Module 13. Classification Methods

Overview:

In this module we will study classification methods, which are used to predict a categorical response variable Y from independent variable X.

We first introduce logistic regression, which is a generalized linear model. Then we will review how to evaluate classification methods. The evaluation methods, particularly cross-validation, are also used to evaluate two additional classification methods; support vector machine and classification tree.

By the end of this module, you should be able to carry out classification in R using logistic regression, support vector machine and classification tree. You should be able to read and interpret the R outputs, and be able to assess the classification performance

Learning Objectives

1. Apply **logistic regression** for classification
2. **Evaluate** classification methods by **ROC curve** and **validation**
3. Fit and use **support vector machine (SVM)** for classification
4. Fit and use classification **tree** for classification.

Readings:

Statistics Using R with Biological Examples pages 301-310.

Applied Statistics for Bioinformatics using R pages 145-172.

Lesson 1: Classification and Logistic Regression

Objectives

By the end of this lesson you will have had the opportunity to:

- Fit a **logistic regression** model in R, and **classify** using the predicted probability
- Carry out **statistical inferences** for simple logistic regression: parameter estimation, confidence intervals and hypothesis test

Overview

In this lesson, we introduce the logistic regression model. We write the model as a generalized linear model. Under this structure, we can conduct many statistical inferences similar to the linear regression model.

The classification is done based on predicted class probabilities from the logistic regression. We demonstrate the simple and multiple logistic regression classification on the iris data set in R.

Classification Methods

In medical settings, groups of patients are often diagnosed into classes corresponding to types of diseases. In bioinformatics, the question arises whether the diagnosis of a patient can be predicted by gene expression values. Related is the question of which genes play an important role in the prediction of class membership.

These types of questions are considered a problem of classification: predicting a factor (class) variable Y using independent variable X. Looking at it this way, the problem is very similar to the regression analysis except for one crucial aspect: Y is a continuous response variable in regression, but Y is a factor variable in classification. That is, now Y takes values only in a fixed set of classes.

It helps to look at the following examples of classifications:

1. First, the doctors provide different treatments for leukemia patients according to the types of leukemia. Y takes values of ALL or AML in such an application.
2. Second, biologists want to classify living things into species. For iris flowers, Y can take values as the species of iris.
3. Third, an important concern for bankers is if a company will be in default (bankrupt) next year. For that application, Y takes values as either default or solvent.

We first look at using logistic regression for classification. Many of the statistical inference concepts can be used for logistic regression.

Logistic Regression for Two Classes

We first consider the simple case that the response variable Y has only two outcomes. Such examples could include: ALL/AML for leukemia patients, default/solvent for companies, rich/poor for countries, etc. Mathematically, it is convenient to denote the two classes as 0 and 1.

With an independent variable X, the logistic regression models the relationship between $p(X) = P(Y = 1 | X)$ and X as

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} . \qquad (1)$$

First, notice that we only need to specify the relationship for $p(X) = P(Y = 1 | X)$. Since there are only two classes, $P(Y = 0 | X) = 1 - p(X)$. So $p(X)$ contains all information about distribution of Y given X.

Second, notice that we do not directly apply a linear model $p(X) = \beta_0 + \beta_1 X$. The linear function $\beta_0 + \beta_1 X$ can take negative values or values greater than one, which are not valid values for probabilities. Rather we apply the linear relationship with a logistic function $f(X) = \frac{e^x}{1 + e^x}$ transformation, which ensures the values are always between 0 and 1.

Third, using simple algebra, the logistic regression has an equivalent form

$$\log[\frac{p(X)}{1 - p(X)}] = \beta_0 + \beta_1 X . \qquad (2)$$

The left-hand side is the logit of p(X). That is, we have the linear regression model after applying a known *link function* (logit function here). This type of model is naturally called the *generalized linear model*, and is fitted in R by function glm().

We will now illustrate the fitting of logistic regression on the iris data set.

Demonstration of Logistic Regression on Iris Data Set

Previously, we have used the iris data set that comes with R installation. It contains five variables: sepal length, sepal width, petal length, petal width, and species of the flowers.

The last variable takes values as three species (classes): setosa, virginica and versicolor. Here we consider the identification of the setosa flowers. That is, our response variable Y has two values: Y=1 for setosa flowers, and Y=0 for non setosa flowers. We will use the first variable "sepal length" as our X. The logistic regression is fitted by the following R commands:

```
y<-as.numeric(iris$Species=='setosa') ## Y=1 for setosa; Y=0 otherwise
SepalLength<-iris$Sepal.Length ##Get first variable as X
reg.lgr <- glm(y~SepalLength, family=binomial(link='logit'))#Fit the generalized linear model, for binomial (0/1 responses) with logit link. That means logistic regression.
```

We can see the fitted solution as

```
> reg.lgr ##Display logistic regression fit
Call:  glm(formula = y ~ SepalLength, family = binomial(link = "logit"))
Coefficients:
(Intercept)  SepalLength
     27.829       -5.176

Degrees of Freedom: 149 Total (i.e. Null);  148 Residual
Null Deviance:      191
Residual Deviance: 71.84        AIC: 75.84
```

We see the estimated coefficients. Therefore

$$\log[\frac{P(Y=1|\text{SepalLength})}{1-P(Y=1|\text{SepalLength})}] = 27.829 - 5.176*\text{SepalLength}\,.$$

Or equivalently, $P(Y=1|\text{SepalLength}) = \frac{e^{27.829-5.176*\text{SepalLength}}}{1+e^{27.829-5.176*\text{SepalLength}}}$.

This provides a prediction probability for being setosa given a flower's sepal length.

Next, we study the statistical inferences of the logistics regression in more details.

Point estimators of Logistic Regression

As we learned in previous modules, the basic statistical inferences include point estimations, confidence intervals and hypothesis tests. We first look at the point estimator for the parameters β_0 and β_1 in the logistic regression model

$$p(X)=\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

If we follow the general principle of maximum likelihood, the standard method is to use maximum likelihood estimators (MLE). For a random sample $(X_1,Y_1),...,(X_n,Y_n)$, the probability of observing Y_i is $p(X_i)$ when $Y_i=1$ and $1-p(X_i)$ when $Y_i=0$. These two expressions can be combined as $[p(X_i)]^{Y_i}[1-p(X_i)]^{1-Y_i}$ by noticing that, for each value of 1 or 0, only one factor $p(X_i)$ or $1-p(X_i)$ appears. Therefore the likelihood is

$$L(\beta_0,\beta_1)=\prod_{i=1}^{n}[p(X_i)]^{Y_i}[1-p(X_i)]^{1-Y_i}=\prod_{i=1}^{n}[\frac{e^{\beta_0+\beta_1 X_i}}{1+e^{\beta_0+\beta_1 X_i}}]^{Y_i}[\frac{1}{1+e^{\beta_0+\beta_1 X_i}}]^{1-Y_i}$$

We can program the estimation of MLE ourselves as before.

```
den<-function(x, y, beta) {#the probability density function
   lin.x<-beta[1]+beta[2]*x            #linear part beta0+beta1*x
   (exp(lin.x)/(1+exp(lin.x)))^y*(1/(1+exp(lin.x)))^(1-y) #logistic
transformation
}
nlk<-function(x,y,beta)  -sum(log(den(x,y,beta))) #negative log-likelihood.
Using logarithm, the product of density() becomes the sum of log(density()).
```

We can then minimize the negative log-likelihood numerically to get MLE. The built-in logistic regression in R also finds the MLE. We can see this by comparing our MLE solution with the built-in fit on the iris data.

Logistic Regression MLE on iris data

We find the MLE on iris data with our program.

```
den<-function(x, y, beta) {#the probability density function
   lin.x<-beta[1]+beta[2]*x              #linear part beta0+beta1*x
   (exp(lin.x)/(1+exp(lin.x)))^y*(1/(1+exp(lin.x)))^(1-y) #logistic
transformation
}
nlk<-function(x,y,beta)  -sum(log(den(x,y,beta))) #negative log-likelihood.
## Fit MLE on iris data
y<-as.numeric(iris$Species=='setosa') ## Y=1 for setosa; Y=0 otherwise
SepalLength<-iris$Sepal.Length         ##Get first variable as X
nlk.iris<-function(beta) nlk(SepalLength, y, beta) #plug-in data, leaving only
the parameters to be estimated
beta.init <- c(0,0)            #Initial values of beta=(0,0)
optim(par=beta.init, fn=nlk.iris)  #Optimize nlk.iris using initial values
beta.init
```

If we run the program, we get:

```
$par
[1] 27.830709 -5.176039
 ... more outputs omitted
```

Compare this with the previous fit, we can see that the answers are the same except some numerical rounding errors.

```
> reg.lgr ##Display logistic regression fit
Call:  glm(formula = y ~ SepalLength, family = binomial(link = "logit"))
Coefficients:
(Intercept)  SepalLength
     27.829       -5.176
```

Estimating Logistic Regression Parameters

We have seen that the preprogrammed logistic regression in R gives the same answer as our MLE. In practice, we will always use built-in R codes to find the parameter estimates. They are coded by professionals and often use optimized numerical algorithms compared to our crude code.

The lesson here, however, is that those built-in programs are just implementing the general estimators like MLE. When dealing with a new model for which the built-in program does not exist, you can program the estimators like MLE on your own.

For logistic regression, we will use the preprogrammed glm() to do parameter estimation. The function also provides confidence intervals and hypothesis tests. Those are variations of the z-intervals and z-tests we saw in earlier modules. Hence those are asymptotic approximate intervals and tests (they are valid only for large sample size). We demonstrate those types of inferences on the iris data set next.

