Linear Regression, Ridge Regression, Lasso Regression

YoungWoong Cho

July 2020

In this project, three different types of linear regression methods are performed: plain linear regression with no regularization, ridge regression with regularization, and lasso regression. After the models were trained, their performances were quantitatively analyzed by the MSE values on the validation datasets.

Dataset : Automobile miles per gallon Data Set

Source : https://archive.ics.uci.edu/ml/datasets/Auto+MPG

Features : cylinders, displacement, horsepower, weight, acceleration, model year, origin

Label : miles per gallon

0.1 Preliminary works

1. Import libraries

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

2. Get a dataset

[ ]:
```
from google.colab import files
upload = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving auto-mpg.csv to auto-mpg.csv
```

[ ]:
```
data = pd.read_csv("auto-mpg.csv")
data.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',␣
↪'acceleration' , 'model year', 'origin', 'car name']
del data['car name']
data.head()
```

[ ]:
```
    mpg  cylinders  displacement  ...  acceleration  model year  origin
0  15.0          8         350.0  ...          11.5          70       1
1  18.0          8         318.0  ...          11.0          70       1
```

```
2  16.0           8         304.0  ...          12.0           70         1
3  17.0           8         302.0  ...          10.5           70         1
4  15.0           8         429.0  ...          10.0           70         1

[5 rows x 8 columns]
```

3. Create train, test and validation sets

```
[ ]: # Normalizing each feature column
for i in range(len(data.iloc[0])):
  max = data.iloc[:,i].max()
  data.iloc[:,i] = data.iloc[:,i]/max

# Creating Train, Test, and Val sets
msk = np.random.rand(len(data)) < 0.8
train = data[msk].dropna(how='any')
not_train = data[~msk].dropna(how='any')
midway = (int)(len(not_train)/2)
val = not_train[:midway]
test = not_train[midway:]
print("Data length: %d, Training data length: %d, Validation data length: %d,␣
↪Testing data length: %d" %(len(data), len(train), len(val), len(test)))
data.head()
```

```
Data length: 391, Training data length: 311, Validation data length: 40, Testing
data length: 40
```

```
[ ]:        mpg  cylinders  displacement  ...  acceleration  model year    origin
0  0.321888        1.0      0.769231  ...      0.463710    0.853659  0.333333
1  0.386266        1.0      0.698901  ...      0.443548    0.853659  0.333333
2  0.343348        1.0      0.668132  ...      0.483871    0.853659  0.333333
3  0.364807        1.0      0.663736  ...      0.423387    0.853659  0.333333
4  0.321888        1.0      0.942857  ...      0.403226    0.853659  0.333333

[5 rows x 8 columns]
```

0.2 Linear Regression

```
[ ]: # Loss function for linear regression
def RSS(X, Y, b):
  """
  Input:
    X: Features; 1*p vector where p is a number of features
    Y: Label; scalar
    b: Coefficients; p*1 vector
  Returns:
    (Y - X*b)^2
```

```
    """
    return (Y - X.dot(b)).T.dot(Y - X.dot(b))
```

```
[ ]: # Training
     # First column is a response vector
     # A column of ones is attached on the left side of the feature matrix train_X
     train_X = np.concatenate( (np.ones( (len(train), 1) ), train.iloc[:, 1:-1].
      ↪to_numpy()), axis=1)
     train_Y = train.iloc[:, 0].to_numpy()

     b = np.linalg.inv(train_X.T.dot(train_X)).dot(train_X.T.dot(train_Y)) # beta␣
      ↪that minimizes RSS

     # Loss function RSS
     RSS_train = RSS(train_X, train_Y, b)

     # MSE
     MSE_train = RSS_train / len(train)

     print("RSS of the training set using this coefficient vector is %f. \nMSE of the␣
      ↪training set using this coefficient vector is %f." %(RSS_train, MSE_train))
```

```
RSS of the training set using this coefficient vector is 1.628525.
MSE of the training set using this coefficient vector is 0.005236.
```

```
[ ]: # Test
     # A column of ones is attached on the left side of the feature matrix train_X
     test_X = np.concatenate( (np.ones( (len(test), 1) ), test.iloc[:, 1:-1].
      ↪to_numpy()), axis=1)
     test_Y = test.iloc[:, 0].to_numpy()

     # RSS
     RSS_test = RSS(test_X, test_Y, b)

     # MSE
     MSE_test = RSS_test / len(test)

     print("RSS of the test set using this coefficient vector is %f. \nMSE of the␣
      ↪test set using this coefficient vector is %f." %(RSS_test, MSE_test))
     print("b0 intercept is %f" %b[0])
```

```
RSS of the test set using this coefficient vector is 0.327808.
MSE of the test set using this coefficient vector is 0.008195.
b0 intercept is -0.312111
```

0.3 Ridge Regression

[ ]:
```
# Loss function for ridge regression
def RSS_ridge(X, Y, b, lamb):
  """
  Input:
    X: Features; 1*p vector where p is a number of features
    Y: Label; scalar
    b: Coefficients; p*1 vector
    lamb: Tuning parameter; scalar
  Returns:
    (Y - X*b)^2 + lamb * b^2
  """
  return (Y - X.dot(b)).T.dot(Y - X.dot(b)) + lamb*b.T.dot(b)
```

