Hit Classification Case Study

Kenny Miller

4/5/2021

Summary

I was tasked with developing a new classification approach for batted balls using more than just launch angle, which is the current Rapsodo approach. The model I developed is a support vector machine that uses interactions between the exit speed, distance, direction, and playing level variables. This model had an accuracy of 81.4% on the holdout data during model training and accurately classified hit types at an 81% clip when compared to the previous system's classifications.

Cleaning

- Verified there were no missing values in the data
- Checked distributions/representations of all 6 variables to ensure they would be appropriate for modeling
- Removed 11,244 rows where exitSpeed was 0 because it does not make sense a batted ball would leave the bat at 0 m/s (this also removed the rows where 0 was the only value in the launchAngle, direction, and distance variables). This appeared to be a potential calibration/recording error in the system used to record the data
- Double-checked distributions again to ensure the removal of rows did not require new transformations; all variables were normally or uniformly distributed and good to use in model development
- Adjusted the hitClass variable to a factor in order to run classification modeling
- Split the data into training and holdout samples; using a simple random sample of 80% of the total rows

Model Development

- Started with a classic decision tree model to predict hitClass
- Decision Tree did well at classifying the hit class but stepped up to a naive Bayes model to see if there were improvements
- Finally tested a support vector machine (SVM) model both with and without interactions
- The SVM model with interactions was the best choice for classifying this data
- Using the full data frame to make predictions/classifications, the SVM model does a good job to predict most hit types but struggles with popups
- Interactions greatly improved the model, as we could control for the variations in the exit speed across the playing levels

Conclusions

During this case, I utilized multiple models to try and identify a new way to classify batted ball hit classes. The best model for classifying the data was a support vector machine that included interactions between the variables in the model. Since launch angle perfectly predicted hit class, it needed to be excluded from the model development.

Future Recommendations

- Recording hit class by eye/user input, instead of using the current launch angle system, in order to better compare how a new system/model compares in classifying hit types to the current Rapsodo approach
- Develop additional models that include launch angle to help classify hits but are not able to explain it perfectly

---

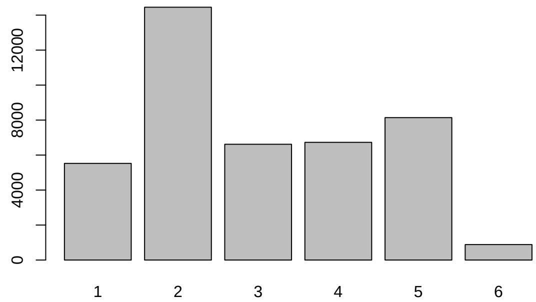
Data Cleaning

- Let's check the hitClass variable first:

```
summary(hits)
```

```
##     hitClass       exitSpeed      launchAngle       direction
##  Min.   :1.000   Min.   : 0.00   Min.   :-78.521   Min.   :-178.531
##  1st Qu.:2.000   1st Qu.: 0.00   1st Qu.:  0.000   1st Qu.:  -4.236
##  Median :3.000   Median :30.15   Median :  7.792   Median :   0.000
##  Mean   :3.004   Mean   :24.21   Mean   : 11.102   Mean   :   1.363
##  3rd Qu.:4.000   3rd Qu.:36.64   3rd Qu.: 21.794   3rd Qu.:   8.828
##  Max.   :6.000   Max.   :60.85   Max.   : 86.193   Max.   : 178.301
##     distance       playingLevel
##  Min.   :  0.00   Min.   :0.000
##  1st Qu.:  0.00   1st Qu.:1.000
##  Median : 27.64   Median :1.000
##  Mean   : 34.84   Mean   :1.265
##  3rd Qu.: 63.05   3rd Qu.:2.000
##  Max.   :139.53   Max.   :3.000
```

```
barplot(table(hits$hitClass))
```




```
table(hits$hitClass)
```

```
##
##     1     2     3     4     5     6
##  5525 14453  6619  6729  8142   885
```

