OR 568 Project

Kelly Johnson

2/7/2022

LOAD AND PREPROCESS

Load Data to use and perform preliminary exploration

```
data(PimaIndiansDiabetes)
dat <- PimaIndiansDiabetes
str(dat)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ pregnant: num  6 1 8 1 0 5 3 10 2 8 ...
##  $ glucose : num  148 85 183 89 137 116 78 115 197 125 ...
##  $ pressure: num  72 66 64 66 40 74 50 0 70 96 ...
##  $ triceps : num  35 29 0 23 35 0 32 0 45 0 ...
##  $ insulin : num  0 0 0 94 168 0 88 0 543 0 ...
##  $ mass    : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ pedigree: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ age     : num  50 31 32 21 33 30 26 29 53 54 ...
##  $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 1 2 2 ...
```

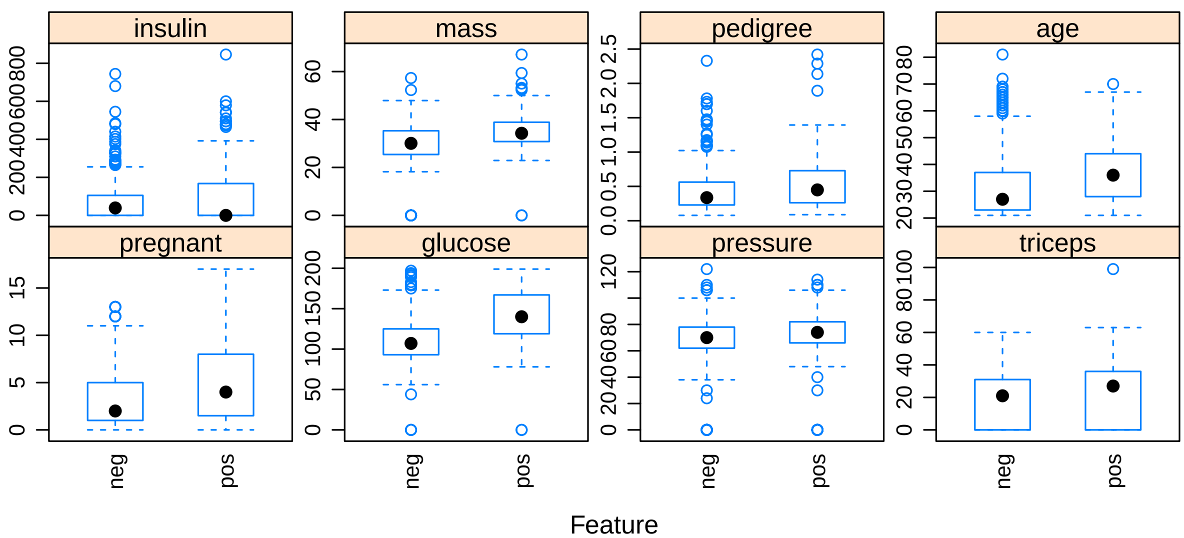
```
summary(dat)
```

```
##     pregnant         glucose         pressure         triceps     
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00  
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00  
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00  
##  Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54  
##  3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00  
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00  
##     insulin           mass          pedigree           age        diabetes
##  Min.   :  0.0   Min.   : 0.00   Min.   :0.0780   Min.   :21.00   neg:500 
##  1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437   1st Qu.:24.00   pos:268 
##  Median : 30.5   Median :32.00   Median :0.3725   Median :29.00           
##  Mean   : 79.8   Mean   :31.99   Mean   :0.4719   Mean   :33.24           
##  3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00           
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200   Max.   :81.00           
```

```r
#Isolate the response
x <- dat[ , -(9)]
y <- dat[,9]
table(y)
```

```
## y
## neg pos
## 500 268
```

```r
#Box Plots
featurePlot(x = x,
            y = y,
            plot = "box",
            scales = list(y = list(relation="free"),
                          x = list(rot = 90)),
            layout = c(4,2 ),
            auto.key = list(columns = 4))
```



There are 768 samples, 8 potential predictors that are numeric and 1 categorical response with two classes:

- diabetes
  - 1 - "pos" (has diabetes)
  - 2 - "neg" (does not have diabetes)

Check for near zero variance

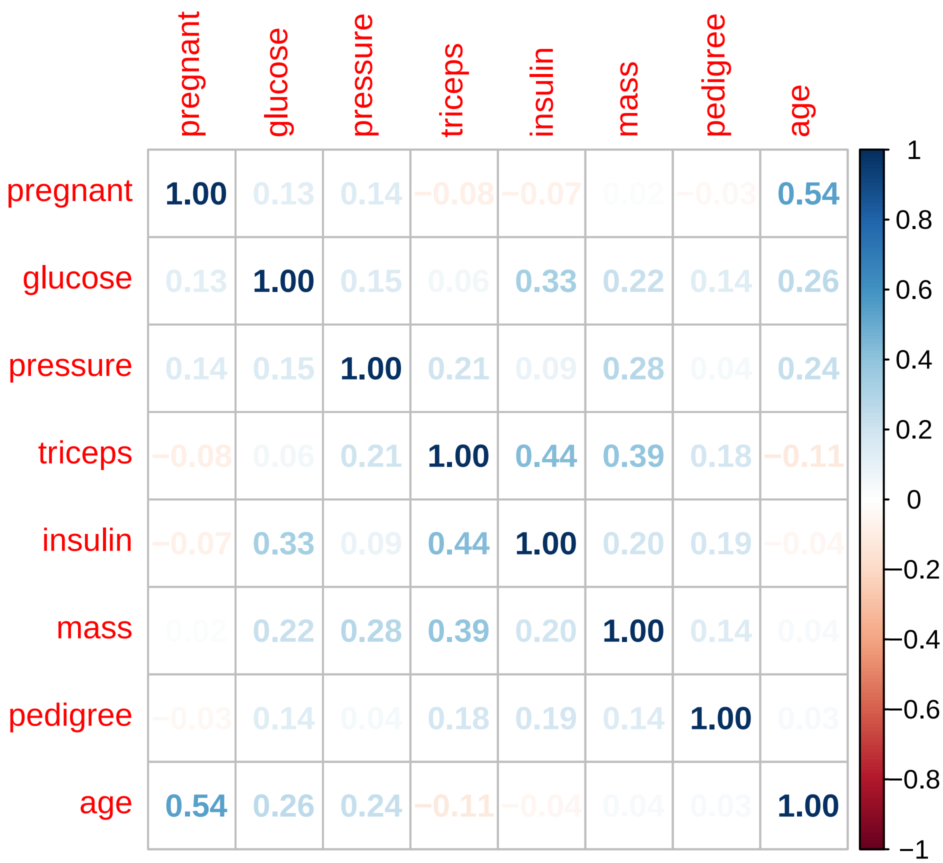
```r
nearZeroVar(x)
```

```
## integer(0)
```

There is no near zero variances

Check for correlated Predictors

```r
corrplot(cor(x), method= "number")
```




```r
findCorrelation(cor(x), cutoff = .55)
```

```
## integer(0)
```

There is no correlation between predictors with a pearsons coefficient greater than 0.55. Age and pregnant have the lowest and insignificant correlation coefficient.

Check for linear depencencies

```r
findLinearCombos(x)
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

There are no linear dependencies for the predictors.

Count Zeros by column

```
#Count the number of zeros per column
zerobycolumn <-colSums(dat==0)
zerobycolumn
```

```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
##      111        5       35      227      374       11        0        0
## diabetes
##        0
```

There is a lot of zeros. Pregnant would be the only zero reading that would be accurate.Therefore, we can impute 0's rather than omitting them in our analyses

To impute the 0's, we first change all of the 0's to NA's with the exeption of the pregnant predictor. Then using the 'bagImpute' method from the caret package, we will impute the missing values. This method is the bagging (bootstrap aggregating) of regression trees. It provides the recovery of missing values for several variables at once, based on regression dependencies. This method takes each predictor in the data, created a bagged tree using all of the other predictors in the train.dummy set. The bagged model is used to predict the missing values. The computational cost of this method is afforded by the size of the dataset. Then columns from the train.dummy set that were predicted we used to replace columns that had missing values.

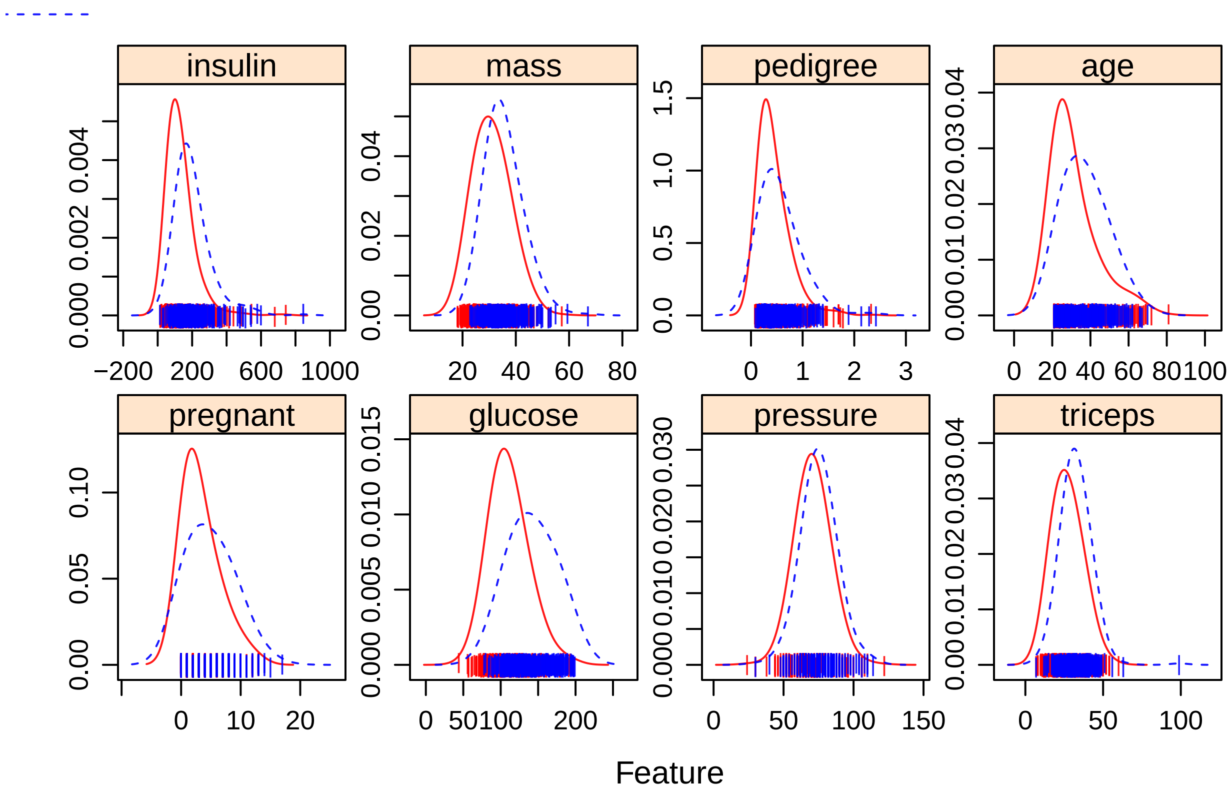
Replace Zeros

```
# replace zeros with NA
x[x == 0] <- NA

#Return Pregnant NA back to 0(zer0)
x$pregnant[is.na(x$pregnant)] <- 0

# Transform all feature to dummy variables.
dummy.vars <- dummyVars(~ ., data = x)
train.dummy <- predict(dummy.vars, x)

#impute
pre.process <- preProcess(train.dummy, method = "bagImpute")
imputed.data <- predict(pre.process, train.dummy)

#Replace zeros with imputed dummy variables
x$glucose <- imputed.data[,2]
x$pressure <- imputed.data[,3]
x$triceps <- imputed.data[,4]
x$insulin <- imputed.data[,5]
x$mass <- imputed.data[,6]

#Check to make sure that it worked
zerobycolumn <-colSums(x==0)
summary(x)
```

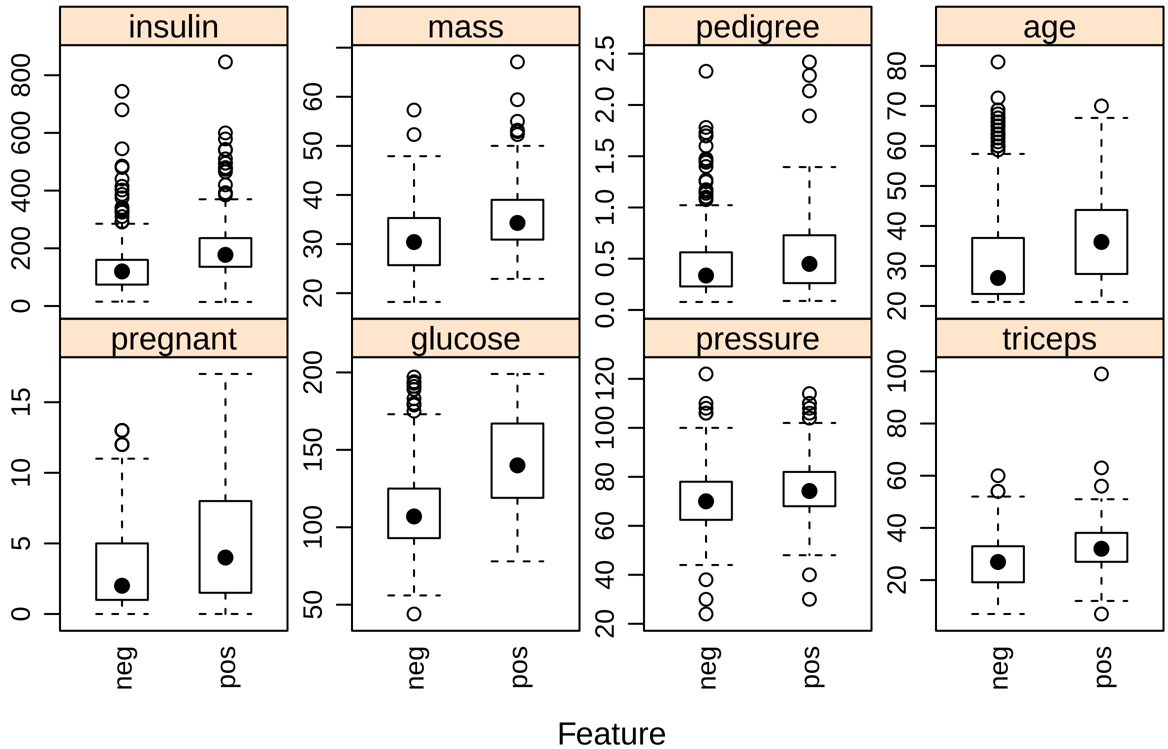
```
##     pregnant         glucose         pressure         triceps     
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00  
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:22.00  
##  Median : 3.000   Median :117.0   Median : 72.00   Median :28.64  
##  Mean   : 3.845   Mean   :121.7   Mean   : 72.36   Mean   :28.88  
##  3rd Qu.: 6.000   3rd Qu.:140.4   3rd Qu.: 80.00   3rd Qu.:35.02  
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00  
##     insulin           mass          pedigree           age       
##  Min.   : 14.0   Min.   :18.20   Min.   :0.0780   Min.   :21.00  
##  1st Qu.: 86.5   1st Qu.:27.50   1st Qu.:0.2437   1st Qu.:24.00  
##  Median :135.8   Median :32.30   Median :0.3725   Median :29.00  
##  Mean   :155.4   Mean   :32.47   Mean   :0.4719   Mean   :33.24  
##  3rd Qu.:191.2   3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00  
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200   Max.   :81.00  
```

