Final

Cesar F. Pardede

15 MAY 2020

Does the count affect the probability of a strike being called?

Introduction

In this paper, we try to answer the question in baseball of whether umpires are more likely to call a fringe pitch a strike if there are already three balls on the count.

First, we have to define a fringe pitch. A fringe pitch is a pitch that is on the border of the strike zone that could be called a ball or a strike ... so we have to define the strike zone to define a fringe pitch. Once we identify our fringe pitches, we can conduct our analysis. We start with a few quick contingency tables to see if anything becomes immediately apparent. Then we continue our analysis by creating a statistical model to either validate our hunches from the contingency tables, or to gain a new perspective. We consider both random forest and logistic regression models, then move forward with the better performing model as measured by the area under each ROC curve (AUC). After we have our model, we use it to predict probabilities of a fringe pitch being called a strike under three conditions:

- 3 balls, any number of strikes
- 3 balls, 2 strikes
- <3 balls, any number of strikes.

Next we create histograms to plot the density of probabilities of a strike being called on a fringe pitch, and use kernel density estimation to turn our results into densities. We conclude our analysis by performing 2-sample Kolmogorov-Smirnov tests (KS test) to determine whether our pitches were drawn from different distributions, to indicate whether umpires are more likely to call a fringe pitch a strike for various counts.

Defining a fringe pitch

If we plot all of the data points, we get something that looks like Figure 1 (top). We can clearly see some kind of boundary region where strikes end and balls begin. We could define the strike zone, by drawing a boundary where we believe most (how many?) of the strikes occur. Instead we allow the strike zone to be

