Chapter 21
Prediction and Internal Statistical Cross Validation

We should start by reviewing Chap. 14 (Model Performance Assessment). Cross-validation is a statistical approach for validating predictive methods, classification models, and clustering techniques. It assesses the reliability and stability of the results of the corresponding statistical analyses (e.g., predictions, classifications, forecasts) based on independent datasets. For prediction of trend, association, clustering, and classification, a model is usually trained on one dataset (*training data*) and subsequently tested on new data (*testing or validation data*). Statistical internal cross-validation uses iterative bootstrapping to define test datasets, evaluates the model predictive performance, and assesses its power to avoid overfitting. *Overfitting* is the process of computing a predictive or classification model that describes random error, i.e., fits to the noise components of the observations, instead of the actual underlying relationships and salient features in the data.

In this Chapter, we will use the Google Flu Trends, Autism, and Parkinson's disease case-studies to (1) illustrate exhaustive and non-exhaustive internal statistical cross-validation; (2) explore alternative forecasting types using linear and non-linear predictions; and (3) compare complementary predictor functions.

21.1 Forecasting Types and Assessment Approaches

In Chap. 7, we discussed the types of classification and prediction methods, including `supervised` and `unsupervised` learning. The former are direct and predictive, as there are known outcome variables that can be predicted, and the corresponding forecasts can be evaluated. The latter are indirect and descriptive, as there are no *a priori* labels or specific outcomes.

There are alternative metrics used for evaluation of model performance, see Chap. 14. For example, assessment of supervised prediction and classification methods depends on the type of the labeled outcome responses: `categorical` (binary or polytomous) vs. `continuous`.

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_21

- `Confusion matrices` reporting accuracy, FP, FN, PPV, NPV, LOR and other metrics may be used to assess predictions of dichotomous (binary) or `polytomous outcomes`.
- R^2, correlations (between predicted and observed outcomes), and RMSE measures may be used to quantify the performance of various supervised forecasting methods on `continuous features`.

21.2 Overfitting

Before we go into the cross-validation of predictive analytics, we will present several examples of *overfitting* that illustrate why a certain amount of skepticism and mistrust may be appropriate when dealing with forecasting models that are based on large and complex data.

21.2.1 Example (US Presidential Elections)

By 2017, there were only **57 US presidential elections** and **45 presidents**. That is a small dataset, and learning from it may be challenging. For instance:

- If the predictor space expands to include things like *having false teeth*, it's pretty easy for the model to go from fitting the generalizable features of the data (the signal, e.g., presidential actions) to matching noise patterns (e.g., irrelevant characteristics like gender of the children of presidents, or types of dentures they may wear).
- When overfitting noise patterns takes place, the quality of the model fit assessed on the historical data may improve (e.g., better R^2, more about the Coefficient of Determination is available online). At the same time, however, the model performance may be suboptimal when used to make inference about prospective data, e.g., future presidential elections.

Figure 21.1 shows a cartoon that includes some of the (unique) noisy presidential characteristics that are thought to be unimportant to electability, fitness for office, or expectations of presidential performance.

21.2.2 Example (Google Flu Trends)

A March 14, 2014 article in Science (DOI: https://doi.org/10.1126/science.1248506), identified problems in a Google Flu Trends (GFT) study, DOI https://doi.org/10.1371/journal.pone.0023610, which may be attributed in part to

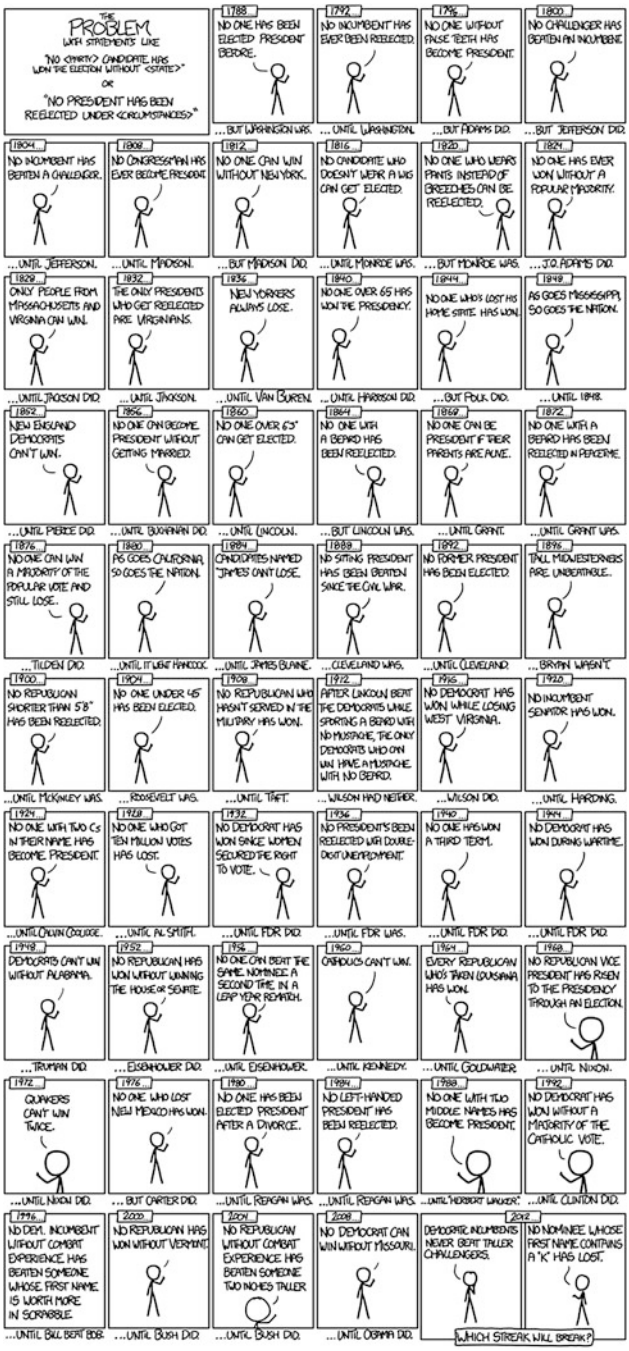

Fig. 21.1 Example of an overfitting based on extreme stratification of traits of presidential candidates

overfitting. The GFT model was built to predict the future Centers for Disease Control and Prevention (CDC) reports of doctor office visits for influenza-like illness (ILI). In February 2013, Nature reported that GFT was predicting more than double the proportion of doctor visits compared to the CDC forecast for the same period.

The GFT model found the best matches among 50 million web search terms to fit 1,152 data points. It predicted quite high odds of finding search terms that match the propensity of the flu, but which are structurally unrelated, and hence are not prospectively predictive. In fact, the GFT investigators reported weeding out seasonal search terms that were unrelated to the flu, which may have been strongly correlated to the CDC data, e.g., high school basketball season. The big GFT data may have overfitted the relatively small number of cases. This false-alarm result was also paired with a false-negative finding. The GFT model also missed the non-seasonal 2009 H1N1 influenza pandemic. This provides a cautionary tale about prediction, overfitting, and prospective validation.

21.2.3 Example (Autism)

Autistic brains constantly overfit visual and cognitive stimuli. To an autistic person, a general conversation of several adults may seem like a cacophony due to super-sensitive detail-oriented hearing and perception tuned to literally pick up all elements of the conversation and clues of the surrounding environment. At the same time, autistic brains may downplay body language, sarcasm, and non-literal cues. We can `miss the forest for the trees` when we start "overfitting", i.e., when we over interpret the noise on top of the actual salient information. Ambient noise, trivial observations, and unrelated perceptions may obfuscate the true communication details.

Human conversations and communications involve exchanges of both critical information and random noise. Fitting a perfect model requires focus only on the "relevant" information. Overfitting occurs when attention is (excessively) consumed with peripheral noise, or worse, overwhelmed by inconsequential noise drowning the salient aspects of the communication exchange.

Any dataset is a mix of signal and noise. The main task of our brains is to sort these components and interpret the useful information while ignoring the noise. However, we should be cognizant that

"One person's noise is another person's treasure map!"

Our predictions are most accurate if we can model as much of the signal and as little of the noise as possible. Note that in these terms, R^2 is a poor metric to identify predictive power – it measures how much of the signal *and* the noise is explained by our model. In practice, it's hard to always identify what's signal and what's noise. This is why practical applications tend to favor simpler models, since the more complicated a model is, the easier it is to overfit the noise component of the observed information.

21.3 Internal Statistical Cross-Validation is an Iterative Process

Internal statistical cross-validation assesses the expected performance of a prediction method in cases (e.g., subjects, units, regions, etc.) drawn from a similar population as the original training data sample. `Internal` validation is distinct from `external` validation, as the latter potentially allows for the existence of differences between the populations: training data, used to develop, or train, the technique, and testing data, used to independently quantify the performance of the technique.

Each step in the internal statistical cross-validation protocol involves:

- Randomly partitioning a sample of data into 2 complementary subsets (training + testing),
- Performing the analysis, fitting or estimating the model using the training set,
- Validating the analysis or evaluating the performance of the model using the separate testing set,
- Increasing the iteration index and repeating the process. Various termination criterial can be chosen like a fixed number of iterations, a desired mean variability, or an upper bound on the error-rate.

One example of internal statistical cross-validation used for predictive diagnostic modeling in Parkinson's disease is available online.

To reduce the noise and variability at each iteration, the final validation results may include the averaged performance results across iterations.

In cases when new observations are hard to obtain (due to costs, reliability, time, or other constraints), cross-validation guards against testing hypotheses suggested by the data themselves (also known as `Type III error` or False-Suggestion).

Cross-validation is different from *conventional-validation* (e.g. 80–20% partitioning the data set into training and testing subsets) where the prediction error (e.g., Root Mean Square Error, RMSE) evaluated on the training data is not a useful estimator of model performance, as it does not generalize across multiple samples.

In general, the errors of the conventional-valuation are based on the results of a specific test dataset and may not accurately represent the model performance. A more appropriate strategy to properly estimate model prediction performance is to use cross-validation (CV), which combines (e.g., averages) multiple prediction errors to measure the expected model performance. CV corrects for the expected stochastic nature of partitioning the training and testing sets and generates a more accurate and robust estimate of the expected model performance.

