HW4: Python problem set and project description

Kim Gannon

2020-11-20

```
library(bis557)
```

This vignette contains the answers from BIS 557 HW4.

I begin by setting up a new conda environment:

```
library(reticulate)
use_condaenv("r-reticulate")
```

Problem 1

In Python, implement a numerically-stable ridge regression that takes into account colinear (or nearly colinear) regression variables. Show that it works by comparint it to the output of your R implementation.

Like the function I wrote in R for Assignment 2, the function below reads in X and Y data arrays, and minimizes MSE with a given penalty lambda. The code is saved in the python/Scripts directory of my project (python_ridge.py). The main difference I noted when building this function was the SVD function within numpy versus that in R - the one in numpy gives an argument "vh" that is the transpose of the one given in R as v.

We compare the output of this function to that of my R-based ridge regression:

```
import seaborn as sns

iris = sns.load_dataset("iris")
X_names = ["sepal_width", "petal_length", "petal_width"]
Y_name = ["sepal_length"]
```

Now, I'll use reticulate to call this function from R.

```
reticulate::source_python("../python/Scripts/python_ridge.py")

betas <- python_ridge(Y_name, X_names, iris, 0.02)
print(betas)
#> [[1]]
#> [1] "constant"     "sepal_width"  "petal_length" "petal_width"
#>
#> [[2]]
#> [1]  1.8370947  0.6558188  0.7104345 -0.5577064
```

Finally, I'll compare the results to the ridge function I wrote in R earlier this semester:

```
data("iris")
irisform <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
```

```
bis557::my_ridge(irisform, iris, 0.02)
#> $coefficients
#>              [,1]
#> [1,]  1.8370947
#> [2,]  0.6558188
#> [3,]  0.7104345
#> [4,] -0.5577064
#>
#> $form
#> Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
#>
#> attr(,"class")
#> [1] "my_ridge"
```

Problem 2

Create an "out of core" implementation of the linear model that reads in contiguous rows of a data frame from a file„updates the model. You may read the data from R and send it in to Python.

As suggested, I'll first read the data in from R using the iread.table iterator.

To begin, I first write Python function that will calculate the beta hat from a given X and Y data chunk - see the function lm_calc.py.

Next, I write a function in R (lm_ooc.R) that grabs a data chunk, calls Python to estimate $\hat{\beta}$, stores the results in a vector, and repeats for every data chunk. Once $\hat{\beta}$ is estimated for every chunk, the function will return our result: an average of the individual $\hat{\beta}$s.

Now, I will demonstrate its effectiveness with the iris data set.

```
library(reticulate)
reticulate::use_condaenv("r-reticulate")
reticulate::source_python('../python/Scripts/lm_calc.py')

data(iris)
write.table(x = iris, file = "iris.csv", sep = ",", row.names = FALSE)
irisform <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
beta_hats <- bis557::lm_ooc(irisform, "iris.csv", nrows=75)
print(beta_hats)
#> [1]  1.9183992  0.5354021  0.7145914 -0.4045466
```

We see below that these numbers are similar to the output of the lm function:

```
irisform <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
lm(irisform,iris)
#>
#> Call:
#> lm(formula = irisform, data = iris)
#>
#> Coefficients:
#>  (Intercept)   Sepal.Width  Petal.Length   Petal.Width
#>       1.8560        0.6508        0.7091       -0.5565
```

Problem 3

Implement your own LASSO regression function in Python. Show that the results are the same as the function implemented in the casl package.

As similarly constructed in Section 7.5 of the CASL textbook, below I have constructed a LASSO function by computing a linear elastic net with $\alpha = 1$. Page 188 of CASL tells us this will yield the LASSO estimator. I will optimize using coordinate descent. First, I create a soft threshold function, to be used in updating the betas. Then, I construct a beta updating algorithm per Equation 7.34. Then, I write the primary function of interest by iterating until the between-iteration difference between slope coefficients falls within a tolerance level.

First, I build and test the soft threshold function - see soft_threshold.py

Then, I write the beta update function using coordinate descent - see lenet_update.py.