Tests for Logistic Regression Parameters on Iris Data

We can get more detailed information on the parameter estimations of the logistic regression fit by summary() function.

```
> summary(reg.lgr)
Call:
glm(formula = y ~ SepalLength, family = binomial(link = "logit"))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.25787  -0.27914  -0.04605   0.31399   2.14316

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   27.8285     4.8276   5.765 8.19e-09 ***
SepalLength   -5.1757     0.8934  -5.793 6.90e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 190.954  on 149  degrees of freedom
Residual deviance:  71.836  on 148  degrees of freedom
AIC: 75.836

Number of Fisher Scoring iterations: 7
```

After the values of parameter estimates, the standard errors are provided. There are also p-values for testing the null hypotheses $H_0 : \beta_0 = 0$ and $H_0 : \beta_1 = 0$. Both p-values are very small, so we reject the null hypotheses.

Similar to the linear regression model, we are more interested in $H_0 : \beta_1 = 0$ than $H_0 : \beta_0 = 0$. Rejecting $H_0 : \beta_1 = 0$ means that the slope for Sepal Length is nonzero. In other words, Sepal Length does contribute to the prediction of Y (or classification of setosa versus non-setosa flowers).

Next we will look at the confidence intervals for the parameters.

Confidence Intervals for Logistic Regression Parameters on Iris Data

We can get the confidence intervals for the parameters using the built-in confint() function.

```
> confint(reg.lgr, level=0.9) #Find 90% 2-sided CIs for the parameters
Waiting for profiling to be done...
                    5 %      95 %
(Intercept)    20.737501 36.746018
SepalLength -6.826734 -3.865255
```

Thus, the 90% CI for β_0 is (20.74, 36.75) and the 90% CI for β_1 is (-6.83, -3.87). Notice that this also gives 95% upper CI for β_1 as (-∞, -3.87) and 95% lower CI for β_1 as (-6.83, ∞).

There is another built-in confint.default() function which gives the Wald's confidence intervals $\hat{\beta}_0 \pm z_{\alpha/2} se(\hat{\beta}_0)$ and $\hat{\beta}_1 \pm z_{\alpha/2} se(\hat{\beta}_1)$.

```
> confint.default(reg.lgr) #Find 95% Wald CIs. The level=0.95 when not
specified.
                    2.5 %    97.5 %
(Intercept)    18.366675 37.290367
SepalLength -6.926727 -3.424669
```

The Wald's CIs require less computing time, but are less accurate since they do not consider the nonlinear effect of the link function. So we should use confint() as long as computation time permits. The confint() function produce so-called profiling confidence intervals.

Next we return to the prediction of Y, given X values.

Prediction by Logistic Regression on Iris Data

For a X value, the logistic regression gives a prediction probability of Y=1 as $p(X)=\frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}}$. This can be calculated in R using the predict() function.

For example, the prediction probability of an iris flower with 5.5cm sepal length being setosa (Y=1) is 34.6%, as calculated by

```
> predict(reg.lgr, newdata=data.frame(SepalLength=5.5), type="response")
        1
0.3457399
```

Here the predict() function takes the input as a fitted object of glm. The X value is specified through "newdata=" option, which takes a data frame as value. The type="response" results in prediction probability $p(X)=\frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}}$. type="link" would give the prediction of the linear part of the model $\log[\frac{p(X)}{1-p(X)}]=\hat{\beta}_0+\hat{\beta}_1 X$.

```
> predict(reg.lgr, data.frame(SepalLength=5.5), type="link")
        1
-0.6378183
```

It is easy to check that $\log[\frac{0.3457399}{1-0.3457399}]=-0.6378183$.

Our ultimate goal of classification is to predict the value of Y. A natural approach is to assign any object to the class with highest probability. For example, an iris flower with 5.5cm sepal length will be classified as non-setosa (Y=0) since P(Y=1)=34.6% and P(Y=0)=65.4%. We can check the results of logistic regression classification on the iris data.

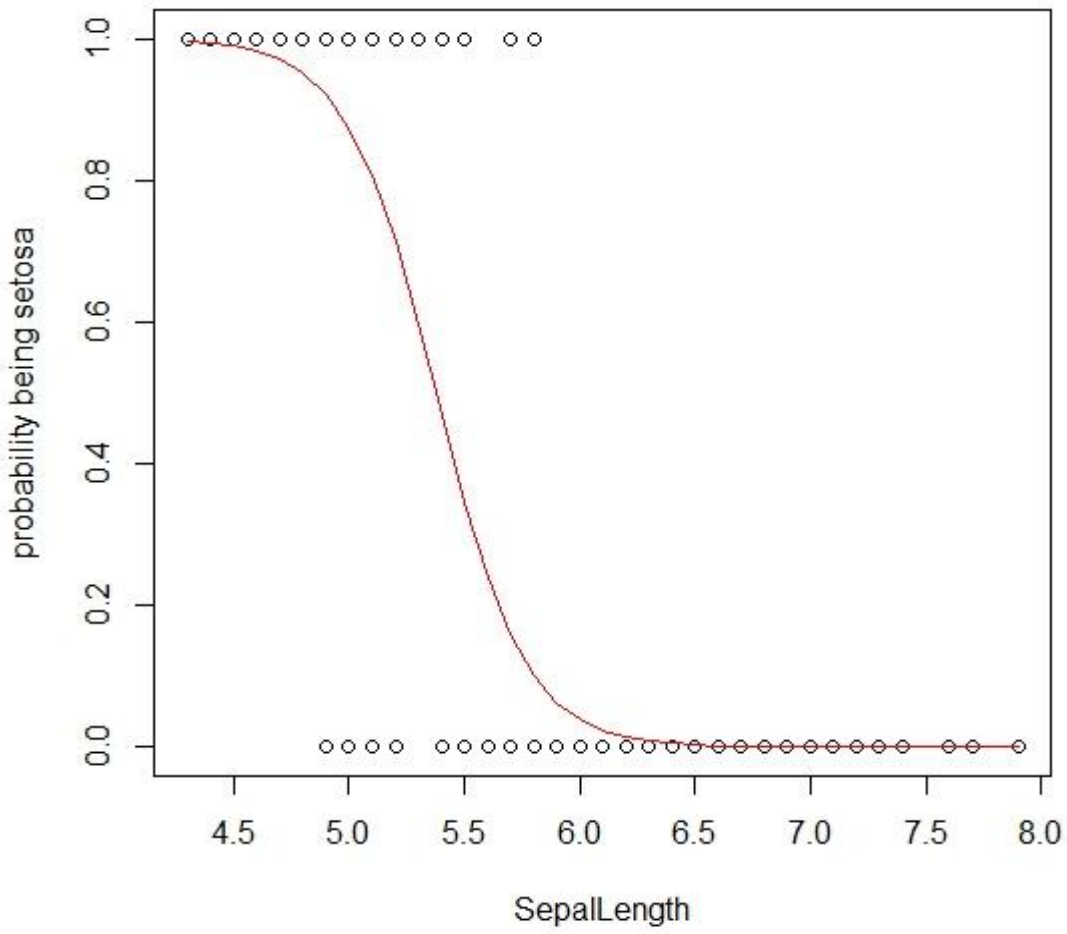
```
> pred.lgr <-(predict(reg.lgr, type="response")>0.5) #cases with prediction probability>0.5 on the original data set, since no "newdata=" is specified.
> pred.lgr <- factor(pred.lgr, levels=c(TRUE,FALSE), labels=c("setosa","not setosa")) #label as setosa versus not setosa.
> true.setosa<-factor(y, levels=c(1,0), labels=c("setosa","not setosa")) #label the true cases as setosa versus not setosa
> table(pred.lgr, true.setosa) #compare real classes with logistic regression classification. This is called a confusion matrix.
                true.setosa
pred.lgr     setosa not setosa
  setosa         40          6
  not setosa     10         94
```

We see that majority of flowers are classified correctly, 10 out of 50 setosa flowers are incorrectly classified, and 6 out of 100 non-setosa flowers are incorrectly classified.

Display of Prediction Probability by Logistic Regression on Iris Data

We can show graphically the logistic regression fit.

```
plot(y~ SepalLength, ylab="probability being setosa") #data plot, set y-label
#get equally spaced (0.1 difference) grid over the range of Sepal Length
values. These are the points we calculate prediction probabilities to draw a
cuve
x<-seq(from=min(SepalLength), to=max(SepalLength), by=0.1)
pred.x<-predict(reg.lgr, newdata=data.frame(SepalLength=x),
type="response") #prediction probabilities on the x values for  the fit in
'reg.lgr'
lines(x, pred.x, col='red') #a red line of predicted probabilities
```



We see that basically, the flowers with smaller sepal lengths are classified as setosa. For the middle range of sepal lengths, both setosa and non-setosa flowers exist. The probability being a setosa decreases along a smooth curve.

Logistic Regression with Multiple Predictor Variables

Like multiple linear regression, we can also use multiple predictor variables in logistic regression. For *m* predictor variables $X_1,...,X_m$, we fit model

$$p(X) = e^{\beta_0+\beta_1X_1+...+\beta_mX_m} / (1+e^{\beta_0+\beta_1X_1+...+\beta_mX_m}).$$

We illustrate this on the iris data set. We will use all four numerical predictors Sepal.Length, Sepal.Width, Petal.Length and Petal.Width.

```
> y<-as.numeric(iris$Species=='setosa') ## Y=1 for setosa; Y=0 otherwise
> iris.lgr <-cbind(iris[,1:4],y) #Combine the four predictor variables with Y into a new data set iris.lgr
> reg.lgr <- glm(y~. , data=iris.lgr, family=binomial(link='logit'))#logistic regression of Y on all other (four) variables, on data set iris.lgr
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Notice "y~." means to regress y on all other variables in the data set.

We get some warning messages here, since we achieved a perfect classification, and prediction probabilities for all cases are either 0% or 100%. We can check this by comparing to the true classes.

