Basketball Wins and Beating the Odds

Jason Blunk and Adam Hall

MA 5790

October 2020

Abstract:

The NBA has been the fastest accelerating professional sports network over the past decade, with much assistance from a strong prevalence on social media as well as a focus on a handful of stars. With this increased popularity, comes increased activity in the gambling industry. The sports gambling industry is a titan in Las Vegas, with every game having multiple avenues to wager and millions of dollars at stake. Our goal in this study is to help examine the relationships between rapidly stabilizing performance metrics for basketball teams, and simulate the business conducted in this industry. These stabilizing metrics involve field goal rates, which are percentages. The goal being that these rates are relatively stable quickly in a season, so the rest of the season could be predicted with reasonable accuracy. While most modern gambling is done on a line, ultimately playoff performance and victories are the largest individual pots of money available - especially in basketball which has a hyper-focus on playoffs and championships (with star players even opting not to play during the regular season in order to be as ready for the playoffs as possible).

Table of Contents

Background

Basketball is one of the largest sports in the world and needs very little introduction. Household names include Michael Jordan, Lebron James and so on. However, the interesting things about the game of basketball is how little the rules of point generation and the game itself have changed over time. While the rules of how shots can come about might be different, the same shot which would be worth three points in 2002 is still worth three in 2020. Teams like the Moneyball A's of 2002 have started mathematically optimizing their offense with more and more advanced statistics, proving that even a low-budget sports team can compete based on nothing but advanced understanding of the game itself. These changes to the game exacerbate but do not invalidate the premise that our metrics measure what is the same event independent of the year.

And these metrics are very much a part of the current NBA. The Dallas Mavericks were the most efficient offense in the history of basketball despite having only two notable players who are widely considered inferior to many other players in the league. James Harden of the Houston Rockets has refined his game to almost a parody of the current NBA in which he only takes shots with the highest expected returns. Some teams are slower to follow suit, but ultimately the meta in which a game takes place is not a variable we consider, since we are looking only at performance. If someone only takes shots of higher return, or if someone is more apt at making them, these will already be reflected in the variables and as such any strategies are accounted for. Basketball, like other sports, is inherently noisy, and it's also very easy to predict a victor of a game post-hoc (the team with the most number of points, which is a statistic by definition, is always classified as the winner) so it is important to only include variables which are more likely to be static, but also likely to have a predictable variance. The variables we selected are the ones we considered to be the variables which fit this description as well as possible.

Variable Introduction and Definitions

For those unfamiliar with advanced basketball analysis, or even basketball in general, the actual metrics we use need to be introduced. In essence, a team with possession of the basketball can attempt a shot with two points, or a shot with a higher degree of difficulty worth three. There are also fouls which can sometimes result in a number of free throws being made, each worth one point. The metrics most readily available and relevant for these are free throw percentage, 3 point percentage and field goal percentage. While field goal percentage includes all non-free throw shots, this is sufficient information to infer the percentage of two pointers, thus encompassing all ways in which the ball is scored. We also include, for reference, the total number of assists (passes made before a successful field goal) and rebounds (the total number of times a ball is recovered after a missed attempt). Through this, we believe we have a very good picture of the performance of a 5 man squad of players.

Basketball has a very deserved reputation of being a superstar centric league. As such, with the very safe and correct assumption that superstars tend to start games, we needed to have a metric to help describe the team as a whole, and have done so by splitting these metrics for both the whole team, as well as the bench (giving us the information we need to determine starter performance through the most readily available metrics). Every metric is applied to the bench as well as the team as a whole. Finally, these same metrics are given to the both the away team and home team.

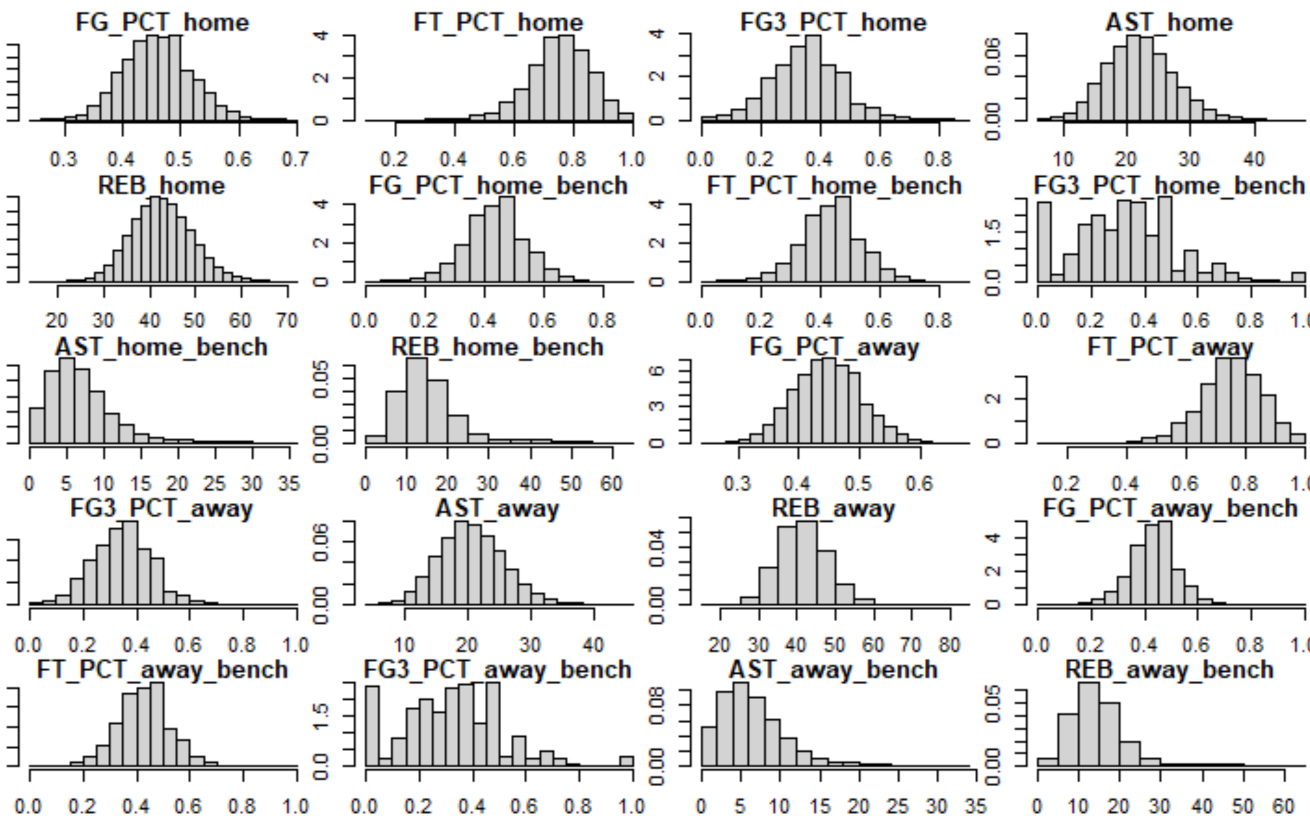
Variable Name	Description
FG_PCT_home	Field Goal Percentage for the Home Team
FT_PCT_home	Free Throw Percentage for the Home Team
FG3_PCT_home	Three Pointer Percentage for the Home Team
AST_home	Total Assists for the Home Team
REB_home	Total Rebounds for the Home Team
FG_PCT_home_bench	Field Goal Percentage for the Home Team Bench
FT_PCT_home_bench	Free Throw Percentage for the Home Team Bench
FG3_PCT_home_bench	Three Pointer Percentage for the Home Team Bench
AST_home_bench	Total Assists for the Home Team Bench
REB_home_bench	Total Rebounds for the Home Team Bench
FG_PCT_away	Field Goal Percentage for the Away Team
FT_PCT_away	Free Throw Percentage for the Away Team
FG3_PCT_away	Three Pointer Percentage for the Away Team
AST_away	Total Assists for the Away Team
REB_away	Total Rebounds for the Away Team
FG_PCT_away_bench	Field Goal Percentage for the Away Team Bench
FT_PCT_away_bench	Free Throw Percentage for the Away Team Bench
FG3_PCT_away_bench	Three Pointer Percentage for the Away Team Bench
AST_away_bench	Total Assists for the Away Team Bench
REB_away_bench	Total Rebounds for the Away Team Bench

Preprocessing of the Predictors

As noted earlier, the sort of 'basketball meta' has changed. We noticed one thing almost immediately with the data: sometimes there would be instances of teams which did not have a field goal attempt, usually a three pointer, from any bench player. As such the rate would necessarily be undefined. These games were removed from the dataset and we believe it to be a safe removal because the game has, in essence, changed permanently and while there might be a freak game once a decade which somehow would fall under this category, it is entirely unlikely. The data points are not likely to be something replicated for anything in the future and thus could safely be removed for purposes of predictive analytics.

Investigating the Predictors

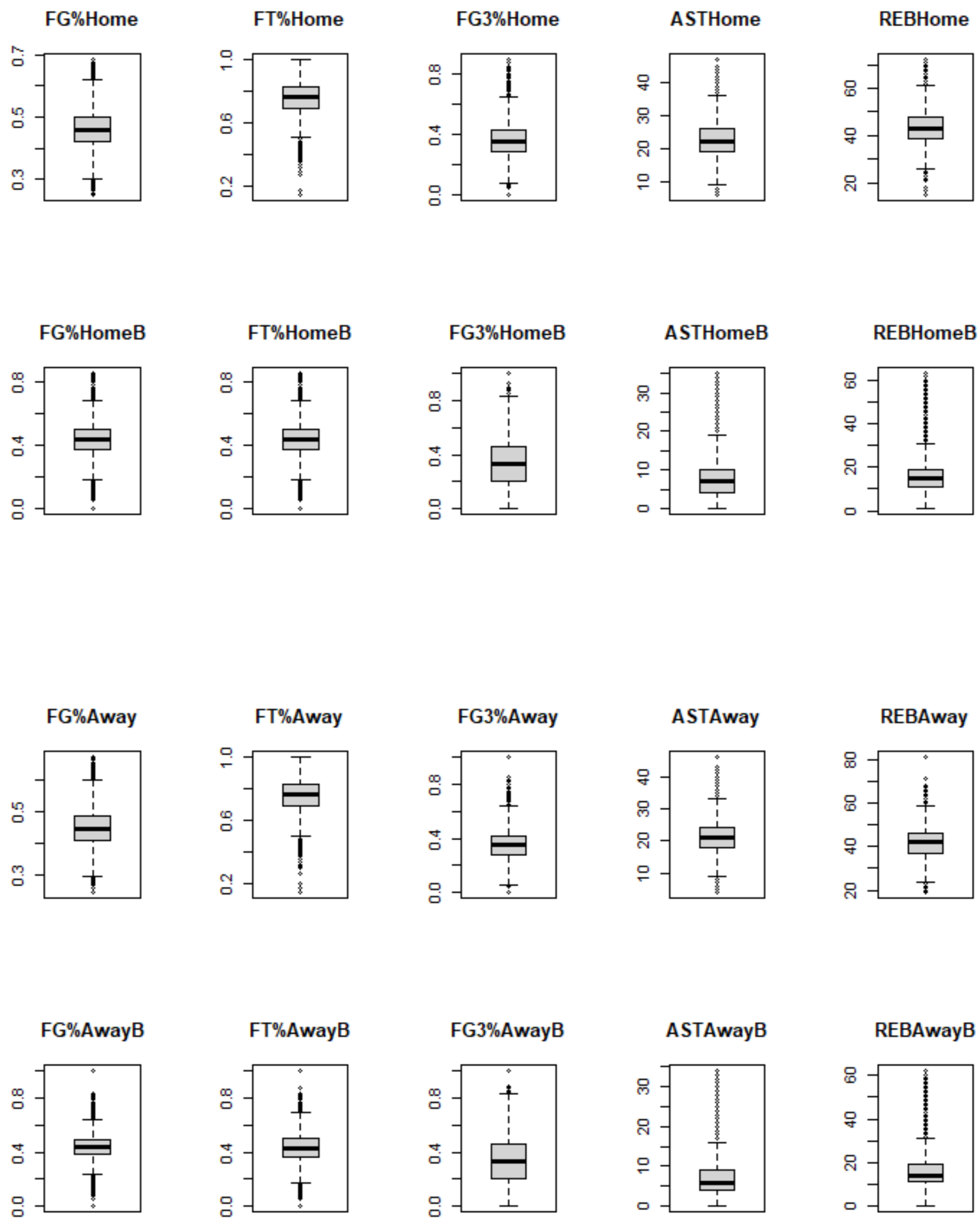
Skewness



Looking at our data we can see that there tends to be skew in mostly every rate variable, and some slight skew in some of the counting variables (Rebounds and Assists). Getting rid of the skew is important, so we will be using a box-cox transformation. Our skewness values are as follows:

FG_PCT_home	FT_PCT_home	FG3_PCT_home	AST_home	REB_home
0.09720	-0.38088	0.06136	0.26080	0.17325
FG_PCT_home_bench	FT_PCT_home_bench	FG3_PCT_home_bench	AST_home_bench	REB_home_bench
-0.11784	-0.11784	0.38447	1.52803	1.76064
FG_PCT_away	FT_PCT_away	FG3_PCT_away	AST_away	REB_away
0.08157	-0.37296	0.08576	0.25408	0.21852
FG_PCT_away_bench	FT_PCT_away_bench	FG3_PCT_away_bench	AST_away_bench	REB_away_bench
-0.15947	-0.02227	0.42845	1.43467	1.76951

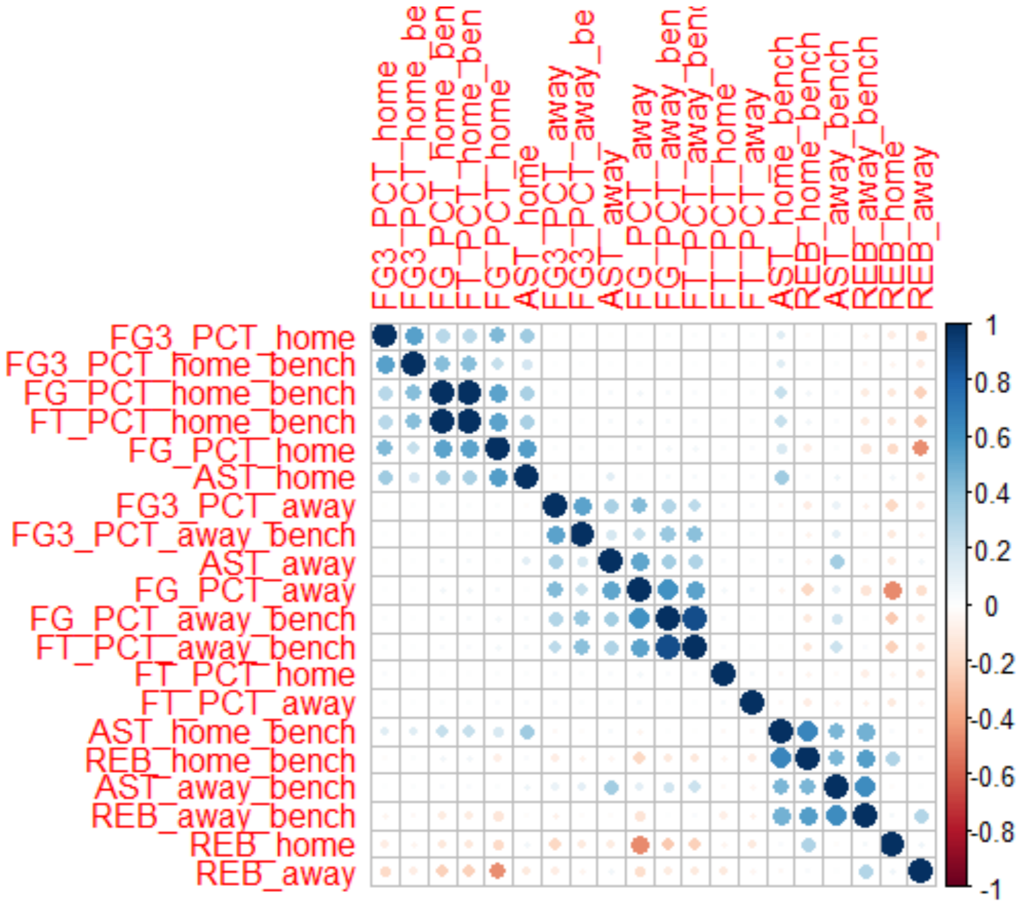
Outliers



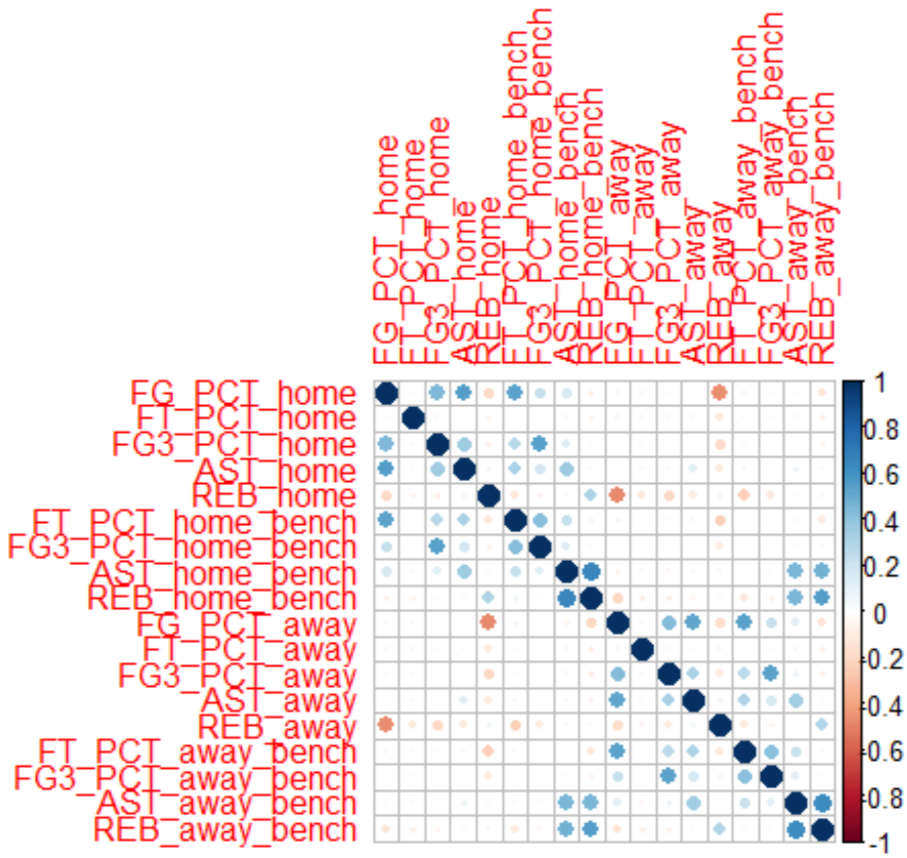
Since many of our predictors contain outliers, we decided to use a Spatial Sign transformation.

Correlations

A correlation plot of the 20 predictors was created to examine any relationship between predictors. In essence, many of the variables are measuring the ability to put a basketball into a basketball hoop, so there is a high possibility that there is correlation between variables. Our initial guess was correct, as there was a correlation between two pairs of variables (one for home and one for away):



It makes sense that there is a correlation between free throws and two pointers for the bench, since both are considered to be “easier” shots. The inability to make a free throw is something that superstars only tend to exhibit, and only by survivor bias (bench players who cannot make free throws are likely not going to be basketball players for much longer). These were removed at a threshold of 0.85, which seems high, but part of what makes this data so neat is that there is very strong independence between our variables. Of encouraging note: the higher field goal percentages for the home team and away team leads to a lower rebound count. This is entirely expected. Our units aren’t the same, with rebounds being a count and scoring being rates.



This final plot gets rid of all heavy correlations. We do see some instances where there are blocks of concentrated localized correlations, however PCA analysis suggests that we would need 17 out of our 18 variables for 95% of the variance. This is only slightly better than suggesting each variable has an equal, independent contribution to the variance and thus the simpler model is just to include all variables as they are.

