models

Gaotong LIU

5/17/2020

```
df.raw = read_csv("winequality-red.csv")
```

```
## Parsed with column specification:
## cols(
##   `fixed acidity` = col_double(),
##   `volatile acidity` = col_double(),
##   `citric acid` = col_double(),
##   `residual sugar` = col_double(),
##   chlorides = col_double(),
##   `free sulfur dioxide` = col_double(),
##   `total sulfur dioxide` = col_double(),
##   density = col_double(),
##   pH = col_double(),
##   sulphates = col_double(),
##   alcohol = col_double(),
##   quality = col_double()
## )
```

```
df = df.raw %>%
  janitor::clean_names() %>%
  mutate(quality = factor(quality,
                           labels = c("q3","q4","q5","q6","q7","q8"))) %>%
  mutate(quality = fct_collapse(quality,
                                poor = c("q3","q4","q5"),
                                good = c("q6","q7","q8")))

set.seed(1)
rowTrain <- createDataPartition(y = df$quality,
                                p = 2/3,
                                list = FALSE)
df.train = df[rowTrain,]
x = model.matrix(quality~., df.train)[,-1]
y = df.train$quality
df.test = df[-rowTrain,]
```

```
ctrl1 <- trainControl(method = "repeatedcv",
                      summaryFunction = twoClassSummary,
                      classProbs = TRUE)

ctrl <- trainControl(method = "cv",number = 5,
                      summaryFunction = twoClassSummary,
                      classProbs = TRUE)
```

LDA and QDA

```
set.seed(1)
model.lda <- train(quality~., df,
```

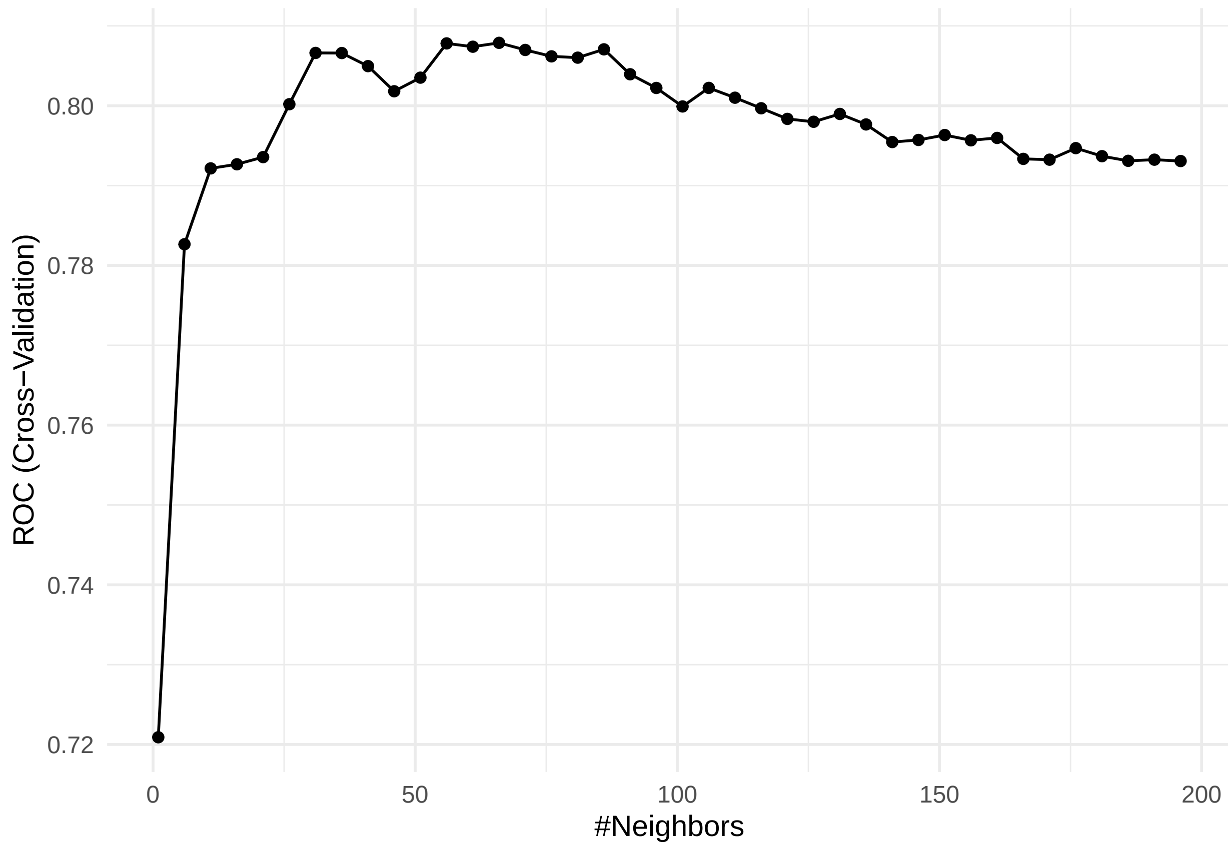
```r
                   subset = rowTrain,
                   method = "lda",
                   metric = "ROC",
                   trControl = ctrl)
set.seed(1)
model.qda <- train(quality~., df,
                   subset = rowTrain,
                   method = "qda",
                   metric = "ROC",
                   trControl = ctrl)
```

