Pitch Classification

Gabriel Zanuttini-Frank

This project seeks to, given a number of variables describing each pitch, identify the pitch types of the unknown pitches in a test set. With the advent of PITCHF/x data, this classification problem has become automatable, and thus is being widely explored within the MLB, both by the league for its broadcasts and statistics as well as by teams, presumably to evaluate the strengths and weaknesses of pitchers and hitters throwing or facing the different pitch types. The datasets used contain PITCHF/x information from several thousand pitches from a variety of pitchers over different seasons. The columns include the pitch ID, pitcher ID, season, pitcher handedness, pitcher height, initial pitch speed as it leaves the pitcher's hand, horizontal and vertical break distance, coordinates identifying the release point of the pitch, distance from the rubber at which the pitch was released, and the spin rate.

```r
train <- read.csv("pitchclassificationtrain.csv", as.is=TRUE)
test <- read.csv("pitchclassificationtest.csv", as.is=TRUE)

train[,c(2:4,13)] <- lapply(train[,c(2:4,13)], factor) # factor pitcherid, yearid, throws, and type
test[,c(2:4)] <- lapply(test[,c(2:4)], factor)
```

My first step is to explore the data. I will check whether there are any missing values and how each variable is distributed.

```r
sum(is.na(train)) # no missing values
sum(is.na(test))
for (i in 2:ncol(train)) { # pitchid not interesting, since all unique
  if (is.numeric(train[,i])) {
    hist(train[,i], main=names(train)[i], breaks=20)
    plot(train[,i], main=names(train)[i])
  } else {
    plot(train[,i], main=names(train)[i])
  }
}
```

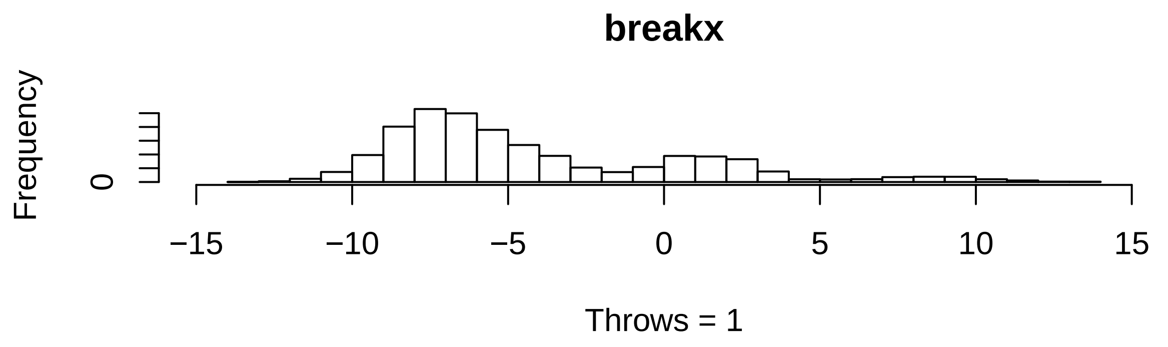
From looking at the histograms, 'initposx' and 'initposz' seem to be bimodal; 'initposz' also has two clusters in the scatterplot, while 'initposx' has three clusters in the scatterplot. Furthermore, 'breakx' has three modes, although they are not as distinct and are not very evident in its scatterplot. Lastly, 'breakz' looks slightly skewed left, but taking the log or the square root to transform it did not yield much improvement. Overall, there are no clear outliers in any of the variables.

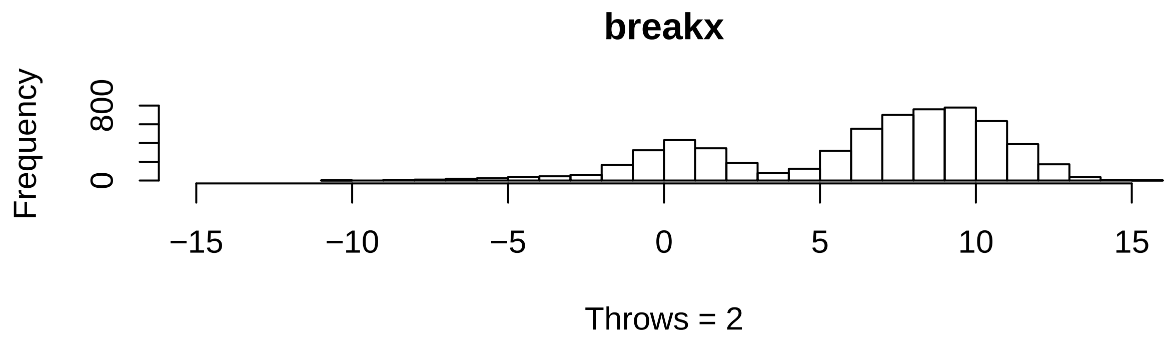
Next, I will eliminate 'pitchid' because it does not provide meaningful information for modeling, and is simply a unique identifier for each row. However, I will save the values to create the final table of predictions.

```r
trainids <- train$pitchid
testids <- test$pitchid
train <- train[,-which(names(train) %in% c("pitchid"))]
test <- test[,-which(names(test) %in% c("pitchid"))]
```

