Assignment 2 Manual (From Scratch)

- **Programmers:**
    - Shaun Pritchard
    - Ismael A Lopez
- **Date:** 11-8-2021
- **Assignment:** 2
- **Prof:** M.DeGiorgio

Overview: Secondary - Assignment 2

We analyzed the credit card data from N=400 training observations that you examined in Programming Assignment 1 using a penalized (regularized) least squares fit of a linear model using elastic net, with model parameters obtained by coordinate descent.

Initially, we each worked independently, then we collaborated afterwards to finalize the assignment deliverables. This resulted in the completion of 2 methods for achieving the same goal, namely implementing ElastNet with coordinate descent. This is the second take on assignment 2. The aim was to display different methods of achieving the same abstraction.

Import data

```
#Math libs
from math import sqrt
from scipy import stats
import os
# Data Science libs
import numpy as np
import pandas as pd
# Graphics libs
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#Timers
!pip install pytictoc
from pytictoc import TicToc
```

```
    Requirement already satisfied: pytictoc in /usr/local/lib/python3.7/dist-pack
```

✓ 0s completed at 3:53 PM

```
# Validate data import
df.head(3)
```

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

Data Pre Proccessing

```
# Assign dummy variables to catigorical feature attributes
df = df.replace({'Male': 0, 'Female':1, 'No': 0, 'Yes': 1})
df.head(3)
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Balar
0	14.891	3606	283	2	34	11	0	0	1	ː
1	106.025	6645	483	3	82	15	1	1	1	(
2	104.593	7075	514	4	71	11	0	0	0	!

```
# separate the predictors from the response
X = df.to_numpy()[:, :-1]
Y = df.to_numpy()[:, -1]
print('Convert dataframe to numpy array:', X.shape, Y.shape)
```

```
Convert dataframe to numpy array: (400, 9) (400,)
```

Set Global Variables

```
# Set local variables
```

```

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

# Set verbose to True
verbose = True

# Set n x m matrix variable
X_p = X

# Set n vector variable
Y_p  = Y
```

Instantiate Data

```
# Randomize N x M and N data
def randomize_data(X_p, Y_p):
  matrix = np.concatenate((X_p, Y_p[:, None]), 1)
  np.random.shuffle(matrix)
  return matrix[:, :-1], matrix[:, -1]
```

```
# Initilize random sample data
x, y = randomize_data(X_p, Y_p)
```

```
# Set Global variable for samples and number of X = N X M features
X1 = x.shape[0]
X2 = x.shape[1]
```

```
# Create a β matrix to store the predictors
β = np.zeros([k, len(λ), len(α), X2])
```

-

```
# Standardize  X
def standardize(x, mean_x, std_x):
  return (x - mean_x) / std_x
```

```
# Center response variables
def centerResponses(y, mean):
  return y - mean
```

```
# predicit x
def predict(x):
  x = standardize(x, mean_x, std_x)
  return np.matmul(x, βx)
```

```
# Calculate MSE score
def score(x_test, y_test,  βx):
  ŷ = np.matmul(x_test, βx)
  mse = np.mean((y_test - ŷ) ** 2)
  return mse
```

Coordinate Descent Algortihm

```
# implement Coordinate Descent
def coordinateDescent(x, y, βx, sum_sq, lamb, alpha):
  for k in range(X2):
```

```
            x_test = x[i_test:i_test + test_x]
            x_train = np.delete(x, np.arange(i_test, i_test + test_x)
            y_test = y[i_test:i_test + test_x]
            y_train = np.delete(y, np.arange(i_test, i_test + test_x)


            # Standardize x and center y  5 K-fold trianing and test
            mean_x, std_x = np.mean(x_train, 0), np.std(x_train, 0)
            mean_res = np.mean(y_train)

            # X training and test
            x_train = standardize(x_train, mean_x, std_x)
            x_test = standardize(x_test, mean_x, std_x)

            # Y training and test
            y_train = centerResponses(y_train, mean_res)[:, None]
            y_test = centerResponses(y_test, mean_res)[:, None]

