Lecture 5: Classification methods

Alan Lee

Department of Statistics
STATS 769 Lecture 5

August 4, 2015

Outline

Introduction

Classsification

Error rates

Methods

Comparison of methods

Today's agenda

In this lecture we continue our discussion of classification methods Some of the ideas should be familiar from STATS 760. Today we will cover the same topics as the last lecture, but from the point of view of classification rather than prediction.

- Trees
- Neural networks
- Boosting and Bagging, particularly random forests
- Use of the `caret` package

We will use the spam data from HTF as a running example.

Classification

- In the last lecture we looked at prediction problems where the response variable was continuous, and the predictions were based on fitting models by least squares.
- Now we go back to the case where the response variable is categorical, representing membership of one of K groups or classes. Thus, the response variable is a factor with K levels (as in logistic regression).
- We will discuss the modifications we need to make to the prediction methods so that they can be used for classification.

Connection with regression

Most of the regression methods we have discussed can be adapted to the classification problem. If Y is the response, taking values $1, 2, \ldots, K$, we can define new binary variables Y_k, $k = 1, 2, \ldots, K$ by

$$Y_k = \begin{cases} 1, & Y = k \\ 0, & Y \neq k \end{cases}$$

We then fit K separate regression models to the variables Y_k by least squares, getting K predictors f_k, $k = 1, 2, \ldots, K$. We then define the overall predictor by

$$f(x) = \text{argmax}_{k=1,2,\ldots,K} f_k(x).$$

Alternative using logistic/multinomial regression

Alternatively, we can use a different goodness of fit criterion: set

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{j=1}^{K} \exp(f_j(x))}$$

where $f_k(x)$ could be a linear function, a neural net etc. We choose the functions to minimise

$$-\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log(p_k(x_i)).$$

For $k = 2$ and f_k linear, this is just logistic regression.

Why is this reasonable?

- If the the π_k are unrestricted, and $y_i \geq 0$, $\sum_i y_i = 1$ the expression

$$-\sum_{k=1}^{K} y_k \log(\pi_k)$$

  takes its minimum value $\sum_{k=1}^{K} y_k \log(y_k)$ when $\pi_i = y_i$.
- The criterion gives the maximum likelihood estimates if the data are assumed to be multinomial.

Error rates

- We defined the prediction error in Lecture 3 as the probability of misclassification, estimated by the error rate, the proportion of misclassified cases. Thus we have an in-sample and an out-of-sample version of the error rate, with the out-of-sample being the more realistic one.
- We can cross-classify cases according to two criteria: (a) the actual value of the response, and (b) the predicted value. The sum of the off-diagonal elements of this table, divided by the number of cases, gives the misclassification rate.
- We can also think of our prediction methods as giving a set of estimated probabilities $p_k(x)$ that represent the estimated probability that a case with covariates x will be in class k.
- Then the minimum value of the fitting criterion

$$-\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(p_k(x_i))$$

can also be used as a measure of the classification error.

Error rates (cont)

As in regression, we can get estimates of the out-of-sample error rate using either a test set, cross-validation or the bootstrap. In both regression and classification, the error when using a prediction rule $f(x)$ is measured by

$$\frac{1}{M}\sum_{i=1}^{M} L(y_i, f(x_i))$$

where $L(y, f(x))$ is the error when predicting a case whose true response is y with the prediction $f(x)$. For least squares, the loss function is $(y - f(x))^2$. For a binary response, with a predictor $f(x)$ taking values 0 and 1, it is

$$L(y, f(x)) = \begin{cases} 0, & y = f(x) \\ 1, & y \neq f(x) \end{cases}$$

Neural nets

- There is one output node for each response category.
- At the kth output node we accumulate

$$f_k(x) = \beta_{0k} + \sum_j \beta_{kj}\sigma(\alpha_{k0} + \sum_m \alpha_{km}^T x).$$

- Outputs are

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{j=1}^K \exp(f_j(x))}$$

- Fitting is done by minimizing the criterion on slide 6.
- Assign a case with covariate x to class having the largest value of $p_k(x)$. (equivalently the largest value of $f_k(x)$)
- Use `linout=FALSE`.
- Make sure the output variable is a factor.

