1

PREDICTING THE NEXT PITCH (BASEBALL)

Final Project Assignment
Data Mining for Business Analytics
STA 9660 S1DA [9489]

John Estefano Ortiz
July 16 2018

[1] (Mutliple) Mariano Rivera
https://en.wikipedia.org/wiki/Mariano_Rivera

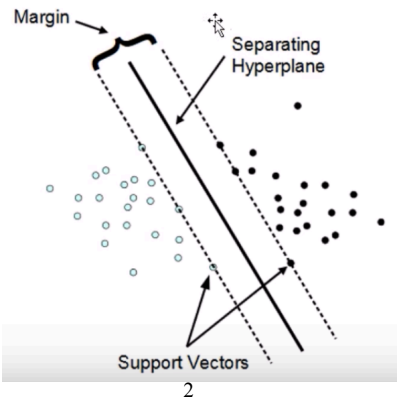
Table of Contents

I. Classification Methodology

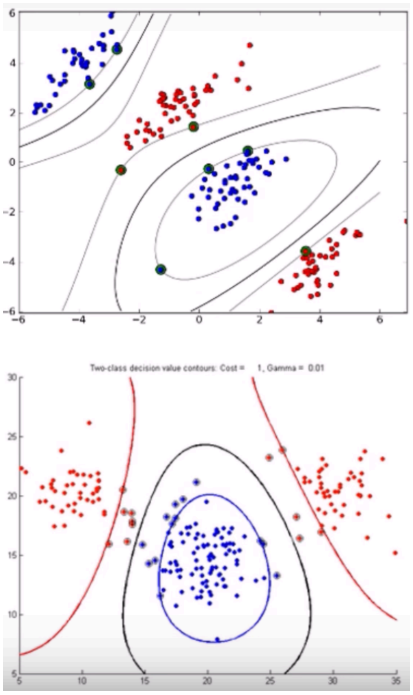
In the spirt of keeping it simple (not really) I chose to use the Support Vector Machines (SVM) classification method as suggested.

The following is my understanding of how the method works.

SVM is a supervised machine learning algorithm. It can be used for classification and regression problems. Since it supervised learning we will need labeled data. Then through computations try and optimally fit a "hyperplane" line (if linear) or curve (if nonlinear) between the group(s) or clusters that are distinctly labeled with the widest margin. Apparently, *SVM achieves this with nonlinear challenges by using a Kernel trick which uses a different "implicit mapping" and taking higher dimensional feature spaces to arrive at that classification* (L) (If you understood that, you are in a better position than me…) You can watch the posted below in the footnotes to learn more.



[2]



[3]

[2] (L)Support Vector Machines (SVM) Overview and Demo using R (Minute 1:10) https://www.youtube.com/watch?v=ueKqDlMxueE

[3] (L)Support Vector Machines (SVM) Overview and Demo using R (Minute 4:30) https://www.youtube.com/watch?v=ueKqDlMxueE

II. Apply methodology (SVM) to chosen data set.

MLB PITCH TYPE SMV CLASSIFICATION

For my data set I chose to go with the baseball sports data. It was extracted from Google Cloud Data using SQL.

I extracted what I thought would be useful data in predicting baseball pitchtypes. And for the purpose of SVM, I will try and build 2 models. One will predict the pitchtype with current balls and current strikes. And the other will predict (I should say post-predict) pitchtype with pitch speed and pitch zone. I thought I should also include whether or not the game is currently tied, a save situation (a team leads by less than 3 runs), or a complete wash (a team leads by more than 3 runs). Below is my SQL query:

```
SELECT
  pitchTypeDescription as ptDescription,
  pitchSpeed,
  pitchZone,
  startingBalls,
  startingStrikes,
  startingOuts,
  (if (ABS(homeCurrentTotalRuns - awayCurrentTotalRuns) BETWEEN 1 AND 3, 1, 0)) as
saveSit,
  (if (homeCurrentTotalRuns - awayCurrentTotalRuns = 0, 1, 0)) as tieGame,
  (if (ABS(homeCurrentTotalRuns - awayCurrentTotalRuns) > 3, 1, 0)) as domSit
FROM
  `bigquery-public-data.baseball.games_wide`
WHERE
  atBatEventType = "PITCH"
  AND
  pitchTypeDescription != "Other"
  AND
  pitchTypeDescription != "Intentional Ball"
  AND
  pitchSpeed > 50
Order by
  pitchSpeed
```

Row	ptDescription	pitchSpeed	pitchZone	startingBalls	startingStrikes	startingOuts	saveSit	tieGame	domSit
1	Fastball	70	0	1	0	2	1	0	0
2	Changeup	75	0	0	1	0	0	0	1
3	Fastball	78	0	0	1	0	0	0	1
4	Fastball	96	0	1	2	0	1	0	0
5	Curveball	57	1	0	0	0	1	0	0
6	Curveball	61	1	0	2	2	1	0	0

