Data Mining Assignment 6

Author: Lukas Gust

Due: March 27th

1 Linear Regression

Description:

We will find coefficients A to estimate $XA \approx Y$, using the provided datasets `X` and `Y`. We will copmare two approaches *least squares* and *ridge regression*.

A: Solve for the coefficients A (or As) using Least Squares and Ridge Regression with $s = \{1, 5, 10, 15, 20, 25, 30\}$. For each set of coefficients, report the error in the estiamate $\hat{Y}$ of Y as `norm(Y-X*A,2)`.

B: Create three row subsets of X and Y

- `X1 = X(1:66,:)` and `Y1 = Y(1:66)`
- `X2 = X(34:100,:)` and `Y2 = Y(34:100)`
- `X3 = [X(1:33,:), X(67:100,:)]` and `Y3 = [Y(1:33), Y(67:100)]`

Repeat the above procedure on these subsets and *cross-validate* the solution on the remainder of X and Y. Specifically, learn the coefficients A using, say, $X1$ and $Y1$ and then measure `norm(Y(67:100) - X(67:100,:)*A,2)`.

C: Which approach works best (averaging the results from the three subsets): Least Squares, or for which value of s using Ridge Regression?

Solution:

A: Using `numpy`'s least squares implementation we obtain the following coefficients and residual.

```
Coefficients:
[[  6.41766065]
 [  0.26256845]
 [  4.88999221]
 [  1.2352192 ]
 [  1.70850062]
 [  5.53776502]
 [  8.48696144]
 [ -1.46156254]
 [ 10.69158285]
 [  5.86161369]
 [  5.75864638]
 [  5.32731348]
 [  4.71614203]
 [  8.82500343]
 [  7.46678008]]
```

```
Squared Residual: 366817.621755
Residual: 605.654705055
```

Now we compute the same values but using Ridge Regression using the following function.

```
def ridge_regress(x, y, s):
    A = np.linalg.inv(x.T.dot(x) + (s**2)*np.eye(15)).dot(x.T).dot(y)/s
    A *= s
    r = np.linalg.norm(y - x.dot(A))
    return A, r
```

```
s: 1
Coeffs:
[[  6.41329395]
 [  0.2708133 ]
 [  4.89271436]
 [  1.2370286 ]
 [  1.71154643]
 [  5.53618856]
 [  8.48291189]
 [ -1.45037748]
 [ 10.68433057]
 [  5.86355149]
 [  5.75774639]
 [  5.32605177]
 [  4.71665642]
 [  8.81745469]
 [  7.46193719]]
Residual: 605.654905335

s: 5
Coeffs:
[[  6.3144084 ]
 [  0.45926442]
 [  4.95292738]
 [  1.28104106]
 [  1.78287946]
 [  5.50081295]
 [  8.38877667]
 [ -1.19348765]
 [ 10.51632748]
 [  5.90546875]
 [  5.7370862 ]
 [  5.29779357]
 [  4.72762651]
 [  8.6447522 ]
 [  7.35073904]]
Residual: 605.770819125

s: 10
Coeffs:
```

```
[[  6.06445027]
 [  0.95121332]
 [  5.09134618]
 [  1.42164898]
 [  1.98614387]
 [  5.41379184]
 [  8.12756065]
 [ -0.51097348]
 [ 10.05592736]
 [  5.9917465 ]
 [  5.6814612 ]
 [  5.22879208]
 [  4.74892976]
 [  8.19207991]
 [  7.05585616]]
Residual: 607.156835367

s: 15
Coeffs:
[[ 5.7770121 ]
 [ 1.54606466]
 [ 5.21987723]
 [ 1.64877225]
 [ 2.2710863 ]
 [ 5.31594819]
 [ 7.77619758]
 [ 0.34013566]
 [ 9.44990537]
 [ 6.04376277]
 [ 5.60667162]
 [ 5.15223466]
 [ 4.75954125]
 [ 7.63924417]
 [ 6.69014743]]
Residual: 611.305417879

s: 20
Coeffs:
[[ 5.51997386]
 [ 2.10271064]
 [ 5.29735333]
 [ 1.93040192]
 [ 2.58600186]
 [ 5.22576702]
 [ 7.4017201 ]
 [ 1.16607527]
 [ 8.82064258]
 [ 6.02672105]
 [ 5.52067306]
 [ 5.08259357]
 [ 4.75250127]
 [ 7.11191047]
 [ 6.33777179]]
```

```
Residual: 618.446855656

s: 25
Coeffs:
[[ 5.31100908]
 [ 2.5638321 ]
 [ 5.3250843 ]
 [ 2.22699673]
 [ 2.89071749]
 [ 5.14402494]
 [ 7.04494982]
 [ 1.87511784]
 [ 8.23729851]
 [ 5.9481749 ]
 [ 5.42668132]
 [ 5.01981945]
 [ 4.73124463]
 [ 6.66033652]
 [ 6.03544159]]
Residual: 627.94392425

s: 30
Coeffs:
[[ 5.14408584]
 [ 2.92312491]
 [ 5.31780451]
 [ 2.50812809]
 [ 3.1616424 ]
 [ 5.0667758 ]
 [ 6.72331031]
 [ 2.44584544]
 [ 7.72558837]
 [ 5.83020526]
 [ 5.32735522]
 [ 4.96020038]
 [ 4.70071303]
 [ 6.28923647]
 [ 5.78774796]]
Residual: 639.168019892
```