```
#Density Plots
transparentTheme(trans = .9)
featurePlot(x = x,
            y = y,
            plot = "density",
            scales = list(x =list(relation="free"),
                          y =list(relation="free")),
            adjust = 2.5,
            pch = "|",
            layout = c(4, 2),
            auto.key = list(columns = 8))
```

```
## Warning in draw.key(simpleKey(...), draw = FALSE): not enough rows for columns
```




```r
#Box Plots
featurePlot(x = x,
            y = y,
            plot = "box",
            scales = list(y = list(relation="free"),
                          x = list(rot = 90)),
            layout = c(4,2 ),
            auto.key = list(columns = 4))
```



Split the data

```
set.seed(2323)
indexes <- createDataPartition(y,times = 1,p = 0.7,list = FALSE)

trainx <- x[indexes,]
testx <- x[-indexes,]
trainy <- y[indexes]
testy <- y[-indexes]


# Examine the proportions of the response class label across the datasets.
prop.table(table(dat$diabetes))
```

```
##
##       neg       pos
## 0.6510417 0.3489583
```

```
prop.table(table(trainy))
```

```
## trainy
##       neg       pos
## 0.6505576 0.3494424
```

```r
prop.table(table(testy))
```

```
## testy
##       neg       pos
## 0.6521739 0.3478261
```

TRAIN MODELS

Logistic Regression Training Models

Here we are falling down the rabbit hole to see if there are any significant differences in four different preprocessing methods with the Logistic Regression Training Models. We will attempt to preprocess with simple center and scaling for all four. Additionally, on each of the other three models we will try the Box Cox transformation, Yeo Johnson Transformation, and Principal Component Analysis(PCA).

```r
#Logistic Regression: Training Model
#Control Object
ctrl <- trainControl(method = "cv", number = 10,
                     classProbs = T, summaryFunction =
                     twoClassSummary,savePredictions = T)
#Center and Scale
set.seed(2345)
lr_train_data <- train(x=trainx,y=trainy,
                              method = "glm",
                              metric = "ROC",
                              preProcess =
                              c("center","scale"),
                              tuneLength = 10,
                              trControl = ctrl )

lr_train_data
```

```
## Generalized Linear Model
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8506767  0.8828571  0.5961988
```

```r
#Yeo Johnson
set.seed(2345)
lr_train_dataYJ <- train(x=trainx,y=trainy,
                              method = "glm",
```

```r
                              metric = "ROC",
                              preProcess =
                              c("center","scale","YeoJohnson"),
                              tuneLength = 10,
                              trControl = ctrl)

lr_train_dataYJ
```

```
## Generalized Linear Model
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Yeo-Johnson transformation (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8591813  0.8685714  0.6333333
```

```r
#Box Cox
set.seed(2345)
lr_train_dataBC <- train(x=trainx,y=trainy,
                              method = "glm",
                              metric = "ROC",
                              preProcess =
                              c("center","scale","BoxCox"),
                              tuneLength = 10,
                              trControl = ctrl)

lr_train_dataBC
```

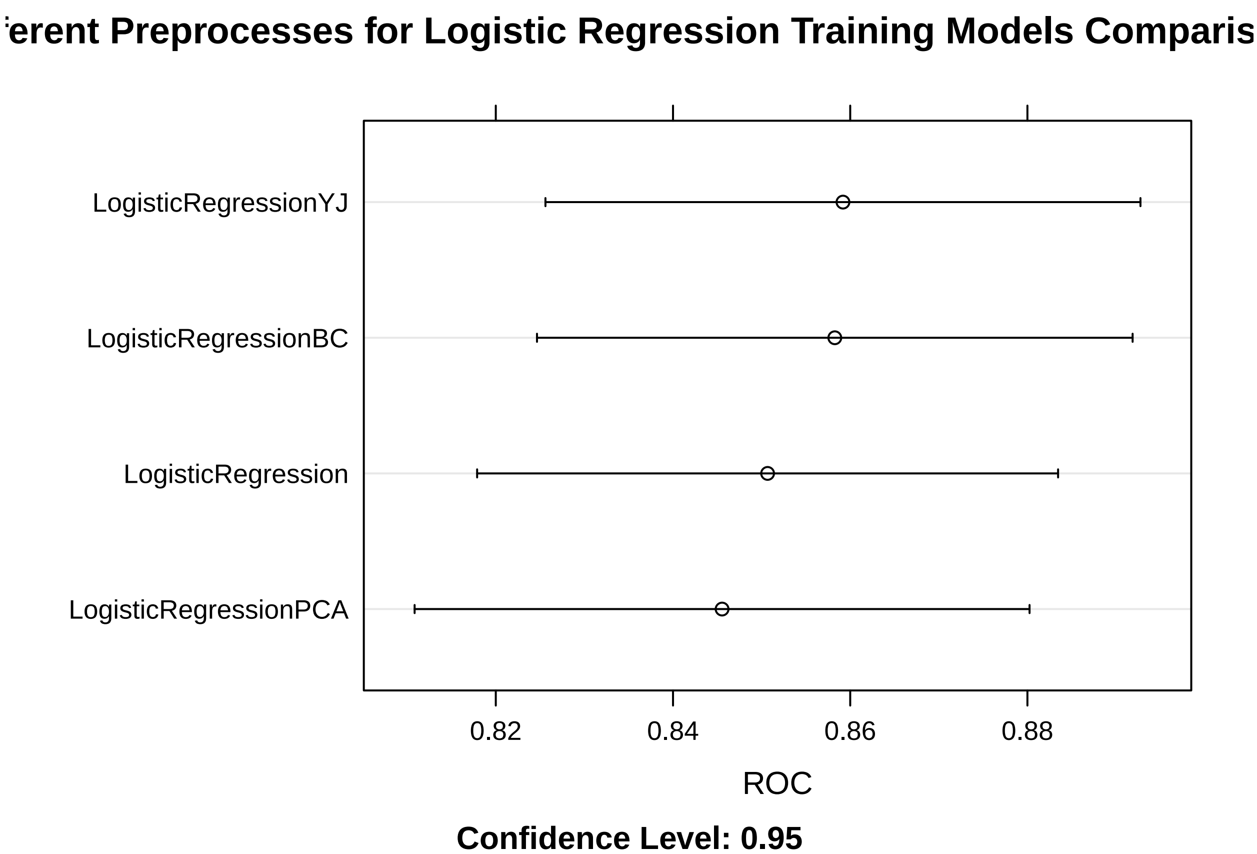
```
## Generalized Linear Model
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Box-Cox transformation (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8582623  0.8685714  0.6333333
```

```r
#PCA
set.seed(2345)
lr_train_dataPCA <- train(x=trainx,y=trainy,
                              method = "glm",
```

```r
                              metric = "ROC",
                              preProcess =
                              c("center","scale","pca"),
                              tuneLength = 10,
                              trControl = ctrl)

lr_train_dataPCA
```

```
## Generalized Linear Model
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), principal component signal
##  extraction (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8455388  0.8685714  0.5807018
```

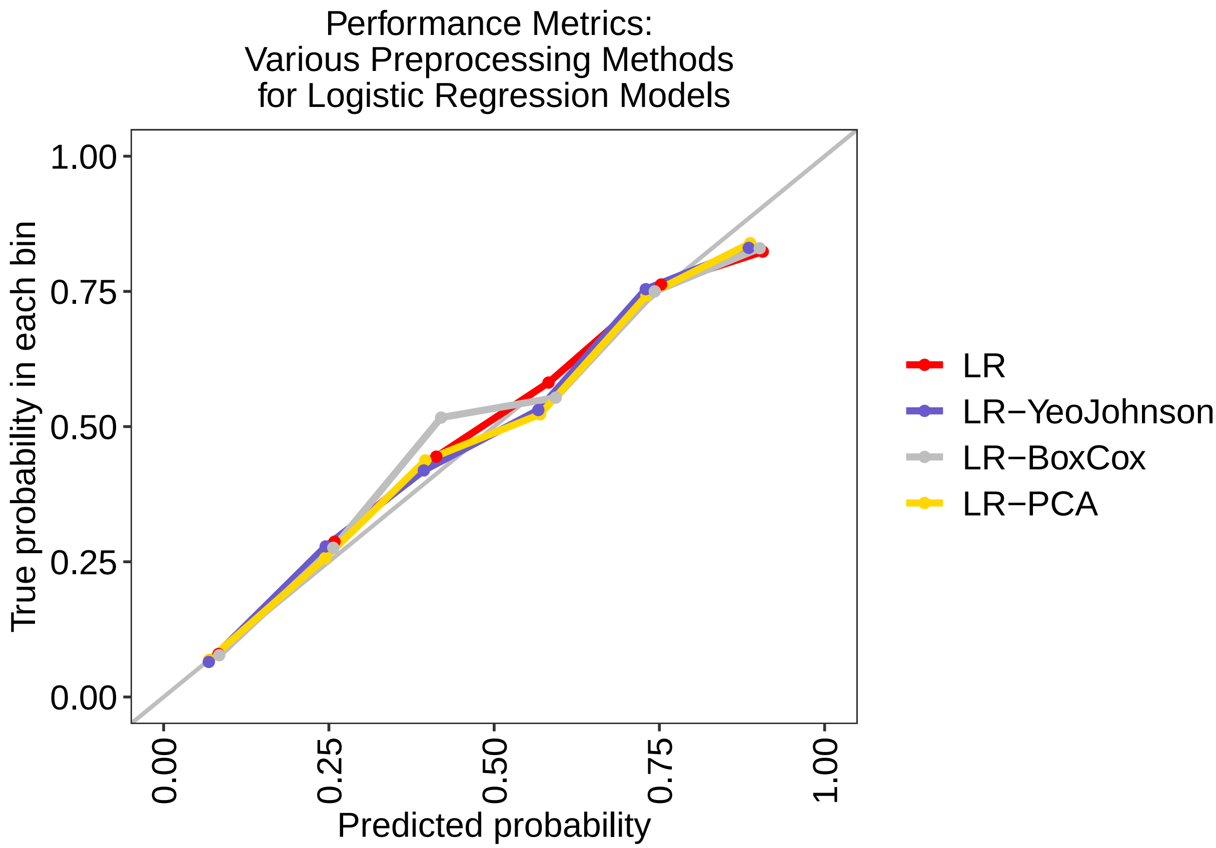
```r
#Comparision of different preprocesses on the logistic regression training model
#(Yeo Johnson, Box Cox, PCA, and simple center and scaling).
lrTrainComp <- list(LogisticRegression = lr_train_data,
                    LogisticRegressionYJ = lr_train_dataYJ,
                    LogisticRegressionBC = lr_train_dataBC,
                    LogisticRegressionPCA = lr_train_dataPCA)

resampleLogisticRegression <- resamples(lrTrainComp)

