Assignment: Regression

Qianlang Chen (u1172983)

CS 5140 Spring 2021

```python
import numpy

X = numpy.loadtxt('./data/X.csv', delimiter=',')
X = numpy.insert(X, 0, 1, 1) # bias column
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, 51) (100,) (50, 20) (50,)
```

Problem 1

Part A

```python
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)
```

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

Using Least Squares:

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

```
Error: 3.4048156890344536
```

Using Ridge Regression:

```python
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: 3.6791753582507223
Error with s = 0.3: 3.8911188767606615
Error with s = 0.7: 4.195034819548265
Error with s = 0.9: 4.324427630424707
Error with s = 1.1: 4.4437451387072455
Error with s = 1.3: 4.5546145230060135
Error with s = 1.5: 4.658189475920739
```

Part B

```python
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): 4.392775779197778
Error of (X2, y2): 3.890628108757094
Error of (X3, y3): 4.720398268070038
Error of (X4, y4): 3.9928547236443577
```

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): 2.9000624224225615
    Error of (X2, y2): 2.493745256464902
    Error of (X3, y3): 2.4792813042683335
```

```
    Error of (X4, y4): 2.625256751154926
    Average error: 2.6245864335776807

With s = 0.3:
    Error of (X1, y1): 2.8136316065046736
    Error of (X2, y2): 2.5221424744252476
    Error of (X3, y3): 2.4031595700892763
    Error of (X4, y4): 2.3551174550365452
    Average error: 2.5235127765139356

With s = 0.7:
    Error of (X1, y1): 2.8975928884921665
    Error of (X2, y2): 2.7235132577998176
    Error of (X3, y3): 2.4160840494727975
    Error of (X4, y4): 2.2935422285125395
    Average error: 2.58268310606933

With s = 0.9:
    Error of (X1, y1): 2.9508290919838958
    Error of (X2, y2): 2.809243893764796
    Error of (X3, y3): 2.4363274010665723
    Error of (X4, y4): 2.3216825487186394
    Average error: 2.6295207338834756

With s = 1.1:
    Error of (X1, y1): 3.0016856215061782
    Error of (X2, y2): 2.8853852851914032
    Error of (X3, y3): 2.459779255858018
    Error of (X4, y4): 2.362213424088662
    Average error: 2.677265896661065

With s = 1.3:
    Error of (X1, y1): 3.04917198231318
    Error of (X2, y2): 2.9536260732791892
    Error of (X3, y3): 2.4847443379924483
    Error of (X4, y4): 2.408480564274436
```

```
    Average error: 2.7240057394648134

With s = 1.5:
    Error of (X1, y1): 3.0932472496061285
    Error of (X2, y2): 3.015333368465654
    Error of (X3, y3): 2.510243429133295
    Error of (X4, y4): 2.4570836242035345
    Average error: 2.768976917852153
```

Part C

It looks like Ridge Regression with $s = 0.3$ worked the best out of these options (2.523 average error across the four splits).

Part D

(Note: the assignment instruction was unclear on whether I should average the test errors from *Part B* or the squared errors – the errors from *Part B* squared. Nor was it clear on whether to report the average errors separately for each Ridge's s-value or combine them. So, I reported all the average errors under different interpretations.)

```python
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):
    n_test = len(split[3])
    print(f'Average error of (X{i + 1}, y{i + 1}):')
    err = cross_validate(*split, least_squares)
    print(f'    Least Squares: {err / n_test}\n'
          f'    Least Squares (SSE): {err**2 / n_test}')
    sum_errs = sum_sse = 0
    for s in S:
        err = cross_validate(*split, ridge, s)
        sum_errs += err
        sum_sse += err**2
        print(f'    Ridge with s = {s}: {err / n_test}\n'
              f'    Ridge with s = {s} (SSE): {err**2 / n_test}')
```

```python
    print(f'    Ridge average: {sum_errs / len(S) / n_test}\n'
          f'    Ridge average (SSE): {sum_sse / len(S) / n_test}')
    print('')
```