Relative to a simpler model, a more complex model may *overfit-the-data* if it has a short foresight, i.e., it may generate accurate fitting results for known data but less accurate results when predicting based on new data. Knowledge from past experiences may include either *relevant* or *irrelevant* (noise) information. In challenging data-driven prediction models where the uncertainty (entropy) is high, more noise is present in past information that needs to be accounted for in prospective

forecasting. However, it is generally hard to discriminate patterns from noise in complex systems, which makes it difficult to decide what part to model and what to ignore. Models that reduce the chance of fitting noise are called **robust**.

21.4 Example (Linear Regression)

Let's demonstrate a simple model assessment using linear regression. Suppose we observe the response values $\{y_1, \cdots, y_n\}$, and the corresponding k predictors represented as a kD vector of covariates $\{x_1, \cdots, x_n\}$, where subjects/cases are indexed by $1 \leq i \leq n$, and the data-elements (variables) are indexed by $1 \leq j \leq k$.

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{pmatrix}.$$

Using least squares to estimate the linear function parameters (effect-sizes), $\beta_1, \cdots, \beta_k$, allows us to compute a hyperplane $y = a + x\beta$ that best fits the observed data $(x_i, y_i)_{1 \leq i \leq n}$. This is expressed as a matrix by:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_k \end{pmatrix}.$$

Corresponding to the system of linear hyperplanes:

$$\begin{cases} y_1 = a_1 + x_{1,1}\beta_1 + x_{1,2}\beta_2 + \cdots + x_{1,k}\beta_k \\ y_2 = a_2 + x_{2,1}\beta_1 + x_{2,2}\beta_2 + \cdots + x_{2,k}\beta_k \\ \qquad\qquad\qquad \vdots \\ y_n = a_1 + x_{n,1}\beta_1 + x_{n,2}\beta_2 + \cdots + x_{n,k}\beta_k \end{cases}.$$

One measure to evaluate the model fit may be the mean squared error (MSE). The MSE for a given value of the parameters α and β on the observed training data $(x_i, y_i)_{1 \leq i \leq n}$ is expressed as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \underbrace{\left(a_1 + x_{i,1}\beta_1 + x_{i,2}\beta_2 + \cdots + x_{i,k}\beta_k \right)}_{\text{predicted value } \hat{y}_i \text{ at } x_{i,1}, \cdots, x_{i,k}} \right)^2.$$

And the corresponding root mean square error (RMSE) is:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \underbrace{\left(a_1 + x_{i,1}\beta_1 + x_{i,2}\beta_2 + \cdots + x_{i,k}\beta_k\right)}_{\text{predicted value } \hat{y}_i \text{ at } x_{i,1},\cdots,\ x_{i,k}}\right)^2}.$$

In the linear model case, the expected value of the MSE (over the distribution of training sets) for the **training set** is $\frac{n-k-1}{n+k+1}E$, where E is the expected value of the MSE for the **testing/validation data**. Therefore, fitting a model and computing the MSE on the training set, we may produce an over optimistic evaluation assessment (smaller RMSE) of how well the model may fit another dataset. This bias represents *in-sample* estimate of the fit, whereas we are interested in the cross-validation estimate as an *out-of-sample* estimate.

In the linear regression model, cross validation may not be as useful, since we can compute the **exact** correction factor $\frac{n-k-1}{n+k+1}$ to obtain an estimate of the (unknown) exact expected *out-of-sample* fit using the (known) *in-sample* MSE (under)estimate. However, even in this situation, cross-validation remains useful as it can be used to select an optimal regularized cost function.

In most other modeling procedures (e.g. logistic regression), there are no simple general closed-form expressions (formulas) to adjust the cross-validation error estimate of the known in-sample fit to estimate the unknown out-of-sample error rate. Cross-validation is general strategy to predict the performance of a model on a validation set using stochastic computation instead of obtaining experimental, theoretical, mathematical, or closed-form analytic error estimates.

21.4.1 Cross-Validation Methods

There are two classes of cross-validation approaches, *exhaustive* and *non-exhaustive*.

21.4.2 Exhaustive Cross-Validation

Exhaustive cross-validation methods are based on determining all possible ways to divide the original sample into training and testing data. For instance, the *Leave-m-out cross-validation* involves using m observations for testing and the remaining $(n - m)$ observations as training. The case when $m = 1$, i.e., leave-one-out method, is only applicable when n is small, due to its huge computational cost. This process is repeated on all partitions of the original sample. This method requires model fitting and validating C_m^n times (n is the total number of observations in the original sample and m is the number of observations left out for validation). This requires a very large number of iterations.

21.4.3 Non-Exhaustive Cross-Validation

Non-exhaustive cross validation methods use bootstrap approximation to avoid computing estimates/errors using all possible partitionings of the original sample. For example, in the **k-fold cross-validation**, the original sample is randomly partitioned into k equal sized subsamples, or *folds*. Of all k subsamples, a single subsample is kept as final testing data for validation of the model. The other $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times, corresponding to the k folds. Each of the k subsamples is used once as the validation data. In the end, the corresponding k results are averaged (or otherwise aggregated) to generate a final pooled model-quality estimation. In k-fold validation, all observations are used for both training and validation, and each observation is used for validation exactly once. In general, k is a parameter that needs to be selected by the investigator (common values may be 5 or 10).

A general case of the `k-fold validation` is $k = n$ (the total number of observations), when it coincides with the **leave-one-out cross-validation**.

A variation of the k-fold validation is **stratified k-fold cross-validation**, where each fold has (approximately) the same mean response value. For instance, if the model represents a binary classification of cases (e.g., controls vs. patients), this implies that each fold contains roughly the same proportion of the two class labels.

Repeated random sub-sampling validation splits randomly the entire dataset into a training set, where the model is fit, and a testing set, where the predictive accuracy is assessed. Again, the results are averaged over all iterative splits. This method has an advantage over k-fold cross validation, as the proportion of the training/testing split is not dependent on the number of iterations (folds). However, its drawback is that some observations may never be selected in the testing/validation subsample, whereas others may be selected multiple times. As validation subsets may overlap, the results may vary each time we repeat the validation protocol, unless we set a seed point in the algorithm.

Asymptotically, as the number of random splits increases, the *repeated random sub-sampling* validation approaches the *leave-k-out cross-validation*.

21.5 Case-Studies

In the examples below, we have intentionally suppressed some of the R output to save space. This is accomplished using this Rmarkdown command, `{r eval=TRUE, results='hide'}`, however, the reader is encouraged to try hands-on all the protocols, to make modifications, to inspect, and finally to interpret the outputs.

21.5.1 Example 1: Prediction of Parkinson's Disease Using `Adaptive Boosting` (AdaBoost)

This Parkinson's Diseases study, which involves heterogeneous neuroimaging, genetics, clinical, and phenotypic data for over 600 volunteers including multivariate data for three cohorts (HC=Healthy Controls, PD=Parkinson's, SWEDD = subjects without evidence for dopaminergic deficit).

```
# update packages
# update.packages()
```

Load the data: 06_PPMI_ClassificationValidationData_Short.csv.

```
ppmi_data <-
read.csv("https://umich.instructure.com/files/330400/download?download_frd=1",
header=TRUE)
```

Binarize the Dx (clinical diagnosis) classes.

```
# binarize the Dx classes
ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control",
"Control", "Patient")
attach(ppmi_data)

head(ppmi_data)
# View (ppmi_data)
```

Obtain a model-free predictive analytics, e.g., AdaBoost classification, and report the results.

```
# Model-free analysis, classification
# install.packages("crossval")
# install.packages("ada")
# library("crossval")
require(crossval)
require(ada)
#set up adaboosting prediction function

# Define a new AdaBoost classification result-reporting function
my.ada <- function (train.x, train.y, test.x, test.y, negative, formula){
  ada.fit <- ada(train.x, train.y)
  predict.y <- predict(ada.fit, test.x)
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
 # negative  is the label of a negative "null" sample (default: "control").
  return (out)
}
```

When group sizes are imbalanced, we may need to rebalance them to avoid potential biases of the dominant cohorts. In this case, we will re-balance the groups using the package `SMOTE` Synthetic Minority Oversampling Technique. SMOTE may be used to handle class imbalance in binary classification, see Chap. 3.

```
# balance cases
# SMOTE: Synthetic Minority Oversampling Technique to handle class misbalanc
e in binary classification.
set.seed(1000)
# install.packages("unbalanced") to deal with unbalanced group data
require(unbalanced)
ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
uniqueID <- unique(ppmi_data$FID_IID)
ppmi_data <- ppmi_data[ppmi_data$VisitID==1, ]
ppmi_data$PD <- factor(ppmi_data$PD)

colnames(ppmi_data)
# ppmi_data.1<-ppmi_data[, c(3:281, 284, 287, 336:340, 341)]
n <- ncol(ppmi_data)
output.1 <- ppmi_data$PD

# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
# remove Default Real Clinical subject classifications!
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD",
"X", "FID_IID"))]
# output <- as.matrix(ppmi_data[, which(names(ppmi_data) %in% {"PD"})])
output <- as.factor(ppmi_data$PD)
c(dim(input), dim(output))

#balance the dataset
set.seed(123)
data.1<-ubBalance(X= input, Y=output, type="ubSMOTE", percOver=300, percUnde
r=150, verbose=TRUE)
balancedData<-cbind(data.1$X, data.1$Y)
table(data.1$Y)

nrow(data.1$X); ncol(data.1$X)
nrow(balancedData); ncol(balancedData)
nrow(input); ncol(input)

colnames(balancedData) <- c(colnames(input), "PD")
```

Next, we'll check the re-balanced cohort sizes (Fig. 21.2).

