Midterm Report

WalterHennings Partner: Ben Schmitz

The following linear models and classification models, respectively, were generated for MATH341 midterm project:

The following classification models were also generated:

In [55]:
```
cbind(c("Full MLR"),c("Subset Selection"),c("MLR Ridge"),c("MLR Lasso"),c("kNN Re
cbind(c("Logistic Regression"),c("kNN Classifier"),c("LDA"),c("QDA"),c("Naive Bay
```

Full MLR	Subset Selection	MLR Ridge	MLR Lasso	kNN Regression

Logistic Regression	kNN Classifier	LDA	QDA	Naive Bayes

Loading in Libraries

In [2]:
```
library(readr)
library(glmnet)
library(ggplot2)
library(boot)
library(caret)
library(leaps)
library(class)
library(FNN)
library(Metrics)
library(pROC)
library(tidyverse)
library(e1071)
library(naivebayes)
library(MASS)
options(warn=-1) #To Suppress Warnings
```

Loading Train and Test Data

In [3]:
```
setwd("C:/Users/whwal/Desktop/Machine Learning/Midterm Project")
train <- read_csv("StudentDataTrain.csv")
test <- read_csv("StudentDataTest.csv")
train = na.omit(train)
test = na.omit(test)
train = train[,-c(1,2)]
test = test[,-c(1,2)]
```

```
Parsed with column specification:
cols(
  Race_Ethc_Visa = col_character(),
  Gender = col_character(),
  HSGPA = col_double(),
  SAT_Total = col_double(),
  Entry_Term = col_double(),
  Term.GPA = col_double(),
  Persistence.NextYear = col_double(),
  N.RegisteredCourse = col_double(),
  N.Ws = col_double(),
  N.DFs = col_double(),
  N.As = col_double(),
  N.PassedCourse = col_double(),
  N.CourseTaken = col_double(),
  Perc.PassedEnrolledCourse = col_double(),
  Perc.Pass = col_double(),
  Perc.Withd = col_double(),
  N.GraduateCourse = col_double(),
  FullTimeStudent = col_double()
)
Parsed with column specification:
cols(
  Race_Ethc_Visa = col_character(),
  Gender = col_character(),
  HSGPA = col_double(),
  SAT_Total = col_double(),
  Entry_Term = col_double(),
  Term.GPA = col_double(),
  Persistence.NextYear = col_double(),
  N.RegisteredCourse = col_double(),
  N.Ws = col_double(),
  N.DFs = col_double(),
  N.As = col_double(),
  N.PassedCourse = col_double(),
  N.CourseTaken = col_double(),
  Perc.PassedEnrolledCourse = col_double(),
  Perc.Pass = col_double(),
  Perc.Withd = col_double(),
  N.GraduateCourse = col_double(),
  FullTimeStudent = col_double()
)
```

Data Exploration/Cleaning

In [4]:
```
cor(train)
```

	HSGPA	SAT_Total	Entry_Term	Term.GPA	Persistence.Nex
HSGPA	1.000000000	-0.003758312	-0.103645005	0.060280036	0.17456
SAT_Total	-0.003758312	1.000000000	-0.038816754	-0.022627828	-0.00760
Entry_Term	-0.103645005	-0.038816754	1.000000000	0.004810567	-0.11131
Term.GPA	0.060280036	-0.022627828	0.004810567	1.000000000	0.47754
Persistence.NextYear	0.174561339	-0.007603759	-0.111316139	0.477542392	1.00000
N.RegisteredCourse	0.017466136	-0.001550347	-0.001491140	-0.001990355	0.00398
N.Ws	0.015323271	0.020477721	-0.184489208	-0.010872277	0.02073
N.DFs	-0.028319672	-0.024496385	0.226961904	0.007662234	-0.07061
N.As	0.005033852	-0.013697361	-0.120398872	0.019541574	0.07820
N.PassedCourse	0.030230332	0.003755455	-0.055260637	-0.001629591	0.04014
N.CourseTaken	0.011234555	-0.011955572	0.090962118	0.003299299	-0.00608
Perc.PassedEnrolledCourse	0.017840237	0.004104826	-0.083494649	0.002810394	0.05231
Perc.Pass	0.030307725	0.022810338	-0.226454081	-0.001818066	0.06583
Perc.Withd	0.007988513	0.025873840	-0.188673739	-0.005920765	0.01123
N.GraduateCourse	0.016572219	-0.020778679	0.003716367	0.010332220	-0.00329
FullTimeStudent	0.003711931	-0.002089033	0.077900207	0.014200012	-0.00056

In this step, we identify any variables that have significant (>0.4) correlation with each other, in order to reduce redundancies.