KNN

```r
set.seed(1)
knn.fit <- train(quality~., df,
                   subset = rowTrain,
                   method = "knn",
                   preProcess = c("center","scale"),
                   tuneGrid = data.frame(k = seq(1,200,by=5)),
                   trControl = ctrl)
```

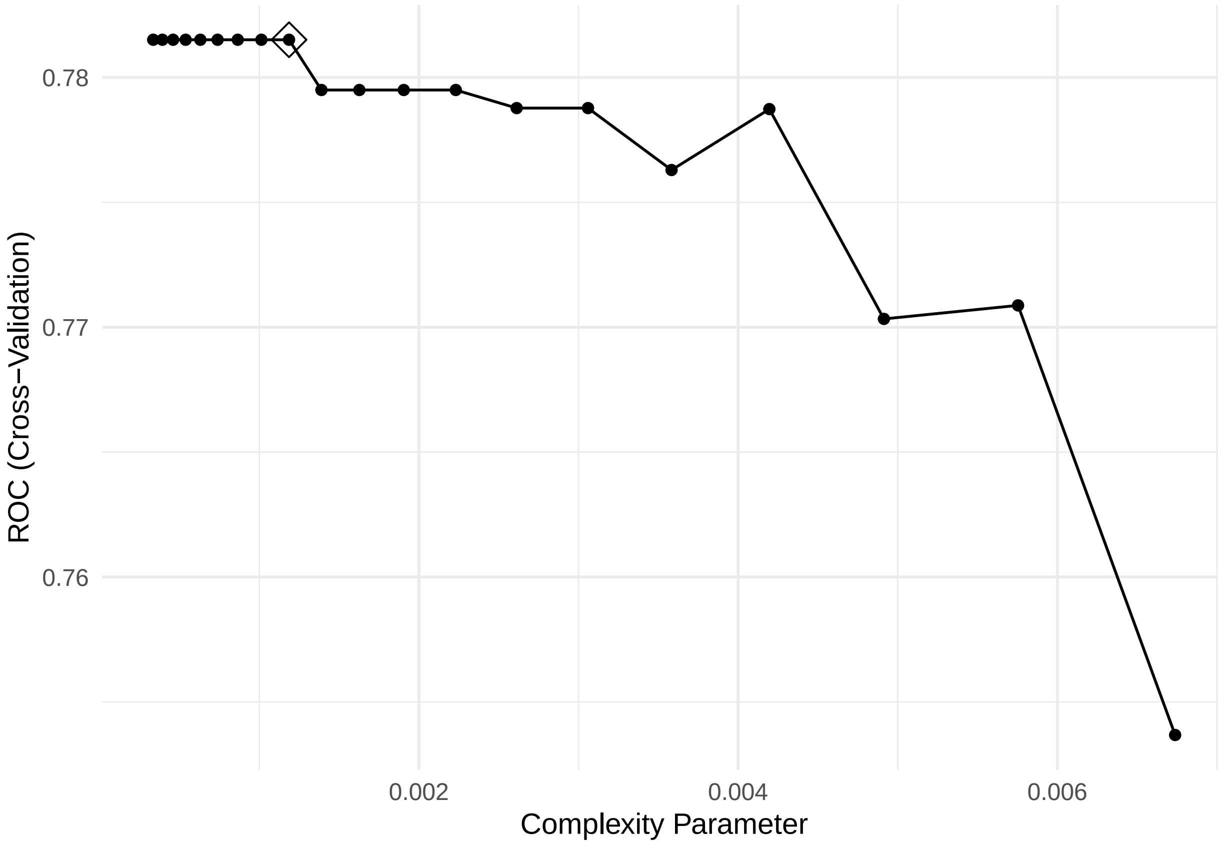
```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```r
ggplot(knn.fit)
```

