- **Programmer: Shaun Pritchard**
- **Date: 10-10-2021**
- **Assignment: 1**
- **Prof: Michael DeGiorgio**

CAP 52625 COMPUTATIONAL FOUDNATIONS OF AI

Ridge Regression using Gradient Descent - Assignment 1

Note: I decided to use symbols to make it easier to view to the implementation of code in contrast to the mathmatical thoeroms learned in class.

Deliverables

- **Deliverable 1:** build graph of dataset N=9 features tuning parameter effect on inferred Ridge regression
- **Deliverable 2:** Illustrate the effect of the tuning parameter on the cross-validation error
- **Deliverable 3:** Indicate the value of λthat generated the smallest CV(5)error
- **Deliverable 4:** retrain the modelof N=400 observations and provide the estimates of the p=9 *best-fit* model parameters.
- **Deliverable 5** Provide all your source code that you wrote from scratch to perform all analyses(aside from plotting scripts, which you do not need to turn in) in this assignment, along with instructions on how to compile and run your code.
- **Deliverable 6** Implement the assignment using statistical or machine learning libraries in a language of your choice.Compare the results with those obtained above, and provide a discussion as to why you believe your results are different if you found them to be different.

> Note:

Deliverable 1-5 located here: https://colab.research.google.com/gist/shaungt1/83c9e75f7062e34897957859165f3a0d/spritchard_cap5625_programming_assignment-1_10182021.ipynb

Deliverable 6 is located at: https://colab.research.google.com/drive/1W0aaP4C2_QJo4NTQ-mrODOepyaWxnQa-?usp=sharing

Import Dataset

```
#Math libs
from math import sqrt
from scipy import stats
# Data Science libs
import numpy as np
import pandas as pd
# Graphics libs
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

```

```
# Mount Google Drive for data access
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

```
# Set up dataframe instance of dataset Credit_N400_p9.csv
df = pd.read_csv('/content/drive/MyDrive/Florida_Atlantic_University/Computati
# Set up datafeme for testing
# df = pd.read_csv('/content/Credit_N400_p9.csv')
```

```
# Check feature (row:col) shape of dataframe
df.shape
```

```
(400, 10)
```

```
# Build copy of dataset for Pre-proccessing
df1 = df.copy()
# Validate new dataframe
df1.head(3)
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	903
2	104.593	7075	514	4	71	11	Male	No	No	580

Preproccess Data

```
# Assign dummy variables to catigorical feature attributes
df1 = df1.replace({'Male': 0, 'Female':1, 'No': 0, 'Yes': 1})
```

```
# Validate new trianing dataframe with dummy variables
df1.head(3)
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	B
0	14.891	3606	283	2	34	11	0	0	1	333
1	106.025	6645	483	3	82	15	1	1	1	903
2	104.593	7075	514	4	71	11	0	0	0	580

```
# Separate independent n X 1 feature X  and convert to numpy array
X = np.array(df1.iloc[:,:-1], dtype='float64')
# Test print X feature data conversion results
print('Matrix shape:{X}\nValidate array:(row:col)'  .format(X = X.shape), '\n'
```

```
Matrix shape:(400, 9)
Validate array:(row:col)
 [[1.48910e+01 3.60600e+03 2.83000e+02 ... 0.00000e+00 0.00000e+00
  1.00000e+00]
 [1.06025e+02 6.64500e+03 4.83000e+02 ... 1.00000e+00 1.00000e+00
  1.00000e+00]
 [1.04593e+02 7.07500e+03 5.14000e+02 ... 0.00000e+00 0.00000e+00
  0.00000e+00]
 ...
 [5.78720e+01 4.17100e+03 3.21000e+02 ... 1.00000e+00 0.00000e+00
  1.00000e+00]
 [3.77280e+01 2.52500e+03 1.92000e+02 ... 0.00000e+00 0.00000e+00
  1.00000e+00]
 [1.87010e+01 5.52400e+03 4.15000e+02 ... 1.00000e+00 0.00000e+00
  0.00000e+00]]
```

```
# Seperate dependant n X 1 feature Y and reshape to (400 x 1) vector numpy arr
Y = np.array(df1.iloc[:,-1], dtype='float64').reshape([-1,1])
# Test print Y feature data conversion results
print('Dependant Feature:{Y}\n \nValidate array:(row:col)\n' .format(Y = Y.sha
for i in Y:
   print(i, end = ' ')
```

```
Dependant Feature:(400, 1)

Validate array:(row:col)

