Support Vector Machines

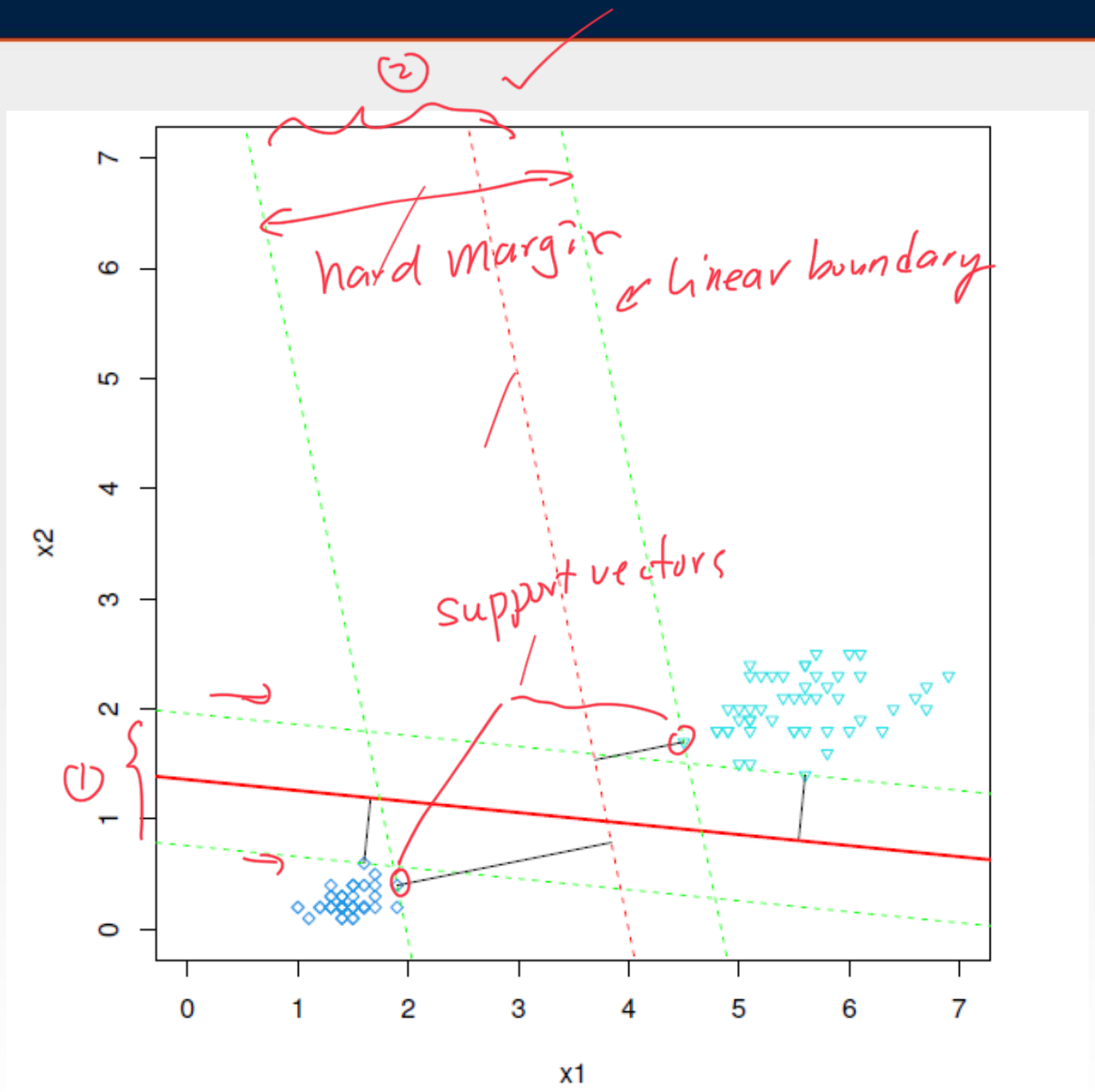
Chapter 13: Nonlinear Classification Models

Support vector machines

- Support vector machine was developed in the computer science community in the 1990s, and it has been considered one of the best "out of the box" classifiers. Specifically, we will learn
  - maximal margin classifier ✓ (hard margin and linear)
  - support vector classifier ✓ (soft margin and linear)
  - support vector machine ✓ (soft margin and non-linear)
- People often loosely refer to these approaches as "support vector machines." To avoid confusion, we will carefully distinguish between these three notions.