Example: Spam

- As you no doubt know, spam is unwanted email, the electronic equivalent of junk mail.
- Most mail programs have a way of recognizing such messages as spam and diverting them to a junk mail folder.
- This is usually done by recognizing key words and characters that are likely to occur in either genuine messages and spam, but not both. These words will be different for every user.

Example: Spam (cont)

- The data for this example consist of measurements on 4601 email messages. There are 58 variables representing the relative frequency of certain words and characters (e.g. the proportion of words in the message that are "George".
- There are additional variables representing the length of strings of consecutive capital letters, and the response variable is a binary variable (0=genuine message, 1=spam).
- The aim is to develop a classification rule for classifying an email as genuine or spam.

The spam data: Neural nets

Suppose data is in a data frame spam, with variables V1,...,V58. the output the variable V58, coded as 0=genuine, 1=spam. We will divide the data set into training and test sets:

```
library(nnet)
X = scale(spam[,-58])
y = factor(spam[,58])
spam = data.frame(X,y)
# pick 75% data for training set
use = sample(dim(spam)[1], 0.75*dim(spam)[1])
training = spam[use,]
test= spam[-use,]
```

The spam data: Neural nets (cont)

Fit the neural net and calculate apparent and test error:

```
> fit2 = nnet(y~., data=training, method="class", size = 5,
maxit=1000, decay=0.1)
> # in-sample error
> mytable.training = table(training$y,predict(fit2)>0.5)
> mytable.training
      FALSE TRUE
    0  2046   44
    1    47 1313
> (44+47)/sum(mytable.training)
[1] 0.02637681
# out-of sample error
mytable.test = table(test$y,predict(fit2, newdata=test)>0.5)
> 1-sum(diag(mytable.test))/sum(mytable.test)
[1] 0.06342311
```

Trees

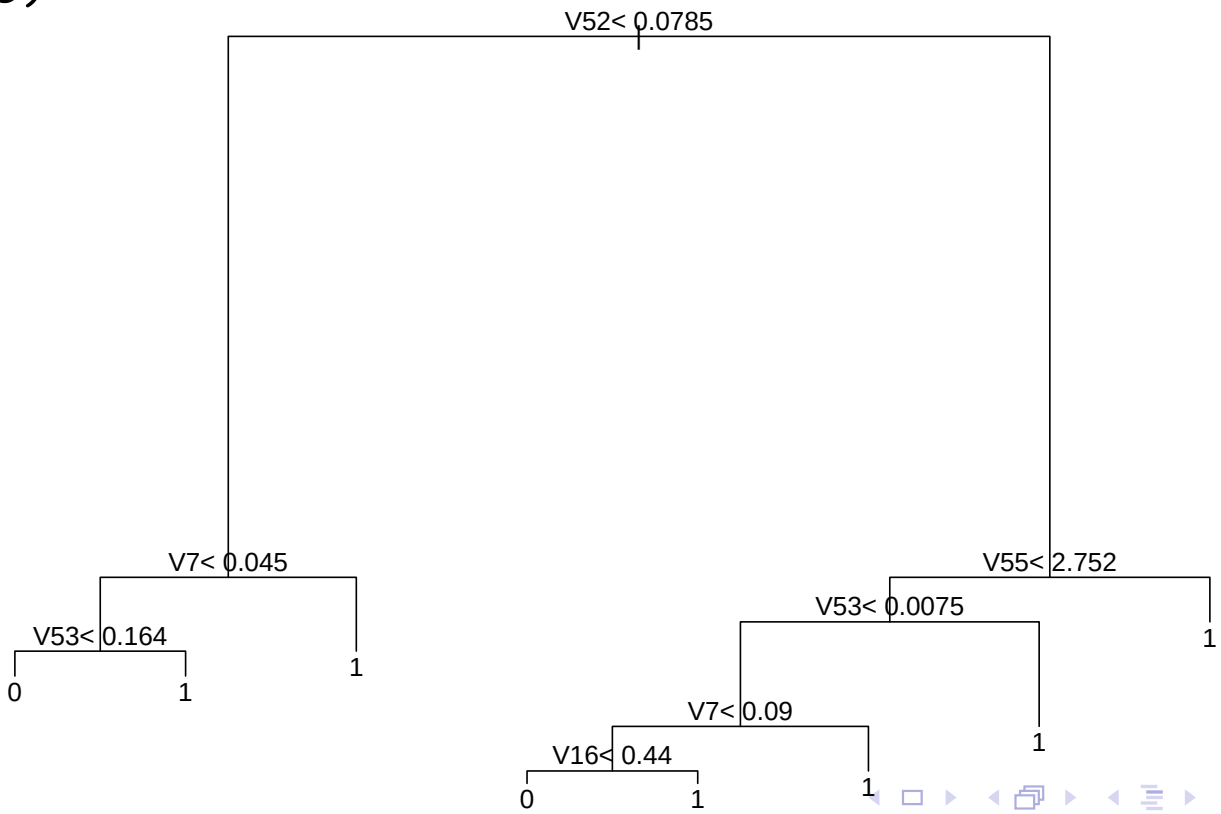
- For classification, we use a different splitting strategy. Suppose at a node, the proportions of cases at the node falling into the different response categories are $\hat{p}_1, \ldots, \hat{p}_K$.
- Define the "node impurity" by the Gini index $1 - \sum_{k=1}^{K} \hat{p}_k^2$. This takes a minimum value of 0 when all the cases are in a single category and a maximum of $1 - 1/K$ when the numbers of cases in each category are the same.
- We then split the cases at the node into two subgroups, in such a way to maximise the difference between the "parent" node impurity and the weighted sum of the impurities of the two " child" nodes (weighted by the numbers at each child node). Thus, we are choosing the split that gives the biggest decrease in the node impurities. (Compare with regression where we split to get the biggest reduction in the residual sum of squares.)