dotplot(resampleLogisticRegression, metric="ROC",
        main="Different Preprocesses for Logistic Regression Training Models Comparision")
```

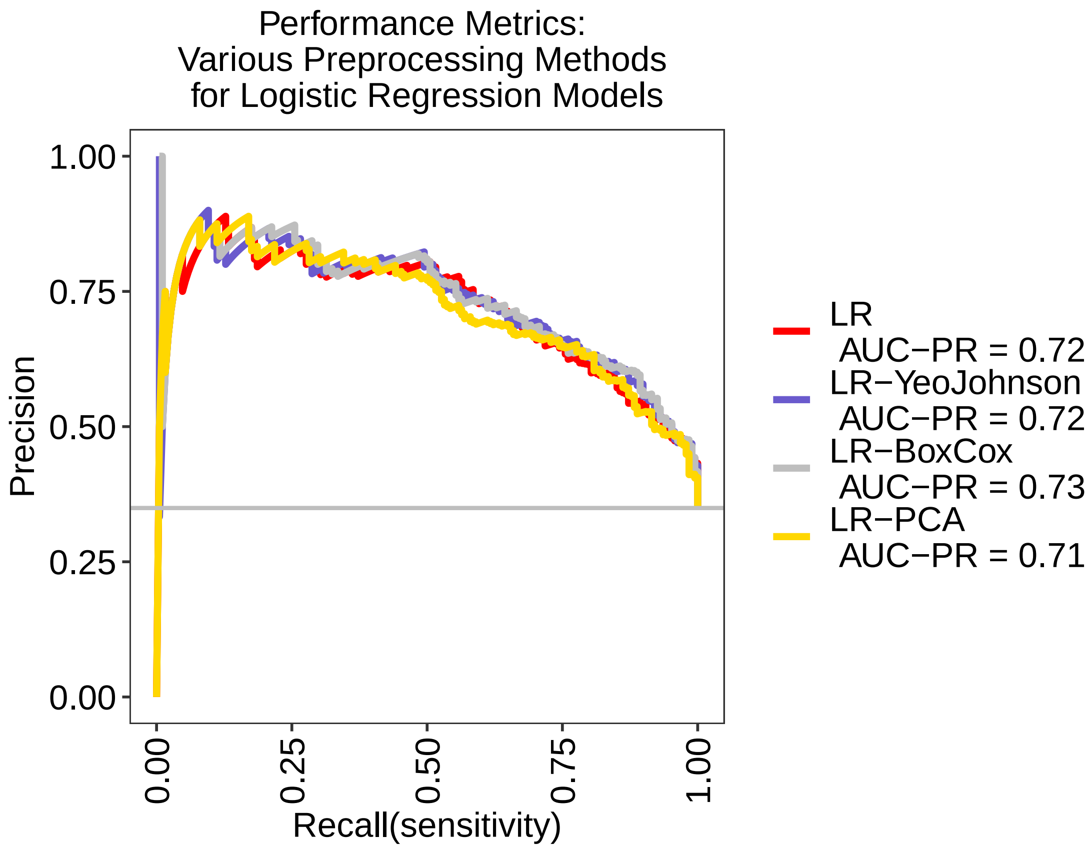


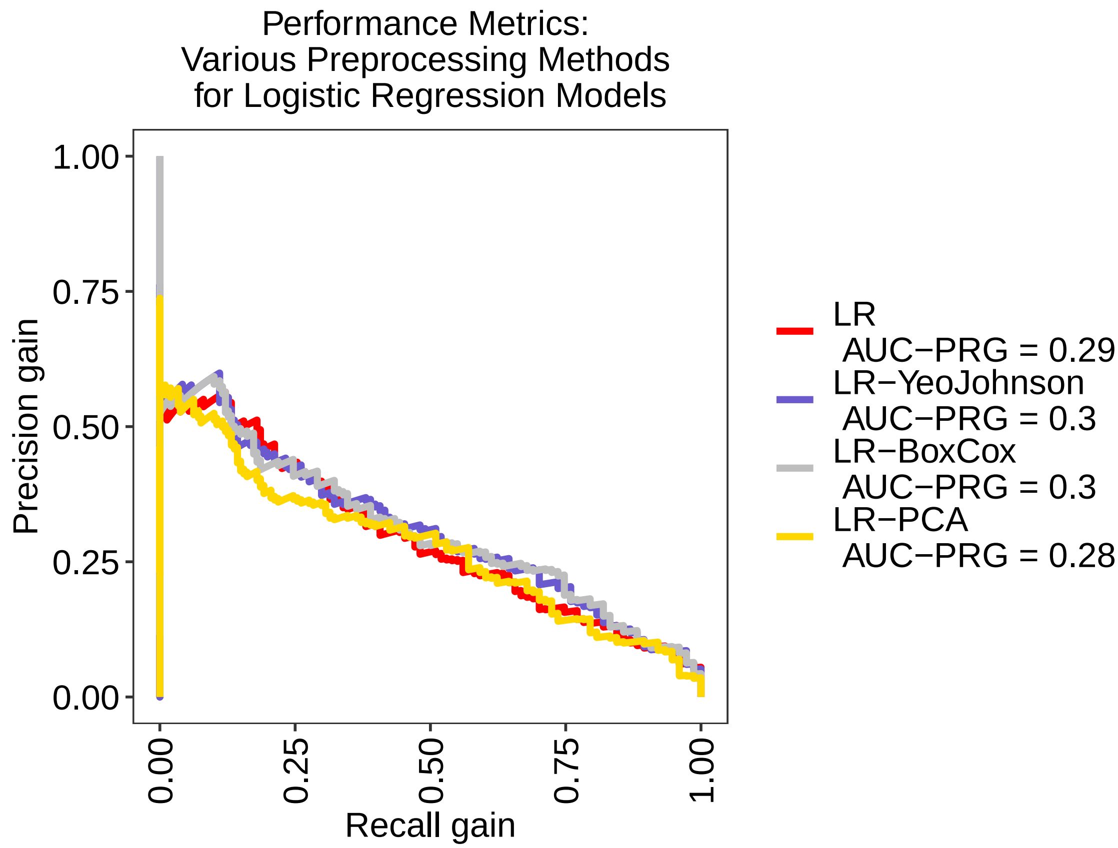

```
#MLeval:evalm() is for machine learning model evaluation.
#The function can accept the Caret 'train' function results
#to evaluate machine learning predictions or a data frame
#of probabilities and ground truth labels can be passed in
#to evaluate

names<- c("LR","LR-YeoJohnson","LR-BoxCox","LR-PCA")
res <- evalm(lrTrainComp, gnames = names,title="Performance Metrics: \nVarious Preprocessing Methods \n
```

```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Group 2 type: cv

## Group 3 type: cv

## Group 4 type: cv

## Observations: 2152
```

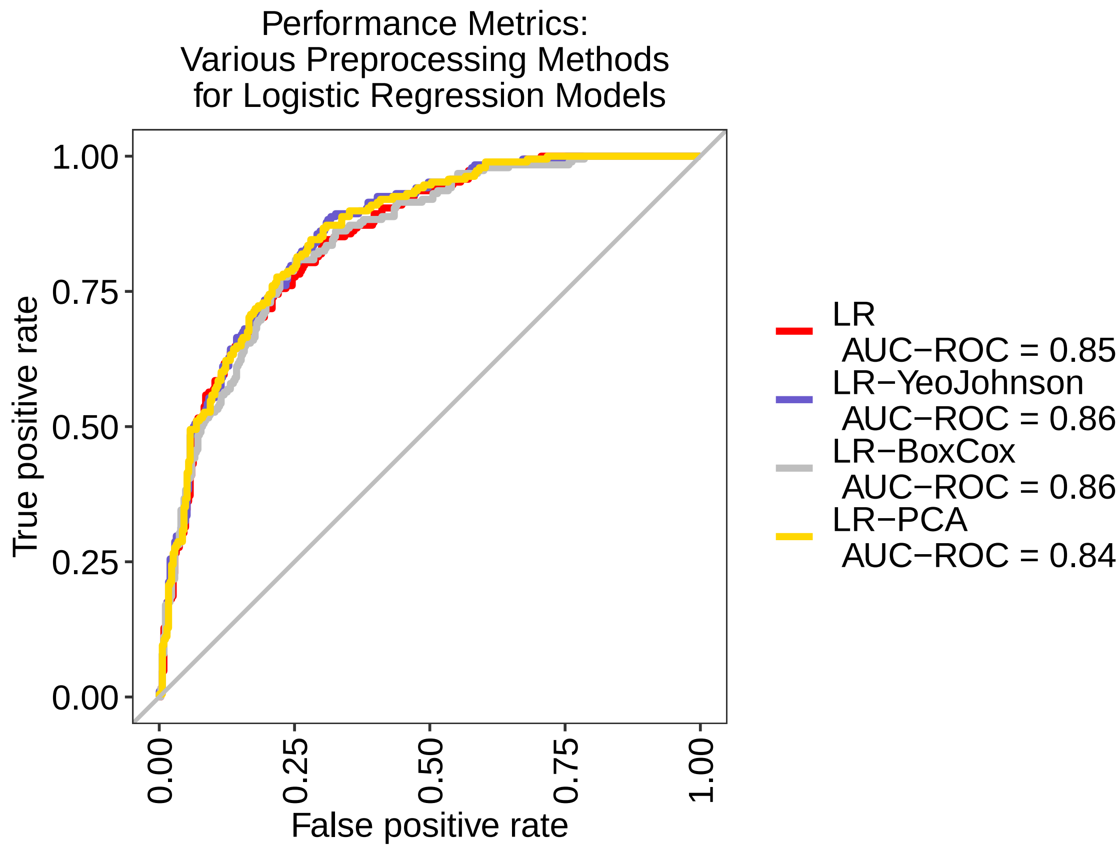
```
## Number of groups: 4

## Observations per group: 538

## Positive: pos

## Negative: neg

## Group: LR

## Positive: 188

## Negative: 350

## Group: LR-YeoJohnson

## Positive: 188

## Negative: 350

## Group: LR-BoxCox

## Positive: 188

## Negative: 350

## Group: LR-PCA

## Positive: 188

## Negative: 350

## ***Performance Metrics***
```


Performance Metrics:
Various Preprocessing Methods
for Logistic Regression Models
True probability in each bin
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted probability
LR
LR−YeoJohnson
LR−BoxCox
LR−PCA


Performance Metrics:
Various Preprocessing Methods
for Logistic Regression Models
1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
LR
AUC−PR = 0.72
LR−YeoJohnson
AUC−PR = 0.72
LR−BoxCox
AUC−PR = 0.73
LR−PCA
AUC−PR = 0.71




```
## LR Optimal Informedness = 0.537568389057751

## LR-YeoJohnson Optimal Informedness = 0.565744680851064

## LR-BoxCox Optimal Informedness = 0.571550151975684

## LR-PCA Optimal Informedness = 0.557082066869301

## LR AUC-ROC = 0.85

## LR-YeoJohnson AUC-ROC = 0.86

## LR-BoxCox AUC-ROC = 0.86

## LR-PCA AUC-ROC = 0.84
```




```
## get ROC
#res$roc

## get calibration curve
#res$cc

## get precision recall gain curve
#res$prg
```

K-Nearest Neighbors (KNN)

```
#Tune Grid
knnTG = expand.grid(.k = c(3:10))

#Center and Scale
set.seed(2345)
knn_train_data <- train(x=trainx,y=trainy,
                        method = "knn",
                        metric = "ROC",
                        tuneGrid = knnTG,
                        preProcess =
                        c("center","scale"),
                        tuneLength = 10,
                        trControl = ctrl )
```

```r
knn_train_data
```

```
## k-Nearest Neighbors
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    3  0.7799206  0.8028571  0.6073099
##    4  0.7997327  0.8085714  0.6502924
##    5  0.8140351  0.8200000  0.6505848
##    6  0.8299290  0.8257143  0.6763158
##    7  0.8315748  0.8371429  0.6494152
##    8  0.8334127  0.8485714  0.6233918
##    9  0.8290017  0.8400000  0.6444444
##   10  0.8286759  0.8342857  0.6017544
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 8.
```

```r
#Yeo Johnson
set.seed(2345)
knn_train_dataYJ <- train(x=trainx,y=trainy,
                              method = "knn",
                              metric = "ROC",
                              tuneGrid = knnTG,
                              preProcess =
                              c("center","scale","YeoJohnson"),
                              tuneLength = 10,
                              trControl = ctrl)

knn_train_dataYJ
```

```
## k-Nearest Neighbors
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Yeo-Johnson transformation (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    3  0.7678530  0.8228571  0.6391813
```

```
##    4  0.7875689  0.8228571  0.5964912
##    5  0.8108730  0.8200000  0.6558480
##    6  0.8103676  0.7971429  0.6391813
##    7  0.8117878  0.8028571  0.6385965
##    8  0.8219256  0.8228571  0.6500000
##    9  0.8273559  0.8114286  0.6497076
##   10  0.8280326  0.8314286  0.6441520
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.
```

```
#Box Cox
set.seed(2345)
knn_train_dataBC <- train(x=trainx,y=trainy,
                              method = "knn",
                              metric = "ROC",
                              tuneGrid = knnTG,
                              preProcess =
                              c("center","scale","BoxCox"),
                              tuneLength = 10,
                              trControl = ctrl)

knn_train_dataBC
```

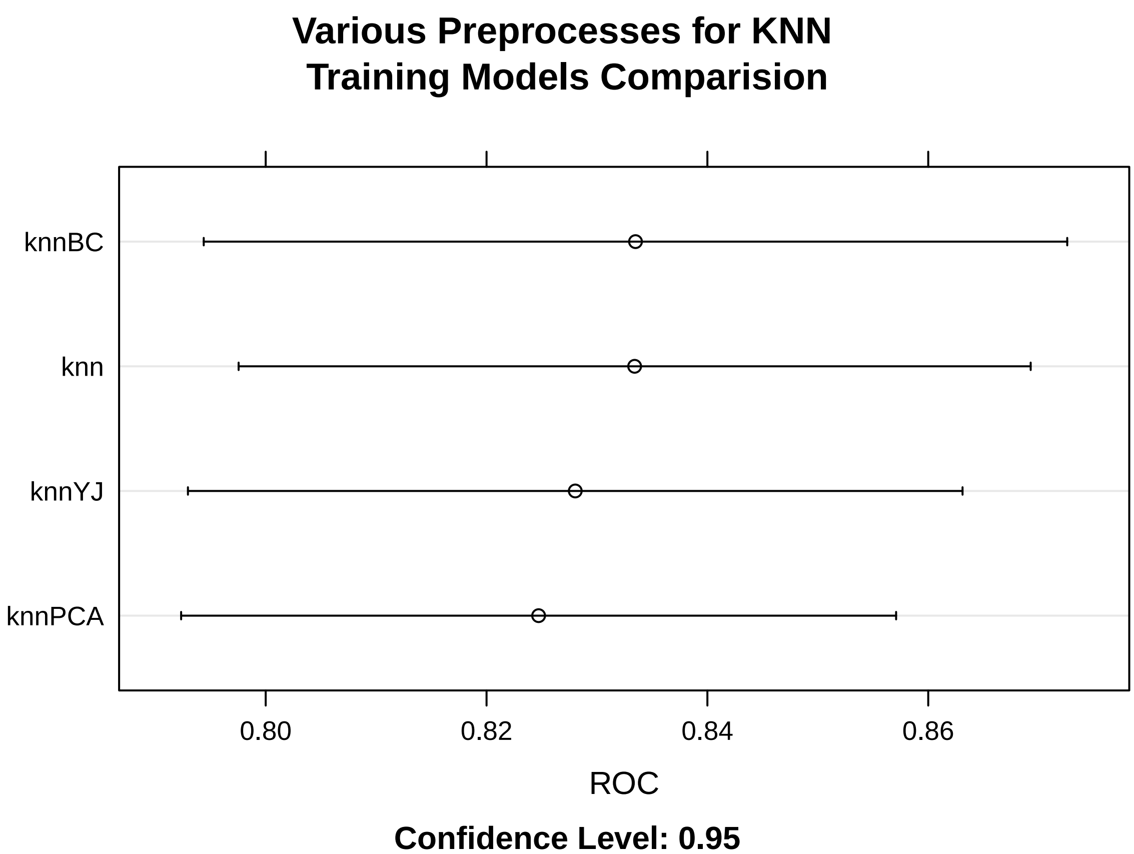
```
## k-Nearest Neighbors
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Box-Cox transformation (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    3  0.7660192  0.8085714  0.6020468
##    4  0.8000334  0.8057143  0.6134503
##    5  0.8108271  0.8314286  0.6447368
##    6  0.8193442  0.8228571  0.6339181
##    7  0.8235422  0.8200000  0.6502924
##    8  0.8284837  0.8257143  0.6178363
##    9  0.8265873  0.8342857  0.6500000
##   10  0.8334879  0.8257143  0.6447368
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.
```

