Chapter 11

Chapter 11. Measuring Performance in Classification Models

RMSE and R2 are not appropriate in the context of classification

11.1 Class Predictions

Probability and predicted class

softmax transformation

Well-calibrated probabilities

calibration plot (if the points fall along a 45 degree line, the model has produced well-calibrated probabilities)

Presenting class probabilities

histogram and confusion matrix for two classes Heat map for three or more classes

Equivocal zones

An approach to improving classification performance is to create an equivocal or indeterminate zone where the class is not formally predicted when the confidence is not high

11.2 Evaluating predicted classes

Confusion matrix

Kappa statistic (Cohen's Kappa) = (O - E) / (1 - E)

Weighted kappa

Two-class problems

Sensitivity = # samples with the event and predicted to have the event / # samples having the event

Specificity = # samples without the event and predicted as nonevents / # samples without the event

Trade-offs between sensitivity and specificity

ROC - receiver operating characteristic curve is a technique for evaluating the trade-off

Youden's J index, J = sensitivity + specificity - 1

Sensitivity and specificity are conditional measures

Unconditional evaluations: positive predicted value (PPV) and negative predicted value (NPV)

PPV = sensitivity * prevalence / (sensitivity * prevalence) + ((1-specificity) * (1-prevalence))

NPV = specificity * (1-prevalence) / (prevalence * (1-sensitivity)) + (specificity * (1-prevalence))

Predictive values are not often used to characterize the model, the reasons: First, prevalence is hard to quantify. Also the prevalence is dynamic.

Non-accurarcy-based criteria

Also consider the metrics that quantify the consequences of correct and incorrect predictions (i.e., the benefits and costs)

Profit calculation also needs to consider prevalence

Probability-cost function (PCF) = P * C(+|-) / P * C(-|+) + (1-P) * C(+|-)

Normalized expected cost (NEC) = PCF * (1-TP) + (1 - PCF) * FP

11.3 Evaluating class probabilities

Receiver operating characteristic (ROC) curves

given a collection of continuous data points, determine an effective threshold such that values above the threshold are indicative of a specific event

The ROC curve is created by evaluating the class probabilities for the model across a continum of thresholds. For each candidate threshold, the resulting true-positive rate (sensitivity) and the false-positive rate (1-specificity) are plotted against each other

In the confusion matrix, it cannot move samples out of both off-diagonal table cells. There is almost always a decrease in either sensitivity or specificity

Comparing ROC curves can be useful in contrasting two or more models with different predictor sets (for the same model), different tuning parameters (within model comparisons), or complete different classifiers (between models)

One advantage of using ROC curves to characterize models is that, since it is a function of sensitivity and specificity, the curve is insensitive to disparities in the class proportions

The ROC curve is only defined for two-class problems but has been extended to handle three or more classed

Lift charts

Rank the samples by their scores and determine the cumulatie event rate as more samples are evaluated

The lift is the number of samples detected by a model above a completely random selection of samples

The lift chart plots the cumulative gain/lift against the cumulative percentage of samples that have been screened

