Lab 4: decision trees

Paul Song

Week 4

Objectives

- Fit classification and regression trees in R
- Implement cost-complexity pruning
- Compare tree-fitting strategies: grow with a constraint, or grow-then-prune?

Throughout the lab you'll work with the `algae` data from the first homework. Initially, you'll fit a regression tree to the `a1` levels; using this as an example, you'll fit a classification tree to predict high and low `a1` levels.

To evaluate the trees, you'll check their prediction error on a 20% holdout subset of the data.

```
# import algae data
algae <- read_csv('algae.csv') %>%
  select(-starts_with('a'), a1) %>%
  mutate(season = factor(season),
         size = factor(size),
         speed = factor(speed))

# hold out 20% of data as a test set
set.seed(12521)
algae_part <- resample_partition(algae, c(test = 0.2, train = 0.8))
train <- as_tibble(algae_part$train)
test <- as_tibble(algae_part$test)
```

Regression trees

Here you'll fit a regression tree to predict algae levels. The cost-complexity pruning procedure may seem somewhat involved, so initially we can explore what will happen if we simply grow a small-ish tree by setting n_{min} (the minimum allowed node size) to a large-ish number.

```
# grow a small regression tree
# NOTE: split = 'deviance' uses RSS for regression
nmin <- 60
tree_opts <- tree.control(nobs = nrow(train),
                          minsize = nmin,
                          mindev = exp(-6))
t_small <- tree(a1 ~ ., data = train,
                control = tree_opts, split = 'deviance')
summary(t_small)
```

The nodes can be examined in detail by checking `$frame`:

```
# examine nodes
t_small$frame %>% select(1:4)
```

This includes information about the splitting variable, the node size, impurity, the prediction, and the cutpoints.

Your turn (1) Display the information in `t_small$frame` for the root node. What is the prediction for observations at the root node?

```
# examine root node
t_small$frame[1,]
t_small$frame[1,]$yval
mean(train$a1)
```

Your turn (2) Display the predictions for each of the leaf nodes (hint: use `filter()`) in ascending order. Compare these with the quantiles of `a1`. Does there seem to be any correspondence?

```
# display leaf node predictions
t_small$frame %>% filter(var == "<leaf>") %>% select(yval) %>% arrange(yval)
# compare with quantiles
quantile(algae$a1, probs = seq(0.3, 0.9, length = 6))
```

The tree can be plotted using `draw.tree`. The function has a few graphical parameters `cex` and `size` that scale the entire figure and the node size, respectively.

```
# plot tree
draw.tree(t_small, cex = 0.75, size = 2.5, digits = 2)
```

The RMSE for this tree on the test data is:

```
# test RMSE
rmse_tsmall <- rmse(t_small, test)
rmse_tsmall
```

The idea behind cost complexity pruning is that it can leverage the better fit and increased flexibility of a large tree without overfitting, and it provides a data-driven way to determine the tree size (rather than an artificial stopping rule). We can grow a much larger tree by reducing the minimum node size.

```
# grow a large tree to prune
nmin <- 2
tree_opts <- tree.control(nobs = nrow(train),
                          minsize = nmin,
                          mindev = exp(-8))
t_0 <- tree(a1 ~ ., data = train,
            control = tree_opts, split = 'deviance')
summary(t_0)
```

Your turn (3) Check the training and test RMSE, and plot the tree `t_0`.

```
# rmses
rmsetrain0 <- rmse(t_0, train)
rmsetrain0
# plot tree
draw.tree(t_0, cex = 0.2, size = 0.2, digits = 2)
```

Now cost-complexity pruning can be implemented using `cv.tree`:

```
# cost-complexity pruning
nfolds <- 8
cv_out <- cv.tree(t_0, K = nfolds)
```

The output is a little unwieldy, so it can be helpful to convert 'by hand' to a tibble. This makes it easier to select the best tuning parameter.

```
# convert to tibble
cv_df <- tibble(alpha = cv_out$k,
                impurity = cv_out$dev,
                size = cv_out$size)

# choose optimal alpha
best_alpha <- slice_min(cv_df, impurity) %>%
  slice_min(size)
```

Your turn (4) Plot the average total tree impurity against the tuning parameter, and describe the trend. Add the best tuning parameter value to the plot as a red point.

```
# plot impurity against tuning parameter
cv_df %>%
  ggplot(aes(alpha, impurity)) +
  geom_point() +
  geom_line() +
  geom_point(data = best_alpha, color = "red") +
  theme_bw()
```

The final tree can be selected using `prune.tree()`:

```
# select final tree
t_opt <- prune.tree(t_0, k = best_alpha$alpha)
summary(t_opt)
```

Your turn (5) Calculate the training and test RMSE for the selected tree `t_opt` and compare this with the training and test RMSE for the small tree. What do you notice? Is there a big improvement in this case?

```
# compare errors with simple tree
rmsetraintopt <- rmse(t_opt, train)
rmsetraintopt
```

A plot of the tree is shown below:

```r
# plot
draw.tree(t_opt, cex = 0.6, size = 2.5, digits = 2)
```

Classification tree

This part is entirely your turn. Let's suppose that an `a1` level is considered high if it exceeds 30. Construct a factor indicating high/low algae levels and fit a classification tree to the new variable using the data in the training partition. Give a plot of the tree and a table of misclassification errors on the test partition.