Deng_Yehong_HW3

February 9, 2020

Conceptual exercises Training/test error for subset selection

1. (5 points) Generate a data set with $p = 20$ features, $n = 1000$ observations, and an associated quantitative response vector generated according to the model

$$Y = X\beta + \epsilon$$

where β has some elements that are exactly equal to zero.

[318]:
```python
import numpy as np
import pandas as pd
np.random.seed(1234)
X = []
X_dic = {}
for i in range(20):
    X.append(np.random.normal(0,10,1000))
X = np.array(X)
X_trans = X.transpose()
X_df = pd.DataFrame()
for j in range(20):
    X_df['X{}'.format(j + 1)] = X_trans[:,j]
X_df.head(10)
```

[318]:
```
          X1         X2         X3         X4         X5         X6  \
0   4.714352 -15.822080 -11.870412  -1.588084  -8.118981  13.777115
1 -11.909757 -16.201902  16.170816  -0.580519 -19.194430   0.188342
2  14.327070   0.465619  -0.426813  -9.282614  -7.877294  -8.008123
3  -3.126519 -16.798289   3.679837   9.123283  25.597029  13.160000
4  -7.205887  13.958923  18.091850   5.383625  -0.045414  13.788188
5   8.871629  -8.449714  10.200683 -18.196796   1.190739   1.185566
6   8.595884   8.140069  15.897211  12.015206 -10.235479   6.739554
7  -6.365235  -0.497423  12.381168   9.468057  21.754655  -1.627619
8   0.156964   5.342468  -9.765244  -8.685828   3.390515   8.602183
9 -22.426850  -8.070091  -8.877504  14.711871   5.751262  -6.098891

          X7         X8         X9        X10        X11        X12  \
0  15.073607 -11.755436 -15.214476 -13.876990   7.411468 -12.324561
1 -16.747214  -0.371710  -3.266494  13.514572   3.267239 -23.198860
```

```
2 -10.551765  -0.930554  22.322605  -0.609708   6.662411   0.665484
3  -9.630160  -5.746781  17.077355 -15.209515   0.635578  13.602440
4  13.263866   3.991317   3.777444  15.411191  -1.844548  -1.310472
5  14.236261  17.401515  -4.374805  18.710805   5.458347   3.631672
6  -3.315694  -2.381793   8.401552 -12.067569   8.506890  -3.257338
7  -3.752939   4.096260   3.723799  23.783793   1.116338   2.473971
8  -2.073415   5.011725  12.553320  -8.167761 -15.211489   5.551569
9  13.251207   1.872369   3.477125  -3.186239  16.350131   7.931130

         X13        X14        X15        X16        X17        X18  \
0  -4.445726  14.457552  10.410972   6.874028  -7.736147  -1.642443
1  11.510025  -9.394676 -14.393629   9.727797   5.871652   2.501816
2  -1.354627  -0.525251  -2.410696  12.862934 -12.287901   0.470752
3  10.019327  -2.050978 -18.042717 -22.482470   5.259837   4.149826
4  -4.060836  19.452012  -6.629941 -22.565776   8.458490  -5.159981
5 -12.187667  -3.817226  17.904804  -4.482575  -1.321668   1.595542
6  -5.755577  -9.026393  -3.248007   2.449215 -17.445764 -14.618446
7   8.803338  -3.222821  -0.872120  -1.799507 -17.113562   2.083616
8 -12.004537  10.365699  18.668098  -8.038104  -0.413990  14.508234
9  -7.410584  15.267762   3.198740  25.283099  14.291073  -1.032939

         X19        X20
0  11.779386 -14.083893
1  -7.256441  -0.178640
2 -11.232715 -24.652302
3  -6.586114   7.794075
4   0.107460  -9.391892
5  -3.745470 -25.400168
6 -17.548792   5.527113
7 -22.460044 -16.787309
8  -9.801665 -20.931211
9   1.118475 -18.696136
```

[315]:
```
beta = []
for i in range(20):
    beta.append(np.random.normal(0,1))
zeros_index = np.random.randint(1,5)
beta[:zeros_index] = np.zeros(zeros_index)
beta = np.array(beta)
```