11.4 Computing

```
library(AppliedPredictiveModeling)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(klaR)
```

```
## Loading required package: MASS
```

```
library(MASS)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(975)
simulatedTrain <- quadBoundaryFunc(500)
simulatedTest <- quadBoundaryFunc(1000)
head(simulatedTrain)
```

```
##           X1          X2      prob  class
## 1  2.4685709  2.28742015 0.9647251 Class1
## 2 -0.1889407 -1.63949455 0.9913938 Class1
## 3 -1.9101460 -2.89194964 1.0000000 Class1
## 4  0.3481279  0.06707434 0.1529697 Class1
## 5  0.1401153  0.86900555 0.5563062 Class1
## 6  0.7717148 -0.91504835 0.2713248 Class2
```

```
rfModel <- randomForest(class ~ X1 + X2, data = simulatedTrain, ntree = 2000)
rfModel
```

```
##
## Call:
##  randomForest(formula = class ~ X1 + X2, data = simulatedTrain,      ntree = 2000)
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 1
##
```

```
##         OOB estimate of  error rate: 15.4%
## Confusion matrix:
##        Class1 Class2 class.error
## Class1    174     42   0.1944444
## Class2     35    249   0.1232394
```

```
qdaModel <- qda(class ~ X1 + X2, data = simulatedTrain)
qdaModel
```

```
## Call:
## qda(class ~ X1 + X2, data = simulatedTrain)
##
## Prior probabilities of groups:
## Class1 Class2
##  0.432  0.568
##
## Group means:
##                X1          X2
## Class1 0.5881427 -0.05429687
## Class2 1.2342071  0.11753625
```

```
qdaTrainPred <- predict(qdaModel, simulatedTrain)
names(qdaTrainPred)
```

```
## [1] "class"     "posterior"
```

```
head(qdaTrainPred$class)
```

```
## [1] Class1 Class1 Class1 Class2 Class1 Class2
## Levels: Class1 Class2
```

```
head(qdaTrainPred$posterior)
```

```
##      Class1      Class2
## 1 0.7313136 0.268686374
## 2 0.8083861 0.191613899
## 3 0.9985019 0.001498068
## 4 0.3549247 0.645075330
## 5 0.5264952 0.473504846
## 6 0.3604055 0.639594534
```

```
qdaTestPred <- predict(qdaModel, simulatedTest)
simulatedTrain$QDAprob <- qdaTrainPred$posterior[,"Class1"]
simulatedTest$QDAprob <- qdaTestPred$posterior[,"Class1"]
```

```
rfTestPred <- predict(rfModel, simulatedTest, type = "prob")
head(rfTestPred)
```

```
##   Class1 Class2
## 1 0.4440 0.5560
```

```
## 2 0.5355 0.4645
## 3 0.9935 0.0065
## 4 0.9330 0.0670
## 5 0.0170 0.9830
## 6 0.2840 0.7160
```

```
simulatedTest$RFprob <- rfTestPred[,"Class1"]
simulatedTest$RFclass <- predict(rfModel, simulatedTest)
```

```
head(simulatedTest)
```

```
##           X1         X2       prob  class   QDAprob RFprob RFclass
## 1  0.6689362 -0.9113505 0.31729132 Class2 0.3830767 0.4440  Class2
## 2 -0.1149877 -0.8272760 0.64476405 Class1 0.5440393 0.5355  Class1
## 3  1.2704375  2.9100370 0.99999790 Class1 0.9846107 0.9935  Class1
## 4 -0.5511148 -0.2145599 0.53333948 Class2 0.5463540 0.9330  Class1
## 5  0.9566129 -0.3315233 0.05333099 Class2 0.2426705 0.0170  Class2
## 6 -0.2132386  0.2364065 0.38441722 Class2 0.4823296 0.2840  Class2
```

Sensitivity and specificity

```
# Class 1 will be used as the event of interest
sensitivity(data = simulatedTest$RFclass, reference = simulatedTest$class, positive = "Class1")
```

```
## [1] 0.8257081
```

```
specificity(data = simulatedTest$RFclass, reference = simulatedTest$class, negative = "Class2")
```

```
## [1] 0.8983364
```

```
posPredValue(data = simulatedTest$RFclass, reference = simulatedTest$class, positive = "Class1")
```

```
## [1] 0.8732719
```

```
negPredValue(data = simulatedTest$RFclass, reference = simulatedTest$class, negative = "Class2")
```

```
## [1] 0.8586572
```

```
# Change the prevalence manually
posPredValue(data = simulatedTest$RFclass, reference = simulatedTest$class, positive = "Class1", preval
```

```
## [1] 0.9865043
```

confusion matrix

```r
confusionMatrix(data = simulatedTest$RFclass, reference = simulatedTest$class, positive = "Class1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class1 Class2
##     Class1    379     55
##     Class2     80    486
##
##                Accuracy : 0.865
##                  95% CI : (0.8422, 0.8856)
##     No Information Rate : 0.541
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.727
##
##  Mcnemar's Test P-Value : 0.03887
##
##             Sensitivity : 0.8257
##             Specificity : 0.8983
##          Pos Pred Value : 0.8733
##          Neg Pred Value : 0.8587
##              Prevalence : 0.4590
##          Detection Rate : 0.3790
##    Detection Prevalence : 0.4340
##       Balanced Accuracy : 0.8620
##
##        'Positive' Class : Class1
##
```

Receiver operating characteristic curves

```r
rocCurve <- roc(response = simulatedTest$class, predictor = simulatedTest$RFprob,
                # This function assumes that the second class is the event of interest, so we reverse t
                levels = rev(levels(simulatedTest$class)))
```