```
Average error of (X1, y1):
    Least Squares: 0.17571103116791112
    Least Squares (SSE): 0.7718591618522658)
    Ridge with s = 0.1: 0.11600249689690247
    Ridge with s = 0.1 (SSE): 0.33641448215789665
    Ridge with s = 0.3: 0.11254526426018695
    Ridge with s = 0.3 (SSE): 0.3166609126848828
    Ridge with s = 0.7: 0.11590371553968666
    Ridge with s = 0.7 (SSE): 0.3358417818976151
    Ridge with s = 0.9: 0.11803316367935583
    Ridge with s = 0.9 (SSE): 0.34829569320394005
    Ridge with s = 1.1: 0.12006742486024713
    Ridge with s = 1.1 (SSE): 0.36040466281427724
    Ridge with s = 1.3: 0.1219668792925272
    Ridge with s = 1.3 (SSE): 0.37189799110894756
    Ridge with s = 1.5: 0.12372988998424514
    Ridge with s = 1.5 (SSE): 0.3827271418878352
    Ridge average: 0.11832126207330734
    Ridge average (SSE): 0.3503203808221992

Average error of (X2, y2):
    Least Squares: 0.15562512435028375
    Least Squares (SSE): 0.6054794832260321)
    Ridge with s = 0.1: 0.09974981025859607
    Ridge with s = 0.1 (SSE): 0.24875061616564798
    Ridge with s = 0.3: 0.1008856989770099
    Ridge with s = 0.3 (SSE): 0.25444810645199645
    Ridge with s = 0.7: 0.1089405303119927
    Ridge with s = 0.7 (SSE): 0.296700978616455
    Ridge with s = 0.9: 0.11236975575059184
    Ridge with s = 0.9 (SSE): 0.31567405018619166
    Ridge with s = 1.1: 0.11541541140765613
```

```
    Ridge with s = 1.1 (SSE): 0.33301792975996297
    Ridge with s = 1.3: 0.11814504293116757
    Ridge with s = 1.3 (SSE): 0.34895627923018574
    Ridge with s = 1.5: 0.12061333473862616
    Ridge with s = 1.5 (SSE): 0.3636894129192971
    Ridge average: 0.1108742263393772
    Ridge average (SSE): 0.3087481961899624

Average error of (X3, y3):
    Least Squares: 0.1888159307228015
    Least Squares (SSE): 0.8912863923679444)
    Ridge with s = 0.1: 0.09917125217073335
    Ridge with s = 0.1 (SSE): 0.24587343142777954
    Ridge with s = 0.3: 0.09612638280357105
    Ridge with s = 0.3 (SSE): 0.23100703677246698
    Ridge with s = 0.7: 0.09664336197891191
    Ridge with s = 0.7 (SSE): 0.23349848536467488
    Ridge with s = 0.9: 0.09745309604266289
    Ridge with s = 0.9 (SSE): 0.23742764820751194
    Ridge with s = 1.1: 0.09839117023432072
    Ridge with s = 1.1 (SSE): 0.24202055950197698
    Ridge with s = 1.3: 0.09938977351969794
    Ridge with s = 1.3 (SSE): 0.24695817700742123
    Ridge with s = 1.5: 0.1004097371653318
    Ridge with s = 1.5 (SSE): 0.2520528829402754
    Ridge average: 0.09822639627360426
    Ridge average (SSE): 0.24126260303172956

Average error of (X4, y4):
    Least Squares: 0.1597141889457743
    Least Squares (SSE): 0.6377155537651624)
    Ridge with s = 0.1: 0.10501027004619705
    Ridge with s = 0.1 (SSE): 0.2756789203793807
    Ridge with s = 0.3: 0.09420469820146181
    Ridge with s = 0.3 (SSE): 0.22186312908071254
    Ridge with s = 0.7: 0.09174168914050158
```

```
Ridge with s = 0.7 (SSE): 0.21041343815881064
Ridge with s = 0.9: 0.09286730194874558
Ridge with s = 0.9 (SSE): 0.2156083942809871
Ridge with s = 1.1: 0.09448853696354648
Ridge with s = 1.1 (SSE): 0.22320209043778724
Ridge with s = 1.3: 0.09633922257097743
Ridge with s = 1.3 (SSE): 0.23203114513950823
Ridge with s = 1.5: 0.09828334496814138
Ridge with s = 1.5 (SSE): 0.24149039745316705
Ridge average: 0.09613358054851019
Ridge average (SSE): 0.23146964499005054
```

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.