B: Since the implementation is in python and the assignment was assuming Matlab the indices may be off by one. We run the same procedure from above on the subsets.

```
X1 = X[0:66]
X1_C = X[66:]

Y1 = Y[0:66]
Y1_C = Y[66:]

X2 = X[34:100]
X2_C = X[0:34]

Y2 = Y[34:100]
```

```
Y2_C = Y[0:34]

X3 = np.vstack([X[0:33], X[67:100]])
X3_C = X[33:67]

Y3 = np.vstack([Y[0:33], Y[67:100]])
Y3_C = Y[33:67]
```

Using least squares we get the following results. The residual is the normal residual from the fitted values. Cross-Val Residual is the residual of the non-fitted values.

```
X1
Coeffs:
[[ 2.93837499]
 [ 5.30919625]
 [ 5.41512021]
 [-3.55623211]
 [ 1.32133517]
 [ 5.79768578]
 [ 9.37297505]
 [-2.46096478]
 [ 8.58540281]
 [ 7.37732721]
 [ 6.63872593]
 [ 7.6307665 ]
 [ 6.36390943]
 [ 7.36108316]
 [ 6.28496322]]
Residual: 441.121873562
Cross-Val Residual: 483.02361036

X2
Coeffs:
[[  7.35734429]
 [ -2.05293525]
 [  5.82743635]
 [  2.22701997]
 [  2.80308908]
 [  6.00253633]
 [ 11.17105387]
 [ -2.62408297]
 [ 12.47457376]
 [  3.13478809]
 [  3.90879956]
 [  3.2143665 ]
 [  5.17503073]
 [ 10.47767815]
 [  6.90077289]]
Residual: 460.317108454
Cross-Val Residual: 432.369392276

X3
Coeffs:
```

```
[[  6.46688093]
 [ -0.4190106 ]
 [  5.05411275]
 [  3.72605859]
 [  1.41241571]
 [  6.6715221 ]
 [  4.55263264]
 [ -1.29318511]
 [ 11.37494853]
 [  7.69240024]
 [  4.88347493]
 [  5.5831921 ]
 [  2.72501327]
 [  8.77743236]
 [  8.80027736]]
Residual: 475.645392334
Cross-Val Residual: 411.176725128
```

With and average Cross-Val Residual of 442.18991

Now we do the same for Ridge Regression for each s.