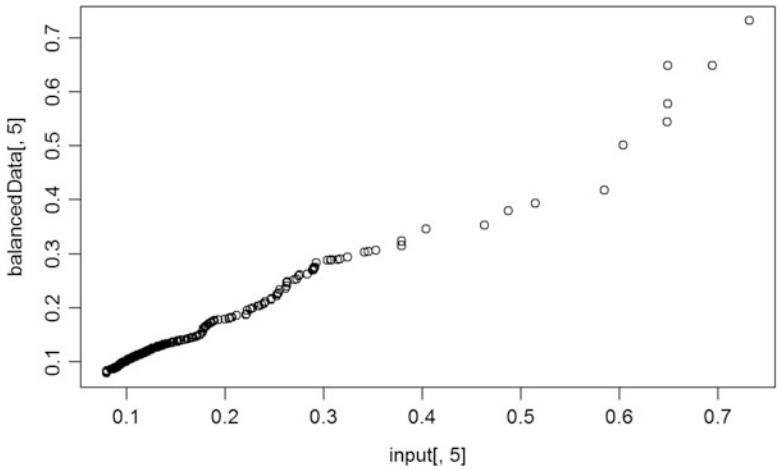

Fig. 21.2 Quantile-quantile plot of the original and rebalanced data distributions for one feature

```
###Check balance
## T test
alpha.0.05 <- 0.05
test.results.bin <- NULL        # binarized/dichotomized p-values
test.results.raw <- NULL        # raw p-values

# get a better error-handling t.test function that gracefully handles NA's a
nd trivial variances
my.t.test.p.value <- function(input1, input2) {
obj <- try(t.test(input1, input2), silent=TRUE)
if (is(obj, "try-error"))
  return(NA)
else
  return(obj$p.value)
}

for (i in 1:ncol(balancedData))
{
test.results.raw[i]  <- my.t.test.p.value(input[, i], balancedData [, i])
    test.results.bin[i] <- ifelse(test.results.raw[i]  > alpha.0.05, 1, 0)
# binarize the p-value (0=significant, 1=otherwise)
    print(c("i=",i,"var=", colnames(balancedData[i]), "t-test_raw_p_value=",
test.results.raw[i]))
}

# we can also employ (e.g., FDR, Bonferonni) correction for multiple
testing!
    # test.results.corr <- stats::p.adjust(test.results.raw, method = "fdr",
n = length(test.results.raw))
    # where methods are "holm", "hochberg", "hommel", "bonferroni", "BH",
"BY", "fdr", "none")
    # plot(test.results.raw, test.results.corr)
    # sum(test.results.raw < alpha.0.05, na.rm=T)/length(test.results.raw)
#check proportion of inconsistencies
# sum(test.results.corr < alpha.0.05, na.rm =T)/length(test.results.corr)

qqplot(input[, 5], balancedData [, 5]) # check visually for differences
between the distributions of the raw (input) and rebalanced data (for only
one variable, in this case)
```

```
# Now, check visually for differences between the distributions of the raw
(input) and rebalanced data.
# par(mar=c(1,1,1,1))
# par(mfrow=c(10,10))
# for(i in c(1:62,64:101)){ qqplot(balancedData [, i],input[, i]) } #except
VisitID
# as the sample-size is changed:
length(input[, 5]); length(balancedData [, 5])
# to plot raw vs. rebalanced pairs (e.g., var="L_insular_cortex_Volume"), we
need to equalize the lengths
#plot (input[, 5] +0*balancedData [, 5], balancedData [, 5])   # [, 5] ==
"L_insular_cortex_Volume"

# print(c("T-test results: ", test.results))
# zeros (0) are significant independent between-group T-test differences,
ones (1) are insignificant

for (i in 1:(ncol(balancedData)-1))
{
```

```
    test.results.raw [i]  <- wilcox.test(input[, i], balancedData [,
i])$p.value
    test.results.bin [i] <- ifelse(test.results.raw [i] > alpha.0.05, 1, 0)
    print(c("i=", i, "Wilcoxon-test=", test.results.raw [i]))
}
print(c("Wilcoxon test results: ", test.results.bin))
# test.results.corr <- stats::p.adjust(test.results.raw, method = "fdr", n =
length(test.results.raw))
# where methods are "holm", "hochberg", "hommel", "bonferroni", "BH", "BY",
"fdr", "none")
# plot(test.results.raw, test.results.corr)
```

The next step will be the actual `cross-validation`.

```
# using raw data:
X <- as.data.frame(input); Y <- output
neg <- "1"   # "Control" == "1"

# using Rebalanced data:
X <- as.data.frame(data.1$X); Y <- data.1$Y
# balancedData<-cbind(data.1$X, data.1$Y); dim(balancedData)

# Side note: There is a function name collision for "crossval", the same met
hod is present in the "mlr" (machine Learning in R) package and in the "cros
sval" package.
# To specify a function call from a specific package do:  packagename::funct
ionname()

set.seed(115)
cv.out <- crossval::crossval(my.ada, X, Y, K = 5, B = 1, negative = neg)
    # the label of a negative "null" sample (default: "control")
out <- diagnosticErrors(cv.out$stat)

print(cv.out$stat)

##    FP    TP    TN    FN
##   0.6 109.6  97.0   0.2

print(out)

##        acc       sens       spec        ppv        npv        lor
##  0.9961427  0.9981785  0.9938525  0.9945554  0.9979424 11.3918119
```

As we can see from the reported metrics, the overall averaged AdaBoost-based diagnostic predictions are quite good.

21.5.2 Example 2: Sleep Dataset

These data contain the effect of two soporific drugs to increase hours of sleep (treatment-compared design) on 10 patients. The data are available in R by default (`sleep {datasets}`).

First, load the data and report some graphs and summaries (Fig. 21.3).

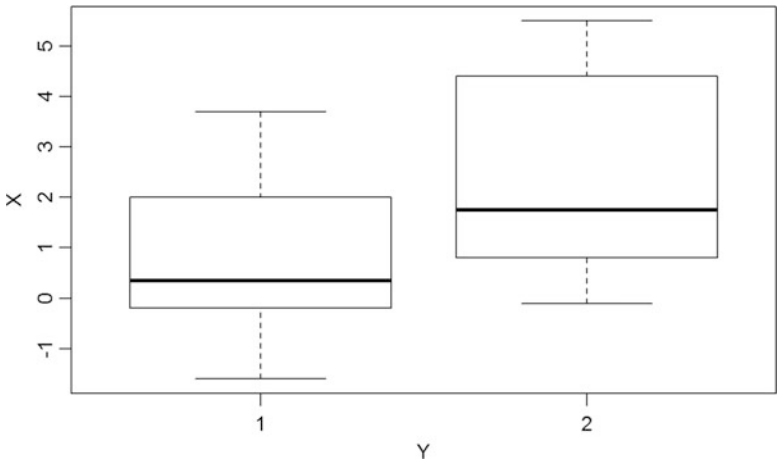

Fig. 21.3 Box-and whisker plots of the hours of sleep for the two cohorts in the sleep dataset

```
data(sleep); str(sleep)
X = as.matrix(sleep[, 1, drop=FALSE]) # increase in hours of sleep,
    # drop is logical, if TRUE the result is coerced to the lowest possible
dimension.
# The default is to drop if only one column is left, but not to drop if only
one row is left.
Y = sleep[, 2] # drug given
plot(X ~ Y)
```

```
levels(Y) # "1" "2"
dim(X) # 20  1
```

Next, we will define a new LDA (linear discriminant analysis) predicting function and perform the `cross-validation` (CV) on the resulting predictor.

```
require("MASS") # for lda function

predfun.lda = function(train.x, train.y, test.x, test.y, negative)
{   lda.fit = lda(train.x, grouping=train.y)
    ynew = predict(lda.fit, test.x)$class
# count TP, FP etc.
    out = confusionMatrix(test.y, ynew, negative=negative)
return( out )
}

# install.packages("crossval")
library("crossval")

set.seed(123456)
cv.out <- crossval::crossval(predfun.lda, X, Y, K=5, B=20, negative="1",
verbose=FALSE)
cv.out$stat
diagnosticErrors(cv.out$stat)
```

Execute the above code and interpret the diagnostic results measuring the performance of the LDA prediction.

21.5.3 Example 3: Model-Based (Linear Regression) Prediction Using the `Attitude` Dataset

These data represent a survey of clerical employees of an organization with 35 employees in 30 (randomly selected) departments. The data include the proportion of favorable responses to 7 questions in each department.

Let's load and summarize the data, which is available in the R {`datasets`} as `attitude`.

```
# ?attitude, colnames(attitude)
# Note: when using a data frame, a time-saver is to use "." to indicate "
include all covariates" in the DF.
# E.g., fit <- lm(Y ~ ., data = D)

data("attitude")
y = attitude[, 1]       # rating variable
x = attitude[, -1]  # date frame with the remaining variables
is.factor(y)
summary( lm(y ~ . , data=x) )   # R-squared:  0.7326
# set up lm prediction function
```

We will demonstrate model-based analytics using `lm` and `lda`, and then will validate the forecasting using CV.

```
predfun.lm = function(train.x, train.y, test.x, test.y)
{     lm.fit = lm(train.y ~ . , data=train.x)
       ynew = predict(lm.fit, test.x )
                    # compute squared error risk (MSE)
       out = mean( (ynew - test.y)^2)
            # note that, in general, when fitting linear model to continuous
outcome variable (Y),
            # we can't use the out<-confusionMatrix(test.y, ynew, negative=n
egative), as it requires a binary outcome
            # this is why we use the MSE as an estimate of the discrepancy b
etween observed & predicted values
return(out)
}
# require("MASS")
#predfun.lda = function(train.x, train.y, test.x, test.y, negative)
#{  lda.fit = lda(train.x, grouping=train.y)
#   ynew = predict(lda.fit, test.x)$class
# count TP, FP etc.
#   out = confusionMatrix(test.y, ynew, negative=negative)
#return( out )
#}
```

```
# prediction MSE using all variables
set.seed(123456)
cv.out.lm = crossval::crossval(predfun.lm, x, y, K=5, B=20, verbose=FALSE)
c(cv.out.lm$stat, cv.out.lm$stat.se)            # 72.581198  3.736784
# reducing to using only two variables
cv.out.lm = crossval::crossval(predfun.lm, x[, c(1, 3)], y, K=5, B=20, verbo
se=FALSE)
c(cv.out.lm$stat, cv.out.lm$stat.se)
# 52.563957  2.015109
```

21.5.4 Example 4: Parkinson's Data (`ppmi_data`)

Let's go back to the more elaborate PD data and start by loading and preprocessing the derived-PPMI data.

```
# ppmi_data <-read.csv("https://umich.instructure.com/files/330400/download?
download_frd=1", header=TRUE)
# ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control", "C
ontrol", "Patient")
# attach(ppmi_data); head(ppmi_data)
# install.packages("crossval")
# library("crossval")
# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
# input <- ppmi_data[ , -which(names(ppmi_data) %in% c("ResearchGroup",
"PD", "X", "FID_IID"))]
# output <- as.factor(ppmi_data$PD)

# remove the irrelevant variables (e.g., visit ID)
output <- as.factor(ppmi_data$PD)
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD", "
X", "FID_IID", "VisitID"))]
```

```
X = as.matrix(input)    # Predictor variables
Y = as.matrix(output)   #  Actual PD clinical assessment
dim(X);
dim(Y)

layout(matrix(c(1, 2, 3, 4), 2, 2))         # optional 4 graphs/page
fit <- lm(Y~X);
plot(fit)           # plot the fit
```

```
levels(as.factor(Y))            # "0" "1"

## [1] "0" "1"

c(dim(X), dim(Y))           # 1043  103

## [1] 422 100 422   1
```

Apply `cross-validation` to assess the performance of the linear model (Fig. 21.4).