Maximal margin classifier

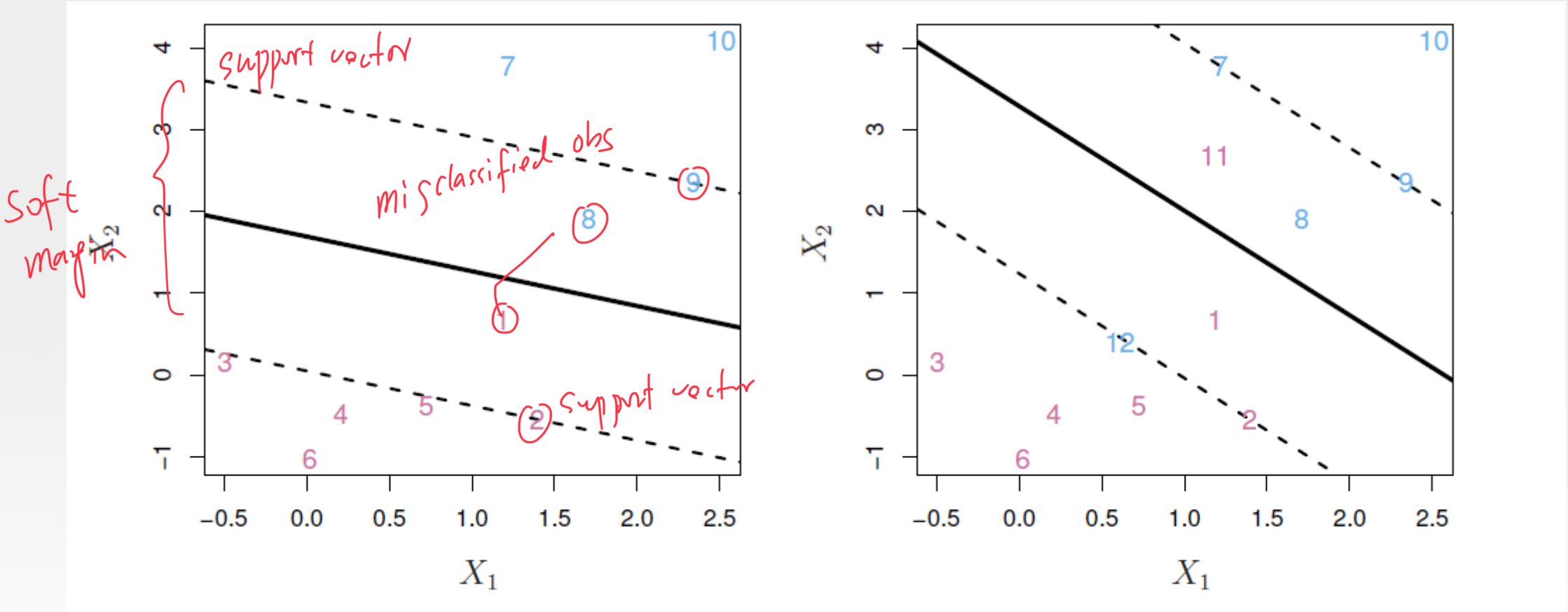
- Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.
- The two observations on the dashed lines are known as support vectors, since the maximal margin hyperplane depends directly on them, not one the other observations



(2) should be chosen.

There is no classification error
↓
(hard margin)

Support vector classifier



- The support vector classifier maximizes a *soft* margin, which tries to separate most of the training observations into the two classes, but may misclassify a few observations.

Remarks

- When we don't allow any samples to be inside the margin, we are talking about the hard margin.
- More often, we use soft margin classification, in which we allow certain samples to be inside the margin
  - By doing so, the overall fit of the model to the data might very well be better than with hard margin classification
  - It would reduce variance at the *cost* of some bias (i.e., the bias-variance trade-off)
- So, here is where the "Cost" is introduced
  - Purpose: modify the optimization problem to optimize both the fit of the line to data and penalizing the amount of samples inside the margin at the same time
  - C defines the weight of how much samples inside the margin contribute to the overall error: the low value of C allows more samples inside the margin; near 0 means every sample can be inside.