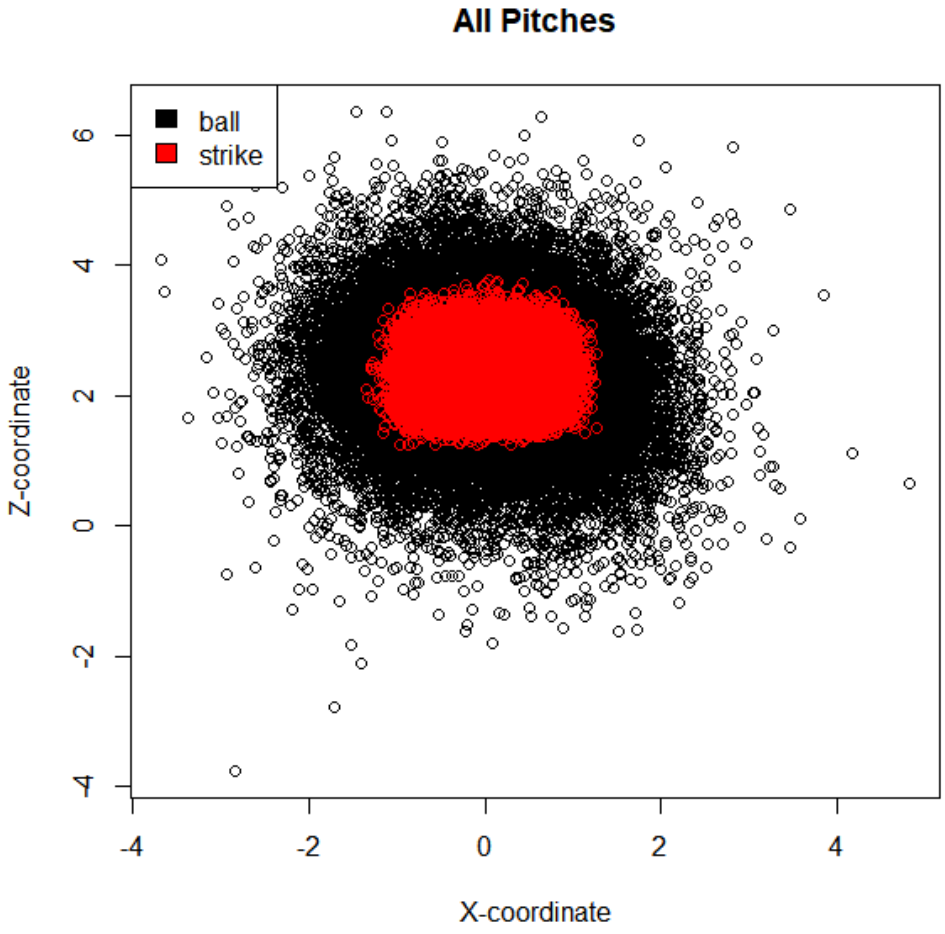

Figure 1: Location of all pitches

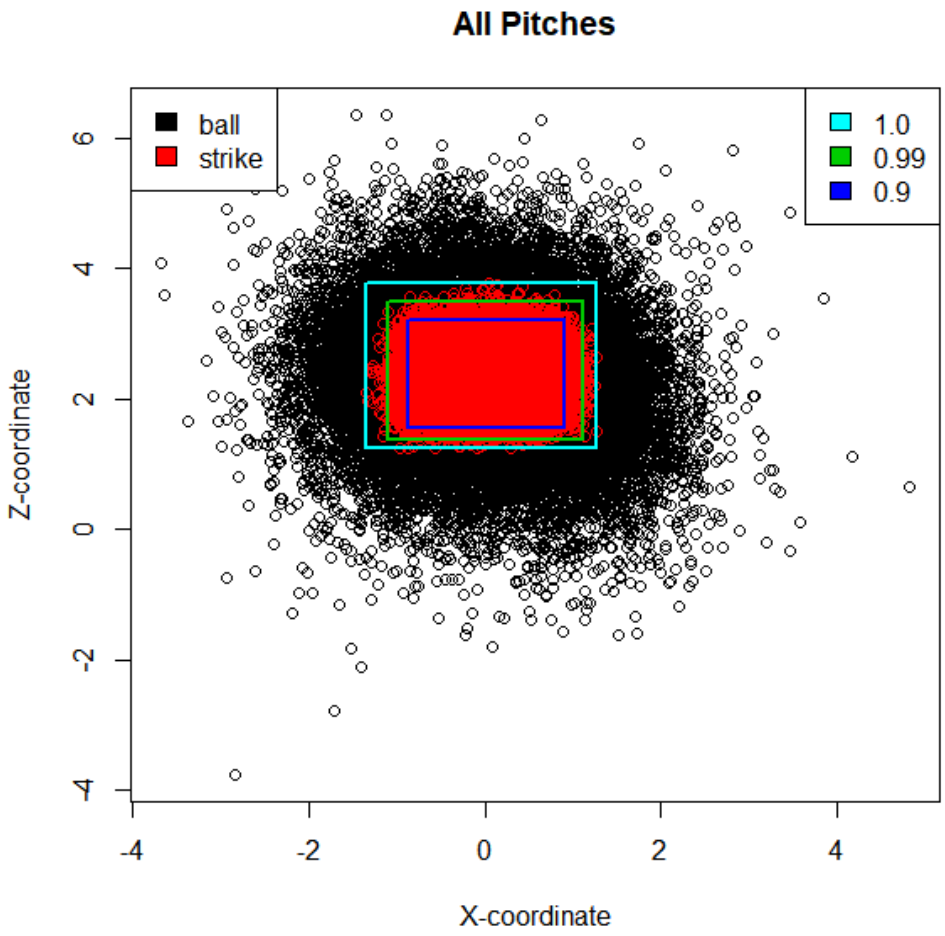

Figure 2: All pitches with strike percentage zones

defined by the data.

To define our strike zone, we start by sorting the x-coordinates and the z-coordinates that resulted in a strike. Then, we choose a percentage of strikes that we want contained within our strike zone, and select the indices of our coordinates that would result in covering that percentage of strikes. In our case, we decide to find our 99% and 90% strike zones. Note that we're not actually covering 99% or 90% of strikes in each zone, but rather 99% and 90% of strikes within each coordinate axis. So our 99% strike zone really covers ∼98% =.99*.99*100% of strikes, and our 90% strike zone covers ∼81%=.90*.90*100% of strikes. This creates the boundaries found in Figure 2.

We can zoom in on this figure and only consider the data points falling within the 100% strike zone in Figure 3. Since there are no strikes outside this region, it's unlikely pitches outside this region would be considered fringe pitches. We can consider fringe pitches to be any pitches intersecting any two strike percentage zones which contain pitches that were called balls (e.g. the 1% - 2% strike zone probably does not contain any balls and would not be considered a fringe region). For this paper, we define our fringe pitches as those pitches existing between the 90% and 100% strike zones, which will be the data set we work with for the remainder of this paper. It is visualized in Figure 4.