[333.] [903.] [580.] [964.] [331.] [1151.] [203.] [872.] [279.] [1350.] [1407.] [0.] [2
```

Center response variables and standarize features

- Convert dataframe objects to numpy arrays
- Center the response variable Y (subtracting the mean)
- Standardizing input features X to a Z score


```
# Center Y response variable(subtract the mean)
Y_p  = Y- np.mean(Y, axis=0)
```

```
# Validate Y response vairables - mean of y_
y_ = np.mean(Y, axis=0)
print('Mean of Y:', y_)
print('Matrix Shape:', Y_p.shape)
```

```
Mean of Y: [520.015]
Matrix Shape: (400, 1)
```

```
# Split centered (row:col) of Y feature
Y_row, Y_col = Y_p.shape
# validate feature split of Y
print('(Y_p) Row x Col:=',Y_row, Y_col)
```

```
(Y_p) Row x Col:= 400 1
```

```
# Standardized X feature n x 1 matrix  as X_p array and reshape
mean_X = np.mean(X, axis=0).reshape([1,-1])
# Center X
X_p = X - mean_X
# Apply standard deviation to new shape[1,-1]
std_X  = np.std(X_p, axis=0).reshape([1,-1])
# Caluate centered features divided by standard deviation
X_p = X_p / std_X
```

```
# Validate X_p feature
print('Matrix Shape:', X_p.shape, '\n', '\n', 'Mean of X:' , '\n', mean_X, '\n
```

```
Matrix Shape: (400, 9)

 Mean of X:
 [[4.5218885e+01 4.7356000e+03 3.5494000e+02 2.9575000e+00 5.5667500e+01
  1.3450000e+01 5.1750000e-01 1.0000000e-01 6.1250000e-01]]

 Standard deviation of X:
 [[3.52001903e+01 2.30531179e+03 1.54530616e+02 1.36955969e+00
```

```
 1.72282310e+01 3.12129781e+00 4.99693656e-01 3.00000000e-01
 4.87179382e-01]]
```

```
#Store and seperate (row:col) in variable for X_p training/test set
X_row , X_col = X.shape
print('(X_p) Row x Col:=', X_row, X_col)
```

```
(X_p) Row x Col:= 400 9
```

Assign local variables

```
# Local Variables

# Tuning Parms
λ  = np.array([1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4])

# Learning Rate
α = 1e-5 # learning & convergence rate

# K-folds
k = 5

#Iterations
q = 1000 # itterations

#log base of lambda
λ_log = np.log10(λ)

#Standerrdized X features
X_p = X_p

#Centered y features
Y_p = Y_p
```

Ridge Regression Batch Gradient Decent

- Implement(inferred) Batch Gradient Descent for Ridge Regression on standardizing feature X_p and centered feature Y_p numpy arrays. Where p is n x p matrix.
- α = the learning and convergence rate.
- λ = L2 regularization tuning parameters.
- q = max number of iterations (1000) as specified.
- βx = Ridge regression beta coefficients parms.

- MSE = is a storage list to contain the mean squared errors for each iteration of the Batch Gradient Descent algorithm.

Note: I decided to use symbols to make it easier to view to the implementation of code in contrast to

Ridge Regression Batch Gradient Decent Algorithm

5:06

$$\beta := \beta - 2\alpha[\lambda\beta - \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)]$$

```
def RidgeRegression_BGD(X, Y, α, λ ) :
    # Empty list to hold MSE errors
    MSE = []
    # Randomly initialize the parameter vector 𝛽 = [𝛽1, 𝛽2, … , 𝛽𝑝]
    𝛽x = np.random.uniform(-1,1,(X_col,1))
    # Instantiate loop to update parameter vectors
    for i in range(q) :
        # Store 𝛽x coefficients in temp variable
        𝛽x_temp = 𝛽x
        # Update 𝛽x parameter vector as 𝛽x ≔ 𝛽 − 2𝛼[𝜆𝛽 − 𝐗𝑇(𝐲 − 𝐗𝛽)]
        𝛽x = 𝛽x - 2*α*( λ *𝛽x -  np.dot( X.T, Y - np.dot(X,𝛽x) ) )
        # Calcualte vector direction of response variables
        ŷ = np.dot(X,𝛽x)
        # Instantiate temp var "MSE_temp"square-root of real Y variables
        # minus the calculated ŷ response
        MSE_temp = np.mean(np.square(Y - ŷ))
        # Append updated MSE_temp caluation to MSE list
        MSE.append(MSE_temp)
        # Caluate the divided absolute values of 𝛽x coefficients minus 𝛽x
        𝛽t = np.abs((𝛽x - 𝛽x_temp)/𝛽x_temp)
        # Calualte the max value of 𝛽t and store in 𝛽m
        𝛽m = np.max(𝛽t)
# Console log to test my code:
#------------- Feature Testing Output-----------------------------------
        if (𝛽m < α):
            print("Testing:\nBatch Gradient Descent(RR_BGD) breaks on: {i} ite
            break
        # Test for convergance error
    if (MSE[-1] < MSE[0]): #Check MSE is lower than the initial value
        pass
    else :
        print("Testing:\n Error not converging with lambda = {λ}param".format(
    # Output updated coefficients and MSE
    return 𝛽x, MSE
```

