mag_final

Bingyu Sun

5/8/2019

##Data import & cleaning * Reclassify response variable to make it binary

```
apple = read_csv("./data/AppleStore.csv") %>%
  janitor::clean_names() %>%
  dplyr::select(-c(x1, id, track_name, currency, ver)) %>%
  mutate(size_bytes = round(size_bytes * 1e-6),
         cont_rating = factor(cont_rating, levels = c("4+", "9+", "12+", "17+")), #ascending order
         user_rating = ifelse(user_rating >= 4, "high", "low"),
         user_rating = factor(user_rating, levels = c("low", "high"))) %>%
  rename(size_megabytes = size_bytes) %>%
  filter(rating_count_tot != 0,
         user_rating_ver != 0) %>% #Remove apps with no user rating, and apps with no rating on current
  mutate(prime_genre = as.integer(ifelse(prime_genre == "Games", 1, 0))) %>%
  dplyr::select(-rating_count_tot, -rating_count_ver, -vpp_lic) %>%
  dplyr::select(user_rating, everything())
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   id = col_double(),
##   track_name = col_character(),
##   size_bytes = col_double(),
##   currency = col_character(),
##   price = col_double(),
##   rating_count_tot = col_double(),
##   rating_count_ver = col_double(),
##   user_rating = col_double(),
##   user_rating_ver = col_double(),
##   ver = col_character(),
##   cont_rating = col_character(),
##   prime_genre = col_character(),
##   sup_devices.num = col_double(),
##   ipadSc_urls.num = col_double(),
##   lang.num = col_double(),
##   vpp_lic = col_double()
## )
```

```
str(apple)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 5754 obs. of  9 variables:
##  $ user_rating     : Factor w/ 2 levels "low","high": 2 2 1 2 2 2 2 2 2 2 ...
##  $ size_megabytes  : num  101 159 101 129 93 10 228 130 49 70 ...
##  $ price           : num  3.99 0 0 0 0 0.99 0 0 9.99 3.99 ...
##  $ user_rating_ver : num  4.5 3.5 4.5 4.5 5 4 4.5 4.5 5 4 ...
##  $ cont_rating     : Factor w/ 4 levels "4+","9+","12+",..: 1 1 1 3 1 1 1 3 1 1 ...
##  $ prime_genre     : int  1 0 0 0 0 1 0 0 0 1 ...
##  $ sup_devices_num : num  38 37 37 37 37 47 37 37 37 38 ...
```

```
##  $ ipad_sc_urls_num: num  5 5 5 5 5 5 0 4 5 0 ...
##  $ lang_num        : num  10 23 3 9 45 1 19 1 1 10 ...
```

```
table(apple$user_rating)
```

```
##
##  low high
## 1285 4469
```

##Split to train/test sets

```
set.seed(1234)
#Split data to traning and testing
trRows = createDataPartition(apple$user_rating,
                             p = .75,
                             list = FALSE)
train_data = apple[trRows,]
test_data = apple[-trRows,]
#in matrix form
x_train = model.matrix(user_rating~., train_data)[,-1]
y_train = train_data$user_rating
x_test = model.matrix(user_rating~., test_data)[,-1]
y_test = test_data$user_rating

#CV method
ctrl1 = trainControl(method = "cv", number = 10)
ctrl2 = trainControl(method = "cv",
                     number = 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)
```

