Project 2 - Work in Pairs

Stat 159, Fall 2016, Prof. Sanchez

Predictive Modeling Process

The second project in Stat 159 involves working collaboratively in teams of two members. This means that you have to pair up with another student and decide which one will be the *owner* of the repository, and which will be the *contributor*.

As stated in the syllabus, you will not be allowed to do the project individually (you must collaborate). Due to the size of the class, it is possible that there is a team with three members. Likewise, if you fail to "submit" your public github repository you will fail the course.

This project is largely based on chapter 6: *Linear Model Selection and Regularization* (from "An Introduction to Statistical Learning" by James et al). Read this chapter, especially sections 6.2, 6.3, 6.6, and 6.7. You may also need to read chapter 5: *Resampling Methods* to learn about cross-validation. The idea of this project is to perform a predictive modeling process applied on the data set *Credit*. The data set is described in page 83, and available as a CSV file at:

http://www-bcf.usc.edu/~gareth/ISL/Credit.csv

You (i.e. the team) will have to build various models to predict the variable `Balance` in terms of ten predictors such as `Income`, `Age`, `Education`, `Gender`, `Ethnicity`, etc.

Exploratory Data Analysis (EDA)

The first step in any model building process is to understand the data. This means that you have to obtain descriptive statistics and summaries of all variables.

For the quantitative variables you should compute:

- Minimum, Maximum, Range
- Median, First and Third quartiles, and interquartile range (IQR)
- Mean and Standard Deviation
- Histograms and boxplots

For the qualitative (categorical) variables you should compute a table of frequencies with both the frequency (i.e. counts) and the relative frequency (i.e. percent or proportions). Likewise, create barcharts of such frequencies (you can choose plotting either the counts or the proportions).

Because we are interested in studying the association between `Balance` and the rest of predictors, you will also obtain:

- matrix of correlations among all quantitative variables.
- scatterplot matrix.
- anova's between `Balance` and all the qualitative variables (see function `aov()`).

- conditional boxplots between `Balance` and the qualitative variables, that is, boxplots of `Balance` conditioned to each of `Gender`, `Ethnicity`, `Student`, and `Married`.

You can add other descriptive statistics and charts that you consider convenient for the EDA phase. Use one or more text files to `sink()` the descriptive statistics; as for the charts, use image files (e.g. png, pdf) to save each plot.

Pre-modeling Data Processing

There are two major processing steps that you need to perform before you can fit any model:

- convert factors into dummy variables
- mean centering and standardization

a) Dummy out categorical variables

The first processing step involves transforming each categorical variable (`Gender`, `Student`, `Married`, and `Ethnicity`) into dummy variables. The main reason to do this is because the function `glmnet()` (used in ridge and lasso regressions) will not work if the input data contains factors.

In order to generate a new data frame in which factors are "dummified", you can use the function `model.matrix()`. Assuming that the `Credit.csv` data is in a data frame `credit`, you can use `model.matrix()` as follows:

```
# dummy out categorical variables
temp_credit <- model.matrix(Balance ~ ., data = credit)
```

`model.matrix()` will expand factors to a set of dummy variables (i.e. binary indicators). For a given factor, the number of binary indicators will be one less than the number of levels. For example, `Gender` has two levels: `Male` and `Female`; thus just one dummy indicator will be generated for `Gender`. Likewise, because `Ethnicity` has three levels (`African American`, `Asian`, and `Caucasian`), only two binary indicators will be generated for this variable.

Notice that the returned `temp_credit` will contain a vector of ones in the first column. This corresponds to an *intercept* term that won't be necessary. To assemble the full data, you need to `cbind()` the response variable `Balance`:

```
# removing column of ones, and appending Balance
new_credit <- cbind(temp_credit[ ,-1], Balance = Balance)
```

b) Mean Centering and Standardizing

The second processing step involves mean-centering and standardizing all the variables. This means that each variable will have mean zero, and standard deviation one. One reason to standardize variables is to have comparable scales. When you perform a regression analysis, the value of the computed coefficients will depend on the measurement scale of the associated predictors. A β

coefficient will be different if the variable is measured in grams or in kilos. To avoid favoring a certain coefficient, it is recommended to mean-center and standardize the variables. Perhaps the easiest way to do this is using the function `scale()`.

```
# scaling and centering
scaled_credit <- scale(new_credit, center = TRUE, scale = TRUE)

# export scaled data
write.csv(scaled_credit, file = "path/of/processed/data/scaled-credit.csv")
```

Once you obtain scaled data, save it as a csv file. This will be the actual data you will use for the model building process.