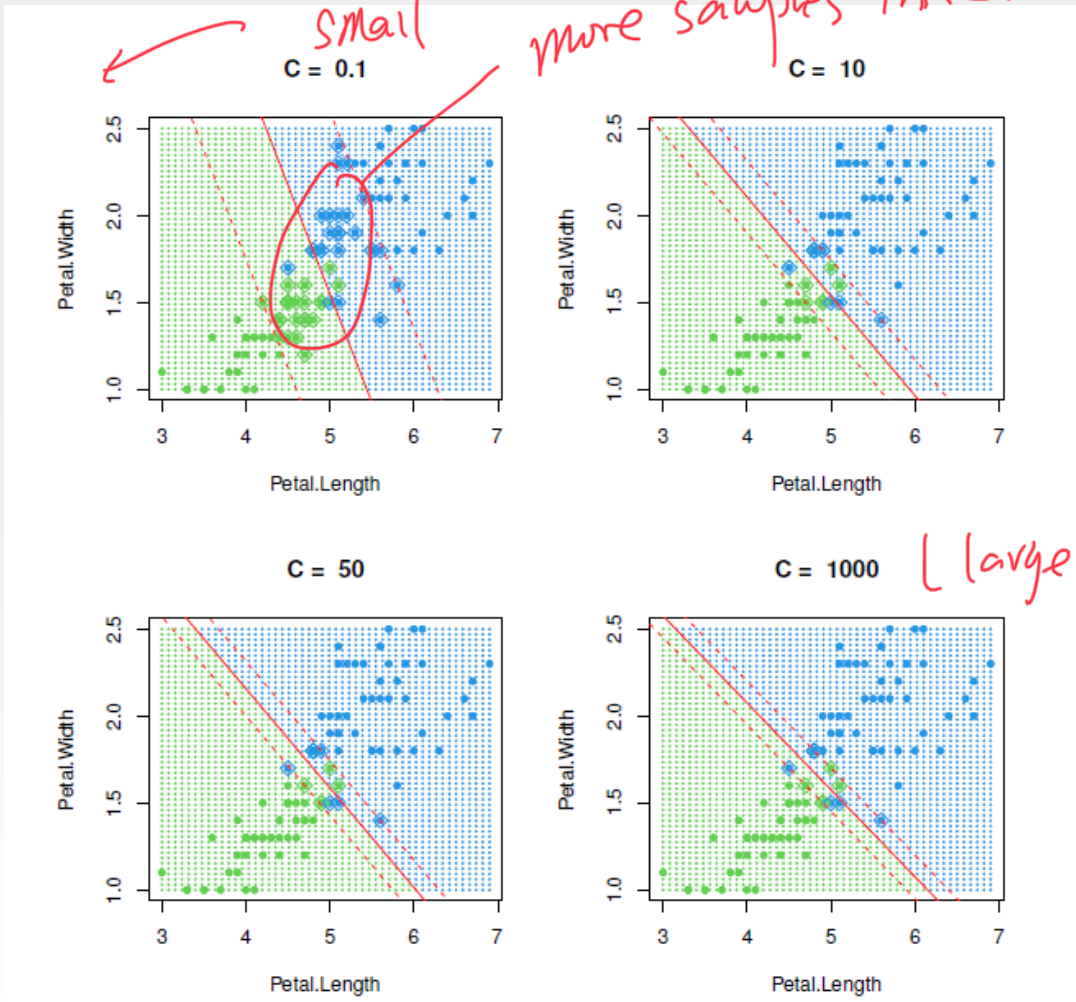
C is small. the margin will be wider

Tuning parameter of cost "C"