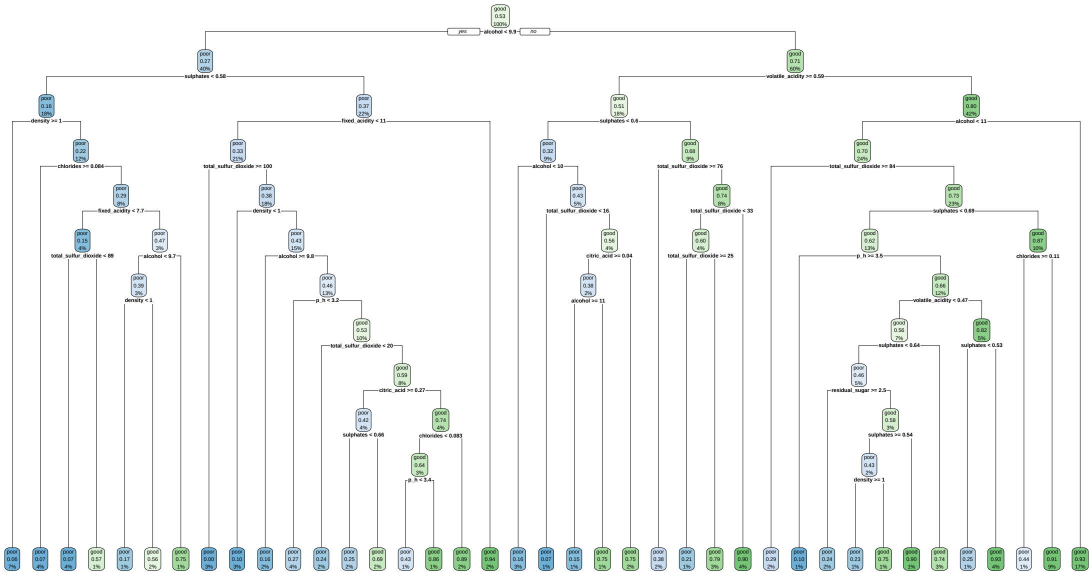


Classification tree

```r
set.seed(1)
rpart.fit <- train(quality~., df,
                   subset = rowTrain,
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-8,-5, len = 20))),
                   trControl = ctrl,
                   metric = "ROC")
ggplot(rpart.fit, highlight = TRUE)
```



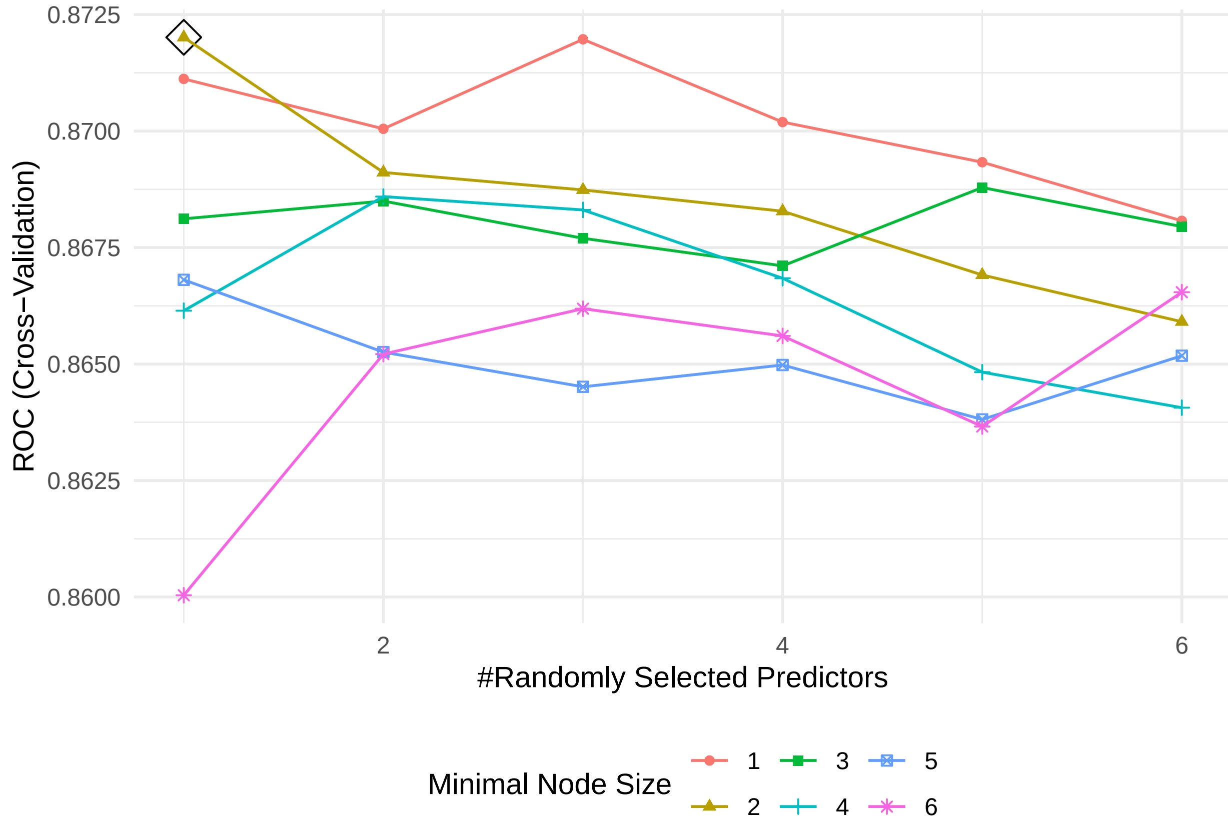

```r
rpart.plot(rpart.fit$finalModel)
```