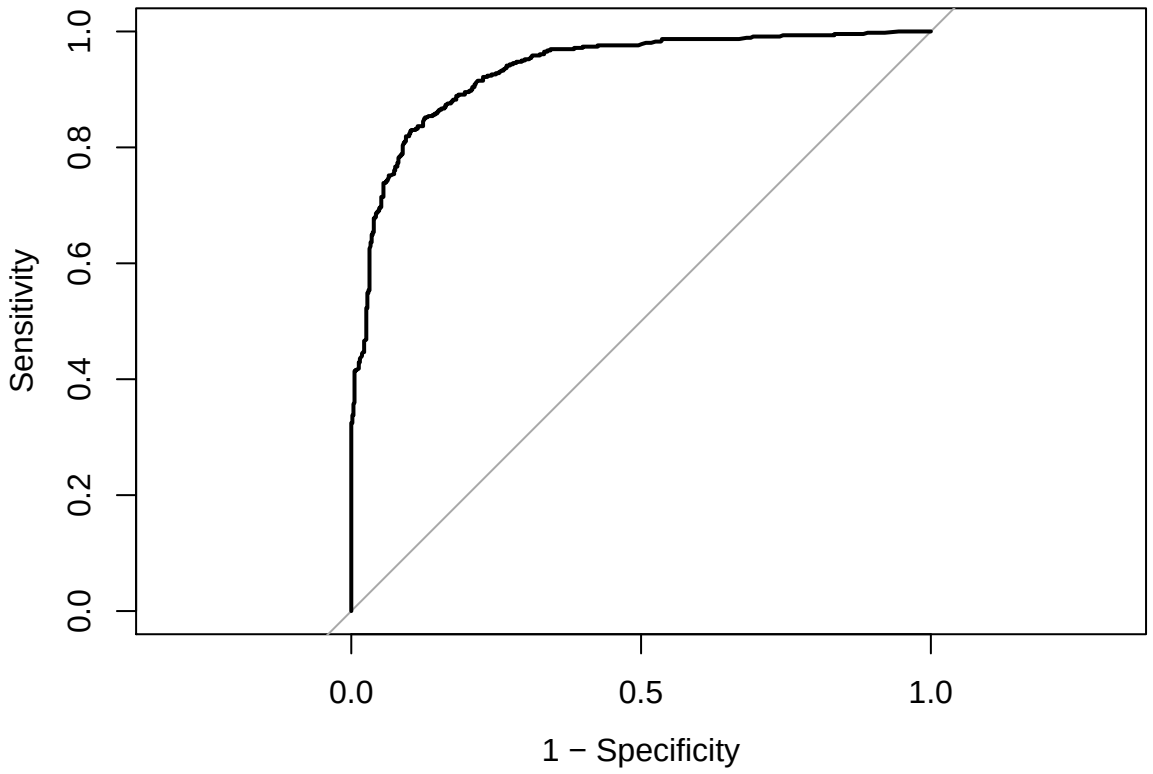
```r
auc(rocCurve)
```

```
## Area under the curve: 0.9335
```

```r
ci.auc(rocCurve)
```

```
## 95% CI: 0.9185-0.9486 (DeLong)
```

```r
plot(rocCurve, legacy.axes = TRUE)
```