Training and Testing Sets

After first understanding the data, the next step is to build and evaluate a model on the data. A standard approach is to take a random sample of the data for model building and use the rest to understand the model performance. The common terminology is to get a **training** (*aka* learning) set to build the model, and use a **test** set to assess the performance.

The *Credit* data set has 400 observations. In order to define the train set, you have to generate a random sample (with no replacement) of size 300. You can use the `sample()` function and an integer vector `1:400` to sample 300 values; this sample will be the *train set*. The rest of 100 observations not included in the train set will form the *test set*. For reproducibility purposes, you should set a random seed before taking the sample—see the function `set.seed()`. Store the train and test vectors in a file. You will use these vectors, together with the scaled data, everytime you build a model.

Regression Models

Fit a multiple linear regression model via Ordinary Least Squares (OLS)—using the `lm()` function. This model will serve as a benchmark to which you will compare the other models. In addition to fitting a model with OLS, we are going to consider other regression alternatives:

- Ridge regression (RR)
- Lasso regression (LR)
- Principal Components regression (PCR)
- Partial Least Squares regression (PLSR)

Ridge and Lasso regressions are shrinkage methods. PCR and PLSR are dimension reduction methods (read Chapter 6 for more information about these approaches). Functions to fit models with ridge and lasso regressions are available in the package `"glmnet"`. Functions to fit models with PCR and PLSR are available in the package `"pls"`. Examples with code on how to use these functions are also available on Chapter 6 (see sections 6.6 and 6.7).

A Git branch for each method. To carry out the model building process under each of the four approaches, you must create a new branch in your project. For instance, to start working with the

ridge regression method, you can create a branch `"ridge"`. Likewise, to work on the lasso method, you can create a branch `"lasso"`. Each team member will work on one shrinkage method and one dimension reduction method. This means that each team member will create two branches.

Model Building Process

For each regression method (except OLS), you will have to follow these steps:

1. Run the corresponding fitting function on the **train set** using ten-fold cross-validation. Because cross-validation works by resampling data, you have to set a random seed before running the fitting function (for reproducibility purposes).
    - `cv.glmnet()` performs 10-fold cross-validation by default. Also, it computes an intercept term and standardizes the variables by default. Because we already mean-centered and standardized all the variables, use the arguments `intercept = FALSE`, and `standardize = FALSE`. Use a grid for the argument `lambda` of: `grid <- 10^seq(10, -2, length = 100)`.
    - with functions `pcr()` and `plsr()` use the argument `validation = "CV"` to perform 10-fold cross-validation.
2. The output from the fitting function will give you a list of models (from which you will select the "best" model); `save()` this output in a `.RData` file.
3. To select the best model:
    - in RR and LR, look for `$lambda.min` from the output of `cv.glmnet()`
    - in PCR and PLSR, look for the minimum `$validation$PRESS` from the outputs of `pcr()` and `plsr()`.
4. Plot the cross-validation errors in terms of the *tunning* parameter to visualize which parameter gives the "best" model:
    - for RR and LR the tunning parameter is λ. You can use the function `plot()` on the outputs of `cv.glmnet()`.
    - for PCR and PLSR the tunning parameter is the number of components. You can use the function `validationplot()`, with the argument `val.type = "MSEP"`, on the outputs of `pcr()` and `plsr()`.
5. Once you identify the "best" model, use the **test set** to compute the test Mean Square Error (test MSE). You'll use this value in the Rmd file to compare the performance of all the models.
6. Last but not least, refit the model on the **full data set** using the parameter chosen by cross-validation. This fit will give you the "official" coefficient estimates (to be used in Rmd file).