Random forests

```
rf.grid <- expand.grid(mtry = 1:6,
                       splitrule = "gini",
                       min.node.size = 1:6)
set.seed(1)
rf.fit <- train(quality~., df,
                subset = rowTrain,
                method = "ranger",
                tuneGrid = rf.grid,
                metric = "ROC",
                trControl = ctrl)

ggplot(rf.fit, highlight = TRUE)
```

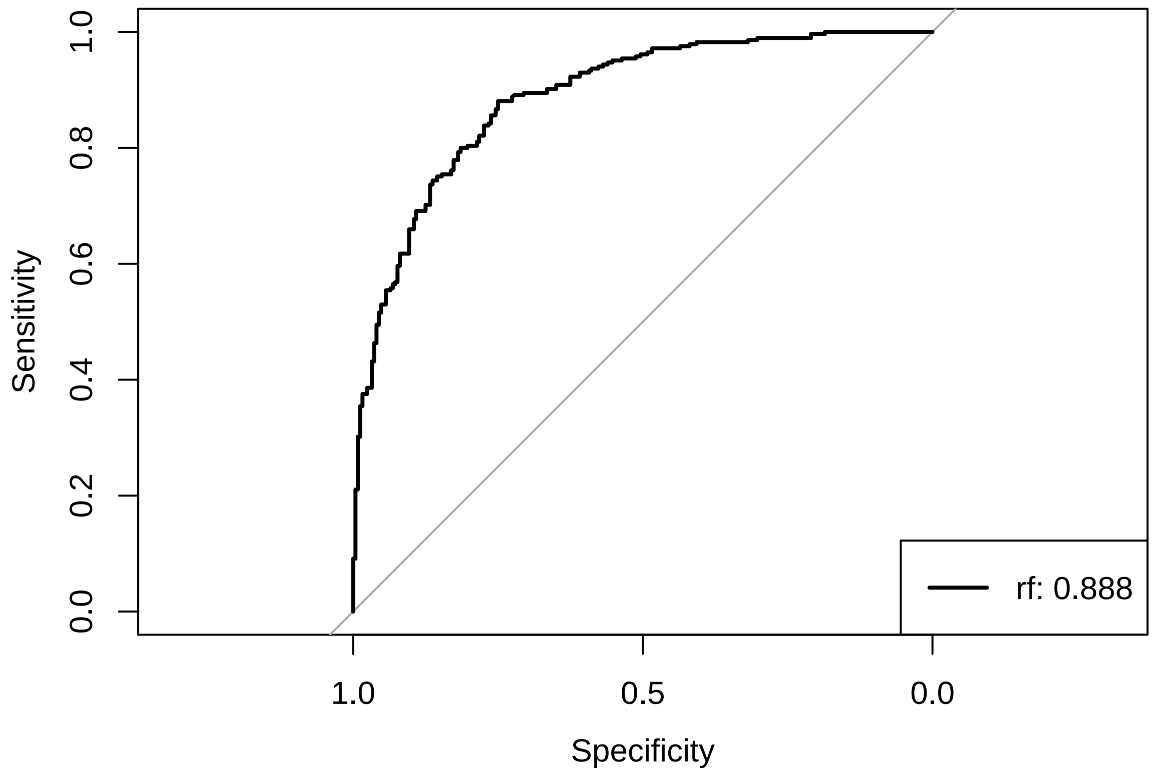



```
rf.pred <- predict(rf.fit, newdata = df[-rowTrain,], type = "prob")[,1]
roc.rf <- roc(df$quality[-rowTrain], rf.pred)
```

```
## Setting levels: control = poor, case = good
```