```
# Test tunning paremters inferred on RidgeRegression_BGD()
```

```
# Create emtpy list to store updated coefficients
β_lst = []
# Create counter for test ouput
count = 0
# Itterate through RidgeRegression_BGD oupt: βx, MSE
for i, l in enumerate(λ):
    # counter
    count += 1
    # print('Tuning parameters {} \n', lmbdas)
    print('Tuning parameter converged at = #{c}λ {} \n'.format(l, c=count))
    # run RidgeRegression_BGD
    βx, MSE = RidgeRegression_BGD(X_p, Y_p, α, l)
    # Append βx beta coefficients to empty list
    β_lst.append(βx)

```

```
Tuning parameter converged at = #1λ 0.01

Tuning parameter converged at = #2λ 0.1

Tuning parameter converged at = #3λ 1.0

Tuning parameter converged at = #4λ 10.0

Tuning parameter converged at = #5λ 100.0

Tuning parameter converged at = #6λ 1000.0

Testing:
Batch Gradient Descent(RR_BGD) breaks on: 418 iteration
Tuning parameter converged at = #7λ 10000.0

Testing:
Batch Gradient Descent(RR_BGD) breaks on: 51 iteration
```

Deliverable 1:

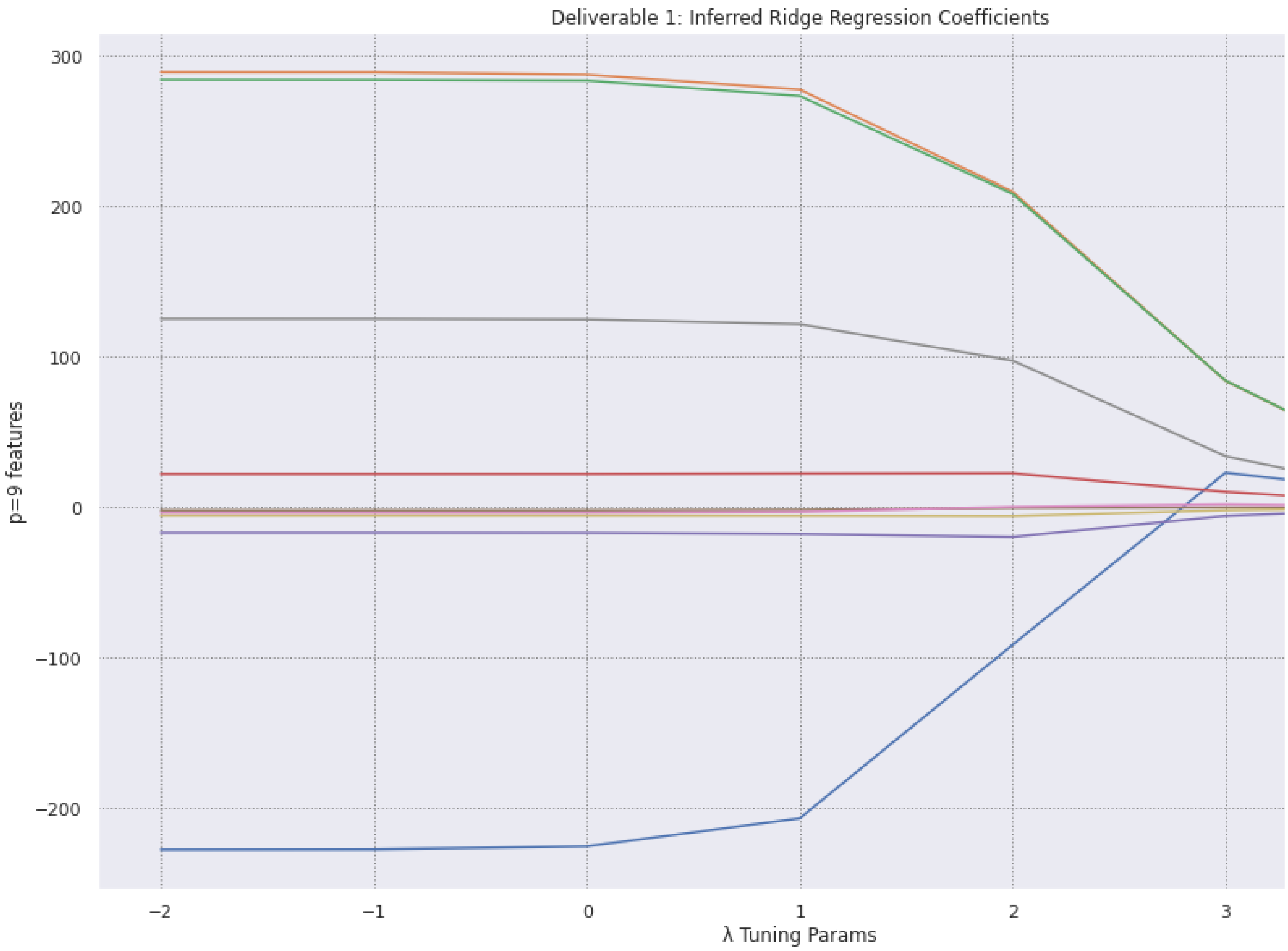
> Build graph of dataset N=9 features tuning parameter effect on inferred Ridge regression