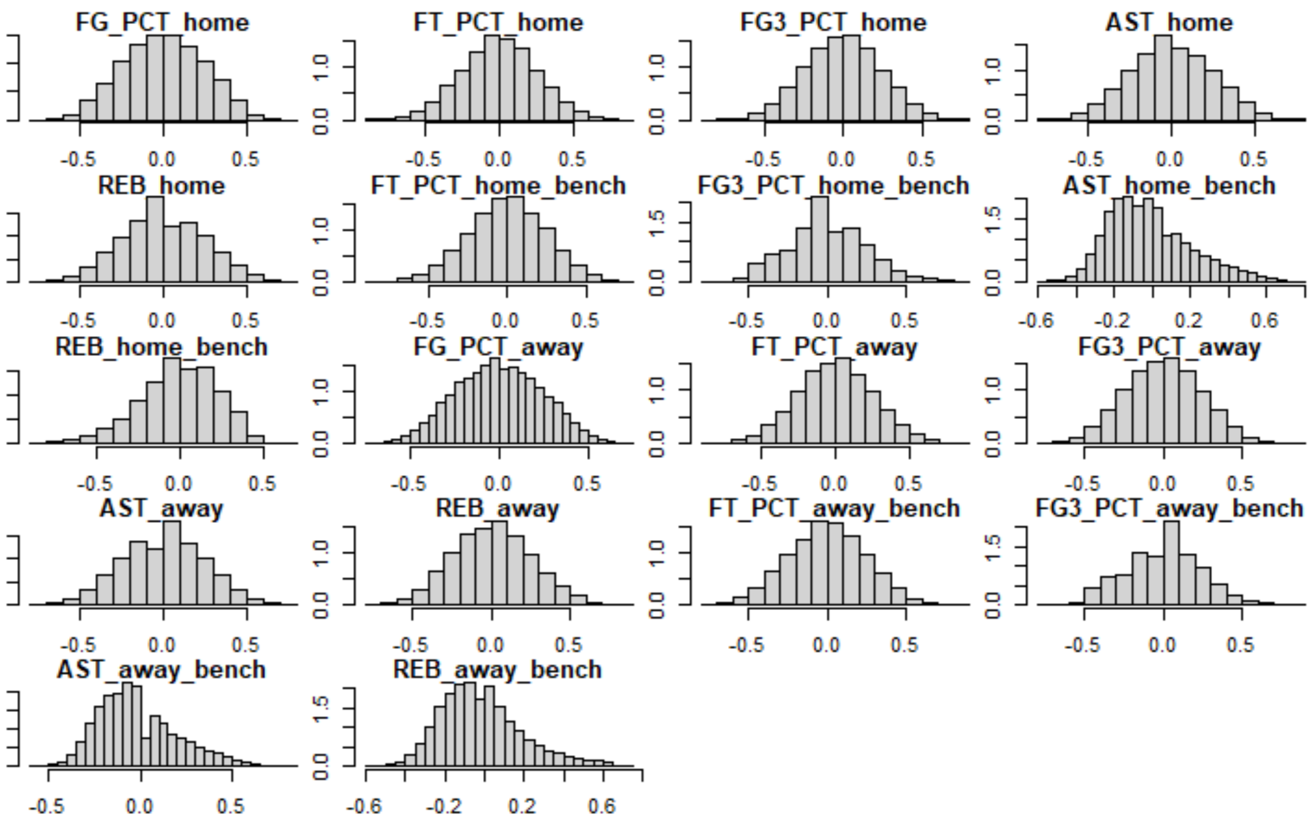
Transformations

Skewness

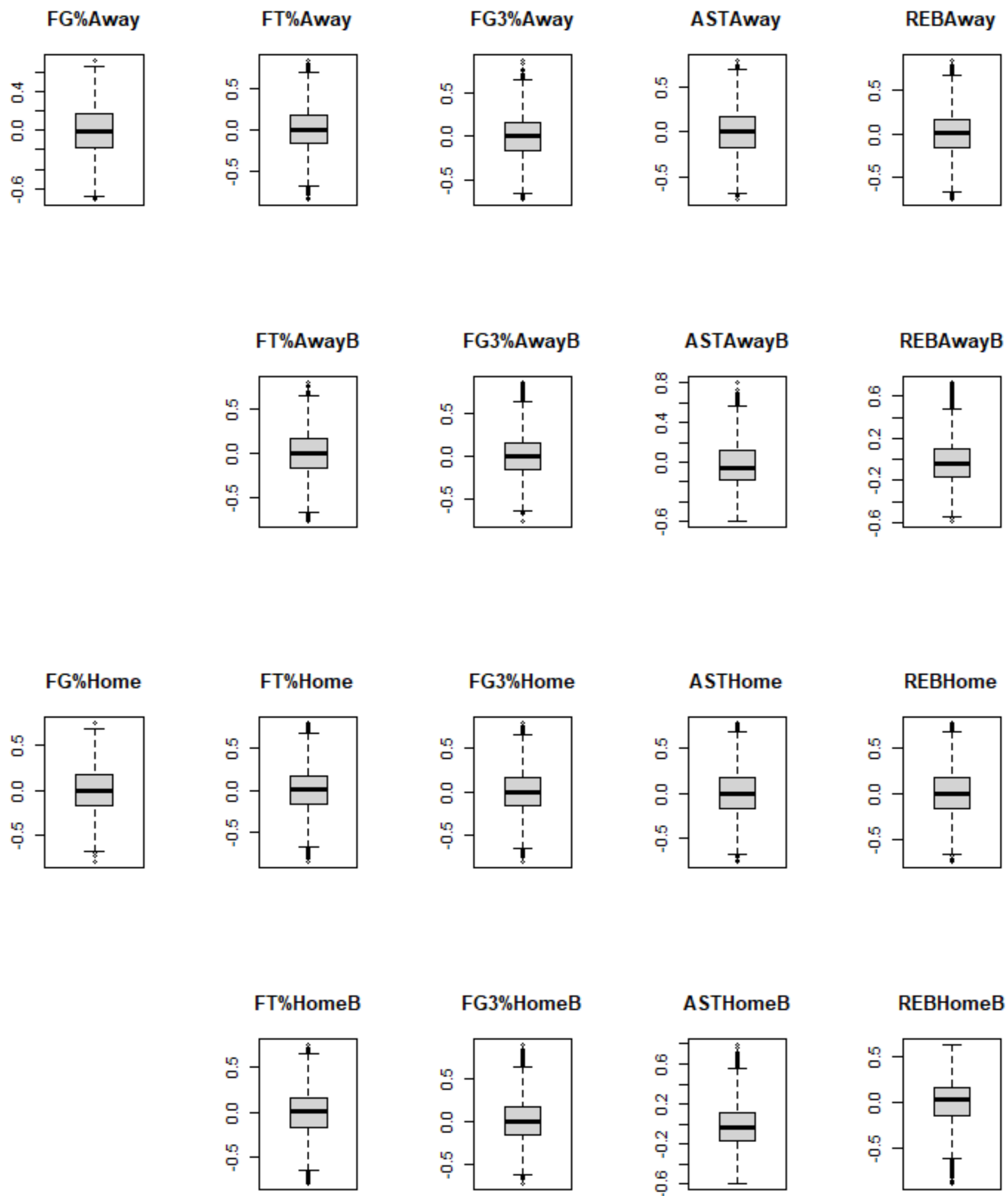
In order to account for the skewness, we applied a handful of transformations to the data. These include:

- Box-Cox transformations
- Center
- Scale
- Spatial sign transformations

Below are the resulting graphs post-transformation.

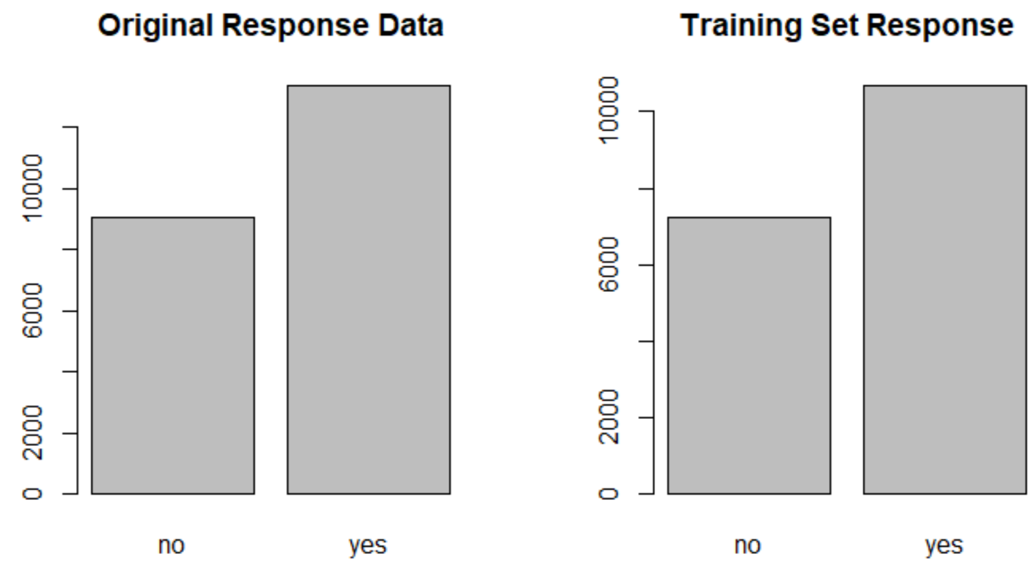


Outliers



Note that the two highly correlated predictors were removed from our data and are not included in these plots. The spatial sign transformations showed some improvement on the outliers but there are still several present.

Splitting of the Data



The actual question we are trying to answer is “does the home team win this game?” With such a large dataset, we can expect a fair training set data partition to resemble the original response training set. One thing to note, for some people, is that the home teams do tend to win their games more often than the away teams. This is shown in our data here, and it is why the frequency of “no” is smaller than the frequency of “yes”. This home field advantage is automatically accounted for by us defining the classification as whether or not the home team wins, so there are no significant issues on that front. Defining our classification as such will not hamper any predictive abilities in any way. It is clear that the response is imbalanced and therefore stratified random sampling has been used when splitting the data into a training and a testing set. 80% of each outcome is used in the training set leaving 20% from each outcome to be used as the test set.

Model Fitting

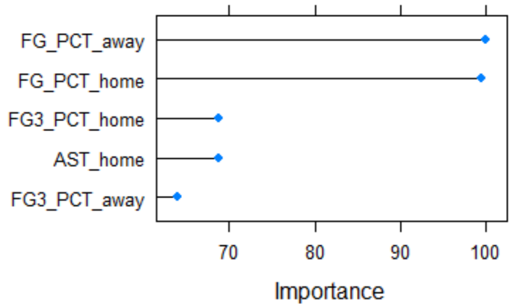
Overview of Models

Model		Training			Testing			
		ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**	**Accuracy**
Linear Models	LDA	0.9238	0.8032	0.8685	0.9215	0.7979	0.8670	0.8391
	Logistic	0.9240	0.7927	0.8755	0.9212	0.7874	0.8737	0.8388
	PLSDA	0.9233	0.7991	0.8708	0.9215	0.7957	0.8696	0.8388
	Penalized	0.9236	0.7883	0.8775	0.9210	0.7841	0.8790	0.8406
Non-Linear Models	FDA	0.9231	0.7940	0.8717	0.9200	0.7929	0.8722	0.8402
	KNN	0.9096	0.7676	0.8656	0.9078	0.7570	0.8737	0.8266
	MDA	0.9238	0.8032	0.8685	0.9215	0.7979	0.8670	0.8391
	Naive Bayes	0.8993	0.7836	0.8375	0.8953	0.7786	0.8396	0.8150
	Neural Net	0.9252	0.7971	0.8749	0.9201	0.7874	0.8737	0.8388
	QDA	0.9216	0.8000	0.8677	0.9178	0.7902	0.8670	0.8359
	RDA	0.9242	0.8037	0.8693	0.9214	0.7962	0.8700	0.8402
	SVM	**0.9254**	**0.7929**	**0.8781**	**0.9214**	**0.7885**	**0.8794**	**0.8426**

The majority of our models are nearly identical in terms of the area under the curve (ROC). Since we plan to use our model for predicting the winner of a game, it is hard to decide on a model based on Sensitivity or Specificity since it is equally important to be able to correctly predict when the home team will win and to be able to correctly predict when they won't. Therefore, we used a combination of ROC and Accuracy as the deciding factor for picking the strongest model. The optimal model was Support Vector Machine with a Radial kernel.

Variable Importance

Field Goal percentage was the most important variable in our model.



Appendix I - Supplemental Output

Linear Models

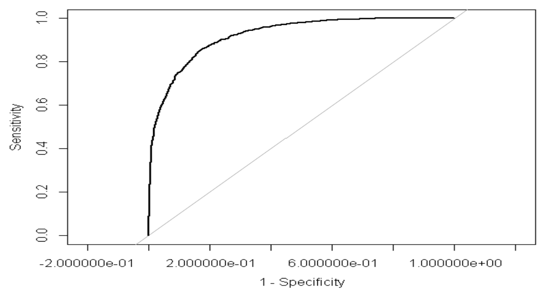
Linear Discriminant Analysis

No tuning parameters available.

Training			Testing		
ROC	Sensitivity	Specificity	ROC	Sensitivity	Specificity
0.9238	0.8032	0.8685	0.9215	0.7979	0.8670

```
Linear Discriminant Analysis

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results:

  ROC        Sens       Spec
  0.9237587  0.8032468  0.8685051

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1445  355
       yes  366 2314

               Accuracy : 0.8391
                 95% CI : (0.828, 0.8497)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6655

 Mcnemar's Test P-Value : 0.7096

            Sensitivity : 0.7979
            Specificity : 0.8670
```

```
         Pos Pred Value : 0.8028
         Neg Pred Value : 0.8634
             Prevalence : 0.4042
         Detection Rate : 0.3225
   Detection Prevalence : 0.4018
      Balanced Accuracy : 0.8324

       'Positive' Class : no
```



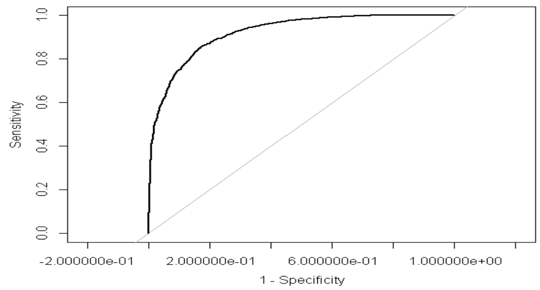
Logistic Model

No tuning parameters available.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9240	0.7927	0.8755	0.9212	0.7874	0.8737

```
Generalized Linear Model

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results:

  ROC       Sens      Spec
  0.923951  0.792667  0.8755339

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1426  337
       yes  385 2332

               Accuracy : 0.8388
                 95% CI : (0.8277, 0.8495)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.664

 Mcnemar's Test P-Value : 0.08026

            Sensitivity : 0.7874
            Specificity : 0.8737
         Pos Pred Value : 0.8088
         Neg Pred Value : 0.8583
             Prevalence : 0.4042
         Detection Rate : 0.3183
   Detection Prevalence : 0.3935
      Balanced Accuracy : 0.8306
```