```
> pred.lgr <-(predict(reg.lgr, type="response")>0.5) #classify into class with >0.5 probability
> pred.lgr <- factor(pred.lgr, levels=c(TRUE,FALSE), labels=c("setosa","not setosa")) #label classes as setosa and not setosa.
> true.setosa<-factor(y, levels=c(1,0), labels=c("setosa","not setosa")) #label true classes
> table(pred.lgr, true.setosa) #confusion matrix
                true.setosa
pred.lgr     setosa not setosa
  setosa         50          0
not setosa       0        100
```

We see that all flowers are correctly classified. This is because the setosa flowers are well separated from the other two species.

The above table comparing the predicted classes versus the true classes is called the **confusion matrix**. A good classifier would have very small off-diagonal entries in the confusion matrix.

Logistic Regression with Singular Parameter Estimations

On the iris data set, we achieved perfect classification. However, once perfect classification is achieved, there are many other parameter values that can also achieve perfect classification. This leads to an identifiability problem: many parameter values give the same likelihood so that we cannot distinguish them on the data set. The standard errors of the parameters become very big so that we can no longer do the usual statistical inferences on the parameters. We can see the standard errors of the parameters using summary().

```
> summary(reg.lgr)
……
Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      -16.946 457457.097       0        1
Sepal.Length      11.759 130504.042       0        1
Sepal.Width        7.842  59415.385       0        1
Petal.Length     -20.088 107724.594       0        1
Petal.Width      -21.608 154350.616       0        1
…..
```

Therefore, the confidence intervals will be so wide that we really have no idea regarding the value of the parameters. However, for classification purposes, we may not care about the parameter values inferences.

We do need some uncertainty assessment on how reliable the classification results are (the logistic regression here says we are 100% sure of all flowers which seem to be overly confident). We will discuss evaluation methods for classification in the next lesson. Before we go to that, we cover the logistic regression for multiple classes next.

Logistic Regression for Multiple Classes

We have done the logistic regression for a response variable Y with only two classes. In practice, there are often more than two classes that we need to classify. In those situations, we can do logistic regression using multinomial instead of binomial probability. We illustrate this on the iris data, which in fact has three classes (species). We will need the R package "VGAM".

```
library(VGAM); data(iris) # load the package and data
##Regress Speices on all other (four) variables on data set iris.
family=multinomial to specify logistic regression for multiple classes
iris.lgr <- vglm(Species~., family=multinomial, data=iris)
pred.prob <- predict(iris.lgr, iris[,1:4], type="response") #get prediction
probability for all cases in data set
pred <- apply(pred.prob, 1, which.max) #Assign to the class with largest
prediction probability
pred <- factor(pred, levels=c("1","2","3"), labels=levels(iris$Species))
#Substitute 1,2,3 by species names in "pred" variable.
table(pred, iris$Species) #Compare classified with true classes
```

Run the R code and we get

```
  pred       setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          1        49
```

We can see that most flowers are correctly classified. Only one virginica is misclassified as versicolor, and one versicolor is misclassified as virginica. The overall misclassification rate is 2/150=0.013 (wrong in 2 out of 150 total cases).

For a new flower with 5cm sepal length, 4cm sepal width, 5cm petal length and 2cm petal width, we find the prediction probability for each species as

```
> newx<-data.frame(Sepal.Length=5, Sepal.Width=4, Petal.Length=5,
Petal.Width=2) #new flowers data in newx
> predict(iris.lgr, newdata=newx, type="response") #use iris.lgr fit on newx
       setosa          versicolor   virginica
1 5.200073e-26  0.1159098 0.8840902
```

So we will classify this new flower as virginica which has the highest (88.4%) probability.

Statistical Inferences of Logistic Regression

Due to the similar structure to linear regression, we can carry out many of the familiar statistical inferences in logistic regression. These include parameter estimators, confidence intervals and hypothesis tests.

However, when we focus on using the logistic regression as a classification tool, these standard statistical inferences may not be what we want. Also, unlike linear regression, the diagnostics of model assumptions are much harder using the residuals (since the responses are binary, either 0 or 1).

In the next lesson, we discuss evaluation methods for classification tools.

Lesson Summary

This lesson introduces logistic regression. We covered how to write the logistic regression model as a generalized linear model, and reviewed how to obtain common statistical inferences in logistic regression: parameters estimation (by MLE), confidence intervals and hypothesis tests.

You should now be able to use R to fit the logistic regression including with multiple predictors and for the response variable of multiple classes. You should also be able to translate the logistic regression fit into classification results.

The next lesson covers evaluation methods for classification tools.

Lesson 2: Evaluation of Classification Methods: ROC curve and validation

Objectives

By the end of this lesson you will have had the opportunity to:

- Describe and calculate **measures** of classification performance: misclassification rate (mcr), false positive rate (fpr), false negative rate (fnr), sensitivity, specificity
- Use the **ROC curve** to assess the classification performance
- Apply **cross-validation** to assess the classification performance

Overview

In this lesson, we introduce some common evaluation methods. We first introduce the definition of several numerical measures of performance. We then show how to summarize some of them in what is known as an ROC curve. Finally, we introduce the need for validation, and demonstrate validation on examples of logistic regression classifiers.

Evaluation of the Classification Performance

To judge classification results, we need some numerical measures for the performance. One natural measure is the misclassification rate: the number of misclassified cases divided by the total number of cases. This rate reflects an overall error rate. However, the error rates may not be same for all classes, and we need to look at more detailed information.

For simplicity, we consider the situation of only two classes (such as AML versus ALL patients). We call one class 'positive' and the other class 'negative'. For each case, we also make a prediction if it belongs to the positive (+) class or the negative (-) class. Then we have the following natural measures:

- *False positive rate* (fpr) is the probability that we predict a positive (+) class when the true class is negative (-). It is estimated by the total number of falsely predicted positive cases divided by the total number of true negative cases.

- *False negative rate* (fnr) is the probability that we predict a negative (-) class when the true class is positive (+). It is estimated by the total number of falsely predicted negative cases divided by the total number of true positive cases.

Notice that those rates are the Type I and Type II error rates in the hypothesis testing. Correspondingly, we also have the *true positive rate* (tpr) which equals 1-fnr, and *true negative rate* (tnr) which equals 1-fpr. The true positive rate (tpr) is called **sensitivity**. This is the power in hypothesis testing, the probability of predicting positive (reject null hypothesis) when the true class is positive. (Recall that a positive result refers to getting what we wanted to prove, the alternative hypothesis.) The true negative rate (tnr) is called **specificity**.

We illustrate these measures on logistic regression classifiers of the Chiaretti (2004)'s ALL data set.

Demonstration on ALL Data Set

Example 1: We use a subset of the ALL data set that were studied before. We focus on the B-cell patients in B1, B2 and B3 stages. There are totally 78 such patients.

We first focus on predicting if a patient belongs to the B1 stage, and separate patients into two classes of 'B1' and 'not'. For distinguishing the three stages, we have selected the top five genes using ANOVA in module 10 homework. We start with logistic regression on the top gene of probe name 1389_at.

```
library("ALL"); data(ALL);  #load package and data
allB123 <- ALL[,which(ALL$BT %in% c("B1","B2","B3"))] #select patients
prob.name <- "1389_at"       #select the gene
expr.data <-exprs(allB123)[prob.name,]  #get expression data for this gene
IsB1 <-  (allB123$BT=="B1")       #A boolean class indicator (in B1?)
data.lgr <- data.frame(IsB1, expr.data)  #A data frame of class indicator and expression data.
fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.lgr) #logistic regression of class indicator on other variables (1389_at expression values)
pred.prob <- predict(fit.lgr, data=data.lgr$expr.data, type="response") #predict class probability (response) using the regression fit on expression data.
pred.B1<- factor(pred.prob> 0.5, levels=c(TRUE,FALSE), labels=c("B1","not")) #classify according to prediction probability, and label TRUE to 'B1' and FALSE to 'not'
IsB1<-factor(IsB1, levels=c(TRUE,FALSE), labels=c("B1","not")) #class indicator label to 'B1' and 'not'
table(pred.B1, IsB1) #confusion matrix that compares predicted versus true classes
```

Run the R codes and we have

```
          IsB1
pred.B1 B1 not
   B1    17   4
   not    2   55
```

So, most of patients are classified correctly. The misclassification rate is (4+2)/78=0.077.

We will look at the estimates of the other measures on next page.

Demonstration on ALL Data Set (Continued)

This is the table (confusion matrix) from last page

```
          IsB1
pred.B1 B1 not
   B1    17   4
   not    2   55
```

Here we take B1 as positive, and non-B1 as negative. So the error rates are

False positive rate: fpr=(# false predicted B1)/(# true 'not')=4/(4+55)=0.068.
False negative rate: fnr=(# false predicted 'not')/(# true B1)=2/(17+2)=0.105.

Also, the sensitivity = tpr = 17/(17+2) = 0.895.
The specificity = tnr = 55/(4+55)=0.932.

This is a pretty good classification result: 89.5% B1 stage patients are correctly classified as B1, 93.2% non-B1 patients are correctly classified as not being in the B1 stage.

Notice that the logistic regression gives prediction probabilities, and we classified patients with >0.5 probability as B1 patients. Threshold values other than 0.5 can also be used, to adjust the sensitivity and specificity of the classification. For example, if we only classify as B1 patients those with prediction probability >0.7, we have

