DS5110 HW 3 - Due Nov. 17

Kylie Ariel Bemis

9/30/2017

Instructions

Create a directory with the following structure:

- `hw3-your-name/hw3-your-name.Rmd`
- `hw3-your-name/hw3-your-name.pdf`
- `hw3-your-name/hw3-your-name-package.tar.gz`

where `hw3-your-name.Rmd` is an R Markdown file that compiles to create `hw3-your-name.pdf`, and `hw3-your-name-package.tar.gz` is your solution for Problem 10.

Do not include data in the directory. Compress the directory as `.zip`.

Your solution should include all of the code necessary to answer the problems. All of your code should run (assuming the data is available). All plots should be generated using `ggplot2`. Missing values and overplotting should be handled appropriately. Axes should be labeled clearly and accurately.

To submit your solution, create a new private post of type "Note" on Piazza, select "Individual Student(s) / Instructor(s)" and type "Instructors", select the folder "hw3", go to Insert->Insert file in the Rich Text Editor, upload your `.zip` homework solution. Title your note "[hw3 solutions] - your name" and post the private note to Piazza. **Be sure to post it only to instructors**

Part A

Problems 1–3 use the `mpg` data from `ggplot2`.

Problem 1

Fit a model that predicts highway miles per gallon using no more than 3 predictor variables. Use plots to justify your choice of predictor variables. Print the values of the fitted model parameters.

Problem 2

Plot the residuals of the fitted model from Problem 1 against the predictor variables in the model and against other potential predictor variables in the dataset. Comment on what you observe in each residual plot.

Problem 3

Fit a new model for predicting highway mileage, adding or removing variables based on the residual plots from Problem 2.

Part B

Problem 4

Write a function that performs cross-validation for a linear model (fit using `lm`) and returns the average root-mean-square-error across all folds. The function should take as arguments (1) a formula used to fit the model, (2) a dataset, and (3) the number of folds to use for cross-validation. The function should partition the dataset, fit a model on each training partition, make predictions on each test partition, and return the average root-mean-square-error.

Problem 5

Use your function from Problem 4 to compare the models you used from Part A. Report the cross-validated root-mean-square-error for the models from Problems 1 and 3. Which model was more predictive?

Part C

Problems 6–10 use example mass spectrometry imaging data. A mass spectrum is a 1D vector of intensities representing relative abundancies at different m/z values (mass-to-charge ratios), where the m/z values represent molecules of different masses. Mass spectrometry imaging (MSI) can be considered a 3D "datacube" with an m/z (mass-to-charge-ratio) dimension and 2 spatial (x/y) dimensions. MSI experiments are commonly stored in an XML-based format called imzML. An imzML file consists for an XML part (extension ".imzML") and a binary part (extension ".ibd"). Download the 3x3 example data from https://ms-imaging.org/wp/imzml/example-files-test/. The specification for the format is described elsewhere on the same website.

Problem 6

Using the `xml2` package, write a function that parses the following information from the "Example_Continuous.imzML" XML file as *numeric* data:

- "x position" of each spectrum
- "y position" of each spectrum
- "external array length" of the m/z array
- "external offset" of the m/z array
- "external array length" of each intensity array
- "external offset" of each intensity array

You are given the functions below, which you may use in your code.

```
insert_ref_groups <- function(x) {
  ref_groups <- xml_root(x) %>%
    xml_child("d1:referenceableParamGroupList") %>%
    xml_children()
  ref <- xml_child(x, "d1:referenceableParamGroupRef")
  name <- xml_attr(ref, "ref")
  ref_groups_exist <- xml_attr(ref_groups, "id") %in% name
  if ( any(ref_groups_exist) )
    group <- ref_groups[[which(ref_groups_exist)]]
  for ( g in xml_children(group) )
    xml_add_child(x, g)
  xml_remove(ref)
  x
}
```

```r
xml_find_by_attribute <- function(x, attr, value) {
  match <- xml_attr(x, attr) == value
  if ( isTRUE(any(match)) ) {
    x[[which(match)]]
  } else {
    NULL
  }
}

get_spectrum_data <- function(x, i) {
  spectrum <- x %>%
    xml_child("d1:run") %>%
    xml_child("d1:spectrumList") %>%
    xml_child(i)
  spectrum <- insert_ref_groups(spectrum)
  scan <- spectrum %>%
    xml_child("d1:scanList") %>%
    xml_child("d1:scan")
  scan <- insert_ref_groups(scan)
  data <- spectrum %>%
    xml_child("d1:binaryDataArrayList") %>%
    xml_children()
  for ( d in data )
    insert_ref_groups(d)
  data <- lapply(data, xml_children)
  for ( i in seq_along(data) ) {
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "m/z array")) )
      names(data)[i] <- "mz"
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "intensity array")) )
      names(data)[i] <- "intensity"
  }
  data$coord <- xml_children(scan)
  data[c("mz", "intensity", "coord")]
}

get_spectra_n <- function(x) {
  x %>%
    xml_child("d1:run") %>%
    xml_child("d1:spectrumList") %>%
    xml_attr("count") %>%
    as.numeric()
}

get_spectra <- function(x) {
  n <- get_spectra_n(x)
  lapply(1:n, function(i) get_spectrum_data(x, i))
}
```

Problem 7

Using the information you parsed in Problem 6 and the base R function `readBin`, write a function that reads the m/z array and all of the intensity arrays in the "Example_Continuous.ibd" binary file.

Problem 8

Write a constructor for a class that stores the coordinates, m/z array, and intensity arrays that you parsed in Problems 6 and 7. You may use either an S3 or an S4 class.

Problem 9

Write methods to access the coordinates, m/z array, and intensity arrays. Write another method to plot an image of the data for a particular m/z value.

Hint: See `geom_tile` for how to plot images in `ggplot2`. For a given m/z `value` and m/z array `mz`, a simple way to calculate its index is `which.min(mz - value)`.

Problem 10

Create an R package for the class and methods you created in Problems 8 and 9. For full credit, it should pass `R CMD check` without errors. (Warnings are okay. You do not need to include documentation.) Build the package and include the `.tar.gz` compressed file in your homework directory.