Trees (cont)

- At each terminal node, we record the class with the highest proportion of cases (the majority group)
- To predict a new case, we pass it through the tree. It winds up at node m say. We assign the new case to the "majority group" of node m.
- Note that each node of the tree corresponds to a set of probabilities (the proportions of cases falling to the various classes.)

The spam data revisited

```
fit3 = rpart(y∼., data=training, method = "class",
parms = list(split="gini"), cp=0.01)
plot(fit3)
text(fit3)
```



What is cp?

The parameter cp controls the tree size. (can also be controlled by limiting the number of terminal nodes.)

- For prediction, we stop growing the tree when a split fails to reduce the RSS by cp times the RSS of the null tree (the one fitting a mean to all the data)
- For classification, when a split fails to reduce node impurity by cp times the node impurity of the null tree.

The decrease in node impurity of a split is measured by $nG - n_L G_L - n_R G_R$ where G is the Gini index of the node being split, n is the number of data points at the node, with corresponding interpretations for G_L, G_R, N_L, N_R, (L=left node, R=right node). See ISLR p 312 for more information.

Random forests for classification.

- These work equally well for classification. The individual trees are grown as described on a previous slide.
- When combining the results, instead of averaging, we use a "majority vote" rule: we classify the case into the group chosen by the majority of the trees.

Comparisons: spam data.

Classification errors

	In-sample	Out-of-sample
Linear	0.110	0.115
Logistic	0.080	0.115
Neural4	0.031	0.065
Neural8	0.011	0.068
Tree	0.097	0.106
Boosted tree	0.058	0.072
Random Forest	0.049	0.048

Code

Tree

```
myfit1 = rpart(y∼., data=training, method = "class",
parms = list(split="gini"), cp=0.01)
```

Boosted tree

```
y.t = factor(y)
training = data.frame(X, y.t)
fm =y.t∼btree(data.frame(X), tree_controls =
ctree_control(maxdepth = 5))
myfit2 = mboost(fm, data = training.t,
family=Binomial(), control = boost_control(mstop
=160, nu = 0.1))
```

Random forests

```
myfit3=randomForest(y.t∼., data=training)
```