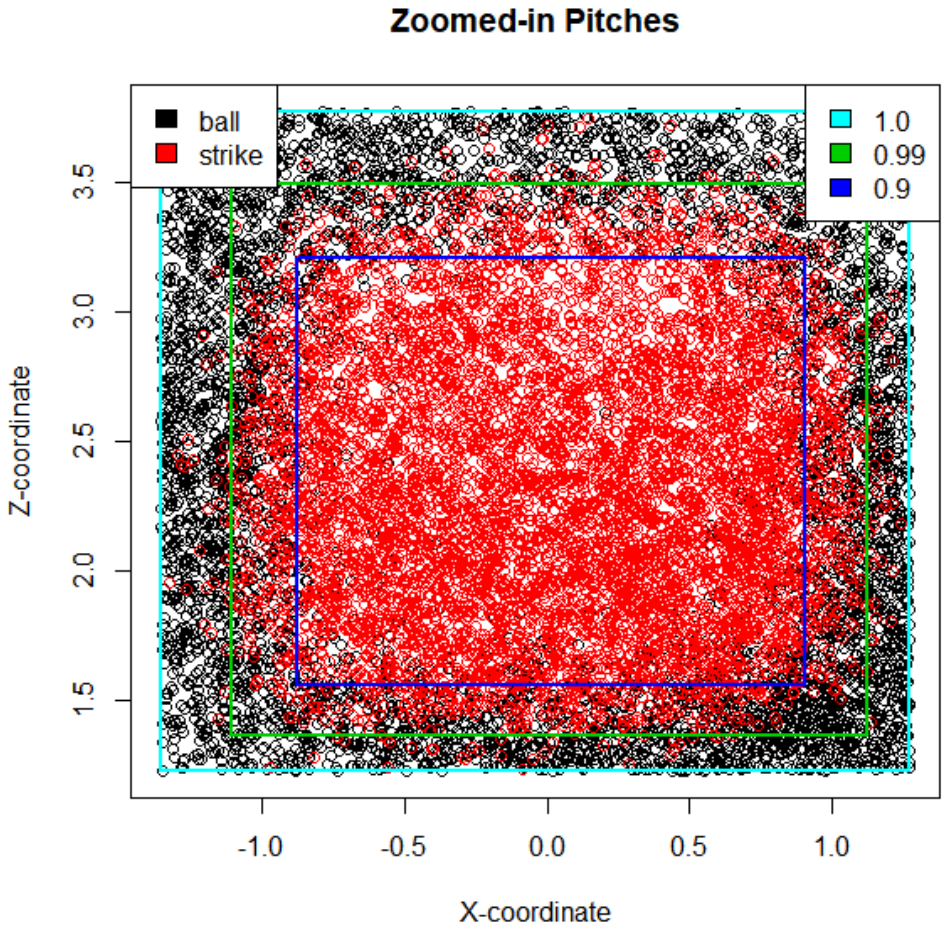

Figure 3: Pitches within 100% strike zone

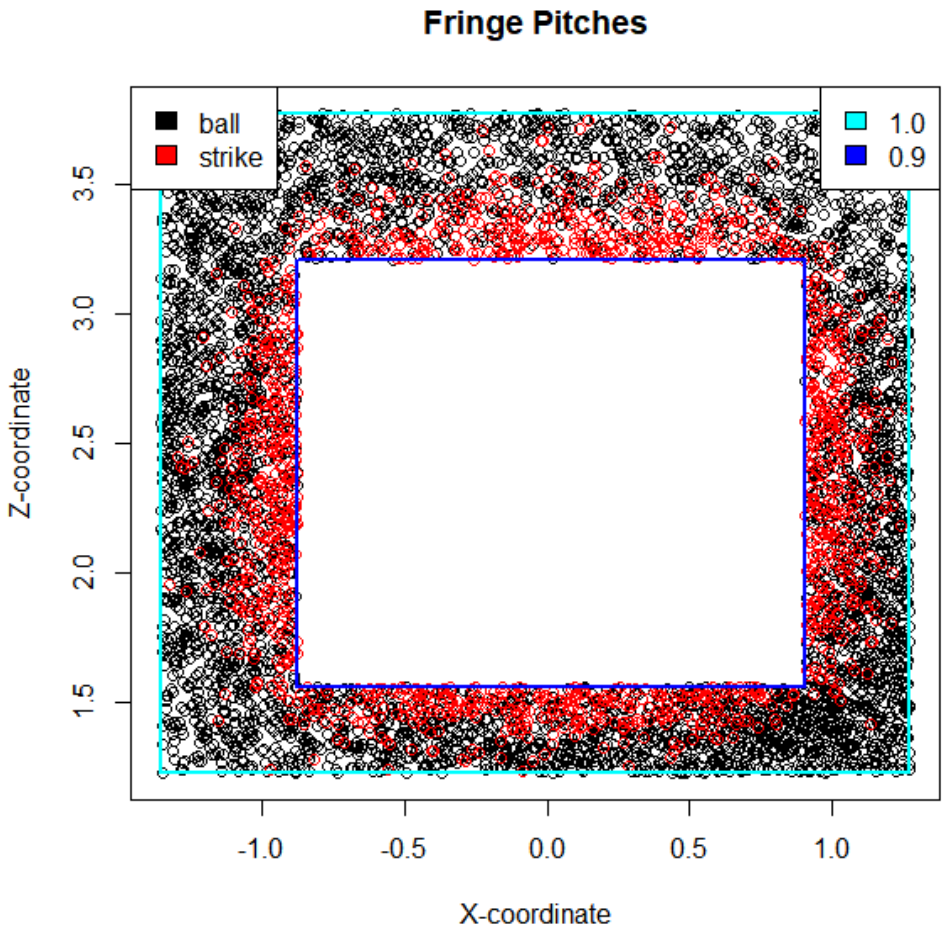

Figure 4: Fringe Pitches

We quickly create proportional contingency tables of our fringe pitches to see if there is a relationship between the count and whether a strike is called under our three conditions. The first table shows the percentage of each call for each number of balls, the second table shows the percentage of each call at each count, and the third table shows the the percentage of each call given 3 balls and 0, 1, or 2 strikes. The contingency tables faintly suggest that a fringe pitch is more likely to be called a strike when the count is lower.

```
                     0         1         2         3
ball           0.7766867 0.7959427 0.7929465 0.7586207
called_strike  0.2233133 0.2040573 0.2070535 0.2413793

                     0         1         2         3         4         5
ball           0.7464789 0.7967699 0.7994350 0.7997382 0.8244111 0.8316327
called_strike  0.2535211 0.2032301 0.2005650 0.2002618 0.1755889 0.1683673

                     3         4         5
ball           0.6302521 0.7651007 0.8316327
called_strike  0.3697479 0.2348993 0.1683673
```

Selecting a statistical model

Now that we have identified our fringe pitches, we can start to create our statistical models. We choose to create random forests and logistic regression models with 5-fold cross validation; using 'balls', 'strikes', 'plate_x' (x-coordinate of the pitch), 'plate_z' (z-coordinate of the pitch) as our explanatory variables; and 'description' (ball or strike) as our response variable. We considered adaptive boosting models, but found they took much longer to generate than random forests without offering much improvement in performance. We measure the performance of each model by comparing the AUC of the ROC curves generated by each model. AUC is our performance metric which is a summary of our models' classifcation accuracy. A higher AUC means higher accuracy, which means better model performance. We plot a ROC curve for our last random forest model as a demonstration. The AUC is the area under the ROC curve, the AUC for each cross-validated model is given in the table below.

```
> # measure performance with auc of roc curves
> auc_rf
                   [,1]
[1,] 0.8159172
[2,] 0.7990638
[3,] 0.7962198
[4,] 0.7929020
[5,] 0.7928791
> auc_lr
                   [,1]
[1,] 0.5913481
[2,] 0.5540172
[3,] 0.5819723
[4,] 0.5952638
[5,] 0.5964763
```

After parameter tuning to find the best performing parameters, we see that random forests outperform