Margin are wider

small

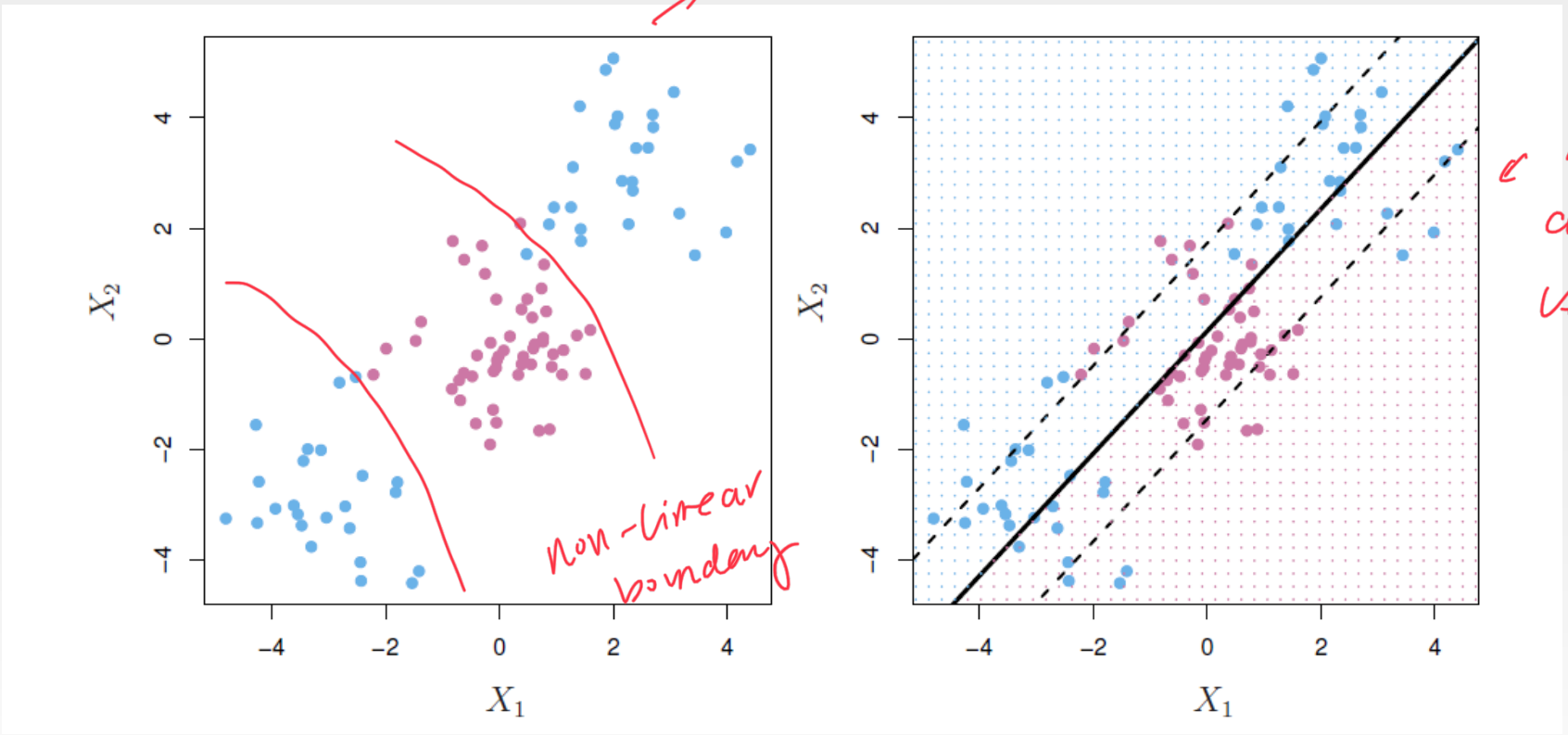
more samples inside of the margin, indicating that the misclassification becomes larger when C is small in the training data.



(larger)

margins are smaller

Linear boundary can fail



- In practice we are sometimes faced with non-linear class boundaries.
- It is clear that a support vector classifier or any linear classifier will perform poorly here.

Support vector machines (SVM)

- The maximal classifier and the support vector classifier considered *linear* classification boundaries.
- To extend the linear nature of the model to nonlinear classification boundaries, we consider kernel function instead of the simple linear relationship. There are three popular kernels,
  - Polynomial
  - Radial basis function → popular in practice
  - Hyperbolic tangent
- The kernel trick allows the SVM model produce extremely flexible decision boundaries.

Remarks on SVM

- The choice of the kernel function parameters and the cost value control the complexity and should be tuned appropriately so that the model does not over-fit the training data.
- Using resampling to find appropriate values of tuning parameters tends to find a reasonable balance between under- and over-fitting.
- From our experience, SVM models tend to be very competitive for most classification problems.