```
> pred.B1<- factor(pred.prob> 0.7, levels=c(TRUE,FALSE),
labels=c("B1","not"))
> table(pred.B1, IsB1) #confusion matrix
       IsB1
pred.B1 B1 not
   B1  16   2
   not  3  57
```

This results in two less misclassified non-B1 patients, but one more misclassified B1 patient.

Now fpr=2/(2+57)=0.034; fnr=3/(16+3)=0.158. The sensitivity=84.2%, the specificity=0.966.

We used the table (confusion matrix) to show the quantities clearly. The rates can be calculated in R directly without the table. For example,

```
sum(pred.B1=="B1" & IsB1 !="B1")/sum(IsB1 !="B1") #calculate fpr
```

Measures of Classification Performance and the ROC Curve

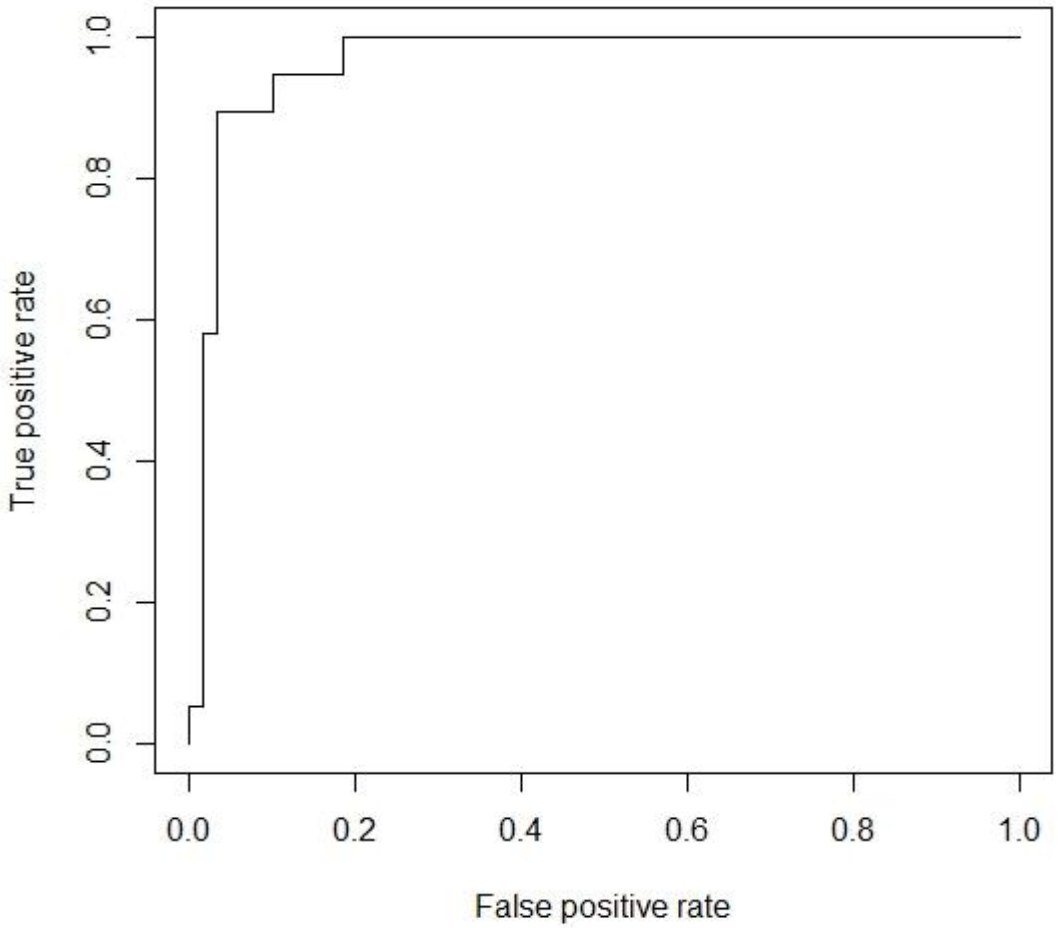
As we change the threshold value in the classification rule in the above example, the performance measures changes. As we increase the threshold value, the sensitivity decreases and the specificity increases. If we want to increase the sensitivity, then we can decrease the threshold value, which also decreases the specificity.

Graphically, we can represent the performance over all threshold values by the **operator characteristic (ROC) curve**. The curve plots the true positive rate (sensitivity) versus the false positive rate (1-specificity) for different threshold values. We use the 'ROCR' package to draw this curve (Install the package first if you have not done so before). We will do this for the logistic regression of the previous example.

Demonstration on ALL Data Set: ROC Curve

We use the following R code:

```
library(ROCR) ##Load library
pred <- prediction(pred.prob, IsB1=="B1") #using pred.prob to predict the response (which need values of T/F, thus we apply ==”B1”)
perf <- performance(pred, "tpr", "fpr" ) #compute tpr and fpr for pred
plot(perf)      #Plot tpr versus fpr, i.e., ROC curve
```



We see that the true positive rate increases very fast for small false positive rate. This indicates a good classifier. We can also get a numerical summary of the ROC curve such as the area under curve (AUC).

Using the R command performance(pred,"auc"), we can find the AUC is 0.965. An AUC value >0.5 indicates better than a random guess. A perfect classifier achieves the largest possible value of AUC=1.

ROC Curves Comparison on ALL Data

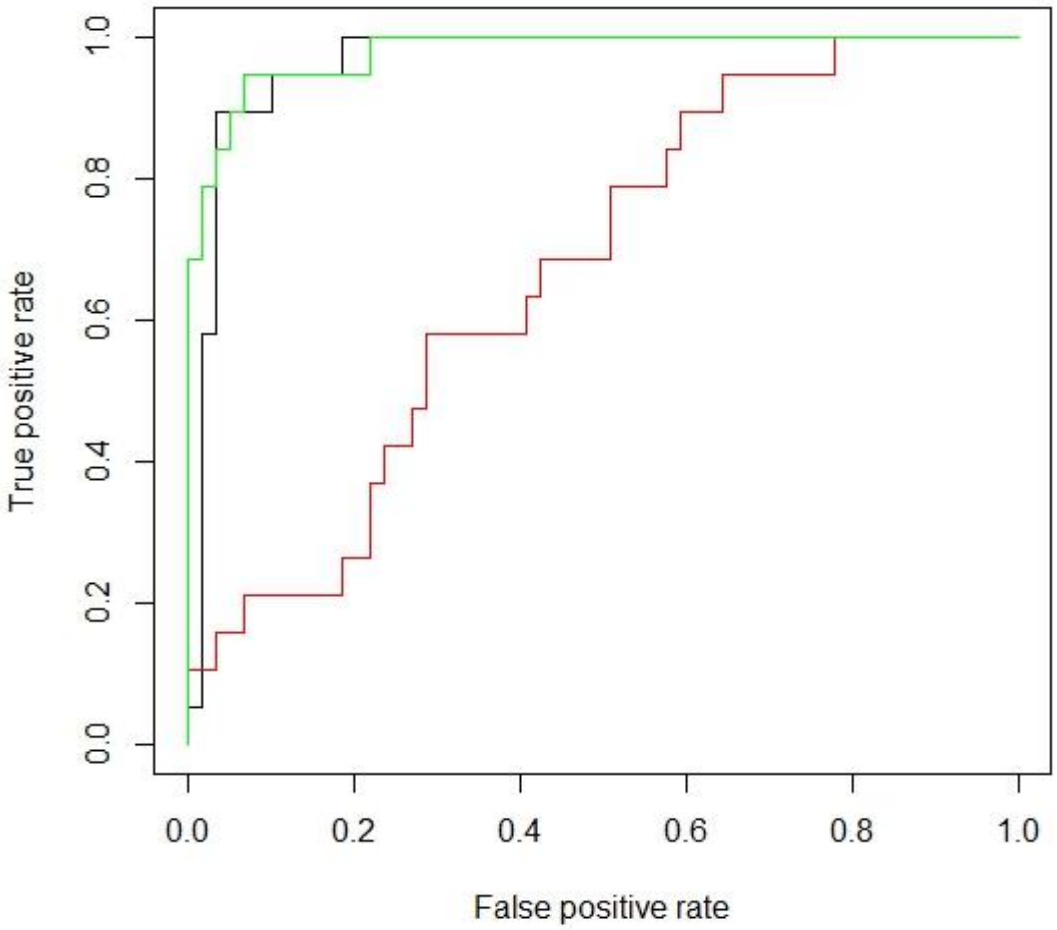
We can use ROC curves to compare different classifiers. Here we consider two more logistic regression classifiers: the first one will use the first gene in the data set (represent an arbitrarily chosen gene without statistical/biological consideration), and the second one will use the top three genes selected by ANOVA in a previous assignment. We add their ROC curves to the plot of ROC curve for the earlier logistic regression classifier. The R code follows:

```
#### Arbitrarily choose a gene (first gene in the data).
expr.data <-exprs(allB123)[1,] #choose first gene.
IsB1 <-  (allB123$BT=="B1") #These commands are same as before.
# Repeating logistic regression except on the first gene expression now.
data.lgr <- data.frame( IsB1,expr.data)
fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.lgr)
pred.prob <- predict(fit.lgr, data=data.lgr$expr.data, type="response")
pred <- prediction(pred.prob, IsB1)
perf <- performance(pred, "tpr", "fpr" )
plot(perf,col="red",add=TRUE) #add red ROC curve for this classifier
### Use the top 3 genes
prob.name <- c("1389_at", "1914_at", "33358_at")
expr.data <-exprs(allB123)[prob.name,]
data.lgr <- data.frame( IsB1, t(expr.data))
fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.lgr)
pred.prob <- predict(fit.lgr, data=data.lgr[,-1], type="response")
pred <- prediction(pred.prob, IsB1)
perf <- performance(pred, "tpr", "fpr" )
plot(perf,col="green",add=TRUE) #add green ROC curve for this classifier
```

We compare the three classifiers through their ROC curves on next page.

ROC Curves Comparison on ALL Data (Cont'd)

Run the R code and we get the graph as



We can see that using the first gene (red line) provides a much weaker classifier (AUC=0.67) than the classifier using the top gene selected by ANOVA (AUC=0.965). Using three top genes in logistic regression (green line) increases the AUC (=0.979) only by a small amount. The green ROC curve and the black ROC curve cross each other, so neither classifier dominates the other over all specificity values.

Need of Using Validation for Evaluation

The quantities we used to summarize the performance, fpr, fnr, tpr and tnr were defined as probabilities. Using basic statistical terms, the population probabilities (over all future cases) are the true parameters of interest. What we have calculated are the empirical probabilities on the observed data set, hence are estimates of true parameters. For these quantities to be useful, they have to be accurate.

For logistic regression classifiers, we can make statistical inferences on those quantities by relating them to the fitted parameter values $\beta_0, \beta_1, ..., \beta_m$. However, there are two issues on such inferences. First, the logistic regression model may not hold, and the diagnostic tools for checking model assumptions are limited. Second, the parameters $\beta_0, \beta_1, ..., \beta_m$ are estimated on the same data set that is also used to calculate performance measures. This often leads to the "over-fitting" problem. That is, the classification measures such as the misclassification rate (mcr) are better on the data set than the true rates for future cases.

The first issue of no reliable statistical inferences also holds for other classifiers that we will introduce later. And other types of classifiers often do not have the regression form, and also do not have ready parameter inferences as the logistic regression classifier. The second issue of over-fitting can be more serious for other more complicated classifiers. Those classifiers can classify all cases in a data set perfectly if no restrictions are put on their parameters. However, that type of performance cannot generalize to classification of future cases.

These issues require us to find some nonparametric evaluation methods that alolow us to avoid the over-fitting effect. The most commonly used idea is **validation**: to evaluate the classification performance on a future validation data set which is not used in the parameter estimation. Since we do not have such a future data set at time of data analysis, we can split the available data into two parts: training data and validation data. The parameters are estimated based on observations in the training data set only. Then the performance of the fitted classifier is evaluated on the hold-out validation data set. This is called cross validation.

Next, we will demonstrate the cross validation of the logistic regression classifier on the gene with probe name "1389_at".

Demonstration of Training and Validation

We revisit the previous example of predicting if a patient belongs to B1 stage, in the data set of 78 B-cell patients in B1, B2 and B3 stages. We divide the data set into 60% (47 patients) for training data and 40% (31) for validation data set. The data set is divided randomly. We fit the logistic regression model on the gene with probe name 1389_at, using the 47 training data. We then compare the misclassification rates (mcr) of the resulting classifier on the training data set and on the validation data set.

```
#The first several lines are same as before, get the data set for logistic
regression
library("ALL"); data(ALL);
allB123 <- ALL[ , which(ALL$BT %in% c("B1","B2","B3"))]
expr.data <-exprs(allB123)[ "1389_at", ]
IsB1 <-  (allB123$BT=="B1")
data.lgr <- data.frame( IsB1,expr.data) #The whole data set in data.lgr
#Training and Validation (testing)
set.seed(131) #Set an arbitrary random seed.
testID <- sample(1:78, 31, replace = FALSE) #randomly select 31 test
(validation) cases out of 78
data.tr<-data.lgr[-testID, ]      #training data, remove test cases with "-"
data.test<-data.lgr[testID, ]    #test (validation) data
fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.tr) #fit
logistic regression on training data.
pred.prob <- predict(fit.lgr, data=data.tr, type="response") #training data
prediction
pred.B1<- factor(pred.prob> 0.5)
mcr.tr<- sum(pred.B1!=data.tr$IsB1)/length(data.tr$IsB1) #mcr on training
data. (# of misclassification)/(# total cases)
pred.prob <- predict(fit.lgr, newdata=data.test, type="response") #test data
prediction
pred.B1<- factor(pred.prob> 0.5)
mcr.test<- sum(pred.B1!=data.test$IsB1)/length(data.test$IsB1) #mcr on test
data
data.frame(mcr.tr, mcr.test) #print out mcr
```

Run it and we get

```
     mcr.tr  mcr.test
1 0.08510638 0.1290323
```

So the misclassification rate (mcr) is 8.5% on training data but 12.9% on validation data. Generally the mcr is lower on the training data, but not always. The measures on validation data are usually a better reflection of the true performance.

We have set the random number generator seed, to ensure getting the same result if we run the code again. Generally, you do not set the seed.

Repeated Cross Validation

In last page, we did the cross validation by randomly splitting the data set into a training data set for parameter and a validation data set to check performance. If we run the cross validation another time, we will get a different split, and the results are different. To reduce the variability, we can run multiple rounds of cross validation. We can repeatedly split the data into training and validation data sets, and average the validation results over the repeated splits.

The leave-p-out cross-validation use p cases as validation data set and the remaining (n-p) cases as training data. The total number of possible combinations of choosing p cases out of n is denoted as $\binom{n}{p}$ which grows very fast for large n. In such cases, we just run the cross-validation for a fixed number nsim rounds. That is, instead of iterating over all possible $\binom{n}{p}$ splits, we just do nsim random splits. This is very similar to the bootstrap or permutation tests. Instead of repeatedly resampling from the original data set, here we repeatedly split the data set.

For leave-one-out cross-validation, there are only n possible such splits. So we can actually iterate through all possible splits instead of doing a fixed nsim random rounds of splits.

We will now demonstrate the leave-p-out cross validation on the previous example of logistic regression classifier.

Demonstration of Leave-one-out Cross-validation

We can iterate through the n=78 patients. Each time we use the i-th patient as the validation data, while training the logistic regression model on the other 77 patients.

```
######## leave-one-out cross-validation
n<-dim(data.lgr)[1]    ##size of the whole sample
mcr.cv.raw<-rep(NA, n) ## A vector to save misclassification rates (mcr) for each delete-one validation
for (i in 1:n) {
#delete each observation from training, then use it for testing.
  data.tr<-data.lgr[-i,]    #remove i-th observation from training
  data.test<-data.lgr[i,]  #the i-th observation is kept for testing
  fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.tr) #train model
  pred.prob <- predict(fit.lgr, newdata=data.test, type="response") #prediction probability
  pred.B1<- (pred.prob> 0.5) #prediction class
 mcr.cv.raw[i]<- sum(pred.B1!=data.test$IsB1)/length(pred.B1)#mcr on testing case
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all n rounds.
```

Run it and we get

```
[1] 0.07692308
```

So the leave-one-out cross-validated misclassification rate (mcr) is 7.7%.

Next we will carry out a leave-five-out cross-validation.

Demonstration of Leave-five-out Cross-validation

There are in total 21111090 possibilities for choosing p=5 patients out of the n=78 patients. We do not want to iterate through all of them. So we will just do nsim=2000 rounds.

```
######## leave-p-out cross-validation at p=5
n<-dim(data.lgr)[1]    #size of the whole sample
p<-5                   #The number of test cases
nsim<-2000     #number of rounds for validation
mcr.cv.raw<-rep(NA, nsim) ## A vector to save raw mcr
for (i in 1:nsim) {
  testID<-sample(n, p, replace=FALSE) #sample p test cases
  data.tr<-data.lgr[-testID,]   #remove the p cases from training data
  data.test<-data.lgr[testID,]  #validation (test) data
  fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.tr) #train
model
  pred.prob <- predict(fit.lgr, newdata=data.test, type="response")
#prediction probability
  pred.B1<- (pred.prob> 0.5) #prediction class
  mcr.cv.raw[i]<- sum(pred.B1!=data.test$IsB1)/length(pred.B1)#mcr on
testing case
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all nsim rounds.
```

Run it and we get

```
> mcr.cv
[1] 0.0777
```

So the leave-five-out cross-validated misclassification rate (mcr) is 7.8%.

Comparison of Validation Results on the ALL data

We have done several validations of the misclassification rate (mcr). Let us compare the results.

The empirical mcr=0.077 for the original logistic regression fit on the whole data set. We used a random 60/40 percent split of data, and get training mcr=0.085 and validation mcr=0.129. The leave-one-out cross-validation mcr=0.077 and the leave-five-out cross-validation mcr=0.078.

First, some of the numbers above contain Monte Carlo error, and would vary if run again. The random 60/40 percent split, if conducted again, would vary and the training/validation mcr will change accordingly. The leave-five-out cross-validation mcr used nsim=2000 runs. That is, it randomly split off 5 data points as validation data, repeated 2000 times. If we calculate the leave-five-out cross-validation mcr again, we will get 2000 different splits and the mcr value will vary accordingly. However, with the average over 2000 runs, the variation is much smaller than the variation in validation mcr using just one run. The leave-one-out cross-validation mcr uses exhaustively n=78 leave-one-out splits, thus will remain the same if calculated again.

Second, we see that both the leave-one-out cross-validation mcr=0.077 and the leave-five-out cross-validation mcr=0.078 are very close to the empirical mcr=0.077. This indicates that there is little over-fitting for the logistic regression classifier on this data set.

Third, we see that the training mcr=0.085 and validation mcr=0.129 are bigger than the empirical mcr=0.077 in the random 60/40 percent split. This can be due to the Monte Carlo error mentioned above. It can also come from the fact that training is done on a smaller (60%) data set. Generally smaller data set lead to worse training and higher mcr.

The proportions of splitting requires some thinking. We do not want the training data to be too small so that the validated measure does not reflect performance on the whole data. But also we need enough validation data to estimate the performance measures. So for one split, we generally keep more than half of the data for training. 50/50 to 90/10 training/validation splits are often used. By using average over multiple runs, we do not need big validation data in each run. So the leave-p-out cross-validation can be done with small p numbers.

K-fold Cross Validation

Another commonly used validation method is the K-fold cross-validation. We randomly split the data into K equal size subsamples. Then we iterate K times. Each time use one subsample as the validation data, and the remaining K-1 subsamples as training data. Each subsample is used exactly once as validation data. The ten-fold cross-validation is commonly used.

When we use the sample size n as K value, the n-fold cross-validation becomes exactly the same as the leave-one-out cross-validation.

The ten-fold cross-validation is commonly used, as computationally it is much faster than the leave-p-out cross-validation for most p values. However, since it averages over only ten rounds, the variation in the resulting estimates are much bigger.

Demonstration of Ten-fold Cross-validation on ALL Data

We now try the 10-fold cross-validation of the logistic regression on the n=78 patients. So the ten subsamples would consists of two subsamples of size 7 and eight subsamples of size 8 (total $2 \cdot 7 + 8 \cdot 8 = 14 + 64 = 78$). We use the createFolds() function in the package "caret" to randomly generate the ten subsamples. You need to install the package first, install.packages('caret'). The following R code estimates the 10-fold cross-validated mcr.