```
#PCA
set.seed(2345)
knn_train_dataPCA <- train(x=trainx,y=trainy,
                              method = "knn",
                              metric = "ROC",
```

```r
                              tuneGrid = knnTG,
                              preProcess =
                              c("center","scale","pca"),
                              tuneLength = 10,
                              trControl = ctrl)

knn_train_dataPCA
```

```
## k-Nearest Neighbors
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), principal component signal
##  extraction (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    3  0.7702297  0.8028571  0.6070175
##    4  0.7864996  0.7914286  0.5532164
##    5  0.8014620  0.8142857  0.5959064
##    6  0.8126358  0.8314286  0.6169591
##    7  0.8116291  0.8457143  0.6122807
##    8  0.8185631  0.8485714  0.5482456
##    9  0.8247160  0.8342857  0.6011696
##   10  0.8242941  0.8428571  0.5856725
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

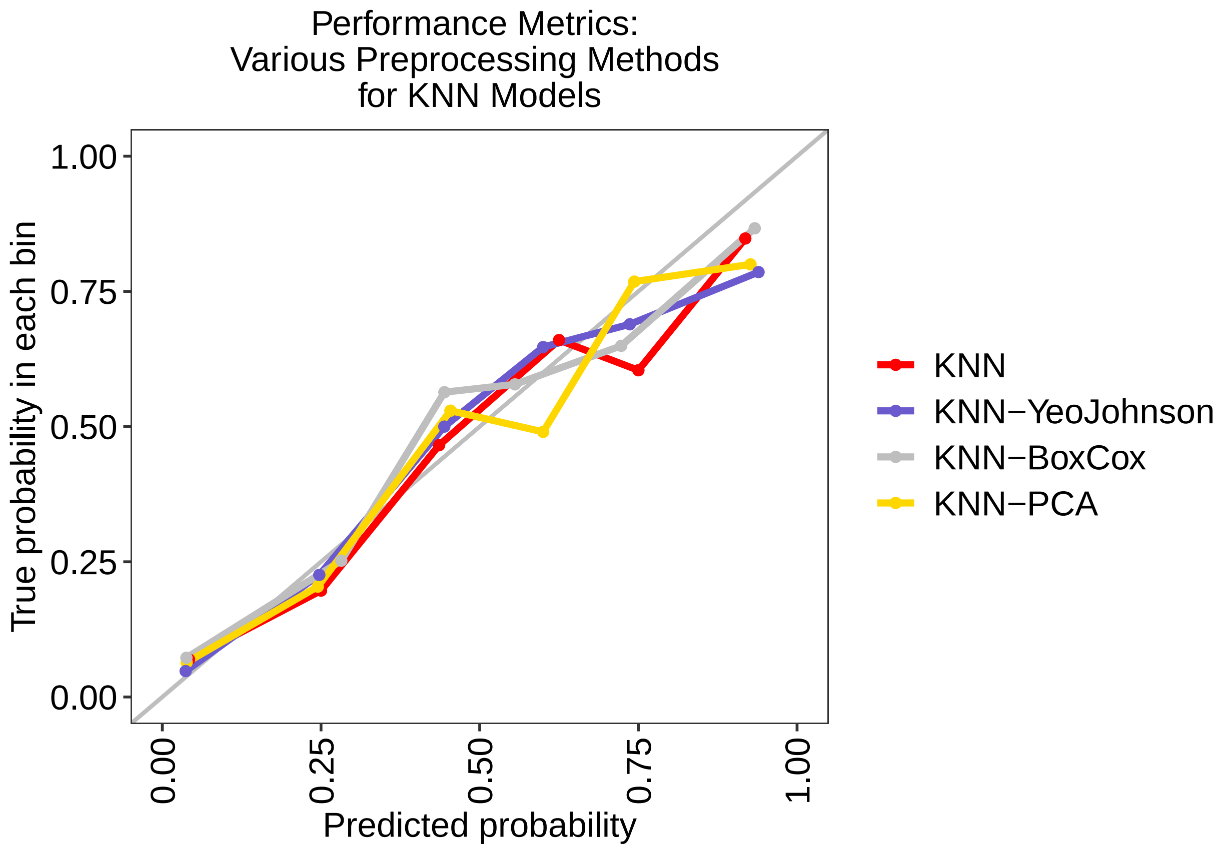
```r
#Comparision of different preprocesses on the knn training model
#(Yeo Johnson, Box Cox, PCA, and simple center and scaling).
knnTrainComp <- list(knn = knn_train_data,
                    knnYJ = knn_train_dataYJ,
                    knnBC = knn_train_dataBC,
                    knnPCA = knn_train_dataPCA)

resampleknn <- resamples(knnTrainComp)

dotplot(resampleknn, metric="ROC",
        main="Various Preprocesses for KNN \nTraining Models Comparision")
```

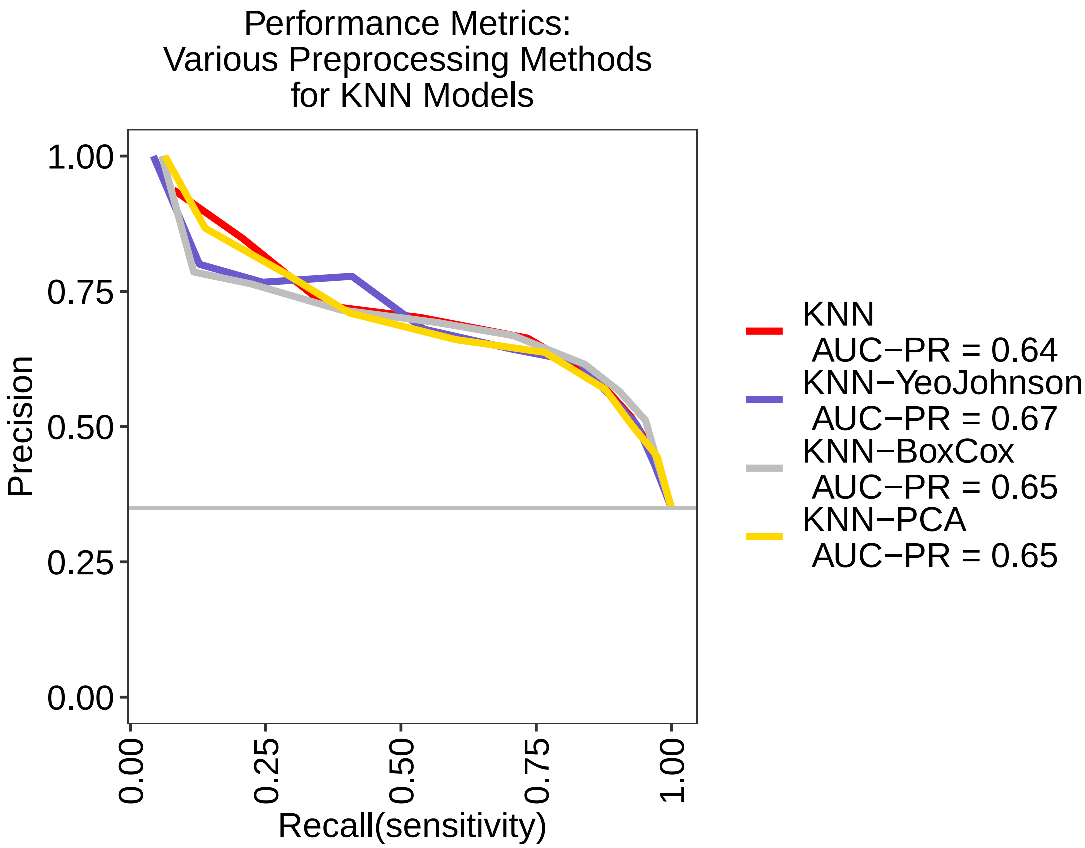


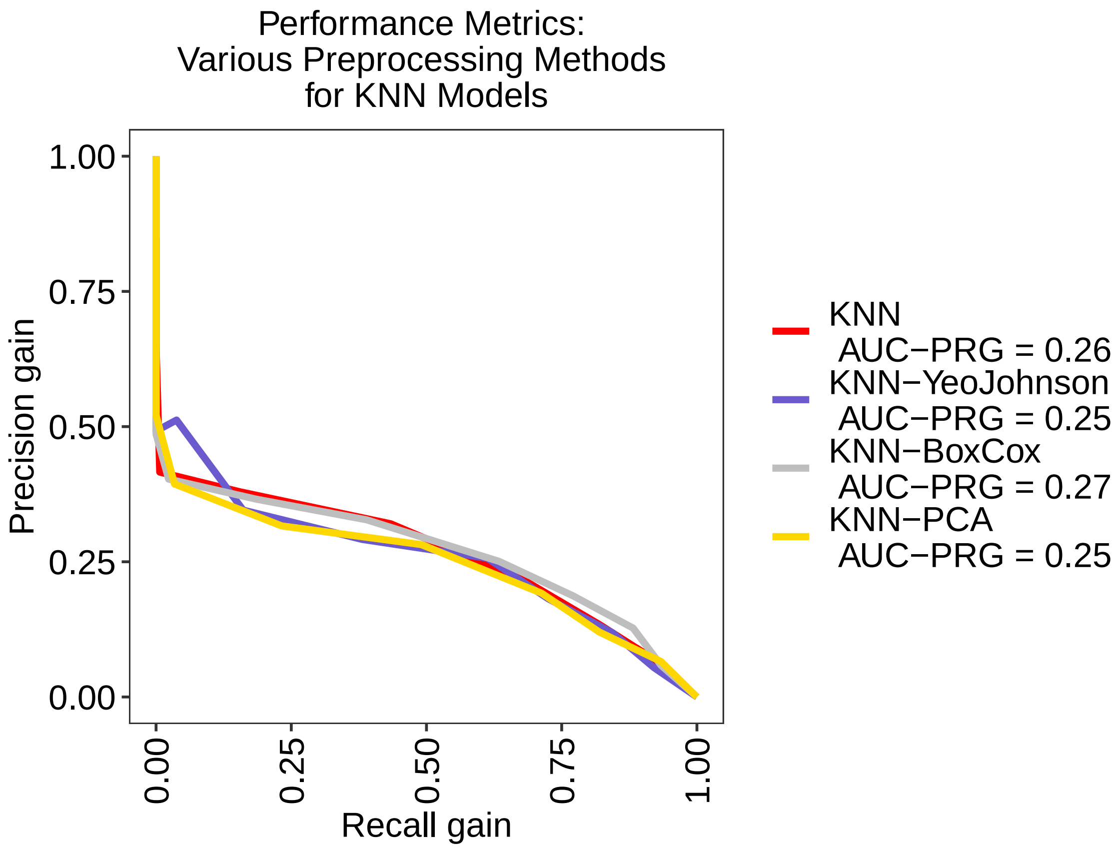

```r
#MLeval:evalm() is for machine learning model evaluation.
#The function can accept the Caret 'train' function results
#to evaluate machine learning predictions or a data frame
#of probabilities and ground truth labels can be passed in
#to evaluate

names2<- c("KNN","KNN-YeoJohnson","KNN-BoxCox","KNN-PCA")
res <- evalm(knnTrainComp, gnames = names2,title="Performance Metrics: \nVarious Preprocessing Methods \
```

```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Group 2 type: cv

## Group 3 type: cv

## Group 4 type: cv

## Observations: 2152
```