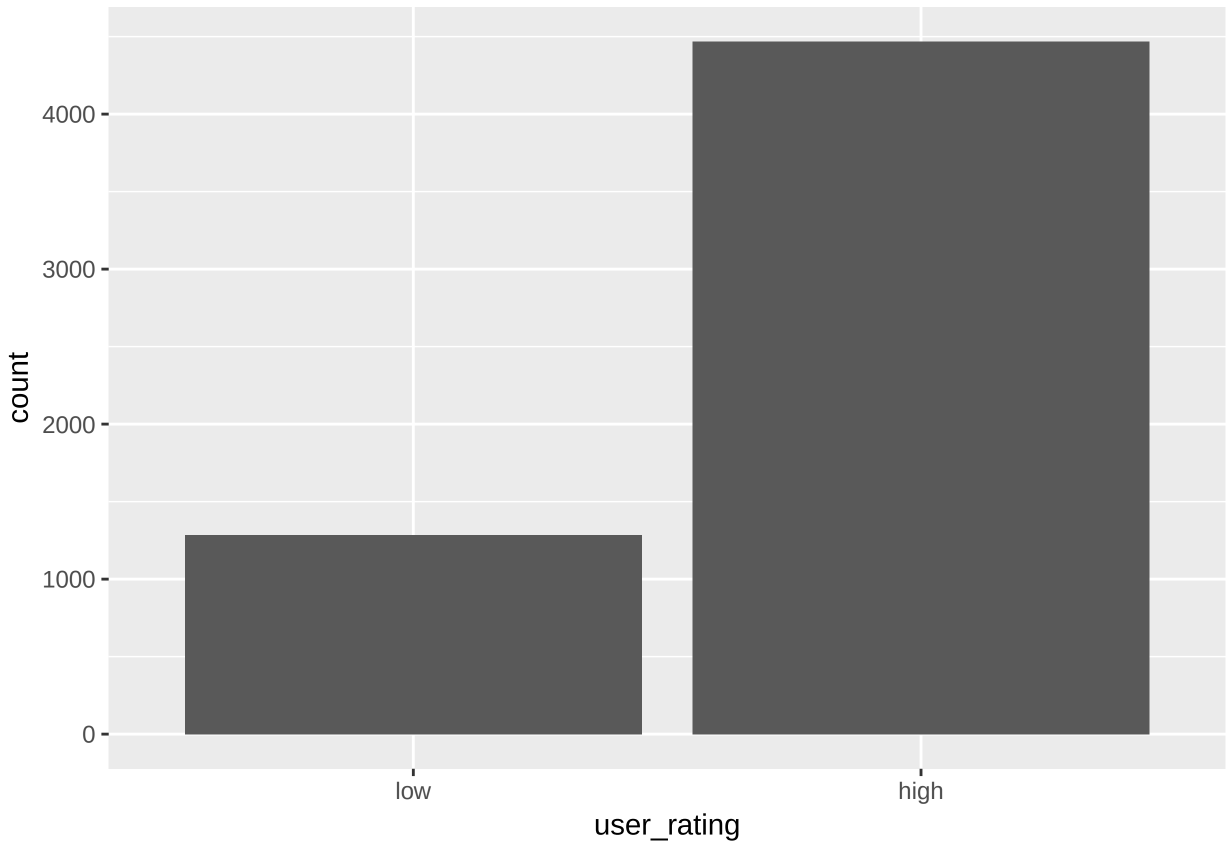
#Supervised learning

##Classification

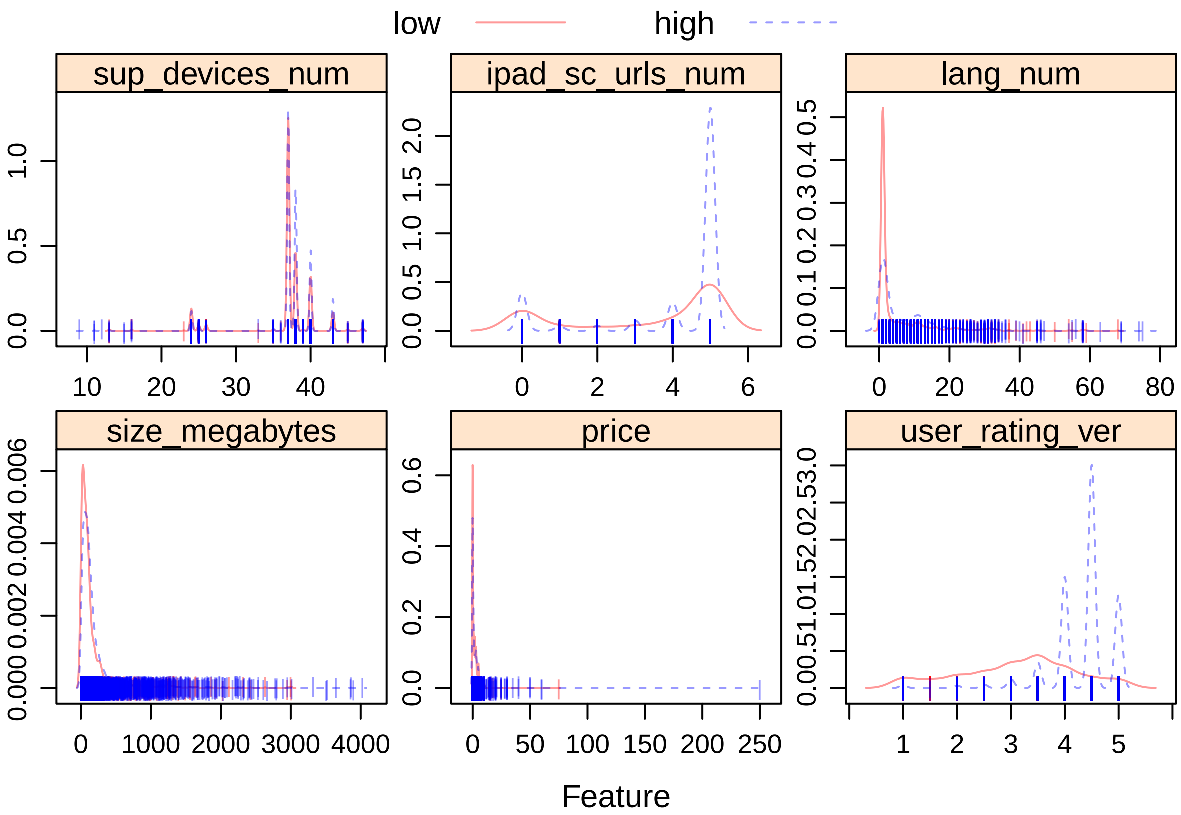
###For linear/non-linear decision boundary

####EDA

```
#barplot for response
apple %>%
  ggplot(aes(x = user_rating)) +
  geom_bar()
```




```
transparentTheme(trans = .4)
featurePlot(x = apple[, c(2:4, 7:9)],
            y = apple$user_rating,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```



1a. Logistic Regression

- For large p, do penalization (ridge, lasso, elastic net)

```
glm.fit <- glm(user_rating~.,
               data = train_data,
               family = binomial)

contrasts(train_data$user_rating)
```

```
##      high
## low     0
## high    1
```

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = user_rating ~ ., family = binomial, data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9099   0.2445   0.3806   0.5370   2.9511
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -6.9282844  0.4912665 -14.103  < 2e-16 ***
## size_megabytes   -0.0001257  0.0001328  -0.947 0.343777
## price             0.0169367  0.0141117   1.200 0.230064
```

```
## user_rating_ver   1.6750654  0.0630475  26.568  < 2e-16 ***
## cont_rating9+     0.0893751  0.1447284   0.618 0.536881
## cont_rating12+    0.1454597  0.1328055   1.095 0.273393
## cont_rating17+   -0.1020242  0.1869774  -0.546 0.585306
## prime_genre       0.5241544  0.1072396   4.888 1.02e-06 ***
## sup_devices_num   0.0261461  0.0111411   2.347 0.018935 *
## ipad_sc_urls_num  0.0746664  0.0251981   2.963 0.003045 **
## lang_num          0.0198464  0.0059875   3.315 0.000918 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4584.6  on 4315  degrees of freedom
## Residual deviance: 3139.5  on 4305  degrees of freedom
## AIC: 3161.5
##
## Number of Fisher Scoring iterations: 5
```

