Hu_Chun_HW3

February 9, 2020

[457]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV,␣
↪ElasticNetCV
from sklearn.metrics import mean_squared_error as mse
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
```

0.1 Conceptual Exercises

0.1.1 Training/test error for subset selection

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.

[427]:
```
np.random.seed(123)
X = [np.random.normal(0, 1, 1000) for n in range(20)]
X = np.array(X)

beta = [np.random.randint(0, 5) for n in range(20)]
beta = np.array(beta)

beta = beta.reshape(-1, 1)

err = np.random.normal(0, 1, 1000)

y = np.sum(X*beta, axis=0) + err
```

Split your data set into a training set containing 100 observations and a test set containing 900 observations.

[428]:
```
X = X.transpose()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=900)
```

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?

[429]:
```
model = LinearRegression()

sfs_sub = sfs(model,
              k_features=5,
              forward=True,
              scoring='neg_mean_squared_error',
              cv=5)
```

[430]:
```
sfs_sub.fit(X_train, y_train)
```

[430]:
```
SequentialFeatureSelector(clone_estimator=True, cv=5,
                          estimator=LinearRegression(copy_X=True,
                                                     fit_intercept=True,
                                                     n_jobs=None,
                                                     normalize=False),
                          fixed_features=None, floating=False, forward=True,
                          k_features=5, n_jobs=1, pre_dispatch='2*n_jobs',
                          scoring='neg_mean_squared_error', verbose=0)
```

[431]:
```
sfs_sub.k_feature_idx_
```

[431]:
```
(2, 7, 11, 15, 17)
```

[432]:
```
num = []
score = []
for i in range(1, 21):
    sfs_fit = sfs(model,
                  k_features=i,
                  forward=True,
                  scoring='neg_mean_squared_error',
                  cv=5)
    sfs_fit.fit(X_train, y_train)
    num.append(i)
    score.append(-sfs_fit.k_score_)

feature_mse = pd.DataFrame({"feature_num": num, "mse": score})
feature_mse
```

[432]:

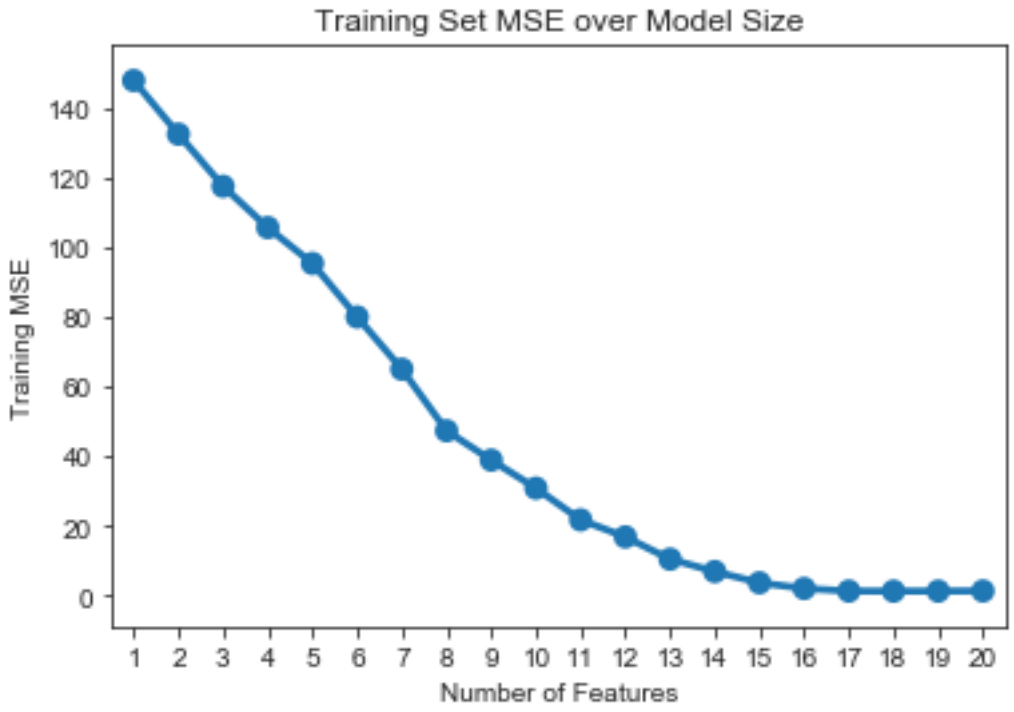
	feature_num	mse
0	1	147.935585
1	2	132.533102
2	3	117.540291
3	4	105.679444
4	5	95.365177
5	6	79.993773
6	7	64.976995
7	8	47.281677