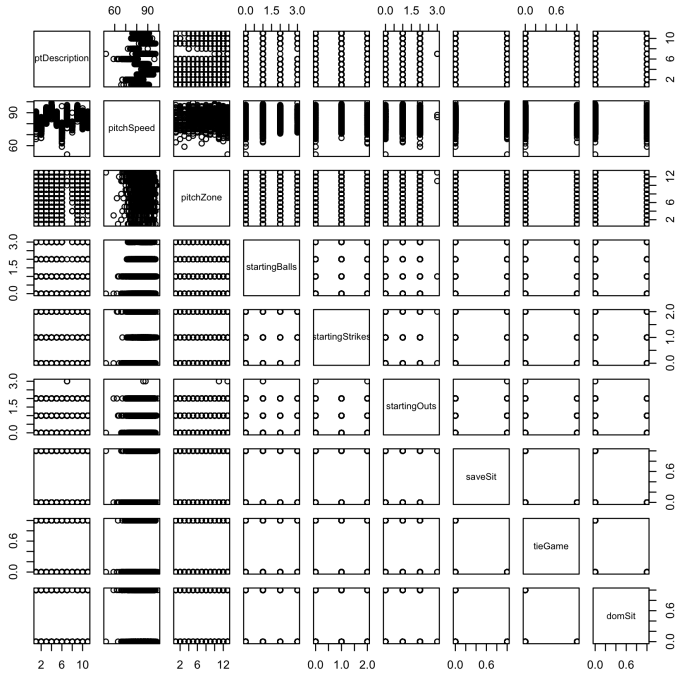
712,816 rows of data!
There 10 different classifications but 357,000 rows are fastballs alone!
Seems like I may need to wrangle some data in R and even things up! Otherwise the model will always predict fastball.

The pitch class with lowest number of rows is “pitchout” with 171 rows. So, I decided to randomly select 171 rows of each.

- I first changed the dataframe into a data.table so I can filter by pitchtype.
- randomly removed the excess rows from all the pitchtypes
- plotted the results on a matrix

```
> mlb2 <- data.table(mlb)
> mlb = read.csv('/Users/johnortiz/Desktop/mlb_project.csv')
> mlb2 <- data.table(mlb)
> mlb2 = mlb2[-sample(which(ptDescription=="Forkball"), 57)]
> mlb2 = mlb2[-sample(which(ptDescription=="Knuckleball"), 4239)]
> mlb2 = mlb2[-sample(which(ptDescription=="Splitter"), 10261)]
> mlb2 = mlb2[-sample(which(ptDescription=="Cutter"), 34483)]
> mlb2 = mlb2[-sample(which(ptDescription=="Sinker"), 45766)]
> mlb2 = mlb2[-sample(which(ptDescription=="Changeup"), 72788)]
> mlb2 = mlb2[-sample(which(ptDescription=="Curveball"), 77690)]
> mlb2 = mlb2[-sample(which(ptDescription=="Slider"), 108385)]
> mlb2 = mlb2[-sample(which(ptDescription=="Fastball"), 357409)]
> dim(mlb2)
[1] 1738    9
> plot(mlb2)
```

- There doesn’t seem to be a pair of variables that can accurately predict the target variable…
- I am going to go ahead and predict a low accuracy rate with these variables…



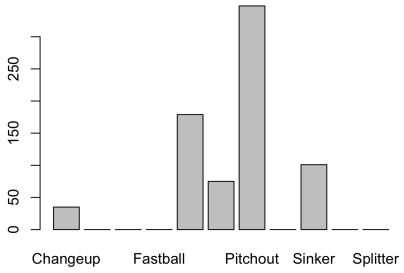
- I then created a training set and test set.
- To use SMV you must first “install.packages(“e1071”)” and load it “library(e1071)
- Ran SMV
- I tried to plot SMV result, but it didn’t work!
- Then I remembered that I can only run SMV on only 2 dependent variables. So, I chose startingBalls and startingStrikes.
- Then that started to give me issues! I figured it was due to the fact that the data was in data table form, so I exported the modified data to a CSV and voila, it worked!
- Again, I split the data into training and test sets, ran SMV and plotted the results.

```
> s = sample(1738, 1000)
> mlb_train = mlb2[s,]
> mlb_test = mlb2[-s,]
> svmfit = svm(ptDescription~., data=mlb_train, kernel = "linear", cost =.1, scale= FALSE)
> plot(svmfit, mlb_train[,])
Error in plot.svm(svmfit, mlb_train[, ]) : missing formula.
> col = c("startingBalls","startingStrikes","ptDescription")
> mlb_train = mlb2[s,col]
Error in `[.data.table`(mlb2, s, col) :
  j (the 2nd argument inside [...]) is a single symbol but column name 'col' is not found. Perhaps you intended DT[, ..col]. This
difference to data.frame is deliberate and explained in FAQ 1.1.
> write.csv(mlb2, file = "mlb2.csv")
> mlb2 = read.csv('/Users/johnortiz/Desktop/mlb2.csv')
> mlb_train = mlb2[s,col]
> mlb_test = mlb2[-s,col]
> svmfit = svm(ptDescription~., data=mlb_train, kernel = "linear", cost =.1, scale= FALSE)
> plot(svmfit, mlb_train[,col])
```