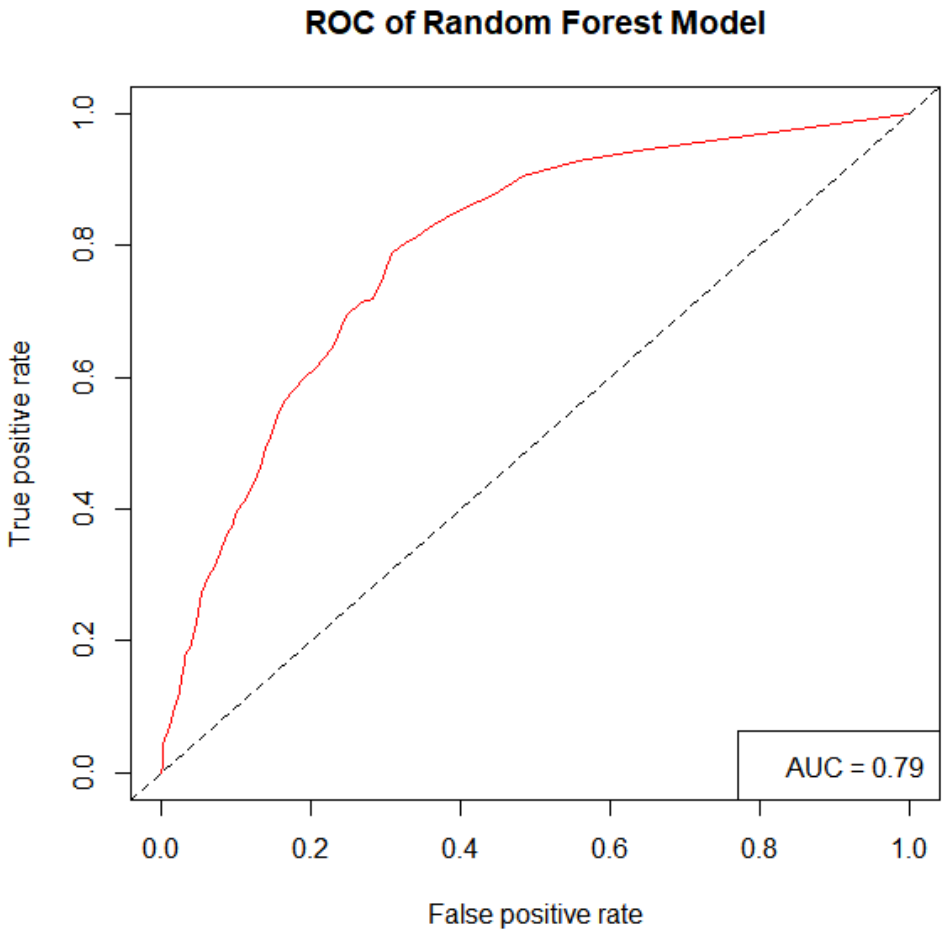

Figure 5: ROC Curve and AUC of a Random Forest Model

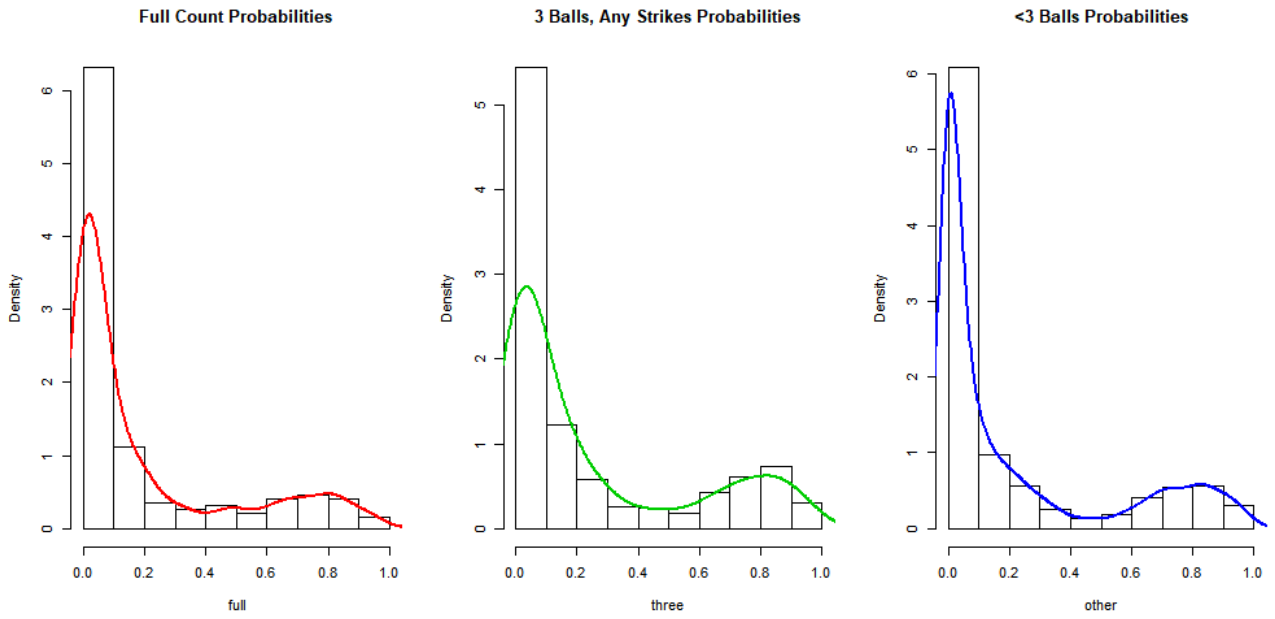

Figure 6: Histograms and Densities of Each Condition

logistic regression at each fold, and that the performance of each model is relatively consistent, so we proceed our analysis with our random forest model.

Does the count affect the probability of a strike being called?

We want to determine whether the probability of a strike differs in three cases: 1) when the count is 3 balls and any number of strikes, 2) when it's a full count (3 balls, 2 strikes), and 3) when there are less than 3 balls. We use our model to predict the probablities of a strike being called for each fringe pitch which falls under each condition, and plot histograms of our probabilites for each condition.

Then, we can come up with a density for each condition using the built-in density() function to perform kernel density estimation; we overlay the generated densities over the histograms in Figure 6, and compare just the densities in Figure 7. We also take our densities and generate cumulative densities for each condition, and plot them in Figure 8. It looks like there *might* be a difference in the probablity of calling a strike under our three conditions, but is the difference *significant*?

To answer this question, we can perform a 2-sample KS tests on our empirical cumulative density functions. Note that we don't use our theoretical cumulative densities, those were plotted for visualization and to give us a hint whether there there are any significant differences in the distribution of probabilities, but we wouldn't use them for a KS test. If we created different distributions for each condition, and then ran a test for whether each condition has a different distribution, our test would be more likely to say that each condition does have a different distribution than if we were to sample from data instead.
To perform a KS test, we need to use our data and come up with an empirical cumulative density function. We can do this easily with the ecdf() function, and we plot the resulting CDF's in Figure 9.