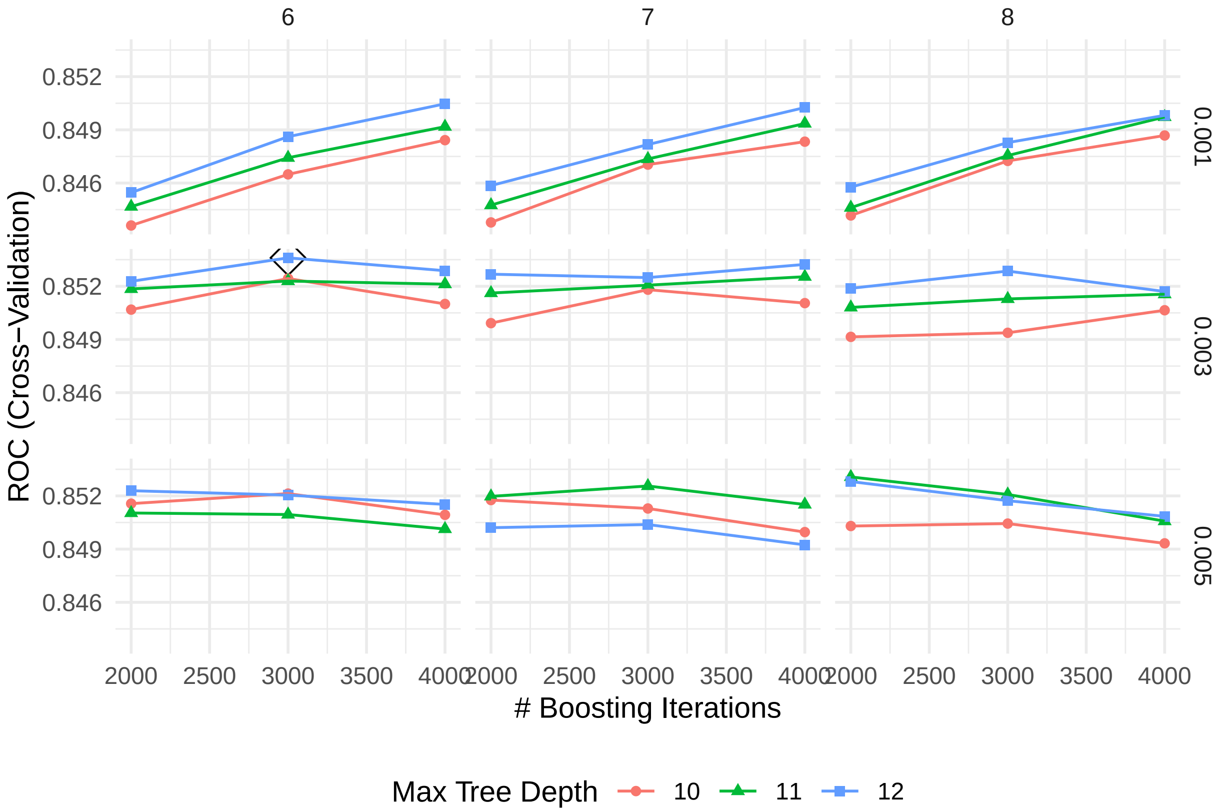
```
## Setting direction: controls > cases
```

```
plot(roc.rf)
auc <- roc.rf$auc[1]
modelNames <- "rf"
legend("bottomright", legend = paste0(modelNames, ": ", round(auc,3)),
       col = 1:6, lwd = 2)
```



boositng- AdaBoost

```
gbmA.grid <- expand.grid(n.trees = c(2000, 3000, 4000),
                         interaction.depth = 10:12,
                         shrinkage = c(0.001,0.003,0.005),
                         n.minobsinnode = 6:8)
set.seed(1)
# adaboost loss function
gbmA.fit <- train(quality~.,
                  df,
                   subset = rowTrain,
                 tuneGrid = gbmA.grid,
                 trControl = ctrl,
                 method = 'gbm',
                 distribution = "adaboost",
                 metric = "ROC",
                 verbose = FALSE)
ggplot(gbmA.fit, highlight = TRUE)
```




```
gbmA.fit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 47    3000                12     0.003              6
```

```
gbmA.pred <- predict(gbmA.fit, newdata = df[-rowTrain,], type = "prob")[,1]
```