[ ]:
```
# Training
# First column is a response vector
# Last column is not used
train_X = np.concatenate( (np.ones( (len(train), 1) ), train.iloc[:, 1:-1].
 ↪to_numpy()), axis=1)
train_Y = train.iloc[:, 0].to_numpy()

# Linear space of lambda is created
lambs = np.linspace(0, 0.01, 1000)

# Empty list to store b is created
b_ridge = []

# Calculate coefficient vectors for each lambda
for lamb in lambs:
  b_ridge.append(np.linalg.inv( train_X.T.dot(train_X) + lamb*np.
 ↪identity(len(train_X[0])) ).dot(train_X.T.dot(train_Y)))
```

[ ]:
```
# Cross-Validation
# Find optimal lambda value using the calculaed weighting factors from training␣
 ↪dataset
val_X = np.concatenate( (np.ones( (len(val), 1) ), val.iloc[:, 1:-1].
 ↪to_numpy()), axis=1)
val_Y = val.iloc[:, 0].to_numpy()

# Empty lists of RSS and MSE are created
RSS_ridge = []
MSE_ridge = []

for b in b_ridge:
  RSS_temp = RSS(val_X, val_Y, b)  # temporary RSS created to avoid␣
 ↪unnecerassily repeated calculation
```

```
    RSS_ridge.append(RSS_temp)
    MSE_ridge.append(RSS_temp/len(val))

# Find
RSS_min = min(RSS_ridge)
MSE_min = RSS_min/len(train)
lamb_opt = lambs[RSS_ridge.index(min(RSS_ridge))]
b_opt = b_ridge[RSS_ridge.index(min(RSS_ridge))]
print("Minimum RSS from the validation set using this coefficient vector is %f.
 ↪\nMinimum MSE of the validation set using this coefficient vector is %f.\nThis␣
 ↪occurs when lambda is %f.\n" %(RSS_min, MSE_min, lamb_opt))
print("Optimal coefficient values are:")
print(b_opt)
```

```
Minimum RSS from the validation set using this coefficient vector is 0.136990.
Minimum MSE of the validation set using this coefficient vector is 0.000440.
This occurs when lambda is 0.000000.

Optimal coefficient values are:
[-0.31211128 -0.09859175  0.09599611  0.03236082 -0.75142705  0.074578
  1.31250979]
```

[ ]:
```
# Test
# A column of ones is attached on the left side of the feature matrix train_X
test_X = np.concatenate( (np.ones( (len(test), 1) ), test.iloc[:, 1:-1].
 ↪to_numpy()), axis=1)
test_Y = test.iloc[:, 0].to_numpy()

# RSS
RSS_test = RSS(test_X, test_Y, b_opt)

# MSE
MSE_test = RSS_test / len(test)

print("RSS of the test set using this coefficient vector is %f. \nMSE of the␣
 ↪test set using this coefficient vector is %f." %(RSS_test, MSE_test))
print("b0 intercept is %f" %b_opt[0])
```

```
RSS of the test set using this coefficient vector is 0.327808.
MSE of the test set using this coefficient vector is 0.008195.
b0 intercept is -0.312111
```

0.4 Lasso Regression

[ ]:
```
# Create training data
train_X = np.concatenate( (np.ones( (len(train), 1) ), train.iloc[:, 1:-1].
 ↪to_numpy()), axis=1)
train_Y = train.iloc[:, 0].to_numpy()

val_X = np.concatenate( (np.ones( (len(val), 1) ), val.iloc[:, 1:-1].
 ↪to_numpy()), axis=1)
val_Y = val.iloc[:, 0].to_numpy()

# Sample lambda values same as ridge
lambs = np.linspace(0, 0.01, 1000)

# b and MSE lists
b_lasso = []
MSE_lasso = []

for lamb in lambs:
  # Reset squared error
  sum = 0

  # Train with current lambda
  lasso_reg = linear_model.Lasso(alpha=lamb, normalize=True)
  lasso_reg.fit(train_X, train_Y)

  # Find optimal lambda using val set and predict y values
  yHat = lasso_reg.predict(val_X)

  for i in range(len(val_X)):
    # Add up squared errors of val sets for this lambda
    sum = sum + (val_Y[i] - yHat[i])**2

  # Find MSE
  MSE_lasso.append(sum/len(val_X))

  # Append b
  b_lasso.append(lasso_reg.coef_)
```

[ ]:
```
data_lasso = pd.DataFrame({'lambda' : lambs, 'MSE' : MSE_lasso})

print(data_lasso)
print("Minimum MSE occurs when lambda is %d and its value is %f"␣
 ↪%(data_lasso['MSE'].idxmin(), data_lasso['MSE'].min()))
```

```
      lambda       MSE
0    0.00000  0.003425
```

```
1     0.00001  0.003370
2     0.00002  0.003321
3     0.00003  0.003272
4     0.00004  0.003235
..        ...       ...
995   0.00996  0.016285
996   0.00997  0.016285
997   0.00998  0.016285
998   0.00999  0.016285
999   0.01000  0.016285

[1000 rows x 2 columns]
Minimum MSE occurs when lambda is 69 and its value is 0.003090
```

Lasso plot

[ ]:
```python
plt.xlabel("Lambda")
plt.ylabel("Coefficients")
plt.plot(lambs, b_lasso)
len(b_lasso[0])
```

[ ]: 7