Review: We see a disproportional amount of ground balls in comparison to the remaining hit classes but this may not be a major issue. Could be an opportunity to review these hits and see if there is an opportunity to add/adjust our hit type buckets. Also, there are not many popups but this most likely isn't an issue

- Now let's check the exitSpeed variable:

```
hist(hits$exitSpeed)
```


```
summary(hits$exitSpeed)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00   30.15   24.21   36.64   60.85
```

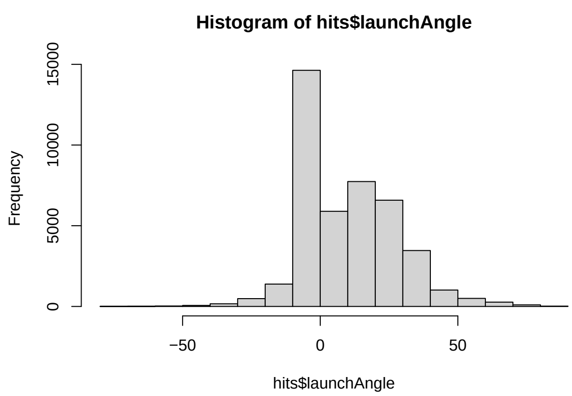
```
summary(hits[which(hits$exitSpeed == 0),]) # let's check what all the 0 values are
```

```
##     hitClass    exitSpeed  launchAngle   direction    distance  playingLevel
##  Min.   :2   Min.   :0   Min.   :0    Min.   :0   Min.   :0   Min.   :0.00
##  1st Qu.:2   1st Qu.:0   1st Qu.:0    1st Qu.:0   1st Qu.:0   1st Qu.:1.00
##  Median :2   Median :0   Median :0    Median :0   Median :0   Median :1.00
##  Mean   :2   Mean   :0   Mean   :0    Mean   :0   Mean   :0   Mean   :1.32
##  3rd Qu.:2   3rd Qu.:0   3rd Qu.:0    3rd Qu.:0   3rd Qu.:0   3rd Qu.:2.00
##  Max.   :2   Max.   :0   Max.   :0    Max.   :0   Max.   :0   Max.   :3.00
```

Review: There are a ton of 0 or near 0 values here, not entirely sure why this is the case. Have there been errors in the calculations/calibration of the data? Outside of the early values where we see a lot of 0 values the data appears to follow a normal distribution well enough that we don't need to worry much about it, but let's continue to explore why there are so many near 0s. It looks like these values are all 0 across the board in every category (this could also explain the large number of ground balls because they are all labeled as grounders), and the system calibration could have been off or there were errors; potentially we need to remove these rows.

- Time to review launchAngle variable:

```
hist(hits$launchAngle)
```




```
summary(hits$launchAngle)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -78.521   0.000   7.792  11.102  21.794  86.193
```

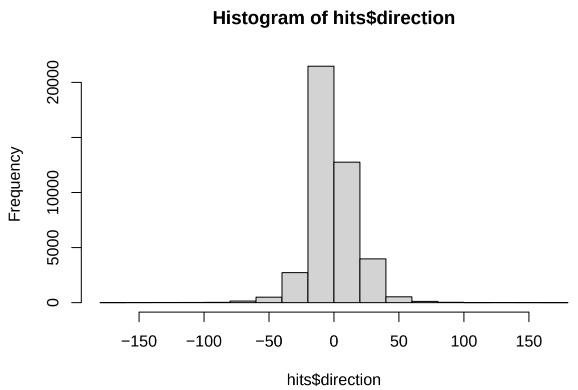
```
summary(hits[which(hits$launchAngle == 0),]) # let's check what all the 0 values are
```

```
##     hitClass    exitSpeed  launchAngle   direction    distance  playingLevel
##  Min.   :2   Min.   :0   Min.   :0    Min.   :0   Min.   :0   Min.   :0.00
##  1st Qu.:2   1st Qu.:0   1st Qu.:0    1st Qu.:0   1st Qu.:0   1st Qu.:1.00
##  Median :2   Median :0   Median :0    Median :0   Median :0   Median :1.00
##  Mean   :2   Mean   :0   Mean   :0    Mean   :0   Mean   :0   Mean   :1.32
##  3rd Qu.:2   3rd Qu.:0   3rd Qu.:0    3rd Qu.:0   3rd Qu.:0   3rd Qu.:2.00
##  Max.   :2   Max.   :0   Max.   :0    Max.   :0   Max.   :0   Max.   :3.00
```