X1:

```
s: 1
Coeffs:
[[ 2.94404236]
 [ 5.3054141 ]
 [ 5.41459831]
 [-3.53881916]
 [ 1.3237142 ]
 [ 5.79300275]
 [ 9.36109156]
 [-2.4414278 ]
 [ 8.58310647]
 [ 7.37552955]
 [ 6.63687018]
 [ 7.62214993]
 [ 6.35541118]
 [ 7.36016142]
 [ 6.28281207]]
Residual: 441.122367778
Cross-Val Residual: 482.731584281

s: 5
Coeffs:
[[ 3.07046758]
 [ 5.22679266]
 [ 5.40182967]
 [-3.14687926]
 [ 1.38749895]
 [ 5.6950534 ]
 [ 9.09843625]
```

```
 [-2.00692676]
 [ 8.51753155]
 [ 7.32770509]
 [ 6.59289145]
 [ 7.4352481 ]
 [ 6.16549435]
 [ 7.33280277]
 [ 6.23079288]]
Residual: 441.398148722
Cross-Val Residual: 476.42195304

s: 10
Coeffs:
[[ 3.36983384]
 [ 5.07320362]
 [ 5.36146303]
 [-2.16741663]
 [ 1.6234979 ]
 [ 5.49911213]
 [ 8.4756159 ]
 [-0.95856063]
 [ 8.25577877]
 [ 7.15555267]
 [ 6.46167969]
 [ 7.01837492]
 [ 5.71189711]
 [ 7.21181187]
 [ 6.07551326]]
Residual: 444.403738592
Cross-Val Residual: 462.924744963

s: 15
Coeffs:
[[ 3.66982193]
 [ 4.94325791]
 [ 5.30138266]
 [-1.04077209]
 [ 2.00512126]
 [ 5.32657187]
 [ 7.79931412]
 [ 0.19547149]
 [ 7.82467616]
 [ 6.88133711]
 [ 6.26498619]
 [ 6.60438484]
 [ 5.24877866]
 [ 6.98450013]
 [ 5.86566675]]
Residual: 452.369857778
Cross-Val Residual: 451.76926446

s: 20
Coeffs:
```

```
[[ 3.88986669]
 [ 4.84337956]
 [ 5.23591756]
 [-0.0232174 ]
 [ 2.4272083 ]
 [ 5.18992241]
 [ 7.20475302]
 [ 1.19884614]
 [ 7.34528029]
 [ 6.56746204]
 [ 6.03450476]
 [ 6.26765808]
 [ 4.8980395 ]
 [ 6.69716625]
 [ 5.65774081]]
Residual: 464.561274076
Cross-Val Residual: 446.509063605

s: 25
Coeffs:
[[ 4.02726697]
 [ 4.75895636]
 [ 5.17309134]
 [ 0.80160281]
 [ 2.80948618]
 [ 5.07380189]
 [ 6.71730502]
 [ 1.98609503]
 [ 6.89997267]
 [ 6.25771961]
 [ 5.79892132]
 [ 6.00322998]
 [ 4.66237413]
 [ 6.39908603]
 [ 5.47657031]]
Residual: 479.338501177
Cross-Val Residual: 446.857835074

s: 30
Coeffs:
[[ 4.1021696 ]
 [ 4.68031712]
 [ 5.11284998]
 [ 1.43631989]
 [ 3.12019376]
 [ 4.96718468]
 [ 6.32338104]
 [ 2.57369579]
 [ 6.51580399]
 [ 5.97131155]
 [ 5.57478617]
 [ 5.79012492]
 [ 4.50610389]
```

```
 [ 6.11764744]
 [ 5.32252723]]
Residual: 495.828766523
Cross-Val Residual: 451.70808544
```