SVM

```
###################################Support Vector Machines#####################################
set.seed(476)
#Given a range of values for the "sigma" inverse width parameter in the Gaussian Radial Basis kernel for
#use with Support Vector Machine
sigmaRangeReduced <- sigest(as.matrix(Smarket.train[,1:8]))
svmRGridReduced <- expand.grid(.sigma = sigmaRangeReduced[1],
.C = 2^(seq(-4, 4)))
SVMTune <- train(x = as.matrix(Smarket.train[,1:8]),
            y = Smarket.train$Direction,
method = "svmRadial", #svmLinear #svmPoly
metric = "ROC",
preProc = c("center", "scale"),
tuneGrid = svmRGridReduced,
fit = FALSE,
trControl = ctrl)
SVMTune
```

SVM output

```
> SVMTune
Support Vector Machines with Radial Basis Function Kernel

998 samples
  8 predictor
  2 classes: 'Down', 'Up'

Pre-processing: centered (8), scaled (8)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
Resampling results across tuning parameters:

  C        ROC        Sens       Spec
   0.0625  0.9937809  0.9708197  0.9365079
   0.1250  0.9940437  0.9501639  0.9596825
   0.2500  0.9956648  0.9521311  0.9666667
   0.5000  0.9965574  0.9626230  0.9682540
   1.0000  0.9969035  0.9737705  0.9644444
   2.0000  0.9971663  0.9737705  0.9688889
   4.0000  0.9971298  0.9750820  0.9692063
   8.0000  0.9975176  0.9796721  0.9692063
  16.0000  0.9975826  0.9806557  0.9638095

Tuning parameter 'sigma' was held constant at a value of 0.03314474
ROC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.03314474 and C = 16.
```

fixed

Model comparison with ROC and Confusion Matrix

Chapter 13: Nonlinear Classification Models

Prediction based on different models

predictors from the test data

```
### Predict the test set based on eight models
#QDA
Smarket.test$QDA<- predict(QDATune,Smarket.test, type = "prob")[,1]
#RDA
Smarket.test$RDA <- predict(RDATune,Smarket.test, type = "prob")[,1]
#MDA
Smarket.test$MDA <- predict(MDATune,Smarket.test, type = "prob")[,1]
#NB
Smarket.test$NB <- predict(NBTune,Smarket.test, type = "prob")[,1]
#KNN
Smarket.test$KNN <- predict(KNNTune,Smarket.test, type = "prob")[,1]
#NN
Smarket.test$NN <- predict(NNTune,Smarket.test, type = "prob")[,1]
#FDA
Smarket.test$FDA <- predict(FDATune, Smarket.test, type = "prob")[,1]
#SVM
Smarket.test$SVM <- predict(SVMTune, Smarket.test[,1:8], type = "prob")[,1]
```