```
8              9   38.845797
9             10   30.776950
10            11   21.666638
11            12   16.768962
12            13   10.383862
13            14    6.813748
14            15    3.588932
15            16    1.945406
16            17    1.205019
17            18    1.183350
18            19    1.198415
19            20    1.241520
```

Training set MSE reaches the lowest value at model size equals to 18.

[433]:
```
ax = sns.pointplot(x="feature_num", y="mse", data=feature_mse)
ax.set(xlabel='Number of Features', ylabel='Training MSE', title='Training Set␣
↪MSE over Model Size')
```

[433]:
```
[Text(0, 0.5, 'Training MSE'),
 Text(0.5, 0, 'Number of Features'),
 Text(0.5, 1.0, 'Training Set MSE over Model Size')]
```



Plot the test set MSE associated with the best model of each size.

[434]:
```
test_num = []
test_score = []
```

```
feature_idx = []
for i in range(1, 21):
    sfs_fit = sfs(model,
                  k_features=i,
                  forward=True,
                  scoring='neg_mean_squared_error',
                  cv=5)
    sfs_fit.fit(X_train, y_train)

    lm = model.fit(X_train[:, sfs_fit.k_feature_idx_], y_train)
    test_err = mse(lm.predict(X_test[:, sfs_fit.k_feature_idx_]), y_test)

    test_num.append(i)
    test_score.append(test_err)
    feature_idx.append(list(sfs_fit.k_feature_idx_))

test_mse = pd.DataFrame({"feature_num": num, "mse": test_score, "feature index":
 ↪ feature_idx})
test_mse
```

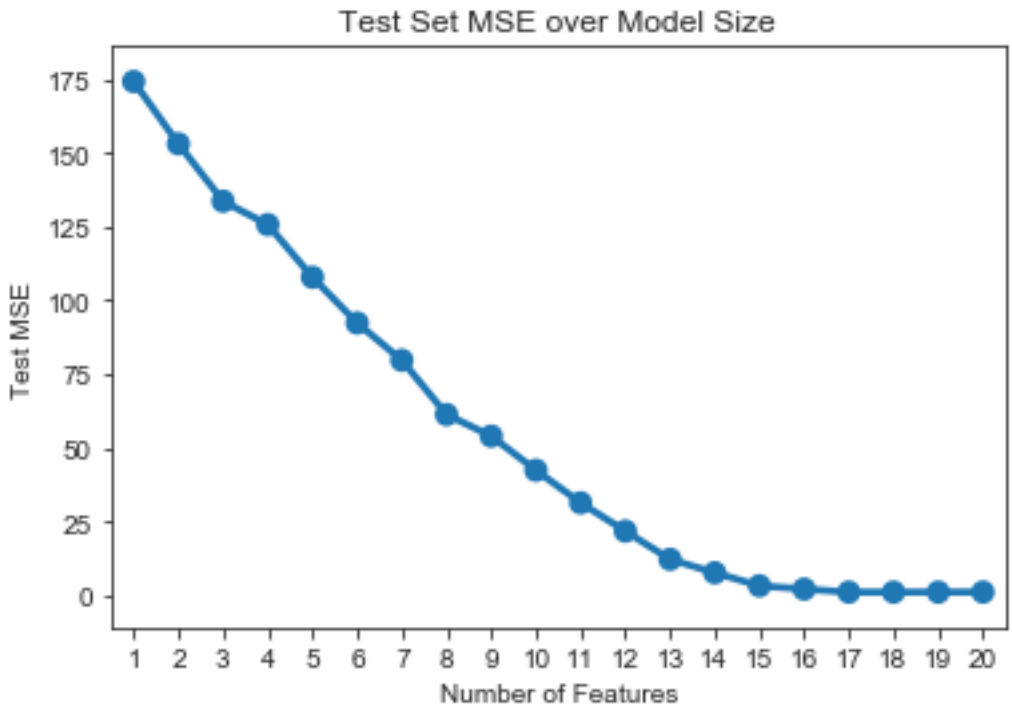
[434]:

	feature_num	mse	feature index
0	1	174.181112	[11]
1	2	153.086469	[7, 11]
2	3	133.643932	[2, 7, 11]
3	4	125.721621	[2, 7, 11, 15]
4	5	107.892774	[2, 7, 11, 15, 17]
5	6	92.403873	[2, 3, 7, 11, 15, 17]
6	7	79.773715	[2, 3, 7, 9, 11, 15, 17]
7	8	61.376733	[2, 3, 7, 9, 11, 12, 15, 17]
8	9	54.073238	[2, 3, 7, 9, 11, 12, 14, 15, 17]
9	10	42.526379	[2, 3, 7, 8, 9, 11, 12, 14, 15, 17]
10	11	31.474496	[0, 2, 3, 7, 8, 9, 11, 12, 14, 15, 17]
11	12	21.899762	[0, 2, 3, 7, 8, 9, 11, 12, 13, 14, 15, 17]
12	13	12.348713	[0, 2, 3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18]
13	14	7.792476	[0, 2, 3, 6, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18]
14	15	3.386154	[0, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 1...
15	16	2.258477	[0, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 1...
16	17	1.232281	[0, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1...
17	18	1.237293	[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...
18	19	1.240610	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
19	20	1.267565	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...

[435]:
```
ax = sns.pointplot(x="feature_num", y="mse", data=test_mse)
ax.set(xlabel='Number of Features', ylabel='Test MSE', title='Test Set MSE over␣
 ↪Model Size')
```

[435]:
```
[Text(0, 0.5, 'Test MSE'),
 Text(0.5, 0, 'Number of Features'),
```