[316]:
```
err = np.random.normal(0, 1, 1000)
Y = np.sum(X_df * beta, axis = 1) + err
```

2. (10 points) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

[197]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_df, Y, test_size = 0.9)
```

3. (10 points) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. For which model size does the training set MSE take on its minimum value?

[198]:
```
#referenced from http://www.science.smith.edu/~jcrouser/SDS293/labs/lab8-py.html
import statsmodels.api as sm
import itertools
def process(feature_set):
    model = sm.OLS(Y_train, X_train[list(feature_set)])
    regr = model.fit()
    RSS = ((regr.predict(X_train[list(feature_set)]) - Y_train)**2).sum()
    return {"model": regr, "RSS": RSS}

def get_Best(k):
    results = []
    for combo in itertools.combinations(X_train.columns, k):
        results.append(process(combo))
    models = pd.DataFrame(results)
    best_model = models.loc[models['RSS'].idxmin()]

    return best_model
```

[199]:
```
models_best = pd.DataFrame(columns = ["RSS", "model"])

for i in range(1,21):
    models_best.loc[i] = get_Best(i)
models_best
```

[199]:

	RSS	model
1	86037.446165	<statsmodels.regression.linear_model.Regressio…
2	61701.829039	<statsmodels.regression.linear_model.Regressio…
3	43480.718808	<statsmodels.regression.linear_model.Regressio…
4	34341.392798	<statsmodels.regression.linear_model.Regressio…
5	23760.619900	<statsmodels.regression.linear_model.Regressio…
6	18327.734084	<statsmodels.regression.linear_model.Regressio…
7	13266.209352	<statsmodels.regression.linear_model.Regressio…
8	9305.030454	<statsmodels.regression.linear_model.Regressio…
9	5370.299659	<statsmodels.regression.linear_model.Regressio…
10	2406.913806	<statsmodels.regression.linear_model.Regressio…
11	1493.478101	<statsmodels.regression.linear_model.Regressio…
12	607.218046	<statsmodels.regression.linear_model.Regressio…
13	136.316295	<statsmodels.regression.linear_model.Regressio…
14	74.079620	<statsmodels.regression.linear_model.Regressio…
15	70.235882	<statsmodels.regression.linear_model.Regressio…

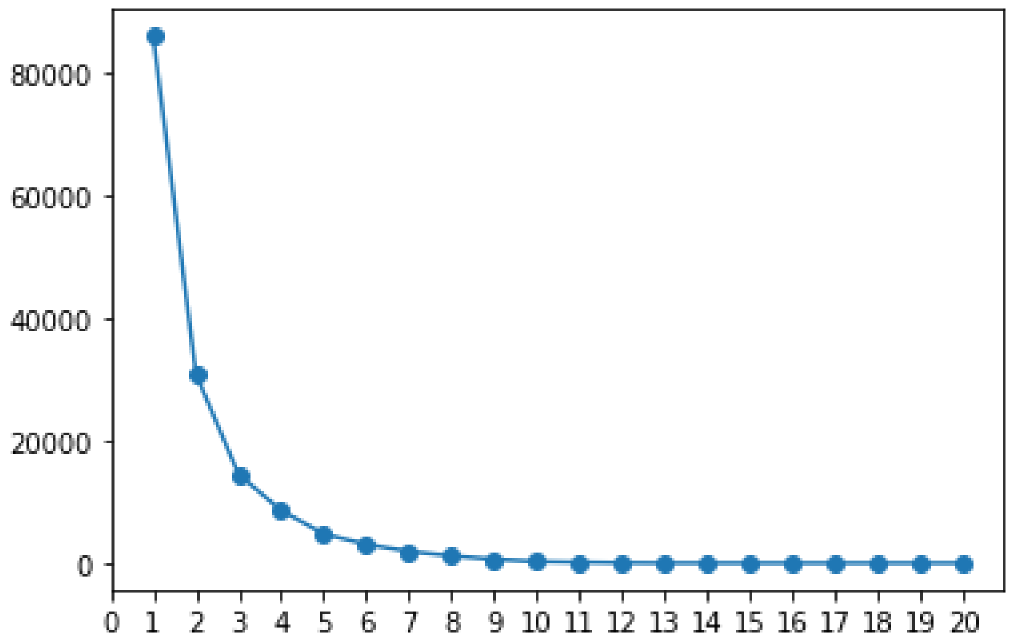
```
16    68.755639  <statsmodels.regression.linear_model.Regressio…
17    67.466786  <statsmodels.regression.linear_model.Regressio…
18    66.505504  <statsmodels.regression.linear_model.Regressio…
19    65.628327  <statsmodels.regression.linear_model.Regressio…
20    65.553434  <statsmodels.regression.linear_model.Regressio…
```

[200]:

```python
import matplotlib.pyplot as plt
mse = []
mse_dic = {}
for i in range(1, 21):
    m = models_best.loc[i,'RSS'] / i
    mse.append(m)
    mse_dic[str(i)] = m
plt.plot(models_best.index, mse, marker = 'o')
plt.xticks(np.arange(0,21))
mse_dic
```

[200]:

```
{'1': 86037.44616521144,
 '2': 30850.91451938499,
 '3': 14493.572935899669,
 '4': 8585.34819948579,
 '5': 4752.123979938449,
 '6': 3054.6223473145255,
 '7': 1895.1727645428007,
 '8': 1163.1288067704606,
 '9': 596.6999620770449,
 '10': 240.69138058482366,
 '11': 135.7707364513901,
 '12': 50.601503848197986,
 '13': 10.485868876468139,
 '14': 5.2914014374179486,
 '15': 4.68239210554819,
 '16': 4.2972274201286895,
 '17': 3.9686344479544404,
 '18': 3.694750215999465,
 '19': 3.4541224745429835,
 '20': 3.277671679278184}
```



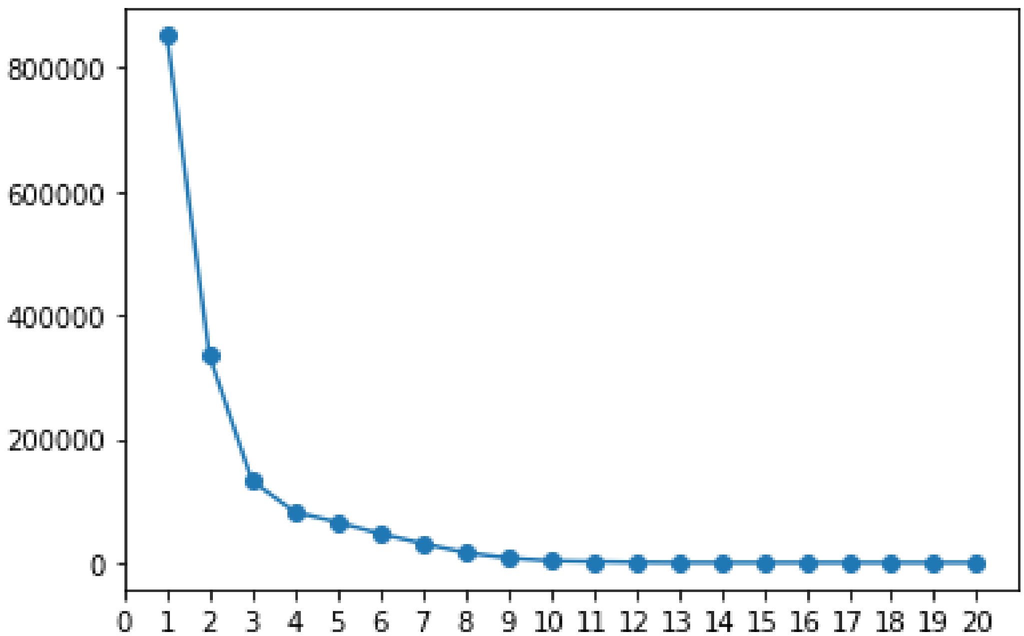
According to the dictionary the dictionary, for size 20, the training set MSE has its minimum 3.277671679278184

4. (5 points) Plot the test set MSE associated with the best model of each size.

[310]:
```
test_mse = []
test_mse_dic = {}
features = []
for i in range(1,21):
    model = models_best.loc[i, 'model']
    x_ls = list(dict(models_best.loc[i, 'model'].params).keys())
    RSS_test = ((model.predict(X_test[x_ls]) - Y_test)**2).sum()
    mse_t = RSS_test / i
    test_mse.append(mse_t)
    test_mse_dic[str(i)] = mse_t
plt.plot(np.arange(1,21), test_mse, marker = 'o')
plt.xticks(np.arange(0,21))
test_mse_dic
```

[310]:
```
{'1': 852684.4158056874,
 '2': 334328.6150933516,
 '3': 132234.3191547112,
 '4': 81974.98115865845,
 '5': 65353.836302870885,
 '6': 46878.94080937676,
 '7': 30930.84416089002,
```

