Part IX
Trees, SVM, and Random Forest Discriminants

1 rpart Analyses – the Pima Dataset

Note the rpart terminology:

size	Number of leaves	Used in plots from `plotcp()`
`nsplit`	Number of splits = size - 1	Used in `printcp()` output
`cp`	Complexity parameter	Appears as CP in graphs and printed output. A smaller `cp` gives a more complex model, i.e., more splits.
rel error	Resubstitution error measure	Multiply by baseline error to get the corresponding absolute error measure. In general, treat this error measure with scepticism.
xerror	Crossvalidation error estimate	Multiply by baseline error to get the corresponding absolute error measure.

After attaching the *MASS* package, type `help(Pima.tr)` to get a description of these data. They are relevant to the investigation of conditions that may pre-dispose to diabetes.

1.1 Fitting the model

Fit an rpart model to the `Pima.tr` data:

```
> library(MASS)
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(Pima.rpart)
```

The formula `type` $\sim$ `.` has the effect of using as explanatory variables all columns of `Pima.tr` except `type`. The parameter cp is a complexity parameter; it penalizes models that are too complex. A small penalty leads to more splits. Note that cp, at this initial fit, has to be small enough so that the minimum of the cross-validated error is attained.

Try this several times. The result will vary somewhat from run to run. Why?

One approach is to choose the model that gives the absolute minimum of the cross-validated error. If the fitting has been repeated several times, the cross-validation errors can be averaged over the separate runs.

A more cautious approach is to choose a model where there is some modest certainty that the final split is giving a better than chance improvement. For this, the suggestion is to choose the smallest number of splits so that cross-validated error rate lies under the dotted line. This is at a height of (minimum cross-validated error rate) + 1 standard error.

The choice of 1 SE, rather than some other multiple of the SE, is somewhat arbitrary. The aim is to identify a model where the numbers of splits stays pretty much constant under successive runs of `rpart`. For this it is necessary to move back somewhat, on the curve that plots the cross-validated error rate against `cp`, from the flat part of the curve where the cross-validated error rate is a minimum. The 1 SE rule identifies a better-defined value of `cp` where the curve has a detectable negative slope.

For example, in one of my runs, the 1 SE rule gave `cp`=0.038, with size=4. The resulting tree can be cut back to this size with:

```
> Pima.rpart4 <- prune(Pima.rpart, cp=0.037)
```

The value of cp is chosen to be less than 0.038, but more than the value that led to a further split.

Plot the tree, thus:

```
> plot(Pima.rpart4)   # NB: plot, not plotcp()
> text(Pima.rpart4)      # Labels the tree
```

Note also the printed output from

```
> printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.97059 0.097791
3 0.073529      2   0.61765 0.89706 0.095752
4 0.058824      3   0.54412 0.88235 0.095305
5 0.014706      4   0.48529 0.77941 0.091785
6 0.010000      7   0.44118 0.89706 0.095752
```

Get the absolute cross-validated error by multiplying the root node error by xerror. With nsplit=3 (a tree of size 4 leaves), this is, in the run I did, 0.3327*0.7006 = 0.233. (The accuracy is obtained by subtracting this from 1.0, i.e., about 77%.)

Where it is not clear from the graph where the minimum (or the minimum+SE, if that is used) lies, it will be necessary to resort to use this printed output for that purpose. It may be necessary to use a value of cp that is smaller than the default in the call to rpart(), in order to be reasonably sure that the optimum (according to one or other criterion) has been found.

Exercise 1: Repeat the above several times, i.e.

```
> library(rpart)
> Pima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(Pima.rpart)
```

(a) Overlay the several plots of the cross-validated error rate against the number of splits. Why does the cross=validated error rate vary somewhat from run to run? Average over the several runs and choose the optimum size of tree based on (i) the minimum cross-validated error rate, and (ii) the minimum cross-validated error rate, plus one SE.

(b) Show the relative error rate on the same graph.

NB: You can get the error rate estimates from:

```
> errmat <- printcp(Pima.rpart)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped
```

```
Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 1.00000 0.098518
3 0.073529      2   0.61765 0.88235 0.095305
4 0.058824      3   0.54412 0.85294 0.094370
5 0.014706      4   0.48529 0.69118 0.088180
6 0.010000      7   0.44118 0.80882 0.092863
```

```
> colnames(errmat)        # Hints at what will come next

[1] "CP"        "nsplit"    "rel error" "xerror"    "xstd"

> resub.err <- 1-0.3327*errmat[,"rel error"]
> cv.err <- 1-0.3327*errmat[,"xerror"]
```

Exercise 2: Prune the model back to give the optimum tree, as determined by the one SE rule. How does the error rate vary with the observed value of `type`? Examine the confusion matrix. This is most easily done using the function `xpred()`. (The following assumes that 3 leaves, i.e., `cp` less than about 0.038 and greater than 0.011, is optimal.)

```
> Pima.rpart <- prune(Pima.rpart, cp=0.037)
> cvhat <- xpred.rpart(Pima.rpart4, cp=0.037)
> tab <- table(Pima.tr$type, cvhat)
> confusion <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> dimnames(confusion) <- list(ActualType=c("No","Yes"),
+ PredictedType=c("No","Yes"))
> print(confusion)

          PredictedType
ActualType        No       Yes
       No  0.8863636 0.1136364
       Yes 0.5000000 0.5000000
```

The table shows how the predicted accuracy changes, depending on whether the correct type is `Yes` or `No`.