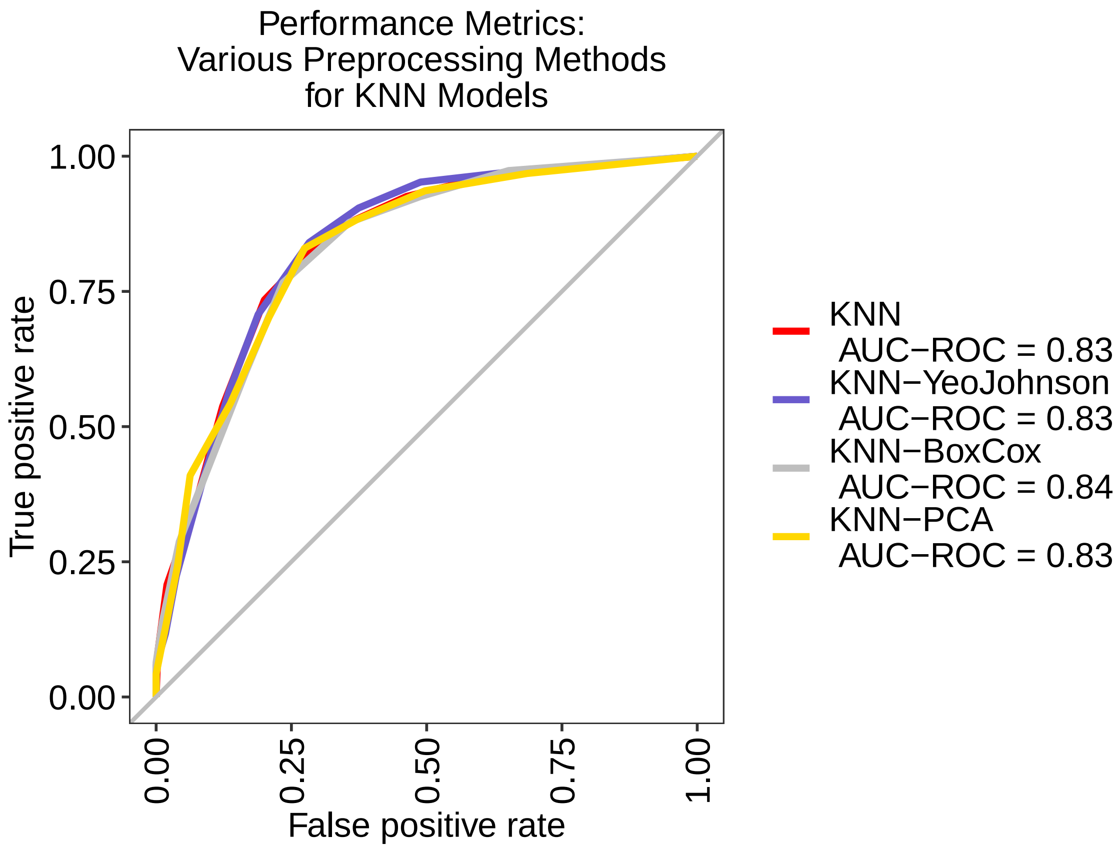
```
## Number of groups: 4

## Observations per group: 538

## Positive: pos

## Negative: neg

## Group: KNN

## Positive: 188

## Negative: 350

## Group: KNN-YeoJohnson

## Positive: 188

## Negative: 350

## Group: KNN-BoxCox

## Positive: 188

## Negative: 350

## Group: KNN-PCA

## Positive: 188

## Negative: 350

## ***Performance Metrics***
```


Performance Metrics:
Various Preprocessing Methods
for KNN Models
True probability in each bin
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted probability
KNN
KNN−YeoJohnson
KNN−BoxCox
KNN−PCA


Performance Metrics:
Various Preprocessing Methods
for KNN Models
1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
KNN
AUC−PR = 0.64
KNN−YeoJohnson
AUC−PR = 0.67
KNN−BoxCox
AUC−PR = 0.65
KNN−PCA
AUC−PR = 0.65




```
## KNN Optimal Informedness = 0.538844984802432

## KNN-YeoJohnson Optimal Informedness = 0.555501519756839

## KNN-BoxCox Optimal Informedness = 0.557568389057751

## KNN-PCA Optimal Informedness = 0.531671732522796

## KNN AUC-ROC = 0.83

## KNN-YeoJohnson AUC-ROC = 0.83

## KNN-BoxCox AUC-ROC = 0.84

## KNN-PCA AUC-ROC = 0.83
```



Linear Support Vector Machine (SVM)

```
#Tune Grid
#We tune our LSVM by having the expand.grid() take on
#different cost values and then choose the C with the
#highest ROC. Then we use train() to run through the
#training and test sets to build the LSVM, and use method = "svmLinear"
#for the linear kernel.

svmTG = expand.grid(C=c(seq(.5,5,by=.5)))

#Center and Scale
set.seed(2345)
svm_train_data <- train(x=trainx,y=trainy,
                        method = "svmLinear",
                        metric = "ROC",
                        tuneGrid = svmTG,
                        preProcess =
                        c("center","scale"),
                        tuneLength = 10,
                        trControl = ctrl )

svm_train_data
```

```
## Support Vector Machines with Linear Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   C    ROC        Sens       Spec
##   0.5  0.8498580  0.8857143  0.5909357
##   1.0  0.8481955  0.8857143  0.5804094
##   1.5  0.8482038  0.8857143  0.5909357
##   2.0  0.8483542  0.8828571  0.5909357
##   2.5  0.8483542  0.8914286  0.5804094
##   3.0  0.8480535  0.8857143  0.5909357
##   3.5  0.8482038  0.8885714  0.5804094
##   4.0  0.8482038  0.8828571  0.5909357
##   4.5  0.8480535  0.8914286  0.5856725
##   5.0  0.8479031  0.8914286  0.5909357
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.5.
```

```
#Yeo Johnson
set.seed(2345)
svm_train_dataYJ <- train(x=trainx,y=trainy,
                              method = "svmLinear",
                              metric = "ROC",
                              tuneGrid = svmTG,
                              preProcess =
                              c("center","scale","YeoJohnson"),
                              tuneLength = 10,
                              trControl = ctrl)

svm_train_dataYJ
```

```
## Support Vector Machines with Linear Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Yeo-Johnson transformation (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   C    ROC        Sens       Spec
##   0.5  0.8579699  0.8742857  0.6225146
##   1.0  0.8579783  0.8742857  0.6225146
##   1.5  0.8579950  0.8742857  0.6172515
```

```
##   2.0  0.8582957  0.8714286  0.6119883
##   2.5  0.8584461  0.8742857  0.6225146
##   3.0  0.8585965  0.8771429  0.6172515
##   3.5  0.8590476  0.8771429  0.6119883
##   4.0  0.8590476  0.8771429  0.6225146
##   4.5  0.8588972  0.8800000  0.6172515
##   5.0  0.8590476  0.8771429  0.6225146
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 3.5.
```

```
#Box Cox
set.seed(2345)
svm_train_dataBC <- train(x=trainx,y=trainy,
                              method = "svmLinear",
                              metric = "ROC",
                              tuneGrid = svmTG,
                              preProcess =
                              c("center","scale","BoxCox"),
                              tuneLength = 10,
                              trControl = ctrl)

svm_train_dataBC
```

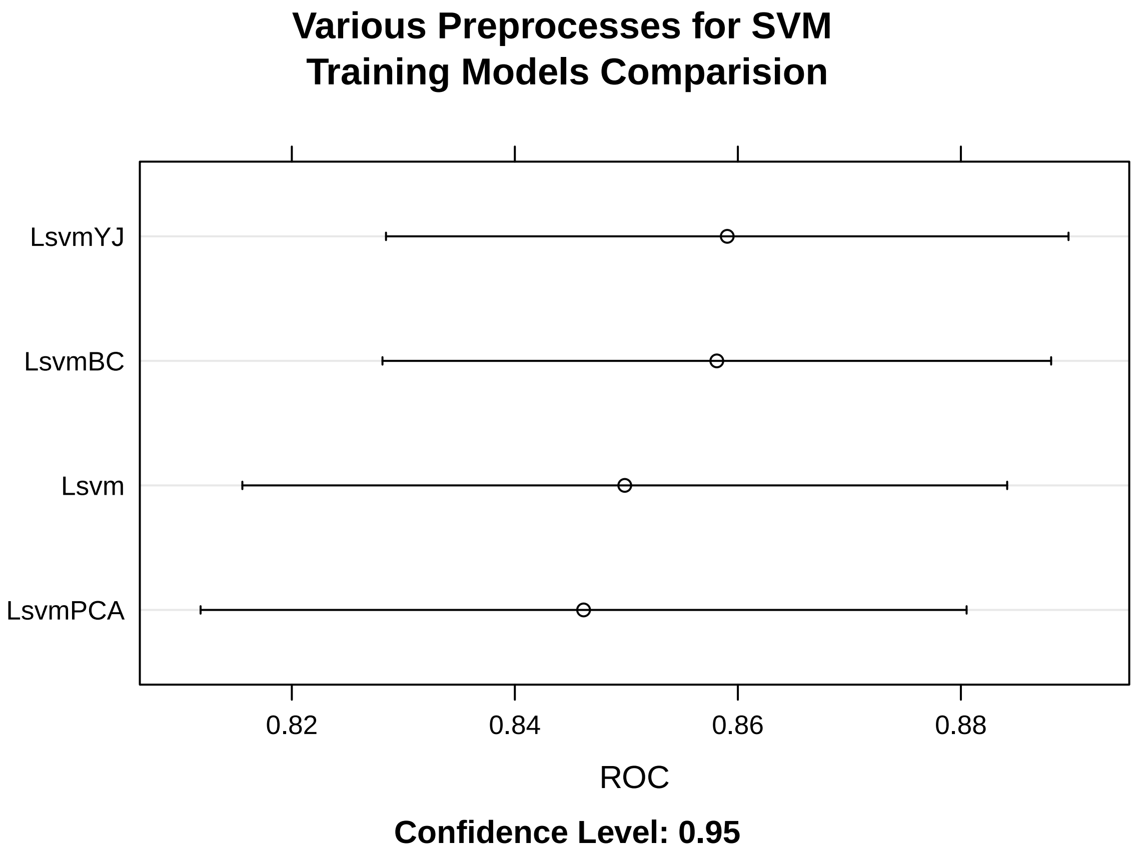
```
## Support Vector Machines with Linear Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Box-Cox transformation (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   C    ROC        Sens       Spec
##   0.5  0.8558480  0.8800000  0.6222222
##   1.0  0.8578112  0.8742857  0.6169591
##   1.5  0.8581119  0.8771429  0.6116959
##   2.0  0.8579616  0.8771429  0.6277778
##   2.5  0.8576608  0.8742857  0.6116959
##   3.0  0.8575104  0.8742857  0.6169591
##   3.5  0.8573601  0.8714286  0.6172515
##   4.0  0.8572097  0.8714286  0.6222222
##   4.5  0.8570593  0.8828571  0.6169591
##   5.0  0.8570593  0.8742857  0.6169591
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.5.
```

```
#PCA
set.seed(2345)
svm_train_dataPCA <- train(x=trainx,y=trainy,
```

```r
                              method = "svmLinear",
                              metric = "ROC",
                              tuneGrid = svmTG,
                              preProcess =
                              c("center","scale","pca"),
                              tuneLength = 10,
                              trControl = ctrl)

svm_train_dataPCA
```

```
## Support Vector Machines with Linear Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), principal component signal
##  extraction (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   C    ROC        Sens       Spec
##   0.5  0.8461654  0.8742857  0.5590643
##   1.0  0.8457143  0.8800000  0.5643275
##   1.5  0.8457143  0.8771429  0.5643275
##   2.0  0.8457143  0.8800000  0.5643275
##   2.5  0.8457143  0.8828571  0.5590643
##   3.0  0.8458647  0.8800000  0.5643275
##   3.5  0.8457143  0.8771429  0.5643275
##   4.0  0.8455639  0.8771429  0.5643275
##   4.5  0.8458647  0.8828571  0.5643275
##   5.0  0.8457143  0.8771429  0.5643275
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.5.
```

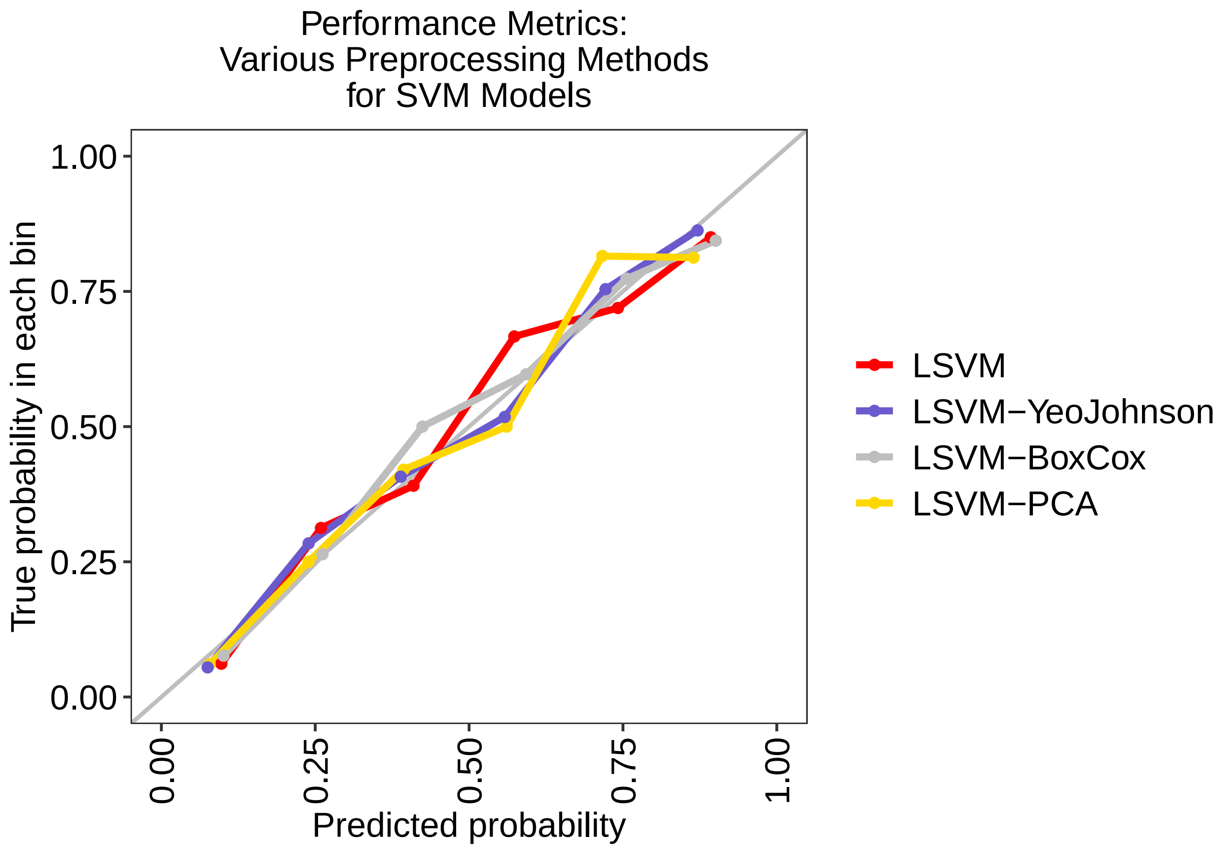
```r
#Comparision of different preprocesses on the knn training model
#(Yeo Johnson, Box Cox, PCA, and simple center and scaling).
svmTrainComp <- list(Lsvm = svm_train_data,
                     LsvmYJ = svm_train_dataYJ,
                     LsvmBC = svm_train_dataBC,
                     LsvmPCA = svm_train_dataPCA)

resamplesvm <- resamples(svmTrainComp)

dotplot(resamplesvm, metric="ROC",
        main="Various Preprocesses for SVM \nTraining Models Comparision")
```