ROCs based on different models

response from the test data

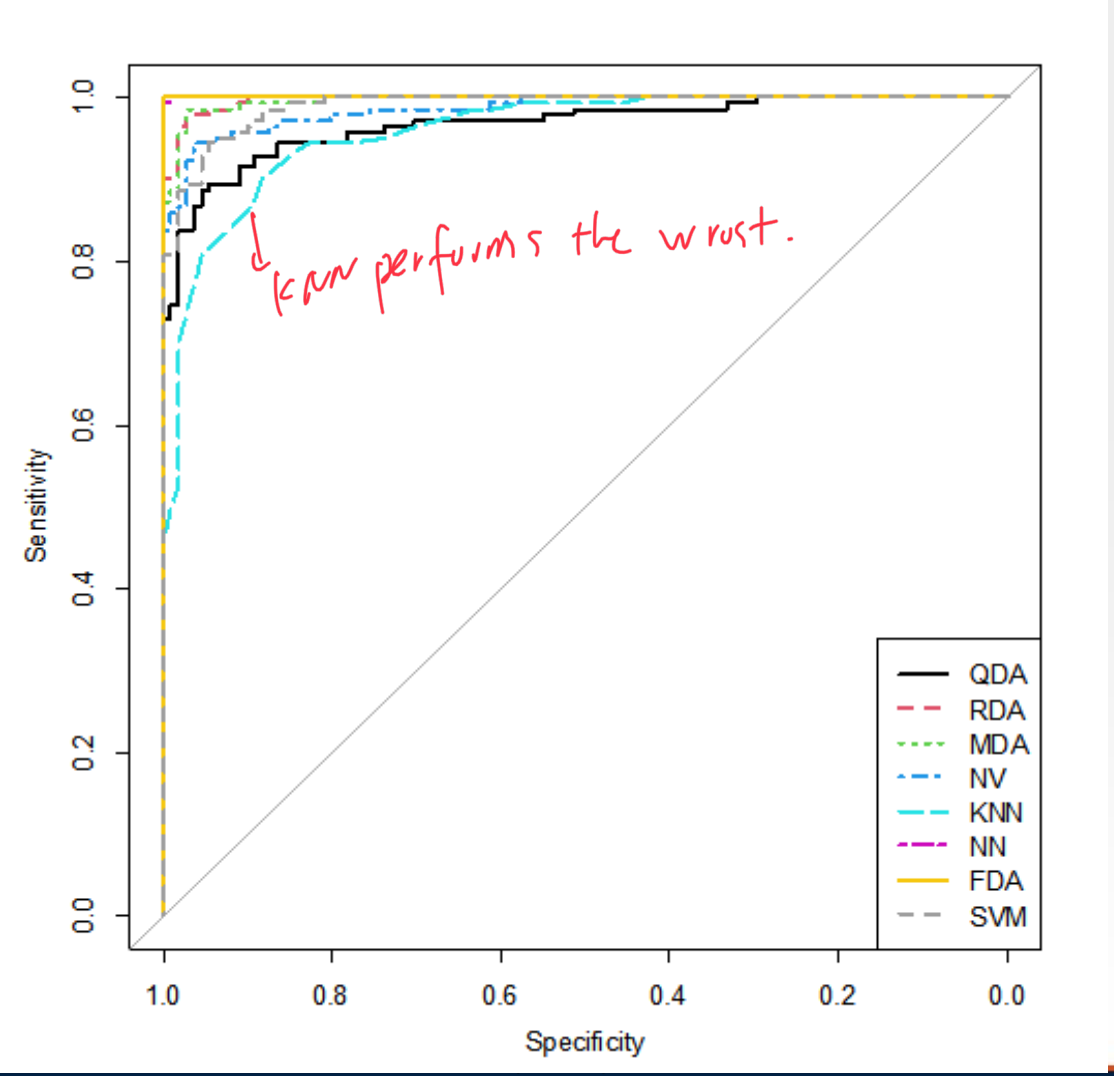
predicted response from QDA

```
#ROC for QDA
QDAROC <- roc(Smarket.test$Direction, Smarket.test$QDA)
plot(QDAROC, col=1, lty=1, lwd=2)

#ROC for RDA
RDAROC <- roc(Smarket.test$Direction, Smarket.test$RDA)
lines(RDAROC, col=2, lty=2, lwd=2)

#ROC for MDA
MDAROC <- roc(Smarket.test$Direction, Smarket.test$MDA)
lines(MDAROC, col=3, lty=3, lwd=2)

#ROC for NB
NBROC <- roc(Smarket.test$Direction, Smarket.test$NB)
lines(NBROC, col=4, lty=4, lwd=2)
```

ROCs based on different models

```
#ROC for KNN
KNNROC <- roc(Smarket.test$Direction, Smarket.test$KNN)
lines(KNNROC, col=5, lty=5, lwd=2)

#ROC for NN
NNROC <- roc(Smarket.test$Direction, Smarket.test$NN)
lines(NNROC, col=6, lty=6, lwd=2)

#ROC for FDA
FDAROC <- roc(Smarket.test$Direction, Smarket.test$FDA)
lines(FDAROC, col=7, lty=7, lwd=2)

#ROC for SVM
SVMROC <- roc(Smarket.test$Direction, Smarket.test$SVM)
lines(SVMROC, col=8, lty=8, lwd=2)

legend('bottomright', c('QDA','RDA','MDA','NV', 'KNN', 'NN', 'FDA','SVM'), col=1:8, lty=1:8,lwd=2)
```

ROCs based on different models



Confusion matrices of different models

response from the test data

```
#Confusion matrix of QDA
confusionMatrix(data = predict(QDATune, Smarket.test), reference = Smarket.test$Direction)

#Confusion Matrix of RDA
confusionMatrix(data = predict(RDATune, Smarket.test), reference = Smarket.test$Direction)

#Confusion matrix of MDA
confusionMatrix(data = predict(MDATune, Smarket.test), reference = Smarket.test$Direction)

#Confusion matrix of NB
confusionMatrix(data = predict(NBTune, Smarket.test), reference = Smarket.test$Direction)
```