In [5]:
```
train = train[,-c(6,7,8,10,11,12)]
test = test[,-c(6,7,8,10,11,12)]
cor(train)
colnames(train) #Left with 10 predictors
```

	HSGPA	SAT_Total	Entry_Term	Term.GPA	Persistence.NextYear
HSGPA	1.000000000	-0.003758312	-0.103645005	0.060280036	0.1745613391
SAT_Total	-0.003758312	1.000000000	-0.038816754	-0.022627828	-0.0076037587
Entry_Term	-0.103645005	-0.038816754	1.000000000	0.004810567	-0.1113161389
Term.GPA	0.060280036	-0.022627828	0.004810567	1.000000000	0.4775423915
Persistence.NextYear	0.174561339	-0.007603759	-0.111316139	0.477542392	1.0000000000
N.As	0.005033852	-0.013697361	-0.120398872	0.019541574	0.0782088497
Perc.Pass	0.030307725	0.022810338	-0.226454081	-0.001818066	0.0658342385
Perc.Withd	0.007988513	0.025873840	-0.188673739	-0.005920765	0.0112324292
N.GraduateCourse	0.016572219	-0.020778679	0.003716367	0.010332220	-0.0032990788
FullTimeStudent	0.003711931	-0.002089033	0.077900207	0.014200012	-0.0005602344

'HSGPA' 'SAT_Total' 'Entry_Term' 'Term.GPA' 'Persistence.NextYear' 'N.As' 'Perc.Pass' 'Perc.Withd' 'N.GraduateCourse' 'FullTimeStudent'

Regression Models

Multi-Linear Regression

We perform Multi-Linear Regression with the train() function from the "caret" package. With it we are able to easily utilize 5-fold Cross Validation.

In [6]:
```
train.control = trainControl(method = "cv", number = 5)
lm.model.fit = train(Term.GPA~., data = test, trControl = train.control, method =
lm.predict.train = predict(lm.model.fit, train)
lm.predict.test = predict(lm.model.fit, test)
lm.train.MSE = mean((lm.predict.train - train$Term.GPA)^2)
lm.test.PMSE = mean((lm.predict.test - test$Term.GPA)^2)
RSS.lm = sum((lm.predict.test - test$Term.GPA)^2)
TSS.lm = sum((test$Term.GPA - mean(test$Term.GPA))^2)
lm.Rsqr = 1 - (RSS.lm/TSS.lm)
```

Ridge Regression

We fit a ridge regression model to the training data. Here, we choose an optimal value of lambda through cross validation. By default, glmnet() performs 10-fold CV, but we change it to 5 using the nfolds argument.

In [7]:
```
set.seed(1)
train.mat = model.matrix(Term.GPA ~ ., data = train)
test.mat = model.matrix(Term.GPA ~ ., data = test)

grid = 10 ^ seq(4, -2, length = 100)
ridge.model = glmnet(train.mat, train$Term.GPA, alpha = 0, lambda = grid, thresh
cv.ridge = cv.glmnet(train.mat, train$Term.GPA, alpha = 0, lambda = grid, thresh
bestlambda.ridge = cv.ridge$lambda.min
bestlambda.ridge #This value is 0.01
```

0.01

Now that we have acquired an optimal lambda, we can now test the model through train and test predictions:

In [8]:
```
ridge.predict.train = predict(ridge.model, s = bestlambda.ridge, newx = train.mat
ridge.predict.test = predict(ridge.model, s = bestlambda.ridge, newx = test.mat)
ridge.train.MSE = mean((ridge.predict.train - train$Term.GPA)^2)
ridge.test.PMSE = mean((ridge.predict.test - test$Term.GPA)^2)
predict(ridge.model, s = bestlambda.ridge, type = "coefficients")
RSS.ridge = sum((ridge.predict.test - test$Term.GPA)^2)
TSS.ridge = sum((test$Term.GPA - mean(test$Term.GPA))^2)
ridge.Rsqr = 1 - (RSS.ridge/TSS.ridge)
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                                 1
(Intercept)         -1.502994e+01
(Intercept)          .
HSGPA               -1.504636e-03
SAT_Total           -8.326111e-05
Entry_Term           7.718919e-03
Persistence.NextYear  1.219804e+00
N.As                -1.237038e-02
Perc.Pass           -5.504550e-02
Perc.Withd          -1.822924e-03
N.GraduateCourse     1.362237e-02
FullTimeStudent      2.387999e-02
```

Lasso Regression

We fit a lasso regression model to the training data. Like in ridge regression, we use 5-fold CV to obtain a lambda that will minimize MSE.

In [9]:
```
lasso.model = glmnet(train.mat, train$Term.GPA, alpha = 1, lambda = grid, thresh
cv.lasso = cv.glmnet(train.mat, train$Term.GPA, alpha = 1, lambda = grid, thresh
bestlambda.lasso = cv.lasso$lambda.min
bestlambda.lasso

lasso.predict.train = predict(lasso.model, s = bestlambda.lasso, newx = train.mat
lasso.predict.test = predict(lasso.model, s = bestlambda.lasso, newx = test.mat)
lasso.train.MSE = mean((lasso.predict.train - train$Term.GPA)^2)
lasso.test.PMSE = mean((lasso.predict.test - test$Term.GPA)^2)
predict(lasso.model, s = bestlambda.lasso, type = "coefficients")
RSS.lasso = sum((lasso.predict.test - test$Term.GPA)^2)
TSS.lasso = sum((test$Term.GPA - mean(test$Term.GPA))^2)
lasso.Rsqr = 1 - (RSS.lasso/TSS.lasso)
```