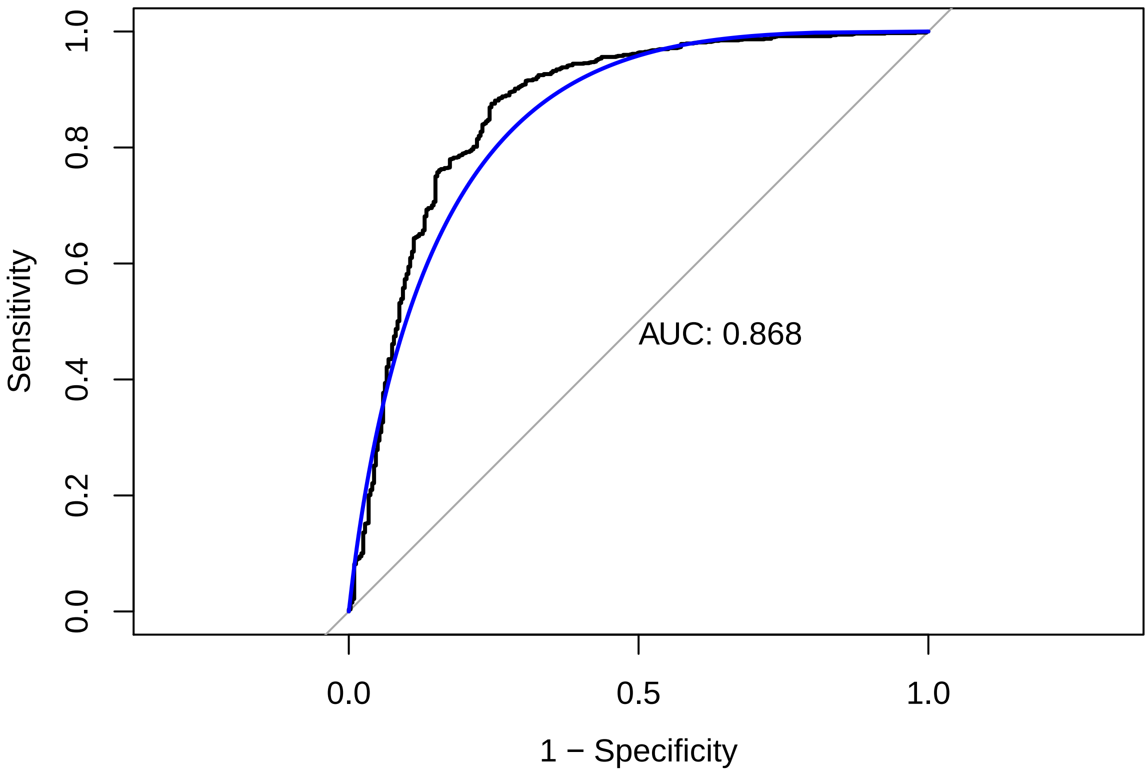
```
test.pred.prob  <- predict(glm.fit, newdata = test_data,
                           type = "response")
test.pred <- rep("low", length(test.pred.prob))
test.pred[test.pred.prob > 0.5] <- "high" #Bayes classifier (cutoff 0.5)

#Evaluate performance on the test data
confusionMatrix(data = factor(test.pred, levels = c("low", "high")),
                reference = test_data$user_rating,
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low   149   35
##       high  172 1082
##
##                Accuracy : 0.8561
##                  95% CI : (0.8368, 0.8738)
##     No Information Rate : 0.7768
##     P-Value [Acc > NIR] : 2.187e-14
##
##                   Kappa : 0.5105
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9687
##             Specificity : 0.4642
##          Pos Pred Value : 0.8628
##          Neg Pred Value : 0.8098
##              Prevalence : 0.7768
##          Detection Rate : 0.7524
##    Detection Prevalence : 0.8720
##       Balanced Accuracy : 0.7164
##
##        'Positive' Class : high
```

```
##
```

```r
#Plot the test ROC curve
roc.glm <- roc(test_data$user_rating, test.pred.prob)
plot(roc.glm, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.glm), col = 4, add = TRUE)
```



For comparison, fit logistic regression using caret

```r
set.seed(1234)
model.glm <- train(x = train_data[2:9],
                   y = train_data$user_rating,
                   method = "glm",
                   metric = "ROC",
                   trControl = ctrl2)
```

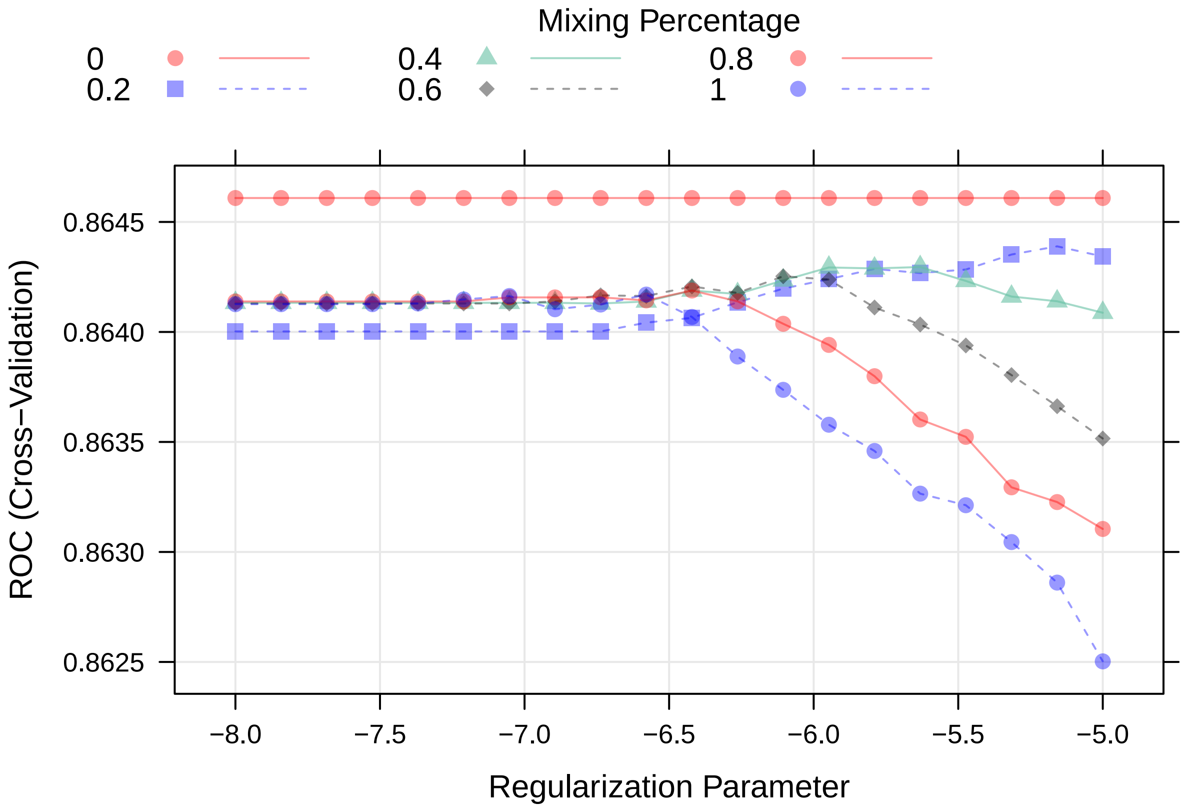
Consider penalization, do regularized logistic regression with glmnet, select the optimal tuning parameters

```r
glmnGrid <- expand.grid(.alpha = seq(0, 1, length = 6),
                        .lambda = exp(seq(-8, -5, length = 20)))
set.seed(1234)
model.glmn <- train(x = x_train,
                    y = y_train,
                    method = "glmnet",
                    tuneGrid = glmnGrid,
                    metric = "ROC",
                    trControl = ctrl2)

model.glmn$bestTune
```

```
##    alpha      lambda
## 20     0 0.006737947
```

```r
plot(model.glmn, xTrans = function(x) log(x))
```