- I went on and did a cross validation on the “cost” parameter with the “tune” function
- The best parameter for cost was 10, so I adjusted, ran SVM and plotted the results
- The boundaries look more defined!

```
> tuned = tune(svm, ptDescription~., data=mlb_train, kernel = "linear", ranges =
list(cost=c(0.001,0.01,0.1,1,10,100)))
> summary(tuned)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.84

- Detailed performance results:
   cost error dispersion
1 1e-03 0.896 0.02590581
2 1e-02 0.900 0.02403701
3 1e-01 0.852 0.02299758
4 1e+00 0.844 0.02633122
5 1e+01 0.840 0.02581989
6 1e+02 0.849 0.02469818

> svmfit = svm(ptDescription~., data=mlb_train, kernel = "linear", cost =10, scale= FALSE)
> plot(svmfit, mlb_train[,col])
```

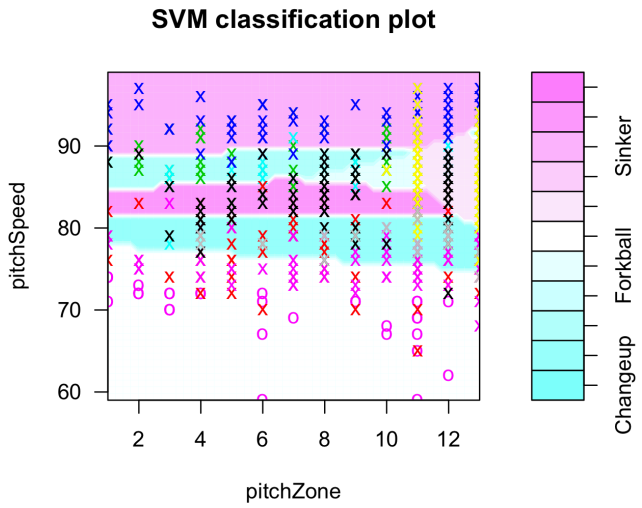
- And finally, here is the good part! Prediction time.
- And Holy Moley was it bad! A 13.14 percent accuracy rate!



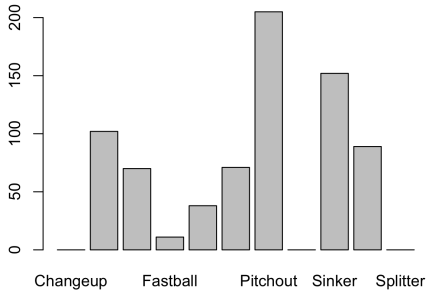

```
> p = predict(svmfit, mlb_test[,col], type="class")
> plot(p)
> table(p, mlb_test[,3])

p            Changeup Curveball Cutter Fastball Forkball Knuckleball Pitchout Screwball Sinker Slider Splitter
  Changeup          5         6      3        2        2           6        0          0      3      5        3
  Curveball         0         0      0        0        0           0        0          0      0      0        0
  Cutter            0         0      0        0        0           0        0          0      0      0        0
  Fastball          0         0      0        0        0           0        0          0      0      0        0
  Forkball         17        20     21       20       30          12        7          2     14      9       27
  Knuckleball      12         4      7       12        8           5       11          1      5      6        4
  Pitchout         27        37     30       41       26          36       47          4     37     30       33
  Screwball         0         0      0        0        0           0        0          0      0      0        0
  Sinker            6         4     15       20        5          12        4          4     10     10       11
  Slider            0         0      0        0        0           0        0          0      0      0        0
  Splitter          0         0      0        0        0           0        0          0      0      0        0
> mean(p== mlb_test[,3])
[1] 0.1314363
```

Next, I chose to do same model but using pitchZone and pitchSpeed as x variables.

```
> col = c("pitchSpeed","pitchZone","ptDescription")
> mlb2 = read.csv('/Users/johnortiz/Desktop/mlb2.csv')
> mlb_train = mlb2[s,col]
> mlb_test = mlb2[-s,col]
> tuned = tune(svm, ptDescription~., data=mlb_train, kernel = "linear", ranges =
list(cost=c(0.001,0.01,0.1,1,10,100)))
> summary(tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.703

- Detailed performance results:
   cost error dispersion
1 1e-03 0.897 0.02311805
2 1e-02 0.747 0.06429965
3 1e-01 0.703 0.07409453
4 1e+00 0.719 0.06674162
5 1e+01 0.723 0.07103207
6 1e+02 0.725 0.06980131

> svmfit = svm(ptDescription~., data=mlb_train, kernel = "linear", cost =0.1, scale= FALSE)
> plot(svmfit, mlb_train[,col])
> p = predict(svmfit, mlb_test[,col], type="class")
> plot(p)
> mean(p== mlb_test[,3])
[1] 0.2926829
```