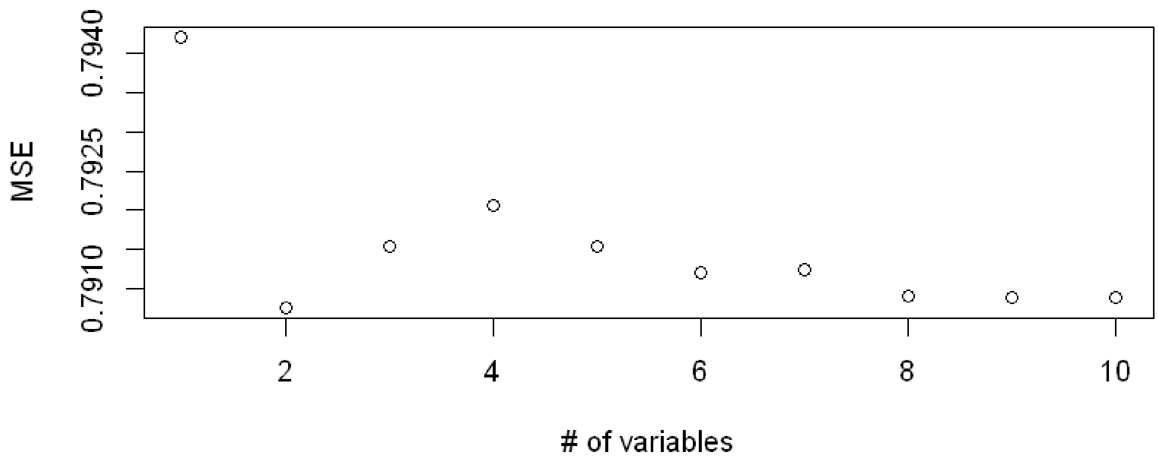
0.01

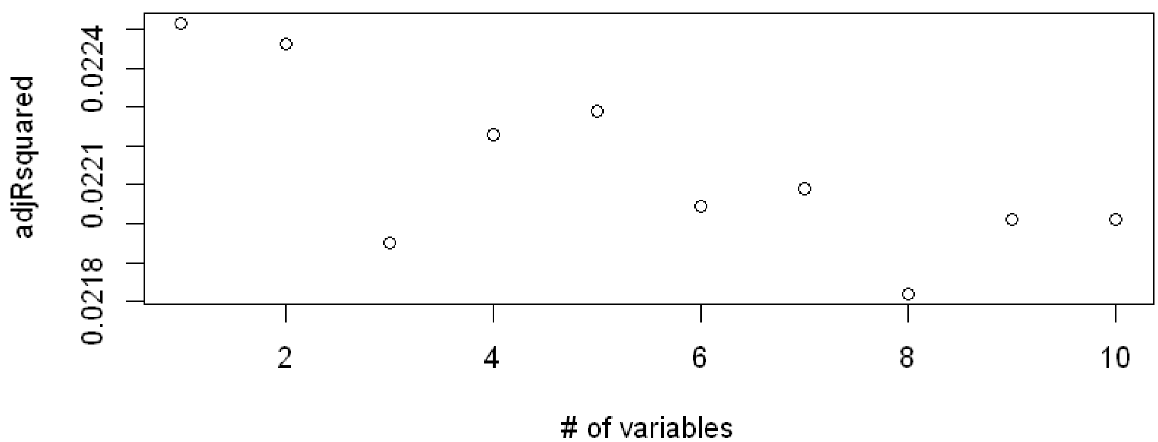
```
11 x 1 sparse Matrix of class "dgCMatrix"
                                1
(Intercept)          -1.330540e+01
(Intercept)            .
HSGPA                -6.391154e-04
SAT_Total            -3.621187e-05
Entry_Term            6.863673e-03
Persistence.NextYear  1.197693e+00
N.As                   .
Perc.Pass            -4.021311e-02
Perc.Withd             .
N.GraduateCourse      2.356422e-03
FullTimeStudent       1.697602e-03
```

Best Subset Selection (Backwards Selection)

We proceed with conducting best subset selection on the training set, specifically backwards selection to iterively remove least contributing predictors. We train the model using the "leapBackward" method and using the train() function from caret, performing 5-fold CV. We then extract a relationship between MSE, adjusted R squared and # of variables.

In [10]:
```
set.seed(123)
train.control <- trainControl(method = "cv", number = 5)

step.model <- train(Term.GPA ~., data = train,
                    method = "leapBackward",
                    tuneGrid = data.frame(nvmax = 1:10),
                    trControl = train.control
                    )
step.adj.rsq = step.model$results$RsquaredSD
step.MSE = (step.model$results$RMSE)^2
par(mfrow=c(2,1))
plot(step.model$results$nvmax,step.MSE, xlab = "# of variables", ylab = "MSE")
plot(step.model$results$nvmax, step.adj.rsq, xlab = "# of variables", ylab = "adj
```





From the graphs, we can assert that the best model has 2 variables, since that is the point where the highest R squared and MSE coincide. We can check this:

In [11]:
```
step.model$bestTune
```

	nvmax
2	2

In [12]:
```
coef(step.model$finalModel, 2)
```

(Intercept)	-18.4944056625729
Entry_Term	0.00921905774701395
Persistence.NextYear	1.22128425825323

Hence, backwards selection creates a model with only 2 variables from the original 10.