```
'8': 16226.520687081165,
'9': 7482.4674704174995,
'10': 2854.9687703348054,
'11': 1752.8718711883462,
'12': 581.7808013162747,
'13': 125.72848803562664,
'14': 74.37567963240333,
'15': 65.81139805385595,
'16': 61.114155054945385,
'17': 58.728981658382224,
'18': 55.914483787437725,
'19': 53.25382578953771,
'20': 53.95246585974844}
```



5. (5 points) For which model size does the test set MSE take on its minimum value? Comment on your results.

[217]:
```
print(models_best.loc[19,'model'].params)
print(beta)
```

```
X1      0.011549
X3     -0.010122
X4      0.010002
X5     -1.699846
X6      1.670016
X7      0.754349
```

```
X8     -0.808657
X9      0.846491
X10     0.076937
X11    -1.391828
X12     0.832116
X13    -0.734209
X14     0.691957
X15    -0.018268
X16     0.305330
X17    -0.332443
X18     0.225389
X19    -0.015089
X20     0.608145
dtype: float64
[ 0.          0.          0.          0.         -1.69566881  1.66362859
  0.75788635 -0.80725066  0.84934261  0.07161185 -1.39349312  0.84343336
 -0.74838974  0.69696246 -0.02318777  0.31478762 -0.3274646   0.22589291
 -0.01273708  0.59368282]
```

According to the dictionary from 1.4, the model size 19 for the test set MSE takes on its minimum. Comapring the training set and the test set, it shows that there can be a discrapancy between the minimum MSEs for them. The model which size is 19 excludes the X2. As we can see in the beta list, the ture beta I have set for X2 is zero.

6. (10 points) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient sizes.

[319]:
```python
difference_dic = {}
beta_dic = {}
for i in range(20):
    beta_dic['X' + str(i + 1)] = beta[i]
#for j in range(1, 21):
    difference_dic["X" + str(j)] = 0
for i in list(dict(models_best.loc[19, 'model'].params).items()):
    difference_dic[i[0]] = beta_dic[i[0]] - i[1]
difference_dic
```

[319]:
```
{'X19': 0.0023523755286643435,
 'X1': -0.0115485105455225,
 'X3': 0.01012202713623285,
 'X4': -0.010002292535482904,
 'X5': 0.004177031275097054,
 'X6': -0.006387253165412998,
 'X7': 0.003537375611054494,
 'X8': 0.001406836796198685,
 'X9': 0.002851772762482274,
 'X10': -0.005325025250633497,
 'X11': -0.0016649767780980707,
```