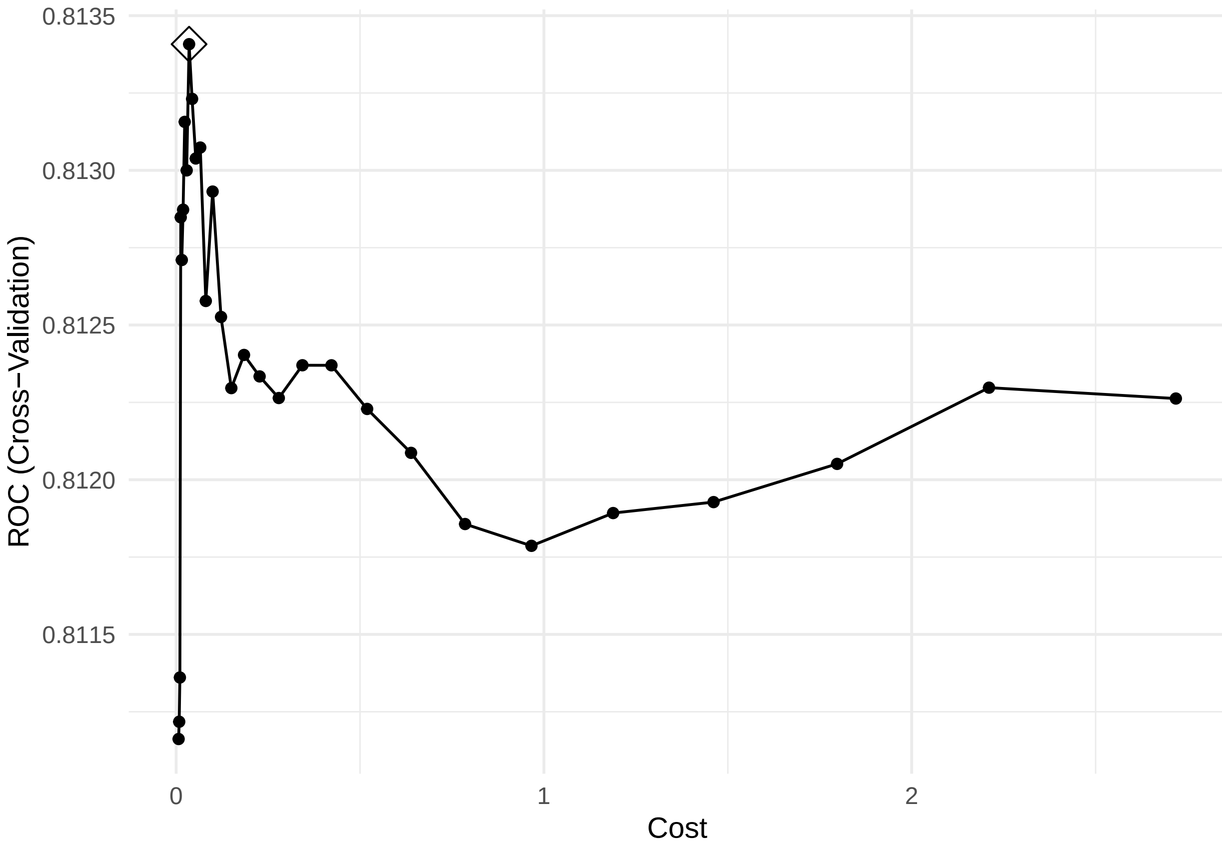
Support Vector Machine

Linear kernel

```
set.seed(1)
svmlinear.fit <- train(quality~.,
                  df,
                   subset = rowTrain,
                  method = "svmLinear2",
                  preProcess = c("center", "scale"),
                  tuneGrid = data.frame(cost = exp(seq(-5, 1, len=30))),
                  trControl = ctrl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```
ggplot(svmlinear.fit, highlight = TRUE)
```




```r
svmlinear.fit$bestTune
```

```
##        cost
## 9 0.0352663
```

```r
# train error
pred.svmlinear.train <- predict(svmlinear.fit$finalModel, data = df.train)
confusionMatrix(data = pred.svmlinear.train,
                reference = df$quality[rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction poor good
##       poor  379  156
##       good  117  414
##
##                Accuracy : 0.7439
##                  95% CI : (0.7166, 0.7699)
##     No Information Rate : 0.5347
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4879
##
##  Mcnemar's Test P-Value : 0.02146
##
##             Sensitivity : 0.7641
##             Specificity : 0.7263
```

```
##          Pos Pred Value : 0.7084
##          Neg Pred Value : 0.7797
##              Prevalence : 0.4653
##          Detection Rate : 0.3555
##    Detection Prevalence : 0.5019
##       Balanced Accuracy : 0.7452
##
##        'Positive' Class : poor
##
```

```
# test error
pred.svmlinear.test <- predict(svmlinear.fit, newdata = df.test)
confusionMatrix(data = pred.svmlinear.test,
                reference = df$quality[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction poor good
##       poor  192   83
##       good   56  202
##
##                Accuracy : 0.7392
##                  95% CI : (0.6997, 0.776)
##     No Information Rate : 0.5347
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4796
##
##  Mcnemar's Test P-Value : 0.02743
##
##             Sensitivity : 0.7742
##             Specificity : 0.7088
##          Pos Pred Value : 0.6982
##          Neg Pred Value : 0.7829
##              Prevalence : 0.4653
##          Detection Rate : 0.3602
##    Detection Prevalence : 0.5159
##       Balanced Accuracy : 0.7415
##
##        'Positive' Class : poor
##
```

Comment: By 10 fold cross validation using `train()` function from caret package, the best cost tuning parameter is 0.519. Then train misclassification error of this linear SVM on entire train dataset is 1-0.7458 = 0.2542. The test missclassification error is 1-0.743 = 0.257. The test error is slightly greater than train error, so our model seems to be a good fit.