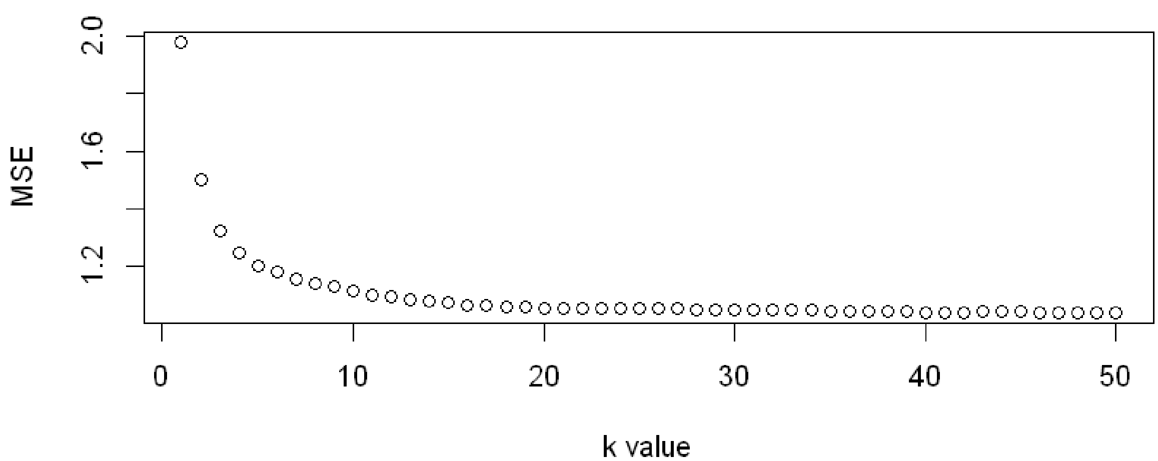
We can test the model using our test data:

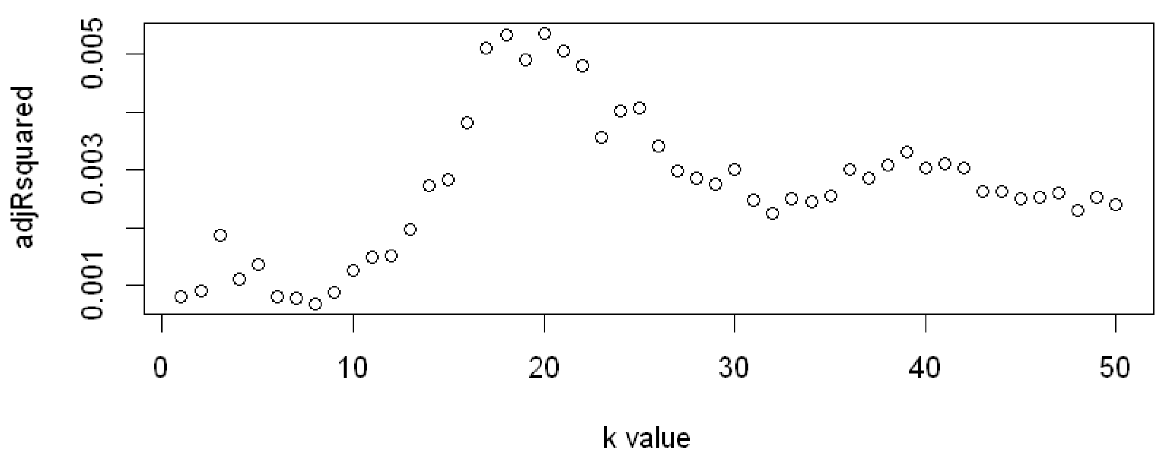
In [23]:
```
step.model.train = lm(Term.GPA~Entry_Term + Persistence.NextYear, data = train)
step.model.test = lm(Term.GPA~Entry_Term + Persistence.NextYear, data = test)
step.pred.train = predict(step.model, data = train)
step.pred.test = predict(step.model, data = test)
step.train.MSE = mean((step.pred.train - train$Term.GPA)^2)
step.test.PMSE = mean((step.pred.test - test$Term.GPA)^2)
RSS.step = sum((step.pred.test - test$Term.GPA)^2)
TSS.step = sum((test$Term.GPA - mean(test$Term.GPA))^2)
step.Rsqr = 1 - (RSS.step/TSS.step)
```

kNN Regression

```
Here we will train a kNN regression model again using the train() function in
caret. In order to output results for different values of k, we will set
tuneGrid equal to a vector 1:50 to test MSE's for values of k ranging from 1 to
50. This is how we will find the best k Nearest Neighbors for our estimation.
```

In [14]:
```
trcontrol.knn = trainControl(method = "cv",
                             number = 5)
knn.model = train(Term.GPA~.,
                  method = "knn",
                  tuneGrid = expand.grid(k = 1:50),
                  trControl = trcontrol.knn,
                  data = train)
knn.MSE = (knn.model$results$RMSE)^2
knn.adjrsqr = knn.model$results$RsquaredSD
par(mfrow=c(2,1))
plot(knn.model$results$k,knn.MSE, xlab = "k value", ylab = "MSE")
plot(knn.model$results$k, knn.adjrsqr, xlab = "k value", ylab = "adjRsquared")
```





We notice a consistent decrease in MSE as we increase k, but we also notice R squared is hghest at k = 20, so that is our optimum k value for this model. We can proceed to model testing.