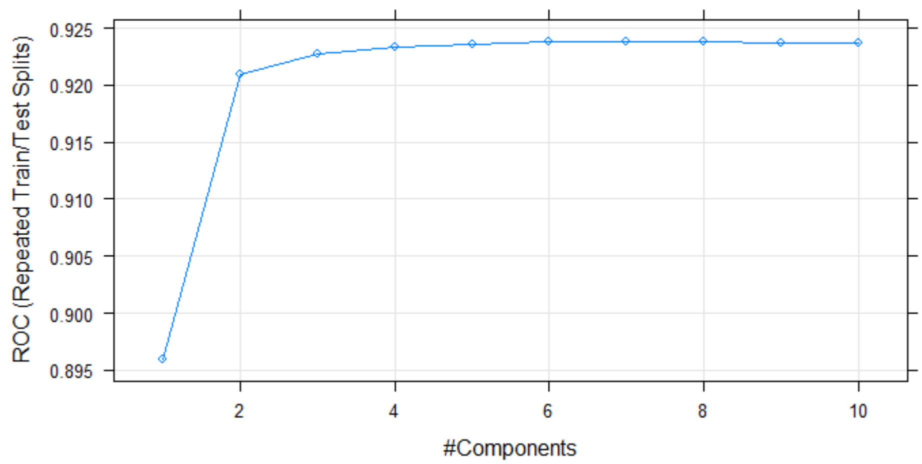
'Positive' Class : no



Partial Least Squares Discriminant Analysis

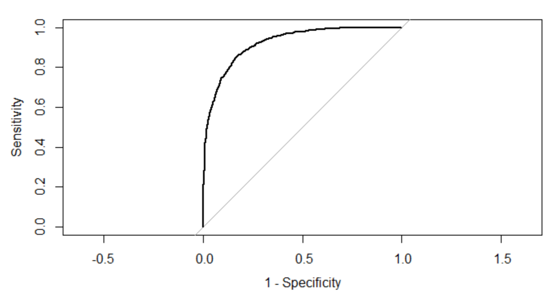
Optimal tuning parameter determined to be ncomp = 6.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9238	0.8017	0.8696	0.9215	0.7951	0.8685




```
Partial Least Squares

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (18), scaled (18)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  ncomp  ROC        Sens       Spec
   1     0.8959895  0.7602650  0.8464893
   2     0.9209689  0.7979901  0.8671263
   3     0.9227784  0.8017670  0.8672911
   4     0.9233907  0.8006184  0.8704983
   5     0.9236089  0.8012811  0.8692994
   6     0.9237686  0.8017007  0.8696441
   7     0.9237682  0.8018112  0.8695541
   8     0.9237602  0.8017449  0.8697190
   9     0.9237596  0.8018112  0.8697190
  10     0.9237591  0.8017891  0.8697490

ROC was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 6.
```

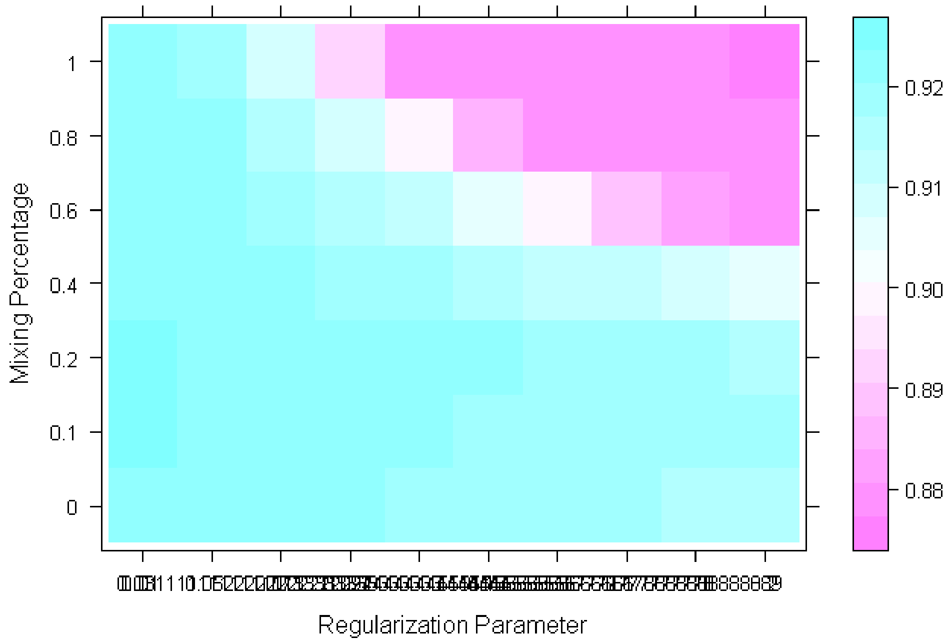
```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1440  351
       yes  371 2318

               Accuracy : 0.8388
                 95% CI : (0.8277, 0.8495)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6648

 Mcnemar's Test P-Value : 0.4795

            Sensitivity : 0.7951
            Specificity : 0.8685
         Pos Pred Value : 0.8040
         Neg Pred Value : 0.8620
             Prevalence : 0.4042
         Detection Rate : 0.3214
   Detection Prevalence : 0.3998
      Balanced Accuracy : 0.8318

       'Positive' Class : no
```



Penalized Logistic Model

Optimal tuning parameter determined to be alpha = 0.1 and lambda = 0.01.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9236	0.7883	0.8775	0.9210	0.7841	0.8790

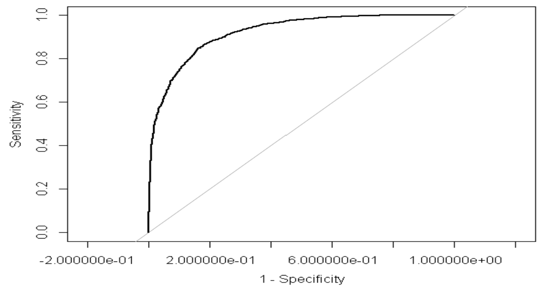


ROC (Repeated Train/Test Splits)

```
glmnet

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (18), scaled (18)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  alpha  lambda      ROC        Sens        Spec
  0.0    0.01000000  0.9230312  0.78252899  0.8801499
  0.0    0.03111111  0.9225352  0.77753727  0.8816336
  0.0    0.05222222  0.9214585  0.76713418  0.8844961
  0.0    0.07333333  0.9204410  0.75814467  0.8875384
  0.0    0.09444444  0.9195061  0.75249034  0.8907306
  0.0    0.11555556  0.9186176  0.74581999  0.8935931
  0.0    0.13666667  0.9178075  0.73780232  0.8960060
```

0.0	0.15777778	0.9170495	0.73064605	0.8982840
0.0	0.17888889	0.9163434	0.72346770	0.9008917
0.0	0.20000000	0.9156802	0.71818885	0.9033646
0.1	0.01000000	0.9236148	0.78831585	0.8775122
0.1	0.03111111	0.9226935	0.77632247	0.8820232
0.1	0.05222222	0.9218512	0.76384318	0.8856201
0.1	0.07333333	0.9211709	0.75392601	0.8900562
0.1	0.09444444	0.9205578	0.74515737	0.8958262
0.1	0.11555556	0.9200123	0.73596908	0.9007568
0.1	0.13666667	0.9195043	0.72715627	0.9050880
0.1	0.15777778	0.9190263	0.71785754	0.9092094
0.1	0.17888889	0.9185767	0.70906681	0.9116673
0.1	0.20000000	0.9181490	0.70025400	0.9147996
0.2	0.01000000	0.9235661	0.78782993	0.8771375
0.2	0.03111111	0.9227528	0.77464384	0.8820082
0.2	0.05222222	0.9222031	0.76112645	0.8878681
0.2	0.07333333	0.9216267	0.74833793	0.8939078
0.2	0.09444444	0.9209899	0.73738266	0.8997377
0.2	0.11555556	0.9202591	0.72609608	0.9056725
0.2	0.13666667	0.9194315	0.71363887	0.9097640
0.2	0.15777778	0.9184737	0.70157924	0.9146946
0.2	0.17888889	0.9173563	0.68753175	0.9192956
0.2	0.20000000	0.9160421	0.67067918	0.9245710
0.4	0.01000000	0.9234063	0.78672557	0.8764931
0.4	0.03111111	0.9226413	0.76993926	0.8845111
0.4	0.05222222	0.9215361	0.75412479	0.8913151
0.4	0.07333333	0.9197702	0.73811154	0.8974597
0.4	0.09444444	0.9171281	0.72024296	0.9029899
0.4	0.11555556	0.9141262	0.69819989	0.9098539
0.4	0.13666667	0.9128440	0.67447819	0.9180817
0.4	0.15777778	0.9113866	0.64879072	0.9280629
0.4	0.17888889	0.9092705	0.61323026	0.9364106
0.4	0.20000000	0.9062486	0.57371618	0.9453129
0.6	0.01000000	0.9232139	0.78586416	0.8766579
0.6	0.03111111	0.9219274	0.76457206	0.8853054
0.6	0.05222222	0.9189918	0.74487024	0.8917197
0.6	0.07333333	0.9137485	0.72068470	0.8989884
0.6	0.09444444	0.9105513	0.69137493	0.9080405
0.6	0.11555556	0.9062490	0.65579238	0.9168827
0.6	0.13666667	0.8990777	0.61055770	0.9268640
0.6	0.15777778	0.8883470	0.54782993	0.9366205
0.6	0.17888889	0.8822705	0.48541137	0.9491645
0.6	0.20000000	0.8789521	0.41789067	0.9630124
0.8	0.01000000	0.9230403	0.78456102	0.8772874
0.8	0.03111111	0.9204774	0.76061844	0.8861296
0.8	0.05222222	0.9136455	0.73349531	0.8921694
0.8	0.07333333	0.9071459	0.69835450	0.9007868
0.8	0.09444444	0.8973082	0.64945334	0.9075459

```
  0.8    0.11555556  0.8840970  0.59379348  0.9173773
  0.8    0.13666667  0.8789551  0.53263390  0.9333983
  0.8    0.15777778  0.8781923  0.46029818  0.9536456
  0.8    0.17888889  0.8781695  0.36673661  0.9724091
  0.8    0.20000000  0.8781178  0.22586416  0.9899438
  1.0    0.01000000  0.9228499  0.78274986  0.8775272
  1.0    0.03111111  0.9179869  0.75578134  0.8861896
  1.0    0.05222222  0.9071272  0.71741579  0.8920045
  1.0    0.07333333  0.8936773  0.66590834  0.8961858
  1.0    0.09444444  0.8799891  0.60938708  0.9062570
  1.0    0.11555556  0.8782021  0.54895638  0.9268790
  1.0    0.13666667  0.8781829  0.46334622  0.9530011
  1.0    0.15777778  0.8781269  0.32795141  0.9788535
  1.0    0.17888889  0.8779859  0.09976808  0.9976920
  1.0    0.20000000  0.8772011  0.00000000  1.0000000

ROC was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.1 and lambda = 0.01.

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1420  323
       yes  391 2346

               Accuracy : 0.8406
                 95% CI : (0.8296, 0.8512)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.6671

 Mcnemar's Test P-Value : 0.01216

            Sensitivity : 0.7841
            Specificity : 0.8790
         Pos Pred Value : 0.8147
         Neg Pred Value : 0.8571
             Prevalence : 0.4042
         Detection Rate : 0.3170
   Detection Prevalence : 0.3891
      Balanced Accuracy : 0.8315

       'Positive' Class : no
```

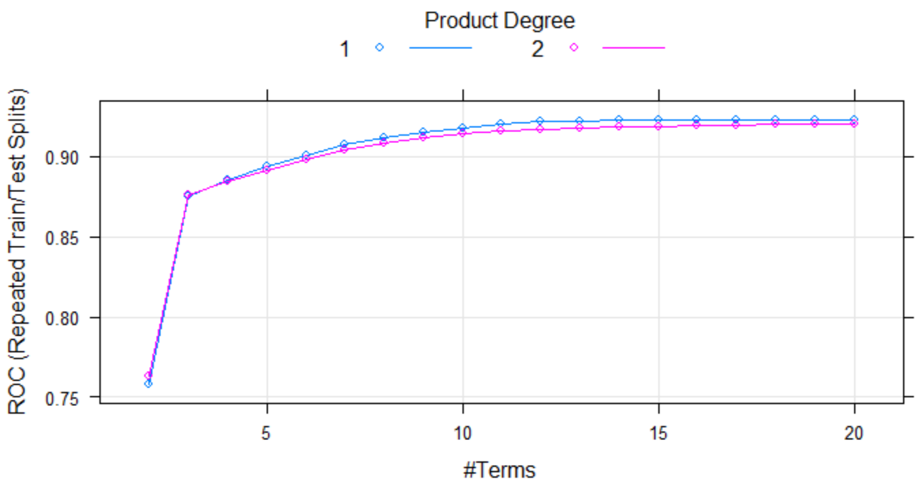

Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
-2.000000e-01
2.000000e-01
6.000000e-01
1.000000e+00
1 - Specificity

Non-Linear Models

Flexible Discriminant Analysis

Optimal tuning parameter determined to be degree = 1 and nprune = 20.