Radial kernel

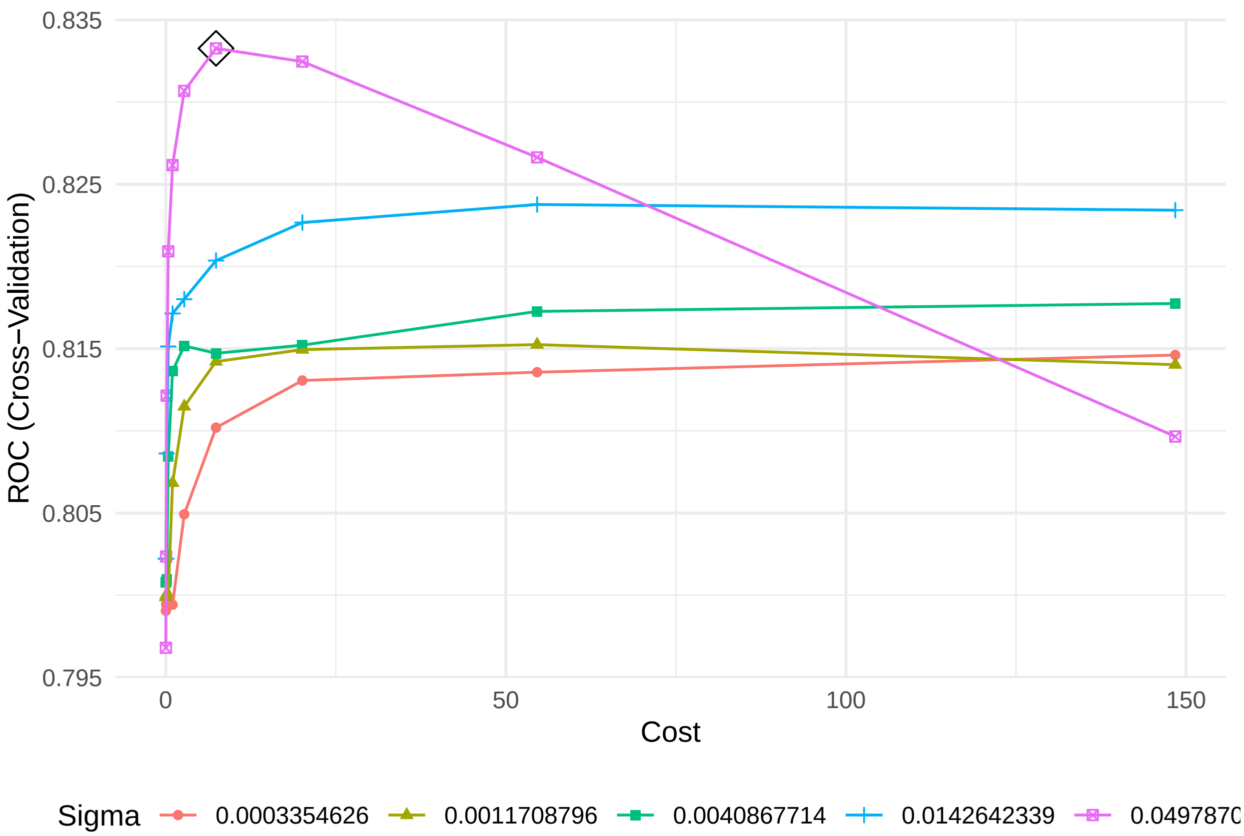
Different from the linear kernel, radial kernel can construct nonlinear classification boundaries.

```
svmr.grid <- expand.grid(C = exp(seq(-4, 5,len=10)),
                         sigma = exp(seq(-8,-3,len=5)))
set.seed(1)
svmradial.fit <- train(quality~.,
```

```r
                        data = df,
                        subset = rowTrain,
                        method = "svmRadial",
                        preProcess = c("center", "scale"),
                        tuneGrid = svmr.grid,
                        trControl = ctrl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```r
ggplot(svmradial.fit, highlight = TRUE)
```




```r
svmradial.fit$bestTune
```

```
##         sigma        C
## 35 0.04978707 7.389056
```

```r
# train error
pred.svmradial.train <- predict(svmradial.fit, newdata = df.train)
confusionMatrix(data = pred.svmradial.train,
                reference = df$quality[rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction poor good
##       poor  410  104
##       good   86  466
##
```

```
##                Accuracy : 0.8218
##                  95% CI : (0.7974, 0.8443)
##     No Information Rate : 0.5347
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6426
##
##  Mcnemar's Test P-Value : 0.2175
##
##             Sensitivity : 0.8266
##             Specificity : 0.8175
##          Pos Pred Value : 0.7977
##          Neg Pred Value : 0.8442
##              Prevalence : 0.4653
##          Detection Rate : 0.3846
##    Detection Prevalence : 0.4822
##       Balanced Accuracy : 0.8221
##
##        'Positive' Class : poor
##
```

```r
# test error
pred.svmradial.test <- predict(svmradial.fit, newdata = df.test)
confusionMatrix(data = pred.svmradial.test,
                reference = df$quality[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction poor good
##       poor  186   64
##       good   62  221
##
##                Accuracy : 0.7636
##                  95% CI : (0.7252, 0.7991)
##     No Information Rate : 0.5347
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5252
##
##  Mcnemar's Test P-Value : 0.929
##
##             Sensitivity : 0.7500
##             Specificity : 0.7754
##          Pos Pred Value : 0.7440
##          Neg Pred Value : 0.7809
##              Prevalence : 0.4653
##          Detection Rate : 0.3490
##    Detection Prevalence : 0.4690
##       Balanced Accuracy : 0.7627
##
##        'Positive' Class : poor
##
```