In [15]:
```
knn.pred.train = predict(knn.model, train)
knn.pred.test = predict(knn.model, test)
knn.MSE.train = mean((knn.pred.train - train$Term.GPA)^2)
knn.PMSE.test = mean((knn.pred.test - test$Term.GPA)^2)
RSS.knn = sum((knn.pred.test - test$Term.GPA)^2)
TSS.knn = sum((test$Term.GPA - mean(test$Term.GPA))^2)
knn.Rsqr = 1 - (RSS.knn/TSS.knn)
```

Multi-Regression Results

In [25]:
```
trainMSE = c(lm.train.MSE,ridge.train.MSE,lasso.train.MSE,step.train.MSE,knn.MSE.
testMSE = c(lm.test.PMSE,ridge.test.PMSE,lasso.test.PMSE,step.test.PMSE,knn.PMSE.
ResultsMLR = cbind(trainMSE, testMSE)
row.names(ResultsMLR) = c("MLR", "Ridge", "Lasso", "Step", "kNN")
ResultsMLR
```

	trainMSE	testMSE
MLR	0.8408511	0.8002639
Ridge	0.7892362	0.8163820
Lasso	0.7899882	0.8134883
Step	0.7906234	1.2494092
kNN	0.9959190	1.0163150

Best value of k nearest neighbors was 20, and lambda was 0.01 for Lasso and Ridge. Unfortunately, linear techniques did not perform well at all. Unfortunately since my R squared value for kNN is negative, I have made a big mistake in the calculations. It seems, however, that the full linear model is the best performing.

Classification Models

Data Exploration

We analyze the correlation matrix:

In [30]:
```
cor(train)
```

	HSGPA	SAT_Total	Entry_Term	Term.GPA	Persistence.NextYear
HSGPA	1.000000000	-0.003758312	-0.103645005	0.060280036	0.1745613391
SAT_Total	-0.003758312	1.000000000	-0.038816754	-0.022627828	-0.0076037587
Entry_Term	-0.103645005	-0.038816754	1.000000000	0.004810567	-0.1113161389
Term.GPA	0.060280036	-0.022627828	0.004810567	1.000000000	0.4775423915
Persistence.NextYear	0.174561339	-0.007603759	-0.111316139	0.477542392	1.0000000000
N.As	0.005033852	-0.013697361	-0.120398872	0.019541574	0.0782088497
Perc.Pass	0.030307725	0.022810338	-0.226454081	-0.001818066	0.0658342385
Perc.Withd	0.007988513	0.025873840	-0.188673739	-0.005920765	0.0112324292
N.GraduateCourse	0.016572219	-0.020778679	0.003716367	0.010332220	-0.0032990788
FullTimeStudent	0.003711931	-0.002089033	0.077900207	0.014200012	-0.0005602344

The Best predictors of Persistence Next Year is Term.GPA, Entry_Term, and HSGPA. Hence we will proceed to explore these predictors.

Logistic Regression

We fit a logistic regression model to the training data using caret with 5-fold CV:

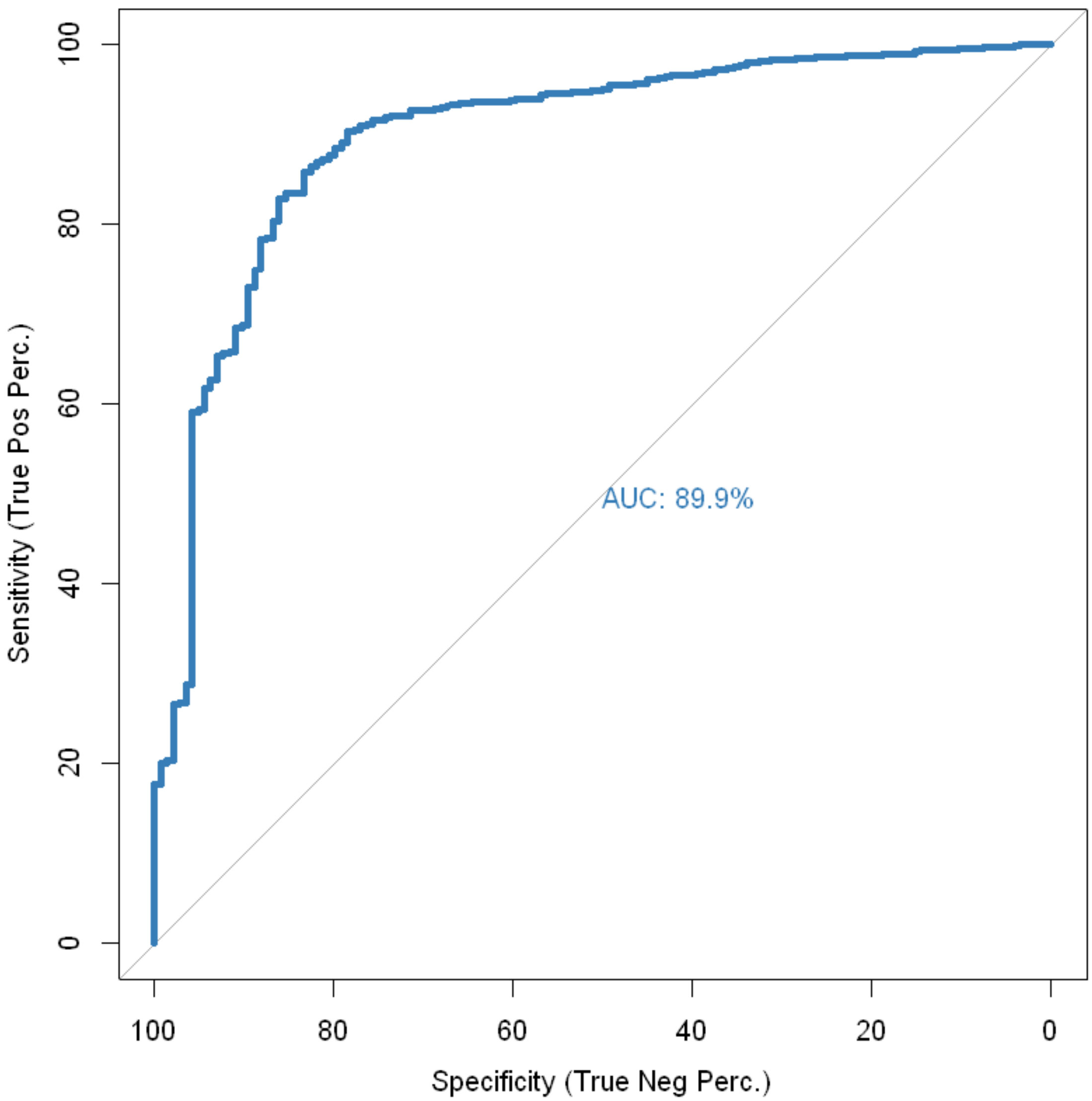
In [32]:
```
log.fit = train(Persistence.NextYear~Term.GPA+Entry_Term+HSGPA, method = "glm", t
               ,data = train)
glm.predict.train = predict(log.fit, train)
glm.predict.train[glm.predict.train>.5]="Persistent"
glm.predict.train[glm.predict.train<=.5]="Dropped"

train.Persistence = train$Persistence.NextYear

train.Persistence[train.Persistence>.5]="Persistent"
train.Persistence[train.Persistence<=.5]="Dropped"

table(glm.predict.train, train.Persistence)

log.Err.train = mean(glm.predict.train != train.Persistence)
log.Acc.train = mean(glm.predict.train == train.Persistence)
log.Prec.train = 561/(561+254)

glm.predict.test = predict(log.fit, test)
glm.predict.test[glm.predict.test>.5]="Persistent" #1
glm.predict.test[glm.predict.test<=.5]="Dropped" #0

test.Persistence = test$Persistence.NextYear
test.Persistence[test.Persistence>.5]="Persistent"
test.Persistence[test.Persistence<=.5]="Dropped"

table(glm.predict.test, test.Persistence)

log.Err.test = mean(glm.predict.test != test.Persistence)
log.Acc.test = mean(glm.predict.test == test.Persistence)
log.Prec.test = 17/(17+7)
log.sens = 1294/(1294 + 7)
log.spec = 17/(17 + 127)
```

```
                 train.Persistence
glm.predict.train Dropped Persistent
       Dropped        318         97
       Persistent     849       4493

                test.Persistence
glm.predict.test Dropped Persistent
      Dropped         17          7
      Persistent     127       1294
```

With Logistic Regression, we are able to make a ROC graph to plot Sensitivity and Specificity Trade-off:

In [33]:
```
par(pty = "s")
glm.roc = roc(test$Persistence.NextYear, predict(log.fit, test), plot = TRUE, per
glm.roc

glm.roc.info = roc(test$Persistence.NextYear,predict(log.fit, test), legacy.axes
glm.roc.df = data.frame(tpp = glm.roc.info$sensitivities*100,
                        fpp = (1 - glm.roc.info$specificities)*100,
                        thresholds = glm.roc.info$thresholds)
```