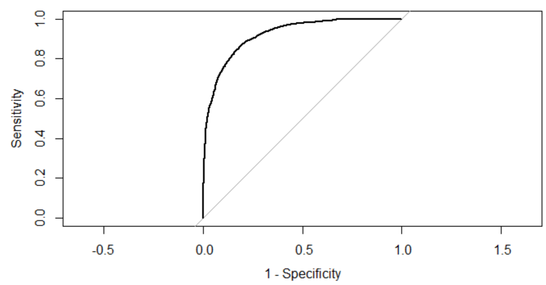
Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9231	0.7940	0.8717	0.9200	0.7929	0.8722




```
Flexible Discriminant Analysis

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  degree  nprune  ROC        Sens       Spec
  1        2      0.7577698  0.5476532  0.8007044
  1        3      0.8756177  0.7055770  0.8533833
  1        4      0.8852846  0.7271342  0.8583439
  1        5      0.8935862  0.7447156  0.8600225
  1        6      0.9008201  0.7551629  0.8605920
  1        7      0.9074994  0.7645058  0.8665118
  1        8      0.9120742  0.7714854  0.8697490
  1        9      0.9154680  0.7773827  0.8718321
  1       10      0.9177731  0.7819989  0.8734807
  1       11      0.9205387  0.7865710  0.8724166
```

```
  1       12      0.9217537  0.7905025  0.8731660
  1       13      0.9223628  0.7911651  0.8736306
  1       14      0.9227483  0.7923136  0.8730461
  1       15      0.9229373  0.7929100  0.8726115
  1       16      0.9229863  0.7931309  0.8722368
  1       17      0.9230477  0.7935505  0.8718172
  1       18      0.9230737  0.7935284  0.8718172
  1       19      0.9231192  0.7939481  0.8716223
  1       20      0.9231339  0.7939923  0.8716673
  2        2      0.7632577  0.5478962  0.8093518
  2        3      0.8757286  0.7046935  0.8557063
  2        4      0.8842348  0.7184318  0.8624803
  2        5      0.8917904  0.7239094  0.8682203
  2        6      0.8984005  0.7338487  0.8718321
  2        7      0.9040159  0.7426394  0.8757287
  2        8      0.9086624  0.7525787  0.8772874
  2        9      0.9118844  0.7542131  0.8810491
  2       10      0.9145671  0.7597791  0.8821731
  2       11      0.9159703  0.7593374  0.8833271
  2       12      0.9172109  0.7641524  0.8822031
  2       13      0.9180492  0.7707344  0.8784564
  2       14      0.9186236  0.7736057  0.8777070
  2       15      0.9187471  0.7715516  0.8789209
  2       16      0.9193123  0.7744451  0.8790408
  2       17      0.9197451  0.7743125  0.8800150
  2       18      0.9201997  0.7744230  0.8800150
  2       19      0.9204008  0.7739150  0.8806145
  2       20      0.9204689  0.7747764  0.8801199

ROC was used to select the optimal model using the largest value.
The final values used for the model were degree = 1 and nprune = 20.


Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1436  341
       yes  375 2328

               Accuracy : 0.8402
                 95% CI : (0.8291, 0.8508)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6672

 Mcnemar's Test P-Value : 0.2175
```

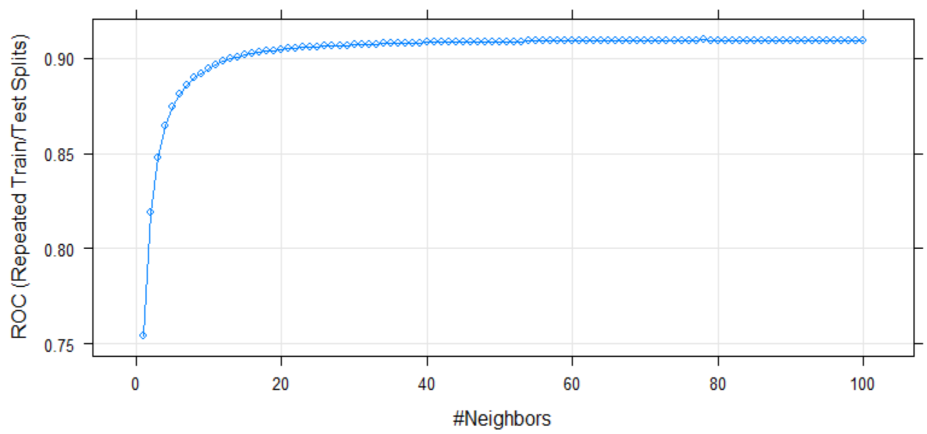
```
         Sensitivity : 0.7929
         Specificity : 0.8722
      Pos Pred Value : 0.8081
      Neg Pred Value : 0.8613
          Prevalence : 0.4042
      Detection Rate : 0.3205
Detection Prevalence : 0.3967
   Balanced Accuracy : 0.8326

    'Positive' Class : no
```



KNN Model

Optimal tuning parameter determined to be k=78 neighbors.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9096	0.7676	0.8656	0.9078	0.7570	0.8737

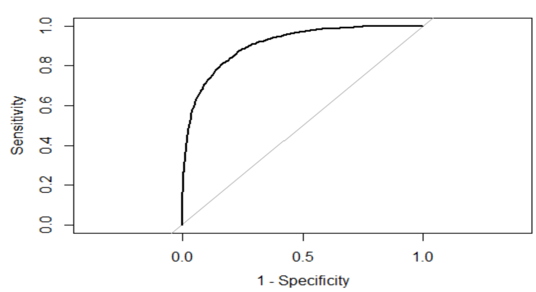



```
k-Nearest Neighbors

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (18), scaled (18)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  k    ROC        Sens       Spec
    1  0.7544786  0.7223854  0.7865717
    2  0.8187145  0.7163335  0.7841289
    3  0.8477390  0.7475870  0.8223754
    4  0.8644812  0.7460188  0.8205021
    5  0.8744421  0.7575925  0.8392207
    6  0.8813693  0.7578134  0.8379318
    7  0.8859895  0.7651905  0.8457400
    8  0.8895664  0.7637327  0.8452154
    9  0.8919757  0.7671784  0.8487973
   10  0.8945271  0.7669575  0.8480779
   11  0.8964476  0.7692766  0.8523342
   12  0.8983844  0.7678189  0.8524691
   13  0.8997468  0.7711982  0.8550618
```

14	0.9007528	0.7702264	0.8558112
15	0.9016075	0.7712424	0.8580592
16	0.9023409	0.7708227	0.8576845
17	0.9029368	0.7717062	0.8583739
18	0.9036282	0.7716400	0.8581641
19	0.9041921	0.7715737	0.8598277
20	0.9046852	0.7718388	0.8591982
21	0.9051125	0.7718388	0.8602173
22	0.9055084	0.7698067	0.8592432
23	0.9057445	0.7715737	0.8605620
24	0.9059661	0.7711099	0.8600974
25	0.9061812	0.7720155	0.8607718
26	0.9064244	0.7704694	0.8601574
27	0.9064563	0.7717725	0.8599925
28	0.9066286	0.7713528	0.8593331
29	0.9067877	0.7711982	0.8597227
30	0.9069543	0.7702706	0.8597078
31	0.9071829	0.7709553	0.8599176
32	0.9073221	0.7704694	0.8605320
33	0.9074112	0.7706019	0.8608318
34	0.9075586	0.7701822	0.8616411
35	0.9077360	0.7700939	0.8616561
36	0.9078714	0.7700276	0.8620157
37	0.9079970	0.7706902	0.8624653
38	0.9081036	0.7709553	0.8624653
39	0.9082136	0.7715958	0.8630049
40	0.9082819	0.7708890	0.8632746
41	0.9084541	0.7718167	0.8630798
42	0.9085077	0.7707344	0.8629599
43	0.9085840	0.7707565	0.8628400
44	0.9086648	0.7712424	0.8628850
45	0.9087278	0.7706461	0.8627801
46	0.9088122	0.7700276	0.8629299
47	0.9088582	0.7717062	0.8628100
48	0.9087910	0.7703368	0.8628850
49	0.9088364	0.7708669	0.8631098
50	0.9088067	0.7703368	0.8633646
51	0.9087646	0.7702264	0.8629899
52	0.9087672	0.7704473	0.8625553
53	0.9088114	0.7705577	0.8630948
54	0.9089008	0.7698951	0.8628100
55	0.9089562	0.7698730	0.8630798
56	0.9089584	0.7698509	0.8639341
57	0.9089751	0.7693650	0.8638142
58	0.9090386	0.7698288	0.8635294
59	0.9090597	0.7699393	0.8643687
60	0.9091305	0.7692104	0.8643237
61	0.9092248	0.7695859	0.8646384

```
   62  0.9093040  0.7692325  0.8651330
   63  0.9093568  0.7692325  0.8647284
   64  0.9093668  0.7683269  0.8646384
   65  0.9093815  0.7686361  0.8650281
   66  0.9094633  0.7685036  0.8650281
   67  0.9094932  0.7689232  0.8649532
   68  0.9095362  0.7690116  0.8647433
   69  0.9095300  0.7683269  0.8653129
   70  0.9095051  0.7683048  0.8654028
   71  0.9095161  0.7680839  0.8654777
   72  0.9095118  0.7680398  0.8654178
   73  0.9095384  0.7675759  0.8656875
   74  0.9095487  0.7683048  0.8655526
   75  0.9095433  0.7681281  0.8655676
   76  0.9095433  0.7675980  0.8659123
   77  0.9095429  0.7681060  0.8659273
   78  0.9095992  0.7675538  0.8655976
   79  0.9095480  0.7667808  0.8663320
   80  0.9095331  0.7673551  0.8661072
   81  0.9095222  0.7669796  0.8662570
   82  0.9095323  0.7677084  0.8661971
   83  0.9095492  0.7668691  0.8663769
   84  0.9095300  0.7670458  0.8661371
   85  0.9094868  0.7671342  0.8662570
   86  0.9095031  0.7669575  0.8667966
   87  0.9094852  0.7670017  0.8663619
   88  0.9094916  0.7665378  0.8665268
   89  0.9094607  0.7669133  0.8663020
   90  0.9094445  0.7663390  0.8666317
   91  0.9094395  0.7664274  0.8661971
   92  0.9094306  0.7664495  0.8666317
   93  0.9094220  0.7659194  0.8664968
   94  0.9094470  0.7659856  0.8664369
   95  0.9094448  0.7655660  0.8668565
   96  0.9094449  0.7654335  0.8666467
   97  0.9094407  0.7659636  0.8668865
   98  0.9094323  0.7656764  0.8668415
   99  0.9094004  0.7651684  0.8668115
  100  0.9094085  0.7650580  0.8667216

ROC was used to select the optimal model using the largest value.
The final value used for the model was k = 78.

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1376  342
```

```
      yes  435 2327

               Accuracy : 0.8266
                 95% CI : (0.8152, 0.8375)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6369

 Mcnemar's Test P-Value : 0.0009652

            Sensitivity : 0.7598
            Specificity : 0.8719
         Pos Pred Value : 0.8009
         Neg Pred Value : 0.8425
             Prevalence : 0.4042
         Detection Rate : 0.3071
   Detection Prevalence : 0.3835
      Balanced Accuracy : 0.8158

       'Positive' Class : no
```