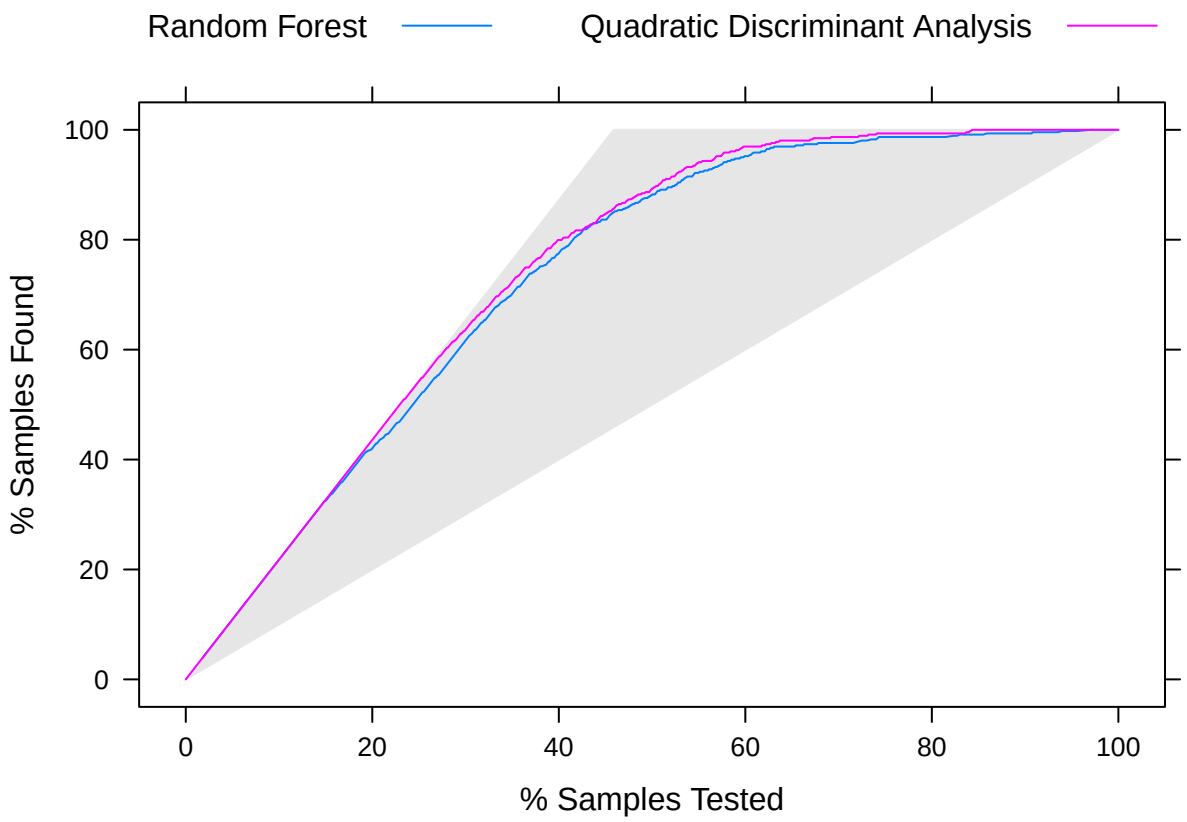

```
# By default, the x-axis goes backwards, used the option legacy.axes = TRUE to get 1-spe on the x-axis
# Also, another curve can be added using add=TRUE the next time plot.auc is used
```

Lift charts

```
labs <- c(RFprob="Random Forest", QDAprob = "Quadratic Discriminant Analysis")
liftCurve <- lift(class ~ RFprob + QDAprob, data = simulatedTest, labels = labs)
liftCurve
```

```
##
## Call:
## lift.formula(x = class ~ RFprob + QDAprob, data = simulatedTest, labels
##  = labs)
##
## Models: Random Forest, Quadratic Discriminant Analysis
## Event: Class1 (45.9%)
```

```
# Add lattice options to produce a legend on top
xyplot(liftCurve,
       auto.key = list(columns = 2,
                       lines = TRUE,
                       points = FALSE))
```