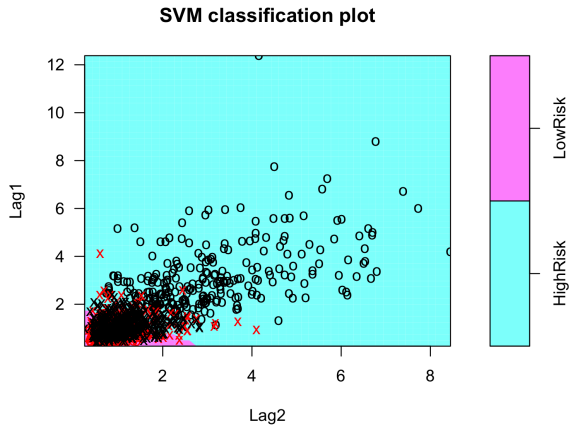
- It seems like this pair of variables did a bit better at 29.26 percent accuracy rate!
- But 29.26% is not impressive by any means.
- Let's compare it with other classification methods (kNN, Naïve Bayes, GLM)
- But first let's apply SMV on the JEC STOCK DATA…

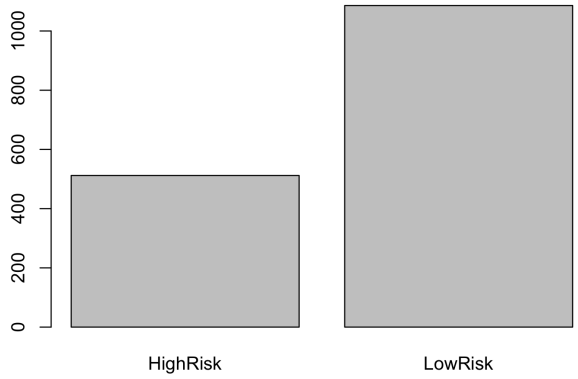


JEC STOCK RANGE SMV CLASSIFICATION

- I created a training set and test set.
- I chose Lag1 and Lag2 as the x-variables to predict the stock's volatility
- Remember! To use SMV you must first "install.packages("e1071")" and load it "library(e1071)
- This time I chose to directly apply the cross validation on the "cost" parameter with the "tune" function
- Best cost parameter was 1
- Then I ran SMV and plotted the results and ran the prediction.
- And Holy Moley was it good! A 70.46 percent accuracy rate!

```
> s2 = sample(2998,1400)
> col2 = c('Lag1','Lag2','Risk')
> JEC_train = JECr[s2,col2]
> JEC_test = JECr[-s2,col2]
> tuned = tune(svm, Risk~Lag1+Lag2, data=JEC_train, kernel = "linear", ranges =
list(cost=c(0.001,0.01,0.1,1,10,100)))
> summary(tuned)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.2757143

- Detailed performance results:
   cost     error dispersion
1 1e-03 0.3728571 0.03623431
2 1e-02 0.2985714 0.04651092
3 1e-01 0.2771429 0.05235930
4 1e+00 0.2757143 0.04904993
5 1e+01 0.2764286 0.04809877
6 1e+02 0.2757143 0.04764285

> svmfit2 = svm(Risk~Lag1+Lag2, data=JEC_train, kernel = "linear", cost =1, scale= FALSE)
> plot(svmfit2, JEC_train[,col2])
> p = predict(svmfit2, JEC_test[,col2], type='class')
> plot(p)
> table(p,JEC_test[,3])

p          HighRisk LowRisk
  HighRisk      420      92
  LowRisk       380     706
> mean(p==JEC_test[,3])
[1] 0.7046308
```





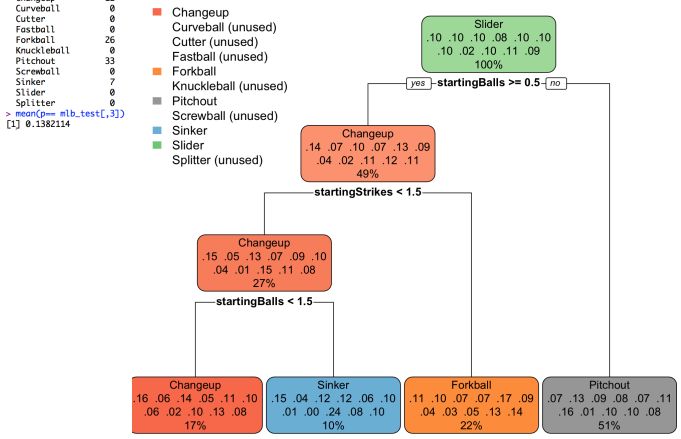
BONUS!!!! I decided to use the decision tree classification method just for “kicks”

MLB PITCH TYPE DECISION TREE CLASSIFICATION

startingStrikes and startingBalls

- Running the decision tree using the startingStrikes and startingBalls as x-variables yielded a classification prediction accuracy of 13.82%
- Which is about the same as SMV… No change here.

