truescore,
k19w3c30_truescore, k21w1c30_truescore,
k21w2c30_truescore, k21w3c30_truescore,
k23w1c30_truescore, k23w2c30_truescore,
k23w3c30_truescore, k25w1c30_truescore,
k25w2c30_truescore, k25w3c30_truescore,
k27w1c30_truescore, k27w2c30_truescore,
k27w3c30_truescore, k29w1c30_truescore,
k29w2c30_truescore, k29w3c30_truescore,
k31w1c30_truescore, k31w2c30_truescore,
k31w3c30_truescore, k33w1c30_truescore,
k33w2c30_truescore, k33w3c30_truescore,
k35w1c30_truescore, k35w2c30_truescore,
k35w3c30_truescore, k37w1c30_truescore,
k37w2c30_truescore, k37w3c30_truescore,
k39w1c30_truescore, k39w2c30_truescore,
k39w3c30_truescore))

```

`knitr::kable(c30_results[1:45,], caption = "c30_results")`

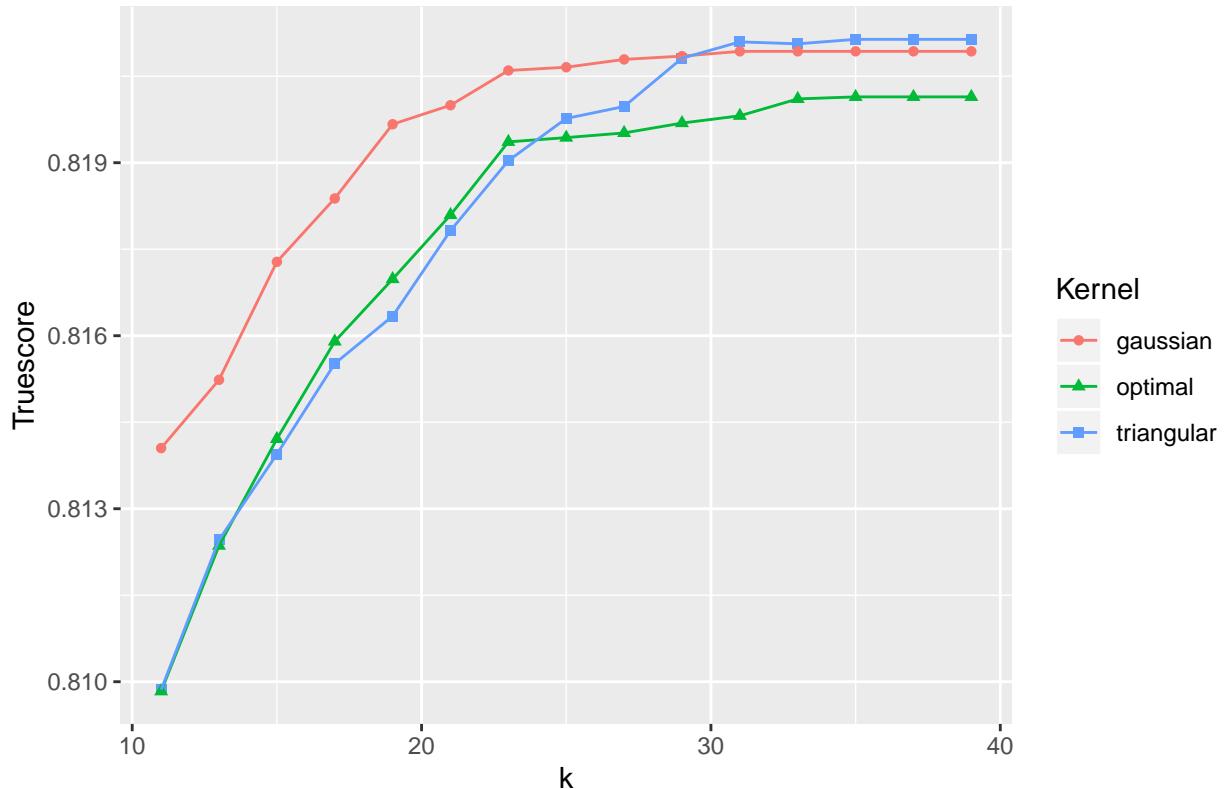
Table 39: c30_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.3	0.810241	0.809484	0.809862
11	gaussian	0.3	0.822490	0.805784	0.814051
11	optimal	0.3	0.809739	0.809930	0.809834
13	triangular	0.3	0.816817	0.808146	0.812458
13	gaussian	0.3	0.825552	0.805173	0.815235
13	optimal	0.3	0.816717	0.808047	0.812359
15	triangular	0.3	0.820884	0.807122	0.813945
15	gaussian	0.3	0.830673	0.804314	0.817281
15	optimal	0.3	0.821185	0.807353	0.814210
17	triangular	0.3	0.824950	0.806296	0.815516
17	gaussian	0.3	0.834137	0.803207	0.818380
17	optimal	0.3	0.825753	0.806280	0.815900
19	triangular	0.3	0.827610	0.805371	0.816339
19	gaussian	0.3	0.837299	0.802762	0.819667
19	optimal	0.3	0.829317	0.805008	0.816982
21	triangular	0.3	0.831827	0.804281	0.817822
21	gaussian	0.3	0.838655	0.802150	0.819996
21	optimal	0.3	0.832229	0.804430	0.818093
23	triangular	0.3	0.834839	0.803819	0.819035
23	gaussian	0.3	0.839608	0.802431	0.820599
23	optimal	0.3	0.835392	0.803934	0.819361

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.3	0.836898	0.803323	0.819767
25	gaussian	0.3	0.840813	0.801440	0.820655
25	optimal	0.3	0.836797	0.802778	0.819435
27	triangular	0.3	0.837851	0.802844	0.819974
27	gaussian	0.3	0.841064	0.801473	0.820791
27	optimal	0.3	0.837651	0.802150	0.819516
29	triangular	0.3	0.840361	0.802150	0.820811
29	gaussian	0.3	0.841165	0.801490	0.820848
29	optimal	0.3	0.838404	0.801787	0.819687
31	triangular	0.3	0.841265	0.801870	0.821095
31	gaussian	0.3	0.842671	0.800284	0.820931
31	optimal	0.3	0.839357	0.801159	0.819813
33	triangular	0.3	0.841466	0.801622	0.821061
33	gaussian	0.3	0.842671	0.800284	0.820931
33	optimal	0.3	0.840261	0.800895	0.820106
35	triangular	0.3	0.842068	0.801226	0.821139
35	gaussian	0.3	0.842671	0.800284	0.820931
35	optimal	0.3	0.840462	0.800780	0.820141
37	triangular	0.3	0.842068	0.801226	0.821139
37	gaussian	0.3	0.842671	0.800284	0.820931
37	optimal	0.3	0.840462	0.800780	0.820141
39	triangular	0.3	0.842068	0.801226	0.821139
39	gaussian	0.3	0.842671	0.800284	0.820931
39	optimal	0.3	0.840462	0.800780	0.820141

```
ggplot(c30_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.30 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.30 (Truescore)

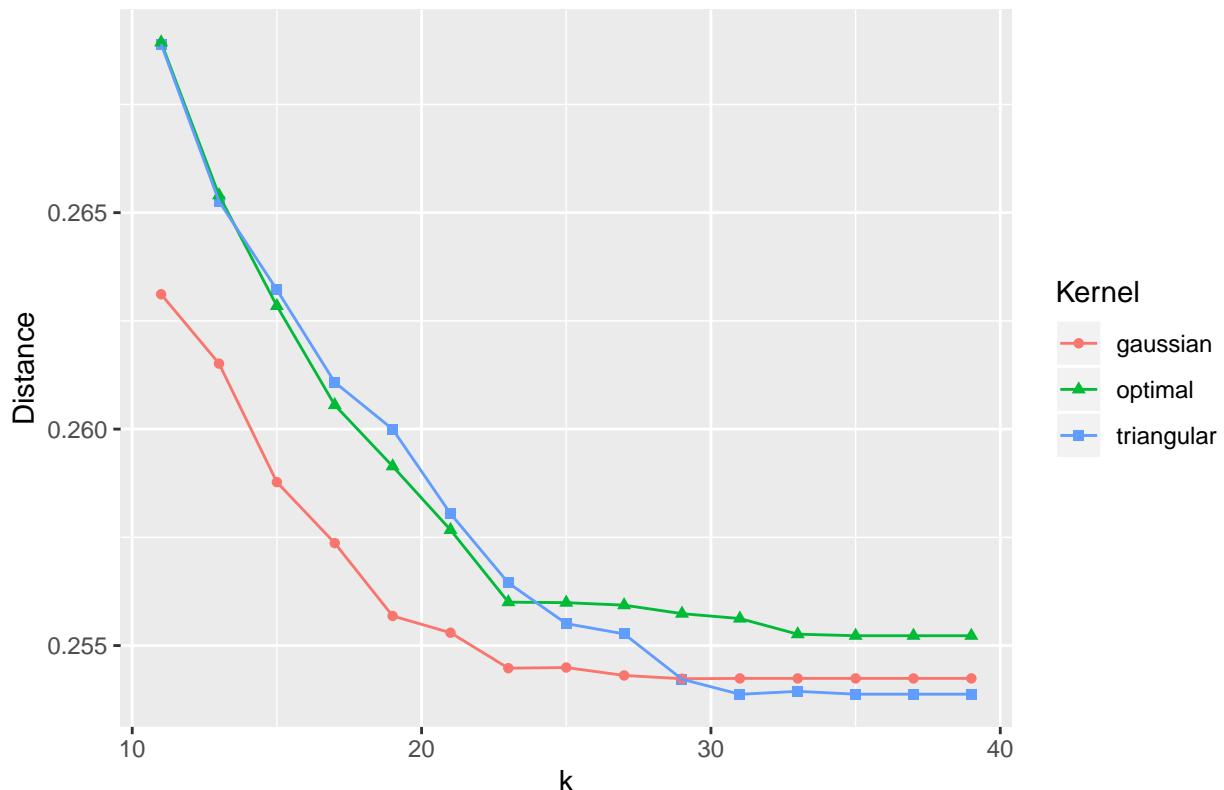


```
# For the Cutoff of 0.30, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c30_results <- c30_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c30_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.30 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.30 (Distance)



```
# For the Cutoff of 0.30, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c30_results$Truescore)
```

```
## [1] 0.821139
```

```
(c30_opt_k_ts <- c30_results$k[which.max(c30_results$Truescore)])
```

```
## [1] 35
```

```
(c30_opt_kernel_ts <- c30_results$Kernel[which.max(c30_results$Truescore)])
```

```
## [1] "triangular"
```

```
(c30_opt_cut_ts <- c30_results$Cut[which.max(c30_results$Truescore)])
```

```
## [1] 0.3
```

```
(c30_opt_tpr_ts <- c30_results$TPR[which.max(c30_results$Truescore)])
```

```
## [1] 0.842068
```

```

(c30_opt_tnr_ts <- c30_results$TNR[which.max(c30_results$Truescore)])

## [1] 0.801226

(c30_opt_d_ts <- c30_results$Distance[which.max(c30_results$Truescore)])

## [1] 0.2538772

min(c30_results$Distance)

## [1] 0.2538746

(c30_opt_k_dist <- c30_results$k[which.min(c30_results$Distance)])

## [1] 31

(c30_opt_kernel_dist <- c30_results$Kernel[which.min(c30_results$Distance)])

## [1] "triangular"

(c30_opt_cut_dist <- c30_results$Cut[which.min(c30_results$Distance)])

## [1] 0.3

(c30_opt_tpr_dist <- c30_results$TPR[which.min(c30_results$Distance)])

## [1] 0.841265

(c30_opt_tnr_dist <- c30_results$TNR[which.min(c30_results$Distance)])

## [1] 0.80187

(c30_opt_t_dist <- c30_results$Truescore[which.min(c30_results$Distance)])

## [1] 0.821095

#####
# 0.31 Cutoff
#####

# For the decision cutoff of 0.31, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c31 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred31, obs))
  confusionMatrix(ss$pred31, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c31) <- kwf_dfs_v

cm_c31_tables <- sapply(cm_c31, "[[", 2)
cm_c31_tables <- as_tibble(cm_c31_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.31.

k11w1c31_tpr <- round(sum(cm_c31_tables[1, 1:10]) /
  (sum(cm_c31_tables[1, 1:10]) + sum(cm_c31_tables[2, 1:10])), 6)
k11w1c31_tnr <- round(sum(cm_c31_tables[4, 1:10]) /
  (sum(cm_c31_tables[4, 1:10]) + sum(cm_c31_tables[3, 1:10])), 6)
k11w1c31_truescore <- round((2 * k11w1c31_tpr * k11w1c31_tnr) /
  (k11w1c31_tpr + k11w1c31_tnr), 6)

k11w2c31_tpr <- round(sum(cm_c31_tables[1, 11:20]) /
  (sum(cm_c31_tables[1, 11:20]) + sum(cm_c31_tables[2, 11:20])), 6)
k11w2c31_tnr <- round(sum(cm_c31_tables[4, 11:20]) /
  (sum(cm_c31_tables[4, 11:20]) + sum(cm_c31_tables[3, 11:20])), 6)
k11w2c31_truescore <- round((2 * k11w2c31_tpr * k11w2c31_tnr) /
  (k11w2c31_tpr + k11w2c31_tnr), 6)

k11w3c31_tpr <- round(sum(cm_c31_tables[1, 21:30]) /
  (sum(cm_c31_tables[1, 21:30]) + sum(cm_c31_tables[2, 21:30])), 6)
k11w3c31_tnr <- round(sum(cm_c31_tables[4, 21:30]) /
  (sum(cm_c31_tables[4, 21:30]) + sum(cm_c31_tables[3, 21:30])), 6)
k11w3c31_truescore <- round((2 * k11w3c31_tpr * k11w3c31_tnr) /
  (k11w3c31_tpr + k11w3c31_tnr), 6)

k13w1c31_tpr <- round(sum(cm_c31_tables[1, 31:40]) /
  (sum(cm_c31_tables[1, 31:40]) + sum(cm_c31_tables[2, 31:40])), 6)
k13w1c31_tnr <- round(sum(cm_c31_tables[4, 31:40]) /
  (sum(cm_c31_tables[4, 31:40]) + sum(cm_c31_tables[3, 31:40])), 6)
k13w1c31_truescore <- round((2 * k13w1c31_tpr * k13w1c31_tnr) /
  (k13w1c31_tpr + k13w1c31_tnr), 6)

k13w2c31_tpr <- round(sum(cm_c31_tables[1, 41:50]) /
  (sum(cm_c31_tables[1, 41:50]) + sum(cm_c31_tables[2, 41:50])), 6)
k13w2c31_tnr <- round(sum(cm_c31_tables[4, 41:50]) /
  (sum(cm_c31_tables[4, 41:50]) + sum(cm_c31_tables[3, 41:50])), 6)
k13w2c31_truescore <- round((2 * k13w2c31_tpr * k13w2c31_tnr) /
  (k13w2c31_tpr + k13w2c31_tnr), 6)

k13w3c31_tpr <- round(sum(cm_c31_tables[1, 51:60]) /
  (sum(cm_c31_tables[1, 51:60]) + sum(cm_c31_tables[2, 51:60])), 6)
k13w3c31_tnr <- round(sum(cm_c31_tables[4, 51:60]) /
  (sum(cm_c31_tables[4, 51:60]) + sum(cm_c31_tables[3, 51:60])), 6)
k13w3c31_truescore <- round((2 * k13w3c31_tpr * k13w3c31_tnr) /

```

```

(k13w3c31_tpr + k13w3c31_tnr), 6)

k15w1c31_tpr <- round(sum(cm_c31_tables[1, 61:70]) /
  (sum(cm_c31_tables[1, 61:70]) + sum(cm_c31_tables[2, 61:70])), 6)
k15w1c31_tnr <- round(sum(cm_c31_tables[4, 61:70]) /
  (sum(cm_c31_tables[4, 61:70]) + sum(cm_c31_tables[3, 61:70])), 6)
k15w1c31_truescore <- round((2 * k15w1c31_tpr * k15w1c31_tnr) /
  (k15w1c31_tpr + k15w1c31_tnr), 6)

k15w2c31_tpr <- round(sum(cm_c31_tables[1, 71:80]) /
  (sum(cm_c31_tables[1, 71:80]) + sum(cm_c31_tables[2, 71:80])), 6)
k15w2c31_tnr <- round(sum(cm_c31_tables[4, 71:80]) /
  (sum(cm_c31_tables[4, 71:80]) + sum(cm_c31_tables[3, 71:80])), 6)
k15w2c31_truescore <- round((2 * k15w2c31_tpr * k15w2c31_tnr) /
  (k15w2c31_tpr + k15w2c31_tnr), 6)

k15w3c31_tpr <- round(sum(cm_c31_tables[1, 81:90]) /
  (sum(cm_c31_tables[1, 81:90]) + sum(cm_c31_tables[2, 81:90])), 6)
k15w3c31_tnr <- round(sum(cm_c31_tables[4, 81:90]) /
  (sum(cm_c31_tables[4, 81:90]) + sum(cm_c31_tables[3, 81:90])), 6)
k15w3c31_truescore <- round((2 * k15w3c31_tpr * k15w3c31_tnr) /
  (k15w3c31_tpr + k15w3c31_tnr), 6)

k17w1c31_tpr <- round(sum(cm_c31_tables[1, 91:100]) /
  (sum(cm_c31_tables[1, 91:100]) + sum(cm_c31_tables[2, 91:100])), 6)
k17w1c31_tnr <- round(sum(cm_c31_tables[4, 91:100]) /
  (sum(cm_c31_tables[4, 91:100]) + sum(cm_c31_tables[3, 91:100])), 6)
k17w1c31_truescore <- round((2 * k17w1c31_tpr * k17w1c31_tnr) /
  (k17w1c31_tpr + k17w1c31_tnr), 6)

k17w2c31_tpr <- round(sum(cm_c31_tables[1, 101:110]) /
  (sum(cm_c31_tables[1, 101:110]) + sum(cm_c31_tables[2, 101:110])), 6)
k17w2c31_tnr <- round(sum(cm_c31_tables[4, 101:110]) /
  (sum(cm_c31_tables[4, 101:110]) + sum(cm_c31_tables[3, 101:110])), 6)
k17w2c31_truescore <- round((2 * k17w2c31_tpr * k17w2c31_tnr) /
  (k17w2c31_tpr + k17w2c31_tnr), 6)

k17w3c31_tpr <- round(sum(cm_c31_tables[1, 111:120]) /
  (sum(cm_c31_tables[1, 111:120]) + sum(cm_c31_tables[2, 111:120])), 6)
k17w3c31_tnr <- round(sum(cm_c31_tables[4, 111:120]) /
  (sum(cm_c31_tables[4, 111:120]) + sum(cm_c31_tables[3, 111:120])), 6)
k17w3c31_truescore <- round((2 * k17w3c31_tpr * k17w3c31_tnr) /
  (k17w3c31_tpr + k17w3c31_tnr), 6)

k19w1c31_tpr <- round(sum(cm_c31_tables[1, 121:130]) /
  (sum(cm_c31_tables[1, 121:130]) + sum(cm_c31_tables[2, 121:130])), 6)
k19w1c31_tnr <- round(sum(cm_c31_tables[4, 121:130]) /
  (sum(cm_c31_tables[4, 121:130]) + sum(cm_c31_tables[3, 121:130])), 6)
k19w1c31_truescore <- round((2 * k19w1c31_tpr * k19w1c31_tnr) /
  (k19w1c31_tpr + k19w1c31_tnr), 6)

```

```

k19w2c31_tpr <- round(sum(cm_c31_tables[1, 131:140]) /
  (sum(cm_c31_tables[1, 131:140]) + sum(cm_c31_tables[2, 131:140])), 6)
k19w2c31_tnr <- round(sum(cm_c31_tables[4, 131:140]) /
  (sum(cm_c31_tables[4, 131:140]) + sum(cm_c31_tables[3, 131:140])), 6)
k19w2c31_truescore <- round((2 * k19w2c31_tpr * k19w2c31_tnr) /
  (k19w2c31_tpr + k19w2c31_tnr), 6)

k19w3c31_tpr <- round(sum(cm_c31_tables[1, 141:150]) /
  (sum(cm_c31_tables[1, 141:150]) + sum(cm_c31_tables[2, 141:150])), 6)
k19w3c31_tnr <- round(sum(cm_c31_tables[4, 141:150]) /
  (sum(cm_c31_tables[4, 141:150]) + sum(cm_c31_tables[3, 141:150])), 6)
k19w3c31_truescore <- round((2 * k19w3c31_tpr * k19w3c31_tnr) /
  (k19w3c31_tpr + k19w3c31_tnr), 6)

k21w1c31_tpr <- round(sum(cm_c31_tables[1, 151:160]) /
  (sum(cm_c31_tables[1, 151:160]) + sum(cm_c31_tables[2, 151:160])), 6)
k21w1c31_tnr <- round(sum(cm_c31_tables[4, 151:160]) /
  (sum(cm_c31_tables[4, 151:160]) + sum(cm_c31_tables[3, 151:160])), 6)
k21w1c31_truescore <- round((2 * k21w1c31_tpr * k21w1c31_tnr) /
  (k21w1c31_tpr + k21w1c31_tnr), 6)

k21w2c31_tpr <- round(sum(cm_c31_tables[1, 161:170]) /
  (sum(cm_c31_tables[1, 161:170]) + sum(cm_c31_tables[2, 161:170])), 6)
k21w2c31_tnr <- round(sum(cm_c31_tables[4, 161:170]) /
  (sum(cm_c31_tables[4, 161:170]) + sum(cm_c31_tables[3, 161:170])), 6)
k21w2c31_truescore <- round((2 * k21w2c31_tpr * k21w2c31_tnr) /
  (k21w2c31_tpr + k21w2c31_tnr), 6)

k21w3c31_tpr <- round(sum(cm_c31_tables[1, 171:180]) /
  (sum(cm_c31_tables[1, 171:180]) + sum(cm_c31_tables[2, 171:180])), 6)
k21w3c31_tnr <- round(sum(cm_c31_tables[4, 171:180]) /
  (sum(cm_c31_tables[4, 171:180]) + sum(cm_c31_tables[3, 171:180])), 6)
k21w3c31_truescore <- round((2 * k21w3c31_tpr * k21w3c31_tnr) /
  (k21w3c31_tpr + k21w3c31_tnr), 6)

k23w1c31_tpr <- round(sum(cm_c31_tables[1, 181:190]) /
  (sum(cm_c31_tables[1, 181:190]) + sum(cm_c31_tables[2, 181:190])), 6)
k23w1c31_tnr <- round(sum(cm_c31_tables[4, 181:190]) /
  (sum(cm_c31_tables[4, 181:190]) + sum(cm_c31_tables[3, 181:190])), 6)
k23w1c31_truescore <- round((2 * k23w1c31_tpr * k23w1c31_tnr) /
  (k23w1c31_tpr + k23w1c31_tnr), 6)

k23w2c31_tpr <- round(sum(cm_c31_tables[1, 191:200]) /
  (sum(cm_c31_tables[1, 191:200]) + sum(cm_c31_tables[2, 191:200])), 6)
k23w2c31_tnr <- round(sum(cm_c31_tables[4, 191:200]) /
  (sum(cm_c31_tables[4, 191:200]) + sum(cm_c31_tables[3, 191:200])), 6)
k23w2c31_truescore <- round((2 * k23w2c31_tpr * k23w2c31_tnr) /
  (k23w2c31_tpr + k23w2c31_tnr), 6)

k23w3c31_tpr <- round(sum(cm_c31_tables[1, 201:210]) /
  (sum(cm_c31_tables[1, 201:210]) + sum(cm_c31_tables[2, 201:210])), 6)

```

```

k23w3c31_tnr <- round(sum(cm_c31_tables[4, 201:210]) /
  (sum(cm_c31_tables[4, 201:210]) + sum(cm_c31_tables[3, 201:210])), 6)
k23w3c31_truescore <- round((2 * k23w3c31_tpr * k23w3c31_tnr) /
  (k23w3c31_tpr + k23w3c31_tnr), 6)

k25w1c31_tpr <- round(sum(cm_c31_tables[1, 211:220]) /
  (sum(cm_c31_tables[1, 211:220]) + sum(cm_c31_tables[2, 211:220])), 6)
k25w1c31_tnr <- round(sum(cm_c31_tables[4, 211:220]) /
  (sum(cm_c31_tables[4, 211:220]) + sum(cm_c31_tables[3, 211:220])), 6)
k25w1c31_truescore <- round((2 * k25w1c31_tpr * k25w1c31_tnr) /
  (k25w1c31_tpr + k25w1c31_tnr), 6)

k25w2c31_tpr <- round(sum(cm_c31_tables[1, 221:230]) /
  (sum(cm_c31_tables[1, 221:230]) + sum(cm_c31_tables[2, 221:230])), 6)
k25w2c31_tnr <- round(sum(cm_c31_tables[4, 221:230]) /
  (sum(cm_c31_tables[4, 221:230]) + sum(cm_c31_tables[3, 221:230])), 6)
k25w2c31_truescore <- round((2 * k25w2c31_tpr * k25w2c31_tnr) /
  (k25w2c31_tpr + k25w2c31_tnr), 6)

k25w3c31_tpr <- round(sum(cm_c31_tables[1, 231:240]) /
  (sum(cm_c31_tables[1, 231:240]) + sum(cm_c31_tables[2, 231:240])), 6)
k25w3c31_tnr <- round(sum(cm_c31_tables[4, 231:240]) /
  (sum(cm_c31_tables[4, 231:240]) + sum(cm_c31_tables[3, 231:240])), 6)
k25w3c31_truescore <- round((2 * k25w3c31_tpr * k25w3c31_tnr) /
  (k25w3c31_tpr + k25w3c31_tnr), 6)

k27w1c31_tpr <- round(sum(cm_c31_tables[1, 241:250]) /
  (sum(cm_c31_tables[1, 241:250]) + sum(cm_c31_tables[2, 241:250])), 6)
k27w1c31_tnr <- round(sum(cm_c31_tables[4, 241:250]) /
  (sum(cm_c31_tables[4, 241:250]) + sum(cm_c31_tables[3, 241:250])), 6)
k27w1c31_truescore <- round((2 * k27w1c31_tpr * k27w1c31_tnr) /
  (k27w1c31_tpr + k27w1c31_tnr), 6)

k27w2c31_tpr <- round(sum(cm_c31_tables[1, 251:260]) /
  (sum(cm_c31_tables[1, 251:260]) + sum(cm_c31_tables[2, 251:260])), 6)
k27w2c31_tnr <- round(sum(cm_c31_tables[4, 251:260]) /
  (sum(cm_c31_tables[4, 251:260]) + sum(cm_c31_tables[3, 251:260])), 6)
k27w2c31_truescore <- round((2 * k27w2c31_tpr * k27w2c31_tnr) /
  (k27w2c31_tpr + k27w2c31_tnr), 6)

k27w3c31_tpr <- round(sum(cm_c31_tables[1, 261:270]) /
  (sum(cm_c31_tables[1, 261:270]) + sum(cm_c31_tables[2, 261:270])), 6)
k27w3c31_tnr <- round(sum(cm_c31_tables[4, 261:270]) /
  (sum(cm_c31_tables[4, 261:270]) + sum(cm_c31_tables[3, 261:270])), 6)
k27w3c31_truescore <- round((2 * k27w3c31_tpr * k27w3c31_tnr) /
  (k27w3c31_tpr + k27w3c31_tnr), 6)

k29w1c31_tpr <- round(sum(cm_c31_tables[1, 271:280]) /
  (sum(cm_c31_tables[1, 271:280]) + sum(cm_c31_tables[2, 271:280])), 6)
k29w1c31_tnr <- round(sum(cm_c31_tables[4, 271:280]) /

```

```

(sum(cm_c31_tables[4, 271:280]) + sum(cm_c31_tables[3, 271:280])), 6)
k29w1c31_truescore <- round((2 * k29w1c31_tpr * k29w1c31_tnr) /
  (k29w1c31_tpr + k29w1c31_tnr), 6)

k29w2c31_tpr <- round(sum(cm_c31_tables[1, 281:290]) /
  (sum(cm_c31_tables[1, 281:290]) + sum(cm_c31_tables[2, 281:290])), 6)
k29w2c31_tnr <- round(sum(cm_c31_tables[4, 281:290]) /
  (sum(cm_c31_tables[4, 281:290]) + sum(cm_c31_tables[3, 281:290])), 6)
k29w2c31_truescore <- round((2 * k29w2c31_tpr * k29w2c31_tnr) /
  (k29w2c31_tpr + k29w2c31_tnr), 6)

k29w3c31_tpr <- round(sum(cm_c31_tables[1, 291:300]) /
  (sum(cm_c31_tables[1, 291:300]) + sum(cm_c31_tables[2, 291:300])), 6)
k29w3c31_tnr <- round(sum(cm_c31_tables[4, 291:300]) /
  (sum(cm_c31_tables[4, 291:300]) + sum(cm_c31_tables[3, 291:300])), 6)
k29w3c31_truescore <- round((2 * k29w3c31_tpr * k29w3c31_tnr) /
  (k29w3c31_tpr + k29w3c31_tnr), 6)

k31w1c31_tpr <- round(sum(cm_c31_tables[1, 301:310]) /
  (sum(cm_c31_tables[1, 301:310]) + sum(cm_c31_tables[2, 301:310])), 6)
k31w1c31_tnr <- round(sum(cm_c31_tables[4, 301:310]) /
  (sum(cm_c31_tables[4, 301:310]) + sum(cm_c31_tables[3, 301:310])), 6)
k31w1c31_truescore <- round((2 * k31w1c31_tpr * k31w1c31_tnr) /
  (k31w1c31_tpr + k31w1c31_tnr), 6)

k31w2c31_tpr <- round(sum(cm_c31_tables[1, 311:320]) /
  (sum(cm_c31_tables[1, 311:320]) + sum(cm_c31_tables[2, 311:320])), 6)
k31w2c31_tnr <- round(sum(cm_c31_tables[4, 311:320]) /
  (sum(cm_c31_tables[4, 311:320]) + sum(cm_c31_tables[3, 311:320])), 6)
k31w2c31_truescore <- round((2 * k31w2c31_tpr * k31w2c31_tnr) /
  (k31w2c31_tpr + k31w2c31_tnr), 6)

k31w3c31_tpr <- round(sum(cm_c31_tables[1, 321:330]) /
  (sum(cm_c31_tables[1, 321:330]) + sum(cm_c31_tables[2, 321:330])), 6)
k31w3c31_tnr <- round(sum(cm_c31_tables[4, 321:330]) /
  (sum(cm_c31_tables[4, 321:330]) + sum(cm_c31_tables[3, 321:330])), 6)
k31w3c31_truescore <- round((2 * k31w3c31_tpr * k31w3c31_tnr) /
  (k31w3c31_tpr + k31w3c31_tnr), 6)

k33w1c31_tpr <- round(sum(cm_c31_tables[1, 331:340]) /
  (sum(cm_c31_tables[1, 331:340]) + sum(cm_c31_tables[2, 331:340])), 6)
k33w1c31_tnr <- round(sum(cm_c31_tables[4, 331:340]) /
  (sum(cm_c31_tables[4, 331:340]) + sum(cm_c31_tables[3, 331:340])), 6)
k33w1c31_truescore <- round((2 * k33w1c31_tpr * k33w1c31_tnr) /
  (k33w1c31_tpr + k33w1c31_tnr), 6)

k33w2c31_tpr <- round(sum(cm_c31_tables[1, 341:350]) /
  (sum(cm_c31_tables[1, 341:350]) + sum(cm_c31_tables[2, 341:350])), 6)
k33w2c31_tnr <- round(sum(cm_c31_tables[4, 341:350]) /
  (sum(cm_c31_tables[4, 341:350]) + sum(cm_c31_tables[3, 341:350])), 6)
k33w2c31_truescore <- round((2 * k33w2c31_tpr * k33w2c31_tnr) /

```

```

(k33w2c31_tpr + k33w2c31_tnr), 6)

k33w3c31_tpr <- round(sum(cm_c31_tables[1, 351:360]) /
  (sum(cm_c31_tables[1, 351:360]) + sum(cm_c31_tables[2, 351:360])), 6)
k33w3c31_tnr <- round(sum(cm_c31_tables[4, 351:360]) /
  (sum(cm_c31_tables[4, 351:360]) + sum(cm_c31_tables[3, 351:360])), 6)
k33w3c31_truescore <- round((2 * k33w3c31_tpr * k33w3c31_tnr) /
  (k33w3c31_tpr + k33w3c31_tnr), 6)

k35w1c31_tpr <- round(sum(cm_c31_tables[1, 361:370]) /
  (sum(cm_c31_tables[1, 361:370]) + sum(cm_c31_tables[2, 361:370])), 6)
k35w1c31_tnr <- round(sum(cm_c31_tables[4, 361:370]) /
  (sum(cm_c31_tables[4, 361:370]) + sum(cm_c31_tables[3, 361:370])), 6)
k35w1c31_truescore <- round((2 * k35w1c31_tpr * k35w1c31_tnr) /
  (k35w1c31_tpr + k35w1c31_tnr), 6)

k35w2c31_tpr <- round(sum(cm_c31_tables[1, 371:380]) /
  (sum(cm_c31_tables[1, 371:380]) + sum(cm_c31_tables[2, 371:380])), 6)
k35w2c31_tnr <- round(sum(cm_c31_tables[4, 371:380]) /
  (sum(cm_c31_tables[4, 371:380]) + sum(cm_c31_tables[3, 371:380])), 6)
k35w2c31_truescore <- round((2 * k35w2c31_tpr * k35w2c31_tnr) /
  (k35w2c31_tpr + k35w2c31_tnr), 6)

k35w3c31_tpr <- round(sum(cm_c31_tables[1, 381:390]) /
  (sum(cm_c31_tables[1, 381:390]) + sum(cm_c31_tables[2, 381:390])), 6)
k35w3c31_tnr <- round(sum(cm_c31_tables[4, 381:390]) /
  (sum(cm_c31_tables[4, 381:390]) + sum(cm_c31_tables[3, 381:390])), 6)
k35w3c31_truescore <- round((2 * k35w3c31_tpr * k35w3c31_tnr) /
  (k35w3c31_tpr + k35w3c31_tnr), 6)

k37w1c31_tpr <- round(sum(cm_c31_tables[1, 391:400]) /
  (sum(cm_c31_tables[1, 391:400]) + sum(cm_c31_tables[2, 391:400])), 6)
k37w1c31_tnr <- round(sum(cm_c31_tables[4, 391:400]) /
  (sum(cm_c31_tables[4, 391:400]) + sum(cm_c31_tables[3, 391:400])), 6)
k37w1c31_truescore <- round((2 * k37w1c31_tpr * k37w1c31_tnr) /
  (k37w1c31_tpr + k37w1c31_tnr), 6)

k37w2c31_tpr <- round(sum(cm_c31_tables[1, 401:410]) /
  (sum(cm_c31_tables[1, 401:410]) + sum(cm_c31_tables[2, 401:410])), 6)
k37w2c31_tnr <- round(sum(cm_c31_tables[4, 401:410]) /
  (sum(cm_c31_tables[4, 401:410]) + sum(cm_c31_tables[3, 401:410])), 6)
k37w2c31_truescore <- round((2 * k37w2c31_tpr * k37w2c31_tnr) /
  (k37w2c31_tpr + k37w2c31_tnr), 6)

k37w3c31_tpr <- round(sum(cm_c31_tables[1, 411:420]) /
  (sum(cm_c31_tables[1, 411:420]) + sum(cm_c31_tables[2, 411:420])), 6)
k37w3c31_tnr <- round(sum(cm_c31_tables[4, 411:420]) /
  (sum(cm_c31_tables[4, 411:420]) + sum(cm_c31_tables[3, 411:420])), 6)
k37w3c31_truescore <- round((2 * k37w3c31_tpr * k37w3c31_tnr) /
  (k37w3c31_tpr + k37w3c31_tnr), 6)

```

```

k39w1c31_tpr <- round(sum(cm_c31_tables[1, 421:430]) /
  (sum(cm_c31_tables[1, 421:430]) + sum(cm_c31_tables[2, 421:430])), 6)
k39w1c31_tnr <- round(sum(cm_c31_tables[4, 421:430]) /
  (sum(cm_c31_tables[4, 421:430]) + sum(cm_c31_tables[3, 421:430])), 6)
k39w1c31_truescore <- round((2 * k39w1c31_tpr * k39w1c31_tnr) /
  (k39w1c31_tpr + k39w1c31_tnr), 6)

k39w2c31_tpr <- round(sum(cm_c31_tables[1, 431:440]) /
  (sum(cm_c31_tables[1, 431:440]) + sum(cm_c31_tables[2, 431:440])), 6)
k39w2c31_tnr <- round(sum(cm_c31_tables[4, 431:440]) /
  (sum(cm_c31_tables[4, 431:440]) + sum(cm_c31_tables[3, 431:440])), 6)
k39w2c31_truescore <- round((2 * k39w2c31_tpr * k39w2c31_tnr) /
  (k39w2c31_tpr + k39w2c31_tnr), 6)

k39w3c31_tpr <- round(sum(cm_c31_tables[1, 441:450]) /
  (sum(cm_c31_tables[1, 441:450]) + sum(cm_c31_tables[2, 441:450])), 6)
k39w3c31_tnr <- round(sum(cm_c31_tables[4, 441:450]) /
  (sum(cm_c31_tables[4, 441:450]) + sum(cm_c31_tables[3, 441:450])), 6)
k39w3c31_truescore <- round((2 * k39w3c31_tpr * k39w3c31_tnr) /
  (k39w3c31_tpr + k39w3c31_tnr), 6)

# Compile the 0.31 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c31_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.31,
  TPR = c(k11w1c31_tpr, k11w2c31_tpr, k11w3c31_tpr, k13w1c31_tpr,
    k13w2c31_tpr, k13w3c31_tpr, k15w1c31_tpr, k15w2c31_tpr,
    k15w3c31_tpr, k17w1c31_tpr, k17w2c31_tpr, k17w3c31_tpr,
    k19w1c31_tpr, k19w2c31_tpr, k19w3c31_tpr, k21w1c31_tpr,
    k21w2c31_tpr, k21w3c31_tpr, k23w1c31_tpr, k23w2c31_tpr,
    k23w3c31_tpr, k25w1c31_tpr, k25w2c31_tpr, k25w3c31_tpr,
    k27w1c31_tpr, k27w2c31_tpr, k27w3c31_tpr, k29w1c31_tpr,
    k29w2c31_tpr, k29w3c31_tpr, k31w1c31_tpr, k31w2c31_tpr,
    k31w3c31_tpr, k33w1c31_tpr, k33w2c31_tpr, k33w3c31_tpr,
    k35w1c31_tpr, k35w2c31_tpr, k35w3c31_tpr, k37w1c31_tpr,
    k37w2c31_tpr, k37w3c31_tpr, k39w1c31_tpr, k39w2c31_tpr,
    k39w3c31_tpr),
  TNR = c(k11w1c31_tnr, k11w2c31_tnr, k11w3c31_tnr, k13w1c31_tnr,
    k13w2c31_tnr, k13w3c31_tnr, k15w1c31_tnr, k15w2c31_tnr,
    k15w3c31_tnr, k17w1c31_tnr, k17w2c31_tnr, k17w3c31_tnr,
    k19w1c31_tnr, k19w2c31_tnr, k19w3c31_tnr, k21w1c31_tnr,
    k21w2c31_tnr, k21w3c31_tnr, k23w1c31_tnr, k23w2c31_tnr,
    k23w3c31_tnr, k25w1c31_tnr, k25w2c31_tnr, k25w3c31_tnr,
    k27w1c31_tnr, k27w2c31_tnr, k27w3c31_tnr, k29w1c31_tnr,
    k29w2c31_tnr, k29w3c31_tnr, k31w1c31_tnr, k31w2c31_tnr,
    k31w3c31_tnr, k33w1c31_tnr, k33w2c31_tnr, k33w3c31_tnr),
  Truescore = k39w1c31_truescore)

```

```

k35w1c31_tnr, k35w2c31_tnr, k35w3c31_tnr, k37w1c31_tnr,
k37w2c31_tnr, k37w3c31_tnr, k39w1c31_tnr, k39w2c31_tnr,
k39w3c31_tnr),
Truescore = c(k11w1c31_truescore, k11w2c31_truescore,
k11w3c31_truescore, k13w1c31_truescore,
k13w2c31_truescore, k13w3c31_truescore,
k15w1c31_truescore, k15w2c31_truescore,
k15w3c31_truescore, k17w1c31_truescore,
k17w2c31_truescore, k17w3c31_truescore,
k19w1c31_truescore, k19w2c31_truescore,
k19w3c31_truescore, k21w1c31_truescore,
k21w2c31_truescore, k21w3c31_truescore,
k23w1c31_truescore, k23w2c31_truescore,
k23w3c31_truescore, k25w1c31_truescore,
k25w2c31_truescore, k25w3c31_truescore,
k27w1c31_truescore, k27w2c31_truescore,
k27w3c31_truescore, k29w1c31_truescore,
k29w2c31_truescore, k29w3c31_truescore,
k31w1c31_truescore, k31w2c31_truescore,
k31w3c31_truescore, k33w1c31_truescore,
k33w2c31_truescore, k33w3c31_truescore,
k35w1c31_truescore, k35w2c31_truescore,
k35w3c31_truescore, k37w1c31_truescore,
k37w2c31_truescore, k37w3c31_truescore,
k39w1c31_truescore, k39w2c31_truescore,
k39w3c31_truescore))

```

`knitr::kable(c31_results[1:45,], caption = "c31_results")`

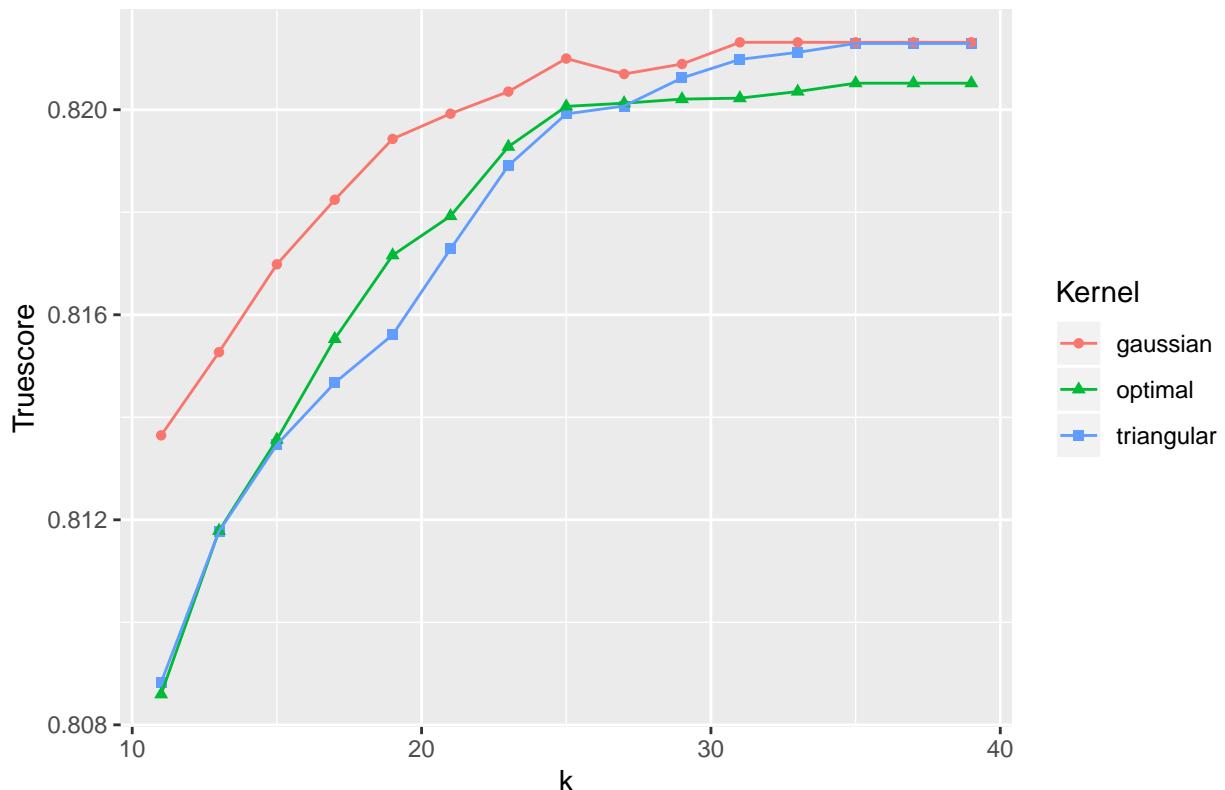
Table 40: c31_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.31	0.802460	0.815297	0.808828
11	gaussian	0.31	0.814709	0.812589	0.813648
11	optimal	0.31	0.803062	0.814207	0.808596
13	triangular	0.31	0.808886	0.814686	0.811776
13	gaussian	0.31	0.818775	0.811796	0.815271
13	optimal	0.31	0.809036	0.814554	0.811786
15	triangular	0.31	0.813303	0.813629	0.813466
15	gaussian	0.31	0.823343	0.810722	0.816984
15	optimal	0.31	0.813203	0.813927	0.813565
17	triangular	0.31	0.816416	0.812936	0.814672
17	gaussian	0.31	0.826456	0.810194	0.818244
17	optimal	0.31	0.817771	0.813299	0.815529
19	triangular	0.31	0.819629	0.811631	0.815610
19	gaussian	0.31	0.829367	0.809731	0.819431
19	optimal	0.31	0.822139	0.812242	0.817161
21	triangular	0.31	0.823544	0.811119	0.817284
21	gaussian	0.31	0.830723	0.809401	0.819923
21	optimal	0.31	0.824950	0.811020	0.817926
23	triangular	0.31	0.827108	0.810871	0.818909
23	gaussian	0.31	0.831225	0.809764	0.820354
23	optimal	0.31	0.827912	0.810822	0.819278

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.31	0.829568	0.810491	0.819919
25	gaussian	0.31	0.833283	0.809071	0.820999
25	optimal	0.31	0.829869	0.810491	0.820066
27	triangular	0.31	0.830472	0.809930	0.820072
27	gaussian	0.31	0.833082	0.808674	0.820697
27	optimal	0.31	0.830271	0.810227	0.820127
29	triangular	0.31	0.832329	0.809236	0.820620
29	gaussian	0.31	0.833484	0.808674	0.820892
29	optimal	0.31	0.831325	0.809385	0.820208
31	triangular	0.31	0.833635	0.808707	0.820982
31	gaussian	0.31	0.835291	0.807799	0.821315
31	optimal	0.31	0.832078	0.808707	0.820226
33	triangular	0.31	0.834237	0.808410	0.821120
33	gaussian	0.31	0.835291	0.807799	0.821315
33	optimal	0.31	0.832781	0.808295	0.820355
35	triangular	0.31	0.834839	0.808179	0.821293
35	gaussian	0.31	0.835291	0.807799	0.821315
35	optimal	0.31	0.833434	0.807997	0.820518
37	triangular	0.31	0.834839	0.808179	0.821293
37	gaussian	0.31	0.835291	0.807799	0.821315
37	optimal	0.31	0.833434	0.807997	0.820518
39	triangular	0.31	0.834839	0.808179	0.821293
39	gaussian	0.31	0.835291	0.807799	0.821315
39	optimal	0.31	0.833434	0.807997	0.820518

```
ggplot(c31_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.31 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.31 (Truescore)

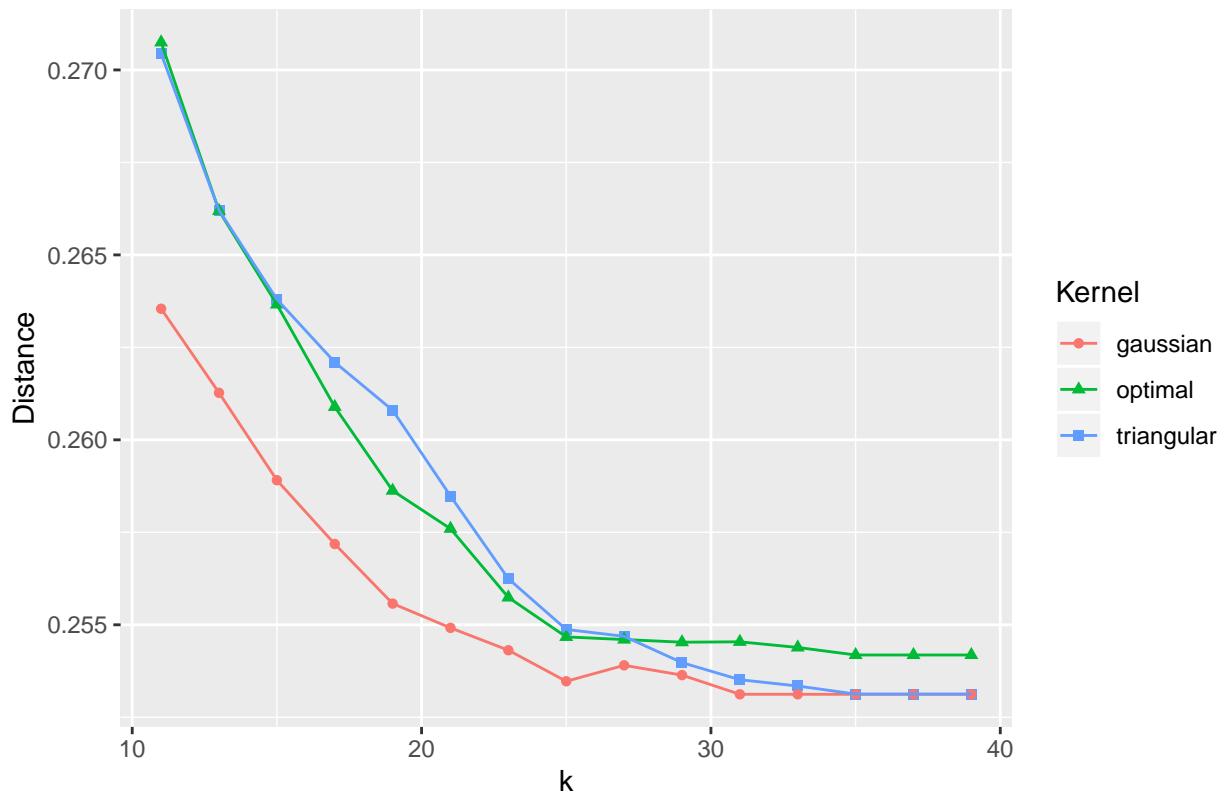


```
# For the Cutoff of 0.31, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c31_results <- c31_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c31_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.31 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.31 (Distance)



```
# For the Cutoff of 0.31, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c31_results$Truescore)
```

```
## [1] 0.821315
```

```
(c31_opt_k_ts <- c31_results$k[which.max(c31_results$Truescore)])
```

```
## [1] 31
```

```
(c31_opt_kernel_ts <- c31_results$Kernel[which.max(c31_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c31_opt_cut_ts <- c31_results$Cut[which.max(c31_results$Truescore)])
```

```
## [1] 0.31
```

```
(c31_opt_tpr_ts <- c31_results$TPR[which.max(c31_results$Truescore)])
```

```
## [1] 0.835291
```

```

(c31_opt_tnr_ts <- c31_results$TNR[which.max(c31_results$Truescore)])

## [1] 0.807799

(c31_opt_d_ts <- c31_results$Distance[which.max(c31_results$Truescore)])

## [1] 0.2531211

min(c31_results$Distance)

## [1] 0.2531211

(c31_opt_k_dist <- c31_results$k[which.min(c31_results$Distance)])

## [1] 31

(c31_opt_kernel_dist <- c31_results$Kernel[which.min(c31_results$Distance)])

## [1] "gaussian"

(c31_opt_cut_dist <- c31_results$Cut[which.min(c31_results$Distance)])

## [1] 0.31

(c31_opt_tpr_dist <- c31_results$TPR[which.min(c31_results$Distance)])

## [1] 0.835291

(c31_opt_tnr_dist <- c31_results$TNR[which.min(c31_results$Distance)])

## [1] 0.807799

(c31_opt_t_dist <- c31_results$Truescore[which.min(c31_results$Distance)])

## [1] 0.821315

#####
# 0.32 Cutoff
#####

# For the decision cutoff of 0.32, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c32 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred32, obs))
  confusionMatrix(ss$pred32, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c32) <- kwf_dfs_v

cm_c32_tables <- sapply(cm_c32, "[[", 2)
cm_c32_tables <- as_tibble(cm_c32_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.32.

k11w1c32_tpr <- round(sum(cm_c32_tables[1, 1:10]) /
  (sum(cm_c32_tables[1, 1:10]) + sum(cm_c32_tables[2, 1:10])), 6)
k11w1c32_tnr <- round(sum(cm_c32_tables[4, 1:10]) /
  (sum(cm_c32_tables[4, 1:10]) + sum(cm_c32_tables[3, 1:10])), 6)
k11w1c32_truescore <- round((2 * k11w1c32_tpr * k11w1c32_tnr) /
  (k11w1c32_tpr + k11w1c32_tnr), 6)

k11w2c32_tpr <- round(sum(cm_c32_tables[1, 11:20]) /
  (sum(cm_c32_tables[1, 11:20]) + sum(cm_c32_tables[2, 11:20])), 6)
k11w2c32_tnr <- round(sum(cm_c32_tables[4, 11:20]) /
  (sum(cm_c32_tables[4, 11:20]) + sum(cm_c32_tables[3, 11:20])), 6)
k11w2c32_truescore <- round((2 * k11w2c32_tpr * k11w2c32_tnr) /
  (k11w2c32_tpr + k11w2c32_tnr), 6)

k11w3c32_tpr <- round(sum(cm_c32_tables[1, 21:30]) /
  (sum(cm_c32_tables[1, 21:30]) + sum(cm_c32_tables[2, 21:30])), 6)
k11w3c32_tnr <- round(sum(cm_c32_tables[4, 21:30]) /
  (sum(cm_c32_tables[4, 21:30]) + sum(cm_c32_tables[3, 21:30])), 6)
k11w3c32_truescore <- round((2 * k11w3c32_tpr * k11w3c32_tnr) /
  (k11w3c32_tpr + k11w3c32_tnr), 6)

k13w1c32_tpr <- round(sum(cm_c32_tables[1, 31:40]) /
  (sum(cm_c32_tables[1, 31:40]) + sum(cm_c32_tables[2, 31:40])), 6)
k13w1c32_tnr <- round(sum(cm_c32_tables[4, 31:40]) /
  (sum(cm_c32_tables[4, 31:40]) + sum(cm_c32_tables[3, 31:40])), 6)
k13w1c32_truescore <- round((2 * k13w1c32_tpr * k13w1c32_tnr) /
  (k13w1c32_tpr + k13w1c32_tnr), 6)

k13w2c32_tpr <- round(sum(cm_c32_tables[1, 41:50]) /
  (sum(cm_c32_tables[1, 41:50]) + sum(cm_c32_tables[2, 41:50])), 6)
k13w2c32_tnr <- round(sum(cm_c32_tables[4, 41:50]) /
  (sum(cm_c32_tables[4, 41:50]) + sum(cm_c32_tables[3, 41:50])), 6)
k13w2c32_truescore <- round((2 * k13w2c32_tpr * k13w2c32_tnr) /
  (k13w2c32_tpr + k13w2c32_tnr), 6)

k13w3c32_tpr <- round(sum(cm_c32_tables[1, 51:60]) /
  (sum(cm_c32_tables[1, 51:60]) + sum(cm_c32_tables[2, 51:60])), 6)
k13w3c32_tnr <- round(sum(cm_c32_tables[4, 51:60]) /
  (sum(cm_c32_tables[4, 51:60]) + sum(cm_c32_tables[3, 51:60])), 6)
k13w3c32_truescore <- round((2 * k13w3c32_tpr * k13w3c32_tnr) /

```

```

(k13w3c32_tpr + k13w3c32_tnr), 6)

k15w1c32_tpr <- round(sum(cm_c32_tables[1, 61:70]) /
  (sum(cm_c32_tables[1, 61:70]) + sum(cm_c32_tables[2, 61:70])), 6)
k15w1c32_tnr <- round(sum(cm_c32_tables[4, 61:70]) /
  (sum(cm_c32_tables[4, 61:70]) + sum(cm_c32_tables[3, 61:70])), 6)
k15w1c32_truescore <- round((2 * k15w1c32_tpr * k15w1c32_tnr) /
  (k15w1c32_tpr + k15w1c32_tnr), 6)

k15w2c32_tpr <- round(sum(cm_c32_tables[1, 71:80]) /
  (sum(cm_c32_tables[1, 71:80]) + sum(cm_c32_tables[2, 71:80])), 6)
k15w2c32_tnr <- round(sum(cm_c32_tables[4, 71:80]) /
  (sum(cm_c32_tables[4, 71:80]) + sum(cm_c32_tables[3, 71:80])), 6)
k15w2c32_truescore <- round((2 * k15w2c32_tpr * k15w2c32_tnr) /
  (k15w2c32_tpr + k15w2c32_tnr), 6)

k15w3c32_tpr <- round(sum(cm_c32_tables[1, 81:90]) /
  (sum(cm_c32_tables[1, 81:90]) + sum(cm_c32_tables[2, 81:90])), 6)
k15w3c32_tnr <- round(sum(cm_c32_tables[4, 81:90]) /
  (sum(cm_c32_tables[4, 81:90]) + sum(cm_c32_tables[3, 81:90])), 6)
k15w3c32_truescore <- round((2 * k15w3c32_tpr * k15w3c32_tnr) /
  (k15w3c32_tpr + k15w3c32_tnr), 6)

k17w1c32_tpr <- round(sum(cm_c32_tables[1, 91:100]) /
  (sum(cm_c32_tables[1, 91:100]) + sum(cm_c32_tables[2, 91:100])), 6)
k17w1c32_tnr <- round(sum(cm_c32_tables[4, 91:100]) /
  (sum(cm_c32_tables[4, 91:100]) + sum(cm_c32_tables[3, 91:100])), 6)
k17w1c32_truescore <- round((2 * k17w1c32_tpr * k17w1c32_tnr) /
  (k17w1c32_tpr + k17w1c32_tnr), 6)

k17w2c32_tpr <- round(sum(cm_c32_tables[1, 101:110]) /
  (sum(cm_c32_tables[1, 101:110]) + sum(cm_c32_tables[2, 101:110])), 6)
k17w2c32_tnr <- round(sum(cm_c32_tables[4, 101:110]) /
  (sum(cm_c32_tables[4, 101:110]) + sum(cm_c32_tables[3, 101:110])), 6)
k17w2c32_truescore <- round((2 * k17w2c32_tpr * k17w2c32_tnr) /
  (k17w2c32_tpr + k17w2c32_tnr), 6)

k17w3c32_tpr <- round(sum(cm_c32_tables[1, 111:120]) /
  (sum(cm_c32_tables[1, 111:120]) + sum(cm_c32_tables[2, 111:120])), 6)
k17w3c32_tnr <- round(sum(cm_c32_tables[4, 111:120]) /
  (sum(cm_c32_tables[4, 111:120]) + sum(cm_c32_tables[3, 111:120])), 6)
k17w3c32_truescore <- round((2 * k17w3c32_tpr * k17w3c32_tnr) /
  (k17w3c32_tpr + k17w3c32_tnr), 6)

k19w1c32_tpr <- round(sum(cm_c32_tables[1, 121:130]) /
  (sum(cm_c32_tables[1, 121:130]) + sum(cm_c32_tables[2, 121:130])), 6)
k19w1c32_tnr <- round(sum(cm_c32_tables[4, 121:130]) /
  (sum(cm_c32_tables[4, 121:130]) + sum(cm_c32_tables[3, 121:130])), 6)
k19w1c32_truescore <- round((2 * k19w1c32_tpr * k19w1c32_tnr) /
  (k19w1c32_tpr + k19w1c32_tnr), 6)

```

```

k19w2c32_tpr <- round(sum(cm_c32_tables[1, 131:140]) /
  (sum(cm_c32_tables[1, 131:140]) + sum(cm_c32_tables[2, 131:140])), 6)
k19w2c32_tnr <- round(sum(cm_c32_tables[4, 131:140]) /
  (sum(cm_c32_tables[4, 131:140]) + sum(cm_c32_tables[3, 131:140])), 6)
k19w2c32_truescore <- round((2 * k19w2c32_tpr * k19w2c32_tnr) /
  (k19w2c32_tpr + k19w2c32_tnr), 6)

k19w3c32_tpr <- round(sum(cm_c32_tables[1, 141:150]) /
  (sum(cm_c32_tables[1, 141:150]) + sum(cm_c32_tables[2, 141:150])), 6)
k19w3c32_tnr <- round(sum(cm_c32_tables[4, 141:150]) /
  (sum(cm_c32_tables[4, 141:150]) + sum(cm_c32_tables[3, 141:150])), 6)
k19w3c32_truescore <- round((2 * k19w3c32_tpr * k19w3c32_tnr) /
  (k19w3c32_tpr + k19w3c32_tnr), 6)

k21w1c32_tpr <- round(sum(cm_c32_tables[1, 151:160]) /
  (sum(cm_c32_tables[1, 151:160]) + sum(cm_c32_tables[2, 151:160])), 6)
k21w1c32_tnr <- round(sum(cm_c32_tables[4, 151:160]) /
  (sum(cm_c32_tables[4, 151:160]) + sum(cm_c32_tables[3, 151:160])), 6)
k21w1c32_truescore <- round((2 * k21w1c32_tpr * k21w1c32_tnr) /
  (k21w1c32_tpr + k21w1c32_tnr), 6)

k21w2c32_tpr <- round(sum(cm_c32_tables[1, 161:170]) /
  (sum(cm_c32_tables[1, 161:170]) + sum(cm_c32_tables[2, 161:170])), 6)
k21w2c32_tnr <- round(sum(cm_c32_tables[4, 161:170]) /
  (sum(cm_c32_tables[4, 161:170]) + sum(cm_c32_tables[3, 161:170])), 6)
k21w2c32_truescore <- round((2 * k21w2c32_tpr * k21w2c32_tnr) /
  (k21w2c32_tpr + k21w2c32_tnr), 6)

k21w3c32_tpr <- round(sum(cm_c32_tables[1, 171:180]) /
  (sum(cm_c32_tables[1, 171:180]) + sum(cm_c32_tables[2, 171:180])), 6)
k21w3c32_tnr <- round(sum(cm_c32_tables[4, 171:180]) /
  (sum(cm_c32_tables[4, 171:180]) + sum(cm_c32_tables[3, 171:180])), 6)
k21w3c32_truescore <- round((2 * k21w3c32_tpr * k21w3c32_tnr) /
  (k21w3c32_tpr + k21w3c32_tnr), 6)

k23w1c32_tpr <- round(sum(cm_c32_tables[1, 181:190]) /
  (sum(cm_c32_tables[1, 181:190]) + sum(cm_c32_tables[2, 181:190])), 6)
k23w1c32_tnr <- round(sum(cm_c32_tables[4, 181:190]) /
  (sum(cm_c32_tables[4, 181:190]) + sum(cm_c32_tables[3, 181:190])), 6)
k23w1c32_truescore <- round((2 * k23w1c32_tpr * k23w1c32_tnr) /
  (k23w1c32_tpr + k23w1c32_tnr), 6)

k23w2c32_tpr <- round(sum(cm_c32_tables[1, 191:200]) /
  (sum(cm_c32_tables[1, 191:200]) + sum(cm_c32_tables[2, 191:200])), 6)
k23w2c32_tnr <- round(sum(cm_c32_tables[4, 191:200]) /
  (sum(cm_c32_tables[4, 191:200]) + sum(cm_c32_tables[3, 191:200])), 6)
k23w2c32_truescore <- round((2 * k23w2c32_tpr * k23w2c32_tnr) /
  (k23w2c32_tpr + k23w2c32_tnr), 6)

k23w3c32_tpr <- round(sum(cm_c32_tables[1, 201:210]) /
  (sum(cm_c32_tables[1, 201:210]) + sum(cm_c32_tables[2, 201:210])), 6)

```

```

k23w3c32_tnr <- round(sum(cm_c32_tables[4, 201:210]) /
  (sum(cm_c32_tables[4, 201:210]) + sum(cm_c32_tables[3, 201:210])), 6)
k23w3c32_truescore <- round((2 * k23w3c32_tpr * k23w3c32_tnr) /
  (k23w3c32_tpr + k23w3c32_tnr), 6)

k25w1c32_tpr <- round(sum(cm_c32_tables[1, 211:220]) /
  (sum(cm_c32_tables[1, 211:220]) + sum(cm_c32_tables[2, 211:220])), 6)
k25w1c32_tnr <- round(sum(cm_c32_tables[4, 211:220]) /
  (sum(cm_c32_tables[4, 211:220]) + sum(cm_c32_tables[3, 211:220])), 6)
k25w1c32_truescore <- round((2 * k25w1c32_tpr * k25w1c32_tnr) /
  (k25w1c32_tpr + k25w1c32_tnr), 6)

k25w2c32_tpr <- round(sum(cm_c32_tables[1, 221:230]) /
  (sum(cm_c32_tables[1, 221:230]) + sum(cm_c32_tables[2, 221:230])), 6)
k25w2c32_tnr <- round(sum(cm_c32_tables[4, 221:230]) /
  (sum(cm_c32_tables[4, 221:230]) + sum(cm_c32_tables[3, 221:230])), 6)
k25w2c32_truescore <- round((2 * k25w2c32_tpr * k25w2c32_tnr) /
  (k25w2c32_tpr + k25w2c32_tnr), 6)

k25w3c32_tpr <- round(sum(cm_c32_tables[1, 231:240]) /
  (sum(cm_c32_tables[1, 231:240]) + sum(cm_c32_tables[2, 231:240])), 6)
k25w3c32_tnr <- round(sum(cm_c32_tables[4, 231:240]) /
  (sum(cm_c32_tables[4, 231:240]) + sum(cm_c32_tables[3, 231:240])), 6)
k25w3c32_truescore <- round((2 * k25w3c32_tpr * k25w3c32_tnr) /
  (k25w3c32_tpr + k25w3c32_tnr), 6)

k27w1c32_tpr <- round(sum(cm_c32_tables[1, 241:250]) /
  (sum(cm_c32_tables[1, 241:250]) + sum(cm_c32_tables[2, 241:250])), 6)
k27w1c32_tnr <- round(sum(cm_c32_tables[4, 241:250]) /
  (sum(cm_c32_tables[4, 241:250]) + sum(cm_c32_tables[3, 241:250])), 6)
k27w1c32_truescore <- round((2 * k27w1c32_tpr * k27w1c32_tnr) /
  (k27w1c32_tpr + k27w1c32_tnr), 6)

k27w2c32_tpr <- round(sum(cm_c32_tables[1, 251:260]) /
  (sum(cm_c32_tables[1, 251:260]) + sum(cm_c32_tables[2, 251:260])), 6)
k27w2c32_tnr <- round(sum(cm_c32_tables[4, 251:260]) /
  (sum(cm_c32_tables[4, 251:260]) + sum(cm_c32_tables[3, 251:260])), 6)
k27w2c32_truescore <- round((2 * k27w2c32_tpr * k27w2c32_tnr) /
  (k27w2c32_tpr + k27w2c32_tnr), 6)

k27w3c32_tpr <- round(sum(cm_c32_tables[1, 261:270]) /
  (sum(cm_c32_tables[1, 261:270]) + sum(cm_c32_tables[2, 261:270])), 6)
k27w3c32_tnr <- round(sum(cm_c32_tables[4, 261:270]) /
  (sum(cm_c32_tables[4, 261:270]) + sum(cm_c32_tables[3, 261:270])), 6)
k27w3c32_truescore <- round((2 * k27w3c32_tpr * k27w3c32_tnr) /
  (k27w3c32_tpr + k27w3c32_tnr), 6)

k29w1c32_tpr <- round(sum(cm_c32_tables[1, 271:280]) /
  (sum(cm_c32_tables[1, 271:280]) + sum(cm_c32_tables[2, 271:280])), 6)
k29w1c32_tnr <- round(sum(cm_c32_tables[4, 271:280]) /

```

```

(sum(cm_c32_tables[4, 271:280]) + sum(cm_c32_tables[3, 271:280])), 6)
k29w1c32_truescore <- round((2 * k29w1c32_tpr * k29w1c32_tnr) /
(k29w1c32_tpr + k29w1c32_tnr), 6)

k29w2c32_tpr <- round(sum(cm_c32_tables[1, 281:290]) /
(sum(cm_c32_tables[1, 281:290]) + sum(cm_c32_tables[2, 281:290])), 6)
k29w2c32_tnr <- round(sum(cm_c32_tables[4, 281:290]) /
(sum(cm_c32_tables[4, 281:290]) + sum(cm_c32_tables[3, 281:290])), 6)
k29w2c32_truescore <- round((2 * k29w2c32_tpr * k29w2c32_tnr) /
(k29w2c32_tpr + k29w2c32_tnr), 6)

k29w3c32_tpr <- round(sum(cm_c32_tables[1, 291:300]) /
(sum(cm_c32_tables[1, 291:300]) + sum(cm_c32_tables[2, 291:300])), 6)
k29w3c32_tnr <- round(sum(cm_c32_tables[4, 291:300]) /
(sum(cm_c32_tables[4, 291:300]) + sum(cm_c32_tables[3, 291:300])), 6)
k29w3c32_truescore <- round((2 * k29w3c32_tpr * k29w3c32_tnr) /
(k29w3c32_tpr + k29w3c32_tnr), 6)

k31w1c32_tpr <- round(sum(cm_c32_tables[1, 301:310]) /
(sum(cm_c32_tables[1, 301:310]) + sum(cm_c32_tables[2, 301:310])), 6)
k31w1c32_tnr <- round(sum(cm_c32_tables[4, 301:310]) /
(sum(cm_c32_tables[4, 301:310]) + sum(cm_c32_tables[3, 301:310])), 6)
k31w1c32_truescore <- round((2 * k31w1c32_tpr * k31w1c32_tnr) /
(k31w1c32_tpr + k31w1c32_tnr), 6)

k31w2c32_tpr <- round(sum(cm_c32_tables[1, 311:320]) /
(sum(cm_c32_tables[1, 311:320]) + sum(cm_c32_tables[2, 311:320])), 6)
k31w2c32_tnr <- round(sum(cm_c32_tables[4, 311:320]) /
(sum(cm_c32_tables[4, 311:320]) + sum(cm_c32_tables[3, 311:320])), 6)
k31w2c32_truescore <- round((2 * k31w2c32_tpr * k31w2c32_tnr) /
(k31w2c32_tpr + k31w2c32_tnr), 6)

k31w3c32_tpr <- round(sum(cm_c32_tables[1, 321:330]) /
(sum(cm_c32_tables[1, 321:330]) + sum(cm_c32_tables[2, 321:330])), 6)
k31w3c32_tnr <- round(sum(cm_c32_tables[4, 321:330]) /
(sum(cm_c32_tables[4, 321:330]) + sum(cm_c32_tables[3, 321:330])), 6)
k31w3c32_truescore <- round((2 * k31w3c32_tpr * k31w3c32_tnr) /
(k31w3c32_tpr + k31w3c32_tnr), 6)

k33w1c32_tpr <- round(sum(cm_c32_tables[1, 331:340]) /
(sum(cm_c32_tables[1, 331:340]) + sum(cm_c32_tables[2, 331:340])), 6)
k33w1c32_tnr <- round(sum(cm_c32_tables[4, 331:340]) /
(sum(cm_c32_tables[4, 331:340]) + sum(cm_c32_tables[3, 331:340])), 6)
k33w1c32_truescore <- round((2 * k33w1c32_tpr * k33w1c32_tnr) /
(k33w1c32_tpr + k33w1c32_tnr), 6)

k33w2c32_tpr <- round(sum(cm_c32_tables[1, 341:350]) /
(sum(cm_c32_tables[1, 341:350]) + sum(cm_c32_tables[2, 341:350])), 6)
k33w2c32_tnr <- round(sum(cm_c32_tables[4, 341:350]) /
(sum(cm_c32_tables[4, 341:350]) + sum(cm_c32_tables[3, 341:350])), 6)
k33w2c32_truescore <- round((2 * k33w2c32_tpr * k33w2c32_tnr) /

```

```

(k33w2c32_tpr + k33w2c32_tnr), 6)

k33w3c32_tpr <- round(sum(cm_c32_tables[1, 351:360]) /
  (sum(cm_c32_tables[1, 351:360]) + sum(cm_c32_tables[2, 351:360])), 6)
k33w3c32_tnr <- round(sum(cm_c32_tables[4, 351:360]) /
  (sum(cm_c32_tables[4, 351:360]) + sum(cm_c32_tables[3, 351:360])), 6)
k33w3c32_truescore <- round((2 * k33w3c32_tpr * k33w3c32_tnr) /
  (k33w3c32_tpr + k33w3c32_tnr), 6)

k35w1c32_tpr <- round(sum(cm_c32_tables[1, 361:370]) /
  (sum(cm_c32_tables[1, 361:370]) + sum(cm_c32_tables[2, 361:370])), 6)
k35w1c32_tnr <- round(sum(cm_c32_tables[4, 361:370]) /
  (sum(cm_c32_tables[4, 361:370]) + sum(cm_c32_tables[3, 361:370])), 6)
k35w1c32_truescore <- round((2 * k35w1c32_tpr * k35w1c32_tnr) /
  (k35w1c32_tpr + k35w1c32_tnr), 6)

k35w2c32_tpr <- round(sum(cm_c32_tables[1, 371:380]) /
  (sum(cm_c32_tables[1, 371:380]) + sum(cm_c32_tables[2, 371:380])), 6)
k35w2c32_tnr <- round(sum(cm_c32_tables[4, 371:380]) /
  (sum(cm_c32_tables[4, 371:380]) + sum(cm_c32_tables[3, 371:380])), 6)
k35w2c32_truescore <- round((2 * k35w2c32_tpr * k35w2c32_tnr) /
  (k35w2c32_tpr + k35w2c32_tnr), 6)

k35w3c32_tpr <- round(sum(cm_c32_tables[1, 381:390]) /
  (sum(cm_c32_tables[1, 381:390]) + sum(cm_c32_tables[2, 381:390])), 6)
k35w3c32_tnr <- round(sum(cm_c32_tables[4, 381:390]) /
  (sum(cm_c32_tables[4, 381:390]) + sum(cm_c32_tables[3, 381:390])), 6)
k35w3c32_truescore <- round((2 * k35w3c32_tpr * k35w3c32_tnr) /
  (k35w3c32_tpr + k35w3c32_tnr), 6)

k37w1c32_tpr <- round(sum(cm_c32_tables[1, 391:400]) /
  (sum(cm_c32_tables[1, 391:400]) + sum(cm_c32_tables[2, 391:400])), 6)
k37w1c32_tnr <- round(sum(cm_c32_tables[4, 391:400]) /
  (sum(cm_c32_tables[4, 391:400]) + sum(cm_c32_tables[3, 391:400])), 6)
k37w1c32_truescore <- round((2 * k37w1c32_tpr * k37w1c32_tnr) /
  (k37w1c32_tpr + k37w1c32_tnr), 6)

k37w2c32_tpr <- round(sum(cm_c32_tables[1, 401:410]) /
  (sum(cm_c32_tables[1, 401:410]) + sum(cm_c32_tables[2, 401:410])), 6)
k37w2c32_tnr <- round(sum(cm_c32_tables[4, 401:410]) /
  (sum(cm_c32_tables[4, 401:410]) + sum(cm_c32_tables[3, 401:410])), 6)
k37w2c32_truescore <- round((2 * k37w2c32_tpr * k37w2c32_tnr) /
  (k37w2c32_tpr + k37w2c32_tnr), 6)

k37w3c32_tpr <- round(sum(cm_c32_tables[1, 411:420]) /
  (sum(cm_c32_tables[1, 411:420]) + sum(cm_c32_tables[2, 411:420])), 6)
k37w3c32_tnr <- round(sum(cm_c32_tables[4, 411:420]) /
  (sum(cm_c32_tables[4, 411:420]) + sum(cm_c32_tables[3, 411:420])), 6)
k37w3c32_truescore <- round((2 * k37w3c32_tpr * k37w3c32_tnr) /
  (k37w3c32_tpr + k37w3c32_tnr), 6)

```

```

k39w1c32_tpr <- round(sum(cm_c32_tables[1, 421:430]) /
  (sum(cm_c32_tables[1, 421:430]) + sum(cm_c32_tables[2, 421:430])), 6)
k39w1c32_tnr <- round(sum(cm_c32_tables[4, 421:430]) /
  (sum(cm_c32_tables[4, 421:430]) + sum(cm_c32_tables[3, 421:430])), 6)
k39w1c32_truescore <- round((2 * k39w1c32_tpr * k39w1c32_tnr) /
  (k39w1c32_tpr + k39w1c32_tnr), 6)

k39w2c32_tpr <- round(sum(cm_c32_tables[1, 431:440]) /
  (sum(cm_c32_tables[1, 431:440]) + sum(cm_c32_tables[2, 431:440])), 6)
k39w2c32_tnr <- round(sum(cm_c32_tables[4, 431:440]) /
  (sum(cm_c32_tables[4, 431:440]) + sum(cm_c32_tables[3, 431:440])), 6)
k39w2c32_truescore <- round((2 * k39w2c32_tpr * k39w2c32_tnr) /
  (k39w2c32_tpr + k39w2c32_tnr), 6)

k39w3c32_tpr <- round(sum(cm_c32_tables[1, 441:450]) /
  (sum(cm_c32_tables[1, 441:450]) + sum(cm_c32_tables[2, 441:450])), 6)
k39w3c32_tnr <- round(sum(cm_c32_tables[4, 441:450]) /
  (sum(cm_c32_tables[4, 441:450]) + sum(cm_c32_tables[3, 441:450])), 6)
k39w3c32_truescore <- round((2 * k39w3c32_tpr * k39w3c32_tnr) /
  (k39w3c32_tpr + k39w3c32_tnr), 6)

# Compile the 0.32 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c32_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.32,
  TPR = c(k11w1c32_tpr, k11w2c32_tpr, k11w3c32_tpr, k13w1c32_tpr,
    k13w2c32_tpr, k13w3c32_tpr, k15w1c32_tpr, k15w2c32_tpr,
    k15w3c32_tpr, k17w1c32_tpr, k17w2c32_tpr, k17w3c32_tpr,
    k19w1c32_tpr, k19w2c32_tpr, k19w3c32_tpr, k21w1c32_tpr,
    k21w2c32_tpr, k21w3c32_tpr, k23w1c32_tpr, k23w2c32_tpr,
    k23w3c32_tpr, k25w1c32_tpr, k25w2c32_tpr, k25w3c32_tpr,
    k27w1c32_tpr, k27w2c32_tpr, k27w3c32_tpr, k29w1c32_tpr,
    k29w2c32_tpr, k29w3c32_tpr, k31w1c32_tpr, k31w2c32_tpr,
    k31w3c32_tpr, k33w1c32_tpr, k33w2c32_tpr, k33w3c32_tpr,
    k35w1c32_tpr, k35w2c32_tpr, k35w3c32_tpr, k37w1c32_tpr,
    k37w2c32_tpr, k37w3c32_tpr, k39w1c32_tpr, k39w2c32_tpr,
    k39w3c32_tpr),
  TNR = c(k11w1c32_tnr, k11w2c32_tnr, k11w3c32_tnr, k13w1c32_tnr,
    k13w2c32_tnr, k13w3c32_tnr, k15w1c32_tnr, k15w2c32_tnr,
    k15w3c32_tnr, k17w1c32_tnr, k17w2c32_tnr, k17w3c32_tnr,
    k19w1c32_tnr, k19w2c32_tnr, k19w3c32_tnr, k21w1c32_tnr,
    k21w2c32_tnr, k21w3c32_tnr, k23w1c32_tnr, k23w2c32_tnr,
    k23w3c32_tnr, k25w1c32_tnr, k25w2c32_tnr, k25w3c32_tnr,
    k27w1c32_tnr, k27w2c32_tnr, k27w3c32_tnr, k29w1c32_tnr,
    k29w2c32_tnr, k29w3c32_tnr, k31w1c32_tnr, k31w2c32_tnr,
    k31w3c32_tnr, k33w1c32_tnr, k33w2c32_tnr, k33w3c32_tnr)

```

```

k35w1c32_tnr, k35w2c32_tnr, k35w3c32_tnr, k37w1c32_tnr,
k37w2c32_tnr, k37w3c32_tnr, k39w1c32_tnr, k39w2c32_tnr,
k39w3c32_tnr),
Truescore = c(k11w1c32_truescore, k11w2c32_truescore,
k11w3c32_truescore, k13w1c32_truescore,
k13w2c32_truescore, k13w3c32_truescore,
k15w1c32_truescore, k15w2c32_truescore,
k15w3c32_truescore, k17w1c32_truescore,
k17w2c32_truescore, k17w3c32_truescore,
k19w1c32_truescore, k19w2c32_truescore,
k19w3c32_truescore, k21w1c32_truescore,
k21w2c32_truescore, k21w3c32_truescore,
k23w1c32_truescore, k23w2c32_truescore,
k23w3c32_truescore, k25w1c32_truescore,
k25w2c32_truescore, k25w3c32_truescore,
k27w1c32_truescore, k27w2c32_truescore,
k27w3c32_truescore, k29w1c32_truescore,
k29w2c32_truescore, k29w3c32_truescore,
k31w1c32_truescore, k31w2c32_truescore,
k31w3c32_truescore, k33w1c32_truescore,
k33w2c32_truescore, k33w3c32_truescore,
k35w1c32_truescore, k35w2c32_truescore,
k35w3c32_truescore, k37w1c32_truescore,
k37w2c32_truescore, k37w3c32_truescore,
k39w1c32_truescore, k39w2c32_truescore,
k39w3c32_truescore))

```

`knitr::kable(c32_results[1:45,], caption = "c32_results")`

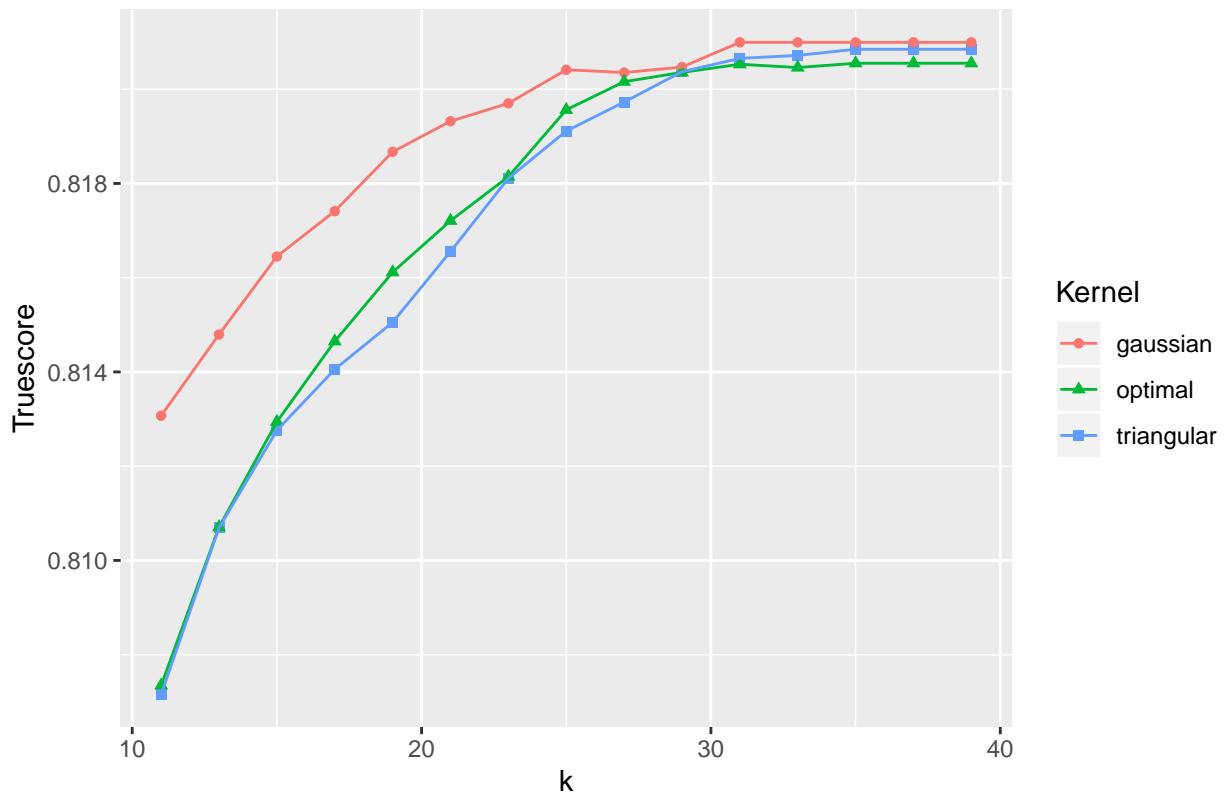
Table 41: c32_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.32	0.793675	0.821111	0.807160
11	gaussian	0.32	0.806878	0.819360	0.813071
11	optimal	0.32	0.792821	0.822416	0.807347
13	triangular	0.32	0.800753	0.820880	0.810692
13	gaussian	0.32	0.810894	0.818733	0.814795
13	optimal	0.32	0.800653	0.821012	0.810705
15	triangular	0.32	0.805371	0.820285	0.812760
15	gaussian	0.32	0.815261	0.817643	0.816450
15	optimal	0.32	0.806225	0.819773	0.812943
17	triangular	0.32	0.808835	0.819344	0.814056
17	gaussian	0.32	0.818072	0.816751	0.817411
17	optimal	0.32	0.809588	0.819773	0.814649
19	triangular	0.32	0.811797	0.818336	0.815053
19	gaussian	0.32	0.820884	0.816470	0.818671
19	optimal	0.32	0.813353	0.818898	0.816116
21	triangular	0.32	0.815060	0.818056	0.816555
21	gaussian	0.32	0.822339	0.816321	0.819319
21	optimal	0.32	0.816215	0.818204	0.817208
23	triangular	0.32	0.818624	0.817593	0.818108
23	gaussian	0.32	0.823544	0.815892	0.819700
23	optimal	0.32	0.818775	0.817511	0.818143

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.32	0.820934	0.817296	0.819111
25	gaussian	0.32	0.825151	0.815727	0.820412
25	optimal	0.32	0.822139	0.816999	0.819561
27	triangular	0.32	0.822590	0.816883	0.819727
27	gaussian	0.32	0.825151	0.815611	0.820353
27	optimal	0.32	0.823594	0.816751	0.820158
29	triangular	0.32	0.824900	0.815892	0.820371
29	gaussian	0.32	0.825351	0.815644	0.820469
29	optimal	0.32	0.824498	0.816255	0.820356
31	triangular	0.32	0.825351	0.816008	0.820653
31	gaussian	0.32	0.826958	0.815116	0.820994
31	optimal	0.32	0.825201	0.815909	0.820529
33	triangular	0.32	0.825753	0.815743	0.820717
33	gaussian	0.32	0.826958	0.815116	0.820994
33	optimal	0.32	0.825653	0.815330	0.820459
35	triangular	0.32	0.826456	0.815314	0.820847
35	gaussian	0.32	0.826958	0.815116	0.820994
35	optimal	0.32	0.825904	0.815264	0.820550
37	triangular	0.32	0.826456	0.815314	0.820847
37	gaussian	0.32	0.826958	0.815116	0.820994
37	optimal	0.32	0.825904	0.815264	0.820550
39	triangular	0.32	0.826456	0.815314	0.820847
39	gaussian	0.32	0.826958	0.815116	0.820994
39	optimal	0.32	0.825904	0.815264	0.820550

```
ggplot(c32_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.32 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.32 (Truescore)

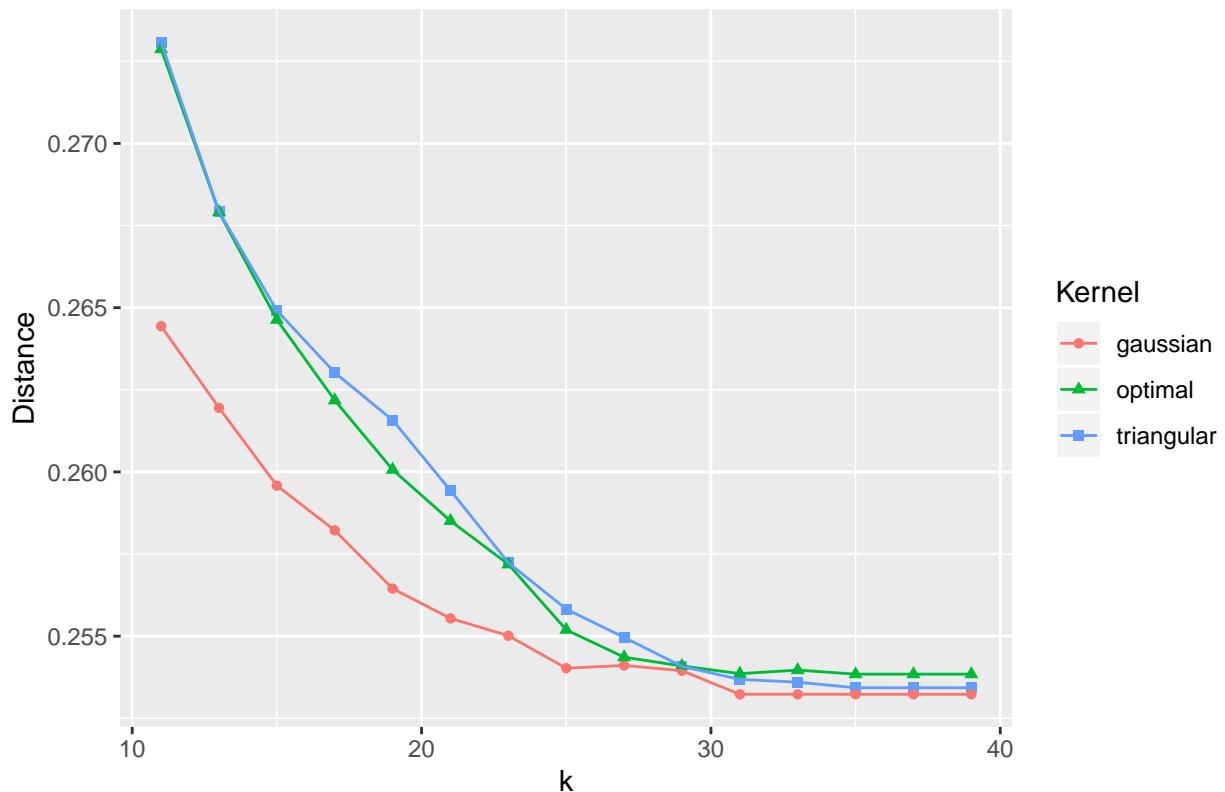


```
# For the Cutoff of 0.32, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c32_results <- c32_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c32_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.32 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.32 (Distance)



```
# For the Cutoff of 0.32, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c32_results$Truescore)
```

```
## [1] 0.820994
```

```
(c32_opt_k_ts <- c32_results$k[which.max(c32_results$Truescore)])
```

```
## [1] 31
```

```
(c32_opt_kernel_ts <- c32_results$Kernel[which.max(c32_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c32_opt_cut_ts <- c32_results$Cut[which.max(c32_results$Truescore)])
```

```
## [1] 0.32
```

```
(c32_opt_tpr_ts <- c32_results$TPR[which.max(c32_results$Truescore)])
```

```
## [1] 0.826958
```

```

(c32_opt_tnr_ts <- c32_results$TNR[which.max(c32_results$Truescore)])

## [1] 0.815116

(c32_opt_d_ts <- c32_results$Distance[which.max(c32_results$Truescore)])

## [1] 0.2532304

min(c32_results$Distance)

## [1] 0.2532304

(c32_opt_k_dist <- c32_results$k[which.min(c32_results$Distance)])

## [1] 31

(c32_opt_kernel_dist <- c32_results$Kernel[which.min(c32_results$Distance)])

## [1] "gaussian"

(c32_opt_cut_dist <- c32_results$Cut[which.min(c32_results$Distance)])

## [1] 0.32

(c32_opt_tpr_dist <- c32_results$TPR[which.min(c32_results$Distance)])

## [1] 0.826958

(c32_opt_tnr_dist <- c32_results$TNR[which.min(c32_results$Distance)])

## [1] 0.815116

(c32_opt_t_dist <- c32_results$Truescore[which.min(c32_results$Distance)])

## [1] 0.820994

#####
# 0.33 Cutoff
#####

# For the decision cutoff of 0.33, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c33 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred33, obs))
  confusionMatrix(ss$pred33, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c33) <- kwf_dfs_v

cm_c33_tables <- sapply(cm_c33, "[[", 2)
cm_c33_tables <- as_tibble(cm_c33_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.33.

k11w1c33_tpr <- round(sum(cm_c33_tables[1, 1:10]) /
  (sum(cm_c33_tables[1, 1:10]) + sum(cm_c33_tables[2, 1:10])), 6)
k11w1c33_tnr <- round(sum(cm_c33_tables[4, 1:10]) /
  (sum(cm_c33_tables[4, 1:10]) + sum(cm_c33_tables[3, 1:10])), 6)
k11w1c33_truescore <- round((2 * k11w1c33_tpr * k11w1c33_tnr) /
  (k11w1c33_tpr + k11w1c33_tnr), 6)

k11w2c33_tpr <- round(sum(cm_c33_tables[1, 11:20]) /
  (sum(cm_c33_tables[1, 11:20]) + sum(cm_c33_tables[2, 11:20])), 6)
k11w2c33_tnr <- round(sum(cm_c33_tables[4, 11:20]) /
  (sum(cm_c33_tables[4, 11:20]) + sum(cm_c33_tables[3, 11:20])), 6)
k11w2c33_truescore <- round((2 * k11w2c33_tpr * k11w2c33_tnr) /
  (k11w2c33_tpr + k11w2c33_tnr), 6)

k11w3c33_tpr <- round(sum(cm_c33_tables[1, 21:30]) /
  (sum(cm_c33_tables[1, 21:30]) + sum(cm_c33_tables[2, 21:30])), 6)
k11w3c33_tnr <- round(sum(cm_c33_tables[4, 21:30]) /
  (sum(cm_c33_tables[4, 21:30]) + sum(cm_c33_tables[3, 21:30])), 6)
k11w3c33_truescore <- round((2 * k11w3c33_tpr * k11w3c33_tnr) /
  (k11w3c33_tpr + k11w3c33_tnr), 6)

k13w1c33_tpr <- round(sum(cm_c33_tables[1, 31:40]) /
  (sum(cm_c33_tables[1, 31:40]) + sum(cm_c33_tables[2, 31:40])), 6)
k13w1c33_tnr <- round(sum(cm_c33_tables[4, 31:40]) /
  (sum(cm_c33_tables[4, 31:40]) + sum(cm_c33_tables[3, 31:40])), 6)
k13w1c33_truescore <- round((2 * k13w1c33_tpr * k13w1c33_tnr) /
  (k13w1c33_tpr + k13w1c33_tnr), 6)

k13w2c33_tpr <- round(sum(cm_c33_tables[1, 41:50]) /
  (sum(cm_c33_tables[1, 41:50]) + sum(cm_c33_tables[2, 41:50])), 6)
k13w2c33_tnr <- round(sum(cm_c33_tables[4, 41:50]) /
  (sum(cm_c33_tables[4, 41:50]) + sum(cm_c33_tables[3, 41:50])), 6)
k13w2c33_truescore <- round((2 * k13w2c33_tpr * k13w2c33_tnr) /
  (k13w2c33_tpr + k13w2c33_tnr), 6)

k13w3c33_tpr <- round(sum(cm_c33_tables[1, 51:60]) /
  (sum(cm_c33_tables[1, 51:60]) + sum(cm_c33_tables[2, 51:60])), 6)
k13w3c33_tnr <- round(sum(cm_c33_tables[4, 51:60]) /
  (sum(cm_c33_tables[4, 51:60]) + sum(cm_c33_tables[3, 51:60])), 6)
k13w3c33_truescore <- round((2 * k13w3c33_tpr * k13w3c33_tnr) /

```

