732A94 Advanced R Programming
Bonus Computer Lab
(2 extra point towards exam if this lab is passed)

Krzysztof Bartoszek, Bayu Brahmantio, Simon Jorstedt, Duc Tran
(designed by Leif Jonsson, Måns Magnusson and Shashi Nagarajan)
(updated and converted to LaTeX by Arash Haratian)

5 September 2024

Lab deadline: **5 November 23:59**
NO RESUBMISSION POSSIBLE
NO LATE SUBMISSION ALLOWED

Instructions

- Ideally this lab should be conducted by students **two by two**.
- The lab consists of writing a package that is version controlled on `github.com` or `gitlab.liu.se`.
- All students in the group should **contribute equally much** to the package. All group members have to contribute to, understand and be able to explain all aspects of the work. In case some member(s) of a group do not contribute equally this has to be reported and in this situation a formal group work contract will be signed, stipulating the consequences for further unequal contributions.
- Other significant collaborations/discussions should be acknowledged in the solution.
- To copy other's code is **NOT** allowed. Your solutions will be checked through URKUND.
- Copying solutions of others and from any online or offline resources is **NOT** allowed.
- Commit continously your addition and changes.
- Collaborations should be done using GitHub (ie you should commit using your own github account) or using GitLab.
- In the lab some functions can be marked with an *. Students **MUST do AT LEAST ONE** exercise marked with an * for each of the Labs 3 - 6 and Bonus. If only one exercise is marked with an *, then it **MUST** be done.
- The deadline for the lab is on the lab's title page.
- The lab should be turned in using an url to the repository containing the package on github/gitlab.liu.se using **LISAM**. This should also include name, liu-id and, if applicable, github user names of the students behind the project. In case of problems or if you do not have access to LISAM the url may be emailed to `baybr79@liu.se` or `marbr987@student.liu.se` or `araha147@student.liu.se` or `krzysztof.bartoszek@liu.se`.
- **NO resubmissions will be allowed for the Bonus lab.**
  NO late submissions will be allowed for the Bonus lab.
- Inside your package you may not depend on any global variable (unless it is a standardR one, like pi). Using them will result in an immediate failure of your code.If at any stage your code changes any options, these changes have to be reverted before your code finishes.
- All notes raised by Travis/GitHub Actions/GitLab CI have to be taken care of or explicitly defended in your submission.
- The seminars are there to discuss your solutions and obtain support with problems. Every group has to present at least once during the seminars in order to pass the lab part.

Contents

Chapter 1

Introduction to machine learning in R

In this lab we will continue to improve our `linreg` package by including a similar function for ridge regression.

Master students should implement one of the exercises marked with (*).

1.1 Ridge regression

You should add a new function in your `linreg` package that you call `ridgereg(formula, data, lambda)`. As with the `linreg()` function it should take a formula object as well as a dataset and return a `ridgereg` object. The `ridgereg()` function should also have the argument `lambda` to specify λ.

Ridge regression can be a good alternative when we have a lot of covariates (when $p > n$) or in the situation of multicollinearity. More information on ridge regression can be found in chapter 3.4.1 in [1].

The hyperparameter that we will tune to find the best model is the λ parameters. Unlike the linear regression situation, different scalings of the covariates in $\mathbf{X}$ will affect the results. So normalize all covariates before you do the analysis.

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{V(\mathbf{x})}}$$

If you want to compare the results you can compare with `lm.ridge()` in the `MASS` package that is parametrized in the same way. But it uses SVD decomposition so there can be small differences in the results.

1.1.1 Computations using least squares

The simple way to calculate the different coefficients and values is to use ordinary linear algebra and calculate:

Regressions coefficients:

$$\hat{\beta}^{\textbf{ridge}} = \left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^{\mathbf{T}}\mathbf{y}$$

The fitted values:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}^{\textbf{ridge}}$$

Calculate these statistics and store it in an object of class `ridgereg`. You can use either S3 objects or RC objects.

Document your function `ridgereg()` using `roxygen2`.

1.1.2 (*) Ridge regression using the QR decomposition

As in the situation with linear regression we can do the calculations using the QR decomposition. It is a little bit trickier that in the linear situation, some hints can be found here
`http://math.stackexchange.com/questions/299481/qr-factorization-for-ridge-regression`.

1.1.3 Implementing methods

As with the `linreg()` function you should implement some methods for your object.

The following methods should be implemented and be documented using `roxygen2`.

`print()` should **print out** the coefficients and coefficient names, similar as done by the `lm` class.

```
data(iris)
mod_object <- lm(Petal.Length~Species, data = iris)
print(mod_object)


Call:
lm(formula = Petal.Length ~ Species, data = iris)

Coefficients:
      (Intercept)  Speciesversicolor   Speciesvirginica
             1.46               2.80               4.09
```

`predict()` should return the predicted values $\hat{\mathbf{y}}$, it should be able to predict for new dataset similar to the `predict()` function for the `lm()` package.