1b. GAM: consider non-linear covariates

```r
# Start with linear model; do not assume nonlinear trait
gam.m1 <- gam(user_rating ~ size_megabytes + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + lang_num ,
              data = train_data,
              family = binomial)
summary(gam.m1)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## user_rating ~ size_megabytes + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
##
## Parametric coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -6.9282844  0.4912673 -14.103  < 2e-16 ***
## size_megabytes      -0.0001257  0.0001328  -0.947 0.343778
## price                0.0169367  0.0141117   1.200 0.230065
## user_rating_ver      1.6750654  0.0630477  26.568  < 2e-16 ***
## cont_rating9+        0.0893751  0.1447286   0.618 0.536881
## cont_rating12+       0.1454597  0.1328057   1.095 0.273393
```

```
## cont_rating17+   -0.1020242  0.1869776  -0.546 0.585307
## prime_genre       0.5241544  0.1072397   4.888 1.02e-06 ***
## sup_devices_num   0.0261461  0.0111411   2.347 0.018935 *
## ipad_sc_urls_num  0.0746664  0.0251981   2.963 0.003045 **
## lang_num          0.0198464  0.0059875   3.315 0.000918 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =  0.376   Deviance explained = 31.5%
## UBRE = -0.2675  Scale est. = 1         n = 4316
```

```r
# add one non-linear component to size bytes
gam.m2 <- gam(user_rating ~ s(size_megabytes) + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + lang_num,
              data = train_data,
              family = binomial)
summary(gam.m2)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
##
## Parametric coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -6.961597   0.488637 -14.247  < 2e-16 ***
## price             0.017887   0.014120   1.267 0.205236
## user_rating_ver   1.676702   0.063074  26.583  < 2e-16 ***
## cont_rating9+     0.109541   0.145741   0.752 0.452284
## cont_rating12+    0.149190   0.132726   1.124 0.260992
## cont_rating17+   -0.098284   0.186893  -0.526 0.598968
## prime_genre       0.536861   0.107984   4.972 6.64e-07 ***
## sup_devices_num   0.025562   0.011166   2.289 0.022058 *
## ipad_sc_urls_num  0.077083   0.025349   3.041 0.002359 **
## lang_num          0.019940   0.005988   3.330 0.000869 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df Chi.sq p-value
## s(size_megabytes) 1.86  2.331  2.452    0.31
##
## R-sq.(adj) =  0.377   Deviance explained = 31.6%
## UBRE = -0.26777  Scale est. = 1         n = 4316
```

```r
#  add one non-linear component to lang_num
gam.m3 <- gam(user_rating ~ s(size_megabytes) + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + s(lang_num),
              data = train_data,
```

```r
              family = binomial)

# add one non-linear component to user rating of current version
gam.m4 <- gam(user_rating ~ s(size_megabytes) + price +
                s(user_rating_ver, k = 9) + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + s(lang_num),
              data = train_data,
              family = binomial)

anova(gam.m1, gam.m2, gam.m3, gam.m4, test = "F")
```

```
## Warning: using F test with a 'binomial' family is inappropriate

## Analysis of Deviance Table
##
## Model 1: user_rating ~ size_megabytes + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
## Model 2: user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
## Model 3: user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + s(lang_num)
## Model 4: user_rating ~ s(size_megabytes) + price + s(user_rating_ver,
##     k = 9) + cont_rating + prime_genre + sup_devices_num + ipad_sc_urls_num +
##     s(lang_num)
##   Resid. Df Resid. Dev     Df Deviance       F    Pr(>F)
## 1    4305.0     3139.5
## 2    4303.7     3136.6 1.3311    2.909  2.1851    0.1312
## 3    4300.6     3106.7 3.0536   29.849  9.7750 1.597e-06 ***
## 4    4293.2     2853.8 7.4311  252.946 34.0387 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

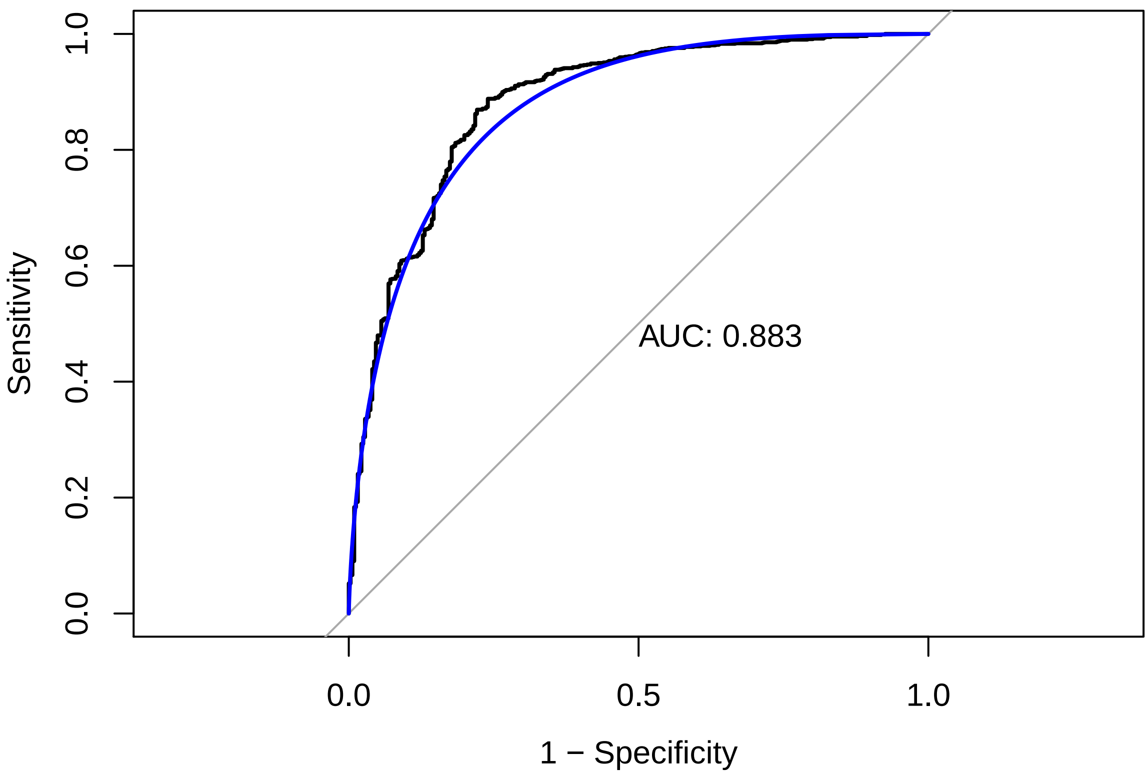
```r
test.pred.prob  <- predict(gam.m4, newdata = test_data,
                           type = "response")
test.pred <- rep("low", length(test.pred.prob))
test.pred[test.pred.prob > 0.5] <- "high" #Bayes classifier (cutoff 0.5)