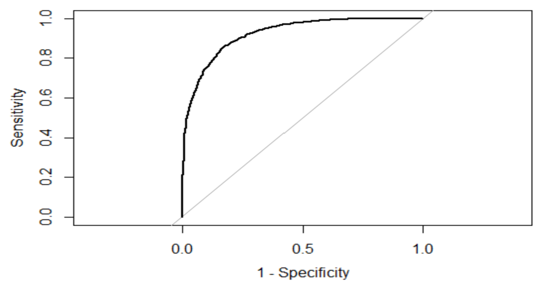
Mixture Discriminant Analysis

Optimal tuning parameter determined to be subclasses = 1

Training			**Testing**		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9238	0.8032	0.8685	0.9215	0.7979	0.8670

```
Mixture Discriminant Analysis

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  subclasses  ROC        Sens       Spec
  1           0.9237587  0.8032468  0.8685051
  2           0.9235575  0.8029818  0.8698689
  3           0.9226769  0.8041745  0.8658674

ROC was used to select the optimal model using the largest value.
The final value used for the model was subclasses = 1.

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1445  355
       yes  366 2314

               Accuracy : 0.8391
                 95% CI : (0.828, 0.8497)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6655

 Mcnemar's Test P-Value : 0.7096

            Sensitivity : 0.7979
            Specificity : 0.8670
         Pos Pred Value : 0.8028
         Neg Pred Value : 0.8634
```

```
            Prevalence : 0.4042
        Detection Rate : 0.3225
  Detection Prevalence : 0.4018
     Balanced Accuracy : 0.8324

      'Positive' Class : no
```



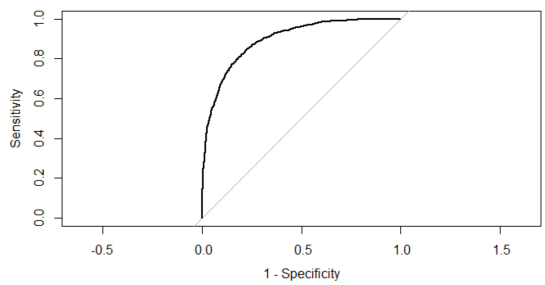
Naive-Bayes

No tuning parameters available.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.8993	0.7836	0.8375	0.8953	0.7786	0.8396

```
Naive Bayes

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results:

  ROC        Sens       Spec
  0.8992646  0.7835892  0.8374972

Tuning parameter 'fL' was held constant at a value of 2 of TRUE
Tuning parameter 'adjust' was held constant at a value of TRUE

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1410  428
       yes  401 2241

               Accuracy : 0.815
                 95% CI : (0.8033, 0.8262)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6167

 Mcnemar's Test P-Value : 0.3665

            Sensitivity : 0.7786
            Specificity : 0.8396
```

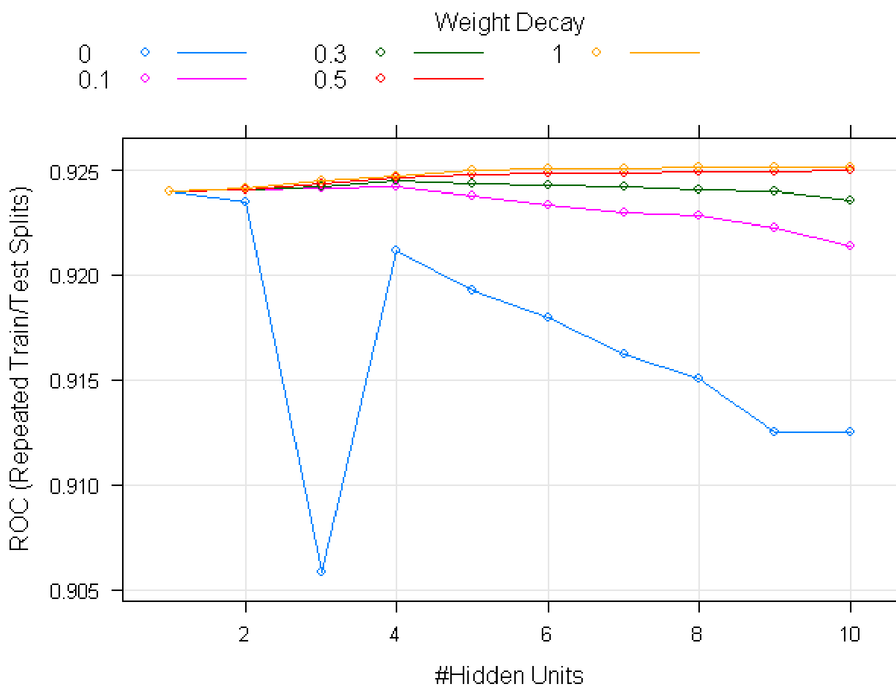
```
         Pos Pred Value : 0.7671
         Neg Pred Value : 0.8482
             Prevalence : 0.4042
         Detection Rate : 0.3147
   Detection Prevalence : 0.4103
      Balanced Accuracy : 0.8091

       'Positive' Class : no
```



Neural Network

Optimal tuning parameter determined to be

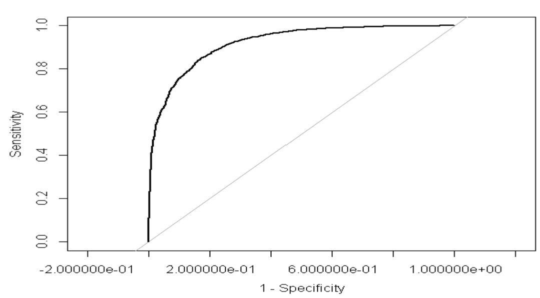
Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9252	0.7971	0.8749	0.9201	0.7874	0.8737




```
Neural Network

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  size  decay  ROC        Sens       Spec
   1    0.0    0.9239532  0.7946328  0.8743350
   1    0.1    0.9239548  0.7973716  0.8724766
   1    0.3    0.9239524  0.7985422  0.8718471
   1    0.5    0.9239563  0.7990723  0.8717272
   1    1.0    0.9239560  0.7993153  0.8713825
   2    0.0    0.9234613  0.7952071  0.8741851
```

```
   2    0.1    0.9240719  0.7973495  0.8734058
   2    0.3    0.9240680  0.7971728  0.8733608
   2    0.5    0.9240370  0.7969078  0.8735407
   2    1.0    0.9241509  0.7959580  0.8732109
   3    0.0    0.9058316  0.7635560  0.8785912
   3    0.1    0.9241554  0.7935947  0.8745747
   3    0.3    0.9242312  0.7969078  0.8735706
   3    0.5    0.9243642  0.7978575  0.8737954
   3    1.0    0.9244910  0.7961789  0.8744549
   4    0.0    0.9211478  0.7945445  0.8725515
   4    0.1    0.9241707  0.7951187  0.8741102
   4    0.3    0.9244645  0.7952733  0.8735556
   4    0.5    0.9246702  0.7960464  0.8741401
   4    1.0    0.9247364  0.7966427  0.8745448
   5    0.0    0.9192492  0.7904583  0.8710378
   5    0.1    0.9237944  0.7951408  0.8737205
   5    0.3    0.9243516  0.7954279  0.8739753
   5    0.5    0.9248107  0.7963335  0.8744099
   5    1.0    0.9250111  0.7964881  0.8750244
   6    0.0    0.9179912  0.7898399  0.8701236
   6    0.1    0.9232996  0.7939702  0.8736306
   6    0.3    0.9242837  0.7953396  0.8743050
   6    0.5    0.9248342  0.7961347  0.8740802
   6    1.0    0.9250845  0.7969961  0.8749494
   7    0.0    0.9162244  0.7910547  0.8680405
   7    0.1    0.9229650  0.7946991  0.8740802
   7    0.3    0.9242219  0.7948316  0.8752492
   7    0.5    0.9248852  0.7968415  0.8744848
   7    1.0    0.9250896  0.7971066  0.8746497
   8    0.0    0.9150822  0.7884042  0.8685500
   8    0.1    0.9227917  0.7947874  0.8737505
   8    0.3    0.9240371  0.7953396  0.8741851
   8    0.5    0.9249038  0.7966207  0.8747996
   8    1.0    0.9251652  0.7971066  0.8750843
   9    0.0    0.9125121  0.7852015  0.8641439
   9    0.1    0.9222314  0.7941027  0.8724466
   9    0.3    0.9239871  0.7948095  0.8749194
   9    0.5    0.9249574  0.7969520  0.8756238
   9    1.0    0.9251824  0.7971066  0.8748895
  10    0.0    0.9125490  0.7857537  0.8654777
  10    0.1    0.9213999  0.7924462  0.8719520
  10    0.3    0.9235792  0.7944119  0.8742750
  10    0.5    0.9249675  0.7960243  0.8747396
  10    1.0    0.9251501  0.7969520  0.8750244

ROC was used to select the optimal model using the largest value.
The final values used for the model were size = 9 and decay = 1.
```

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1426  337
       yes  385 2332

               Accuracy : 0.8388
                 95% CI : (0.8277, 0.8495)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.664

 Mcnemar's Test P-Value : 0.08026

            Sensitivity : 0.7874
            Specificity : 0.8737
         Pos Pred Value : 0.8088
         Neg Pred Value : 0.8583
             Prevalence : 0.4042
         Detection Rate : 0.3183
   Detection Prevalence : 0.3935
      Balanced Accuracy : 0.8306

       'Positive' Class : no
```



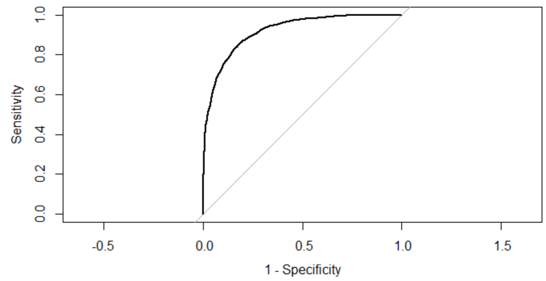
Quadratic Discriminant Analysis

No tuning parameters available.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9216	0.8000	0.8677	0.9178	0.7902	0.8670

```
Quadratic Discriminant Analysis

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results:

  ROC        Sens  Spec
  0.9216154  0.8   0.8667066

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1431  355
       yes  380 2314

               Accuracy : 0.8359
                 95% CI : (0.8248, 0.8467)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6586

 Mcnemar's Test P-Value : 0.376

            Sensitivity : 0.7902
            Specificity : 0.8670
         Pos Pred Value : 0.8012
         Neg Pred Value : 0.8589
             Prevalence : 0.4042
         Detection Rate : 0.3194
   Detection Prevalence : 0.3987
      Balanced Accuracy : 0.8286
```

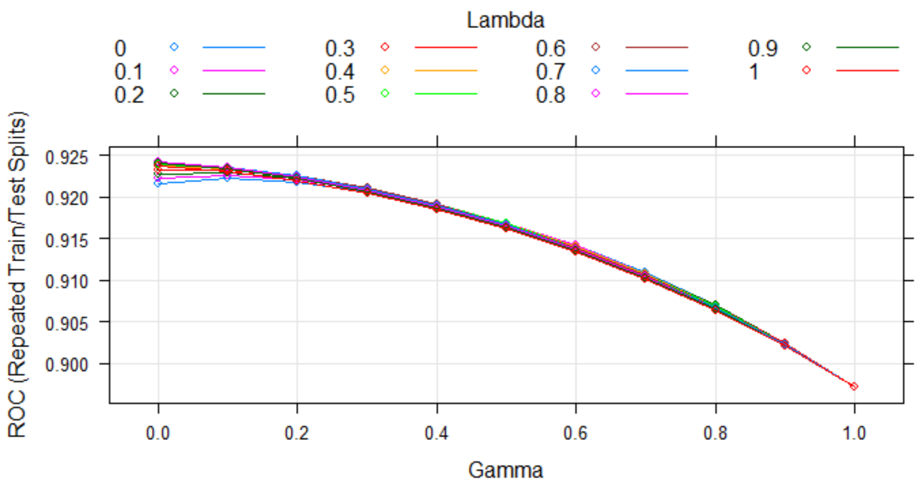
```
'Positive' Class : no
```



Regularized Discriminant Analysis

Optimal tuning parameter determined to be gamma = 0 and lambda = 0.7.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9242	0.8037	0.8693	0.9214	0.7962	0.8700

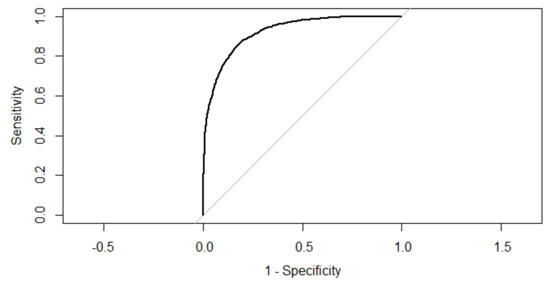