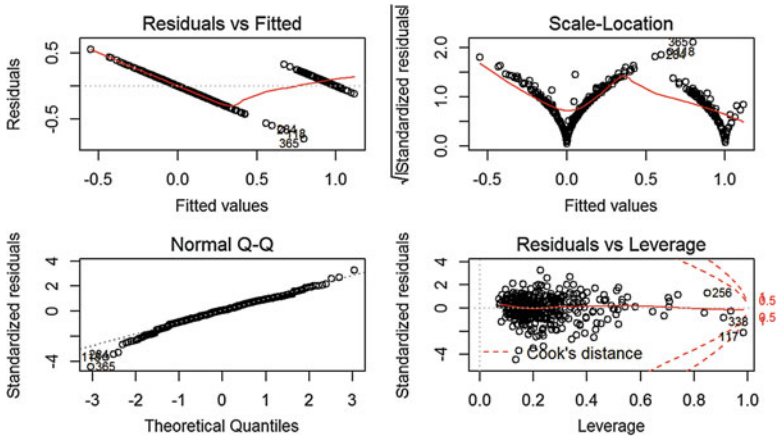

Fig. 21.4 Residual plots provide exploratory analytics of the model quality

```
set.seed(12345)
# cv.out.lm = crossval::crossval(predfun.lm, as.data.frame(X), as.numeric(Y)
, K=5, B=20)

cv.out.lda = crossval::crossval(predfun.lda, X, Y, K=5, B=20, negative="1",
verbose=FALSE)
# K=Number of folds; B=Number of repetitions.

# Results
#cv.out.lda$stat;
#cv.out.lda;
diagnosticErrors(cv.out.lda$stat)

##       acc      sens      spec       ppv       npv       lor
## 0.9617299 0.9513333 0.9872951 0.9945984 0.8918919 7.3258500

#cv.out.lm$stat;
#cv.out.lm;
#diagnosticErrors(cv.out.lm$stat)
```

21.6 Summary of CV output

The cross-validation (CV) output object includes the following three components:

- `stat.cv`: Vector of statistics returned by predfun for each cross validation run.
- `stat`: Mean statistic returned by predfun averaged over all cross validation runs.
- `stat.se`: Variability measuring the corresponding standard error.

21.7 Alternative Predictor Functions

We have already seen a number of `predict()` functions, e.g., Chap. 18. Below, we will add to the collection of predictive analytics and forecasting functions.

21.7.1 Logistic Regression

We already saw the *logit* model in Chap. 18. Now, we will demonstrate a logit-predictor function by applying it to the PD dataset.

```
# ppmi_data <-read.csv("https://umich.instructure.com/files/330400/download?
download_frd=1", header=TRUE)
# ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control",
"Control", "Patient")
# install.packages("crossval"); library("crossval")
# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)

# remove the irrelevant variables (e.g., visit ID)
output <- as.factor(ppmi_data$PD)
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD",
"X", "FID_IID", "VisitID"))]
X = as.matrix(input)    # Predictor variables
Y = as.matrix(output)
```

Note that the predicted values are in *log* terms, so they need to be *exponentiated* to be correctly interpreted.

```
lm.logit <- glm(as.numeric(Y) ~ ., data = as.data.frame(X), family =
"binomial")
ynew <- predict(lm.logit, as.data.frame(X)); #plot(ynew)
ynew2 <- ifelse(exp(ynew)<0.5, 0, 1); # plot(ynew2)

predfun.logit = function(train.x, train.y, test.x, test.y, neg)
{   lm.logit <- glm(train.y ~ ., data = train.x, family = "binomial")
    ynew = predict(lm.logit, test.x )
 # compute TP, FP, TN, FN
ynew2 <- ifelse(exp(ynew)<0.5, 0, 1)
    out = confusionMatrix(test.y, ynew2, negative=neg)  # Binary outcome,
we can use confusionMatrix
return( out )
}
# Reduce the bag of explanatory variables, purely to simplify the
interpretation of the analytics in this example!
input.short <- input[, which(names(input) %in% c("R_fusiform_gyrus_Volume",
            "R_fusiform_gyrus_ShapeIndex", "R_fusiform_gyrus_Curvedness",
 "Sex",  "Weight", "Age", "chr12_rs34637584_GT", "chr17_rs11868035_GT",
    "UPDRS_Part_I_Summary_Score_Baseline",
"UPDRS_Part_I_Summary_Score_Month_03",
"UPDRS_Part_II_Patient_Questionnaire_Summary_Score_Baseline",
   "UPDRS_Part_III_Summary_Score_Baseline",
   "X_Assessment_Non.Motor_Epworth_Sleepiness_Scale_Summary_Score_Baseline"
))]
X = as.matrix(input.short)

cv.out.logit = crossval::crossval(predfun.logit, as.data.frame(X),
as.numeric(Y), K=5, B=2, neg="1", verbose=FALSE)
cv.out.logit$stat.cv
```

```
##       FP TP TN FN
## B1.F1  1 50 31  2
## B1.F2  0 60 19  6
## B1.F3  2 55 19  8
## B1.F4  3 58 23  0
## B1.F5  3 60 21  1
## B2.F1  2 56 22  4
## B2.F2  0 57 23  5
## B2.F3  3 60 20  1
## B2.F4  1 58 23  2
## B2.F5  1 54 27  3
```

```
diagnosticErrors(cv.out.logit$stat)
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9431280 0.9466667 0.9344262 0.9726027 0.8769231 5.5331424
```

Caution: Note that if you forget to exponentiate the values of the predicted logistic model (see *ynew2* in `predict.logit`), you will get nonsense results, e.g., all cases may be predicted to be in one class, trivial sensitivity, or incorrect NPP.

21.7.2 Quadratic Discriminant Analysis (QDA)

In Chaps. 8 and 21, we discussed the *linear* and *quadratic* discriminant analysis models. Let's now introduce a `predfun.qda()` function.

```
predfun.qda = function(train.x, train.y, test.x, test.y, negative)
{
  require("MASS") # for lda function
  qda.fit = qda(train.x, grouping=train.y)
  ynew = predict(qda.fit, test.x)$class
  out.qda = confusionMatrix(test.y, ynew, negative=negative)
  return( out.qda )
}

cv.out.qda = crossval::crossval(predfun.qda, as.data.frame(input.short),
as.factor(Y), K=5, B=20, neg="1")

## Error in qda.default(x, grouping, ...): rank deficiency in group 1
diagnosticErrors(cv.out.lda$stat); diagnosticErrors(cv.out.qda$stat);

## Error in diagnosticErrors(cv.out.qda$stat): object 'cv.out.qda' not found
```

This error message: "*Error in qda.default(x, grouping, ...): rank deficiency in group 1*" indicates that there is a rank deficiency, i.e. some variables are collinear and one or more covariance matrices cannot be inverted to obtain the estimates in group 1 (Controls).

If you remove the strongly correlated data elements ("R_fusiform_gyrus_Volume", "R_fusiform_gyrus_ShapeIndex", and "R_fusiform_gyrus_Curvedness"), the rank-deficiency problem goes away.

```
input.short2 <- input[, which(names(input) %in% c("R_fusiform_gyrus_Volume",
"Sex",  "Weight", "Age" , "chr17_rs11868035_GT",
"UPDRS_Part_I_Summary_Score_Baseline",
"UPDRS_Part_II_Patient_Questionnaire_Summary_Score_Baseline",
    "UPDRS_Part_III_Summary_Score_Baseline",
    "X_Assessment_Non.Motor_Epworth_Sleepiness_Scale_Summary_Score_Baseline"
))]
X = as.matrix(input.short2)
cv.out.qda= crossval::crossval(predfun.qda, as.data.frame(X), as.numeric(Y),
K=5, B=2, neg="1")
```

It makes sense to contrast the QDA and GLM/Logit predictions.

```
diagnosticErrors(cv.out.qda$stat); diagnosticErrors(cv.out.logit$stat)

##       acc      sens      spec       ppv       npv       lor
## 0.9407583 0.9533333 0.9098361 0.9629630 0.8880000 5.3285694

##       acc      sens      spec       ppv       npv       lor
## 0.9431280 0.9466667 0.9344262 0.9726027 0.8769231 5.5331424
```

Clearly, both the QDA and Logit model predictions are quite similar and reliable.