```

(k13w3c33_tpr + k13w3c33_tnr), 6)

k15w1c33_tpr <- round(sum(cm_c33_tables[1, 61:70]) /
  (sum(cm_c33_tables[1, 61:70]) + sum(cm_c33_tables[2, 61:70])), 6)
k15w1c33_tnr <- round(sum(cm_c33_tables[4, 61:70]) /
  (sum(cm_c33_tables[4, 61:70]) + sum(cm_c33_tables[3, 61:70])), 6)
k15w1c33_truescore <- round((2 * k15w1c33_tpr * k15w1c33_tnr) /
  (k15w1c33_tpr + k15w1c33_tnr), 6)

k15w2c33_tpr <- round(sum(cm_c33_tables[1, 71:80]) /
  (sum(cm_c33_tables[1, 71:80]) + sum(cm_c33_tables[2, 71:80])), 6)
k15w2c33_tnr <- round(sum(cm_c33_tables[4, 71:80]) /
  (sum(cm_c33_tables[4, 71:80]) + sum(cm_c33_tables[3, 71:80])), 6)
k15w2c33_truescore <- round((2 * k15w2c33_tpr * k15w2c33_tnr) /
  (k15w2c33_tpr + k15w2c33_tnr), 6)

k15w3c33_tpr <- round(sum(cm_c33_tables[1, 81:90]) /
  (sum(cm_c33_tables[1, 81:90]) + sum(cm_c33_tables[2, 81:90])), 6)
k15w3c33_tnr <- round(sum(cm_c33_tables[4, 81:90]) /
  (sum(cm_c33_tables[4, 81:90]) + sum(cm_c33_tables[3, 81:90])), 6)
k15w3c33_truescore <- round((2 * k15w3c33_tpr * k15w3c33_tnr) /
  (k15w3c33_tpr + k15w3c33_tnr), 6)

k17w1c33_tpr <- round(sum(cm_c33_tables[1, 91:100]) /
  (sum(cm_c33_tables[1, 91:100]) + sum(cm_c33_tables[2, 91:100])), 6)
k17w1c33_tnr <- round(sum(cm_c33_tables[4, 91:100]) /
  (sum(cm_c33_tables[4, 91:100]) + sum(cm_c33_tables[3, 91:100])), 6)
k17w1c33_truescore <- round((2 * k17w1c33_tpr * k17w1c33_tnr) /
  (k17w1c33_tpr + k17w1c33_tnr), 6)

k17w2c33_tpr <- round(sum(cm_c33_tables[1, 101:110]) /
  (sum(cm_c33_tables[1, 101:110]) + sum(cm_c33_tables[2, 101:110])), 6)
k17w2c33_tnr <- round(sum(cm_c33_tables[4, 101:110]) /
  (sum(cm_c33_tables[4, 101:110]) + sum(cm_c33_tables[3, 101:110])), 6)
k17w2c33_truescore <- round((2 * k17w2c33_tpr * k17w2c33_tnr) /
  (k17w2c33_tpr + k17w2c33_tnr), 6)

k17w3c33_tpr <- round(sum(cm_c33_tables[1, 111:120]) /
  (sum(cm_c33_tables[1, 111:120]) + sum(cm_c33_tables[2, 111:120])), 6)
k17w3c33_tnr <- round(sum(cm_c33_tables[4, 111:120]) /
  (sum(cm_c33_tables[4, 111:120]) + sum(cm_c33_tables[3, 111:120])), 6)
k17w3c33_truescore <- round((2 * k17w3c33_tpr * k17w3c33_tnr) /
  (k17w3c33_tpr + k17w3c33_tnr), 6)

k19w1c33_tpr <- round(sum(cm_c33_tables[1, 121:130]) /
  (sum(cm_c33_tables[1, 121:130]) + sum(cm_c33_tables[2, 121:130])), 6)
k19w1c33_tnr <- round(sum(cm_c33_tables[4, 121:130]) /
  (sum(cm_c33_tables[4, 121:130]) + sum(cm_c33_tables[3, 121:130])), 6)
k19w1c33_truescore <- round((2 * k19w1c33_tpr * k19w1c33_tnr) /
  (k19w1c33_tpr + k19w1c33_tnr), 6)

```

```

k19w2c33_tpr <- round(sum(cm_c33_tables[1, 131:140]) /
  (sum(cm_c33_tables[1, 131:140]) + sum(cm_c33_tables[2, 131:140])), 6)
k19w2c33_tnr <- round(sum(cm_c33_tables[4, 131:140]) /
  (sum(cm_c33_tables[4, 131:140]) + sum(cm_c33_tables[3, 131:140])), 6)
k19w2c33_truescore <- round((2 * k19w2c33_tpr * k19w2c33_tnr) /
  (k19w2c33_tpr + k19w2c33_tnr), 6)

k19w3c33_tpr <- round(sum(cm_c33_tables[1, 141:150]) /
  (sum(cm_c33_tables[1, 141:150]) + sum(cm_c33_tables[2, 141:150])), 6)
k19w3c33_tnr <- round(sum(cm_c33_tables[4, 141:150]) /
  (sum(cm_c33_tables[4, 141:150]) + sum(cm_c33_tables[3, 141:150])), 6)
k19w3c33_truescore <- round((2 * k19w3c33_tpr * k19w3c33_tnr) /
  (k19w3c33_tpr + k19w3c33_tnr), 6)

k21w1c33_tpr <- round(sum(cm_c33_tables[1, 151:160]) /
  (sum(cm_c33_tables[1, 151:160]) + sum(cm_c33_tables[2, 151:160])), 6)
k21w1c33_tnr <- round(sum(cm_c33_tables[4, 151:160]) /
  (sum(cm_c33_tables[4, 151:160]) + sum(cm_c33_tables[3, 151:160])), 6)
k21w1c33_truescore <- round((2 * k21w1c33_tpr * k21w1c33_tnr) /
  (k21w1c33_tpr + k21w1c33_tnr), 6)

k21w2c33_tpr <- round(sum(cm_c33_tables[1, 161:170]) /
  (sum(cm_c33_tables[1, 161:170]) + sum(cm_c33_tables[2, 161:170])), 6)
k21w2c33_tnr <- round(sum(cm_c33_tables[4, 161:170]) /
  (sum(cm_c33_tables[4, 161:170]) + sum(cm_c33_tables[3, 161:170])), 6)
k21w2c33_truescore <- round((2 * k21w2c33_tpr * k21w2c33_tnr) /
  (k21w2c33_tpr + k21w2c33_tnr), 6)

k21w3c33_tpr <- round(sum(cm_c33_tables[1, 171:180]) /
  (sum(cm_c33_tables[1, 171:180]) + sum(cm_c33_tables[2, 171:180])), 6)
k21w3c33_tnr <- round(sum(cm_c33_tables[4, 171:180]) /
  (sum(cm_c33_tables[4, 171:180]) + sum(cm_c33_tables[3, 171:180])), 6)
k21w3c33_truescore <- round((2 * k21w3c33_tpr * k21w3c33_tnr) /
  (k21w3c33_tpr + k21w3c33_tnr), 6)

k23w1c33_tpr <- round(sum(cm_c33_tables[1, 181:190]) /
  (sum(cm_c33_tables[1, 181:190]) + sum(cm_c33_tables[2, 181:190])), 6)
k23w1c33_tnr <- round(sum(cm_c33_tables[4, 181:190]) /
  (sum(cm_c33_tables[4, 181:190]) + sum(cm_c33_tables[3, 181:190])), 6)
k23w1c33_truescore <- round((2 * k23w1c33_tpr * k23w1c33_tnr) /
  (k23w1c33_tpr + k23w1c33_tnr), 6)

k23w2c33_tpr <- round(sum(cm_c33_tables[1, 191:200]) /
  (sum(cm_c33_tables[1, 191:200]) + sum(cm_c33_tables[2, 191:200])), 6)
k23w2c33_tnr <- round(sum(cm_c33_tables[4, 191:200]) /
  (sum(cm_c33_tables[4, 191:200]) + sum(cm_c33_tables[3, 191:200])), 6)
k23w2c33_truescore <- round((2 * k23w2c33_tpr * k23w2c33_tnr) /
  (k23w2c33_tpr + k23w2c33_tnr), 6)

k23w3c33_tpr <- round(sum(cm_c33_tables[1, 201:210]) /
  (sum(cm_c33_tables[1, 201:210]) + sum(cm_c33_tables[2, 201:210])), 6)

```

```

k23w3c33_tnr <- round(sum(cm_c33_tables[4, 201:210]) /
  (sum(cm_c33_tables[4, 201:210]) + sum(cm_c33_tables[3, 201:210])), 6)
k23w3c33_truescore <- round((2 * k23w3c33_tpr * k23w3c33_tnr) /
  (k23w3c33_tpr + k23w3c33_tnr), 6)

k25w1c33_tpr <- round(sum(cm_c33_tables[1, 211:220]) /
  (sum(cm_c33_tables[1, 211:220]) + sum(cm_c33_tables[2, 211:220])), 6)
k25w1c33_tnr <- round(sum(cm_c33_tables[4, 211:220]) /
  (sum(cm_c33_tables[4, 211:220]) + sum(cm_c33_tables[3, 211:220])), 6)
k25w1c33_truescore <- round((2 * k25w1c33_tpr * k25w1c33_tnr) /
  (k25w1c33_tpr + k25w1c33_tnr), 6)

k25w2c33_tpr <- round(sum(cm_c33_tables[1, 221:230]) /
  (sum(cm_c33_tables[1, 221:230]) + sum(cm_c33_tables[2, 221:230])), 6)
k25w2c33_tnr <- round(sum(cm_c33_tables[4, 221:230]) /
  (sum(cm_c33_tables[4, 221:230]) + sum(cm_c33_tables[3, 221:230])), 6)
k25w2c33_truescore <- round((2 * k25w2c33_tpr * k25w2c33_tnr) /
  (k25w2c33_tpr + k25w2c33_tnr), 6)

k25w3c33_tpr <- round(sum(cm_c33_tables[1, 231:240]) /
  (sum(cm_c33_tables[1, 231:240]) + sum(cm_c33_tables[2, 231:240])), 6)
k25w3c33_tnr <- round(sum(cm_c33_tables[4, 231:240]) /
  (sum(cm_c33_tables[4, 231:240]) + sum(cm_c33_tables[3, 231:240])), 6)
k25w3c33_truescore <- round((2 * k25w3c33_tpr * k25w3c33_tnr) /
  (k25w3c33_tpr + k25w3c33_tnr), 6)

k27w1c33_tpr <- round(sum(cm_c33_tables[1, 241:250]) /
  (sum(cm_c33_tables[1, 241:250]) + sum(cm_c33_tables[2, 241:250])), 6)
k27w1c33_tnr <- round(sum(cm_c33_tables[4, 241:250]) /
  (sum(cm_c33_tables[4, 241:250]) + sum(cm_c33_tables[3, 241:250])), 6)
k27w1c33_truescore <- round((2 * k27w1c33_tpr * k27w1c33_tnr) /
  (k27w1c33_tpr + k27w1c33_tnr), 6)

k27w2c33_tpr <- round(sum(cm_c33_tables[1, 251:260]) /
  (sum(cm_c33_tables[1, 251:260]) + sum(cm_c33_tables[2, 251:260])), 6)
k27w2c33_tnr <- round(sum(cm_c33_tables[4, 251:260]) /
  (sum(cm_c33_tables[4, 251:260]) + sum(cm_c33_tables[3, 251:260])), 6)
k27w2c33_truescore <- round((2 * k27w2c33_tpr * k27w2c33_tnr) /
  (k27w2c33_tpr + k27w2c33_tnr), 6)

k27w3c33_tpr <- round(sum(cm_c33_tables[1, 261:270]) /
  (sum(cm_c33_tables[1, 261:270]) + sum(cm_c33_tables[2, 261:270])), 6)
k27w3c33_tnr <- round(sum(cm_c33_tables[4, 261:270]) /
  (sum(cm_c33_tables[4, 261:270]) + sum(cm_c33_tables[3, 261:270])), 6)
k27w3c33_truescore <- round((2 * k27w3c33_tpr * k27w3c33_tnr) /
  (k27w3c33_tpr + k27w3c33_tnr), 6)

k29w1c33_tpr <- round(sum(cm_c33_tables[1, 271:280]) /
  (sum(cm_c33_tables[1, 271:280]) + sum(cm_c33_tables[2, 271:280])), 6)
k29w1c33_tnr <- round(sum(cm_c33_tables[4, 271:280]) /

```

```

(sum(cm_c33_tables[4, 271:280]) + sum(cm_c33_tables[3, 271:280])), 6)
k29w1c33_truescore <- round((2 * k29w1c33_tpr * k29w1c33_tnr) /
(k29w1c33_tpr + k29w1c33_tnr), 6)

k29w2c33_tpr <- round(sum(cm_c33_tables[1, 281:290]) /
(sum(cm_c33_tables[1, 281:290]) + sum(cm_c33_tables[2, 281:290])), 6)
k29w2c33_tnr <- round(sum(cm_c33_tables[4, 281:290]) /
(sum(cm_c33_tables[4, 281:290]) + sum(cm_c33_tables[3, 281:290])), 6)
k29w2c33_truescore <- round((2 * k29w2c33_tpr * k29w2c33_tnr) /
(k29w2c33_tpr + k29w2c33_tnr), 6)

k29w3c33_tpr <- round(sum(cm_c33_tables[1, 291:300]) /
(sum(cm_c33_tables[1, 291:300]) + sum(cm_c33_tables[2, 291:300])), 6)
k29w3c33_tnr <- round(sum(cm_c33_tables[4, 291:300]) /
(sum(cm_c33_tables[4, 291:300]) + sum(cm_c33_tables[3, 291:300])), 6)
k29w3c33_truescore <- round((2 * k29w3c33_tpr * k29w3c33_tnr) /
(k29w3c33_tpr + k29w3c33_tnr), 6)

k31w1c33_tpr <- round(sum(cm_c33_tables[1, 301:310]) /
(sum(cm_c33_tables[1, 301:310]) + sum(cm_c33_tables[2, 301:310])), 6)
k31w1c33_tnr <- round(sum(cm_c33_tables[4, 301:310]) /
(sum(cm_c33_tables[4, 301:310]) + sum(cm_c33_tables[3, 301:310])), 6)
k31w1c33_truescore <- round((2 * k31w1c33_tpr * k31w1c33_tnr) /
(k31w1c33_tpr + k31w1c33_tnr), 6)

k31w2c33_tpr <- round(sum(cm_c33_tables[1, 311:320]) /
(sum(cm_c33_tables[1, 311:320]) + sum(cm_c33_tables[2, 311:320])), 6)
k31w2c33_tnr <- round(sum(cm_c33_tables[4, 311:320]) /
(sum(cm_c33_tables[4, 311:320]) + sum(cm_c33_tables[3, 311:320])), 6)
k31w2c33_truescore <- round((2 * k31w2c33_tpr * k31w2c33_tnr) /
(k31w2c33_tpr + k31w2c33_tnr), 6)

k31w3c33_tpr <- round(sum(cm_c33_tables[1, 321:330]) /
(sum(cm_c33_tables[1, 321:330]) + sum(cm_c33_tables[2, 321:330])), 6)
k31w3c33_tnr <- round(sum(cm_c33_tables[4, 321:330]) /
(sum(cm_c33_tables[4, 321:330]) + sum(cm_c33_tables[3, 321:330])), 6)
k31w3c33_truescore <- round((2 * k31w3c33_tpr * k31w3c33_tnr) /
(k31w3c33_tpr + k31w3c33_tnr), 6)

k33w1c33_tpr <- round(sum(cm_c33_tables[1, 331:340]) /
(sum(cm_c33_tables[1, 331:340]) + sum(cm_c33_tables[2, 331:340])), 6)
k33w1c33_tnr <- round(sum(cm_c33_tables[4, 331:340]) /
(sum(cm_c33_tables[4, 331:340]) + sum(cm_c33_tables[3, 331:340])), 6)
k33w1c33_truescore <- round((2 * k33w1c33_tpr * k33w1c33_tnr) /
(k33w1c33_tpr + k33w1c33_tnr), 6)

k33w2c33_tpr <- round(sum(cm_c33_tables[1, 341:350]) /
(sum(cm_c33_tables[1, 341:350]) + sum(cm_c33_tables[2, 341:350])), 6)
k33w2c33_tnr <- round(sum(cm_c33_tables[4, 341:350]) /
(sum(cm_c33_tables[4, 341:350]) + sum(cm_c33_tables[3, 341:350])), 6)
k33w2c33_truescore <- round((2 * k33w2c33_tpr * k33w2c33_tnr) /

```

```

(k33w2c33_tpr + k33w2c33_tnr), 6)

k33w3c33_tpr <- round(sum(cm_c33_tables[1, 351:360]) /
  (sum(cm_c33_tables[1, 351:360]) + sum(cm_c33_tables[2, 351:360])), 6)
k33w3c33_tnr <- round(sum(cm_c33_tables[4, 351:360]) /
  (sum(cm_c33_tables[4, 351:360]) + sum(cm_c33_tables[3, 351:360])), 6)
k33w3c33_truescore <- round((2 * k33w3c33_tpr * k33w3c33_tnr) /
  (k33w3c33_tpr + k33w3c33_tnr), 6)

k35w1c33_tpr <- round(sum(cm_c33_tables[1, 361:370]) /
  (sum(cm_c33_tables[1, 361:370]) + sum(cm_c33_tables[2, 361:370])), 6)
k35w1c33_tnr <- round(sum(cm_c33_tables[4, 361:370]) /
  (sum(cm_c33_tables[4, 361:370]) + sum(cm_c33_tables[3, 361:370])), 6)
k35w1c33_truescore <- round((2 * k35w1c33_tpr * k35w1c33_tnr) /
  (k35w1c33_tpr + k35w1c33_tnr), 6)

k35w2c33_tpr <- round(sum(cm_c33_tables[1, 371:380]) /
  (sum(cm_c33_tables[1, 371:380]) + sum(cm_c33_tables[2, 371:380])), 6)
k35w2c33_tnr <- round(sum(cm_c33_tables[4, 371:380]) /
  (sum(cm_c33_tables[4, 371:380]) + sum(cm_c33_tables[3, 371:380])), 6)
k35w2c33_truescore <- round((2 * k35w2c33_tpr * k35w2c33_tnr) /
  (k35w2c33_tpr + k35w2c33_tnr), 6)

k35w3c33_tpr <- round(sum(cm_c33_tables[1, 381:390]) /
  (sum(cm_c33_tables[1, 381:390]) + sum(cm_c33_tables[2, 381:390])), 6)
k35w3c33_tnr <- round(sum(cm_c33_tables[4, 381:390]) /
  (sum(cm_c33_tables[4, 381:390]) + sum(cm_c33_tables[3, 381:390])), 6)
k35w3c33_truescore <- round((2 * k35w3c33_tpr * k35w3c33_tnr) /
  (k35w3c33_tpr + k35w3c33_tnr), 6)

k37w1c33_tpr <- round(sum(cm_c33_tables[1, 391:400]) /
  (sum(cm_c33_tables[1, 391:400]) + sum(cm_c33_tables[2, 391:400])), 6)
k37w1c33_tnr <- round(sum(cm_c33_tables[4, 391:400]) /
  (sum(cm_c33_tables[4, 391:400]) + sum(cm_c33_tables[3, 391:400])), 6)
k37w1c33_truescore <- round((2 * k37w1c33_tpr * k37w1c33_tnr) /
  (k37w1c33_tpr + k37w1c33_tnr), 6)

k37w2c33_tpr <- round(sum(cm_c33_tables[1, 401:410]) /
  (sum(cm_c33_tables[1, 401:410]) + sum(cm_c33_tables[2, 401:410])), 6)
k37w2c33_tnr <- round(sum(cm_c33_tables[4, 401:410]) /
  (sum(cm_c33_tables[4, 401:410]) + sum(cm_c33_tables[3, 401:410])), 6)
k37w2c33_truescore <- round((2 * k37w2c33_tpr * k37w2c33_tnr) /
  (k37w2c33_tpr + k37w2c33_tnr), 6)

k37w3c33_tpr <- round(sum(cm_c33_tables[1, 411:420]) /
  (sum(cm_c33_tables[1, 411:420]) + sum(cm_c33_tables[2, 411:420])), 6)
k37w3c33_tnr <- round(sum(cm_c33_tables[4, 411:420]) /
  (sum(cm_c33_tables[4, 411:420]) + sum(cm_c33_tables[3, 411:420])), 6)
k37w3c33_truescore <- round((2 * k37w3c33_tpr * k37w3c33_tnr) /
  (k37w3c33_tpr + k37w3c33_tnr), 6)

```

```

k39w1c33_tpr <- round(sum(cm_c33_tables[1, 421:430]) /
  (sum(cm_c33_tables[1, 421:430]) + sum(cm_c33_tables[2, 421:430])), 6)
k39w1c33_tnr <- round(sum(cm_c33_tables[4, 421:430]) /
  (sum(cm_c33_tables[4, 421:430]) + sum(cm_c33_tables[3, 421:430])), 6)
k39w1c33_truescore <- round((2 * k39w1c33_tpr * k39w1c33_tnr) /
  (k39w1c33_tpr + k39w1c33_tnr), 6)

k39w2c33_tpr <- round(sum(cm_c33_tables[1, 431:440]) /
  (sum(cm_c33_tables[1, 431:440]) + sum(cm_c33_tables[2, 431:440])), 6)
k39w2c33_tnr <- round(sum(cm_c33_tables[4, 431:440]) /
  (sum(cm_c33_tables[4, 431:440]) + sum(cm_c33_tables[3, 431:440])), 6)
k39w2c33_truescore <- round((2 * k39w2c33_tpr * k39w2c33_tnr) /
  (k39w2c33_tpr + k39w2c33_tnr), 6)

k39w3c33_tpr <- round(sum(cm_c33_tables[1, 441:450]) /
  (sum(cm_c33_tables[1, 441:450]) + sum(cm_c33_tables[2, 441:450])), 6)
k39w3c33_tnr <- round(sum(cm_c33_tables[4, 441:450]) /
  (sum(cm_c33_tables[4, 441:450]) + sum(cm_c33_tables[3, 441:450])), 6)
k39w3c33_truescore <- round((2 * k39w3c33_tpr * k39w3c33_tnr) /
  (k39w3c33_tpr + k39w3c33_tnr), 6)

# Compile the 0.33 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c33_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.33,
  TPR = c(k11w1c33_tpr, k11w2c33_tpr, k11w3c33_tpr, k13w1c33_tpr,
    k13w2c33_tpr, k13w3c33_tpr, k15w1c33_tpr, k15w2c33_tpr,
    k15w3c33_tpr, k17w1c33_tpr, k17w2c33_tpr, k17w3c33_tpr,
    k19w1c33_tpr, k19w2c33_tpr, k19w3c33_tpr, k21w1c33_tpr,
    k21w2c33_tpr, k21w3c33_tpr, k23w1c33_tpr, k23w2c33_tpr,
    k23w3c33_tpr, k25w1c33_tpr, k25w2c33_tpr, k25w3c33_tpr,
    k27w1c33_tpr, k27w2c33_tpr, k27w3c33_tpr, k29w1c33_tpr,
    k29w2c33_tpr, k29w3c33_tpr, k31w1c33_tpr, k31w2c33_tpr,
    k31w3c33_tpr, k33w1c33_tpr, k33w2c33_tpr, k33w3c33_tpr,
    k35w1c33_tpr, k35w2c33_tpr, k35w3c33_tpr, k37w1c33_tpr,
    k37w2c33_tpr, k37w3c33_tpr, k39w1c33_tpr, k39w2c33_tpr,
    k39w3c33_tpr),
  TNR = c(k11w1c33_tnr, k11w2c33_tnr, k11w3c33_tnr, k13w1c33_tnr,
    k13w2c33_tnr, k13w3c33_tnr, k15w1c33_tnr, k15w2c33_tnr,
    k15w3c33_tnr, k17w1c33_tnr, k17w2c33_tnr, k17w3c33_tnr,
    k19w1c33_tnr, k19w2c33_tnr, k19w3c33_tnr, k21w1c33_tnr,
    k21w2c33_tnr, k21w3c33_tnr, k23w1c33_tnr, k23w2c33_tnr,
    k23w3c33_tnr, k25w1c33_tnr, k25w2c33_tnr, k25w3c33_tnr,
    k27w1c33_tnr, k27w2c33_tnr, k27w3c33_tnr, k29w1c33_tnr,
    k29w2c33_tnr, k29w3c33_tnr, k31w1c33_tnr, k31w2c33_tnr,
    k31w3c33_tnr, k33w1c33_tnr, k33w2c33_tnr, k33w3c33_tnr),
  Truescore = k39w1c33_truescore)

```

```

k35w1c33_tnr, k35w2c33_tnr, k35w3c33_tnr, k37w1c33_tnr,
k37w2c33_tnr, k37w3c33_tnr, k39w1c33_tnr, k39w2c33_tnr,
k39w3c33_tnr),
Truescore = c(k11w1c33_truescore, k11w2c33_truescore,
k11w3c33_truescore, k13w1c33_truescore,
k13w2c33_truescore, k13w3c33_truescore,
k15w1c33_truescore, k15w2c33_truescore,
k15w3c33_truescore, k17w1c33_truescore,
k17w2c33_truescore, k17w3c33_truescore,
k19w1c33_truescore, k19w2c33_truescore,
k19w3c33_truescore, k21w1c33_truescore,
k21w2c33_truescore, k21w3c33_truescore,
k23w1c33_truescore, k23w2c33_truescore,
k23w3c33_truescore, k25w1c33_truescore,
k25w2c33_truescore, k25w3c33_truescore,
k27w1c33_truescore, k27w2c33_truescore,
k27w3c33_truescore, k29w1c33_truescore,
k29w2c33_truescore, k29w3c33_truescore,
k31w1c33_truescore, k31w2c33_truescore,
k31w3c33_truescore, k33w1c33_truescore,
k33w2c33_truescore, k33w3c33_truescore,
k35w1c33_truescore, k35w2c33_truescore,
k35w3c33_truescore, k37w1c33_truescore,
k37w2c33_truescore, k37w3c33_truescore,
k39w1c33_truescore, k39w2c33_truescore,
k39w3c33_truescore))

```

`knitr::kable(c33_results[1:45,], caption = "c33_results")`

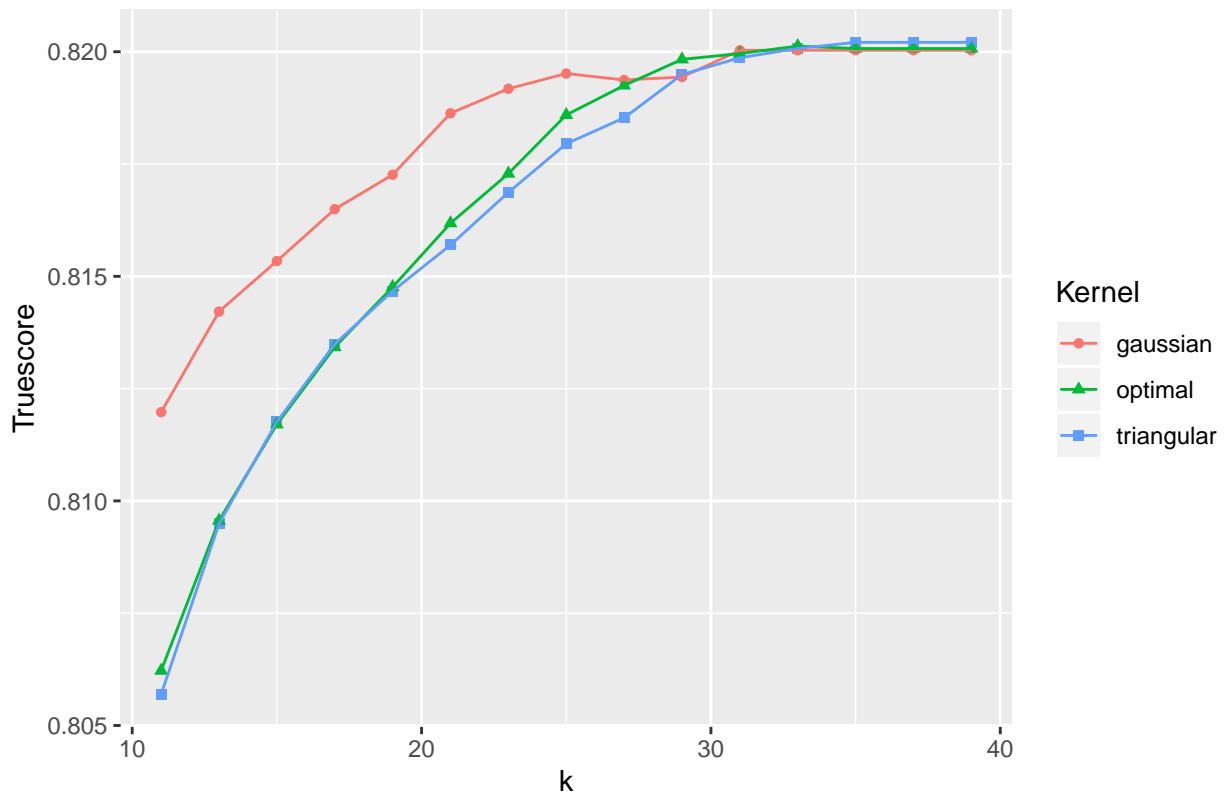
Table 42: c33_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.33	0.786094	0.826297	0.805694
11	gaussian	0.33	0.798946	0.825439	0.811976
11	optimal	0.33	0.786094	0.827404	0.806220
13	triangular	0.33	0.793022	0.826661	0.809492
13	gaussian	0.33	0.803765	0.824943	0.814216
13	optimal	0.33	0.792972	0.826859	0.809561
15	triangular	0.33	0.797942	0.826066	0.811760
15	gaussian	0.33	0.806827	0.824035	0.815340
15	optimal	0.33	0.797741	0.826165	0.811704
17	triangular	0.33	0.801606	0.825736	0.813492
17	gaussian	0.33	0.809538	0.823572	0.816495
17	optimal	0.33	0.801054	0.826182	0.813424
19	triangular	0.33	0.804418	0.825191	0.814672
19	gaussian	0.33	0.811546	0.823060	0.817262
19	optimal	0.33	0.804167	0.825637	0.814761
21	triangular	0.33	0.807179	0.824398	0.815698
21	gaussian	0.33	0.814408	0.822895	0.818630
21	optimal	0.33	0.807731	0.824811	0.816182
23	triangular	0.33	0.809538	0.824332	0.816868
23	gaussian	0.33	0.815914	0.822466	0.819177
23	optimal	0.33	0.810442	0.824249	0.817287

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.33	0.812149	0.823836	0.817951
25	gaussian	0.33	0.816667	0.822383	0.819515
25	optimal	0.33	0.813303	0.823952	0.818593
27	triangular	0.33	0.813855	0.823258	0.818529
27	gaussian	0.33	0.816416	0.822350	0.819372
27	optimal	0.33	0.814859	0.823688	0.819250
29	triangular	0.33	0.816265	0.822746	0.819493
29	gaussian	0.33	0.816365	0.822532	0.819437
29	optimal	0.33	0.816416	0.823275	0.819831
31	triangular	0.33	0.817219	0.822532	0.819867
31	gaussian	0.33	0.818022	0.822053	0.820033
31	optimal	0.33	0.816817	0.823126	0.819959
33	triangular	0.33	0.817972	0.822185	0.820073
33	gaussian	0.33	0.818022	0.822053	0.820033
33	optimal	0.33	0.817721	0.822548	0.820127
35	triangular	0.33	0.818424	0.822003	0.820210
35	gaussian	0.33	0.818022	0.822053	0.820033
35	optimal	0.33	0.817871	0.822284	0.820072
37	triangular	0.33	0.818424	0.822003	0.820210
37	gaussian	0.33	0.818022	0.822053	0.820033
37	optimal	0.33	0.817871	0.822284	0.820072
39	triangular	0.33	0.818424	0.822003	0.820210
39	gaussian	0.33	0.818022	0.822053	0.820033
39	optimal	0.33	0.817871	0.822284	0.820072

```
ggplot(c33_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.33 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.33 (Truescore)

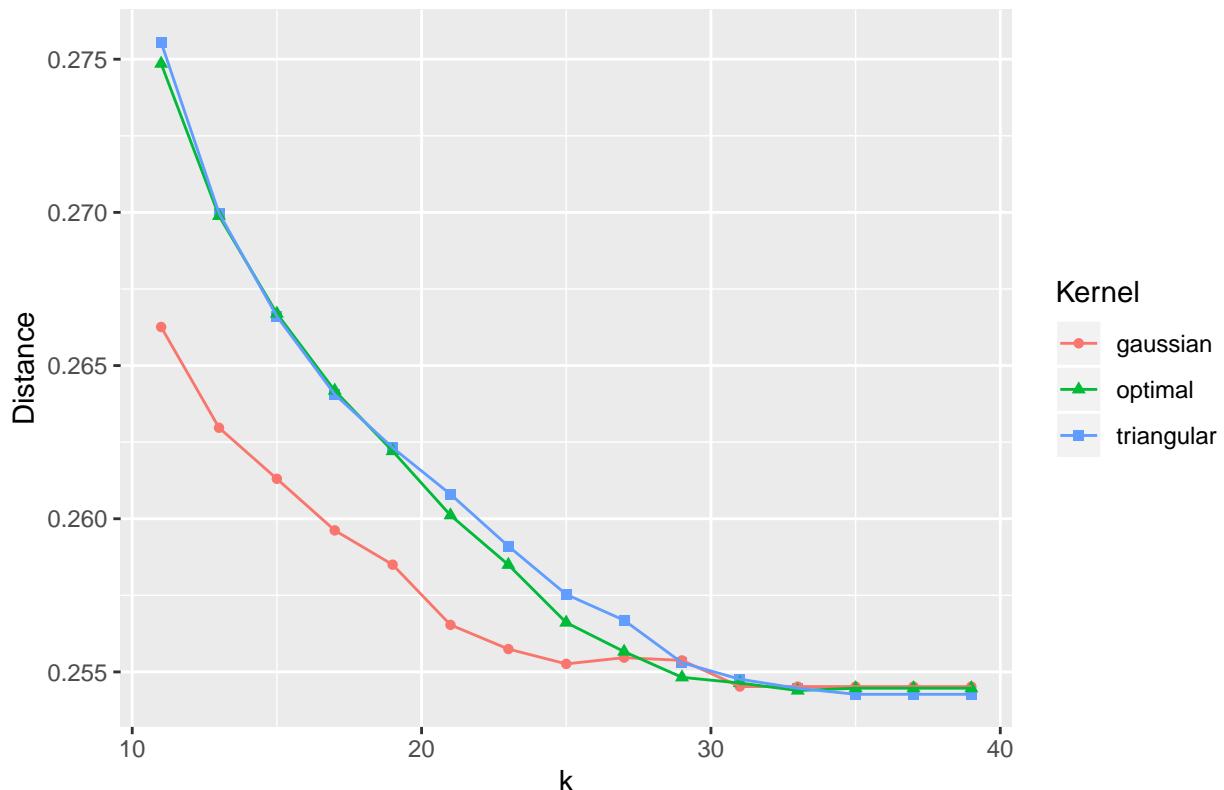


```
# For the Cutoff of 0.33, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c33_results <- c33_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c33_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.33 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.33 (Distance)



```
# For the Cutoff of 0.33, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c33_results$Truescore)
```

```
## [1] 0.82021
```

```
(c33_opt_k_ts <- c33_results$k[which.max(c33_results$Truescore)])
```

```
## [1] 35
```

```
(c33_opt_kernel_ts <- c33_results$Kernel[which.max(c33_results$Truescore)])
```

```
## [1] "triangular"
```

```
(c33_opt_cut_ts <- c33_results$Cut[which.max(c33_results$Truescore)])
```

```
## [1] 0.33
```

```
(c33_opt_tpr_ts <- c33_results$TPR[which.max(c33_results$Truescore)])
```

```
## [1] 0.818424
```

```

(c33_opt_tnr_ts <- c33_results$TNR[which.max(c33_results$Truescore)])

## [1] 0.822003

(c33_opt_d_ts <- c33_results$Distance[which.max(c33_results$Truescore)])

## [1] 0.2542691

min(c33_results$Distance)

## [1] 0.2542691

(c33_opt_k_dist <- c33_results$k[which.min(c33_results$Distance)])

## [1] 35

(c33_opt_kernel_dist <- c33_results$Kernel[which.min(c33_results$Distance)])

## [1] "triangular"

(c33_opt_cut_dist <- c33_results$Cut[which.min(c33_results$Distance)])

## [1] 0.33

(c33_opt_tpr_dist <- c33_results$TPR[which.min(c33_results$Distance)])

## [1] 0.818424

(c33_opt_tnr_dist <- c33_results$TNR[which.min(c33_results$Distance)])

## [1] 0.822003

(c33_opt_t_dist <- c33_results$Truescore[which.min(c33_results$Distance)])

## [1] 0.82021

#####
# 0.34 Cutoff
#####

# For the decision cutoff of 0.34, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c34 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred34, obs))
  confusionMatrix(ss$pred34, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c34) <- kwf_dfs_v

cm_c34_tables <- sapply(cm_c34, "[[", 2)
cm_c34_tables <- as_tibble(cm_c34_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.34.

k11w1c34_tpr <- round(sum(cm_c34_tables[1, 1:10]) /
  (sum(cm_c34_tables[1, 1:10]) + sum(cm_c34_tables[2, 1:10])), 6)
k11w1c34_tnr <- round(sum(cm_c34_tables[4, 1:10]) /
  (sum(cm_c34_tables[4, 1:10]) + sum(cm_c34_tables[3, 1:10])), 6)
k11w1c34_truescore <- round((2 * k11w1c34_tpr * k11w1c34_tnr) /
  (k11w1c34_tpr + k11w1c34_tnr), 6)

k11w2c34_tpr <- round(sum(cm_c34_tables[1, 11:20]) /
  (sum(cm_c34_tables[1, 11:20]) + sum(cm_c34_tables[2, 11:20])), 6)
k11w2c34_tnr <- round(sum(cm_c34_tables[4, 11:20]) /
  (sum(cm_c34_tables[4, 11:20]) + sum(cm_c34_tables[3, 11:20])), 6)
k11w2c34_truescore <- round((2 * k11w2c34_tpr * k11w2c34_tnr) /
  (k11w2c34_tpr + k11w2c34_tnr), 6)

k11w3c34_tpr <- round(sum(cm_c34_tables[1, 21:30]) /
  (sum(cm_c34_tables[1, 21:30]) + sum(cm_c34_tables[2, 21:30])), 6)
k11w3c34_tnr <- round(sum(cm_c34_tables[4, 21:30]) /
  (sum(cm_c34_tables[4, 21:30]) + sum(cm_c34_tables[3, 21:30])), 6)
k11w3c34_truescore <- round((2 * k11w3c34_tpr * k11w3c34_tnr) /
  (k11w3c34_tpr + k11w3c34_tnr), 6)

k13w1c34_tpr <- round(sum(cm_c34_tables[1, 31:40]) /
  (sum(cm_c34_tables[1, 31:40]) + sum(cm_c34_tables[2, 31:40])), 6)
k13w1c34_tnr <- round(sum(cm_c34_tables[4, 31:40]) /
  (sum(cm_c34_tables[4, 31:40]) + sum(cm_c34_tables[3, 31:40])), 6)
k13w1c34_truescore <- round((2 * k13w1c34_tpr * k13w1c34_tnr) /
  (k13w1c34_tpr + k13w1c34_tnr), 6)

k13w2c34_tpr <- round(sum(cm_c34_tables[1, 41:50]) /
  (sum(cm_c34_tables[1, 41:50]) + sum(cm_c34_tables[2, 41:50])), 6)
k13w2c34_tnr <- round(sum(cm_c34_tables[4, 41:50]) /
  (sum(cm_c34_tables[4, 41:50]) + sum(cm_c34_tables[3, 41:50])), 6)
k13w2c34_truescore <- round((2 * k13w2c34_tpr * k13w2c34_tnr) /
  (k13w2c34_tpr + k13w2c34_tnr), 6)

k13w3c34_tpr <- round(sum(cm_c34_tables[1, 51:60]) /
  (sum(cm_c34_tables[1, 51:60]) + sum(cm_c34_tables[2, 51:60])), 6)
k13w3c34_tnr <- round(sum(cm_c34_tables[4, 51:60]) /
  (sum(cm_c34_tables[4, 51:60]) + sum(cm_c34_tables[3, 51:60])), 6)
k13w3c34_truescore <- round((2 * k13w3c34_tpr * k13w3c34_tnr) /

```

```

(k13w3c34_tpr + k13w3c34_tnr), 6)

k15w1c34_tpr <- round(sum(cm_c34_tables[1, 61:70]) /
  (sum(cm_c34_tables[1, 61:70]) + sum(cm_c34_tables[2, 61:70])), 6)
k15w1c34_tnr <- round(sum(cm_c34_tables[4, 61:70]) /
  (sum(cm_c34_tables[4, 61:70]) + sum(cm_c34_tables[3, 61:70])), 6)
k15w1c34_truescore <- round((2 * k15w1c34_tpr * k15w1c34_tnr) /
  (k15w1c34_tpr + k15w1c34_tnr), 6)

k15w2c34_tpr <- round(sum(cm_c34_tables[1, 71:80]) /
  (sum(cm_c34_tables[1, 71:80]) + sum(cm_c34_tables[2, 71:80])), 6)
k15w2c34_tnr <- round(sum(cm_c34_tables[4, 71:80]) /
  (sum(cm_c34_tables[4, 71:80]) + sum(cm_c34_tables[3, 71:80])), 6)
k15w2c34_truescore <- round((2 * k15w2c34_tpr * k15w2c34_tnr) /
  (k15w2c34_tpr + k15w2c34_tnr), 6)

k15w3c34_tpr <- round(sum(cm_c34_tables[1, 81:90]) /
  (sum(cm_c34_tables[1, 81:90]) + sum(cm_c34_tables[2, 81:90])), 6)
k15w3c34_tnr <- round(sum(cm_c34_tables[4, 81:90]) /
  (sum(cm_c34_tables[4, 81:90]) + sum(cm_c34_tables[3, 81:90])), 6)
k15w3c34_truescore <- round((2 * k15w3c34_tpr * k15w3c34_tnr) /
  (k15w3c34_tpr + k15w3c34_tnr), 6)

k17w1c34_tpr <- round(sum(cm_c34_tables[1, 91:100]) /
  (sum(cm_c34_tables[1, 91:100]) + sum(cm_c34_tables[2, 91:100])), 6)
k17w1c34_tnr <- round(sum(cm_c34_tables[4, 91:100]) /
  (sum(cm_c34_tables[4, 91:100]) + sum(cm_c34_tables[3, 91:100])), 6)
k17w1c34_truescore <- round((2 * k17w1c34_tpr * k17w1c34_tnr) /
  (k17w1c34_tpr + k17w1c34_tnr), 6)

k17w2c34_tpr <- round(sum(cm_c34_tables[1, 101:110]) /
  (sum(cm_c34_tables[1, 101:110]) + sum(cm_c34_tables[2, 101:110])), 6)
k17w2c34_tnr <- round(sum(cm_c34_tables[4, 101:110]) /
  (sum(cm_c34_tables[4, 101:110]) + sum(cm_c34_tables[3, 101:110])), 6)
k17w2c34_truescore <- round((2 * k17w2c34_tpr * k17w2c34_tnr) /
  (k17w2c34_tpr + k17w2c34_tnr), 6)

k17w3c34_tpr <- round(sum(cm_c34_tables[1, 111:120]) /
  (sum(cm_c34_tables[1, 111:120]) + sum(cm_c34_tables[2, 111:120])), 6)
k17w3c34_tnr <- round(sum(cm_c34_tables[4, 111:120]) /
  (sum(cm_c34_tables[4, 111:120]) + sum(cm_c34_tables[3, 111:120])), 6)
k17w3c34_truescore <- round((2 * k17w3c34_tpr * k17w3c34_tnr) /
  (k17w3c34_tpr + k17w3c34_tnr), 6)

k19w1c34_tpr <- round(sum(cm_c34_tables[1, 121:130]) /
  (sum(cm_c34_tables[1, 121:130]) + sum(cm_c34_tables[2, 121:130])), 6)
k19w1c34_tnr <- round(sum(cm_c34_tables[4, 121:130]) /
  (sum(cm_c34_tables[4, 121:130]) + sum(cm_c34_tables[3, 121:130])), 6)
k19w1c34_truescore <- round((2 * k19w1c34_tpr * k19w1c34_tnr) /
  (k19w1c34_tpr + k19w1c34_tnr), 6)

```

```

k19w2c34_tpr <- round(sum(cm_c34_tables[1, 131:140]) /
  (sum(cm_c34_tables[1, 131:140]) + sum(cm_c34_tables[2, 131:140])), 6)
k19w2c34_tnr <- round(sum(cm_c34_tables[4, 131:140]) /
  (sum(cm_c34_tables[4, 131:140]) + sum(cm_c34_tables[3, 131:140])), 6)
k19w2c34_truescore <- round((2 * k19w2c34_tpr * k19w2c34_tnr) /
  (k19w2c34_tpr + k19w2c34_tnr), 6)

k19w3c34_tpr <- round(sum(cm_c34_tables[1, 141:150]) /
  (sum(cm_c34_tables[1, 141:150]) + sum(cm_c34_tables[2, 141:150])), 6)
k19w3c34_tnr <- round(sum(cm_c34_tables[4, 141:150]) /
  (sum(cm_c34_tables[4, 141:150]) + sum(cm_c34_tables[3, 141:150])), 6)
k19w3c34_truescore <- round((2 * k19w3c34_tpr * k19w3c34_tnr) /
  (k19w3c34_tpr + k19w3c34_tnr), 6)

k21w1c34_tpr <- round(sum(cm_c34_tables[1, 151:160]) /
  (sum(cm_c34_tables[1, 151:160]) + sum(cm_c34_tables[2, 151:160])), 6)
k21w1c34_tnr <- round(sum(cm_c34_tables[4, 151:160]) /
  (sum(cm_c34_tables[4, 151:160]) + sum(cm_c34_tables[3, 151:160])), 6)
k21w1c34_truescore <- round((2 * k21w1c34_tpr * k21w1c34_tnr) /
  (k21w1c34_tpr + k21w1c34_tnr), 6)

k21w2c34_tpr <- round(sum(cm_c34_tables[1, 161:170]) /
  (sum(cm_c34_tables[1, 161:170]) + sum(cm_c34_tables[2, 161:170])), 6)
k21w2c34_tnr <- round(sum(cm_c34_tables[4, 161:170]) /
  (sum(cm_c34_tables[4, 161:170]) + sum(cm_c34_tables[3, 161:170])), 6)
k21w2c34_truescore <- round((2 * k21w2c34_tpr * k21w2c34_tnr) /
  (k21w2c34_tpr + k21w2c34_tnr), 6)

k21w3c34_tpr <- round(sum(cm_c34_tables[1, 171:180]) /
  (sum(cm_c34_tables[1, 171:180]) + sum(cm_c34_tables[2, 171:180])), 6)
k21w3c34_tnr <- round(sum(cm_c34_tables[4, 171:180]) /
  (sum(cm_c34_tables[4, 171:180]) + sum(cm_c34_tables[3, 171:180])), 6)
k21w3c34_truescore <- round((2 * k21w3c34_tpr * k21w3c34_tnr) /
  (k21w3c34_tpr + k21w3c34_tnr), 6)

k23w1c34_tpr <- round(sum(cm_c34_tables[1, 181:190]) /
  (sum(cm_c34_tables[1, 181:190]) + sum(cm_c34_tables[2, 181:190])), 6)
k23w1c34_tnr <- round(sum(cm_c34_tables[4, 181:190]) /
  (sum(cm_c34_tables[4, 181:190]) + sum(cm_c34_tables[3, 181:190])), 6)
k23w1c34_truescore <- round((2 * k23w1c34_tpr * k23w1c34_tnr) /
  (k23w1c34_tpr + k23w1c34_tnr), 6)

k23w2c34_tpr <- round(sum(cm_c34_tables[1, 191:200]) /
  (sum(cm_c34_tables[1, 191:200]) + sum(cm_c34_tables[2, 191:200])), 6)
k23w2c34_tnr <- round(sum(cm_c34_tables[4, 191:200]) /
  (sum(cm_c34_tables[4, 191:200]) + sum(cm_c34_tables[3, 191:200])), 6)
k23w2c34_truescore <- round((2 * k23w2c34_tpr * k23w2c34_tnr) /
  (k23w2c34_tpr + k23w2c34_tnr), 6)

k23w3c34_tpr <- round(sum(cm_c34_tables[1, 201:210]) /
  (sum(cm_c34_tables[1, 201:210]) + sum(cm_c34_tables[2, 201:210])), 6)

```

```

k23w3c34_tnr <- round(sum(cm_c34_tables[4, 201:210]) /
  (sum(cm_c34_tables[4, 201:210]) + sum(cm_c34_tables[3, 201:210])), 6)
k23w3c34_truescore <- round((2 * k23w3c34_tpr * k23w3c34_tnr) /
  (k23w3c34_tpr + k23w3c34_tnr), 6)

k25w1c34_tpr <- round(sum(cm_c34_tables[1, 211:220]) /
  (sum(cm_c34_tables[1, 211:220]) + sum(cm_c34_tables[2, 211:220])), 6)
k25w1c34_tnr <- round(sum(cm_c34_tables[4, 211:220]) /
  (sum(cm_c34_tables[4, 211:220]) + sum(cm_c34_tables[3, 211:220])), 6)
k25w1c34_truescore <- round((2 * k25w1c34_tpr * k25w1c34_tnr) /
  (k25w1c34_tpr + k25w1c34_tnr), 6)

k25w2c34_tpr <- round(sum(cm_c34_tables[1, 221:230]) /
  (sum(cm_c34_tables[1, 221:230]) + sum(cm_c34_tables[2, 221:230])), 6)
k25w2c34_tnr <- round(sum(cm_c34_tables[4, 221:230]) /
  (sum(cm_c34_tables[4, 221:230]) + sum(cm_c34_tables[3, 221:230])), 6)
k25w2c34_truescore <- round((2 * k25w2c34_tpr * k25w2c34_tnr) /
  (k25w2c34_tpr + k25w2c34_tnr), 6)

k25w3c34_tpr <- round(sum(cm_c34_tables[1, 231:240]) /
  (sum(cm_c34_tables[1, 231:240]) + sum(cm_c34_tables[2, 231:240])), 6)
k25w3c34_tnr <- round(sum(cm_c34_tables[4, 231:240]) /
  (sum(cm_c34_tables[4, 231:240]) + sum(cm_c34_tables[3, 231:240])), 6)
k25w3c34_truescore <- round((2 * k25w3c34_tpr * k25w3c34_tnr) /
  (k25w3c34_tpr + k25w3c34_tnr), 6)

k27w1c34_tpr <- round(sum(cm_c34_tables[1, 241:250]) /
  (sum(cm_c34_tables[1, 241:250]) + sum(cm_c34_tables[2, 241:250])), 6)
k27w1c34_tnr <- round(sum(cm_c34_tables[4, 241:250]) /
  (sum(cm_c34_tables[4, 241:250]) + sum(cm_c34_tables[3, 241:250])), 6)
k27w1c34_truescore <- round((2 * k27w1c34_tpr * k27w1c34_tnr) /
  (k27w1c34_tpr + k27w1c34_tnr), 6)

k27w2c34_tpr <- round(sum(cm_c34_tables[1, 251:260]) /
  (sum(cm_c34_tables[1, 251:260]) + sum(cm_c34_tables[2, 251:260])), 6)
k27w2c34_tnr <- round(sum(cm_c34_tables[4, 251:260]) /
  (sum(cm_c34_tables[4, 251:260]) + sum(cm_c34_tables[3, 251:260])), 6)
k27w2c34_truescore <- round((2 * k27w2c34_tpr * k27w2c34_tnr) /
  (k27w2c34_tpr + k27w2c34_tnr), 6)

k27w3c34_tpr <- round(sum(cm_c34_tables[1, 261:270]) /
  (sum(cm_c34_tables[1, 261:270]) + sum(cm_c34_tables[2, 261:270])), 6)
k27w3c34_tnr <- round(sum(cm_c34_tables[4, 261:270]) /
  (sum(cm_c34_tables[4, 261:270]) + sum(cm_c34_tables[3, 261:270])), 6)
k27w3c34_truescore <- round((2 * k27w3c34_tpr * k27w3c34_tnr) /
  (k27w3c34_tpr + k27w3c34_tnr), 6)

k29w1c34_tpr <- round(sum(cm_c34_tables[1, 271:280]) /
  (sum(cm_c34_tables[1, 271:280]) + sum(cm_c34_tables[2, 271:280])), 6)
k29w1c34_tnr <- round(sum(cm_c34_tables[4, 271:280]) /

```

```

(sum(cm_c34_tables[4, 271:280]) + sum(cm_c34_tables[3, 271:280])), 6)
k29w1c34_truescore <- round((2 * k29w1c34_tpr * k29w1c34_tnr) /
(k29w1c34_tpr + k29w1c34_tnr), 6)

k29w2c34_tpr <- round(sum(cm_c34_tables[1, 281:290]) /
(sum(cm_c34_tables[1, 281:290]) + sum(cm_c34_tables[2, 281:290])), 6)
k29w2c34_tnr <- round(sum(cm_c34_tables[4, 281:290]) /
(sum(cm_c34_tables[4, 281:290]) + sum(cm_c34_tables[3, 281:290])), 6)
k29w2c34_truescore <- round((2 * k29w2c34_tpr * k29w2c34_tnr) /
(k29w2c34_tpr + k29w2c34_tnr), 6)

k29w3c34_tpr <- round(sum(cm_c34_tables[1, 291:300]) /
(sum(cm_c34_tables[1, 291:300]) + sum(cm_c34_tables[2, 291:300])), 6)
k29w3c34_tnr <- round(sum(cm_c34_tables[4, 291:300]) /
(sum(cm_c34_tables[4, 291:300]) + sum(cm_c34_tables[3, 291:300])), 6)
k29w3c34_truescore <- round((2 * k29w3c34_tpr * k29w3c34_tnr) /
(k29w3c34_tpr + k29w3c34_tnr), 6)

k31w1c34_tpr <- round(sum(cm_c34_tables[1, 301:310]) /
(sum(cm_c34_tables[1, 301:310]) + sum(cm_c34_tables[2, 301:310])), 6)
k31w1c34_tnr <- round(sum(cm_c34_tables[4, 301:310]) /
(sum(cm_c34_tables[4, 301:310]) + sum(cm_c34_tables[3, 301:310])), 6)
k31w1c34_truescore <- round((2 * k31w1c34_tpr * k31w1c34_tnr) /
(k31w1c34_tpr + k31w1c34_tnr), 6)

k31w2c34_tpr <- round(sum(cm_c34_tables[1, 311:320]) /
(sum(cm_c34_tables[1, 311:320]) + sum(cm_c34_tables[2, 311:320])), 6)
k31w2c34_tnr <- round(sum(cm_c34_tables[4, 311:320]) /
(sum(cm_c34_tables[4, 311:320]) + sum(cm_c34_tables[3, 311:320])), 6)
k31w2c34_truescore <- round((2 * k31w2c34_tpr * k31w2c34_tnr) /
(k31w2c34_tpr + k31w2c34_tnr), 6)

k31w3c34_tpr <- round(sum(cm_c34_tables[1, 321:330]) /
(sum(cm_c34_tables[1, 321:330]) + sum(cm_c34_tables[2, 321:330])), 6)
k31w3c34_tnr <- round(sum(cm_c34_tables[4, 321:330]) /
(sum(cm_c34_tables[4, 321:330]) + sum(cm_c34_tables[3, 321:330])), 6)
k31w3c34_truescore <- round((2 * k31w3c34_tpr * k31w3c34_tnr) /
(k31w3c34_tpr + k31w3c34_tnr), 6)

k33w1c34_tpr <- round(sum(cm_c34_tables[1, 331:340]) /
(sum(cm_c34_tables[1, 331:340]) + sum(cm_c34_tables[2, 331:340])), 6)
k33w1c34_tnr <- round(sum(cm_c34_tables[4, 331:340]) /
(sum(cm_c34_tables[4, 331:340]) + sum(cm_c34_tables[3, 331:340])), 6)
k33w1c34_truescore <- round((2 * k33w1c34_tpr * k33w1c34_tnr) /
(k33w1c34_tpr + k33w1c34_tnr), 6)

k33w2c34_tpr <- round(sum(cm_c34_tables[1, 341:350]) /
(sum(cm_c34_tables[1, 341:350]) + sum(cm_c34_tables[2, 341:350])), 6)
k33w2c34_tnr <- round(sum(cm_c34_tables[4, 341:350]) /
(sum(cm_c34_tables[4, 341:350]) + sum(cm_c34_tables[3, 341:350])), 6)
k33w2c34_truescore <- round((2 * k33w2c34_tpr * k33w2c34_tnr) /

```

```

(k33w2c34_tpr + k33w2c34_tnr), 6)

k33w3c34_tpr <- round(sum(cm_c34_tables[1, 351:360]) /
  (sum(cm_c34_tables[1, 351:360]) + sum(cm_c34_tables[2, 351:360])), 6)
k33w3c34_tnr <- round(sum(cm_c34_tables[4, 351:360]) /
  (sum(cm_c34_tables[4, 351:360]) + sum(cm_c34_tables[3, 351:360])), 6)
k33w3c34_truescore <- round((2 * k33w3c34_tpr * k33w3c34_tnr) /
  (k33w3c34_tpr + k33w3c34_tnr), 6)

k35w1c34_tpr <- round(sum(cm_c34_tables[1, 361:370]) /
  (sum(cm_c34_tables[1, 361:370]) + sum(cm_c34_tables[2, 361:370])), 6)
k35w1c34_tnr <- round(sum(cm_c34_tables[4, 361:370]) /
  (sum(cm_c34_tables[4, 361:370]) + sum(cm_c34_tables[3, 361:370])), 6)
k35w1c34_truescore <- round((2 * k35w1c34_tpr * k35w1c34_tnr) /
  (k35w1c34_tpr + k35w1c34_tnr), 6)

k35w2c34_tpr <- round(sum(cm_c34_tables[1, 371:380]) /
  (sum(cm_c34_tables[1, 371:380]) + sum(cm_c34_tables[2, 371:380])), 6)
k35w2c34_tnr <- round(sum(cm_c34_tables[4, 371:380]) /
  (sum(cm_c34_tables[4, 371:380]) + sum(cm_c34_tables[3, 371:380])), 6)
k35w2c34_truescore <- round((2 * k35w2c34_tpr * k35w2c34_tnr) /
  (k35w2c34_tpr + k35w2c34_tnr), 6)

k35w3c34_tpr <- round(sum(cm_c34_tables[1, 381:390]) /
  (sum(cm_c34_tables[1, 381:390]) + sum(cm_c34_tables[2, 381:390])), 6)
k35w3c34_tnr <- round(sum(cm_c34_tables[4, 381:390]) /
  (sum(cm_c34_tables[4, 381:390]) + sum(cm_c34_tables[3, 381:390])), 6)
k35w3c34_truescore <- round((2 * k35w3c34_tpr * k35w3c34_tnr) /
  (k35w3c34_tpr + k35w3c34_tnr), 6)

k37w1c34_tpr <- round(sum(cm_c34_tables[1, 391:400]) /
  (sum(cm_c34_tables[1, 391:400]) + sum(cm_c34_tables[2, 391:400])), 6)
k37w1c34_tnr <- round(sum(cm_c34_tables[4, 391:400]) /
  (sum(cm_c34_tables[4, 391:400]) + sum(cm_c34_tables[3, 391:400])), 6)
k37w1c34_truescore <- round((2 * k37w1c34_tpr * k37w1c34_tnr) /
  (k37w1c34_tpr + k37w1c34_tnr), 6)

k37w2c34_tpr <- round(sum(cm_c34_tables[1, 401:410]) /
  (sum(cm_c34_tables[1, 401:410]) + sum(cm_c34_tables[2, 401:410])), 6)
k37w2c34_tnr <- round(sum(cm_c34_tables[4, 401:410]) /
  (sum(cm_c34_tables[4, 401:410]) + sum(cm_c34_tables[3, 401:410])), 6)
k37w2c34_truescore <- round((2 * k37w2c34_tpr * k37w2c34_tnr) /
  (k37w2c34_tpr + k37w2c34_tnr), 6)

k37w3c34_tpr <- round(sum(cm_c34_tables[1, 411:420]) /
  (sum(cm_c34_tables[1, 411:420]) + sum(cm_c34_tables[2, 411:420])), 6)
k37w3c34_tnr <- round(sum(cm_c34_tables[4, 411:420]) /
  (sum(cm_c34_tables[4, 411:420]) + sum(cm_c34_tables[3, 411:420])), 6)
k37w3c34_truescore <- round((2 * k37w3c34_tpr * k37w3c34_tnr) /
  (k37w3c34_tpr + k37w3c34_tnr), 6)

```

```

k39w1c34_tpr <- round(sum(cm_c34_tables[1, 421:430]) /
  (sum(cm_c34_tables[1, 421:430]) + sum(cm_c34_tables[2, 421:430])), 6)
k39w1c34_tnr <- round(sum(cm_c34_tables[4, 421:430]) /
  (sum(cm_c34_tables[4, 421:430]) + sum(cm_c34_tables[3, 421:430])), 6)
k39w1c34_truescore <- round((2 * k39w1c34_tpr * k39w1c34_tnr) /
  (k39w1c34_tpr + k39w1c34_tnr), 6)

k39w2c34_tpr <- round(sum(cm_c34_tables[1, 431:440]) /
  (sum(cm_c34_tables[1, 431:440]) + sum(cm_c34_tables[2, 431:440])), 6)
k39w2c34_tnr <- round(sum(cm_c34_tables[4, 431:440]) /
  (sum(cm_c34_tables[4, 431:440]) + sum(cm_c34_tables[3, 431:440])), 6)
k39w2c34_truescore <- round((2 * k39w2c34_tpr * k39w2c34_tnr) /
  (k39w2c34_tpr + k39w2c34_tnr), 6)

k39w3c34_tpr <- round(sum(cm_c34_tables[1, 441:450]) /
  (sum(cm_c34_tables[1, 441:450]) + sum(cm_c34_tables[2, 441:450])), 6)
k39w3c34_tnr <- round(sum(cm_c34_tables[4, 441:450]) /
  (sum(cm_c34_tables[4, 441:450]) + sum(cm_c34_tables[3, 441:450])), 6)
k39w3c34_truescore <- round((2 * k39w3c34_tpr * k39w3c34_tnr) /
  (k39w3c34_tpr + k39w3c34_tnr), 6)

# Compile the 0.34 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c34_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.34,
  TPR = c(k11w1c34_tpr, k11w2c34_tpr, k11w3c34_tpr, k13w1c34_tpr,
    k13w2c34_tpr, k13w3c34_tpr, k15w1c34_tpr, k15w2c34_tpr,
    k15w3c34_tpr, k17w1c34_tpr, k17w2c34_tpr, k17w3c34_tpr,
    k19w1c34_tpr, k19w2c34_tpr, k19w3c34_tpr, k21w1c34_tpr,
    k21w2c34_tpr, k21w3c34_tpr, k23w1c34_tpr, k23w2c34_tpr,
    k23w3c34_tpr, k25w1c34_tpr, k25w2c34_tpr, k25w3c34_tpr,
    k27w1c34_tpr, k27w2c34_tpr, k27w3c34_tpr, k29w1c34_tpr,
    k29w2c34_tpr, k29w3c34_tpr, k31w1c34_tpr, k31w2c34_tpr,
    k31w3c34_tpr, k33w1c34_tpr, k33w2c34_tpr, k33w3c34_tpr,
    k35w1c34_tpr, k35w2c34_tpr, k35w3c34_tpr, k37w1c34_tpr,
    k37w2c34_tpr, k37w3c34_tpr, k39w1c34_tpr, k39w2c34_tpr,
    k39w3c34_tpr),
  TNR = c(k11w1c34_tnr, k11w2c34_tnr, k11w3c34_tnr, k13w1c34_tnr,
    k13w2c34_tnr, k13w3c34_tnr, k15w1c34_tnr, k15w2c34_tnr,
    k15w3c34_tnr, k17w1c34_tnr, k17w2c34_tnr, k17w3c34_tnr,
    k19w1c34_tnr, k19w2c34_tnr, k19w3c34_tnr, k21w1c34_tnr,
    k21w2c34_tnr, k21w3c34_tnr, k23w1c34_tnr, k23w2c34_tnr,
    k23w3c34_tnr, k25w1c34_tnr, k25w2c34_tnr, k25w3c34_tnr,
    k27w1c34_tnr, k27w2c34_tnr, k27w3c34_tnr, k29w1c34_tnr,
    k29w2c34_tnr, k29w3c34_tnr, k31w1c34_tnr, k31w2c34_tnr,
    k31w3c34_tnr, k33w1c34_tnr, k33w2c34_tnr, k33w3c34_tnr),
  Truescore = k39w1c34_truescore)

```

```

k35w1c34_tnr, k35w2c34_tnr, k35w3c34_tnr, k37w1c34_tnr,
k37w2c34_tnr, k37w3c34_tnr, k39w1c34_tnr, k39w2c34_tnr,
k39w3c34_tnr),
Truescore = c(k11w1c34_truescore, k11w2c34_truescore,
k11w3c34_truescore, k13w1c34_truescore,
k13w2c34_truescore, k13w3c34_truescore,
k15w1c34_truescore, k15w2c34_truescore,
k15w3c34_truescore, k17w1c34_truescore,
k17w2c34_truescore, k17w3c34_truescore,
k19w1c34_truescore, k19w2c34_truescore,
k19w3c34_truescore, k21w1c34_truescore,
k21w2c34_truescore, k21w3c34_truescore,
k23w1c34_truescore, k23w2c34_truescore,
k23w3c34_truescore, k25w1c34_truescore,
k25w2c34_truescore, k25w3c34_truescore,
k27w1c34_truescore, k27w2c34_truescore,
k27w3c34_truescore, k29w1c34_truescore,
k29w2c34_truescore, k29w3c34_truescore,
k31w1c34_truescore, k31w2c34_truescore,
k31w3c34_truescore, k33w1c34_truescore,
k33w2c34_truescore, k33w3c34_truescore,
k35w1c34_truescore, k35w2c34_truescore,
k35w3c34_truescore, k37w1c34_truescore,
k37w2c34_truescore, k37w3c34_truescore,
k39w1c34_truescore, k39w2c34_truescore,
k39w3c34_truescore))

```

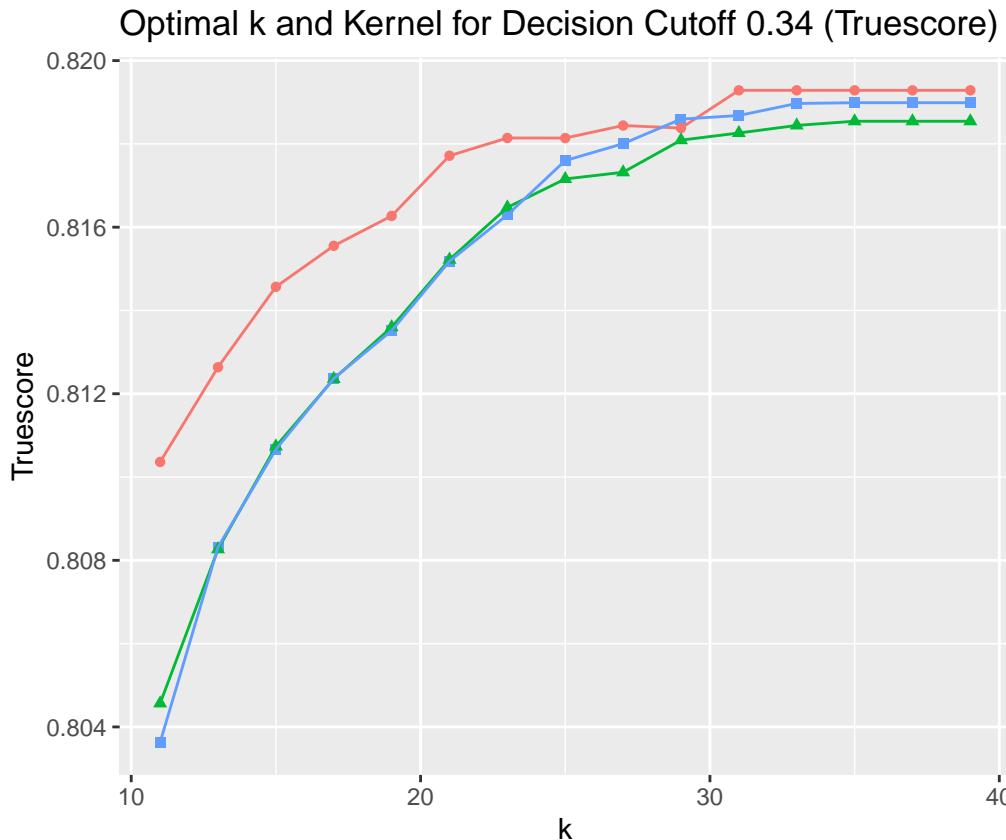
`knitr::kable(c34_results[1:45,], caption = "c34_results")`

Table 43: c34_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.34	0.777008	0.832144	0.803631
11	gaussian	0.34	0.790110	0.831682	0.810363
11	optimal	0.34	0.777962	0.833053	0.804566
13	triangular	0.34	0.785492	0.832508	0.808317
13	gaussian	0.34	0.794779	0.831318	0.812638
13	optimal	0.34	0.784839	0.833135	0.808266
15	triangular	0.34	0.790010	0.832425	0.810663
15	gaussian	0.34	0.798494	0.831302	0.814568
15	optimal	0.34	0.789910	0.832689	0.810736
17	triangular	0.34	0.793223	0.832458	0.812367
17	gaussian	0.34	0.801155	0.830476	0.815552
17	optimal	0.34	0.793323	0.832326	0.812357
19	triangular	0.34	0.796034	0.831781	0.813515
19	gaussian	0.34	0.803363	0.829601	0.816271
19	optimal	0.34	0.796135	0.831847	0.813599
21	triangular	0.34	0.799448	0.831550	0.815183
21	gaussian	0.34	0.806074	0.829700	0.817716
21	optimal	0.34	0.799598	0.831467	0.815221
23	triangular	0.34	0.801958	0.831153	0.816295
23	gaussian	0.34	0.806777	0.829832	0.818142
23	optimal	0.34	0.802661	0.830773	0.816475

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.34	0.804618	0.831005	0.817599
25	gaussian	0.34	0.807430	0.829138	0.818140
25	optimal	0.34	0.804618	0.830096	0.817158
27	triangular	0.34	0.806074	0.830294	0.818005
27	gaussian	0.34	0.808283	0.828857	0.818441
27	optimal	0.34	0.805070	0.829947	0.817319
29	triangular	0.34	0.807781	0.829700	0.818594
29	gaussian	0.34	0.808133	0.828890	0.818380
29	optimal	0.34	0.806878	0.829617	0.818090
31	triangular	0.34	0.808534	0.829089	0.818682
31	gaussian	0.34	0.810392	0.828378	0.819286
31	optimal	0.34	0.807480	0.829336	0.818262
33	triangular	0.34	0.809287	0.828890	0.818971
33	gaussian	0.34	0.810392	0.828378	0.819286
33	optimal	0.34	0.808384	0.828758	0.818444
35	triangular	0.34	0.809588	0.828610	0.818989
35	gaussian	0.34	0.810392	0.828378	0.819286
35	optimal	0.34	0.808735	0.828593	0.818544
37	triangular	0.34	0.809588	0.828610	0.818989
37	gaussian	0.34	0.810392	0.828378	0.819286
37	optimal	0.34	0.808735	0.828593	0.818544
39	triangular	0.34	0.809588	0.828610	0.818989
39	gaussian	0.34	0.810392	0.828378	0.819286
39	optimal	0.34	0.808735	0.828593	0.818544

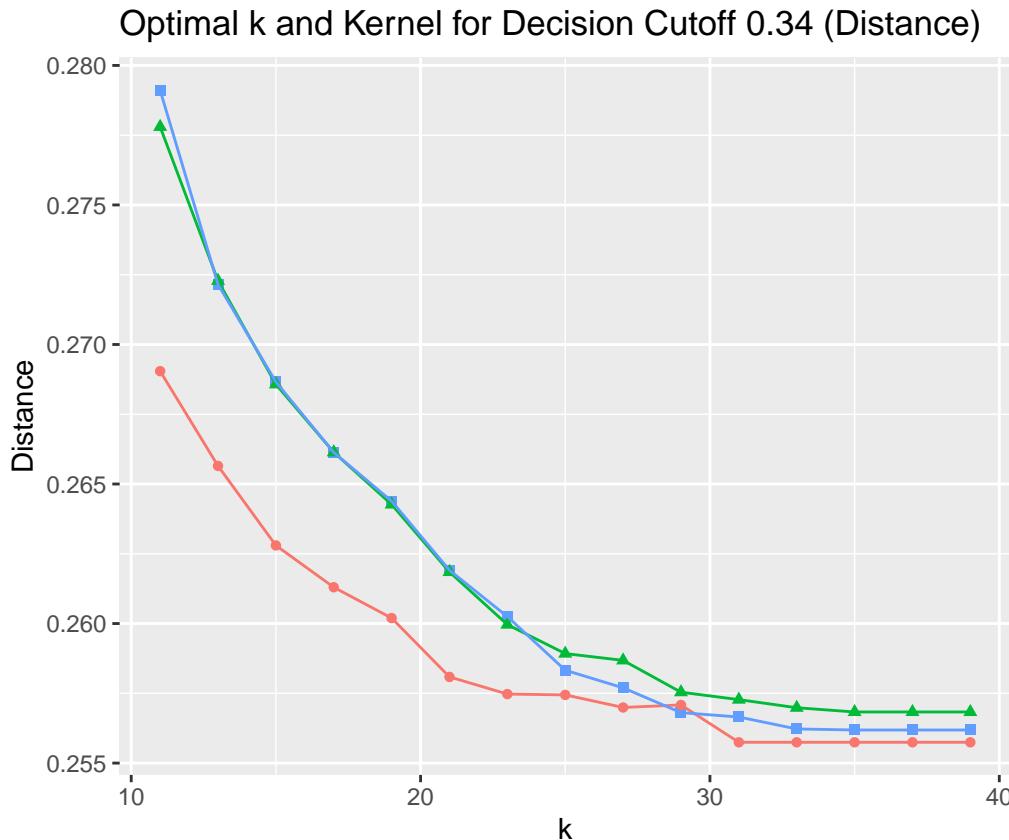
```
ggplot(c34_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.34 (Truescore)")
```



```
# For the Cutoff of 0.34, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c34_results <- c34_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c34_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.34 (Distance)")
```



```
# For the Cutoff of 0.34, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c34_results$Truescore)
```

```
## [1] 0.819286
```

```
(c34_opt_k_ts <- c34_results$k[which.max(c34_results$Truescore)])
```

```
## [1] 31
```

```
(c34_opt_kernel_ts <- c34_results$Kernel[which.max(c34_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c34_opt_cut_ts <- c34_results$Cut[which.max(c34_results$Truescore)])
```

```
## [1] 0.34
```

```
(c34_opt_tpr_ts <- c34_results$TPR[which.max(c34_results$Truescore)])
```

```
## [1] 0.810392
```

```

(c34_opt_tnr_ts <- c34_results$TNR[which.max(c34_results$Truescore)])

## [1] 0.828378

(c34_opt_d_ts <- c34_results$Distance[which.max(c34_results$Truescore)])

## [1] 0.2557446

min(c34_results$Distance)

## [1] 0.2557446

(c34_opt_k_dist <- c34_results$k[which.min(c34_results$Distance)])

## [1] 31

(c34_opt_kernel_dist <- c34_results$Kernel[which.min(c34_results$Distance)])

## [1] "gaussian"

(c34_opt_cut_dist <- c34_results$Cut[which.min(c34_results$Distance)])

## [1] 0.34

(c34_opt_tpr_dist <- c34_results$TPR[which.min(c34_results$Distance)])

## [1] 0.810392

(c34_opt_tnr_dist <- c34_results$TNR[which.min(c34_results$Distance)])

## [1] 0.828378

(c34_opt_t_dist <- c34_results$Truescore[which.min(c34_results$Distance)])

## [1] 0.819286

#####
# 0.35 Cutoff
#####

# For the decision cutoff of 0.35, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c35 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred35, obs))
  confusionMatrix(ss$pred35, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c35) <- kwf_dfs_v

cm_c35_tables <- sapply(cm_c35, "[[", 2)
cm_c35_tables <- as_tibble(cm_c35_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.35.

k11w1c35_tpr <- round(sum(cm_c35_tables[1, 1:10]) /
  (sum(cm_c35_tables[1, 1:10]) + sum(cm_c35_tables[2, 1:10])), 6)
k11w1c35_tnr <- round(sum(cm_c35_tables[4, 1:10]) /
  (sum(cm_c35_tables[4, 1:10]) + sum(cm_c35_tables[3, 1:10])), 6)
k11w1c35_truescore <- round((2 * k11w1c35_tpr * k11w1c35_tnr) /
  (k11w1c35_tpr + k11w1c35_tnr), 6)

k11w2c35_tpr <- round(sum(cm_c35_tables[1, 11:20]) /
  (sum(cm_c35_tables[1, 11:20]) + sum(cm_c35_tables[2, 11:20])), 6)
k11w2c35_tnr <- round(sum(cm_c35_tables[4, 11:20]) /
  (sum(cm_c35_tables[4, 11:20]) + sum(cm_c35_tables[3, 11:20])), 6)
k11w2c35_truescore <- round((2 * k11w2c35_tpr * k11w2c35_tnr) /
  (k11w2c35_tpr + k11w2c35_tnr), 6)

k11w3c35_tpr <- round(sum(cm_c35_tables[1, 21:30]) /
  (sum(cm_c35_tables[1, 21:30]) + sum(cm_c35_tables[2, 21:30])), 6)
k11w3c35_tnr <- round(sum(cm_c35_tables[4, 21:30]) /
  (sum(cm_c35_tables[4, 21:30]) + sum(cm_c35_tables[3, 21:30])), 6)
k11w3c35_truescore <- round((2 * k11w3c35_tpr * k11w3c35_tnr) /
  (k11w3c35_tpr + k11w3c35_tnr), 6)

k13w1c35_tpr <- round(sum(cm_c35_tables[1, 31:40]) /
  (sum(cm_c35_tables[1, 31:40]) + sum(cm_c35_tables[2, 31:40])), 6)
k13w1c35_tnr <- round(sum(cm_c35_tables[4, 31:40]) /
  (sum(cm_c35_tables[4, 31:40]) + sum(cm_c35_tables[3, 31:40])), 6)
k13w1c35_truescore <- round((2 * k13w1c35_tpr * k13w1c35_tnr) /
  (k13w1c35_tpr + k13w1c35_tnr), 6)

k13w2c35_tpr <- round(sum(cm_c35_tables[1, 41:50]) /
  (sum(cm_c35_tables[1, 41:50]) + sum(cm_c35_tables[2, 41:50])), 6)
k13w2c35_tnr <- round(sum(cm_c35_tables[4, 41:50]) /
  (sum(cm_c35_tables[4, 41:50]) + sum(cm_c35_tables[3, 41:50])), 6)
k13w2c35_truescore <- round((2 * k13w2c35_tpr * k13w2c35_tnr) /
  (k13w2c35_tpr + k13w2c35_tnr), 6)

k13w3c35_tpr <- round(sum(cm_c35_tables[1, 51:60]) /
  (sum(cm_c35_tables[1, 51:60]) + sum(cm_c35_tables[2, 51:60])), 6)
k13w3c35_tnr <- round(sum(cm_c35_tables[4, 51:60]) /
  (sum(cm_c35_tables[4, 51:60]) + sum(cm_c35_tables[3, 51:60])), 6)
k13w3c35_truescore <- round((2 * k13w3c35_tpr * k13w3c35_tnr) /

```

```

(k13w3c35_tpr + k13w3c35_tnr), 6)

k15w1c35_tpr <- round(sum(cm_c35_tables[1, 61:70]) /
  (sum(cm_c35_tables[1, 61:70]) + sum(cm_c35_tables[2, 61:70])), 6)
k15w1c35_tnr <- round(sum(cm_c35_tables[4, 61:70]) /
  (sum(cm_c35_tables[4, 61:70]) + sum(cm_c35_tables[3, 61:70])), 6)
k15w1c35_truescore <- round((2 * k15w1c35_tpr * k15w1c35_tnr) /
  (k15w1c35_tpr + k15w1c35_tnr), 6)

k15w2c35_tpr <- round(sum(cm_c35_tables[1, 71:80]) /
  (sum(cm_c35_tables[1, 71:80]) + sum(cm_c35_tables[2, 71:80])), 6)
k15w2c35_tnr <- round(sum(cm_c35_tables[4, 71:80]) /
  (sum(cm_c35_tables[4, 71:80]) + sum(cm_c35_tables[3, 71:80])), 6)
k15w2c35_truescore <- round((2 * k15w2c35_tpr * k15w2c35_tnr) /
  (k15w2c35_tpr + k15w2c35_tnr), 6)

k15w3c35_tpr <- round(sum(cm_c35_tables[1, 81:90]) /
  (sum(cm_c35_tables[1, 81:90]) + sum(cm_c35_tables[2, 81:90])), 6)
k15w3c35_tnr <- round(sum(cm_c35_tables[4, 81:90]) /
  (sum(cm_c35_tables[4, 81:90]) + sum(cm_c35_tables[3, 81:90])), 6)
k15w3c35_truescore <- round((2 * k15w3c35_tpr * k15w3c35_tnr) /
  (k15w3c35_tpr + k15w3c35_tnr), 6)

k17w1c35_tpr <- round(sum(cm_c35_tables[1, 91:100]) /
  (sum(cm_c35_tables[1, 91:100]) + sum(cm_c35_tables[2, 91:100])), 6)
k17w1c35_tnr <- round(sum(cm_c35_tables[4, 91:100]) /
  (sum(cm_c35_tables[4, 91:100]) + sum(cm_c35_tables[3, 91:100])), 6)
k17w1c35_truescore <- round((2 * k17w1c35_tpr * k17w1c35_tnr) /
  (k17w1c35_tpr + k17w1c35_tnr), 6)

k17w2c35_tpr <- round(sum(cm_c35_tables[1, 101:110]) /
  (sum(cm_c35_tables[1, 101:110]) + sum(cm_c35_tables[2, 101:110])), 6)
k17w2c35_tnr <- round(sum(cm_c35_tables[4, 101:110]) /
  (sum(cm_c35_tables[4, 101:110]) + sum(cm_c35_tables[3, 101:110])), 6)
k17w2c35_truescore <- round((2 * k17w2c35_tpr * k17w2c35_tnr) /
  (k17w2c35_tpr + k17w2c35_tnr), 6)

k17w3c35_tpr <- round(sum(cm_c35_tables[1, 111:120]) /
  (sum(cm_c35_tables[1, 111:120]) + sum(cm_c35_tables[2, 111:120])), 6)
k17w3c35_tnr <- round(sum(cm_c35_tables[4, 111:120]) /
  (sum(cm_c35_tables[4, 111:120]) + sum(cm_c35_tables[3, 111:120])), 6)
k17w3c35_truescore <- round((2 * k17w3c35_tpr * k17w3c35_tnr) /
  (k17w3c35_tpr + k17w3c35_tnr), 6)

k19w1c35_tpr <- round(sum(cm_c35_tables[1, 121:130]) /
  (sum(cm_c35_tables[1, 121:130]) + sum(cm_c35_tables[2, 121:130])), 6)
k19w1c35_tnr <- round(sum(cm_c35_tables[4, 121:130]) /
  (sum(cm_c35_tables[4, 121:130]) + sum(cm_c35_tables[3, 121:130])), 6)
k19w1c35_truescore <- round((2 * k19w1c35_tpr * k19w1c35_tnr) /
  (k19w1c35_tpr + k19w1c35_tnr), 6)

```

```

k19w2c35_tpr <- round(sum(cm_c35_tables[1, 131:140]) /
  (sum(cm_c35_tables[1, 131:140]) + sum(cm_c35_tables[2, 131:140])), 6)
k19w2c35_tnr <- round(sum(cm_c35_tables[4, 131:140]) /
  (sum(cm_c35_tables[4, 131:140]) + sum(cm_c35_tables[3, 131:140])), 6)
k19w2c35_truescore <- round((2 * k19w2c35_tpr * k19w2c35_tnr) /
  (k19w2c35_tpr + k19w2c35_tnr), 6)

k19w3c35_tpr <- round(sum(cm_c35_tables[1, 141:150]) /
  (sum(cm_c35_tables[1, 141:150]) + sum(cm_c35_tables[2, 141:150])), 6)
k19w3c35_tnr <- round(sum(cm_c35_tables[4, 141:150]) /
  (sum(cm_c35_tables[4, 141:150]) + sum(cm_c35_tables[3, 141:150])), 6)
k19w3c35_truescore <- round((2 * k19w3c35_tpr * k19w3c35_tnr) /
  (k19w3c35_tpr + k19w3c35_tnr), 6)

k21w1c35_tpr <- round(sum(cm_c35_tables[1, 151:160]) /
  (sum(cm_c35_tables[1, 151:160]) + sum(cm_c35_tables[2, 151:160])), 6)
k21w1c35_tnr <- round(sum(cm_c35_tables[4, 151:160]) /
  (sum(cm_c35_tables[4, 151:160]) + sum(cm_c35_tables[3, 151:160])), 6)
k21w1c35_truescore <- round((2 * k21w1c35_tpr * k21w1c35_tnr) /
  (k21w1c35_tpr + k21w1c35_tnr), 6)

k21w2c35_tpr <- round(sum(cm_c35_tables[1, 161:170]) /
  (sum(cm_c35_tables[1, 161:170]) + sum(cm_c35_tables[2, 161:170])), 6)
k21w2c35_tnr <- round(sum(cm_c35_tables[4, 161:170]) /
  (sum(cm_c35_tables[4, 161:170]) + sum(cm_c35_tables[3, 161:170])), 6)
k21w2c35_truescore <- round((2 * k21w2c35_tpr * k21w2c35_tnr) /
  (k21w2c35_tpr + k21w2c35_tnr), 6)

k21w3c35_tpr <- round(sum(cm_c35_tables[1, 171:180]) /
  (sum(cm_c35_tables[1, 171:180]) + sum(cm_c35_tables[2, 171:180])), 6)
k21w3c35_tnr <- round(sum(cm_c35_tables[4, 171:180]) /
  (sum(cm_c35_tables[4, 171:180]) + sum(cm_c35_tables[3, 171:180])), 6)
k21w3c35_truescore <- round((2 * k21w3c35_tpr * k21w3c35_tnr) /
  (k21w3c35_tpr + k21w3c35_tnr), 6)

k23w1c35_tpr <- round(sum(cm_c35_tables[1, 181:190]) /
  (sum(cm_c35_tables[1, 181:190]) + sum(cm_c35_tables[2, 181:190])), 6)
k23w1c35_tnr <- round(sum(cm_c35_tables[4, 181:190]) /
  (sum(cm_c35_tables[4, 181:190]) + sum(cm_c35_tables[3, 181:190])), 6)
k23w1c35_truescore <- round((2 * k23w1c35_tpr * k23w1c35_tnr) /
  (k23w1c35_tpr + k23w1c35_tnr), 6)

k23w2c35_tpr <- round(sum(cm_c35_tables[1, 191:200]) /
  (sum(cm_c35_tables[1, 191:200]) + sum(cm_c35_tables[2, 191:200])), 6)
k23w2c35_tnr <- round(sum(cm_c35_tables[4, 191:200]) /
  (sum(cm_c35_tables[4, 191:200]) + sum(cm_c35_tables[3, 191:200])), 6)
k23w2c35_truescore <- round((2 * k23w2c35_tpr * k23w2c35_tnr) /
  (k23w2c35_tpr + k23w2c35_tnr), 6)

k23w3c35_tpr <- round(sum(cm_c35_tables[1, 201:210]) /
  (sum(cm_c35_tables[1, 201:210]) + sum(cm_c35_tables[2, 201:210])), 6)

```

```

k23w3c35_tnr <- round(sum(cm_c35_tables[4, 201:210]) /
  (sum(cm_c35_tables[4, 201:210]) + sum(cm_c35_tables[3, 201:210])), 6)
k23w3c35_truescore <- round((2 * k23w3c35_tpr * k23w3c35_tnr) /
  (k23w3c35_tpr + k23w3c35_tnr), 6)

k25w1c35_tpr <- round(sum(cm_c35_tables[1, 211:220]) /
  (sum(cm_c35_tables[1, 211:220]) + sum(cm_c35_tables[2, 211:220])), 6)
k25w1c35_tnr <- round(sum(cm_c35_tables[4, 211:220]) /
  (sum(cm_c35_tables[4, 211:220]) + sum(cm_c35_tables[3, 211:220])), 6)
k25w1c35_truescore <- round((2 * k25w1c35_tpr * k25w1c35_tnr) /
  (k25w1c35_tpr + k25w1c35_tnr), 6)

k25w2c35_tpr <- round(sum(cm_c35_tables[1, 221:230]) /
  (sum(cm_c35_tables[1, 221:230]) + sum(cm_c35_tables[2, 221:230])), 6)
k25w2c35_tnr <- round(sum(cm_c35_tables[4, 221:230]) /
  (sum(cm_c35_tables[4, 221:230]) + sum(cm_c35_tables[3, 221:230])), 6)
k25w2c35_truescore <- round((2 * k25w2c35_tpr * k25w2c35_tnr) /
  (k25w2c35_tpr + k25w2c35_tnr), 6)

k25w3c35_tpr <- round(sum(cm_c35_tables[1, 231:240]) /
  (sum(cm_c35_tables[1, 231:240]) + sum(cm_c35_tables[2, 231:240])), 6)
k25w3c35_tnr <- round(sum(cm_c35_tables[4, 231:240]) /
  (sum(cm_c35_tables[4, 231:240]) + sum(cm_c35_tables[3, 231:240])), 6)
k25w3c35_truescore <- round((2 * k25w3c35_tpr * k25w3c35_tnr) /
  (k25w3c35_tpr + k25w3c35_tnr), 6)

k27w1c35_tpr <- round(sum(cm_c35_tables[1, 241:250]) /
  (sum(cm_c35_tables[1, 241:250]) + sum(cm_c35_tables[2, 241:250])), 6)
k27w1c35_tnr <- round(sum(cm_c35_tables[4, 241:250]) /
  (sum(cm_c35_tables[4, 241:250]) + sum(cm_c35_tables[3, 241:250])), 6)
k27w1c35_truescore <- round((2 * k27w1c35_tpr * k27w1c35_tnr) /
  (k27w1c35_tpr + k27w1c35_tnr), 6)

k27w2c35_tpr <- round(sum(cm_c35_tables[1, 251:260]) /
  (sum(cm_c35_tables[1, 251:260]) + sum(cm_c35_tables[2, 251:260])), 6)
k27w2c35_tnr <- round(sum(cm_c35_tables[4, 251:260]) /
  (sum(cm_c35_tables[4, 251:260]) + sum(cm_c35_tables[3, 251:260])), 6)
k27w2c35_truescore <- round((2 * k27w2c35_tpr * k27w2c35_tnr) /
  (k27w2c35_tpr + k27w2c35_tnr), 6)

k27w3c35_tpr <- round(sum(cm_c35_tables[1, 261:270]) /
  (sum(cm_c35_tables[1, 261:270]) + sum(cm_c35_tables[2, 261:270])), 6)
k27w3c35_tnr <- round(sum(cm_c35_tables[4, 261:270]) /
  (sum(cm_c35_tables[4, 261:270]) + sum(cm_c35_tables[3, 261:270])), 6)
k27w3c35_truescore <- round((2 * k27w3c35_tpr * k27w3c35_tnr) /
  (k27w3c35_tpr + k27w3c35_tnr), 6)

k29w1c35_tpr <- round(sum(cm_c35_tables[1, 271:280]) /
  (sum(cm_c35_tables[1, 271:280]) + sum(cm_c35_tables[2, 271:280])), 6)
k29w1c35_tnr <- round(sum(cm_c35_tables[4, 271:280]) /

```

```

(sum(cm_c35_tables[4, 271:280]) + sum(cm_c35_tables[3, 271:280])), 6)
k29w1c35_truescore <- round((2 * k29w1c35_tpr * k29w1c35_tnr) /
(k29w1c35_tpr + k29w1c35_tnr), 6)

k29w2c35_tpr <- round(sum(cm_c35_tables[1, 281:290]) /
(sum(cm_c35_tables[1, 281:290]) + sum(cm_c35_tables[2, 281:290])), 6)
k29w2c35_tnr <- round(sum(cm_c35_tables[4, 281:290]) /
(sum(cm_c35_tables[4, 281:290]) + sum(cm_c35_tables[3, 281:290])), 6)
k29w2c35_truescore <- round((2 * k29w2c35_tpr * k29w2c35_tnr) /
(k29w2c35_tpr + k29w2c35_tnr), 6)

k29w3c35_tpr <- round(sum(cm_c35_tables[1, 291:300]) /
(sum(cm_c35_tables[1, 291:300]) + sum(cm_c35_tables[2, 291:300])), 6)
k29w3c35_tnr <- round(sum(cm_c35_tables[4, 291:300]) /
(sum(cm_c35_tables[4, 291:300]) + sum(cm_c35_tables[3, 291:300])), 6)
k29w3c35_truescore <- round((2 * k29w3c35_tpr * k29w3c35_tnr) /
(k29w3c35_tpr + k29w3c35_tnr), 6)

k31w1c35_tpr <- round(sum(cm_c35_tables[1, 301:310]) /
(sum(cm_c35_tables[1, 301:310]) + sum(cm_c35_tables[2, 301:310])), 6)
k31w1c35_tnr <- round(sum(cm_c35_tables[4, 301:310]) /
(sum(cm_c35_tables[4, 301:310]) + sum(cm_c35_tables[3, 301:310])), 6)
k31w1c35_truescore <- round((2 * k31w1c35_tpr * k31w1c35_tnr) /
(k31w1c35_tpr + k31w1c35_tnr), 6)

k31w2c35_tpr <- round(sum(cm_c35_tables[1, 311:320]) /
(sum(cm_c35_tables[1, 311:320]) + sum(cm_c35_tables[2, 311:320])), 6)
k31w2c35_tnr <- round(sum(cm_c35_tables[4, 311:320]) /
(sum(cm_c35_tables[4, 311:320]) + sum(cm_c35_tables[3, 311:320])), 6)
k31w2c35_truescore <- round((2 * k31w2c35_tpr * k31w2c35_tnr) /
(k31w2c35_tpr + k31w2c35_tnr), 6)

k31w3c35_tpr <- round(sum(cm_c35_tables[1, 321:330]) /
(sum(cm_c35_tables[1, 321:330]) + sum(cm_c35_tables[2, 321:330])), 6)
k31w3c35_tnr <- round(sum(cm_c35_tables[4, 321:330]) /
(sum(cm_c35_tables[4, 321:330]) + sum(cm_c35_tables[3, 321:330])), 6)
k31w3c35_truescore <- round((2 * k31w3c35_tpr * k31w3c35_tnr) /
(k31w3c35_tpr + k31w3c35_tnr), 6)

k33w1c35_tpr <- round(sum(cm_c35_tables[1, 331:340]) /
(sum(cm_c35_tables[1, 331:340]) + sum(cm_c35_tables[2, 331:340])), 6)
k33w1c35_tnr <- round(sum(cm_c35_tables[4, 331:340]) /
(sum(cm_c35_tables[4, 331:340]) + sum(cm_c35_tables[3, 331:340])), 6)
k33w1c35_truescore <- round((2 * k33w1c35_tpr * k33w1c35_tnr) /
(k33w1c35_tpr + k33w1c35_tnr), 6)

k33w2c35_tpr <- round(sum(cm_c35_tables[1, 341:350]) /
(sum(cm_c35_tables[1, 341:350]) + sum(cm_c35_tables[2, 341:350])), 6)
k33w2c35_tnr <- round(sum(cm_c35_tables[4, 341:350]) /
(sum(cm_c35_tables[4, 341:350]) + sum(cm_c35_tables[3, 341:350])), 6)
k33w2c35_truescore <- round((2 * k33w2c35_tpr * k33w2c35_tnr) /

```