As mentioned above, save all the outputs from the model building stage. For example, say you are working on the branch `"lasso"`. You can `save()` the output of `cv.glmnet()`, the value of λ for the "best" model, the test MSE, and the "official" coefficients of the model on the full data set using the parameter chosen by cross-validation. Because `save()` produces binary files, you could `sink()` as well some of the primary results (e.g. λ for the "best" model, "official" coefficients, test MSE) to a text file. Likewise, export the plot with cross-validation errors to an image file.

Once you finish the model building stage for a given regression method, you can merge the branch with the "master" branch.

Keep in mind that you are using the mean centered and standardized variables (both the response and the predictors). Because of this, you won't get similar results to those of the book. If you are curious about being able to obtain simmilar results like in the book, then follow the code on sections 6.6 and 6.7 (this is not mandatory, and is not part of this project).

Report

One of the primary deliverables in the project will be a report in PDF format. You can generate an HTML document as additional material, but we will only consider the pdf version for grading purposes.

To prepare the report, you will break down its content into separate section files (`.Rmd`). One of the main reasons to split up the report into several files is collaboration. In this way, it's less likely to have conflicts. Both members can be editing the report content at the same time by working on different files: e.g. one member may edit the `01-introduction.Rmd` file while the other member may edit the `04-analysis.Rmd` file without stepping on each others toes.

```
report/
  report.pdf
  report.Rmd
  sections/
    00-abstract.Rmd
    01-introduction.Rmd
    02-data.Rmd
    03-methods.Rmd
    04-analysis.Rmd
    05-results.Rmd
    06-conclusions.Rmd
```

You can start writing the sections files before you finish the regression analyses. But you must finish the model building stage for each regression method in order to have the outputs necessary to complete the report.

- Include a table of regression coefficients for all methods. This table should have as many rows as predictors, and five columns (one column per regression methods: ols, ridge, lasso, pcr, and plsr).
- Include another table with the test MSE values for the regression techniques: ridge, lasso, pcr, and plsr.
- Likewise, include a plot in which the official coefficients are compared. A good option is to use `ggplot()` to create a bar chart with facetting (e.g. `facet_wrap()`) of such coefficients. If you choose this approach, the facets should be made in terms of the regression methods (one facet for each method). Another suggestion is to plot trend lines (i.e. the profiles) of the coefficients (one line connecting the coefficients of each method).

Slides

To give the project an extra feature, you must also prepare some slides. RStudio provides different Rmd alternatives to create slides. On one hand you can make HTML presentations with ioslides, Slidy, and reveal.js. On the other hand you can make PDF presentations via Beamer. All of these options have their pros and cons. But in terms of reproducibility, they are definitely better than any of the typical slideware like Power Point, Keynote, or Google Slides.

Use an Rmd file to make your presentation with **ioslides**. More information on the settings and options to create these slides can be found here:

http://rmarkdown.rstudio.com/ioslides_presentation_format.html

Think about your slides as companion material for this project: something that you can show to an audience when giving a talk (instead of the report). What to include in the presentation? Include slides about the data, the prediction goal, the list of applied methods, the most relevant findings, with some plots and tables, and main conclusions. Pretend that you only have between 10 and 15 minutes to give a talk with your slides (this should help you decide about the number of slides and their content).

File Structure

The file-structure for the project is as follows:

```
stat159-fall2016-project2/
   README.md
   Makefile
   LICENSE
   session-info.txt
   .gitignore
   code/
      functions/
         ...
      scripts/
         ...
      tests/
         ...
   data/
      Credit.csv
      ...
   images/
      ...
   report/
      report.pdf
      report.Rmd
      sections/
         00-abstract.Rmd
         01-introduction.Rmd
         02-data.Rmd
```

```
        03-methods.Rmd
        04-analysis.Rmd
        05-results.Rmd
        06-conclusions.Rmd
    slides/
        ...
```

Note that the file tree diagram doesn't show all the files. You should make some decisions on what type of files, and how many files, will be included.