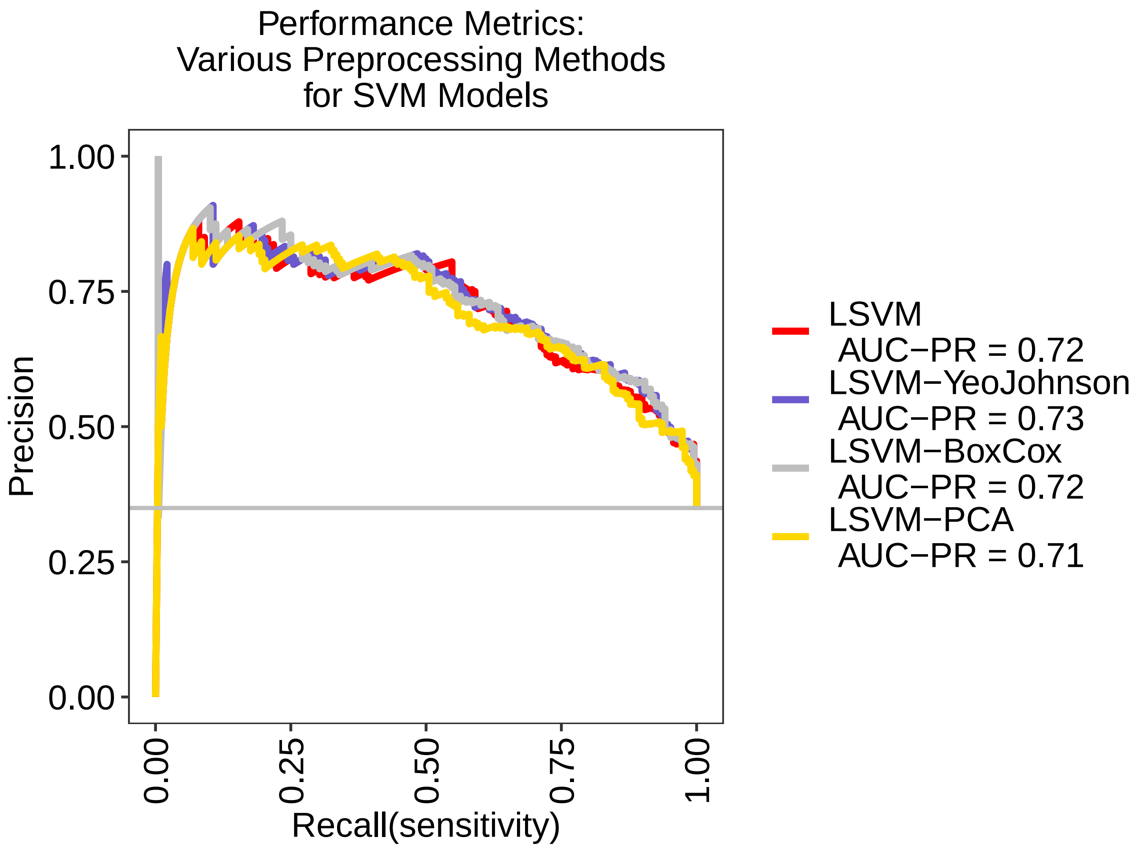

```
#MLeval:evalm() is for machine learning model evaluation.
#The function can accept the Caret 'train' function results
#to evaluate machine learning predictions or a data frame
#of probabilities and ground truth labels can be passed in
#to evaluate

names3<- c("LSVM","LSVM-YeoJohnson","LSVM-BoxCox","LSVM-PCA")
res <- evalm(svmTrainComp, gnames = names3,title="Performance Metrics: \nVarious Preprocessing Methods
```

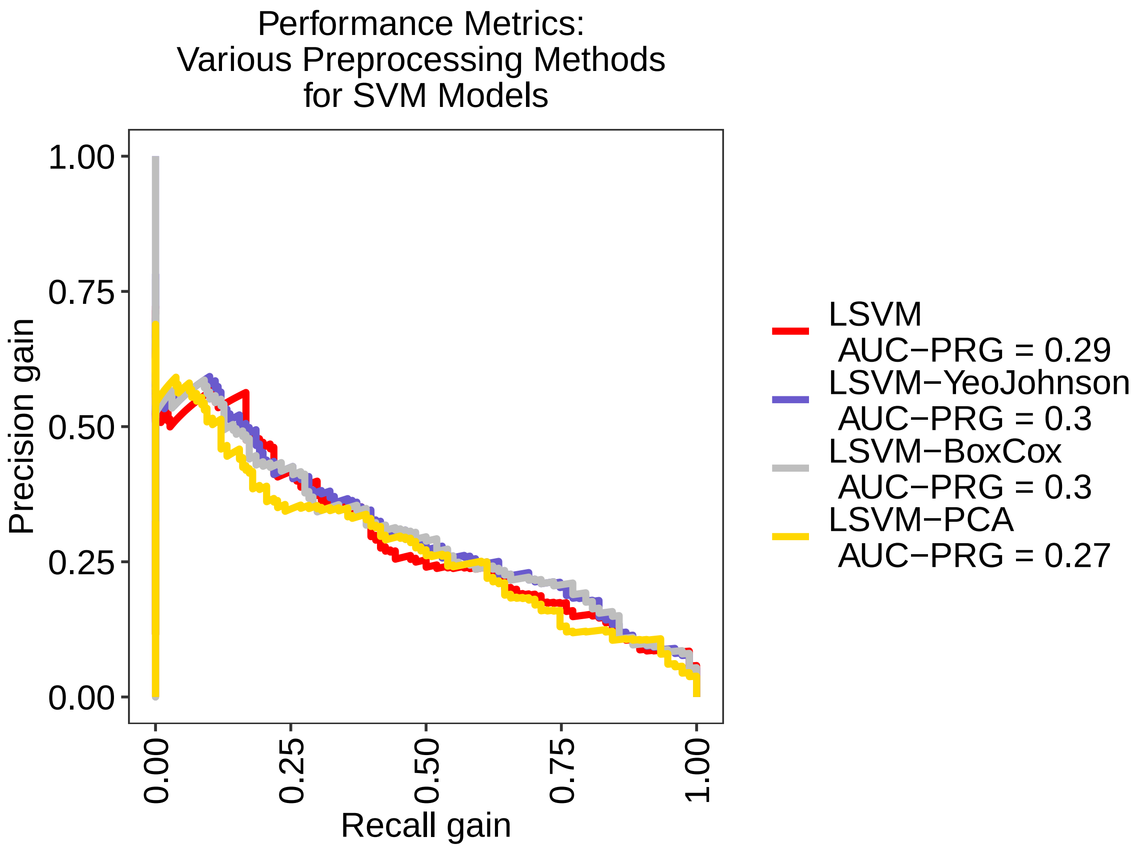
```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Group 2 type: cv

## Group 3 type: cv

## Group 4 type: cv

## Observations: 2152
```

```
## Number of groups: 4

## Observations per group: 538

## Positive: pos

## Negative: neg

## Group: LSVM

## Positive: 188

## Negative: 350

## Group: LSVM-YeoJohnson

## Positive: 188

## Negative: 350

## Group: LSVM-BoxCox

## Positive: 188

## Negative: 350

## Group: LSVM-PCA

## Positive: 188

## Negative: 350

## ***Performance Metrics***
```

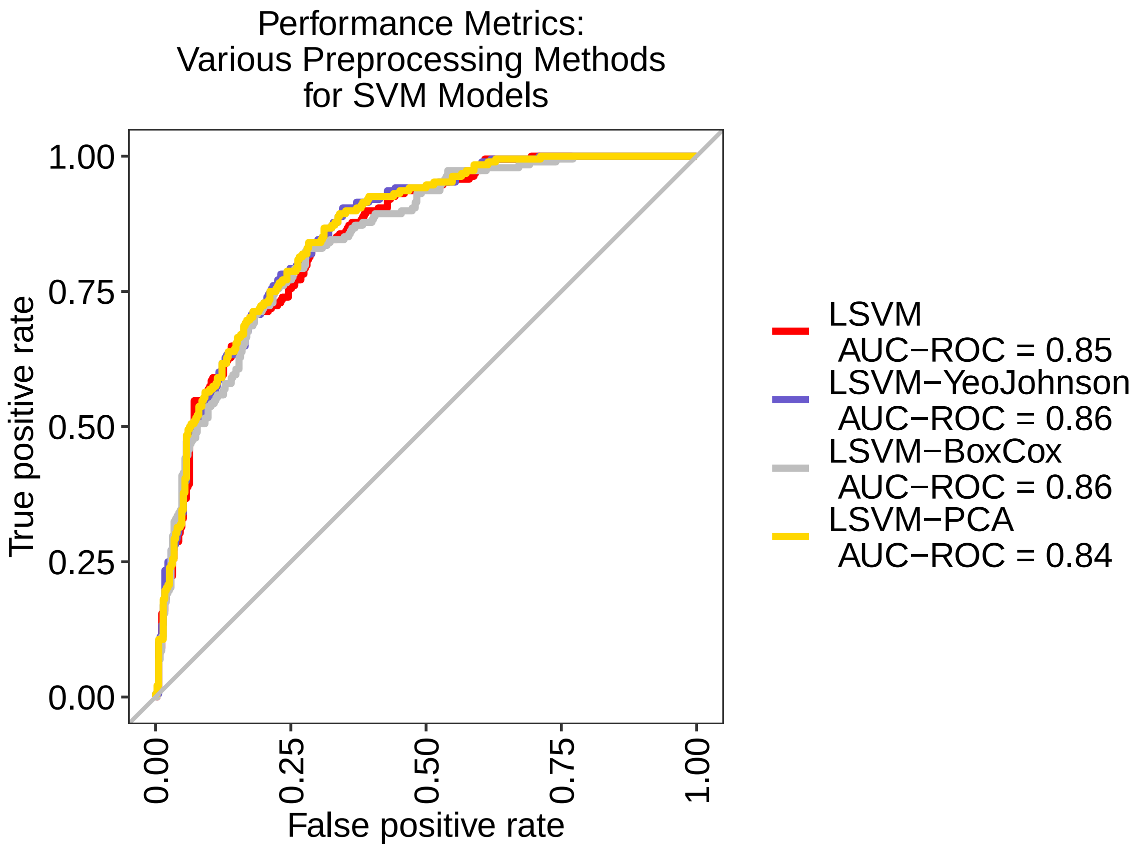


Performance Metrics:
Various Preprocessing Methods
for SVM Models
True probability in each bin
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted probability
LSVM
LSVM−YeoJohnson
LSVM−BoxCox
LSVM−PCA


Performance Metrics:
Various Preprocessing Methods
for SVM Models
1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
LSVM
AUC−PR = 0.72
LSVM−YeoJohnson
AUC−PR = 0.73
LSVM−BoxCox
AUC−PR = 0.72
LSVM−PCA
AUC−PR = 0.71




```
## LSVM Optimal Informedness = 0.546534954407295

## LSVM-YeoJohnson Optimal Informedness = 0.557568389057751

## LSVM-BoxCox Optimal Informedness = 0.558541033434651

## LSVM-PCA Optimal Informedness = 0.549787234042553

## LSVM AUC-ROC = 0.85

## LSVM-YeoJohnson AUC-ROC = 0.86

## LSVM-BoxCox AUC-ROC = 0.86

## LSVM-PCA AUC-ROC = 0.84
```