The results of our KS tests are found in the table below. Notice that the p-values for the 2-sample tests between pitches with a full count and three balls, and pitches with a full count and other (less than 3 balls) are larger than traditional significance values. This indicates the probability distribution of calling a strike under these two conditions are similar. However, the 2-sample KS test between pitches with 3 balls

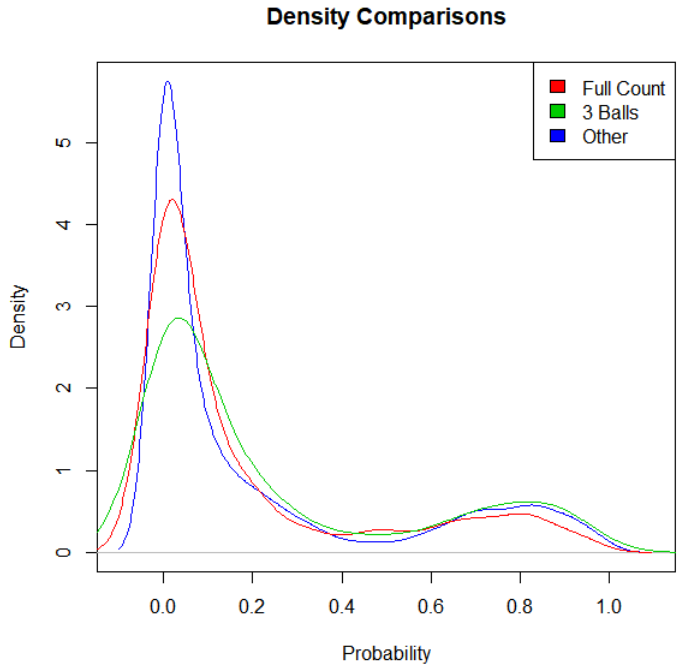

Figure 7: Density Comparisons

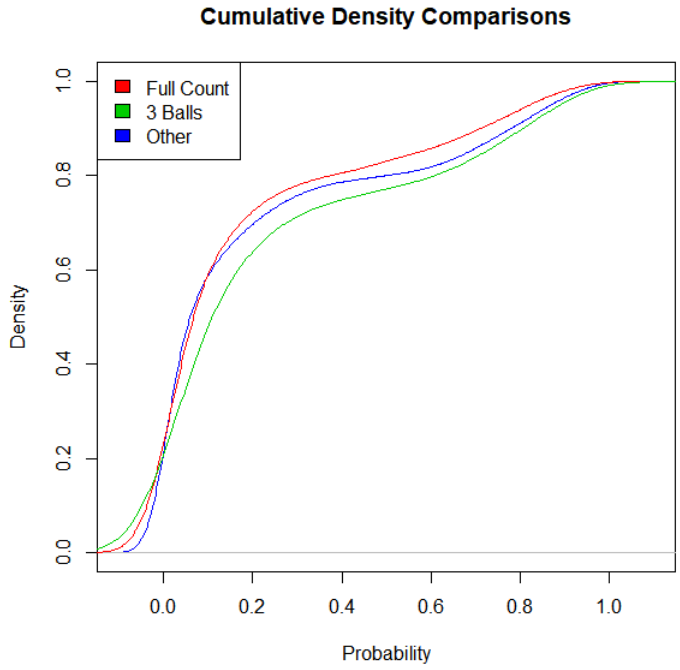

Figure 8: Cumulative Density Comparisons

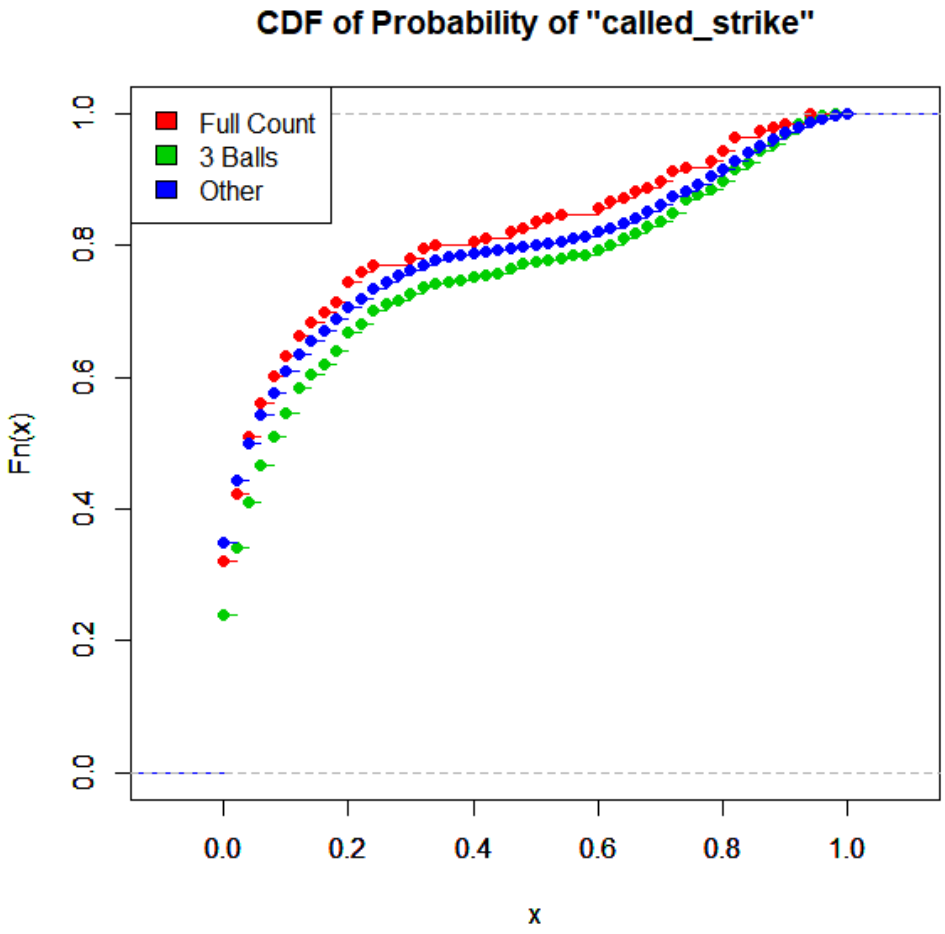

Figure 9: Empirical Cumulative Density

and pitches with less than 3 balls is very small, indicating the probability of calling a strike under these two conditions come from different distributions.

```
> # 2-sample ks-test and p-val on data
> ks.test(full, three)

Two-sample Kolmogorov-Smirnov test

data:  full and three
D = 0.098566, p-value = 0.1374
alternative hypothesis: two-sided

> ks.test(full, other)

Two-sample Kolmogorov-Smirnov test

data:  full and other
D = 0.041542, p-value = 0.8992
alternative hypothesis: two-sided

> ks.test(three, other)

Two-sample Kolmogorov-Smirnov test

data:  three and other
D = 0.11039, p-value = 5.737e-05
alternative hypothesis: two-sided
```

We believe the reason is the varying sample sizes for each condition (note: length(three) == 464, length(full) == 196, length(other) == 5712). Lets try running the KS tests again, but, this time, sample randomly from each category for the minimum number of samples (i.e. sample 196 pitches from each pitch condition because there are only 196 pitches in the full count condition). The results of our KS tests on the subsampled data are below, and we plot the sample-size-adjusted empirical CDF's in Figure 10:

```
# 2-sample ks-test and p-val on subsamples
> ks.test(full, three_t)

Two-sample Kolmogorov-Smirnov test

data:  full and three_t
D = 0.071429, p-value = 0.6994
alternative hypothesis: two-sided

> ks.test(full, other_t)

Two-sample Kolmogorov-Smirnov test

data:  full and other_t
D = 0.071429, p-value = 0.6994
alternative hypothesis: two-sided
```

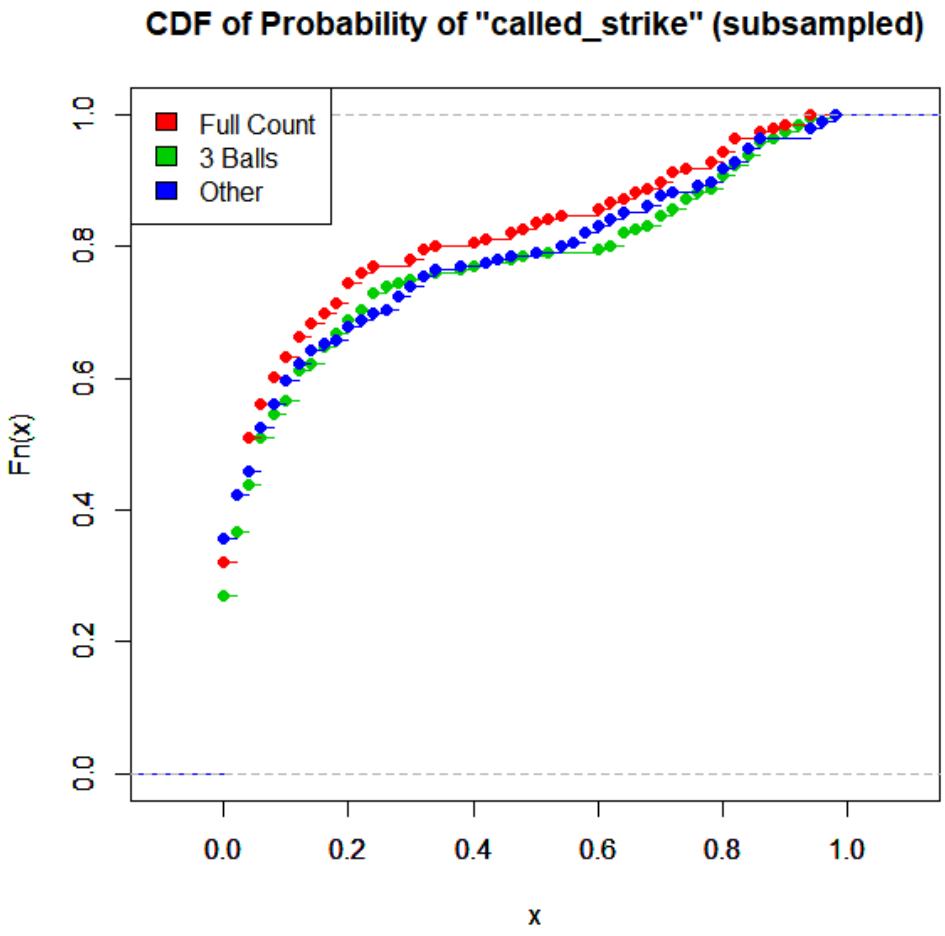

Figure 10: Empirical Cumulative Density (adjusted for sample size)

```
> ks.test(three_t, other_t)

Two-sample Kolmogorov-Smirnov test

data:  three_t and other_t
D = 0.086735, p-value = 0.4523
alternative hypothesis: two-sided
```

Now, after adjusting for sample size, we see that it's unlikely for each pitch condition to have different distributions.

Conclusion and Remarks

Our analysis shows that the count probably does not impact the probability of a strike being called on a fringe pitch. There might be a slight chance that a higher count may influence an umpire to call a strike on a fringe pitch, but it is not significant. Now we take a moment to note the limitations and shortcomings of our analysis:

Consider how we defined a fringe pitch in the first place. We decided a fringe pitch was a pitch landing between the 100% and 90% strike zones. Had we chosen different strike zones, or a different method entirely of determining a fringe pitch, the rest of our analysis could have been very different. While we did allow the strike zones to emerge naturally from the data, determining which pitches were considered on the fringe depended on the percentages we chose. I would try to improve this by finding a way for the fringe region to emerge naturally from the data (as much as possible), probably by finding determining regions that have a 50/50 mixture of balls and strikes (or some other percentage).