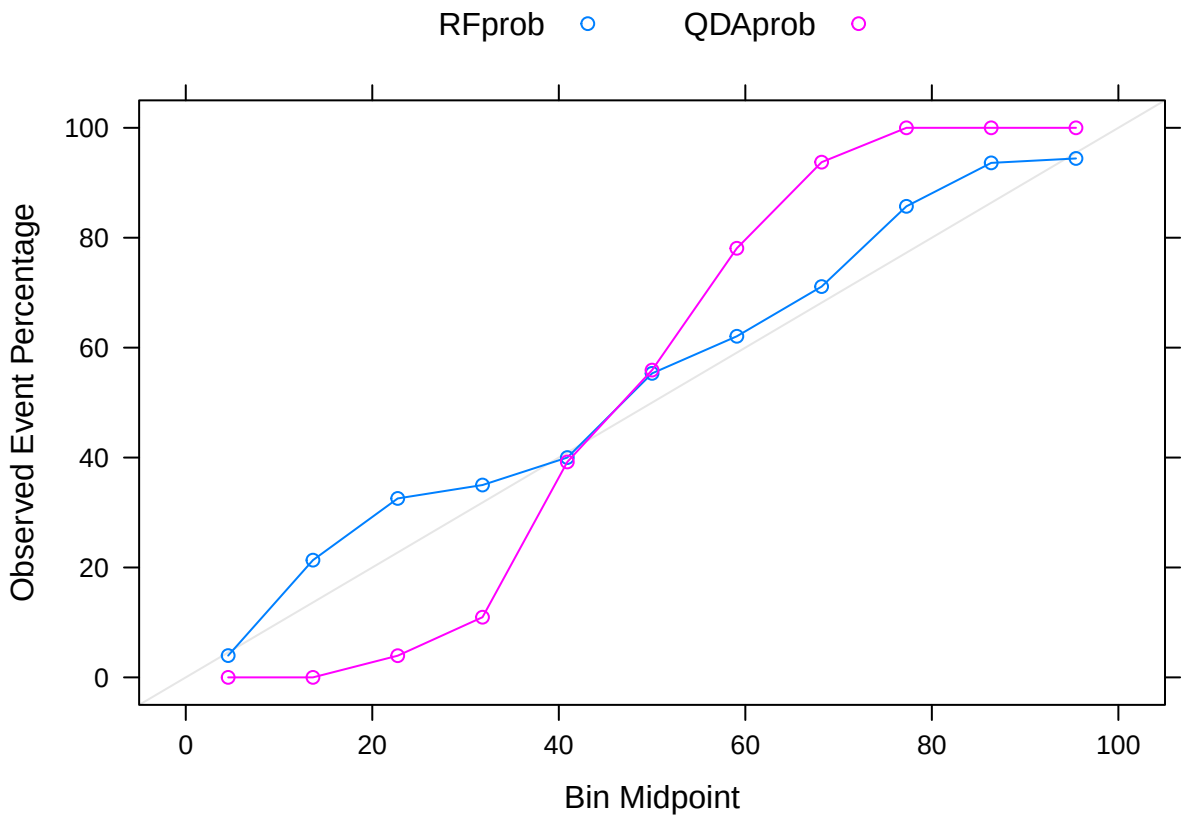


Calibrating probabilities

```
calCurve <- calibration(class ~ RFprob + QDAprob, data = simulatedTest)
calCurve
```

```
##
## Call:
## calibration.formula(x = class ~ RFprob + QDAprob, data = simulatedTest)
##
## Models: RFprob, QDAprob
## Event:  Class1
## Cuts: 11
```

```
xyplot(calCurve, auto.key=list(columns = 2))
```




```
# The glm() function models the probabilities of the second factor level, so the function revevel() is
sigmoidalCal <- glm(relevel(class, ref = "Class2") ~ QDAprob, data = simulatedTrain, family = binomial)
coef(summary(sigmoidalCal))
```

```
##               Estimate Std. Error   z value     Pr(>|z|)
## (Intercept) -5.701055  0.5005652 -11.38924 4.731132e-30
## QDAprob     11.717292  1.0705197  10.94542 6.989017e-28
```

```
sigmoidProbs <- predict(sigmoidalCal, newdata = simulatedTest[,"QDAprob", drop = FALSE], type = "respons
simulatedTest$QDAsigmoid <- sigmoidProbs
```

```
BayesCal <- NaiveBayes(class ~ QDAprob, data = simulatedTrain, usekernel = TRUE)
# Like qda(), the predict function for this model creates both the classes and the probabilities

BayesProbs <- predict(BayesCal, newdata = simulatedTest[,"QDAprob", drop = FALSE])

simulatedTest$QDABayes <- BayesProbs$posterior[,"Class1"]

# The probability values before and after calibration

head(simulatedTest[,c(5:6,8,9)])
```

```
##     QDAprob RFprob QDAsigmoid  QDABayes
## 1 0.3830767 0.4440 0.22927068 0.2515696
## 2 0.5440393 0.5355 0.66231139 0.6383383
```

```
## 3 0.9846107 0.9935 0.99708776 0.9995061
## 4 0.5463540 0.9330 0.66835048 0.6430232
## 5 0.2426705 0.0170 0.05428903 0.0566883
## 6 0.4823296 0.2840 0.48763794 0.5109129
```

```
calCurve2 <- calibration(class ~ QDAprob + QDABayes + QDAsigmoid, data = simulatedTest)
xyplot(calCurve2)
```