30100HW3

February 8, 2020

[152]:
```python
import numpy as np
import itertools
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNetCV
import math
```

1 Conceptual exercises

1.1 1

[4]:
```python
np.random.seed(1234)
```

[48]:
```python
ds = []
for _ in range(20):
    ds.append(np.random.normal(0, 10, 1000))
ds = pd.DataFrame(ds).transpose()
rename_dict = {}
for i in range(20):
    rename_dict[i] = f'x{i+1}'
ds = ds.rename(columns=rename_dict)
ds
```

[48]:
```
            x1         x2         x3         x4         x5         x6  \
0     5.005098  -3.079597  -6.736742  -9.208900  17.163524   2.679199
1   -23.916511  -8.349183   8.364086  -3.255462   1.983732   2.690373
2    -2.688546  -0.941587  -1.934186  -1.875205  -6.867739  -6.585818
3   -21.432797 -12.918548  -4.963914  -3.189294 -10.652574   1.542849
4    20.115298 -15.521232  11.820163 -12.628821   5.766747  -2.701025
..         …          …          …          …          …          …
```

```
995   6.255667  -7.334920   4.010327 -15.985313   3.790038  10.618489
996  10.105843  -2.217488  -7.195990  -3.874800 -12.369471  -4.999170
997  -4.877899   0.470569 -10.988603   5.314530   4.851118   4.393106
998  -3.074163  -7.765926  -8.789487   5.664828   5.228831   8.439872
999   1.651616 -10.461592  -9.229005   2.196724  13.691822  -4.487877

            x7         x8        x9        x10        x11        x12  \
0     4.718801  14.519283 -1.322301 -14.801647  -0.196367   6.485250
1     6.617009   6.221157 -3.880052  12.562306   3.021221 -16.964511
2     1.965547  25.181092  1.729895  -3.274528 -10.170728  -4.940939
3     0.236012   3.734927 -0.722333  10.166517 -14.101019   4.560306
4   -26.380349  -8.906178 -8.403334  -9.211666   7.796728  -3.623239
..         …          …         …          …          …          …
995   0.678319   4.362874 -1.420523  -0.474641  -9.520225 -12.767077
996 -14.624940  -2.325923 -7.209664 -19.349156 -25.102776  -4.646223
997   1.058657  -8.959477  4.329335   3.340279 -19.169683  11.370569
998   0.784879  -0.305834 -2.321722  -0.674759   9.398248   2.967133
999   0.540586   2.419024  1.592510   6.560977  -1.908010   6.040977

           x13        x14        x15        x16        x17        x18  \
0     8.134716 -24.535596  14.247538  24.306004   0.069961 -15.330970
1   -13.999435  -2.246745   3.662489  -6.339808   4.829360   7.180375
2   -12.454654 -19.363850  -8.435162   2.229563 -22.446687  -1.573028
3   -12.795223   5.721978  -4.179870 -11.481876   4.492177 -17.443767
4    -3.081797  -0.711870 -10.933869   7.924502  13.960252  18.186273
..         …          …          …          …          …          …
995 -20.172819  -3.954662  16.274452   8.340085   7.391269   7.410582
996   5.548532   6.945687   9.234932   3.963585  -3.810432 -15.907529
997   0.512573   5.496581   4.270261   1.547216  -2.432065  10.344513
998  14.815367  10.325031   9.325506   5.689447   2.816422   0.682594
999  -9.791717   7.452824  -1.588970   6.332869 -16.878943  -9.474782

           x19        x20
0     4.774943 -21.887444
1     2.364700 -11.155318
2    -4.863731   2.178076
3   -11.859413   6.450934
4   -15.750461  -0.168575
..         …          …
995   6.650144   8.809316
996  -8.002547   0.170180
997  -1.361390   5.731568
998  -1.879308  15.362828
999   1.056359   1.375688

[1000 rows x 20 columns]
```

[49]:
```
beta = np.random.normal(0, 1, 20)
beta0 = np.random.randint(11)
beta[:beta0] = np.zeros(beta0)
beta
```

[49]:
```
array([ 0.        ,  0.        ,  0.        ,  0.        ,  1.58543276,
       -1.01818822, -0.11834515,  0.11149148, -0.39270171, -0.186964  ,
       -0.39819717, -0.1118996 , -0.28438328, -0.44222926,  0.29220768,
       -0.46236187, -0.77583907, -0.47675005,  0.10803055,  0.57764624])
```