```
## 10 fold cross-validation
require(caret)
n<-dim(data.lgr)[1]    #size of the whole sample
index<-1:n #index for data points
K<-10      #number of folds
flds <- createFolds(index, k=K) #create K folds
mcr.cv.raw<-rep(NA, K) ## A vector to save raw mcr
for (i in 1:K) {
  testID<-flds[[i]]  #the i-th fold as validation set
  data.tr<-data.lgr[-testID,]   #remove the test cases from training data
  data.test<-data.lgr[testID,]  #validation (test) data
  fit.lgr <- glm(IsB1~., family=binomial(link='logit'),  data=data.tr) #train
model
   pred.prob <- predict(fit.lgr, newdata=data.test, type="response")
#prediction probability
   pred.B1<- (pred.prob> 0.5) #prediction class
  mcr.cv.raw[i]<- sum(pred.B1!=data.test$IsB1)/length(pred.B1)#mcr on
testing case
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over K folds.
```

Run it and we get

```
> mcr.cv
[1] 0.08035714
```

So the 10-fold cross-validated misclassification rate (mcr) is 8.0%.

There are some variations when we run this a few more times. My twenty runs of the 10-fold cross-validation resulted in an estimated mcr from 5.4% to 10.4%.

Evaluation of Other Classifiers

We have introduced evaluation measures for classifier performance such as misclassification rates (mcr), false positive rates (fpr), sensitivity, specificity, et al. We also considered nonparametric evaluation by using cross validation.

We demonstrate the cross validation of misclassification rate on one logistic regression classifier example. The cross validation of other performance measures on other classifiers can be carried out similarly.

In the next lessons, we will introduce two more classifiers. We will not go into the mathematical details of their construction. We will just show how to use them for classifications. Just knowing how to run these classifiers allow us to evaluate their performance using methods in this lesson. Also, there are tuning parameters (such as number of variables used as predictors in logistic regression) for those classifiers. Often the cross-validation is programmed in R functions for the classifiers to choose the tuning parameter. We will just use the R implementations without going into details.

Lesson Summary

This lesson introduced evaluation methods for the classification performance. You should now be able to define certain measures, including misclassification rates, false positive/negative rates, true positive/negative rates, sensitivity and specificity. We then covered how to describe the classifier performance using a ROC curve. You should now be able to draw the ROC curve.

We then introduced evaluations through training and validation. You should be able to conduct leave-p-out validations and K-fold validations.

The next lesson introduces a new classifier, the support vector machine.

Lesson 3: Support Vector Machine (SVM)

Objectives

By the end of this lesson you will have had the opportunity to:

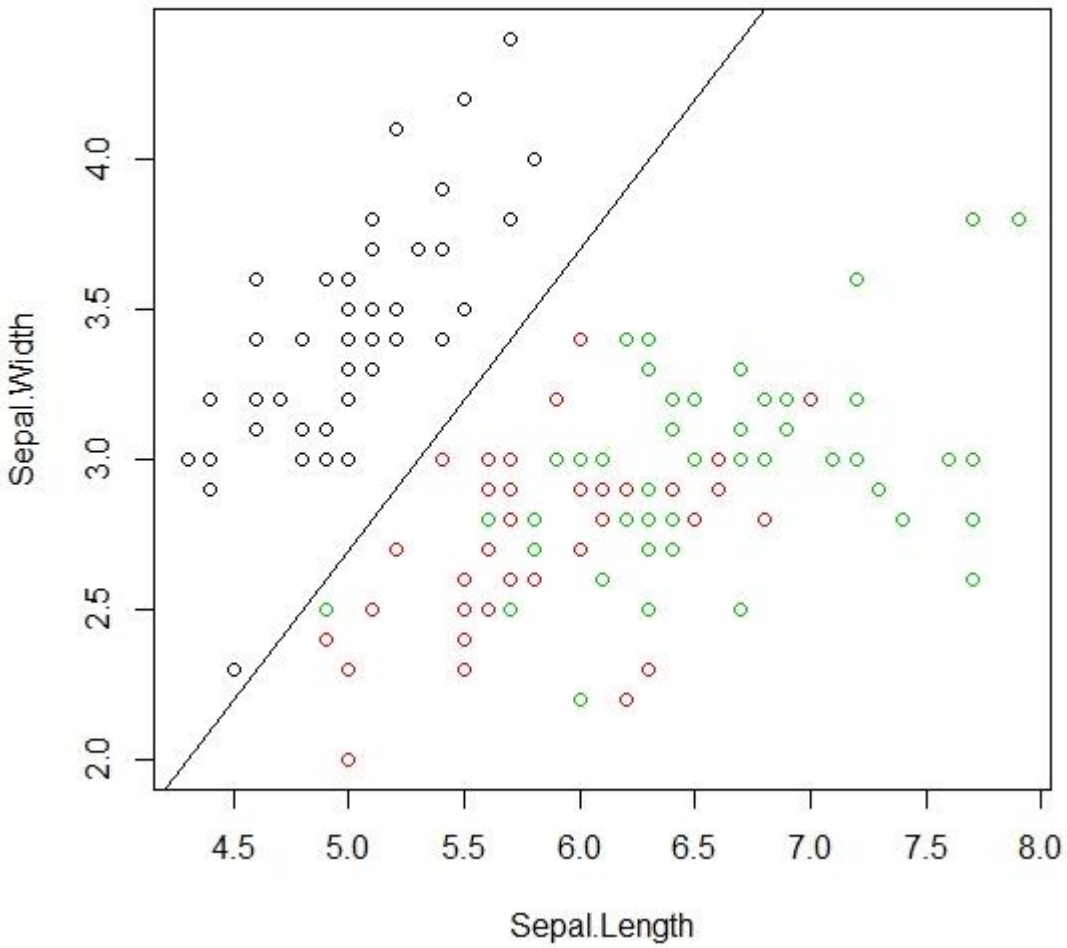
- Apply **Support Vector Machine (SVM)** to conduct classification
- Use **cross-validation** to evaluate the performance of SVM classifier

Overview

In this lesson, we will introduce the SVM classifier. We will demonstrate the training of the SVM classifier, and then use cross-validation to evaluate its performance.

Support Vector Machine (SVM)

The support vector machine finds separating lines (hyper planes) to divide the data into classes. The following plot shows one separating line for setosa flowers from other two species.



The support vector machine aims to find the best separating line according to a numerical criterion. It can also find other nonlinear separating boundaries by using nonlinear kernel. Here we just illustrate the support vector machine with linear kernel on the classification of ALL data.

Demonstration of Support Vector Machine on ALL data

We demonstrate the application of support vector machine on diagnosis of B-cell state B1, B2, and B3 from gene expressions in the Chiaretti (2004) ALL data set. Following the example on page 157 in Applied Statistics for Bioinformatics using R, we first reduce the number of genes using a gene filter by ANOVA. Particularly we select the genes with ANOVA *p*-values smaller than 0.000001. This results in 29 selected genes. The following code selected the 29 genes for 78 patients and saved the data set 'probedat'. We then do classification on this data set.

```
library("hgu95av2.db");library(ALL);data(ALL)            #load library
ALLB123 <- ALL[,ALL$BT %in% c("B1","B2","B3")]   #patients in
B1,B2,B3 stages
pano <- apply(exprs(ALLB123), 1, function(x) anova(lm(x ~
ALLB123$BT))$Pr[1]) #do ANOVA on every gene
names <- featureNames(ALL)[pano<0.000001]  #get probe names for genes
passing the ANOVA gene filter
symb <- mget(names, env = hgu95av2SYMBOL) #get the gene names
ALLBTnames <- ALLB123[names, ]   #keep only the gene passing filter
probedat <- as.matrix(exprs(ALLBTnames))  #save expression values in
probedat
row.names(probedat)<-unlist(symb) #change row names to gene names
```

We now apply the support vector machine using svm() function in the 'e1071' package (install it with install.packages('e1071')).

