Economics 1660: Big Data Prof. Daniel Björkegren

PS 6: Regularization

PART A: (Source: HTF page 95)
The Ridge regression problem is:

$$\hat{\beta}^{ridge} = argmin \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

Consider the ridge regression problem above. Show that this problem is equivalent to the problem:

$$\hat{\beta}^c = argmin \left\{ \sum_{i=1}^{N} [y_i - \beta_0^c - \sum_{j=1}^{p} (x_{ij} - \bar{x}_j)\beta_j^c]^2 + \lambda \sum_{j=1}^{p} \beta_j^{c\,2} \right\}$$

Give the correspondence between $\hat{\beta}^c$ and the original β.

PART B: Prediction with Linear Models

For this assignment, we will use ridge regression to estimate our *days_until_funded* variable in the loans dataset. We will use the *loans_AB.csv* file.

Before creating a model, think about the variable you will be predicting, and consider which other variables within the dataset might be correlated with it.

1. Give an example of a variable in the dataset that you expect to be correlated with *days_until_funded*. Will there be a positive or negative correlation? Provide justification for you answer.

We will now use ridge regression to develop a predictive model for the *days_until_funded* variable.

The solution to the ridge regression problem is given by:

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

where $\widehat{\boldsymbol{\beta}}$ is a vector of your predicted coefficients, $\boldsymbol{X}$ is a matrix whose rows represent the values of the independent variables, and $\boldsymbol{y}$ is a vector of the corresponding dependent variable values (in our case, the values of *days_until_funded* for each loan).

Use Python to do the following, and answer the questions that follow.

- Read in the file *loans_PS6.csv* and randomly assign each loan to either your training data or testing data. (You can simply call the function random.randint(0,1) to randomly return either 0 or 1, and then add a loan to either training or testing data accordingly. To load the *random* library, write "import random" at the top of your file.)
- Use your training data to build a ridge regression model with at least 30 variables of your choosing (you'll see more meaningful results if you use many variables). You must include at least one variable generated from the loan's text description (a keyword dummy, the length of the description, etc.), one factor variable (such as a country dummy, a sector dummy, or a gender dummy), and one continuous variable (such as loan amount or repayment term). Begin by setting lambda to 0 in your model.
- We want to include a constant term, but we usually don't want to penalize it like the other variables (when we penalize the other variables, we want the constant term to absorb any slack). In Part A you showed that the standard ridge regression can be rewritten in terms of centered variables, where we subtract the mean $\bar{x}_j$ from each x_{ij}, and subtract the mean $\bar{y}$ from each y_i. Your code should center the data in this manner before running the regression.
- Fill out the code to compute $\widehat{\boldsymbol{\beta}}$ and print out this vector.
- Now use your model to calculate the value of *days_until_funded* that your model predicts for each loan in both the training set and the testing set.
- Using the predictions and actual values, calculate the mean squared errors for both the training data and testing data.

HINTS:

- Since computing a ridge regression model involves dealing with matrices, you might want to make use of Python's NumPy library. It is simple to use, and an explanation of the functions you might need can be found on the last page of this assignment.
- You will probably want to maintain 4 lists as you loop through the loans in the csv file to read them in: a training $\boldsymbol{X}$, a testing $\boldsymbol{X}$, a training $\boldsymbol{y}$, and a testing $\boldsymbol{y}$. The 'X's should be lists of lists where each sub-list has the loan's values of your independent variables. The 'y's should be lists of the loans' values of *days_until_funded*. Append to these lists as you read in the data, and then convert to NumPy matrices once all of the data has been read in.
- Although we only used binary variables in constructing our decision trees, you are free (and expected) to use continuous variables in this assignment.

2. Report the value of the $\hat{\beta}$ vector that was produced by running the ridge regression with $\lambda = 0$. Indicate which independent variable corresponds with which $\hat{\beta}_i$.

3. Comment on the sign (positive or negative) of each $\hat{\beta}_i$. Is each sign what you would expect given what types of loans might take longer to fund? Did the regression yield any interesting insights? *(Don't worry about statistical significance here.)*

4. When $\lambda = 0$, what method does your estimate correspond with?

5. Vary λ, and:
    a. Plot how each $\hat{\beta}_i$ changes as λ increases. *(If you run a large model you may choose to only depict a subset of coefficients for a cleaner picture)*
    b. Plot how the MSE in both the training and test data changes as λ increases
    Comment on the results you obtain.

6. Are your results invariant to the scale of your variables? Try changing the scales (you can multiply each variable by a different constant), and compare the results you obtain. Does changing the scales change the underlying meaning of the data?

BONUS: Ridge regression penalizes the squared magnitude of coefficients and as a result shrinks coefficients smoothly to zero. LASSO penalizes the absolute value of coefficients and often ends up zeroing out some coefficients, which in practice ends up being helpful for applied work. Repeat the above exercise using LASSO instead of ridge. The algorithm to compute LASSO estimates is more complicated than ridge; see HTF section 3.4.4 (Algorithm 3.2 and 3.2a). This is quite challenging.

PART C

Your friend runs a firm. After hearing about your experience using big data, she has asked to help her optimize the price of her product. She has provided historical data for you to analyze.

1. Thinking the problem through

Your friend's firm produces one type of product; the marginal cost of producing the good is *mc*, and there is no fixed cost. She can decide the price to charge for the product *p*. If she sets the price to *p*, *D(p)* individuals will purchase the product.

a. Write the expression representing the firm's profit as a function of *p*
b. Use calculus to find the optimal price in terms of the primitives *mc, p,* and *D(p)*
c. Rearrange the expression and solve for the optimal price in terms of the price elasticity of demand, $\varepsilon = \frac{p}{D(p)} \frac{dD(p)}{dp}$
d. Based on the expressions you've derived, how can you use *D'(p)* to determine whether the price should be increased or decreased?

2. Data

The file demand_monopoly.csv includes price and quantity observations for her firm which is the only firm providing this good at the moment.
Relevant variables are:
t: time period
p: price
s: market share (quantity)

a. Use regression to estimate the elasticity ε (hint: write a regression function of *log D(p)* on *log(p)* and interpret the coefficient on log price. You may use whichever software package you prefer, including python, R, or STATA.)

b. Interpret your results. Should your friend raise or lower prices? Explain. If your results are puzzling, try to understand why.

NUMPY INTRODUCTION

NumPy is a very popular Python library that is useful for working with multidimensional arrays and matrices. You can explore the Python docs for more information, but here are some key functions and features that will be helpful in this assignment:

To use, include the line “import numpy” at the top of your code

numpy.matrix(*X*)	returns a matrix object given a list of lists *X*
Example:	numpy.matrix([[1,2,3], [4,5,6]]) returns $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
numpy.identity(*n*)	returns an identity matrix of size *n x n*
numpy.dot(*X, Y*)	returns the product of numpy matrices *X* and *Y*
numpy.linalg.inv(*X*)	returns the inverse of numpy matrix *X*
X.transpose()	returns the transpose of numpy matrix *X*