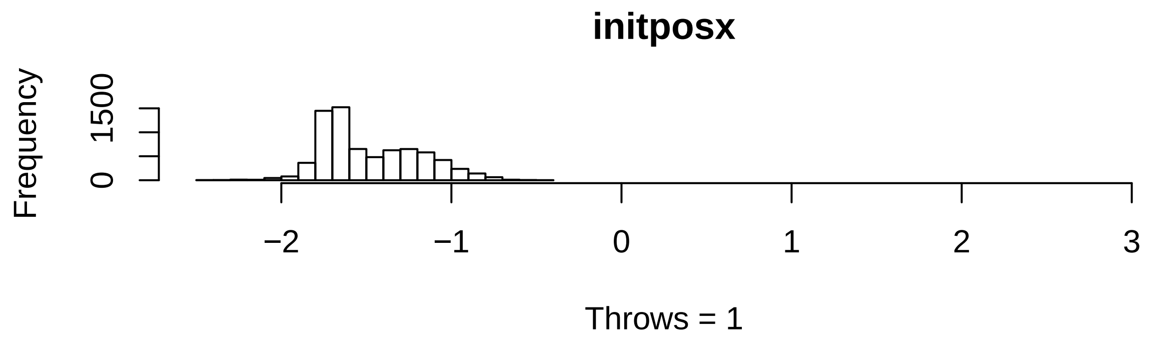
Explore how the distributions of certain variables change based on the handedness of the pitcher:

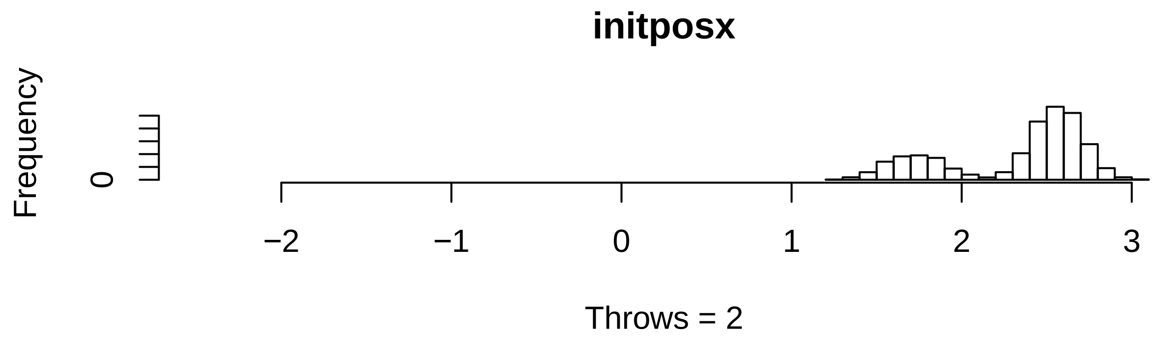
```r
par(mfrow=c(2,1))
hist(train$breakx[train$throws==1], main="breakx", xlab="Throws = 1", breaks=20, xlim=c(-15,15))
hist(train$breakx[train$throws==2], main="breakx", xlab="Throws = 2", breaks=20, xlim=c(-15,15))
```