```

(k33w2c35_tpr + k33w2c35_tnr), 6)

k33w3c35_tpr <- round(sum(cm_c35_tables[1, 351:360]) /
  (sum(cm_c35_tables[1, 351:360]) + sum(cm_c35_tables[2, 351:360])), 6)
k33w3c35_tnr <- round(sum(cm_c35_tables[4, 351:360]) /
  (sum(cm_c35_tables[4, 351:360]) + sum(cm_c35_tables[3, 351:360])), 6)
k33w3c35_truescore <- round((2 * k33w3c35_tpr * k33w3c35_tnr) /
  (k33w3c35_tpr + k33w3c35_tnr), 6)

k35w1c35_tpr <- round(sum(cm_c35_tables[1, 361:370]) /
  (sum(cm_c35_tables[1, 361:370]) + sum(cm_c35_tables[2, 361:370])), 6)
k35w1c35_tnr <- round(sum(cm_c35_tables[4, 361:370]) /
  (sum(cm_c35_tables[4, 361:370]) + sum(cm_c35_tables[3, 361:370])), 6)
k35w1c35_truescore <- round((2 * k35w1c35_tpr * k35w1c35_tnr) /
  (k35w1c35_tpr + k35w1c35_tnr), 6)

k35w2c35_tpr <- round(sum(cm_c35_tables[1, 371:380]) /
  (sum(cm_c35_tables[1, 371:380]) + sum(cm_c35_tables[2, 371:380])), 6)
k35w2c35_tnr <- round(sum(cm_c35_tables[4, 371:380]) /
  (sum(cm_c35_tables[4, 371:380]) + sum(cm_c35_tables[3, 371:380])), 6)
k35w2c35_truescore <- round((2 * k35w2c35_tpr * k35w2c35_tnr) /
  (k35w2c35_tpr + k35w2c35_tnr), 6)

k35w3c35_tpr <- round(sum(cm_c35_tables[1, 381:390]) /
  (sum(cm_c35_tables[1, 381:390]) + sum(cm_c35_tables[2, 381:390])), 6)
k35w3c35_tnr <- round(sum(cm_c35_tables[4, 381:390]) /
  (sum(cm_c35_tables[4, 381:390]) + sum(cm_c35_tables[3, 381:390])), 6)
k35w3c35_truescore <- round((2 * k35w3c35_tpr * k35w3c35_tnr) /
  (k35w3c35_tpr + k35w3c35_tnr), 6)

k37w1c35_tpr <- round(sum(cm_c35_tables[1, 391:400]) /
  (sum(cm_c35_tables[1, 391:400]) + sum(cm_c35_tables[2, 391:400])), 6)
k37w1c35_tnr <- round(sum(cm_c35_tables[4, 391:400]) /
  (sum(cm_c35_tables[4, 391:400]) + sum(cm_c35_tables[3, 391:400])), 6)
k37w1c35_truescore <- round((2 * k37w1c35_tpr * k37w1c35_tnr) /
  (k37w1c35_tpr + k37w1c35_tnr), 6)

k37w2c35_tpr <- round(sum(cm_c35_tables[1, 401:410]) /
  (sum(cm_c35_tables[1, 401:410]) + sum(cm_c35_tables[2, 401:410])), 6)
k37w2c35_tnr <- round(sum(cm_c35_tables[4, 401:410]) /
  (sum(cm_c35_tables[4, 401:410]) + sum(cm_c35_tables[3, 401:410])), 6)
k37w2c35_truescore <- round((2 * k37w2c35_tpr * k37w2c35_tnr) /
  (k37w2c35_tpr + k37w2c35_tnr), 6)

k37w3c35_tpr <- round(sum(cm_c35_tables[1, 411:420]) /
  (sum(cm_c35_tables[1, 411:420]) + sum(cm_c35_tables[2, 411:420])), 6)
k37w3c35_tnr <- round(sum(cm_c35_tables[4, 411:420]) /
  (sum(cm_c35_tables[4, 411:420]) + sum(cm_c35_tables[3, 411:420])), 6)
k37w3c35_truescore <- round((2 * k37w3c35_tpr * k37w3c35_tnr) /
  (k37w3c35_tpr + k37w3c35_tnr), 6)

```

```

k39w1c35_tpr <- round(sum(cm_c35_tables[1, 421:430]) /
  (sum(cm_c35_tables[1, 421:430]) + sum(cm_c35_tables[2, 421:430])), 6)
k39w1c35_tnr <- round(sum(cm_c35_tables[4, 421:430]) /
  (sum(cm_c35_tables[4, 421:430]) + sum(cm_c35_tables[3, 421:430])), 6)
k39w1c35_truescore <- round((2 * k39w1c35_tpr * k39w1c35_tnr) /
  (k39w1c35_tpr + k39w1c35_tnr), 6)

k39w2c35_tpr <- round(sum(cm_c35_tables[1, 431:440]) /
  (sum(cm_c35_tables[1, 431:440]) + sum(cm_c35_tables[2, 431:440])), 6)
k39w2c35_tnr <- round(sum(cm_c35_tables[4, 431:440]) /
  (sum(cm_c35_tables[4, 431:440]) + sum(cm_c35_tables[3, 431:440])), 6)
k39w2c35_truescore <- round((2 * k39w2c35_tpr * k39w2c35_tnr) /
  (k39w2c35_tpr + k39w2c35_tnr), 6)

k39w3c35_tpr <- round(sum(cm_c35_tables[1, 441:450]) /
  (sum(cm_c35_tables[1, 441:450]) + sum(cm_c35_tables[2, 441:450])), 6)
k39w3c35_tnr <- round(sum(cm_c35_tables[4, 441:450]) /
  (sum(cm_c35_tables[4, 441:450]) + sum(cm_c35_tables[3, 441:450])), 6)
k39w3c35_truescore <- round((2 * k39w3c35_tpr * k39w3c35_tnr) /
  (k39w3c35_tpr + k39w3c35_tnr), 6)

# Compile the 0.35 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c35_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.35,
  TPR = c(k11w1c35_tpr, k11w2c35_tpr, k11w3c35_tpr, k13w1c35_tpr,
    k13w2c35_tpr, k13w3c35_tpr, k15w1c35_tpr, k15w2c35_tpr,
    k15w3c35_tpr, k17w1c35_tpr, k17w2c35_tpr, k17w3c35_tpr,
    k19w1c35_tpr, k19w2c35_tpr, k19w3c35_tpr, k21w1c35_tpr,
    k21w2c35_tpr, k21w3c35_tpr, k23w1c35_tpr, k23w2c35_tpr,
    k23w3c35_tpr, k25w1c35_tpr, k25w2c35_tpr, k25w3c35_tpr,
    k27w1c35_tpr, k27w2c35_tpr, k27w3c35_tpr, k29w1c35_tpr,
    k29w2c35_tpr, k29w3c35_tpr, k31w1c35_tpr, k31w2c35_tpr,
    k31w3c35_tpr, k33w1c35_tpr, k33w2c35_tpr, k33w3c35_tpr,
    k35w1c35_tpr, k35w2c35_tpr, k35w3c35_tpr, k37w1c35_tpr,
    k37w2c35_tpr, k37w3c35_tpr, k39w1c35_tpr, k39w2c35_tpr,
    k39w3c35_tpr),
  TNR = c(k11w1c35_tnr, k11w2c35_tnr, k11w3c35_tnr, k13w1c35_tnr,
    k13w2c35_tnr, k13w3c35_tnr, k15w1c35_tnr, k15w2c35_tnr,
    k15w3c35_tnr, k17w1c35_tnr, k17w2c35_tnr, k17w3c35_tnr,
    k19w1c35_tnr, k19w2c35_tnr, k19w3c35_tnr, k21w1c35_tnr,
    k21w2c35_tnr, k21w3c35_tnr, k23w1c35_tnr, k23w2c35_tnr,
    k23w3c35_tnr, k25w1c35_tnr, k25w2c35_tnr, k25w3c35_tnr,
    k27w1c35_tnr, k27w2c35_tnr, k27w3c35_tnr, k29w1c35_tnr,
    k29w2c35_tnr, k29w3c35_tnr, k31w1c35_tnr, k31w2c35_tnr,
    k31w3c35_tnr, k33w1c35_tnr, k33w2c35_tnr, k33w3c35_tnr),
  Truescore = k39w1c35_truescore)

```

```

k35w1c35_tnr, k35w2c35_tnr, k35w3c35_tnr, k37w1c35_tnr,
k37w2c35_tnr, k37w3c35_tnr, k39w1c35_tnr, k39w2c35_tnr,
k39w3c35_tnr),
Truescore = c(k11w1c35_truescore, k11w2c35_truescore,
k11w3c35_truescore, k13w1c35_truescore,
k13w2c35_truescore, k13w3c35_truescore,
k15w1c35_truescore, k15w2c35_truescore,
k15w3c35_truescore, k17w1c35_truescore,
k17w2c35_truescore, k17w3c35_truescore,
k19w1c35_truescore, k19w2c35_truescore,
k19w3c35_truescore, k21w1c35_truescore,
k21w2c35_truescore, k21w3c35_truescore,
k23w1c35_truescore, k23w2c35_truescore,
k23w3c35_truescore, k25w1c35_truescore,
k25w2c35_truescore, k25w3c35_truescore,
k27w1c35_truescore, k27w2c35_truescore,
k27w3c35_truescore, k29w1c35_truescore,
k29w2c35_truescore, k29w3c35_truescore,
k31w1c35_truescore, k31w2c35_truescore,
k31w3c35_truescore, k33w1c35_truescore,
k33w2c35_truescore, k33w3c35_truescore,
k35w1c35_truescore, k35w2c35_truescore,
k35w3c35_truescore, k37w1c35_truescore,
k37w2c35_truescore, k37w3c35_truescore,
k39w1c35_truescore, k39w2c35_truescore,
k39w3c35_truescore))

```

`knitr::kable(c35_results[1:45,], caption = "c35_results")`

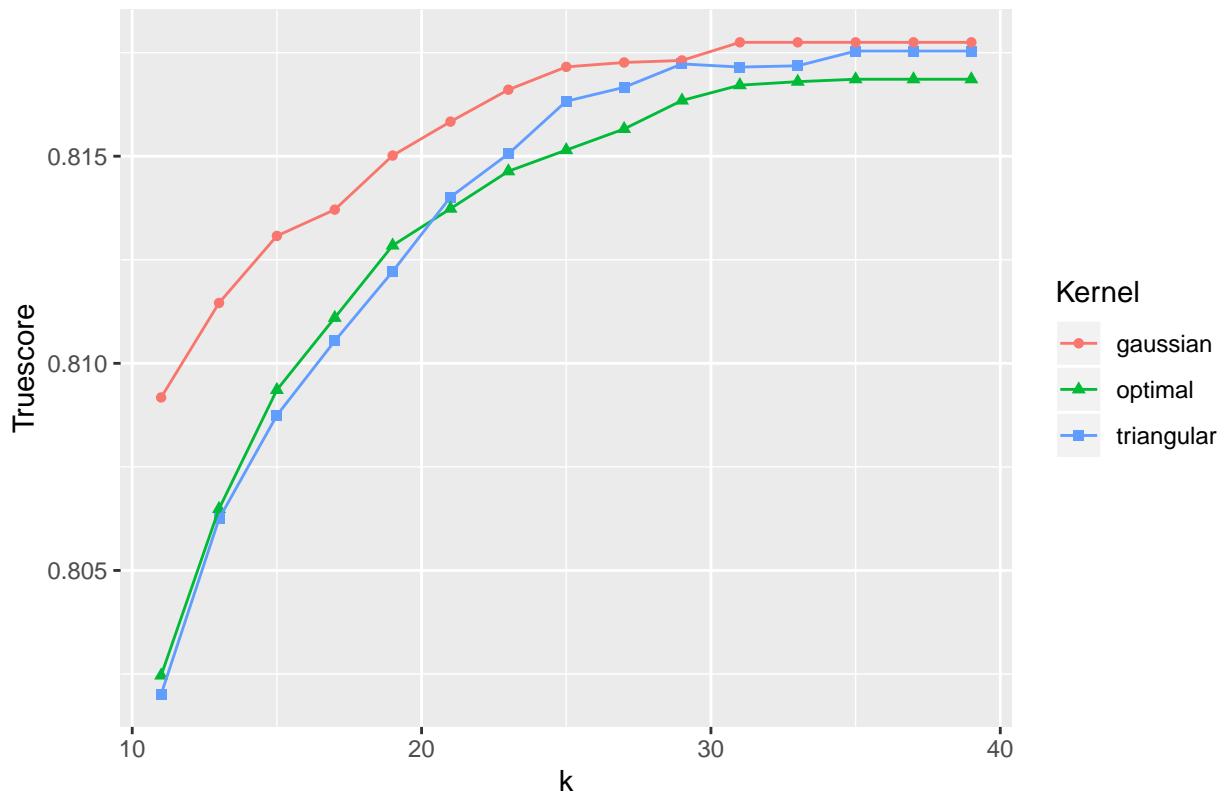
Table 44: c35_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.35	0.768825	0.838189	0.802010
11	gaussian	0.35	0.782681	0.837528	0.809176
11	optimal	0.35	0.769478	0.838404	0.802464
13	triangular	0.35	0.776506	0.838354	0.806246
13	gaussian	0.35	0.786697	0.837826	0.811457
13	optimal	0.35	0.777008	0.838288	0.806486
15	triangular	0.35	0.781124	0.838387	0.808743
15	gaussian	0.35	0.789659	0.837925	0.813076
15	optimal	0.35	0.782028	0.838668	0.809358
17	triangular	0.35	0.784337	0.838553	0.810539
17	gaussian	0.35	0.791667	0.837016	0.813710
17	optimal	0.35	0.785141	0.838833	0.811099
19	triangular	0.35	0.787701	0.838305	0.812216
19	gaussian	0.35	0.794679	0.836422	0.815016
19	optimal	0.35	0.788755	0.838453	0.812845
21	triangular	0.35	0.791064	0.838338	0.814015
21	gaussian	0.35	0.796536	0.836092	0.815835
21	optimal	0.35	0.790813	0.838024	0.813734
23	triangular	0.35	0.793624	0.837677	0.815056
23	gaussian	0.35	0.797992	0.836108	0.816605
23	optimal	0.35	0.793273	0.837182	0.814636

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.35	0.796486	0.837182	0.816327
25	gaussian	0.35	0.799649	0.835447	0.817156
25	optimal	0.35	0.794930	0.836422	0.815148
27	triangular	0.35	0.797440	0.836835	0.816663
27	gaussian	0.35	0.800201	0.835068	0.817263
27	optimal	0.35	0.796034	0.836273	0.815658
29	triangular	0.35	0.798896	0.836422	0.817228
29	gaussian	0.35	0.800251	0.835117	0.817312
29	optimal	0.35	0.797691	0.835893	0.816345
31	triangular	0.35	0.799247	0.835877	0.817152
31	gaussian	0.35	0.801606	0.834556	0.817749
31	optimal	0.35	0.798745	0.835513	0.816715
33	triangular	0.35	0.799699	0.835447	0.817182
33	gaussian	0.35	0.801606	0.834556	0.817749
33	optimal	0.35	0.799297	0.835084	0.816799
35	triangular	0.35	0.800653	0.835150	0.817538
35	gaussian	0.35	0.801606	0.834556	0.817749
35	optimal	0.35	0.799548	0.834935	0.816858
37	triangular	0.35	0.800653	0.835150	0.817538
37	gaussian	0.35	0.801606	0.834556	0.817749
37	optimal	0.35	0.799548	0.834935	0.816858
39	triangular	0.35	0.800653	0.835150	0.817538
39	gaussian	0.35	0.801606	0.834556	0.817749
39	optimal	0.35	0.799548	0.834935	0.816858

```
ggplot(c35_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.35 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.35 (Truescore)

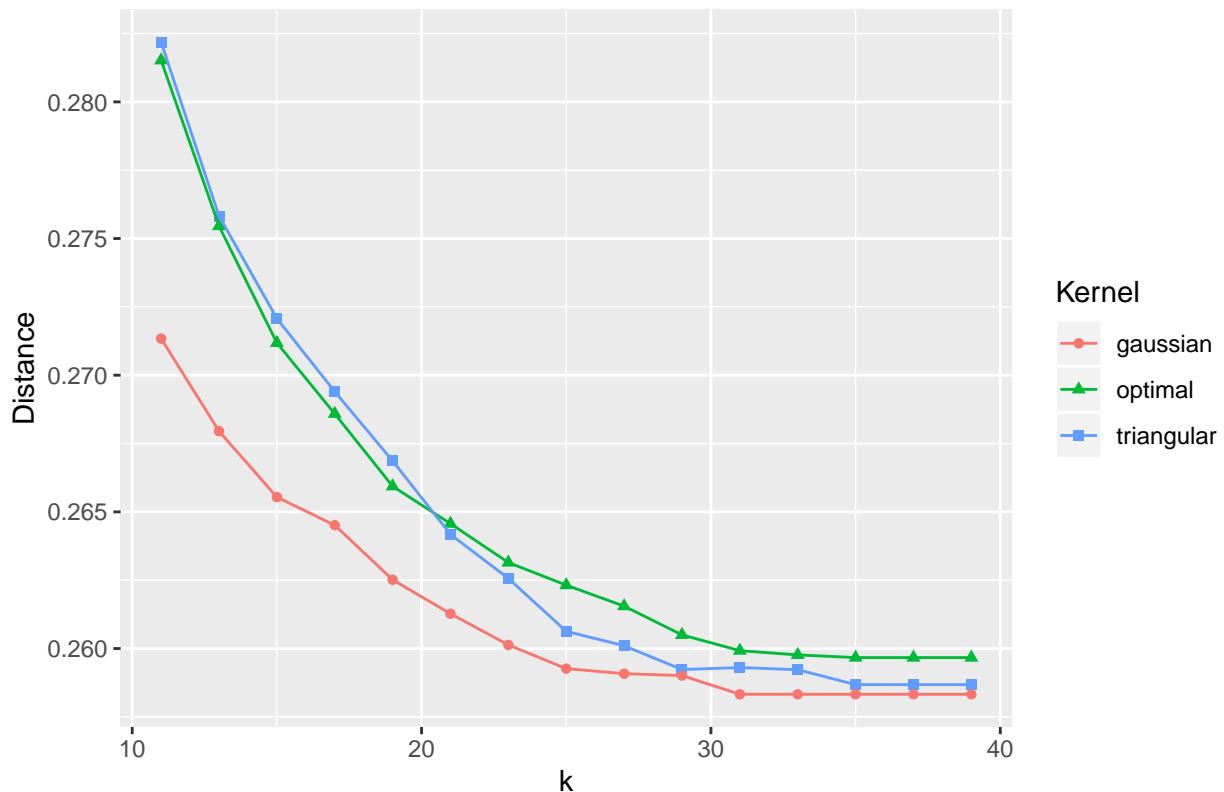


```
# For the Cutoff of 0.35, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c35_results <- c35_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c35_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.35 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.35 (Distance)



```
# For the Cutoff of 0.35, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c35_results$Truescore)
```

```
## [1] 0.817749
```

```
(c35_opt_k_ts <- c35_results$k[which.max(c35_results$Truescore)])
```

```
## [1] 31
```

```
(c35_opt_kernel_ts <- c35_results$Kernel[which.max(c35_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c35_opt_cut_ts <- c35_results$Cut[which.max(c35_results$Truescore)])
```

```
## [1] 0.35
```

```
(c35_opt_tpr_ts <- c35_results$TPR[which.max(c35_results$Truescore)])
```

```
## [1] 0.801606
```

```

(c35_opt_tnr_ts <- c35_results$TNR[which.max(c35_results$Truescore)])

## [1] 0.834556

(c35_opt_d_ts <- c35_results$Distance[which.max(c35_results$Truescore)])

## [1] 0.2583252

min(c35_results$Distance)

## [1] 0.2583252

(c35_opt_k_dist <- c35_results$k[which.min(c35_results$Distance)])

## [1] 31

(c35_opt_kernel_dist <- c35_results$Kernel[which.min(c35_results$Distance)])

## [1] "gaussian"

(c35_opt_cut_dist <- c35_results$Cut[which.min(c35_results$Distance)])

## [1] 0.35

(c35_opt_tpr_dist <- c35_results$TPR[which.min(c35_results$Distance)])

## [1] 0.801606

(c35_opt_tnr_dist <- c35_results$TNR[which.min(c35_results$Distance)])

## [1] 0.834556

(c35_opt_t_dist <- c35_results$Truescore[which.min(c35_results$Distance)])

## [1] 0.817749

#####
# 0.36 Cutoff
#####

# For the decision cutoff of 0.36, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c36 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred36, obs))
  confusionMatrix(ss$pred36, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c36) <- kwf_dfs_v

cm_c36_tables <- sapply(cm_c36, "[[", 2)
cm_c36_tables <- as_tibble(cm_c36_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.36.

k11w1c36_tpr <- round(sum(cm_c36_tables[1, 1:10]) /
  (sum(cm_c36_tables[1, 1:10]) + sum(cm_c36_tables[2, 1:10])), 6)
k11w1c36_tnr <- round(sum(cm_c36_tables[4, 1:10]) /
  (sum(cm_c36_tables[4, 1:10]) + sum(cm_c36_tables[3, 1:10])), 6)
k11w1c36_truescore <- round((2 * k11w1c36_tpr * k11w1c36_tnr) /
  (k11w1c36_tpr + k11w1c36_tnr), 6)

k11w2c36_tpr <- round(sum(cm_c36_tables[1, 11:20]) /
  (sum(cm_c36_tables[1, 11:20]) + sum(cm_c36_tables[2, 11:20])), 6)
k11w2c36_tnr <- round(sum(cm_c36_tables[4, 11:20]) /
  (sum(cm_c36_tables[4, 11:20]) + sum(cm_c36_tables[3, 11:20])), 6)
k11w2c36_truescore <- round((2 * k11w2c36_tpr * k11w2c36_tnr) /
  (k11w2c36_tpr + k11w2c36_tnr), 6)

k11w3c36_tpr <- round(sum(cm_c36_tables[1, 21:30]) /
  (sum(cm_c36_tables[1, 21:30]) + sum(cm_c36_tables[2, 21:30])), 6)
k11w3c36_tnr <- round(sum(cm_c36_tables[4, 21:30]) /
  (sum(cm_c36_tables[4, 21:30]) + sum(cm_c36_tables[3, 21:30])), 6)
k11w3c36_truescore <- round((2 * k11w3c36_tpr * k11w3c36_tnr) /
  (k11w3c36_tpr + k11w3c36_tnr), 6)

k13w1c36_tpr <- round(sum(cm_c36_tables[1, 31:40]) /
  (sum(cm_c36_tables[1, 31:40]) + sum(cm_c36_tables[2, 31:40])), 6)
k13w1c36_tnr <- round(sum(cm_c36_tables[4, 31:40]) /
  (sum(cm_c36_tables[4, 31:40]) + sum(cm_c36_tables[3, 31:40])), 6)
k13w1c36_truescore <- round((2 * k13w1c36_tpr * k13w1c36_tnr) /
  (k13w1c36_tpr + k13w1c36_tnr), 6)

k13w2c36_tpr <- round(sum(cm_c36_tables[1, 41:50]) /
  (sum(cm_c36_tables[1, 41:50]) + sum(cm_c36_tables[2, 41:50])), 6)
k13w2c36_tnr <- round(sum(cm_c36_tables[4, 41:50]) /
  (sum(cm_c36_tables[4, 41:50]) + sum(cm_c36_tables[3, 41:50])), 6)
k13w2c36_truescore <- round((2 * k13w2c36_tpr * k13w2c36_tnr) /
  (k13w2c36_tpr + k13w2c36_tnr), 6)

k13w3c36_tpr <- round(sum(cm_c36_tables[1, 51:60]) /
  (sum(cm_c36_tables[1, 51:60]) + sum(cm_c36_tables[2, 51:60])), 6)
k13w3c36_tnr <- round(sum(cm_c36_tables[4, 51:60]) /
  (sum(cm_c36_tables[4, 51:60]) + sum(cm_c36_tables[3, 51:60])), 6)
k13w3c36_truescore <- round((2 * k13w3c36_tpr * k13w3c36_tnr) /

```

```

(k13w3c36_tpr + k13w3c36_tnr), 6)

k15w1c36_tpr <- round(sum(cm_c36_tables[1, 61:70]) /
  (sum(cm_c36_tables[1, 61:70]) + sum(cm_c36_tables[2, 61:70])), 6)
k15w1c36_tnr <- round(sum(cm_c36_tables[4, 61:70]) /
  (sum(cm_c36_tables[4, 61:70]) + sum(cm_c36_tables[3, 61:70])), 6)
k15w1c36_truescore <- round((2 * k15w1c36_tpr * k15w1c36_tnr) /
  (k15w1c36_tpr + k15w1c36_tnr), 6)

k15w2c36_tpr <- round(sum(cm_c36_tables[1, 71:80]) /
  (sum(cm_c36_tables[1, 71:80]) + sum(cm_c36_tables[2, 71:80])), 6)
k15w2c36_tnr <- round(sum(cm_c36_tables[4, 71:80]) /
  (sum(cm_c36_tables[4, 71:80]) + sum(cm_c36_tables[3, 71:80])), 6)
k15w2c36_truescore <- round((2 * k15w2c36_tpr * k15w2c36_tnr) /
  (k15w2c36_tpr + k15w2c36_tnr), 6)

k15w3c36_tpr <- round(sum(cm_c36_tables[1, 81:90]) /
  (sum(cm_c36_tables[1, 81:90]) + sum(cm_c36_tables[2, 81:90])), 6)
k15w3c36_tnr <- round(sum(cm_c36_tables[4, 81:90]) /
  (sum(cm_c36_tables[4, 81:90]) + sum(cm_c36_tables[3, 81:90])), 6)
k15w3c36_truescore <- round((2 * k15w3c36_tpr * k15w3c36_tnr) /
  (k15w3c36_tpr + k15w3c36_tnr), 6)

k17w1c36_tpr <- round(sum(cm_c36_tables[1, 91:100]) /
  (sum(cm_c36_tables[1, 91:100]) + sum(cm_c36_tables[2, 91:100])), 6)
k17w1c36_tnr <- round(sum(cm_c36_tables[4, 91:100]) /
  (sum(cm_c36_tables[4, 91:100]) + sum(cm_c36_tables[3, 91:100])), 6)
k17w1c36_truescore <- round((2 * k17w1c36_tpr * k17w1c36_tnr) /
  (k17w1c36_tpr + k17w1c36_tnr), 6)

k17w2c36_tpr <- round(sum(cm_c36_tables[1, 101:110]) /
  (sum(cm_c36_tables[1, 101:110]) + sum(cm_c36_tables[2, 101:110])), 6)
k17w2c36_tnr <- round(sum(cm_c36_tables[4, 101:110]) /
  (sum(cm_c36_tables[4, 101:110]) + sum(cm_c36_tables[3, 101:110])), 6)
k17w2c36_truescore <- round((2 * k17w2c36_tpr * k17w2c36_tnr) /
  (k17w2c36_tpr + k17w2c36_tnr), 6)

k17w3c36_tpr <- round(sum(cm_c36_tables[1, 111:120]) /
  (sum(cm_c36_tables[1, 111:120]) + sum(cm_c36_tables[2, 111:120])), 6)
k17w3c36_tnr <- round(sum(cm_c36_tables[4, 111:120]) /
  (sum(cm_c36_tables[4, 111:120]) + sum(cm_c36_tables[3, 111:120])), 6)
k17w3c36_truescore <- round((2 * k17w3c36_tpr * k17w3c36_tnr) /
  (k17w3c36_tpr + k17w3c36_tnr), 6)

k19w1c36_tpr <- round(sum(cm_c36_tables[1, 121:130]) /
  (sum(cm_c36_tables[1, 121:130]) + sum(cm_c36_tables[2, 121:130])), 6)
k19w1c36_tnr <- round(sum(cm_c36_tables[4, 121:130]) /
  (sum(cm_c36_tables[4, 121:130]) + sum(cm_c36_tables[3, 121:130])), 6)
k19w1c36_truescore <- round((2 * k19w1c36_tpr * k19w1c36_tnr) /
  (k19w1c36_tpr + k19w1c36_tnr), 6)

```

```

k19w2c36_tpr <- round(sum(cm_c36_tables[1, 131:140]) /
  (sum(cm_c36_tables[1, 131:140]) + sum(cm_c36_tables[2, 131:140])), 6)
k19w2c36_tnr <- round(sum(cm_c36_tables[4, 131:140]) /
  (sum(cm_c36_tables[4, 131:140]) + sum(cm_c36_tables[3, 131:140])), 6)
k19w2c36_truescore <- round((2 * k19w2c36_tpr * k19w2c36_tnr) /
  (k19w2c36_tpr + k19w2c36_tnr), 6)

k19w3c36_tpr <- round(sum(cm_c36_tables[1, 141:150]) /
  (sum(cm_c36_tables[1, 141:150]) + sum(cm_c36_tables[2, 141:150])), 6)
k19w3c36_tnr <- round(sum(cm_c36_tables[4, 141:150]) /
  (sum(cm_c36_tables[4, 141:150]) + sum(cm_c36_tables[3, 141:150])), 6)
k19w3c36_truescore <- round((2 * k19w3c36_tpr * k19w3c36_tnr) /
  (k19w3c36_tpr + k19w3c36_tnr), 6)

k21w1c36_tpr <- round(sum(cm_c36_tables[1, 151:160]) /
  (sum(cm_c36_tables[1, 151:160]) + sum(cm_c36_tables[2, 151:160])), 6)
k21w1c36_tnr <- round(sum(cm_c36_tables[4, 151:160]) /
  (sum(cm_c36_tables[4, 151:160]) + sum(cm_c36_tables[3, 151:160])), 6)
k21w1c36_truescore <- round((2 * k21w1c36_tpr * k21w1c36_tnr) /
  (k21w1c36_tpr + k21w1c36_tnr), 6)

k21w2c36_tpr <- round(sum(cm_c36_tables[1, 161:170]) /
  (sum(cm_c36_tables[1, 161:170]) + sum(cm_c36_tables[2, 161:170])), 6)
k21w2c36_tnr <- round(sum(cm_c36_tables[4, 161:170]) /
  (sum(cm_c36_tables[4, 161:170]) + sum(cm_c36_tables[3, 161:170])), 6)
k21w2c36_truescore <- round((2 * k21w2c36_tpr * k21w2c36_tnr) /
  (k21w2c36_tpr + k21w2c36_tnr), 6)

k21w3c36_tpr <- round(sum(cm_c36_tables[1, 171:180]) /
  (sum(cm_c36_tables[1, 171:180]) + sum(cm_c36_tables[2, 171:180])), 6)
k21w3c36_tnr <- round(sum(cm_c36_tables[4, 171:180]) /
  (sum(cm_c36_tables[4, 171:180]) + sum(cm_c36_tables[3, 171:180])), 6)
k21w3c36_truescore <- round((2 * k21w3c36_tpr * k21w3c36_tnr) /
  (k21w3c36_tpr + k21w3c36_tnr), 6)

k23w1c36_tpr <- round(sum(cm_c36_tables[1, 181:190]) /
  (sum(cm_c36_tables[1, 181:190]) + sum(cm_c36_tables[2, 181:190])), 6)
k23w1c36_tnr <- round(sum(cm_c36_tables[4, 181:190]) /
  (sum(cm_c36_tables[4, 181:190]) + sum(cm_c36_tables[3, 181:190])), 6)
k23w1c36_truescore <- round((2 * k23w1c36_tpr * k23w1c36_tnr) /
  (k23w1c36_tpr + k23w1c36_tnr), 6)

k23w2c36_tpr <- round(sum(cm_c36_tables[1, 191:200]) /
  (sum(cm_c36_tables[1, 191:200]) + sum(cm_c36_tables[2, 191:200])), 6)
k23w2c36_tnr <- round(sum(cm_c36_tables[4, 191:200]) /
  (sum(cm_c36_tables[4, 191:200]) + sum(cm_c36_tables[3, 191:200])), 6)
k23w2c36_truescore <- round((2 * k23w2c36_tpr * k23w2c36_tnr) /
  (k23w2c36_tpr + k23w2c36_tnr), 6)

k23w3c36_tpr <- round(sum(cm_c36_tables[1, 201:210]) /
  (sum(cm_c36_tables[1, 201:210]) + sum(cm_c36_tables[2, 201:210])), 6)

```

```

k23w3c36_tnr <- round(sum(cm_c36_tables[4, 201:210]) /
  (sum(cm_c36_tables[4, 201:210]) + sum(cm_c36_tables[3, 201:210])), 6)
k23w3c36_truescore <- round((2 * k23w3c36_tpr * k23w3c36_tnr) /
  (k23w3c36_tpr + k23w3c36_tnr), 6)

k25w1c36_tpr <- round(sum(cm_c36_tables[1, 211:220]) /
  (sum(cm_c36_tables[1, 211:220]) + sum(cm_c36_tables[2, 211:220])), 6)
k25w1c36_tnr <- round(sum(cm_c36_tables[4, 211:220]) /
  (sum(cm_c36_tables[4, 211:220]) + sum(cm_c36_tables[3, 211:220])), 6)
k25w1c36_truescore <- round((2 * k25w1c36_tpr * k25w1c36_tnr) /
  (k25w1c36_tpr + k25w1c36_tnr), 6)

k25w2c36_tpr <- round(sum(cm_c36_tables[1, 221:230]) /
  (sum(cm_c36_tables[1, 221:230]) + sum(cm_c36_tables[2, 221:230])), 6)
k25w2c36_tnr <- round(sum(cm_c36_tables[4, 221:230]) /
  (sum(cm_c36_tables[4, 221:230]) + sum(cm_c36_tables[3, 221:230])), 6)
k25w2c36_truescore <- round((2 * k25w2c36_tpr * k25w2c36_tnr) /
  (k25w2c36_tpr + k25w2c36_tnr), 6)

k25w3c36_tpr <- round(sum(cm_c36_tables[1, 231:240]) /
  (sum(cm_c36_tables[1, 231:240]) + sum(cm_c36_tables[2, 231:240])), 6)
k25w3c36_tnr <- round(sum(cm_c36_tables[4, 231:240]) /
  (sum(cm_c36_tables[4, 231:240]) + sum(cm_c36_tables[3, 231:240])), 6)
k25w3c36_truescore <- round((2 * k25w3c36_tpr * k25w3c36_tnr) /
  (k25w3c36_tpr + k25w3c36_tnr), 6)

k27w1c36_tpr <- round(sum(cm_c36_tables[1, 241:250]) /
  (sum(cm_c36_tables[1, 241:250]) + sum(cm_c36_tables[2, 241:250])), 6)
k27w1c36_tnr <- round(sum(cm_c36_tables[4, 241:250]) /
  (sum(cm_c36_tables[4, 241:250]) + sum(cm_c36_tables[3, 241:250])), 6)
k27w1c36_truescore <- round((2 * k27w1c36_tpr * k27w1c36_tnr) /
  (k27w1c36_tpr + k27w1c36_tnr), 6)

k27w2c36_tpr <- round(sum(cm_c36_tables[1, 251:260]) /
  (sum(cm_c36_tables[1, 251:260]) + sum(cm_c36_tables[2, 251:260])), 6)
k27w2c36_tnr <- round(sum(cm_c36_tables[4, 251:260]) /
  (sum(cm_c36_tables[4, 251:260]) + sum(cm_c36_tables[3, 251:260])), 6)
k27w2c36_truescore <- round((2 * k27w2c36_tpr * k27w2c36_tnr) /
  (k27w2c36_tpr + k27w2c36_tnr), 6)

k27w3c36_tpr <- round(sum(cm_c36_tables[1, 261:270]) /
  (sum(cm_c36_tables[1, 261:270]) + sum(cm_c36_tables[2, 261:270])), 6)
k27w3c36_tnr <- round(sum(cm_c36_tables[4, 261:270]) /
  (sum(cm_c36_tables[4, 261:270]) + sum(cm_c36_tables[3, 261:270])), 6)
k27w3c36_truescore <- round((2 * k27w3c36_tpr * k27w3c36_tnr) /
  (k27w3c36_tpr + k27w3c36_tnr), 6)

k29w1c36_tpr <- round(sum(cm_c36_tables[1, 271:280]) /
  (sum(cm_c36_tables[1, 271:280]) + sum(cm_c36_tables[2, 271:280])), 6)
k29w1c36_tnr <- round(sum(cm_c36_tables[4, 271:280]) /

```

```

(sum(cm_c36_tables[4, 271:280]) + sum(cm_c36_tables[3, 271:280])), 6)
k29w1c36_truescore <- round((2 * k29w1c36_tpr * k29w1c36_tnr) /
(k29w1c36_tpr + k29w1c36_tnr), 6)

k29w2c36_tpr <- round(sum(cm_c36_tables[1, 281:290]) /
(sum(cm_c36_tables[1, 281:290]) + sum(cm_c36_tables[2, 281:290])), 6)
k29w2c36_tnr <- round(sum(cm_c36_tables[4, 281:290]) /
(sum(cm_c36_tables[4, 281:290]) + sum(cm_c36_tables[3, 281:290])), 6)
k29w2c36_truescore <- round((2 * k29w2c36_tpr * k29w2c36_tnr) /
(k29w2c36_tpr + k29w2c36_tnr), 6)

k29w3c36_tpr <- round(sum(cm_c36_tables[1, 291:300]) /
(sum(cm_c36_tables[1, 291:300]) + sum(cm_c36_tables[2, 291:300])), 6)
k29w3c36_tnr <- round(sum(cm_c36_tables[4, 291:300]) /
(sum(cm_c36_tables[4, 291:300]) + sum(cm_c36_tables[3, 291:300])), 6)
k29w3c36_truescore <- round((2 * k29w3c36_tpr * k29w3c36_tnr) /
(k29w3c36_tpr + k29w3c36_tnr), 6)

k31w1c36_tpr <- round(sum(cm_c36_tables[1, 301:310]) /
(sum(cm_c36_tables[1, 301:310]) + sum(cm_c36_tables[2, 301:310])), 6)
k31w1c36_tnr <- round(sum(cm_c36_tables[4, 301:310]) /
(sum(cm_c36_tables[4, 301:310]) + sum(cm_c36_tables[3, 301:310])), 6)
k31w1c36_truescore <- round((2 * k31w1c36_tpr * k31w1c36_tnr) /
(k31w1c36_tpr + k31w1c36_tnr), 6)

k31w2c36_tpr <- round(sum(cm_c36_tables[1, 311:320]) /
(sum(cm_c36_tables[1, 311:320]) + sum(cm_c36_tables[2, 311:320])), 6)
k31w2c36_tnr <- round(sum(cm_c36_tables[4, 311:320]) /
(sum(cm_c36_tables[4, 311:320]) + sum(cm_c36_tables[3, 311:320])), 6)
k31w2c36_truescore <- round((2 * k31w2c36_tpr * k31w2c36_tnr) /
(k31w2c36_tpr + k31w2c36_tnr), 6)

k31w3c36_tpr <- round(sum(cm_c36_tables[1, 321:330]) /
(sum(cm_c36_tables[1, 321:330]) + sum(cm_c36_tables[2, 321:330])), 6)
k31w3c36_tnr <- round(sum(cm_c36_tables[4, 321:330]) /
(sum(cm_c36_tables[4, 321:330]) + sum(cm_c36_tables[3, 321:330])), 6)
k31w3c36_truescore <- round((2 * k31w3c36_tpr * k31w3c36_tnr) /
(k31w3c36_tpr + k31w3c36_tnr), 6)

k33w1c36_tpr <- round(sum(cm_c36_tables[1, 331:340]) /
(sum(cm_c36_tables[1, 331:340]) + sum(cm_c36_tables[2, 331:340])), 6)
k33w1c36_tnr <- round(sum(cm_c36_tables[4, 331:340]) /
(sum(cm_c36_tables[4, 331:340]) + sum(cm_c36_tables[3, 331:340])), 6)
k33w1c36_truescore <- round((2 * k33w1c36_tpr * k33w1c36_tnr) /
(k33w1c36_tpr + k33w1c36_tnr), 6)

k33w2c36_tpr <- round(sum(cm_c36_tables[1, 341:350]) /
(sum(cm_c36_tables[1, 341:350]) + sum(cm_c36_tables[2, 341:350])), 6)
k33w2c36_tnr <- round(sum(cm_c36_tables[4, 341:350]) /
(sum(cm_c36_tables[4, 341:350]) + sum(cm_c36_tables[3, 341:350])), 6)
k33w2c36_truescore <- round((2 * k33w2c36_tpr * k33w2c36_tnr) /

```

```

(k33w2c36_tpr + k33w2c36_tnr), 6)

k33w3c36_tpr <- round(sum(cm_c36_tables[1, 351:360]) /
  (sum(cm_c36_tables[1, 351:360]) + sum(cm_c36_tables[2, 351:360])), 6)
k33w3c36_tnr <- round(sum(cm_c36_tables[4, 351:360]) /
  (sum(cm_c36_tables[4, 351:360]) + sum(cm_c36_tables[3, 351:360])), 6)
k33w3c36_truescore <- round((2 * k33w3c36_tpr * k33w3c36_tnr) /
  (k33w3c36_tpr + k33w3c36_tnr), 6)

k35w1c36_tpr <- round(sum(cm_c36_tables[1, 361:370]) /
  (sum(cm_c36_tables[1, 361:370]) + sum(cm_c36_tables[2, 361:370])), 6)
k35w1c36_tnr <- round(sum(cm_c36_tables[4, 361:370]) /
  (sum(cm_c36_tables[4, 361:370]) + sum(cm_c36_tables[3, 361:370])), 6)
k35w1c36_truescore <- round((2 * k35w1c36_tpr * k35w1c36_tnr) /
  (k35w1c36_tpr + k35w1c36_tnr), 6)

k35w2c36_tpr <- round(sum(cm_c36_tables[1, 371:380]) /
  (sum(cm_c36_tables[1, 371:380]) + sum(cm_c36_tables[2, 371:380])), 6)
k35w2c36_tnr <- round(sum(cm_c36_tables[4, 371:380]) /
  (sum(cm_c36_tables[4, 371:380]) + sum(cm_c36_tables[3, 371:380])), 6)
k35w2c36_truescore <- round((2 * k35w2c36_tpr * k35w2c36_tnr) /
  (k35w2c36_tpr + k35w2c36_tnr), 6)

k35w3c36_tpr <- round(sum(cm_c36_tables[1, 381:390]) /
  (sum(cm_c36_tables[1, 381:390]) + sum(cm_c36_tables[2, 381:390])), 6)
k35w3c36_tnr <- round(sum(cm_c36_tables[4, 381:390]) /
  (sum(cm_c36_tables[4, 381:390]) + sum(cm_c36_tables[3, 381:390])), 6)
k35w3c36_truescore <- round((2 * k35w3c36_tpr * k35w3c36_tnr) /
  (k35w3c36_tpr + k35w3c36_tnr), 6)

k37w1c36_tpr <- round(sum(cm_c36_tables[1, 391:400]) /
  (sum(cm_c36_tables[1, 391:400]) + sum(cm_c36_tables[2, 391:400])), 6)
k37w1c36_tnr <- round(sum(cm_c36_tables[4, 391:400]) /
  (sum(cm_c36_tables[4, 391:400]) + sum(cm_c36_tables[3, 391:400])), 6)
k37w1c36_truescore <- round((2 * k37w1c36_tpr * k37w1c36_tnr) /
  (k37w1c36_tpr + k37w1c36_tnr), 6)

k37w2c36_tpr <- round(sum(cm_c36_tables[1, 401:410]) /
  (sum(cm_c36_tables[1, 401:410]) + sum(cm_c36_tables[2, 401:410])), 6)
k37w2c36_tnr <- round(sum(cm_c36_tables[4, 401:410]) /
  (sum(cm_c36_tables[4, 401:410]) + sum(cm_c36_tables[3, 401:410])), 6)
k37w2c36_truescore <- round((2 * k37w2c36_tpr * k37w2c36_tnr) /
  (k37w2c36_tpr + k37w2c36_tnr), 6)

k37w3c36_tpr <- round(sum(cm_c36_tables[1, 411:420]) /
  (sum(cm_c36_tables[1, 411:420]) + sum(cm_c36_tables[2, 411:420])), 6)
k37w3c36_tnr <- round(sum(cm_c36_tables[4, 411:420]) /
  (sum(cm_c36_tables[4, 411:420]) + sum(cm_c36_tables[3, 411:420])), 6)
k37w3c36_truescore <- round((2 * k37w3c36_tpr * k37w3c36_tnr) /
  (k37w3c36_tpr + k37w3c36_tnr), 6)

```

```

k39w1c36_tpr <- round(sum(cm_c36_tables[1, 421:430]) /
  (sum(cm_c36_tables[1, 421:430]) + sum(cm_c36_tables[2, 421:430])), 6)
k39w1c36_tnr <- round(sum(cm_c36_tables[4, 421:430]) /
  (sum(cm_c36_tables[4, 421:430]) + sum(cm_c36_tables[3, 421:430])), 6)
k39w1c36_truescore <- round((2 * k39w1c36_tpr * k39w1c36_tnr) /
  (k39w1c36_tpr + k39w1c36_tnr), 6)

k39w2c36_tpr <- round(sum(cm_c36_tables[1, 431:440]) /
  (sum(cm_c36_tables[1, 431:440]) + sum(cm_c36_tables[2, 431:440])), 6)
k39w2c36_tnr <- round(sum(cm_c36_tables[4, 431:440]) /
  (sum(cm_c36_tables[4, 431:440]) + sum(cm_c36_tables[3, 431:440])), 6)
k39w2c36_truescore <- round((2 * k39w2c36_tpr * k39w2c36_tnr) /
  (k39w2c36_tpr + k39w2c36_tnr), 6)

k39w3c36_tpr <- round(sum(cm_c36_tables[1, 441:450]) /
  (sum(cm_c36_tables[1, 441:450]) + sum(cm_c36_tables[2, 441:450])), 6)
k39w3c36_tnr <- round(sum(cm_c36_tables[4, 441:450]) /
  (sum(cm_c36_tables[4, 441:450]) + sum(cm_c36_tables[3, 441:450])), 6)
k39w3c36_truescore <- round((2 * k39w3c36_tpr * k39w3c36_tnr) /
  (k39w3c36_tpr + k39w3c36_tnr), 6)

# Compile the 0.36 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c36_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.36,
  TPR = c(k11w1c36_tpr, k11w2c36_tpr, k11w3c36_tpr, k13w1c36_tpr,
    k13w2c36_tpr, k13w3c36_tpr, k15w1c36_tpr, k15w2c36_tpr,
    k15w3c36_tpr, k17w1c36_tpr, k17w2c36_tpr, k17w3c36_tpr,
    k19w1c36_tpr, k19w2c36_tpr, k19w3c36_tpr, k21w1c36_tpr,
    k21w2c36_tpr, k21w3c36_tpr, k23w1c36_tpr, k23w2c36_tpr,
    k23w3c36_tpr, k25w1c36_tpr, k25w2c36_tpr, k25w3c36_tpr,
    k27w1c36_tpr, k27w2c36_tpr, k27w3c36_tpr, k29w1c36_tpr,
    k29w2c36_tpr, k29w3c36_tpr, k31w1c36_tpr, k31w2c36_tpr,
    k31w3c36_tpr, k33w1c36_tpr, k33w2c36_tpr, k33w3c36_tpr,
    k35w1c36_tpr, k35w2c36_tpr, k35w3c36_tpr, k37w1c36_tpr,
    k37w2c36_tpr, k37w3c36_tpr, k39w1c36_tpr, k39w2c36_tpr,
    k39w3c36_tpr),
  TNR = c(k11w1c36_tnr, k11w2c36_tnr, k11w3c36_tnr, k13w1c36_tnr,
    k13w2c36_tnr, k13w3c36_tnr, k15w1c36_tnr, k15w2c36_tnr,
    k15w3c36_tnr, k17w1c36_tnr, k17w2c36_tnr, k17w3c36_tnr,
    k19w1c36_tnr, k19w2c36_tnr, k19w3c36_tnr, k21w1c36_tnr,
    k21w2c36_tnr, k21w3c36_tnr, k23w1c36_tnr, k23w2c36_tnr,
    k23w3c36_tnr, k25w1c36_tnr, k25w2c36_tnr, k25w3c36_tnr,
    k27w1c36_tnr, k27w2c36_tnr, k27w3c36_tnr, k29w1c36_tnr,
    k29w2c36_tnr, k29w3c36_tnr, k31w1c36_tnr, k31w2c36_tnr,
    k31w3c36_tnr, k33w1c36_tnr, k33w2c36_tnr, k33w3c36_tnr)

```

```

k35w1c36_tnr, k35w2c36_tnr, k35w3c36_tnr, k37w1c36_tnr,
k37w2c36_tnr, k37w3c36_tnr, k39w1c36_tnr, k39w2c36_tnr,
k39w3c36_tnr),
Truescore = c(k11w1c36_truescore, k11w2c36_truescore,
k11w3c36_truescore, k13w1c36_truescore,
k13w2c36_truescore, k13w3c36_truescore,
k15w1c36_truescore, k15w2c36_truescore,
k15w3c36_truescore, k17w1c36_truescore,
k17w2c36_truescore, k17w3c36_truescore,
k19w1c36_truescore, k19w2c36_truescore,
k19w3c36_truescore, k21w1c36_truescore,
k21w2c36_truescore, k21w3c36_truescore,
k23w1c36_truescore, k23w2c36_truescore,
k23w3c36_truescore, k25w1c36_truescore,
k25w2c36_truescore, k25w3c36_truescore,
k27w1c36_truescore, k27w2c36_truescore,
k27w3c36_truescore, k29w1c36_truescore,
k29w2c36_truescore, k29w3c36_truescore,
k31w1c36_truescore, k31w2c36_truescore,
k31w3c36_truescore, k33w1c36_truescore,
k33w2c36_truescore, k33w3c36_truescore,
k35w1c36_truescore, k35w2c36_truescore,
k35w3c36_truescore, k37w1c36_truescore,
k37w2c36_truescore, k37w3c36_truescore,
k39w1c36_truescore, k39w2c36_truescore,
k39w3c36_truescore))

```

`knitr::kable(c36_results[1:45,], caption = "c36_results")`

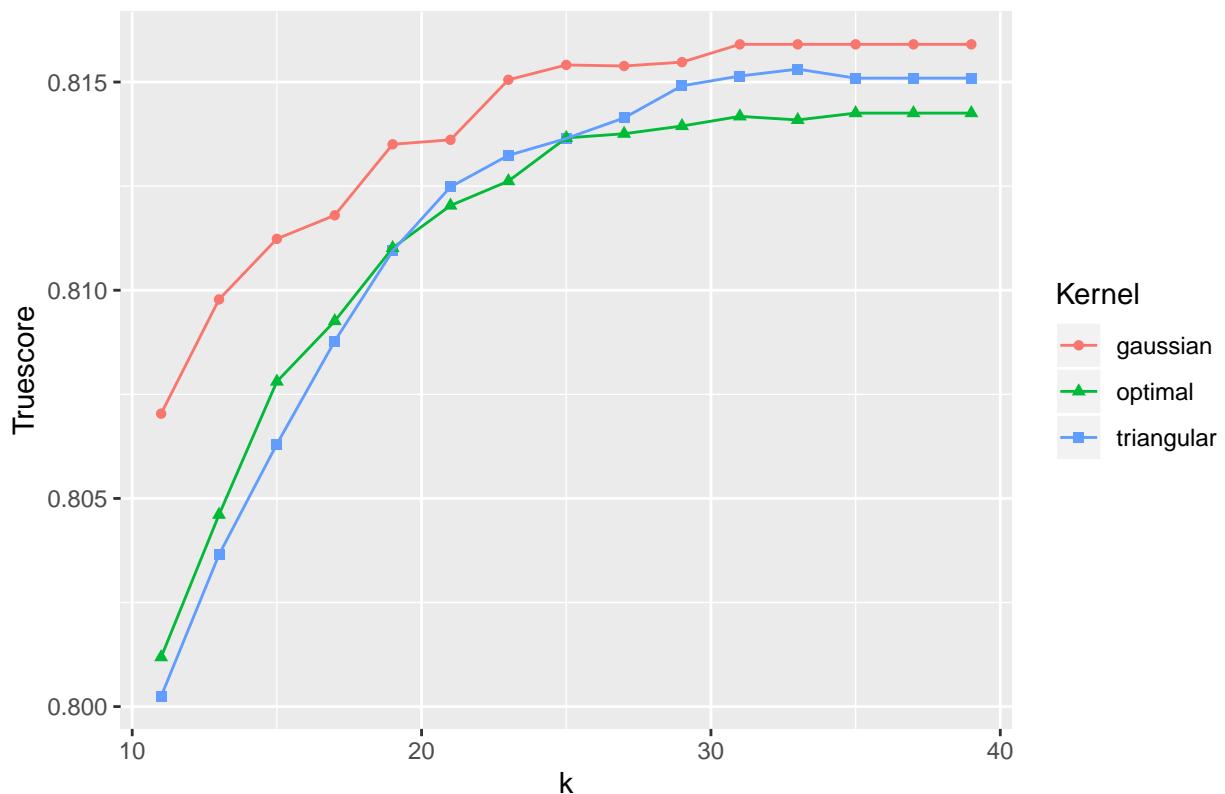
Table 45: c36_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.36	0.760643	0.844201	0.800247
11	gaussian	0.36	0.773143	0.844036	0.807036
11	optimal	0.36	0.761898	0.844746	0.801186
13	triangular	0.36	0.766867	0.844135	0.803648
13	gaussian	0.36	0.778363	0.843838	0.809779
13	optimal	0.36	0.768574	0.844185	0.804607
15	triangular	0.36	0.771386	0.844531	0.806303
15	gaussian	0.36	0.780873	0.844052	0.811234
15	optimal	0.36	0.774197	0.844465	0.807806
17	triangular	0.36	0.775803	0.844680	0.808778
17	gaussian	0.36	0.782631	0.843227	0.811800
17	optimal	0.36	0.776857	0.844482	0.809259
19	triangular	0.36	0.779869	0.844597	0.810943
19	gaussian	0.36	0.785793	0.843243	0.813505
19	optimal	0.36	0.780271	0.844284	0.811016
21	triangular	0.36	0.782380	0.844994	0.812482
21	gaussian	0.36	0.786596	0.842549	0.813612
21	optimal	0.36	0.782229	0.844201	0.812034
23	triangular	0.36	0.784739	0.843887	0.813239
23	gaussian	0.36	0.789558	0.842252	0.815054
23	optimal	0.36	0.783886	0.843540	0.812620

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.36	0.785894	0.843425	0.813644
25	gaussian	0.36	0.790562	0.841872	0.815411
25	optimal	0.36	0.786195	0.843111	0.813659
27	triangular	0.36	0.787149	0.843045	0.814139
27	gaussian	0.36	0.790763	0.841592	0.815386
27	optimal	0.36	0.786898	0.842516	0.813758
29	triangular	0.36	0.789157	0.842401	0.814910
29	gaussian	0.36	0.790863	0.841674	0.815478
29	optimal	0.36	0.787751	0.841938	0.813944
31	triangular	0.36	0.789960	0.841988	0.815145
31	gaussian	0.36	0.791918	0.841393	0.815906
31	optimal	0.36	0.788504	0.841575	0.814176
33	triangular	0.36	0.790462	0.841773	0.815311
33	gaussian	0.36	0.791918	0.841393	0.815906
33	optimal	0.36	0.788604	0.841278	0.814090
35	triangular	0.36	0.790361	0.841426	0.815094
35	gaussian	0.36	0.791918	0.841393	0.815906
35	optimal	0.36	0.788956	0.841228	0.814254
37	triangular	0.36	0.790361	0.841426	0.815094
37	gaussian	0.36	0.791918	0.841393	0.815906
37	optimal	0.36	0.788956	0.841228	0.814254
39	triangular	0.36	0.790361	0.841426	0.815094
39	gaussian	0.36	0.791918	0.841393	0.815906
39	optimal	0.36	0.788956	0.841228	0.814254

```
ggplot(c36_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.36 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.36 (Truescore)

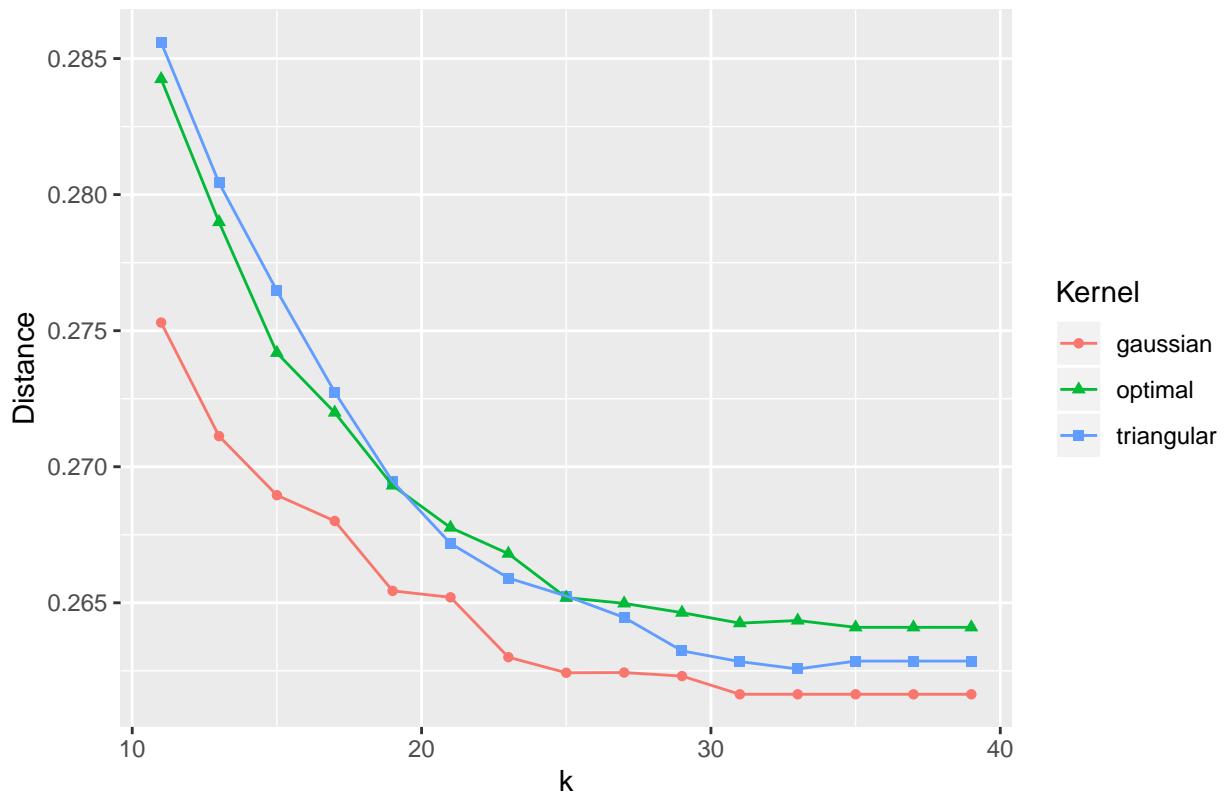


```
# For the Cutoff of 0.36, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c36_results <- c36_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c36_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.36 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.36 (Distance)



```
# For the Cutoff of 0.36, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c36_results$Truescore)
```

```
## [1] 0.815906
```

```
(c36_opt_k_ts <- c36_results$k[which.max(c36_results$Truescore)])
```

```
## [1] 31
```

```
(c36_opt_kernel_ts <- c36_results$Kernel[which.max(c36_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c36_opt_cut_ts <- c36_results$Cut[which.max(c36_results$Truescore)])
```

```
## [1] 0.36
```

```
(c36_opt_tpr_ts <- c36_results$TPR[which.max(c36_results$Truescore)])
```

```
## [1] 0.791918
```

```

(c36_opt_tnr_ts <- c36_results$TNR[which.max(c36_results$Truescore)])

## [1] 0.841393

(c36_opt_d_ts <- c36_results$Distance[which.max(c36_results$Truescore)])

## [1] 0.2616377

min(c36_results$Distance)

## [1] 0.2616377

(c36_opt_k_dist <- c36_results$k[which.min(c36_results$Distance)])

## [1] 31

(c36_opt_kernel_dist <- c36_results$Kernel[which.min(c36_results$Distance)])

## [1] "gaussian"

(c36_opt_cut_dist <- c36_results$Cut[which.min(c36_results$Distance)])

## [1] 0.36

(c36_opt_tpr_dist <- c36_results$TPR[which.min(c36_results$Distance)])

## [1] 0.791918

(c36_opt_tnr_dist <- c36_results$TNR[which.min(c36_results$Distance)])

## [1] 0.841393

(c36_opt_t_dist <- c36_results$Truescore[which.min(c36_results$Distance)])

## [1] 0.815906

#####
# 0.37 Cutoff
#####

# For the decision cutoff of 0.37, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c37 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred37, obs))
  confusionMatrix(ss$pred37, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c37) <- kwf_dfs_v

cm_c37_tables <- sapply(cm_c37, "[[", 2)
cm_c37_tables <- as_tibble(cm_c37_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.37.

k11w1c37_tpr <- round(sum(cm_c37_tables[1, 1:10]) /
  (sum(cm_c37_tables[1, 1:10]) + sum(cm_c37_tables[2, 1:10])), 6)
k11w1c37_tnr <- round(sum(cm_c37_tables[4, 1:10]) /
  (sum(cm_c37_tables[4, 1:10]) + sum(cm_c37_tables[3, 1:10])), 6)
k11w1c37_truescore <- round((2 * k11w1c37_tpr * k11w1c37_tnr) /
  (k11w1c37_tpr + k11w1c37_tnr), 6)

k11w2c37_tpr <- round(sum(cm_c37_tables[1, 11:20]) /
  (sum(cm_c37_tables[1, 11:20]) + sum(cm_c37_tables[2, 11:20])), 6)
k11w2c37_tnr <- round(sum(cm_c37_tables[4, 11:20]) /
  (sum(cm_c37_tables[4, 11:20]) + sum(cm_c37_tables[3, 11:20])), 6)
k11w2c37_truescore <- round((2 * k11w2c37_tpr * k11w2c37_tnr) /
  (k11w2c37_tpr + k11w2c37_tnr), 6)

k11w3c37_tpr <- round(sum(cm_c37_tables[1, 21:30]) /
  (sum(cm_c37_tables[1, 21:30]) + sum(cm_c37_tables[2, 21:30])), 6)
k11w3c37_tnr <- round(sum(cm_c37_tables[4, 21:30]) /
  (sum(cm_c37_tables[4, 21:30]) + sum(cm_c37_tables[3, 21:30])), 6)
k11w3c37_truescore <- round((2 * k11w3c37_tpr * k11w3c37_tnr) /
  (k11w3c37_tpr + k11w3c37_tnr), 6)

k13w1c37_tpr <- round(sum(cm_c37_tables[1, 31:40]) /
  (sum(cm_c37_tables[1, 31:40]) + sum(cm_c37_tables[2, 31:40])), 6)
k13w1c37_tnr <- round(sum(cm_c37_tables[4, 31:40]) /
  (sum(cm_c37_tables[4, 31:40]) + sum(cm_c37_tables[3, 31:40])), 6)
k13w1c37_truescore <- round((2 * k13w1c37_tpr * k13w1c37_tnr) /
  (k13w1c37_tpr + k13w1c37_tnr), 6)

k13w2c37_tpr <- round(sum(cm_c37_tables[1, 41:50]) /
  (sum(cm_c37_tables[1, 41:50]) + sum(cm_c37_tables[2, 41:50])), 6)
k13w2c37_tnr <- round(sum(cm_c37_tables[4, 41:50]) /
  (sum(cm_c37_tables[4, 41:50]) + sum(cm_c37_tables[3, 41:50])), 6)
k13w2c37_truescore <- round((2 * k13w2c37_tpr * k13w2c37_tnr) /
  (k13w2c37_tpr + k13w2c37_tnr), 6)

k13w3c37_tpr <- round(sum(cm_c37_tables[1, 51:60]) /
  (sum(cm_c37_tables[1, 51:60]) + sum(cm_c37_tables[2, 51:60])), 6)
k13w3c37_tnr <- round(sum(cm_c37_tables[4, 51:60]) /
  (sum(cm_c37_tables[4, 51:60]) + sum(cm_c37_tables[3, 51:60])), 6)
k13w3c37_truescore <- round((2 * k13w3c37_tpr * k13w3c37_tnr) /

```

```

(k13w3c37_tpr + k13w3c37_tnr), 6)

k15w1c37_tpr <- round(sum(cm_c37_tables[1, 61:70]) /
  (sum(cm_c37_tables[1, 61:70]) + sum(cm_c37_tables[2, 61:70])), 6)
k15w1c37_tnr <- round(sum(cm_c37_tables[4, 61:70]) /
  (sum(cm_c37_tables[4, 61:70]) + sum(cm_c37_tables[3, 61:70])), 6)
k15w1c37_truescore <- round((2 * k15w1c37_tpr * k15w1c37_tnr) /
  (k15w1c37_tpr + k15w1c37_tnr), 6)

k15w2c37_tpr <- round(sum(cm_c37_tables[1, 71:80]) /
  (sum(cm_c37_tables[1, 71:80]) + sum(cm_c37_tables[2, 71:80])), 6)
k15w2c37_tnr <- round(sum(cm_c37_tables[4, 71:80]) /
  (sum(cm_c37_tables[4, 71:80]) + sum(cm_c37_tables[3, 71:80])), 6)
k15w2c37_truescore <- round((2 * k15w2c37_tpr * k15w2c37_tnr) /
  (k15w2c37_tpr + k15w2c37_tnr), 6)

k15w3c37_tpr <- round(sum(cm_c37_tables[1, 81:90]) /
  (sum(cm_c37_tables[1, 81:90]) + sum(cm_c37_tables[2, 81:90])), 6)
k15w3c37_tnr <- round(sum(cm_c37_tables[4, 81:90]) /
  (sum(cm_c37_tables[4, 81:90]) + sum(cm_c37_tables[3, 81:90])), 6)
k15w3c37_truescore <- round((2 * k15w3c37_tpr * k15w3c37_tnr) /
  (k15w3c37_tpr + k15w3c37_tnr), 6)

k17w1c37_tpr <- round(sum(cm_c37_tables[1, 91:100]) /
  (sum(cm_c37_tables[1, 91:100]) + sum(cm_c37_tables[2, 91:100])), 6)
k17w1c37_tnr <- round(sum(cm_c37_tables[4, 91:100]) /
  (sum(cm_c37_tables[4, 91:100]) + sum(cm_c37_tables[3, 91:100])), 6)
k17w1c37_truescore <- round((2 * k17w1c37_tpr * k17w1c37_tnr) /
  (k17w1c37_tpr + k17w1c37_tnr), 6)

k17w2c37_tpr <- round(sum(cm_c37_tables[1, 101:110]) /
  (sum(cm_c37_tables[1, 101:110]) + sum(cm_c37_tables[2, 101:110])), 6)
k17w2c37_tnr <- round(sum(cm_c37_tables[4, 101:110]) /
  (sum(cm_c37_tables[4, 101:110]) + sum(cm_c37_tables[3, 101:110])), 6)
k17w2c37_truescore <- round((2 * k17w2c37_tpr * k17w2c37_tnr) /
  (k17w2c37_tpr + k17w2c37_tnr), 6)

k17w3c37_tpr <- round(sum(cm_c37_tables[1, 111:120]) /
  (sum(cm_c37_tables[1, 111:120]) + sum(cm_c37_tables[2, 111:120])), 6)
k17w3c37_tnr <- round(sum(cm_c37_tables[4, 111:120]) /
  (sum(cm_c37_tables[4, 111:120]) + sum(cm_c37_tables[3, 111:120])), 6)
k17w3c37_truescore <- round((2 * k17w3c37_tpr * k17w3c37_tnr) /
  (k17w3c37_tpr + k17w3c37_tnr), 6)

k19w1c37_tpr <- round(sum(cm_c37_tables[1, 121:130]) /
  (sum(cm_c37_tables[1, 121:130]) + sum(cm_c37_tables[2, 121:130])), 6)
k19w1c37_tnr <- round(sum(cm_c37_tables[4, 121:130]) /
  (sum(cm_c37_tables[4, 121:130]) + sum(cm_c37_tables[3, 121:130])), 6)
k19w1c37_truescore <- round((2 * k19w1c37_tpr * k19w1c37_tnr) /
  (k19w1c37_tpr + k19w1c37_tnr), 6)

```

```

k19w2c37_tpr <- round(sum(cm_c37_tables[1, 131:140]) /
  (sum(cm_c37_tables[1, 131:140]) + sum(cm_c37_tables[2, 131:140])), 6)
k19w2c37_tnr <- round(sum(cm_c37_tables[4, 131:140]) /
  (sum(cm_c37_tables[4, 131:140]) + sum(cm_c37_tables[3, 131:140])), 6)
k19w2c37_truescore <- round((2 * k19w2c37_tpr * k19w2c37_tnr) /
  (k19w2c37_tpr + k19w2c37_tnr), 6)

k19w3c37_tpr <- round(sum(cm_c37_tables[1, 141:150]) /
  (sum(cm_c37_tables[1, 141:150]) + sum(cm_c37_tables[2, 141:150])), 6)
k19w3c37_tnr <- round(sum(cm_c37_tables[4, 141:150]) /
  (sum(cm_c37_tables[4, 141:150]) + sum(cm_c37_tables[3, 141:150])), 6)
k19w3c37_truescore <- round((2 * k19w3c37_tpr * k19w3c37_tnr) /
  (k19w3c37_tpr + k19w3c37_tnr), 6)

k21w1c37_tpr <- round(sum(cm_c37_tables[1, 151:160]) /
  (sum(cm_c37_tables[1, 151:160]) + sum(cm_c37_tables[2, 151:160])), 6)
k21w1c37_tnr <- round(sum(cm_c37_tables[4, 151:160]) /
  (sum(cm_c37_tables[4, 151:160]) + sum(cm_c37_tables[3, 151:160])), 6)
k21w1c37_truescore <- round((2 * k21w1c37_tpr * k21w1c37_tnr) /
  (k21w1c37_tpr + k21w1c37_tnr), 6)

k21w2c37_tpr <- round(sum(cm_c37_tables[1, 161:170]) /
  (sum(cm_c37_tables[1, 161:170]) + sum(cm_c37_tables[2, 161:170])), 6)
k21w2c37_tnr <- round(sum(cm_c37_tables[4, 161:170]) /
  (sum(cm_c37_tables[4, 161:170]) + sum(cm_c37_tables[3, 161:170])), 6)
k21w2c37_truescore <- round((2 * k21w2c37_tpr * k21w2c37_tnr) /
  (k21w2c37_tpr + k21w2c37_tnr), 6)

k21w3c37_tpr <- round(sum(cm_c37_tables[1, 171:180]) /
  (sum(cm_c37_tables[1, 171:180]) + sum(cm_c37_tables[2, 171:180])), 6)
k21w3c37_tnr <- round(sum(cm_c37_tables[4, 171:180]) /
  (sum(cm_c37_tables[4, 171:180]) + sum(cm_c37_tables[3, 171:180])), 6)
k21w3c37_truescore <- round((2 * k21w3c37_tpr * k21w3c37_tnr) /
  (k21w3c37_tpr + k21w3c37_tnr), 6)

k23w1c37_tpr <- round(sum(cm_c37_tables[1, 181:190]) /
  (sum(cm_c37_tables[1, 181:190]) + sum(cm_c37_tables[2, 181:190])), 6)
k23w1c37_tnr <- round(sum(cm_c37_tables[4, 181:190]) /
  (sum(cm_c37_tables[4, 181:190]) + sum(cm_c37_tables[3, 181:190])), 6)
k23w1c37_truescore <- round((2 * k23w1c37_tpr * k23w1c37_tnr) /
  (k23w1c37_tpr + k23w1c37_tnr), 6)

k23w2c37_tpr <- round(sum(cm_c37_tables[1, 191:200]) /
  (sum(cm_c37_tables[1, 191:200]) + sum(cm_c37_tables[2, 191:200])), 6)
k23w2c37_tnr <- round(sum(cm_c37_tables[4, 191:200]) /
  (sum(cm_c37_tables[4, 191:200]) + sum(cm_c37_tables[3, 191:200])), 6)
k23w2c37_truescore <- round((2 * k23w2c37_tpr * k23w2c37_tnr) /
  (k23w2c37_tpr + k23w2c37_tnr), 6)

k23w3c37_tpr <- round(sum(cm_c37_tables[1, 201:210]) /
  (sum(cm_c37_tables[1, 201:210]) + sum(cm_c37_tables[2, 201:210])), 6)

```

```

k23w3c37_tnr <- round(sum(cm_c37_tables[4, 201:210]) /
  (sum(cm_c37_tables[4, 201:210]) + sum(cm_c37_tables[3, 201:210])), 6)
k23w3c37_truescore <- round((2 * k23w3c37_tpr * k23w3c37_tnr) /
  (k23w3c37_tpr + k23w3c37_tnr), 6)

k25w1c37_tpr <- round(sum(cm_c37_tables[1, 211:220]) /
  (sum(cm_c37_tables[1, 211:220]) + sum(cm_c37_tables[2, 211:220])), 6)
k25w1c37_tnr <- round(sum(cm_c37_tables[4, 211:220]) /
  (sum(cm_c37_tables[4, 211:220]) + sum(cm_c37_tables[3, 211:220])), 6)
k25w1c37_truescore <- round((2 * k25w1c37_tpr * k25w1c37_tnr) /
  (k25w1c37_tpr + k25w1c37_tnr), 6)

k25w2c37_tpr <- round(sum(cm_c37_tables[1, 221:230]) /
  (sum(cm_c37_tables[1, 221:230]) + sum(cm_c37_tables[2, 221:230])), 6)
k25w2c37_tnr <- round(sum(cm_c37_tables[4, 221:230]) /
  (sum(cm_c37_tables[4, 221:230]) + sum(cm_c37_tables[3, 221:230])), 6)
k25w2c37_truescore <- round((2 * k25w2c37_tpr * k25w2c37_tnr) /
  (k25w2c37_tpr + k25w2c37_tnr), 6)

k25w3c37_tpr <- round(sum(cm_c37_tables[1, 231:240]) /
  (sum(cm_c37_tables[1, 231:240]) + sum(cm_c37_tables[2, 231:240])), 6)
k25w3c37_tnr <- round(sum(cm_c37_tables[4, 231:240]) /
  (sum(cm_c37_tables[4, 231:240]) + sum(cm_c37_tables[3, 231:240])), 6)
k25w3c37_truescore <- round((2 * k25w3c37_tpr * k25w3c37_tnr) /
  (k25w3c37_tpr + k25w3c37_tnr), 6)

k27w1c37_tpr <- round(sum(cm_c37_tables[1, 241:250]) /
  (sum(cm_c37_tables[1, 241:250]) + sum(cm_c37_tables[2, 241:250])), 6)
k27w1c37_tnr <- round(sum(cm_c37_tables[4, 241:250]) /
  (sum(cm_c37_tables[4, 241:250]) + sum(cm_c37_tables[3, 241:250])), 6)
k27w1c37_truescore <- round((2 * k27w1c37_tpr * k27w1c37_tnr) /
  (k27w1c37_tpr + k27w1c37_tnr), 6)

k27w2c37_tpr <- round(sum(cm_c37_tables[1, 251:260]) /
  (sum(cm_c37_tables[1, 251:260]) + sum(cm_c37_tables[2, 251:260])), 6)
k27w2c37_tnr <- round(sum(cm_c37_tables[4, 251:260]) /
  (sum(cm_c37_tables[4, 251:260]) + sum(cm_c37_tables[3, 251:260])), 6)
k27w2c37_truescore <- round((2 * k27w2c37_tpr * k27w2c37_tnr) /
  (k27w2c37_tpr + k27w2c37_tnr), 6)

k27w3c37_tpr <- round(sum(cm_c37_tables[1, 261:270]) /
  (sum(cm_c37_tables[1, 261:270]) + sum(cm_c37_tables[2, 261:270])), 6)
k27w3c37_tnr <- round(sum(cm_c37_tables[4, 261:270]) /
  (sum(cm_c37_tables[4, 261:270]) + sum(cm_c37_tables[3, 261:270])), 6)
k27w3c37_truescore <- round((2 * k27w3c37_tpr * k27w3c37_tnr) /
  (k27w3c37_tpr + k27w3c37_tnr), 6)

k29w1c37_tpr <- round(sum(cm_c37_tables[1, 271:280]) /
  (sum(cm_c37_tables[1, 271:280]) + sum(cm_c37_tables[2, 271:280])), 6)
k29w1c37_tnr <- round(sum(cm_c37_tables[4, 271:280]) /

```

```

(sum(cm_c37_tables[4, 271:280]) + sum(cm_c37_tables[3, 271:280])), 6)
k29w1c37_truescore <- round((2 * k29w1c37_tpr * k29w1c37_tnr) /
(k29w1c37_tpr + k29w1c37_tnr), 6)

k29w2c37_tpr <- round(sum(cm_c37_tables[1, 281:290]) /
(sum(cm_c37_tables[1, 281:290]) + sum(cm_c37_tables[2, 281:290])), 6)
k29w2c37_tnr <- round(sum(cm_c37_tables[4, 281:290]) /
(sum(cm_c37_tables[4, 281:290]) + sum(cm_c37_tables[3, 281:290])), 6)
k29w2c37_truescore <- round((2 * k29w2c37_tpr * k29w2c37_tnr) /
(k29w2c37_tpr + k29w2c37_tnr), 6)

k29w3c37_tpr <- round(sum(cm_c37_tables[1, 291:300]) /
(sum(cm_c37_tables[1, 291:300]) + sum(cm_c37_tables[2, 291:300])), 6)
k29w3c37_tnr <- round(sum(cm_c37_tables[4, 291:300]) /
(sum(cm_c37_tables[4, 291:300]) + sum(cm_c37_tables[3, 291:300])), 6)
k29w3c37_truescore <- round((2 * k29w3c37_tpr * k29w3c37_tnr) /
(k29w3c37_tpr + k29w3c37_tnr), 6)

k31w1c37_tpr <- round(sum(cm_c37_tables[1, 301:310]) /
(sum(cm_c37_tables[1, 301:310]) + sum(cm_c37_tables[2, 301:310])), 6)
k31w1c37_tnr <- round(sum(cm_c37_tables[4, 301:310]) /
(sum(cm_c37_tables[4, 301:310]) + sum(cm_c37_tables[3, 301:310])), 6)
k31w1c37_truescore <- round((2 * k31w1c37_tpr * k31w1c37_tnr) /
(k31w1c37_tpr + k31w1c37_tnr), 6)

k31w2c37_tpr <- round(sum(cm_c37_tables[1, 311:320]) /
(sum(cm_c37_tables[1, 311:320]) + sum(cm_c37_tables[2, 311:320])), 6)
k31w2c37_tnr <- round(sum(cm_c37_tables[4, 311:320]) /
(sum(cm_c37_tables[4, 311:320]) + sum(cm_c37_tables[3, 311:320])), 6)
k31w2c37_truescore <- round((2 * k31w2c37_tpr * k31w2c37_tnr) /
(k31w2c37_tpr + k31w2c37_tnr), 6)

k31w3c37_tpr <- round(sum(cm_c37_tables[1, 321:330]) /
(sum(cm_c37_tables[1, 321:330]) + sum(cm_c37_tables[2, 321:330])), 6)
k31w3c37_tnr <- round(sum(cm_c37_tables[4, 321:330]) /
(sum(cm_c37_tables[4, 321:330]) + sum(cm_c37_tables[3, 321:330])), 6)
k31w3c37_truescore <- round((2 * k31w3c37_tpr * k31w3c37_tnr) /
(k31w3c37_tpr + k31w3c37_tnr), 6)

k33w1c37_tpr <- round(sum(cm_c37_tables[1, 331:340]) /
(sum(cm_c37_tables[1, 331:340]) + sum(cm_c37_tables[2, 331:340])), 6)
k33w1c37_tnr <- round(sum(cm_c37_tables[4, 331:340]) /
(sum(cm_c37_tables[4, 331:340]) + sum(cm_c37_tables[3, 331:340])), 6)
k33w1c37_truescore <- round((2 * k33w1c37_tpr * k33w1c37_tnr) /
(k33w1c37_tpr + k33w1c37_tnr), 6)

k33w2c37_tpr <- round(sum(cm_c37_tables[1, 341:350]) /
(sum(cm_c37_tables[1, 341:350]) + sum(cm_c37_tables[2, 341:350])), 6)
k33w2c37_tnr <- round(sum(cm_c37_tables[4, 341:350]) /
(sum(cm_c37_tables[4, 341:350]) + sum(cm_c37_tables[3, 341:350])), 6)
k33w2c37_truescore <- round((2 * k33w2c37_tpr * k33w2c37_tnr) /

```

```

(k33w2c37_tpr + k33w2c37_tnr), 6)

k33w3c37_tpr <- round(sum(cm_c37_tables[1, 351:360]) /
  (sum(cm_c37_tables[1, 351:360]) + sum(cm_c37_tables[2, 351:360])), 6)
k33w3c37_tnr <- round(sum(cm_c37_tables[4, 351:360]) /
  (sum(cm_c37_tables[4, 351:360]) + sum(cm_c37_tables[3, 351:360])), 6)
k33w3c37_truescore <- round((2 * k33w3c37_tpr * k33w3c37_tnr) /
  (k33w3c37_tpr + k33w3c37_tnr), 6)

k35w1c37_tpr <- round(sum(cm_c37_tables[1, 361:370]) /
  (sum(cm_c37_tables[1, 361:370]) + sum(cm_c37_tables[2, 361:370])), 6)
k35w1c37_tnr <- round(sum(cm_c37_tables[4, 361:370]) /
  (sum(cm_c37_tables[4, 361:370]) + sum(cm_c37_tables[3, 361:370])), 6)
k35w1c37_truescore <- round((2 * k35w1c37_tpr * k35w1c37_tnr) /
  (k35w1c37_tpr + k35w1c37_tnr), 6)

k35w2c37_tpr <- round(sum(cm_c37_tables[1, 371:380]) /
  (sum(cm_c37_tables[1, 371:380]) + sum(cm_c37_tables[2, 371:380])), 6)
k35w2c37_tnr <- round(sum(cm_c37_tables[4, 371:380]) /
  (sum(cm_c37_tables[4, 371:380]) + sum(cm_c37_tables[3, 371:380])), 6)
k35w2c37_truescore <- round((2 * k35w2c37_tpr * k35w2c37_tnr) /
  (k35w2c37_tpr + k35w2c37_tnr), 6)

k35w3c37_tpr <- round(sum(cm_c37_tables[1, 381:390]) /
  (sum(cm_c37_tables[1, 381:390]) + sum(cm_c37_tables[2, 381:390])), 6)
k35w3c37_tnr <- round(sum(cm_c37_tables[4, 381:390]) /
  (sum(cm_c37_tables[4, 381:390]) + sum(cm_c37_tables[3, 381:390])), 6)
k35w3c37_truescore <- round((2 * k35w3c37_tpr * k35w3c37_tnr) /
  (k35w3c37_tpr + k35w3c37_tnr), 6)

k37w1c37_tpr <- round(sum(cm_c37_tables[1, 391:400]) /
  (sum(cm_c37_tables[1, 391:400]) + sum(cm_c37_tables[2, 391:400])), 6)
k37w1c37_tnr <- round(sum(cm_c37_tables[4, 391:400]) /
  (sum(cm_c37_tables[4, 391:400]) + sum(cm_c37_tables[3, 391:400])), 6)
k37w1c37_truescore <- round((2 * k37w1c37_tpr * k37w1c37_tnr) /
  (k37w1c37_tpr + k37w1c37_tnr), 6)

k37w2c37_tpr <- round(sum(cm_c37_tables[1, 401:410]) /
  (sum(cm_c37_tables[1, 401:410]) + sum(cm_c37_tables[2, 401:410])), 6)
k37w2c37_tnr <- round(sum(cm_c37_tables[4, 401:410]) /
  (sum(cm_c37_tables[4, 401:410]) + sum(cm_c37_tables[3, 401:410])), 6)
k37w2c37_truescore <- round((2 * k37w2c37_tpr * k37w2c37_tnr) /
  (k37w2c37_tpr + k37w2c37_tnr), 6)

k37w3c37_tpr <- round(sum(cm_c37_tables[1, 411:420]) /
  (sum(cm_c37_tables[1, 411:420]) + sum(cm_c37_tables[2, 411:420])), 6)
k37w3c37_tnr <- round(sum(cm_c37_tables[4, 411:420]) /
  (sum(cm_c37_tables[4, 411:420]) + sum(cm_c37_tables[3, 411:420])), 6)
k37w3c37_truescore <- round((2 * k37w3c37_tpr * k37w3c37_tnr) /
  (k37w3c37_tpr + k37w3c37_tnr), 6)

```

```

k39w1c37_tpr <- round(sum(cm_c37_tables[1, 421:430]) /
  (sum(cm_c37_tables[1, 421:430]) + sum(cm_c37_tables[2, 421:430])), 6)
k39w1c37_tnr <- round(sum(cm_c37_tables[4, 421:430]) /
  (sum(cm_c37_tables[4, 421:430]) + sum(cm_c37_tables[3, 421:430])), 6)
k39w1c37_truescore <- round((2 * k39w1c37_tpr * k39w1c37_tnr) /
  (k39w1c37_tpr + k39w1c37_tnr), 6)

k39w2c37_tpr <- round(sum(cm_c37_tables[1, 431:440]) /
  (sum(cm_c37_tables[1, 431:440]) + sum(cm_c37_tables[2, 431:440])), 6)
k39w2c37_tnr <- round(sum(cm_c37_tables[4, 431:440]) /
  (sum(cm_c37_tables[4, 431:440]) + sum(cm_c37_tables[3, 431:440])), 6)
k39w2c37_truescore <- round((2 * k39w2c37_tpr * k39w2c37_tnr) /
  (k39w2c37_tpr + k39w2c37_tnr), 6)

k39w3c37_tpr <- round(sum(cm_c37_tables[1, 441:450]) /
  (sum(cm_c37_tables[1, 441:450]) + sum(cm_c37_tables[2, 441:450])), 6)
k39w3c37_tnr <- round(sum(cm_c37_tables[4, 441:450]) /
  (sum(cm_c37_tables[4, 441:450]) + sum(cm_c37_tables[3, 441:450])), 6)
k39w3c37_truescore <- round((2 * k39w3c37_tpr * k39w3c37_tnr) /
  (k39w3c37_tpr + k39w3c37_tnr), 6)

# Compile the 0.37 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c37_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.37,
  TPR = c(k11w1c37_tpr, k11w2c37_tpr, k11w3c37_tpr, k13w1c37_tpr,
    k13w2c37_tpr, k13w3c37_tpr, k15w1c37_tpr, k15w2c37_tpr,
    k15w3c37_tpr, k17w1c37_tpr, k17w2c37_tpr, k17w3c37_tpr,
    k19w1c37_tpr, k19w2c37_tpr, k19w3c37_tpr, k21w1c37_tpr,
    k21w2c37_tpr, k21w3c37_tpr, k23w1c37_tpr, k23w2c37_tpr,
    k23w3c37_tpr, k25w1c37_tpr, k25w2c37_tpr, k25w3c37_tpr,
    k27w1c37_tpr, k27w2c37_tpr, k27w3c37_tpr, k29w1c37_tpr,
    k29w2c37_tpr, k29w3c37_tpr, k31w1c37_tpr, k31w2c37_tpr,
    k31w3c37_tpr, k33w1c37_tpr, k33w2c37_tpr, k33w3c37_tpr,
    k35w1c37_tpr, k35w2c37_tpr, k35w3c37_tpr, k37w1c37_tpr,
    k37w2c37_tpr, k37w3c37_tpr, k39w1c37_tpr, k39w2c37_tpr,
    k39w3c37_tpr),
  TNR = c(k11w1c37_tnr, k11w2c37_tnr, k11w3c37_tnr, k13w1c37_tnr,
    k13w2c37_tnr, k13w3c37_tnr, k15w1c37_tnr, k15w2c37_tnr,
    k15w3c37_tnr, k17w1c37_tnr, k17w2c37_tnr, k17w3c37_tnr,
    k19w1c37_tnr, k19w2c37_tnr, k19w3c37_tnr, k21w1c37_tnr,
    k21w2c37_tnr, k21w3c37_tnr, k23w1c37_tnr, k23w2c37_tnr,
    k23w3c37_tnr, k25w1c37_tnr, k25w2c37_tnr, k25w3c37_tnr,
    k27w1c37_tnr, k27w2c37_tnr, k27w3c37_tnr, k29w1c37_tnr,
    k29w2c37_tnr, k29w3c37_tnr, k31w1c37_tnr, k31w2c37_tnr,
    k31w3c37_tnr, k33w1c37_tnr, k33w2c37_tnr, k33w3c37_tnr),
  Truescore = k39w1c37_truescore)

```

```

k35w1c37_tnr, k35w2c37_tnr, k35w3c37_tnr, k37w1c37_tnr,
k37w2c37_tnr, k37w3c37_tnr, k39w1c37_tnr, k39w2c37_tnr,
k39w3c37_tnr),
Truescore = c(k11w1c37_truescore, k11w2c37_truescore,
k11w3c37_truescore, k13w1c37_truescore,
k13w2c37_truescore, k13w3c37_truescore,
k15w1c37_truescore, k15w2c37_truescore,
k15w3c37_truescore, k17w1c37_truescore,
k17w2c37_truescore, k17w3c37_truescore,
k19w1c37_truescore, k19w2c37_truescore,
k19w3c37_truescore, k21w1c37_truescore,
k21w2c37_truescore, k21w3c37_truescore,
k23w1c37_truescore, k23w2c37_truescore,
k23w3c37_truescore, k25w1c37_truescore,
k25w2c37_truescore, k25w3c37_truescore,
k27w1c37_truescore, k27w2c37_truescore,
k27w3c37_truescore, k29w1c37_truescore,
k29w2c37_truescore, k29w3c37_truescore,
k31w1c37_truescore, k31w2c37_truescore,
k31w3c37_truescore, k33w1c37_truescore,
k33w2c37_truescore, k33w3c37_truescore,
k35w1c37_truescore, k35w2c37_truescore,
k35w3c37_truescore, k37w1c37_truescore,
k37w2c37_truescore, k37w3c37_truescore,
k39w1c37_truescore, k39w2c37_truescore,
k39w3c37_truescore))

```

`knitr::kable(c37_results[1:45,], caption = "c37_results")`

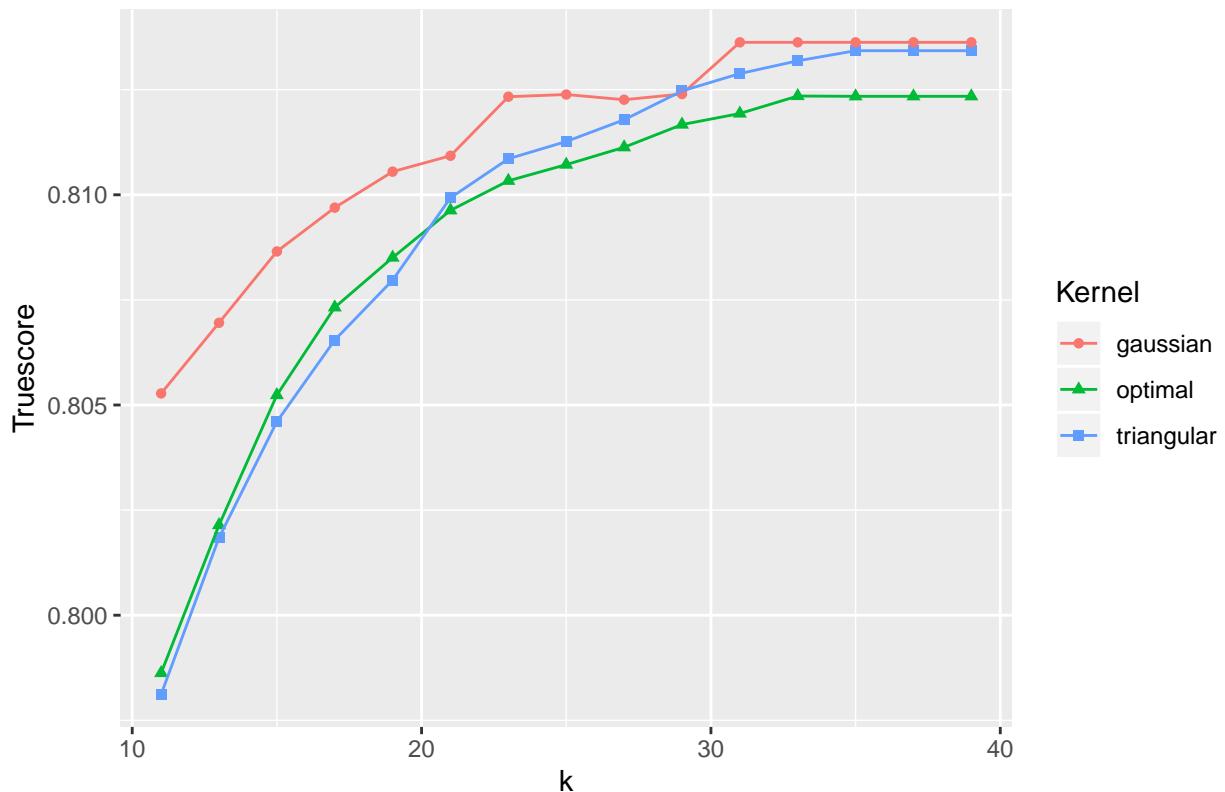
Table 46: c37_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.37	0.752661	0.849420	0.798119
11	gaussian	0.37	0.765010	0.850015	0.805275
11	optimal	0.37	0.753263	0.849800	0.798625
13	triangular	0.37	0.758785	0.850081	0.801843
13	gaussian	0.37	0.768223	0.849800	0.806955
13	optimal	0.37	0.759337	0.850064	0.802143
15	triangular	0.37	0.763253	0.850692	0.804604
15	gaussian	0.37	0.771034	0.850130	0.808652
15	optimal	0.37	0.764458	0.850609	0.805236
17	triangular	0.37	0.766767	0.850676	0.806545
17	gaussian	0.37	0.773394	0.849569	0.809694
17	optimal	0.37	0.767972	0.850923	0.807322
19	triangular	0.37	0.769578	0.850395	0.807971
19	gaussian	0.37	0.775301	0.849156	0.810550
19	optimal	0.37	0.770482	0.850477	0.808506
21	triangular	0.37	0.773042	0.850510	0.809928
21	gaussian	0.37	0.776104	0.849024	0.810928
21	optimal	0.37	0.772691	0.850279	0.809630
23	triangular	0.37	0.775201	0.849949	0.810856
23	gaussian	0.37	0.778916	0.848743	0.812332
23	optimal	0.37	0.774096	0.850130	0.810333

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.37	0.776104	0.849767	0.811267
25	gaussian	0.37	0.779418	0.848264	0.812385
25	optimal	0.37	0.775552	0.849222	0.810717
27	triangular	0.37	0.777410	0.849338	0.811784
27	gaussian	0.37	0.779468	0.847934	0.812261
27	optimal	0.37	0.776556	0.848925	0.811130
29	triangular	0.37	0.779016	0.848925	0.812469
29	gaussian	0.37	0.779468	0.848231	0.812397
29	optimal	0.37	0.777811	0.848611	0.811670
31	triangular	0.37	0.780070	0.848578	0.812883
31	gaussian	0.37	0.782028	0.847884	0.813626
31	optimal	0.37	0.778614	0.848231	0.811933
33	triangular	0.37	0.780823	0.848347	0.813186
33	gaussian	0.37	0.782028	0.847884	0.813626
33	optimal	0.37	0.779518	0.848066	0.812349
35	triangular	0.37	0.781325	0.848281	0.813427
35	gaussian	0.37	0.782028	0.847884	0.813626
35	optimal	0.37	0.779618	0.847934	0.812342
37	triangular	0.37	0.781325	0.848281	0.813427
37	gaussian	0.37	0.782028	0.847884	0.813626
37	optimal	0.37	0.779618	0.847934	0.812342
39	triangular	0.37	0.781325	0.848281	0.813427
39	gaussian	0.37	0.782028	0.847884	0.813626
39	optimal	0.37	0.779618	0.847934	0.812342

```
ggplot(c37_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.37 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.37 (Truescore)

