PS132B Problem Set 5

Due 12:29PM Thursday March 9, 2023

Please submit this assignment by uploading your R Markdown code file (`.Rmd`) AND either an html (`.html`) or pdf (`.pdf`) output onto bCourses before the due time, with easy-to-recognize file names (e.g., `pset5_KirkBansak.Rmd`). Your homework will be graded based on completeness, accuracy, and readability of both code and written answers.

Unless a package is explicitly mentioned in the problem set, you should not use any `R` packages beyond the following packages that are automatically loaded by default when you open up `RStudio`: `base`, `datasets`, `grDevices`, `graphics`, `methods`, `stats`, `utils`. In addition, you may also use `ggplot2`.

The point allocation is given by:

Q1.1	Q1.2	Q1.3	Q1.4	Bonus	Q1.5	Q1.6	Q1.7	Q1.8	Q1.9
5	5	5	5	2	5	5	15	5	5

Q2.1	Q2.2	Q2.3	Q2.4	Q2.5	Q2.6	Bonus	Q2.7	Q2.8	**Total (Bonus)**
1	8	6	5	3	5	2	7	10	**100 (4)**

Introduction

In this problem set we will use logistic regression and LASSO to perform supervised learning on text documents. In *The Impression of Influence*, Grimmer, Westwood, and Messing analyze the rate at which members of Congress claim credit for government spending in their press releases. Members of Congress issue a lot of press releases: from 2005 to 2010, House members issued nearly *170,000* press releases.

The object `CreditClaim.RData` contains data for 797 such documents, taken from *The Impression of Influence*. The format of the object when you load it is a list called `credit_claim`. The first element of this list (`x`) is a document term matrix, which contains the counts (i.e. number of occurrences) for each word in each document. That is, the columns of `x` correspond to word counts that will be used as predictors. The second element of `credit_claim` (`y`) is an indicator for whether or not the document (press release) is one in which the Member of Congress is claiming credit for government spending. The `y` vector has been hand-coded by experts who carefully read each press release, so we know it is an accurate reflection of whether or not the press release is indeed credit-claiming.

This problem set will involve building classification models that predict whether documents are credit-claiming or not as a function of their word counts. In a real-life use case, the ability to build such a classification model can be extremely valuable: it is extremely time-consuming to hand-code document labels (e.g. credit-claiming or not), so it is standard for practitioners to hand-code some number of documents, then build a classifier, and then use the classifier to label the remaining documents. In doing so, it is vital to (a) build the most accurate classification model possible and (b) estimate the out-of-sample performance of one's model to assess how accurate it is likely to be in classifying all the remaining documents.

Q1: Limited Logistic Regression and LOOCV

1) To begin this question, run the following code, which will store the document term matrix (predictor variables) and the vector of labels (outcome variable) separately

```
load("CreditClaim.RData")
x <- credit_claim$x
y <- credit_claim$y
```

What proportion of the documents are claiming credit?

2) Note the number of observations vs. the number of predictors. What would happen if you tried to use this dataset to fit a logistic regression with all of the predictors?

3) Identify the twenty words that are the most prevalent (occur most often on average) across the documents. Print those twenty words and comment briefly: what do you notice about the words? (Hint: the `colMeans()` function may be useful for this.)

4) Using the full set of observations, fit a logistic regression that predicts the credit-claiming label (y) with the *20* most common words (only the columns of x corresponding to those 20 words).

Bonus) You will notice that one of the variables has been dropped from the regression (i.e. you will see NA instead of a coefficient value). Why do you think this is the case?

5) Using the predicted probabilities from the logistic regression and applying a threshold of 0.5, classify each document in the data as credit claiming or not.

6) Compute the in-sample classification error.

Note) Now we are going to compare the in-sample fit to an estimate of out-of-sample fit. To do this, instead of performing a single training-test split, we're going to use *leave one out cross validation* (LOOCV). You may find it easier to implement this process by column-binding y and the limited x matrix (just the 20 variables) into a single matrix and then converting it into a data.frame, if you have not already done so.

7) For each of the documents, perform the following procedure, again using a logistic regression with the 20 most common words as the predictors:

   a) For document i (each observation or row of the data), fit the logistic regression to all documents except for i (we leave this document out of the model).
   b) Make a prediction for document i using the logistic regression you just fit, specifically classifying it as credit claiming or not, employing a 0.5 threshold.

8) Compute the out-of-sample error based on this cross-validation procedure.

9) How does the classification error from the in-sample fit (Part 6) compare to the classification error for this estimated out-of-sample fit?

Q2: LASSO and k-Fold Cross-Validation

For this question, you can use the `glmnet` package.

1) First, separate the data into training and test sets using the following code:

```
load("CreditClaim.RData")
x <- credit_claim$x
y <- credit_claim$y
n.total <- length(y)
prop.train <- 0.7
set.seed(54321)
r <- sample(1:n.total,round(prop.train*n.total), replace = FALSE)
x.train <- x[r,]
x.test <- x[-r,]
y.train <- y[r]
y.test <- y[-r]
```

2) Now run the following chunk of code:

```
set.seed(123)
cv.results <- cv.glmnet(x = x.train, y = y.train,
                        family = "binomial", nfolds = 5, alpha = 1)
```

   Explain what this code chunk is doing. That is, explain what the function `cv.glmnet` is performing, what data are being used, what the response variable is, and how many predictors are being employed. Further explain what the `family = "binomial"`, `nfolds = 5`, and `alpha = 1` arguments are all doing.

3) Determine how many λ values were tested. Given the number of folds and number of λ values used, how many separate LASSO models did the `cv.glmnet` command above fit? In addition, what loss function is being employed to compute CV error by default in this case?

4) Without using the short-cut command `plot(cv.results)`, plot the cross-validation error on the y-axis against the $log(\lambda)$ value on the x-axis.

5) What is the optimal λ value according to this cross-validation procedure (i.e. the value resulting in the lowest CV error)?

6) Extract the optimal λ value and store it. Now, using the optimal λ value, use the `glmnet` function to fit a single LASSO model on the training data. How many coefficients remain in this model (i.e. how many coefficients were not shrunk to zero)? Hint: Use the `?predict.glmnet` command to view the documentation on the various tasks the `predict` function can perform with a `glmnet` object.

Bonus) Print the names of the coefficients that remain in the model.

7) Use the LASSO model you fit in part (6) to make predictions for the test data. Apply this LASSO model to the test data to compute predicted probabilities for all observations in the test data. Convert these predicted probabilities into classification labels using 0.5 as the threshold. Compute the test set classification error.

8) Apply the bootstrap to the LASSO in order to compute a 95% confidence interval on the predicted probability that the *first row of the test data* is credit-claiming. Only the *training data* should be used in training the bootstrapped LASSO models, and hold λ fixed at the `lambda.1se` value for this procedure (i.e. **not** the `lambda.min` value that you used above).

   Note: This process is a bit computationally intensive, so you can keep the number of bootstrap iterations to a few hundred. (When testing out and debugging the function on your own, I recommend further limiting the number of iterations.) If you were implementing this for a real project, however, you would want to increase the number of iterations to the thousands.