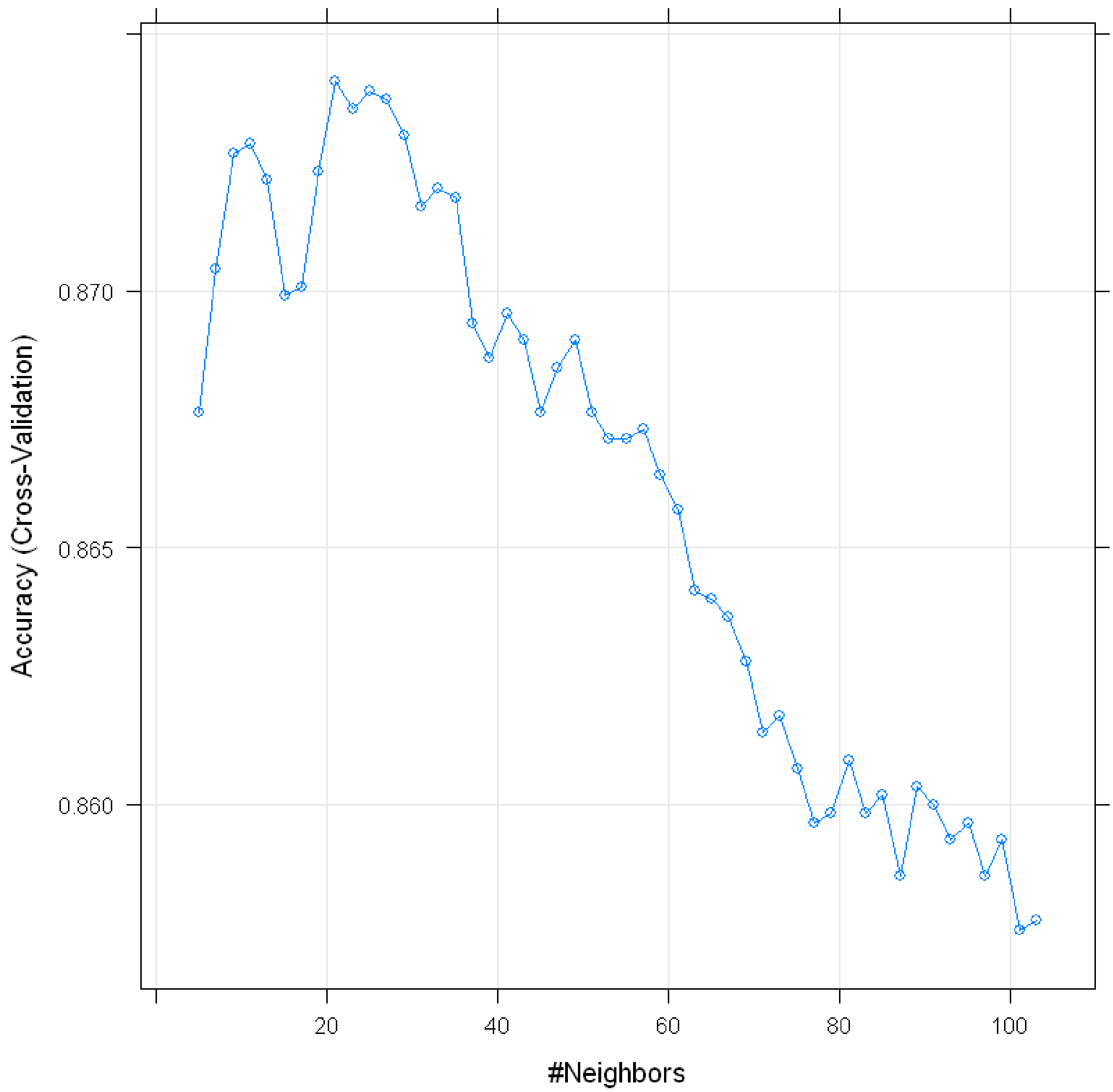
```
Setting levels: control = 0, case = 1
Setting direction: controls < cases


Call:
roc.default(response = test$Persistence.NextYear, predictor = predict(log.fit,
test), percent = TRUE, plot = TRUE, xlab = "Specificity (True Neg Perc.)",
ylab = "Sensitivity (True Pos Perc.)", col = "#377eb8", lwd = 4,     print.auc
= TRUE)

Data: predict(log.fit, test) in 144 controls (test$Persistence.NextYear 0) < 13
01 cases (test$Persistence.NextYear 1).
Area under the curve: 89.91%


Setting levels: control = 0, case = 1
Setting direction: controls < cases
```



kNN Classifier

We use train() to train a kNN classifier using our test data. Again, we iterate through values of k neighbors from 1 to 50:

In [34]:
```
set.seed(123)
knn.fit.class = train(as.factor(Persistence.NextYear)~Term.GPA+Entry_Term+HSGPA,
                   method = "knn",
                   tuneLength = 50,
                   trControl = train.control,
                   preProcess = c("center", "scale"),
                   data = train,
                     metric = "Accuracy")
plot(knn.fit.class)
```



We observe the optimal k nearest neighbors to be k = 21. We can now make predictions on this model using the test set:

In [44]:
```
knn.predict.train = predict(knn.fit.class,train)
knn.predict.test = predict(knn.fit.class,test)
table(knn.predict.train,train.Persistence)
table(knn.predict.test,test.Persistence)
knn.Err.train = mean(knn.predict.train != train$Persistence.NextYear)
knn.Acc.train = mean(knn.predict.train == train$Persistence.NextYear)
knn.Prec.train = 723/(723+226)
knn.Err.test = mean(knn.predict.test != test$Persistence.NextYear)
knn.Acc.test = mean(knn.predict.test == test$Persistence.NextYear)
knn.Prec.test = 113/(113+62)
knn.sens = 1239/(1239+62)
knn.spec = 113/(113+31)
```

```
                 train.Persistence
knn.predict.train Dropped Persistent
                0     718        229
                1     449       4361

                test.Persistence
knn.predict.test Dropped Persistent
               0     113         62
               1      31       1239
```

Naive Bayes Classifier

We will use train() to train a naive bayes classifier using 5-fold CV.