```
library(e1071) #load library
B.stage <- factor(ALLBTnames$BT) #the classes
exprs.gene <- t(probedat) #use above data set of selected gene expression
values. Transpose since the classification is done for rows (patients)
svmest <- svm(B.stage~exprs.gene, type = "C-classification", kernel =
"linear") #Do svm for classification of B.stage based on exprs.gene, use linear
kernel.
svmpred <-predict(svmest, exprs.gene) #use fit to get predicted classes
table(svmpred, B.stage) #confusion matrix compare prediction and true classes
```

Run it and we get

```
        B.stage
svmpred B1 B2 B3
     B1 19  0  0
     B2  0 36  1
     B3  0  0 22
```

The patients are well classified with misclassification rate of only 1/78=0.013.

Demonstration of Support Vector Machine on ALL data (cont'd)

We now check the performance of the support vector machine classification by leave-one-out cross validation.

```
######## leave-one-out cross-validation
B.stage <- factor(ALLBTnames$BT) #the classes
exprs.gene<- t(probedat)   #use above data set of selected gene expression values
n<-length(B.stage)    ##size of the whole sample
mcr.cv.raw<-rep(NA, n) ## A vector to save misclassification rates (mcr) for each delete-one validation
for (i in 1:n) {
#delete each observation from training, then use it for testing.
  svmest <- svm(exprs.gene[-i,], B.stage[-i], type = "C-classification", kernel = "linear")
  svmpred <- predict(svmest, t(exprs.gene[i,])) #get prediction. transpose t() is used to make the vector into 1 by ncol matrix
  mcr.cv.raw[i]<- mean(svmpred!=B.stage[i]) #misclassification proportion
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all n rounds.
```

Run this R code and we get

```
> mcr.cv
[1] 0.2307692
```

So the cross-validated misclassification rate is much higher than the empirical misclassification rate of 1/78=0.013. **The SVM over-fitted here.** The classification model does not generalize well to future cases.

Demonstration of Support Vector Machine on ALL data (continued)

We can also do leave-ten-out cross validation.

```
######## leave-p-out cross-validation at p=10
B.stage <- factor(ALLBTnames$BT) #the classes
exprs.gene<- t(probedat)   #use above data set of selected gene expression values
n<-length(B.stage)    ##size of the whole sample
p<-10       #The number of test cases
nsim<-1000     #number of sampling runs for leave-p-out validation
mcr.cv.raw<-rep(NA, nsim) ## A vector to save raw mcr
for (i in 1:nsim) {
  testID<-sample(n, p, replace=FALSE) #sample p validation cases
   svmest <- svm(exprs.gene[-testID,], B.stage[-testID], type = "C-classification", kernel = "linear")
  svmpred <- predict(svmest, newdata=exprs.gene[testID,]) #use fit to get predicted classes
  mcr.cv.raw[i]<- mean(svmpred!=B.stage[testID]) #misclassification proportion
}
mcr.cv<-mean(mcr.cv.raw)
```

Run this R code and we get

```
> mcr.cv
[1] 0.2439
```

This is close to the leave-one-out cross-validated misclassification rate, and much higher than the empirical misclassification rate of 1/78=0.013.

Demonstration of Support Vector Machine on ALL data (cont'd)

We can also do 10-fold cross validation.

```
## 10-fold cross-validation
require(caret)
B.stage <- factor(ALLBTnames$BT) #the classes
exprs.gene<- t(probedat)   #use above data set of selected gene expression values
n<-length(B.stage)    ##size of the whole sample
index<-1:n #index for data points
K<-10      #number of folds
flds <- createFolds(index, k=K) #create K folds
mcr.cv.raw<-rep(NA, K) ## A vector to save raw mcr
for (i in 1:K) {
  testID<-flds[[i]]  #the i-th fold as validation set
  svmest <- svm(exprs.gene[-testID,], B.stage[-testID], type = "C-classification", kernel = "linear") #train SVM on training data
  svmpred <- predict(svmest, newdata=exprs.gene[testID,]) #use fit to get predicted classes
  mcr.cv.raw[i]<- mean(svmpred!=B.stage[testID]) #mcr on validation fold
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over K folds.
```

Run this R code and we get

```
> mcr.cv
[1] 0.2553571
```

This 10-fold cross-validated misclassification rate (mcr) varies a lot. Trying a few more times, we get mcr values ranging from 19.3% to 32.1%. However, the general pattern still holds: the cross-validated mcr is much higher than the empirical mcr=0.013. So we can observe the over-fitting with 10-fold cross-validation also. Too many genes were used in SVM here.

Support Vector Machine versus Logistic Regression

When there are only two classes, the logistic regression classify a case into Y=1 when $p(X) = e^{\beta_0+\beta_1 X_1+...+\beta_m X_m} / (1+e^{\beta_0+\beta_1 X_1+...+\beta_m X_m}) > 0.5$ which is equivalent to $\beta_0 + \beta_1 X_1 + ... + \beta_m X_m > 0$. So the logistic regression is also using a linear separating hyperplane, similar to support vector machine. In practice, these two classifiers often have similar performance in such situations.

Historically, the support vector machine grew from the computer science community. An interesting attractive feature is that it can also use nonlinear separating boundaries through use of a different kernel function. This is done in R implementation by, for example,

svm(X, Y, type = "C-classification", kernel = "polynomial")

Other kernel functions can also be used, use ?svm to see the R help file for what choices of kernel are available.

Mathematically, similar kernel tricks can be used on logistic regression. But kernel logistic regression is rarely used for historical reasons: the kernel support vector machine is already very popular. So, there is not much interest to implement a kernel logistic regression classifier with similar performance.

In this course, it suffices for you to know how to use support vector classifier in R.

Lesson Summary

This lesson first introduced support vector machine classifier. We discussed its basic goal: to find the best separating hyper plane (linear boundary). Also, it can find the best separating nonlinear boundary through specifying a kernel function.

You should now be able to use support vector machine in R to classify data. You should also be able to apply cross-validation to assess the real performance of the support vector machine classifier.

The next lesson introduces the classification tree.

Lesson 4: Classification Tree and Comparison of Classifiers

Objectives

By the end of this lesson you will have had the opportunity to:

- Fit the **classification tree** in R
- Use **cross-validation** to assess performance of the classification tree
- Compare logistic regression, SVM and classification tree on the iris data set with PCA

Overview

In this lesson, we will introduce the classification tree. We demonstrate the training of the classification tree in R. We then use cross-validation to evaluate its performance. Finally, we show an example of combining principal component analysis (PCA) with the classifiers. We compare the performance of logistic regression, SVM and classification tree on the iris data set.

Classification Tree

The classification tree classifies an object into a category by going through series of decision rules. These decision rules can be represented in a tree structure, and are simple to interpret. However, often they do not achieve the same prediction accuracy of other classification methods.

We show how to produce the classification tree with the R package "rpart". We will illustrate its application on the ALL data set next. If you did not install this package before, first install it using install.packages('rpart').

Demonstration of Classification Tree on ALL Data

We use the same subset of the Chiaretti (2004) ALL data set as in the last lesson's support vector machine example. The filtering codes are repeated here.

```
library("hgu95av2.db");library(ALL);data(ALL)            #load library
ALLB123 <- ALL[,ALL$BT %in% c("B1","B2","B3")]   #patients in
B1,B2,B3 stages
pano <- apply(exprs(ALLB123), 1, function(x) anova(lm(x ~
ALLB123$BT))$Pr[1]) #do ANOVA on every gene
names <- featureNames(ALL)[pano<0.000001]  #get probe names for genes
passing the ANOVA gene filter
symb <- mget(names, env = hgu95av2SYMBOL) #get the gene names
ALLBTnames <- ALLB123[names, ]   #keep only the gene passing filter
probedat <- as.matrix(exprs(ALLBTnames))  #save expression values in
probedat
row.names(probedat)<-unlist(symb) #change row names to gene names
```

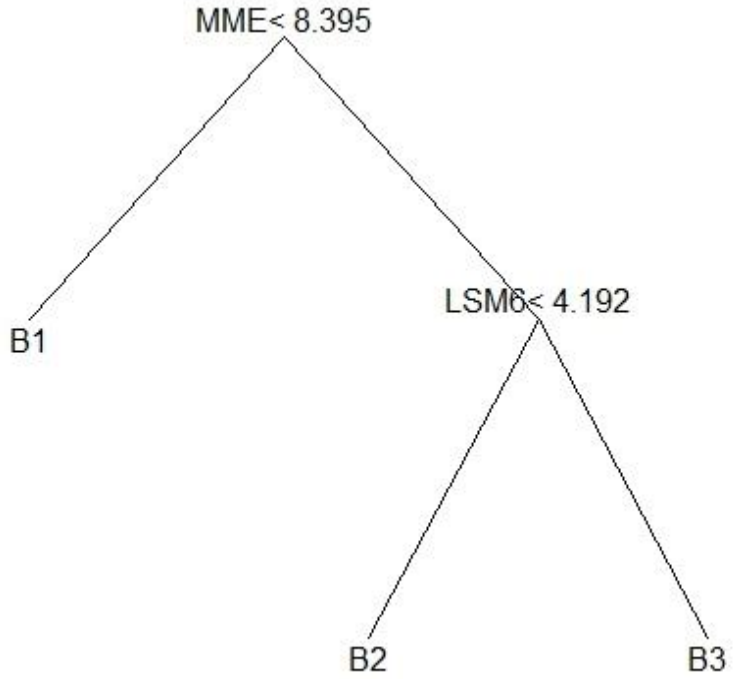
We now fit the classification tree

```
require(rpart)  #load library
B.stage <- factor(ALLBTnames$BT) #The true classes: B1,B2,B3 stages
c.tr <- rpart(B.stage ~ ., data = data.frame(t(probedat))) #Fit the tree on data
set 'probedat'. The transpose is needed since we are classifying the patients
(rows).
plot(c.tr, branch=0,margin=0.1); #Plot the tree with V-shaped branch (=0),
leave margins for later text additions.
text(c.tr, digits=3)  #Add text for the decision rules on the tree. use 3 digits
for numbers
rpartpred <- predict(c.tr, type="class")  #predicted classes from the tree
table(rpartpred, B.stage)   #confusion matrix compare predicted versus true
classes
```

We display the R outputs of running this code on the next page.

Demonstration of Classification Tree on ALL Data (Cont'd)

Running the R code from last page, we get a tree graph and a table.



	diagnosed		
rpartpred	B1	B2	B3
B1	17	2	0
B2	1	33	5
B3	1	1	18

We can see that the rules for this classification tree are very simple to interpret. If gene MME expression value is less than 8.395, the patient is classified as stage B1. Otherwise we then check the expression value of gene LSM6: less than 4.192 is classified into B2 stage, at least 4.192 is classified as B3 stage. Notice that the tree selects only two out of the 29 genes to make the decision.

From the table, we see that the misclassification rate is 10/78=0.13.

Next, we check the performance with leave-ten-out cross-validation.

Demonstration of Classification Tree on ALL Data (Cont'd)

We will carry out leave-ten-out cross validation for misclassification rate of the tree classifier.