[50]:
```
theta = np.random.normal(0, 1, 1000)
y = (ds * beta).sum(axis=1) + theta
y
```

[50]:
```
0      24.037575
1      -4.599835
2      31.262268
3      -1.899849
4      -9.211562
         …
995     5.546280
996    11.032350
997     3.569553
998    -6.732175
999    38.727093
Length: 1000, dtype: float64
```

1.2 2

[51]:
```
x_train, x_test, y_train, y_test = train_test_split(ds, y, test_size = 0.9)
```

1.3 3

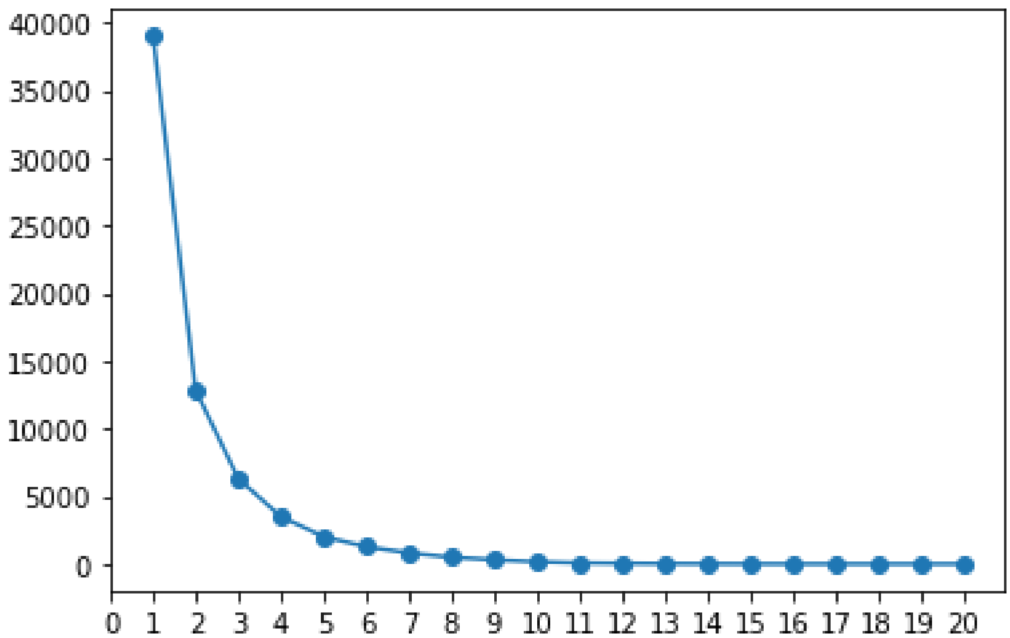
[54]:
```
def process_subset(feature_set):
    model = sm.OLS(y_train, x_train[list(feature_set)])
    regr = model.fit()
    RSS = ((regr.predict(x_train[list(feature_set)]) - y_train) ** 2).sum()
    return {"model":regr, "RSS":RSS}

def get_best(k):
    results = []
    for combo in itertools.combinations(x_train.columns, k):
        results.append(process_subset(combo))
    models = pd.DataFrame(results)
    best_model = models.loc[models['RSS'].idxmin()]
    return best_model
```

[55]:
```
models_best = pd.DataFrame(columns=["RSS", "model"])

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

models_best
```

[55]:

	RSS	model
1	39010.066387	<statsmodels.regression.linear_model.Regressio…
2	25669.268951	<statsmodels.regression.linear_model.Regressio…
3	18953.355559	<statsmodels.regression.linear_model.Regressio…
4	13993.382006	<statsmodels.regression.linear_model.Regressio…
5	10100.344049	<statsmodels.regression.linear_model.Regressio…
6	7650.588640	<statsmodels.regression.linear_model.Regressio…
7	5694.414089	<statsmodels.regression.linear_model.Regressio…
8	3938.579019	<statsmodels.regression.linear_model.Regressio…
9	2769.514440	<statsmodels.regression.linear_model.Regressio…
10	1595.456447	<statsmodels.regression.linear_model.Regressio…
11	811.859166	<statsmodels.regression.linear_model.Regressio…
12	588.054369	<statsmodels.regression.linear_model.Regressio…
13	354.491249	<statsmodels.regression.linear_model.Regressio…
14	237.277909	<statsmodels.regression.linear_model.Regressio…
15	152.146899	<statsmodels.regression.linear_model.Regressio…
16	90.742581	<statsmodels.regression.linear_model.Regressio…
17	88.705877	<statsmodels.regression.linear_model.Regressio…
18	87.308638	<statsmodels.regression.linear_model.Regressio…
19	87.302342	<statsmodels.regression.linear_model.Regressio…
20	87.300702	<statsmodels.regression.linear_model.Regressio…