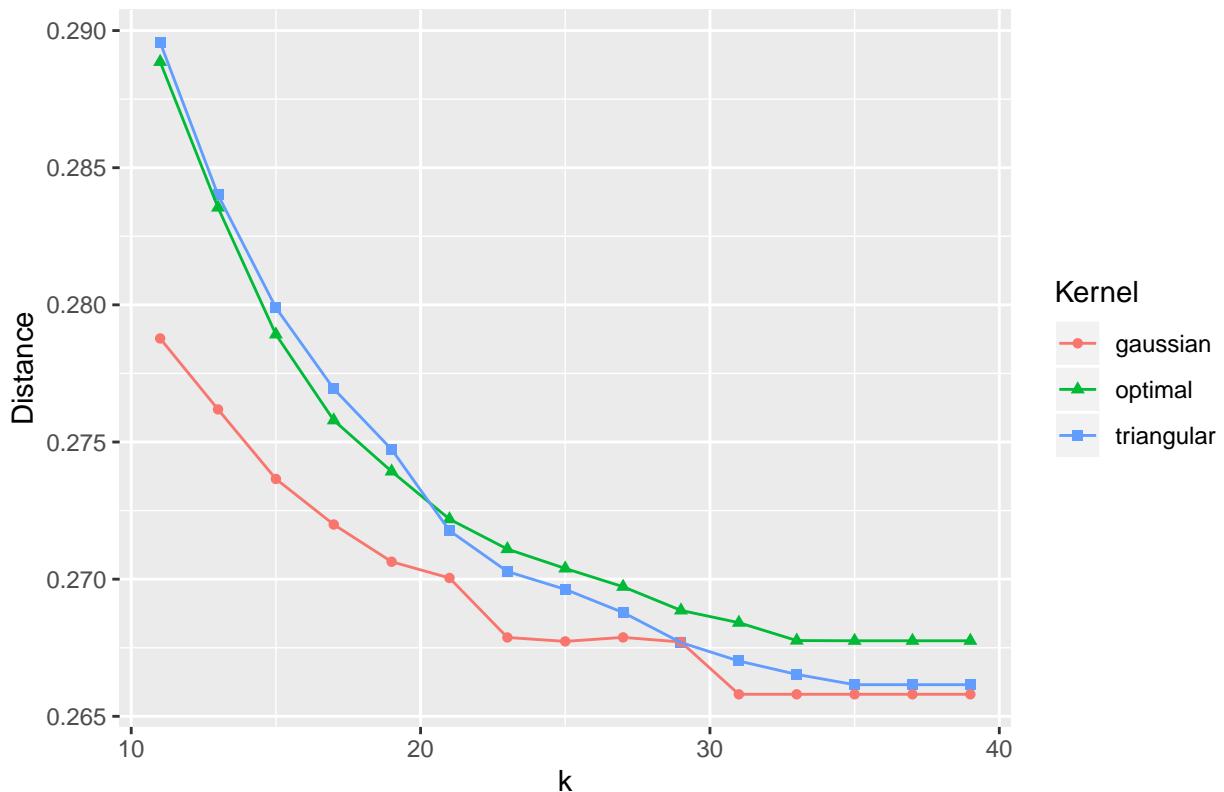


```
# For the Cutoff of 0.37, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c37_results <- c37_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c37_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.37 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.37 (Distance)



```
# For the Cutoff of 0.37, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c37_results$Truescore)
```

```
## [1] 0.813626
```

```
(c37_opt_k_ts <- c37_results$k[which.max(c37_results$Truescore)])
```

```
## [1] 31
```

```
(c37_opt_kernel_ts <- c37_results$Kernel[which.max(c37_results$Truescore)])
```

```
## [1] "gaussian"
```

```
(c37_opt_cut_ts <- c37_results$Cut[which.max(c37_results$Truescore)])
```

```
## [1] 0.37
```

```
(c37_opt_tpr_ts <- c37_results$TPR[which.max(c37_results$Truescore)])
```

```
## [1] 0.782028
```

```

(c37_opt_tnr_ts <- c37_results$TNR[which.max(c37_results$Truescore)])

## [1] 0.847884

(c37_opt_d_ts <- c37_results$Distance[which.max(c37_results$Truescore)])

## [1] 0.2658027

min(c37_results$Distance)

## [1] 0.2658027

(c37_opt_k_dist <- c37_results$k[which.min(c37_results$Distance)])

## [1] 31

(c37_opt_kernel_dist <- c37_results$Kernel[which.min(c37_results$Distance)])

## [1] "gaussian"

(c37_opt_cut_dist <- c37_results$Cut[which.min(c37_results$Distance)])

## [1] 0.37

(c37_opt_tpr_dist <- c37_results$TPR[which.min(c37_results$Distance)])

## [1] 0.782028

(c37_opt_tnr_dist <- c37_results$TNR[which.min(c37_results$Distance)])

## [1] 0.847884

(c37_opt_t_dist <- c37_results$Truescore[which.min(c37_results$Distance)])

## [1] 0.813626

#####
# 0.38 Cutoff
#####

# For the decision cutoff of 0.38, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c38 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred38, obs))
  confusionMatrix(ss$pred38, ss$obs)
}

```

```

}, simplify = FALSE, USE.NAMES = TRUE)

names(cm_c38) <- kwf_dfs_v

cm_c38_tables <- sapply(cm_c38, "[[", 2)
cm_c38_tables <- as_tibble(cm_c38_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.38.

k11w1c38_tpr <- round(sum(cm_c38_tables[1, 1:10]) /
  (sum(cm_c38_tables[1, 1:10]) + sum(cm_c38_tables[2, 1:10])), 6)
k11w1c38_tnr <- round(sum(cm_c38_tables[4, 1:10]) /
  (sum(cm_c38_tables[4, 1:10]) + sum(cm_c38_tables[3, 1:10])), 6)
k11w1c38_truescore <- round((2 * k11w1c38_tpr * k11w1c38_tnr) /
  (k11w1c38_tpr + k11w1c38_tnr), 6)

k11w2c38_tpr <- round(sum(cm_c38_tables[1, 11:20]) /
  (sum(cm_c38_tables[1, 11:20]) + sum(cm_c38_tables[2, 11:20])), 6)
k11w2c38_tnr <- round(sum(cm_c38_tables[4, 11:20]) /
  (sum(cm_c38_tables[4, 11:20]) + sum(cm_c38_tables[3, 11:20])), 6)
k11w2c38_truescore <- round((2 * k11w2c38_tpr * k11w2c38_tnr) /
  (k11w2c38_tpr + k11w2c38_tnr), 6)

k11w3c38_tpr <- round(sum(cm_c38_tables[1, 21:30]) /
  (sum(cm_c38_tables[1, 21:30]) + sum(cm_c38_tables[2, 21:30])), 6)
k11w3c38_tnr <- round(sum(cm_c38_tables[4, 21:30]) /
  (sum(cm_c38_tables[4, 21:30]) + sum(cm_c38_tables[3, 21:30])), 6)
k11w3c38_truescore <- round((2 * k11w3c38_tpr * k11w3c38_tnr) /
  (k11w3c38_tpr + k11w3c38_tnr), 6)

k13w1c38_tpr <- round(sum(cm_c38_tables[1, 31:40]) /
  (sum(cm_c38_tables[1, 31:40]) + sum(cm_c38_tables[2, 31:40])), 6)
k13w1c38_tnr <- round(sum(cm_c38_tables[4, 31:40]) /
  (sum(cm_c38_tables[4, 31:40]) + sum(cm_c38_tables[3, 31:40])), 6)
k13w1c38_truescore <- round((2 * k13w1c38_tpr * k13w1c38_tnr) /
  (k13w1c38_tpr + k13w1c38_tnr), 6)

k13w2c38_tpr <- round(sum(cm_c38_tables[1, 41:50]) /
  (sum(cm_c38_tables[1, 41:50]) + sum(cm_c38_tables[2, 41:50])), 6)
k13w2c38_tnr <- round(sum(cm_c38_tables[4, 41:50]) /
  (sum(cm_c38_tables[4, 41:50]) + sum(cm_c38_tables[3, 41:50])), 6)
k13w2c38_truescore <- round((2 * k13w2c38_tpr * k13w2c38_tnr) /
  (k13w2c38_tpr + k13w2c38_tnr), 6)

k13w3c38_tpr <- round(sum(cm_c38_tables[1, 51:60]) /
  (sum(cm_c38_tables[1, 51:60]) + sum(cm_c38_tables[2, 51:60])), 6)
k13w3c38_tnr <- round(sum(cm_c38_tables[4, 51:60]) /
  (sum(cm_c38_tables[4, 51:60]) + sum(cm_c38_tables[3, 51:60])), 6)
k13w3c38_truescore <- round((2 * k13w3c38_tpr * k13w3c38_tnr) /

```

```

(k13w3c38_tpr + k13w3c38_tnr), 6)

k15w1c38_tpr <- round(sum(cm_c38_tables[1, 61:70]) /
  (sum(cm_c38_tables[1, 61:70]) + sum(cm_c38_tables[2, 61:70])), 6)
k15w1c38_tnr <- round(sum(cm_c38_tables[4, 61:70]) /
  (sum(cm_c38_tables[4, 61:70]) + sum(cm_c38_tables[3, 61:70])), 6)
k15w1c38_truescore <- round((2 * k15w1c38_tpr * k15w1c38_tnr) /
  (k15w1c38_tpr + k15w1c38_tnr), 6)

k15w2c38_tpr <- round(sum(cm_c38_tables[1, 71:80]) /
  (sum(cm_c38_tables[1, 71:80]) + sum(cm_c38_tables[2, 71:80])), 6)
k15w2c38_tnr <- round(sum(cm_c38_tables[4, 71:80]) /
  (sum(cm_c38_tables[4, 71:80]) + sum(cm_c38_tables[3, 71:80])), 6)
k15w2c38_truescore <- round((2 * k15w2c38_tpr * k15w2c38_tnr) /
  (k15w2c38_tpr + k15w2c38_tnr), 6)

k15w3c38_tpr <- round(sum(cm_c38_tables[1, 81:90]) /
  (sum(cm_c38_tables[1, 81:90]) + sum(cm_c38_tables[2, 81:90])), 6)
k15w3c38_tnr <- round(sum(cm_c38_tables[4, 81:90]) /
  (sum(cm_c38_tables[4, 81:90]) + sum(cm_c38_tables[3, 81:90])), 6)
k15w3c38_truescore <- round((2 * k15w3c38_tpr * k15w3c38_tnr) /
  (k15w3c38_tpr + k15w3c38_tnr), 6)

k17w1c38_tpr <- round(sum(cm_c38_tables[1, 91:100]) /
  (sum(cm_c38_tables[1, 91:100]) + sum(cm_c38_tables[2, 91:100])), 6)
k17w1c38_tnr <- round(sum(cm_c38_tables[4, 91:100]) /
  (sum(cm_c38_tables[4, 91:100]) + sum(cm_c38_tables[3, 91:100])), 6)
k17w1c38_truescore <- round((2 * k17w1c38_tpr * k17w1c38_tnr) /
  (k17w1c38_tpr + k17w1c38_tnr), 6)

k17w2c38_tpr <- round(sum(cm_c38_tables[1, 101:110]) /
  (sum(cm_c38_tables[1, 101:110]) + sum(cm_c38_tables[2, 101:110])), 6)
k17w2c38_tnr <- round(sum(cm_c38_tables[4, 101:110]) /
  (sum(cm_c38_tables[4, 101:110]) + sum(cm_c38_tables[3, 101:110])), 6)
k17w2c38_truescore <- round((2 * k17w2c38_tpr * k17w2c38_tnr) /
  (k17w2c38_tpr + k17w2c38_tnr), 6)

k17w3c38_tpr <- round(sum(cm_c38_tables[1, 111:120]) /
  (sum(cm_c38_tables[1, 111:120]) + sum(cm_c38_tables[2, 111:120])), 6)
k17w3c38_tnr <- round(sum(cm_c38_tables[4, 111:120]) /
  (sum(cm_c38_tables[4, 111:120]) + sum(cm_c38_tables[3, 111:120])), 6)
k17w3c38_truescore <- round((2 * k17w3c38_tpr * k17w3c38_tnr) /
  (k17w3c38_tpr + k17w3c38_tnr), 6)

k19w1c38_tpr <- round(sum(cm_c38_tables[1, 121:130]) /
  (sum(cm_c38_tables[1, 121:130]) + sum(cm_c38_tables[2, 121:130])), 6)
k19w1c38_tnr <- round(sum(cm_c38_tables[4, 121:130]) /
  (sum(cm_c38_tables[4, 121:130]) + sum(cm_c38_tables[3, 121:130])), 6)
k19w1c38_truescore <- round((2 * k19w1c38_tpr * k19w1c38_tnr) /
  (k19w1c38_tpr + k19w1c38_tnr), 6)

```

```

k19w2c38_tpr <- round(sum(cm_c38_tables[1, 131:140]) /
  (sum(cm_c38_tables[1, 131:140]) + sum(cm_c38_tables[2, 131:140])), 6)
k19w2c38_tnr <- round(sum(cm_c38_tables[4, 131:140]) /
  (sum(cm_c38_tables[4, 131:140]) + sum(cm_c38_tables[3, 131:140])), 6)
k19w2c38_truescore <- round((2 * k19w2c38_tpr * k19w2c38_tnr) /
  (k19w2c38_tpr + k19w2c38_tnr), 6)

k19w3c38_tpr <- round(sum(cm_c38_tables[1, 141:150]) /
  (sum(cm_c38_tables[1, 141:150]) + sum(cm_c38_tables[2, 141:150])), 6)
k19w3c38_tnr <- round(sum(cm_c38_tables[4, 141:150]) /
  (sum(cm_c38_tables[4, 141:150]) + sum(cm_c38_tables[3, 141:150])), 6)
k19w3c38_truescore <- round((2 * k19w3c38_tpr * k19w3c38_tnr) /
  (k19w3c38_tpr + k19w3c38_tnr), 6)

k21w1c38_tpr <- round(sum(cm_c38_tables[1, 151:160]) /
  (sum(cm_c38_tables[1, 151:160]) + sum(cm_c38_tables[2, 151:160])), 6)
k21w1c38_tnr <- round(sum(cm_c38_tables[4, 151:160]) /
  (sum(cm_c38_tables[4, 151:160]) + sum(cm_c38_tables[3, 151:160])), 6)
k21w1c38_truescore <- round((2 * k21w1c38_tpr * k21w1c38_tnr) /
  (k21w1c38_tpr + k21w1c38_tnr), 6)

k21w2c38_tpr <- round(sum(cm_c38_tables[1, 161:170]) /
  (sum(cm_c38_tables[1, 161:170]) + sum(cm_c38_tables[2, 161:170])), 6)
k21w2c38_tnr <- round(sum(cm_c38_tables[4, 161:170]) /
  (sum(cm_c38_tables[4, 161:170]) + sum(cm_c38_tables[3, 161:170])), 6)
k21w2c38_truescore <- round((2 * k21w2c38_tpr * k21w2c38_tnr) /
  (k21w2c38_tpr + k21w2c38_tnr), 6)

k21w3c38_tpr <- round(sum(cm_c38_tables[1, 171:180]) /
  (sum(cm_c38_tables[1, 171:180]) + sum(cm_c38_tables[2, 171:180])), 6)
k21w3c38_tnr <- round(sum(cm_c38_tables[4, 171:180]) /
  (sum(cm_c38_tables[4, 171:180]) + sum(cm_c38_tables[3, 171:180])), 6)
k21w3c38_truescore <- round((2 * k21w3c38_tpr * k21w3c38_tnr) /
  (k21w3c38_tpr + k21w3c38_tnr), 6)

k23w1c38_tpr <- round(sum(cm_c38_tables[1, 181:190]) /
  (sum(cm_c38_tables[1, 181:190]) + sum(cm_c38_tables[2, 181:190])), 6)
k23w1c38_tnr <- round(sum(cm_c38_tables[4, 181:190]) /
  (sum(cm_c38_tables[4, 181:190]) + sum(cm_c38_tables[3, 181:190])), 6)
k23w1c38_truescore <- round((2 * k23w1c38_tpr * k23w1c38_tnr) /
  (k23w1c38_tpr + k23w1c38_tnr), 6)

k23w2c38_tpr <- round(sum(cm_c38_tables[1, 191:200]) /
  (sum(cm_c38_tables[1, 191:200]) + sum(cm_c38_tables[2, 191:200])), 6)
k23w2c38_tnr <- round(sum(cm_c38_tables[4, 191:200]) /
  (sum(cm_c38_tables[4, 191:200]) + sum(cm_c38_tables[3, 191:200])), 6)
k23w2c38_truescore <- round((2 * k23w2c38_tpr * k23w2c38_tnr) /
  (k23w2c38_tpr + k23w2c38_tnr), 6)

k23w3c38_tpr <- round(sum(cm_c38_tables[1, 201:210]) /
  (sum(cm_c38_tables[1, 201:210]) + sum(cm_c38_tables[2, 201:210])), 6)

```

```

k23w3c38_tnr <- round(sum(cm_c38_tables[4, 201:210]) /
  (sum(cm_c38_tables[4, 201:210]) + sum(cm_c38_tables[3, 201:210])), 6)
k23w3c38_truescore <- round((2 * k23w3c38_tpr * k23w3c38_tnr) /
  (k23w3c38_tpr + k23w3c38_tnr), 6)

k25w1c38_tpr <- round(sum(cm_c38_tables[1, 211:220]) /
  (sum(cm_c38_tables[1, 211:220]) + sum(cm_c38_tables[2, 211:220])), 6)
k25w1c38_tnr <- round(sum(cm_c38_tables[4, 211:220]) /
  (sum(cm_c38_tables[4, 211:220]) + sum(cm_c38_tables[3, 211:220])), 6)
k25w1c38_truescore <- round((2 * k25w1c38_tpr * k25w1c38_tnr) /
  (k25w1c38_tpr + k25w1c38_tnr), 6)

k25w2c38_tpr <- round(sum(cm_c38_tables[1, 221:230]) /
  (sum(cm_c38_tables[1, 221:230]) + sum(cm_c38_tables[2, 221:230])), 6)
k25w2c38_tnr <- round(sum(cm_c38_tables[4, 221:230]) /
  (sum(cm_c38_tables[4, 221:230]) + sum(cm_c38_tables[3, 221:230])), 6)
k25w2c38_truescore <- round((2 * k25w2c38_tpr * k25w2c38_tnr) /
  (k25w2c38_tpr + k25w2c38_tnr), 6)

k25w3c38_tpr <- round(sum(cm_c38_tables[1, 231:240]) /
  (sum(cm_c38_tables[1, 231:240]) + sum(cm_c38_tables[2, 231:240])), 6)
k25w3c38_tnr <- round(sum(cm_c38_tables[4, 231:240]) /
  (sum(cm_c38_tables[4, 231:240]) + sum(cm_c38_tables[3, 231:240])), 6)
k25w3c38_truescore <- round((2 * k25w3c38_tpr * k25w3c38_tnr) /
  (k25w3c38_tpr + k25w3c38_tnr), 6)

k27w1c38_tpr <- round(sum(cm_c38_tables[1, 241:250]) /
  (sum(cm_c38_tables[1, 241:250]) + sum(cm_c38_tables[2, 241:250])), 6)
k27w1c38_tnr <- round(sum(cm_c38_tables[4, 241:250]) /
  (sum(cm_c38_tables[4, 241:250]) + sum(cm_c38_tables[3, 241:250])), 6)
k27w1c38_truescore <- round((2 * k27w1c38_tpr * k27w1c38_tnr) /
  (k27w1c38_tpr + k27w1c38_tnr), 6)

k27w2c38_tpr <- round(sum(cm_c38_tables[1, 251:260]) /
  (sum(cm_c38_tables[1, 251:260]) + sum(cm_c38_tables[2, 251:260])), 6)
k27w2c38_tnr <- round(sum(cm_c38_tables[4, 251:260]) /
  (sum(cm_c38_tables[4, 251:260]) + sum(cm_c38_tables[3, 251:260])), 6)
k27w2c38_truescore <- round((2 * k27w2c38_tpr * k27w2c38_tnr) /
  (k27w2c38_tpr + k27w2c38_tnr), 6)

k27w3c38_tpr <- round(sum(cm_c38_tables[1, 261:270]) /
  (sum(cm_c38_tables[1, 261:270]) + sum(cm_c38_tables[2, 261:270])), 6)
k27w3c38_tnr <- round(sum(cm_c38_tables[4, 261:270]) /
  (sum(cm_c38_tables[4, 261:270]) + sum(cm_c38_tables[3, 261:270])), 6)
k27w3c38_truescore <- round((2 * k27w3c38_tpr * k27w3c38_tnr) /
  (k27w3c38_tpr + k27w3c38_tnr), 6)

k29w1c38_tpr <- round(sum(cm_c38_tables[1, 271:280]) /
  (sum(cm_c38_tables[1, 271:280]) + sum(cm_c38_tables[2, 271:280])), 6)
k29w1c38_tnr <- round(sum(cm_c38_tables[4, 271:280]) /

```

```

(sum(cm_c38_tables[4, 271:280]) + sum(cm_c38_tables[3, 271:280])), 6)
k29w1c38_truescore <- round((2 * k29w1c38_tpr * k29w1c38_tnr) /
(k29w1c38_tpr + k29w1c38_tnr), 6)

k29w2c38_tpr <- round(sum(cm_c38_tables[1, 281:290]) /
(sum(cm_c38_tables[1, 281:290]) + sum(cm_c38_tables[2, 281:290])), 6)
k29w2c38_tnr <- round(sum(cm_c38_tables[4, 281:290]) /
(sum(cm_c38_tables[4, 281:290]) + sum(cm_c38_tables[3, 281:290])), 6)
k29w2c38_truescore <- round((2 * k29w2c38_tpr * k29w2c38_tnr) /
(k29w2c38_tpr + k29w2c38_tnr), 6)

k29w3c38_tpr <- round(sum(cm_c38_tables[1, 291:300]) /
(sum(cm_c38_tables[1, 291:300]) + sum(cm_c38_tables[2, 291:300])), 6)
k29w3c38_tnr <- round(sum(cm_c38_tables[4, 291:300]) /
(sum(cm_c38_tables[4, 291:300]) + sum(cm_c38_tables[3, 291:300])), 6)
k29w3c38_truescore <- round((2 * k29w3c38_tpr * k29w3c38_tnr) /
(k29w3c38_tpr + k29w3c38_tnr), 6)

k31w1c38_tpr <- round(sum(cm_c38_tables[1, 301:310]) /
(sum(cm_c38_tables[1, 301:310]) + sum(cm_c38_tables[2, 301:310])), 6)
k31w1c38_tnr <- round(sum(cm_c38_tables[4, 301:310]) /
(sum(cm_c38_tables[4, 301:310]) + sum(cm_c38_tables[3, 301:310])), 6)
k31w1c38_truescore <- round((2 * k31w1c38_tpr * k31w1c38_tnr) /
(k31w1c38_tpr + k31w1c38_tnr), 6)

k31w2c38_tpr <- round(sum(cm_c38_tables[1, 311:320]) /
(sum(cm_c38_tables[1, 311:320]) + sum(cm_c38_tables[2, 311:320])), 6)
k31w2c38_tnr <- round(sum(cm_c38_tables[4, 311:320]) /
(sum(cm_c38_tables[4, 311:320]) + sum(cm_c38_tables[3, 311:320])), 6)
k31w2c38_truescore <- round((2 * k31w2c38_tpr * k31w2c38_tnr) /
(k31w2c38_tpr + k31w2c38_tnr), 6)

k31w3c38_tpr <- round(sum(cm_c38_tables[1, 321:330]) /
(sum(cm_c38_tables[1, 321:330]) + sum(cm_c38_tables[2, 321:330])), 6)
k31w3c38_tnr <- round(sum(cm_c38_tables[4, 321:330]) /
(sum(cm_c38_tables[4, 321:330]) + sum(cm_c38_tables[3, 321:330])), 6)
k31w3c38_truescore <- round((2 * k31w3c38_tpr * k31w3c38_tnr) /
(k31w3c38_tpr + k31w3c38_tnr), 6)

k33w1c38_tpr <- round(sum(cm_c38_tables[1, 331:340]) /
(sum(cm_c38_tables[1, 331:340]) + sum(cm_c38_tables[2, 331:340])), 6)
k33w1c38_tnr <- round(sum(cm_c38_tables[4, 331:340]) /
(sum(cm_c38_tables[4, 331:340]) + sum(cm_c38_tables[3, 331:340])), 6)
k33w1c38_truescore <- round((2 * k33w1c38_tpr * k33w1c38_tnr) /
(k33w1c38_tpr + k33w1c38_tnr), 6)

k33w2c38_tpr <- round(sum(cm_c38_tables[1, 341:350]) /
(sum(cm_c38_tables[1, 341:350]) + sum(cm_c38_tables[2, 341:350])), 6)
k33w2c38_tnr <- round(sum(cm_c38_tables[4, 341:350]) /
(sum(cm_c38_tables[4, 341:350]) + sum(cm_c38_tables[3, 341:350])), 6)
k33w2c38_truescore <- round((2 * k33w2c38_tpr * k33w2c38_tnr) /

```

```

(k33w2c38_tpr + k33w2c38_tnr), 6)

k33w3c38_tpr <- round(sum(cm_c38_tables[1, 351:360]) /
  (sum(cm_c38_tables[1, 351:360]) + sum(cm_c38_tables[2, 351:360])), 6)
k33w3c38_tnr <- round(sum(cm_c38_tables[4, 351:360]) /
  (sum(cm_c38_tables[4, 351:360]) + sum(cm_c38_tables[3, 351:360])), 6)
k33w3c38_truescore <- round((2 * k33w3c38_tpr * k33w3c38_tnr) /
  (k33w3c38_tpr + k33w3c38_tnr), 6)

k35w1c38_tpr <- round(sum(cm_c38_tables[1, 361:370]) /
  (sum(cm_c38_tables[1, 361:370]) + sum(cm_c38_tables[2, 361:370])), 6)
k35w1c38_tnr <- round(sum(cm_c38_tables[4, 361:370]) /
  (sum(cm_c38_tables[4, 361:370]) + sum(cm_c38_tables[3, 361:370])), 6)
k35w1c38_truescore <- round((2 * k35w1c38_tpr * k35w1c38_tnr) /
  (k35w1c38_tpr + k35w1c38_tnr), 6)

k35w2c38_tpr <- round(sum(cm_c38_tables[1, 371:380]) /
  (sum(cm_c38_tables[1, 371:380]) + sum(cm_c38_tables[2, 371:380])), 6)
k35w2c38_tnr <- round(sum(cm_c38_tables[4, 371:380]) /
  (sum(cm_c38_tables[4, 371:380]) + sum(cm_c38_tables[3, 371:380])), 6)
k35w2c38_truescore <- round((2 * k35w2c38_tpr * k35w2c38_tnr) /
  (k35w2c38_tpr + k35w2c38_tnr), 6)

k35w3c38_tpr <- round(sum(cm_c38_tables[1, 381:390]) /
  (sum(cm_c38_tables[1, 381:390]) + sum(cm_c38_tables[2, 381:390])), 6)
k35w3c38_tnr <- round(sum(cm_c38_tables[4, 381:390]) /
  (sum(cm_c38_tables[4, 381:390]) + sum(cm_c38_tables[3, 381:390])), 6)
k35w3c38_truescore <- round((2 * k35w3c38_tpr * k35w3c38_tnr) /
  (k35w3c38_tpr + k35w3c38_tnr), 6)

k37w1c38_tpr <- round(sum(cm_c38_tables[1, 391:400]) /
  (sum(cm_c38_tables[1, 391:400]) + sum(cm_c38_tables[2, 391:400])), 6)
k37w1c38_tnr <- round(sum(cm_c38_tables[4, 391:400]) /
  (sum(cm_c38_tables[4, 391:400]) + sum(cm_c38_tables[3, 391:400])), 6)
k37w1c38_truescore <- round((2 * k37w1c38_tpr * k37w1c38_tnr) /
  (k37w1c38_tpr + k37w1c38_tnr), 6)

k37w2c38_tpr <- round(sum(cm_c38_tables[1, 401:410]) /
  (sum(cm_c38_tables[1, 401:410]) + sum(cm_c38_tables[2, 401:410])), 6)
k37w2c38_tnr <- round(sum(cm_c38_tables[4, 401:410]) /
  (sum(cm_c38_tables[4, 401:410]) + sum(cm_c38_tables[3, 401:410])), 6)
k37w2c38_truescore <- round((2 * k37w2c38_tpr * k37w2c38_tnr) /
  (k37w2c38_tpr + k37w2c38_tnr), 6)

k37w3c38_tpr <- round(sum(cm_c38_tables[1, 411:420]) /
  (sum(cm_c38_tables[1, 411:420]) + sum(cm_c38_tables[2, 411:420])), 6)
k37w3c38_tnr <- round(sum(cm_c38_tables[4, 411:420]) /
  (sum(cm_c38_tables[4, 411:420]) + sum(cm_c38_tables[3, 411:420])), 6)
k37w3c38_truescore <- round((2 * k37w3c38_tpr * k37w3c38_tnr) /
  (k37w3c38_tpr + k37w3c38_tnr), 6)

```

```

k39w1c38_tpr <- round(sum(cm_c38_tables[1, 421:430]) /
  (sum(cm_c38_tables[1, 421:430]) + sum(cm_c38_tables[2, 421:430])), 6)
k39w1c38_tnr <- round(sum(cm_c38_tables[4, 421:430]) /
  (sum(cm_c38_tables[4, 421:430]) + sum(cm_c38_tables[3, 421:430])), 6)
k39w1c38_truescore <- round((2 * k39w1c38_tpr * k39w1c38_tnr) /
  (k39w1c38_tpr + k39w1c38_tnr), 6)

k39w2c38_tpr <- round(sum(cm_c38_tables[1, 431:440]) /
  (sum(cm_c38_tables[1, 431:440]) + sum(cm_c38_tables[2, 431:440])), 6)
k39w2c38_tnr <- round(sum(cm_c38_tables[4, 431:440]) /
  (sum(cm_c38_tables[4, 431:440]) + sum(cm_c38_tables[3, 431:440])), 6)
k39w2c38_truescore <- round((2 * k39w2c38_tpr * k39w2c38_tnr) /
  (k39w2c38_tpr + k39w2c38_tnr), 6)

k39w3c38_tpr <- round(sum(cm_c38_tables[1, 441:450]) /
  (sum(cm_c38_tables[1, 441:450]) + sum(cm_c38_tables[2, 441:450])), 6)
k39w3c38_tnr <- round(sum(cm_c38_tables[4, 441:450]) /
  (sum(cm_c38_tables[4, 441:450]) + sum(cm_c38_tables[3, 441:450])), 6)
k39w3c38_truescore <- round((2 * k39w3c38_tpr * k39w3c38_tnr) /
  (k39w3c38_tpr + k39w3c38_tnr), 6)

# Compile the 0.38 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c38_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.38,
  TPR = c(k11w1c38_tpr, k11w2c38_tpr, k11w3c38_tpr, k13w1c38_tpr,
    k13w2c38_tpr, k13w3c38_tpr, k15w1c38_tpr, k15w2c38_tpr,
    k15w3c38_tpr, k17w1c38_tpr, k17w2c38_tpr, k17w3c38_tpr,
    k19w1c38_tpr, k19w2c38_tpr, k19w3c38_tpr, k21w1c38_tpr,
    k21w2c38_tpr, k21w3c38_tpr, k23w1c38_tpr, k23w2c38_tpr,
    k23w3c38_tpr, k25w1c38_tpr, k25w2c38_tpr, k25w3c38_tpr,
    k27w1c38_tpr, k27w2c38_tpr, k27w3c38_tpr, k29w1c38_tpr,
    k29w2c38_tpr, k29w3c38_tpr, k31w1c38_tpr, k31w2c38_tpr,
    k31w3c38_tpr, k33w1c38_tpr, k33w2c38_tpr, k33w3c38_tpr,
    k35w1c38_tpr, k35w2c38_tpr, k35w3c38_tpr, k37w1c38_tpr,
    k37w2c38_tpr, k37w3c38_tpr, k39w1c38_tpr, k39w2c38_tpr,
    k39w3c38_tpr),
  TNR = c(k11w1c38_tnr, k11w2c38_tnr, k11w3c38_tnr, k13w1c38_tnr,
    k13w2c38_tnr, k13w3c38_tnr, k15w1c38_tnr, k15w2c38_tnr,
    k15w3c38_tnr, k17w1c38_tnr, k17w2c38_tnr, k17w3c38_tnr,
    k19w1c38_tnr, k19w2c38_tnr, k19w3c38_tnr, k21w1c38_tnr,
    k21w2c38_tnr, k21w3c38_tnr, k23w1c38_tnr, k23w2c38_tnr,
    k23w3c38_tnr, k25w1c38_tnr, k25w2c38_tnr, k25w3c38_tnr,
    k27w1c38_tnr, k27w2c38_tnr, k27w3c38_tnr, k29w1c38_tnr,
    k29w2c38_tnr, k29w3c38_tnr, k31w1c38_tnr, k31w2c38_tnr,
    k31w3c38_tnr, k33w1c38_tnr, k33w2c38_tnr, k33w3c38_tnr),
  Truescore = k39w1c38_truescore)

```

```

k35w1c38_tnr, k35w2c38_tnr, k35w3c38_tnr, k37w1c38_tnr,
k37w2c38_tnr, k37w3c38_tnr, k39w1c38_tnr, k39w2c38_tnr,
k39w3c38_tnr),
Truescore = c(k11w1c38_truescore, k11w2c38_truescore,
k11w3c38_truescore, k13w1c38_truescore,
k13w2c38_truescore, k13w3c38_truescore,
k15w1c38_truescore, k15w2c38_truescore,
k15w3c38_truescore, k17w1c38_truescore,
k17w2c38_truescore, k17w3c38_truescore,
k19w1c38_truescore, k19w2c38_truescore,
k19w3c38_truescore, k21w1c38_truescore,
k21w2c38_truescore, k21w3c38_truescore,
k23w1c38_truescore, k23w2c38_truescore,
k23w3c38_truescore, k25w1c38_truescore,
k25w2c38_truescore, k25w3c38_truescore,
k27w1c38_truescore, k27w2c38_truescore,
k27w3c38_truescore, k29w1c38_truescore,
k29w2c38_truescore, k29w3c38_truescore,
k31w1c38_truescore, k31w2c38_truescore,
k31w3c38_truescore, k33w1c38_truescore,
k33w2c38_truescore, k33w3c38_truescore,
k35w1c38_truescore, k35w2c38_truescore,
k35w3c38_truescore, k37w1c38_truescore,
k37w2c38_truescore, k37w3c38_truescore,
k39w1c38_truescore, k39w2c38_truescore,
k39w3c38_truescore))

```

`knitr::kable(c38_results[1:45,], caption = "c38_results")`

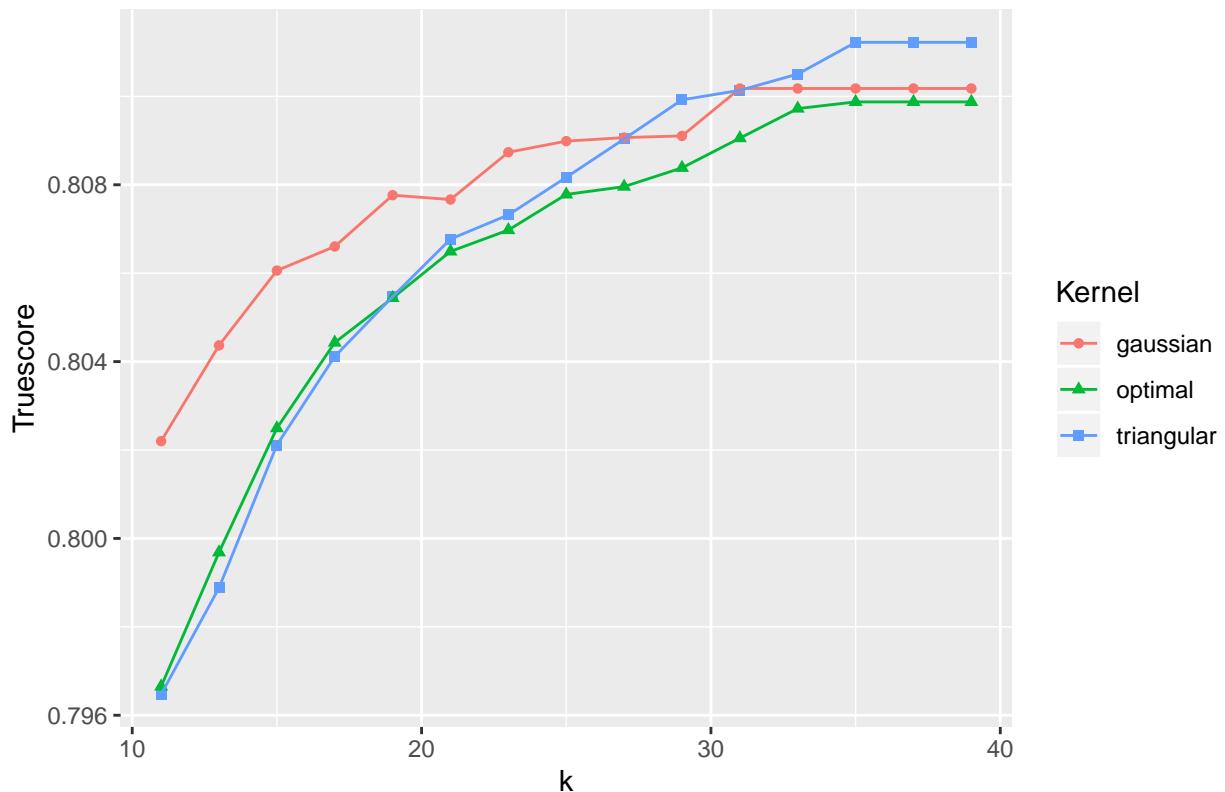
Table 47: c38_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.38	0.745482	0.854953	0.796474
11	gaussian	0.38	0.754769	0.855994	0.802201
11	optimal	0.38	0.746637	0.853847	0.796651
13	triangular	0.38	0.749197	0.855630	0.798884
13	gaussian	0.38	0.758886	0.855647	0.804367
13	optimal	0.38	0.751004	0.855118	0.799686
15	triangular	0.38	0.754367	0.856308	0.802112
15	gaussian	0.38	0.761747	0.855845	0.806059
15	optimal	0.38	0.755070	0.856275	0.802494
17	triangular	0.38	0.757932	0.856291	0.804115
17	gaussian	0.38	0.763102	0.855366	0.806604
17	optimal	0.38	0.758584	0.856175	0.804430
19	triangular	0.38	0.760291	0.856357	0.805470
19	gaussian	0.38	0.765512	0.854953	0.807764
19	optimal	0.38	0.760392	0.856159	0.805439
21	triangular	0.38	0.762851	0.856060	0.806772
21	gaussian	0.38	0.765110	0.855234	0.807666
21	optimal	0.38	0.762600	0.855746	0.806492
23	triangular	0.38	0.764157	0.855647	0.807318
23	gaussian	0.38	0.767219	0.855003	0.808736
23	optimal	0.38	0.763755	0.855383	0.806976

k	Kernel	Cut	TPR	TNR	Truescore
25	triangular	0.38	0.765813	0.855482	0.808168
25	gaussian	0.38	0.768072	0.854507	0.808987
25	optimal	0.38	0.764960	0.855680	0.807781
27	triangular	0.38	0.767520	0.855300	0.809036
27	gaussian	0.38	0.768323	0.854375	0.809067
27	optimal	0.38	0.765462	0.855449	0.807958
29	triangular	0.38	0.769478	0.854854	0.809922
29	gaussian	0.38	0.768323	0.854458	0.809105
29	optimal	0.38	0.766466	0.855151	0.808383
31	triangular	0.38	0.769930	0.854772	0.810136
31	gaussian	0.38	0.770382	0.854309	0.810178
31	optimal	0.38	0.767821	0.854970	0.809055
33	triangular	0.38	0.770582	0.854788	0.810504
33	gaussian	0.38	0.770382	0.854309	0.810178
33	optimal	0.38	0.769227	0.854722	0.809724
35	triangular	0.38	0.771787	0.854904	0.811222
35	gaussian	0.38	0.770382	0.854309	0.810178
35	optimal	0.38	0.769578	0.854623	0.809874
37	triangular	0.38	0.771787	0.854904	0.811222
37	gaussian	0.38	0.770382	0.854309	0.810178
37	optimal	0.38	0.769578	0.854623	0.809874
39	triangular	0.38	0.771787	0.854904	0.811222
39	gaussian	0.38	0.770382	0.854309	0.810178
39	optimal	0.38	0.769578	0.854623	0.809874

```
ggplot(c38_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.38 (Truescore)")
```

Optimal k and Kernel for Decision Cutoff 0.38 (Truescore)

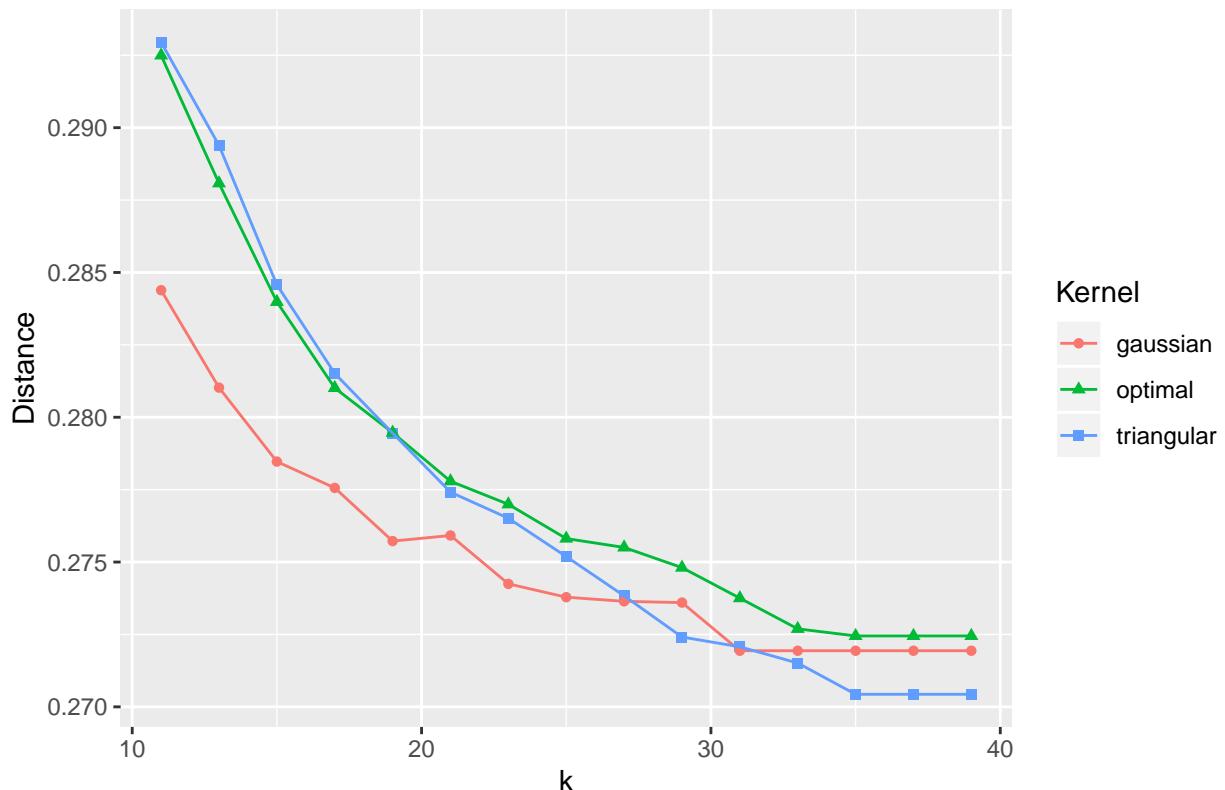


```
# For the Cutoff of 0.38, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c38_results <- c38_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c38_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.38 (Distance)")
```

Optimal k and Kernel for Decision Cutoff 0.38 (Distance)



```
# For the Cutoff of 0.38, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c38_results$Truescore)
```

```
## [1] 0.811222
```

```
(c38_opt_k_ts <- c38_results$k[which.max(c38_results$Truescore)])
```

```
## [1] 35
```

```
(c38_opt_kernel_ts <- c38_results$Kernel[which.max(c38_results$Truescore)])
```

```
## [1] "triangular"
```

```
(c38_opt_cut_ts <- c38_results$Cut[which.max(c38_results$Truescore)])
```

```
## [1] 0.38
```

```
(c38_opt_tpr_ts <- c38_results$TPR[which.max(c38_results$Truescore)])
```

```
## [1] 0.771787
```

```

(c38_opt_tnr_ts <- c38_results$TNR[which.max(c38_results$Truescore)])

## [1] 0.854904

(c38_opt_d_ts <- c38_results$Distance[which.max(c38_results$Truescore)])

## [1] 0.270433

min(c38_results$Distance)

## [1] 0.270433

(c38_opt_k_dist <- c38_results$k[which.min(c38_results$Distance)])

## [1] 35

(c38_opt_kernel_dist <- c38_results$Kernel[which.min(c38_results$Distance)])

## [1] "triangular"

(c38_opt_cut_dist <- c38_results$Cut[which.min(c38_results$Distance)])

## [1] 0.38

(c38_opt_tpr_dist <- c38_results$TPR[which.min(c38_results$Distance)])

## [1] 0.771787

(c38_opt_tnr_dist <- c38_results$TNR[which.min(c38_results$Distance)])

## [1] 0.854904

(c38_opt_t_dist <- c38_results$Truescore[which.min(c38_results$Distance)])

## [1] 0.811222

#####
# 0.39 Cutoff
#####

# For the decision cutoff of 0.39, generate a confusion matrix for
# every combination of k (11:39 by two), kernel (1:3) and fold (1:10).

cm_c39 <- sapply(kwf_dfs_1, function(x) {
  ss <- subset(x, select = c(pred39, obs))
  confusionMatrix(ss$pred39, ss$obs)
}, simplify = FALSE, USE.NAMES = TRUE)

rm(kwf_dfs_1)
gc(reset = TRUE)

```

```

##           used     (Mb) gc trigger     (Mb) max used     (Mb)
## Ncells   4272237  228.2    7420952  396.4    4272237  228.2
## Vcells  214491969 1636.5  395060576 3014.1  214491969 1636.5

names(cm_c39) <- kwf_dfs_v

cm_c39_tables <- sapply(cm_c39, "[[", 2)
cm_c39_tables <- as_tibble(cm_c39_tables)

# For each combination of k and kernel, sum the True Positives, False Negatives,
# True Negatives, and False Positives respectively across all 10
# folds. Then use these totals to compute the Truescore for
# each combination of k and kernel when the decision cutoff is 0.39.

k11w1c39_tpr <- round(sum(cm_c39_tables[1, 1:10]) /
  (sum(cm_c39_tables[1, 1:10]) + sum(cm_c39_tables[2, 1:10])), 6)
k11w1c39_tnr <- round(sum(cm_c39_tables[4, 1:10]) /
  (sum(cm_c39_tables[4, 1:10]) + sum(cm_c39_tables[3, 1:10])), 6)
k11w1c39_truescore <- round((2 * k11w1c39_tpr * k11w1c39_tnr) /
  (k11w1c39_tpr + k11w1c39_tnr), 6)

k11w2c39_tpr <- round(sum(cm_c39_tables[1, 11:20]) /
  (sum(cm_c39_tables[1, 11:20]) + sum(cm_c39_tables[2, 11:20])), 6)
k11w2c39_tnr <- round(sum(cm_c39_tables[4, 11:20]) /
  (sum(cm_c39_tables[4, 11:20]) + sum(cm_c39_tables[3, 11:20])), 6)
k11w2c39_truescore <- round((2 * k11w2c39_tpr * k11w2c39_tnr) /
  (k11w2c39_tpr + k11w2c39_tnr), 6)

k11w3c39_tpr <- round(sum(cm_c39_tables[1, 21:30]) /
  (sum(cm_c39_tables[1, 21:30]) + sum(cm_c39_tables[2, 21:30])), 6)
k11w3c39_tnr <- round(sum(cm_c39_tables[4, 21:30]) /
  (sum(cm_c39_tables[4, 21:30]) + sum(cm_c39_tables[3, 21:30])), 6)
k11w3c39_truescore <- round((2 * k11w3c39_tpr * k11w3c39_tnr) /
  (k11w3c39_tpr + k11w3c39_tnr), 6)

k13w1c39_tpr <- round(sum(cm_c39_tables[1, 31:40]) /
  (sum(cm_c39_tables[1, 31:40]) + sum(cm_c39_tables[2, 31:40])), 6)
k13w1c39_tnr <- round(sum(cm_c39_tables[4, 31:40]) /
  (sum(cm_c39_tables[4, 31:40]) + sum(cm_c39_tables[3, 31:40])), 6)
k13w1c39_truescore <- round((2 * k13w1c39_tpr * k13w1c39_tnr) /
  (k13w1c39_tpr + k13w1c39_tnr), 6)

k13w2c39_tpr <- round(sum(cm_c39_tables[1, 41:50]) /
  (sum(cm_c39_tables[1, 41:50]) + sum(cm_c39_tables[2, 41:50])), 6)
k13w2c39_tnr <- round(sum(cm_c39_tables[4, 41:50]) /
  (sum(cm_c39_tables[4, 41:50]) + sum(cm_c39_tables[3, 41:50])), 6)
k13w2c39_truescore <- round((2 * k13w2c39_tpr * k13w2c39_tnr) /
  (k13w2c39_tpr + k13w2c39_tnr), 6)

k13w3c39_tpr <- round(sum(cm_c39_tables[1, 51:60]) /
  (sum(cm_c39_tables[1, 51:60]) + sum(cm_c39_tables[2, 51:60])), 6)
k13w3c39_tnr <- round(sum(cm_c39_tables[4, 51:60]) /
  (sum(cm_c39_tables[4, 51:60]) + sum(cm_c39_tables[3, 51:60])), 6)

```

```

k13w3c39_truescore <- round((2 * k13w3c39_tpr * k13w3c39_tnr) /
  (k13w3c39_tpr + k13w3c39_tnr), 6)

k15w1c39_tpr <- round(sum(cm_c39_tables[1, 61:70]) /
  (sum(cm_c39_tables[1, 61:70]) + sum(cm_c39_tables[2, 61:70])), 6)
k15w1c39_tnr <- round(sum(cm_c39_tables[4, 61:70]) /
  (sum(cm_c39_tables[4, 61:70]) + sum(cm_c39_tables[3, 61:70])), 6)
k15w1c39_truescore <- round((2 * k15w1c39_tpr * k15w1c39_tnr) /
  (k15w1c39_tpr + k15w1c39_tnr), 6)

k15w2c39_tpr <- round(sum(cm_c39_tables[1, 71:80]) /
  (sum(cm_c39_tables[1, 71:80]) + sum(cm_c39_tables[2, 71:80])), 6)
k15w2c39_tnr <- round(sum(cm_c39_tables[4, 71:80]) /
  (sum(cm_c39_tables[4, 71:80]) + sum(cm_c39_tables[3, 71:80])), 6)
k15w2c39_truescore <- round((2 * k15w2c39_tpr * k15w2c39_tnr) /
  (k15w2c39_tpr + k15w2c39_tnr), 6)

k15w3c39_tpr <- round(sum(cm_c39_tables[1, 81:90]) /
  (sum(cm_c39_tables[1, 81:90]) + sum(cm_c39_tables[2, 81:90])), 6)
k15w3c39_tnr <- round(sum(cm_c39_tables[4, 81:90]) /
  (sum(cm_c39_tables[4, 81:90]) + sum(cm_c39_tables[3, 81:90])), 6)
k15w3c39_truescore <- round((2 * k15w3c39_tpr * k15w3c39_tnr) /
  (k15w3c39_tpr + k15w3c39_tnr), 6)

k17w1c39_tpr <- round(sum(cm_c39_tables[1, 91:100]) /
  (sum(cm_c39_tables[1, 91:100]) + sum(cm_c39_tables[2, 91:100])), 6)
k17w1c39_tnr <- round(sum(cm_c39_tables[4, 91:100]) /
  (sum(cm_c39_tables[4, 91:100]) + sum(cm_c39_tables[3, 91:100])), 6)
k17w1c39_truescore <- round((2 * k17w1c39_tpr * k17w1c39_tnr) /
  (k17w1c39_tpr + k17w1c39_tnr), 6)

k17w2c39_tpr <- round(sum(cm_c39_tables[1, 101:110]) /
  (sum(cm_c39_tables[1, 101:110]) + sum(cm_c39_tables[2, 101:110])), 6)
k17w2c39_tnr <- round(sum(cm_c39_tables[4, 101:110]) /
  (sum(cm_c39_tables[4, 101:110]) + sum(cm_c39_tables[3, 101:110])), 6)
k17w2c39_truescore <- round((2 * k17w2c39_tpr * k17w2c39_tnr) /
  (k17w2c39_tpr + k17w2c39_tnr), 6)

k17w3c39_tpr <- round(sum(cm_c39_tables[1, 111:120]) /
  (sum(cm_c39_tables[1, 111:120]) + sum(cm_c39_tables[2, 111:120])), 6)
k17w3c39_tnr <- round(sum(cm_c39_tables[4, 111:120]) /
  (sum(cm_c39_tables[4, 111:120]) + sum(cm_c39_tables[3, 111:120])), 6)
k17w3c39_truescore <- round((2 * k17w3c39_tpr * k17w3c39_tnr) /
  (k17w3c39_tpr + k17w3c39_tnr), 6)

k19w1c39_tpr <- round(sum(cm_c39_tables[1, 121:130]) /
  (sum(cm_c39_tables[1, 121:130]) + sum(cm_c39_tables[2, 121:130])), 6)
k19w1c39_tnr <- round(sum(cm_c39_tables[4, 121:130]) /
  (sum(cm_c39_tables[4, 121:130]) + sum(cm_c39_tables[3, 121:130])), 6)
k19w1c39_truescore <- round((2 * k19w1c39_tpr * k19w1c39_tnr) /

```

```

(k19w1c39_tpr + k19w1c39_tnr), 6)

k19w2c39_tpr <- round(sum(cm_c39_tables[1, 131:140]) /
  (sum(cm_c39_tables[1, 131:140]) + sum(cm_c39_tables[2, 131:140])), 6)
k19w2c39_tnr <- round(sum(cm_c39_tables[4, 131:140]) /
  (sum(cm_c39_tables[4, 131:140]) + sum(cm_c39_tables[3, 131:140])), 6)
k19w2c39_truescore <- round((2 * k19w2c39_tpr * k19w2c39_tnr) /
  (k19w2c39_tpr + k19w2c39_tnr), 6)

k19w3c39_tpr <- round(sum(cm_c39_tables[1, 141:150]) /
  (sum(cm_c39_tables[1, 141:150]) + sum(cm_c39_tables[2, 141:150])), 6)
k19w3c39_tnr <- round(sum(cm_c39_tables[4, 141:150]) /
  (sum(cm_c39_tables[4, 141:150]) + sum(cm_c39_tables[3, 141:150])), 6)
k19w3c39_truescore <- round((2 * k19w3c39_tpr * k19w3c39_tnr) /
  (k19w3c39_tpr + k19w3c39_tnr), 6)

k21w1c39_tpr <- round(sum(cm_c39_tables[1, 151:160]) /
  (sum(cm_c39_tables[1, 151:160]) + sum(cm_c39_tables[2, 151:160])), 6)
k21w1c39_tnr <- round(sum(cm_c39_tables[4, 151:160]) /
  (sum(cm_c39_tables[4, 151:160]) + sum(cm_c39_tables[3, 151:160])), 6)
k21w1c39_truescore <- round((2 * k21w1c39_tpr * k21w1c39_tnr) /
  (k21w1c39_tpr + k21w1c39_tnr), 6)

k21w2c39_tpr <- round(sum(cm_c39_tables[1, 161:170]) /
  (sum(cm_c39_tables[1, 161:170]) + sum(cm_c39_tables[2, 161:170])), 6)
k21w2c39_tnr <- round(sum(cm_c39_tables[4, 161:170]) /
  (sum(cm_c39_tables[4, 161:170]) + sum(cm_c39_tables[3, 161:170])), 6)
k21w2c39_truescore <- round((2 * k21w2c39_tpr * k21w2c39_tnr) /
  (k21w2c39_tpr + k21w2c39_tnr), 6)

k21w3c39_tpr <- round(sum(cm_c39_tables[1, 171:180]) /
  (sum(cm_c39_tables[1, 171:180]) + sum(cm_c39_tables[2, 171:180])), 6)
k21w3c39_tnr <- round(sum(cm_c39_tables[4, 171:180]) /
  (sum(cm_c39_tables[4, 171:180]) + sum(cm_c39_tables[3, 171:180])), 6)
k21w3c39_truescore <- round((2 * k21w3c39_tpr * k21w3c39_tnr) /
  (k21w3c39_tpr + k21w3c39_tnr), 6)

k23w1c39_tpr <- round(sum(cm_c39_tables[1, 181:190]) /
  (sum(cm_c39_tables[1, 181:190]) + sum(cm_c39_tables[2, 181:190])), 6)
k23w1c39_tnr <- round(sum(cm_c39_tables[4, 181:190]) /
  (sum(cm_c39_tables[4, 181:190]) + sum(cm_c39_tables[3, 181:190])), 6)
k23w1c39_truescore <- round((2 * k23w1c39_tpr * k23w1c39_tnr) /
  (k23w1c39_tpr + k23w1c39_tnr), 6)

k23w2c39_tpr <- round(sum(cm_c39_tables[1, 191:200]) /
  (sum(cm_c39_tables[1, 191:200]) + sum(cm_c39_tables[2, 191:200])), 6)
k23w2c39_tnr <- round(sum(cm_c39_tables[4, 191:200]) /
  (sum(cm_c39_tables[4, 191:200]) + sum(cm_c39_tables[3, 191:200])), 6)
k23w2c39_truescore <- round((2 * k23w2c39_tpr * k23w2c39_tnr) /
  (k23w2c39_tpr + k23w2c39_tnr), 6)

```

```

k23w3c39_tpr <- round(sum(cm_c39_tables[1, 201:210]) /
  (sum(cm_c39_tables[1, 201:210]) + sum(cm_c39_tables[2, 201:210])), 6)
k23w3c39_tnr <- round(sum(cm_c39_tables[4, 201:210]) /
  (sum(cm_c39_tables[4, 201:210]) + sum(cm_c39_tables[3, 201:210])), 6)
k23w3c39_truescore <- round((2 * k23w3c39_tpr * k23w3c39_tnr) /
  (k23w3c39_tpr + k23w3c39_tnr), 6)

k25w1c39_tpr <- round(sum(cm_c39_tables[1, 211:220]) /
  (sum(cm_c39_tables[1, 211:220]) + sum(cm_c39_tables[2, 211:220])), 6)
k25w1c39_tnr <- round(sum(cm_c39_tables[4, 211:220]) /
  (sum(cm_c39_tables[4, 211:220]) + sum(cm_c39_tables[3, 211:220])), 6)
k25w1c39_truescore <- round((2 * k25w1c39_tpr * k25w1c39_tnr) /
  (k25w1c39_tpr + k25w1c39_tnr), 6)

k25w2c39_tpr <- round(sum(cm_c39_tables[1, 221:230]) /
  (sum(cm_c39_tables[1, 221:230]) + sum(cm_c39_tables[2, 221:230])), 6)
k25w2c39_tnr <- round(sum(cm_c39_tables[4, 221:230]) /
  (sum(cm_c39_tables[4, 221:230]) + sum(cm_c39_tables[3, 221:230])), 6)
k25w2c39_truescore <- round((2 * k25w2c39_tpr * k25w2c39_tnr) /
  (k25w2c39_tpr + k25w2c39_tnr), 6)

k25w3c39_tpr <- round(sum(cm_c39_tables[1, 231:240]) /
  (sum(cm_c39_tables[1, 231:240]) + sum(cm_c39_tables[2, 231:240])), 6)
k25w3c39_tnr <- round(sum(cm_c39_tables[4, 231:240]) /
  (sum(cm_c39_tables[4, 231:240]) + sum(cm_c39_tables[3, 231:240])), 6)
k25w3c39_truescore <- round((2 * k25w3c39_tpr * k25w3c39_tnr) /
  (k25w3c39_tpr + k25w3c39_tnr), 6)

k27w1c39_tpr <- round(sum(cm_c39_tables[1, 241:250]) /
  (sum(cm_c39_tables[1, 241:250]) + sum(cm_c39_tables[2, 241:250])), 6)
k27w1c39_tnr <- round(sum(cm_c39_tables[4, 241:250]) /
  (sum(cm_c39_tables[4, 241:250]) + sum(cm_c39_tables[3, 241:250])), 6)
k27w1c39_truescore <- round((2 * k27w1c39_tpr * k27w1c39_tnr) /
  (k27w1c39_tpr + k27w1c39_tnr), 6)

k27w2c39_tpr <- round(sum(cm_c39_tables[1, 251:260]) /
  (sum(cm_c39_tables[1, 251:260]) + sum(cm_c39_tables[2, 251:260])), 6)
k27w2c39_tnr <- round(sum(cm_c39_tables[4, 251:260]) /
  (sum(cm_c39_tables[4, 251:260]) + sum(cm_c39_tables[3, 251:260])), 6)
k27w2c39_truescore <- round((2 * k27w2c39_tpr * k27w2c39_tnr) /
  (k27w2c39_tpr + k27w2c39_tnr), 6)

k27w3c39_tpr <- round(sum(cm_c39_tables[1, 261:270]) /
  (sum(cm_c39_tables[1, 261:270]) + sum(cm_c39_tables[2, 261:270])), 6)
k27w3c39_tnr <- round(sum(cm_c39_tables[4, 261:270]) /
  (sum(cm_c39_tables[4, 261:270]) + sum(cm_c39_tables[3, 261:270])), 6)
k27w3c39_truescore <- round((2 * k27w3c39_tpr * k27w3c39_tnr) /
  (k27w3c39_tpr + k27w3c39_tnr), 6)

k29w1c39_tpr <- round(sum(cm_c39_tables[1, 271:280]) /

```

```

(sum(cm_c39_tables[1, 271:280]) + sum(cm_c39_tables[2, 271:280])), 6)
k29w1c39_tnr <- round(sum(cm_c39_tables[4, 271:280]) /
  (sum(cm_c39_tables[4, 271:280]) + sum(cm_c39_tables[3, 271:280])), 6)
k29w1c39_truescore <- round((2 * k29w1c39_tpr * k29w1c39_tnr) /
  (k29w1c39_tpr + k29w1c39_tnr), 6)

k29w2c39_tpr <- round(sum(cm_c39_tables[1, 281:290]) /
  (sum(cm_c39_tables[1, 281:290]) + sum(cm_c39_tables[2, 281:290])), 6)
k29w2c39_tnr <- round(sum(cm_c39_tables[4, 281:290]) /
  (sum(cm_c39_tables[4, 281:290]) + sum(cm_c39_tables[3, 281:290])), 6)
k29w2c39_truescore <- round((2 * k29w2c39_tpr * k29w2c39_tnr) /
  (k29w2c39_tpr + k29w2c39_tnr), 6)

k29w3c39_tpr <- round(sum(cm_c39_tables[1, 291:300]) /
  (sum(cm_c39_tables[1, 291:300]) + sum(cm_c39_tables[2, 291:300])), 6)
k29w3c39_tnr <- round(sum(cm_c39_tables[4, 291:300]) /
  (sum(cm_c39_tables[4, 291:300]) + sum(cm_c39_tables[3, 291:300])), 6)
k29w3c39_truescore <- round((2 * k29w3c39_tpr * k29w3c39_tnr) /
  (k29w3c39_tpr + k29w3c39_tnr), 6)

k31w1c39_tpr <- round(sum(cm_c39_tables[1, 301:310]) /
  (sum(cm_c39_tables[1, 301:310]) + sum(cm_c39_tables[2, 301:310])), 6)
k31w1c39_tnr <- round(sum(cm_c39_tables[4, 301:310]) /
  (sum(cm_c39_tables[4, 301:310]) + sum(cm_c39_tables[3, 301:310])), 6)
k31w1c39_truescore <- round((2 * k31w1c39_tpr * k31w1c39_tnr) /
  (k31w1c39_tpr + k31w1c39_tnr), 6)

k31w2c39_tpr <- round(sum(cm_c39_tables[1, 311:320]) /
  (sum(cm_c39_tables[1, 311:320]) + sum(cm_c39_tables[2, 311:320])), 6)
k31w2c39_tnr <- round(sum(cm_c39_tables[4, 311:320]) /
  (sum(cm_c39_tables[4, 311:320]) + sum(cm_c39_tables[3, 311:320])), 6)
k31w2c39_truescore <- round((2 * k31w2c39_tpr * k31w2c39_tnr) /
  (k31w2c39_tpr + k31w2c39_tnr), 6)

k31w3c39_tpr <- round(sum(cm_c39_tables[1, 321:330]) /
  (sum(cm_c39_tables[1, 321:330]) + sum(cm_c39_tables[2, 321:330])), 6)
k31w3c39_tnr <- round(sum(cm_c39_tables[4, 321:330]) /
  (sum(cm_c39_tables[4, 321:330]) + sum(cm_c39_tables[3, 321:330])), 6)
k31w3c39_truescore <- round((2 * k31w3c39_tpr * k31w3c39_tnr) /
  (k31w3c39_tpr + k31w3c39_tnr), 6)

k33w1c39_tpr <- round(sum(cm_c39_tables[1, 331:340]) /
  (sum(cm_c39_tables[1, 331:340]) + sum(cm_c39_tables[2, 331:340])), 6)
k33w1c39_tnr <- round(sum(cm_c39_tables[4, 331:340]) /
  (sum(cm_c39_tables[4, 331:340]) + sum(cm_c39_tables[3, 331:340])), 6)
k33w1c39_truescore <- round((2 * k33w1c39_tpr * k33w1c39_tnr) /
  (k33w1c39_tpr + k33w1c39_tnr), 6)

k33w2c39_tpr <- round(sum(cm_c39_tables[1, 341:350]) /
  (sum(cm_c39_tables[1, 341:350]) + sum(cm_c39_tables[2, 341:350])), 6)
k33w2c39_tnr <- round(sum(cm_c39_tables[4, 341:350]) /

```

```

(sum(cm_c39_tables[4, 341:350]) + sum(cm_c39_tables[3, 341:350])), 6)
k33w2c39_truescore <- round((2 * k33w2c39_tpr * k33w2c39_tnr) /
(k33w2c39_tpr + k33w2c39_tnr), 6)

k33w3c39_tpr <- round(sum(cm_c39_tables[1, 351:360]) /
(sum(cm_c39_tables[1, 351:360]) + sum(cm_c39_tables[2, 351:360])), 6)
k33w3c39_tnr <- round(sum(cm_c39_tables[4, 351:360]) /
(sum(cm_c39_tables[4, 351:360]) + sum(cm_c39_tables[3, 351:360])), 6)
k33w3c39_truescore <- round((2 * k33w3c39_tpr * k33w3c39_tnr) /
(k33w3c39_tpr + k33w3c39_tnr), 6)

k35w1c39_tpr <- round(sum(cm_c39_tables[1, 361:370]) /
(sum(cm_c39_tables[1, 361:370]) + sum(cm_c39_tables[2, 361:370])), 6)
k35w1c39_tnr <- round(sum(cm_c39_tables[4, 361:370]) /
(sum(cm_c39_tables[4, 361:370]) + sum(cm_c39_tables[3, 361:370])), 6)
k35w1c39_truescore <- round((2 * k35w1c39_tpr * k35w1c39_tnr) /
(k35w1c39_tpr + k35w1c39_tnr), 6)

k35w2c39_tpr <- round(sum(cm_c39_tables[1, 371:380]) /
(sum(cm_c39_tables[1, 371:380]) + sum(cm_c39_tables[2, 371:380])), 6)
k35w2c39_tnr <- round(sum(cm_c39_tables[4, 371:380]) /
(sum(cm_c39_tables[4, 371:380]) + sum(cm_c39_tables[3, 371:380])), 6)
k35w2c39_truescore <- round((2 * k35w2c39_tpr * k35w2c39_tnr) /
(k35w2c39_tpr + k35w2c39_tnr), 6)

k35w3c39_tpr <- round(sum(cm_c39_tables[1, 381:390]) /
(sum(cm_c39_tables[1, 381:390]) + sum(cm_c39_tables[2, 381:390])), 6)
k35w3c39_tnr <- round(sum(cm_c39_tables[4, 381:390]) /
(sum(cm_c39_tables[4, 381:390]) + sum(cm_c39_tables[3, 381:390])), 6)
k35w3c39_truescore <- round((2 * k35w3c39_tpr * k35w3c39_tnr) /
(k35w3c39_tpr + k35w3c39_tnr), 6)

k37w1c39_tpr <- round(sum(cm_c39_tables[1, 391:400]) /
(sum(cm_c39_tables[1, 391:400]) + sum(cm_c39_tables[2, 391:400])), 6)
k37w1c39_tnr <- round(sum(cm_c39_tables[4, 391:400]) /
(sum(cm_c39_tables[4, 391:400]) + sum(cm_c39_tables[3, 391:400])), 6)
k37w1c39_truescore <- round((2 * k37w1c39_tpr * k37w1c39_tnr) /
(k37w1c39_tpr + k37w1c39_tnr), 6)

k37w2c39_tpr <- round(sum(cm_c39_tables[1, 401:410]) /
(sum(cm_c39_tables[1, 401:410]) + sum(cm_c39_tables[2, 401:410])), 6)
k37w2c39_tnr <- round(sum(cm_c39_tables[4, 401:410]) /
(sum(cm_c39_tables[4, 401:410]) + sum(cm_c39_tables[3, 401:410])), 6)
k37w2c39_truescore <- round((2 * k37w2c39_tpr * k37w2c39_tnr) /
(k37w2c39_tpr + k37w2c39_tnr), 6)

k37w3c39_tpr <- round(sum(cm_c39_tables[1, 411:420]) /
(sum(cm_c39_tables[1, 411:420]) + sum(cm_c39_tables[2, 411:420])), 6)
k37w3c39_tnr <- round(sum(cm_c39_tables[4, 411:420]) /
(sum(cm_c39_tables[4, 411:420]) + sum(cm_c39_tables[3, 411:420])), 6)
k37w3c39_truescore <- round((2 * k37w3c39_tpr * k37w3c39_tnr) /

```

```

(k37w3c39_tpr + k37w3c39_tnr), 6)

k39w1c39_tpr <- round(sum(cm_c39_tables[1, 421:430]) /
  (sum(cm_c39_tables[1, 421:430]) + sum(cm_c39_tables[2, 421:430])), 6)
k39w1c39_tnr <- round(sum(cm_c39_tables[4, 421:430]) /
  (sum(cm_c39_tables[4, 421:430]) + sum(cm_c39_tables[3, 421:430])), 6)
k39w1c39_truescore <- round((2 * k39w1c39_tpr * k39w1c39_tnr) /
  (k39w1c39_tpr + k39w1c39_tnr), 6)

k39w2c39_tpr <- round(sum(cm_c39_tables[1, 431:440]) /
  (sum(cm_c39_tables[1, 431:440]) + sum(cm_c39_tables[2, 431:440])), 6)
k39w2c39_tnr <- round(sum(cm_c39_tables[4, 431:440]) /
  (sum(cm_c39_tables[4, 431:440]) + sum(cm_c39_tables[3, 431:440])), 6)
k39w2c39_truescore <- round((2 * k39w2c39_tpr * k39w2c39_tnr) /
  (k39w2c39_tpr + k39w2c39_tnr), 6)

k39w3c39_tpr <- round(sum(cm_c39_tables[1, 441:450]) /
  (sum(cm_c39_tables[1, 441:450]) + sum(cm_c39_tables[2, 441:450])), 6)
k39w3c39_tnr <- round(sum(cm_c39_tables[4, 441:450]) /
  (sum(cm_c39_tables[4, 441:450]) + sum(cm_c39_tables[3, 441:450])), 6)
k39w3c39_truescore <- round((2 * k39w3c39_tpr * k39w3c39_tnr) /
  (k39w3c39_tpr + k39w3c39_tnr), 6)

# Compile the 0.39 cutoff results in a table, and identify the combination of
# of k and kernel that maximizes the Truescore.

c39_results <- tibble(k = c(11, 11, 11, 13, 13, 13, 15, 15, 15,
  17, 17, 17, 19, 19, 19, 21, 21, 21,
  23, 23, 23, 25, 25, 25, 27, 27, 27,
  29, 29, 29, 31, 31, 31, 33, 33, 33,
  35, 35, 35, 37, 37, 37, 39, 39, 39),
  Kernel = rep(c("triangular", "gaussian", "optimal"), 15),
  Cut = 0.39,
  TPR = c(k11w1c39_tpr, k11w2c39_tpr, k11w3c39_tpr, k13w1c39_tpr,
    k13w2c39_tpr, k13w3c39_tpr, k15w1c39_tpr, k15w2c39_tpr,
    k15w3c39_tpr, k17w1c39_tpr, k17w2c39_tpr, k17w3c39_tpr,
    k19w1c39_tpr, k19w2c39_tpr, k19w3c39_tpr, k21w1c39_tpr,
    k21w2c39_tpr, k21w3c39_tpr, k23w1c39_tpr, k23w2c39_tpr,
    k23w3c39_tpr, k25w1c39_tpr, k25w2c39_tpr, k25w3c39_tpr,
    k27w1c39_tpr, k27w2c39_tpr, k27w3c39_tpr, k29w1c39_tpr,
    k29w2c39_tpr, k29w3c39_tpr, k31w1c39_tpr, k31w2c39_tpr,
    k31w3c39_tpr, k33w1c39_tpr, k33w2c39_tpr, k33w3c39_tpr,
    k35w1c39_tpr, k35w2c39_tpr, k35w3c39_tpr, k37w1c39_tpr,
    k37w2c39_tpr, k37w3c39_tpr, k39w1c39_tpr, k39w2c39_tpr,
    k39w3c39_tpr),
  TNR = c(k11w1c39_tnr, k11w2c39_tnr, k11w3c39_tnr, k13w1c39_tnr,
    k13w2c39_tnr, k13w3c39_tnr, k15w1c39_tnr, k15w2c39_tnr,
    k15w3c39_tnr, k17w1c39_tnr, k17w2c39_tnr, k17w3c39_tnr,
    k19w1c39_tnr, k19w2c39_tnr, k19w3c39_tnr, k21w1c39_tnr,
    k21w2c39_tnr, k21w3c39_tnr, k23w1c39_tnr, k23w2c39_tnr,
    k23w3c39_tnr, k25w1c39_tnr, k25w2c39_tnr, k25w3c39_tnr,
    k27w1c39_tnr, k27w2c39_tnr, k27w3c39_tnr, k29w1c39_tnr,
    k29w2c39_tnr, k29w3c39_tnr, k31w1c39_tnr, k31w2c39_tnr,
    k31w3c39_tnr, k33w1c39_tnr, k33w2c39_tnr, k33w3c39_tnr,
    k35w1c39_tnr, k35w2c39_tnr, k35w3c39_tnr, k37w1c39_tnr,
    k37w2c39_tnr, k37w3c39_tnr, k39w1c39_tnr, k39w2c39_tnr,
    k39w3c39_tnr),
  Truescore = c(0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39,
    0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39))

```

```

k29w2c39_tnr, k29w3c39_tnr, k31w1c39_tnr, k31w2c39_tnr,
k31w3c39_tnr, k33w1c39_tnr, k33w2c39_tnr, k33w3c39_tnr,
k35w1c39_tnr, k35w2c39_tnr, k35w3c39_tnr, k37w1c39_tnr,
k37w2c39_tnr, k37w3c39_tnr, k39w1c39_tnr, k39w2c39_tnr,
k39w3c39_tnr),
Truescore = c(k11w1c39_truescore, k11w2c39_truescore,
k11w3c39_truescore, k13w1c39_truescore,
k13w2c39_truescore, k13w3c39_truescore,
k15w1c39_truescore, k15w2c39_truescore,
k15w3c39_truescore, k17w1c39_truescore,
k17w2c39_truescore, k17w3c39_truescore,
k19w1c39_truescore, k19w2c39_truescore,
k19w3c39_truescore, k21w1c39_truescore,
k21w2c39_truescore, k21w3c39_truescore,
k23w1c39_truescore, k23w2c39_truescore,
k23w3c39_truescore, k25w1c39_truescore,
k25w2c39_truescore, k25w3c39_truescore,
k27w1c39_truescore, k27w2c39_truescore,
k27w3c39_truescore, k29w1c39_truescore,
k29w2c39_truescore, k29w3c39_truescore,
k31w1c39_truescore, k31w2c39_truescore,
k31w3c39_truescore, k33w1c39_truescore,
k33w2c39_truescore, k33w3c39_truescore,
k35w1c39_truescore, k35w2c39_truescore,
k35w3c39_truescore, k37w1c39_truescore,
k37w2c39_truescore, k37w3c39_truescore,
k39w1c39_truescore, k39w2c39_truescore,
k39w3c39_truescore))

```

`knitr::kable(c39_results[1:45,], caption = "c39_results")`

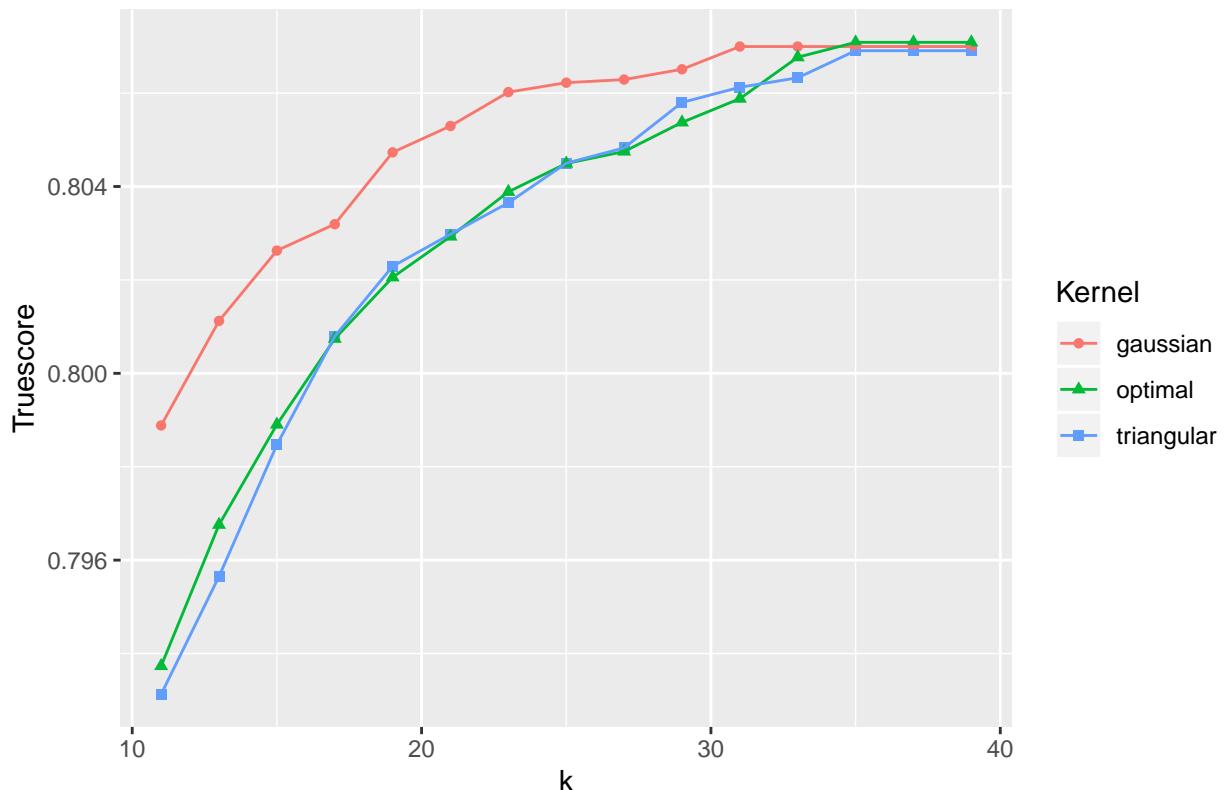
Table 48: c39_results

k	Kernel	Cut	TPR	TNR	Truescore
11	triangular	0.39	0.735693	0.860288	0.793127
11	gaussian	0.39	0.744980	0.861196	0.798884
11	optimal	0.39	0.737550	0.859181	0.793733
13	triangular	0.39	0.739608	0.860883	0.795651
13	gaussian	0.39	0.748795	0.861312	0.801122
13	optimal	0.39	0.741817	0.860486	0.796757
15	triangular	0.39	0.744026	0.861510	0.798470
15	gaussian	0.39	0.751707	0.860949	0.802628
15	optimal	0.39	0.744829	0.861444	0.798903
17	triangular	0.39	0.747791	0.861874	0.800790
17	gaussian	0.39	0.752661	0.860998	0.803193
17	optimal	0.39	0.747841	0.861692	0.800740
19	triangular	0.39	0.750552	0.861692	0.802291
19	gaussian	0.39	0.755622	0.860668	0.804731
19	optimal	0.39	0.750301	0.861477	0.802055
21	triangular	0.39	0.751958	0.861428	0.802979
21	gaussian	0.39	0.756325	0.861048	0.805296
21	optimal	0.39	0.751807	0.861510	0.802929
23	triangular	0.39	0.753263	0.861263	0.803651

k	Kernel	Cut	TPR	TNR	Truescore
23	gaussian	0.39	0.757631	0.861015	0.806021
23	optimal	0.39	0.753614	0.861345	0.803886
25	triangular	0.39	0.754920	0.861048	0.804499
25	gaussian	0.39	0.758384	0.860503	0.806223
25	optimal	0.39	0.754719	0.861296	0.804493
27	triangular	0.39	0.755472	0.861081	0.804827
27	gaussian	0.39	0.758735	0.860205	0.806290
27	optimal	0.39	0.755271	0.861163	0.804749
29	triangular	0.39	0.757329	0.860899	0.805800
29	gaussian	0.39	0.759137	0.860189	0.806510
29	optimal	0.39	0.756677	0.860767	0.805373
31	triangular	0.39	0.758032	0.860734	0.806125
31	gaussian	0.39	0.760040	0.860139	0.806997
31	optimal	0.39	0.757831	0.860437	0.805881
33	triangular	0.39	0.758584	0.860486	0.806328
33	gaussian	0.39	0.760040	0.860139	0.806997
33	optimal	0.39	0.759388	0.860453	0.806768
35	triangular	0.39	0.759739	0.860321	0.806908
35	gaussian	0.39	0.760040	0.860139	0.806997
35	optimal	0.39	0.759940	0.860470	0.807087
37	triangular	0.39	0.759739	0.860321	0.806908
37	gaussian	0.39	0.760040	0.860139	0.806997
37	optimal	0.39	0.759940	0.860470	0.807087
39	triangular	0.39	0.759739	0.860321	0.806908
39	gaussian	0.39	0.760040	0.860139	0.806997
39	optimal	0.39	0.759940	0.860470	0.807087

```
ggplot(c39_results, aes(k, Truescore, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.39 (Truescore)")
```

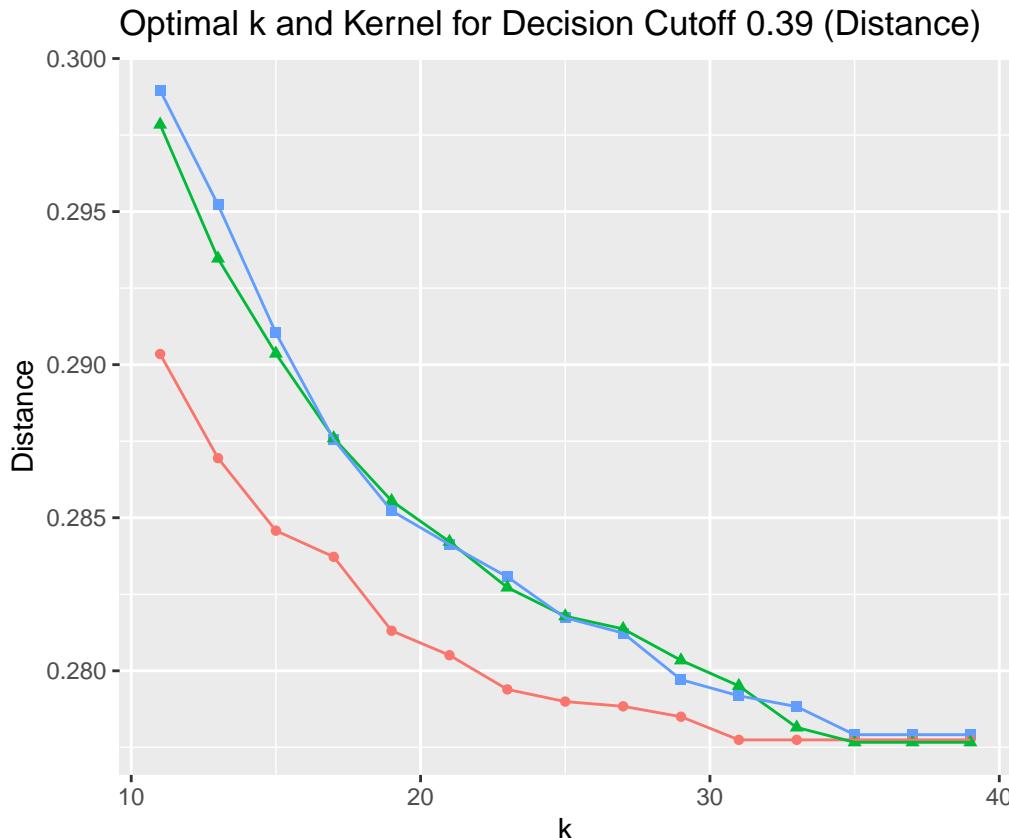
Optimal k and Kernel for Decision Cutoff 0.39 (Truescore)



```
# For the Cutoff of 0.39, compute the Minimum Distance to (0, 1) for each
# combination of k and Kernel.
```

```
c39_results <- c39_results %>% mutate(Distance = sqrt((1 - TPR)^2 + (1 - TNR)^2))

ggplot(c39_results, aes(k, Distance, shape = Kernel, color = Kernel)) +
  geom_point() + geom_line() +
  labs(title = "Optimal k and Kernel for Decision Cutoff 0.39 (Distance)")
```



```
# For the Cutoff of 0.39, identify the optimal combination of values for k and Kernel
# based on each of our assessment methods, Truescore and Minimum Distance to (0, 1).
```

```
max(c39_results$Truescore)
```

```
## [1] 0.807087
```

```
(c39_opt_k_ts <- c39_results$k[which.max(c39_results$Truescore)])
```

```
## [1] 35
```

```
(c39_opt_kernel_ts <- c39_results$Kernel[which.max(c39_results$Truescore)])
```

```
## [1] "optimal"
```

```
(c39_opt_cut_ts <- c39_results$Cut[which.max(c39_results$Truescore)])
```

```
## [1] 0.39
```

```
(c39_opt_tpr_ts <- c39_results$TPR[which.max(c39_results$Truescore)])
```

```
## [1] 0.75994
```

```

(c39_opt_tnr_ts <- c39_results$TNR[which.max(c39_results$Truescore)])

## [1] 0.86047

(c39_opt_d_ts <- c39_results$Distance[which.max(c39_results$Truescore)])

## [1] 0.2776642

min(c39_results$Distance)

## [1] 0.2776642

(c39_opt_k_dist <- c39_results$k[which.min(c39_results$Distance)])

## [1] 35

(c39_opt_kernel_dist <- c39_results$Kernel[which.min(c39_results$Distance)])

## [1] "optimal"

(c39_opt_cut_dist <- c39_results$Cut[which.min(c39_results$Distance)])

## [1] 0.39

(c39_opt_tpr_dist <- c39_results$TPR[which.min(c39_results$Distance)])

## [1] 0.75994

(c39_opt_tnr_dist <- c39_results$TNR[which.min(c39_results$Distance)])

## [1] 0.86047

(c39_opt_t_dist <- c39_results$Truescore[which.min(c39_results$Distance)])

## [1] 0.807087

```

For each cutoff that we tested, we had determined the model that generated the highest truescore and shortest distance. It was only one more step to identify which model among these best performers generated the very highest truescore and very shortest distance. That would, in turn, reveal the optimal combination of k, kernel, and cutoff for our weighted knn model.

```

# Compile the best performing models (based on Maximum Truescore and
# Shortest Distance) for every Cutoff that was tested.

results_optkKernelcut <- tibble(
  Model = c("c25k27tri_ts", "c25k21tri_d", "c26k33tri_ts", "c26k25tri_d",
            "c27k29gau_ts", "c27k29gau_d", "c28k23gau_ts", "c28k23gau_d",
            "c29k31gau_ts", "c29k23gau_d", "c30k35tri_ts", "c30k31tri_d",
            "c31k31gau_ts", "c31k31gau_d", "c32k31gau_ts", "c32k31gau_d",
            "c33k35tri_ts", "c33k35tri_d", "c34k31gau_ts", "c34k31gau_d",
            "c35k31gau_ts", "c35k31gau_d", "c36k31gau_ts", "c36k31gau_d",
            "c37k31gau_ts", "c37k31gau_d", "c38k35tri_ts", "c38k35tri_d",
            "c39k35opt_ts", "c39k35opt_d"),
  Criterion = rep(c("Truescore", "Distance"), 15),
  Cut = c("0.25", "0.25", "0.26", "0.26", "0.27", "0.27", "0.28", "0.28",
          "0.29", "0.29", "0.30", "0.30", "0.31", "0.31", "0.32", "0.32",
          "0.33", "0.33", "0.34", "0.34", "0.35", "0.35", "0.36", "0.36",
          "0.37", "0.37", "0.38", "0.38", "0.39", "0.39"),
  OptKs = c(c25_opt_k_ts, c25_opt_k_dist, c26_opt_k_ts, c26_opt_k_dist,
            c27_opt_k_ts, c27_opt_k_dist, c28_opt_k_ts, c28_opt_k_dist,
            c29_opt_k_ts, c29_opt_k_dist, c30_opt_k_ts, c30_opt_k_dist,
            c31_opt_k_ts, c31_opt_k_dist, c32_opt_k_ts, c32_opt_k_dist,
            c33_opt_k_ts, c33_opt_k_dist, c34_opt_k_ts, c34_opt_k_dist,
            c35_opt_k_ts, c35_opt_k_dist, c36_opt_k_ts, c36_opt_k_dist,
            c37_opt_k_ts, c37_opt_k_dist, c38_opt_k_ts, c38_opt_k_dist,
            c39_opt_k_ts, c39_opt_k_dist),
  OptKernels = c(c25_opt_kernel_ts, c25_opt_kernel_dist, c26_opt_kernel_ts,
                 c26_opt_kernel_dist, c27_opt_kernel_ts, c27_opt_kernel_dist,
                 c28_opt_kernel_ts, c28_opt_kernel_dist, c29_opt_kernel_ts,
                 c29_opt_kernel_dist, c30_opt_kernel_ts, c30_opt_kernel_dist,
                 c31_opt_kernel_ts, c31_opt_kernel_dist, c32_opt_kernel_ts,
                 c32_opt_kernel_dist, c33_opt_kernel_ts, c33_opt_kernel_dist,
                 c34_opt_kernel_ts, c34_opt_kernel_dist, c35_opt_kernel_ts,
                 c35_opt_kernel_dist, c36_opt_kernel_ts, c36_opt_kernel_dist,
                 c37_opt_kernel_ts, c37_opt_kernel_dist, c38_opt_kernel_ts,
                 c38_opt_kernel_dist, c39_opt_kernel_ts, c39_opt_kernel_dist),
  MaxTruescores = c(max(c25_results$Truescore), c25_opt_t_dist,
                    max(c26_results$Truescore), c26_opt_t_dist,
                    max(c27_results$Truescore), c27_opt_t_dist,
                    max(c28_results$Truescore), c28_opt_t_dist,
                    max(c29_results$Truescore), c29_opt_t_dist,
                    max(c30_results$Truescore), c30_opt_t_dist,
                    max(c31_results$Truescore), c31_opt_t_dist,
                    max(c32_results$Truescore), c32_opt_t_dist,
                    max(c33_results$Truescore), c33_opt_t_dist,
                    max(c34_results$Truescore), c34_opt_t_dist,
                    max(c35_results$Truescore), c35_opt_t_dist,
                    max(c36_results$Truescore), c36_opt_t_dist,
                    max(c37_results$Truescore), c37_opt_t_dist,
                    max(c38_results$Truescore), c38_opt_t_dist,
                    max(c39_results$Truescore), c39_opt_t_dist),
  MinDistances = c(c25_opt_d_ts, min(c25_results$Distance),
                    c26_opt_d_ts, min(c26_results$Distance),
                    c27_opt_d_ts, min(c27_results$Distance),

```

```

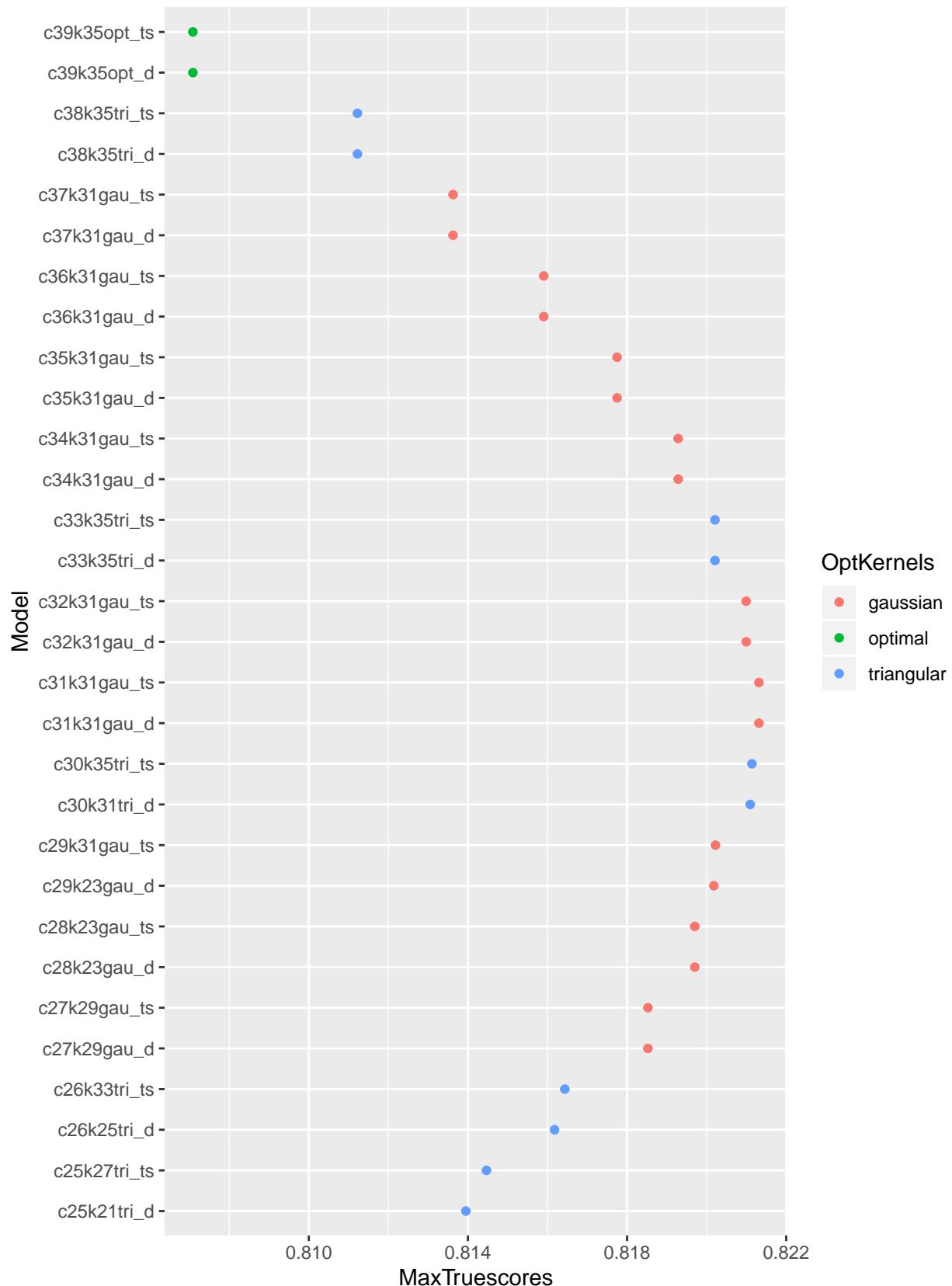
c28_opt_d_ts, min(c28_results$Distance),
c29_opt_d_ts, min(c29_results$Distance),
c30_opt_d_ts, min(c30_results$Distance),
c31_opt_d_ts, min(c31_results$Distance),
c32_opt_d_ts, min(c32_results$Distance),
c33_opt_d_ts, min(c33_results$Distance),
c34_opt_d_ts, min(c34_results$Distance),
c35_opt_d_ts, min(c35_results$Distance),
c36_opt_d_ts, min(c36_results$Distance),
c37_opt_d_ts, min(c37_results$Distance),
c38_opt_d_ts, min(c38_results$Distance),
c39_opt_d_ts, min(c39_results$Distance)),
TPR = c(c25_opt_tpr_ts, c25_opt_tpr_dist, c26_opt_tpr_ts, c26_opt_tpr_dist,
c27_opt_tpr_ts, c27_opt_tpr_dist, c28_opt_tpr_ts, c28_opt_tpr_dist,
c29_opt_tpr_ts, c29_opt_tpr_dist, c30_opt_tpr_ts, c30_opt_tpr_dist,
c31_opt_tpr_ts, c31_opt_tpr_dist, c32_opt_tpr_ts, c32_opt_tpr_dist,
c33_opt_tpr_ts, c33_opt_tpr_dist, c34_opt_tpr_ts, c34_opt_tpr_dist,
c35_opt_tpr_ts, c35_opt_tpr_dist, c36_opt_tpr_ts, c36_opt_tpr_dist,
c37_opt_tpr_ts, c37_opt_tpr_dist, c38_opt_tpr_ts, c38_opt_tpr_dist,
c39_opt_tpr_ts, c39_opt_tpr_dist),
TNR = c(c25_opt_tnr_ts, c25_opt_tnr_dist, c26_opt_tnr_ts, c26_opt_tnr_dist,
c27_opt_tnr_ts, c27_opt_tnr_dist, c28_opt_tnr_ts, c28_opt_tnr_dist,
c29_opt_tnr_ts, c29_opt_tnr_dist, c30_opt_tnr_ts, c30_opt_tnr_dist,
c31_opt_tnr_ts, c31_opt_tnr_dist, c32_opt_tnr_ts, c32_opt_tnr_dist,
c33_opt_tnr_ts, c33_opt_tnr_dist, c34_opt_tnr_ts, c34_opt_tnr_dist,
c35_opt_tnr_ts, c35_opt_tnr_dist, c36_opt_tnr_ts, c36_opt_tnr_dist,
c37_opt_tnr_ts, c37_opt_tnr_dist, c38_opt_tnr_ts, c38_opt_tnr_dist,
c39_opt_tnr_ts, c39_opt_tnr_dist))

# Identify values of k, Kernel and Cutoff that result in the highest Truescore and
# shortest Distance to (0, 1).

ggplot(results_optkKernelcut, aes(MaxTruescores, Model, color = OptKernels)) +
  geom_point() +
  labs(title = "Best Performing Model (by Truescore) for Each Cutoff")

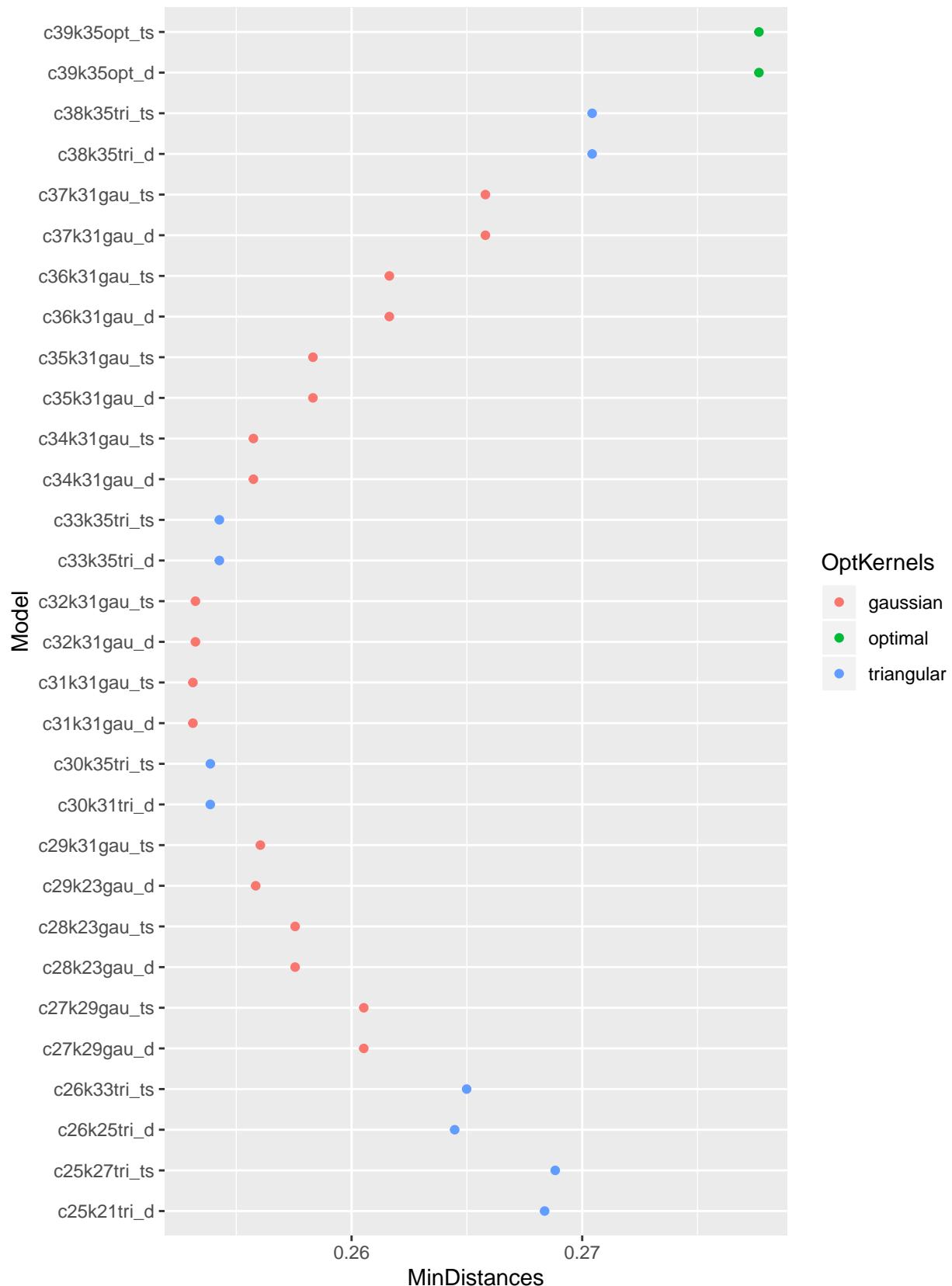
```

Best Performing Model (by Truescore) for Each Cutoff



```
ggplot(results_optkKernelcut, aes(MinDistances, Model, color = OptKernels)) +  
  geom_point() +  
  labs(title = "Best Performing Model (by Distance) for Each Cutoff")
```

Best Performing Model (by Distance) for Each Cutoff



```

max(results_optkKernelcut$MaxTruescores)

## [1] 0.821315

(kknn_opt_k <-
  results_optkKernelcut$OptKs[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] 31

(kknn_opt_kernel <-
  results_optkKernelcut$OptKernels[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] "gaussian"

(kknn_opt_cut <-
  results_optkKernelcut$Cut[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] "0.31"

(kknn_opt_tpr <-
  results_optkKernelcut$TPR[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] 0.835291

(kknn_opt_tnr <-
  results_optkKernelcut$TNR[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] 0.807799

(kknn_opt_d <-
  results_optkKernelcut$MinDistances[which.max(results_optkKernelcut$MaxTruescores)]) 

## [1] 0.2531211

```

Our analysis indicated that the best weighted knn model would have a neighborhood size of 31 with a gaussian kernel and a probability cutoff of 0.31. These were the parameter values that resulted in both the highest truescore and the shortest distance to $(0, 1)$, based on 10-fold cross-validation.

We next needed to fit a model with our new optimal parameters to the entire train set, and then use the model to predict outcomes based on the predictors in the test set.