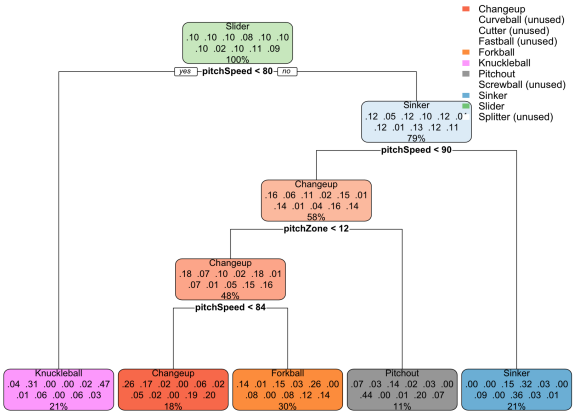
```
> col = c("startingBalls","startingStrikes","ptDescription")
> mlb_train = mlb2[s,col]
> mlb_test = mlb2[-s,col]
> dtm <- rpart( ptDescription ~., data = mlb_train, method='class')
> rpart.plot(dtm)
> p = predict(dtm, mlb_test[,col], type="class")
> table(p, mlb_test[,3])

p             Changeup Curveball Cutter Fastball Forkball Knuckleball Pitchout Screwball Sinker Slider
  Changeup          18         8     19       23       12           14       15         4     12     13
  Curveball          0         0      0        0        0            0        0         0      0      0
  Cutter             0         0      0        0        0            0        0         0      0      0
  Fastball           0         0      0        0        0            0        0         0      0      0
  Forkball          15        19     20       18       30           12        7         2     13      8
  Knuckleball        0         0      0        0        0            0        0         0      0      0
  Pitchout          27        37     30       41       26           36       47         4     37     30
  Screwball          0         0      0        0        0            0        0         0      0      0
  Sinker             7         7      7       13        3            9        0         1      7      9
  Slider             0         0      0        0        0            0        0         0      0      0
  Splitter           0         0      0        0        0            0        0         0      0      0

p             Splitter
  Changeup          12
  Curveball          0
  Cutter             0
  Fastball           0
  Forkball          26
  Knuckleball        0
  Pitchout          33
  Screwball          0
  Sinker             7
  Slider             0
  Splitter           0
> mean(p== mlb_test[,3])
[1] 0.1382114
```



pitchSpeed and pitchZone

- Running the decision tree using the pitchSpeed and pitchZone as x-variables yielded a classification prediction accuracy of 32.24%
- Which is about 2% increase vs SMV…Again, Not really a significant change here.

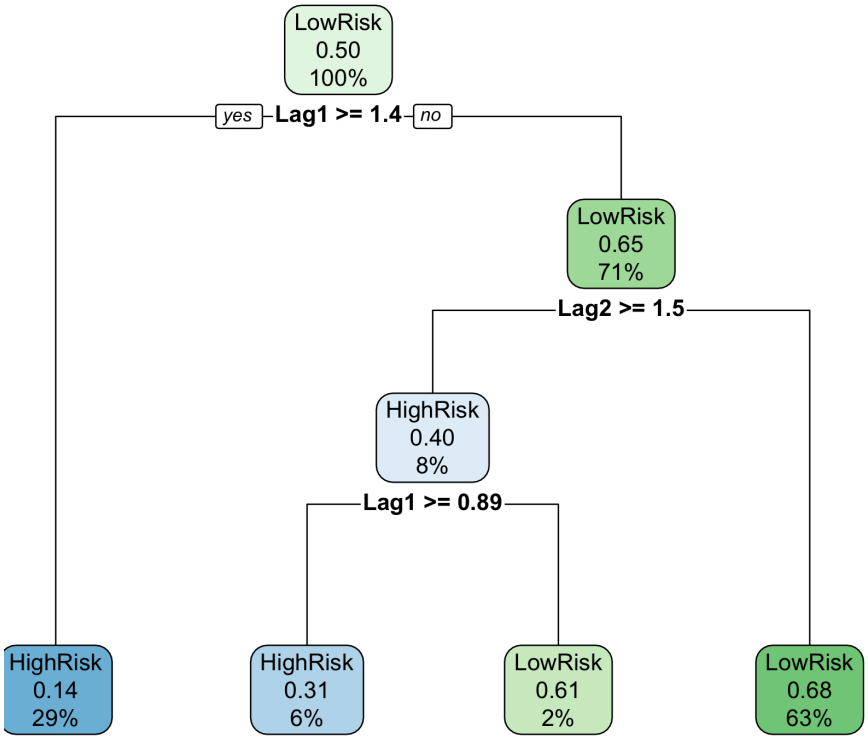
```
> col = c("pitchSpeed","pitchZone","ptDescription")
> mlb_train = mlb2[s,col]
> mlb_test = mlb2[-s,col]
> dtm <- rpart( ptDescription ~., data = mlb_train, method='class')
> plot(dtm)
> rpart.plot(dtm)
> table(p, mlb_test[,3])

p             Changeup Curveball Cutter Fastball Forkball Knuckleball Pitchout Screwball Sinker Slider
  Changeup          25        23      5        0        2            2        5         1      2     15
  Curveball          0         0      0        0        0            0        0         0      0      0
  Cutter             0         0      0        0        0            0        0         0      0      0
  Fastball           0         0      0        0        0            0        0         0      0      0
  Forkball          24         3     42       12       62            0       18         0      7     27
  Knuckleball        8        39      0        1        2           68        3         9      0      4
  Pitchout           8         6      9        2        1            1       28         0      5     13
  Screwball          0         0      0        0        0            0        0         0      0      0
  Sinker             2         0     20       80        4            0       15         1     55      1
  Slider             0         0      0        0        0            0        0         0      0      0
  Splitter           0         0      0        0        0            0        0         0      0      0

p             Splitter
  Changeup          33
  Curveball          0
  Cutter             0
  Fastball           0
  Forkball          30
  Knuckleball        4
  Pitchout           7
  Screwball          0
  Sinker             4
  Slider             0
  Splitter           0
> mean(p== mlb_test[,3])
[1] 0.3224932
```