In [36]:
```
nb.fit = train(as.factor(Persistence.NextYear)~Term.GPA+Entry_Term+HSGPA, method
              trControl = train.control,
              data = train,
              metric = "Accuracy")
nb.predict.train = predict(nb.fit,train)
nb.predict.test = predict(nb.fit,test)
table(nb.predict.train, train.Persistence)
nb.Err.train = mean(nb.predict.train != train$Persistence.NextYear)
nb.Acc.train = mean(nb.predict.train == train$Persistence.NextYear)
nb.Prec.train = 568/(568+136)
nb.predict.test = predict(nb.fit, test)
table(nb.predict.test, test.Persistence)
nb.Err.test = mean(nb.predict.test != test$Persistence.NextYear)
nb.Acc.test = mean(nb.predict.test == test$Persistence.NextYear)
nb.Prec.test = 94/(94+25)
nb.sens = 1276/(1276 + 25)
nb.spec = 94/(94 + 50)
```

```
                train.Persistence
nb.predict.train Dropped Persistent
               0     568        136
               1     599       4454

               test.Persistence
nb.predict.test Dropped Persistent
              0      94         25
              1      50       1276
```

LDA

Here we perform linear discriminant analysis and train the model using train() and 5-fold CV.

In [37]:
```
lda.fit = train(as.factor(Persistence.NextYear)~Term.GPA+HSGPA+Entry_Term,method
               data = train)
lda.predict.train = predict(lda.fit,train)
lda.predict.test = predict(lda.fit,test)
table(lda.predict.train, train.Persistence)
lda.Err.train = mean(lda.predict.train != train$Persistence.NextYear)
lda.Acc.train = mean(lda.predict.train == train$Persistence.NextYear)
lda.Prec.train = 550/(568+258)
lda.predict.test = predict(lda.fit, test)
table(lda.predict.test, test.Persistence)
lda.Err.test = mean(lda.predict.test != test$Persistence.NextYear)
lda.Acc.test = mean(lda.predict.test == test$Persistence.NextYear)
lda.Prec.test = 45/(45+22)
lda.sens = 1279/(1279 + 22)
lda.spec = 45/(45 + 99)
```

```
                 train.Persistence
lda.predict.train Dropped Persistent
                0     550        258
                1     617       4332

                test.Persistence
lda.predict.test Dropped Persistent
               0      45         22
               1      99       1279
```

Here we perform quantitative descriptive analysis and train the model using train() and 5-fold CV

In [38]:
```
qda.fit = train(as.factor(Persistence.NextYear)~Term.GPA+HSGPA+Entry_Term,method
               data = train)
qda.predict.train = predict(qda.fit,train)
qda.predict.test = predict(qda.fit,test)
table(qda.predict.train, train.Persistence)
qda.Err.train = mean(qda.predict.train != train$Persistence.NextYear)
qda.Acc.train = mean(qda.predict.train == train$Persistence.NextYear)
qda.Prec.train = 603/(603+312)
qda.predict.test = predict(qda.fit, test)
table(lda.predict.test, test.Persistence)
qda.Err.test = mean(qda.predict.test != test$Persistence.NextYear)
qda.Acc.test = mean(qda.predict.test == test$Persistence.NextYear)
qda.Prec.test = 45/(45+22)
qda.sens = 1279/(1279 + 22)
qda.spec = 45/(45 + 99)
```

```
                 train.Persistence
qda.predict.train Dropped Persistent
                0     603        312
                1     564       4278

                test.Persistence
lda.predict.test Dropped Persistent
               0      45         22
               1      99       1279
```

Classification Results

In [49]:
```
train.err = c(log.Err.train,knn.Err.train,nb.Err.train,lda.Err.train,qda.Err.trai
test.err = c(log.Err.test,knn.Err.test,nb.Err.test,lda.Err.test,qda.Err.test)
train.prec = c(log.Prec.train,knn.Prec.train,nb.Prec.train,lda.Prec.train,qda.Pre
test.prec = c(log.Prec.test,knn.Prec.test,nb.Prec.test,lda.Prec.test,qda.Prec.tes
train.acc = c(log.Acc.train,knn.Acc.train,nb.Acc.train,lda.Acc.train,qda.Acc.trai
test.acc = c(log.Acc.test,knn.Acc.test,nb.Acc.test,lda.Acc.test,qda.Acc.test)
sensitivity = c(log.sens,knn.sens,nb.sens,lda.sens,qda.sens)
specificity = c(log.spec,knn.spec,nb.spec,lda.spec,qda.spec)
results.class = cbind(train.err,test.err,train.prec,test.prec,train.acc,test.acc,
row.names(results.class) = c("Logistic Reg","kNN","Naive Bayes","LDA","QDA")
results.class
```

	train.err	test.err	train.prec	test.prec	train.acc	test.acc	sensitivity	specificity
Logistic Reg	0.1643217	0.09273356	0.6883436	0.7083333	0.8356783	0.9072664	0.9946195	0.1180556
kNN	0.1177697	0.06435986	0.7618546	0.6457143	0.8822303	0.9356401	0.9523444	0.7847222
Naive Bayes	0.1276707	0.05190311	0.8068182	0.7899160	0.8723293	0.9480969	0.9807840	0.6527778
LDA	0.1519889	0.08373702	0.6658596	0.6716418	0.8480111	0.9162630	0.9830899	0.3125000
QDA	0.1521626	0.08719723	0.6590164	0.6716418	0.8478374	0.9128028	0.9830899	0.3125000

From the Results table, it suffices to say that the superior model in this case is kNN with k = 21. While Naive Bayes comes is also a healthy model, kNN has better sensitivity to specificity trade off.