HW4

February 2, 2025

[168]:
```
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
```

1 Problem 6a and 6b

My strategy would be to use Lasso regression to create a sparse model, and use grid search to find the best value of alpha that gives us only ten non-zero coefficients. Fortunately for me, the first value of alpha I used, 0.5, returned a model with exactly 10 coefficients with indexes: [1, 2, 4, 6, 10, 12, 16, 18, 22, 26]

[176]:
```
mys_df = pd.read_csv('mystery.dat', header=None)
#print(mys_df)
#len(mys_df)
X, y = mys_df.iloc[:, :-1], mys_df.iloc[:, -1]
display(mys_df)
model = Lasso(alpha=0.5)
model.fit(X, y)
y_pred = model.predict(X)
lasso_mse = mean_squared_error(y, y_pred)
print(lasso_mse)
print(model.sparse_coef_)
```

```
          0        1        2        3        4        5        6        7   \
0   0.63311 -1.71313 -0.48056 -0.32540 -0.05102  0.05634 -1.63462 -0.58081
1   0.82710 -0.45099  0.62209 -0.24694  0.53069  0.84492  0.37463 -0.61650
2  -0.25135 -0.22821 -0.65147  0.52365 -0.58971  0.02787  0.27812  0.21289
3   0.46192  0.16546  2.87388 -0.65411  0.76601  1.54346 -1.08101 -1.00728
4  -1.50107  2.05339  0.03820  0.27116 -0.07920 -0.53648  0.32249 -0.57844
..      …        …        …        …        …        …        …        …
```

```
96  -0.68050 -0.43706  1.08467 -0.08860 -0.75584 -0.51020 -0.12227  0.76625
97  -0.04866  0.29133 -1.72828 -0.45218  0.59248  0.51923  0.18951 -0.91447
98   0.85169 -0.63504 -1.96589  0.04994  0.98394  0.96469 -0.67908 -0.90963
99   0.50464  1.18625 -0.54312  0.11166 -0.40399 -0.63077 -1.34732 -0.97678
100  0.62023  0.99255 -0.83880 -2.00780 -1.50414 -0.77133 -1.58600 -0.90523

         8        9   …       91       92       93       94       95  \
0    0.70627 -2.06938  …  0.69346  0.49371 -0.15578  1.02650  0.48640
1   -0.00887  0.51328  …  1.12702  0.53821  1.69800  0.65812  0.18004
2    1.08754 -0.28801  … -0.08484  0.00259  1.98580  0.39629 -1.37305
3   -0.01883  0.41995  …  0.72389  1.27442 -0.69487  0.47128  0.54426
4   -0.66211 -0.73749  …  0.07916 -0.34523  1.09813  1.78102 -1.06170
..        …        …  …        …        …        …        …        …
96   0.09844  0.79460  …  1.39146 -2.31016  0.34400  0.46904  1.13855
97   0.36181 -0.20676  …  1.02579 -1.11850  1.11086 -1.74939  1.30350
98  -0.98677 -0.85608  … -0.48633  0.45576  0.35485 -0.95341  0.47911
99  -0.62806 -0.23150  … -0.38290  0.23060 -1.06765 -0.12734 -0.94222
100 -1.25421  0.86554  … -0.85442  1.39371 -0.91005  0.74747 -0.08224

         96       97       98       99      100
0    0.32758 -2.28887 -0.00430 -0.39673 -6.07560
1    0.31097  0.42096  0.43610 -0.09575  4.03525
2    1.66343 -1.25645 -0.41212  0.78800 -3.57768
3    1.18577  0.00389  0.90909  1.44143 -1.01789
4   -1.74101  1.96249 -0.86213 -1.88139  4.54025
..        …        …        …        …        …
96   0.54741  2.63132 -1.86386 -0.01147 -2.87495
97   1.47213  0.83292 -1.64844  0.99595  4.22825
98  -0.92630  1.76321 -0.05456 -0.40289 -4.60459
99   0.94762 -1.79318  0.16694 -1.12890 -2.31289
100  0.18000 -0.65921  0.06566  0.15351 -1.24437

[101 rows x 101 columns]

3.1938312812617617
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 10 stored elements and shape (1, 100)>
  Coords        Values
  (0, 1)        0.5980707934148933
  (0, 2)        0.4905843803857638
  (0, 4)        0.6183269480966214
  (0, 6)        0.837668121698656
  (0, 10)       0.47713177855877054
  (0, 12)       0.3273188407108959
  (0, 16)       0.17272449651470947
  (0, 18)       0.37472883195505524
  (0, 22)       0.709102367009201
  (0, 26)       0.520022742355967
```