JEC STOCK RANGE DECISION TREE CLASSIFICATION

- Running the decision tree using the Lag1 and Lag2 as x-variables yielded a classification prediction accuracy of 68.83%
- Which is about 1.5% decrease vs SMV…Again, Not really a significant change here. But still an effective model

```
> dtm2 = rpart(Risk~Lag1+Lag2, data=JEC_train, method='class')
> rpart.plot(dtm2)
> p = predict(dtm2, JEC_test[,col2], type='class')
> table(p,JEC_test[,3])

p          HighRisk LowRisk
  HighRisk      436     134
  LowRisk       364     664
> mean(p==JEC_test[,3])
[1] 0.6883605
```



Apply Methodology (bestglm)

MLB DATASET

Given that my chosen dataset does not have a binomial target variable I am afraid I won't be able to use the bestglm function! So, looked for an alternative function that can perform best model selection a multinomial classification problem. I came across GLMNET! Then I saw this:
It looks like it is used for regression and not classification? I not I will probably need to create dummy variables, etc.

```
glmnet(x, y, family=c("gaussian","binomial","poisson","multinomial","cox","mgaussian"),
    weights, offset=NULL, alpha = 1, nlambda = 100,
    lambda.min.ratio = ifelse(nobs
```

```
#multinomial
g4=sample(1:4,100,replace=TRUE)
fit3=glmnet(x,g4,family="multinomial")
fit3a=glmnet(x,g4,family="multinomial",type.multinomial="grouped")
#poisson
N=500; p=20
nzc=5
x=matrix(rnorm(N*p),N,p)
beta=rnorm(nzc)
f = x[,seq(nzc)]%*%beta
mu=exp(f)
y=rpois(N,mu)
fit=glmnet(x,y,family="poisson")
plot(fit)
pfit = predict(fit,x,s=0.001,type="response")
plot(pfit,y)
```

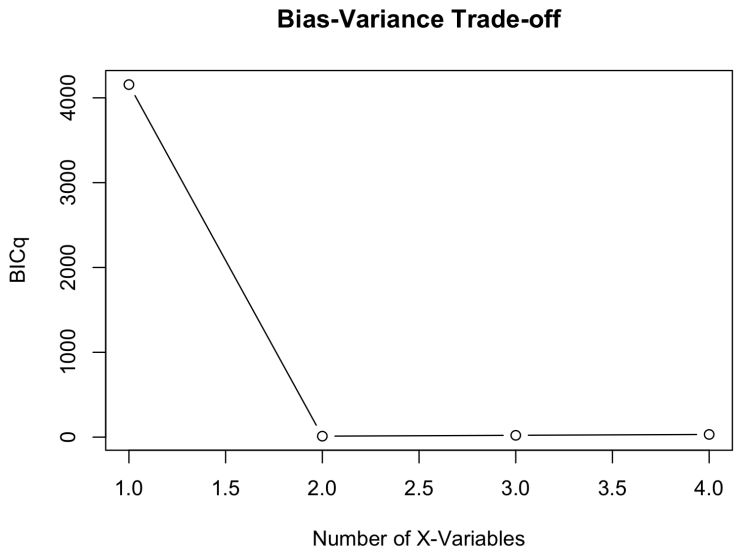
https://www.rdocumentation.org/packages/glmnet/versions/2.0-16/topics/glmnet