```
 'X12': 0.011316876739103976,
 'X13': -0.014181010106822711,
 'X14': 0.005005018192484223,
 'X15': -0.004920234971970604,
 'X16': 0.009457246900911154,
 'X17': 0.004978417810654834,
 'X18': 0.0005037017191039384,
 'X20': -0.014462404294268483}
```

As we can see in the dictionary, the differences between the coeffcients and the true betas are very small. Moreover, in general, most coefficients are smaller than the true betas.

7. (10 points) Create a plot displaying

$$\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$$

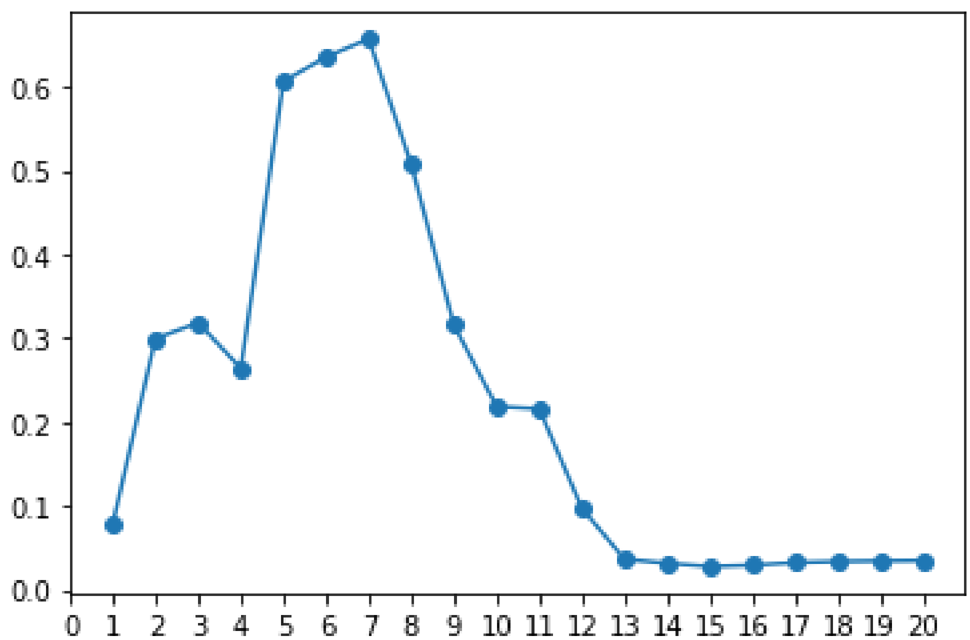
for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot?

[317]:
```python
beta_dic = {}
for i in range(20):
    beta_dic['X' + str(i + 1)] = beta[i]

def get_one_res(r):
    res = 0
    for i in list(dict(models_best.loc[r, 'model'].params).items()):
        diff = beta_dic[i[0]] - i[1]
        res += (diff ** 2)
    return (res ** 0.5)

res_ = [get_one_res(k) for k in range(1,21)]
print(res_)
plt.plot(np.arange(1,21), res_, marker = 'o')
plt.xticks(np.arange(0,21))
```

```
[0.07845350456507294, 0.2994134193967109, 0.3183885190260245,
0.2638965214331392, 0.6062287640840333, 0.6352724498744213, 0.6569724119938397,
0.5072234730491388, 0.31556827359214884, 0.218355518574315, 0.2154695920232972,
0.0969516406950457, 0.036155119546721695, 0.03125960036507476,
0.027445055542458074, 0.028979037097520005, 0.03204944072965362,
0.03331533770652615, 0.03399222592772646, 0.03452101219581696]
```

[317]:
```
([<matplotlib.axis.XTick at 0x2a16163ac08>,
  <matplotlib.axis.XTick at 0x2a161540f88>,
  <matplotlib.axis.XTick at 0x2a15525d208>,
  <matplotlib.axis.XTick at 0x2a16164e508>,
```