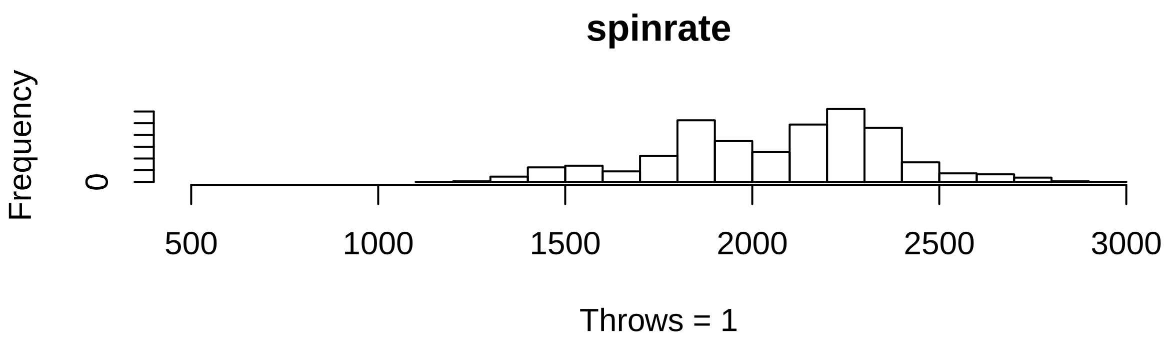


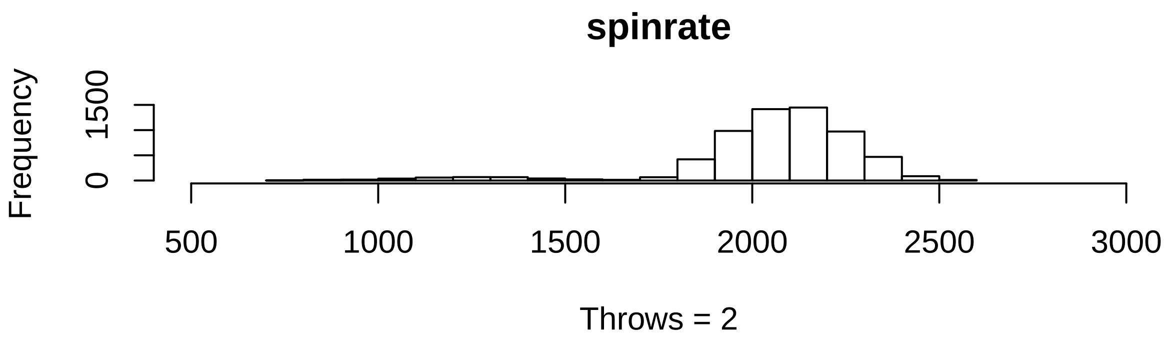
```r
hist(train$initposx[train$throws==1], main="initposx", xlab="Throws = 1", breaks=20, xlim=c(-2.5,3))
hist(train$initposx[train$throws==2], main="initposx", xlab="Throws = 2", breaks=20, xlim=c(-2.5,3))
```





```r
hist(train$spinrate[train$throws==1], main="spinrate", xlab="Throws = 1", breaks=20, xlim=c(500,3000))
hist(train$spinrate[train$throws==2], main="spinrate", xlab="Throws = 2", breaks=20, xlim=c(500,3000))
```





For some variables, the distributions are simply flipped around the midpoint between lefties and righites (breakx and initposx), while for others the distributions are noticeably different (spinrate). Therefore, in addition to building models on the entire dataset, I will build separate models for lefties and righties. Since lefties and righties' pitch statistics are distributed differently, my thought is that building a separate model for each one will more accurately capture and therefore predict the pitch type.

Split data by pitcher handedness:

```
train1 <- train[train$throws==1,]
train2 <- train[train$throws==2,]
test1 <- test[test$throws==1,]
test2 <- test[test$throws==2,]
```

Since there are six possible types that pitches can be classified as, this is a problem of multiclass classification rather than binary classification. As a result, many common classification methods will not be straightforward. I could reduce this to a binary classification problem by using a one-vs-rest or one-vs-one strategy, but this strategy could lead to several problems. Therefore, I will proceed with three methods that can handle multiclass classification: k-nearest neighbors, linear discriminant analysis, and random forests.

K-Nearest Neighbors

```
# Drop factor variables because KNN doesn't handle them:
  nofactortrain <- train[,-which(names(train) %in% c("pitcherid", "yearid", "throws"))]
nofactortest <- test[,-which(names(test) %in% c("pitcherid", "yearid", "throws"))]
knntrain <- nofactortrain
knntest <- nofactortest
```

Split data for KNN by pitcher handedness

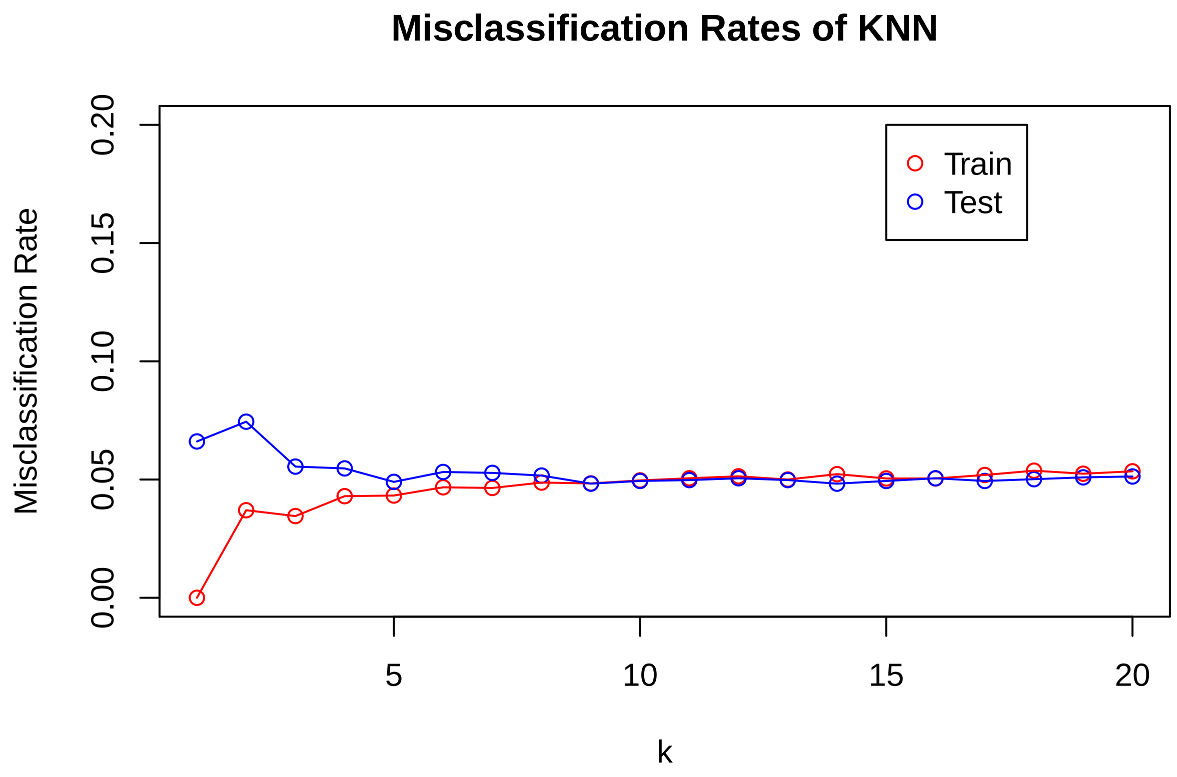
```r
knntrain1 <- knntrain[train$throws==1,]
knntrain2 <- knntrain[train$throws==2,]
knntest1 <- knntest[test$throws==1,]
knntest2 <- knntest[test$throws==2,]
```

