Assignment: Regression

Qianlang Chen (u1172983)

CS 5140 Spring 2021

```
import numpy

X = numpy.loadtxt('./data/X.csv', delimiter=',')
y = numpy.loadtxt('./data/y.csv', delimiter=',')
M = numpy.loadtxt('./data/M.csv', delimiter=',')
W = numpy.loadtxt('./data/W.csv', delimiter=',')
print(X.shape, y.shape, M.shape, W.shape)
```

```
(100, 50) (100,) (50, 20) (50,)
```

Problem 1

Part A

```
from numpy import linalg

def least_squares(X, y): return linalg.inv(X.T @ X) @ X.T @ y.T

def ridge(X, y, s):
    return (linalg.inv(X.T @ X + s * numpy.identity(X.shape[1]))
            @ X.T @ y.T)

def sse(X, y, alpha): return linalg.norm(y.T - X @ alpha, 2)
```

Using Least Squares:

```
alpha = least_squares(X, y)
print(f'Error: {sse(X, y, alpha)}')
```

```
Error: 3.452257117069318
```

Using Ridge Regression:

```
S = [.1, .3, .7, .9, 1.1, 1.3, 1.5]
for s in S:
    alpha = ridge(X, y, s)
    print(f'Error with s = {s}: {sse(X, y, alpha)}')
```

```
Error with s = 0.1: 3.697108163766999
Error with s = 0.3: 3.9003300572591115
Error with s = 0.7: 4.198778810955157
Error with s = 0.9: 4.327017561622948
Error with s = 1.1: 4.445519765867784
Error with s = 1.3: 4.555758705764217
Error with s = 1.5: 4.658819389955557
```

Part B

```
def cross_validate(X_learn, y_learn, X_test, y_test, regression, *args):
    alpha = regression(X_learn, y_learn, *args)
    return sse(X_test, y_test, alpha)

X1, X1_r = X[:75, :], X[75:, :]
y1, y1_r = y[:75], y[75:]
X2, X2_r = X[25:, :], X[:25, :]
y2, y2_r = y[25:], y[:25]
X3, X3_r = (numpy.vstack((X[:50, :], X[75:, :])), X[50:75, :])
y3, y3_r = (numpy.concatenate((y[:50], y[75:])), y[50:75])
X4, X4_r = (numpy.vstack((X[:25, :], X[50:, :])), X[25:50, :])
y4, y4_r = (numpy.concatenate((y[:25], y[50:])), y[25:50])
```

Using Least Squares:

```
print('Error of (X1, y1):'
      f' {cross_validate(X1, y1, X1_r, y1_r, least_squares)}')
print('Error of (X2, y2):'
      f' {cross_validate(X2, y2, X2_r, y2_r, least_squares)}')
print('Error of (X3, y3):'
      f' {cross_validate(X3, y3, X3_r, y3_r, least_squares)}')
print('Error of (X4, y4):'
      f' {cross_validate(X4, y4, X4_r, y4_r, least_squares)}')
```

```
Error of (X1, y1): 4.574549134311085
Error of (X2, y2): 3.953705254240132
Error of (X3, y3): 4.239879523801711
Error of (X4, y4): 3.9381553312790407
```

Using Ridge Regression:

```
for s in S:
    print(f'With s = {s}:')
    errors = [cross_validate(X1, y1, X1_r, y1_r, ridge, s),
              cross_validate(X2, y2, X2_r, y2_r, ridge, s),
```

```
            cross_validate(X3, y3, X3_r, y3_r, ridge, s),
            cross_validate(X4, y4, X4_r, y4_r, ridge, s)]
    print(f'    Error of (X1, y1): {errors[0]}')
    print(f'    Error of (X2, y2): {errors[1]}')
    print(f'    Error of (X3, y3): {errors[2]}')
    print(f'    Error of (X4, y4): {errors[3]}')
    print(f'    Average error: {sum(errors) / 4}')
    print('')
```

```
With s = 0.1:
    Error of (X1, y1): 2.9032557648579354
    Error of (X2, y2): 2.435781393405244
    Error of (X3, y3): 2.4562579841221472
    Error of (X4, y4): 2.6224416883243347
    Average error: 2.604434207677415

With s = 0.3:
    Error of (X1, y1): 2.812360286833187
    Error of (X2, y2): 2.4362818335730734
    Error of (X3, y3): 2.3902049265642913
    Error of (X4, y4): 2.3506279951743188
    Average error: 2.497368760536218

With s = 0.7:
    Error of (X1, y1): 2.871140069262489
    Error of (X2, y2): 2.6205294719938363
    Error of (X3, y3): 2.399715705029355
    Error of (X4, y4): 2.295868919335064
    Average error: 2.546813541405186

With s = 0.9:
    Error of (X1, y1): 2.918800501686161
    Error of (X2, y2): 2.7029498819203344
    Error of (X3, y3): 2.417625648786458
    Error of (X4, y4): 2.3251598957110566
    Average error: 2.5911339820260024
```

```
With s = 1.1:
    Error of (X1, y1): 2.966809395453503
    Error of (X2, y2): 2.77732331042384
    Error of (X3, y3): 2.438798749191682
    Error of (X4, y4): 2.3660843057969854
    Average error: 2.6372539402165023

With s = 1.3:
    Error of (X1, y1): 3.0131279773121937
    Error of (X2, y2): 2.844756655775371
    Error of (X3, y3): 2.4616191724047525
    Error of (X4, y4): 2.41237064332693
    Average error: 2.682968612204812

With s = 1.5:
    Error of (X1, y1): 3.057072486774334
    Error of (X2, y2): 2.9062788229288645
    Error of (X3, y3): 2.485140097765302
    Error of (X4, y4): 2.4608297993422954
    Average error: 2.727330301702699
```

Part C

It looks like Ridge Regression with $s = 0.3$ worked the best out of these options (2.497 average error across the four splits).

Part D

```
splits = [(X1, y1, X1_r, y1_r), (X2, y2, X2_r, y2_r),
          (X3, y3, X3_r, y3_r), (X4, y4, X4_r, y4_r)]
for i, split in enumerate(splits):
    sum_errors = (cross_validate(*split, least_squares)**2
                  / len(split[3]))
    for s in S:
```

```
        sum_errors += (cross_validate(*split, ridge, s)**2
                       / len(split[3]))
    print(f'Average error of (X{i + 1}, y{i + 1}):'
          f' {sum_errors / (1 + len(S))}')
```

```
Average error of (X1, y1): 0.40627098126668415
Average error of (X2, y2): 0.3296300999101577
Average error of (X3, y3): 0.297548200445271
Average error of (X4, y4): 0.2803133639664369
```

The above four train/test splits are all 75/25 splits, but all those 75 items and 25 items are consecutive items from the original data array. Since there might be bias going on in the ordering of the items in the original data array (like if the data of people from the same region are grouped together), this might introduce bias to our models and ultimately influence our choice of s.

Part E

We must assume that the ordering of the items in the original data array does not matter; otherwise, we should draw random items instead of consecutive items from the data array when doing our train/test splits in order to overcome this problem.