Makefile targets

Your `Makefile` should have the following Phony targets:

- `all`: will be associated to phony targets `eda`, `regressions`, and `report`
- `data`: will download the file `Credit.csv` to the folder `data/`
- `tests`: will run the unit tests of your functions
- `eda`: will perform the exploratory data analysis
- `ols`: OLS regression
- `ridge`: Rdige Regression
- `lasso`: Lasso Regression
- `pcr`: Principal Components Regression
- `plsr`: Partial Least Squares Regression
- `regressions`: all five types of regressions
- `report`: will generate `report.pdf` (or `report.html`)
- `slides`: will generate `slides.html`
- `session`: will generate `session-info.txt`
- `clean`: will delete the generated report (pdf and/or html)

The rule for the target `regressions` should be written like this:

```
regressions:
    make ols
    make ridge
    make lasso
    make pcr
    make plsr
```

Based on the phony targets, you can add as many other targets as you consider convenient.

Your own decisions

You have total freedom and complete responsibility to make decisions on the following aspects:

- Functions:

  - naming style
  - number of functions
  - number of files to organize the functions

- Unit tests of functions:
  - contexts
  - test groups
  - expectations
  - number of files to organize the tests

- Script files:
  - naming style
  - number of files to organize the scripts

- Data files:
  - naming style
  - type of files (e.g. `.RData`, `.txt`, etc).
  - number of files to organize data-related files

- Image files:
  - naming style
  - type of files (e.g. `png`, `pdf`, `jpeg`, etc).
  - number of files to organize image files

- README files:
  - number of secondary readme files (in `data/`, `code/`, `images/`, `report/`)

Guidelines

Be consistent with the style system you decide to use. For instance, you can choose to use hyphens in the names of files: e.g. `raw-data.csv`, `eda-script.R`. Or you can choose to use underscores: e.g. `raw_data.csv`, `eda_script.R`. But you should avoid mixing hyphens and underscores: `raw_data.csv` and `eda-script.R`.

Functions and code: The same idea applies to function names: chosse a consistent naming style.

- choose descriptive function names, and arguments.
- avoid writing long functions (try to not exceed more than 10 lines of code).
- add descriptions for what a function does, what is the expected input(s), and what is the returned output.
- write unit tests for the functions that you create.

Commits: Come up with a consistent styleguide for writing commit messages. For example, commits that have to do with fixing bugs may be labeled `"bugfix"`:

```
git commit -m "bugfix: add closing '}' in function get_coefficients()"
```

Likewise, all commits that have to do with the source `.Rmd` files of the report could be labeled with `"report"`:

```
git commit -m "report: add chunk for table of coefficients in 05-results.Rmd"
```

Report and slides:

- all the tables and figures in the report must include captions.
- do not hard code values that depend on the data and the analysis output; use instead code chunks or **inline code**.

Credits: Give credit where credit is due. It is possible that you may end up using ideas, code, and other resources from other authors, friends, peers, etc. If that's the case, please acknowledge the sources. You can add a note in your readme file(s), link back to the consulted references, or simply credit the original creators.

Grading

Here are the main requirements that we will grade about your project:

- Public Github Repository (of each team member)
- Git branches for each regression method (after merging, do not delete branches!)
- Main `README` file must have:
  - title of project
  - description of the project's structure (primary directories and main files)
  - instructions for other users about how to reproduce your project
  - list of Make commands for phony targets
  - information about licenses: one creative commons license for media-content https://creativecommons.org/choose/, one open-source license for code http://choosealicense.com/
  - authors names (of both members)
- Use random seeds (`set.seed()`) for generating train and tests sets, and when calling functions that involve cross-validations. You should be able to reproduce your own analysis.
- Report in PDF format
- Slides
- Your `Makefile` should include:
  - declaration of variables
  - use of Make automatic variables
  - comments for rules, targets or dependencies that need further description
  - all required phony targets
- Session Information in `session-info.txt`:

  - R's session information `devtools::session_info()`, with versions of all the used R packages
  - git version
  - latex version
  - pandoc version
  - version of other software tools you use

In addition to not meeting the previous requirements, points will be deducted for:

- having inconsistent styleguides
- writing bad (e.g. uninformative, dummy) commit messages
- use of absolute filepath names (this breaks reproducibility)
- hard coding values in the `Rmd` files

You can discuss the various tasks of the project with other teams, but you must write your own code and narratives.