How would you expect the overall estimate of predictive accuracy to change, using the same fitted rpart model for prediction:

- if 40% were in the `Yes` category?
- if 20% were in the `Yes` category?

2 rpart Analyses – `Pima.tr` and `Pima.te`

Exercise 3 These exercises will use the two data sets `Pima.tr` and `Pima.te`.
What are the respective proportions of the two types in the two data sets?

Exercise 3a: Refit the model.

```
> trPima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> plotcp(trPima.rpart)
> printcp(trPima.rpart)
```

```
Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 1.01471 0.098864
3 0.073529      2   0.61765 0.94118 0.097014
4 0.058824      3   0.54412 0.89706 0.095752
5 0.014706      4   0.48529 0.79412 0.092331
6 0.010000      7   0.44118 0.75000 0.090647
```

Choose values of cp that will give the points on the graph given by plotcp(trPima.rpart). These (except for the first; what happens there?) are the geometric means of the successive pairs that are printed, and can be obtained thus:

```
> trPima.rpart <- rpart(type ~ ., data=Pima.tr, method="class")
> cp.all <- printcp(trPima.rpart)[, "CP"]

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.91176 0.096186
3 0.073529      2   0.61765 0.79412 0.092331
4 0.058824      3   0.54412 0.77941 0.091785
5 0.014706      4   0.48529 0.66176 0.086846
6 0.010000      7   0.44118 0.70588 0.088822

> n <- length(cp.all)
> cp.all <- sqrt(cp.all*c(Inf, cp.all[-n]))
> nsize <- printcp(trPima.rpart)[, "nsplit"] + 1

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34
```

```
n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.91176 0.096186
3 0.073529      2   0.61765 0.79412 0.092331
4 0.058824      3   0.54412 0.77941 0.091785
5 0.014706      4   0.48529 0.66176 0.086846
6 0.010000      7   0.44118 0.70588 0.088822
```

Observe that `nsize` is one greater than the number of splits.

Prune back successively to these points. In each case determine the cross-validated error for the training data and the error for test data. Plot both these errors against the size of tree, on the same graph. Are they comparable? The following will get you started:

```
> tr.cverr <- printcp(trPima.rpart)[, "xerror"] * 0.34

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, method = "class")

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.91176 0.096186
3 0.073529      2   0.61765 0.79412 0.092331
4 0.058824      3   0.54412 0.77941 0.091785
5 0.014706      4   0.48529 0.66176 0.086846
6 0.010000      7   0.44118 0.70588 0.088822

> n <- length(cp.all)
> trPima0.rpart <- trPima.rpart
> te.cverr <- numeric(n)
> for (i in n:1){
+    trPima0.rpart <- prune(trPima0.rpart, cp=cp.all[i])
+    hat <- predict(trPima0.rpart, newdata=Pima.te, type="class")
+    tab <- table(hat, Pima.te$type)
+    te.cverr[i] <- 1-sum(tab[row(tab)==col(tab)])/sum(tab)
+  }
```

Comment on the comparison, and also on the dependence on the number of splits.

3 Analysis Using *svm*

Exercise 4: Compare the result also with the result from an SVM (Support Vector Machine) model. For getting predictions for the current test data, it will be necessary, for models fitted using `svm()`, to use the `newdata` argument for `predict()`. Follow the prototype

```
> library(e1071)
> library(MASS)
> trPima.svm <- svm(type ~ ., data=Pima.tr)
> hat <- predict(trPima.svm, newdata=Pima.te)
> tab <- table(Pima.te$type, hat)
> 1-sum(tab[row(tab)==col(tab)])/sum(tab)
> confusion.svm <- rbind(tab[1,]/sum(tab[1,]), tab[2,]/sum(tab[2,]))
> print(confusion.svm)
```

4 Analysis Using *randomForest*

Random forests can be fit pretty much automatically, with little need or opportunity for tuning. For datasets where there is a relatively complex form of dependence on explanatory factors and variables, random forests may give unrivalled accuracy.

Fiting proceeds by fitting (in fact, overfitting) many (by default, 500) trees, with each tree fitted to a different bootstrap sample of the data, with a different random sample of variables also. Each tree is taken to its full extent. The predicted class is then determined ny a simple vote over all trees.