```
Text(0.5, 1.0, 'Test Set MSE over Model Size')]
```



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

Test set MSE reaches the lowest value at model size equals to 17.

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.

[436]:
```
best_feats = test_mse["feature index"][16]
best_feats
```

[436]:
```
[0, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19]
```

[437]:
```
best_model = model.fit(X[:,best_feats], y)
best_model.coef_
```

[437]:
```
array([3.0151403 , 3.96238589, 3.97518812, 1.99959093, 2.05721518,
       3.99554478, 2.98090868, 3.92733144, 0.92701278, 2.97974286,
       3.93148804, 2.95806522, 2.96609713, 2.97196397, 4.01121667,
       2.98439607, 0.96061048])
```

[438]:
```
true_model = beta[best_feats]
true_coef = []
for i in range(len(true_model)):
    true_coef.append(true_model[i][0])
```

[439]:
```
compare_model = pd.DataFrame({"feature": best_feats, "best model": best_model.
 ↪coef_, "true model": true_coef})
compare_model
```

[439]:

	feature	best model	true model
0	0	3.015140	3
1	2	3.962386	4
2	3	3.975188	4
3	5	1.999591	2
4	6	2.057215	2
5	7	3.995545	4
6	8	2.980909	3
7	9	3.927331	4
8	10	0.927013	1
9	11	2.979743	3
10	12	3.931488	4
11	13	2.958065	3
12	14	2.966097	3
13	15	2.971964	3
14	17	4.011217	4
15	18	2.984396	3
16	19	0.960610	1

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.

We generated a few betas at 0, however, none of them showed up in our best model. Comparing the coefficient we get from the best model and the true model used to generate the data, most of the coefficients are smaller than the original beta and a few of the coefficients are around the same value.

Create a plot displaying

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

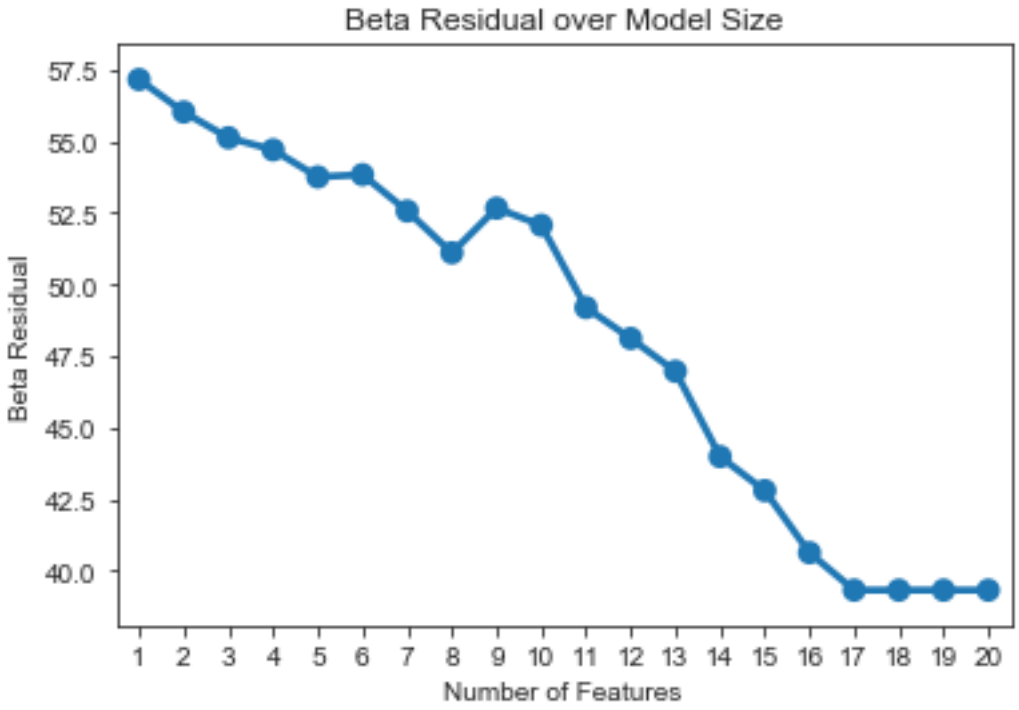
for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot?

[442]:
```
feature_num = list(i for i in range(1,21))