Recall that our the AUC for our random forest model was around 80%. While this isn't bad, it's not great, and the accuracy of your predictions will depend on the performance of this model. A more accurate prediction model may yield different conclusions, and confidence in your conclusions partly depends on the power of your model.

Another issue came up when we conducted our KS test the first time. We saw that as the counts got higher, the sample sizes got smaller. We had to take subsamples from two of our categories to even out the sample sizes and not allow a set of pitches to dominate the KS test.

Appendix: Pardede536_Final.R

```
# Cesar F. Pardede
# Math 536
# Final
# 15 MAY 2020

# read in data and extract only columns we think are related to the
   problem
data = read.csv('F:/Documents/CSU Fullerton/Spring 2020/Math 536/Final/
   baseball (1).csv')[c('description', 'balls', 'strikes', 'plate_x', '
   plate_z')]
# dim(data) == c(20528, 5)

# plot all points
plot(data$plate_x[data$description == 'ball'], data$plate_z[
   data$description == 'ball'], col=1, main='All Pitches', xlab='X-
   coordinate', ylab='Z-coordinate')
points(data$plate_x[data$description == 'called_strike'], data$plate_z[
   data$description == 'called_strike'], col=2)
legend('topleft', c('ball', 'strike'), fill = c(1,2))

# sort x and z values that resulted in a strike
x_sort_strike = sort(data$plate_x[data$description == 'called_strike'])
z_sort_strike = sort(data$plate_z[data$description == 'called_strike'])
xlen = length(x_sort_strike)
zlen = length(z_sort_strike)

# function to plot "percent zones"
zone_plotter <- function(x1, x2, y1, y2, color){
        lines(c(x1, x1), c(y1, y2), col=color, lwd=2)
        lines(c(x1, x2), c(y2, y2), col=color, lwd=2)
        lines(c(x2, x2), c(y2, y1), col=color, lwd=2)
        lines(c(x2, x1), c(y1, y1), col=color, lwd=2)
}

a1 = 0.99; a2 = 0.90
# 99%*99% of strikes
zone_plotter(x_sort_strike[(1-a1)/2*xlen], x_sort_strike[(1-(1-a1)/2)*
   xlen], z_sort_strike[(1-a1)/2*zlen], z_sort_strike[(1-(1-a1)/2)*zlen],
    3)
# 90%*90% of strikes
zone_plotter(x_sort_strike[(1-a2)/2*xlen], x_sort_strike[(1-(1-a2)/2)*
   xlen], z_sort_strike[(1-a2)/2*zlen], z_sort_strike[(1-(1-a2)/2)*zlen],
    4)
# 100% of strikes
zone_plotter(x_sort_strike[1], x_sort_strike[xlen], z_sort_strike[1],
   z_sort_strike[zlen], 5)
legend('topright', c("1.0", as.character(a1), as.character(a2)), fill = c
```

```
   (5, 3, 4))

# Lets zoom in
zoomed_data = data[which(data$plate_x > x_sort_strike[1] & data$plate_x <
    x_sort_strike[xlen] & data$plate_z > z_sort_strike[1] & data$plate_z
   < z_sort_strike[zlen]),]
plot(zoomed_data$plate_x[zoomed_data$description == 'ball'],
   zoomed_data$plate_z[zoomed_data$description == 'ball'], col=1, main='
   Zoomed-in Pitches', xlab='X-coordinate', ylab='Z-coordinate')
points(zoomed_data$plate_x[zoomed_data$description == 'called_strike'],
   zoomed_data$plate_z[zoomed_data$description == 'called_strike'], col
   =2)
zone_plotter(x_sort_strike[(1-a1)/2*xlen], x_sort_strike[(1-(1-a1)/2)*
   xlen], z_sort_strike[(1-a1)/2*zlen], z_sort_strike[(1-(1-a1)/2)*zlen],
    3)
zone_plotter(x_sort_strike[(1-a2)/2*xlen], x_sort_strike[(1-(1-a2)/2)*
   xlen], z_sort_strike[(1-a2)/2*zlen], z_sort_strike[(1-(1-a2)/2)*zlen],
    4)
zone_plotter(x_sort_strike[1], x_sort_strike[xlen], z_sort_strike[1],
   z_sort_strike[zlen], 5)
legend('topleft', c('ball', 'strike'), fill = c(1,2))
legend('topright', c("1.0", as.character(a1), as.character(a2)), fill = c
   (5, 3, 4))

# Define fringe pitches as those pitches within the 100% and 90%*90%
   strike zone
fringe_pitches = zoomed_data[-which(zoomed_data$plate_x>x_sort_strike[(1-
   a2)/2*xlen] & zoomed_data$plate_x<x_sort_strike[(1-(1-a2)/2)*xlen] &
zoomed_data$plate_z>z_sort_strike[(1-a2)/2*zlen] & zoomed_data$plate_z<
   z_sort_strike[(1-(1-a2)/2)*zlen]),]

fringe_strikes = fringe_pitches[fringe_pitches$description == '
   called_strike',]
fringe_balls = fringe_pitches[fringe_pitches$description == 'ball',]
plot(fringe_balls$plate_x, fringe_balls$plate_z, main='Fringe Pitches',
   xlab='X-coordinate', ylab='Z-coordinate')
points(fringe_strikes$plate_x, fringe_strikes$plate_z, col=2)
zone_plotter(x_sort_strike[(1-a2)/2*xlen], x_sort_strike[(1-(1-a2)/2)*
   xlen], z_sort_strike[(1-a2)/2*zlen], z_sort_strike[(1-(1-a2)/2)*zlen],
    4)
zone_plotter(x_sort_strike[1], x_sort_strike[xlen], z_sort_strike[1],
   z_sort_strike[zlen], 5)
legend('topleft', c('ball', 'strike'), fill = c(1,2))
legend('topright', c("1.0", as.character(a2)), fill = c(5,4))

# Now we do analysis on the fringe pitches only
# dim(fringe_pitches) == c(6176, 5) # -> 6176/20528 pitches fall into
   fringe category
# quick contingency tables faintly suggest strikes are more likely to be
   called when the count is lower