            # compute b_k given this fold
            sum_sq = np.sum(x_train ** 2, 0)


            # initialize random Beta for this lambda and fold
            βx = np.random.uniform(low = -1, high = 1, size = (X2, 1)
lambda:0.1; alpha:0.2; CV MSE:10107.311060939737
lambda:0.1; alpha:0.4; CV MSE:10106.527487675436
lambda:0.1; alpha:0.6; CV MSE:10105.892396306765
lambda:0.1; alpha:0.8; CV MSE:10105.393168314593
lambda:0.1; alpha:1; CV MSE:10105.003601773618
lambda:1.0; alpha:0; CV MSE:10108.282592895757
lambda:1.0; alpha:0.2; CV MSE:10104.379860088233
lambda:1.0; alpha:0.4; CV MSE:10106.733188516877
lambda:1.0; alpha:0.6; CV MSE:10111.187330508856
lambda:1.0; alpha:0.8; CV MSE:10116.592263579734
lambda:1.0; alpha:1; CV MSE:10122.579137367928
lambda:10.0; alpha:0; CV MSE:10108.289391793101
lambda:10.0; alpha:0.2; CV MSE:10159.856252431346
lambda:10.0; alpha:0.4; CV MSE:10269.140903521058
lambda:10.0; alpha:0.6; CV MSE:10427.910296865932
lambda:10.0; alpha:0.8; CV MSE:10632.115972887845
lambda:10.0; alpha:1; CV MSE:10876.415631027066
lambda:100.0; alpha:0; CV MSE:10108.20024592934
lambda:100.0; alpha:0.2; CV MSE:12574.807391369948
lambda:100.0; alpha:0.4; CV MSE:16986.66346897991
lambda:100.0; alpha:0.6; CV MSE:21675.239671371124
```

```
 [ 0.12942747]
 [ 0.02113179]
 [-0.00072954]
 [-0.00558078]
 [ 0.00299435]
 [ 0.04321191]
 [ 0.00227012]]
```

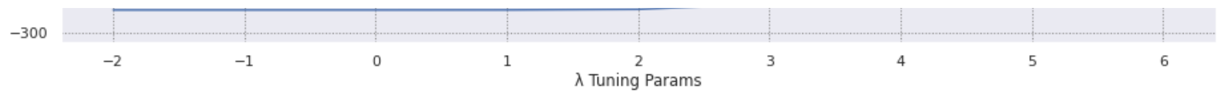
```
print(mse_score)
```

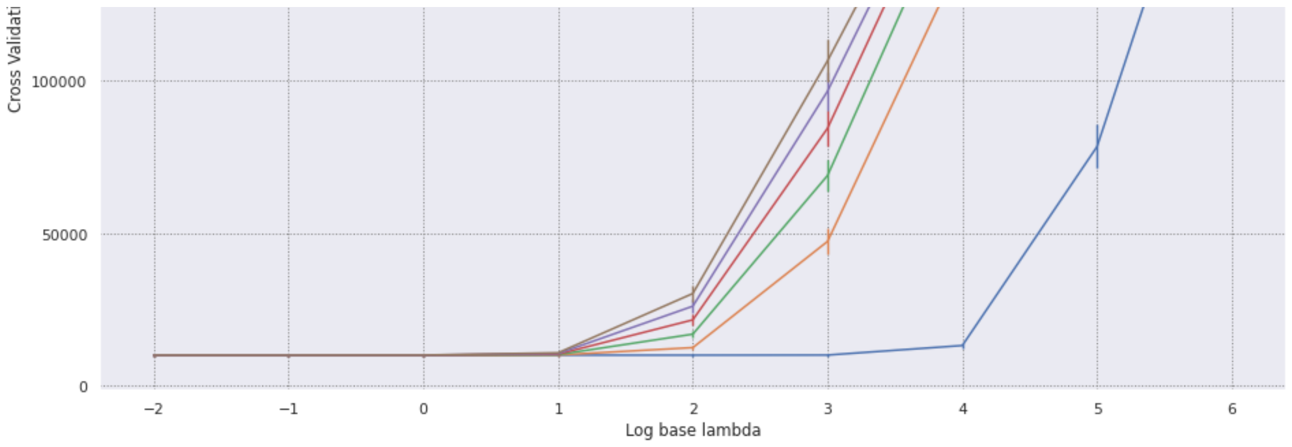
```
168106.3319076381
```

```
Best lambda:= 1.0
```

```
print('Best alpha:=', best_alpha)
```

```
Best alpha:= 0.2
```


−300
−2
−1
0
1
2
3
4
5
6
λ Tuning Params


Cross Validati
100000
50000
0
−2
−1
0
1
2
3
4
5
6
Log base lambda