beta_res = []
for r in range(1,21):
    feats = best_feats[:r]
    lm = model.fit(X_train[:,feats], y_train)
    coef_hat = lm.coef_

    b = np.zeros(20)
    for i ,f in enumerate(feats):
        b[f] = coef_hat[i]
```

```
    calc = (((beta - b)**2).sum())**.5
    beta_res.append(calc)
```

[443]:
```
beta_residual = pd.DataFrame({"feature num": feature_num, "beta residual":␣
 ↪beta_res})
ax = sns.pointplot(x="feature num", y="beta residual", data=beta_residual)
ax.set(xlabel='Number of Features', ylabel='Beta Residual', title='Beta␣
 ↪Residual over Model Size')
```

[443]:
```
[Text(0, 0.5, 'Beta Residual'),
 Text(0.5, 0, 'Number of Features'),
 Text(0.5, 1.0, 'Beta Residual over Model Size')]
```



Both graphs showed a similar decreasing trend in their values as model size increases. Model size at 17 produced both the lowest beta residual and the lowest test MSE, suggesting a consistent pattern across our models.

0.2 Application Exercises

Your task is to construct a series of statistical/machine learning models to accurately predict an individual's egalitarianism using model selection and regularization methods. Use all the available predictors for each model unless otherwise specified.

Fit a least squares linear model on the training set, and report the test MSE.

[451]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
```

[452]:
```
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)
```

[453]:
```
lm = model.fit(X_train, y_train)
ypred = lm.predict(X_test)
```

[455]:
```
test_mse = mse(y_test, ypred)
print("Test MSE for Linear Regression: ", test_mse)
```

```
Test MSE for Linear Regression:  63.21362962301499
```

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

[460]:
```
ridge = RidgeCV(alphas=(0.1, 1.0, 10), cv=10).fit(X_train, y_train)
ypred = ridge.predict(X_test)
```

[461]:
```
test_mse = mse(y_test, ypred)
print("Test MSE for Ridge Regression: ", test_mse)
```

```
Test MSE for Ridge Regression:  62.4992024395781
```

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.

[462]:
```
lasso = LassoCV(alphas=(0.1, 1.0, 10), cv=10).fit(X_train, y_train)
print("Test MSE for Lasso Regression: ", mse(y_test, lasso.predict(X_test)))
```

```
Test MSE for Lasso Regression:  62.77841555477389
```

[463]:
```
print("Non-zero coefficient estimates: ", (lasso.coef_ != 0).sum())
```

```
Non-zero coefficient estimates:  24
```

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.

[465]:
```
elasticnet = ElasticNetCV(l1_ratio=np.linspace(.1,1,11), cv=10).fit(X_train,␣
↪y_train)
print("Test MSE for Elastic Net: ", mse(y_test, elasticnet.predict(X_test)))
```

```
Test MSE for Elastic Net:  62.7780157899344
```

[466]:
```
print("Non-zero coefficient estimates: ", (elasticnet.coef_ != 0).sum())
```

```
Non-zero coefficient estimates:  24
```

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?

There is no much difference among the test MSE from these models. All models generate test MSE at around 63 and the number of non-zero coefficient estimates is the same for Lasso Regression and Elastic Net Regression. In terms of test MSE, our prediction of an individual's egalitarianism is worse than coin flips (accuracy = 0.5). To obtain a more accurate prediction, we may consider using other models that could fit our dataset better.