Finally, I write a function that iterates iterate until we reach the same stopping condition as specified in CASL p. 190: where "if the maximum difference between slope coefficients in successive iterations is small, then we are close enough to the optimal value, [or] . . . if we have reached a maximum number of iterations." See lenet.py for the implementation.

To test this function, I will use the iris data set and compare output with the CASL function. Note that my function defaults to a maximum of 1000 iterations, but all other parameters have been configured to match the default of the casl_lenet function.

```
library(reticulate)
reticulate::use_condaenv("r-reticulate")
reticulate::source_python('../python/Scripts/soft_threshold.py')
reticulate::source_python('../python/Scripts/lenet_update.py')
reticulate::source_python('../python/Scripts/lenet.py')

data(iris)
irisform <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
mmx <- bis557::make_model_matrices(irisform, iris)
X <- mmx$X
Y <- mmx$Y
l <- 0.001
alph <- 1 #true for all LASSO

W <- 1/150
bta <- c(rep(0,ncol(X)))
beta_hat <- lenet(Y, X, l, alph, bta, W=W, tol=0.00001)
print(beta_hat)
#> [[1]]
#> [1] 0
#>
#> [[2]]
#> [1] 1.84484
#>
#> [[3]]
#> [1] 0
#>
#> [[4]]
#> [1] 0
```

Now I do the same with the CASL function. Since I could not succesfully install the casl function from R, I copied and pasted the relevant R functions (caslPutil_soft_thresh, casl_lenet_update, and casl_lenet) from

Github into my BIS557 package.

```
data(iris)
irisform <- Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width
mmx <- bis557::make_model_matrices(irisform, iris)
X <- mmx$X
Y <- mmx$Y
l <- 0.001
bta <- c(rep(0,ncol(X)))
bis557:::casl_lenet(X, Y, lambda=l, maxit=1000)
#>             [,1]
#> [1,]  1.8142254
#> [2,]  0.6638585
#> [3,]  0.7028158
#> [4,] -0.5358817
```

Problem 4

Propose a final project for this class.

Throughout the past three decades, machine learning methods, specifically regularization methods, have migrated into econometrics (Athey 2018, Wooldridge 2009). While regularization methods like ridge regression and LASSO were initially developed in prediction settings, economists have begun to apply methods into both prediction and causal inference settings (Athey 2017, Mullainathan 2017, Varian 2014, Ruiz et. al. 2019, Chernozhukov et. al. 2015). LASSO is the most popular regularization technique among economists, both because of its relative ease of coefficient interpretability (after applying post-LASSO methods) and its variable selection properties. However, economists have run into two primary problems with this transition: First, LASSO variable selection can, in practice, be highly dependent on the order of the variables within a formula. Since the coordinate descent algorithm used by most LASSO regression packages conducts updates sequentially, one of two variables that are highly correlated may be selected based on which one appears first. Second, LASSO was developed and purportedly tends to work best in sparse data sets - those where the number of observations where n is less than number of regressors p. Such data does not tend to exist in many economics settings, although there are data sets with large values of p (although less than n) where economists have applied LASSO for model specification (Belloni and Chernozhukov 2011).

This analysis will shed light on both questions. I intend to examine whether the variable selection and predictive qualities of LASSO varies with the number of available candidate regressors. Following the approach of Belloni and Chernozhukov (2011), I will employ a classic model used in the teaching of econometrics: a regression of $Y = log(earnings)$ on $X = education$ (Angrist et. al. 2006). I have obtained a data extract from the US Census which contains, along with the two variables of interest, nearly one hundred covariates, many of which are collinear. I will begin by running a LASSO regression on all regressors for a 10% sample of observations, noting the variables that were selected and obtaining a value of lambda that minimizes MSE through cross-validation. I will then randomly generate other variables to serve as regressors and determine whether the selected variables or MSE changes. Then, I identify some highly correlated covariates (e.g. features of the income distribution) and permute their order, noting how variable selection and out-of-sample MSE changes.