```

```
prop.table(table(fringe_pitches$description, fringe_pitches$balls),
   margin=2)
prop.table(table(fringe_pitches$description, fringe_pitches$balls+
   fringe_pitches$strikes), margin=2)
prop.table(table(fringe_pitches[fringe_pitches$balls==3,'description'],
   fringe_pitches[fringe_pitches$balls==3, 'balls']+fringe_pitches[
   fringe_pitches$balls==3, 'strikes']), margin=2)

# can we make a model?
# Lets do a random forest model - adaptive boosting takes too long, and
# format data
desc = as.factor(fringe_pitches$description)
balls = as.numeric(fringe_pitches$balls)
strikes = as.numeric(fringe_pitches$strikes)
xplate = as.numeric(fringe_pitches$plate_x)
zplate = as.numeric(fringe_pitches$plate_z)
fringe_df = data.frame(balls, strikes, xplate, zplate, desc)

library(randomForest)
library(ROCR)
library(pROC)

# cross-validation of models
k = 5
d = dim(fringe_df)
d1 = floor(d[1]/k)*k
indexmat = matrix(sample(1:d1, d1), 5)
auc_rf = auc_lr = matrix(NA, k, 1)
for (i in 1:k){
        index = indexmat[i,]
        train = fringe_df[-index,]
        test = fringe_df[index,]

        model_rf = randomForest(desc~balls+strikes+xplate+zplate,
        data = train, mtry=4, ntree=50, # not much improvement after 50
           trees
        control=rpart.control(minsplit=2,cp=10e-2))
        model_lr = glm(desc~balls+strikes+xplate+zplate,
        data = train, family='binomial')

        test_rf = predict(model_rf,type="prob", newdata=test)
        pred_rf = prediction(test_rf[,2], test$desc)
        # perf_rf = performance(pred_rf,'tpr', 'fpr') # plot ROC curve
        auc_rf[i] = performance(pred_rf,'auc')@y.values[[1]]

        pred_lr = predict.glm(model_lr, test, type = 'response')
        auc_lr[i] = roc(test$desc, pred_lr, plot = F, print.auc = T)$auc
}
# measure performance with auc of roc curves
auc_rf; auc_lr
```

```
# plot ROC curve
perf_rf = performance(pred_rf,'tpr', 'fpr')
plot(perf_rf, col=2, main='ROC of Random Forest Model')
abline(0,1, lty=2)
auc_text = sprintf('AUC = %3.2f', auc_rf[i])
legend('bottomright', auc_text)

# predict probability of strike in 3 cases
# 1. Three balls (3 balls, any strikes)
# 2. Full count (3 balls, 2 strikes)
# 3. Other (<3 balls, any strikes)
three = predict(model_rf,type="prob", newdata=fringe_df[fringe_df$balls
   ==3,])[,2]
full = predict(model_rf,type="prob", newdata=fringe_df[fringe_df$balls
   ==3&fringe_df$strikes==2,])[,2]
other = predict(model_rf,type="prob", newdata=fringe_df[fringe_df$balls
   !=3,])[,2]

# generate densities using built-in kernel density estimation
three_d = density(three)
full_d = density(full)
other_d = density(other)
# create cdf from densities
three_cd = three_d
three_cd$y = cumsum(three_d$y)/max(cumsum(three_d$y))
full_cd = full_d
full_cd$y = cumsum(full_d$y)/max(cumsum(full_d$y))
other_cd = other_d
other_cd$y = cumsum(other_d$y)/max(cumsum(other_d$y))

# plot histograms and densities together
par(mfrow=c(1,3))
hist(full, probability = T, main='Full Count Probabilities')
lines(full_d, col=2, lwd=2)
hist(three, probability = T, main='3 Balls, Any Strikes Probabilities')
lines(three_d, col=3, lwd=2)
hist(other, probability = T, main='<3 Balls Probabilities')
lines(other_d, col=4, lwd=2)
par(mfrow=c(1,1))

# just densities for comparison
plot(other_d, col=4, main='Density Comparisons', xlab='Probability')
lines(full_d, col=2)
lines(three_d, col=3)
legend('topright', c('Full Count', '3 Balls', 'Other'), fill = c(2, 3, 4)
   )

# cumulative densities
plot(other_cd,  col=4, main='Cumulative Density Comparisons', xlab='
```

```
   Probability')
lines(full_cd, col=2)
lines(three_cd, col=3)
legend('topleft', c('Full Count', '3 Balls', 'Other'), fill = c(2, 3, 4))

# plot empirical cumulative densities
plot(ecdf(three), col=3, main = 'CDF of Probability of "called_strike"',
   xlim=c(-0.1, 1.1))
par(new=T)
plot(ecdf(full), col=2, main = '', xlim=c(-0.1, 1.1))
par(new=T)
plot(ecdf(other), col=4, main = '', xlim=c(-0.1, 1.1))
legend('topleft', c('Full Count', '3 Balls', 'Other'), fill = c(2, 3, 4))

# 2-sample ks-test and p-val on data
ks.test(full, three)
ks.test(full, other)
ks.test(three, other)

# length(three)
# length(full)
# length(other)

# subsample
three_t = sample(three, 196)
other_t = sample(other, 196)
# 2-sample ks-test and p-val on subsamples
ks.test(full, three_t)
ks.test(full, other_t)
ks.test(three_t, other_t)

# plot empirical cumulative densities on subsamples
plot(ecdf(three_t), col=3, main = 'CDF of Probability of "called_strike"
   (subsampled)', xlim=c(-0.1, 1.1))
par(new=T)
plot(ecdf(full), col=2, main = '', xlim=c(-0.1, 1.1))
par(new=T)
plot(ecdf(other_t), col=4, main = '', xlim=c(-0.1, 1.1))
legend('topleft', c('Full Count', '3 Balls', 'Other'), fill = c(2, 3, 4))
```