21.7.3 *Foundation of LDA and QDA for Prediction, Dimensionality Reduction, and Forecasting*

Previously, in Chap. 8 we saw some examples of LDA/QDA methods. Now, we'll provide more details. Both LDA (Linear Discriminant Analysis) and QDA (Quadratic Discriminant Analysis) use probabilistic models of the class conditional distribution of the data $P(X \mid Y = k)$ for each class k. Their predictions are obtained by using the Bayesian theorem (http://wiki.socr.umich.edu/index.php/SMHS_BayesianInference#Bayesian_Rule):

$$P(Y = k \mid X) = \frac{P(X \mid Y = k)P(Y = k)}{P(X)} = \frac{P(X \mid Y = k)P(Y = k)}{\sum_{l=0}^{\infty} P(X \mid Y = l)P(Y = l)}$$

Thus, we select the class k, which **maximizes** this conditional probability (maximum likelihood estimation). In linear and quadratic discriminant analysis, $P(X \mid Y)$ is modelled as a multivariate Gaussian distribution with density:

$$P(X \mid Y = k) = \frac{1}{(2\pi)^n |\Sigma_k|^{\frac{1}{2}}} \times e^{\left(-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1}(X-\mu_k)\right)}.$$

This model can be used to classify data by using the training data to **estimate**:

(1) The class prior probabilities $P(Y = k)$ by counting the proportion of observed instances of class k,

(2) The class means μ_k by computing the empirical sample class means, and
(3) The covariance matrices by computing either the empirical sample class covariance matrices, or by using a regularized estimator, e.g., LASSO).

In the **linear case** (LDA), the Gaussians for each class are assumed to share the same covariance matrix: $\Sigma_k = \Sigma$ for each class k. This leads to linear decision surfaces separating different classes. This is clear from comparing the log-probability ratios of a pair of 2 classes (k and l):

$LOR = log\left(\frac{P(Y=k|X)}{P(Y=l|X)}\right)$, (the LOR $= 0 \Leftrightarrow$ the two probabilities are identical, i.e., same class)

$$LOR = log\left(\frac{P(Y=k \mid X)}{P(Y=l \mid X)}\right) = (\mu_k - \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) = \frac{1}{2}\left(\mu_k^T \Sigma^{-1} \mu_k - \mu_l^T \Sigma^{-1} \mu_l\right).$$

But, in the more general, **quadratic case** of QDA, there are no assumptions on the covariance matrices Σ_k of the Gaussians, leading to more flecible quadratic decision surfaces separating the classes.

LDA (Linear Discriminant Analysis)

LDA is similar to GLM (e.g., ANOVA and regression analyses), as it also attempts to express one dependent variable as a linear combination of the other features or data elements, However, ANOVA uses categorical independent variables and a continuous dependent variable, whereas LDA has continuous independent variables and a categorical dependent variable (i.e., Dx/class label). Logistic regression and probit regression are more similar to LDA than ANOVA, as they also explain a categorical variable by the values of continuous independent variables.

```
predfun.lda = function(train.x, train.y, test.x, test.y, neg)
{
  require("MASS")
  lda.fit = lda(train.x, grouping=train.y)
  ynew = predict(lda.fit, test.x)$class
  out.lda = confusionMatrix(test.y, ynew, negative=neg)
  return( out.lda )
}
```

QDA (Quadratic Discriminant Analysis)

Similarly to LDA, the QDA prediction function can be defined by:

```
predfun.qda = function(train.x, train.y, test.x, test.y, neg)
{
  require("MASS") # for lda function
  qda.fit = qda(train.x, grouping=train.y)
  ynew = predict(qda.fit, test.x)$class
  out.qda = confusionMatrix(test.y, ynew, negative=neg)
  return( out.qda )
}
```

21.7.4 Neural Networks

We already saw Artificial Neural Networks (NNs) in Chap. 11. Applying NNs is not straightforward. We have to create a design matrix with an indicator column for the response feature. In addition, we need to write a *predict function* to translate the output of `neuralnet()` into analytical forecasts.

```
# predict nn
library("neuralnet")
pred = function(nn, dat) {
    yhat = compute(nn, dat)$net.result
    yhat = apply(yhat, 1, which.max)-1
    return(yhat)
}

my.neural <- function (train.x, train.y, test.x,
test.y,method,layer=c(5,5)){
  train.x <- as.data.frame(train.x)
  train.y <- as.data.frame(train.y)
  colnames(train.x) <- paste0('V', 1:ncol(X))
  colnames(train.y) <- "V1"
  train_y_ind = model.matrix(~factor(train.y$V1)-1)
  colnames(train_y_ind) = paste0('out', 0:1)
  train = cbind(train.x, train_y_ind)
  y_names = paste0('out', 0:1)
  x_names = paste0('V', 1:ncol(train.x))
  nn = neuralnet(
    paste(paste(y_names, collapse='+'),
          '~',
          paste(x_names, collapse='+')),
    train,
    hidden=layer,
      linear.output=FALSE,
    lifesign='full', lifesign.step=1000)
  #predict
  predict.y <- pred(nn, test.x)
  out <- crossval::confusionMatrix(test.y, predict.y,negative = 0)
  return (out)
}

set.seed(1234)
cv.out.nn <- crossval::crossval(my.neural, scale(X), Y, K = 5, B =
1,layer=c(20,20),verbose = F) # scale predictors is necessary.

## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps:  63 error: 1.02185
time: 0.08 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps:79 error: 1.01374
time: 0.2 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps:  73 error: 1.02399
time: 0.09 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps:  66 error: 1.03016
time: 0.09 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps:  72 error: 1.01491
time: 0.11 secs
```

```
crossval::diagnosticErrors(cv.out.nn$stat)
```

```
##          acc         sens         spec          ppv          npv
## 0.9454976303 0.9016393443 0.9633333333 0.9090909091 0.9601328904
##          lor
## 5.4841051313
```

Again the forecasting results on the PD dataset are quite good.

21.7.5 SVM

In Chap. 11, we also saw SVM classification. Let's try cross-validation using Linear and Gaussian (radial) kernel SVM. We may expect that linear SVM would achieve a similar result to Gaussian, or even better than Gaussian SVM, since this dataset has a large k (# features) compared with n (# cases), which we explored in detail in Chap. 11.

```
library("e1071")

my.svm <- function (train.x, train.y, test.x,
test.y,method,cost=1,gamma=1/ncol(dx_norm),coef0=0,degree=3){
  svm_l.fit <- svm(x = train.x, y=as.factor(train.y),kernel = method)
  predict.y <- predict(svm_l.fit, test.x)
  out <- crossval::confusionMatrix(test.y, predict.y,negative = 0)
  return (out)
}

# Linear kernel
set.seed(123)
cv.out.svml <- crossval::crossval(my.svm, as.data.frame(X), Y, K = 5, B = 1,
method = "linear",cost=tune_svm$best.parameters$cost,verbose = F)
diagnosticErrors(cv.out.svml$stat)
```

```
##          acc         sens         spec          ppv          npv
## 0.9502369668 0.9098360656 0.9666666667 0.9173553719 0.9634551495
##          lor
## 5.6789307585
```

```
# Gaussian kernel
set.seed(123)
cv.out.svmg <- crossval::crossval(my.svm, as.data.frame(X), Y, K = 5, B = 1,
method = "radial",cost=tune_svmg$best.parameters$cost,gamma=tune_svmg$best.p
arameters$gamma,verbose = F)
diagnosticErrors(cv.out.svmg$stat)
```

```
##          acc         sens         spec          ppv          npv
## 0.9454976303 0.9262295082 0.9533333333 0.8897637795 0.9694915254
##          lor
## 5.5470977226
```

Indeed, both types of kernels yield good quality predictors according to the assessment metrics reported by the `diagnosticErrors()` method.

21.7.6 k-Nearest Neighbors Algorithm (k-NN)

As we saw in Chap. 7, *k-NN* is a non-parametric method for either classification or regression, where the **input** consists of the k closest **training examples** in the feature space, but the **output** depends on whether k-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership (labels). Objects in the testing data are classified by a majority vote of their neighbors. Each object is assigned to a class that is most common among its k nearest neighbors (k is always a small positive integer). When $k = 1$, then an object is assigned to the class of its single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object representing the average of the values of its k nearest neighbors.

Let's now build the corresponding `predfun.knn()` method.

```
# X = as.matrix(input)  # Predictor variables X = as.matrix(input.short2)
#  Y = as.matrix(output)    # Outcome
# KNN (k-nearest neighbors)
library("class")
# knn.fit.test <- knn(X, X, cl = Y, k=3, prob=F); predict(as.matrix(knn.fit.
test), X)$class
# table(knn.fit.test, Y); confusionMatrix(Y, knn.fit.test, negative="1")
# This can be used for polytomous variable (multiple classes)

predfun.knn = function(train.x, train.y, test.x, test.y, neg)
{
  require("class")
  knn.fit = knn(train.x, test.x, cl = train.y, prob=T)  # knn is already
a prediction function!!!
  # ynew = predict(knn.fit, test.x)$class           # no need of another
prediction, in this case
  out.knn = confusionMatrix(test.y, knn.fit, negative=neg)
  return( out.knn )
}
cv.out.knn = crossval::crossval(predfun.knn, X, Y, K=5, B=2, neg="1")

cv.out.knn = crossval::crossval(predfun.knn, X, Y, K=5, B=2, neg="1")
#Compare all 3 classifiers (lda, qda, knn, and logit)
diagnosticErrors(cv.out.lda$stat); diagnosticErrors(cv.out.qda$stat);
diagnosticErrors(cv.out.qda$stat); diagnosticErrors(cv.out.logit$stat);
```

We can also examine the performance of k-NN prediction on the PPMI (Parkinson's disease) data. Start by partitioning the data into `training` and `testing` sets.

```
# TRAINING: 75% of the sample size
sample_size <- floor(0.75 * nrow(input))
## set the seed to make your partition reproducible
set.seed(1234)
input.train.ind <- sample(seq_len(nrow(input)), size = sample_size)
input.train <- input[input.train.ind, ]
output.train <- as.matrix(output)[input.train.ind, ]

# TESTING DATA
input.test <- input[-input.train.ind, ]
output.test <- as.matrix(output)[-input.train.ind, ]
```

Then, we can fit the k-NN model and report the results.

```
library("class")
knn_model <- knn(train= input.train, input.test, cl=as.factor(output.train),
k=2)
#plot(knn_model)
summary(knn_model)
attributes(knn_model)

# cross-validation
knn_model.cv <- knn.cv(train= input.train, cl=as.factor(output.train), k=2)
summary(knn_model.cv)
```

21.7.7 k-Means Clustering (k-MC)