Exercise 5: Repeat the previous exercise, but now using `randomForest()`

```
> library(randomForest)
> Pima.rf <- randomForest(type~., data=Pima.tr, xtest=Pima.te[,-8],
+                         ytest=Pima.te$type)
> Pima.rf

Call:
 randomForest(formula = type ~ ., data = Pima.tr, xtest = Pima.te[,      -8], ytest = Pima.te$type)
              Type of random forest: classification
                    Number of trees: 500
No. of variables tried at each split: 2

       OOB estimate of  error rate: 29.5%
Confusion matrix:
     No Yes class.error
No  107  25   0.1893939
Yes  34  34   0.5000000
                Test set error rate: 23.19%
Confusion matrix:
     No Yes class.error
No  190  33   0.1479821
Yes  44  65   0.4036697
```

Note that OOB = Out of Bag Error rate, calculated using an approach that has much the same effect as cross-validation, applied to the data specified by the `data` parameter. Notice that `randomForest()` will optionally give, following the one function call, an assessment of predictive error for a completely separate set of test data that has had no role in training the model. Where such a test set is available, this provides a reassuring check that `randomForest()` is not over-fitting.

Exercise 6: The function `tuneRF()` can be used for such limited tuning as `randomForest()` allows. Look up `help(tuneRF()`, and run this function in order to find an optimal value for the parameter `mtry`. Then repeat the above with this optimum value of `mtry`, and again compare the OOB error with the error on the test set.

Exercise 7: Comment on the difference between `rpart()` and `randomForest()`: (1) in the level of automation; (2) in error rate for the data sets for which you have a comparison; (3) in speed of execution.

5 Class Weights

Exercise 8: Analysis with and without specification of class weights Try the following:

```
> Pima.rf <- randomForest(type ~ ., data=Pima.tr, method="class")
> Pima.rf.4 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.6,.4))
> Pima.rf.1 <- randomForest(type ~ ., data=Pima.tr, method="class", classwt=c(.9,.1))
```

What class weights have been implicitly assumed, in the first calculation?

Compare the three confusion matrices. The effect of the class weights is not entirely clear. They do not function as prior probabilites. Prior probabilities can be fudged by manipulating the sizes of the bootstrap samples from the different classes.

6 Plots that show the "distances" between points

In analyses with `randomForest`, the proportion of times (over all trees) that any pair of observations ("points") appears at the same terminal node can be use as a measure of proximity between the pair. An ordination method can then be used to find a low-dimensional representation of the points that as far as possible preserves the distances or (for non-metric scaling) preserves the ordering of the distances. Here is an example:

```
> Pima.rf <- randomForest(type~., data=Pima.tr, proximity=TRUE)
> Pima.prox <- predict(Pima.rf, proximity=TRUE)
> Pima.cmd <- cmdscale(1-Pima.prox$proximity)
> Pima.cmd3 <- cmdscale(1-Pima.prox$proximity, k=3)
> library(lattice)
> cloud(Pima.cmd3[,1] ~ Pima.cmd3[,2]*Pima.cmd3[,2], groups=Pima.tr$type)
```

Exercise 9: What is the plot from `cloud()` saying? Why is this not overly surprising?:

Note: The function `randomForest()` has the parameter `classwt`, which seems to have little effect.

6.1 Prior probabilities

The effect of assigning prior probabilities can be achieved by choosing the elements of the parameter `sampsize` (the bootstrap sample sizes) so that they are in the ratio of the required prior probabilities.

```
> ## Default
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      68))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 28.5%
Confusion matrix:
     No Yes class.error
No  108  24   0.1818182
Yes  33  35   0.4852941

> ## Simulate a prior that is close to 0.8:0.2
> randomForest(type ~ ., data=Pima.tr, sampsize=c(132,33))
```

```
Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(132,      33))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 27%
Confusion matrix:
     No Yes class.error
No  120  12   0.0909091
Yes  42  26   0.6176471

>   # Notice the dramatically increased accuracy for the No's
> ## Simulate a prior that is close to 0.1:0.9
> randomForest(type ~ ., data=Pima.tr, sampsize=c(17,68))

Call:
 randomForest(formula = type ~ ., data = Pima.tr, sampsize = c(17,      68))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 39%
Confusion matrix:
    No Yes class.error
No  60  72   0.5454545
Yes  6  62   0.0882353
```

With small sample sizes, it may be beneficial to fit a greater number of trees.

7 Further Examples

(a) Look up the help page for the dataset `Vehicle` from the *mlbench* package:

  (a) Fit a linear discriminant analysis model (`lda()`). Plot the first two sets of discriminant scores, identifying the different vehicle types;

  (b) Try quadratic discriminant analysis (`qda()`) for comparison;

  (c) Compare the accuracy with that from fitting a random forest model. Use the proximities to derive a three-dimensional representation of the data. Examine both the projection onto two dimensions and (using `rgl()` from the *rgl* package) the three-dimensional representation.

  (d) Compare the two (or more; you might derive more than one plot from the proximities) low-dimensional plots. Do any of the plots offer any clue on why quadratic discriminant analysis is so effective?

(b) Repeat Exercise 1, now with the dataset `fgl` (forensic glass data) in the *MASS* package. Note that, for these data, `qda()` fails. Why does it fail?