```

# Fit an optimized kknn model (k = 31, Kernel = "gaussian"(w2)) to the
# entire predictor matrix and outcome vector (train_set2_orig). Then,
# apply the kknn model (kknn_fitval_k31w2) and optimal cutoff of 0.31
# to predict class probabilities based on the predictors in test_set2_orig.

train_set2_orig <- train_set[, c("launch_angle", "launch_speed", "spray_angle_Kolp",
                                 "spray_angle_adj", "hp_to_1b", "num_if_alignment",

```

```

            "events")]
test_set2_orig <- test_set[, c("launch_angle", "launch_speed", "spray_angle_Kolp",
                             "spray_angle_adj", "hp_to_1b", "num_if_alignment",
                             "events")]

kknn_fitval_k31w2 <- kknn(formula = events ~ ., train = train_set2_orig,
                           test = test_set2_orig, k = 31, kernel = "gaussian")

kknn_y_hat_prob_tbl <- as_tibble(kknn_fitval_k31w2$prob)
kknn_y_hat_prob_tbl <- kknn_y_hat_prob_tbl %>%
  mutate(preds = ifelse(kknn_y_hat_prob_tbl$single > 0.31, "single", "field_out"))
kknn_y_hat_prob_tbl$preds <- factor(kknn_y_hat_prob_tbl$preds,
                                      levels = c("single", "field_out"))

```

We were now in a position to see whether our weighted knn model outperformed our standard knn model.

```

# Assess the predictive ability of our weighted knn (kknn) model (kknn_fitval_k31w2c31).

confusionMatrix(data = kknn_y_hat_prob_tbl$preds, reference = test_set2_orig$events)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction  single field_out
##   single      4208     2940
##   field_out    773     12197
##
##          Accuracy : 0.8154
##                 95% CI : (0.81, 0.8208)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.5677
##
##     Mcnemar's Test P-Value : < 2.2e-16
##
##          Sensitivity : 0.8448
##          Specificity : 0.8058
##     Pos Pred Value : 0.5887
##     Neg Pred Value : 0.9404
##          Prevalence : 0.2476
##     Detection Rate : 0.2092
##     Detection Prevalence : 0.3553
##          Balanced Accuracy : 0.8253
##
##     'Positive' Class : single
##

(kknn_k31w2c31_tpr <- round(sensitivity(kknn_y_hat_prob_tbl$preds,
                                             reference = factor(test_set2_orig$events)), 6))

## [1] 0.84481

```

```

(kknn_k31w2c31_tnr <- round(specificity(kknn_y_hat_prob_tbl$preds,
                                         reference = factor(test_set2_orig$events)), 6))

## [1] 0.805774

(kknn_k31w2c31_fpr <- round(1 - kknn_k31w2c31_tnr, 6))

## [1] 0.194226

(kknn_k31w2c31_truescore <- round((2 * kknn_k31w2c31_tpr * kknn_k31w2c31_tnr) /
                                         (kknn_k31w2c31_tpr + kknn_k31w2c31_tnr) , 6))

## [1] 0.82483

(kknn_k31w2c31_distance <-
  round(sqrt((1 - kknn_k31w2c31_tpr)^2 + (kknn_k31w2c31_fpr)^2), 6))

## [1] 0.248611

assessment_results <- tibble(Model = c("Baseline", "Log Regression", "Log Regression",
                                         "knn_k5", "knn_k7", "knn_k7c40", "knn_k23c31",
                                         "kknn_k31w2c31"),
                                `Cutoff Method` = c(NA, "Truescore", "Distance",
                                                   "default (0.50)", "default (0.50)",
                                                   "both", "both", "both"),
                                `Truescore` = c(bl_truescore, max_truescore,
                                                min_distance_truescore, knn_k5_truescore,
                                                knn_k7_truescore, knn_k7c40_truescore,
                                                knn_k23c31_truescore,
                                                kknn_k31w2c31_truescore),
                                TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                         knn_k5_tpr, knn_k7_tpr, knn_k7c40_tpr,
                                         knn_k23c31_tpr, kknn_k31w2c31_tpr),
                                TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                         knn_k5_tnr, knn_k7_tnr, knn_k7c40_tnr,
                                         knn_k23c31_tnr, kknn_k31w2c31_tnr),
                                Distance = c(NA, max_truescore_distance, min_distance,
                                             knn_k5_distance, knn_k7_distance,
                                             knn_k7c40_distance, knn_k23c31_distance,
                                             kknn_k31w2c31_distance),
                                FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                         knn_k5_fpr, knn_k7_fpr, knn_k7c40_fpr,
                                         knn_k23c31_fpr, kknn_k31w2c31_fpr),
                                `Best Cutoff` = c(NA, max_truescore_cutoff,
                                                 min_distance_cutoff, "default",
                                                 "default", knn_k7c40_cutoff,
                                                 knn_opt_cut, 0.31))
knitr::kable(assessment_results[1:8, ], caption = "Assessment Results")

```

Table 49: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log Regression	Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
knn_k5	default (0.50)	0.746224	0.634812	0.905067	0.377326	0.094933	default
knn_k7	default (0.50)	0.749218	0.635615	0.912268	0.374798	0.087732	default
knn_k7c40	both	0.805345	0.772937	0.840589	0.277434	0.159411	0.4
knn_k23c31	both	0.820183	0.824935	0.815485	0.254349	0.184515	0.31
kknn_k31w2c31	both	0.824830	0.844810	0.805774	0.248611	0.194226	0.31

Our weighted knn model (kknn_k31w2c31) did indeed outperform our standard knn model (knn_k23c31), but only marginally. The truescore improved by .004647, to 0.824830, while the distance improved by .005738, to 0.248611. These were our new targets. The model's TPR and TNR were 0.844810 and 0.805774, respectively. The typical bias in favor of the majority class had been overcome, and then some.

Our trek through the land of k-nearest neighbor models demonstrated the power of treating the decision threshold as another parameter to be optimized. The greatest progress resulted not from altering the value of k, nor did it come from altering the kernel used for weighting, nor did it come from the use of 10-fold cross-validation; instead, it came from changing the cutoff used to decide whether a given probability pointed to one class or another.

Could other algorithms approach the predictive ability of our weighted knn model?

Build and Assess a Classification Tree Model

We next sought to develop a decision or classification tree with an optimized complexity parameter (cp).

One appealing characteristic of classification trees is that they mimic the human decision making process. A classification tree makes a decision (prediction) based on the answers to a series of questions. Each question further reduces the scope of possibilities until a satisfactory level of certainty is reached.

In R, a classification tree model can be built with the rpart function (Recursive Partitioning and Regression Trees). The algorithm “split[s] the data recursively, which means that the subsets that arise from a split are further split until a predetermined termination criterion is reached. At each step, the split is made based on the independent variable that results in the largest possible reduction in heterogeneity of the dependent (predicted) variable.”¹⁵

There are different methods for splitting the data to achieve homogeneity. We opted for the Gini Index, which is rpart’s default method. A pure table consisting of a single class has a Gini Index of 0. When all classes have an equal probability, the Gini Index is 1.

The initial tree was built using 10-fold cross validation. We also specified the minimum number of observations in any terminal node to be 2, with a complexity parameter (cp) of 0. The cp allows the user to specify the extent to which a split must improve the fit to avoid being pruned off during cross-validation. Our parameters were set to permit the construction of a complete or near-complete tree that could be pruned later with the optimal cp identified in this first step of the algorithm.

Our imbalanced classes, once again, presented our primary challenge. The standard decision tree algorithm assumes the optimal value of cp is that which minimizes the misclassification rate (False Positives + False Negatives / Number of Observations). As we’ve seen, this approach can lead to underperforming models when classes are not balanced. We’ve also seen that one way to accommodate imbalanced classes is to

¹⁵ Awati, Kailash, “A gentle introduction to decision trees using R” (2016). <https://eight2late.wordpress.com/2016/02/16/a-gentle-introduction-to-decision-trees-using-r/>

identify the value of the parameter (cp in this case) that maximizes the truescore or minimizes the distance to (0, 1). This customized approach worked well for our final k-Nearest Neighbors Model, and would, no doubt, work well here for identifying the optimal value of cp for pruning a classification tree. But some implementations of standard machine learning algorithms also provide optional parameters that can counter the majority class bias created by imbalanced classes. The rpart package is one such implementation.

We decided to use the rpart function's loss parameter to counter the effects of our unbalanced classes. The loss parameter allows a user to specify a loss matrix that defines the relative cost of making different types of prediction mistakes. Our singles class represented only 25% of our observations, so our loss matrix effectively informed rpart that it's twice as costly to generate a false negative (predict a single to be an out) than a false positive (predict an out to be a single). Our hope was that the resulting cp would produce a pruned tree that predicted, better than our previous models (and with the help of an optimal decision cutoff), the outcomes of the observations in our test set.

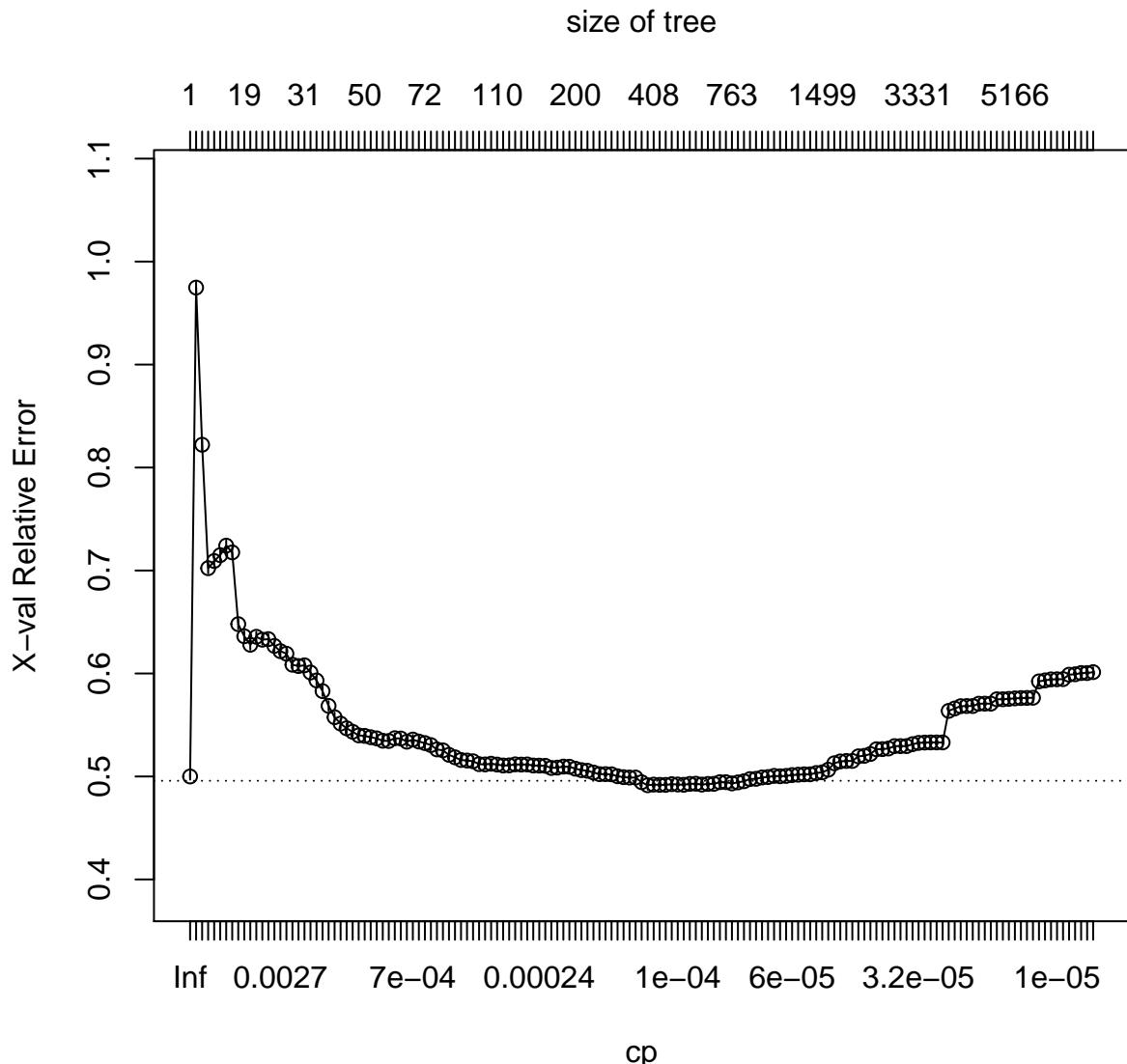
```
# Build a complete tree with cp set to 0.

train_set3 <- dplyr::select(train_set2, events, launch_angle, launch_speed,
                           spray_angle_Kolp, spray_angle_adj, if_fielding_alignment,
                           hp_to_1b)

set.seed(1)
ctree_train <- rpart(events ~ ., method = "class", data = train_set3,
                      parms = list(loss = matrix(c(0, 2, 1, 0), byrow = TRUE, nrow = 2)),
                      control = rpart.control(xval = 10, minbucket = 2, cp = 0))
```

Having constructed a complete tree, we followed a multi-step process to identify the optimal value of cp for our final model. In the final step of the process, we used what is referred to as “Breiman’s one standard error rule” to identify optimal cp. In the context of classification trees, the rule directs the selection of the smallest tree whose error is no more than one standard error above the error of the best tree. Following this rule, the smaller tree would not be worse than the best tree in a statistically significant way.

```
plotcp(ctree_train)
```



```

ctree_train$cptable

# Find the lowest estimated cross-validated error (xerror), along with
# its standard error (xstd).

(cp_min_xerror_index <- which.min(ctree_train$cptable[, "xerror"]))

## 77
## 77

(cp_min_xerror <- ctree_train$cptable[cp_min_xerror_index, "xerror"])

## [1] 0.4914659

```

```

(cp_min_xerror_xstd <- ctree_train$cptable[cp_min_xerror_index, "xstd"])

## [1] 0.004334567

# Identify the highest CP (the smallest tree) whose xerror is still within one
# standard error of the lowest xerror (Breiman's "one standard error rule").

(cp_max_range_xerror <- cp_min_xerror + cp_min_xerror_xstd)

## [1] 0.4958004

(cp_min_range_xerror <- cp_min_xerror - cp_min_xerror_xstd)

## [1] 0.4871313

(opt_cp <- max(ctree_train$cptable[, "CP"] [ctree_train$cptable[, "xerror"] < cp_max_range_xerror]))

## [1] 0.0001338688

```

We now used our optimal cp of 0.0001338688 to prune our tree. The `prune()` function recursively snips off the least important splits, based on the complexity parameter (cp). This helps avoid overfitting the model.

```

# Use opt_cp to prune our classification tree.

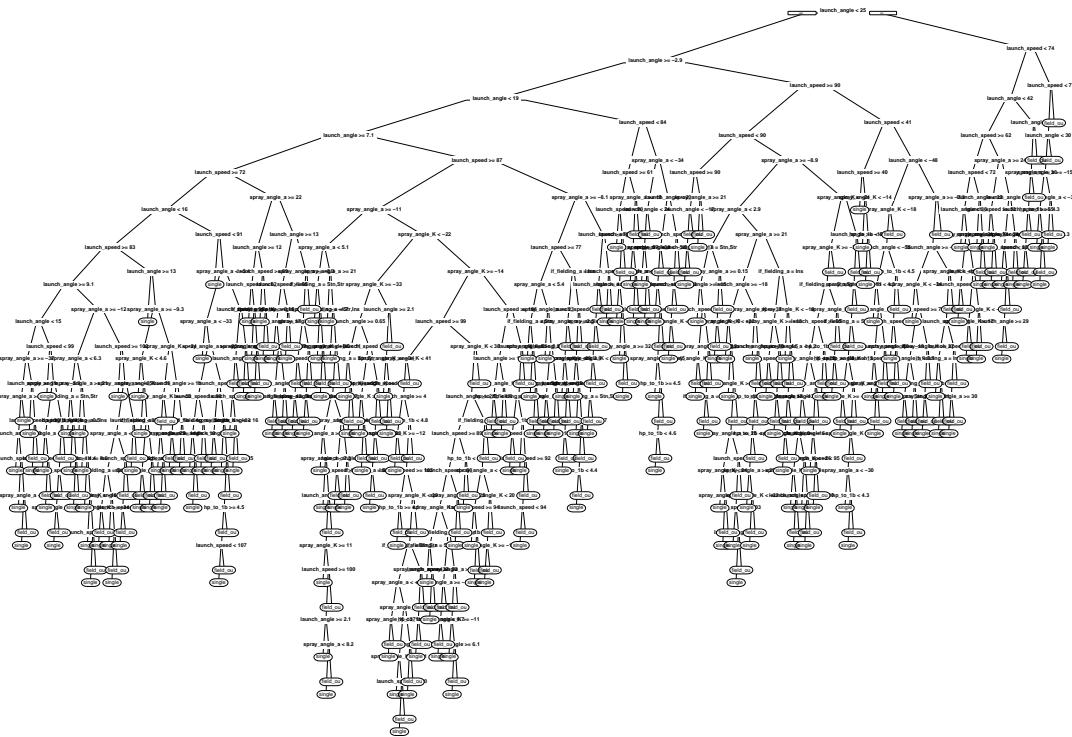
ctree_pruned_train <- prune(ctree_train, opt_cp)

# Review and visualize our pruned classification tree object.

print(ctree_pruned_train)

prp(ctree_pruned_train)

```



```
# If the above plot is generated in Rstudio, the text can be made
# readable by exporting it and saving it as a pdf file that can be enlarged.
```

We next needed to identify the optimal decision cutoff to be used for predicting the outcomes of the observations in the test set. For this purpose, we generated probabilities by using caret::predict against the entire train set.

```
# Use our pruned classification tree and the train set to generate
# probabilities and predict outcomes across a range of cutoffs (0.15 to 0.39).
```

```
ctree_pruned_probs <-
  predict(ctree_pruned_train, method = "prob") # Omitting the newdata argument is
                                                # equivalent to predicting against
                                                # the train set.

ctree_pruned_probs <- as_tibble(ctree_pruned_probs)

ctree_pruned_probs <- ctree_pruned_probs %>%
  mutate(pred15 = ifelse(ctree_pruned_probs$single > 0.15, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
  mutate(pred16 = ifelse(ctree_pruned_probs$single > 0.16, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
  mutate(pred17 = ifelse(ctree_pruned_probs$single > 0.17, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
  mutate(pred18 = ifelse(ctree_pruned_probs$single > 0.18, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
```

```

    mutate(pred19 = ifelse(ctree_pruned_probs$single > 0.19, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred20 = ifelse(ctree_pruned_probs$single > 0.20, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred21 = ifelse(ctree_pruned_probs$single > 0.21, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred22 = ifelse(ctree_pruned_probs$single > 0.22, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred23 = ifelse(ctree_pruned_probs$single > 0.23, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred24 = ifelse(ctree_pruned_probs$single > 0.24, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred25 = ifelse(ctree_pruned_probs$single > 0.25, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred26 = ifelse(ctree_pruned_probs$single > 0.26, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred27 = ifelse(ctree_pruned_probs$single > 0.27, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred28 = ifelse(ctree_pruned_probs$single > 0.28, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred29 = ifelse(ctree_pruned_probs$single > 0.29, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred30 = ifelse(ctree_pruned_probs$single > 0.30, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred31 = ifelse(ctree_pruned_probs$single > 0.31, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred32 = ifelse(ctree_pruned_probs$single > 0.32, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred33 = ifelse(ctree_pruned_probs$single > 0.33, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred34 = ifelse(ctree_pruned_probs$single > 0.34, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred35 = ifelse(ctree_pruned_probs$single > 0.35, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred36 = ifelse(ctree_pruned_probs$single > 0.36, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred37 = ifelse(ctree_pruned_probs$single > 0.37, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred38 = ifelse(ctree_pruned_probs$single > 0.38, "single", "field_out"))
ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate(pred39 = ifelse(ctree_pruned_probs$single > 0.39, "single", "field_out"))

# Convert all character columns to factors using dplyr package
# (https://gist.github.com/ramhiser/character2factor.r).

ctree_pruned_probs <- ctree_pruned_probs %>%
    mutate_if(sapply(ctree_pruned_probs, is.character), as.factor)

sapply(ctree_pruned_probs, class)

# Make sure all of the factor outcomes have the same levels in the same order.

ctree_pruned_probs$pred15 <- factor(ctree_pruned_probs$pred15,

```

```

            levels = c("single", "field_out"))
ctree_pruned_probs$pred16 <- factor(ctree_pruned_probs$pred16,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred17 <- factor(ctree_pruned_probs$pred17,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred18 <- factor(ctree_pruned_probs$pred18,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred19 <- factor(ctree_pruned_probs$pred19,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred20 <- factor(ctree_pruned_probs$pred20,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred21 <- factor(ctree_pruned_probs$pred21,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred22 <- factor(ctree_pruned_probs$pred22,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred23 <- factor(ctree_pruned_probs$pred23,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred24 <- factor(ctree_pruned_probs$pred24,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred25 <- factor(ctree_pruned_probs$pred25,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred26 <- factor(ctree_pruned_probs$pred26,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred27 <- factor(ctree_pruned_probs$pred27,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred28 <- factor(ctree_pruned_probs$pred28,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred29 <- factor(ctree_pruned_probs$pred29,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred30 <- factor(ctree_pruned_probs$pred30,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred31 <- factor(ctree_pruned_probs$pred31,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred32 <- factor(ctree_pruned_probs$pred32,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred33 <- factor(ctree_pruned_probs$pred33,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred34 <- factor(ctree_pruned_probs$pred34,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred35 <- factor(ctree_pruned_probs$pred35,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred36 <- factor(ctree_pruned_probs$pred36,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred37 <- factor(ctree_pruned_probs$pred37,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred38 <- factor(ctree_pruned_probs$pred38,
                                      levels = c("single", "field_out"))
ctree_pruned_probs$pred39 <- factor(ctree_pruned_probs$pred39,
                                      levels = c("single", "field_out"))
sapply(ctree_pruned_probs, levels)

# Add a column to ctree_pruned_probs containing the actual outcomes

```

```
ctree_pruned_probs <- add_column(ctree_pruned_probs, obs = train_set3$events,
                                .before = "single")
```

The optimal Cutoff would be that which maximized the truescore or minimized the distance to (0, 1).

```
# Compute the Truescore and Distance for each cutoff (0.15 to 0.39).

c15_tpr <- sensitivity(ctree_pruned_probs$pred15, ctree_pruned_probs$obs)
c15_tnr <- specificity(ctree_pruned_probs$pred15, ctree_pruned_probs$obs)
c15_truescore <- round((2 * c15_tpr * c15_tnr) / (c15_tpr + c15_tnr), 6)
c15_distance <- round(sqrt((1 - c15_tpr)^2 + (1 - c15_tnr)^2), 6)

c16_tpr <- sensitivity(ctree_pruned_probs$pred16, ctree_pruned_probs$obs)
c16_tnr <- specificity(ctree_pruned_probs$pred16, ctree_pruned_probs$obs)
c16_truescore <- round((2 * c16_tpr * c16_tnr) / (c16_tpr + c16_tnr), 6)
c16_distance <- round(sqrt((1 - c16_tpr)^2 + (1 - c16_tnr)^2), 6)

c17_tpr <- sensitivity(ctree_pruned_probs$pred17, ctree_pruned_probs$obs)
c17_tnr <- specificity(ctree_pruned_probs$pred17, ctree_pruned_probs$obs)
c17_truescore <- round((2 * c17_tpr * c17_tnr) / (c17_tpr + c17_tnr), 6)
c17_distance <- round(sqrt((1 - c17_tpr)^2 + (1 - c17_tnr)^2), 6)

c18_tpr <- sensitivity(ctree_pruned_probs$pred18, ctree_pruned_probs$obs)
c18_tnr <- specificity(ctree_pruned_probs$pred18, ctree_pruned_probs$obs)
c18_truescore <- round((2 * c18_tpr * c18_tnr) / (c18_tpr + c18_tnr), 6)
c18_distance <- round(sqrt((1 - c18_tpr)^2 + (1 - c18_tnr)^2), 6)

c19_tpr <- sensitivity(ctree_pruned_probs$pred19, ctree_pruned_probs$obs)
c19_tnr <- specificity(ctree_pruned_probs$pred19, ctree_pruned_probs$obs)
c19_truescore <- round((2 * c19_tpr * c19_tnr) / (c19_tpr + c19_tnr), 6)
c19_distance <- round(sqrt((1 - c19_tpr)^2 + (1 - c19_tnr)^2), 6)

c20_tpr <- sensitivity(ctree_pruned_probs$pred20, ctree_pruned_probs$obs)
c20_tnr <- specificity(ctree_pruned_probs$pred20, ctree_pruned_probs$obs)
c20_truescore <- round((2 * c20_tpr * c20_tnr) / (c20_tpr + c20_tnr), 6)
c20_distance <- round(sqrt((1 - c20_tpr)^2 + (1 - c20_tnr)^2), 6)

c21_tpr <- sensitivity(ctree_pruned_probs$pred21, ctree_pruned_probs$obs)
c21_tnr <- specificity(ctree_pruned_probs$pred21, ctree_pruned_probs$obs)
c21_truescore <- round((2 * c21_tpr * c21_tnr) / (c21_tpr + c21_tnr), 6)
c21_distance <- round(sqrt((1 - c21_tpr)^2 + (1 - c21_tnr)^2), 6)

c22_tpr <- sensitivity(ctree_pruned_probs$pred22, ctree_pruned_probs$obs)
c22_tnr <- specificity(ctree_pruned_probs$pred22, ctree_pruned_probs$obs)
c22_truescore <- round((2 * c22_tpr * c22_tnr) / (c22_tpr + c22_tnr), 6)
c22_distance <- round(sqrt((1 - c22_tpr)^2 + (1 - c22_tnr)^2), 6)

c23_tpr <- sensitivity(ctree_pruned_probs$pred23, ctree_pruned_probs$obs)
c23_tnr <- specificity(ctree_pruned_probs$pred23, ctree_pruned_probs$obs)
c23_truescore <- round((2 * c23_tpr * c23_tnr) / (c23_tpr + c23_tnr), 6)
c23_distance <- round(sqrt((1 - c23_tpr)^2 + (1 - c23_tnr)^2), 6)

c24_tpr <- sensitivity(ctree_pruned_probs$pred24, ctree_pruned_probs$obs)
```

```

c24_tnr <- specificity(ctree_pruned_probs$pred24, ctree_pruned_probs$obs)
c24_truescore <- round((2 * c24_tpr * c24_tnr) / (c24_tpr + c24_tnr), 6)
c24_distance <- round(sqrt((1 - c24_tpr)^2 + (1 - c24_tnr)^2), 6)

c25_tpr <- sensitivity(ctree_pruned_probs$pred25, ctree_pruned_probs$obs)
c25_tnr <- specificity(ctree_pruned_probs$pred25, ctree_pruned_probs$obs)
c25_truescore <- round((2 * c25_tpr * c25_tnr) / (c25_tpr + c25_tnr), 6)
c25_distance <- round(sqrt((1 - c25_tpr)^2 + (1 - c25_tnr)^2), 6)

c26_tpr <- sensitivity(ctree_pruned_probs$pred26, ctree_pruned_probs$obs)
c26_tnr <- specificity(ctree_pruned_probs$pred26, ctree_pruned_probs$obs)
c26_truescore <- round((2 * c26_tpr * c26_tnr) / (c26_tpr + c26_tnr), 6)
c26_distance <- round(sqrt((1 - c26_tpr)^2 + (1 - c26_tnr)^2), 6)

c27_tpr <- sensitivity(ctree_pruned_probs$pred27, ctree_pruned_probs$obs)
c27_tnr <- specificity(ctree_pruned_probs$pred27, ctree_pruned_probs$obs)
c27_truescore <- round((2 * c27_tpr * c27_tnr) / (c27_tpr + c27_tnr), 6)
c27_distance <- round(sqrt((1 - c27_tpr)^2 + (1 - c27_tnr)^2), 6)

c28_tpr <- sensitivity(ctree_pruned_probs$pred28, ctree_pruned_probs$obs)
c28_tnr <- specificity(ctree_pruned_probs$pred28, ctree_pruned_probs$obs)
c28_truescore <- round((2 * c28_tpr * c28_tnr) / (c28_tpr + c28_tnr), 6)
c28_distance <- round(sqrt((1 - c28_tpr)^2 + (1 - c28_tnr)^2), 6)

c29_tpr <- sensitivity(ctree_pruned_probs$pred29, ctree_pruned_probs$obs)
c29_tnr <- specificity(ctree_pruned_probs$pred29, ctree_pruned_probs$obs)
c29_truescore <- round((2 * c29_tpr * c29_tnr) / (c29_tpr + c29_tnr), 6)
c29_distance <- round(sqrt((1 - c29_tpr)^2 + (1 - c29_tnr)^2), 6)

c30_tpr <- sensitivity(ctree_pruned_probs$pred30, ctree_pruned_probs$obs)
c30_tnr <- specificity(ctree_pruned_probs$pred30, ctree_pruned_probs$obs)
c30_truescore <- round((2 * c30_tpr * c30_tnr) / (c30_tpr + c30_tnr), 6)
c30_distance <- round(sqrt((1 - c30_tpr)^2 + (1 - c30_tnr)^2), 6)

c31_tpr <- sensitivity(ctree_pruned_probs$pred31, ctree_pruned_probs$obs)
c31_tnr <- specificity(ctree_pruned_probs$pred31, ctree_pruned_probs$obs)
c31_truescore <- round((2 * c31_tpr * c31_tnr) / (c31_tpr + c31_tnr), 6)
c31_distance <- round(sqrt((1 - c31_tpr)^2 + (1 - c31_tnr)^2), 6)

c32_tpr <- sensitivity(ctree_pruned_probs$pred32, ctree_pruned_probs$obs)
c32_tnr <- specificity(ctree_pruned_probs$pred32, ctree_pruned_probs$obs)
c32_truescore <- round((2 * c32_tpr * c32_tnr) / (c32_tpr + c32_tnr), 6)
c32_distance <- round(sqrt((1 - c32_tpr)^2 + (1 - c32_tnr)^2), 6)

c33_tpr <- sensitivity(ctree_pruned_probs$pred33, ctree_pruned_probs$obs)
c33_tnr <- specificity(ctree_pruned_probs$pred33, ctree_pruned_probs$obs)
c33_truescore <- round((2 * c33_tpr * c33_tnr) / (c33_tpr + c33_tnr), 6)
c33_distance <- round(sqrt((1 - c33_tpr)^2 + (1 - c33_tnr)^2), 6)

c34_tpr <- sensitivity(ctree_pruned_probs$pred34, ctree_pruned_probs$obs)
c34_tnr <- specificity(ctree_pruned_probs$pred34, ctree_pruned_probs$obs)
c34_truescore <- round((2 * c34_tpr * c34_tnr) / (c34_tpr + c34_tnr), 6)
c34_distance <- round(sqrt((1 - c34_tpr)^2 + (1 - c34_tnr)^2), 6)

```

```

c35_tpr <- sensitivity(ctree_pruned_probs$pred35, ctree_pruned_probs$obs)
c35_tnr <- specificity(ctree_pruned_probs$pred35, ctree_pruned_probs$obs)
c35_truescore <- round((2 * c35_tpr * c35_tnr) / (c35_tpr + c35_tnr), 6)
c35_distance <- round(sqrt((1 - c35_tpr)^2 + (1 - c35_tnr)^2), 6)

c36_tpr <- sensitivity(ctree_pruned_probs$pred36, ctree_pruned_probs$obs)
c36_tnr <- specificity(ctree_pruned_probs$pred36, ctree_pruned_probs$obs)
c36_truescore <- round((2 * c36_tpr * c36_tnr) / (c36_tpr + c36_tnr), 6)
c36_distance <- round(sqrt((1 - c36_tpr)^2 + (1 - c36_tnr)^2), 6)

c37_tpr <- sensitivity(ctree_pruned_probs$pred37, ctree_pruned_probs$obs)
c37_tnr <- specificity(ctree_pruned_probs$pred37, ctree_pruned_probs$obs)
c37_truescore <- round((2 * c37_tpr * c37_tnr) / (c37_tpr + c37_tnr), 6)
c37_distance <- round(sqrt((1 - c37_tpr)^2 + (1 - c37_tnr)^2), 6)

c38_tpr <- sensitivity(ctree_pruned_probs$pred38, ctree_pruned_probs$obs)
c38_tnr <- specificity(ctree_pruned_probs$pred38, ctree_pruned_probs$obs)
c38_truescore <- round((2 * c38_tpr * c38_tnr) / (c38_tpr + c38_tnr), 6)
c38_distance <- round(sqrt((1 - c38_tpr)^2 + (1 - c38_tnr)^2), 6)

c39_tpr <- sensitivity(ctree_pruned_probs$pred39, ctree_pruned_probs$obs)
c39_tnr <- specificity(ctree_pruned_probs$pred39, ctree_pruned_probs$obs)
c39_truescore <- round((2 * c39_tpr * c39_tnr) / (c39_tpr + c39_tnr), 6)
c39_distance <- round(sqrt((1 - c39_tpr)^2 + (1 - c39_tnr)^2), 6)

# Compile the cutoff results in a table.

cutoff_results <- tibble(cp = opt_cp,
                          cut = seq(0.15, 0.39, 0.01),
                          tpr = c(c15_tpr, c16_tpr, c17_tpr, c18_tpr, c19_tpr,
                                  c20_tpr, c21_tpr, c22_tpr, c23_tpr, c24_tpr,
                                  c25_tpr, c26_tpr, c27_tpr, c28_tpr, c29_tpr,
                                  c30_tpr, c31_tpr, c32_tpr, c33_tpr, c34_tpr,
                                  c35_tpr, c36_tpr, c37_tpr, c38_tpr, c39_tpr),
                          tnr = c(c15_tnr, c16_tnr, c17_tnr, c18_tnr, c19_tnr,
                                  c20_tnr, c21_tnr, c22_tnr, c23_tnr, c24_tnr,
                                  c25_tnr, c26_tnr, c27_tnr, c28_tnr, c29_tnr,
                                  c30_tnr, c31_tnr, c32_tnr, c33_tnr, c34_tnr,
                                  c35_tnr, c36_tnr, c37_tnr, c38_tnr, c39_tnr),
                          truescore = c(c15_truescore, c16_truescore, c17_truescore,
                                        c18_truescore, c19_truescore, c20_truescore,
                                        c21_truescore, c22_truescore, c23_truescore,
                                        c24_truescore, c25_truescore, c26_truescore,
                                        c27_truescore, c28_truescore, c29_truescore,
                                        c30_truescore, c31_truescore, c32_truescore,
                                        c33_truescore, c34_truescore, c35_truescore,
                                        c36_truescore, c37_truescore, c38_truescore,
                                        c39_truescore),
                          distance = c(c15_distance, c16_distance, c17_distance,
                                       c18_distance, c19_distance, c20_distance,
                                       c21_distance, c22_distance, c23_distance,
                                       c24_distance, c25_distance, c26_distance,
                                       c27_distance, c28_distance, c29_distance,

```

```

c30_distance, c31_distance, c32_distance,
c33_distance, c34_distance, c35_distance,
c36_distance, c37_distance, c38_distance,
c39_distance))

knitr::kable(cutoff_results[1:25, ], caption = "Cutoff Results")

```

Table 50: Cutoff Results

cp	cut	tpr	tnr	truescore	distance
0.0001339	0.15	0.8716365	0.8172959	0.843592	0.223289
0.0001339	0.16	0.8648092	0.8294520	0.846762	0.217631
0.0001339	0.17	0.8570783	0.8424339	0.849693	0.212729
0.0001339	0.18	0.8518072	0.8505269	0.851167	0.210484
0.0001339	0.19	0.8471386	0.8573481	0.852213	0.209084
0.0001339	0.20	0.8421185	0.8641364	0.852985	0.208292
0.0001339	0.21	0.8377510	0.8696528	0.853404	0.208123
0.0001339	0.22	0.8320783	0.8764741	0.853699	0.208462
0.0001339	0.23	0.8291667	0.8797278	0.853699	0.208924
0.0001339	0.24	0.8288655	0.8800581	0.853695	0.208981
0.0001339	0.25	0.8223896	0.8866151	0.853296	0.210717
0.0001339	0.26	0.8223896	0.8866151	0.853296	0.210717
0.0001339	0.27	0.8223896	0.8866151	0.853296	0.210717
0.0001339	0.28	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.29	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.30	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.31	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.32	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.33	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.34	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.35	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.36	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.37	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.38	0.8219880	0.8869620	0.853240	0.210869
0.0001339	0.39	0.8219880	0.8869620	0.853240	0.210869

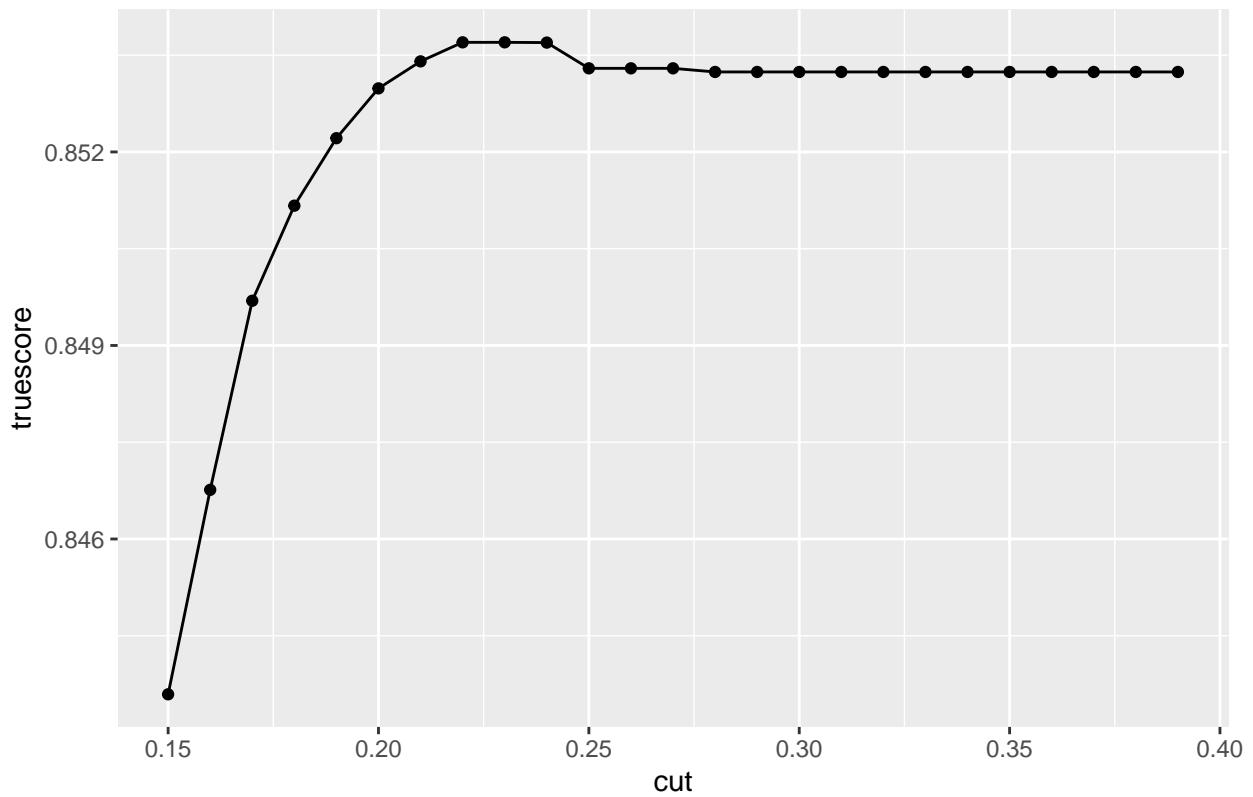
```

# Identify the optimal cutoff based on each of our assessment methods,
# Truescore and Minimum Distance to (0, 1).

ggplot(cutoff_results, aes(cut, truescore)) +
  geom_point() + geom_line() +
  labs(title = "Optimal cp and cutoff by Truescore")

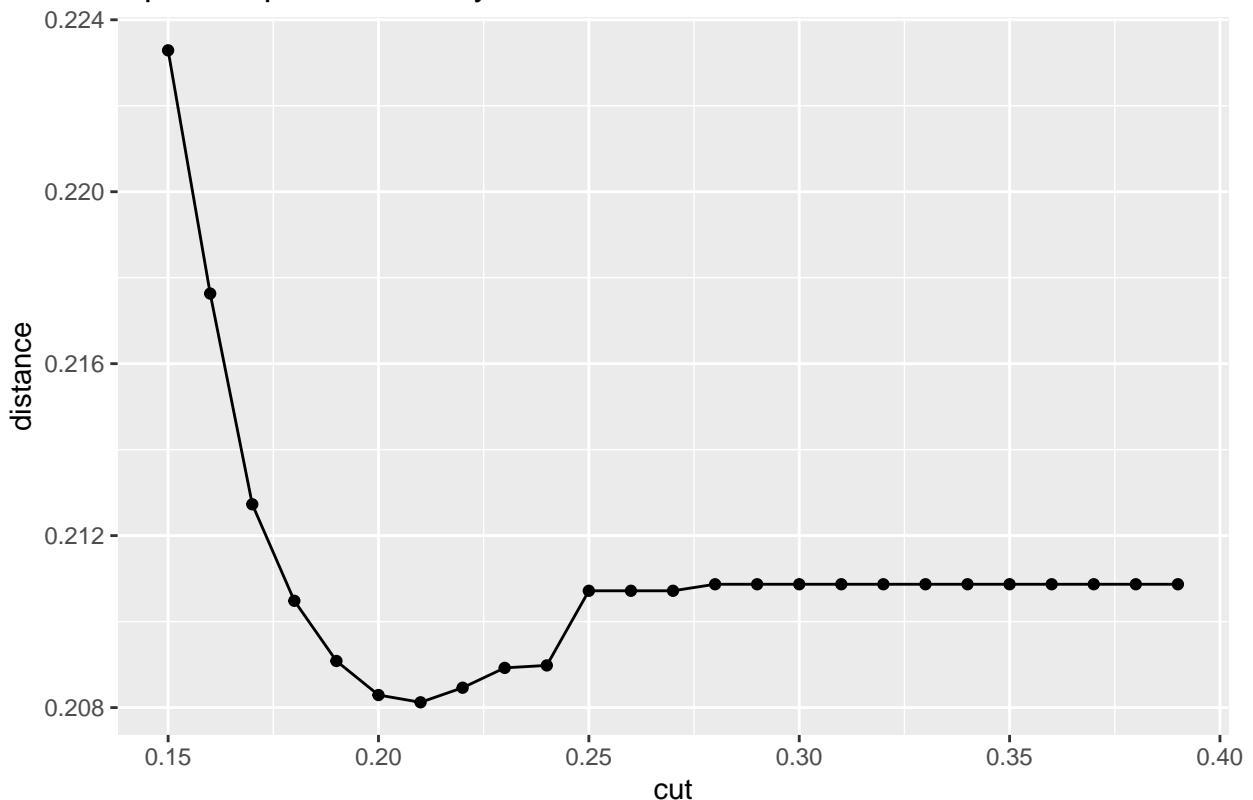
```

Optimal cp and cutoff by Truescore



```
ggplot(cutoff_results, aes(cut, distance)) +  
  geom_point() + geom_line() +  
  labs(title = "Optimal cp and cutoff by Distance")
```

Optimal cp and cutoff by Distance



```
max(cutoff_results$truescore)

## [1] 0.853699

(opt_cut_ts <- cutoff_results$cut[which.max(cutoff_results$truescore)])

## [1] 0.22

(opt_tpr_ts <- cutoff_results$tpr[which.max(cutoff_results$truescore)])

## [1] 0.8320783

(opt_tnr_ts <- cutoff_results$tnr[which.max(cutoff_results$truescore)])

## [1] 0.8764741

min(cutoff_results$distance)

## [1] 0.208123
```

```

(opt_cut_dist <- cutoff_results$cut[which.min(cutoff_results$distance)])  

## [1] 0.21  

(opt_tpr_dist <- cutoff_results$tpr[which.min(cutoff_results$distance)])  

## [1] 0.837751  

(opt_tnr_dist <- cutoff_results$tnr[which.min(cutoff_results$distance)])  

## [1] 0.8696528  

ctree_opt_cut_ts <- opt_cut_ts  

ctree_opt_cut_dist <- opt_cut_dist

```

Our Maximum Truescore criterion revealed an optimal cutoff of 0.22, while Minimum Distance indicated an optimal cutoff of 0.21. We applied both when using our pruned classification tree (*ctree_pruned_train*) to predict outcomes based on our test set predictors.

```

# Use our pruned classification tree (constructed using a cp of 0.0001338688)  

# to predict probabilities based on our test set predictors.  

test_set3 <- dplyr::select(test_set2, events, launch_angle, launch_speed,  

                           spray_angle_Kolp, spray_angle_adj, if_fielding_alignment,  

                           hp_to_1b)  

ctree_pruned_test_preds <- predict(ctree_pruned_train, newdata = test_set3[, -1],  

                                     method = "prob")  

ctree_pruned_test_preds <- as_tibble(ctree_pruned_test_preds)  

# Use our predicted probabilities for the test set to predict outcomes using  

# the optimal cutoff identified by our Maximum Truescore Method (0.22).  

ctree_pruned_test_preds <- ctree_pruned_test_preds %>%  

  mutate(pred22 = ifelse(ctree_pruned_test_preds$single > 0.22, "single", "field_out"))  

# Use our predicted probabilities for the test set to predict outcomes using  

# the optimal cutoff identified by our Minimum Distance Method (0.21).  

ctree_pruned_test_preds <- ctree_pruned_test_preds %>%  

  mutate(pred21 = ifelse(ctree_pruned_test_preds$single > 0.21, "single", "field_out"))  

# Convert all character columns to factors using dplyr package  

# (https://gist.github.com/ramhiser/character2factor.r). Make sure  

# the correct levels are ordered appropriately.  

ctree_pruned_test_preds <- ctree_pruned_test_preds %>%  

  mutate_if(sapply(ctree_pruned_test_preds, is.character), as.factor)  

sapply(ctree_pruned_test_preds, class)

```

```

##      single field_out    pred22    pred21
## "numeric" "numeric"   "factor"   "factor"

ctree_pruned_test_preds$pred22 <- factor(ctree_pruned_test_preds$pred22,
                                             levels = c("single", "field_out"))
ctree_pruned_test_preds$pred21 <- factor(ctree_pruned_test_preds$pred21,
                                             levels = c("single", "field_out"))

(levels(ctree_pruned_test_preds$pred22))

## [1] "single"     "field_out"

(levels(ctree_pruned_test_preds$pred21))

## [1] "single"     "field_out"

```

Were our decision tree's test set predictions better than those produced by our previous models? We were about to find out.

```

# Assess the performance of our final pruned classification tree.

ctree_pruned_test_preds <- add_column(ctree_pruned_test_preds, obs = test_set3$events,
                                         .before = "single")

(ctree_pruned_c22_tpr <- sensitivity(ctree_pruned_test_preds$pred22,
                                         ctree_pruned_test_preds$obs))

## [1] 0.808673

(ctree_pruned_c22_tnr <- specificity(ctree_pruned_test_preds$pred22,
                                         ctree_pruned_test_preds$obs))

## [1] 0.8619277

(ctree_pruned_c22_truescore <- round((2 * ctree_pruned_c22_tpr * ctree_pruned_c22_tnr) /
                                         (ctree_pruned_c22_tpr + ctree_pruned_c22_tnr), 6))

## [1] 0.834452

(ctree_pruned_c22_distance <-
  round(sqrt((1 - ctree_pruned_c22_tpr)^2 + (1 - ctree_pruned_c22_tnr)^2), 6))

## [1] 0.235945

(ctree_pruned_c21_tpr <- sensitivity(ctree_pruned_test_preds$pred21,
                                         ctree_pruned_test_preds$obs))

## [1] 0.813692

```

```

(ctree_pruned_c21_tnr <- specificity(ctree_pruned_test_preds$pred21,
                                         ctree_pruned_test_preds$obs))

## [1] 0.8552553

(ctree_pruned_c21_truescore <- round((2 * ctree_pruned_c21_tpr * ctree_pruned_c21_tnr) /
                                         (ctree_pruned_c21_tpr + ctree_pruned_c21_tnr), 6))

## [1] 0.833956

(ctree_pruned_c21_distance <-
  round(sqrt((1 - ctree_pruned_c21_tpr)^2 + (1 - ctree_pruned_c21_tnr)^2), 6))

## [1] 0.235927

assessment_results <-
  tibble(Model = c("Baseline", "Log Regression", "Log Regression",
                  "kNN (k23c31)", "Wtd kNN (k31w2c31)",
                  "Class. Tree", "Class. Tree"),
         `Cutoff Method` = c(NA, "Truescore", "Distance", "both",
                            "both", "Truescore", "Distance"),
         `Truescore` = c(bl_truescore, max_truescore,
                         min_distance_truescore,
                         knn_k23c31_truescore,
                         kknn_k31w2c31_truescore,
                         ctree_pruned_c22_truescore,
                         ctree_pruned_c21_truescore),
         TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                 knn_k7c40_tpr, knn_k23c31_tpr,
                 ctree_pruned_c22_tpr, ctree_pruned_c21_tpr),
         TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                 knn_k23c31_tnr, kknn_k31w2c31_tnr,
                 ctree_pruned_c22_tnr, ctree_pruned_c21_tnr),
         Distance = c(NA, max_truescore_distance,
                     min_distance, knn_k23c31_distance,
                     kknn_k31w2c31_distance,
                     ctree_pruned_c22_distance,
                     ctree_pruned_c21_distance),
         `Best Cutoff` = c(NA, max_truescore_cutoff,
                          min_distance_cutoff,
                          knn_opt_cut, kknn_opt_cut,
                          ctree_opt_cut_ts,
                          ctree_opt_cut_dist))
knitr::kable(assessment_results[1:7, ], caption = "Assessment Results")

```

Table 51: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	Best Cutoff
Baseline	NA	0.385984	0.259787	0.7506110	NA	NA
Log Regression	Truescore	0.408139	0.383256	0.4362160	0.835599	0.749984

Model	Cutoff Method	Truescore	TPR	TNR	Distance	Best Cutoff
Log Regression	Distance	0.399327	0.328448	0.5092160	0.831776	0.764628
kNN (k23c31)	both	0.820183	0.772937	0.8154850	0.254349	0.31
Wtd kNN (k31w2c31)	both	0.824830	0.824935	0.8057740	0.248611	0.31
Class. Tree	Truescore	0.834452	0.808673	0.8619277	0.235945	0.22
Class. Tree	Distance	0.833956	0.813692	0.8552553	0.235927	0.21

Yes! Both of our classification trees slightly outperformed our Weighted kNN Model. The tree utilizing the Maximum Truescore Cutoff Method produced a truescore of 0.834452 (as compared to the previous best of 0.824830 by our Weighted kNN Model). The tree utilizing the Minimum Distance Cutoff Method produced a Distance of 0.235927 (as compared to the previous best of 0.248611 by Weighted kNN).

One nice feature of rpart is that it explicitly computes the importance of each predictor variable, and then scales the values to sum to 100. This scaled version of variable importance can be obtained by using rpart's summary function; unfortunately, the output is so lengthy that it's best to save it to a file in order to review it. The file argument in the summary function allows you to specify that the output should be saved to file.

```
# Review the Variable Importance computations from the summary() function,
# scaled to sum to 100.

summary(ctree_pruned_train, file = "ctree_pruned_train_summary.csv")

ctree_pruned_train_var_imp <-
  tibble(Predictor = c("launch_angle", "launch_speed",
                      "spray_angle_Kolp", "spray_angle_adj",
                      "if_fielding_alignment", "hp_to_1b"),
         `ctree Importance` = c(48, 32, 4, 13, 2, "< 1"))
knitr::kable(ctree_pruned_train_var_imp[1:6, ],
             caption = "Variable Importance: Pruned Classification Tree")
```

Table 52: Variable Importance: Pruned Classification Tree

Predictor	ctree Importance
launch_angle	48
launch_speed	32
spray_angle_Kolp	4
spray_angle_adj	13
if_fielding_alignment	2
hp_to_1b	< 1

Our data exploration suggested the same order of importance, but it was a bit surprising to see *launch_angle* and *launch_speed* accounting for so much as 80% of the model's predictive ability. Still, the Spray Angle predictors combined to account for nearly all of the remaining 20%, not an insignificant portion. We had to do a great deal of data transformation to get the Spray Angle concept quantified in such a way that it could be incorporated in a model. We see the full integration of Spray Angle in the Baseball Savant arena as a worthwhile area for improvement.

Build and Assess a Random Forest Model

A random forest algorithm creates many classification trees. Each tree is constructed using a random sample (with replacement) of N observations from the original data; this sample serves as the training set. In addition, a random selection of m predictor variables is used to build each tree. N is typically the number of observations in the original training set, and m is a constant that's less than the total number of predictor variables. At the end of the process, each tree essentially votes for a particular class. The forest's prediction is the class receiving the most votes.

Given its design, a random forest algorithm ought to outperform a single classification tree. To see whether this was true, we first trained a random forest model.

The *classwt* parameter in the *randomForest* implementation unfortunately does not achieve its intended purpose of overcoming majority class bias, our perpetual challenge throughout this project. So we turned to the function's *cutoff* parameter. Yes, an honest to goodness cutoff parameter built right into the model training function! Thank you, thank you, thank you. Somebody else had apparently recognized the power and legitimacy of altering decision thresholds when classes are imbalanced.

```
# Use the randomForest function (randomForest package) to fit a
# random forest model to our training data. Use the cutoff parameter to
# test decision cutoffs ranging from 0.24 to 0.33. Determine the optimal
# cutoff by using both our Maximum Truescore Method and our Minimum
# Distance Method.

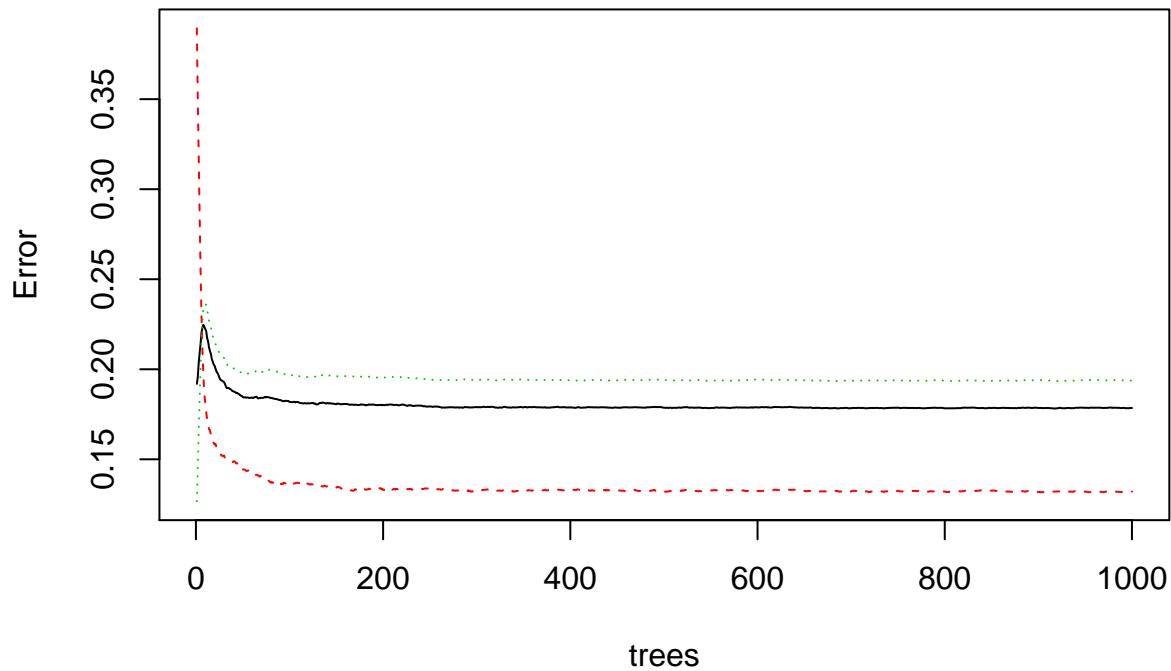
library(randomForest)

set.seed(1)
(rforest_train_c24 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.24, "field_out" = 0.76),
                                      importance = TRUE))

## 
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single =
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
##
##   OOB estimate of  error rate: 17.85%
##   Confusion matrix:
##   single field_out class.error
##   single    17289     2631  0.1320783
##   field_out  11731    48815  0.1937535

plot(rforest_train_c24)
```

rforest_train_c24

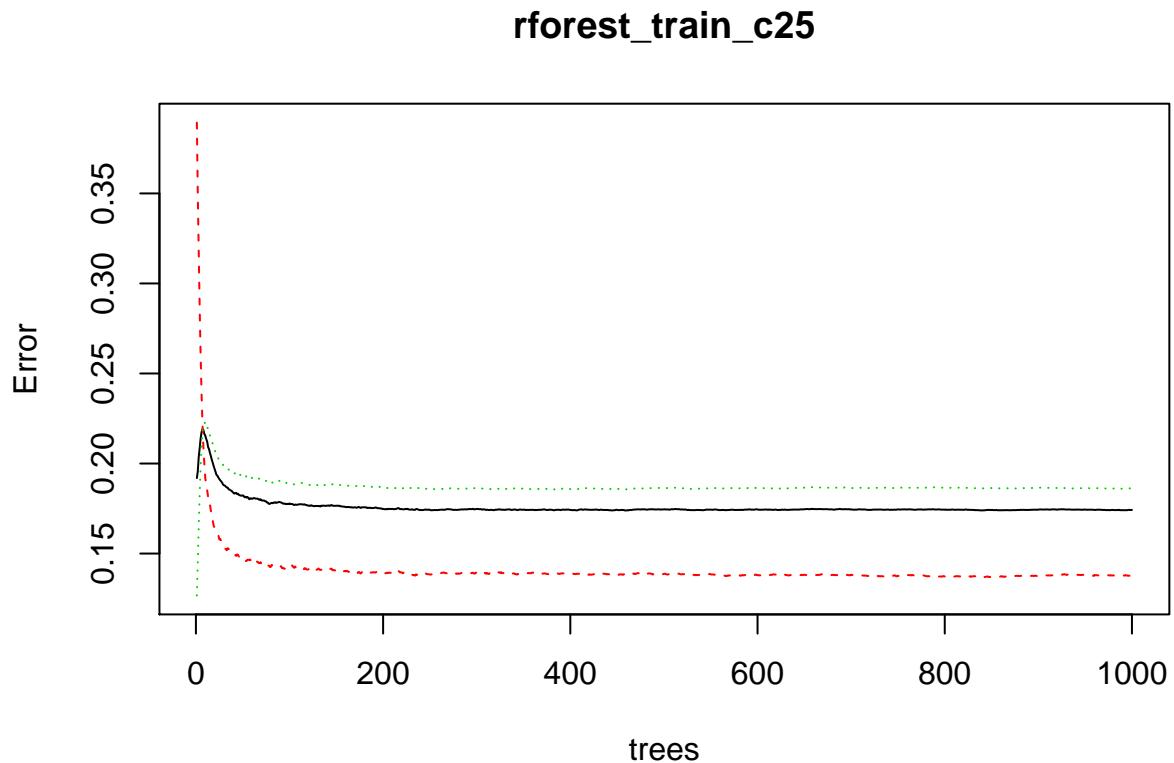


```
cm_c24 <- confusionMatrix(rforest_train_c24$predicted, train_set3$events)
tpr_c24 <- cm_c24$byClass[[1]]
tnr_c24 <- cm_c24$byClass[[2]]
rf_c24_truescore <- round((2 * tpr_c24 * tnr_c24) /
                           (tpr_c24 + tnr_c24), 6)
rf_c24_distance <- round(sqrt((1 - tpr_c24)^2 + (1 - tnr_c24)^2), 6)

set.seed(1)
(rforest_train_c25 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.25, "field_out" = 0.75),
                                      importance = TRUE))

##
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single
##   Type of random forest: classification
##           Number of trees: 1000
##   No. of variables tried at each split: 2
##
##           OOB estimate of  error rate: 17.42%
##   Confusion matrix:
##       single field_out class.error
##   single     17173     2747  0.1379016
##   field_out  11273     49273  0.1861890
```

```
plot(rforest_train_c25)
```



```
cm_c25 <- confusionMatrix(rforest_train_c25$predicted, train_set3$events)
tpr_c25 <- cm_c25$byClass[[1]]
tnr_c25 <- cm_c25$byClass[[2]]
rf_c25_truescore <- round((2 * tpr_c25 * tnr_c25) /
                           (tpr_c25 + tnr_c25), 6)
rf_c25_distance <- round(sqrt((1 - tpr_c25)^2 + (1 - tnr_c25)^2), 6)

set.seed(1)
(rforest_train_c26 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.26, "field_out" = 0.74),
                                      importance = TRUE))

## 
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
## 
##   OOB estimate of  error rate: 16.97%
##   Confusion matrix:
```

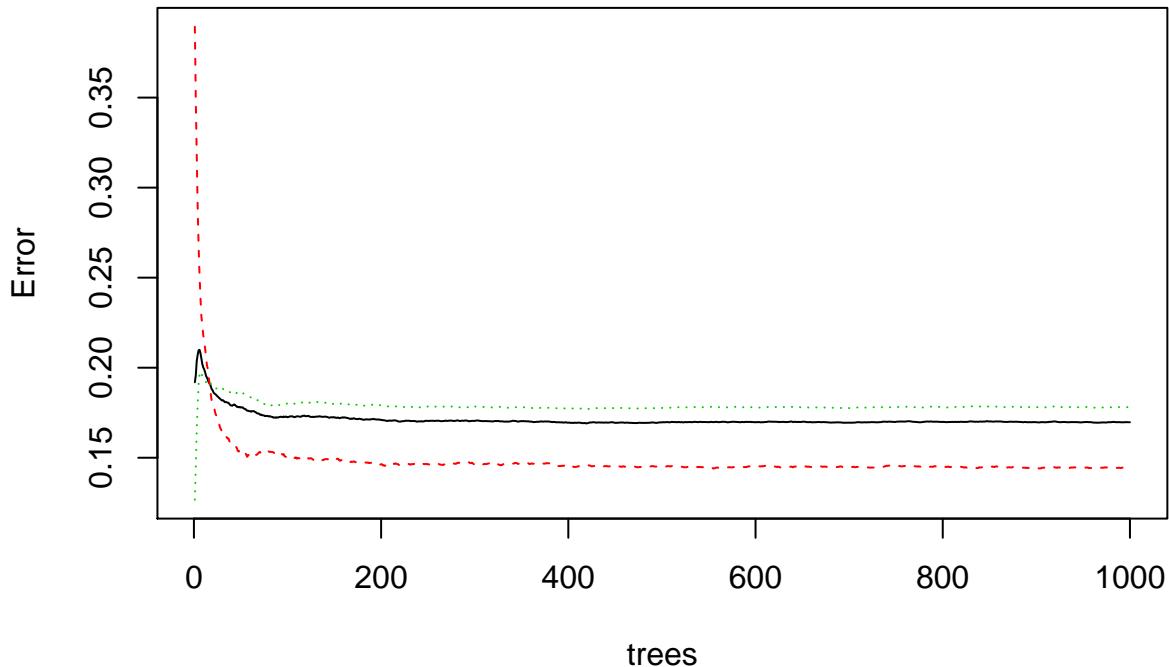
```

##           single field_out class.error
## single      17040      2880  0.1445783
## field_out   10775     49771  0.1779639

```

```
plot(rforest_train_c26)
```

rforest_train_c26



```
cm_c26 <- confusionMatrix(rforest_train_c26$predicted, train_set3$events)
```

```
tpr_c26 <- cm_c26$byClass[[1]]
```

```
tnr_c26 <- cm_c26$byClass[[2]]
```

```
rf_c26_truescore <- round((2 * tpr_c26 * tnr_c26) /
  (tpr_c26 + tnr_c26), 6)
```

```
rf_c26_distance <- round(sqrt((1 - tpr_c26)^2 + (1 - tnr_c26)^2), 6)
```

```
set.seed(1)
```

```
(rforest_train_c27 <- randomForest(events ~ ., data = train_set3,
  ntree = 1000, mtry = 2,
  cutoff = c("single" = 0.27, "field_out" = 0.73),
  importance = TRUE))
```

```
##
```

```
## Call:
```

```
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,
```

```
##               Type of random forest: classification
```

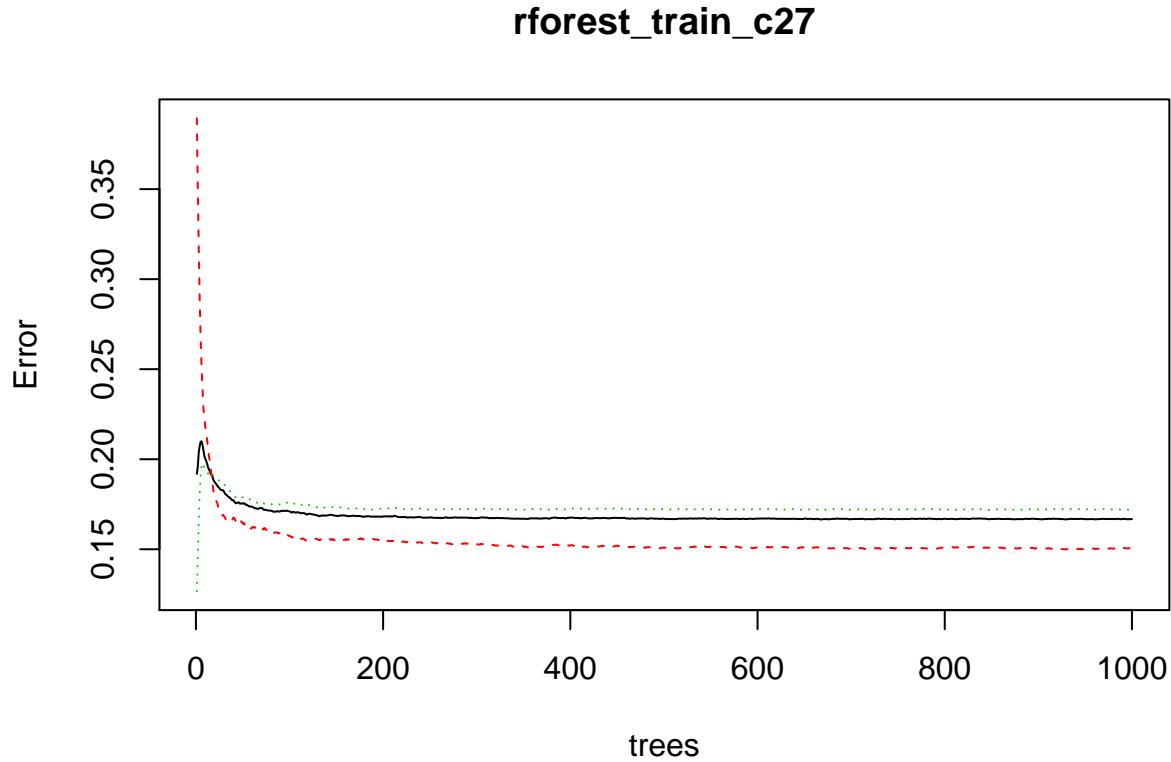
```
##               Number of trees: 1000
```

```

## No. of variables tried at each split: 2
##
##          OOB estimate of  error rate: 16.68%
## Confusion matrix:
##              single field_out class.error
## single      16916      3004  0.1508032
## field_out   10415      50131  0.1720180

plot(rforest_train_c27)

```



```

cm_c27 <- confusionMatrix(rforest_train_c27$predicted, train_set3$events)
tpr_c27 <- cm_c27$byClass[[1]]
tnr_c27 <- cm_c27$byClass[[2]]
rf_c27_truescore <- round((2 * tpr_c27 * tnr_c27) /
                           (tpr_c27 + tnr_c27), 6)
rf_c27_distance <- round(sqrt((1 - tpr_c27)^2 + (1 - tnr_c27)^2), 6)

set.seed(1)
(rforest_train_c28 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.28, "field_out" = 0.72),
                                      importance = TRUE))

##

```

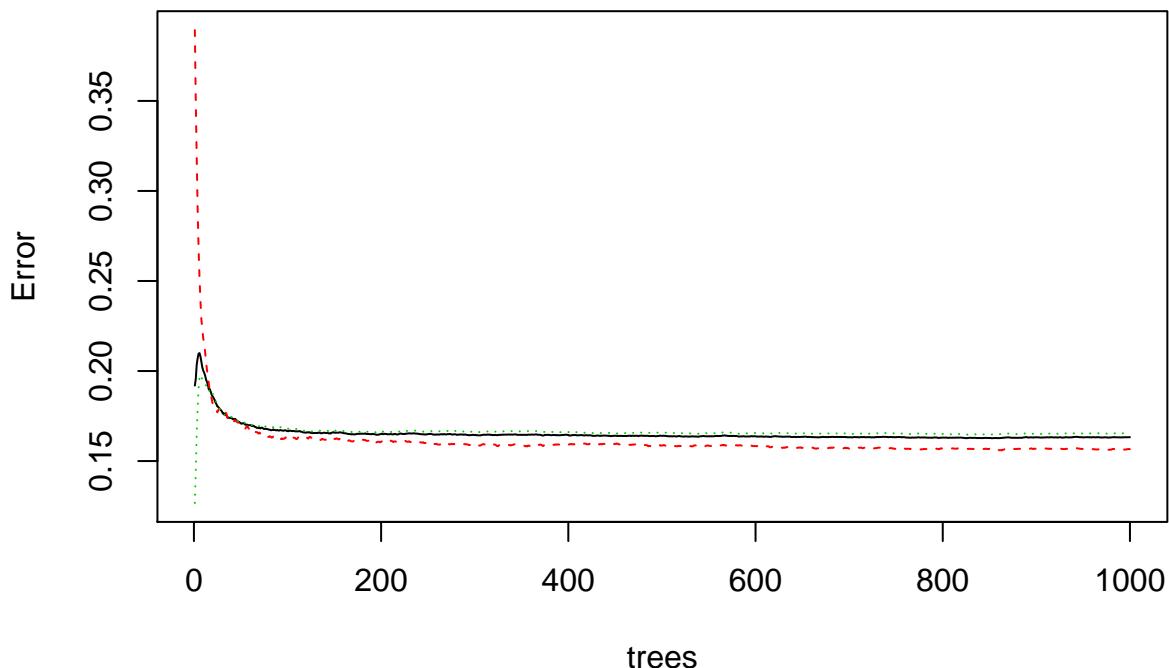
```

## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,           mtry = 2, cutoff = c(single
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
##
##   OOB estimate of error rate: 16.32%
## Confusion matrix:
##   single field_out class.error
## single     16802     3118  0.1565261
## field_out  10015     50531  0.1654114

plot(rforest_train_c28)

```

rforest_train_c28



```

cm_c28 <- confusionMatrix(rforest_train_c28$predicted, train_set3$events)
tpr_c28 <- cm_c28$byClass[[1]]
tnr_c28 <- cm_c28$byClass[[2]]
rf_c28_truescore <- round((2 * tpr_c28 * tnr_c28) /
                           (tpr_c28 + tnr_c28), 6)
rf_c28_distance <- round(sqrt((1 - tpr_c28)^2 + (1 - tnr_c28)^2), 6)

set.seed(1)
(rforest_train_c29 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,

```

```

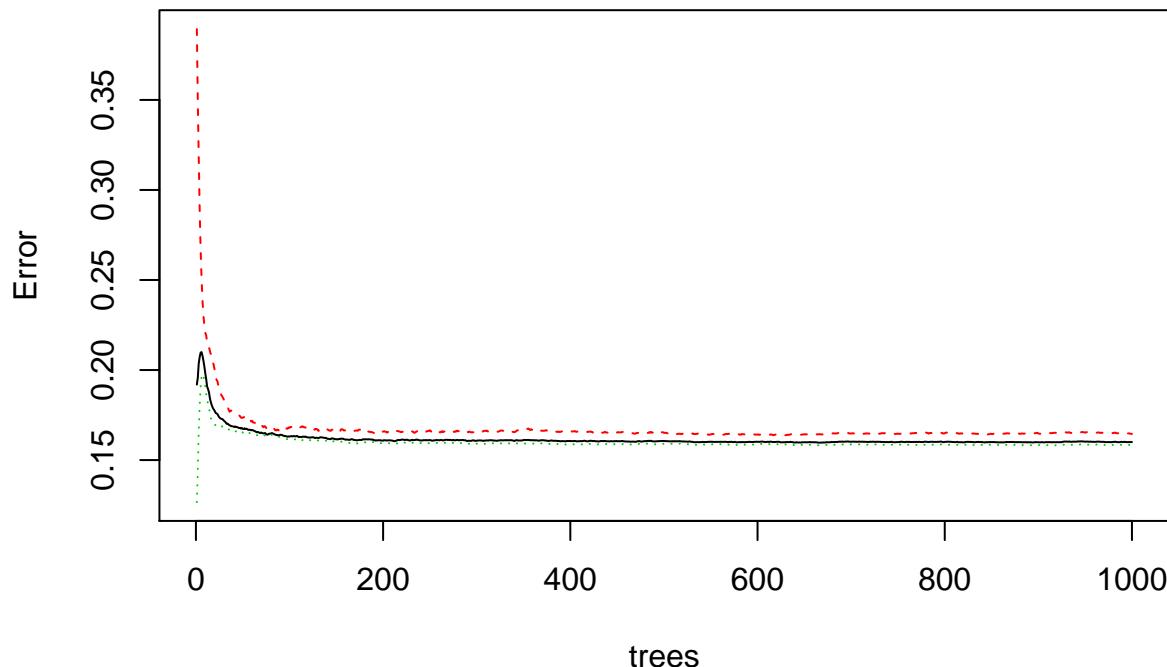
cutoff = c("single" = 0.29, "field_out" = 0.71),
importance = TRUE))

## 
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single =
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
##
##   OOB estimate of  error rate: 16%
## Confusion matrix:
##             single field_out class.error
## single     16642      3278  0.1645582
## field_out    9599      50947  0.1585406

plot(rforest_train_c29)

```

rforest_train_c29



```

cm_c29 <- confusionMatrix(rforest_train_c29$predicted, train_set3$events)
tpr_c29 <- cm_c29$byClass[[1]]
tnr_c29 <- cm_c29$byClass[[2]]
rf_c29_truescore <- round((2 * tpr_c29 * tnr_c29) /
                           (tpr_c29 + tnr_c29), 6)
rf_c29_distance <- round(sqrt((1 - tpr_c29)^2 + (1 - tnr_c29)^2), 6)

```

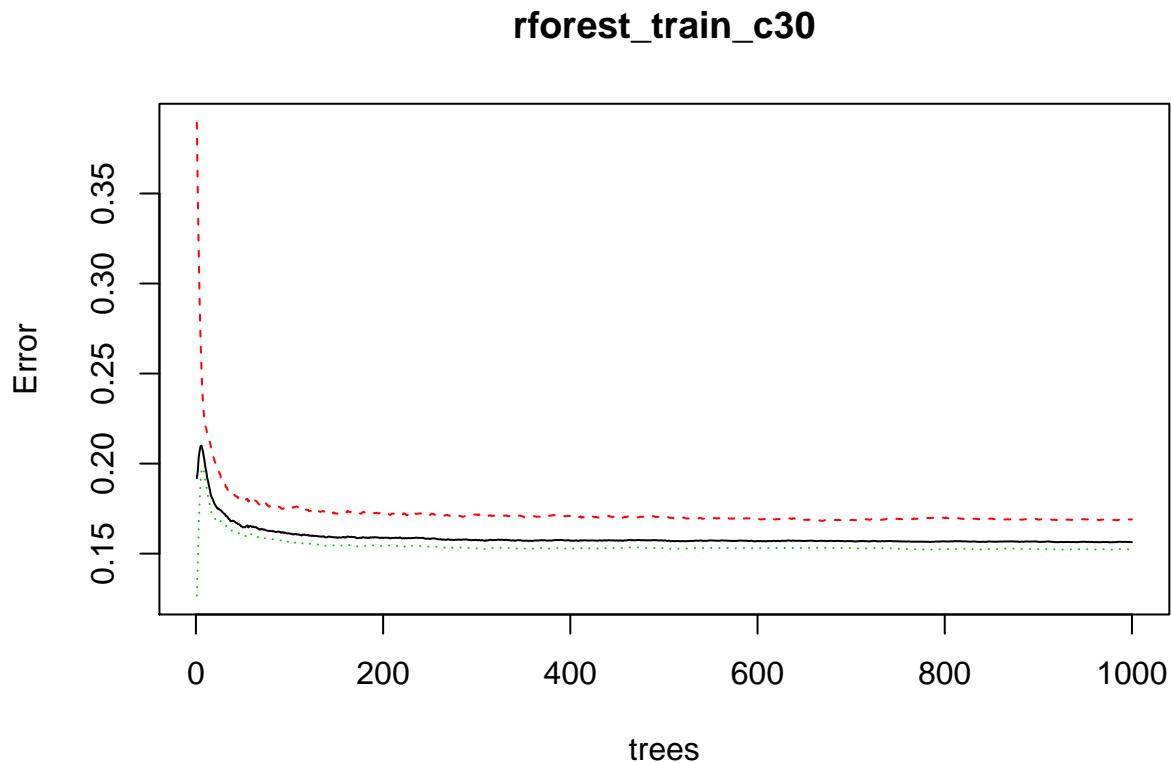
```

set.seed(1)
(rforest_train_c30 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.30, "field_out" = 0.70),
                                      importance = TRUE))

##
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single =
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
##
##   OOB estimate of  error rate: 15.63%
##   Confusion matrix:
##     single field_out class.error
##   single     16556     3364  0.1688755
##   field_out    9213    51333  0.1521653

plot(rforest_train_c30)

```



```

cm_c30 <- confusionMatrix(rforest_train_c30$predicted, train_set3$events)
tpr_c30 <- cm_c30$byClass[[1]]
tnr_c30 <- cm_c30$byClass[[2]]
rf_c30_truescore <- round((2 * tpr_c30 * tnr_c30) /
                           (tpr_c30 + tnr_c30), 6)
rf_c30_distance <- round(sqrt((1 - tpr_c30)^2 + (1 - tnr_c30)^2), 6)

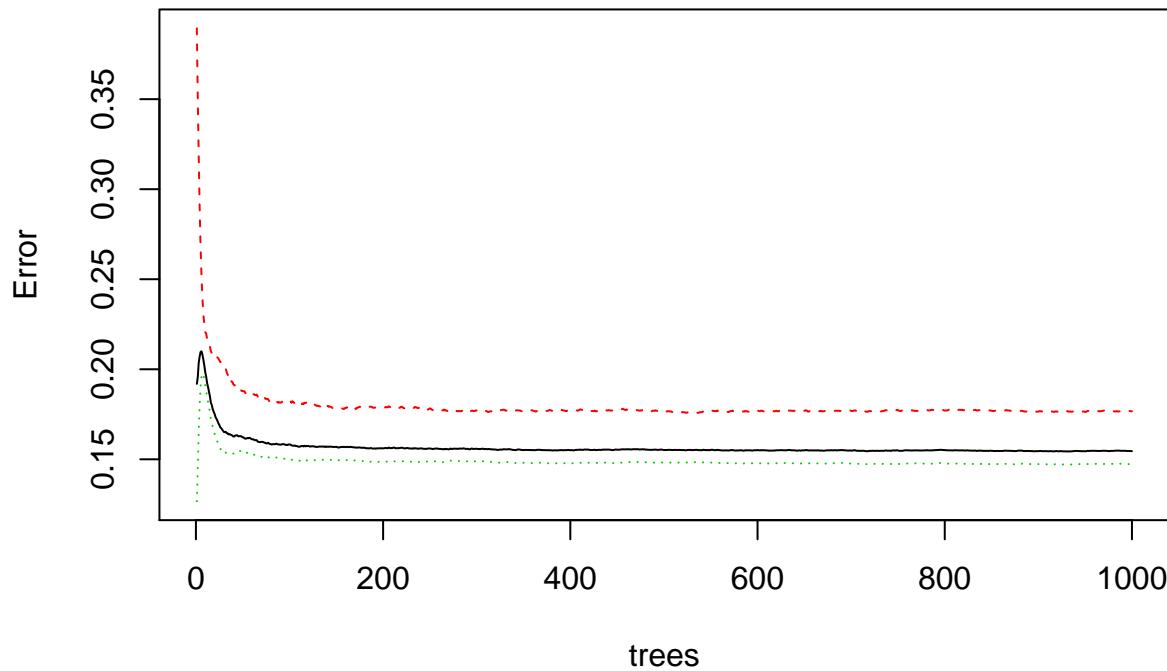
set.seed(1)
(rforest_train_c31 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.31, "field_out" = 0.69),
                                      importance = TRUE))

##
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single
##   Type of random forest: classification
##           Number of trees: 1000
## No. of variables tried at each split: 2
##
##           OOB estimate of  error rate: 15.45%
## Confusion matrix:
##             single field_out class.error
## single     16401     3519    0.1766566
## field_out    8916     51630    0.1472599

plot(rforest_train_c31)

```

rforest_train_c31

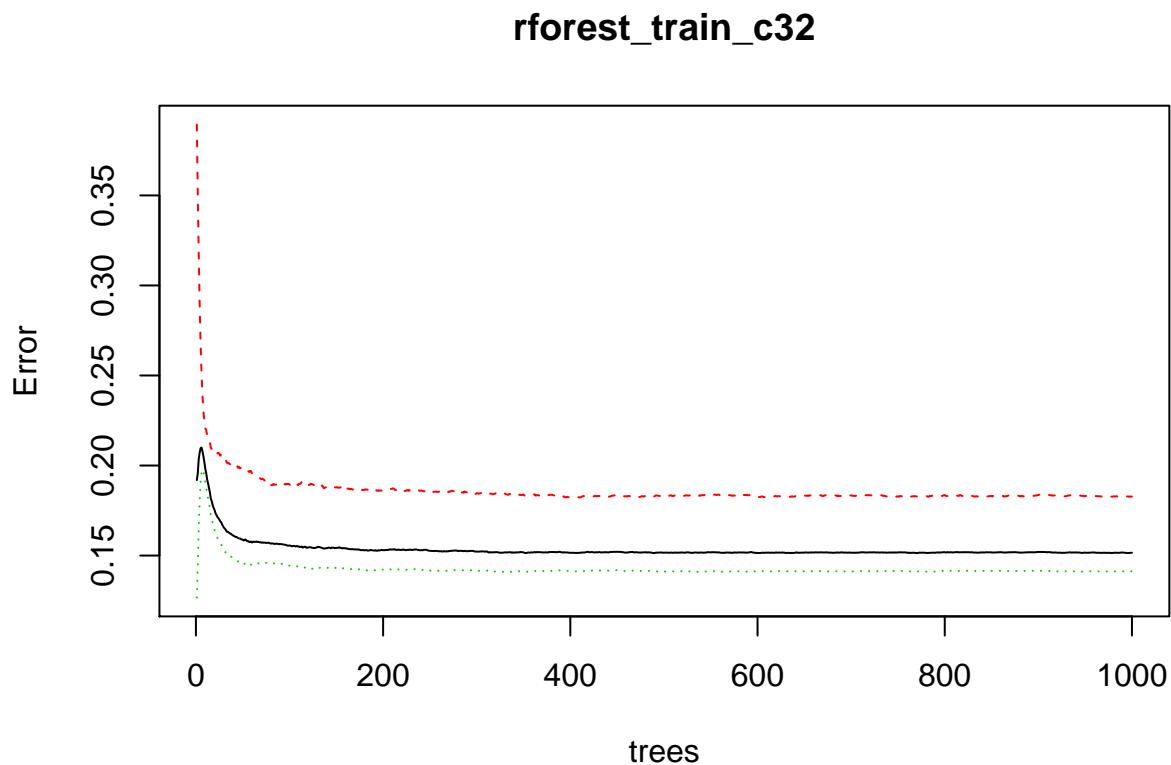


```
cm_c31 <- confusionMatrix(rforest_train_c31$predicted, train_set3$events)
tpr_c31 <- cm_c31$byClass[[1]]
tnr_c31 <- cm_c31$byClass[[2]]
rf_c31_truescore <- round((2 * tpr_c31 * tnr_c31) /
                           (tpr_c31 + tnr_c31), 6)
rf_c31_distance <- round(sqrt((1 - tpr_c31)^2 + (1 - tnr_c31)^2), 6)

set.seed(1)
(rforest_train_c32 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.32, "field_out" = 0.68),
                                      importance = TRUE))

##
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single
##   Type of random forest: classification
##           Number of trees: 1000
##   No. of variables tried at each split: 2
##
##           OOB estimate of  error rate: 15.16%
##   Confusion matrix:
##     single field_out class.error
##   single    16279     3641  0.1827811
##   field_out    8557     51989  0.1413306
```

```
plot(rforest_train_c32)
```



```
cm_c32 <- confusionMatrix(rforest_train_c32$predicted, train_set3$events)
tpr_c32 <- cm_c32$byClass[[1]]
tnr_c32 <- cm_c32$byClass[[2]]
rf_c32_truescore <- round((2 * tpr_c32 * tnr_c32) /
                           (tpr_c32 + tnr_c32), 6)
rf_c32_distance <- round(sqrt((1 - tpr_c32)^2 + (1 - tnr_c32)^2), 6)

set.seed(1)
(rforest_train_c33 <- randomForest(events ~ ., data = train_set3,
                                      ntree = 1000, mtry = 2,
                                      cutoff = c("single" = 0.33, "field_out" = 0.67),
                                      importance = TRUE))

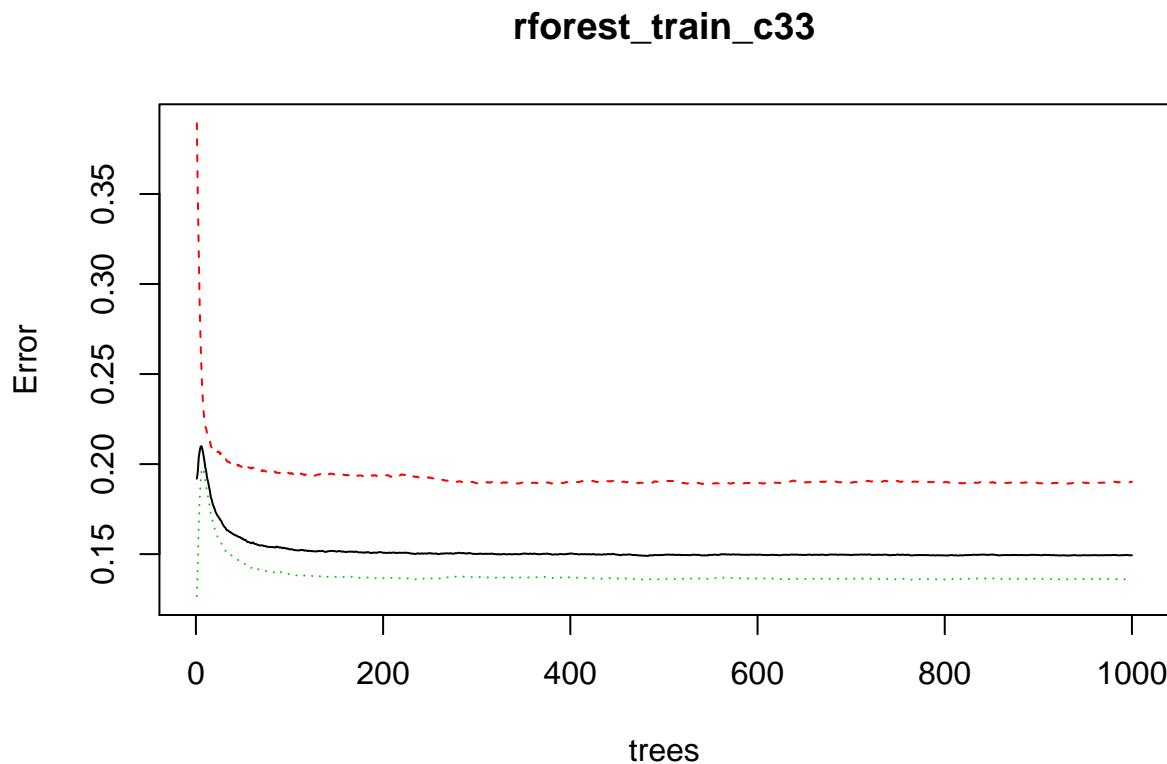
## 
## Call:
##   randomForest(formula = events ~ ., data = train_set3, ntree = 1000,      mtry = 2, cutoff = c(single =
## 
##   Type of random forest: classification
##   Number of trees: 1000
##   No. of variables tried at each split: 2
## 
##   OOB estimate of  error rate: 14.93%
##   Confusion matrix:
```

```

##           single field_out class.error
## single      16131      3789   0.1902108
## field_out    8228      52318   0.1358967

plot(rforest_train_c33)

```



```

cm_c33 <- confusionMatrix(rforest_train_c33$predicted, train_set3$events)
tpr_c33 <- cm_c33$byClass[[1]]
tnr_c33 <- cm_c33$byClass[[2]]
rf_c33_truescore <- round((2 * tpr_c33 * tnr_c33) /
                           (tpr_c33 + tnr_c33), 6)
rf_c33_distance <- round(sqrt((1 - tpr_c33)^2 + (1 - tnr_c33)^2), 6)

# Compile the cutoff results in a table.

rf_cutoff_results <- tibble(
  cut = seq(0.24, 0.33, 0.01),
  tpr = c(tpr_c24, tpr_c25, tpr_c26, tpr_c27, tpr_c28,
          tpr_c29, tpr_c30, tpr_c31, tpr_c32, tpr_c33),
  tnr = c(tnr_c24, tnr_c25, tnr_c26, tnr_c27, tnr_c28,
          tnr_c29, tnr_c30, tnr_c31, tnr_c32, tnr_c33),
  truescore = c(rf_c24_truescore, rf_c25_truescore,
               rf_c26_truescore, rf_c27_truescore,
               rf_c28_truescore, rf_c29_truescore,
               rf_c30_truescore, rf_c31_truescore,
               rf_c32_truescore, rf_c33_truescore)

```

```

            rf_c32_truescore, rf_c33_truescore),
distance = c(rf_c24_distance, rf_c25_distance,
            rf_c26_distance, rf_c27_distance,
            rf_c28_distance, rf_c29_distance,
            rf_c30_distance, rf_c31_distance,
            rf_c32_distance, rf_c33_distance))

knitr::kable(rf_cutoff_results[1:10, ], caption = "Random Forest Cutoff Results")

```

Table 53: Random Forest Cutoff Results

cut	tpr	tnr	truescore	distance
0.24	0.8679217	0.8062465	0.835948	0.234489
0.25	0.8620984	0.8138110	0.837259	0.231696
0.26	0.8554217	0.8220361	0.838397	0.229290
0.27	0.8491968	0.8279820	0.838455	0.228761
0.28	0.8434739	0.8345886	0.839008	0.227731
0.29	0.8354418	0.8414594	0.838440	0.228505
0.30	0.8311245	0.8478347	0.839396	0.227317
0.31	0.8233434	0.8527401	0.837784	0.229985
0.32	0.8172189	0.8586694	0.837432	0.231048
0.33	0.8097892	0.8641033	0.836065	0.233769

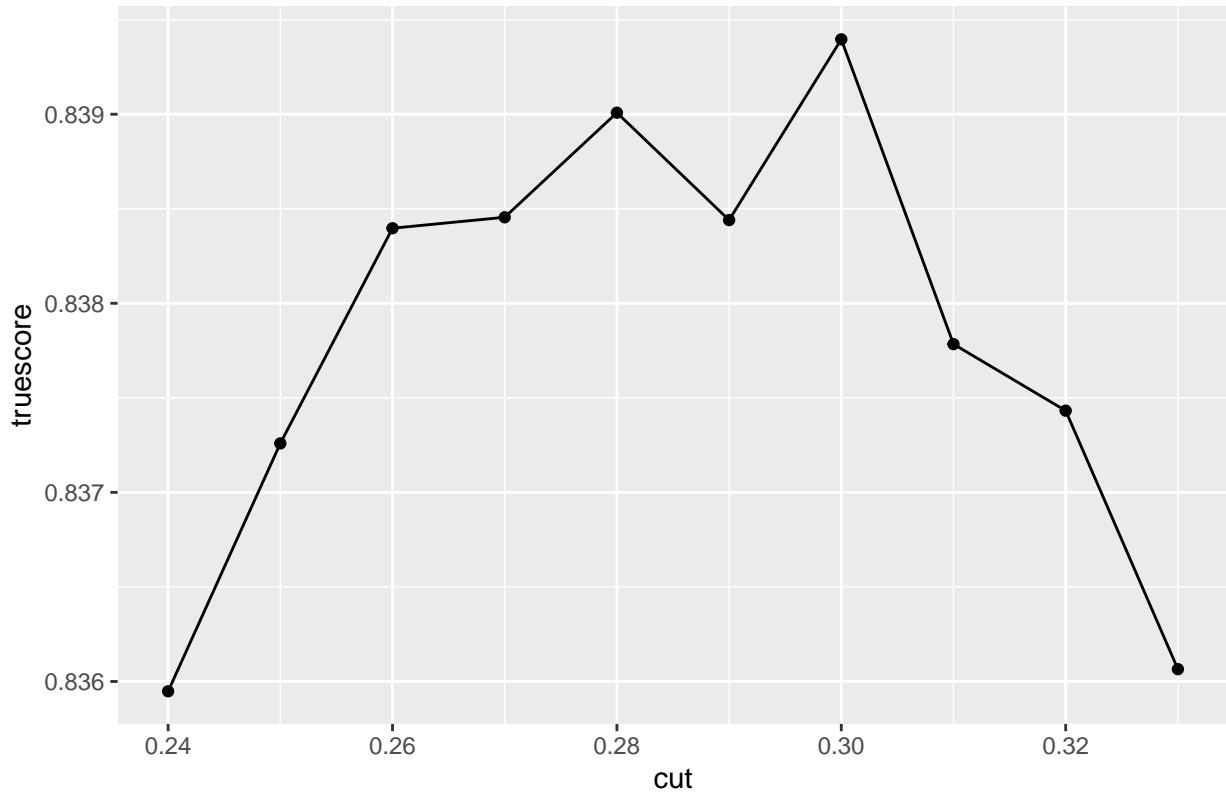
```

# Identify the optimal cutoff based on each of our assessment methods,
# Truescore and Minimum Distance to (0, 1).

ggplot(rf_cutoff_results, aes(cut, truescore)) +
  geom_point() + geom_line() +
  labs(title = "RF Optimal Cutoff by Truescore")

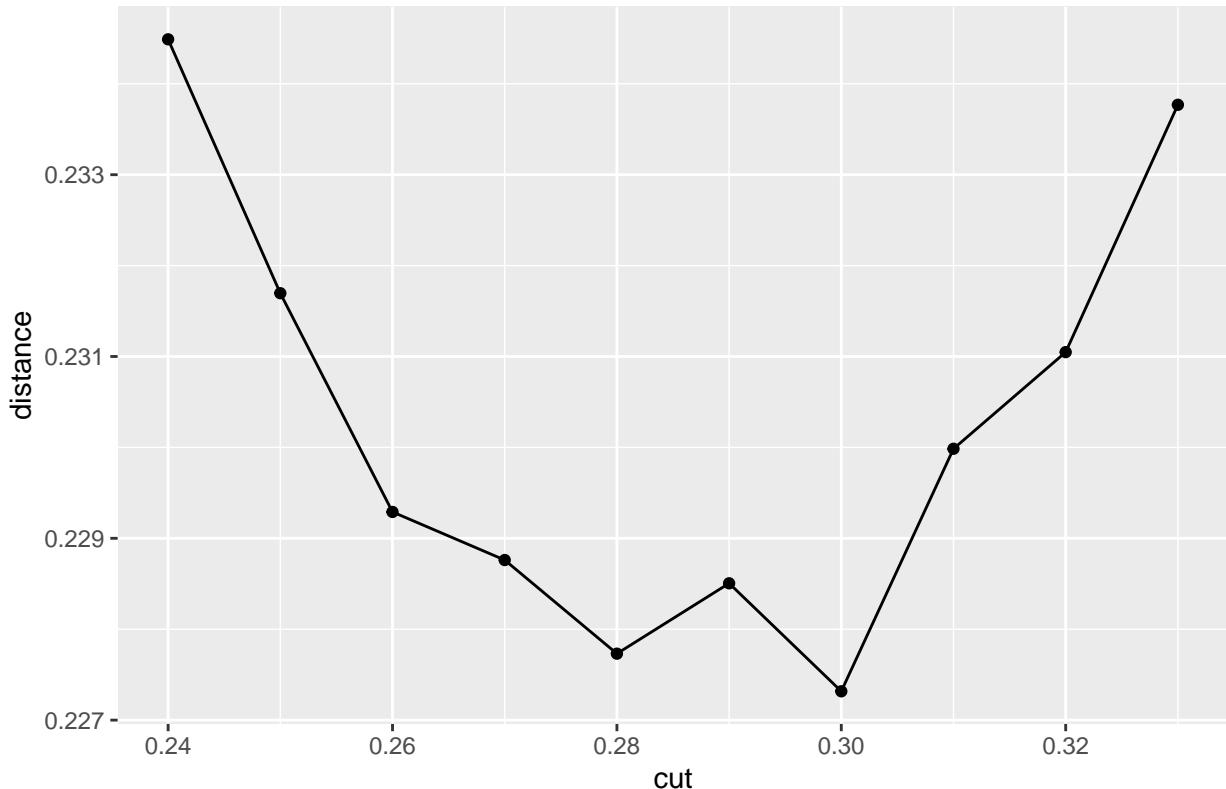
```

RF Optimal Cutoff by Truescore



```
ggplot(rf_cutoff_results, aes(cut, distance)) +  
  geom_point() + geom_line() +  
  labs(title = "RF Optimal Cutoff by Distance")
```

RF Optimal Cutoff by Distance



```
(rf_opt_cut_ts <- rf_cutoff_results$cut[which.max(rf_cutoff_results$truescore)])
```

```
## [1] 0.3
```

```
(rf_opt_cut_dist <- rf_cutoff_results$cut[which.min(rf_cutoff_results$distance)])
```

```
## [1] 0.3
```

It always makes our job a bit easier when a single optimal cutoff generates predictions that result in both the highest truescore and the shortest distance to $(0, 1)$. Our trained random forest model, `rforest_train_c30`, based on a thousand trees, an mtry value of two, and a cutoff of 0.30, was now ready to make some predictions against our test set.