[60]:
```
lasso_cv = LassoCV(cv=5, random_state=0).fit(X,y)
y_pred_lassocv = lasso_cv.predict(X)
lassocv_mse = mean_squared_error(y, y_pred_lassocv)
print(lassocv_mse)
print(lasso_cv.coef_)
len(lasso_cv.coef_[lasso_cv.coef_ > 0.0])
```

```
0.8159516590379202
[-1.57049877e-02  9.38569808e-01  8.12176686e-01  0.00000000e+00
  1.02776529e+00  0.00000000e+00  1.02155363e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00  8.72073041e-01 -0.00000000e+00
  8.10093203e-01  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  7.18318297e-01 -0.00000000e+00  7.36031636e-01  0.00000000e+00
  0.00000000e+00  0.00000000e+00  9.32765601e-01 -0.00000000e+00
 -0.00000000e+00 -2.04996233e-02  8.48910599e-01  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -1.06169822e-06  2.90737359e-02 -6.53193565e-02
 -0.00000000e+00  0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -1.88131728e-02 -0.00000000e+00
 -0.00000000e+00 -3.56532521e-02  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -8.59642945e-02 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00 -6.71213698e-02
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  1.69587424e-02  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  5.45055943e-03  0.00000000e+00  1.76422497e-02
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.51213022e-02]
```

[60]: 15

[100]:
```
#tried elasticnet for both models out of curiosity, works better for this model
↪because there are many more dimensions
elastic_net = ElasticNet(alpha=1, l1_ratio=0.5)
elastic_net.fit(X, y)
print(elastic_net.sparse_coef_)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 12 stored elements and shape (1, 100)>
  Coords        Values
  (0, 1)        0.42428833583123365
  (0, 2)        0.35012726582270093
  (0, 4)        0.40577630597992737
```

```
  (0, 6)        0.5758411417682037
  (0, 10)       0.31793604890743876
  (0, 12)       0.23268248090680677
  (0, 16)       0.0968860196509212
  (0, 18)       0.2999168517355811
  (0, 22)       0.4938692386863265
  (0, 26)       0.3530058324492764
  (0, 31)       -0.026192151841342704
  (0, 59)       -0.0012496837920519788
```

[72]:

```python
heart_df = pd.read_csv('heart.csv')
print(heart_df)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   …  …  ..       …   …  …      …      …      …
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     …  ..   …     …
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

[186]:

```python
X, y = heart_df.iloc[:, :-1], heart_df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 103/303,␣
 ↪random_state = 42)
heart_l1_model = LogisticRegression(penalty='l1', solver='liblinear',␣
 ↪max_iter=1000).fit(X_train, y_train)
y_pred_1 = heart_l1_model.predict(X_test)
print(heart_model.coef_)
```

```
print(1-accuracy_score(y_test, y_pred_1))
```

```
[[ 0.02333912 -1.40812215  0.8659617  -0.01567635 -0.00180536  0.16733695
   0.10014146  0.03031501 -0.80671514 -0.74277557  0.27823236 -0.60748336
  -0.85121358]]
0.19417475728155342
```

[187]:
```
heart_l2_model = LogisticRegression(random_state=0, max_iter=1000).fit(X_train,␣
 ↪y_train)
y_pred_2 = heart_l2_model.predict(X_test)
print(heart_l2_model.coef_)
print(1-accuracy_score(y_test, y_pred_2))
```

```
[[ 2.19680208e-02 -1.09303757e+00  8.88157857e-01 -8.56182574e-03
  -8.56396046e-04  1.47484464e-01  4.99261077e-01  1.84834262e-02
  -8.60965028e-01 -4.81234703e-01  7.51554896e-01 -1.37301366e+00
  -1.21976904e+00]]
0.19417475728155342
```

[188]:
```
cv_heart_model = LogisticRegression(penalty='l1', solver='liblinear',␣
 ↪max_iter=1000)
kf = KFold(n_splits=5, shuffle=True, random_state=7)
scores = cross_val_score(cv_heart_model, X, y, cv=kf)
print(1-np.mean(scores))
```

```
0.16770491803278686
```

2 Problem 7

2.0.1 a. The three most influential features seem to be sex, ca, and thal.

2.0.2 b. The test error was 0.194.

2.0.3 c. The test error using 5-fold CV was 0.1677, lower than the test error of the original model.