```
# Output Deviverable 1: inferred tuning parmeters of ridge regresion
sns.set_theme()
sns.set_style("darkgrid", {"grid.color": ".5","image.cmap": "mako", "grid.line
plt.figure()
plt.figure(figsize=(16, 10), dpi=70)
plt.xlabel('λ Tuning Params')
plt.ylabel('p=9 features')
```

```
plt.title('Deliverable 1: Inferred Ridge Regression Coefficients')
for i in range(X_col) :
    βj = [ βx[i,0] for βx in β_lst  ]
    legend = 'Beta_λ_{}'.format(i)
    sns.lineplot(x=λ_log,  y=βj , label=legend, )
# Output Deliverable1.jpg to file
plt.savefig('SPritchard_CAP5625_Assignment1_Deliverable1.jpg')
plt.show()


```

```
<Figure size 432x288 with 0 Axes>
```



(5)K-Fold Grid Search Cross Validation Algorithm

- (5)K-fold Grid Cross Validation calualted Batch Gradient Decent Ridge Regression with hyperparameter tuning

- Use grid search CV to trian and test 5 k folds
- Calculate test and trainging errors
- Test MSE on tuning params

```
# Implement start and end of test k-fold data split
X_row = X.shape[0]
# Divide absolute row features by k = 5
X_row_test = X_row // k
# Store data in k-fold array (Kfold/Kfold_ )
Kfold = [ X_row_test * ind for ind in range(k)] # initial k-folds
Kfold_ = [ ind + X_row_test for ind in Kfold ]  # End k-folds
```

```
# Grid Cross Validation for Batch Gradient Decent Ridge Regression with hyperp
# Instantiate empty list to hold Cross vlaidations errors
CV = []
# Add a counter to iterate tuning params and errors
for i, l in enumerate(λ) :
    # print('(5)K-fold CV tuning parameter error = {}'.format(l))
    MSE = []
    # Loop through K trianing and test vectors
    for i in range(k):
        #Hold-out 5 k-folds arrays (80 x 9)
        CV_fold = Kfold[i]
        CV_fold_ = Kfold_[i]
        # Seperate training feature variables
        X_train = np.row_stack(( X[0:CV_fold,:] , X[CV_fold_:, :] ))
        Y_train = np.row_stack(( Y[0:CV_fold,:] , Y[CV_fold_:, :] ))
        # Seperate testing feature variables
        X_test   = X[CV_fold:CV_fold_, :]
        Y_test   = Y[CV_fold:CV_fold_, :]
        # Standardize X test set
        X_test_ = (X_test - np.mean(X_test, axis=0))/np.std(X_test, axis=0)
        # Center Y test set
        Y_test_ =  Y_test - np.mean(Y_test, axis=0)
        # Implement ridge regressionand MSE on test data
        𝛽x, _ = RidgeRegression_BGD(X_p, Y_p, α, l)
        # Product transofrmation of test data on trining set
        ŷ = np.dot(X_test_, 𝛽x)
        # Claulate average squareroot of Y(test)- ŷ variables
        err = np.mean(np.square(Y_test_ - ŷ))
        # Append calualtion to MSE list
        MSE.append(err)
    #Caluate average of updated MSE
    CV_err = np.mean(MSE)
```

```
    # Append averaged MSE variables to CV list
    CV.append(CV_err)
```

```
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 420 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 423 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 428 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 422 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 427 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 55 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 54 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 51 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 55 iteration
Testing:
Batch Gradient Descent(RR_BGD) breaks on: 49 iteration
```