`coef()` should return the ridge regression coefficients $\hat{\beta}^{\text{ridge}}$

1.1.4 Write a test suite

Write a simple test suite for your `ridgereg()` that test that you get similar coefficients as `lm.ridge()` in package `MASS`.

1.1.5 Handling large datasets with `dplyr`

Create a function you call `visualize_airport_delays()` without any arguments that creates a plot that visualizes the mean delay of flights for different airports by longitude and latitude using `ggplot2`. The datasets can be found in the `nycflights13` package.

The data handling should be done using `dplyr` verbs as much as possible. See the cheat sheet **here** `https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf` for more information how to data munge using `dplyr`. Remember that `delays` is a variable in the `flights` dataset and airport information is in the `airports` dataset.

1.2 Create a vignettes for `ridgereg()`, `dplyr` and the `caret` package

Create a vignette called `ridgereg` where you show how to do a simple prediction problem using your own `ridgereg()` function.

Use the caret package and your `ridgereg()` function to create a predictive model for the `BostonHousing` data found in the `mlbench` package or (*) data from your own API. If you prefer you may use the tidymodels package instead of caret.

The vignette should include the following:

1. Divide the `BostonHousing` data (or your own API data) into a test and training dataset using the `caret` package.
2. Fit a linear regression model and a fit a linear regression model with forward selection of covariates on the training dataset.
   (a) Hints:

i. You may do this by using any `R` package that is available on CRAN, but you may wish to use the `caret` package; see **here** (`https://topepo.github.io/caret/train-models-by-tag.html#Linear_Regression`)

ii. Remember that when using the forward selection algorithm on a dataset containing `n` covariates, the algorithm **should** be able to select amongst models trained on 0 (only intercept model), 1 (single covariate model), ..., `n` (model that uses all covariates, i.e. no removal of covariates) covariates.

iii. If you do use `caret::train` for this task, make sure to check that the condition mentioned in hint (b) holds. You could check this, for example, by calling the `summary` function on output of the `caret::train` command; if this condition does not hold, check if tuning any parameter(s) of `caret::train` helps.

iv. See **here** (`https://topepo.github.io/caret/model-training-and-tuning.html#grids`) for help on how to tune `caret::train` parameters.

3. Evaluate the performance of this model on the training dataset.

4. Fit a ridge regression model using your `ridgereg()` function to the training dataset for different values of λ. How to include custom models in caret is described **here** `https://topepo.github.io/caret/using-your-own-model-in-train.html` .

5. Find the best hyperparameter value for λ using 10-fold cross-validation on the training set. More information how to use the `caret` package for training can be found **here** `https://cran.r-project.org/web/packages/caret/vignettes/caret.html` and **here** `https://topepo.github.io/caret/model-training-and-tuning.html`.

6. Evaluate the performance of all three models on the test dataset and write some concluding comments.

1.2.1 (*) Predictive modeling of flight delays using `ridgereg()`

Create a new vignette called flight_delay where you try to predict the delay of each flight using your own `ridgereg()` function. If the data is too large, you can scale it down a bit, but the purpose is to try to do predictions using larger datasets.

1. Read in the weather dataset and the flights dataset from the `nycflights13` package and remove eventual variables you do not believe to have a predictive value.

2. Add extra weather data from the weather dataset and create interaction effects you think could be of interest for the prediction.

3. Use the caret package to divide the flight dataset into three sets: test, train and validation (with the proportions 5%, 80% and 15%.

4. Train ridge regressions models for different values of λ and evaluate the root mean squared error (see **here** `https://heuristically.wordpress.com/2013/07/12/calculate-rmse-and-mae-in-r-and-sas/`) on the validation set. Try to find an optimal value for λ.

5. When you found a good value for λ, use this to predict the test set and report the RMSE of your predicted model.

1.3 Seminar and examination

During the seminar you will bring your own computer and demonstrate your package and what you found difficult in the project.

We will present as many packages as possible during the seminar and you should

1. Show that the package can be built using R Studio and that all unit tests is passing.

2. Show your vignette/analysis.

1.3.1 Examination

Turn in a the adress to your github or gitlab repo with the package using LISAM. To pass the lab you need to:

1. Have the R package up on GitHub with a Travis CI, GitHub Actions, or GitLab CI pass/fail badge.
2. The test suites for the implemented function(s) should be included in the package.
3. The package should build without warnings (pass) on Travis CI, GitHub Actions, or GitLab CI.
4. All issues raised by Travis CI / GitHub Actions / GitLab CI should be taken care or justified why they are not a problem or cannot be corrected. Be careful with namespace issues, these you HAVE to take care of.

Bibliography

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.