In Chap. 13, we showed that k-MC aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean, which acts as a prototype of a cluster. The k-MC partitions the data space into Voronoi cells. In general, there is no computationally tractable solution for this, i.e., the problem is NP-hard. However, there are efficient algorithms that converge quickly to local optima, e.g., the *expectation-maximization* algorithm for mixtures of Gaussian distributions via an iterative refinement approach (Figs. 21.5, 21.6 and 21.7).

```
kmeans_model <- kmeans(input.train, 2)
layout(matrix(1, 1))
# tiff("C:/Users/User/Desktop/test.tiff", width = 10, height = 10, units = '
in', res = 300)
fpc::plotcluster(input.train, output.train, col = kmeans_model$cluster)
```

```
cluster::clusplot(input.train, kmeans_model$cluster, color=TRUE, shade=TRUE,
labels=2, lines=0)
```

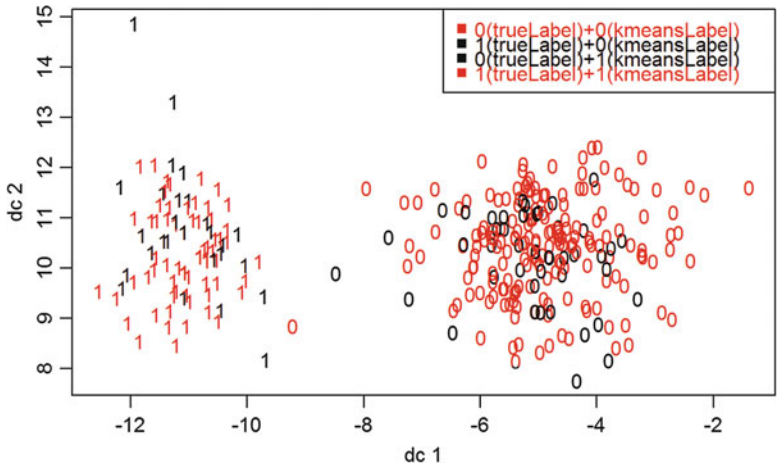

Fig. 21.5 k-Means clustering plot () of the Parkinson's disease data (PPMI)

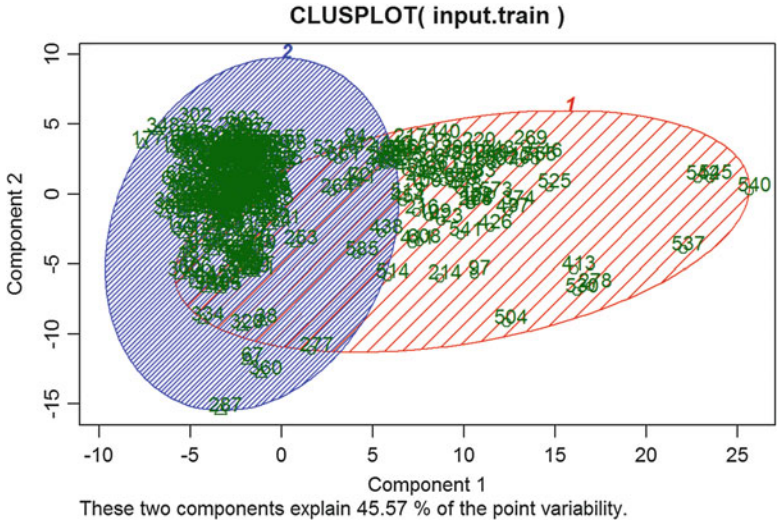

Fig. 21.6 Clusterplot of the k-means clustering of the PPMI data

```
par(mfrow=c(10,10))
# the next figure is very large and will not render in RStudio, you may need
to save it as PDF file!
# pdf("C:/Users/User/Desktop/test.pdf", width = 50, height = 50)
# with(ppmi_data[,1:10], pairs(input.train[,1:10], col=c(1:2)[kmeans_model$c
luster]))
# dev.off()
with(ppmi_data[,1:10], pairs(input.train[,1:10],
col=c(1:2)[kmeans_model$cluster]))
```

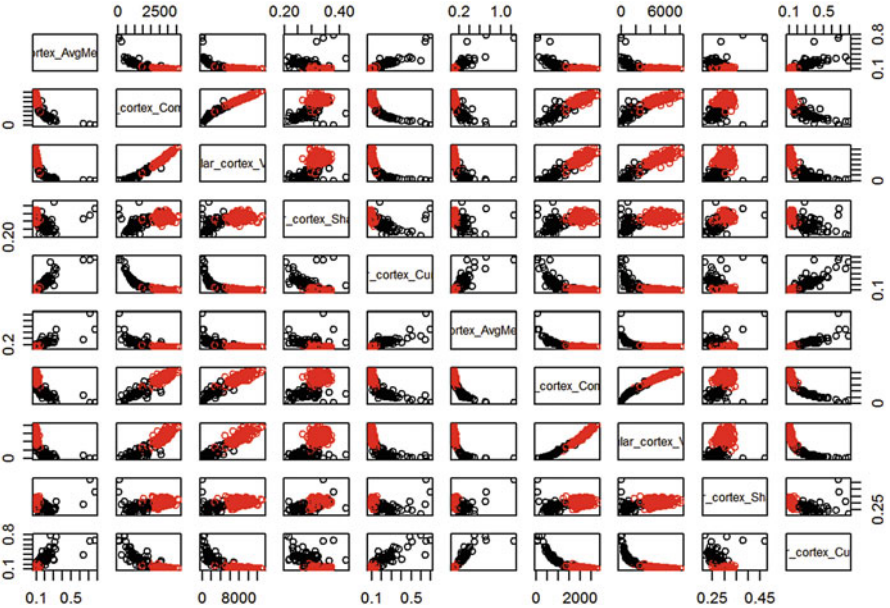

Fig. 21.7 Pair plots of the two clustering lables along the first 10 PPMI features

```
# plot(input.train, col = kmeans_model$cluster)
# points(kmeans_model$centers, col = 1:2, pch = 8, cex = 2)

## cluster centers "fitted" to each obs.:
fitted.kmeans <- fitted(kmeans_model);  head(fitted.kmeans)

##   L_insular_cortex_AvgMeanCurvature L_insular_cortex_ComputeArea
## 2                       0.1071299082                  2635.580514
## 2                       0.1071299082                  2635.580514
## 1                       0.2221893533                  1134.578902
## 2                       0.1071299082                  2635.580514
## 2                       0.1071299082                  2635.580514
## 2                       0.1071299082                  2635.580514
##   L_insular_cortex_Volume L_insular_cortex_ShapeIndex
## 2             7969.485443                0.3250065829
## 2             7969.485443                0.3250065829
## 1             2111.385018                0.2788562513
## 2             7969.485443                0.3250065829
## 2             7969.485443                0.3250065829
## 2             7969.485443                0.3250065829
…

resid.kmeans <- (input.train - fitted(kmeans_model))

# define the sum of squares function
ss <- function(data) sum(scale(data, scale = FALSE)^2)
```

```
## Equalities
cbind(kmeans_model[c("betweenss", "tot.withinss", "totss")], # the same two
columns
                c (ss(fitted.kmeans), ss(resid.kmeans),  ss(input.train)))

##              [,1]        [,2]
## betweenss    15462062254 15462062254
## tot.withinss 12249286905 12249286905
## totss        27711349159 27711349159

# validation
stopifnot(all.equal(kmeans_model$totss,        ss(input.train)),
        all.equal(kmeans_model$tot.withinss, ss(resid.kmeans)),
         ## these three are the same:
        all.equal(kmeans_model$betweenss,    ss(fitted.kmeans)),
        all.equal(kmeans_model$betweenss, kmeans_model$totss -
kmeans_model$tot.withinss),
        ## and hence also
        all.equal(ss(input.train), ss(fitted.kmeans) + ss(resid.kmeans))
)
# kmeans(input.train, 1)$withinss
# trivial one-cluster, (its W.SS == ss(input.train))
clust_kmeans2 = kmeans(scale(X), center=X[1:2,],iter.max=100,
algorithm='Lloyd')
```

We may get empty clusters, instead of two clusters, when we randomly select two points as the initial centers. The way to solve this problem is using k-means++.

```
# k++ initialize
kpp_init = function(dat, K) {
  x = as.matrix(dat)
  n = nrow(x)
  # Randomly choose a first center
  centers = matrix(NA, nrow=K, ncol=ncol(x))
  centers[1,] = as.matrix(x[sample(1:n, 1),])
  for (k in 2:K) {
    # Calculate dist^2 to closest center for each point
    dists = matrix(NA, nrow=n, ncol=k-1)
    for (j in 1:(k-1)) {
      temp = sweep(x, 2, centers[j,], '-')
      dists[,j] = rowSums(temp^2)
    }
    dists = rowMeans(dists)
    # Draw next center with probability proportional to dist^2
    cumdists = cumsum(dists)
    prop = runif(1, min=0, max=cumdists[n])
    centers[k,] = as.matrix(x[min(which(cumdists > prop)),])
  }
  return(centers)
}
clust_kmeans2_plus = kmeans(scale(X), kpp_init(scale(X), 2), iter.max=100, a
lgorithm='Lloyd')
```

Now let's evaluate the model. The first step is to justify the selection of `k=2`. We use the method `silhouette()` in package `cluster`. Recall from Chap. 14 that the *silhouette value* is between -1 and 1. Negative silhouette values represent "misclustered" cases (Fig. 21.8).