```
######## leave-p-out cross-validation at p=10
B.stage <- factor(ALLBTnames$BT) #the classes
exprs.gene<- data.frame(t(probedat) )  #use above data set of selected gene expression values
n<-length(B.stage)    ##size of the whole sample
p<-10       #The number of test cases
nsim<-1000     #number of sampling runs for leave-p-out validation
tree.cv.raw<-rep(NA, nsim) ## A vector to save raw mcr
for (i in 1:nsim) {
  testID<-sample(n, p, replace=FALSE) #sample p validation cases
  tr.est <- rpart(B.stage[-testID] ~ ., data = exprs.gene[-testID,]) #Fit the tree on data leaving out the p validation cases
  tr.pred <- predict(c.tr, newdata=exprs.gene[testID,], type="class")
#predicted classes from the tree
  tree.cv.raw[i]<- mean(tr.pred!=B.stage[testID]) #misclassification proportion
}
tree.cv<-mean(tree.cv.raw)
```

Run this R code and we get

```
> tree.cv
[1] 0.1262
```

This leave-ten-out cross-validated misclassification rate is very close to the empirical misclassification rate of 10/78=0.13. Compared to the support vector machine fit we did before, the classification tree has larger empirical misclassification rate (0.13 versus 0.013), but a smaller cross-validated misclassification rate (0.13 versus 0.24). The tree classifier did not over-fit here.

In fact, the rpart() function used 10-fold cross-validation to control selection of the variables and choice of decision rules on the tree. This is programmed inside the function. The cross-validation can also be used to select variables in the support vector machine. Those are not programmed inside the svm() function, and you will need to program it yourself or find some other package to do so.

Classification after Dimension Reduction by PCA

In the previous example, we ran classification using 29 filter genes. The reason is that 12625 genes are too much to handle, and likely overfits since we only have 78 patients. We often want the number of predictor variables (also called features) to be smaller than the number of cases (sample size n=78 in above example). Therefore, we generally want to reduce the dimension first before applying a classifier.

Besides filtering, we covered another dimension reduction method in previous module: the principal component analysis (PCA). Instead of choosing some predictor variables, we can use the first K principal components as predictor variables. We illustrate this on the iris data next.

Classification after PCA on Iris Data

In the first lesson of this module, we used the iris data set for classification examples. We use the first four numerical variables: sepal length, sepal width, petal length and petal width, to predict the last variable "Species" which has values of three classes. To reduce the number of variables used for classification, we can conduct the principal component analysis (PCA).

```
> pca.iris<-prcomp(iris[,1:4], scale=TRUE) #Apply PCA on first four variables in iris data, scale each variable first
> summary(pca.iris)  #print out summary of the PCA fit
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

We see that the first two principal components explain 96% of total variation, and the first three principal components explain 99% of total variation. The last component can be ignored without losing much information. We prepare a reduced data set with the first three principal components, by taking the first three columns in the $x field of the fitted PCA object in R.

```
Species<-iris$Species        #response variable with true classes
data.pca<-pca.iris$x[,1:3]  #keep only the first three principal components
n<-length(Species)           #sample size n
```

Next we apply the classification tree on the reduced data set. We report the empirical misclassification rate (mcr) and the leave-one-out cross-validated mcr.

Classification Tree after PCA on Iris Data

One the reduced data set with only the first three principal components, we apply classification tree to predict the species.

```
iris2<-data.frame(Species, data.pca)      #combine response variable with
PCA data
fit <- rpart(Species ~ ., data = iris2, method = "class") #Fit tree iris2 data
pred.tr<-predict(fit, iris2, type = "class")  #predict classes from the tree
mcr.tr <- mean(pred.tr!=Species)   #misclassification rate
### leave-one-out cross validation
mcr.cv.raw<-rep(NA, n)        #A vector to save mcr validation
for (i in 1:n) {
  fit.tr <- rpart(Species ~ ., data = iris2[-i,], method = "class") #train the tree
without i-th observation
  pred <- predict(fit.tr, iris2[i,], type = "class")#use tree to predict i-th
observation class
  mcr.cv.raw[i]<- mean(pred!=Species[i]) #check misclassifion
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all n rounds.
```

Run the R code, and we can get the empirical mcr and the leave-one-out cross-validated mcr.

```
> c(mcr.tr, mcr.cv)

[1] 0.06666667 0.14000000
```

The tree has a low 6.7% misclassification rate. But it overfitted, the leave-one-out cross-validated mcr=14% is much higher.

Next we will apply the logistic regression on the PCA reduced data set.

Logistic Regression after PCA on Iris Data

One the reduced data set with only the first three principal components, we apply logistic regression to predict the species.

```
iris2.lgr <- vglm(Species~., family=multinomial, data=iris2) #Logistic Regression
pred.prob <- predict(iris2.lgr, iris2[,-1], type="response") #get prediction probability for all cases in data set
pred.lgr <- apply(pred.prob, 1, which.max) #Assign to the class with largest prediction probability
pred.lgr <- factor(pred.lgr, levels=c("1","2","3"), labels=levels(iris2$Species)) #relabel 1,2,3 by species names
mcr.lgr <- mean(pred.lgr!=iris2$Species) #misclassification rate
### leave-one-out cross validation
mcr.cv.raw<-rep(NA, n)   #A vector to save mcr validation
for (i in 1:n) {
  fit.lgr <-  vglm(Species~., family=multinomial, data=iris2[-i,]) #fit logistic regression without i-th observation
  pred.prob <- predict(fit.lgr, iris2[i,-1], type="response") #get prediction probability
  pred <- apply(pred.prob, 1, which.max) #Assign class
  pred <- factor(pred, levels=c("1","2","3"), labels=levels(iris2$Species)) #relabel 1,2,3 by species names
  mcr.cv.raw[i]<- mean(pred!=Species[i]) #check misclassification
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all n rounds.
```

Run the R code, and we can get the empirical mcr and the leave-one-out cross-validated mcr.

```
> c(mcr.lgr, mcr.cv)
[1] 0.01333333 0.02666667
```

The logistic regression classifier has a low 1.3% misclassification rate. The leave-one-out cross-validated mcr=2.7% is bigger.

Next we will apply the SVM on the PCA reduced data set.

Support Vector Machine (SVM) after PCA on Iris Data

One the reduced data set with only the first three principal components, we apply SVM to predict the species.

```
iris2.svm <- svm(data.pca, Species, type = "C-classification", kernel =
"linear") #train SVM
svmpred <- predict(iris2.svm , data.pca) #get SVM prediction.
mcr.svm<- mean(svmpred!=Species) #misclassification rate
### leave-one-out cross validation
mcr.cv.raw<-rep(NA, n)     #A vector to save mcr validation
for (i in 1:n) {
  svmest <- svm(data.pca[-i,], Species[-i], type = "C-classification", kernel =
"linear") #train SVM without i-th observation
  svmpred <- predict(svmest, t(data.pca[i,])) #predict i-th observation. Here
transpose t() is used to make the vector back into a 1 by ncol matrix
  mcr.cv.raw[i]<- mean(svmpred!=Species[i]) #misclassification rate
}
mcr.cv<-mean(mcr.cv.raw) #average the mcr over all n rounds.
```

Run the R code, and we can get the empirical mcr and the leave-one-out cross-validated mcr.

```
> c(mcr.svm, mcr.cv)

[1] 0.02666667 0.04666667
```

The SVM classifier has a 2.7% misclassification rate. The leave-one-out cross-validated mcr=4.7% is bigger.

Compare the three classifiers, the logistic regression classifier with PCA reduced data performs the best on the iris data set.

No one classifier always performs best. On iris data, the logistic regression does best, while classification tree does better on the B-cell stage ALL data set as earlier. To judge the performance, we do not depend on the empirical misclassification rate which may reflect over-fitted performance. We choose the classifier with better performance using the cross-validated misclassification rate.

Lesson Summary

This lesson introduced the classification tree. You should now be able to fit it in R, and be able to interpret the fitted tree. You should also be able to use cross-validation to assess the real performance of the classifier.

Finally, we illustrated how to combine the classifier with the dimension reduction technique PCA.

Module Summary

This module covered classification methods including logistic regression, support vector machine and classification tree. You should know how to train these classifiers on data in R.

You should also have understanding of statistical inferences from the logistic regression. And know how to plot the classification tree.

We covered how to evaluate the performance of a classifier. You should be able to calculate the numerical measure for summarizing the performance. We introduced the concept of validation. You should be able to carry out leave-p-out and K-fold cross-validation, and use these to evaluate the classifiers.