```
# Use our optimized randomForest algorithm (rforest_train_c30) to predict
# "single" or "field_out" based on our test data.
```

```
rforest_predict <- predict(rforest_train_c30, newdata = test_set3, type = "response")
rforest_predict_prob <- predict(rforest_train_c30, newdata = test_set3, type = "prob")
```

```
# Assess the predictive ability of the randomForest algorithm.
```

```
confusionMatrix(rforest_predict, test_set3$events)
```

```
## Confusion Matrix and Statistics
```

```

##          Reference
## Prediction single field_out
##      single     4178     2284
##    field_out     803     12853
##
##                  Accuracy : 0.8466
##                  95% CI : (0.8415, 0.8515)
##      No Information Rate : 0.7524
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.6255
##
## McNemar's Test P-Value : < 2.2e-16
##
##      Sensitivity : 0.8388
##      Specificity : 0.8491
##      Pos Pred Value : 0.6465
##      Neg Pred Value : 0.9412
##      Prevalence : 0.2476
##      Detection Rate : 0.2077
##      Detection Prevalence : 0.3212
##      Balanced Accuracy : 0.8439
##
##      'Positive' Class : single
##


(rforest_tpr <- round(sensitivity(rforest_predict,
                                    reference = factor(test_set3$events)), 6))

## [1] 0.838787

(rforest_tnr <- round(specificity(rforest_predict,
                                    reference = factor(test_set3$events)), 6))

## [1] 0.849111

(rforest_truescore <-
  round((2 * rforest_tpr * rforest_tnr) / (rforest_tpr + rforest_tnr) , 6))

## [1] 0.843917

(rforest_distance <-
  round(sqrt((1 - rforest_tpr)^2 + (1 - rforest_tnr)^2), 6))

## [1] 0.22081

assessment_results <- tibble(Model = c("Baseline", "Log Regression", "Log Regression",
                                         "kNN (k23c31)", "Wtd kNN (k31w2c31)",
                                         "Classification Tree", "Classification Tree",
                                         "Classification Tree"))

```

```

    "Random Forest"),
`Cutoff Method` = c(NA, "Truescore", "Distance", "both",
                     "both", "Truescore", "Distance",
                     "both"),
`Truescore` = c(bl_truescore, max_truescore,
                min_distance_truescore,
                knn_k23c31_truescore,
                kknn_k31w2c31_truescore,
                ctree_pruned_c22_truescore,
                ctree_pruned_c21_truescore,
                rforest_truescore),
TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
        knn_k7c40_tpr, knn_k23c31_tpr,
        ctree_pruned_c22_tpr, ctree_pruned_c21_tpr,
        rforest_tpr),
TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
        knn_k23c31_tnr, kknn_k31w2c31_tnr,
        ctree_pruned_c22_tnr, ctree_pruned_c21_tnr,
        rforest_tnr),
Distance = c(NA, max_truescore_distance,
            min_distance, knn_k23c31_distance,
            kknn_k31w2c31_distance,
            ctree_pruned_c22_distance,
            ctree_pruned_c21_distance,
            rforest_distance),
`Best Cutoff` = c(NA, max_truescore_cutoff,
                  min_distance_cutoff, knn_opt_cut,
                  kknn_opt_cut, ctree_opt_cut_ts,
                  ctree_opt_cut_dist, rf_opt_cut_ts))

knitr::kable(assessment_results[1:8, ], caption = "Assessment Results")

```

Table 54: Assessment Results

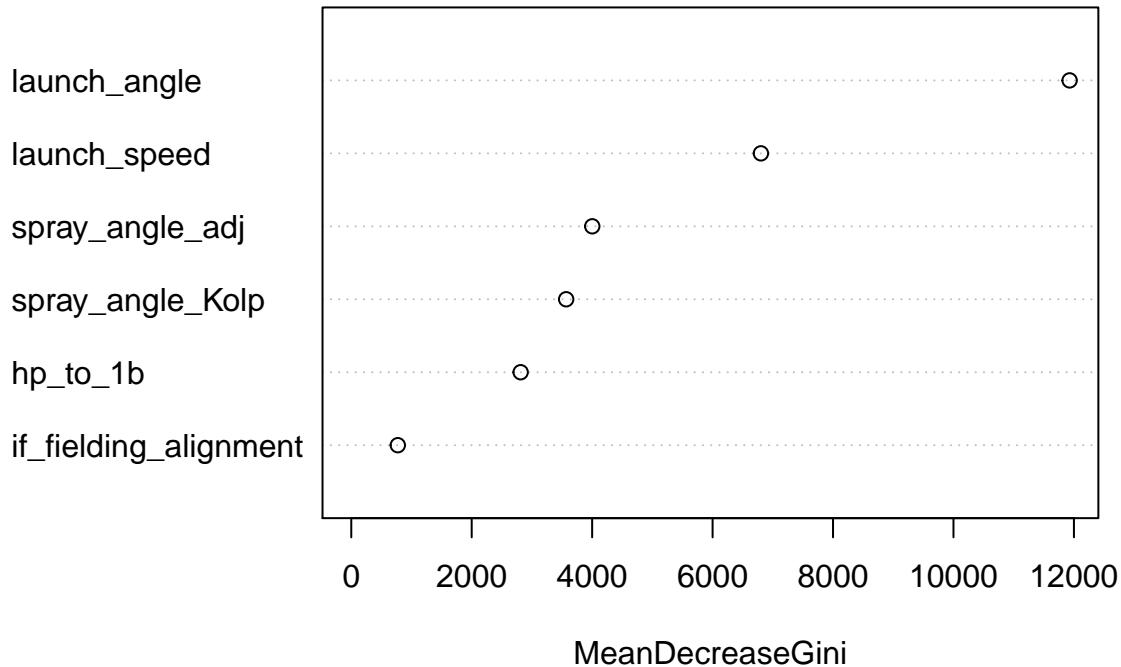
Model	Cutoff Method	Truescore	TPR	TNR	Distance	Best Cutoff
Baseline	NA	0.385984	0.259787	0.7506110	NA	NA
Log Regression	Truescore	0.408139	0.383256	0.4362160	0.835599	0.749984
Log Regression	Distance	0.399327	0.328448	0.5092160	0.831776	0.764628
kNN (k23c31)	both	0.820183	0.772937	0.8154850	0.254349	0.31
Wtd kNN (k31w2c31)	both	0.824830	0.824935	0.8057740	0.248611	0.31
Classification Tree	Truescore	0.834452	0.808673	0.8619277	0.235945	0.22
Classification Tree	Distance	0.833956	0.813692	0.8552553	0.235927	0.21
Random Forest	both	0.843917	0.838787	0.8491110	0.220810	0.3

With a truescore of .844, our *randomForest* algorithm managed to outperform our classification tree by nearly a full percentage point. It also improved upon the previously shortest Distance to (0, 1) by 1.5 percentage points. It's no surprise then to see that it also achieved a better balance between TPR and TNR than any of our previous models. Furthermore, our *randomForest* model actually performed better on the test set than on the train set, indicating that it was not overtrained. The *randomForest* algorithm lived up to its reputation as a superior approach in machine learning.

```
# Which variables did the randomForest algorithm consider to be the most important?

varImpPlot(rforest_train_c30, sort = TRUE, type = 2, main = "Importance of Variables")
```

Importance of Variables



```
importance(rforest_train_c30, type = 2)

##                                     MeanDecreaseGini
## launch_angle                  11928.5627
## launch_speed                  6802.3234
## spray_angle_Kolp              3568.5863
## spray_angle_adj                4001.7952
## if_fielding_alignment          772.9345
## hp_to_1b                      2812.9411

# Compare variable importance as estimated by our classification tree model and our
# random forest model.

rforest_train_var_imp <- as_tibble(importance(rforest_train_c30, type = 2))
rforest_train_var_imp <-
  mutate(rforest_train_var_imp,
         rforest_imp = round((MeanDecreaseGini /
                               sum(rforest_train_var_imp$MeanDecreaseGini)) * 100))

variable_imp <-
```

```

tibble(Predictor = c("launch_angle", "launch_speed", "spray_angle_Kolp",
                     "spray_angle_adj", "if_fielding_alignment", "hp_to_1b"),
       `ctree` = c(48, 32, 4, 13, 2, "< 1"),
       `rforest` = rforest_train_var_imp$rforest_imp)

knitr::kable(variable_imp[1:6, ], caption = "Variable Importance by Model")

```

Table 55: Variable Importance by Model

Predictor	ctree	rforest
launch_angle	48	40
launch_speed	32	23
spray_angle_Kolp	4	12
spray_angle_adj	13	13
if_fielding_alignment	2	3
hp_to_1b	< 1	9

Comparing our random forest and classification tree models in terms of variable importance, the Random Forest Model accorded more importance to *spray_angle_Kolp* and *hp_to_1b*, and less importance to *launch_angle* and *launch_speed*. The differences are fairly significant. We considered the proportions computed by Random Forest to be more in line with what we would have ideally hoped for, with all the predictors making more equal and meaningful contributions. *launch_angle* is still the most influential variable for predicting singles, as it should be. And *hp_to_1b* being accorded 9% significance justifies its inclusion in the study.

Mistaken Predictions

Since our Random Forest Model, *rforest_train_c30*, was our top performer, we decided to analyze its prediction mistakes to see if there might be ways to improve our ability to distinguish singles from outs. Our first step was to create data frames containing information about each batted ball having an outcome that was incorrectly predicted by the model.

While creating these data frames, we also decided to make some adjustments to our spray angle categories based on earlier findings. At the outset of this project, when we were first exploring the data, you may recall that some density plots caused us to revise our estimates of the spray angles corresponding to holes in the infield defense when the infielders are in a standard alignment. We wanted to incorporate those new estimates before undertaking our analysis of the mistaken predictions. Specifically, the gaps were now located at -30 to -23 (between the 3rd baseman and shortstop), -8 to 8 (up the middle, between the shortstop and second baseman), and 27 to 34 (between the second baseman and the first baseman).

```

# Create a data frame showing all erroneous predictions by our
# rforest_train_c30 algorithm, along with their potentially
# relevant characteristics.

# Combine all predictions with actual outcomes.

rf_predict_prob_tbl <- as_tibble(rforest_predict_prob)
rf_predict_tbl <- as_tibble(enframe(rforest_predict))
rf_predict_results <- bind_cols(rf_predict_prob_tbl, rf_predict_tbl)
rf_predict_results <- rf_predict_results %>%
  dplyr::mutate(obs_nmbr = name, prob_single = single, prob_out = field_out,

```

```

        pred = value) %>%
dplyr::select(obs_nmbr, prob_single, prob_out, pred)
rf_predict_results <- bind_cols(rf_predict_results, as_tibble(test_set3$events))
rf_predict_results <- dplyr::rename(rf_predict_results, obs = value)
rf_predict_results$prob_single <- as.numeric(rf_predict_results$prob_single)
rf_predict_results$prob_out <- as.numeric(rf_predict_results$prob_out)

# Identify the mistaken predictions.
rf_predict_results <- rf_predict_results %>%
  mutate(eval = ifelse(rf_predict_results$pred != rf_predict_results$obs, "mistake",
                      "-"))

# Quantify the magnitude of each prediction mistake.

rf_predict_results <- rf_predict_results %>%
  mutate(prob_diff = abs(prob_single - prob_out))

# Add additional information about each batted ball event.

rf_predict_results <- bind_cols(rf_predict_results, test_set[, 1:38])
rf_predict_results <- dplyr::select(rf_predict_results, -(hc_x), -(hc_y))

# Add new categorical variables based on *spray_angle_Kolp* and *spray_angle_adj*.
# Include the revised spray angle boundaries for the categories.

rf_predict_results <- rf_predict_results %>%
  mutate(rev_spray_angle_Kolp_cat = cut(rf_predict_results$spray_angle_Kolp,
                                         breaks = c(-90, -45.1, -40, -30, -23, -8, 8, 27, 34,
                                                   43, 45, 90),
                                         labels = c("-90:-45.1", "LF Line (-45:-40)",
                                                   "3rd Baseman (-39.9:-30)",
                                                   "Hole 5-6 (-29.9:-23)",
                                                   "Shortstop (-22.9:-8)",
                                                   "Hole Middle (-7.9:8)",
                                                   "2nd Baseman (8.1:27)",
                                                   "Hole 3-4 (27.1:34)",
                                                   "1st Baseman (34.1:43)",
                                                   "RF Line (43.1:45)", "45.1:90"))))

rf_predict_results <- rf_predict_results %>%
  mutate(rev_spray_angle_adj_cat = cut(rf_predict_results$spray_angle_adj,
                                         breaks = c(-90, -45.1, -43.1, -34.1, -27.1, -8.1, 8,
                                                   25, 35, 43, 45, 90),
                                         labels = c("-90:-45.1", "Pulled Down Line (-45:-43.1)",
                                                   "Pulled Corner Inf (-43:-34.1)",
                                                   "Pulled Side Hole (-34:-27.1)",
                                                   "Pulled Mid Inf (-27:-8.1)",
                                                   "Middle Hole (-8:7.9)",
                                                   "Oppo Mid Inf (8:22.9)",
                                                   "Oppo Side Hole (23:29.9)",
                                                   "Oppo Corner Inf (30:39.9)",
                                                   "Oppo Down Line (40:45)", "45.1:90"))))

```

```

# Reorder the columns in rf_predict_results.

rf_predict_results <- rf_predict_results %>%
  dplyr::select(obs_nmbr, prob_single, prob_out, pred, obs, eval, prob_diff,
    launch_angle, launch_speed, spray_angle_Kolp, spray_angle_adj,
    if_fielding_alignment, hp_to_1b,
    game_date, batter, player_name, age, stand, position, team, events,
    des, bb_type, adv_bb_type, launch_speed_cat,
    hc_x_Kolp, hc_y_Kolp, rev_spray_angle_Kolp_cat,
    rev_spray_angle_adj_cat, num_if_alignment,
    hp_to_1b_cat, of_fielding_alignment, hit_location, hit_distance_sc,
    launch_speed_angle, game_pk, game_year, game_type, home_team, away_team,
    pitcher, p_throws, description)

# Create a new data frame with just the mistaken predictions.

rf_predict_mistakes <- rf_predict_results %>% dplyr::filter(eval == "mistake")

# Create a new data frame with just the correct predictions.

rf_predict_correct <- rf_predict_results %>% dplyr::filter(eval != "mistake")

# Create a sample of correct predictions equal in size to the number of
# mistaken predictions.

nrow(rf_predict_mistakes)

## [1] 3087

set.seed(1)
rf_predict_correct_sample_3087 <- sample_n(rf_predict_correct, 3087, replace = FALSE)

```

We had a total of 3,087 mistaken predictions out of the 20,118 batted ball events in test_set3, a 15.3% error rate. Recall, though, that our goal was not to minimize this error rate. A lower error rate could have been achieved had we been willing to accept a much lower true positive rate. Instead, we sought to achieve a balance between the true positive rate and the true negative rate (as measured by our truescore metric) because our ultimate goal was to be able to distinguish singles from outs.

```

# Was it more common to mistakenly predict singles (false positive)
# or outs (false negative)?

prop_false_pos <- mean(rf_predict_mistakes$pred == "single")
prop_false_neg <- mean(rf_predict_mistakes$pred == "field_out")

# Add a column to our data frame of prediction mistakes that describes the
# type of mistake that was made.

rf_predict_mistakes <- rf_predict_mistakes %>%
  mutate(eval_type = ifelse(pred == "single", "FalsePos", "FalseNeg"))
rf_predict_mistakes <- rf_predict_mistakes %>%
  dplyr::select(1:6, eval_type, everything())

```

```

# Add a column to our data frame of correct predictions that describes the
# type of correct prediction that was made.

rf_predict_correct_sample_3087 <- rf_predict_correct_sample_3087 %>%
  mutate(eval_type = ifelse(pred == "single", "TruePos", "TrueNeg"))
rf_predict_correct_sample_3087 <- rf_predict_correct_sample_3087 %>%
  dplyr::select(1:6, eval_type, everything())

# Combine our data frames of prediction mistakes and correct predictions.

predictions_6174 <- bind_rows(rf_predict_mistakes, rf_predict_correct_sample_3087)

```

Erroneous single predictions (false positives) comprised nearly three-fourths of the mistakes made by our random forest algorithm. These were batted balls that actually resulted in outs when the model predicted they were singles. They're represented by the blue points in the following plot of singles predictions. An equally sized sample of correct singles predictions were included for reference, and are represented by the red points.

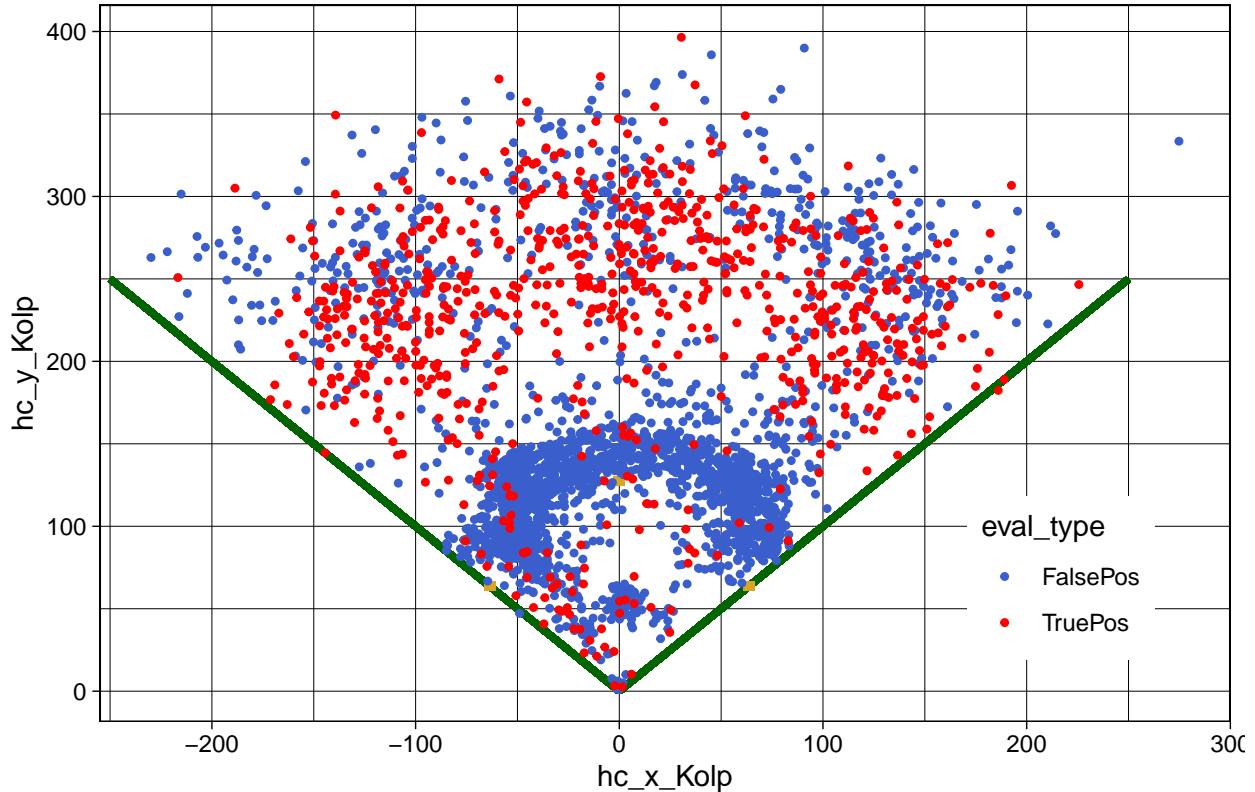
```

# Visualize false positive and true positive predictions by rforest_train_c30 model.

(Pos_predictions <- ggplot(dplyr::filter(predictions_6174,
                                         eval_type %in% c("FalsePos", "TruePos")),
                           aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c("royalblue3", "red")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = .9) + theme_linedraw() +
  labs(title = "False Positive Predictions by rforest_train_c30 Model") +
  theme(legend.position = c(.85, .2)))

```

False Positive Predictions by rforest_train_c30 Model

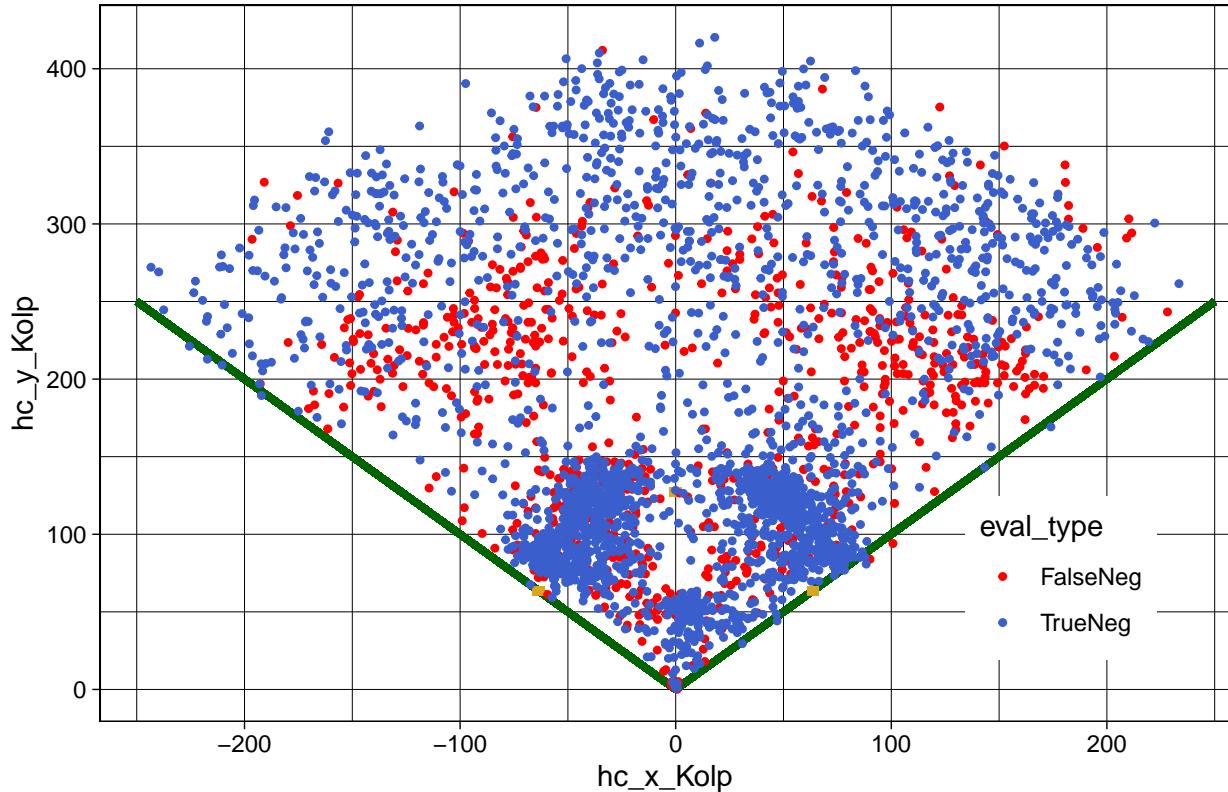


The model made far fewer false negative mistakes, that is, predictions of outs when, in actuality, the batted balls resulted in singles. They're represented by the red points in the following plot of outs predictions. An equally sized sample of correct outs predictions were included for reference, and are represented by the blue points.

```
# Visualize false negative and true negative predictions by rforest_train_c30 model.
```

```
(Neg_predictions <- ggplot(dplyr::filter(predictions_6174,
                                             eval_type %in% c("FalseNeg", "TrueNeg")),
                            aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = .9) + theme_linedraw() +
  labs(title = "False Negative Predictions by rforest_train_c30 Model") +
  theme(legend.position = c(.85, .2)))
```

False Negative Predictions by rforest_train_c30 Model



Here, we see the inverse of what we saw in the plot of singles predictions. Many of the false negatives reached the shallow outfield, while correctly predicted outs were either stopped in the infield or caught deeper in the outfield. So the question for false negatives was, what was it about those batted balls that made the model think they were going to be stopped in the infield or caught deeper in the outfield?

To answer our questions, we first examined the values of the predictor variables for the false positives, beginning with *launch_angle* and its categorical version, *adv_bb_type*. The idea was to compare, for each of the model's features, the distributions of false positives and true positives, and the distributions of false negatives and true negatives. This would first be done with a visualization of Tukey's five number summary, in the form of box plots. Density plots would then round out the picture, filling in the details. We were hoping to see patterns showing the kinds of batted balls the model mistakenly took for singles, and the types of batted balls the model mistakenly took for outs. First, the false positives.

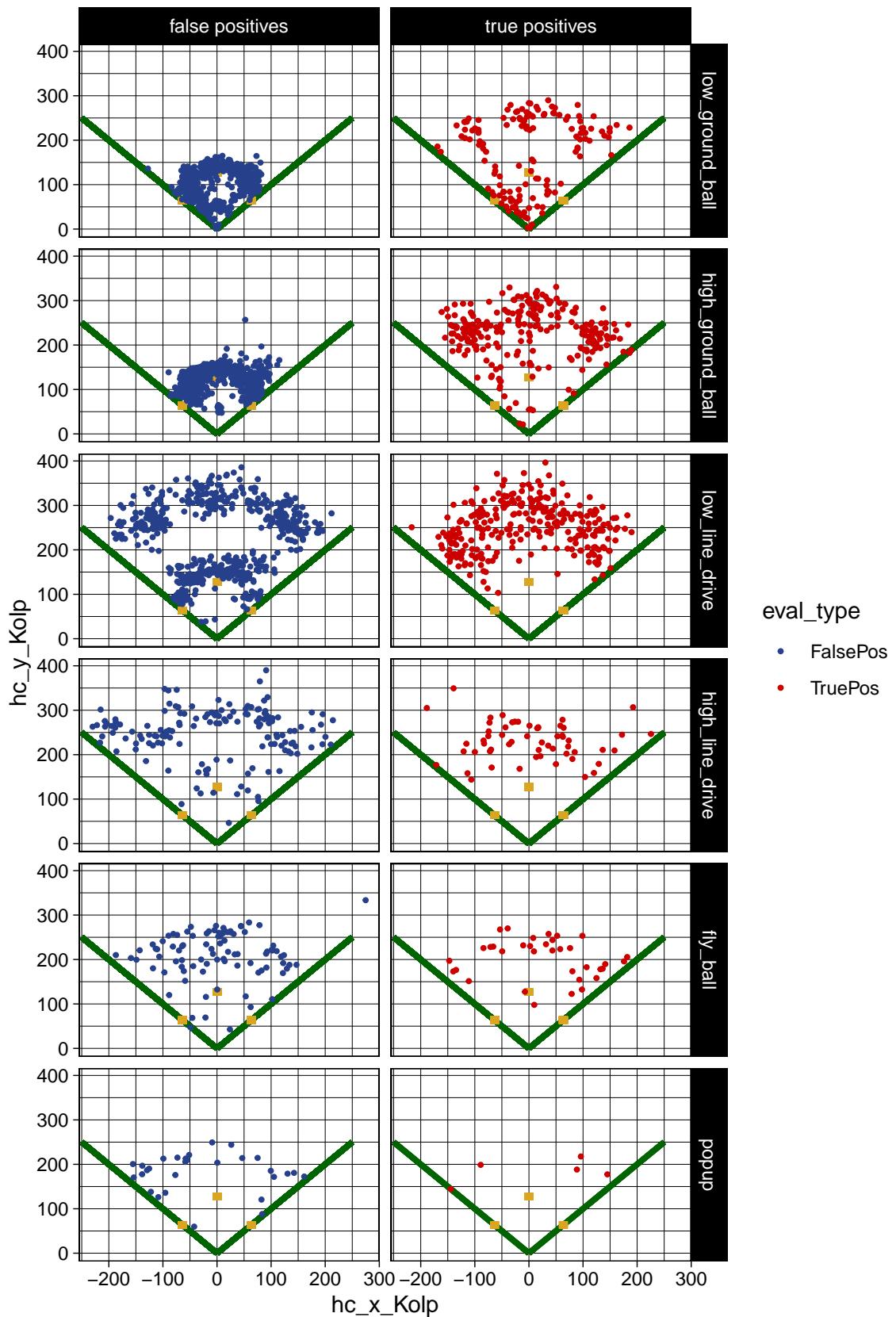
```
# Visualize false positive and true positive predictions by adv_bb_type.

pos_labels <- c("false positives", "true positives")
names(pos_labels) <- c("FalsePos", "TruePos")

ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
       aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c(FalsePos = "royalblue4", TruePos = "red3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
```

```
geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
  geom_point(size = 0.7) + theme_linedraw() +
  labs(title = "False Positives by adv_bb_type") +
  facet_grid(adv_bb_type ~ eval_type, labeller = labeller(eval_type = pos_labels))
```

False Positives by adv_bb_type



Most of the false positives appear to be ground balls or low line drives. While high ground balls and low line drives are the best launch angle categories for singles, low ground balls have some of the worst launch angles for singles.

```
# Using boxplots, compare false positive and true positive predictions by launch_angle.

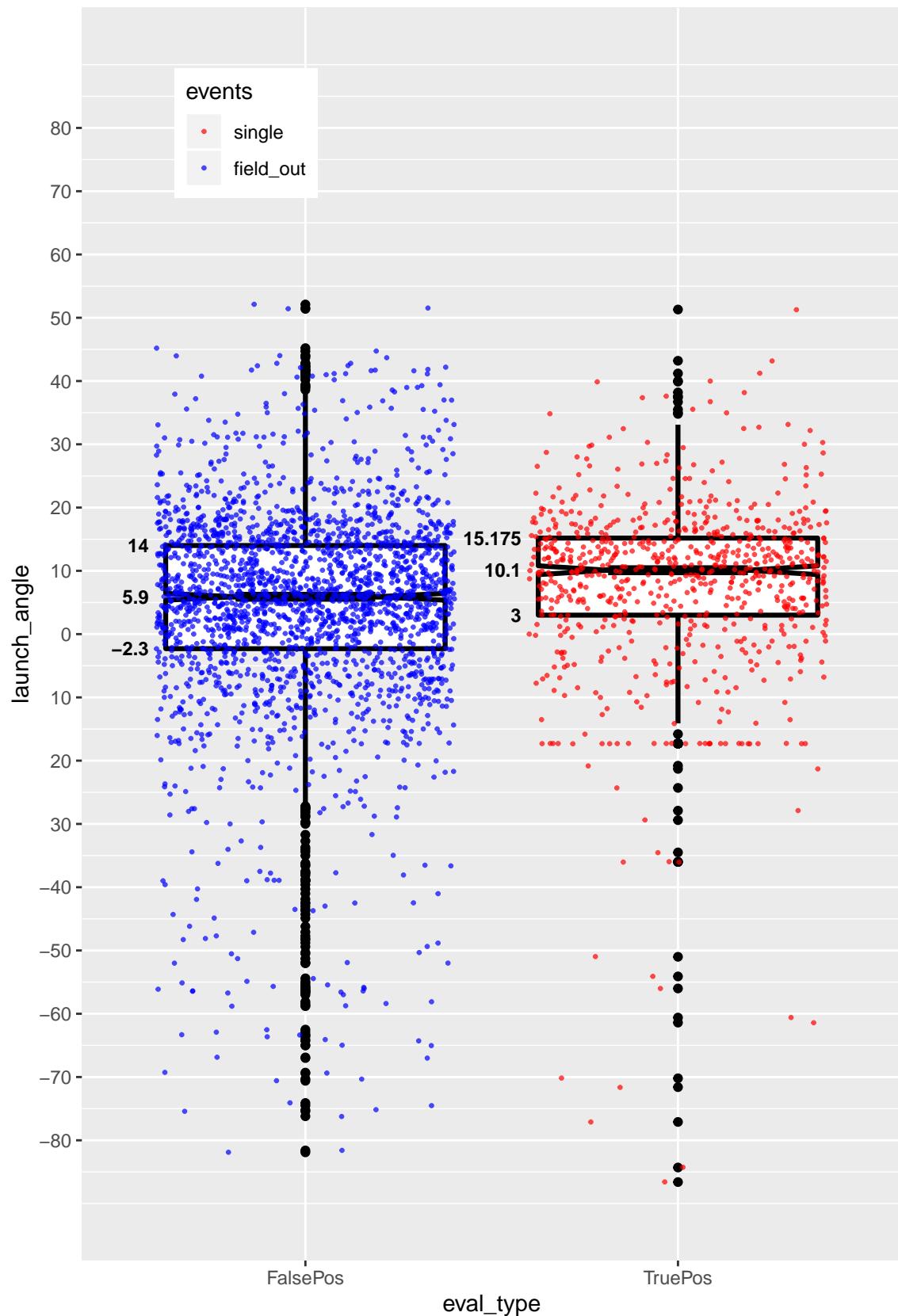
pos_predictions_by_launch_angle <-
  ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
    aes(x = eval_type, y = launch_angle, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-80, 80, by = 10),
    labels = c("-80", "-70", "-60", "-50", "-40", "30", "20", "10", "0",
              "10", "20", "30", "40", "50", "60", "70", "80"),
    limits = c(-90, 90)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Positives, by Launch Angle") +
  theme(legend.position = c(.20, .90))

pos_predictions_by_launch_angle_data <- layer_data(pos_predictions_by_launch_angle)

launch_angle_FalsePos_1quartile <- pos_predictions_by_launch_angle_data[1, 2]
launch_angle_FalsePos_median <- pos_predictions_by_launch_angle_data[1, 3]
launch_angle_FalsePos_3quartile <- pos_predictions_by_launch_angle_data[1, 4]
launch_angle_TruePos_1quartile <- pos_predictions_by_launch_angle_data[2, 2]
launch_angle_TruePos_median <- pos_predictions_by_launch_angle_data[2, 3]
launch_angle_TruePos_3quartile <- pos_predictions_by_launch_angle_data[2, 4]

(pos_predictions_by_launch_angle <-
  ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
    aes(x = eval_type, y = launch_angle, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-80, 80, by = 10),
    labels = c("-80", "-70", "-60", "-50", "-40", "30", "20", "10", "0",
              "10", "20", "30", "40", "50", "60", "70", "80"),
    limits = c(-90, 90)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Positives, by Launch Angle") +
  theme(legend.position = c(.20, .90)) +
  annotate("text", x = c(.58, .58, .58), size = 3, fontface = 2, hjust = 1,
    y = c(launch_angle_FalsePos_1quartile, launch_angle_FalsePos_median,
          launch_angle_FalsePos_3quartile),
    label = c(launch_angle_FalsePos_1quartile, launch_angle_FalsePos_median,
              launch_angle_FalsePos_3quartile)) +
  annotate("text", x = c(1.58, 1.58, 1.58), size = 3, fontface = 2, hjust = 1,
    y = c(launch_angle_TruePos_1quartile, launch_angle_TruePos_median,
          launch_angle_TruePos_3quartile),
    label = c(launch_angle_TruePos_1quartile, launch_angle_TruePos_median,
              launch_angle_TruePos_3quartile)))
```

True vs False Positives, by Launch Angle

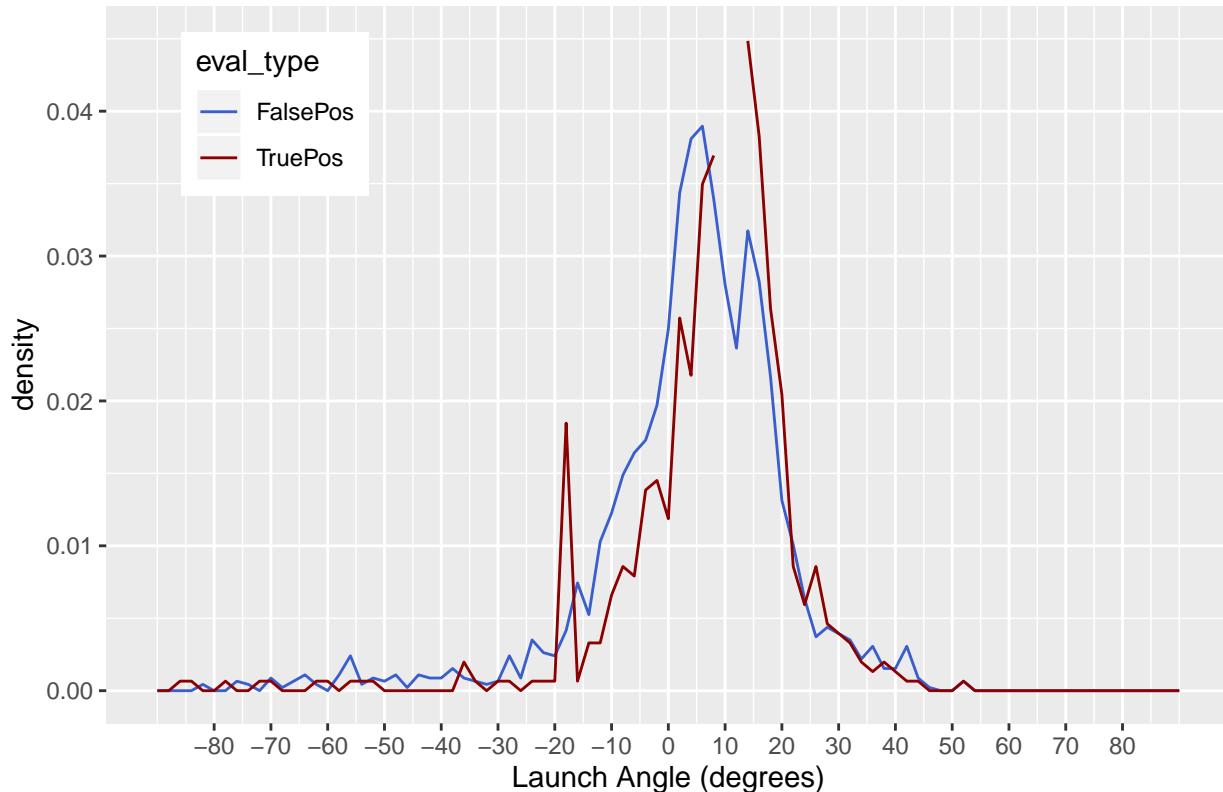


The box plots affirm that the false positives are dipping into negative launch angles (low ground balls). Overall, the false positives are being hit at lower launch angles than actual singles, evidenced by a median that's more than four points lower than that of actual singles.

```
# Compare the launch_angle densities of False Positives and True Positives.

x_scale <- scale_x_continuous(breaks = seq(-80, 80, by = 10),
                               labels = c("-80", "-70", "-60", "-50", "-40", "-30",
                                         "-20", "-10", "0", "10", "20", "30", "40",
                                         "50", "60", "70", "80"),
                               limits = c(-90, 90))
y_scale <- scale_y_continuous(limits = c(0, 0.045))
(la_density_pos <- ggplot(dplyr::filter(predictions_6174,
                                         eval_type == "FalsePos" | eval_type == "TruePos"),
                           aes(x = launch_angle, y = stat(density), color = eval_type)) +
  geom_freqpoly(binwidth = 2) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(FalsePos = "royalblue3", TruePos = "red4")) +
  labs(title = "Density Plots of Launch Angle by FalsePos and TruePos",
       x = "Launch Angle (degrees)") +
  theme(legend.position = c(.15, .85)))
```

Density Plots of Launch Angle by FalsePos and TruePos



The density plots show false positives occurring at a greater frequency below a launch angle of about 7 degrees.

What about launch speeds?

```

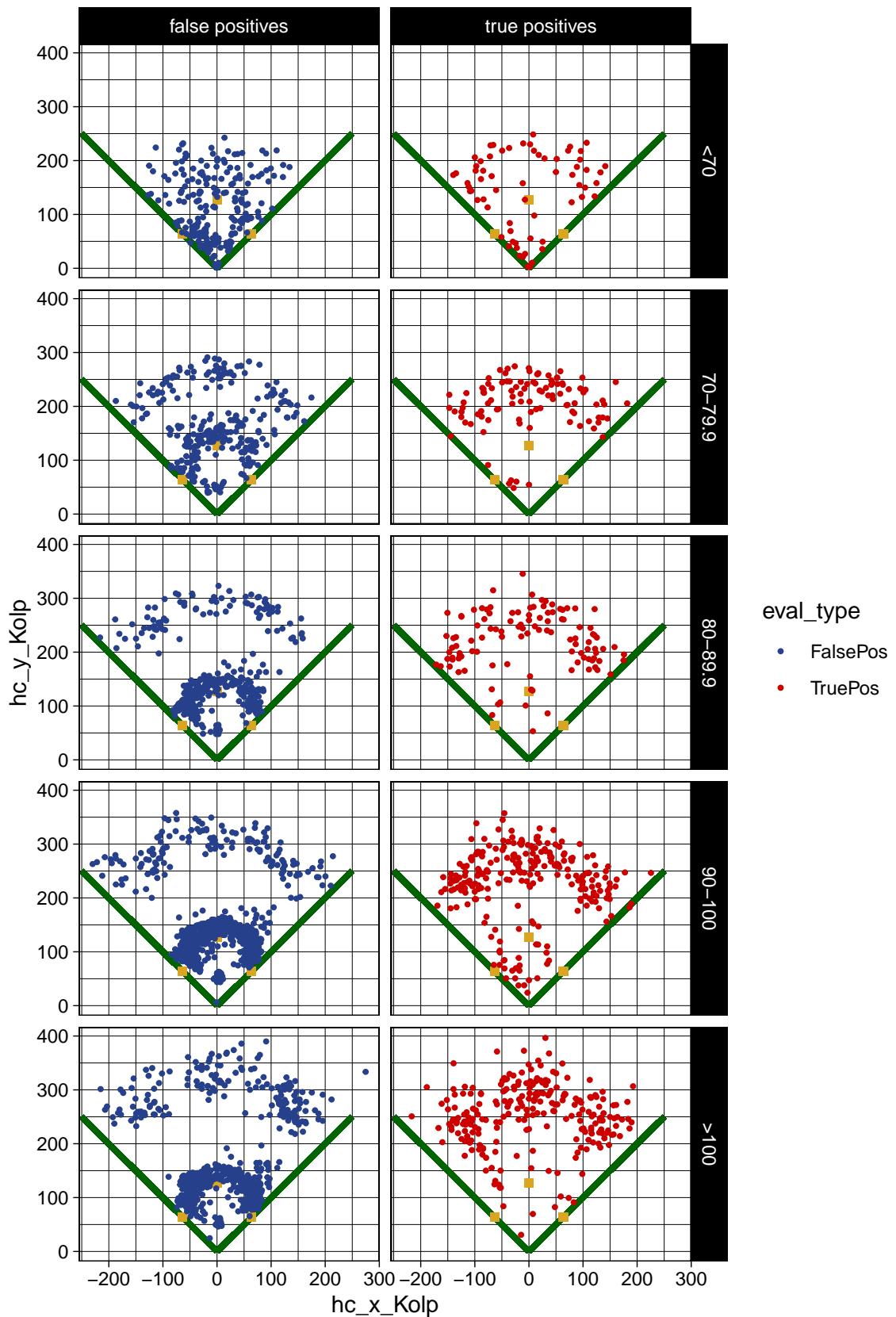
# Visualize false positive and true positive predictions by launch_speed_cat.

pos_labels <- c("false positives", "true positives")
names(pos_labels) <- c("FalsePos", "TruePos")

ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
       aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c(FalsePos = "royalblue4", TruePos = "red3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = .7) + theme_linedraw() +
  labs(title = "False Positives by launch_speed_cat") +
  facet_grid(launch_speed_cat ~ eval_type, labeller = labeller(eval_type = pos_labels))

```

False Positives by launch_speed_cat



Many of the false positives with higher exit velocities (above 90 miles per hour) are being successfully fielded in the infield. This is consistent with what we learned earlier about the false positives being hit at lower launch angles.

```
# Using boxplots, compare false positive and true positive predictions by launch_speed.

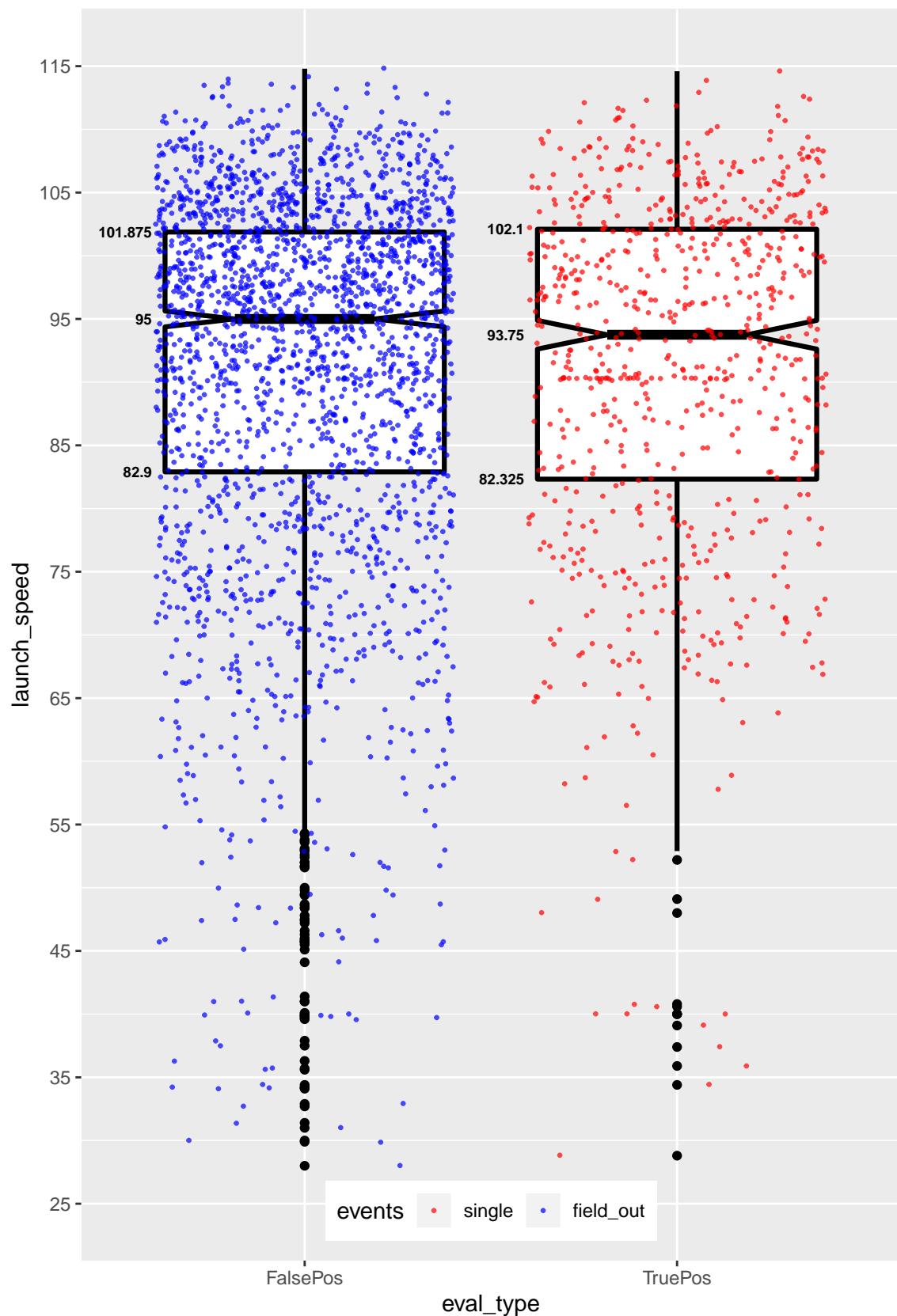
pos_predictions_by_launch_speed <-
  ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
         aes(x = eval_type, y = launch_speed, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(25, 115, by = 10),
                     labels = c("25", "35", "45", "55", "65", "75", "85", "95", "105",
                               "115"),
                     limits = c(25, 115)) +
  scale_color_manual(values = c(field_out = "Blue", single = "Red")) +
  ggtitle("True vs False Positives, by Launch Speed") +
  theme(legend.position = c(.5, .04), legend.direction = "horizontal")

pos_predictions_by_launch_speed_data <- layer_data(pos_predictions_by_launch_speed)

launch_speed_FalsePos_1quartile <- pos_predictions_by_launch_speed_data[1, 2]
launch_speed_FalsePos_median <- pos_predictions_by_launch_speed_data[1, 3]
launch_speed_FalsePos_3quartile <- pos_predictions_by_launch_speed_data[1, 4]
launch_speed_TruePos_1quartile <- pos_predictions_by_launch_speed_data[2, 2]
launch_speed_TruePos_median <- pos_predictions_by_launch_speed_data[2, 3]
launch_speed_TruePos_3quartile <- pos_predictions_by_launch_speed_data[2, 4]

(pos_predictions_by_launch_speed <-
  ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
         aes(x = eval_type, y = launch_speed, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(25, 115, by = 10),
                     labels = c("25", "35", "45", "55", "65", "75", "85", "95", "105",
                               "115"),
                     limits = c(25, 115)) +
  scale_color_manual(values = c(field_out = "Blue", single = "Red")) +
  ggtitle("True vs False Positives, by Launch Speed") +
  theme(legend.position = c(.5, .04), legend.direction = "horizontal") +
  annotate("text", x = c(.59, .59, .59), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_speed_FalsePos_1quartile, launch_speed_FalsePos_median,
                launch_speed_FalsePos_3quartile),
          label = c(launch_speed_FalsePos_1quartile, launch_speed_FalsePos_median,
                    launch_speed_FalsePos_3quartile)) +
  annotate("text", x = c(1.59, 1.59, 1.59), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_speed_TruePos_1quartile, launch_speed_TruePos_median,
                launch_speed_TruePos_3quartile),
          label = c(launch_speed_TruePos_1quartile, launch_speed_TruePos_median,
                    launch_speed_TruePos_3quartile)))
```

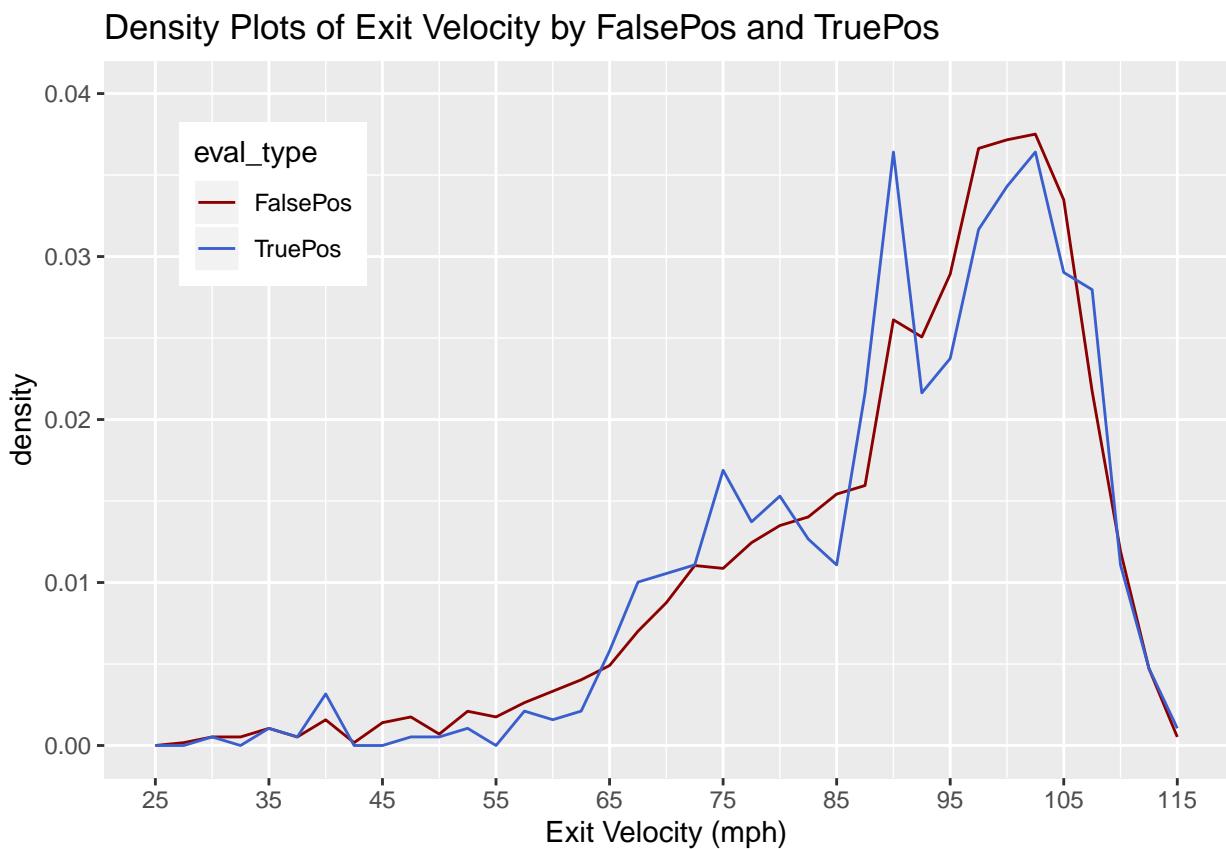
True vs False Positives, by Launch Speed



The false positives aren't being hit significantly harder or softer than actual singles, though the median launch speed for false positives is slightly higher (95 as compared to 93.75 miles per hour).

```
# Compare the launch_speed densities of False Positives and True Positives.

x_scale <- scale_x_continuous(breaks = seq(25, 115, by = 10),
                               labels = c("25", "35", "45", "55", "65", "75", "85", "95",
                                         "105", "115"),
                               limits = c(25, 115))
y_scale <- scale_y_continuous(limits = c(0, 0.04))
ls_density_pos <-
  ggplot(dplyr::filter(predictions_6174,
                        eval_type == "FalsePos" | eval_type == "TruePos"),
         aes(x = launch_speed, y = stat(density), color = eval_type)) +
  geom_freqpoly(binwidth = 2.5) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(FalsePos = "red4", TruePos = "royalblue3")) +
  labs(title = "Density Plots of Exit Velocity by FalsePos and TruePos",
       x = "Exit Velocity (mph)") +
  theme(legend.position = c(.15, .8))
```



Here, we can see more precisely that false positives occur with greater frequency at launch speeds between 93 and 105 miles per hour.

Kolp's version of spray angle was the next feature we examined.

```

# Visualize false positive and true positive predictions by rev_spray_angle_Kolp_cat.

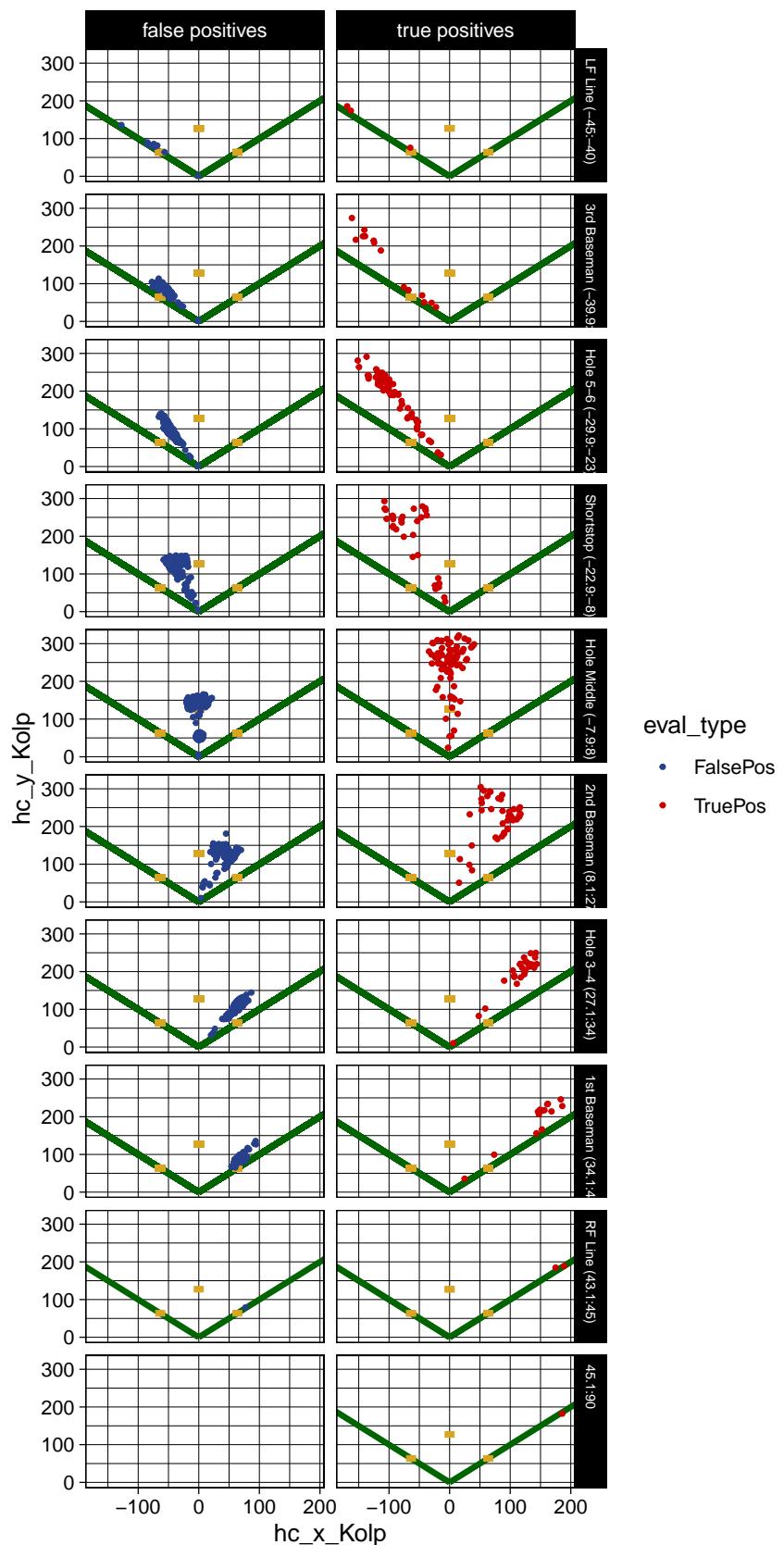
predictions_6174_std_gb <-
  dplyr::filter(predictions_6174, if_fielding_alignment == "Standard" &
                 eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_std_gb <- predictions_6174_std_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
                adv_bb_type == "high_ground_ball")

pos_labels <- c("false positives", "true positives")
names(pos_labels) <- c("FalsePos", "TruePos")

(falsePos_predictions_by_rev_spray_angle_Kolp_cat <-
  ggplot(dplyr::filter(predictions_6174_std_gb,
                        eval_type %in% c("FalsePos", "TruePos")),
         aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c(FalsePos = "royalblue4", TruePos = "red3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green",
               size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = .7) + theme_linedraw() +
  labs(title = "False Positives by spray_angle_Kolp (Standard infield)") +
  facet_grid(rev_spray_angle_Kolp_cat ~ eval_type,
             labeller = labeller(eval_type = pos_labels)) +
  theme(strip.text.y = element_text(size = 6, hjust = 0)))

```

False Positives by spray_angle_Kolp (Standard infield)



```
# facet_grid(eval_type ~ rev_spray_angle_Kolp_cat,
# labeller = labeller(eval_type = pos_labels)))
```

The batted ball plots were not very informative here. The most we can say is that erroneously predicted singles are being stopped by infielders in virtually every category of *spray_angle_Kolp*.

```
# Using boxplots, compare false positive and true positive predictions by
# spray_angle_Kolp.

predictions_6174_std_gb <-
  dplyr::filter(predictions_6174, if_fielding_alignment == "Standard" &
    eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_std_gb <- predictions_6174_std_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
    adv_bb_type == "high_ground_ball")

pos_predictions_by_spray_angle_Kolp <-
  ggplot(predictions_6174_std_gb, aes(x = eval_type, y = spray_angle_Kolp,
    color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
    labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
      "-10", "-5", "0", "5", "10", "15", "20", "25", "30",
      "35", "40", "45", "50"),
    limits = c(-50, 50)) +
  scale_color_manual(values = c(single = "Red", field_out = "Blue")) +
  ggtitle("True vs False Positives, by spray_angle_Kolp (Inf Standard)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal")

pos_predictions_by_spray_angle_Kolp_data <-
  layer_data(pos_predictions_by_spray_angle_Kolp)

spray_angle_Kolp_FalsePos_1quartile <-
  round(pos_predictions_by_spray_angle_Kolp_data[1, 2], 1)
spray_angle_Kolp_FalsePos_median <-
  round(pos_predictions_by_spray_angle_Kolp_data[1, 3], 1)
spray_angle_Kolp_FalsePos_3quartile <-
  round(pos_predictions_by_spray_angle_Kolp_data[1, 4], 1)
spray_angle_Kolp_TruePos_1quartile <-
  round(pos_predictions_by_spray_angle_Kolp_data[2, 2], 1)
spray_angle_Kolp_TruePos_median <-
  round(pos_predictions_by_spray_angle_Kolp_data[2, 3], 1)
spray_angle_Kolp_TruePos_3quartile <-
  round(pos_predictions_by_spray_angle_Kolp_data[2, 4], 1)

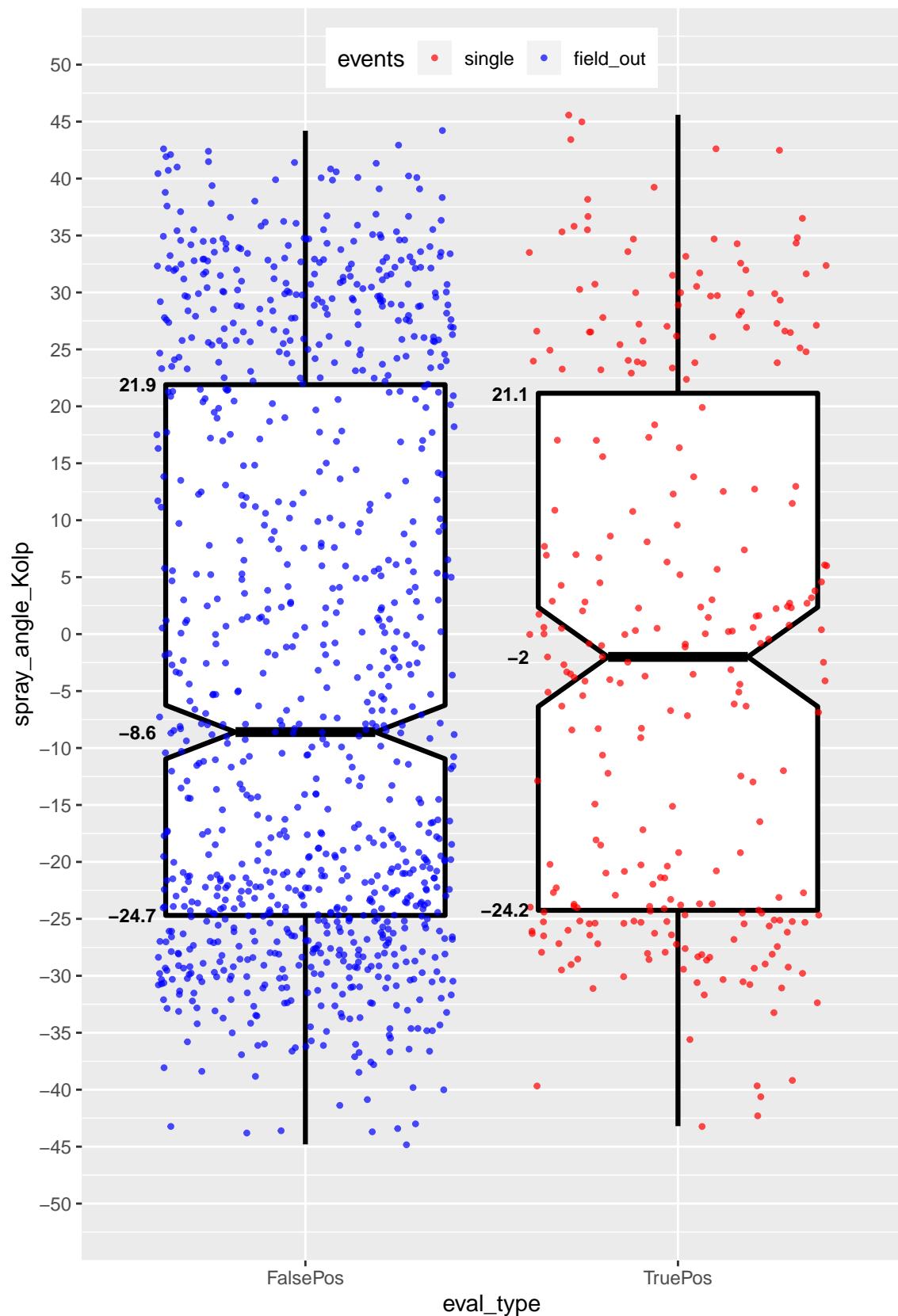
(pos_predictions_by_spray_angle_Kolp <-
  ggplot(predictions_6174_std_gb, aes(x = eval_type,
    y = spray_angle_Kolp, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
    labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
```

```

        "-10", "-5", "0", "5", "10", "15", "20", "25", "30",
        "35", "40", "45", "50"),
limits = c(-50, 50)) +
scale_color_manual(values = c(single = "Red", field_out = "Blue")) +
ggtitle("True vs False Positives, by spray_angle_Kolp (Inf Standard)") +
theme(legend.position = c(.5, .96), legend.direction = "horizontal") +
annotate("text", x = c(.60, .60, .60), size = 3, fontface = 2, hjust = 1,
y = c(spray_angle_Kolp_FalsePos_1quartile, spray_angle_Kolp_FalsePos_median,
spray_angle_Kolp_FalsePos_3quartile),
label = c(spray_angle_Kolp_FalsePos_1quartile,
spray_angle_Kolp_FalsePos_median,
spray_angle_Kolp_FalsePos_3quartile)) +
annotate("text", x = c(1.60, 1.60, 1.60), size = 3, fontface = 2, hjust = 1,
y = c(spray_angle_Kolp_TruePos_1quartile, spray_angle_Kolp_TruePos_median,
spray_angle_Kolp_TruePos_3quartile),
label = c(spray_angle_Kolp_TruePos_1quartile,
spray_angle_Kolp_TruePos_median,
spray_angle_Kolp_TruePos_3quartile)))

```

True vs False Positives, by spray_angle_Kolp (Inf Standard)



The medians reflected in the boxplots tell us the false positives are being hit more toward the left side of the field than are actual singles. The false positives also appear to be most concentrated near the 3-4 and 5-6 holes in the infield, though it is difficult to judge proportions from the boxplots. We hoped the density plots would provide more clarity.

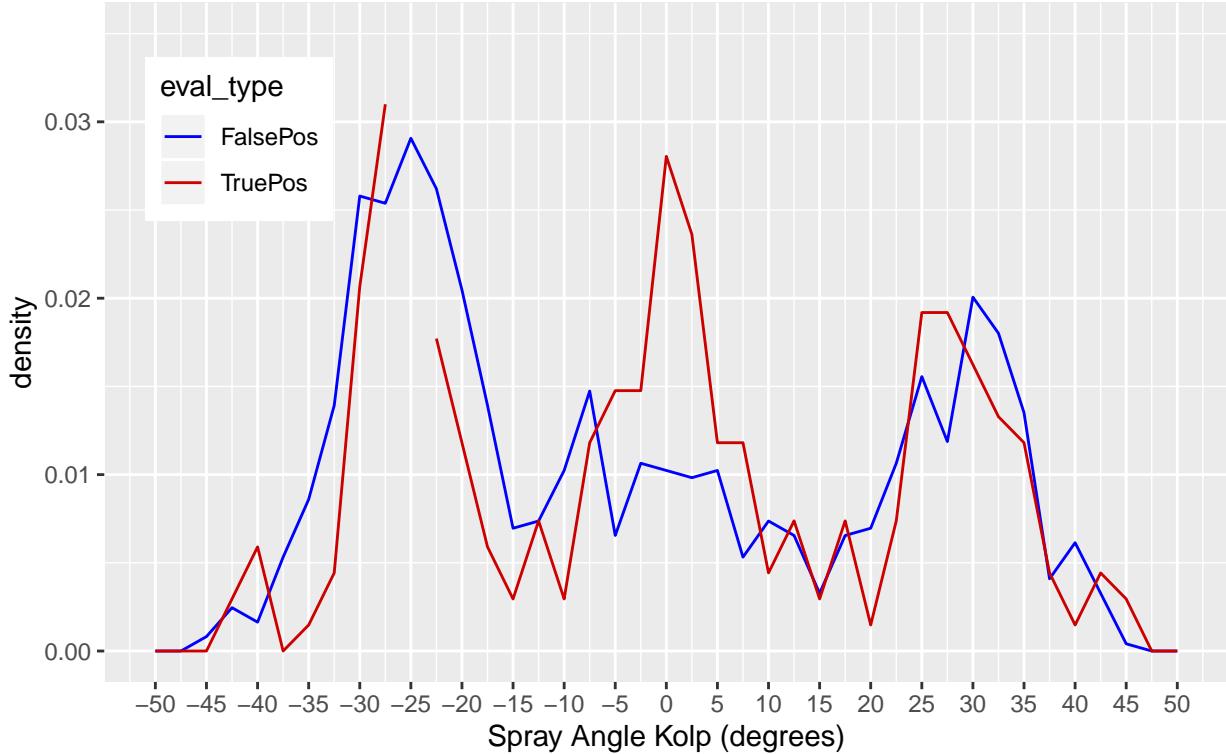
```
# Compare the spray_angle_Kolp densities of False Positives and True Positives.

predictions_6174_std_gb <-
  dplyr::filter(predictions_6174,
    if_fielding_alignment == "Standard" &
      eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_std_gb <- predictions_6174_std_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
    adv_bb_type == "high_ground_ball")

x_scale <- scale_x_continuous(breaks = seq(-50, 50, by = 5),
  labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20",
            "-15", "-10", "-5", "0",
            "5", "10", "15", "20", "25", "30", "35", "40",
            "45", "50"),
  limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))

(sa_Kolp_density_pos <-
  ggplot(predictions_6174_std_gb, aes(x = spray_angle_Kolp, y = stat(density),
    color = eval_type)) +
  geom_freqpoly(binwidth = 2.5) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(FalsePos = "blue", TruePos = "red3")) +
  labs(title = "Density Plots of spray_angle_Kolp by FalsePos and TruePos
  (Standard If Alignment)", x = "Spray Angle Kolp (degrees)") +
  theme(legend.position = c(.12, .80))
```

Density Plots of spray_angle_Kolp by FalsePos and TruePos (Standard If Alignment)



The density plots may have added some clarity, but it's not a simple picture. False positives occurred more frequently on balls hit toward the third baseman and shortstop, and to a lesser extent on balls hit in the 3-4 hole.

We moved on to our *spray_angle_adj* feature.

```
# Visualize false positive and true positive predictions by rev_spray_angle_adj_cat.

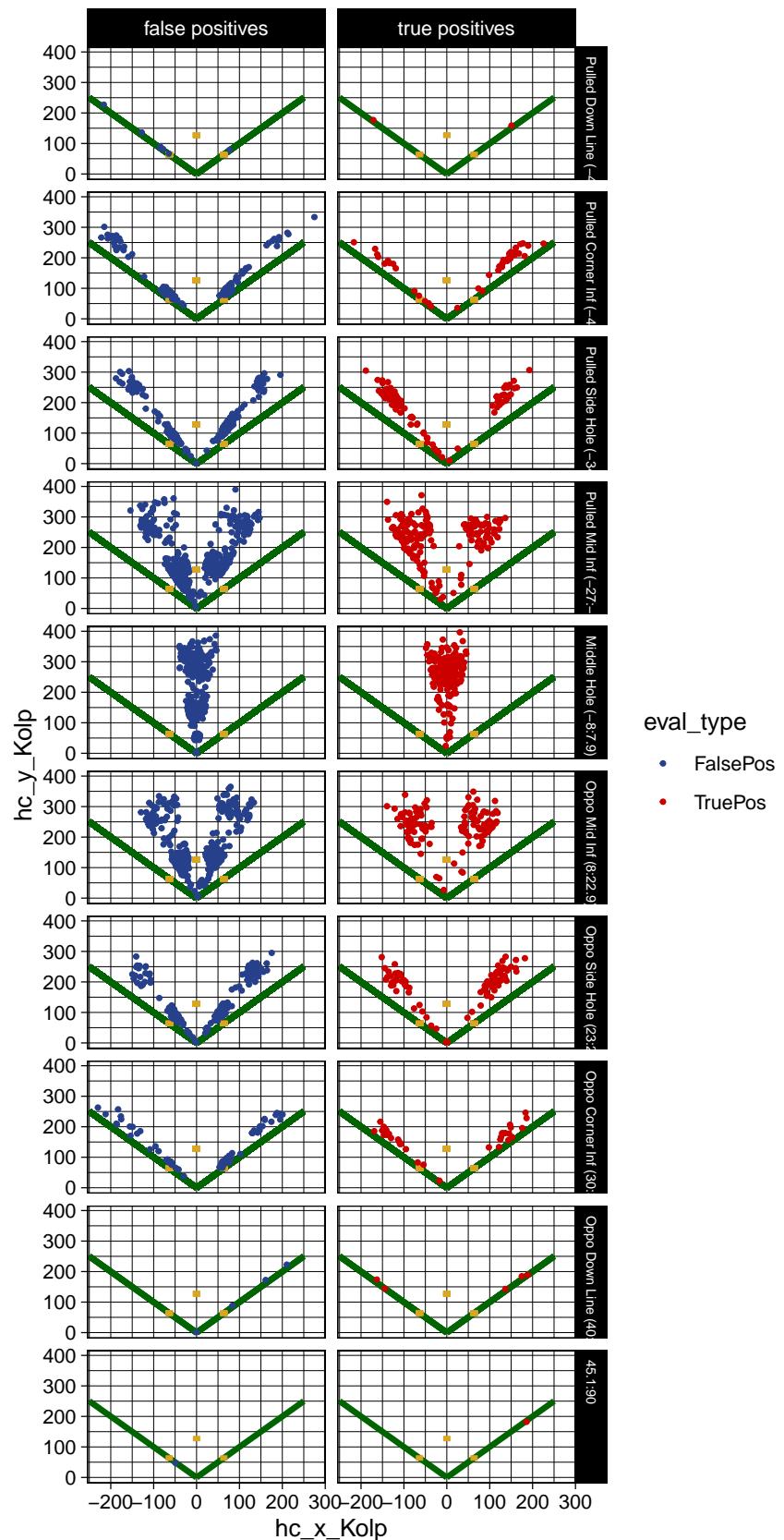
predictions_6174_shift_gb <-
  dplyr::filter(predictions_6174, if_fielding_alignment == "Infield shift" &
    eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_shift_gb <- predictions_6174_shift_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
    adv_bb_type == "high_ground_ball")

pos_labels <- c("false positives", "true positives")
names(pos_labels) <- c("FalsePos", "TruePos")

(falsePos_predictions_by_rev_spray_angle_adj_cat <-
  ggplot(dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos")),
    aes(hc_x_Kolp, hc_y_Kolp, color = eval_type)) +
  scale_color_manual(values = c(FalsePos = "royalblue4", TruePos = "red3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green",
    size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green",
    size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
```

```
    fill = "goldenrod") +
geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
geom_point(size = .7) + theme_linedraw() +
labs(title = "False Positives by spray_angle_adj (Infield shift)") +
facet_grid(rev_spray_angle_adj_cat ~ eval_type,
           labeller = labeller(eval_type = pos_labels)) +
theme(strip.text.y = element_text(size = 6, hjust = 0)))
```

False Positives by spray_angle_adj (Infield shift)



```
#facet_grid(eval_type ~ rev_spray_angle_adj_cat,
#labeller = labeller(eval_type = pos_labels)))
```

Again, all we can really say, based on the batted ball plots, is that more of the false positives are being stopped in the infield, regardless of whether the ball is being pulled or hit the opposite way.

```
# Using boxplots, compare false positive and true positive predictions by spray_angle_adj.

predictions_6174_shift_gb <-
  dplyr::filter(predictions_6174, if_fielding_alignment == "Infield shift" &
    eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_shift_gb <- predictions_6174_shift_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
    adv_bb_type == "high_ground_ball")

pos_predictions_by_spray_angle_adj <- ggplot(predictions_6174_shift_gb,
                                               aes(x = eval_type, y = spray_angle_adj,
                                                   color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
                     labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                               "-10", "-5", "0", "5", "10", "15", "20", "25", "30",
                               "35", "40", "45", "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c(single = "Red", field_out = "Blue")) +
  ggtitle("True vs False Positives, by spray_angle_adj (Infield shift)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal")

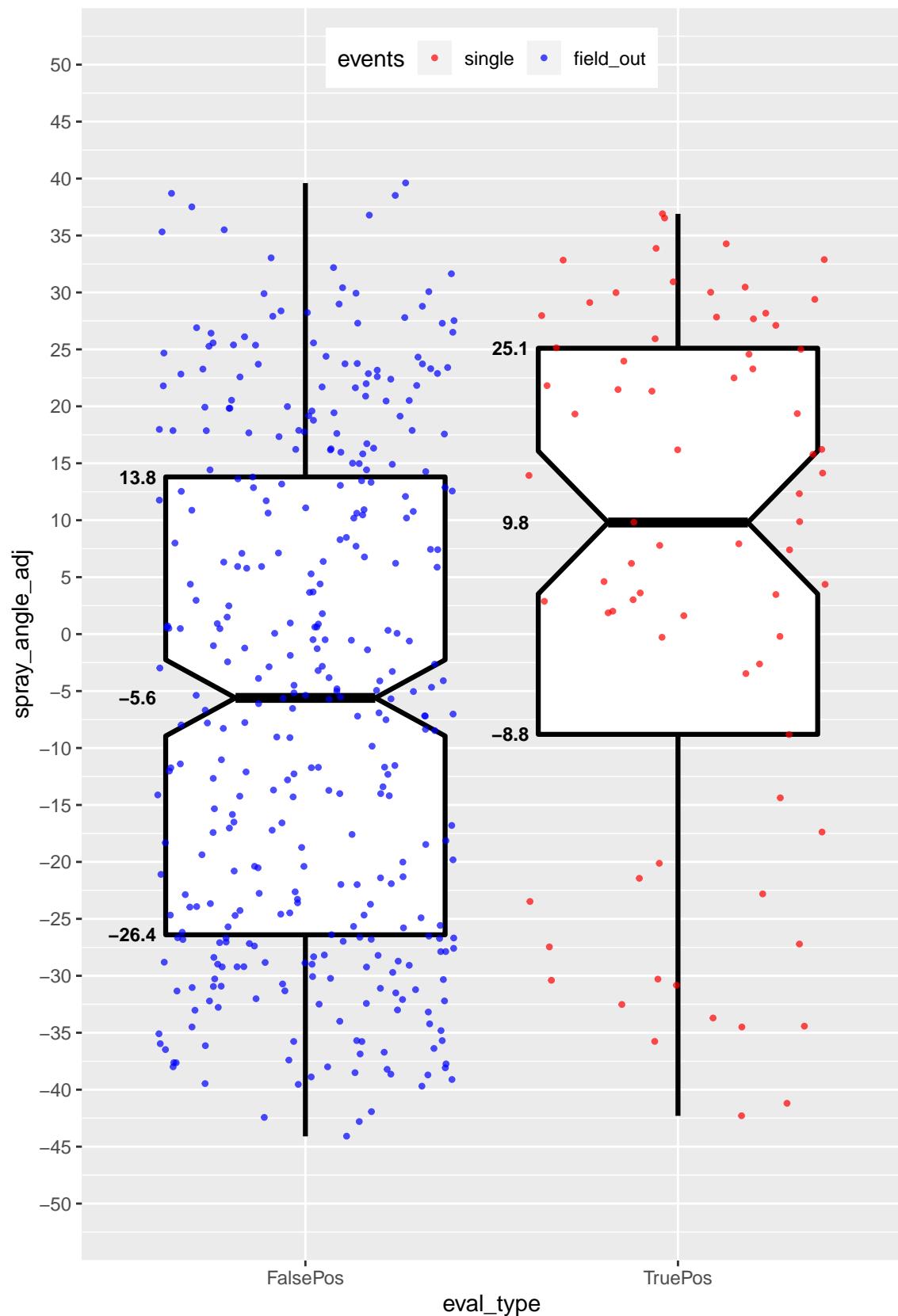
pos_predictions_by_spray_angle_adj_data <- layer_data(pos_predictions_by_spray_angle_adj)

spray_angle_adj_FalsePos_1quartile <-
  round(pos_predictions_by_spray_angle_adj_data[1, 2], 1)
spray_angle_adj_FalsePos_median <-
  round(pos_predictions_by_spray_angle_adj_data[1, 3], 1)
spray_angle_adj_FalsePos_3quartile <-
  round(pos_predictions_by_spray_angle_adj_data[1, 4], 1)
spray_angle_adj_TruePos_1quartile <-
  round(pos_predictions_by_spray_angle_adj_data[2, 2], 1)
spray_angle_adj_TruePos_median <-
  round(pos_predictions_by_spray_angle_adj_data[2, 3], 1)
spray_angle_adj_TruePos_3quartile <-
  round(pos_predictions_by_spray_angle_adj_data[2, 4], 1)

(pos_predictions_by_spray_angle_adj <-
  ggplot(predictions_6174_shift_gb, aes(x = eval_type, y = spray_angle_adj,
                                         color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
                     labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                               "-10", "-5", "0", "5", "10", "15", "20", "25", "30",
                               "35", "40", "45", "50"),
```

```
limits = c(-50, 50)) +
scale_color_manual(values = c(single = "Red", field_out = "Blue")) +
ggtitle("True vs False Positives, by spray_angle_adj (Infield shift)") +
theme(legend.position = c(.5, .96), legend.direction = "horizontal") +
annotate("text", x = c(.60, .60, .60), size = 3, fontface = 2, hjust = 1,
y = c(spray_angle_adj_FalsePos_1quartile, spray_angle_adj_FalsePos_median,
spray_angle_adj_FalsePos_3quartile),
label = c(spray_angle_adj_FalsePos_1quartile,
spray_angle_adj_FalsePos_median,
spray_angle_adj_FalsePos_3quartile)) +
annotate("text", x = c(1.60, 1.60, 1.60), size = 3, fontface = 2, hjust = 1,
y = c(spray_angle_adj_TruePos_1quartile, spray_angle_adj_TruePos_median,
spray_angle_adj_TruePos_3quartile),
label = c(spray_angle_adj_TruePos_1quartile,
spray_angle_adj_TruePos_median,
spray_angle_adj_TruePos_3quartile)))
```

True vs False Positives, by spray_angle_adj (Infield shift)



Here we can clearly see that more of the false positives are being hit to the pull side of the field, while the actual singles are being hit to the opposite side of the field. That makes sense against a shift.

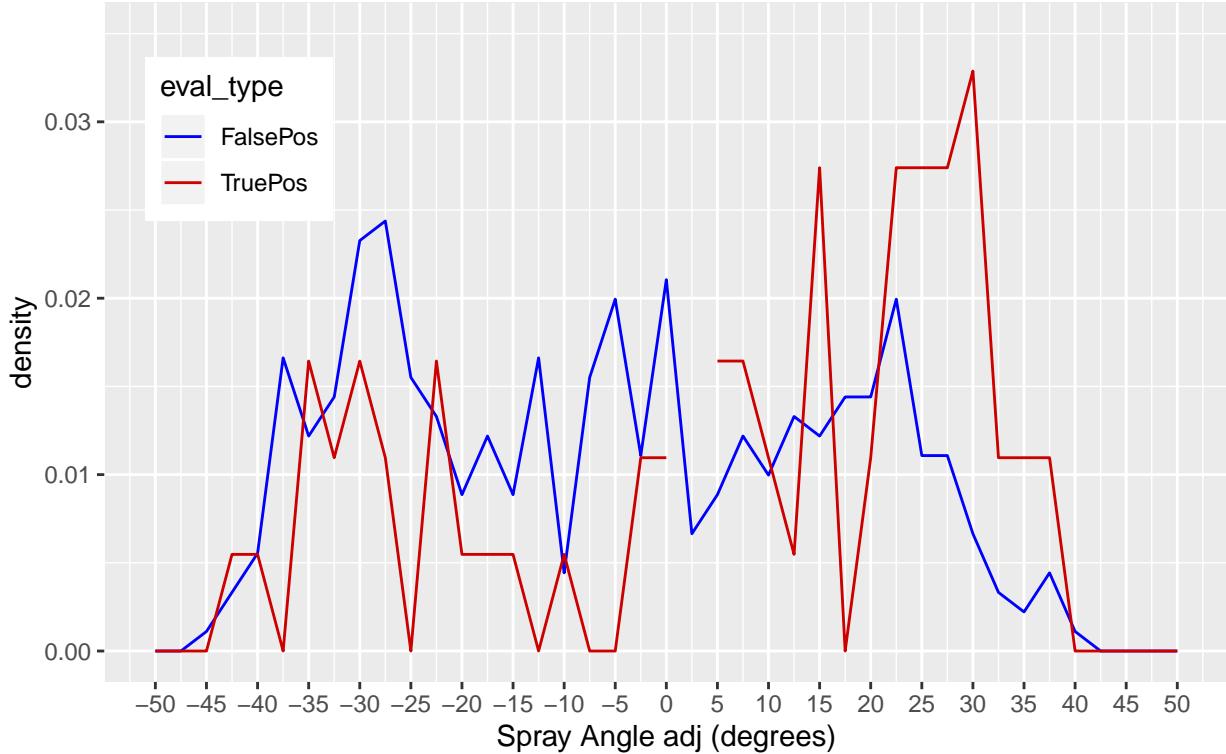
```
# Compare the spray_angle_adj densities of False Positives and True Positives.

predictions_6174_shift_gb <-
  dplyr::filter(predictions_6174,
    if_fielding_alignment == "Infield shift" &
      eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_shift_gb <- predictions_6174_shift_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
    adv_bb_type == "high_ground_ball")

x_scale <- scale_x_continuous(breaks = seq(-50, 50, by = 5),
  labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20",
    "-15", "-10", "-5", "0", "5", "10", "15", "20",
    "25", "30", "35", "40", "45", "50"),
  limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))

(sa_adj_density_pos <- ggplot(predictions_6174_shift_gb,
  aes(x = spray_angle_adj, y = stat(density),
    color = eval_type)) +
  geom_freqpoly(binwidth = 2.5) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(FalsePos = "blue", TruePos = "red3")) +
  labs(title = "Density Plots of spray_angle_adj by FalsePos and TruePos
  (If Shift)", x = "Spray Angle adj (degrees)")) +
  theme(legend.position = c(.12, .80))
```

Density Plots of spray_angle_adj by FalsePos and TruePos (If Shift)



The density plots confirm that the model is being fooled by balls pulled against a shifted infield. Balls that would normally be singles are being turned into outs.

Our last feature to be examined was hp_to_1b, and then we'll see if we can draw any conclusions about the types of batted balls the model is misclassifying as singles.