Here I define a function that standardizes the given training and testing datasets using the means and standard deviations from the training data's variables.

```r
standardize.test <- function(train, test) {
means <- lapply(train[,1:8], mean)
sds <- lapply(train[,1:8], sd)
train[,1:8] <- scale(train[,1:8])
for (i in 1:8) {
test[,i] <- (test[,i]-means[[i]])/sds[[i]]
}
return(test)
}

# Standardize all KNN datasets
knntest <- standardize.test(knntrain, knntest)
knntest1 <- standardize.test(knntrain1, knntest1)
knntest2 <- standardize.test(knntrain2, knntest2)
knntrain[,1:8] <- scale(knntrain[,1:8])
knntrain1[,1:8] <- scale(knntrain1[,1:8])
knntrain2[,1:8] <- scale(knntrain2[,1:8])
```

I will now use the training data to plot how the sub-training and validation misclassification rates change with k.

```r
sample.ind <- sample(2, nrow(knntrain), replace = T, prob = c(0.8,0.2))
knnsubtrain <- knntrain[sample.ind==1,]
knnval <- knntrain[sample.ind==2,]

# apply KNN
misclassratesTEST <- c()
for (i in 1:20) {
knn1 <- knn(knnsubtrain[,1:8], knnval[,1:8], knnsubtrain$type, k=i)
misclassratesTEST <- c(misclassratesTEST, 1-sum(knnval$type==knn1)/nrow(knnval))
}

misclassratesTRAIN <- c()
for (i in 1:20) {
knn1 <- knn(knnsubtrain[,1:8], knnsubtrain[,1:8], knnsubtrain$type, k=i)
misclassratesTRAIN <- c(misclassratesTRAIN, 1-sum(knnsubtrain$type==knn1)/nrow(knnsubtrain))
}

par(mfrow=c(1,1))
plot(misclassratesTRAIN, col="red", xlab="k", ylab="Misclassification Rate", ylim=c(0,0.2), main="Miscla
points(misclassratesTEST, col="blue")
lines(misclassratesTRAIN, col="red")
lines(misclassratesTEST, col="blue")
legend(15, .2, legend=c("Train", "Test"), col=c("red", "blue"), pch=1)
```



As we would expect, the testing (validation) misclassification rate decreases (improves) as k increases from 1, reaching a minimum somewhere in the range of (5,15). Conversely, the training misclassification rate increases from 0 (k=1) to a stable rate similar to that of the testing data.