Review: Again there are a ton of near 0 values but this most likely is the same line of thinking as with the exitSpeed variable. Variable distribution looks super outside of those 0 values.

- Review of direction variable:

```
hist(hits$direction)
```



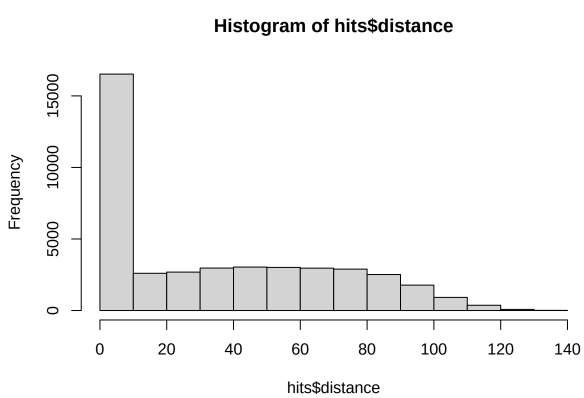

```
summary(hits$direction)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -178.531   -4.236    0.000    1.363    8.828  178.301
```

Review: The direction variable looks good probably want to adjust or remove those 0 values similar to the exitSpeed variable.

- Review of the distance variable:

```
hist(hits$distance)
```




```
summary(hits$distance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00   27.64   34.84   63.05  139.53
```

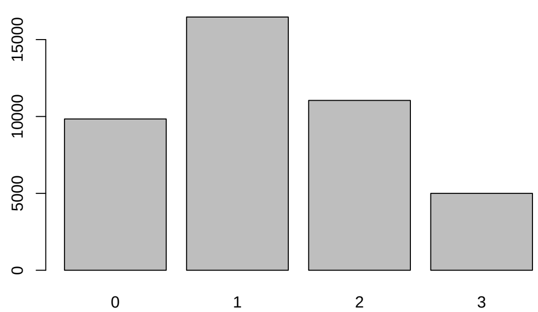
```
summary(hits[which(hits$distance == 0),]) # let's check what all the 0 values are
```

```
##     hitClass   exitSpeed  launchAngle   direction    distance  playingLevel
##  Min.   :2   Min.   :0   Min.   :0    Min.   :0   Min.   :0   Min.   :0.00
##  1st Qu.:2   1st Qu.:0   1st Qu.:0    1st Qu.:0   1st Qu.:0   1st Qu.:1.00
##  Median :2   Median :0   Median :0    Median :0   Median :0   Median :1.00
##  Mean   :2   Mean   :0   Mean   :0    Mean   :0   Mean   :0   Mean   :1.32
##  3rd Qu.:2   3rd Qu.:0   3rd Qu.:0    3rd Qu.:0   3rd Qu.:0   3rd Qu.:2.00
##  Max.   :2   Max.   :0   Max.   :0    Max.   :0   Max.   :0   Max.   :3.00
```

Review: The distance variable appears to follow a more uniform distribution, which isn't an issue at all, other than that major spike near 0, let's review that. Again all of the 0 values are when each variable is 0 across the board, I'll most likely want to remove these rows.

- Review of the playingLevel variable:

```
barplot(table(hits$playingLevel))
```




```
table(hits$playingLevel)
```

```
##
##     0     1     2     3
##  9839 16468 11045  5001
```

Review: Looks like a good enough distribution of the levels, we can double check after the final adjustments.

