Introduction to Data Science Final

Ryan Greenup ; 1780-5315

October 24, 2019

Contents

06_Practical; Trees Ryan Greenup 17805315 23 August 2019

“‘{r setup, include=FALSE, include = FALSE, results = “hide”, eval = TRUE} knitr::opts_chunk$set(echo = TRUE)

if(require(‘pacman’)){ library(‘pacman’) }else{ install.packages(‘pacman’) library(‘pacman’) }

pacman::p_load(caret, scales, ggplot2, rmarkdown, shiny, ISLR, class, BiocManager, corrplot, plotly, tidyverse, latex2exp, stringr, reshape2, cowplot, ggpubr, rstudioapi, wesanderson, RColorBrewer, colorspace, gridExtra, grid, car, boot, colourpicker, tree, ggtree, mise, rpart, rpart.plot, knitr, MASS, magrittr) #Mass isn't available for R 3.5...

set.seed(1) # Set the seed such that we might have comparable results

Wipe all the damn variables to prevent mistakes

mise()

kabstr <- function(df){

data.frame(variable = names(df), class = sapply(df, typeof), first_values = sapply(df, function(x) paste0(head(x), collapse = “,”)), row.names = NULL) %>% kable()

} # Pretty Structure Print

```
* feed the column into `factor()`, specify the `levels = c()` in
    ascending order, specify `ordered = TRUE`


### Preview the data:

```{r}
CSeat.tb <- as_tibble(Carseats)


thresh <- (mean(CSeat.tb$Sales)+0.5*sd(CSeat.tb$Sales)) %>% round()

CSeat.tb$CatSales <- ifelse(CSeat.tb$Sales > thresh, "High", "Low")
CSeat.tb$CatSales <- factor(x = CSeat.tb$CatSales, levels = c("Low",
    "High"), ordered = TRUE)




CSeat.tb <- CSeat.tb[,c(12, 2:11)]

  # Anotherway to reorder
    dplyr::select(CSeat.tb, CatSales, everything())
```

Fit

Now let's use all of the data to create a classification tree of the sales category.

Remember that we can use . to represent all the variables of a data frame;

- If `Sales` was still in the data frame we could pull it out by prefixing it with - but I think that's awfully confusing, instead I would recommend specifying a subset that pulls out undesired columns or just making another dataframe, it's way too easy to make a mistake like that IMO

```
CatSalesModTree <- tree(formula = CatSales ~ ., data =
    as.data.frame(CSeat.tb))
CatSalesModTree
nrow(CSeat.tb)

CatSalesModTree %>% summary()
```

Plot the Model
```

Base

This summary is next to useless so let's plot it, when plotting it it is necessary to call `text` afterwards in order order to have lables:

```
plot(CatSalesModTree)
text (CatSalesModTree, pretty = 0, cex = 0.6)

# #
# CatSalesModTree.rpart <- rpart(formula = CatSales ~ ., data =
    CSeat.tb)
# rpart.plot(CatSalesModTree.rpart, box.palette="RdBu",
    shadow.col="gray", nn=TRUE)
```

Use `rpart`

The base plot looks truly awful, instead I would recommend using the `rpart` package, it creates a tree model using identical syntax to `tree` and then you can call `rpart.plot` directly over the model and it will work.

Be mindful that the `tree` package will split the data until each node has less than 20 observations, whereas the 'rpart' package automatically performs 10-fold cross validation [1], the ratio contained in the plot is the ratio of terms satisfying the specified condition and the percentage is the percentage of all observations thence contained [2]

```
#plot(CatSalesModTree)
#text (CatSalesModTree, pretty = 0, cex = 0.6)

CatSalesModTree.rpart <- rpart(formula = CatSales ~ ., data =
    CSeat.tb)
rpart.plot(CatSalesModTree.rpart, box.palette="OrGy",
    shadow.col="gray", nn=TRUE)
```

[1] rpart.control has a default list entry of `xval=10` which corresponds to the number of CV folds.
[2] Refer to the rpart Vignette

I have no clue if an `rpart` tree object behaves well so I'm just going to use one for plotting and the other for modelling up until I have time to look into them.