[87]:
```
mse = []
for i in range(20):
    mse.append(models_best.loc[i + 1, 'RSS'] / (i + 1))

plt.plot(models_best.index, mse, marker='o')
plt.xticks(np.arange(0, 21));
```



[116]:
```
print(f'For size {np.argmin(mse) + 1} the training set MSE takes on its minimum␣
↪{min(mse)}')
```

```
For size 20 the training set MSE takes on its minimum 4.365035104234773
```

1.4 4

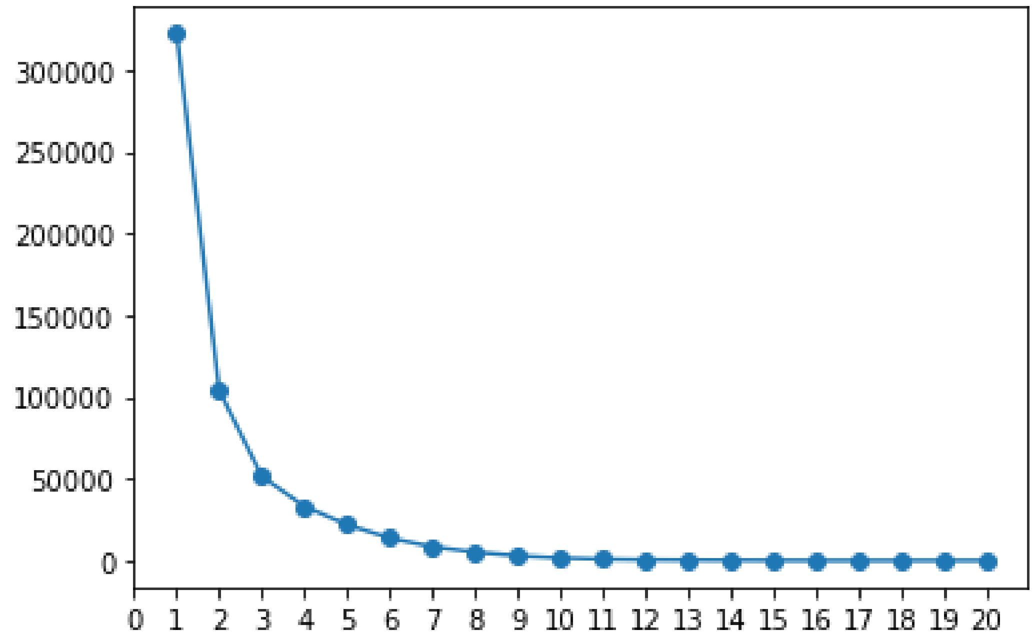
[113]:
```
mse_test = []
for i in range(20):
    mse_test.append(((models_best.loc[i+1, 'model'].predict(
        x_test[list(dict(models_best.loc[i+1, 'model'].params).keys())]) -␣
↪y_test) ** 2).sum() / (i + 1))
```

[114]:
```
mse_test
```

[114]:
```
[322114.03239958896,
 103934.3681542287,
 51539.097651151016,
 33112.99395260199,
 21923.160781032428,
 13778.95883531264,
 8433.08107160877,
 5174.51892631219,
 2951.8692869078586,
 1615.2732672489935,
 858.3144106057827,
```

```
494.17147943308504,
378.8235378295103,
251.20330792464742,
153.03066860356512,
58.48921158184273,
56.01204254900738,
53.930049750927964,
51.06975852913004,
51.53097848425839]
```

[115]:
```
plt.plot(np.arange(1, 21), mse_test, marker='o')
plt.xticks(np.arange(0, 21));
```



1.5 5

[117]:
```
print(f'For size {np.argmin(mse_test) + 1} the test set MSE takes on its␣
↪minimum {min(mse_test)}')
```

```
For size 19 the training set MSE takes on its minimum 51.06975852913004
```

This result shows that we cannot select the model only based on MSE of the training set, which leads to the problem of overfitting. According to the test set MSE, whose size is much larger than the training set, we should choose the model with 19 features.