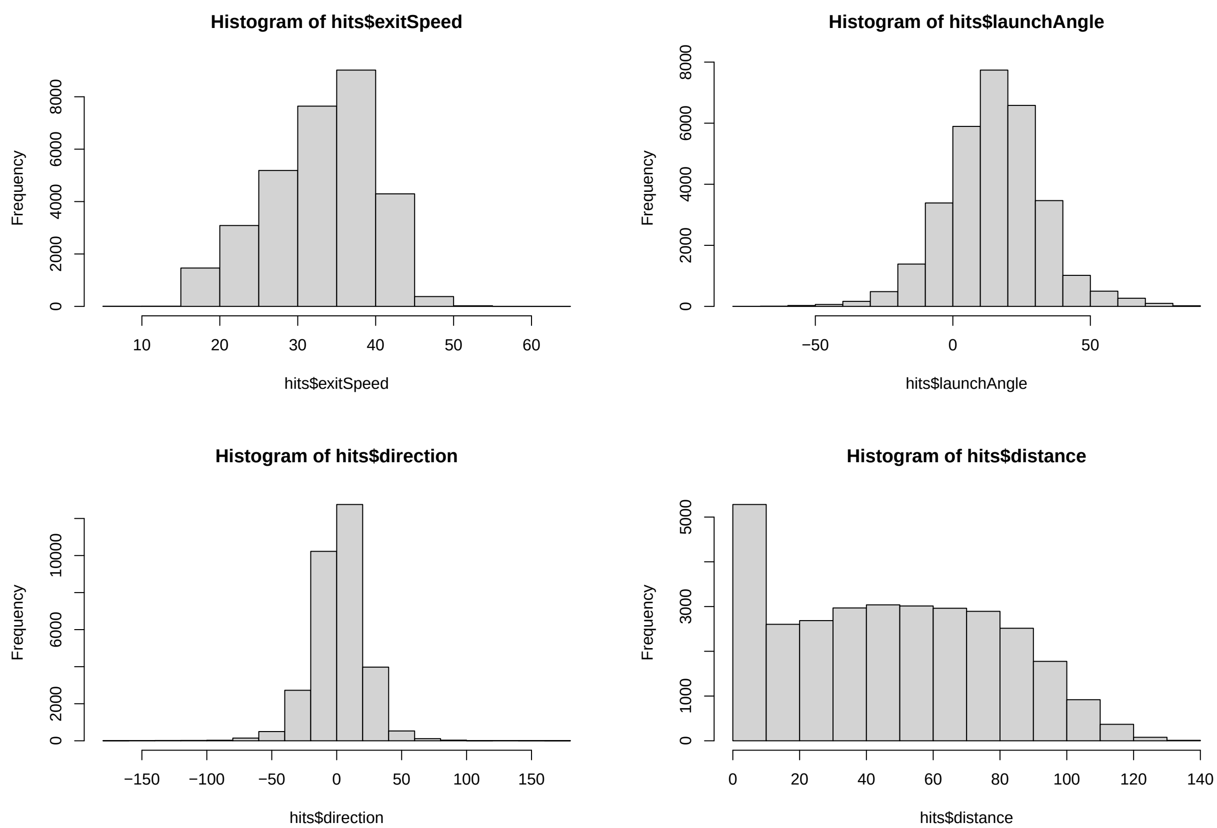
Final Data Cleaning Decisions!

Since all four hit related variables have no variation from 0 when one of them is 0, I will remove these 11,000 + rows because it does not make sense to include rows where we say a batted ball had exitSpeed of 0 m/s.

```
# hits.original <- hits
nrow(hits[which(hits$exitSpeed == 0),])/nrow(hits)
```

```
## [1] 0.265483
```

```r
# 26.5% is a large proportion of data to remove but is necessary
hits <- hits[-which(hits$exitSpeed == 0),]
hits$hitClass <- as.factor(hits$hitClass) # adjust hitClass to a factor in order to predict levels
# leave playing level as an integer to see how that predicts, may adjust to factor later
hits$oldPredicted <- ifelse(hits$launchAngle < 0, 1,
                        ifelse(hits$launchAngle >= 0 & hits$launchAngle < 6, 2,
                          ifelse(hits$launchAngle >= 6 & hits$launchAngle < 15, 3,
                            ifelse(hits$launchAngle >= 15 & hits$launchAngle < 24, 4,
                              ifelse(hits$launchAngle >= 24 & hits$launchAngle < 50, 5, 6)))))
hits$oldPredicted <- as.factor(hits$oldPredicted)
# after removing the 0 value rows let's double check the histograms
hist(hits$exitSpeed);hist(hits$launchAngle);hist(hits$direction);hist(hits$distance)
```