X2:

```
s: 1
Coeffs:
[[  7.34917723]
 [ -2.03172203]
 [  5.83355116]
 [  2.22956854]
 [  2.79859205]
 [  6.0000272 ]
 [ 11.15481816]
 [ -2.59747264]
 [ 12.45891622]
 [  3.13997222]
 [  3.90951741]
 [  3.21317438]
 [  5.18287443]
 [ 10.45935566]
 [  6.89435529]]
Residual: 460.317884754
Cross-Val Residual: 432.12549152

s: 5
Coeffs:
[[  7.16954541]
 [ -1.56520285]
 [  5.95878832]
 [  2.28753178]
 [  2.71672878]
 [  5.94207844]
 [ 10.7962827 ]
 [ -2.01450296]
 [ 12.10452679]
 [  3.25622609]
 [  3.93209171]
 [  3.19342451]
 [  5.34537399]
 [ 10.05442627]
 [  6.75117413]]
Residual: 460.737029154
Cross-Val Residual: 426.930452699

s: 10
Coeffs:
[[  6.75422925]
 [ -0.48709288]
 [  6.17314603]
 [  2.44474621]
```

```
 [  2.65413605]
 [  5.78735867]
 [  9.94923011]
 [ -0.68280604]
 [ 11.20649535]
 [  3.5390986 ]
 [  4.03174105]
 [  3.20106966]
 [  5.63882215]
 [  9.10526219]
 [  6.40620748]]
Residual: 464.928665182
Cross-Val Residual: 416.072928284

s: 15
Coeffs:
[[  6.32480315]
 [  0.61444606]
 [  6.26316239]
 [  2.66664754]
 [  2.77890924]
 [  5.5947248 ]
 [  9.03290933]
 [  0.65755406]
 [ 10.15919231]
 [  3.83887119]
 [  4.19152093]
 [  3.29911322]
 [  5.79441884]
 [  8.11877835]
 [  6.03642203]]
Residual: 474.903716986
Cross-Val Residual: 406.537371916

s: 20
Coeffs:
[[ 5.96144139]
 [ 1.50194819]
 [ 6.21757229]
 [ 2.92131624]
 [ 3.02510895]
 [ 5.4034904 ]
 [ 8.22584941]
 [ 1.72251937]
 [ 9.19919487]
 [ 4.07130938]
 [ 4.34440934]
 [ 3.45271122]
 [ 5.78437083]
 [ 7.31213332]
 [ 5.72753466]]
Residual: 488.879359233
Cross-Val Residual: 400.263716088
```

```
s: 25
Coeffs:
[[ 5.66456745]
 [ 2.15802562]
 [ 6.09653799]
 [ 3.17271742]
 [ 3.29408651]
 [ 5.2273677 ]
 [ 7.56057555]
 [ 2.49811954]
 [ 8.39973413]
 [ 4.21970714]
 [ 4.45209357]
 [ 3.60868054]
 [ 5.67579874]
 [ 6.70352649]
 [ 5.48929392]]
Residual: 504.874492844
Cross-Val Residual: 397.211135262

s: 30
Coeffs:
[[ 5.41801338]
 [ 2.62771105]
 [ 5.94620115]
 [ 3.39460041]
 [ 3.53212323]
 [ 5.06761622]
 [ 7.02110976]
 [ 3.04353499]
 [ 7.75436723]
 [ 4.29744977]
 [ 4.50977857]
 [ 3.73772419]
 [ 5.5253856 ]
 [ 6.24930783]
 [ 5.30440243]]
Residual: 522.137620538
Cross-Val Residual: 397.178242741
```

X3:

```
s: 1
Coeffs:
[[  6.45870259]
 [ -0.40370896]
 [  5.05268058]
 [  3.71693032]
 [  1.43151615]
 [  6.66021274]
 [  4.56317342]
 [ -1.27752969]
```

```
 [ 11.3552267 ]
 [  7.69731797]
 [  4.88503877]
 [  5.57929829]
 [  2.73294889]
 [  8.76322162]
 [  8.78680155]]
Residual: 475.645961536
Cross-Val Residual: 410.998376688

s: 5
Coeffs:
[[  6.27868637]
 [ -0.06660617]
 [  5.02406276]
 [  3.53713162]
 [  1.82778246]
 [  6.42702911]
 [  4.77829747]
 [ -0.92188075]
 [ 10.92920984]
 [  7.78879439]
 [  4.91708242]
 [  5.49894605]
 [  2.89687172]
 [  8.45265813]
 [  8.49147131]]
Residual: 475.944598146
Cross-Val Residual: 407.57419476

s: 10
Coeffs:
[[ 5.86637883]
 [ 0.72102376]
 [ 4.97519893]
 [ 3.26225789]
 [ 2.59429263]
 [ 5.9854219 ]
 [ 5.16685421]
 [-0.02101353]
 [ 9.98082325]
 [ 7.87560346]
 [ 4.97394388]
 [ 5.34183765]
 [ 3.21268114]
 [ 7.74422331]
 [ 7.81301552]]
Residual: 478.784353367
Cross-Val Residual: 402.915480514

s: 15
Coeffs:
[[ 5.46264336]
```

```
 [ 1.5453729 ]
 [ 4.94256459]
 [ 3.1651814 ]
 [ 3.21339589]
 [ 5.63982704]
 [ 5.4285438 ]
 [ 1.00105433]
 [ 9.01859406]
 [ 7.76482718]
 [ 5.00330597]
 [ 5.19691079]
 [ 3.47691811]
 [ 7.02257943]
 [ 7.11930154]]
Residual: 485.368629901
Cross-Val Residual: 402.431269017

s: 20
Coeffs:
[[ 5.16005715]
 [ 2.22597427]
 [ 4.92312986]
 [ 3.21762771]
 [ 3.62800114]
 [ 5.41024432]
 [ 5.54357378]
 [ 1.88495719]
 [ 8.20741462]
 [ 7.49191851]
 [ 4.99482846]
 [ 5.06487846]
 [ 3.67249466]
 [ 6.43042945]
 [ 6.55412518]]
Residual: 494.747147116
Cross-Val Residual: 405.598348678

s: 25
Coeffs:
[[ 4.95102427]
 [ 2.73612091]
 [ 4.90751975]
 [ 3.33428309]
 [ 3.90089088]
 [ 5.24867512]
 [ 5.56331582]
 [ 2.56284945]
 [ 7.55656662]
 [ 7.14729996]
 [ 4.95543534]
 [ 4.93780961]
 [ 3.81803587]
 [ 5.9724822 ]
```

```
 [ 6.12435307]]
Residual: 506.015589342
Cross-Val Residual: 411.150272789

s: 30
Coeffs:
[[ 4.80388388]
 [ 3.10151747]
 [ 4.88867011]
 [ 3.46029114]
 [ 4.08145032]
 [ 5.12085488]
 [ 5.52840547]
 [ 3.05522306]
 [ 7.03703536]
 [ 6.79314706]
 [ 4.89457867]
 [ 4.81462162]
 [ 3.924044  ]
 [ 5.6175743 ]
 [ 5.79790689]]
Residual: 518.987705246
Cross-Val Residual: 418.542552802
```