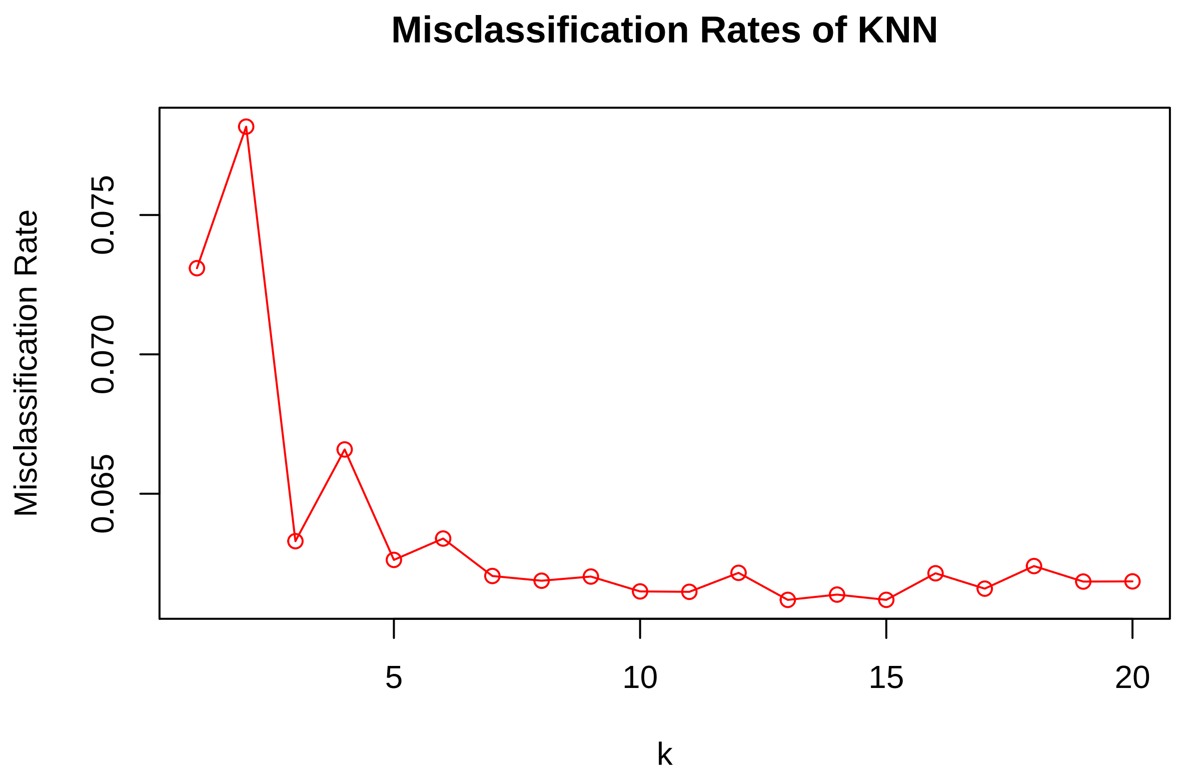
The two functions below collectively run KNN. 'getoptimalk' takes the training set and uses cross-validation to find the value of *k* that produces the lowest average misclassification rate. 'knnclass' then runs k-nearest neighbors and outputs the predicted values.

```r
typelevels <- c("2","3","4","8","9","10")
getoptimalk <- function(training) {
misclassrate <- rep(0,20)
for (i in 1:10) {
sample.ind <- sample(2, nrow(training), replace = T, prob = c(0.8,0.2))
knnsubtrain <- training[sample.ind==1,]
knnval <- training[sample.ind==2,]

misclassratesTEST <- c()
for (j in 1:20) {
knn1 <- knn(knnsubtrain[,1:8], knnval[,1:8], knnsubtrain$type, k=j)
levels(knn1) <- c(levels(knn1), typelevels[!(typelevels %in% levels(knn1))])
misclassratesTEST <- c(misclassratesTEST, 1-sum(knnval$type==knn1)/nrow(knnval))
}

misclassrate <- misclassrate + misclassratesTEST
}
avgerror <- misclassrate/10
bestk <- which(avgerror==min(avgerror))
return(list(bestk, avgerror))
}
```

```r
knnclass <- function(xtrain, xtest, ytrain, bestk) {
ytest <- knn(xtrain, xtest, ytrain, k=bestk)
levels(ytest) <- c(levels(ytest), typelevels[!(typelevels %in% levels(ytest))])
return(ytest)
}
```

Using these functions, I will run KNN on the entire training dataset as well as on the training sets split by pitcher type. With the split data, I calculate the average misclassification rate, weighted by the number of observations in each dataset.

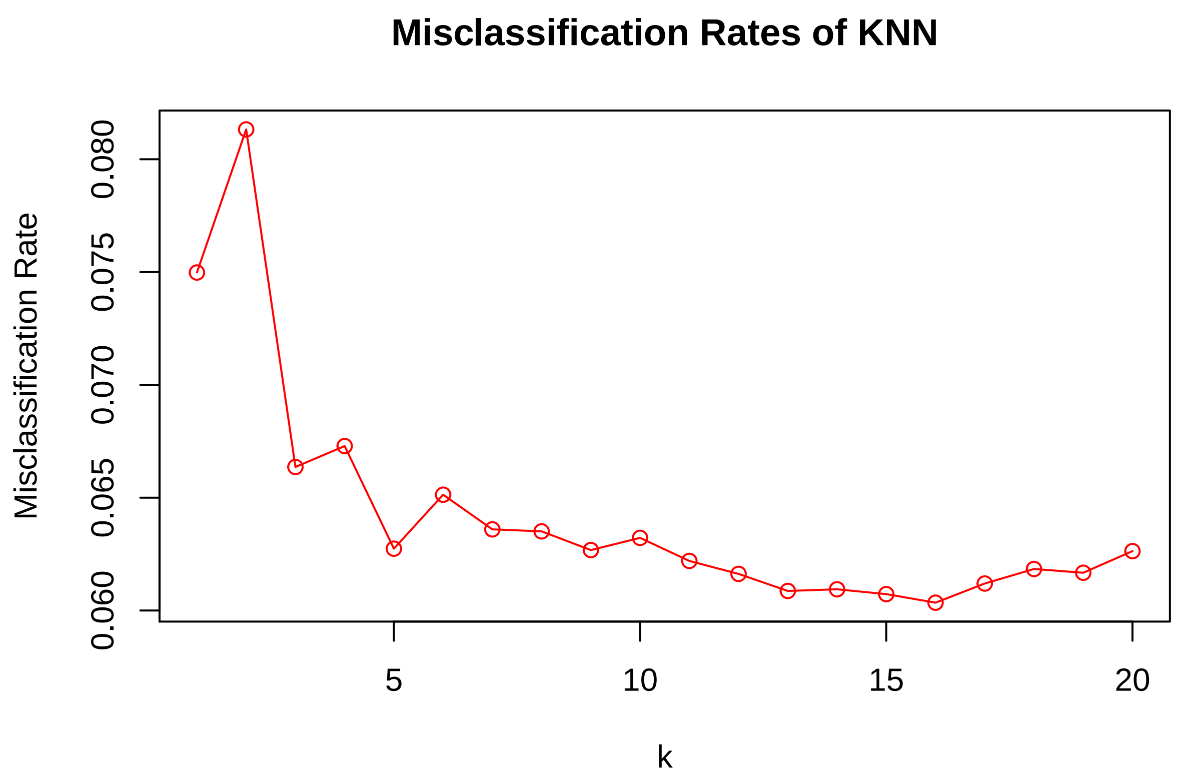
```r
bestk <- getoptimalk(knntrain)
plot(bestk[[2]], col="red", xlab="k", ylab="Misclassification Rate", main="Misclassification Rates of K
lines(bestk[[2]], col="red")
```



```r
min(bestk[[2]])  # misclassification error rate using optimal value of k
```

```
## [1] 0.06119185
```

```r
# Pitcher throws 1 vs. 2
bestk1 <- getoptimalk(knntrain1)
bestk2 <- getoptimalk(knntrain2)
avgerrors <- (bestk1[[2]]*nrow(knntrain1) + bestk2[[2]]*nrow(knntrain2))/nrow(knntrain)
plot(avgerrors, col="red", xlab="k", ylab="Misclassification Rate", main="Misclassification Rates of KN
lines(avgerrors, col="red")
```



```
min(avgerrors)  # misclassification error rate using optimal value of k
```

```
## [1] 0.06034804
```