Comment: The best tuning parameter is sigma = 0.050, C = 54.598, the train error is 1 - 0.8565 = 0.1435, and the test error is 1- 0.7598 = 0.2402. Both the train and test errors are smaller than the linear kernel SVM.

Conclusion

performance

```
resamp <- resamples(list(rf = rf.fit,
                         knn = knn.fit,
                         lda = model.lda,
                         qda = model.qda,
                         rpart = rpart.fit,
                         boosting = gbmA.fit,
                         svmlinear = svmlinear.fit,
                         svmradinal = svmradial.fit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: rf, knn, lda, qda, rpart, boosting, svmlinear, svmradinal
## Number of resamples: 5
##
## ROC
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## rf         0.8511430 0.8597377 0.8725855 0.8720115 0.8806140 0.8959773
## knn        0.7887648 0.7935495 0.8002193 0.8078730 0.8144161 0.8424154
## lda        0.7956761 0.8103845 0.8139474 0.8133477 0.8155237 0.8312068
## qda        0.7770175 0.7815878 0.7918660 0.7875407 0.7918660 0.7953659
## rpart      0.7485823 0.7539474 0.7643098 0.7815089 0.8132642 0.8274411
## boosting   0.8256247 0.8500797 0.8608896 0.8536029 0.8613158 0.8701046
## svmlinear  0.7967393 0.8102073 0.8141060 0.8134078 0.8192982 0.8266879
## svmradinal 0.8070175 0.8228779 0.8313158 0.8332665 0.8431685 0.8619529
##            NA's
## rf            0
## knn           0
## lda           0
## qda           0
## rpart         0
## boosting      0
## svmlinear     0
## svmradinal    0
##
## Sens
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## rf         0.7373737 0.7400000 0.7676768 0.7722424 0.7979798 0.8181818
## knn        0.6500000 0.7070707 0.7171717 0.7097980 0.7171717 0.7575758
## lda        0.7272727 0.7300000 0.7474747 0.7561010 0.7878788 0.7878788
## qda        0.5959596 0.6262626 0.6600000 0.6552323 0.6868687 0.7070707
## rpart      0.5800000 0.6363636 0.6767677 0.6654949 0.7070707 0.7272727
## boosting   0.7070707 0.7300000 0.8080808 0.7722626 0.8080808 0.8080808
## svmlinear  0.7373737 0.7400000 0.7878788 0.7702222 0.7878788 0.7979798
```

```
## svmradinal 0.7300000 0.7373737 0.7575758 0.7500404 0.7575758 0.7676768
##            NA's
## rf            0
## knn           0
## lda           0
## qda           0
## rpart         0
## boosting      0
## svmlinear     0
## svmradinal    0
##
## Spec
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## rf         0.7807018 0.7807018 0.8070175 0.8210526 0.8596491 0.8771930
## knn        0.6929825 0.7280702 0.7631579 0.7561404 0.7807018 0.8157895
## lda        0.6315789 0.7280702 0.7368421 0.7350877 0.7807018 0.7982456
## qda        0.7456140 0.7543860 0.7719298 0.7894737 0.8245614 0.8508772
## rpart      0.7017544 0.7192982 0.7280702 0.7456140 0.7719298 0.8070175
## boosting   0.7719298 0.7894737 0.7982456 0.8105263 0.8245614 0.8684211
## svmlinear  0.6403509 0.7017544 0.7192982 0.7245614 0.7543860 0.8070175
## svmradinal 0.7192982 0.7368421 0.7456140 0.7666667 0.7982456 0.8333333
##            NA's
## rf            0
## knn           0
## lda           0
## qda           0
## rpart         0
## boosting      0
## svmlinear     0
## svmradinal    0
```