```
Regularized Discriminant Analysis

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  gamma  lambda  ROC        Sens       Spec
  0.0    0.0     0.9216154  0.8000000  0.8667066
  0.0    0.1     0.9222730  0.8014357  0.8675009
  0.0    0.2     0.9228286  0.8024517  0.8682653
  0.0    0.3     0.9233050  0.8032027  0.8687898
  0.0    0.4     0.9236842  0.8039757  0.8694043
  0.0    0.5     0.9239619  0.8042187  0.8691195
  0.0    0.6     0.9241319  0.8040420  0.8691795
  0.0    0.7     0.9241935  0.8036886  0.8693293
  0.0    0.8     0.9241581  0.8037769  0.8694043
  0.0    0.9     0.9240158  0.8036444  0.8689697
  0.0    1.0     0.9237587  0.8032468  0.8685051
  0.1    0.0     0.9222105  0.8018112  0.8663769
```

0.1	0.1	0.9226298	0.8026505	0.8668415
0.1	0.2	0.9229827	0.8033131	0.8675159
0.1	0.3	0.9232624	0.8036886	0.8681004
0.1	0.4	0.9234734	0.8041966	0.8682503
0.1	0.5	0.9236000	0.8036002	0.8683402
0.1	0.6	0.9236481	0.8039315	0.8683102
0.1	0.7	0.9236275	0.8034235	0.8680555
0.1	0.8	0.9235415	0.8030260	0.8676208
0.1	0.9	0.9233808	0.8024075	0.8669015
0.1	1.0	0.9231449	0.8013694	0.8665717
0.2	0.0	0.9217962	0.8019437	0.8643387
0.2	0.1	0.9220657	0.8015903	0.8647134
0.2	0.2	0.9222802	0.8019437	0.8651930
0.2	0.3	0.9224431	0.8020099	0.8654627
0.2	0.4	0.9225456	0.8025621	0.8655077
0.2	0.5	0.9225933	0.8028493	0.8656276
0.2	0.6	0.9225847	0.8026726	0.8656725
0.2	0.7	0.9225240	0.8019879	0.8652829
0.2	0.8	0.9224089	0.8017449	0.8648033
0.2	0.9	0.9222395	0.8013694	0.8645935
0.2	1.0	0.9220119	0.8007289	0.8643087
0.3	0.0	0.9206964	0.8012369	0.8617010
0.3	0.1	0.9208663	0.8011044	0.8624204
0.3	0.2	0.9209884	0.8013694	0.8623754
0.3	0.3	0.9210704	0.8012590	0.8627201
0.3	0.4	0.9211130	0.8012590	0.8629899
0.3	0.5	0.9211052	0.8007068	0.8628100
0.3	0.6	0.9210682	0.8004638	0.8625253
0.3	0.7	0.9209809	0.7997570	0.8622405
0.3	0.8	0.9208561	0.7987852	0.8617460
0.3	0.9	0.9206917	0.7981226	0.8618659
0.3	1.0	0.9204754	0.7972391	0.8615661
0.4	0.0	0.9190320	0.7986306	0.8594380
0.4	0.1	0.9191293	0.7985202	0.8595129
0.4	0.2	0.9191897	0.7984539	0.8594380
0.4	0.3	0.9192221	0.7981447	0.8594230
0.4	0.4	0.9192175	0.7977029	0.8594530
0.4	0.5	0.9191837	0.7967311	0.8594680
0.4	0.6	0.9191181	0.7963777	0.8593181
0.4	0.7	0.9190215	0.7954500	0.8591982
0.4	0.8	0.9188876	0.7952954	0.8592581
0.4	0.9	0.9187288	0.7948758	0.8591982
0.4	1.0	0.9185369	0.7944119	0.8593631
0.5	0.0	0.9168573	0.7965323	0.8559610
0.5	0.1	0.9168955	0.7965102	0.8559311
0.5	0.2	0.9169125	0.7960685	0.8562158
0.5	0.3	0.9169081	0.7956709	0.8561709
0.5	0.4	0.9168737	0.7949862	0.8564406

0.5	0.5	0.9168248	0.7946991	0.8568153
0.5	0.6	0.9167401	0.7939923	0.8569352
0.5	0.7	0.9166411	0.7932634	0.8569352
0.5	0.8	0.9165145	0.7926670	0.8572199
0.5	0.9	0.9163594	0.7920265	0.8576695
0.5	1.0	0.9161907	0.7915185	0.8578644
0.6	0.0	0.9141568	0.7924903	0.8521544
0.6	0.1	0.9141504	0.7921590	0.8525141
0.6	0.2	0.9141325	0.7919602	0.8526789
0.6	0.3	0.9140981	0.7915848	0.8528587
0.6	0.4	0.9140431	0.7914743	0.8528587
0.6	0.5	0.9139789	0.7911872	0.8529786
0.6	0.6	0.9138972	0.7910326	0.8531135
0.6	0.7	0.9137948	0.7912755	0.8534283
0.6	0.8	0.9136819	0.7911872	0.8536081
0.6	0.9	0.9135518	0.7907013	0.8537280
0.6	1.0	0.9134056	0.7901712	0.8540427
0.7	0.0	0.9108934	0.7895969	0.8492619
0.7	0.1	0.9108637	0.7891993	0.8492619
0.7	0.2	0.9108245	0.7893760	0.8494417
0.7	0.3	0.9107749	0.7890226	0.8495766
0.7	0.4	0.9107201	0.7888901	0.8495167
0.7	0.5	0.9106509	0.7890447	0.8495466
0.7	0.6	0.9105694	0.7891773	0.8496516
0.7	0.7	0.9104792	0.7889343	0.8496665
0.7	0.8	0.9103794	0.7884705	0.8496516
0.7	0.9	0.9102721	0.7883600	0.8495766
0.7	1.0	0.9101495	0.7880287	0.8492919
0.8	0.0	0.9070260	0.7866593	0.8452154
0.8	0.1	0.9069890	0.7864384	0.8449906
0.8	0.2	0.9069407	0.7863280	0.8450206
0.8	0.3	0.9068902	0.7863501	0.8448558
0.8	0.4	0.9068344	0.7863280	0.8446909
0.8	0.5	0.9067717	0.7862617	0.8446010
0.8	0.6	0.9067046	0.7860629	0.8445710
0.8	0.7	0.9066322	0.7859304	0.8444361
0.8	0.8	0.9065559	0.7861734	0.8442563
0.8	0.9	0.9064760	0.7861955	0.8441813
0.8	1.0	0.9063932	0.7861292	0.8441214
0.9	0.0	0.9024751	0.7812921	0.8411990
0.9	0.1	0.9024392	0.7815130	0.8409442
0.9	0.2	0.9024031	0.7818222	0.8407194
0.9	0.3	0.9023686	0.7821093	0.8405845
0.9	0.4	0.9023324	0.7823302	0.8404496
0.9	0.5	0.9022944	0.7826615	0.8401499
0.9	0.6	0.9022552	0.7831695	0.8398951
0.9	0.7	0.9022120	0.7836113	0.8397752
0.9	0.8	0.9021689	0.7838100	0.8394605

```
  0.9    0.9     0.9021231  0.7842518  0.8391907
  0.9    1.0     0.9020723  0.7846273  0.8390259
  1.0    0.0     0.8971342  0.7759470  0.8371525
  1.0    0.1     0.8971358  0.7765213  0.8367179
  1.0    0.2     0.8971373  0.7773385  0.8361933
  1.0    0.3     0.8971396  0.7778023  0.8356538
  1.0    0.4     0.8971416  0.7785091  0.8352042
  1.0    0.5     0.8971440  0.7788625  0.8347246
  1.0    0.6     0.8971463  0.7794368  0.8343499
  1.0    0.7     0.8971482  0.7799227  0.8337804
  1.0    0.8     0.8971504  0.7805411  0.8332409
  1.0    0.9     0.8971521  0.7811154  0.8328662
  1.0    1.0     0.8971537  0.7814688  0.8324166

ROC was used to select the optimal model using the largest value.
The final values used for the model were gamma = 0 and lambda = 0.7.


Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1442  347
       yes  369 2322

               Accuracy : 0.8402
                 95% CI : (0.8291, 0.8508)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6675

 Mcnemar's Test P-Value : 0.4326

            Sensitivity : 0.7962
            Specificity : 0.8700
         Pos Pred Value : 0.8060
         Neg Pred Value : 0.8629
             Prevalence : 0.4042
         Detection Rate : 0.3219
   Detection Prevalence : 0.3993
      Balanced Accuracy : 0.8331

       'Positive' Class : no
```

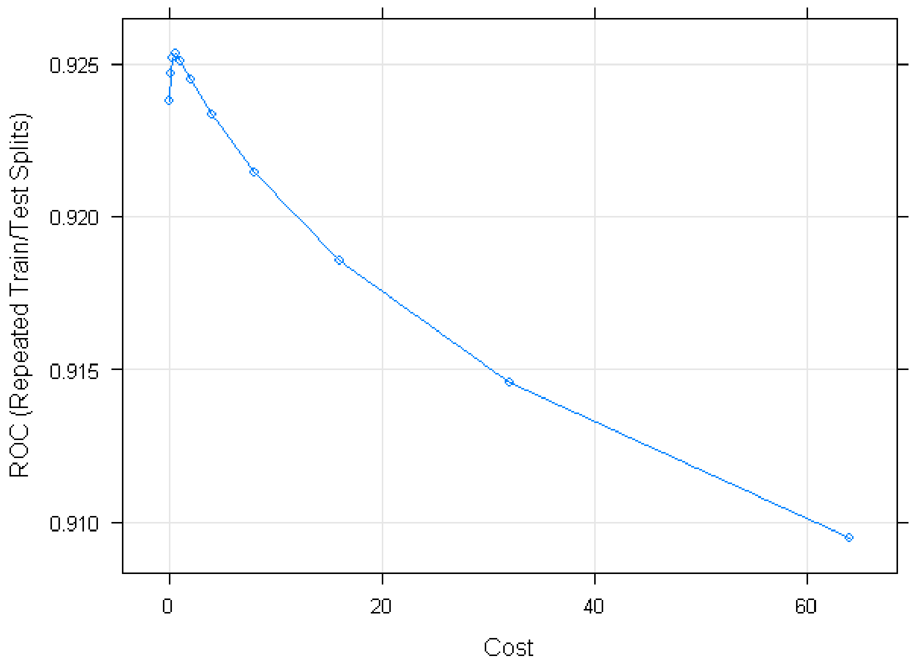


Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
-0.5
0.0
0.5
1.0
1.5
1 - Specificity

Support Vector Machines

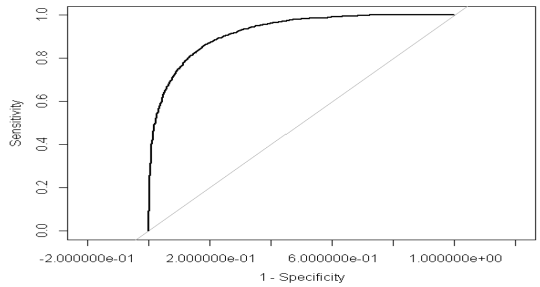
Optimal tuning parameter determined to be sigma = 0.019949 and C = 0.5.

Training			Testing		
ROC	**Sensitivity**	**Specificity**	**ROC**	**Sensitivity**	**Specificity**
0.9254	0.7929	0.8781	0.9214	0.7885	0.8794




```
Support Vector Machines with Radial Basis Function Kernel

17923 samples
   18 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (18), scaled (18)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 13443, 13443, 13443, 13443, 13443, 13443, ...
Resampling results across tuning parameters:

  C         ROC        Sens       Spec
   0.0625  0.9237790  0.7948537  0.8754440
   0.1250  0.9246563  0.7956930  0.8760135
   0.2500  0.9251801  0.7942352  0.8765830
   0.5000  0.9253520  0.7929100  0.8781117
   1.0000  0.9250767  0.7914301  0.8796103
```

```
   2.0000  0.9244577  0.7878962  0.8804646
   4.0000  0.9233556  0.7831474  0.8818434
   8.0000  0.9214567  0.7781557  0.8827726
  16.0000  0.9185315  0.7748205  0.8814088
  32.0000  0.9145591  0.7730094  0.8789209
  64.0000  0.9094530  0.7697626  0.8750094

Tuning parameter 'sigma' was held constant at a value of 0.019949
ROC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.019949 and C = 0.5.

Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  1428  322
       yes  383 2347

               Accuracy : 0.8426
                 95% CI : (0.8316, 0.8532)
    No Information Rate : 0.5958
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.6715

 Mcnemar's Test P-Value : 0.02384

            Sensitivity : 0.7885
            Specificity : 0.8794
         Pos Pred Value : 0.8160
         Neg Pred Value : 0.8597
             Prevalence : 0.4042
         Detection Rate : 0.3187
   Detection Prevalence : 0.3906
      Balanced Accuracy : 0.8339

       'Positive' Class : no
```



Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
-2.000000e-01
2.000000e-01
6.000000e-01
1.000000e+00
1 - Specificity

Appendix II - R Code

Processing

```
data<-read.csv('bballdata.csv')
str(data)

# Separate Predictors and Response from non-used variables
temp <- data[,-c(1,2,3,4,5,6,7,8,14,20,21,27)]

# Coerce Predictors to be Numeric.
temp$FG3_PCT_home_bench <- as.character(temp$FG3_PCT_home_bench)
temp$FT_PCT_away_bench <- as.character(temp$FT_PCT_away_bench)
temp$FG3_PCT_away_bench <- as.character(temp$FG3_PCT_away_bench)
temp$FG3_PCT_home_bench <- as.numeric(temp$FG3_PCT_home_bench)
temp$FT_PCT_away_bench <- as.numeric(temp$FT_PCT_away_bench)
temp$FG3_PCT_away_bench <- as.numeric(temp$FG3_PCT_away_bench)
str(temp)

# Remove missing observations
row.has.na <- apply(temp, 1, function(x){any(is.na(x))})
sum(row.has.na)
temp <- temp[!row.has.na,]
table(is.na(temp)) #Verify removal of rows containing NA

# Separate Predictors from Response
pred <- temp[,-21]
response <- factor(temp[,21])
levels(response) = c('no', 'yes')
plot(response) #slightly unbalanced

# Histograms
library(psych)
multi.hist(pred, density=FALSE); #Histograms
library(e1071)
print(skewValues <- apply(pred, 2, skewness)) #Skewness Values

# Outliers
par(mfrow=c(2,5))
boxplot(pred$FG_PCT_home, main="FG%Home")
boxplot(pred$FT_PCT_home, main="FT%Home")
boxplot(pred$FG3_PCT_home, main="FG3%Home")
boxplot(pred$AST_home, main="ASTHome")
boxplot(pred$REB_home, main="REBHome")
boxplot(pred$FG_PCT_home_bench, main="FG%HomeB") ## Removed correlation
boxplot(pred$FT_PCT_home_bench, main="FT%HomeB")
```

```
boxplot(pred$FG3_PCT_home_bench, main="FG3%HomeB")
boxplot(pred$AST_home_bench, main="ASTHomeB")
boxplot(pred$REB_home_bench, main="REBHomeB")
boxplot(pred$FG_PCT_away, main="FG%Away")
boxplot(pred$FT_PCT_away, main="FT%Away")
boxplot(pred$FG3_PCT_away, main="FG3%Away")
boxplot(pred$AST_away, main="ASTAway")
boxplot(pred$REB_away, main="REBAway")
boxplot(pred$FG_PCT_away_bench, main="FG%AwayB") ## Removed correlation
boxplot(pred$FT_PCT_away_bench, main="FT%AwayB")
boxplot(pred$FG3_PCT_away_bench, main="FG3%AwayB")
boxplot(pred$AST_away_bench, main="ASTAwayB")
boxplot(pred$REB_away_bench, main="REBAwayB")

# Correlation - Remove predictors with very high correlation
library(corrplot)
library(caret)
par(mfrow=c(1,1))
correlations <- cor(pred)
corrplot(correlations, order = 'hclust') #Correlation Plot
print(highCorr <- findCorrelation(correlations, cutoff = .85))
pred <- pred[,-highCorr] #removed two predictors

# Apply Transformations
cleanup <- preProcess(pred,method=c("BoxCox","center","scale", "spatialSign"))
predictors <- predict(cleanup,pred)

# Histograms After Transformations
library(psych)
multi.hist(predictors, density=FALSE); #Histograms
library(e1071)
print(skewValues <- apply(predictors, 2, skewness)) #Skewness Values

# Outliers After Transformations
par(mfrow=c(2,5))
boxplot(predictors$FG_PCT_home, main="FG%Home")
boxplot(predictors$FT_PCT_home, main="FT%Home")
boxplot(predictors$FG3_PCT_home, main="FG3%Home")
boxplot(predictors$AST_home, main="ASTHome")
boxplot(predictors$REB_home, main="REBHome")
boxplot(predictors$FG_PCT_home_bench, main="FG%HomeB") ## Removed for correlation
boxplot(predictors$FT_PCT_home_bench, main="FT%HomeB")
boxplot(predictors$FG3_PCT_home_bench, main="FG3%HomeB")
boxplot(predictors$AST_home_bench, main="ASTHomeB")
boxplot(predictors$REB_home_bench, main="REBHomeB")
boxplot(predictors$FG_PCT_away, main="FG%Away")
boxplot(predictors$FT_PCT_away, main="FT%Away")
boxplot(predictors$FG3_PCT_away, main="FG3%Away")
```

```
boxplot(predictors$AST_away, main="ASTAway")
boxplot(predictors$REB_away, main="REBAway")
boxplot(predictors$FG_PCT_away_bench, main="FG%AwayB") ## Removed for correlation
boxplot(predictors$FT_PCT_away_bench, main="FT%AwayB")
boxplot(predictors$FG3_PCT_away_bench, main="FG3%AwayB")
boxplot(predictors$AST_away_bench, main="ASTAwayB")
boxplot(predictors$REB_away_bench, main="REBAwayB")
```

Training/Testing Split

```
set.seed(314)
trainingRows <- createDataPartition(response, p = .80, list= FALSE)
#separate the data by injury with 80% training and 20% testing.
trP <- predictors[trainingRows, ]
trR <- response[trainingRows]
teP <- predictors[-trainingRows, ]
teR <- response[-trainingRows]
```

Linear Methods

```
ctrl <- trainControl(method = "LGOCV",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

###### Linear Discriminant Analysis ######
set.seed(314)
LDAFull <- train(trP, trR ,
                 method = "lda",
                 metric = "ROC",
                 trControl = ctrl)
LDAFull

## LDA Test Confusion
pred = predict(LDAFull, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## LDA Test ROC
pred = predict(LDAFull, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
```

```
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### Logistic Regression ######
set.seed(314)
logistic <- train(trP, trR ,
                  method = "glm",
                  metric = "ROC",
                  trControl = ctrl)
logistic

## Logistic Test Confusion
pred = predict(logistic, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## Logistic Test ROC
pred = predict(logistic, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### Partial Least Squares Discriminant Analysis ######
set.seed(314)
plsFit2 <- train(trP, trR ,
                 method = "pls",
                 tuneGrid = expand.grid(.ncomp = 1:10),
                 preProc = c("center","scale"),
                 metric = "ROC",
                 trControl = ctrl)

plsFit2
plot(plsFit2)

## PLS Test Confusion
pred = predict(plsFit2, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## PLS Test ROC
pred = predict(plsFit2, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))
```

```
###### Penalized Model ######
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                        .lambda = seq(.01, .2, length = 10))
set.seed(314)
glmnTuned <- train(trP,trR,
                   method = "glmnet",
                   tuneGrid = glmnGrid,
                   preProc = c("center", "scale"),
                   metric = "ROC",
                   trControl = ctrl)
glmnTuned
plot(glmnTuned, plotType = "level")

## GLMN Test Confusion
pred = predict(glmnTuned, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## GLMN Test ROC
pred = predict(glmnTuned, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))
```

Non-Linear Methods

```
ctrl <- trainControl(method = "LGOCV",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

###### FDA Method ######
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:20)
set.seed(314)
fdaTuned <- train(x = trP,
                  y = trR,
                  method = "fda",
                  metric = "ROC",
                  tuneGrid = marsGrid,
                  trControl = ctrl)

fdaTuned
plot(fdaTuned)
```

```
## FDA Test Confusion
pred = predict(fdaTuned, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## FDA Test ROC
pred = predict(fdaTuned, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### KNN Method ######
library(caret)
set.seed(314)
knnFit <- train(x = trP,
                y = trR,
                method = "knn",
                metric = "ROC",
                preProc = c("center", "scale"),
                tuneGrid = data.frame(.k = 1:80),
                trControl = ctrl)
knnFit
plot(knnFit)

## KNN Test Confusion
pred = predict(knnFit, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## KNN Test ROC
pred = predict(knnFit, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
par(mfrow=c(1,1))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### MDA Method ######
set.seed(314)
mdaFit <- train(x = trP,
                y = trR,
                method = "mda",
                metric = "ROC",
                tuneGrid = expand.grid(.subclasses = 1:3),
```

```
                trControl = ctrl)
mdaFit

## MDA Test Confusion
pred = predict(mdaFit, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## MDA Test ROC
pred = predict(mdaFit, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

####### Naive Bayes Method #######
library(klaR)
set.seed(314)
nbFit <- train( x = trP,
                y = trR,
                method = "nb",
                metric = "ROC",
                tuneGrid = data.frame(.fL = 2,.usekernel = TRUE,.adjust = TRUE),
                trControl = ctrl)

nbFit

## NB Test Confusion
pred = predict(nbFit, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## NB Test ROC
pred = predict(nbFit, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### Neural Net Method ######
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, .3, .5, 1))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (18 + 1) + (maxSize+1)*2)
set.seed(314)
nnetFit <- train(x = trP,
                 y = trR,
```

```
                 method = "nnet",
                 metric = "ROC",
                 tuneGrid = nnetGrid,
                 trace = FALSE,
                 maxit = 2000,
                 MaxNWts = numWts,
                 trControl = ctrl)
nnetFit
plot(nnetFit)

## NNet Test Confusion
pred = predict(nnetTune, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## NNet Test ROC
pred = predict(nnetTune, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))


###### QDA Method ######
set.seed(314)
qdaFit <- train(x = trP,
                y = trR,
                method = "qda",
                metric = "ROC",
                trControl = ctrl)
qdaFit

## QDA Test Confusion
pred = predict(qdaFit, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## QDA Test ROC
pred = predict(qdaFit, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))

###### RDA Method ######
set.seed(314)
```

```
rdaGrid <- expand.grid(.gamma = seq(0, 1, length = 11),
                       .lambda = seq(0, 1, length = 11))
rdaFit <- train(x = trP,
                y = trR,
                method = "rda",
                metric = "ROC",
                tuneGrid = rdaGrid,
                trControl = ctrl)
rdaFit
plot(rdaFit)

## RDA Test Confusion
pred = predict(rdaFit, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)

## RDA Test ROC
pred = predict(rdaFit, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))


###### SVMR Method ######
library(kernlab)
library(caret)
sigmaRangeReduced <- sigest(as.matrix(trP))
svmRGridReduced <- expand.grid(.sigma = sigmaRangeReduced[1],
                               .C = 2^(seq(-4, 6)))
set.seed(314)
svmRModel <- train(x = trP,
                   y = trR,
                   method = "svmRadial",
                   metric = "ROC",
                   preProc = c("center", "scale"),
                   tuneGrid = svmRGridReduced,
                   fit = FALSE,
                   trControl = ctrl)
svmRModel
plot(svmRModel)

## SVMR Test Confusion
pred = predict(svmRModel, teP)
test.conf = confusionMatrix(pred, teR)
print(test.conf)
```

```
## SVMR Test ROC
library(pROC)
pred = predict(svmRModel, teP, type = "prob")
test.roc = roc(response = teR,
               predictor = pred$yes,
               levels = rev(levels(teR)))
plot(test.roc, legacy.axes = TRUE)
print(auc(test.roc))
```

Variable Imporance for Optimal Model

```
library(caret)
plot(varImp(svmRModel), 5, main="SVMR Predictor Importance")
```