Confusion matrices of different models

```
#Confusion matrix of KNN
confusionMatrix(data = predict(KNNTune, Smarket.test), reference = Smarket.test$Direction)

#Confusion matrix of NN
confusionMatrix(data = predict(NNTune, Smarket.test), reference = Smarket.test$Direction)

#Confusion matrix of FDA
confusionMatrix(data = predict(FDATune, Smarket.test), reference = Smarket.test$Direction)

#Confusion matrix of SVM
confusionMatrix(data = predict(SVMTune, Smarket.test[,1:8]), reference = Smarket.test$Direction)
```

Summary of output

→ Confusion matrix of the model

Model	Accuracy	Kappa	Sensitivity	Specificity
QDA	0.9127	0.8232	0.9099	0.9149
RDA	0.9524	0.9025	0.9009	0.9929
MDA	0.9683	0.9353	0. 9459	0.9858
NB	0.9444	0.8873	0.9369	0.9504
KNN	0.7460	0.4511	0.4234	1.0000
NN	0.9960	0.9920	1.0000	0.9929
FDA	0.9365	0.8692	0.8559	1.0000
SVM	0.9365	0.8709	0.9189	0.9504
Logistic	0.9960	0.9919	0.9910	1.0000
LDA	0.9524	0.9025	0.9009	0.9929
PLSDA	0.9405	0.8775	0.8649	1.0000
Penalized model	0.9841	0.9677	0.9640	1.0000
NSC	0.9524	0.9023	0.8919	1.0000

You may use t-test to see if Accuracy of all model is different at 5% significant level.

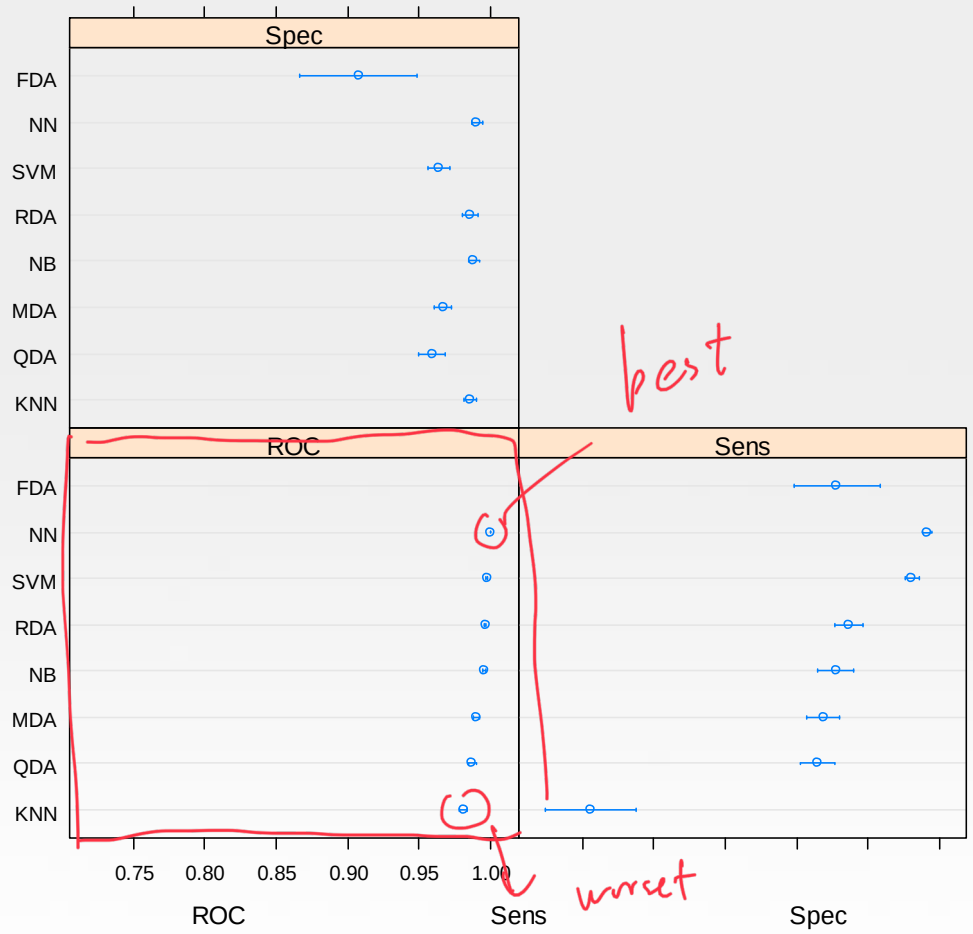
Model comparisons based on resamples of Training data