We can see from the plots above that splitting the training dataset by pitcher hand yields a similar optimal value of k to the entire training set, and produces a slightly higher error.

```
knnpreds <- knnclass(knntrain[,1:8], knntest, knntrain$type, bestk[[1]])

knnpreds1 <- knnclass(knntrain1[,1:8], knntest1, knntrain1$type, bestk1[[1]])
knnpreds2 <- knnclass(knntrain2[,1:8], knntest2, knntrain2$type, bestk2[[1]])

knnbyhand <- data.frame(cbind(test$throws, NA))
names(knnbyhand) <- c("throws", "preds")
knnbyhand$preds[knnbyhand$throws==1] <- levels(knnpreds1)[knnpreds1]
knnbyhand$preds[knnbyhand$throws==2] <- levels(knnpreds2)[knnpreds2]
knnbyhand$preds <- factor(knnbyhand$preds)
knnbyhand <- knnbyhand$preds
```

```
confusionMatrix(knnpreds, knnbyhand)$table
```

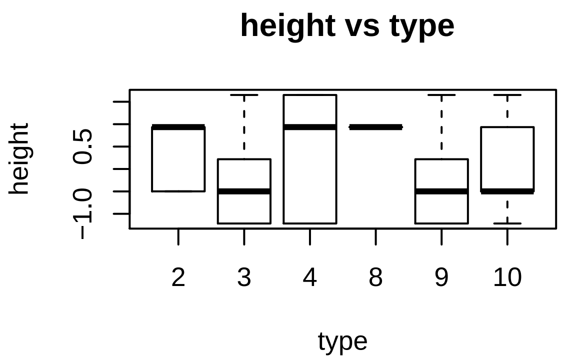
```
##             Reference
## Prediction   10    2    3    4    8    9
##         10 1531    0    0    4    0   88
##         2     0  272    0    0    0    0
##         3     0    0 1118    0    0    0
##         4     5    0    0  732    0    3
##         8     0    0    0    0  241    0
##         9    84    0    0    5    0 1769
```

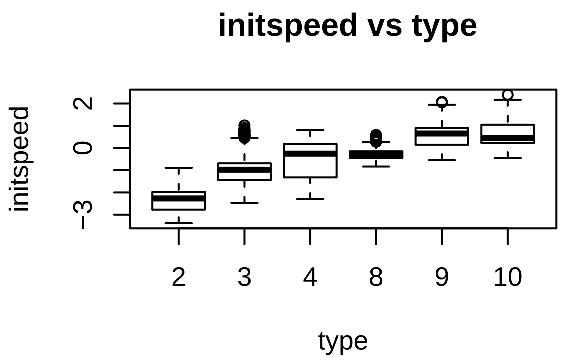
Finally, I compared the predictions made by KNN on the entire dataset and those made by splitting the data. As is shown in the confusion matrix above, these predictions were almost identical. As a result, I will conclude that splitting lefties and righties does not significantly change KNN (except for slightly on pitched labeled 9 and 10), so I will use the predictions on the entire training set as my final KNN predictions.

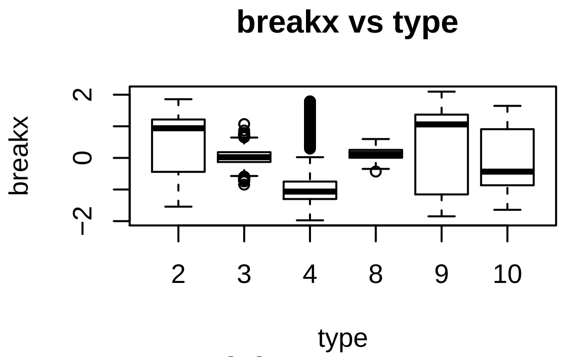
LDA

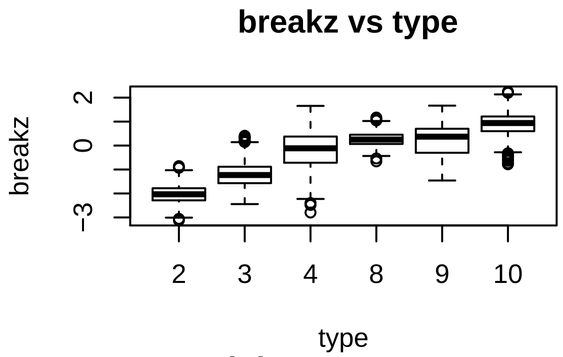
My second attempt at classifying pitch types was using linear discriminant analysis.