Radial Support Vector Machine (SVM)

```
#Tune Grid
#We tune our LSVM by having the expand.grid() take on
#different cost values and then choose the C with the
#highest ROC. Then we use train() to run through the
#training and test sets to build the LSVM, and use method = "svmLinear"
#for the linear kernel.

RsvmTG = expand.grid(sigma = c(2,3,4,5),
                    C = c(.2,.4,.6,.8))

#Center and Scale
set.seed(2345)
Rsvm_train_data <- train(x=trainx,y=trainy,
                              method = "svmRadial",
                              metric = "ROC",
                              tuneGrid = RsvmTG,
                              preProcess =
                              c("center","scale"),
                              tuneLength = 10,
                              trControl = ctrl )

Rsvm_train_data
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   sigma  C    ROC        Sens       Spec
##   2      0.2  0.7730242  0.8371429  0.42631579
##   2      0.4  0.7730242  0.8342857  0.41812865
##   2      0.6  0.7730242  0.8457143  0.39824561
##   2      0.8  0.7730242  0.8400000  0.38742690
##   3      0.2  0.7628154  0.8742857  0.25818713
##   3      0.4  0.7628154  0.8942857  0.22777778
##   3      0.6  0.7629657  0.8942857  0.22280702
##   3      0.8  0.7629657  0.8971429  0.23859649
##   4      0.2  0.7573768  0.9457143  0.11140351
##   4      0.4  0.7573768  0.9171429  0.17923977
##   4      0.6  0.7573768  0.9571429  0.11111111
##   4      0.8  0.7573768  0.9514286  0.12192982
##   5      0.2  0.7513450  0.9600000  0.08479532
##   5      0.4  0.7513450  0.9714286  0.03187135
##   5      0.6  0.7513450  0.9657143  0.06929825
##   5      0.8  0.7514202  0.9685714  0.09005848
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 2 and C = 0.2.
```

```r
#Yeo Johnson
set.seed(2345)
Rsvm_train_dataYJ <- train(x=trainx,y=trainy,
                              method = "svmRadial",
                              metric = "ROC",
                              tuneGrid = RsvmTG,
                              preProcess =
                              c("center","scale","YeoJohnson"),
                              tuneLength = 10,
                              trControl = ctrl)

Rsvm_train_dataYJ
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Yeo-Johnson transformation (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
```

```
## Resampling results across tuning parameters:
##
##   sigma  C    ROC        Sens       Spec
##   2      0.2  0.7564327  0.9028571  0.30760234
##   2      0.4  0.7564327  0.9057143  0.30730994
##   2      0.6  0.7565831  0.9114286  0.28654971
##   2      0.8  0.7565831  0.9142857  0.27602339
##   3      0.2  0.7387636  0.9371429  0.12251462
##   3      0.4  0.7387636  0.9600000  0.13304094
##   3      0.6  0.7387636  0.9400000  0.15380117
##   3      0.8  0.7387636  0.9457143  0.11695906
##   4      0.2  0.7188722  0.9628571  0.05263158
##   4      0.4  0.7189474  0.9800000  0.05263158
##   4      0.6  0.7189474  0.9771429  0.04736842
##   4      0.8  0.7187970  0.9685714  0.05263158
##   5      0.2  0.6456391  0.9771429  0.04269006
##   5      0.4  0.7113534  0.9914286  0.00000000
##   5      0.6  0.7113534  0.9857143  0.01608187
##   5      0.8  0.7113534  0.9828571  0.02631579
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 2 and C = 0.6.
```

```r
#Box Cox
set.seed(2345)
Rsvm_train_dataBC <- train(x=trainx,y=trainy,
                           method = "svmRadial",
                           metric = "ROC",
                           tuneGrid = RsvmTG,
                           preProcess =
                           c("center","scale","BoxCox"),
                           tuneLength = 10,
                           trControl = ctrl)

Rsvm_train_dataBC
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), Box-Cox transformation (7)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   sigma  C    ROC        Sens       Spec
##   2      0.2  0.7430911  0.9085714  0.280994152
##   2      0.4  0.7429323  0.9114286  0.286257310
##   2      0.6  0.7430911  0.9114286  0.254678363
##   2      0.8  0.7430911  0.9257143  0.265204678
##   3      0.2  0.7186633  0.9542857  0.111988304
##   3      0.4  0.7188137  0.9600000  0.105847953
```

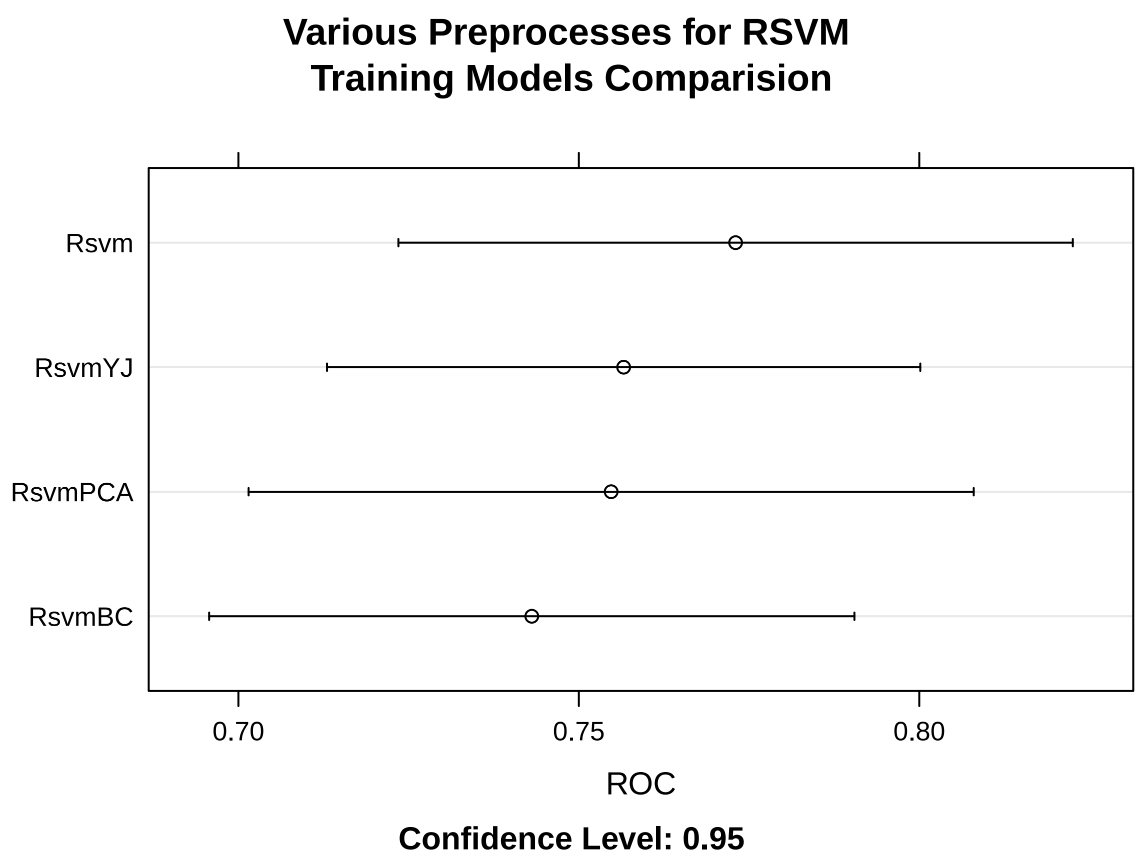
```
##   3      0.6  0.7188137  0.9542857  0.095321637
##   3      0.8  0.7188137  0.9657143  0.079532164
##   4      0.2  0.7054637  0.9657143  0.021052632
##   4      0.4  0.7054637  0.9800000  0.036842105
##   4      0.6  0.7053133  0.9800000  0.005263158
##   4      0.8  0.7054637  0.9828571  0.021052632
##   5      0.2  0.6304177  0.9828571  0.010818713
##   5      0.4  0.6592899  0.9914286  0.000000000
##   5      0.6  0.6982373  0.9857143  0.000000000
##   5      0.8  0.6982373  0.9857143  0.005263158
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 2 and C = 0.2.
```

```r
#PCA
set.seed(2345)
Rsvm_train_dataPCA <- train(x=trainx,y=trainy,
                              method = "svmRadial",
                              metric = "ROC",
                              tuneGrid = RsvmTG,
                              preProcess =
                              c("center","scale","pca"),
                              tuneLength = 10,
                              trControl = ctrl)

Rsvm_train_dataPCA
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8), principal component signal
##  extraction (8)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 484, 484, 484, 484, 484, 485, ...
## Resampling results across tuning parameters:
##
##   sigma  C    ROC        Sens       Spec
##   2      0.2  0.7547452  0.8057143  0.44122807
##   2      0.4  0.7547452  0.8228571  0.45029240
##   2      0.6  0.7547452  0.8571429  0.38274854
##   2      0.8  0.7547452  0.8571429  0.39766082
##   3      0.2  0.7442189  0.9142857  0.22309942
##   3      0.4  0.7442189  0.9000000  0.23888889
##   3      0.6  0.7442189  0.9114286  0.22777778
##   3      0.8  0.7442189  0.9085714  0.23830409
##   4      0.2  0.7435756  0.9514286  0.10614035
##   4      0.4  0.7435756  0.9285714  0.15350877
##   4      0.6  0.7435756  0.9428571  0.12251462
##   4      0.8  0.7435756  0.9428571  0.12719298
##   5      0.2  0.7356976  0.9542857  0.07456140
##   5      0.4  0.7358480  0.9600000  0.04239766
```

```
##   5      0.6  0.7358480  0.9514286  0.05789474
##   5      0.8  0.7359983  0.9342857  0.13771930
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 2 and C = 0.2.
```

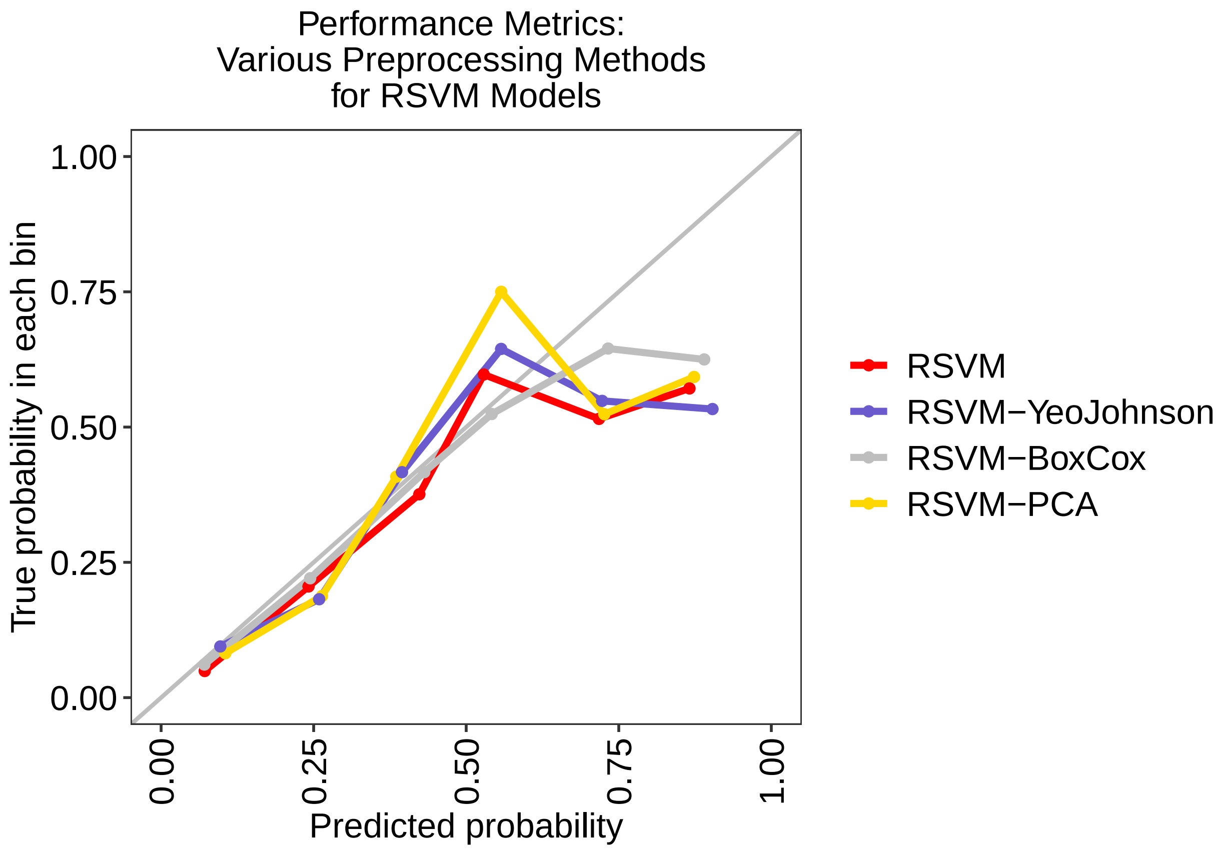
```r
#Comparision of different preprocesses on the Radial SVM training model
#(Yeo Johnson, Box Cox, PCA, and simple center and scaling).
RsvmTrainComp <- list(Rsvm = Rsvm_train_data,
                      RsvmYJ = Rsvm_train_dataYJ,
                      RsvmBC = Rsvm_train_dataBC,
                      RsvmPCA = Rsvm_train_dataPCA)

resampleRsvm <- resamples(RsvmTrainComp)

dotplot(resampleRsvm, metric="ROC",
        main="Various Preprocesses for RSVM \nTraining Models Comparision")
```