#Evaluate performance on the test data
confusionMatrix(data = factor(test.pred, levels = c("low", "high")),
                reference = test_data$user_rating,
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low   207   70
##       high  114 1047
##
##                Accuracy : 0.872
##                  95% CI : (0.8537, 0.8889)
##     No Information Rate : 0.7768
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6121
```

```
##
##  Mcnemar's Test P-Value : 0.001524
##
##             Sensitivity : 0.9373
##             Specificity : 0.6449
##          Pos Pred Value : 0.9018
##          Neg Pred Value : 0.7473
##              Prevalence : 0.7768
##          Detection Rate : 0.7281
##    Detection Prevalence : 0.8074
##       Balanced Accuracy : 0.7911
##
##        'Positive' Class : high
##
```

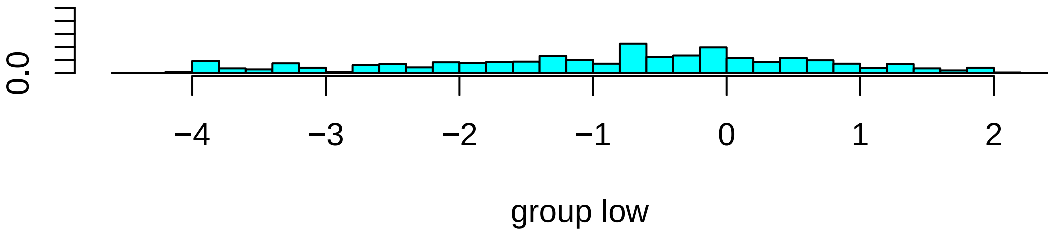
```
#Plot the test ROC curve
roc.glm <- roc(test_data$user_rating, test.pred.prob)
plot(roc.glm, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc.glm), col = 4, add = TRUE)
```

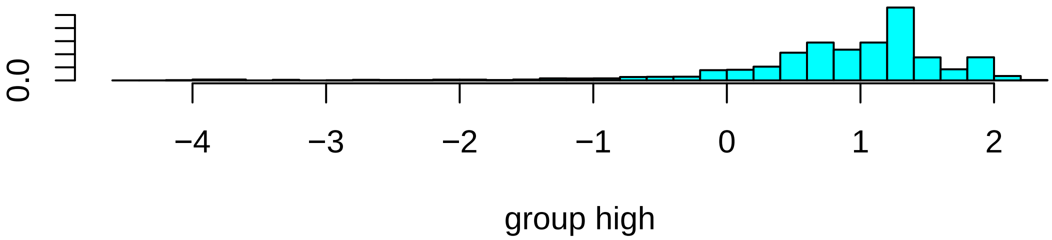


2a. Linear discriminate analysis (LDA)

-Problem for logistic regression: if two classes are widely separated, model is unstable, large variance -Adv: So consider discriminant alaysis, for more than 2 classes, low-dimension views (good when have large p) * assume X normally distributed within each class, assume covariance are the same across classes