Fig. 21.8 Silhouette plot of the 2-class k-means clustering of the Parkonson's disease data

```
clust_k2 = clust_kmeans2_plus$cluster
require(cluster)

## Loading required package: cluster

# X = as.matrix(input.short2)
# as the data is too large for the silhouette plot, we'll just subsample and
plot 100 random cases
subset_int <- sample(nrow(X),100)  #100 cases from 661 total cases
dis = dist(as.data.frame(scale(X[subset_int,])))
sil_k2 = silhouette(clust_k2[subset_int],dis) #best
plot(sil_k2)
```

```
summary(sil_k2)

## Silhouette of 100 units in 2 clusters from silhouette.default(x = clust_k
2[subset_int], dist = dis) :
##  Cluster sizes and average silhouette widths:
##           48           52
## 0.1895633766 0.1018642857
## Individual silhouette widths:
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
## -0.06886907  0.06533312  0.14169240  0.14395980  0.22658680  0.33585520

mean(sil_k2<0)
## [1] 0.01666666667
```

The result is pretty good. Only a very small number of samples are "misclustered" (having negative silhouette values). Furthermore, you can observe that when `k=3` or `k=4`, the overall silhouette decreases, which indicates suboptimal clustering.

```
dis = dist(as.data.frame(scale(X)))
clust_kmeans3_plus = kmeans(scale(X), kpp_init(scale(X), 3), iter.max=100, a
lgorithm='Lloyd')
summary(silhouette(clust_kmeans3_plus$cluster,dis))

## Silhouette of 422 units in 3 clusters from silhouette.default(x =
clust_kmeans3_plus$cluster, dist = dis) :
##  Cluster sizes and average silhouette widths:
##           139           157           126
## 0.08356111542 0.19458813829 0.17237138090
## Individual silhouette widths:
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
## -0.06355399  0.08376430  0.16639550  0.15138420  0.21855670  0.33107050

clust_kmeans4_plus = kmeans(scale(X), kpp_init(scale(X), 4), iter.max=100, a
lgorithm='Lloyd')
summary(silhouette(clust_kmeans4_plus$cluster,dis))

## Silhouette of 422 units in 4 clusters from silhouette.default(x =
clust_kmeans4_plus$cluster, dist = dis) :
##  Cluster sizes and average silhouette widths:
##            138           121           111            52
## 0.124165755516 0.170228092125 0.193359499726 0.008929262925
## Individual silhouette widths:
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
## -0.16300240  0.08751445  0.15091580  0.14137370  0.21035560  0.32293680
```

Then, let's calculate the unsupervised classification error. Here, p represents the percentage of class 0 cases, which provides the weighting factor for labelling each cluster.

```
mat = matrix(1,nrow = length(Y))
p = sum(Y==0)/length(Y)
for (i in 1:2){
  id = which(clust_k2==i)
  if(sum(Y[id]==0)>length(id)*p){
    mat[id] = 0
  }
}
caret::confusionMatrix(Y,mat)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 195 105
##          1   1 121
##
##                Accuracy : 0.7488152
##                  95% CI : (0.7045933, 0.7895087)
##     No Information Rate : 0.535545
```

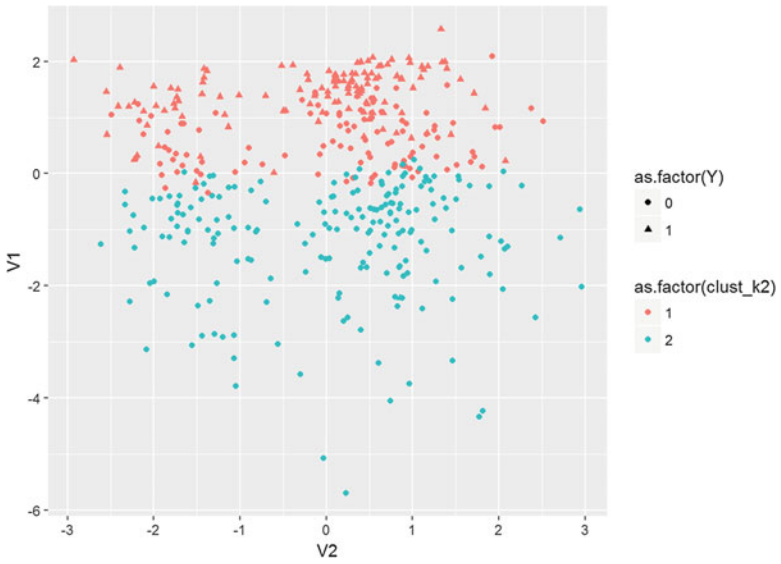

Fig. 21.9 Multi-dimensional scalling plot (2D projection) of the k-means clustering depicting the agreement between testing data labels (glyph shapes) and the predicted class lables (glyph colors)

```
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.5122558
##  Mcnemar's Test P-Value : < 0.00000000000000022204
##
##             Sensitivity : 0.9948980
##             Specificity : 0.5353982
##          Pos Pred Value : 0.6500000
##          Neg Pred Value : 0.9918033
##              Prevalence : 0.4644550
##          Detection Rate : 0.4620853
##    Detection Prevalence : 0.7109005
##       Balanced Accuracy : 0.7651481
##
##        'Positive' Class : 0
```

It achieves 69% accuracy, which is reasonable for unsupervised classification.

Finally, let's visualize the results by superimposing the data into the first two multi-dimensional scaling (MDS) dimensions (Fig. 21.9).

```
library("ggplot2")
mds = as.data.frame(cmdscale(dis, k=2))
mds_temp = cbind(
  mds, as.factor(clust_k2))
names(mds_temp) = c('V1', 'V2', 'cluster k=2')
gp_cluster = ggplot(mds_temp, aes(x=V2, y=V1, color=as.factor(clust_k2))) +
  geom_point(aes(shape = as.factor(Y))) + theme()
gp_cluster
```

21.7.8 Spectral Clustering

Suppose the multivariate dataset is represented as a set of data points A. We can define a similarity matrix $S = s_{(i,j)}$, where $s_{(i,j)}$ represents a measure of the similarity between points $i, j \in A$. Spectral clustering uses the spectrum of the similarity matrix of the high-dimensional data and performs dimensionality reduction for clustering into fewer dimensions. The spectrum of a matrix is the set of its eigenvalues. In general, if $M : \Omega \xrightarrow{\text{linear operator}} \Omega$ maps a vector space Ω into itself, its spectrum is the set of scalars $\lambda = \{\lambda_i\}$ such that $(T - \lambda I)v = 0$, where I is the identity matrix and v are the eigen vectors (or eigen-functions) for the operator T. The **determinant** of the matrix equals the product of its eigenvalues, i.e., $det(T) = \Pi_i \lambda_i$, the **trace** of the matrix $tr(T) = \Sigma_i \lambda_i$, and the **pseudo-determinant** for a singular matrix is the product of its nonzero eigenvalues, $pseudo_{det}(T) = \Pi_{\lambda_i \neq 0} \lambda_i$.

To partition the data into two sets (S_1, S_2), denote v to be the second-smallest eigenvector of the Laplacian matrix:

$$L = I - D^{-\frac{1}{2}} S D^{\frac{1}{2}}$$

of the similarity matrix S, where D is the diagonal matrix $D_{i,i} = \Sigma_j S_{i,j}$.

This actual (S_1, S_2) partitioning of the cases in the data may be done in different ways. For instance, S_1 may use the median m of the components in v and group all data points whose component in v is greater than m. Then, the remaining cases can be labeled as part of S_2. This approach may be used iteratively for *hierarchical clustering* by repeatedly partitioning the subsets.

The *specc* method in the *kernlab* package implements a spectral clustering algorithm where the data-clustering is performed by embedding the data into the subspace of the eigenvectors of an affinity matrix.

```
# install.packages("kernlab")
library("kernlab")

# review and choose a dataset (for example the Iris data
data()
#plot(iris)
```

Let's look at a few simple cases of spectral clustering. We are suppressing some of the outputs to save space (e.g., `#plot(my_data, col= data_sc))`.

Iris Petal Data

Let's look at the *iris* dataset we saw in Chap. 3.

```
my_data <- iris; data(my_data)
num_clusters <- 3
data_sc <- specc(my_data, centers= num_clusters)
data_sc
centers(data_sc)
withinss(data_sc)
#plot(my_data, col= data_sc)
```

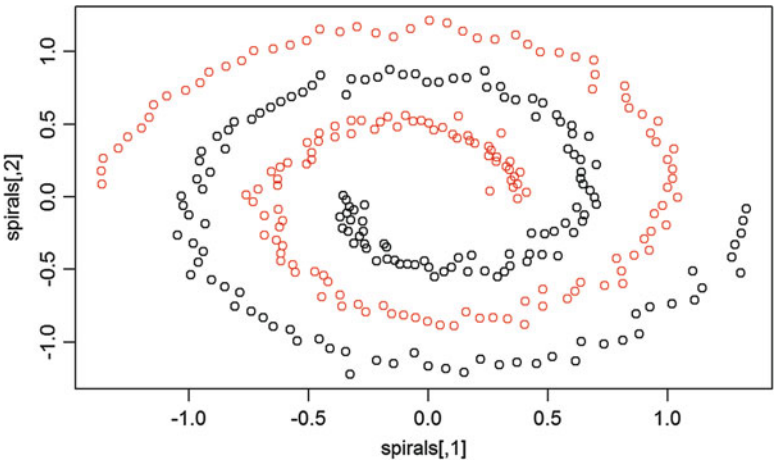

Fig. 21.10 Spirals data spectal clustering results

Spirals Data

Another simple dataset is `kernlab::spirals` (Fig. 21.10).

```
# library("kernlab")
data(spirals)
num_clusters <- 2
data_sc <- specc(spirals, centers= num_clusters)
data_sc

## Spectral Clustering object of class "specc"
##
##  Cluster memberships:
##
## 1 1 2 2 1 2 2 2 1 2 2 1 1 2 2 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 2 1 2 2 1 2 1
2 1 1 2 2 2 2 1 1 1 1 2 1 2 1 1 2 2 2 1 1 1 1 1 2 2 1 2 1 1 1 2 2 1 2 2 2
1 1 1 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 2 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 2 1 2 1
1 1 1 2 2 1 1 2 1 1 1 2 1 2 1 1 1 1 1 2 2 2 2 2 2 1 1 1 2 2 1 2 1 1 1 2 2 2 1
2 2 2 2 2 2 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 2 2 2 1 1 2
2 2 2 2 2 2 2 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 1 2 1 2 1 2 1 1 1 2 2 2 2 1 1
1 2 1 1 2 2 2 2 2 1 1 2 2 2 2 1 1 1 2 2 2 2 2 2 2 1 1 2 2 1 1 1 1 1 1 1 2 2 2
2 1 2 1 2 1 1 2 2 2 1 2 1 1 1 1 2 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 2 2 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  367.501471756465
##
## Centers:
##                [,1]          [,2]
## [1,]  0.01997200551 -0.1761483316
## [2,] -0.01770984369  0.1775136857
##
## Cluster size:
## [1] 150 150
```

```
##
## Within-cluster sum of squares:
## [1] 117.3429096 118.1182272

centers(data_sc)

##                [,1]          [,2]
## [1,]  0.01997200551 -0.1761483316
## [2,] -0.01770984369  0.1775136857

withinss(data_sc)

## [1] 117.3429096 118.1182272

plot(spirals, col= data_sc)
```

Income Data

A customer *income* dataset representing a marketing survey is included in `kernlab::income` (Fig. 21.11).