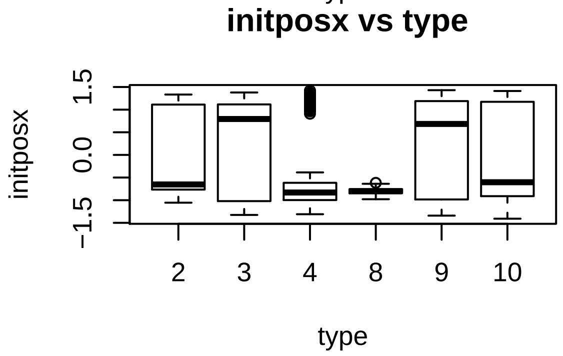
```r
par(mfrow=c(2,2))
for (i in 1:8) {
plot(knnsubtrain$type, knnsubtrain[,i], main=paste("", names(knnsubtrain)[i], "vs type"), xlab="type", 
}
```

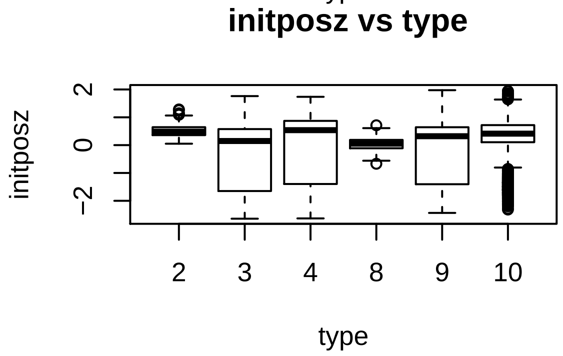


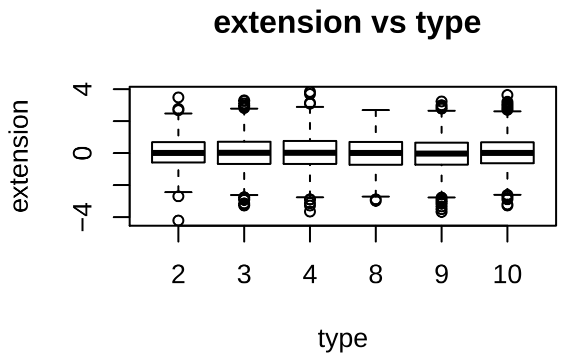


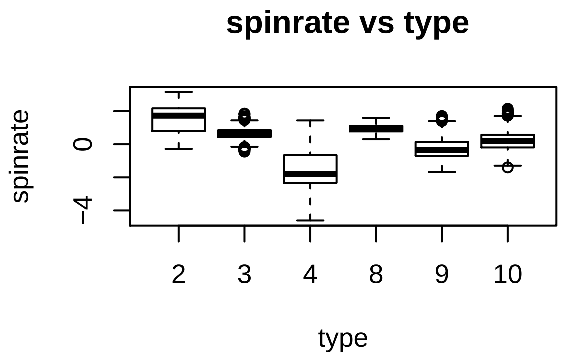












I made boxplots of each variable by pitch type in order to identify which variables would and would not be useful in distinguishing between pitch types. All variables seemed to distinguish between certain combinations of pitch types except for 'extension' so I will build my model on all of the variables except for that one.

```r
varnames <- paste(names(knnsubtrain)[c(1:6,8)], collapse="+")
ldaform <- as.formula(paste("type", varnames, sep=" ~ "))
lda1 <- lda(ldaform, data=knnsubtrain)
type.pred <- predict(lda1, newdata=knnval)$class
lda.error <- sum(type.pred!=knnval$type)/length(type.pred)
lda.error # test error of model
```

```
## [1] 0.1325988
```

The error above from the LDA model is significantly higher than the ~0.5 given by KNN. Therefore, I will not use LDA built on the entire training set as my final method of prediction over KNN.

```r
sample.ind1 <- sample(2, nrow(knntrain1), replace = T, prob = c(0.8,0.2))
knnsubtrain1 <- knntrain1[sample.ind1==1,]
knnval1 <- knntrain1[sample.ind1==2,]
lda.hand1 <- lda(ldaform, data=knnsubtrain1)
type.pred1 <- predict(lda.hand1, newdata=knnval1)$class
lda.error1 <- sum(type.pred1!=knnval1$type)/length(type.pred1)
lda.error1 # test error of model
```

```
## [1] 0.09734513
```

```r
sample.ind2 <- sample(2, nrow(knntrain2), replace = T, prob = c(0.8,0.2))
knnsubtrain2 <- knntrain2[sample.ind2==1,]
knnval2 <- knntrain2[sample.ind2==2,]
lda.hand2 <- lda(ldaform, data=knnsubtrain2)
```

```
## Warning in lda.default(x, grouping, ...): group 8 is empty
```

```r
type.pred2 <- predict(lda.hand2, newdata=knnval2)$class
lda.error2 <- sum(type.pred2!=knnval2$type)/length(type.pred2)
lda.error2 # test error of model
```

```
## [1] 0.06294256
```

```r
(lda.error1*nrow(knntrain1) + lda.error2*nrow(knntrain2))/nrow(knntrain)  # total weighted error after
```

```
## [1] 0.08159377
```