```
lda.fit <- lda(user_rating~., data = train_data)
plot(lda.fit)
```



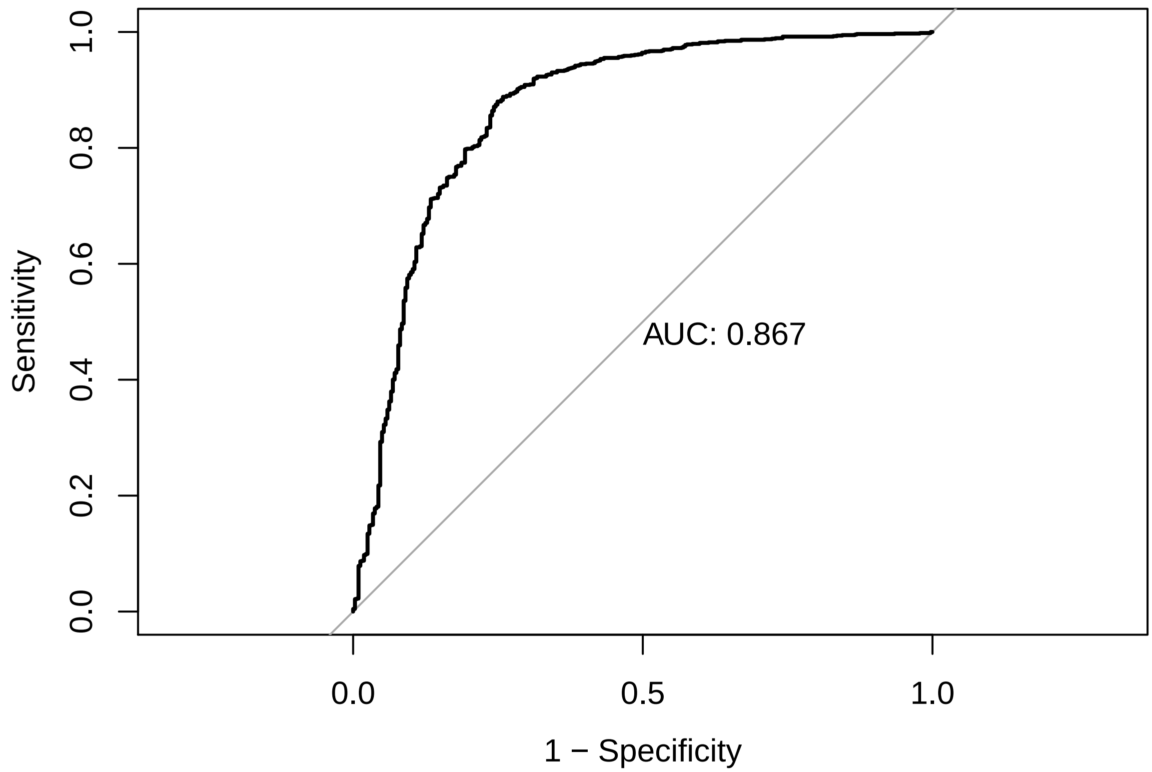



```r
#Evaluate the test set performance using ROC
lda.pred <- predict(lda.fit, newdata = test_data)
head(lda.pred$posterior)
```

```
##          low      high
## 1 0.02647858 0.9735214
## 2 0.07014324 0.9298568
## 3 0.12014310 0.8798569
## 4 0.36628011 0.6337199
## 5 0.35345710 0.6465429
## 6 0.07640427 0.9235957
```

```r
roc.lda <- roc(y_test, lda.pred$posterior[,2], #probability of being positive
               levels = c("low", "high"))

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



Fit LDA with Caret for model comparison

```
set.seed(1234)
model.lda <- train(x = x_train,
                   y = y_train,
                   method = "lda",
                   metric = "ROC",
                   trControl = ctrl2)
```

2b. Quadratic Discriminate analysis (QDA)

- No equal covariance assumption

```
# use qda() in MASS
qda.fit <- qda(user_rating~., data = train_data)

qda.pred <- predict(qda.fit, newdata = test_data)
head(qda.pred$posterior)
```

```
##          low      high
## 1 0.04410839 0.9558916
## 2 0.19323839 0.8067616
## 3 0.85371209 0.1462879
## 4 0.38534119 0.6146588
## 5 0.32271283 0.6772872
## 6 0.11846099 0.8815390
```

For model comparison

```
set.seed(1234)
model.qda <- train(x = x_train,
```

```
                y = y_train,
                method = "qda",
                metric = "ROC",
                trControl = ctrl2)
```

3. **Naivew Bayes**

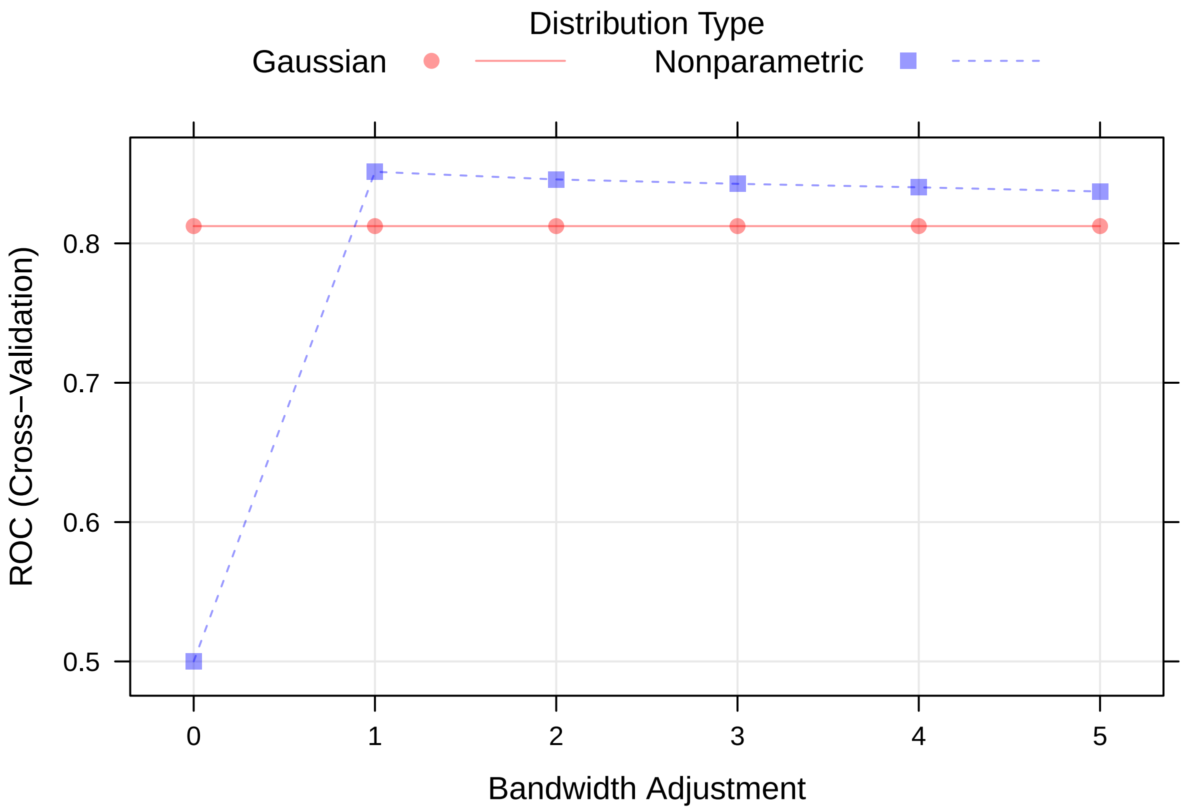
- good for large p, works for mixed p (continuous, categorical)

```
set.seed(1234)

nbGrid <- expand.grid(usekernel = c(FALSE,TRUE),
                      fL = 1,
                      adjust = seq(0,5,by = 1))

model.nb <- train(x = x_train,
                  y = y_train,
                  method = "nb",
                  tuneGrid = nbGrid,
                  metric = "ROC",
                  trControl = ctrl2)

plot(model.nb)
```



