

CS 4340 - Project 5

Austin Hester

December 05, 2017

Introduction

The goal of this project is to implement linear regression using validation and ridge regularization.

I did linear regression by matrix multiplication, so regularization does not seem to help much, if at all.

Our training data is:

Training Data	
X	y
-2	14
-1	11
0	10
1	11
2	14
3	19
4	26
5	35
6	46
7	59
8	74
9	91
10	110

Our y is given by

$$y = f(x) = x^2 + 10$$

Animated GIF of 3-Fold Validation

The Code

```
# Austin Hester
# Linear Regression with Regularization
# CS 4340 – Intro to Machine Learning
# 12.04.17

import numpy as np
import random
import matplotlib.pyplot as plt

def make_x ():
    X = [ [ 1, i-2 ] for i in range ( 13 ) ]
    return np.array ( X )

def get_y ( X ):
    y = [ x**2 + 10. for x in X ]
    return np.array ( y )

def get_estimated_y ( X, w_lin ):
    return np.array ( [ round ( yi ) for yi in np.dot ( X, w_lin ) ] )

def estimate_single ( N, w_lin ):
    return np.dot ( np.array ( [1, N] ), w_lin )

def print_results ( X, y, y_hat, w_lin ):
    print ( "X =\t", X )
    print ( "Wlin =\t", w_lin )
    print ( "y =\t", y )
    print ( "y^ =\t", y_hat.T )
    slope, intercept = np.polyfit ( X, y_hat, 1 )
    print ( "Equation of linear regression line: ", "y = ",
            slope, "*x + ", intercept )

def plot ( X, y, y_hat, w_lin , turn=0 ):
    fig = plt.figure ( figsize=(6,6) )
    fig.suptitle ( "Linear Regression of Y on X" )
    # Plot training points
    ax = fig.add_subplot ( 1,1,1 )
    ax.scatter ( X, y, cmap='prism' )
    # Plot line learned from linear regression
    slope, intercept = np.polyfit ( X, y_hat, 1 )
    ablines = [ slope * i + intercept for i in X ]
    cx = fig.add_subplot ( 1,1,1 )
    cx.plot ( X, ablines , 'or-' )
```

```

plt.ylabel ( 'y & y^' )
plt.xlabel ( 'x' )
plt.xlim ( -2, 10 )
plt.ylim ( 0, 120 )
#fig.savefig ( "run%d.png" % turn )

def get_MSE ( y, y_hat, w_norm, lam=1 ):
    total = 0
    for i in range ( y.size ):
        total += ( y [i] - y_hat [i] )**2
    return total / y.size + ( lam * w_norm )

def validate ( X, y, turn=0, lam=1 ):

    if ( turn == 0 ):

        Xt, w_lin = linear_regression ( X [:8], y [:8] )
        y_hat = get_estimated_y (X, w_lin)
        w_norm = np.dot ( w_lin.T, w_lin )
        print( "lambda = ", lam )
        print("MSE turn ", turn+1, " = ", get_MSE (y, y_hat, w_norm, lam )
              )
        return Xt, w_lin
    if ( turn == 1 ):
        middle = X[:4] + X[8:12]
        ymiddle = y[:4] + y[8:12]
        Xt, w_lin = linear_regression ( middle, ymiddle )
        y_hat = get_estimated_y (X, w_lin)
        w_norm = np.dot ( w_lin.T, w_lin )
        print( "lambda = ", lam )
        print("MSE turn ", turn+1, " = ", get_MSE (y, y_hat, w_norm, lam
              ))
        return Xt, w_lin
    if ( turn == 2 ):
        Xt, w_lin = linear_regression ( X [4:], y [4:] )
        y_hat = get_estimated_y (X, w_lin)
        w_norm = np.dot ( w_lin.T, w_lin )
        print( "lambda = ", lam )
        print("MSE turn ", turn+1, " = ", get_MSE (y, y_hat, w_norm, lam )
              )
        return Xt, w_lin

def regularize_ridge ( X, y, y_hat, w_lin, lam=1 ):
    w_norm = np.dot ( w_lin.T, w_lin )
    validate ( X, y )
    return get_MSE ( y, y_hat, w_norm, lam )