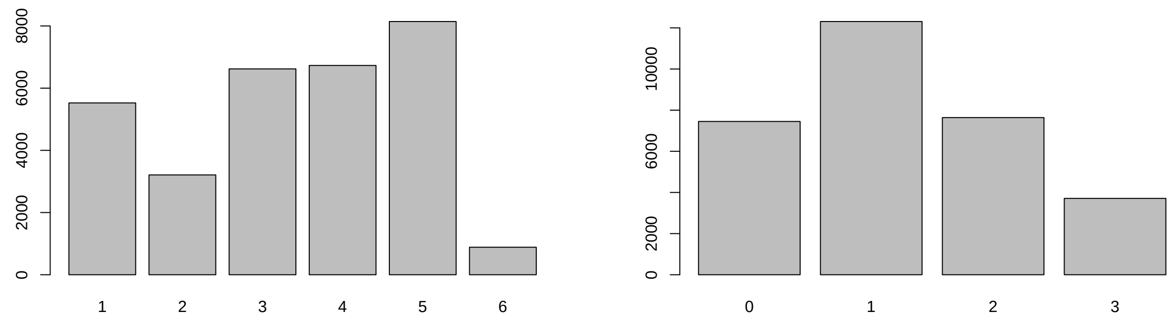



```r
# all of the histograms look great with the removal of those rows!
# need to double check the distributions of Levels and Classes
table(hits$hitClass);table(hits$playingLevel)
```

```
##
##    1    2    3    4    5    6
## 5525 3209 6619 6729 8142  885

##
##     0     1     2     3
##  7451 12308  7639  3711
```

```
barplot(table(hits$hitClass)); barplot(table(hits$playingLevel))
```



Final Decisions:

- Remove 11,244 rows that are 0s for every value of exitSpeed, launchAngle, direction, and distance because these appear to be rows where system calibration and recording are inaccurate
- Make hitClass a factor variable in order to run predictive/classification models
- I have decided to leave playingLevel as an integer in the first iteration of modeling instead of changing it to a factor, this may change after reviewing models
- Add in column of current system hit classifications to check against new systems and convert to a factor in order to run the check
- Verify distributions of all predictor variables and verify proper (somewhat equal) representation in all levels of hitClass and playingLevel
- Everything looks good, let's proceed to model development

Model Creation

After reviewing the data I believe that a decision tree model (or potentially a naive Bayes model) will work best to predict hitClass over the current model using only launchAngle.

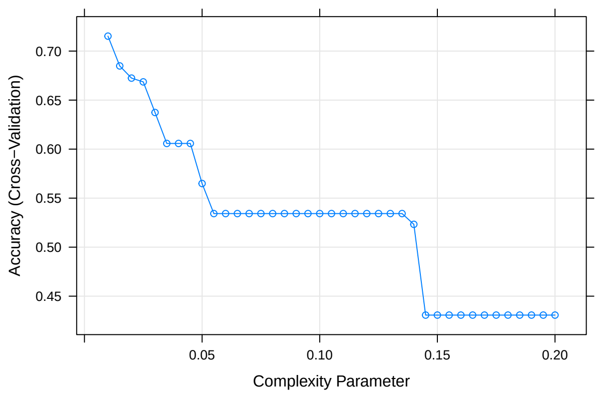
```
# check accuracy of the current model
mean(hits$hitClass == hits$oldPredicted)
```