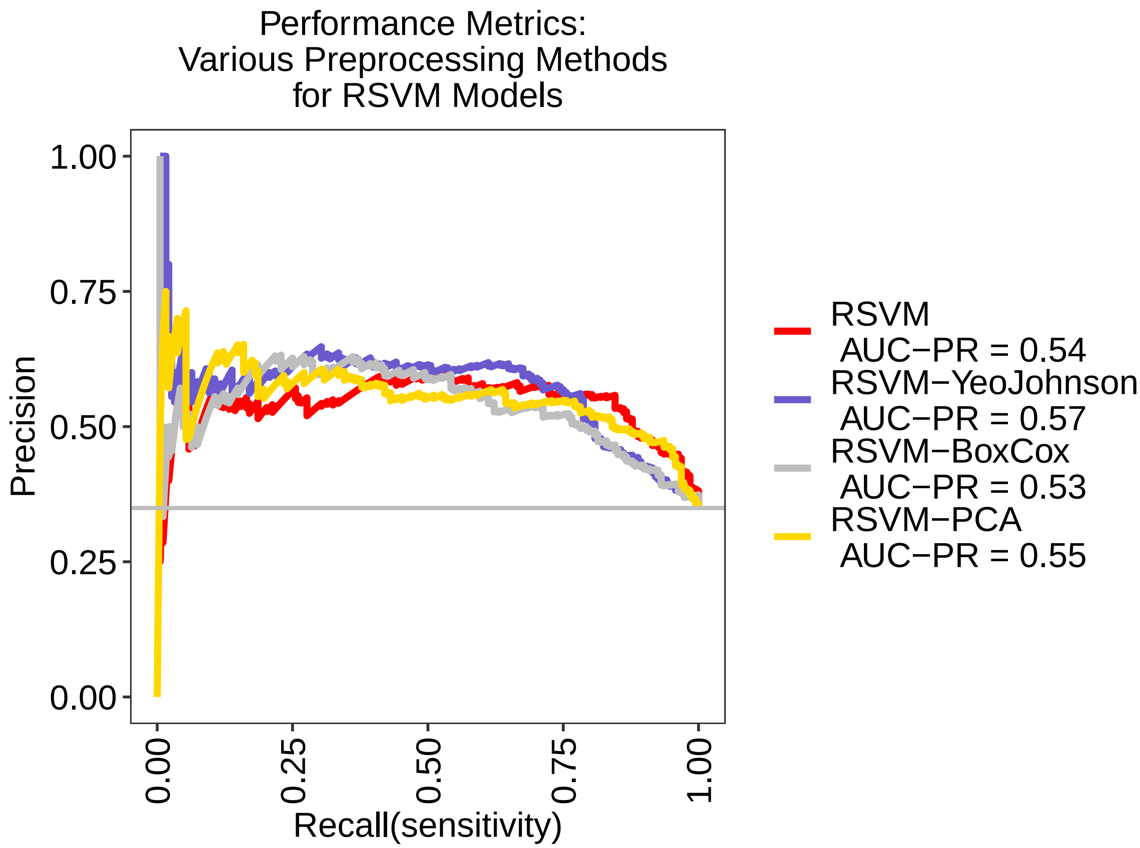



```r
#MLeval:evalm() is for machine learning model evaluation.
#The function can accept the Caret 'train' function results
#to evaluate machine learning predictions or a data frame
#of probabilities and ground truth labels can be passed in
#to evaluate

names4<- c("RSVM","RSVM-YeoJohnson","RSVM-BoxCox","RSVM-PCA")
res <- evalm(RsvmTrainComp, gnames = names4,title="Performance Metrics: \nVarious Preprocessing Methods
```

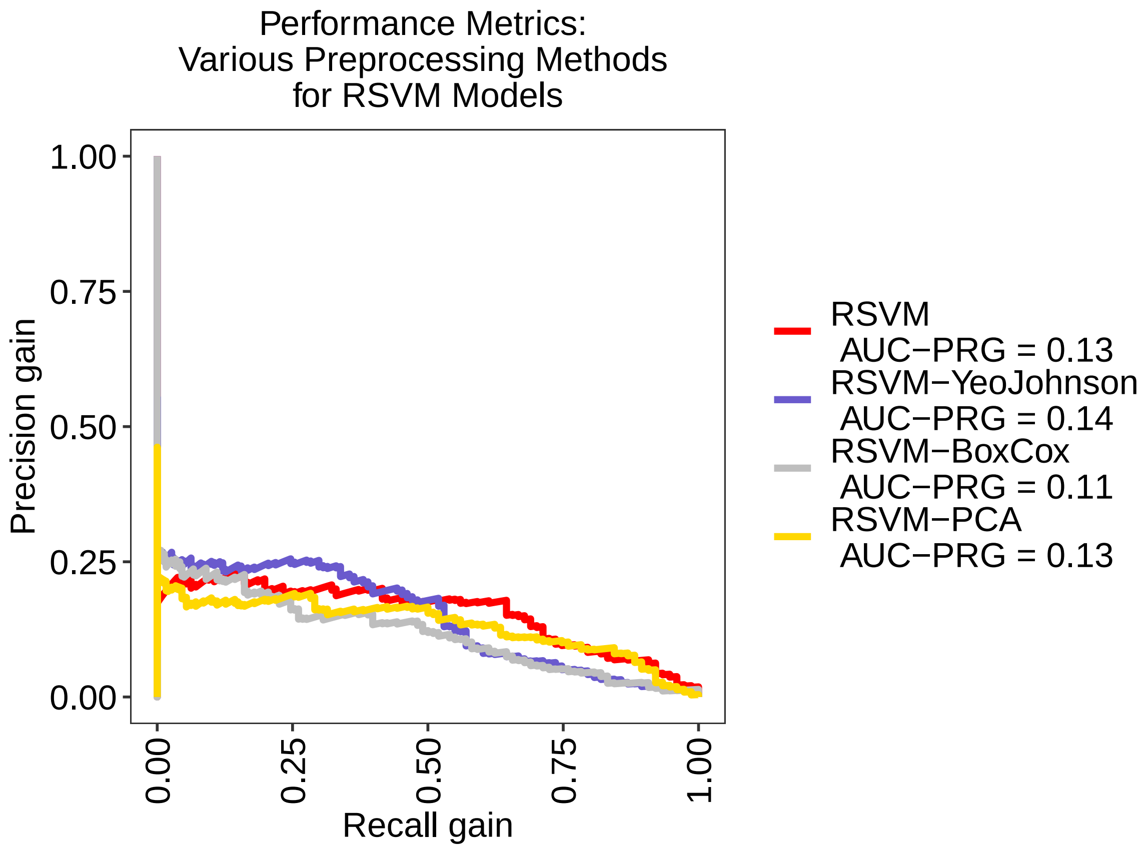
```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Group 2 type: cv

## Group 3 type: cv

## Group 4 type: cv

## Observations: 2152

## Number of groups: 4

## Observations per group: 538

## Positive: pos

## Negative: neg

## Group: RSVM

## Positive: 188

## Negative: 350

## Group: RSVM-YeoJohnson

## Positive: 188

## Negative: 350

## Group: RSVM-BoxCox

## Positive: 188

## Negative: 350

## Group: RSVM-PCA

## Positive: 188

## Negative: 350
```

```
## ***Performance Metrics***
```





Performance Metrics:
Various Preprocessing Methods
for RSVM Models
1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
RSVM
AUC−PR = 0.54
RSVM−YeoJohnson
AUC−PR = 0.57
RSVM−BoxCox
AUC−PR = 0.53
RSVM−PCA
AUC−PR = 0.55




```
## RSVM Optimal Informedness = 0.485744680851064
```

```
## RSVM-YeoJohnson Optimal Informedness = 0.453343465045593
```

```
## RSVM-BoxCox Optimal Informedness = 0.389209726443769
```

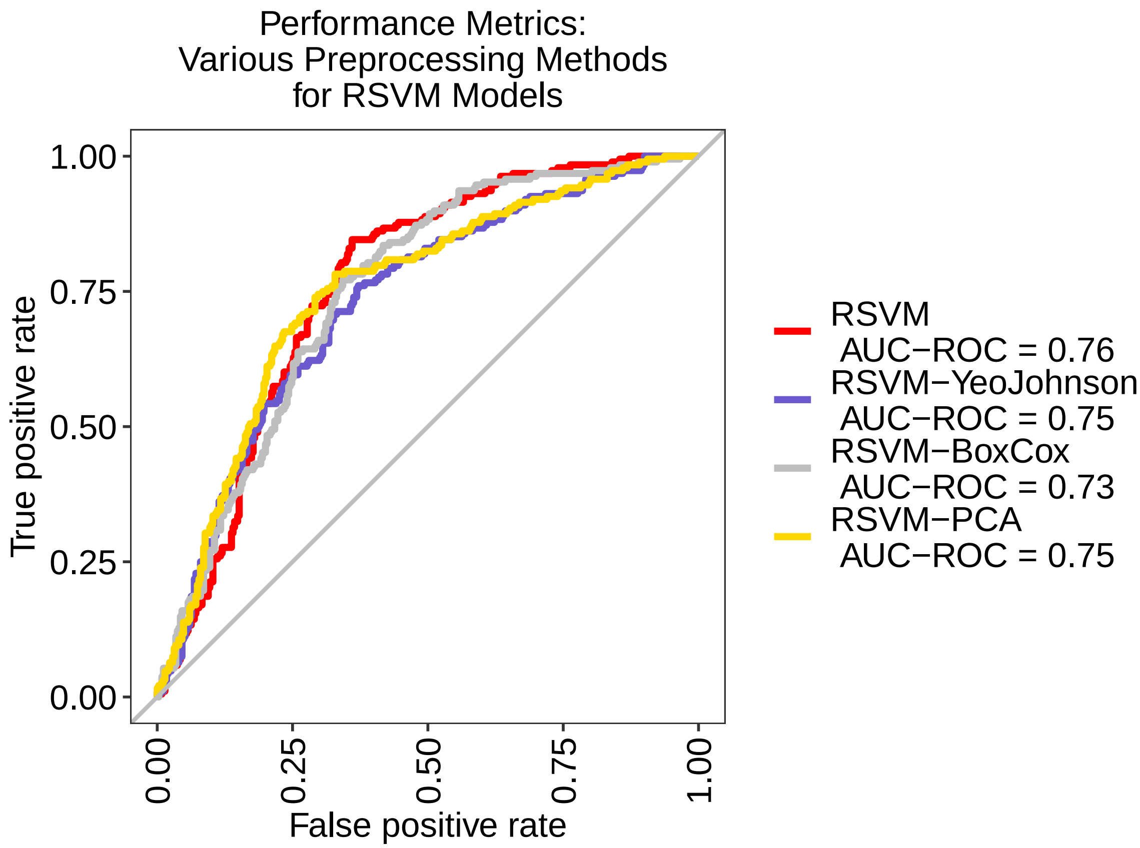
```
## RSVM-PCA Optimal Informedness = 0.428419452887538
```

```
## RSVM AUC-ROC = 0.76
```

```
## RSVM-YeoJohnson AUC-ROC = 0.75
```

```
## RSVM-BoxCox AUC-ROC = 0.73
```

```
## RSVM-PCA AUC-ROC = 0.75
```



TEST MODELS

Logistic Regression Predictions

```
# #Centered and Scaled Logistic Regression
# lrpredict = predict(lr_train_data,testx)
# #Confusion Matrix
# lrcm = confusionMatrix( data=lrpredict, reference=testy,positive = "pos")
# lrcm
# #Prediction Probabilities
# lrprob <- predict(lr_train_data,testx,type="prob")
# #ROC
# lrROC <- roc(testy,lrprob$pos)
# lrROC
#
# plot(lrROC, type = "s", col = rgb(.2, .2, .2, .2), add = TRUE, legacy.axes = TRUE)
# plot(lrROC, col = 1, lty = 2, main = "ROC")
# plot(roc2, col = 4, lty = 3, add = TRUE)
# # AUC - Area under the curve
# colAUC(lrprob$pos, testy, plotROC = T)
#
#
# ## Get the confusion matrices for the hold-out set
```

```
# lrCM <- confusionMatrix(lrFit, norm = "none")
# lrCM
#
# ## Get the area under the ROC curve for the hold-out set
# lrRoc <- roc(response = lr_train_data$pred$obs,
#               predictor = lr_train_data$pred$successful,
#               levels = rev(levels(lr_train_data$pred$obs)))
# plot(lrRoc, legacy.axes = TRUE)
# lrImp <- varImp(lr_train_data, scale = FALSE)
# lrImp
#
# plot(lrRoc, type = "s", col = rgb(.2, .2, .2, .2), legacy.axes = TRUE)
# plot(ldaRoc, add = TRUE, type = "s", legacy.axes = TRUE)
#
#
# #Yeo Johnson
# lrpredictYJ = predict( lr_train_dataYJ,testx)
# lrcmYJ = confusionMatrix( data=lrpredictYJ, reference=testy,positive = "pos" )
# lrcmYJ
#
# #Box Cox
# lrpredictBC = predict( lr_train_dataBC,testx)
# lrcmBC = confusionMatrix( data=lrpredictBC, reference=testy,positive = "pos" )
# lrcmBC
#
# #PCA
# lrpredictPCA = predict( lr_train_dataPCA,testx)
# lrcmPCA = confusionMatrix( data=lrpredictPCA, reference=testy,positive = "pos" )
# lrcmPCA
#
#
# #Comparision of different preprocesses on the logistic regression test model(Yeo Johnson, Box Cox, PC
# lrTestComp <- list(LogisticRegression = lrpredict, LogisticRegressionYJ = lrpredictYJ, LogisticRegres
# resampleLogisticRegressionTest <- resamples(lrTestComp)
#
# dotplot(resampleLogisticRegressionTest, metric="ROC",main="Different Preprocesses for Logistic Regres
#
#
# result_rf <- c(cm_rf$byClass['Sensitivity'], cm_rf$byClass['Specificity'], cm_rf$byClass['Precision']
#                 cm_rf$byClass['Recall'], cm_rf$byClass['F1'], roc_rf$auc)
#
# result_xgb <- c(cm_xgb$byClass['Sensitivity'], cm_xgb$byClass['Specificity'], cm_xgb$byClass['Precisi
#                 cm_xgb$byClass['Recall'], cm_xgb$byClass['F1'], roc_xgb$auc)
#
# result_knn <- c(cm_knn$byClass['Sensitivity'], cm_knn$byClass['Specificity'], cm_knn$byClass['Precisi
#                 cm_knn$byClass['Recall'], cm_knn$byClass['F1'], roc_knn$auc)
#
# result_glm <- c(cm_glm$byClass['Sensitivity'], cm_glm$byClass['Specificity'], cm_glm$byClass['Precisi
#                 cm_glm$byClass['Recall'], cm_glm$byClass['F1'], roc_glm$auc)
#
# result_rpart <- c(cm_rpart$byClass['Sensitivity'], cm_rpart$byClass['Specificity'], cm_rpart$byClass[
#                 cm_rpart$byClass['Recall'], cm_rpart$byClass['F1'], roc_rpart$auc)
#
```

```r
#
# all_results <- data.frame(rbind(result_rf, result_xgb, result_knn, result_glm, result_rpart))
# names(all_results) <- c("Sensitivity", "Specificity", "Precision", "Recall", "F1", "AUC")
# all_results
```