```

```

def linear_regression ( X, y ):
    XxXT = np.dot ( X.T, X )           # Compute  $X.T * X \leftarrow A$ 
    XxXT_inv = np.linalg.inv ( XxXT )  # Compute inverse of  $A \leftarrow B$ 
    Xt = np.dot ( XxXT_inv, X.T )      # Psuedo-inverse of  $X = B * X.T \leftarrow C$ 
    w_lin = np.dot ( Xt, y.T )          # Compute weight vector  $w_{lin} = C * y$ 
    return Xt, w_lin.T

def run ():
    # Get training points
    X = make_x ()
    X1d = X.T [1] [:]
    y = get_y ( X1d )
    # Run linear regression on those points
    Xt, w_lin = linear_regression ( X, y )
    y_hat = get_estimated_y ( X, w_lin )
    plot ( X1d, y, y_hat , w_lin )
    # Regularize and Validate
    w_norm = np.dot ( w_lin.T, w_lin )
    print ( "MSE = ", get_MSE(y, y_hat , w_norm, 0.1 ))
    print_results ( X1d, y, y_hat , w_lin )
    for j in [ 0.1, 1, 10, 100 ]:
        for i in range(3):
            print ( "_____")
            print ( "Validation", i+1 )
            Xt, w_lin = validate ( X, y, i, j )
            print( w_lin )
            y_hat = get_estimated_y ( X, w_lin )
            plot ( X1d, y, y_hat , w_lin , i+1 )
            print_results ( X1d, y, y_hat , w_lin )

run ()
#plt.show ()

```

Notes

a) Our training data.

Training Data	
X	yl
-2	14
-1	11
0	10
1	11
2	14
3	19
4	26
5	35
6	46
7	59
8	74
9	91
10	110

b) The equation of the line obtained in part (1) is:

$$y = 8x + 8$$

c) For the following data: turn 1 is the first 8 points, turn 2 is all but the middle 4 points, and turn 3 is the last 8 points. I know there are 13 points total, but 13 is prime and we can ignore 1 here or there.

$$\lambda = 0.1$$

MSE turn 1 = 746.8

MSE turn 2 = 204.4

MSE turn 3 = 550.7

$$\lambda = 1$$

MSE turn 1 = 907.0

MSE turn 2 = 451.0

MSE turn 3 = 1016.7

$$\lambda = 10$$

MSE turn 1 = 2509.0

MSE turn 2 = 2917.0

MSE turn 3 = 5676.7

$$\lambda = 100$$

MSE turn 1 = 18529.0

MSE turn 2 = 27577.0

MSE turn 3 = 52276.7

d) I chose $\lambda = 0.1$ because it gives the lowest MSE (besides zero). I do not really see the point of regularization when using linear regression. Linear regression is a non-iterative process, and it does not use MSE or SSE when obtaining a weight vector.

e) Our final equation of the line is:

$$y = 7x + 15$$

Example Run

Here is example output of a run of 3-fold validation

```
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 8.  8.]
y = [ 14.  11.  10.  11.  14.  19.  26.  35.  46.  59.  74.  91.
      110.]
y^ = [ -8. -0.  8.  16.  24.  32.  40.  48.  56.  64.  72.  80.  88.]
Equation of linear regression line: y = 8.0 *x + 8.0
```

```
Validation 1
lambda = 0.1
MSE turn 1 = 746.8
[ 13.  3.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 13.  3.]
y = [ 14.  11.  10.  11.  14.  19.  26.  35.  46.  59.  74.  91.
      110.]
y^ = [ 7.  10.  13.  16.  19.  22.  25.  28.  31.  34.  37.  40.  43.]
Equation of linear regression line: y = 3.0 *x + 13.0
```

```
Validation 2
lambda = 0.1
MSE turn 2 = 204.4
[ 15.  7.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 15.  7.]
y = [ 14.  11.  10.  11.  14.  19.  26.  35.  46.  59.  74.  91.
      110.]
y^ = [ 1.  8.  15.  22.  29.  36.  43.  50.  57.  64.  71.  78.  85.]
Equation of linear regression line: y = 7.0 *x + 15.0
```

```
Validation 3
lambda = 0.1
MSE turn 3 = 550.777777778
[-19.33333333 12.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [-19.33333333 12.]
y = [ 14.  11.  10.  11.  14.  19.  26.  35.  46.  59.  74.  91.]
```

```

110.]
y^ = [ -43. -31. -19. -7. 5. 17. 29. 41. 53. 65. 77. 89.
101.]
Equation of linear regression line: y = 12.0 *x + -19.0

```

```

Validation 1
lambda = 1
MSE turn 1 = 907.0
[ 13. 3.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 13. 3.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 7. 10. 13. 16. 19. 22. 25. 28. 31. 34. 37. 40. 43.]
Equation of linear regression line: y = 3.0 *x + 13.0

```

```

Validation 2
lambda = 1
MSE turn 2 = 451.0
[ 15. 7.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 15. 7.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 1. 8. 15. 22. 29. 36. 43. 50. 57. 64. 71. 78. 85.]
Equation of linear regression line: y = 7.0 *x + 15.0

```

```

Validation 3
lambda = 1
MSE turn 3 = 1016.77777778
[-19.33333333 12. ]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [-19.33333333 12. ]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ -43. -31. -19. -7. 5. 17. 29. 41. 53. 65. 77. 89.
101.]
Equation of linear regression line: y = 12.0 *x + -19.0

```

```

Validation 1
lambda = 10
MSE turn 1 = 2509.0
[ 13. 3.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 13. 3.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 7. 10. 13. 16. 19. 22. 25. 28. 31. 34. 37. 40. 43.]
Equation of linear regression line: y = 3.0 *x + 13.0

```

```

Validation 2
lambda = 10
MSE turn 2 = 2917.0
[ 15. 7.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 15. 7.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 1. 8. 15. 22. 29. 36. 43. 50. 57. 64. 71. 78. 85.]
Equation of linear regression line: y = 7.0 *x + 15.0

```

```

Validation 3
lambda = 10
MSE turn 3 = 5676.77777778
[-19.33333333 12. ]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [-19.33333333 12. ]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ -43. -31. -19. -7. 5. 17. 29. 41. 53. 65. 77. 89.
101.]
Equation of linear regression line: y = 12.0 *x + -19.0

```

```

Validation 1
lambda = 100
MSE turn 1 = 18529.0
[ 13. 3.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 13. 3.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 7. 10. 13. 16. 19. 22. 25. 28. 31. 34. 37. 40. 43.]
Equation of linear regression line: y = 3.0 *x + 13.0

```

```

Validation 2
lambda = 100
MSE turn 2 = 27577.0
[ 15. 7.]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]
Wlin = [ 15. 7.]
y = [ 14. 11. 10. 11. 14. 19. 26. 35. 46. 59. 74. 91.
110.]
y^ = [ 1. 8. 15. 22. 29. 36. 43. 50. 57. 64. 71. 78. 85.]
Equation of linear regression line: y = 7.0 *x + 15.0

```

```

Validation 3
lambda = 100
MSE turn 3 = 52276.77777778
[-19.33333333 12. ]
X = [-2 -1 0 1 2 3 4 5 6 7 8 9 10]

```

```

Wlin =  [-19.33333333  12.      ]
y  =    [  14.    11.    10.    11.    14.    19.    26.    35.    46.    59.    74.    91.
 110.]
y^ =    [ -43.   -31.   -19.    -7.     5.    17.    29.    41.    53.    65.    77.    89.
 101.]
Equation of linear regression line:  y =  12.0 *x +  -19.0

```