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

```
# hitClass is confirmed to be based on the original model/approach by Rapsodo (oldPredicted can
# be removed now); I emailed Nicholas to see if there is another way to compare accuracies!
# since launchAngle will perfectly predicted hitClass, we need to remove it from
# any model developments
hits.analysis <- hits[, c(1,2,4,5,6)]
set.seed(23); train.rows <- sample(1:nrow(hits.analysis), 0.8*nrow(hits.analysis))
TRAIN <- hits.analysis[train.rows,]; HOLDOUT <- hits.analysis[-train.rows,]
```

Decision Tree Model

```
TREE <- train(hitClass~., data=TRAIN, method="rpart",
             trControl=trainControl(method="cv",number=10),
             tuneGrid=data.frame(cp=seq(from=0.01, to=0.20, by=0.005)),
             preProc = c("center", "scale"))
plot(TREE)
```



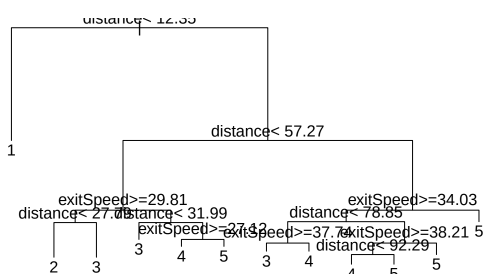

```
TREE$results[rownames(TREE$bestTune),]
```

```
##     cp  Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.01 0.7152333 0.6376354 0.009947093 0.01301002
```

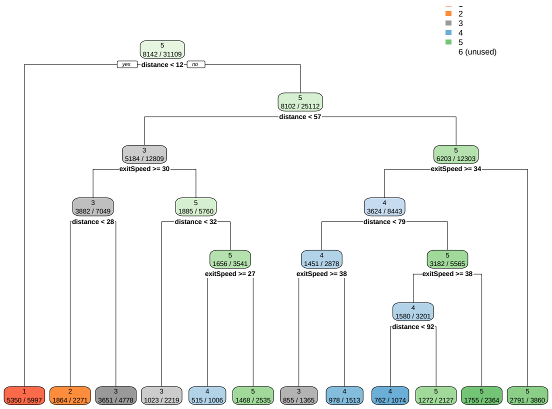
```
postResample(predict(TREE,newdata=HOLDOUT),HOLDOUT$hitClass)
```

```
##  Accuracy     Kappa
## 0.7262938 0.6517144
```

```
rtree <- rpart(hitClass~., data=hits.analysis, cp=0.01)
plot(rtree); text(rtree, digits = 2)
```



```r
rpart.plot(rtree, extra = 2)
```



Conclusion: A more complex tree with limited penalty for additional nodes does well in predicting the hitClass for our batters. Let's only explore the complexity parameter down to 0.01. This tree model is relatively accurate on the training and on the holdout data, but let's look at the naive Bayes model.

Naive Bayes Model

```r
x <- hits.analysis[,c("exitSpeed", "direction", "distance", "playingLevel")]
y <- hits.analysis$hitClass
# using same 80% training rows as sampled above
x.train <- x[train.rows,]
y.train <- y[train.rows]
x.test <- x[-train.rows,]
y.test <- y[-train.rows]
bayes <- suppressWarnings(train(x.train, y.train, method = "nb",
                                tuneGrid = data.frame(usekernel=FALSE,fL=0,adjust=1),
                                trControl = trainControl(method = "cv", number = 10)))
y.pred <- suppressWarnings(predict(bayes, newdata = x.test))
table(y.test,y.pred) # confusion matrix
```

```
##       y.pred
## y.test    1    2    3    4    5    6
##      1 1108   61    0    0    0    9
##      2   61  432  123    0    0    4
##      3   11  220  805  214   44   14
##      4    5   41  324  495  431   19
##      5    5   18  180  135 1225   64
##      6    4   13   33   17   27   80
```

```r
sum(y.test==y.pred)/length(y.test) # prediction accuracy is 0.6661845
```

```
## [1] 0.6661845
```

Conclusion: It appears we do not predict well using the naive Bayes approach, let's see if an SVM model with and without interactions works better than this or the tree.