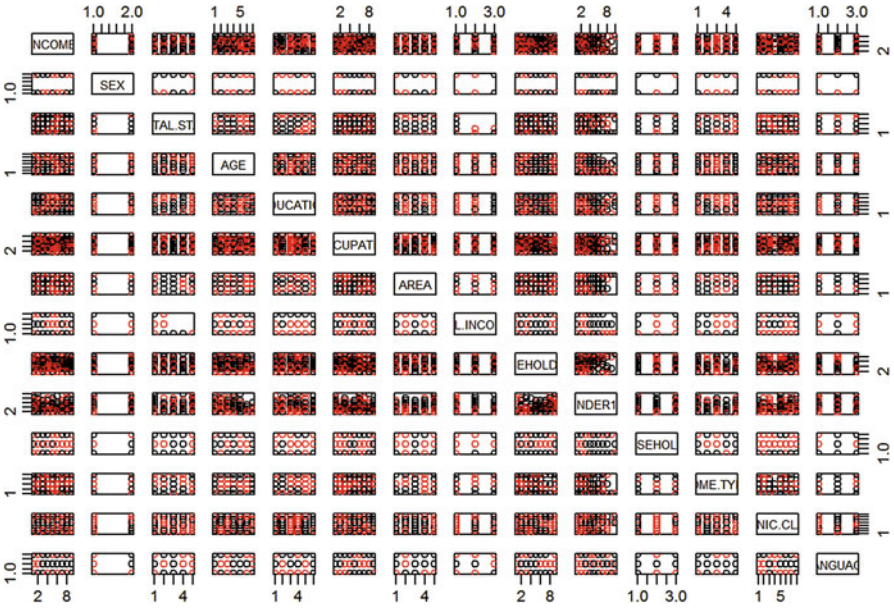

Fig. 21.11 Pair plots of the two-class spectral clustering of the income dataset

```
data(income)
num_clusters <- 2
data_sc <- specc(income, centers= num_clusters)
data_sc

## Spectral Clustering object of class "specc"
##
##  Cluster memberships:
##
## 2 1 2 2 2 1 2 1 2 1 1 1 2 1
##
## String kernel function.  Type =  spectrum
##  Hyperparameters : sub-sequence/string length =  4
##  Normalized
##
## Cluster size:
## [1] 7 7

centers(data_sc)

##      [,1]
## [1,]   NA

withinss(data_sc)

## logical(0)

plot(income, col= data_sc)
```

21.8 Compare the Results

Now let's compare all eight classifiers (`AdaBoost`, `LDA`, `QDA`, `knn`, `logit`, `Neural Network`, `linear SVM` and `Gaussian SVM`) we presented above (Table 21.1).

```
# get AdaBoost CV results
set.seed(123)
cv.out.ada <- crossval::crossval(my.ada, as.data.frame(X), Y, K = 5, B = 1,
negative = neg)

## Number of folds: 5
## Total number of CV fits: 5
##
## Round # 1 of 1
## CV Fit # 1 of 5
## CV Fit # 2 of 5
## CV Fit # 3 of 5
## CV Fit # 4 of 5
## CV Fit # 5 of 5

# get k-Means CV results
my.kmeans <- function (train.x, train.y, test.x, test.y, negative, formula){
  kmeans.fit <- kmeans(scale(test.x), kpp_init(scale(test.x), 2),
iter.max=100, algorithm='Lloyd')
```

Table 21.1 Comparison of alternative predictive analytic strategies for the PPMI dataset

	acc	sens	spec	ppv	npv	lor
AdaBoost	**0.9739336493**	**0.9933333333**	**0.9262295082**	**0.9706840391**	**0.9826086957**	**7.5341095473**
LDA	0.9617298578	0.9513333333	0.9872950820	0.9945983621	0.8918918919	7.3258499745
QDA	0.9407582938	0.9533333333	0.9098360656	0.9629629630	0.8880000000	5.3285694097
knn	0.6113744076	0.7133333333	0.3606557377	0.7328767123	0.3384615385	0.3391095260
logit	0.9431279621	0.9466666667	0.9344262295	0.9726027397	0.8769230769	5.5331424226
Neural Network	0.9454976303	0.9016393443	0.9633333333	0.9090909091	0.9601328904	5.4841051313
linear SVM	0.9502369668	0.9098360656	0.9666666667	0.9173553719	0.9634551495	5.6789307585
Gaussian SVM	0.9454976303	0.9262295082	0.9533333333	0.8897637795	0.9694915254	5.5470977226
k-Means	0.5331753555	0.5400000000	0.5163934426	0.7330316742	0.3134328358	0.2259399326
Spectral Clustering	0.5592417062	0.5900000000	0.4836065574	0.7375000000	0.3241758242	0.2983680947

```
  predict.y <- kmeans.fit$cluster
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
 # negative  is the label of a negative "null" sample (default: "control").
  return (out)
}
set.seed(123)
cv.out.kmeans <- crossval::crossval(my.kmeans, as.data.frame(X), Y, K = 5,
B = 2, negative = neg)

## Number of folds: 5
## Total number of CV fits: 10
##
## Round # 1 of 2
## CV Fit # 1 of 10
## CV Fit # 2 of 10
## CV Fit # 3 of 10
## CV Fit # 4 of 10
## CV Fit # 5 of 10
##
## Round # 2 of 2
## CV Fit # 6 of 10
## CV Fit # 7 of 10
## CV Fit # 8 of 10
## CV Fit # 9 of 10
## CV Fit # 10 of 10

# get spectral clustering CV results
my.sc <- function (train.x, train.y, test.x, test.y, negative, formula){
  sc.fit <- specc(scale(test.x), centers= 2)
  predict.y <- sc.fit@.Data
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
 # negative  is the label of a negative "null" sample (default: "control").
  return (out)
}
set.seed(123)
cv.out.sc <- crossval::crossval(my.sc, as.data.frame(X), Y, K = 5, B = 2,
negative = neg)

## Number of folds: 5
## Total number of CV fits: 10
##
## Round # 1 of 2
## CV Fit # 1 of 10
## CV Fit # 2 of 10
## CV Fit # 3 of 10
## CV Fit # 4 of 10
## CV Fit # 5 of 10
##
## Round # 2 of 2
## CV Fit # 6 of 10
## CV Fit # 7 of 10
## CV Fit # 8 of 10
## CV Fit # 9 of 10
## CV Fit # 10 of 10
```

```
require(knitr)

## Loading required package: knitr

res_tab=rbind(diagnosticErrors(cv.out.ada$stat),diagnosticErrors(
cv.out.lda$stat),diagnosticErrors(cv.out.qda$stat),diagnosticErrors(
cv.out.knn$stat),diagnosticErrors(cv.out.logit$stat),diagnosticErrors(
cv.out.nn$stat),diagnosticErrors(cv.out.svml$stat),diagnosticErrors(
cv.out.svmg$stat),diagnosticErrors(cv.out.kmeans$stat),diagnosticErrors(
cv.out.sc$stat))
rownames(res_tab) <- c("AdaBoost", "LDA", "QDA", "knn", "logit",
"Neural Network", "linear SVM", "Gaussian SVM", "k-Means",
"Spectral Clustering")
kable(res_tab,caption = "Compare Result")
```

Leaving `knn`, `kmeans` and `specc` aside, the other methods achieve pretty good results. In the PD case study, the reason for suboptimal results in some clustering methods may be rooted in lack of training (e.g., `specc` and `kmeans`) or the curse of (high) dimensionality, which we saw in Chap. 7. As the data are rather sparse, predicting from the nearest neighbors may not be too reliable.

21.9 Assignment: 21. Prediction and Internal Statistical Cross-Validation

Demonstrate cross-validation on these two case-studies independently:

- Example 1: ALS (Amyotrophic Lateral Sclerosis)
- Example 2: Quality of Life in Chronic Illness (Case06_QoL_Symptom_ChronicIllness.csv)

Go through the following protocol:

- Review the case-study.
- Choose appropriate dichotomous, polytomous or continuous outcome variables, e.g., use `ALSFRS_slope` for ALS, `CHRONICDISEASESCORE` for *case 06* and cast them as dichotomous outcomes.
- Apply appropriate data preprocessing.
- Perform regression modeling (e.g., OLS, glmnet, Forward or Backward model selection, etc.) for continuous outcomes.
- Perform classification and prediction using various methods (e.g., LDA, QDA, AdaBoost, SVM, Neural Network, KNN) for discrete outcomes.
- Apply cross-validation on these regression and classification methods, respectively.
- Report standard error for regression approaches.
- Report appropriate quality metrics that can be used to rank the forecasting approaches based on the predictive power of their results.
- Compare the result of model-driven and data-driven (e.g., KNN).

- Compare the sensitivity and specificity.
- Use unsupervised classification methods (*k-Means*) and spectral clustering.
- Evaluate and justify a *k-Means* model and report the agreement of the derived clusters and the real labels.
- Report the classification error of *k-means* and also compare with the result of *k-means*++.

References

Elder, J, Nisbet, R, Miner, G (eds.) (2009) *Handbook of Statistical Analysis and Data Mining Applications*, Academic Press, ISBN 0080912036, 9780080912035.

Hastie, T, Tibshirani, R, Friedman, J. (2013) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics*, New York, ISBN 1489905189, 9781489905185.

Hothorn, T, Everitt, BS. (2014) *A Handbook of Statistical Analyses using R*, CRC Press, ISBN 1482204592, 9781482204599.

https://en.wikipedia.org/wiki/Coefficient_of_determination

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0157077