```
 <matplotlib.axis.XTick at 0x2a16164e148>,
 <matplotlib.axis.XTick at 0x2a16165aa48>,
 <matplotlib.axis.XTick at 0x2a16165ab08>,
 <matplotlib.axis.XTick at 0x2a1616523c8>,
 <matplotlib.axis.XTick at 0x2a1616527c8>,
 <matplotlib.axis.XTick at 0x2a161654948>,
 <matplotlib.axis.XTick at 0x2a168862808>,
 <matplotlib.axis.XTick at 0x2a161663ac8>,
 <matplotlib.axis.XTick at 0x2a161663c88>,
 <matplotlib.axis.XTick at 0x2a1688d2688>,
 <matplotlib.axis.XTick at 0x2a1688c8c48>,
 <matplotlib.axis.XTick at 0x2a161663e48>,
 <matplotlib.axis.XTick at 0x2a1616541c8>,
 <matplotlib.axis.XTick at 0x2a1688d5c08>,
 <matplotlib.axis.XTick at 0x2a1688e1148>,
 <matplotlib.axis.XTick at 0x2a14e21cc08>,
 <matplotlib.axis.XTick at 0x2a14e21c448>],
<a list of 21 Text xticklabel objects>)
```



Unlike the test MSE plot, this plot first increases then decrease monotonically. Moreover, this plot start to monotonically decrease when r > 8. This demonstrates while the MSE may decrease constantly as the size of subset increase, the bias may not start to decrease until the subset contain enough predictiors. moreover, as we can see in this plot as well as the printed residuals, the lowest residials exist when r = 15, but in the MSE graph, the minimum MSE took place when size = 19. This means that we should sometime sacrifice some bias to get better model with lower test error.

2 Application exercises

1. (10 points) Fit a least squares linear model on the training set, and report the test MSE.

[282]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
x_train = gss_train.drop(['egalit_scale'], axis = 1)
x_test = gss_test.drop(['egalit_scale'], axis = 1)
```

[290]:
```
#referenced from https://scikit-learn.org/stable/supervised_learning.
↪html#supervised-learning
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import mean_squared_error as mse
```

[291]:
```
OLS = LinearRegression().fit(x_train, y_train)
MSE = mse(OLS.predict(x_test), y_test)
print('The test MSE for the least squares linear model is', MSE)
```

```
The test MSE for the least squares linear model is 63.21362962301501
```

2. (10 points) Fit a ridge regression model on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE.

[293]:
```
ridge = RidgeCV(alphas = (0.1, 1, 10), cv = 10).fit(x_train, y_train)
MSE = mse(ridge.predict(x_test), y_test)
print('The test MSE for the ridge regression model is', MSE)
```

```
The test MSE for the ridge regression model is 62.49920243957809
```

3. (10 points) Fit a lasso regression on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE, along with the number of non-zero coefficient estimates.

[294]:
```
lasso = LassoCV(alphas = (0.1, 1, 10), cv = 10).fit(x_train, y_train)
MSE = mse(lasso.predict(x_test), y_test)

print('The test MSE for the lasso regression model is', MSE)
```

```
The test MSE for the lasso regression model is 62.7784155547739
```

[306]:
```
print("the number of non-zero coefficients is",(lasso.coef_ != 0).sum())
```

```
the number of non-zero coefficients is 24
```

4. (10 points) Fit an elastic net regression model on the training set, with α and λ chosen by 10-fold cross-validation. That is, estimate models with $\alpha = 0, 0.1, 0.2, \ldots, 1$ using the same values for λ across each model. Select the combination of α and λ with the lowest

cross-validation MSE. For that combination, report the test MSE along with the number of non-zero coefficient estimates.

[303]:
```
#hide warnings referenced from https://stackoverflow.com/questions/9031783/
↪hide-all-warnings-in-ipython
import warnings
warnings.filterwarnings('ignore')
elastic_net = ElasticNetCV(l1_ratio = .1, alphas = np.arange(0,1,0.1), cv = 10).
↪fit(x_train, y_train)
MSE =  mse(elastic_net.predict(x_test), y_test)
print("The test MSE for the elastic net regression model is", MSE)
```

```
The test MSE for the elastic net regression model is 62.467338809713716
```

[307]:
```
print("the number of non-zero coefficients is",(elastic_net.coef_ != 0).sum())
```

```
the number of non-zero coefficients is 68
```

5. (5 points) Comment on the results obtained. How accurately can we predict an individual's egalitarianism? Is there much difference among the test errors resulting from these approaches?

In general, we can predict an individual's egalitarianism at a MSE that close to 62.5.In other words, there is not much difference among the test errors from different approaches. More specifically, though, among different approaches, the least squares linear model has the highest MSE at 63.21362962301501, and the eastic net regression model has the lowest 62.467338809713716. Moreover, when we compare lasso regression with elastic net regression, the lasso model has fewer non-zero coefficients, 24, than the elastic net model, 68.