Evaluate the tree by using a validation Split

In order to very quickly evaluate the performance of this model, split the data and use the model on unseen data.

```
train <- sample(1:nrow(Carseats), 200)

#Create the Categorical variable

CSeat <- Carseats
CSeat$CatSales <- ifelse(Carseats$Sales <
   round(mean(Carseats$Sales)), "Low", "High")
CSeat$CatSales <- factor(CSeat$CatSales, levels = c("Low", "High"),
   ordered = TRUE)

#Create the model on the training data only
carseats.tree <- tree(CatSales ~ ., data = CSeat[train,names(CSeat)
   != "Sales"])
 # plot(carseats.tree) ;text(carseats.tree, pretty = 0, cex = 0.6)

carseats.tree.rpart <- rpart(CatSales ~ ., data = CSeat[train,-1],
   model = TRUE) # Model = True stops later errors
#rpart.plot(carseats.tree.rpart, nn = TRUE)

# Create Predictions on the testing data
test.pred <- predict(object = carseats.tree, newdata =
   CSeat[-train,], type = "class" )

# Filter out the observations in the testing data
test.obs <- CSeat$CatSales[-train]

# Now create the confusion Matrix
  # This package prevents making mistakes
  conf.mat <-  caret::confusionMatrix(data = test.pred, reference =
     test.obs)
  # This could otherwise be created by using, always go prediction,
     reference as a standard
  table("prediction" = test.pred, "reference" = test.obs)
```

The Misclassification Rate for unseed testing data is `r (1-conf.mat$overall[1] )%>% round(2)%>% as.numeric()%>% percent()` which is reasonably low, suggesting that the accuracy

of this model is acceptable, particularly because this models primary purpose is interetability not predictive capacity.

Tree Pruning

This model may be overfit, it is necessary instead to determine whether or not to 'prune' the tree, that is reduce the model flexibility by reducing the number of nodes at the end of the tree. So, in this case, the number of nodes at the end of the tree is an indicator of the model flexibility, by default, the `tree` package creates nodes by choosing a threshold that minimises the RSS until the number of observations in each node is 20.

The problem with this strategy is that the model may be overfit, it may be to flexible and the results will suffer from very little model bias but far too much variance.

The process of reducing flexibility in order to improve the model by balancing bias and variation is called 'pruning'.

In order to prune the tree the best strategy to implement is 10-fold cross validation, fourtunately this is all built in and we don't really have to think about it, the `cv.tree()` function will do it automatically.

Using the the `cv.tree()` package

The `cv.tree` package minimises the deviance by default, the texbook provides [3] that the misclassification rate is preferable if preiction accuracy of the final purened tree is the goal, deviance is an analogy to RSS in the classification sence [4], but this is outside the scope of this unit.

In order to specify to `cv.tree` that the cross validation process should aim to minimise the '**misclassification rate**' rather than the '**deviance**' specify the function as `prune.misclass` to `cv.tree()`; be super careful, because we are changing the function, the `dev` output in the list will NOT be deviance, the values will be misclassificaiton

```
set.seed(3)
 tree.mod.cv <- cv.tree(object = carseats.tree, FUN = prune.misclass)
# tree.mod.cv <- cv.tree(object = CatSalesModTree, FUN =
   prune.misclass)
names(tree.mod.cv)[2] <- "misclassification"

tree.mod.cv #%>% str()
```

[3] classref
[4] What is Deviance

This gives us a correspondence between the limiting size of the final node (which is inversely proportional to the flexibility of the model) and the expected misclassification rate on testing data.

Because this is Cross Validation, the error will be an estimation of error unseen by the model, hence the size that reflects the minimum value should be returned.

Plot the Testing Error

```
nodeSize <- factor(tree.mod.cv$size, ordered = TRUE)
cv.error <- tibble(nodeSize = tree.mod.cv$size, error =
    tree.mod.cv$misclassification)

minval <- cv.error[cv.error$error<=min(cv.error$error),]
minNode <- min(minval$nodeSize)

ggplot(cv.error, aes(x = nodeSize, y = error), groups = 1) +
  geom_point(size = 4, col = "IndianRed") +
  geom_line(alpha = 0.7, aes(col = -error), lwd = 1, show.legend =
      FALSE) +
  labs(y = "Estimated Misclassification Rate", x = "Maximum Leaf
      Size", ttitle = "Cross Validation of Tree Model") +
  geom_vline(xintercept = minNode, col = "purple") +
#  geom_hline(yintercept = minval$error) +
  theme_classic()
```

The plot suggests that the minimum misclassification rate for this model will occur when the nodesize is limited to `r minNode`.