name of a method you specify.

```
#Resamples of Tranining data
res = resamples(list(QDA = QDATune, RDA =RDATune,MDA = MDATune, NB = NBTune, KNN = KNNTune,
NN = NNTune, FDA = FDATune, SVM = SVMTune ))
dotplot(res)

#We can add the models from chapters 12 and 13
res1 = resamples(list(Logistic = logisticTune, LDA =ldaTune,PLSDA = plsdaTune,
Penalized = glmnTune, NSC = nscTune,QDA = QDATune, RDA =RDATune,MDA = MDATune, NB = NBTune, KNN = KNNTune,
NN = NNTune, FDA = FDATune, SVM = SVMTune ))
dotplot(res1)
```

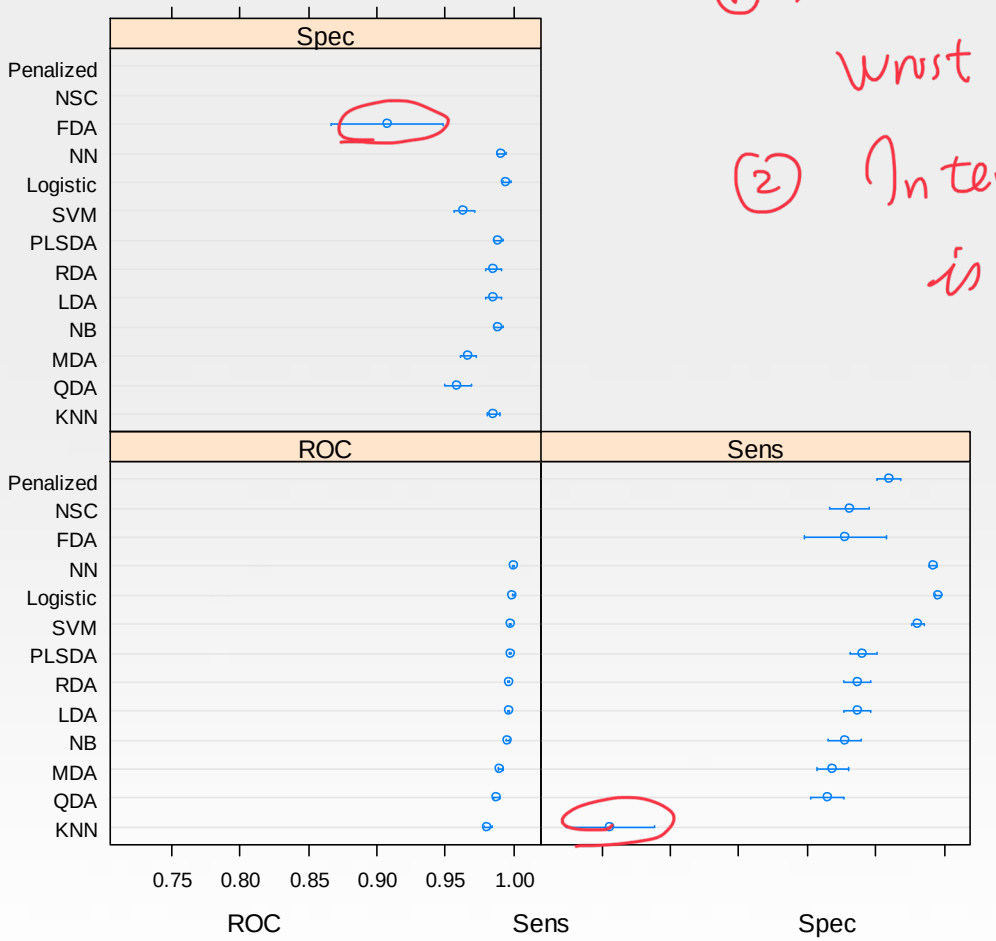
If we just have a training data and we don't want split our data into the traing & test data since the sample size is small, we can use 'resamples' to compare the models based on Training data.

Models from Chapter 13



Confidence Level: 0.95

Models from Chapters 12 and 13



① In terms of specificity, FDA is the worst

② In terms of Sensitivity, KNN is the worst.