4. **KNN**

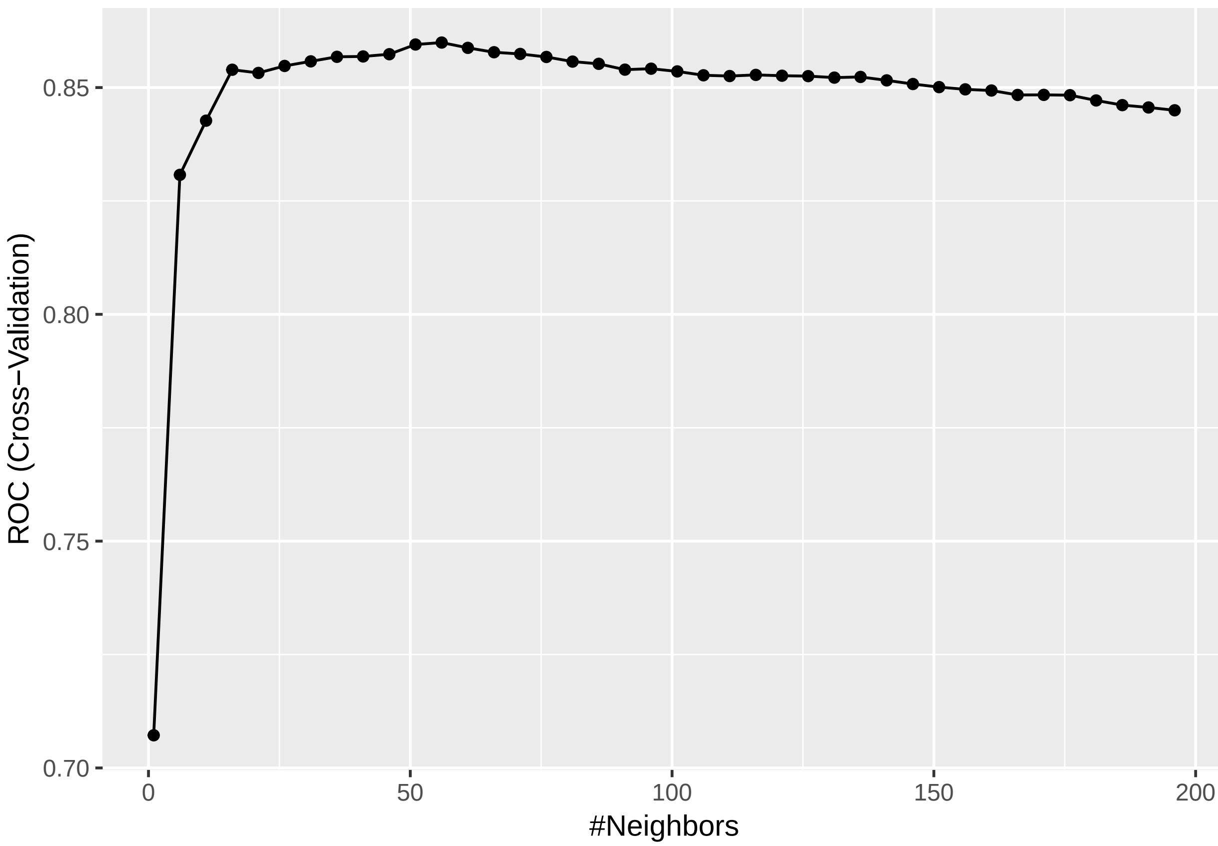
- center and scale first if method is based on distance
- super flexible -Disadv: no assumed model form, don't know relationship btw response and predictor

```r
set.seed(1234)

model.knn <- train(x = x_train,
                   y = y_train,
                   method = "knn",
                   preProcess = c("center","scale"),
                   tuneGrid = data.frame(k = seq(1,200,by=5)),
                   trControl = ctrl2)
```

```
## Warning in train.default(x = x_train, y = y_train, method = "knn",
## preProcess = c("center", : The metric "Accuracy" was not in the result set.
## ROC will be used instead.
```

```r
ggplot(model.knn)
```