1.6 6

[121]:
```
models_best.loc[19, 'model'].params
```

[121]:
```
x1      0.012490
x2      0.000884
x4     -0.014487
x5      1.588279
x6     -1.012021
x7     -0.128142
x8      0.121791
x9     -0.407448
x10    -0.167499
x11    -0.398141
x12    -0.095428
x13    -0.283046
x14    -0.445094
x15     0.288721
x16    -0.457373
x17    -0.769812
x18    -0.474541
x19     0.105706
x20     0.572873
dtype: float64
```

[119]:
```
for i in range(20):
    print(f'x{i+1}', beta[i])
```

```
x1 0.0
x2 0.0
x3 0.0
x4 0.0
x5 1.5854327611453554
x6 -1.0181882190138873
x7 -0.11834515435708044
x8 0.11149147872040578
x9 -0.3927017056183946
x10 -0.1869640022032156
x11 -0.3981971746582488
x12 -0.11189960269111264
x13 -0.2843832825990808
x14 -0.4422292576079531
x15 0.29220768468032265
x16 -0.4623618673589449
x17 -0.7758390748967039
x18 -0.47675005094706435
x19 0.10803054514846445
x20 0.5776462358381345
```

[122]:
```
coef = list(models_best.loc[19, 'model'].params)
coef.insert(2, 0)
for i in range(20):
    print(f'x{i+1}', abs(coef[i] - beta[i]))
```

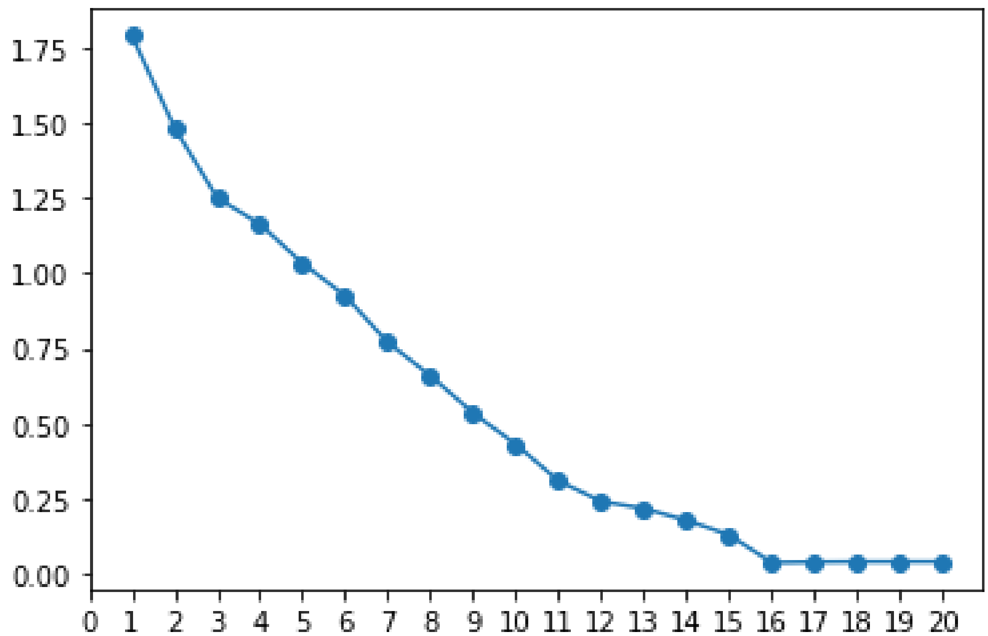
```
x1 0.012490251075332241
x2 0.0008842819533762397
x3 0.0
x4 0.014486753650521066
x5 0.0028467091819182055
x6 0.006167020853428129
x7 0.00979687116087867
x8 0.010299417082740625
x9 0.014746060126698046
x10 0.0194649979888947
x11 5.573156781479849e-05
x12 0.016471464974746372
x13 0.001337146941851386
x14 0.002864278816503696
x15 0.003486551493197265
x16 0.0049892814509169825
x17 0.006027032463996385
x18 0.0022086060737820934
x19 0.002324623861825831
x20 0.004773623958508333
```

According to the comparison of the two arrays of coefficients, we can find that the selected model's estimation of coefficients is very close to the true model, with three coefficients assigned wrong significance but small values which are actually close to zero. And the largest difference between the estimated value and the true value is roughly 0.01, which is rather small compared with the degree of the value of coefficients.