Splitting the two kinds of pitchers significantly improves LDA's performance on the holdout dataset. However, this error is still higher than that of KNN, so I will proceed with KNN as my final method for now.

Random Forest

My third and final method of prediction is random forests. Below I define a function that runs random forests and evaluates them using five-fold cross validation, returning the average error of the model.

```r
fivefoldcv <- function(train, form, ntrees, minnodesize, nvars) {
  samp <- sample(cut(seq(1, nrow(train)), breaks=5, labels=FALSE))
  error <- c()
  for (i in 1:5) {
    holdout <- train[samp==i,]
    training <- train[samp!=i,]
    rf1 <- ranger(form, data=training, num.trees=ntrees, min.node.size=minnodesize, mtry=nvars, write.f
    # rf1$forest$levels <- c(rf1$forest$levels, typelevels[!(typelevels %in% rf1$forest$levels)])
    rf1$forest$levels <- typelevels
    pred <- predict(rf1, data=holdout)
    error <- c(error, sum(pred$predictions!=holdout$type)/nrow(holdout))
  }
  err <- mean(error)
  return(err)
}
```

Next, I run 'fivefoldcv' on the entire training dataset, with the following parameters: ntrees = 250, min.node.size = 20, and mtry = 4. These parameters regulate the bias and variance, in addition to the correlation between trees. I tried varying each of these parameters to lower the error, and found that the three listed above performed best.

```
rf.form <- as.formula(paste("type",paste(names(train)[1:11],collapse="+"),sep="~"))
fivefoldcv(train, rf.form, 250, 20, 4)
```

```
## [1] 0.04487157
```

This error is noticeably lower than that of KNN, so I will use these predictions as my final ones.

Now, build a random forest model on all of the training data:

```
rf.final <- ranger(rf.form, data=train, num.trees=250, min.node.size=20, mtry=4, write.forest=TRUE, resp
pred.rf <- predict(rf.final, data=test)

sum(pred.rf$predictions==knnpreds)/nrow(test)
```

```
## [1] 0.965311
```

```
confusionMatrix(knnpreds, pred.rf$predictions)$table
```

```
## Warning in confusionMatrix.default(knnpreds, pred.rf$predictions): Levels
## are not in the same order for reference and data. Refactoring data to
## match.
```

```
##           Reference
## Prediction    2    3    4    8    9   10
##         2   272    0    0    0    0    0
##         3     0 1118    0    0    0    0
##         4     0    0  734    0    5    1
##         8     0    0    0  241    0    0
##         9     0    0    3    0 1758   97
##         10    0    0    3    0   94 1526
```

We can see that ~97% of pitches are classified in the same way by both KNN and random forest, which makes the predictions feel more reliable. Also, as mentioned earlier, pitch types 9 and 10 are still the most difficult to distinguish between.

Finally, I will try random forests with the split data:

```
rftrain1 <- train1[,-3]
rftrain2 <- train2[,-3]
rftest1 <- test1[,-3]
rftest2 <- test2[,-3]

rf.form.split <- as.formula(paste("type",paste(names(rftrain1)[1:10],collapse="+"),sep="~"))
rf.err1 <- fivefoldcv(rftrain1, rf.form.split, 250, 20, 4)
rf.err2 <- fivefoldcv(rftrain2, rf.form.split, 250, 20, 4)
```

```
## Warning: Dropped unused factor level(s) in dependent variable: 8.
## Warning: Dropped unused factor level(s) in dependent variable: 8.

## Warning: Dropped unused factor level(s) in dependent variable: 8.

## Warning: Dropped unused factor level(s) in dependent variable: 8.

## Warning: Dropped unused factor level(s) in dependent variable: 8.
```

```
rferrorhand <- (rf.err1*nrow(rftrain1) + rf.err2*nrow(rftrain2))/nrow(train)  # total error after split
rferrorhand
```

```
## [1] 0.0452388
```

The weighted average error from using random forests is roughly the same after splitting the data as it is when using the entire training dataset. This is not surprising because many of the trees likely used the 'throws' variable as a decision node toward the top of the tree. Therefore, this serves the same purpose as manually splitting the dataset beforehand.

Since random forests performed best on the holdout sets using cross-validation, I will now produce final pitch type classification predictions of the pitches in the test data.

```
finalclasses <- data.frame(testids, pred.rf$predictions)
names(finalclasses) <- c("pitchid","pred.type")
write.csv(finalclasses, "Zanuttini-Frank_pitchTypes.csv", row.names=FALSE)
```