```
# Visualize false positive and true positive predictions by hp_to_1b_cat.
```

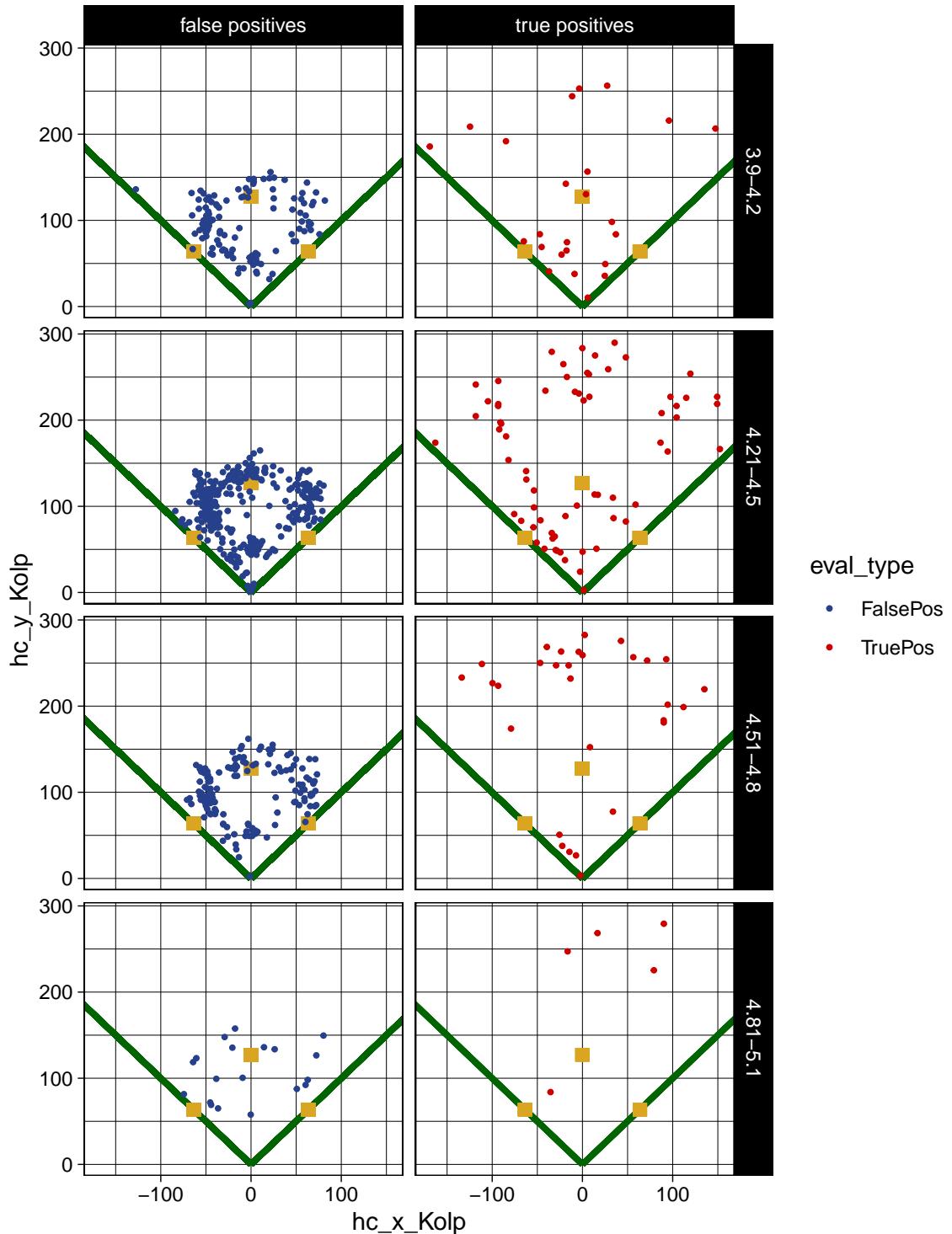
```
predictions_6174_lowgb_topweak <-
  dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_lowgb_topweak <- predictions_6174_lowgb_topweak %>%
  dplyr::filter(adv_bb_type == "low_ground_ball")
predictions_6174_lowgb_topweak <- predictions_6174_lowgb_topweak %>%
  dplyr::filter(launch_speed_angle == "1" |
    launch_speed_angle == "2")

pos_labels <- c("false positives", "true positives")
names(pos_labels) <- c("FalsePos", "TruePos")

(falsePos_predictions_by_hp_to_1b_cat <- ggplot(predictions_6174_lowgb_topweak,
                                                 aes(hc_x_Kolp, hc_y_Kolp,
                                                     color = eval_type)) +
  scale_color_manual(values = c(FalsePos = "royalblue4", TruePos = "red3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green",
               size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green",
               size = 1.3) +
```

```
geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = .7) + theme_linedraw() +
  labs(title = "False Positives by hp_to_1b_cat") +
  facet_grid(hp_to_1b_cat ~ eval_type, labeller = labeller(eval_type = pos_labels)))
```

False Positives by hp_to_1b_cat



Recall that the population for this feature is topped or weakly hit ground balls. As we have come to expect, the false positives are mostly balls that reached infielders in their traditional locations on the field. The singles were either balls that stopped short of the infielders, or ones that sneaked between infielders and rolled into the outfield. Did speed matter? It's difficult to judge the proportions from these batted ball plots. Let's see what the boxplots have to say.

```

# Using boxplots, compare false positive and true positive predictions by hp_to_1b.

predictions_6174_lowgb_topweak <-
  dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_lowgb_topweak <- predictions_6174_lowgb_topweak %>%
  dplyr::filter(adv_bb_type == "low_ground_ball")

pos_predictions_by_hp_to_1b <- ggplot(predictions_6174_lowgb_topweak,
                                         aes(x = eval_type, y = hp_to_1b,
                                              color = eval_type)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(3.90, 5.10, by = 0.1),
                     labels = c("3.90", "4.00", "4.10", "4.20", "4.30", "4.40", "4.50",
                               "4.60", "4.70", "4.80", "4.90", "5.00", "5.10"),
                     limits = c(3.90, 5.10)) +
  scale_color_manual(values = c(FalsePos = "Blue", TruePos = "Red")) +
  ggtitle("False vs True Positives, by hp_to_1b") +
  theme(legend.position = c(.90, .90))

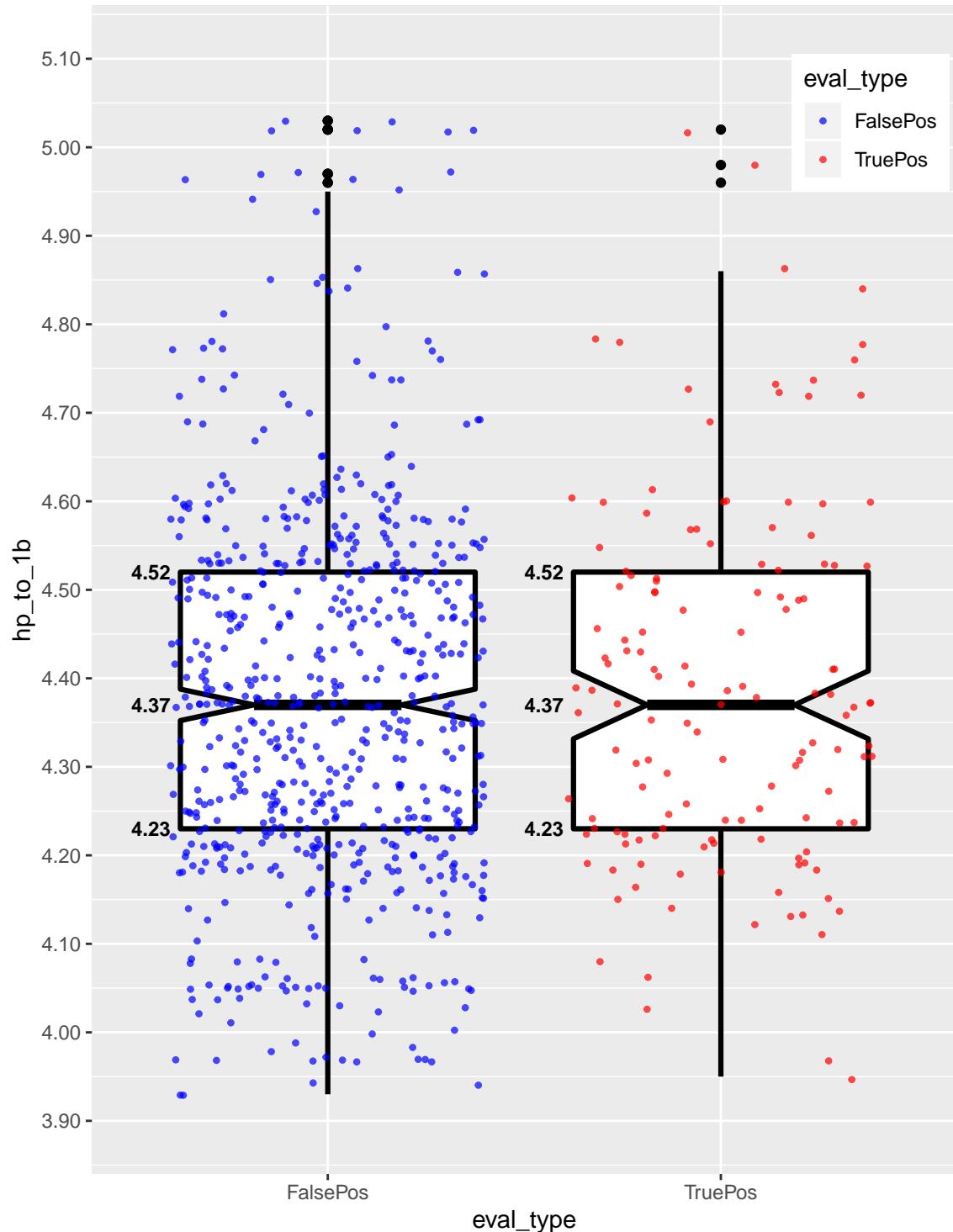
pos_predictions_by_hp_to_1b_data <- layer_data(pos_predictions_by_hp_to_1b)

hp_to_1b_FalsePos_1quartile <- round(pos_predictions_by_hp_to_1b_data[1, 2], 2)
hp_to_1b_FalsePos_median <- round(pos_predictions_by_hp_to_1b_data[1, 3], 2)
hp_to_1b_FalsePos_3quartile <- round(pos_predictions_by_hp_to_1b_data[1, 4], 2)
hp_to_1b_TruePos_1quartile <- round(pos_predictions_by_hp_to_1b_data[2, 2], 2)
hp_to_1b_TruePos_median <- round(pos_predictions_by_hp_to_1b_data[2, 3], 2)
hp_to_1b_TruePos_3quartile <- round(pos_predictions_by_hp_to_1b_data[2, 4], 2)

(pos_predictions_by_hp_to_1b <- ggplot(predictions_6174_lowgb_topweak,
                                         aes(x = eval_type, y = hp_to_1b,
                                              color = eval_type)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.85) +
  scale_y_continuous(breaks = seq(3.90, 5.10, by = 0.1),
                     labels = c("3.90", "4.00", "4.10", "4.20", "4.30", "4.40", "4.50",
                               "4.60", "4.70", "4.80", "4.90", "5.00", "5.10"),
                     limits = c(3.90, 5.10)) +
  scale_color_manual(values = c(FalsePos = "Blue", TruePos = "Red")) +
  ggtitle("False vs True Positives, by hp_to_1b") +
  theme(legend.position = c(.90, .90)) +
  annotate("text", x = c(.60, .60, .60), size = 3, fontface = 2, hjust = 1,
          y = c(hp_to_1b_FalsePos_1quartile, hp_to_1b_FalsePos_median,
                hp_to_1b_FalsePos_3quartile),
          label = c(hp_to_1b_FalsePos_1quartile, hp_to_1b_FalsePos_median,
                    hp_to_1b_FalsePos_3quartile)) +
  annotate("text", x = c(1.60, 1.60, 1.60), size = 3, fontface = 2, hjust = 1,
          y = c(hp_to_1b_TruePos_1quartile, hp_to_1b_TruePos_median,
                hp_to_1b_TruePos_3quartile),
          label = c(hp_to_1b_TruePos_1quartile, hp_to_1b_TruePos_median,
                    hp_to_1b_TruePos_3quartile)))

```

False vs True Positives, by hp_to_1b



The boxplots suggest false positives do not have different `hp_to_1b` values as compared to actual singles. Will the density plots show anything more?

```
# Compare the hp_to_1b densities of False Positives and True Positives.
```

```

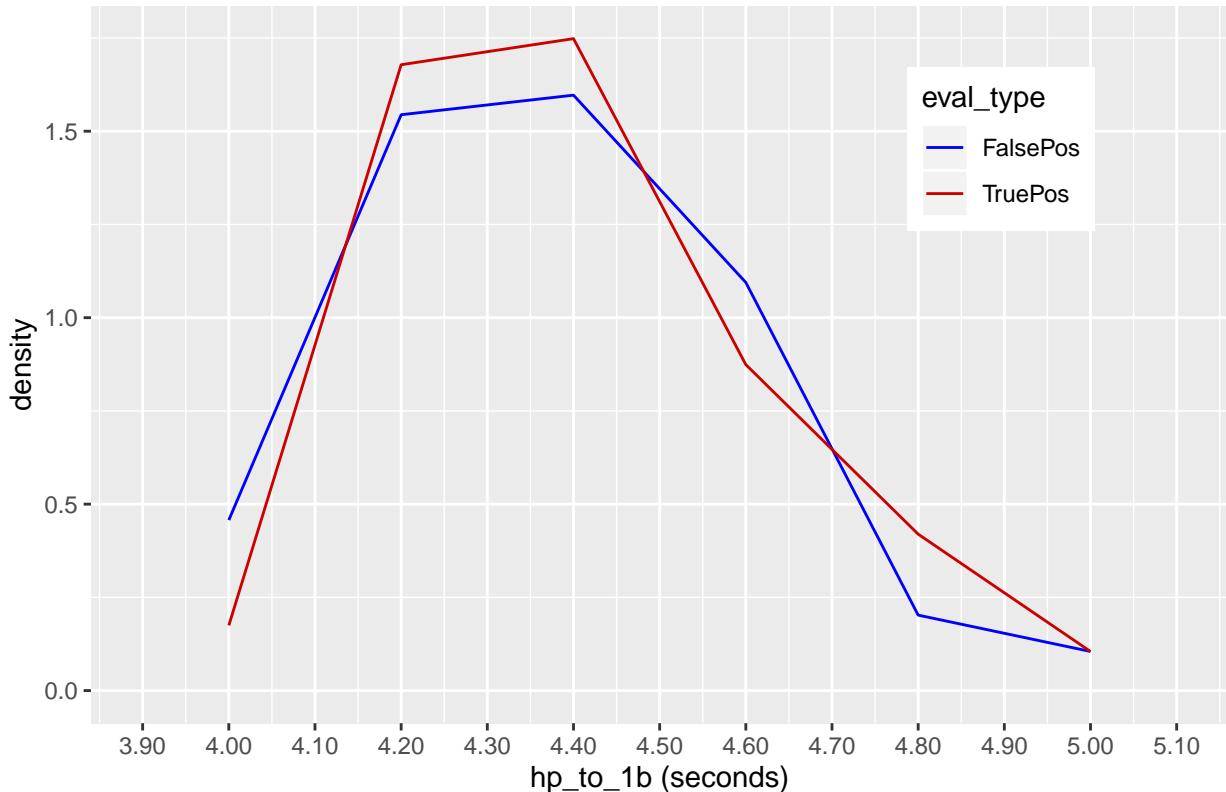
predictions_6174_lowgb_topweak <-
  dplyr::filter(predictions_6174, eval_type %in% c("FalsePos", "TruePos"))
predictions_6174_lowgb_topweak <- predictions_6174_lowgb_topweak %>%
  dplyr::filter(adv_bb_type == "low_ground_ball")

x_scale <- scale_x_continuous(breaks = seq(3.90, 5.10, by = 0.1),
                               labels = c("3.90", "4.00", "4.10", "4.20", "4.30", "4.40",
                                         "4.50", "4.60", "4.70", "4.80", "4.90",
                                         "5.00", "5.10"),
                               limits = c(3.90, 5.10))
y_scale <- scale_y_continuous()

(hp_to_1b_density_pos <- ggplot(predictions_6174_lowgb_topweak,
                                   aes(x = hp_to_1b, y = stat(density),
                                       color = eval_type)) +
  geom_freqpoly(binwidth = 0.2) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(FalsePos = "blue", TruePos = "red3")) +
  labs(title = "Density Plots of hp_to_1b by FalsePos and TruePos",
       x = "hp_to_1b (seconds)") +
  theme(legend.position = c(.80, .80)))

```

Density Plots of hp_to_1b by FalsePos and TruePos



Here we can see that false positives occur in greater proportions for the very fastest batters (under 4.1 seconds), as well as some below average runners (4.5 to 4.7 seconds). Our least impactful feature presents a muddled picture of our false positives.

To summarize our examination of false positives, we can say that the model tended to erroneously predict singles on ground balls hit fairly hard to the left side of the field, and on ground balls pulled into a shifted infield.

Yet, we wondered whether another analytical approach might provide a more granular characterization of false positives. It seemed that examining each feature of the model one at a time could potentially produce a composite picture that missed any batted balls other than those reflecting the very most common characteristics. We sought an approach that looked at all the features of each batted ball simultaneously, and we found it in clustering algorithms.

K-Means Clustering

Given the number of observations in our dataset of false positives, we needed to select a clustering algorithm that partitions the observations into multiple groups, repeatedly reshuffling the observations until the most cohesive groups possible are formed, based on a given criterion. Because one of our features, *if_fielding_alignment*, was categorical, we initially experimented with a technique known as partitioning around medoids (PAM), which accommodates categorical predictors. But the clusters it formed were not helpful, so we turned to another technique, K-means clustering.

False Positives with a Standard Infield Alignment

To make K-means clustering work for our data, we had to further subset the false positives by *if_fielding_alignment* before attempting to cluster them. Thus, the clustering process would only involve our five numeric features, and we would be clustering infield shift data separately from standard alignment data. We also scaled the data at the outset.

```
# Use K-means clustering to identify groups of batted balls that tend to be misclassified
# as singles or outs.

rf_predict_mistakes2 <- dplyr::select(rf_predict_mistakes, obs_nmbr,
                                         prob_single, prob_out, pred,
                                         obs, eval, eval_type, prob_diff,
                                         launch_angle, launch_speed, spray_angle_Kolp,
                                         spray_angle_adj, hp_to_1b,
                                         if_fielding_alignment, everything())

rf_predict_FalsePos_std <- filter(rf_predict_mistakes2, eval_type == "FalsePos",
                                    if_fielding_alignment == "Standard")
rf_predict_FalsePos_std_scaled <- scale(rf_predict_FalsePos_std[, 9:13])

rf_predict_FalsePos_shift <- filter(rf_predict_mistakes2, eval_type == "FalsePos",
                                    if_fielding_alignment == "Infield shift")
rf_predict_FalsePos_shift_scaled <- scale(rf_predict_FalsePos_shift[, 9:13])

rf_predict_FalseNeg_std <- filter(rf_predict_mistakes2, eval_type == "FalseNeg",
                                    if_fielding_alignment == "Standard")
rf_predict_FalseNeg_std_scaled <- scale(rf_predict_FalseNeg_std[, 9:13])

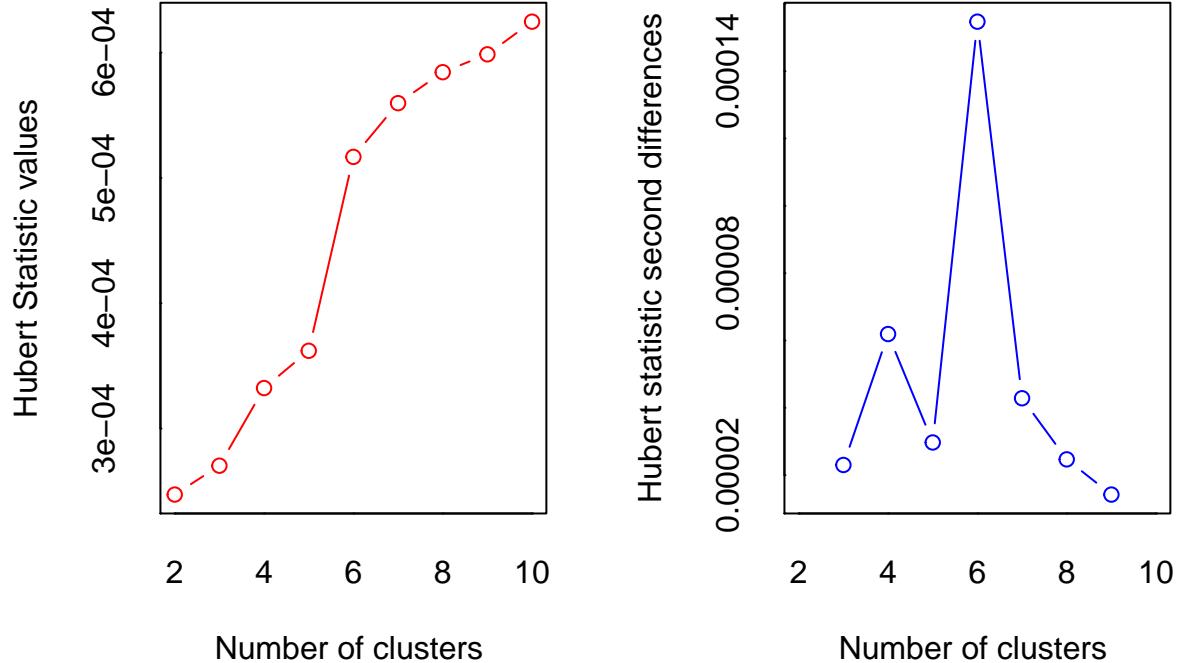
rf_predict_FalseNeg_shift <- filter(rf_predict_mistakes2, eval_type == "FalseNeg",
                                    if_fielding_alignment == "Infield shift")
rf_predict_FalseNeg_shift_scaled <- scale(rf_predict_FalseNeg_shift[, 9:13])
```

Our first task was to inform the K-means algorithm of the number of clusters to be created. To identify the optimal number of clusters to form from *rf_predict_FalsePos_std_scaled*, we used the NbClust package.

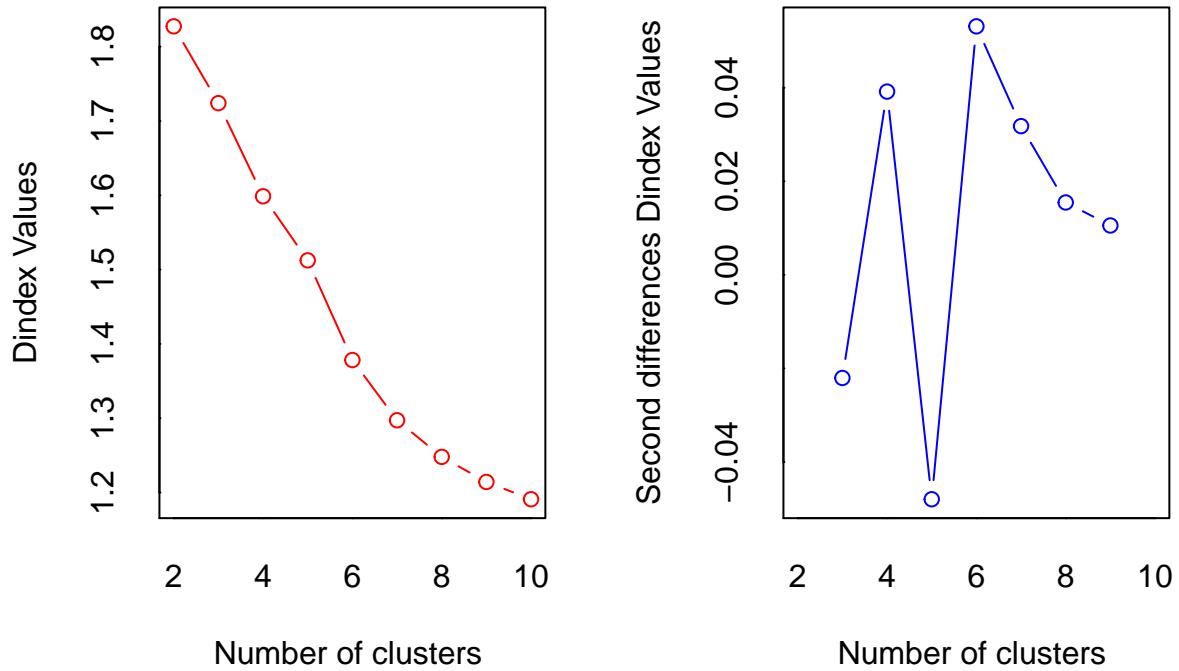
The NbClust function uses up to 30 indices to propose the optimal number of clusters based on the differing results obtained by varying the number of clusters.

```
# Use NbClust package to identify optimal number of clusters in
# rf_predict_FalsePos_std_scaled.

library(NbClust)
set.seed(1234)
nc <- NbClust(rf_predict_FalsePos_std_scaled, distance = "euclidean", min.nc = 2,
               max.nc = 10, method = "kmeans")
```



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
## In the plot of Hubert index, we seek a significant knee that corresponds to a
## significant increase of the value of the measure i.e the significant peak in Hubert
## index second differences plot.
```



```

## *** : The D index is a graphical method of determining the number of clusters.
## In the plot of D index, we seek a significant knee (the significant peak in Dindex
## second differences plot) that corresponds to a significant increase of the value of
## the measure.
##
## *****
## * Among all indices:
## * 4 proposed 2 as the best number of clusters
## * 2 proposed 3 as the best number of clusters
## * 2 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 6 proposed 6 as the best number of clusters
## * 3 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 3 proposed 10 as the best number of clusters
##
## ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 6
##
## *****

```

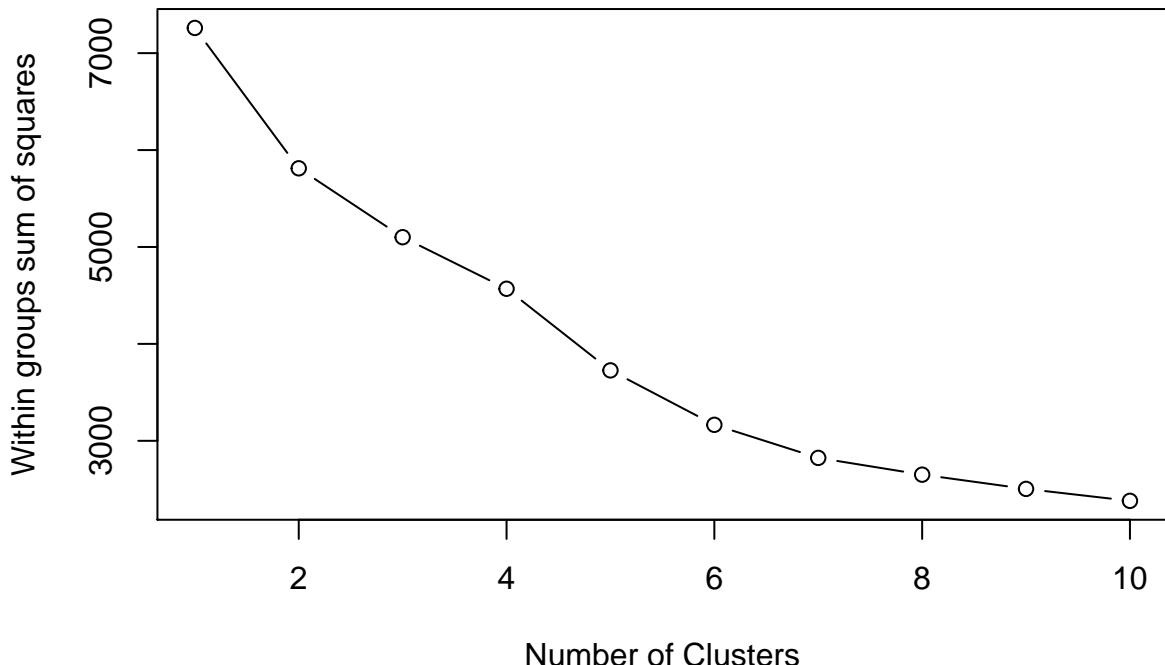
```
table(nc$Best.nc[1,])
```

```
##  
##  0   1   2   3   4   5   6   7   8   9  10  
##  2   1   4   2   2   1   6   3   1   1   3
```

The number of clusters most frequently selected as optimal by the NbClust indices was 6.

Alternatively, Kabacoff proposes plotting the total within-groups sums of squares against the number of clusters, using the following function:¹⁶

```
wssplot <- function(data, nc=15, seed=1234){  
  wss <- (nrow(data)-1)*sum(apply(data,2,var))  
  for (i in 2:nc){  
    set.seed(seed)  
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}  
  plot(1:nc, wss, type="b", xlab="Number of Clusters",  
    ylab="Within groups sum of squares")}  
  
wssplot(rf_predict_FalsePos_std_scaled, nc = 10, seed = 1234)
```



While the resulting plot could be interpreted as proposing 6 clusters, the tell-tale bend was somewhat ambiguous. We nevertheless decided to use K-means clustering to attempt to form 6 cohesive groups of batted balls in the `rf_predict_FalsePos_std_scaled` data frame.

¹⁶Kabacoff, Robert I. 2015. “*R in Action*”. 2nd ed. New York: Manning.

```

# Use K-means clustering to identify the types of batted balls that tend to be
# misclassified as singles against an infield in Standard alignment.

set.seed(1234)
km_fit <- kmeans(rf_predict_FalsePos_std_scaled, 6, iter.max = 10, nstart = 50)
round.aggregate(rf_predict_FalsePos_std[, 9:13], by = list(cluster = km_fit$cluster),
               median), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1       1          3.7      98.20      -25.55      -25.55
## 2       2         21.3      73.05       4.10      11.70
## 3       3          4.3      98.45      27.95     -27.95
## 4       4          5.7      97.40      -6.40      -7.40
## 5       5          2.7      97.40       7.70      21.30
## 6       6        -50.9      62.45     -20.40     -17.25
##   hp_to_1b
## 1 4.37
## 2 4.33
## 3 4.26
## 4 4.72
## 5 4.29
## 6 4.30

round.aggregate(rf_predict_FalsePos_std[, 9:13], by = list(cluster = km_fit$cluster),
               mad), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1       1          9.56      8.60       6.15      6.15
## 2       2         12.82      7.93      23.57     18.68
## 3       3          7.56      9.04       7.71      7.71
## 4       4         11.86      8.60      18.38     18.68
## 5       5         11.64      7.64      26.91     14.83
## 6       6         15.27     14.31      14.01     21.94
##   hp_to_1b
## 1 0.19
## 2 0.20
## 3 0.19
## 4 0.19
## 5 0.17
## 6 0.21

km_fit$size

## [1] 398 230 184 189 386 66

km_clusters <- fitted(km_fit, method = "classes")
rf_predict_FalsePos_std <- add_column(rf_predict_FalsePos_std, km_clusters,
                                         .after = "obs_nmb")

# Create a tibble for each cluster of false positives against an infield in
# Standard alignment.

```

```

FalsePos_std_cluster1 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 1)
FalsePos_std_cluster2 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 2)
FalsePos_std_cluster3 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 3)
FalsePos_std_cluster4 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 4)
FalsePos_std_cluster5 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 5)
FalsePos_std_cluster6 <- rf_predict_FalsePos_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 6)

```

A quick look at the medians and median absolute deviations of our five numeric features shows a fair amount of variation between clusters. And we can see that the clusters vary in size from 66 to 398.

First, we generated batted ball plots for the smaller subset of false positives we were now using.

```

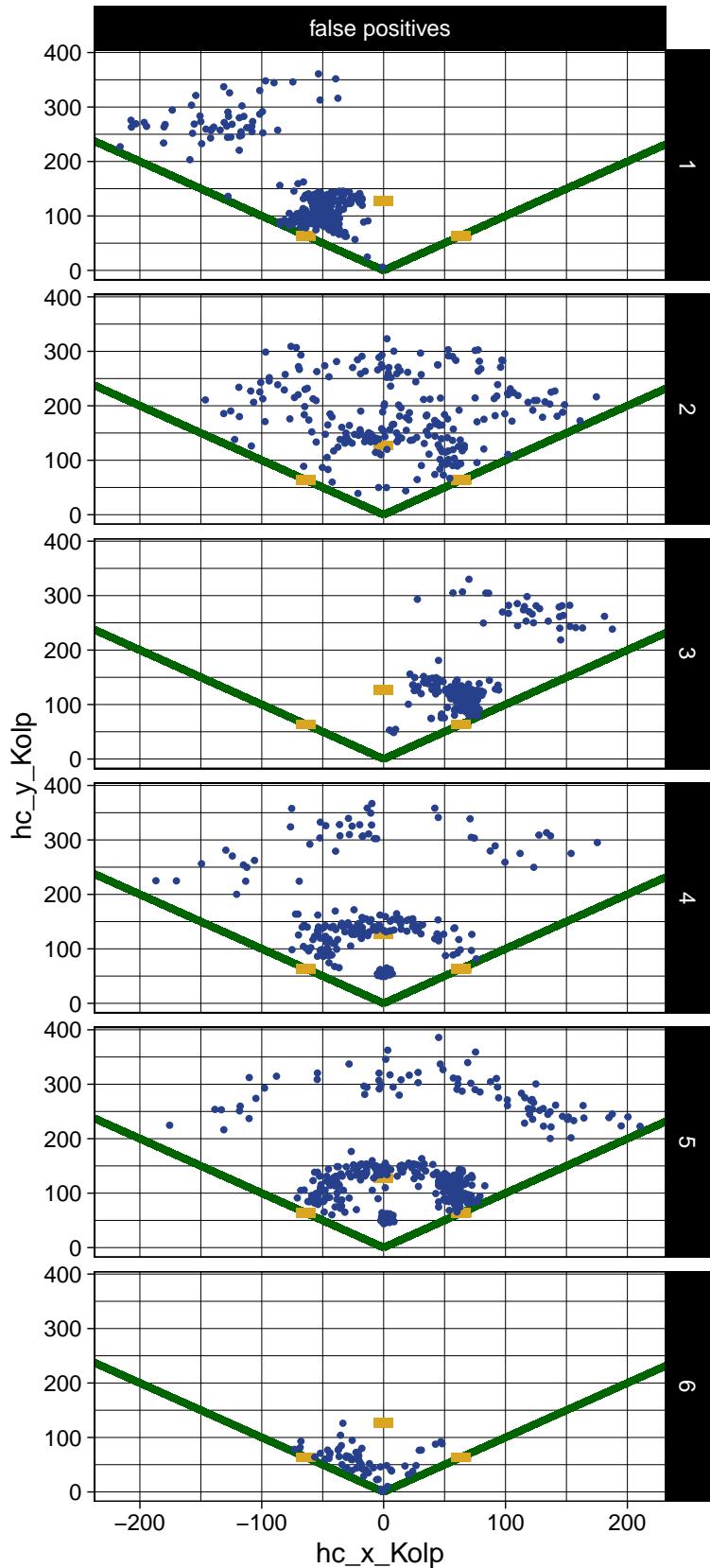
# Visualize false positive predictions by cluster, when the infield was in a
# Standard alignment.

pos_labels <- c("false positives")
names(pos_labels) <- c("FalsePos")

ggplot(rf_predict_FalsePos_std, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green",
               size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green",
               size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.7, color = "royalblue4") + theme_linedraw() +
  labs(title = "FalsePos_std Preds by Cluster") +
  facet_grid(km_clusters ~ eval_type, labeller = labeller(eval_type = pos_labels))

```

FalsePos_std Preds by Cluster



We found that the most helpful way to describe each cluster was by first determining how many times each combination of features was represented by a batted ball in the cluster. The feature combinations (or feature sets) most frequently represented in the cluster told us a lot about the groups of batted balls captured by the K-means clustering algorithm. We'll describe the results one cluster at a time.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 1. (Source of code in the first statement
# below was from Cath on stackoverflow, Dec 16, 2015,
# https://stackoverflow.com/questions/34312324/r-count-all-combinations)

FalsePos_std_cluster1_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster1))~.,
                      FalsePos_std_cluster1, length))
FalsePos_std_cluster1_combs <- FalsePos_std_cluster1_combs %>% arrange(desc(n))
FalsePos_std_cluster1_combs <- mutate(FalsePos_std_cluster1_combs,
                                       n_pct = round(n/sum(n)*100, 2),
                                       n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_std_cluster1_combs <-
  add_column(FalsePos_std_cluster1_combs,
             comb_rank = min_rank(desc(FalsePos_std_cluster1_combs$n)),
             .before = "adv_bb_type")
```

The first cluster of false positives against a standard infield alignment comprises 27% of the observations in *rf_predict_FalsePos_std*, and is the largest of our six clusters. Its 398 batted balls represented 125 distinct combinations of features. Of the 29 most common feature sets (comprising 60% of the 398 total batted balls), 16 were high ground balls and nine were low ground balls. Twelve of the 29 feature sets had exit velocities exceeding 100 miles per hour, and 13 were between 90 and 100 miles per hour. Every one of the 29 feature sets were balls pulled to the left side of the infield (either toward the shortstop, the 5-6 hole, or the third baseman) by right-handed hitters with average or below average speed. So, with regard to Cluster 1, we can think of it as solidly hit ground balls pulled to the left side.

Summary statistics may have allowed us to form the same description of Cluster 1 as above. But the nice thing about viewing each combination of features (feature set) in the cluster is that it allows us to see at a glance the individual features comprising each combination. In the case of Cluster 1, for example, we would like to know whether the model made any egregious mistakes in predicting a single when the batted ball actually resulted in an out. The combination least likely to result in a single was a low ground ball hit between 80 and 90 miles per hour and pulled toward the shortstop or third baseman by a right-handed batter with an average or slower time to first base. Not a single batted ball in the entire cluster matched this feature set. In fact, there were only nine low ground balls with launch speeds between 80 and 90; all but one of them were hit toward gaps in the infield, and the one exception was hit by a batter who could reach first base in 4.2 seconds.

The launch speed category for balls hit between 90 and 100 miles per hour could be viewed as single-neutral. The lower 90s are considered less friendly to singles, while the upper 90s are considered single-friendly. But even when we included this category in our search for egregious false positives, there were only 20 of these that were low ground balls hit at the shortstop or third baseman. Eight of the 20 had launch angles within 5 degrees of being considered a high ground ball. Thirteen had exit velocities in the upper 90s. And 17 were within two degrees of an infield hole. According to our model, the probability that these balls would result in single was 0.4 or less for 15 of the 20 potentially egregious false positives.

As a common sense eye check, we were able to review video for all but two of these 20 false positives. The video confirmed the balls were solidly hit, but only a few involved better than average defensive plays. The vast majority were fairly routine plays, involving balls hit directly toward the shortstop or third baseman. In other words, they didn't look anything like possible singles, and it didn't seem like they deserved to be assigned a greater than 30% chance of being a single. One explanation is that, with a reduced decision

threshold of 0.30, the model was bound to scoop up some routine plays for the singles bucket. The fact that we only found 20 of these egregious false positives out of the 398 in Cluster 1 (about 5%) probably speaks well for the model's performance, given the data it had to work with.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 3.

FalsePos_std_cluster3_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster3))~.,
                      FalsePos_std_cluster3, length))
FalsePos_std_cluster3_combs <- FalsePos_std_cluster3_combs %>% arrange(desc(n))
FalsePos_std_cluster3_combs <- mutate(FalsePos_std_cluster3_combs,
                                       n_pct = round(n/sum(n)*100, 2),
                                       n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_std_cluster3_combs <-
  add_column(FalsePos_std_cluster3_combs,
             comb_rank = min_rank(desc(FalsePos_std_cluster3_combs$n)),
             .before = "adv_bb_type")
```

The next cluster we'll describe is Cluster 3, because it's very similar to Cluster 1, except that it's for left-handed batters. The third cluster contains 184 batted balls, comprising 13% of the false positives against a standard infield alignment. The 184 batted balls formed 64 distinct combinations of features. The 23 most common feature sets represented 72% of the batted balls in Cluster 3. Of these 23 feature sets, 17 are populated by ground balls, particularly high ground balls. Exit velocities are split about equally between the 90-100 mph category and the above 100 mph category. Ten have spray angles toward the 3-4 hole, and 13 have spray angles toward the second baseman or first baseman. Ten of the feature sets include batters with the quickest times to first base (3.9 to 4.2 seconds). To sum up, Cluster 3 is about solidly hit ground balls pulled to the right side of the infield by left-handed hitters with average or above average speed to first base.

As with Cluster 1, we wanted to know whether the model made any egregious mistakes in predicting a single when the batted ball actually resulted in an out. The combination least likely to result in a single was a low ground ball hit between 80 and 90 miles per hour and pulled toward the second baseman or first baseman by a left-handed batter with an average or slower time to first base. But there were no such batted balls in Cluster 3. There were only three low ground balls with launch speeds between 80 and 90, and all were hit toward the 3-4 hole, one by a batter who could reach first base in 4.1 seconds.

As we continued to search for egregious false positives in Cluster 3, we included balls hit between 90 and 100 miles per hour (a single-neutral category). There were 12 such low ground balls hit toward the second baseman or the first baseman. We were able to review video for 9 of them, and it showed three of the the ground balls requiring good defensive plays to get the batters out. The other six were fairly routine grounders to the second baseman or the first baseman. So if we say nine of the 12 were badly mistaken predictions by the model, that amounts to 5% of all the batted balls in Cluster 3. You'll recall that Cluster 1 had a similar rate of egregious false positives.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 2.

FalsePos_std_cluster2_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster2))~.,
                      FalsePos_std_cluster2, length))
FalsePos_std_cluster2_combs <- FalsePos_std_cluster2_combs %>% arrange(desc(n))
FalsePos_std_cluster2_combs <- mutate(FalsePos_std_cluster2_combs,
                                       n_pct = round(n/sum(n)*100, 2),
                                       n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
```

```

FalsePos_std_cluster2_combs <-
  add_column(FalsePos_std_cluster2_combs,
             comb_rank = min_rank(desc(FalsePos_std_cluster2_combs$n)),
             .before = "adv_bb_type")

```

The second cluster consists of 230 observations comprising 16% of the data. The 230 batted balls formed 138 distinct combinations of features, so this is a fairly diverse group of batted balls compared to Clusters 1 and 3. The 50 most common feature sets represent 62% of the batted balls in Cluster 2. Of these 50 feature sets, 40 are populated by line drives and fly balls. Exit velocities range from below 70 mph to 89 mph, so these aren't particularly well hit balls. Forty-five of the 50 feature sets contain balls hit toward the middle part of the field, toward (or over the heads of) the pitcher, second baseman or shortstop. About half of the feature sets have balls being hit to the opposite side of the field, and only seven include balls hit toward the middle infielder on the pull side. Nine of the feature sets include batters with the quickest times to first base (3.9 to 4.2 seconds), and nine include batters with below average times to first base (4.5 to 4.8 seconds). Most have average times to first base.

Cluster 2, then, is about softly hit liners and flies toward the middle or opposite side of the field by both left-handed and right-handed batters with average speed to first base. When we looked at additional information we had about the individual batted balls in this cluster (*rf_predict_FalsePos_std*), it became apparent we were dealing largely with a cluster of flares (balls hit into the shallow part of the outfield, beyond the typical locations of infielders and in front of the typical locations of outfielders). Cluster 2's flares never hit the ground because they were caught by infielders racing back or outfielders charging in.

Are there egregious singles predictions in Cluster 2? The types of batted balls in this cluster that are least likely to result in singles are fly balls in the 80 to 89 miles per hour category. No such batted ball exists in Cluster 2. If we expand our search to include fly balls hit less than 70 miles per hour, there are 25 such batted balls, and they're being caught mostly by middle infielders (180 to 210 feet from home plate), but also by corner outfielders (210 to 240 feet from home plate). After reviewing the video, we could characterize most of them as nice running plays, with a few routine plays and a few difficult plays mixed in. It would be harsh to consider the false positives in Cluster 2 egregious mistakes. Fielders are having to run down these balls to get the out. There isn't much separating them from singles.

```

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 4.

```

```

FalsePos_std_cluster4_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster4))~.,
                       FalsePos_std_cluster4, length))
FalsePos_std_cluster4_combs <- FalsePos_std_cluster4_combs %>% arrange(desc(n))
FalsePos_std_cluster4_combs <- mutate(FalsePos_std_cluster4_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_std_cluster4_combs <-
  add_column(FalsePos_std_cluster4_combs,
             comb_rank = min_rank(desc(FalsePos_std_cluster4_combs$n)),
             .before = "adv_bb_type")

```

Nearly as diverse as Cluster 2, the fourth cluster has 106 distinct feature sets for just 189 batted balls (13% of the false positives against a standard infield alignment). The 37 most frequent combinations of features include 63% of the batted balls in the cluster. With Cluster 4, we're back to ground balls, with some low line drives mixed in. They're solidly hit, about evenly split between the 90 to 100 mph category and the above 100 mph category. But these balls are hit up the middle and pulled toward the middle infielder by both left-handed and right-handed batters with below average speed to first base. About three-fourths of these outs involved the pitcher, middle infielders, or center fielder. Comparing Cluster 4 to the other clusters

we've examined so far, it's most similar to Clusters 1 and 3 except Cluster 4's batted balls are hit more toward the middle of the field by slower batters.

We've been thinking about home-to-first times mattering only on topped or weakly hit grounders, but it's worth considering whether a batter's lack of speed allows an infielder to position himself deeper and thus able to reach more grounders than would otherwise be possible. Plus, an infielder's bobble can still result in an out when there's a slower runner to first. So there are sound reasons to believe these batters' below average home-to-first times had some effect, however small, on these balls becoming outs rather than singles. Whether the model took such an effect into account when making its predictions is difficult to know.

We do know that the middle infield hole in the vicinity of second base is the best infield gap for singles, and there are a significant number of batted balls in Cluster 4 that were hit in excess of 100 mph. So we ought to be seeing a lot balls in Cluster 4 that look like legitimate singles, even if they turned out to be outs. But were there other balls in Cluster 4 that looked liked they had no chance of being a single? Those least likely to result in singles were low ground balls hit less than 90 miles per hour toward a middle infielder. There were four low grounders hit less than 90 mph, but they were hit up the middle, two being stopped by the pitcher, and two being stopped by the second baseman.

When we expanded our search for egregious mistakes in Cluster 4 by including balls hit between 90 and 100 mph, we found eight batted balls that met our criteria. Video showed two of them to be routine plays. The rest could be considered nice plays or even difficult defensive plays. So overall, the model doesn't appear to be making many indefensible decisions with regard to the fourth cluster.

We moved on to Cluster 5.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 5.

FalsePos_std_cluster5_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster5))~.,
    FalsePos_std_cluster5, length))
FalsePos_std_cluster5_combs <- FalsePos_std_cluster5_combs %>% arrange(desc(n))
FalsePos_std_cluster5_combs <- mutate(FalsePos_std_cluster5_combs,
  n_pct = round(n/sum(n)*100, 2),
  n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_std_cluster5_combs <-
  add_column(FalsePos_std_cluster5_combs,
    comb_rank = min_rank(desc(FalsePos_std_cluster5_combs$n)),
    .before = "adv_bb_type")
```

The fifth cluster, with 386 batted balls, is our second largest cluster, representing 27% of false positives against a standard infield alignment. The 386 balls are contained within 148 distinct feature sets. The 43 most common feature sets contain 65% of the data in Cluster 5. Here we have another cluster of ground balls, with some low line drives mixed in. Thirty-five of the 43 most common combinations contain primarily balls in the 90 to 100 mph category and the above 100 mph category. These balls are hit up the middle or to the opposite side by both left-handed and right-handed batters with mostly average speed to first base. Three-hundred and six of the 386 batted balls in the cluster were fielded by infielders, including pitchers. Cluster 5, then, consists mainly of solidly hit grounders to the opposite side (and, to a lesser extent, up the middle) by both left-handed and right-handed batters. It's our first ground ball cluster that includes balls hit the opposite way.

If we're looking for some especially poor predictions in Cluster 5, we're once again looking for low ground balls hit between 80 and 90 miles per hour toward an infielder. We found three of them. One was an 84.6 mph grounder hit directly at the first baseman, so it was a routine play. Another one had an incorrect spray angle (Kolp) of -17.6 degrees in our dataset. The ball was hit just to the left of second base, and probably had a spray angle around -5 degrees. The shortstop had already positioned himself at that location, so the

infield alignment should have been “Strategic” rather than “Standard”. When the ball came off the bat, it looked like a single up the middle. The third potentially bad prediction involved a second baseman who had to range quite a bit to his left to field the ball. So of the three low grounders hit between 80 and 90 mph at an infielder, only one was routine enough to say it had no realistic chance of resulting in a single.

When we expanded our search for egregious false positives in Cluster 5 to include balls hit between 70 and 80 mph, we found two more candidates. One was a routine play by the first baseman, but the other required the third baseman to take two large steps to his left before throwing to second base for a force out. Had there been no opportunity for a force out at second, it’s doubtful the batter, a fast runner, would have been out at first base. There were 36 low ground balls hit between 90 and 100 mph toward an infielder. No doubt we would have found another handful of routine plays among them, but overall, the model appears to be making reasonable decisions with the data it has.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 6.

FalsePos_std_cluster6_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_std_cluster6))~.,
    FalsePos_std_cluster6, length))
FalsePos_std_cluster6_combs <- FalsePos_std_cluster6_combs %>% arrange(desc(n))
FalsePos_std_cluster6_combs <- mutate(FalsePos_std_cluster6_combs,
  n_pct = round(n/sum(n)*100, 2),
  n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_std_cluster6_combs <-
  add_column(FalsePos_std_cluster6_combs,
    comb_rank = min_rank(desc(FalsePos_std_cluster6_combs$n)),
    .before = "adv_bb_type")
```

The sixth cluster, by far our smallest at 66 observations (less than 5% of our data), is the most clearly defined of all our clusters. Still, the 66 batted balls appear in 42 distinct combinations of features. They are all topped and weakly hit low ground balls fielded primarily on the left side of the infield in front of the pitcher or third baseman. They’re prototypical infield hits, except that in this case, they were turned into outs.

To see how they were turned into outs, we reviewed some video. There was no obvious way to narrow our search for easy outs. They were no more likely to have been made by third baseman than by pitchers. Pitchers have to move to their right away from first base, and then stop and pivot before throwing back to their left to get the batter out at first base. Third basemen typically charge forward to the ball and flip to first base in one motion. That’s not an easy play either, but it’s one that a major league third baseman is trained and expected to make. So we reviewed video on balls hit by slow-footed batters and fielded by either the third baseman or the pitcher.

Cluster 6 had eight batted balls fielded by the third baseman and hit by batters with below average speed to first base (hp_to_1b values greater than 4.5 seconds). One of the eight was a routine play, two were difficult plays, and the rest were nice defensive plays (somewhere between routine and difficult). Even against the slowest batters, the third baseman’s execution has to be flawless to get the out. While charging the ball, he has to field the ball cleanly and, while on the run, make an accurate throw to the inside of first base to avoid hitting the batter running down the baseline. It’s hard to blame the model for misclassifying these kinds of plays.

A quick look at video for plays made by the pitcher revealed a handful of additional routine plays, usually when the pitcher didn’t have to move very far to field the ball. Usually, the pitcher did have to move a fair distance, and it was generally a surprise to see the pitcher execute his movements quickly and accurately enough to nab the runner at first base. So our model seemed almost as blameless for these kinds of false positives.

What conclusions could we draw from our six clusters of false positives against a standard infield alignment? Before we employed K-means clustering, we concluded that our top-performing random forest model tended to erroneously predict singles on ground balls hit fairly hard (at least 90 miles per hour) to the left side of the field against a standard infield alignment. K-means clustering corroborated this finding, and expanded on it.

Cluster 1 and, to a large extent, Cluster 4, and to a lesser extent, Cluster 5 are all populated with those very types of batted balls. Harder hit grounders to the left side are probably the most common batted balls among all the false positives, accounting for 426 of the 1,453 false positives against a standard alignment. But clustering painted a much more complex picture of erroneously predicted singles. Let's start from the premise, borne out by K-means clustering, that many different types of batted balls comprise the false positives against a standard infield alignment. From that premise, let's try to make some valid generalizations:

- 1) There were two truly unique clusters. Cluster 2 was the only one consisting primarily of balls hit in the air. Clusters 2 and 6 were the only clusters featuring softly hit balls.

Cluster 6 (5% of the false positives) contained topped and weakly hit low ground balls fielded on the left side of the infield by the third baseman or the pitcher. Think of these as swinging bunts or attempted bunts. It's easy to see why any model would struggle to predict single or out on these types of plays. But they account for relatively few of the false positives.

The other unique cluster was Cluster 2 (16% of the false positives), which contained softly hit line drives and fly balls hit toward the middle or opposite side of the field, toward or over the head of the pitcher, second baseman, or shortstop. Think of these as flares heading toward the shallow parts of the outfield with an infilder running out and an outfielder charging in trying to catch the ball. As with the batted balls in Cluster 6, the difference between a single and an out on these plays is often a matter of inches.

- 2) The remaining false positives (nearly 80% of all false positives) consisted primarily of solidly hit ground balls, divided up into four clusters depending on whether they were pulled or hit the opposite way, and whether they were hit to the left side, right side, or up the middle. Ultimately, it didn't seem to matter much whether it was a low ground ball or a high ground ball, hit by a left-handed or right-handed batter, hit toward an infilder or toward a gap, pulled or oppo, to the left, middle, or right. So long as the ball was hit at least 90 miles per hour on the ground, the model made a disproportionate number of mistakes by essentially predicting these balls were going to make it through the infield and into the outfield.

If we had to name one feature that was clearly overrepresented among the false positives against a standard infield, it would be launch angles of 13 degree or less, basically, ground balls and the lowest of the low line drives. And with ground balls, it really does come down to whether it makes it through the infield or is stopped. At the major league level, if the ball is within reach of an infilder, the ball will almost always be stopped. And at the major league level, if the ball is stopped, the batter will almost always be thrown out at first base. So singles are made between the infilders.

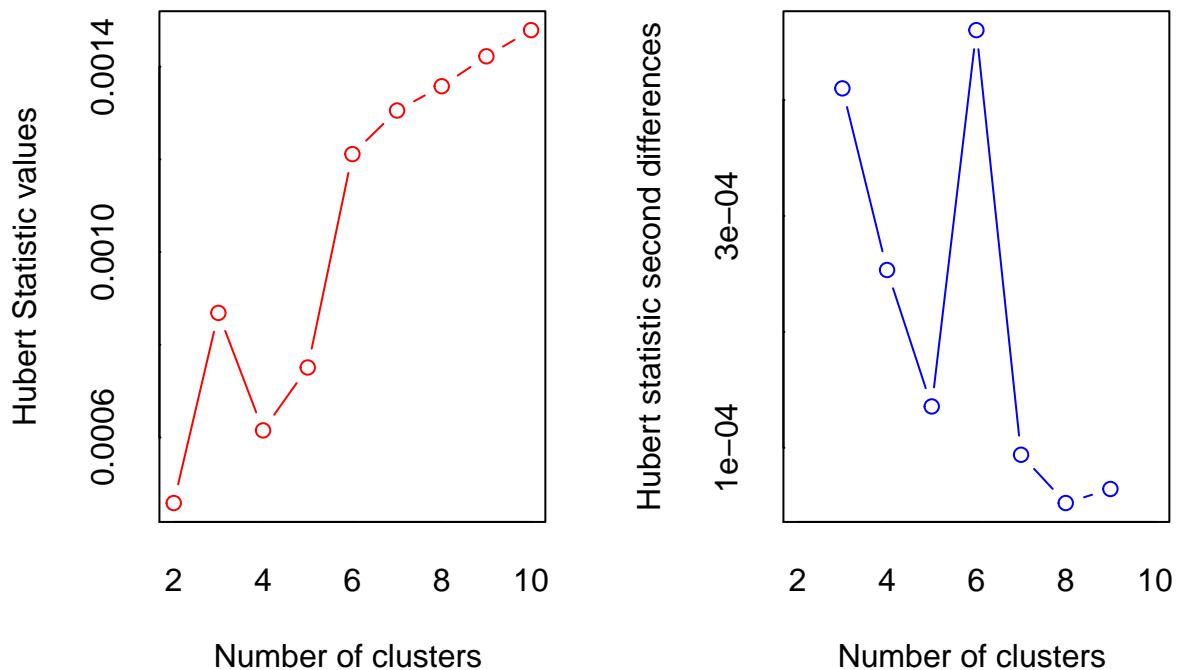
For a model to reliably predict whether a ground ball will pass in between infilders, it has to know, at a minimum, the location of the infilders when the ball hits the bat, and the velocity of the ball as it travels through the infield. The velocity of the ball as it travels through the infield can be reasonably inferred from its velocity as it comes off the bat. But the location of the infilders, that's the tricky part. Infilders move around a lot these days. Data indicating whether the infilders are in a standard, strategic, or shift alignment is certainly better than nothing. After all, our model estimated the combined significance of *spray_angle_Kolp* and *spray_angle_adj* to be 25 on a scale of 1 to 100. We can only presume these features carried more meaning as a result of the infield alignment data. But we still believe the performance of our model could have been significantly improved with higher quality data on the location of the infilders.

False Positives with an Infield Shift Alignment

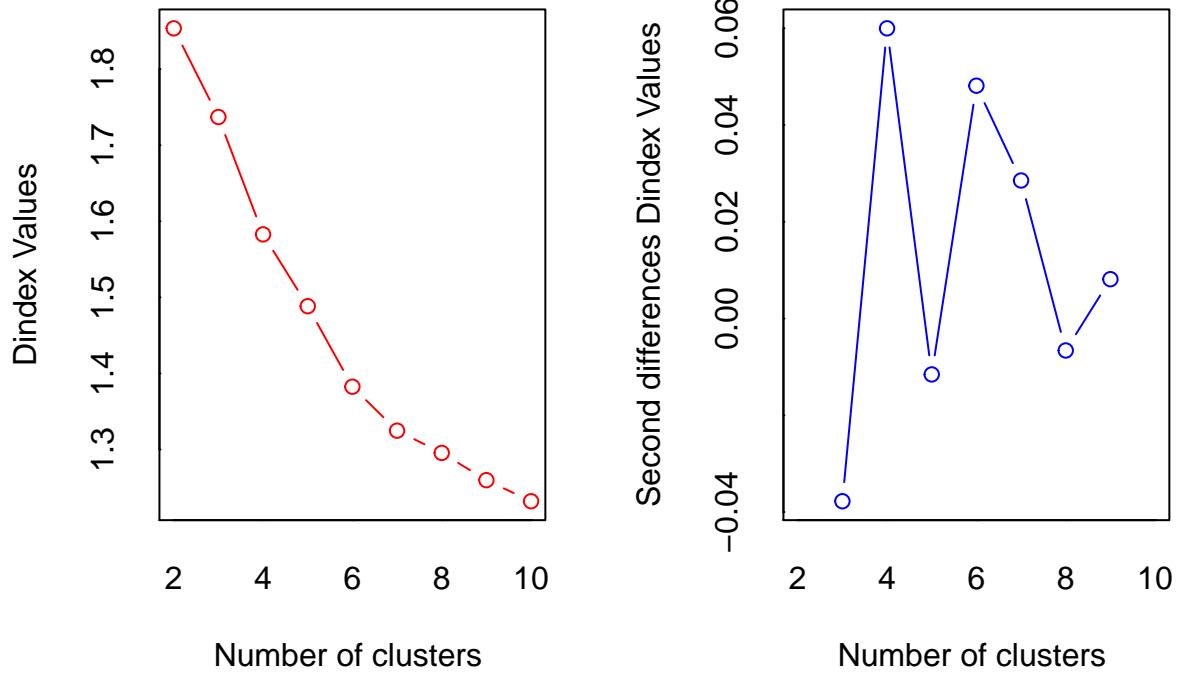
The prediction error rate when the infield was shifted was 0.156, a bit higher than the 0.150 error rate for a standard infield alignment. In our test set, 786 of the 5,045 batted balls against a shift were incorrectly predicted by the model. And 611 of these 786 prediction errors (78%) were false positives. What did these 611 false positives against the shift look like? We again turned to K-means clustering to help describe the most common sets of features.

```
# Use NbClust package to identify optimal number of clusters in
# rf_predict_FalsePos_shift_scaled.

library(NbClust)
set.seed(1234)
nc <- NbClust(rf_predict_FalsePos_shift_scaled, distance = "euclidean", min.nc = 2,
               max.nc = 10, method = "kmeans")
```



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
## In the plot of Hubert index, we seek a significant knee that corresponds to a
## significant increase of the value of the measure i.e the significant peak in Hubert
## index second differences plot.
##
```



```

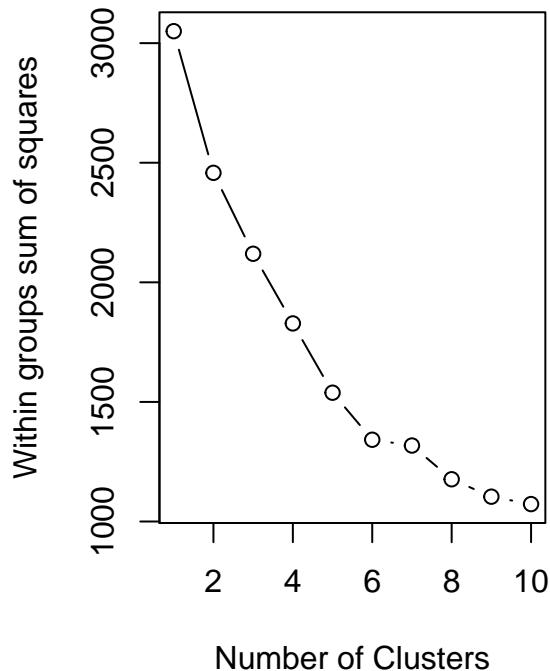
## *** : The D index is a graphical method of determining the number of clusters.
## In the plot of D index, we seek a significant knee (the significant peak in Dindex
## second differences plot) that corresponds to a significant increase of the value of
## the measure.
##
## *****
## * Among all indices:
## * 4 proposed 2 as the best number of clusters
## * 2 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 11 proposed 6 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
##
## ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 6
##
## *****

```

```



```



NbClust and wssplot both identified 6 as the optimal number of clusters for false positives against a shift.

```

# Use K-means clustering to identify the types of batted balls that tend to be
# misclassified as singles against a shifted infield.

```

```

set.seed(1234)
km_fit <- kmeans(rf_predict_FalsePos_shift_scaled, 6, iter.max = 10, nstart = 50)
round(aggregate(rf_predict_FalsePos_shift[, 9:13], by = list(cluster = km_fit$cluster),
               median), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1         1          0.85       97.2      -5.65      10.85
## 2         2         -38.90       58.5     -17.90      21.60
## 3         3          7.60       99.8     -27.95     -27.95
## 4         4          9.10       99.3      26.50     -26.50
## 5         5         20.30       72.0     -7.70      11.90
## 6         6          9.65       98.0      4.30     -7.35
##   hp_to_1b
## 1    4.35
## 2    4.28
## 3    4.45
## 4    4.34
## 5    4.39
## 6    4.79

round(aggregate(rf_predict_FalsePos_shift[, 9:13], by = list(cluster = km_fit$cluster),
               mad), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1         1          9.86       9.27      15.72      16.90
## 2         2         20.76      16.75      16.16      9.34
## 3         3          6.67       8.52       7.49      7.49
## 4         4          8.60       8.75      10.53     10.53
## 5         5         10.82       8.60      20.02     20.16
## 6         6          8.97       9.56      19.72     17.72
##   hp_to_1b
## 1    0.16
## 2    0.15
## 3    0.19
## 4    0.16
## 5    0.16
## 6    0.17

km_fit$size

## [1] 176 27 88 141 89 90

km_clusters <- fitted(km_fit, method = "classes")
rf_predict_FalsePos_shift <- add_column(rf_predict_FalsePos_shift, km_clusters,
                                         .after = "obs_nmbr")

```

Next, we wanted to plot the batted balls against a shift if they resulted in false positive predictions.

```
# Create a tibble for each cluster of false positives against a fully
# shifted infield.
```

```

FalsePos_shift_cluster1 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 1)
FalsePos_shift_cluster2 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 2)
FalsePos_shift_cluster3 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 3)
FalsePos_shift_cluster4 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 4)
FalsePos_shift_cluster5 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 5)
FalsePos_shift_cluster6 <- rf_predict_FalsePos_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 6)

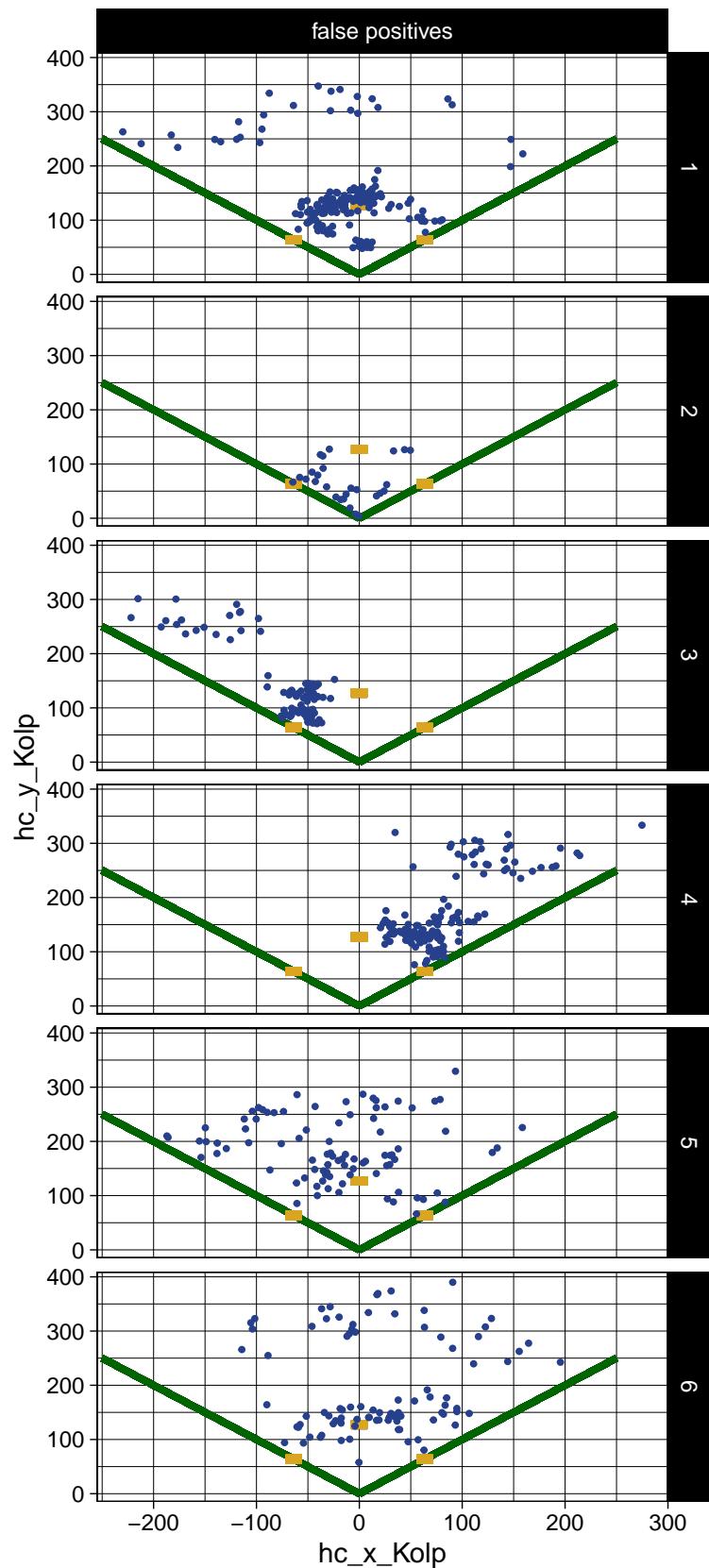
# Visualize false positive predictions by cluster, when the infield was fully
# shifted.

pos_labels <- c("false positives")
names(pos_labels) <- c("FalsePos")

ggplot(rf_predict_FalsePos_shift, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.7, color = "royalblue4") + theme_linedraw() +
  labs(title = "FalsePos_shift Preds by Cluster") +
  facet_grid(km_clusters ~ eval_type, labeller = labeller(eval_type = pos_labels))

```

FalsePos_shift Preds by Cluster



Before drawing any premature conclusions based solely on the batted ball plots, we wanted to see what we could learn by examining the most common combinations of features in each cluster.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 1.

FalsePos_shift_cluster1_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster1))~.,
                      FalsePos_shift_cluster1, length))
FalsePos_shift_cluster1_combs <- FalsePos_shift_cluster1_combs %>% arrange(desc(n))
FalsePos_shift_cluster1_combs <- mutate(FalsePos_shift_cluster1_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster1_combs <-
  add_column(FalsePos_shift_cluster1_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster1_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 2.

FalsePos_shift_cluster2_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster2))~.,
                      FalsePos_shift_cluster2, length))
FalsePos_shift_cluster2_combs <- FalsePos_shift_cluster2_combs %>% arrange(desc(n))
FalsePos_shift_cluster2_combs <- mutate(FalsePos_shift_cluster2_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster2_combs <-
  add_column(FalsePos_shift_cluster2_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster2_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 3.

FalsePos_shift_cluster3_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster3))~.,
                      FalsePos_shift_cluster3, length))
FalsePos_shift_cluster3_combs <- FalsePos_shift_cluster3_combs %>% arrange(desc(n))
FalsePos_shift_cluster3_combs <- mutate(FalsePos_shift_cluster3_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster3_combs <-
  add_column(FalsePos_shift_cluster3_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster3_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 4.

FalsePos_shift_cluster4_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster4))~.,
                      FalsePos_shift_cluster4, length))
```

```

FalsePos_shift_cluster4_combs <- FalsePos_shift_cluster4_combs %>% arrange(desc(n))
FalsePos_shift_cluster4_combs <- mutate(FalsePos_shift_cluster4_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster4_combs <-
  add_column(FalsePos_shift_cluster4_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster4_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 5.

FalsePos_shift_cluster5_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster5))~.,
                      FalsePos_shift_cluster5, length))
FalsePos_shift_cluster5_combs <- FalsePos_shift_cluster5_combs %>% arrange(desc(n))
FalsePos_shift_cluster5_combs <- mutate(FalsePos_shift_cluster5_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster5_combs <-
  add_column(FalsePos_shift_cluster5_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster5_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 6.

FalsePos_shift_cluster6_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalsePos_shift_cluster6))~.,
                      FalsePos_shift_cluster6, length))
FalsePos_shift_cluster6_combs <- FalsePos_shift_cluster6_combs %>% arrange(desc(n))
FalsePos_shift_cluster6_combs <- mutate(FalsePos_shift_cluster6_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalsePos_shift_cluster6_combs <-
  add_column(FalsePos_shift_cluster6_combs,
             comb_rank = min_rank(desc(FalsePos_shift_cluster6_combs$n)),
             .before = "adv_bb_type")

```

Cluster 1 was the largest cluster, containing 176 (29%) of our 611 false positives. The 32 most common feature sets contained nearly 70% of the batted balls in the cluster. These feature sets consist primarily of solidly hit (90 mph or greater) ground balls hit up the middle or toward the middle infielder on the opposite side by mostly left-handed batters with average speed. Against the shift, balls hit up the middle are covered by the shortstop (playing to the right of second base) and the third baseman (playing in the shortstop's traditional location, but cheating a bit toward the middle). It's not surprising, then, that these ground balls are resulting in outs. But why wouldn't the model be making this connection between a shifted infield and grounders hit at these spray angles?

Cluster 3 had 88 batted balls in it, 14% of our false positives against a shift. Despite its relatively small size, the cluster had 61 unique combinations of features, so it was clearly more diverse than Cluster 1. Both ground balls and low line drives comprised the 18 most common feature sets. These balls were hit at least 90 miles per hour, and the spray angles were clearly restricted to the left side of the field, pulled by right-handed batters with average or below average speed to first base. What made the model predict that these pulled grounders and liners were going to get through a shifted infield?

Cluster 4, our second largest cluster with 141 false positives (23% of all false positives against a shift), is similar to Cluster 3, but for left-handed batters. Its 19 most common combinations of features included nearly two-thirds of the batted balls in this cluster. They revealed a cluster of high ground balls and low line drives mostly hit at least 90 miles per hour, and pulled to the right side by left-handed batters with average speed to first base. Some of these plays are being made by a shifted second baseman playing in short right field. Again, why would the model predict that these pulled grounders and liners were going to squeeze through a shifted infield?

Cluster 6 contained 90 false positives with 62 distinct feature sets. The 18 most common feature sets was comprised mainly of high ground balls and low line drives hit at least 80 miles per hour up the middle and also pulled toward the second baseman by batters with below average speed. As mentioned before, a shifted infield blankets the middle hole with the third baseman and one of the middle infielders. The other middle infielder is stationed in short right field at spray angles directed toward the second baseman's traditional location. So there are infielders in position to stop most of the batted balls in this cluster. Plus, the unusually high hp to 1b times of these batters don't boost their chances of reaching base safely.

Cluster 2 consists of only 27 batted balls in 20 distinct feature sets. But it's a unique group of topped or weakly hit low ground balls toward the left side or toward the second baseman. The third baseman and pitcher are making most of these plays, just as with the cluster (Cluster 6) of swinging bunts and bunt attempts against a standard infield alignment.

Finally, Cluster 5, containing 89 false positives in 71 distinct feature sets, is another unique group of batted balls among the 611 false positives against a shift. Mostly line drives and fly balls, with some high ground balls and popups mixed in as well, most of these balls had exit velocities under 80 miles per hour and were headed up the middle or toward or over the head of a middle infielder. Both left-handed and right-handed batters hit most of these balls to the opposite side of the field. Does this cluster sound familiar? It's very similar to the cluster of flares we saw against a standard infield alignment. Here, against the shift, the balls with lower launch angles are going to be fielded by the third baseman or a middle infielder playing toward the middle. The balls with higher launch angles are going to be run down in the shallow outfield by an outfielder charging in. As we stated before, these are close plays because the balls are hit high enough and soft enough to hang for a while in the air, allowing a fielder some time to try to reach it.

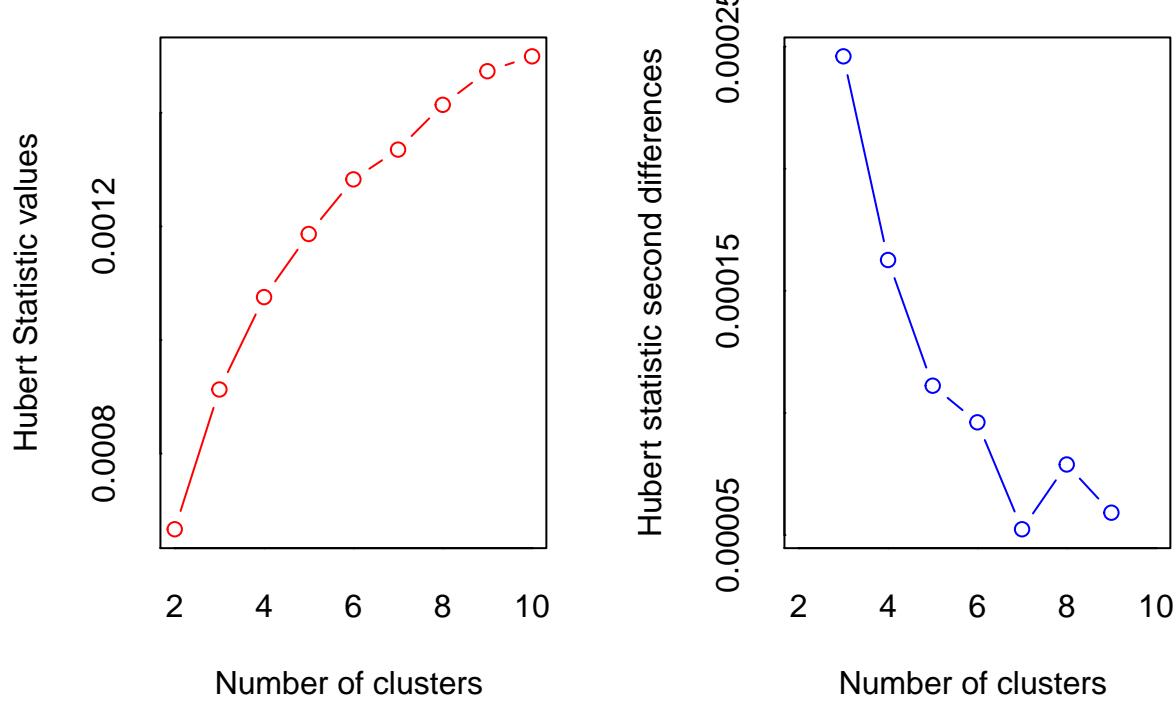
K-means clustering tells a story about false positives against a shift, and it's somewhat similar to the one it told about false positives against a standard infield alignment. False positives against a shift are dominated by four clusters of fairly well-struck ground balls and low line drives. As most shifts are deployed against left-handed batters, it's no surprise that three of the four clusters involve lefties pulling the ball to the right side or hitting it up the middle or toward the shortstop's traditional location, all areas that are generally covered by shifted infielders. This made us wonder whether our top-performing random forest model was even recognizing the interaction between infield alignment and spray angle. Yet, it managed to correctly predict the outcome of 80% of all ground balls and low line drives in the test set. The spray angle features must have helped to some degree, but their full potential won't be realized until we have more precise data on the locations of the infielders at the time the ball was hit.

False Negatives with a Standard Infield Alignment

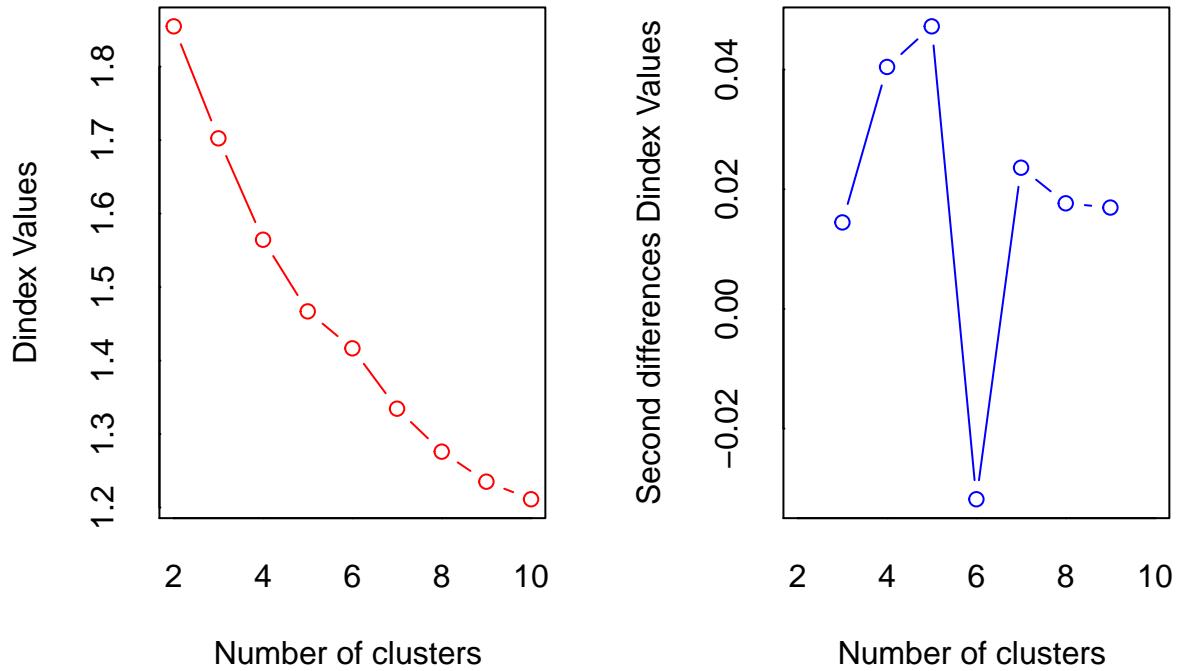
There were only 550 misclassified outs against a standard infield, as compared to 1,453 misclassified singles against a standard infield. We applied K-means clustering to help describe the most common combinations of features.

```
# Use NbClust package to identify optimal number of clusters in
# rf_predict_FalseNeg_std_scaled.

library(NbClust)
set.seed(1234)
nc <- NbClust(rf_predict_FalseNeg_std_scaled, distance = "euclidean", min.nc = 2,
               max.nc = 10, method = "kmeans")
```



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
## In the plot of Hubert index, we seek a significant knee that corresponds to a
## significant increase of the value of the measure i.e the significant peak in Hubert
## index second differences plot.
##
```



```

## *** : The D index is a graphical method of determining the number of clusters.
## In the plot of D index, we seek a significant knee (the significant peak in Dindex
## second differences plot) that corresponds to a significant increase of the value of
## the measure.
##
## *****
## * Among all indices:
## * 7 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 4 proposed 5 as the best number of clusters
## * 2 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
## ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 2
##
## *****

##
```

```

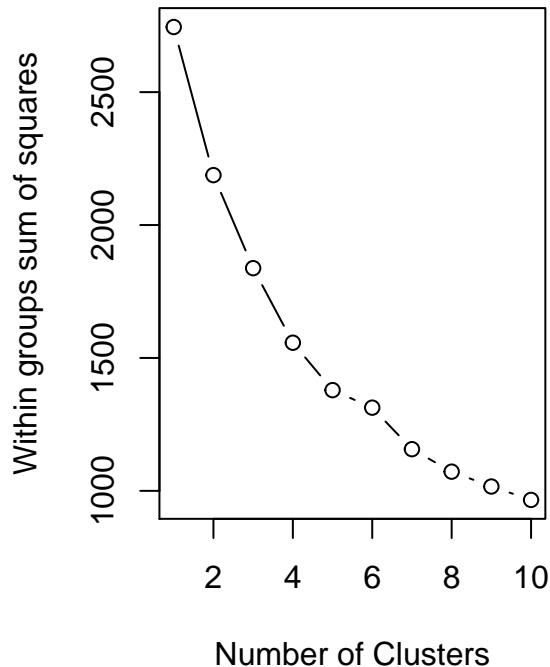
##  0   1   2   3   4   5   7   8   9  10
##  2   1   7   3   3   4   2   1   1   2

# Identify optimal number of clusters in rf_predict_FalseNeg_std_scaled by
# plotting the total within-groups sums of squares against the number of
# clusters. (reference: Kabacoff)

wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}

wssplot(rf_predict_FalseNeg_std_scaled, nc = 10, seed = 1234)

```



NbClust identified 2 as the optimal number of clusters for false negatives against a standard infield alignment. The K-means clustering algorithm then grouped the 550 false negatives into a cluster of 387 and a cluster of 163.

```

# Use K-means clustering to identify the types of batted balls that tend to be
# misclassified as outs against an infield in Standard alignment.

set.seed(1234)
km_fit <- kmeans(rf_predict_FalseNeg_std_scaled, 2, iter.max = 10, nstart = 50)

```

```

round.aggregate(rf_predict_FalseNeg_std[, 9:13], by = list(cluster = km_fit$cluster),
               median), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1          1        -8.4       91.6      -16.5      -22.8
## 2          2        11.1       81.2      13.1       18.0
##   hp_to_1b
## 1    4.35
## 2    4.33

round.aggregate(rf_predict_FalseNeg_std[, 9:13], by = list(cluster = km_fit$cluster),
               mad), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1          1        14.38      12.90      18.38      11.56
## 2          2        20.31      18.98      20.76      13.49
##   hp_to_1b
## 1    0.18
## 2    0.18

km_fit$size

## [1] 387 163

km_clusters <- fitted(km_fit, method = "classes")
rf_predict_FalseNeg_std <- add_column(rf_predict_FalseNeg_std, km_clusters,
                                         .after = "obs_nmbr")

```

We next plotted the batted balls in each of the two clusters. We would use the batted ball plots to supplement our analysis of the most common combinations of features in each cluster.

```

# Create a tibble for each cluster of false negatives against an infield in
# Standard alignment.

FalseNeg_std_cluster1 <- rf_predict_FalseNeg_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 1)
FalseNeg_std_cluster2 <- rf_predict_FalseNeg_std %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 2)

# Visualize false negative predictions by cluster, when the infield was in a
# Standard alignment.

neg_labels <- c("false negatives")
names(neg_labels) <- c("FalseNeg")

ggplot(rf_predict_FalseNeg_std, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "white", size = 1.5)

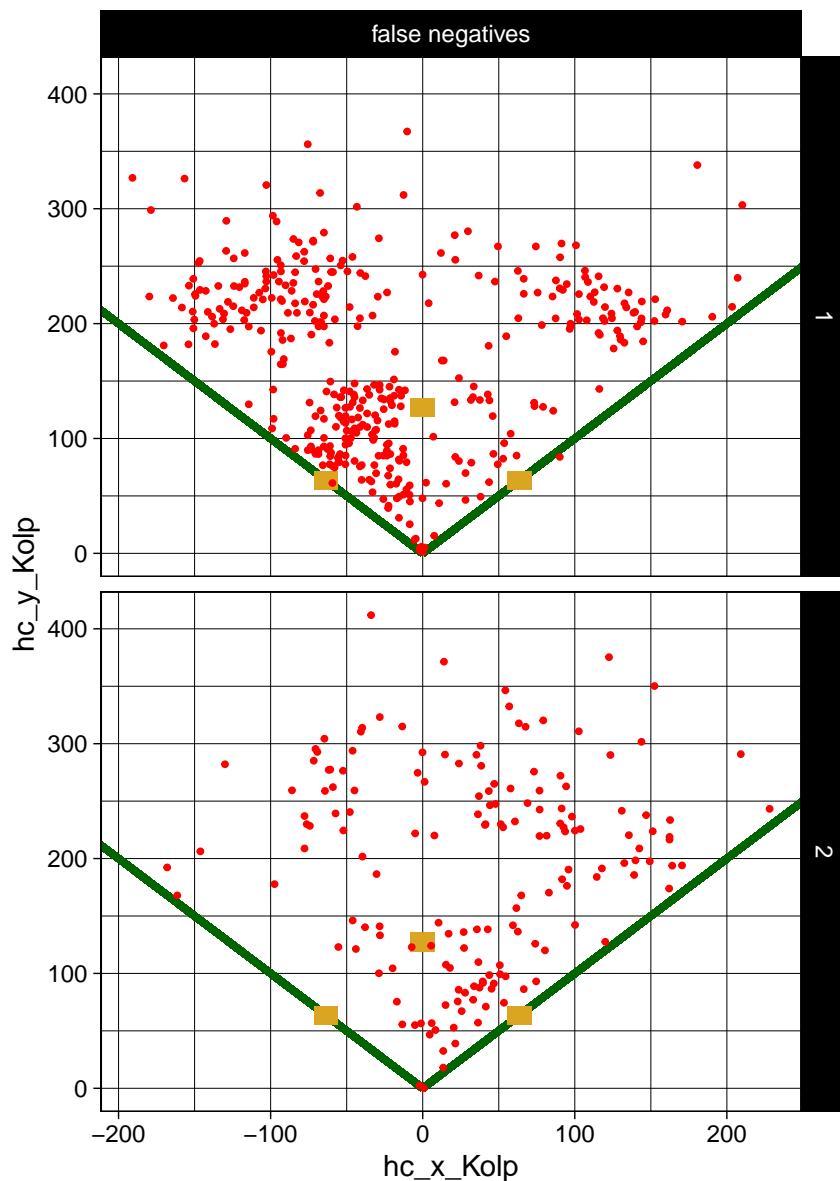
```

```

        fill = "goldenrod") +
geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
          fill = "goldenrod") +
geom_point(size = 0.7, color = "red") + theme_linedraw() +
labs(title = "FalseNeg_std Preds by Cluster") +
facet_grid(km_clusters ~ eval_type, labeller = labeller(eval_type = neg_labels))

```

FalseNeg_std Preds by Cluster



We next computed the most common combinations of features in each cluster.

```

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 1.

```

```

FalseNeg_std_cluster1_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_std_cluster1))~.,
                      FalseNeg_std_cluster1, length))
FalseNeg_std_cluster1_combs <- FalseNeg_std_cluster1_combs %>% arrange(desc(n))
FalseNeg_std_cluster1_combs <- mutate(FalseNeg_std_cluster1_combs,
                                       n_pct = round(n/sum(n)*100, 2),
                                       n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_std_cluster1_combs <-
  add_column(FalseNeg_std_cluster1_combs,
             comb_rank = min_rank(desc(FalseNeg_std_cluster1_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 2.

FalseNeg_std_cluster2_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_std_cluster2))~.,
                      FalseNeg_std_cluster2, length))
FalseNeg_std_cluster2_combs <- FalseNeg_std_cluster2_combs %>% arrange(desc(n))
FalseNeg_std_cluster2_combs <- mutate(FalseNeg_std_cluster2_combs,
                                       n_pct = round(n/sum(n)*100, 2),
                                       n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_std_cluster2_combs <-
  add_column(FalseNeg_std_cluster2_combs,
             comb_rank = min_rank(desc(FalseNeg_std_cluster2_combs$n)),
             .before = "adv_bb_type")

```

The 45 most common feature sets in Cluster 1 consisted mainly of ground balls (particularly low ground balls) of all velocities pulled by both left-handed and right-handed batters with a large range of hp to 1b times. Only about half of these balls made it to the outfield, and about 71% were hit to the left side.

Here, we see a cluster in which the single-out dichotomy doesn't depend so much on whether the ball made it through the infield. There are a lot of ground balls in this cluster that are stopped on the left side of the infield, and yet they turned out to be singles when the model thought they would be outs.

Our curiosity compelled us to check some video, which showed the vast majority of these balls to be legitimate infield hits. While the shortstop or third baseman was able to reach the ball, the exit velocity and spray angle required them to either charge the ball, dive left, dive right, or, in the case of the shortstop, cover a lot of ground toward the 5-6 hole or the middle hole. If the infielder was able to make the throw to first base, the batter was able to beat the throw.

So the balls being stopped in the infield were tough predictions, often coming down to the defensive skills of the individual infielders and the batter's jump out of the batter's box. However, the balls reaching left field were squeezing between the shortstop and third baseman, indicating our spray angle and infield alignment data was not precise enough to permit the model to make correct predictions.

Cluster 2, with 163 false negatives in 124 distinct feature sets, is similar to Cluster 1 in that a significant portion of these balls (36% in this case) were stopped in the infield, and the balls were hit by both left-handed and right-handed batters with a large range of hp to 1b times. The similarities stop there, however. These balls were hit to the opposite side of the field, and the spray angles have a smaller range, extending from the first baseman to the shortstop. Fewer of these balls were hit over 100 miles per hour, and, while ground balls (dominated by low ground balls) comprise half of the batted balls in this cluster, there are a significant number of high line drives and popups as well.

The video for Cluster 2 revealed a more varied collection of infield hits, including balls lost in the sun, and

balls that drop amidst confused infielders. These outcomes were even less predictable than the infield hits in Cluster 1.

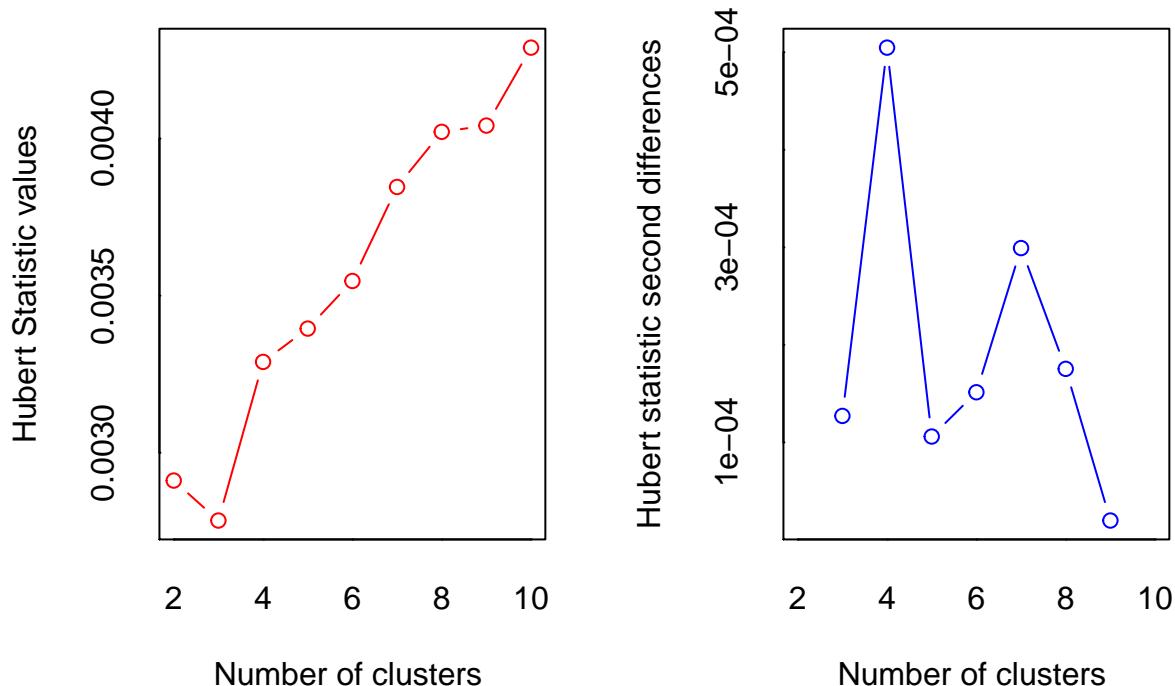
Similarly, the balls reaching the outfield included grounders hit between infielders, but also an array of popups and flares hanging in the air begging to be caught, but not being caught for a variety of reasons. Sometimes, the outfielder took a wrong first step, or didn't pick up the ball right away, or was shaded in the opposite direction from the ball and thus had a long run. We didn't really concern ourselves with the positioning of the outfielders in this project because it doesn't come into play as much with singles, but had we incorporated outfield alignment as a feature, our expectation would be that we would run into the same challenge we did with infield alignment: inaccurate and imprecise data on the location of the outfielders at the time the ball was struck.