Support Vector Machine Model

```r
svmFit<-train(hitClass~.,data=TRAIN,method="svmLinear",trControl=trainControl(method="cv",number=10))
svmFit # accuracy on training data is 0.8073696
```

```
## Support Vector Machines with Linear Kernel
##
## 24887 samples
##     4 predictor
##     6 classes: '1', '2', '3', '4', '5', '6'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 22398, 22399, 22398, 22398, 22399, 22399, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8067666  0.7554268
##
## Tuning parameter 'C' was held constant at a value of 1
```

```r
pred <- predict(svmFit, newdata = HOLDOUT)
table(HOLDOUT$hitClass, pred) # confusion matrix
```

```
##    pred
##        1    2    3    4    5    6
##   1 1153   24    0    0    0    1
##   2   30  511   79    0    0    0
##   3    1   71 1080  154    1    1
##   4    5    0  117  872  320    1
##   5    5    0    1  202 1418    1
##   6    7    7   20   58   76    6
```

```r
sum(HOLDOUT$hitClass == pred)/nrow(HOLDOUT) # prediction accuracy is 0.8100289; we see improvement
```

```
## [1] 0.8100289
```

```r
# let's try with interactions
svmFit2<-train(hitClass~.^2,data=TRAIN,method="svmLinear",trControl=trainControl(method="cv",number=10))
svmFit2 # accuracy on training data is 0.8125522; looking good!
```

```
## Support Vector Machines with Linear Kernel
##
## 24887 samples
##     4 predictor
##     6 classes: '1', '2', '3', '4', '5', '6'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 22398, 22397, 22398, 22398, 22399, 22399, ...
## Resampling results:
##
##   Accuracy   Kappa
```

```
##   0.8127539  0.7630971
##
## Tuning parameter 'C' was held constant at a value of 1
```

```r
pred2 <- predict(svmFit2, newdata = HOLDOUT)
table(HOLDOUT$hitClass, pred2) # confusion matrix
```

```
##    pred2
##        1    2    3    4    5    6
##   1 1152   25    0    0    0    1
##   2   31  512   77    0    0    0
##   3    1   67 1082  155    0    3
##   4    5    0  122  888  298    2
##   5    6    0    2  198 1421    0
##   6    7    5   17   58   76   11
```

```r
sum(HOLDOUT$hitClass == pred2)/nrow(HOLDOUT) # prediction accuracy is 0.8142077; minor improvement
```

```
## [1] 0.8142077
```

Conclusion: Both versions of the support vector machines seem to struggle most with classifying popups, at least when compared to the current approach, but interactions do improve our predictions. I will use an SVM model with interactions as the final approach.

Final Model

```r
# use full data set
SVM <- train(hitClass~.^2, data = hits.analysis,
             method = "svmLinear",
             trControl = trainControl(method = "cv", number = 10))
hits$newPredicted <- predict(SVM, newdata=hits)
mean(hits$hitClass == hits$newPredicted) # 0.8133659
```

```
## [1] 0.8133659
```

```r
table(hits$hitClass,hits$newPredicted)
```

```
##
##        1    2    3    4    5    6
##   1 5389  131    0    0    0    5
##   2  212 2648  348    0    0    1
##   3    6  328 5499  771    2   13
##   4   14    0  700 4570 1438    7
##   5   39    0    9  936 7156    2
##   6   55   11  103  292  383   41
```

```r
SVM$coefnames
```

```
##  [1] "exitSpeed"              "direction"              "distance"
##  [4] "playingLevel"           "exitSpeed:direction"    "exitSpeed:distance"
##  [7] "exitSpeed:playingLevel" "direction:distance"     "direction:playingLevel"
## [10] "distance:playingLevel"
```

Conclusion: The final support vector machine model is used to add a new column to the original “hits” data frame. These are my new predictions and recommendations for classifying hit type. We see interactions are very important to helping classify what type of hit a batted ball is.