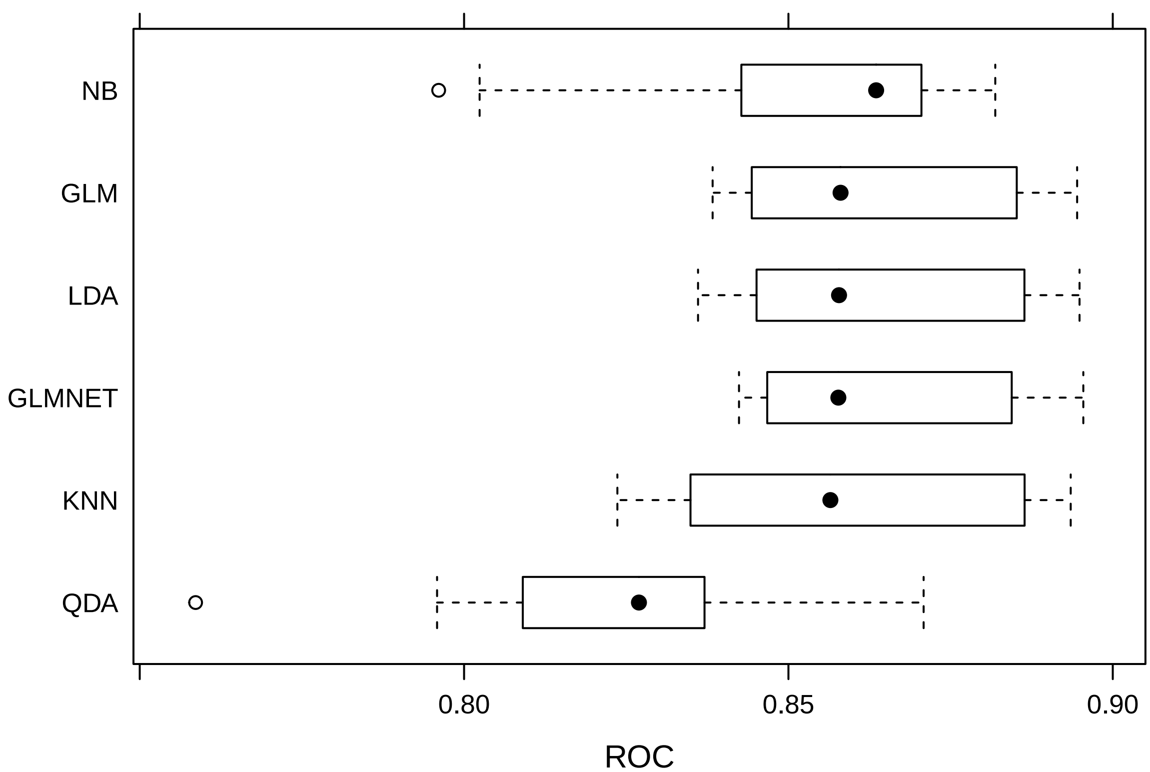
Compare models

```r
res <- resamples(list(GLM = model.glm, GLMNET = model.glmn,
                      LDA = model.lda, QDA = model.qda,
                      NB = model.nb, KNN = model.knn))
summary(res)
```

```
##
## Call:
## summary.resamples(object = res)
##
## Models: GLM, GLMNET, LDA, QDA, NB, KNN
## Number of resamples: 10
```

```
##
## ROC
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GLM     0.8383136 0.8475258 0.8580379 0.8640024 0.8828052 0.8945068    0
## GLMNET 0.8423846 0.8493703 0.8576959 0.8646088 0.8823328 0.8954608    0
## LDA     0.8360671 0.8478760 0.8578047 0.8640890 0.8840033 0.8948761    0
## QDA     0.7586132 0.8104789 0.8269557 0.8216898 0.8368384 0.8708417    0
## NB      0.7960849 0.8439136 0.8635255 0.8513984 0.8703148 0.8818895    0
## KNN     0.8236318 0.8383162 0.8564708 0.8599178 0.8845057 0.8935168    0
##
## Sens
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GLM     0.3608247 0.4389497 0.4766967 0.4689863 0.4935299 0.5520833    0
## GLMNET 0.3298969 0.3687983 0.4270833 0.4212092 0.4690722 0.5104167    0
## LDA     0.3814433 0.4623067 0.4896907 0.4845468 0.5182292 0.5625000    0
## QDA     0.4536082 0.5156250 0.5412371 0.5436426 0.5658827 0.6354167    0
## NB      0.1237113 0.1550419 0.1968965 0.1826138 0.2083333 0.2187500    0
## KNN     0.3402062 0.4062500 0.4145189 0.4191044 0.4494201 0.4845361    0
##
## Spec
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GLM     0.9492537 0.9507463 0.9567830 0.9579335 0.9626866 0.9731343    0
## GLMNET 0.9553571 0.9589552 0.9656716 0.9659906 0.9716418 0.9791045    0
## LDA     0.9432836 0.9500355 0.9567164 0.9576359 0.9634328 0.9761194    0
## QDA     0.8895522 0.9000000 0.9107143 0.9090085 0.9186567 0.9253731    0
## NB      0.9880597 0.9880597 0.9895656 0.9916453 0.9962687 0.9970238    0
## KNN     0.9582090 0.9641791 0.9657249 0.9683769 0.9753931 0.9791045    0
```

```r
bwplot(res, metric = "ROC")
```



Visualize ROCs

```r
lda.pred <- predict(model.lda, newdata = test_data, type = "prob")[,2]
glm.pred <- predict(model.glm, newdata = test_data, type = "prob")[,2]
glmn.pred <- predict(model.glmn, newdata = test_data, type = "prob")[,2]
nb.pred <- predict(model.nb, newdata = test_data, type = "prob")[,2]
qda.pred <- predict(model.qda, newdata = test_data, type = "prob")[,2]
knn.pred <- predict(model.knn, newdata = test_data, type = "prob")[,2]


roc.lda <- roc(y_test, lda.pred)
roc.glm <- roc(y_test, glm.pred)
roc.glmn <- roc(y_test, glmn.pred)
roc.nb <- roc(y_test, nb.pred)
roc.qda <- roc(y_test, qda.pred)
roc.knn <- roc(y_test, knn.pred)

auc <- c(roc.glm$auc[1], roc.glmn$auc[1], roc.lda$auc[1],
         roc.qda$auc[1], roc.nb$auc[1], roc.knn$auc[1])

plot(roc.glm, legacy.axes = TRUE)
plot(roc.glmn, col = 2, add = TRUE)
plot(roc.lda, col = 3, add = TRUE)
plot(roc.qda, col = 4, add = TRUE)
plot(roc.nb, col = 5, add = TRUE)
plot(roc.knn, col = 6, add = TRUE)
modelNames <- c("glm","glmn","lda","qda","nb","knn")
legend("bottomright", legend = paste0(modelNames, ": ", round(auc,3)),
       col = 1:6, lwd = 2)
```

Tree-based methods

- No assumption, less strictive than linear methods, less flexible than knn
- Good interpretation

Regression

1. Regression tree

Classification

2. Classification tree