False Negatives with a Shifted Infield Alignment

We only had 175 misclassified outs against a shifted infield. But we were still able to use the same clustering approach to describe the most common feature sets.

```
# Use NbClust package to identify optimal number of clusters in
# rf_predict_FalseNeg_shift_scaled.

library(NbClust)
set.seed(1234)
nc <- NbClust(rf_predict_FalseNeg_shift_scaled, distance = "euclidean", min.nc = 2,
               max.nc = 10, method = "kmeans")
```

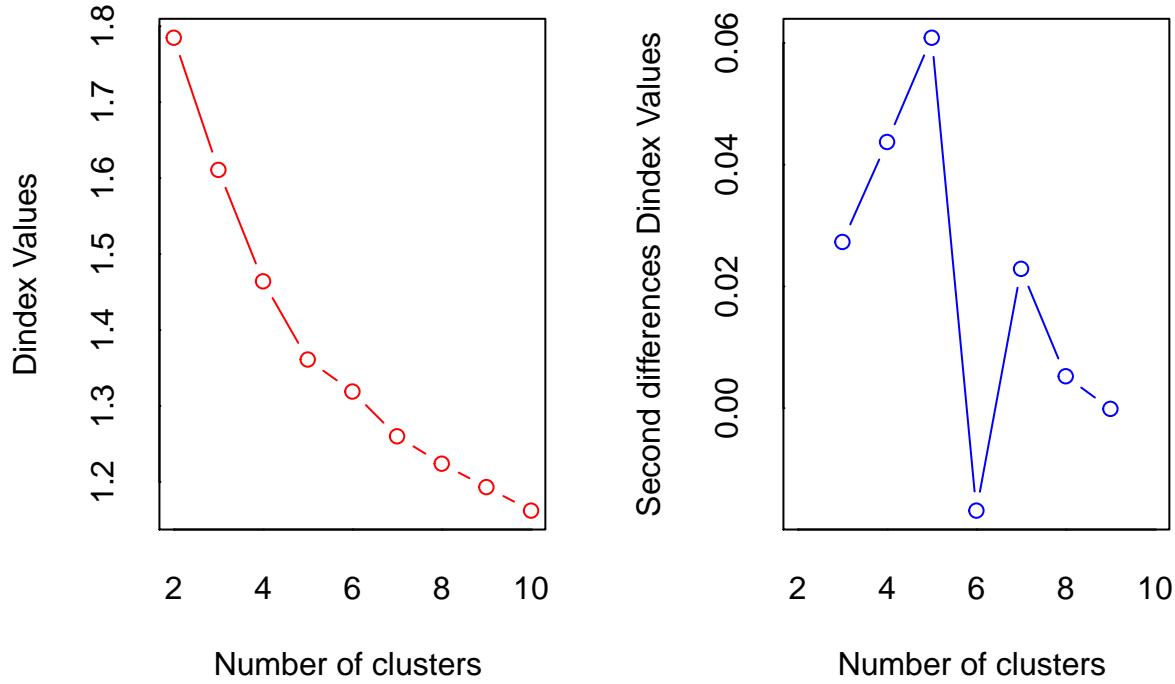


```
## *** : The Hubert index is a graphical method of determining the number of clusters.
```

```

## In the plot of Hubert index, we seek a significant knee that corresponds to a
## significant increase of the value of the measure i.e the significant peak in Hubert
## index second differences plot.
##

```



```

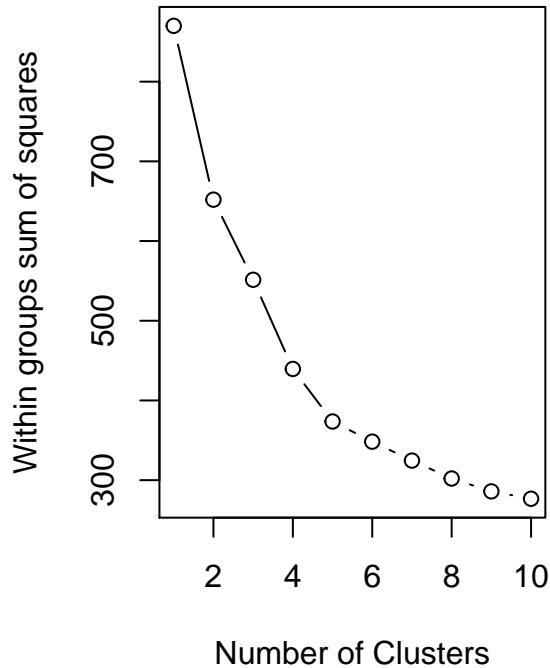
## *** : The D index is a graphical method of determining the number of clusters.
## In the plot of D index, we seek a significant knee (the significant peak in Dindex
## second differences plot) that corresponds to a significant increase of the value of
## the measure.
##
## *****
## * Among all indices:
## * 6 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 9 proposed 5 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
##
## ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is 5
##
## *****

```

```



```



Despite the small number of false negatives against a shift, both NbClust and wssplot identified five as the optimal number of clusters for false negatives against a shift. So the K-means clustering algorithm created

five clusters, one larger cluster and four quite small clusters. Surprisingly, they were actually quite distinct from one another.

```
# Use K-means clustering to identify the types of batted balls that tend to be
# misclassified as outs against a shifted infield.

set.seed(1234)
km_fit <- kmeans(rf_predict_FalseNeg_shift_scaled, 5, iter.max = 10, nstart = 50)
round.aggregate(rf_predict_FalseNeg_shift[, 9:13], by = list(cluster = km_fit$cluster),
               median), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1      1        0.55       98.4      28.60      -28.60
## 2      2        1.20       88.5      21.40      -16.85
## 3      3       -4.65       94.9     -23.15     -23.15
## 4      4       -15.05       69.4      -9.20      12.00
## 5      5        28.20       80.7     -8.30      13.90
##   hp_to_1b
## 1    4.33
## 2    4.86
## 3    4.46
## 4    4.36
## 5    4.44

round.aggregate(rf_predict_FalseNeg_shift[, 9:13], by = list(cluster = km_fit$cluster),
               mad), 2)

##   cluster launch_angle launch_speed spray_angle_Kolp spray_angle_adj
## 1      1        9.86       8.38      9.04      9.04
## 2      2        7.93       6.38     10.75     14.68
## 3      3       12.53       5.04     11.12     11.12
## 4      4       23.05      11.93     15.86     18.24
## 5      5       15.42      13.94     19.72     13.64
##   hp_to_1b
## 1    0.13
## 2    0.16
## 3    0.13
## 4    0.20
## 5    0.18

km_fit$size

## [1] 76 12 28 32 27

km_clusters <- fitted(km_fit, method = "classes")
rf_predict_FalseNeg_shift <- add_column(rf_predict_FalseNeg_shift, km_clusters,
                                         .after = "obs_nmbr")
```

We then plotted the batted balls in each cluster to supplement our analysis of the most common combinations of features.

```

# Create a tibble for each cluster of false negatives against a shifted infield.

FalseNeg_shift_cluster1 <- rf_predict_FalseNeg_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 1)
FalseNeg_shift_cluster2 <- rf_predict_FalseNeg_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 2)
FalseNeg_shift_cluster3 <- rf_predict_FalseNeg_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 3)
FalseNeg_shift_cluster4 <- rf_predict_FalseNeg_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 4)
FalseNeg_shift_cluster5 <- rf_predict_FalseNeg_shift %>%
  dplyr::select(adv_bb_type, launch_speed_cat, rev_spray_angle_Kolp_cat,
                rev_spray_angle_adj_cat, hp_to_1b_cat) %>% filter(km_clusters == 5)

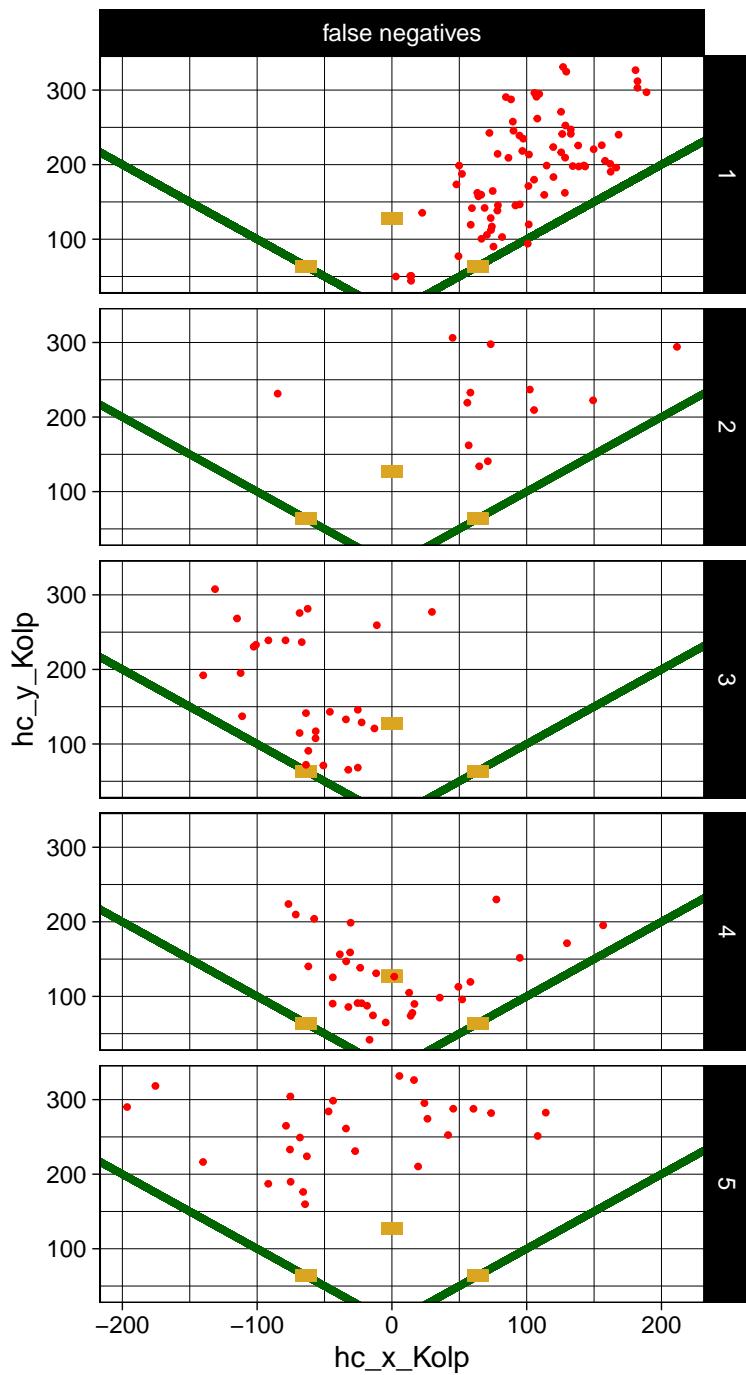
# Visualize false negative predictions by cluster, when the infield was fully
# shifted.

neg_labels <- c("false negatives")
names(neg_labels) <- c("FalseNeg")

ggplot(rf_predict_FalseNeg_shift, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.7, color = "red") + theme_linedraw() +
  labs(title = "FalseNeg_shift Preds by Cluster") +
  facet_grid(km_clusters ~ eval_type, labeller = labeller(eval_type = neg_labels))

```

FalseNeg_shift Preds by Cluster



We then identified the most common combinations of features in each cluster.

```
# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 1.
```

```
FalseNeg_shift_cluster1_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_shift_cluster1))~.,
    FalseNeg_shift_cluster1, length))
```

```

FalseNeg_shift_cluster1_combs <- FalseNeg_shift_cluster1_combs %>% arrange(desc(n))
FalseNeg_shift_cluster1_combs <- mutate(FalseNeg_shift_cluster1_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_shift_cluster1_combs <-
  add_column(FalseNeg_shift_cluster1_combs,
             comb_rank = min_rank(desc(FalseNeg_shift_cluster1_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 2.

FalseNeg_shift_cluster2_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_shift_cluster2))~.,
                      FalseNeg_shift_cluster2, length))
FalseNeg_shift_cluster2_combs <- FalseNeg_shift_cluster2_combs %>% arrange(desc(n))
FalseNeg_shift_cluster2_combs <- mutate(FalseNeg_shift_cluster2_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_shift_cluster2_combs <-
  add_column(FalseNeg_shift_cluster2_combs,
             comb_rank = min_rank(desc(FalseNeg_shift_cluster2_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 3.

FalseNeg_shift_cluster3_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_shift_cluster3))~.,
                      FalseNeg_shift_cluster3, length))
FalseNeg_shift_cluster3_combs <- FalseNeg_shift_cluster3_combs %>% arrange(desc(n))
FalseNeg_shift_cluster3_combs <- mutate(FalseNeg_shift_cluster3_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_shift_cluster3_combs <-
  add_column(FalseNeg_shift_cluster3_combs,
             comb_rank = min_rank(desc(FalseNeg_shift_cluster3_combs$n)),
             .before = "adv_bb_type")

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 4.

FalseNeg_shift_cluster4_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_shift_cluster4))~.,
                      FalseNeg_shift_cluster4, length))
FalseNeg_shift_cluster4_combs <- FalseNeg_shift_cluster4_combs %>% arrange(desc(n))
FalseNeg_shift_cluster4_combs <- mutate(FalseNeg_shift_cluster4_combs,
                                         n_pct = round(n/sum(n)*100, 2),
                                         n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_shift_cluster4_combs <-
  add_column(FalseNeg_shift_cluster4_combs,
             comb_rank = min_rank(desc(FalseNeg_shift_cluster4_combs$n)),
             .before = "adv_bb_type")

```

```

# Determine the number of times each combination of features (feature set) was
# represented by a batted ball in Cluster 5.

FalseNeg_shift_cluster5_combs <-
  as_tibble(aggregate(cbind(n = 1:nrow(FalseNeg_shift_cluster5))~.,
    FalseNeg_shift_cluster5, length))
FalseNeg_shift_cluster5_combs <- FalseNeg_shift_cluster5_combs %>% arrange(desc(n))
FalseNeg_shift_cluster5_combs <- mutate(FalseNeg_shift_cluster5_combs,
  n_pct = round(n/sum(n)*100, 2),
  n_cum_pct = round(cumsum(n)/sum(n)*100, 2))
FalseNeg_shift_cluster5_combs <-
  add_column(FalseNeg_shift_cluster5_combs,
  comb_rank = min_rank(desc(FalseNeg_shift_cluster5_combs$n)),
  .before = "adv_bb_type")

```

The first cluster contains 76 false negatives against the shift. They make up 40 distinct feature sets, the first 17 of which include 70% of the batted balls in Cluster 1. This is our familiar cluster of solidly hit (at least 90 mph) ground balls, except these were all pulled into the shift by left-handed hitters with average speed.

About one-third of these balls were stopped by an infielder, generally the one positioned in short right field, after having taken several steps to his left or right. The batter, having the advantage of starting his sprint from the left side of the plate, was able to beat the infielder's throw to first base.

The balls getting past infielders were simply being pounded through the narrower gaps in the shift. The fact that the gaps in a shifted infield are smaller makes it even more important to have precise information about the location of the infielders. Video showed that even within the shifted alignment category (three infielders on one side of second base), the infielders are positioned in a variety of locations.

Like Cluster 1, the second cluster's 12 false negatives against the shift are mainly ground balls pulled by left-handed batters. But these grounders were hit at lower exit velocities (between 80 and 100 miles per hour) by batters with slow times to first base. A higher proportion of these balls got through the infield, so the batters' lack of speed turns out to be less significant.

Video revealed that while some of these plays were legitimate singles into gaps, there were also some fluky plays involving infielders slipping on grass, making errors that were ruled singles, and playing in (closer to home plate) to stop a runner on third base from scoring. Infielders playing in raises still further doubt about the adequacy of spray angle data without knowing the infielders' precise coordinates, informing the model of both the horizontal and vertical location of an infielder when the ball is hit.

Cluster 3 is a return to solidly hit (at least 90 mph) ground balls, except these were pulled into the shift by right-handed hitters. About half of these balls were stopped by infielders. While these batters didn't have the advantage of starting their sprint to first base from the left side of home plate, they instead gained the advantage of hitting the ball to the left side of the infield, forcing longer throws to first base by the third baseman and shortstop. The balls that made it to the outfield were simply hammered through holes in the shift. When the third baseman or shortstop was able to stop one of these grounders, they were usually too off balance to make a good throw due to having to dive or move laterally.

Cluster 4 consists of topped and weakly hit (less than 80 miles per hour) low ground balls heading toward the middle and opposite side of the field (between the traditional positions of the shortstop and second baseman). The shifted infielders stopped about three-fourths of these balls but were unable to throw the batters out at first base. Most of these balls were legitimate singles, requiring the infielder run a good distance to get to the ball. A few should have been ruled errors. And a few "beat the shift," being hit far enough to the opposite side to get into the huge gap that's inevitably created by a full shift.

The video brought to light an interesting aspect of shifted infields, and that is that shifted infielders are often being asked to cover areas of the field unfamiliar to them. This likely doesn't make much of a difference for

athletic infielders with good instincts, but for infielders with a limited range of defensive skills, asking them to shift to a different location in the infield can sometimes really expose their deficiencies. Third basemen, in particular, now have to approach batted balls from different angles, cover more ground, and make throws from unfamiliar angles and distances. It may be more difficult to hit a ball through a shifted infield, but it's not as impenetrable as it appears, for this reason.

The fifth and last cluster is composed mainly of outfield balls hit in the air between 70 and 100 miles per hour between the left fielder and the right fielder. The balls not hit to straight-away center field were mostly hit to the opposite field. Most of these false negatives were the result of the batter getting jammed by an inside pitch, or hitting the ball off the end of the bat, or balls being lost in sun, or balls falling amidst confused fielders. Still, most were legitimate singles that just dropped in outfield gaps even though they weren't hit that well. These would have been tough predictions for any model to make.

Results

We were quite satisfied with the results of our initial effort to develop a model capable of distinguishing singles from outs. There were three primary reasons for our satisfaction: 1) we were trying to predict a minority class (singles), which is a notorious challenge in the area of machine learning; 2) our models intentionally considered only predictors that a batter could conceivably control, and thus didn't take into account the many external factors that can cause a batted ball to result in a single instead of an out; and 3) our uncertainty about the quality and usefulness of our spray angle data, given the amount of manipulation that was necessary to produce it and the limited amount of information we had about the locations of infielders when the ball was hit.

In light of these considerations, the performance of our kNN, classification tree, and random forest algorithms was indeed a pleasant surprise. Attaining both a true positive rate and a true negative rate above 0.80 was unexpected. Our rforest_train_c30 model, with a true positive rate of 0.839 and a true negative rate of 0.849, outperformed the others by achieving a truescore of 0.844 and a distance of 0.221 from (0, 1). And it produced its results in an efficient manner. We had created a solid foundation for further research.

```
# Assess the predictive ability of the randomForest algorithm.

confusionMatrix(rforest_predict, test_set3$events)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction   single field_out
##   single        4178      2284
##   field_out     803       12853
##
##                  Accuracy : 0.8466
##                         95% CI : (0.8415, 0.8515)
##      No Information Rate : 0.7524
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.6255
##
##      Mcnemar's Test P-Value : < 2.2e-16
##
##                  Sensitivity : 0.8388
##                  Specificity  : 0.8491
##      Pos Pred Value : 0.6465
##      Neg Pred Value : 0.9412
```

```

##          Prevalence : 0.2476
##          Detection Rate : 0.2077
##  Detection Prevalence : 0.3212
##          Balanced Accuracy : 0.8439
##
##          'Positive' Class : single
##


(rforest_tpr <- round(sensitivity(rforest_predict,
                                reference = factor(test_set3$events)), 6))

## [1] 0.838787

(rforest_tnr <- round(specificity(rforest_predict,
                                reference = factor(test_set3$events)), 6))

## [1] 0.849111

(rforest_truescore <- round((2 * rforest_tpr * rforest_tnr) /
                               (rforest_tpr + rforest_tnr), 6))

## [1] 0.843917

(rforest_distance <- round(sqrt((1 - rforest_tpr)^2 + (1 - rforest_tnr)^2), 6))

## [1] 0.22081

assessment_results <- tibble(Model = c("Baseline", "Log Regression", "Log Regression",
                                         "kNN (k23c31)", "Wtd kNN (k31w2c31)",
                                         "Class. Tree", "Class. Tree",
                                         "Random Forest"),
                                `Cutoff Method` = c(NA, "Truescore", "Distance", "both",
                                                   "both", "Truescore", "Distance",
                                                   "both"),
                                `Truescore` = c(bl_truescore, max_truescore,
                                                min_distance_truescore,
                                                knn_k23c31_truescore,
                                                kknn_k31w2c31_truescore,
                                                ctree_pruned_c22_truescore,
                                                ctree_pruned_c21_truescore,
                                                rforest_truescore),
                                TPR = c(bl_tpr, max_truescore_tpr,
                                         min_distance_tpr, knn_k7c40_tpr, knn_k23c31_tpr,
                                         ctree_pruned_c22_tpr, ctree_pruned_c21_tpr,
                                         rforest_tpr),
                                TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                         knn_k23c31_tnr, kknn_k31w2c31_tnr,
                                         ctree_pruned_c22_tnr, ctree_pruned_c21_tnr,
                                         rforest_tnr),
                                Distance = c(NA, max_truescore_distance, min_distance,
                                             max_truescore_tnr, min_distance_tnr))

```

```

knn_k23c31_distance, kknn_k31w2c31_distance,
ctree_pruned_c22_distance,
ctree_pruned_c21_distance,
rforest_distance),
`Best Cutoff` = c(NA, max_truescore_cutoff,
min_distance_cutoff,
knn_opt_cut, kknn_opt_cut,
ctree_opt_cut_ts, ctree_opt_cut_dist,
rf_opt_cut_ts))

knitr::kable(assessment_results[1:8, ], caption = "Assessment Results")

```

Table 56: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	Distance	Best Cutoff
Baseline	NA	0.385984	0.259787	0.7506110	NA	NA
Log Regression	Truescore	0.408139	0.383256	0.4362160	0.835599	0.749984
Log Regression	Distance	0.399327	0.328448	0.5092160	0.831776	0.764628
kNN (k23c31)	both	0.820183	0.772937	0.8154850	0.254349	0.31
Wtd kNN (k31w2c31)	both	0.824830	0.824935	0.8057740	0.248611	0.31
Class. Tree	Truescore	0.834452	0.808673	0.8619277	0.235945	0.22
Class. Tree	Distance	0.833956	0.813692	0.8552553	0.235927	0.21
Random Forest	both	0.843917	0.838787	0.8491110	0.220810	0.3

Our random forest model accorded the most importance, 40%, to *launch_angle*, followed by *launch_speed* at 23%, and our *spray_angle* predictors (*spray_angle_Kolp* (12%), *spray_angle_adj* (13%), and *if_fielding_alignment* (3%)). As expected, the feature considered least important by our random forest model was *hp_to_1b* at 9%.

These levels of importance are generally consistent with other data analyses performed throughout this project. Exit velocity consistently took a back seat to launch angle. *Spray_angle_Kolp* and *spray_angle_adj* each conveyed different but equally valuable information regarding the horizontal direction of batted balls. We knew our *hp_to_1b* feature would only come into play on a fraction of the batted balls in our data, so its relative level of importance seemed appropriate, and justified its inclusion in the study.

By including infield defensive alignment in our model, we hoped to enhance the influence of our spray angle features. The extent to which this actually occurred is unclear. This issue will be addressed further in our Conclusion. Aside from infield alignment, our selected features made meaningful contributions to the predictive ability of our top performing model, *rforest_train_c30*.

Conclusion

We found quality data, transformed it, and then used it to make predictions about whether a batted ball was going to result in a hit or an out. The predictions turned out to be pretty darn good, but we were left with the impression that they could have been better.

When you stop to think about it, though, it's pretty amazing that a few pieces of information about how a baseball comes off a bat in the first milliseconds after contact can usually tell us the result of the play. A seemingly infinite number of things can happen once the bat hits the ball. And in the context of this paper, there was so much we intentionally didn't know that would have been helpful in predicting the outcome of a batted ball.

Does the pitcher like to work this batter inside, outside, up, down? How skilled are the nine defensive players who are trying to get the batter out? Where are the fielders located when the ball is hit? Which direction is the wind blowing? Is the grass wet and slippery? Is it a high sky with a blinding sun? Did one of the teams get a short night's rest due to travel? Is the shortstop's range limited by a minor injury? Is the rookie center fielder's lack of familiarity with the corner outfielders going to result in confusion on balls hit between them? Despite these and other relevant factors being unknown to us, we can still determine with a reasonable level of certainty whether the batter will reach first base safely or make an out, based only on the direction and velocity of the ball off the bat, and perhaps how fast the batter is capable of running to first base.

Factors like the condition of the field and the state of the umpire's indigestion will always be there to remind us that we'll never be able to predict outcomes on a baseball field with 100% certainty. Even if all the external factors could somehow be held constant, biology and the laws of physics make it virtually impossible for a human batter to consistently replicate the muscle movements he knows will result in a single. So we can all rest easy in knowing our great game can never be reduced to an algorithm.

The question, then, is how close can we come to a full understanding of how batters make singles and outs? In this project, our self-imposed constraints required us to use predictors that are conceivably within the control of the batter. Given that requirement, should we be able to predict single-out outcomes with 90% certainty? 95% certainty? What will it take to get there? To what extent can improvements in the quality of existing data get us there? Are there additional measurements that can be collected and made available that would help the cause?

Of course, we wouldn't have been able to develop the model we did had MLB not invested in the advanced technology that can transform what happens on a baseball field into a pile of numbers. And still, it seems so much more can be done with regard to collecting data and making it available to the public. We have to keep reminding ourselves that Baseball Savant does not exist to furnish data scientists with sustenance for research.¹⁷ But in the course of performing its real mission of providing common fans with fun and interesting metrics, it comes so darn close to feeding more serious research that we can taste it!

Understandably, Baseball Savant has thus far focused on Launch Angle and Exit Velocity as the primary predictors of batting results. But these two predictors by themselves can't explain all we would like to know. Spray Angle, an admittedly more complex predictor, is out there waiting to help.

For this study, we were able to scrape up enough data and manipulate it in enough ways to paint a somewhat hazy but informative picture of Spray Angle's impact on the outcome of a batted ball. But this is an area where the quality of data could be significantly improved. The spray angle of every batted ball should be measured directly and not inferred from the batted ball's landing coordinates. The coordinates of the nearest fielders when the ball is hit should also be made available, though it's unclear the extent to which various types of machine learning algorithms are capable of utilizing an interactive variable such as infield alignment, which makes Spray Angle more informative, but by itself, tells us little about the batted ball's outcome. Lastly, the merits of making Home Plate to First Base measurements available for each batted ball should now be evident.

It's our hope that this study satisfactorily demonstrates that high quality spray angle data can notably enhance the performance of many types of predictive algorithms used in baseball research. We look forward to the day MLB's research priorities turn their attention toward Spray Angle, a very informative feature of all batted balls.

In the context of this study, the random forest algorithm demonstrated superior performance in using a handful of batted ball features to distinguish a single from an out. It recognized a single to be a single 84% of the time. It recognized an out to be an out 85% of the time.

While the k-nearest neighbors and classification tree algorithms performed nearly as well, they were not designed to facilitate the identification of an optimal decision threshold. Being able to test alternatives to the simple majority decision rule proved important in our case because we were faced with the challenge of a binary outcome with unbalanced classes.

¹⁷Baccellieri, Emma, "Major League Baseball's Statcast Can Break Sabermetrics" (Dec 18, 2017). <https://deadspin.com/major-league-baseballs-statcast-can-break-sabermetrics-1820987737>

It was apparent throughout our study that altering the decision cutoff was a powerful tool for overcoming majority class bias in our predictive model. Yet many black box algorithms such as knn and classification trees, after going through the trouble of computing and outputting precise probability estimates, fail to offer any alternative cutoff capabilities. Unbalanced outcomes, common in machine learning scenarios, warrant packaged algorithms with greater built-in flexibility for testing alternative decision points as well as alternative criteria for evaluation (e.g. harmonic mean of sensitivity and specificity, shortest distance to (0, 1)).

There are, of course, numerous machine-learning algorithms that were not tested during this study. It's quite possible that one of these untried algorithms could outperform our application of random forest. We suspect any improvement would be marginal given the data currently available. But this would be another area for further research.

.....

We set out trying to determine whether a single can be reliably distinguished from a field out based on information about how the ball comes off the bat. We thought it was a challenging question that would put our data science skills to the test. And while it has no obvious practical usefulness to teams, the media, or even fantasy baseball owners, we also thought it an interesting question. We believe the answer is yes, singles do come off the bat differently than field outs. But this leaves some related interesting questions unanswered. Are singles more easily distinguishable from extra base hits than they are from outs? To what extent can a batter develop some degree of control over the mechanical act of hitting the ball at a certain launch angle, exit velocity, and spray angle? Even if a batter can learn this kind of control, is a single a mere afterthought, the fortuitous byproduct of a failed extra base hit?

These days, there's a broad data-based recognition by baseball people that runs are most efficiently scored with extra-base hits, especially home runs. A batter's ability to get on base is still valued because he can then score when another batter gets an extra-base hit. But a batter's ability to merely reach first base and nothing more is less valued. A batter's ability to reach first base via a single is even less valued. So the question is, do any players even try to get a single any more? Does it ever make sense to try to get a single? Are there players in the game today whose offensive skills are limited to hitting singles? Were there ever such players? There are so many good questions that we couldn't help dipping into the data one last time.

We examined singles as a proportion of on-base events, defined as singles divided by hits plus walks plus hit by pitches (and hereafter referred to as the singles proportion). Note that the singles proportion (and the proportion of any other individual on-base event, such as home runs) by itself doesn't have much to do with the overall level of offense in the league, typically measured by runs per game. Many other variables have a more direct effect on the number of runs scored, so the correlation between a given on-base event's proportion and runs per game is not strong. You could say that the proportions have more to do with how runs were scored than how many runs were scored. They tell us something about the importance of a given type of on-base event (singles or home runs) relative to that of other on-base events.

We began in 1920, after the so-called dead ball era of 1901 to 1919, a period when the same baseball was used until it literally fell apart. The baseballs were soft from overuse, disfigured by scuff marks, and dark with dirt, tobacco juice, spit and myriad other foreign substances. Thus, pitches were difficult to see, and moved in unpredictable ways toward home plate. When a pitch was hit, the ball didn't travel as far.¹⁸ Teams scraped for runs by bunting, executing the hit and run, and stealing bases. The small ball style of play produced a meager 3.92 runs per game between 1901 and 1919. And singles comprised a whopping 58% of all on-base events. Singles would never again be as prominent as they were in the dead ball era.

To see what happened with singles after modern baseball's infancy, we plotted the singles proportion from 1920 through the 2019 season.

¹⁸ McDonald, Jason, "What Are the Major Eras of Major League Baseball History?" (2013). huffpost.com

```

# Import a data file into an R data frame.

singles_props <-
  read_csv("https://raw.githubusercontent.com/jfmusso/HarvardX/master/Singles_Proportions.csv",
  col_types = cols(
    Year = col_number(),
    `1B` = col_integer(),
    H = col_integer(),
    BB = col_integer(),
    HBP = col_integer()
  )
)

class(singles_props[])

```

[1] "tbl_df" "tbl" "data.frame"

```

# Compute the average singles proportion for the period 1920 to 2019.

avg_singles_prop <- round(sum(singles_props$`1B`) / (sum(singles_props$H) +
  sum(singles_props$BB) +
  sum(singles_props$HBP)), 3)

# Plot the season-by-season singles proportions over the period
# 1920 to 2019.

ggplot(data = singles_props, aes(x = Year, y = `1B/(H+BB+HBP)`)) +
  geom_line(color = "royalblue3", size = 1.1) +
  scale_x_continuous(breaks = seq(1920, 2020, by = 5),
    labels = c("1920", "1925", "1930", "1935", "1940", "1945", "1950",
    "1955", "1960", "1965", "1970", "1975", "1980", "1985",
    "1990", "1995", "2000", "2005", "2010", "2015", "2020"),
    limits = c(1920, 2019)) +
  scale_y_continuous(breaks = seq(0.42, 0.580, by = .02),
    labels = c("0.42", "0.44", "0.46", "0.48", "0.50", "0.52",
    "0.54", "0.56", "0.58"),
    limits = c(0.43, 0.585)) +
  theme(axis.text.x = element_text(angle = 45)) +
  geom_hline(yintercept = avg_singles_prop) +
  labs(title = "Singles as a Proportion of On-Base Events", x = "Year",
    y = "Singles/(H+BB+HBP)") +
  annotate("text", size = 4, fontface = 1, x = 1923, y = 0.506,
    label = "Avg 0.503") +
  annotate("text", size = 4, fontface = 1, x = 1930.5, y = 0.583,
    label = "A") +
  annotate("text", size = 4, fontface = 1, x = 1944, y = 0.553,
    label = "B") +
  annotate("text", size = 4, fontface = 1, x = 1955, y = 0.514,
    label = "C") +
  annotate("text", size = 4, fontface = 1, x = 1974, y = 0.542,
    label = "D") +
  annotate("text", size = 4, fontface = 1, x = 1989, y = 0.518,

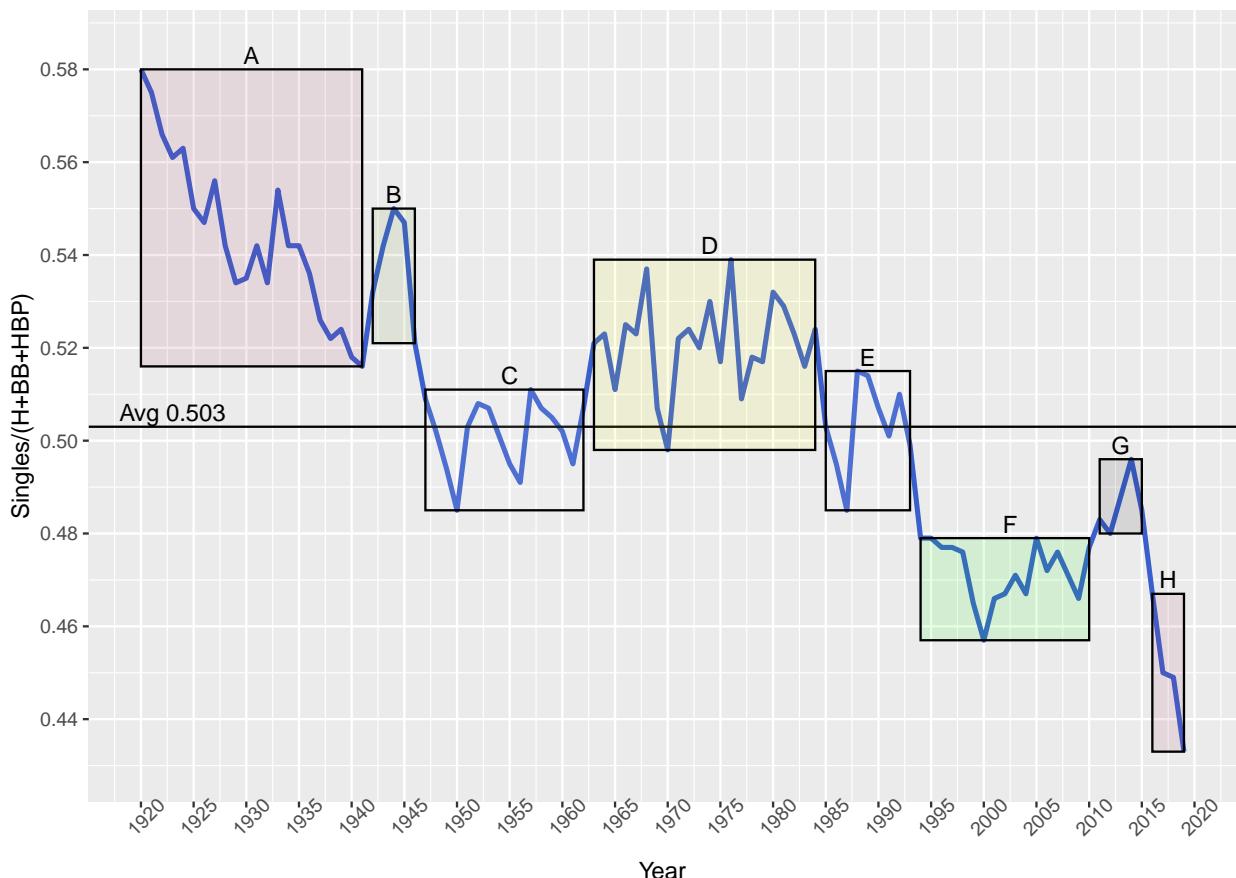
```

```

        label = "E") +
  annotate("text", size = 4, fontface = 1, x = 2002.5, y = 0.482,
           label = "F") +
  annotate("text", size = 4, fontface = 1, x = 2013, y = 0.499,
           label = "G") +
  annotate("text", size = 4, fontface = 1, x = 2017.5, y = 0.470,
           label = "H") +
  annotate("rect", xmin = 1920, xmax = 1941, ymin = 0.516, ymax = 0.580,
           alpha = .1, color = "black", fill = "maroon") +
  annotate("rect", xmin = 1942, xmax = 1946, ymin = 0.521, ymax = 0.550,
           alpha = .1, color = "black", fill = "olivedrab") +
  annotate("rect", xmin = 1947, xmax = 1962, ymin = 0.485, ymax = 0.511,
           alpha = .1, color = "black", fill = "white") +
  annotate("rect", xmin = 1963, xmax = 1984, ymin = 0.498, ymax = 0.539,
           alpha = .1, color = "black", fill = "yellow") +
  annotate("rect", xmin = 1985, xmax = 1993, ymin = 0.485, ymax = 0.515,
           alpha = .1, color = "black", fill = "white") +
  annotate("rect", xmin = 1994, xmax = 2010, ymin = 0.457, ymax = 0.479,
           alpha = .1, color = "black", fill = "green") +
  annotate("rect", xmin = 2011, xmax = 2015, ymin = 0.480, ymax = 0.496,
           alpha = .1, color = "black", fill = "gray9") +
  annotate("rect", xmin = 2016, xmax = 2019, ymin = 0.433, ymax = 0.467,
           alpha = .1, color = "black", fill = "maroon")

```

Singles as a Proportion of On–Base Events



At first glance, the plot reveals several facts worth noting: 1) Since the dead ball era, an average of 50% of all on-base events have been singles. 2) The singles proportion has experienced a number of rises and falls during the last 100 seasons. 3) The singles proportion's long-term trend is downward.

We know that the number of home runs has increased since the dead ball era. So we plotted the home run proportion to be able to compare it to the singles proportion.

```
# Compute the average home run proportion for the period 1920 to 2019.

avg_hr_prop <- round(sum(singles_props$`HR`) / (sum(singles_props$H) +
                                                 sum(singles_props$BB) +
                                                 sum(singles_props$HBP)), 3)

# Compare the season-by-season singles proportions and home run proportions
# over the period 1920 to 2019.

singles_prop_plot <- ggplot(data = singles_props, aes(x = Year, y = `1B/H+BB+HBP`)) +
  geom_line(color = "royalblue3", size = 1.1) +
  scale_x_continuous(breaks = seq(1920, 2020, by = 5),
                      labels = c("1920", "1925", "1930", "1935", "1940", "1945", "1950",
                                "1955", "1960", "1965", "1970", "1975", "1980", "1985",
                                "1990", "1995", "2000", "2005", "2010", "2015", "2020"),
                      limits = c(1920, 2019)) +
  scale_y_continuous(breaks = seq(0.42, 0.580, by = .02),
                      labels = c("0.42", "0.44", "0.46", "0.48", "0.50", "0.52", "0.54",
                                "0.56", "0.58"),
                      limits = c(0.43, 0.585)) +
  theme(axis.text.x = element_text(angle = 45)) +
  geom_hline(yintercept = avg_singles_prop) +
  labs(title = "Singles as a Proportion of On-Base Events", x = "Year",
       y = "Singles/(H+BB+HBP)") +
  annotate("text", size = 4, fontface = 1, x = 1924, y = 0.508,
           label = "Avg 0.503") +
  annotate("text", size = 4, fontface = 1, x = 1930.5, y = 0.5845,
           label = "A") +
  annotate("text", size = 4, fontface = 1, x = 1944, y = 0.5545,
           label = "B") +
  annotate("text", size = 4, fontface = 1, x = 1955, y = 0.5155,
           label = "C") +
  annotate("text", size = 4, fontface = 1, x = 1974, y = 0.5435,
           label = "D") +
  annotate("text", size = 4, fontface = 1, x = 1989, y = 0.5195,
           label = "E") +
  annotate("text", size = 4, fontface = 1, x = 2002.5, y = 0.4835,
           label = "F") +
  annotate("text", size = 4, fontface = 1, x = 2013, y = 0.5005,
           label = "G") +
  annotate("text", size = 4, fontface = 1, x = 2017.5, y = 0.4715,
           label = "H") +
  annotate("rect", xmin = 1920, xmax = 1941, ymin = 0.516, ymax = 0.580,
           alpha = .1, color = "black", fill = "maroon") +
  annotate("rect", xmin = 1942, xmax = 1946, ymin = 0.521, ymax = 0.550,
           alpha = .1, color = "black", fill = "olivedrab") +
  annotate("rect", xmin = 1947, xmax = 1962, ymin = 0.485, ymax = 0.511,
           alpha = .1, color = "black", fill = "white") +
```

```

annotate("rect", xmin = 1963, xmax = 1984, ymin = 0.498, ymax = 0.539,
        alpha = .1, color = "black", fill = "yellow") +
annotate("rect", xmin = 1985, xmax = 1993, ymin = 0.485, ymax = 0.515,
        alpha = .1, color = "black", fill = "white") +
annotate("rect", xmin = 1994, xmax = 2010, ymin = 0.457, ymax = 0.479,
        alpha = .1, color = "black", fill = "green") +
annotate("rect", xmin = 2011, xmax = 2015, ymin = 0.480, ymax = 0.496,
        alpha = .1, color = "black", fill = "gray9") +
annotate("rect", xmin = 2016, xmax = 2019, ymin = 0.433, ymax = 0.467,
        alpha = .1, color = "black", fill = "maroon")

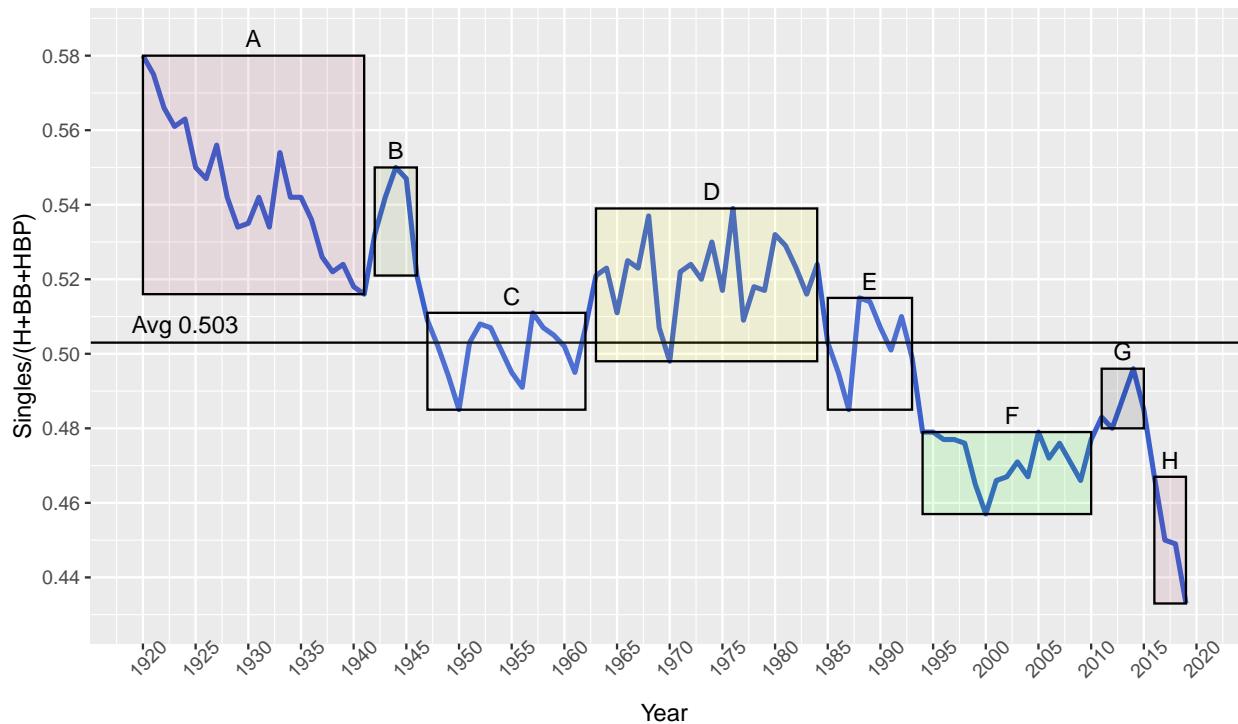
homers_prop_plot <- ggplot(data = singles_props, aes(x = Year, y = `HR/H+BB+HBP`)) +
  geom_line(color = "red3", size = 1.1) +
  scale_x_continuous(breaks = seq(1920, 2020, by = 5),
    labels = c("1920", "1925", "1930", "1935", "1940", "1945", "1950",
              "1955", "1960", "1965", "1970", "1975", "1980", "1985",
              "1990", "1995", "2000", "2005", "2010", "2015", "2020"),
    limits = c(1920, 2019)) +
  scale_y_continuous(breaks = seq(0.02, 0.12, by = .01),
    labels = c("0.02", "0.03", "0.04", "0.05", "0.06", "0.07",
              "0.08", "0.09", "0.10", "0.11", "0.12"),
    limits = c(0.02, 0.117)) +
  theme(axis.text.x = element_text(angle = 45)) +
  geom_hline(yintercept = avg_hr_prop) +
  labs(title = "Home Runs as a Proportion of On-Base Events", x = "Year",
       y = "Home Runs/(H+BB+HBP)") +
  annotate("text", size = 4, fontface = 1, x = 1924, y = 0.069,
           label = "Avg 0.066") +
  annotate("text", size = 4, fontface = 1, x = 1930.5, y = 0.053,
           label = "A") +
  annotate("text", size = 4, fontface = 1, x = 1944, y = 0.043,
           label = "B") +
  annotate("text", size = 4, fontface = 1, x = 1955, y = 0.080,
           label = "C") +
  annotate("text", size = 4, fontface = 1, x = 1974, y = 0.077,
           label = "D") +
  annotate("text", size = 4, fontface = 1, x = 1989, y = 0.087,
           label = "E") +
  annotate("text", size = 4, fontface = 1, x = 2002.5, y = 0.092,
           label = "F") +
  annotate("text", size = 4, fontface = 1, x = 2013, y = 0.088,
           label = "G") +
  annotate("text", size = 4, fontface = 1, x = 2017.5, y = 0.116,
           label = "H") +
  annotate("rect", xmin = 1920, xmax = 1941, ymin = 0.021, ymax = 0.050,
          alpha = .1, color = "black", fill = "maroon") +
  annotate("rect", xmin = 1942, xmax = 1946, ymin = 0.030, ymax = 0.040,
          alpha = .1, color = "black", fill = "olivedrab") +
  annotate("rect", xmin = 1947, xmax = 1962, ymin = 0.048, ymax = 0.077,
          alpha = .1, color = "black", fill = "white") +
  annotate("rect", xmin = 1963, xmax = 1984, ymin = 0.048, ymax = 0.074,
          alpha = .1, color = "black", fill = "yellow") +
  annotate("rect", xmin = 1985, xmax = 1993, ymin = 0.059, ymax = 0.084,

```

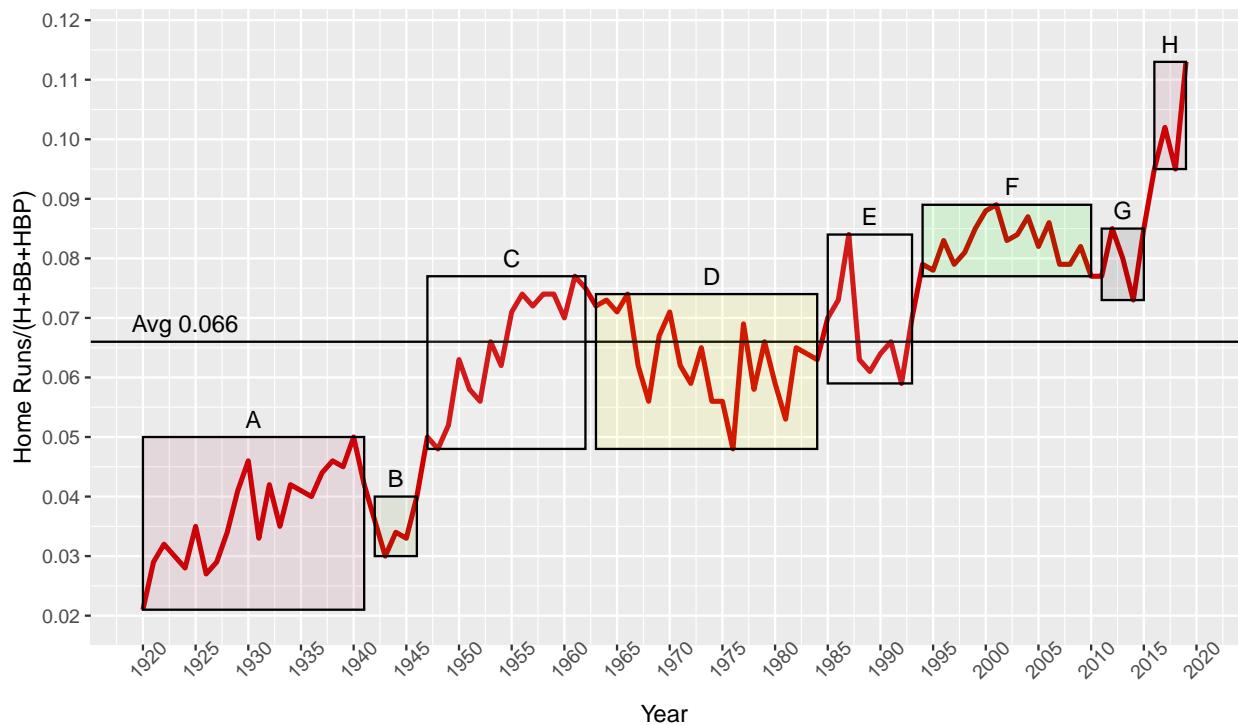
```
alpha = .1, color = "black", fill = "white") +
annotate("rect", xmin = 1994, xmax = 2010, ymin = 0.077, ymax = 0.089,
         alpha = .1, color = "black", fill = "green") +
annotate("rect", xmin = 2011, xmax = 2015, ymin = 0.073, ymax = 0.085,
         alpha = .1, color = "black", fill = "gray9") +
annotate("rect", xmin = 2016, xmax = 2019, ymin = 0.095, ymax = 0.113,
         alpha = .1, color = "black", fill = "maroon")

grid.arrange(singles_prop_plot, homers_prop_plot, ncol = 1)
```

Singles as a Proportion of On-Base Events



Home Runs as a Proportion of On-Base Events



The new plots reveal a strong inverse relationship between the singles proportion and the home run proportion. To see how strong, we computed the correlations between the proportions for each on-base event, plus we threw in stolen bases per game, strikeouts per game, and sacrifice hits per game, just out of curiosity.

```

# Compute the correlations between each on-base event's proportions.

cor_data <- dplyr::select(singles_props, Year, `1B/H+BB+HBP`, `2B/H+BB+HBP`,
                           `3B/H+BB+HBP`, `HR/H+BB+HBP`, `BB/H+BB+HBP`,
                           `HBP/H+BB+HBP`, `SB/G`, `SO/G`, `SH/G`)

cor_data <- cor_data %>% rename(Year = Year, `1B` = `1B/H+BB+HBP`, `2B` = `2B/H+BB+HBP`,
                                   `3B` = `3B/H+BB+HBP`, HR = `HR/H+BB+HBP`,
                                   BB = `BB/H+BB+HBP`, HBP = `HBP/H+BB+HBP`, SB = `SB/G`,
                                   SO = `SO/G`, SH = `SH/G`)

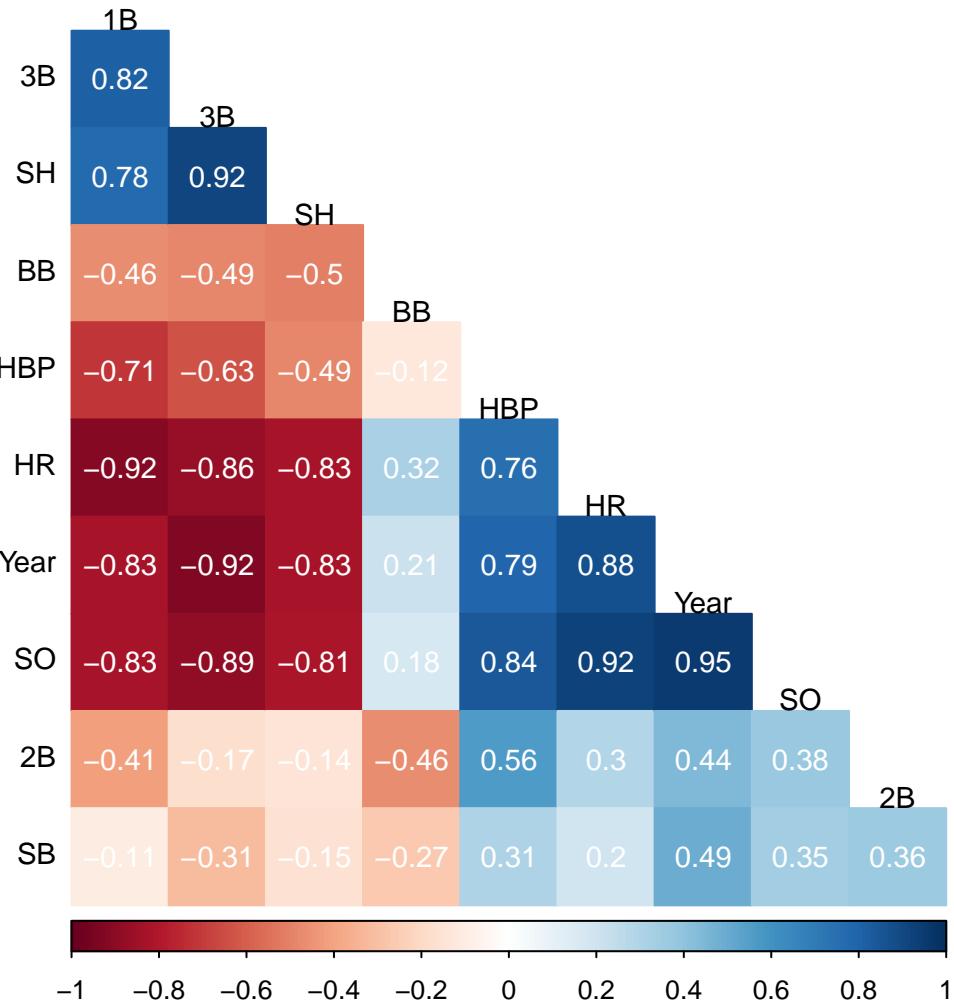
props_cor <- round(cor(cor_data), 3)

# Create a correlogram to visualize the correlations and group them accordingly.

corrplot(props_cor, method = "color", type = "lower",
         diag = FALSE, addCoef.col = "white", order = "hclust",
         tl.col = "black", tl.srt = 0, tl.cex = 0.9, number.cex = 0.95,
         number.font = 15)
title("Correlations Between On-Base Event Proportions", line = 2.5)

```

Correlations Between On-Base Event Proportions



The correlogram makes it easier to understand the correlations by identifying the strongest correlations with the darkest colors. Positive correlations are shown in shades of blue, while negative correlations are shown in shades of red. Knowing this, several observations jump out at us:

- 1) The singles proportion, the triples proportion, and sacrifice hits per game have strong positive correlations with each other, which means they have a strong tendency to move together in the same direction. If one of them decreases, the other two are likely to decrease as well. For the sake of brevity, we'll refer to this group of proportions as the singles proportion. Just keep in mind that when we talk about changes in the singles proportion, the triples proportion and sacrifice hits per game are probably tagging along.
- 2) The home run proportion, strikeouts per game, and the hit by pitch proportion also tend to move together in the same direction. If one of them increases, the other two are likely to increase as well. Again, for the sake of brevity, we'll refer to this group of proportions as the home run proportion. Just remember that when we discuss the home run proportion, the hit by pitch proportion and strikeouts per game are likely coming along for the ride.
- 3) There is a strong negative correlation between the singles proportion and the home run proportion.

Historically, they've moved in opposite directions. If the singles proportion goes down, the home run proportion is likely heading up. Another way to say this is that if singles become a less important part of offense, home runs become a more important part of offense. It doesn't have to be this way, but historically, this is what has happened; they've moved in opposite directions.

- 4) The correlogram also reveals the very strongest long-term trends over the last 100 years of baseball history, since the first dead ball era ended. These can be seen in the row and column labeled "Year." The singles proportion has a strong negative correlation with Year; that is, as the year has gotten larger (more recent), the singles proportion has tended to fall. The long-term home run trend is just the opposite. As the year has gotten larger, the home run proportion has tended to rise. This is a significant phenomenon that shows just how dramatically the game of baseball has changed since its early years. Teams today employ a different approach to scoring runs than teams in the 1920s. They look for different types of players to fill their rosters, and then use them in different ways with different strategies.

With that big picture in mind, we took a bit more detailed look at changes in the singles proportion, beginning in 1920. A baseball historian would have a fun challenge trying to tie each instance of variation to specific events, a more daunting task than it seems. We now know, for example, that any change in the baseball manufacturing process, no matter how trivial, and regardless of whether those involved are even aware of the change, has the potential to alter not only how many runs are scored, but how those runs are produced. This makes it difficult to attribute any change in the singles proportion to one particular cause. Reality is always more complicated than the generalizations of historians. The truth is that changes in the singles proportion over time are likely the net effect of multiple factors, some offsetting each other, and some compounding each other, and some not even known to us.

Having said that, let's do it anyway, because it's fun and because it's the only way humans can make some sense of a complicated world. Bill James, the baseball writer, identified 366 "dividing" events in baseball history that fell into 10 broad categories: 1) rule changes; 2) changes to the structure of the league (e.g., wild cards and interleague play); 3) changes in the conditions in which the game is played (e.g., night baseball, artificial turf, steroid use, equipment changes); 4) changes in managerial strategy (e.g., 5-man pitching rotation, closers); 5) radical changes in statistical achievements, such as the number of stolen bases; 6) franchise moves; 7) new ballparks; 8) external events such as World War II; 9) certain players' careers; and 10) commissioners. James considered stadium architecture and game equipment to be "the two largest dynamics of change in baseball."¹⁹ Some of these forces for change are more likely to directly impact the singles proportion than others. But it should now be apparent how easy it would be to go down the rabbit hole.

The Live Ball Era (1920-1941): The Singles Proportion Experiences a Lengthy Decline (Segment A)

We opted to stay at a more superficial level and not try to offer an explanation every time the plot of singles proportions changed direction. To start, we divided the plot into eight segments based on significant changes in the trend. For example, the first segment, labeled "A", shows a fairly steady and dramatic rate of decline in the singles proportion from 1920, when it was 0.580, through 1941, when it was 0.516. These years coincide with what is customarily referred to as the live ball era.

It might have been more accurate to name it the clean new ball era. Spitballs and "emery" balls were banned, and umpires were directed to frequently replace scuffed, dirty, and discolored balls with clean new white balls.²⁰ Once batters could see the ball again, and unnatural breaking pitches were abolished, the proportion of walks and home runs began trending up at the same time the singles proportion started heading down. The only interruption occurred when Baseball introduced the cushioned-cork center into baseballs in

¹⁹James, Bill, "Dividing Baseball History into Eras" (2012). https://www.billjamesonline.com/dividing_baseball_history_into_eras/

²⁰Schwarz, Alan, "The history of rule changes". https://www.espn.com/mlb/columns/schwarz_alan/1503763.html

1931 in reaction to the startling rise in home runs during 1929 and 1930.²¹ But it wasn't long before the home run proportion resumed its more normal upward trajectory. Runs per game remained relatively steady throughout this period, averaging 4.84, nearly a full run above the run rate during the dead ball era.

So here was an example of rule changes and playing conditions having an immediate and detectable effect on both the number of runs scored, and the manner in which those runs were produced. It was no longer as beneficial to advance baserunners into scoring position by giving up an out. And it made less sense to fill roster spots with players who could hit for a higher average and run fast, unless they could also hit for power.

A more gradual change fueling the decline of the singles proportion was the emergence of concrete and steel stadiums, most of which had outfield fences that batters could reach more easily. In addition, the newer parks tended to move the fans further away from the field, creating more foul territory. This had the effect of making it easier for batters to make outs and more difficult to get hits. These two features of modern stadiums, homer-friendly dimensions and larger foul territory, meant that stringing together a series of hits in an inning became a less efficient way for teams to score runs, and powering extra base hits (particularly home runs) became a more efficient way to score runs, even though home runs came with a lot of strikeouts.

The effects of the new parks weren't as sudden and apparent as those driven by ball-related changes. It's more helpful to think of the modern stadium effects as constantly there in the background season after season, putting upward pressure on home runs and downward pressure on the importance of singles and small ball strategies such as sacrifice hits.

The first wave of modern parks opened between 1909 and 1923, and hung around until the early 1950s, when a second wave of parks began entering the league. Yet the singles proportion continued dropping throughout the live ball era and even after. According to Bill James, "Almost every change in ballparks between 1930 and 1968 took hits out of the league.... There is no change in ballparks that I am aware of between 1930 and 1968 that significantly favored batting averages. Some changes favored home run hitters, but none or almost none favored singles hitters."²²

We also have an example of a player's career (as well as a radically new statistical achievement) affecting the singles proportion in the live ball era. In 1920, his first season with the New York Yankees, Babe Ruth played in the field full-time for the first time in his career (not being required to pitch any longer). In addition, his home park was now the homer-friendly Polo Grounds. The result was 54 home runs, nearly doubling the single season home run record he had set the previous year with the Boston Red Sox.

Players both current and prospective could now see with their own eyes that it was possible to have a powerful uppercut swing and still hit the ball consistently. Ruth provided the model. Like the emergence of homer-friendly parks, this effect likely grew throughout the live ball era, as players who adopted Ruth's approach to hitting in their youth gradually entered the league.

World War II Effects (1942-1946): Replacement Players Play with a Replacement Ball (Segment B)

It took over 20 years for the singles proportion to drop from 0.580 to 0.516 in 1941. But it only took three seasons during World War II to rise back up to 0.550 in 1944. Meanwhile, the proportion of extra base hits fell enough to sink offensive levels back to 4.07 runs per game between 1942 and 1946. Scoring runs hadn't been that difficult since the dead ball era.

Here we have an external event, World War II, impacting the singles proportion in two ways: diminishing the talent pool, and changing the materials used to manufacture baseballs. Rubber, an important ingredient of a baseball's core as well as a component of tanks and planes, was banned in items not essential to the war effort. The redesigned ball was made of less elastic materials. The dead ball and with it, small ball, were

²¹Jaffe, Jay, "Was MLB's Juiced Era Actually A Juiced-Ball Era?" (2012). <https://deadspin.com/was-mlbs-juiced-era-actually-a-juiced-ball-era-5937432>

²²James, Bill, 2001. "*The New Bill James Historical Baseball Abstract*". First Free Press trade paperback ed. 2003. New York.

back.²³ More importantly, Ted Williams, Joe DiMaggio, Stan Musial, and most of the other good players were in military service. More than 500 major leaguers in all had their careers interrupted for a greater cause.²⁴

Post-War Stability (1947-1962): The Singles Proportion Dives into a Period of Sameness (Segment C)

As soon as the war ended and the talent pool was replenished, the singles proportion resumed its dramatic descent that began in the live ball era. It declined every season from 1944 (0.550) to 1950 (0.485), the largest uninterrupted drop in baseball history. Then, for the first time, the singles proportion appeared to establish a state of equilibrium, hovering above and then below the all-time average of 0.503 in a cycle that repeated itself three and a half times between 1947 and 1962.

Of all the time segments in our plots of proportions, this is the one in which the inverse relationship between the singles proportion and the home run proportion does not hold up as well. While the singles proportion remained fairly stable, the home run proportion, aided by MLB's decision to shrink the strike zone for the 1950 season,²⁵ resumed its relentless trek upward from the live ball era. Home runs as a proportion of on-base events increased steadily and significantly throughout this post-war period, from 0.048 in 1948 to 0.077 in 1961.

The home run had never played such an important role in generating runs. The level of offense was not an issue during this period, but the nature of offense was. Rosters were filled out with players who could hit the occasional home run, but couldn't do much else. The re-emergence of base stealers at the very end of this era offered some hope, however.²⁶

The Second Dead Ball Era and its Aftermath (1963-1984): Rule Tinkering Triggers Rise in Singles Proportion and Drop in Home Run Proportion (Segment D)

The state baseball had reached by the late 1950s and early 1960s showed promise. The league had averaged 4.46 runs per game throughout the stable post-war period, significantly above the first dead ball era and the World War II years, but also significantly below the live ball era. While home runs were at historic levels, offenses had begun to diversify, making use of the stolen base, sacrifice hit, and players who hit for average. In addition, four new franchises in 1961 and 1962 added new flavor and contrast to the league.²⁷ Ah, but all good things must come to an end.

Triggered by Roger Maris breaking Babe Ruth's single-season home run record in 1961, the Baseball Rules Committee feared that home runs were becoming too routine. Their solution was to expand the strike zone prior to the 1963 season.²⁸ It might have worked, but they missed their mark. The strike zone was extended both upward, from the armpits to the shoulders, and downward, from the top of the knee to the bottom of the knee.²⁹

The change likely had a more dramatic effect than the Committee intended. The singles proportion immediately rose to 0.521 in 1963, a level not seen since the war years. The home run proportion eventually fell as well. From MLB's perspective, the problem was that runs per game collapsed to World War II levels.

²³Rymer, Zachary D., "The Evolution of the Baseball From the Dead-Ball Era Through Today" (June 18, 2013). <https://bleacherreport.com/articles/1676509-the-evolution-of-the-baseball-from-the-dead-ball-era-through-today>; Calcaterra, Craig, "Today in Baseball History: 'Balata ball' threatens a new Dead Ball Era" (May 4, 2020). <https://sports.yahoo.com/today-baseball-history-balata-ball-131736928.html>

²⁴Constantino, Rocco, "The Top 200 Moments That Shaped MLB's History" (May 17, 2012). <https://bleacherreport.com/articles/1157233-200-events-that-defined-shaped-and-changed-major-league-baseball/>

²⁵Schwarz

²⁶James (2001)

²⁷James (2001); Constantino

²⁸Treder, Steve, "Re-Imagining the Big Zone Sixties, Part 1: 1963-1965" (Nov 30, 2004). <https://tht.fangraphs.com/re-imagining-the-big-zone-sixties-part-1-1963-1965/>

²⁹<https://www.mlb.com/official-information/umpires/strike-zone>

Pitching dominated because the expansion of the strike zone caused strikeouts to shoot up, and walks to drop off dramatically. The quality of batters' contact no doubt suffered as well, with hitters being forced to swing at pitches they could previously take for balls. Thus began the second dead ball era.

We'd be remiss not to mention the more gradual effects of the second wave of ballparks that entered the league beginning in 1953. These weren't necessarily newly constructed parks; some were parks new to the league as a result of franchises moving to new locations. Regardless, most of the second-wave parks coming into the league in the 1950s and 1960s facilitated the growing domination of pitching that characterized the second dead ball era.³⁰

As the period continued, the behavior of the single and homer proportions can best be described as erratically stable, as the Rules Committee kept hitting buttons that sometimes overcorrected for the problem they perceived to be at hand. The singles proportion swung wildly around 0.520 for the next 21 seasons. The home run proportion swung a little less wildly around 0.063 throughout the same period, ending in 1984. That is eight years after runs per game moved back up to the levels of the late 1950s, and thus eight years after the second dead ball era is considered to have ended. More about that later. First, we'll describe some of the peaks and valleys experienced by the singles proportion between 1963 and 1976.

Offense eventually crashed to 3.42 runs per game in 1968, a level not reached since 1908, the heart of the first dead ball era. Baseball felt compelled to act again by restoring the smaller strike zone, lowering the mound, and actually enforcing the maximum permissible mound height (several teams with good pitching were believed to have pitcher's mounds higher than the rules allowed).³¹ In addition, league policy finally started requiring that the visual background for the batter (an area located in the center field stands known as the "batter's eye") protect the hitter's ability to see an incoming pitch.³²

These latest changes had their desired effect, jacking offense back up above 4 runs per game in 1969. The singles proportion, which had spiked to 0.537 in 1968, the highest since the war years, now fell back to 0.507, its largest season-to-season movement to this point in the 20th century. Predictably, the home run proportion, which had collapsed to 0.056 in 1968, now jumped back up to 1950s levels.

Strangely, the boost in offense was short-lived. Offense had suffered death by a thousand cuts (larger fielder's gloves, night baseball, etc.), and only some of the cuts had been treated. Thus, the roller-coaster ride of the second dead ball era continued, fueled by Baseball's seemingly quixotic search for balance, a balance that stubbornly refused to reveal itself. By 1971, runs per game was back down below 4.0, and the singles proportion was back up above 0.52. The American League, which had been lagging behind the National League in runs per game, brought out the big guns this time by implementing the designated hitter rule in 1973.³³ A position player off the bench could now hit for the pitcher. This clearly helped run production, at least in the American League.

Then, in 1974, MLB started using baseballs covered in cowhide rather than horsehide due to a shortage of horses.³⁴ The singles proportion proceeded to climb to a period-high 0.539 in 1976. It didn't stay there for long, though. In 1977, MLB began using balls manufactured by Rawlings instead of Spalding.³⁵ Whether intended or not, the singles proportion crashed again, every bit as hard as it did in 1969. And with that, the second dead ball era was truly over. Runs per game never fell below 4.0 again. Hits were more plentiful; outs were less frequent. The Rules Committee had come full circle. Runs per game were back up to the levels of the late 1950s, a few years before the second dead ball era commenced.

As mentioned earlier, the erratic stability of the singles proportion and home run proportion continued for another eight years, through the 1984 season. Both proportions kept fluctuating within intervals signaling greater reliance on the single and less reliance on the home run, compared to the preceding post-war period. What was the state of baseball at this point? The second dead ball era had ended, runs were up, but the singles proportion and home run proportion had not changed their behavior. Could something else have

³⁰James (2001)

³¹Schwarz; Constantino

³²James (2001)

³³McDonald

³⁴Jaffe (2012)

³⁵Jaffe (2012)

changed about the way runs were being created, aside from the fact that there were just more hits in general? The answer is yes. For the first time in our review, the singles and homer proportions don't tell the whole story. And for the first time since the transition from the first dead ball era to the live ball era, the doubles proportion jumped to a new level.

An important driver behind this odd eight-year period was a third wave of ballparks entering the league during the 1970s.³⁶ Almost all of them had fast AstroTurf playing surfaces. The newly constructed ones were sterile and uniform multi-purpose sports stadiums. But it was primarily the AstroTurf that affected the way offenses generated runs. The prototype was Houston's Astrodome, beginning in 1966.³⁷ Six more AstroTurf fields were added between 1970 and 1973, and three more in 1977. There would be ten AstroTurf fields in baseball until 1994. Most were in the National League.

Ground balls scooted across AstroTurf faster than grass, and line drives and fly balls bounced higher. Speed and quickness now mattered for both batters and fielders, perhaps more than at any time since the first dead ball era. Balls that weren't hit directly at an infielder could make it to the wall for a double or triple. The overall level of runs, 4.3 per game, was mid-level, historically. But the new playing conditions allowed all aspects of the game to emerge. Teams like the Royals and Cardinals, who did the best job constructing rosters that would flourish on AstroTurf, won many games during the late 1970s and 1980 despite consistently finishing near the bottom of the league in home runs. Singles, doubles, triples, stolen bases, and fielding percentage, these were their marks of honor.

The best thing about these years was that good teams could win in a variety of ways, and good players could possess different skill sets. The league featured players with high batting averages, players who stole a lot of bases, or hit a lot of homers, or drew a lot of walks, or hit a lot of doubles. There were dominant pitchers as well. Some starters won a lot of games, or completed a lot of games, or struck out a lot of batters, and some relievers saved a lot of games.

It was an unintended but welcome respite from the inexorable march of the home run. MLB had always been more concerned about the balance between offense and pitching. This was a new type of balance that fans hadn't seen. Attendance increased. But baseball purists struggled with the way the baseball behaved on this new surface. Batted balls that were previously routine outs were now singles, and singles were now doubles or triples. The speed of the game had changed.

In describing the 1963-1984 period, we have to some extent violated our promise not to go down the rabbit hole. But we thought it helpful to provide a sense of the events underlying some of the large peaks and valleys crossed by the singles proportion between 1963 and 1984. Simply put, there was a lot going on in this period; a lot of variables impacted the game, and we only touched the surface.

Baseball made a well-intentioned effort to exert some control over the direction of the game, but, unable to foresee all the consequences of its actions amidst the complex milieu in which baseball existed, it found itself having to make one correction after another to try to get it right. Arguably, they ultimately succeeded. The product that rolled out of MLB's factory in 1977 just might have achieved an optimal balance in the game, even if it was, in part, accidental.

The Single's Last Days (1985-1993): Singles Proportion Collapses Again to Record Low Before Stabilizing Around All-Time Average (Segment E)

The 1985 season saw the beginning of a decline in the singles proportion that would eventually take it down to 0.485 by 1987. MLB then abolished the high strike,³⁸ and the singles proportion bounced right back up in 1988 before settling down to average 0.503 for the period ending in 1993. Overall, it wasn't a dramatic change, but it was a distinctly lower level than the singles proportion of the second dead ball era and its aftermath.

³⁶James (2001)

³⁷Constantino

³⁸<http://m.mlb.com/glossary/rules/strike-zone>

Predictably, the initial decline in the singles proportion brought with it a resurgence in the power hitting game. Unlike in 1950 and 1969, the 1988 contraction of the strike zone had the effect of erasing a substantial increase in the home run proportion to a then record level in 1987, and then keeping the home run proportion suppressed through the 1993 season. Pitchers were likely working the bottom of the strike zone more intensely once the high strike was abolished.

For the period as a whole, the doubles proportion and home run proportions both bumped up, along with the walks proportion, but offense overall remained around the same 4.3 runs per game achieved during the AstroTurf period of 1977 to 1984. Turf surfaces continued to exert an important influence throughout this period (speed still mattered), but other factors seemed to be coming into play now. We might view this period as an extension of the 1977-84 turf period with a modest increase in slugging (including home runs) and a modest decrease in batting average.

Two gradual changes in the way teams used their pitchers are worth noting here. By the mid-80s, most teams had moved from a 4-man starting pitching rotation to a 5-man rotation.³⁹ In addition, most teams now used closers, dominant relief pitchers who specialized in preserving narrow leads in the ninth inning. But any connection between these changes and the power bump in this period is not immediately apparent.

Another force of gradual change that began in the 1980s was players increasingly devoting their off seasons to strength and conditioning programs rather than supplementing their incomes.⁴⁰ As more and more players did so, and as the programs grew in sophistication, durability and performance improved, and any gains in strength were generally reflected in more extra base hits for batters and more strikeouts for pitchers.

The Steroid Era (1994-2010): The Singles Proportion Topples to New Depths and Doesn't Get Back Up

Looking back at the singles proportion and the home run proportion with a long-term macro perspective from 1920 to the end of the 1993 season, you would say baseball went through a transformative period during the live ball era that was temporarily interrupted by World War II before settling into relative stability marked by short-term peaks and valleys from 1947 to 1993. Sometimes, offense had the upper hand, and other times pitching had the upper hand. Sometimes, offense was dominated by the home run, and other times the power game was balanced by speed and contact hitters. But neither the singles proportion nor the home run proportion strayed very far for very long before turning around and coming back home.

This might have continued indefinitely, but it didn't. Baseball entered uncharted territory in 1994 and stayed there through 2010. It was more than a shift in balance; it was a transformation. And if you measure it by the singles proportion, it lasted the better part of two decades. The singles proportion set or tied a new record low (post-1919) every season from 1994 through 2000, when it bottomed out at 0.457. For the entire period, it averaged 0.472, 29 points lower than the next lowest era, the post-war period.

Of course, it wasn't just the singles proportion that changed so dramatically. Also surpassing their post-1919 record highs (both single-season and era average) were the doubles proportion, home run proportion, strikeouts per game, and the hit by pitch proportion. Run production as a whole in this era was comparable to that in the live ball era. But it was the outsized role of power hitting in generating runs that made this period so unique.

As it was all happening, observers naturally credited the ball itself for the power hitting explosion. Some analysts continue to insist the ball is the only explanation that can plausibly account for such a sudden and large jump. As time passed and further evidence came to light, the more commonly accepted explanation for the power explosion was widespread use of performance enhancing drugs, steroids, by players.⁴¹ Both the ball and steroids may have been factors, and we'll probably never know to what extent each contributed to the transformation of the game beginning in 1993-94.

³⁹James, Bill, "The Three Man Starting Rotation" (Nov 20, 2015). https://www.billjamesonline.com/the_three_man_starting_rotation/

⁴⁰James (2001)

⁴¹Jaffe (2012); Lindbergh, Ben, "How Much of a Role Did Steroids Play in the Steroid Era?" (Sep 28, 2018). <https://www.theringer.com/mlb/2018/9/28/17913536/mark-mcgwire-sammy-sosa-steroid-era-home-run-chase>

Looking at the issue from the perspective of the singles proportion and its inverse, the home run proportion, it wasn't the magnitude of the change that made this period transformative. Nor was it the suddenness of the change. In 1994, the singles proportion experienced a one-year drop of twenty points, from 0.499 to 0.479. That was only the fifth largest drop since 1920. If we compare 1994 to 1992, the singles proportion's two-year drop of 30 points was only the fourth largest drop since 1920. If we sum the singles proportion's two-year drop and the home run proportion's two-year rise for each season, the 1994 combined two-year change of 50 points was only the fifth largest since 1920. It took eight seasons for the singles proportion to reach its nadir, 0.457 in 2000, and another ten seasons for it to slowly climb back to a still low 0.477 in 2010. Similarly, it took nine seasons for the home run proportion to reach its peak of 0.089 in 2001, and another nine seasons for it to slowly crawl back to a still high 0.077 in 2010.

We stated earlier that after the transformative live ball era, "...neither the singles proportion nor the home run proportion strayed very far for very long before turning around and coming back home." In the steroid era, the singles proportion and home run proportion never turned around and came back home. They just hung around in their extreme states for at least 17 seasons until their extreme states became new norms. There was only one way to score runs during this period, and that was to wait for your pitch and swing for the fences. For many batters, the single, the triple, and advancing baserunners to the next base had become mere afterthoughts, the fortuitous byproducts of a failed home run.

With runs per game approaching live ball era levels and fans excited by home run races, did MLB attempt any corrections, as it had throughout the history of the game whenever offense or pitching gained too much of an upper hand? Not much. The strike zone was expanded down to the bottom of the knees in 1996,⁴² but this had little effect. Team owners may have viewed the power explosion as a second Ruthian revolution that would increase the popularity of the game. In any case, they rode the power wave until the stigma associated with performance enhancing drugs became too great. They finally banned steroids before the 2005 season, and increased the length of suspensions before 2006,⁴³ but it would be another five seasons before the singles proportion began behaving differently.

The fact that power hitters remained almost as predominant after steroid use became punishable raises more questions. Why didn't we see a more sudden drop off in power hitting? There was never a straight downhill decline. The most we can really say is that the power numbers stopped climbing in subsequent seasons. But they never returned to pre-steroid era levels. One possibility, confirmed by drug testing, is that some players continued to use steroids.⁴⁴ Another possibility is that the approach to hitting favored by analytics never changed, regardless of steroid use. In other words, even the complete absence of steroids didn't diminish the impact of the then commonly accepted notion that, given all the conditions under which the modern game is played, extra base hits are the most efficient way to score runs.

With regard to "the conditions under which the modern game is played," we can't neglect to mention the effect of a fourth wave of stadiums entering the league in the 1990s. There were eleven to be exact, most of which were new, had natural grass surfaces, and favored hitting. One of them was Coors Field in Denver (1995), the best hitter's park in the history of baseball.⁴⁵ They permitted, if not encouraged, the power explosion.

There may have been other contributing factors that, in combination, had the effect of softening the impact of the steroid ban on power performance. Bill James has cited several. For example, by the 1990s, it was finally acknowledged by everyone in the baseball world that offseason strength training done properly could improve a power hitter's performance. Another such factor: batters, emboldened by strict new rules discouraging brushback pitches and charging the mound to initiate fights, were now willing to crowd the plate in order to hit for power to the opposite field. This was the primary reason the hit by pitch proportion increased so much during this period. Finally, bats were designed with increasingly thinner handles to concentrate more weight in the barrel, increasing the speed of the barrel as it travelled through the strike zone.

⁴²<https://www.mlb.com/official-information/umpires/strike-zone>

⁴³Rymer, Zachary D., "Full Timeline of MLB's Failed Attempts to Rid the Game of PEDs" (June 10, 2013). <https://bleacherreport.com/articles/1667581-full-timeline-of-mlbs FAILED-attempts-to-rid-the-game-of-peds>

⁴⁴Lindbergh

⁴⁵James (2001)

Whatever factors contributed to the power explosion of 1994 to 2010, the widely adopted approach to hitting, wait for your pitch and swing really hard, tended to produce a limited set of results that changed the character of the game: strikeouts, walks, hit batters, home runs, doubles.

Speed, whether on the bases or in the field, didn't matter as much. Sacrifice hits and advancing baserunners didn't matter as much. Nor did defensive skills in general. Stretching doubles into triples, one of the most exciting plays in baseball, didn't matter as much either. Fewer balls were being hit into play, and strategic decision points occurred less frequently. Teams just sat back and waited for the next power hitter in the lineup to drive in runs. Games were longer and moved along more slowly, so a greater proportion of game time contained no action. When the object of the game became to outslug the other team, the effect was like that of an invasive species, crowding out other entertaining aspects of the game. MLB seemed more blind than ever to changes in the character of the game, so long as more runs were being scored and attendance kept increasing.

The Umpire Strikes Back (2011-2015): Conformity to the Rulebook Strike Zone Permits the Singles Proportion One Last Goodbye

In 2011, the singles proportion moved toward pre-steroid era levels not seen for 18 seasons, and remained there for five years. It peaked at 0.496 in 2014 before retreating again. The 2015 season would be the last with a singles proportion above steroid era levels.

The doubles and home run proportions did not change during this period, but the walks proportion plunged to levels not seen since the 1930s, and strikeouts per game set a new all-time record every season. When we've seen this combination of changes earlier in baseball history, it was accompanied by a change in the strike zone.

While there was no *official* change to the strike zone during this period, there is evidence that umpires' enforcement of the strike zone changed after 2009, when MLB began using an improved camera system in all major league stadiums to, among other things, monitor ball-strike calls by umpires. In addition to the technology change, a new umpire evaluation and training system was implemented following collective bargaining between MLB and the umpires' union in 2009. Umpires began receiving reports of their performance after every game. At the same time this was happening, MLB took unusual disciplinary action by firing three umpires because of poor safe-out calls they made during the 2009 postseason.⁴⁶

A 2014 study by Brian Mills examined, among other things, the impact of changes in monitoring, evaluation and training on the performance of MLB umpires from 1988 through 2013. Mills found substantial improvement in umpire performance after 2009, when MLB implemented the new monitoring, training and evaluation system, and took disciplinary action against poorly performing umpires. That is, umpires called balls and strikes in a way that was more consistent with the strike zone defined in the MLB Rulebook. Specifically, they called more strikes near the bottom of the strike zone, and fewer strikes off of the outside edge of the plate. Overall, the accuracy with which strikes were called improved from 0.768 in 2007 to 0.857 in 2013. The accuracy with which balls were called also improved, from 0.874 in 2007 to 0.900 in 2013. Overall accuracy improved from 0.840 in 2007 to 0.886 in 2013.⁴⁷

The net effect, according to Mills, was a large increase in the rate of strikes called by umpires that "likely resulted in a decrease in offense [in 2010-11, runs per game finally started to move down in a significant way from steroid era levels] often attributed to the league's crackdown on the use of performance enhancing drugs.... [A]ny claim of success in PED testing should be considered with some skepticism."⁴⁸ Mills' findings provide a sound explanation for the rise in the singles proportion during this five-year period. When the strike zone expands, whether the result of a change in the rules or a change in umpire performance, batters

⁴⁶Mills, Brian, Expert Workers, Performance Standards, and On-the-Job Training: Evaluating Major League Baseball Umpires (August 27, 2014). Available at SSRN: <https://ssrn.com/abstract=2478447> or <http://dx.doi.org/10.2139/ssrn.2478447>

⁴⁷Mills

⁴⁸Research by Jon Roegele made similar findings and drew similar conclusions using somewhat different methodologies. Roegele, Jon. "The Strike Zone During the PITCHfx Era". The Hardball Times, Jan 30, 2014. <https://tht.fangraphs.com/the-strike-zone-during-the-pitchfx-era/>

are forced to swing at pitches they would otherwise take for balls, and thus make weaker contact. The consequent increase in the singles proportion was even more assured in this case because the strike zone expanded downward, resulting in more ground balls.⁴⁹

Reduced-Drag Ball (2016-2019): The Single Disappears from Offensive Playbooks

The 2011 to 2015 umpire-driven interlude abruptly ended in 2016, when the singles proportion experienced a two-season drop of 30 points to 0.467, comparable to the singles proportions we saw during the steroid era. The home run proportion rocketed past its steroid era levels, rising 22 points over 2015 and 2016 to land at 0.095. Neither proportion stopped there. The singles proportion bested its all-time low the next season, and continued to fall in 2018 and 2019, finally landing at 0.433. Its 63 point fall between 2014 and 2019 was the largest five-year drop in the history of the game. Not to be outdone, the home run proportion experienced a 40 point rise during the same five-year period, finishing the 2019 season at a remarkable 0.113. Baseball was truly in uncharted territory now.

This time, there was hard evidence of the primary cause, thanks to newly available Statcast data and more advanced laboratory testing. An exhaustive report commissioned by the league in 2017 and published in May 2018 by a group of scientists and mathematicians pointed to the most notorious catalyst for change throughout baseball history, the baseball.⁵⁰ The report found that “changes in the aerodynamic properties of the baseball” resulted in batted balls having greater carry. That is, balls were travelling farther because of decreased drag.⁵¹ The researchers continued their work through the rest of 2018 and 2019, in an attempt to definitively establish which physical properties changed to reduce air resistance.

The report was monumental not only for identifying the primary cause of the latest power explosion (home runs were now beyond steroid era levels), but also for demonstrating that the slightest change in the baseball manufacturing process has the potential to change, intentionally or not, the character of the game. Suddenly, all those who had pointed to the ball to explain previous evolutions throughout baseball history gained a bit more credibility.

In December 2019, another report released by MLB made it apparent the scientists and statisticians were still struggling to identify all the factors affecting drag.⁵² However, by using more sophisticated lab equipment, they were able to determine that differences in seam height accounted for about 35% of the change in drag.⁵³ According to one of the scientists on the committee, “the change in seam height of a fraction of the thickness of a sheet of paper would give you a measurable effect in the change in the drag.” Reduced drag, or increased carry, was again identified as the primary cause of the increase in home runs, accounting for 60%. Launch conditions, that is, more players adapting their swing to hit the ball at higher launch angles, accounted for the other 40%.⁵⁴

No mention was made of independent research showing that the leather cover of the ball was now smoother.⁵⁵ Instead, the report found that surface roughness wasn’t correlated with drag coefficient.⁵⁶ The report also found no evidence that changes in the ball were the intentional result of changes in the manufacturing process. Any variation in aerodynamic drag, both between balls and from season to season, was merely a

⁴⁹Lemire, Joe, “Why Baseball’s Strike Zone Is Changing” (May 23, 2016). <https://www.vocativ.com/321368/why-baseballs-strike-zone-is-changing/index.html>

⁵⁰Albert et al., Report of the Committee Studying Home Run Rates in Major League Baseball (May 24, 2018). http://www.mlb.com/documents/7/9/4/278128794/Full_Report_of_the_Committee_Studying_Home_Run_Rates_in_Major_League_Baseball_052418.pdf

⁵¹Lemire, Joe, “MLB Research Determines Reduced Drag Boosted Home Run Surge” (May 25, 2018). <https://www.sporttechie.com/mlb-baseball-research-home-run-drag-aerodynamics/>

⁵²Kram, Zach, “How the Baseball Became an Unreliable Narrator” (Dec 16, 2019). <https://www.theringer.com/year-in-review/2019/12/16/21023481/juiced-dejuiced-ball-home-runs-investigation>

⁵³Glaser, Kyle, “Study Concludes Seam Height Changes Contributed To 2019 Home Run Spike” (Dec 11, 2019). <https://www.baseballamerica.com/stories/study-concludes-seam-height-changes-contributed-to-2019-home-run-spike/>

⁵⁴Jaffe, Jay, “The Home Run Committee’s Latest Report Isn’t the Final Word on Juiced Baseballs” (Dec 12, 2019). <https://blogs.fangraphs.com/the-home-run-committees-latest-report-isnt-the-final-word-on-juiced-baseballs/>

⁵⁵Jaffe (2019)

⁵⁶Passan, Jeff, “Uptick in home runs attributed to seam heights and batting techniques, not juiced balls” (Dec 11, 2019). https://www.espn.com/mlb/story/_/id/28273986/uptick-home-runs-attributed-seam-heights-batting-techniques-not-juiced-balls

consequence of “using a product made of natural materials and constructed in part by hand.” MLB promised tighter quality control to minimize ball-to-ball variation, but sounded no alarm bells.⁵⁷

With runs per game once again approaching steroid era levels, and the current MLB commissioner stating that fans like home runs and strike outs, MLB appears resigned to the bumpy ride that comes with a baseball that can vary significantly from pitch to pitch, month to month, and season to season. Will the singles and home run proportions ever reverse course again? Probably. Will they ever return to pre-steroid era levels? It sure seems unlikely, at least until MLB fully implements an automated ball-strike system. To understand the implications of that, we need only look back at the 2011-15 period when human umpires enforced the strike zone defined in the MLB Rulebook. In the meantime, Major League Baseball appears willing to do whatever it believes necessary to stay relevant in a world with limitless entertainment options - even if it means changing the game.

For now, we have more doubles, home runs, strikeouts, walks, hit batters, and runs. And we have fewer singles, triples, stolen bases, and balls in play. Making contact and using speed, either on the bases or in the field, are no longer highly valued. In short, there's a lot less action on the field.

.....

The days when teams willingly sacrificed power for defense and speed in their middle infielders and outfielders seem a distant memory now. One of the most striking things when watching old films of games from the 1960s and 70s is how slender, no, how skinny so many players appear because we're now so accustomed to seeing large muscles on professional athletes. So we'll end with a fun leader board of sorts to honor the leather-flashing, sometimes speedy, mostly normal-bodied contact hitter.

The leader board includes the top 50 all-time (post dead ball era) career leaders (minimum 1500 singles) in singles proportion, the players whose singles comprised the largest percentage of their on-base events. These top 50, 18 of whom are in the Hall of Fame, played on 286 All-Star teams and 48 World Series teams. They earned 11 Most Valuable Player awards, six Rookie of the Year awards, 38 Batting Titles, 41 Silver Slugger Awards and 110 Gold Glove Awards. Twenty of them have at least 250 career stolen bases. Only seven had more than 200 career home runs, none among the top 30 players.

With the retirement of Ichiro Suzuki in 2019, there are no active players among our top 50 leaders. It makes one wonder whether this leader board will look any different 20 years from now. Perhaps it's fixed, a done deal, and the names are already being chiseled into stone, like a monument honoring a bygone era when a game called baseball was played in a very different way by very different people.

(Note that the careers of some players on the leader board predate at least some of the awards. The MVP Award began in 1931, All-Stars were selected beginning in 1933, Rookie of the Year began in 1947, Gold Glove in 1957, and Silver Slugger in 1980. Abbreviations appearing in the Honors column of the Leader Board signify the following: HOF is Hall of Fame, MVP is Most Valuable Player, AS is All-Star, WS is World Series, RoY is Rookie of the Year, GG is Gold Glove, SS is Silver Slugger, BT is Batting Title, and TSNPoY is The Sporting News Player of the Year, which began in 1936.)

```
# Import a data file into an R data frame.
```

```
singles_props_leaderboard <-  
  read_csv("https://raw.githubusercontent.com/jfmusso/HarvardX/master/SinglesProp_Leaderboard2.csv",  
    col_types = cols(  
      Player = col_character(),  
      Pos = col_character(),  
      From = col_number(),  
      To = col_number(),  
      Prop = col_double(),
```

⁵⁷Glaser

```

`1B` = col_integer(),
`1B Rk` = col_integer(),
Honors = col_character()
)

)

class(singles_props_leaderboard[])
}

## [1] "tbl_df"     "tbl"        "data.frame"

knitr::kable(singles_props_leaderboard,
             caption = "Singles Proportion: Top 50 Career Leaders
(min. 1500 singles, post-1919)")

```

Table 57: Singles Proportion: Top 50 Career Leaders (min. 1500 singles, post-1919)

Player	Pos	From	To	Prop	1B	Honors
Lloyd Waner	CF	1927	1945	0.700	2033	HoF, AS
Maury Wills	SS	1959	1972	0.691	1866	MVP, 7xAS, 3xWS, 2xGG, ASMVP, TSN PoY
Larry Bowa	SS	1970	1985	0.677	1815	5xAS, WS, 2xGG
Juan Pierre	CF	2000	2013	0.665	1850	WS
Ichiro Suzuki	RF	2001	2019	0.663	2514	MVP, RoY, 10xAS, 10xGG, 3xSS, 2xBT, ASMVP
Tommy Davis	LF	1959	1976	0.655	1661	3xAS, WS, 2xBT
Doc Cramer	CF	1929	1948	0.652	2163	5xAS, WS
Willie Wilson	CF	1976	1994	0.645	1738	2xAS, WS, GG, 2xSS, BT
Garry Templeton	SS	1976	1991	0.642	1591	3xAS, 2xSS
Willie McGee	CF	1982	1999	0.637	1731	MVP, 4xAS, WS, 3xGG, SS, 2xBT
George Sisler	1B	1920	1930	0.637	1522	HoF, MVP, 2xBT
Dick Groat	SS	1952	1967	0.632	1680	MVP, 8xAS, 2xWS, BT
Bill Russell	SS	1969	1986	0.626	1530	3xAS, WS
Pie Traynor	3B	1920	1937	0.625	1823	HoF, 2xAS, WS
Bill Buckner	1B	1969	1990	0.622	1994	AS, BT
Steve Sax	2B	1981	1994	0.621	1570	RoY, 5xAS, 2xWS, SS
Placido Polanco	2B	1998	2013	0.621	1658	2xAS, 3xGG, SS, ALCS MVP
Nellie Fox	2B	1947	1965	0.613	2161	HoF, MVP, 15xAS, 3xGG
Bill Mazeroski	2B	1956	1972	0.613	1522	HoF, 10xAS, 2xWS, 8xGG, TSN PoY
Luis Aparicio	SS	1956	1973	0.613	2108	HoF, RoY, 13xAS, WS, 9xGG
Willie Davis	CF	1960	1979	0.609	1846	2xAS, 2xWS, 3xGG
Mark Grudzielanek	2B	1995	2010	0.607	1523	AS, GG
Red Schoendienst	2B	1945	1963	0.605	1860	HoF, 10xAS, 2xWS
Bert Campaneris	SS	1964	1983	0.604	1771	6xAS, 3xWS
Tony Gwynn	RF	1982	2001	0.601	2378	HoF, 15xAS, 5xGG, 7xSS, 8xBT
Sam Rice	RF	1920	1934	0.601	1925	HoF, WS
Don Kessinger	SS	1964	1979	0.600	1583	6xAS, 2xGG
Al Dark	SS	1946	1960	0.596	1533	RoY, 3xAS, WS
Luis Castillo	2B	1996	2010	0.595	1608	3xAS, WS, 3xGG
Frankie Frisch	2B	1920	1937	0.594	2135	HoF, MVP, 3xAS, 4xWS
Steve Garvey	1B	1969	1987	0.593	1844	MVP, 10xAS, WS, 4xGG, 2xNLCS MVP, 2xASMVP
Charlie Grimm	1B	1920	1936	0.590	1682	NA
Harvey Kuenn	SS	1952	1966	0.590	1593	RoY, 10xAS, BT

Player	Pos	From	To	Prop	1B	Honors
Roberto Clemente	RF	1955	1972	0.589	2154	HoF, MVP, 15xAS, 2xWS, 12xGG, WS MVP, 4xBT
Rod Carew	1B	1967	1985	0.587	2404	HoF, MVP, RoY, 18xAS, 7xBT, TSN PoY
Lou Brock	LF	1961	1979	0.586	2247	HoF, 6xAS, 2xWS, TSN PoY
Dave Concepcion	SS	1970	1988	0.580	1788	9xAS, 2xWS, 5xGG, 2xSS, ASMVP
Kirby Puckett	CF	1984	1995	0.579	1626	HoF, 10xAS, 2xWS, 6xGG, 6xSS, BT, ALCS MVP, ASMVP
Carney Lansford	3B	1978	1992	0.576	1551	AS, WS, SS, BT
Tony Taylor	2B	1958	1976	0.574	1548	2xAS
Omar Vizquel	SS	1989	2012	0.573	2264	3xAS, 11xGG
Al Oliver	CF	1968	1985	0.571	1918	7xAS, WS, 3xSS, BT
Heinie Manush	LF	1923	1939	0.569	1763	HoF, AS, BT
George Kell	3B	1943	1957	0.568	1541	HoF, 10xAS, BT
Michael Young	SS	2000	2013	0.568	1689	7xAS, GG, BT, ASMVP
Garret Anderson	LF	1994	2010	0.568	1684	3xAS, WS, 2xSS, ASMVP
Bill Terry	1B	1923	1936	0.567	1554	HoF, 3xAS, WS, BT
Edgar Renteria	SS	1996	2011	0.560	1722	5xAS, 2xWS, 2xGG, 3xSS, WS MVP
Ivan Rodriguez	C	1991	2011	0.559	1910	HoF, MVP, 14xAS, WS, 13xGG, 7xSS, NLCS MVP
Vada Pinson	CF	1958	1975	0.558	1889	4xAS, GG