1.7 7

[136]:
```
beta_dict = {}
for i in range(20):
    beta_dict[f'x{i+1}'] = beta[i]

def deviation_coef(r):
    sub_dict = {key: beta_dict[key] - dict(models_best.loc[r, 'model'].params).
 ↪get(key, 0)
                       for key in beta_dict.keys()}
    res = 0
    for i in sub_dict.values():
        res += i ** 2
    return math.sqrt(res)

res = [deviation_coef(i) for i in range(1, 21)]
```

```
plt.plot(np.arange(1, 21), res, marker='o')
plt.xticks(np.arange(0, 21));
```



This plot's decreasing trend is much more smooth than the one of the test set MSE, which drops shaply when r = 2 and 3. This difference shows that when the size of coefficients increases approaching the size of the true model, the improvement on MSE gradually decreases, with the first additions of coefficients much more significant , whereas the improvement on the squared sum of the errors of coefficients is relatively stable. Furthermore, the trend of the squared sum of the errors of coefficients do not significantly decrease until it reaches the true size of coefficients, which renders the plot more informative compared with the test MSE plot.

2 Application exercises

[138]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
gss_train.head()
```

[138]:
```
   age  attend  authoritarianism  black  born  childs  colath  colrac  colcom  \
0   21       0                 4      0     0       0       1       1       0
1   42       0                 4      0     0       2       0       1       1
2   70       1                 1      1     0       3       0       1       1
3   35       3                 2      0     0       2       0       1       0
4   24       3                 6      0     1       3       1       1       0

   colmil  …  zodiac_GEMINI  zodiac_CANCER  zodiac_LEO  zodiac_VIRGO  \
```

```
0      1  ...          0            0          0           0
1      0  ...          0            0          0           0
2      0  ...          0            0          0           0
3      1  ...          0            0          0           0
4      0  ...          0            0          0           0

   zodiac_LIBRA  zodiac_SCORPIO  zodiac_SAGITTARIUS  zodiac_CAPRICORN  \
0             0               0                   0                 0
1             0               0                   0                 0
2             0               0                   0                 0
3             0               1                   0                 0
4             0               1                   0                 0

   zodiac_AQUARIUS  zodiac_PISCES
0                0              0
1                0              0
2                0              0
3                0              0
4                0              0

[5 rows x 78 columns]
```

[139]:
```
x_train = gss_train.drop(['egalit_scale'], axis=1)
x_test = gss_test.drop(['egalit_scale'], axis=1)
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
```

2.1 1

[141]:
```
lr = LinearRegression().fit(x_train, y_train)
lr_mse = mean_squared_error(lr.predict(x_test), y_test)
print('the test MSE of least squares linear: ', lr_mse)
```

```
the test MSE of least squares linear:  63.213629623014974
```

2.2 2

[144]:
```
ridge = RidgeCV(cv=10).fit(x_train, y_train)
ridge_mse = mean_squared_error(ridge.predict(x_test), y_test)
print('the test MSE of ridge: ', ridge_mse)
```

```
the test MSE of ridge:  62.499202439578106
```

2.3 3

```
[146]: lasso = LassoCV(cv=10).fit(x_train, y_train)
lasso_mse = mean_squared_error(lasso.predict(x_test), y_test)
print('the test MSE of lasso: ', lasso_mse)
```

```
the test MSE of lasso:  62.7780157899344
```

```
[150]: print('the number of non-zero coefficient estimates is: ', lasso.coef_.size -␣
 ↪list(lasso.coef_).count(0))
```

```
the number of non-zero coefficient estimates is:  24
```

2.4 4

```
[158]: import warnings
warnings.filterwarnings("ignore")
alpha = np.arange(0, 1.1, step=0.1)
en = ElasticNetCV(cv=10, alphas=alpha).fit(x_train, y_train)
en_mse = mean_squared_error(en.predict(x_test), y_test)
print('l1 ratio: ', en.l1_ratio_)
print('alpha: ', en.alpha_)
print('the test MSE of elastic net: ', en_mse)
```

```
l1 ratio:  0.5
alpha:  0.1
the test MSE of elastic net:  62.507086087221204
```

2.5 5

There is no significant difference among the test errors from these approaches, which are all very close to 62.5, with the test MSE of the linear regression model a bit higher. From my perspective, the MSE of 62.5 is not low enough for an accurate prediction of an individual's egalitarianism, considering that the range of the egalitarianism scale is from 1 to 35.