Prune the Tree

Prune the tree by using the `prune.misclass` function:

```
Cars.Clas.prunce.cv <- prune.misclass(CatSalesModTree, best =
    minNode)
plot(Cars.Clas.prunce.cv)
text(Cars.Clas.prunce.cv, pretty = 0, cex = 0.7)
```

Asess the pruned model against the test data

Compare the model returned by CV to the unseen testing data:

```
test.preds <- predict(object = Cars.Clas.prunce.cv, newdata =
    CSeat[-train, -1], type = "class")
test.obs  <- CSeat[-train, -1]$CatSales

cfConfMat <- confusionMatrix(data = test.preds, reference = test.obs)
mcr <- 1-cfConfMat$overall[1]
```

from this it can be determined that the misclassification rate for this pruned model is now `r percent(round(mcr, 2))`

This tree represents the best trade off between a model with too much variance and a model with too much bias.

Regression Trees

Consider the `Boston` Data set:

```
# kable(head(Boston))
head(Boston)
# kabstr(Boston)
# dim(Boston)
# summary(Boston)
```

This data set is a measurement of the crime rate throughout the Boston area.

Seperate training Data

Split the data into a training and test set in order to evaulate the difference caused by the cross validation technique

```
set.seed(1)
train = sample(1:nrow(Boston), size = 0.52*nrow(Boston))
```

```
names(Boston)
```

Fit a Regression Tree to the Training Data

In order to fit the regression tree to the training data use the train variable as a subset:

```
bost.mod.tree <- tree(formula =medv ~., data = Boston, subset =
    train)
# summary(bost.mod.tree)
plot(bost.mod.tree)
text(bost.mod.tree, pretty = 0, cex = 0.7)
```

Evalueate the Performance

```
modSum <- summary(bost.mod.tree)
rss <- modSum$dev
rmse <- sqrt(modSum$dev/length(length(bost.mod.tree$y)))
rmsernd <- signif(rmse, 1) * 1000
leaves <- summary(bost.mod.tree)$size
```

This model has `r leaves` terminal nodes, the performance of this model can be evaulated by measuring the ***RSSS*** which is evaulated by the mode, this provides that the average distance from data point to model prediction is $\pm$ `r signif(rmse, 1)` K USD.

Prune the tree with CV

Perform Cross Validation

```
cvvals <- cv.tree(object = bost.mod.tree, K = 10)
error <- sqrt(cvvals$dev)/length(bost.mod.tree$y)
TermNodes <- factor(x = cvvals$size, ordered = TRUE)

dvdf <- tibble(TermNodes, error)
```

```
#Return the best number of leaves
minerror <- min(dvdf$error)
optn <- min(dvdf$TermNodes[dvdf$error==minerror]) #There could be
    multiple optimum values
```

Visualise the validaiton Error

```
ggplot(dvdf, aes(x = TermNodes, y = error, group = 1)) +
  geom_point(col = "IndianRed", size = 3) +
  geom_line(col = "RoyalBlue") +
  geom_vline(xintercept = as.numeric(optn), col = "purple") +
  theme_classic() +
  labs(y = "Expected Error From model to Unseen Data ($50 K)", x =
      "Number of Terminal Nodes")
```

The model suggests that the appropriate number of terminal nodes to choose is `r optn` and hence the model ought to be left unchanged, however we will prune it for the sake of practice

Prune the Tree

In order to prune the tree use the `prune.tree` function and specify `best=` as the desired number of leaves:

```
prun.reg.mod <- prune.tree(tree = bost.mod.tree, best = 5)
plot(prun.reg.mod)
text(prun.reg.mod, pretty = 0, cex = 0.7)
```

Using rpart

This could have all been done in one call to rpart like so, however here the data is not split in order to create a more accurate model.

```
BosnonSyn <- Boston
bosTree <- rpart(formula = medv ~ ., data = Boston)
rpart.plot(bosTree)

nicenames <- c("Crime Rate", "Large Zoning Proportion", "Proportion
    of Industry", "Near River?", "NO Polution Level PPM", "Number of
    Rooms", "Proportion of 'Old' Buildings", "Distance to CBD's",
    "Accessability of Highway", "Tax Rate", "Ratio of Teachers",
    "Number of Residents of African-Americans Descent", "Property
    Value")
```