C: Now we can compare the cross validated residuals to see which performed the best. From above we know the average residual across the subsets for least squares was 442.18991. After some averaging across the subsets for each s. we can find the best s i.e. the one that has the smallest cross validation residual. That value is $s = 20$ with a residual of 417.45704. So Ridge Regression with $s = 20$ has the best generalization out of all the types across each subset.

2 Orthogonal Matching Pursuit

Description:

Consider a linear equation $W = MS$ where M is a measurement matrix filled with random values $\{-1, 0, +1\}$, and W is the output of the sparse signal S when measured by M. Use Orthogonal Matching Pursuit to recover the non-zero entries from S. Record the order in which you find each entry and the residual vector after each step.

Solution:

We compute the entries in the following manner. If all of the entries in the residual are zero we stop.

```
r = W
s = np.zeros(100)
for _ in range(1000):
    j = np.argmax(r.T.dot(M))
    s[j] = 1
    r = r - np.reshape(M[:,j], (80,1))

    print('Epoch: ' + str(_))
    print('Entry: ' + str(j))
```

```
    print(np.linalg.norm(r))
    print()
    if not np.count_nonzero(r):
        break

print(s)
```

With the following output:

```
Epoch: 0
Entry: 42
14.5258390463

Epoch: 1
Entry: 65
11.4455231423

Epoch: 2
Entry: 15
9.2736184955

Epoch: 3
Entry: 64
6.7082039325

Epoch: 4
Entry: 88
0.0

[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

End

```
Counter({'a': 330000, 'b': 600000, 'c': 450000})
```

and the output of the interpretation is:

```
'a' Maybe
'c' Maybe
'b' No.
```

For `S2` the counts are:

```
Counter({'a': 865008,
         'b': 371334,
         'c': 572379,
         'g': 1,
         'p': 1,
         'q': 1,
         'x': 1})
```

and the output of the interpretation is:

```
'a' must occur more than 20% of the time
'c' Maybe
'b' No.
'x' No.
'q' No.
'p' No.
'g' No.
```

B: After building the Count-Min Sketch for `S1` the estimated counts for `a,b,c` are:

```
a : 557077
b : 826791
c : 677106
```

and the output of the interpretation is:

```
'a' No
'b' Maybe
'c' Maybe
```

For `S2` the counts for `a,b,c` are:

```
a : 1235102
b : 868829
c : 943539
```

and the output of the interpretation:

```
'a' Maybe
'b' Maybe
'c' Maybe
```

C: If the words are separated by spaces and the sub-sequent tweets were also separated by a space then we would just treat each object as a word instead of a character. The interpretation would be the same and the algorithms would not need to change. The way we read the stream would be slightly different. We would need to make sure that we are treating each word as an object in the algorithms and we must ignore white space, punctuation, stop words, etc. Because these items would have a large amount of frequency.

D: The advantage of the Count-Min Sketch is that we can updated it at any time. Meaning that if we want to subtract a frequency from it we can. This requires additional implementation, but is not difficult.

End