JEC STOCK RANGE DATA

- I feel like including this is not necessary. The data only has 4 variables where the risk variable (target) is directly derived from one of them; JECrange.
- So natural the best model would be the one with JECrange and it alone. Which is what happened.

```
> best.out=bestglm(JECr,IC='BICq',family=binomial,TopModels=1)
Morgan-Tatar search since family is non-gaussian.
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
3: glm.fit: algorithm did not converge
4: glm.fit: fitted probabilities numerically 0 or 1 occurred
5: glm.fit: algorithm did not converge
6: glm.fit: fitted probabilities numerically 0 or 1 occurred
7: glm.fit: algorithm did not converge
8: glm.fit: fitted probabilities numerically 0 or 1 occurred
> best.out
BICq(q = 0.25)
BICq equivalent for q in (0, 0.981334097942187)
Best Model:
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1984831   478806.1 -4.145375 3.392587e-05
JECrange     1927019   464860.1  4.145374 3.392602e-05
> best.out=bestglm(JECr,IC='CV',family=binomial,TopModels=1)
Morgan-Tatar search since family is non-gaussian.
There were 50 or more warnings (use warnings() to see the first 50)
> best.out
CVd(d = 2571, REP = 1000)
BICq equivalent for q in (0, 0.981334097942187)
Best Model:
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1984831   478806.1 -4.145375 3.392587e-05
JECrange     1927019   464860.1  4.145374 3.392602e-05
```

```
> best.out$Subsets$BICq
[1] 4156.11049   11.36234   21.48400   31.65149
> plot(best.out$Subsets$BICq,type='b',xlab='Number of X-Variables',ylab='BICq',main='Bias-Variance Trade-
off')
```



- Reviewing the Bias-Variance Trade-off we notice that the model with 2 x-variables or more starts outperforms the model with the single JECrange variable!
- Given what bestglm outputted, this doesn't make since to me. I think I may have used the model incorrect. Or I just don't know how to interpret the Bia-Variance Trade-off…
- From what I understand is since the best model given by bestglm has one x-variable, the BICq of 1 should be the lowest. But the graph doesn't show this!

III. Compare the performance of your new classification method against those we have used. (kNN, Naïve Bayes, GLM)

MLB DATASET

```
> head(mlb2)
  ptDescription pitchSpeed pitchZone startingBalls startingStrikes startingOuts saveSit tieGame domSit
1     Changeup         72        12             2               1            1       0       0      1
2     Changeup         74        11             0               1            1       1       0      0
3     Changeup         75        11             3               2            1       0       1      0
4     Changeup         76         8             2               2            1       0       1      0
5     Changeup         76         9             1               0            0       0       0      1
6     Changeup         77         9             2               2            2       1       0      0
```

kNN

- kNN is not a good model for the MLB Dataset. This model scored an accuracy forecast of 5.69%...
- WHY?? What is going on here?

```
> X.mlb = mlb2[,2:5]
> Y.mlb = mlb2[,1]
> TrainX.mlb = X.mlb[s,]
> TrainY.mlb = Y.mlb[s]
> TestX.mlb = X.mlb[-s,]
> TestY.mlb = Y.mlb[-s]
> knn.pred = knn(TrainX.mlb,TestX.mlb,TrainY.mlb, 25)
> table.out= table(knn.pred, TestY.mlb)
> (table.out[1,1]+table.out[2,2])/sum(table.out)
[1] 0.05691057
```

Naïve Bayes

```
> model = train(TrainX.mlb,TrainY.mlb,'nb',trControl=trainControl(method='cv',number=10))
> table.out=table(predict(model$finalModel, TestX.mlb)$class, TestY.mlb)
> table.out
             TestY.mlb
              Changeup Curveball Cutter Fastball Forkball Knuckleball Pitchout Screwball Sinker Slider Splitter
  Changeup          12         9      1        0        3           0        0         0      1     11       11
  Curveball         10        29      2        0        1          19        1         6      0      4       10
  Cutter             5         0     21       11       12           0        0         1      9     11       10
  Fastball           0         0      1       18        0           0        0         0      6      0        0
  Forkball          17         4     21        6       44           0        1         0      4     13       16
  Knuckleball        4        16      0        0        1          49        0         2      0      1        2
  Pitchout          16         8     20       18        6           3       67         2     17     17       18
  Screwball          0         0      0        1        0           0        0         0      0      0        0
  Sinker             0         0     10       40        2           0        0         0     32      0        3
  Slider             0         2      0        1        0           0        0         0      0      2        2
  Splitter           3         3      0        0        2           0        0         0      0      1        6
> (table.out[1,1]+table.out[2,2])/sum(table.out)
[1] 0.05555556
```

- Naïve Bayes is also not a good model for the MLB Dataset. This model scored an accuracy forecast of 5.55%...
- Again, WHY?? What is going on here?

Given that my chosen dataset does not have a binomial target variable I am afraid I won't be able to use the glm function!