```
# console log  to test CV code
#-------------- Feature Testing ----------------------------------
print('Initial (5)K-folds', Kfold, '\n')
print('Ending  (5)K-folds', Kfold_, '\n')
print(X_test.shape, Y_test.shape, X_train.shape, Y_train.shape, '\n')
print(X_test[0,0], Y_test[0,0], X_train[0,0], Y_train[0,0], '\n')
print("Mean Square Error",err, '\n')
print("MSE value",MSE, '\n')
print("CV_error", CV_err, '\n')
```

```
Initial (5)K-folds [0, 80, 160, 240, 320]

Ending  (5)K-folds [80, 160, 240, 320, 400]

(80, 9) (80, 1) (320, 9) (320, 1)

16.279 5.0 14.890999999999998 333.0

Mean Square Error 167309.80185008282

MSE value [167455.13745409623, 220062.24519126484, 192200.85842315012, 159943.406401254

CV_error 181394.28986396975
```

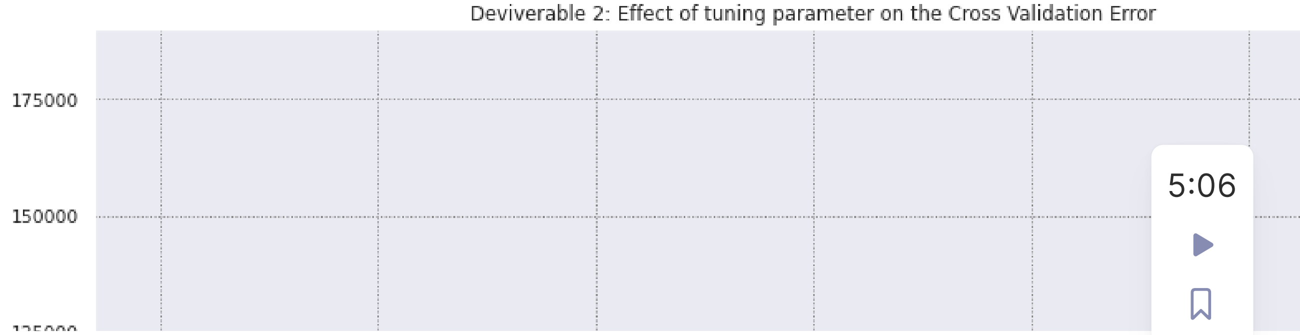
Deliverable 2

- Illustrate the effect of the tuning parameter on the cross validation error by generating a plot.
- Grid Cross Validation tuning errors for each tuning parameter value, perform five-fold cross validation and choose the value of λ that gives the smallest value.

$$\mathrm{CV}_{(5)} = \frac{1}{5}\sum_{i=1}^{5} \mathrm{MSE}_i$$

```
# Illustrate the effect of the tuning parameter on the cross validation error
sns.set_theme()
sns.set_style("darkgrid", {"grid.color": ".5","image.cmap": "mako", "grid.line
plt.figure()
plt.figure(figsize=(16, 10), dpi=70)
plt.title('Deviverable 2: Effect of tuning parameter on the Cross Validation E
plt.xlabel('λ Tuning Params')
plt.ylabel('CV Error')
sns.lineplot(x=λ_log, y=CV , color='purple', markersize=12)
plt.savefig('SPritchard_CAP5625_Assignment1_Deliverable2.jpg')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Deliverable 3

- Indicate the value of λ that generated the smallest CV(5) error

```
#Find minimum MSE error
err = min(MSE)
# index MSE error
i = MSE.index(err)
# Itereate to find MSe from λ list
l = λ[i]
# Output final results of lowest λ tuning param
print("Best CV error of λ = {e}\nBest tuning param of λ = {l}".format(e=err,
```

```
Best CV error of λ = 159943.40640125488
Best tuning param of λ = 10.0
```

Deliverable 4

- Given the optimal λ, retrain your model on the entire dataset of N = 400 observations and provide the estimates of the p = 9 best-fit model parameters.

```
# Retrain model based on λ = 10.0
βx, _dh = RidgeRegression_BGD(X_p, Y_p, α, l)
# Output best fit model params of  βx based on on λ = 10.0 tuning param
print('Best fit model parameters', '\n', βx)
```

```
Best fit model parameters
 [[-206.6352321 ]
 [ 277.44127207]
 [ 273.40973826]
 [  22.38621464]
 [ -17.61875517]
 [  -1.74246626]
 [  -3.0415899 ]
 [ 121.65556358]
 [  -5.61633766]]
```

5:06

0s completed at 1:43 PM