JEC STOCK DATASET

kNN

```
> y.train = Y.JEC[train]
> x.test = X.JEC[-train,]
> y.train = Y.JEC[-train]
> x.train = X.JEC[train,]
> y.train = Y.JEC[train]
> x.test = X.JEC[-train,]
> y.test = Y.JEC[-train]
> model = train(x.train,y.train,'nb',trControl=trainControl(method='cv',number=10))
Warning messages:
1: In FUN(X[[i]], ...) :
  Numerical 0 probability for all classes with observation 112
2: In FUN(X[[i]], ...) :
  Numerical 0 probability for all classes with observation 45
3: In FUN(X[[i]], ...) :
  Numerical 0 probability for all classes with observation 125
4: In nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  :
  There were missing values in resampled performance measures.
> table(predict(model$finalModel, x.test)$class, y.test)
         y.test
          HighRisk LowRisk
  HighRisk      293      69
  LowRisk       207     430
> table.out = table(predict(model$finalModel, x.test)$class, y.test)
> (table.out[1,1]+table.out[2,2])/sum(table.out)
[1] 0.7237237
```

Naïve Bayes

```
> InSample=Shuffle[1:1400]
> OutSample=Shuffle[1401:2998]
> X.JEC=JECrange[,3:4]
> Y.JEC=JECrange[,2]
> median(Y.JEC)
[1] 1.030001
> Y.JEC[Y.JEC>1.030001]="HighRisk"
> Y.JEC[Y.JEC<=1.030001]="LowRisk"
> Y.JEC[1:6]
[1] "HighRisk" "HighRisk" "HighRisk" "HighRisk" "HighRisk" "LowRisk"
> as.factor(Y.JEC[1:6])
[1] HighRisk HighRisk HighRisk HighRisk HighRisk LowRisk
Levels: HighRisk LowRisk
> Y.JEC = as.factor(Y.JEC)
> TrainX.JEC = X.JEC[InSample,]
> TrainY.JEC = Y.JEC[InSample]
> TestX.JEC = X.JEC[OutSample,]
> TestY.JEC = Y.JEC[OutSample]
> knn.pred = knn(TrainX.JEC,TestX.JEC,TrainY.JEC,25)
> table.out = table(knn.pred,TestY.JEC)
> (table.out[1,1]+table.out[2,2])/sum(table.out)
[1] 0.6996245
```

- As you can see, both kNN and Naïve Bayes models have acceptable accuracy forecast rates for the JEC Stock data set with 72.37% and 69.96% respectfully.

GLM

- Here we ran GLM on the same data.
- As you can see the model worked as it outputted the Coefficients (I honestly don't know if I did this right)
- But while trying to use the model to predict the target variable using probabilities I kept getting the same error. INVALID FACTOR LEVEL.
- So, I was unable to perform the model valuation…

```
> glm.fit=glm(Risk~Lag1+Lag2, data=JEC_train, family=binomial)
> summary(glm.fit)

Call:
glm(formula = Risk ~ Lag1 + Lag2, family = binomial, data = JEC_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0077  -0.9710   0.5220   0.8784   2.8274

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   2.7708     0.1858  14.910  < 2e-16 ***
Lag1         -1.5288     0.1520 -10.058  < 2e-16 ***
Lag2         -0.8024     0.1322  -6.068  1.3e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1940.8  on 1399  degrees of freedom
Residual deviance: 1476.0  on 1397  degrees of freedom
AIC: 1482

Number of Fisher Scoring iterations: 6

> glm.probs = predict(glm.fit,type='response')
> glm.probs[1:4]
      897      2302      2813      1100
0.5442726 0.6878288 0.7913089 0.4530329
> glm.forecast[glm.probs>.5]='High'
Warning message:
In `[<-.factor`(`*tmp*`, glm.probs > 0.5, value = "High") :
  invalid factor level, NA generated
```

COMPARISON SUMMARY

Dataset	Classification Method	Forecast Accuracy %
MLB S/B	SMV	13.14%
MLB S/B	Decision Tree	13.82%
MLB PS/PZ	SMV	29.26%
MLB PS/PZ	Decision Tree	32.24%
MLB	kNN	5.69%
MLB	Naïve Bayes	5.55%
JECrange	SMV	70.46%
JECrange	Decision Tree	68.83%
JECrange	kNN	72.37%
JECrange	Naïve Bayes	69.96%

- As you can see, the best pair of x-variables for the MLB dataset was the pitch speed and pitch zone. But this is not useful considering the pitch has to be thrown. We are trying to predict which pitch will be thrown next! So, I must take another look at the raw data and select other variables. Perhaps I can run a multinomial bestglm model on it!
- For some reason the kNN and Naïve Bayes model did work so well. I think it's because it was using all the x -variables to fit the model! …
- All models worked pretty well for JECrange! Hooray! They were all very close, so choosing either model is fine.

Works Cited

L, Melvin. *Support Vector Machines (SVM) Overview and Demo using R*. 16 5 2016. YouTube Video. <https://www.youtube.com/watch?v=ueKqDlMxueE>.

Mutliple. n.d. <https://en.wikipedia.org/wiki/Mariano_Rivera>.