modelhw3

February 9, 2020

[77]:
```
# import the packages
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from itertools import combinations
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV,␣
↪ElasticNetCV
from sklearn.metrics import roc_auc_score, mean_squared_error
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from mlxtend.feature_selection import SequentialFeatureSelector as sfs

import warnings
warnings.filterwarnings('ignore')
```

1 Conceptual exercises

1.1 Training/test error for subset selection

1.1.1 1. 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.

[13]:
```
#set random seed
np.random.seed(2020)
#simulate X
X = np.random.normal(0,5,(1000, 20))
#simulate error term
epsilon = np.random.normal(0,1, 1000)
```

[14]:
```
#simulate beta
beta =np.array([np.random.randint(-5,5) for i in range(20)])
zero_idx = [np.random.randint(0, 19) for i in range(10)]
beta[zero_idx] = 0
beta
```

[14]:
```
array([-4,  0, -1, -4,  1,  0, -1, -2, -5,  4,  0,  0, -2,  0,  3,  4,  0,
        0,  0,  0])
```

[15]:
```
#generate Y
Y = np.dot(X, beta) + epsilon
```

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

[16]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.90,␣
↪random_state=124)
```

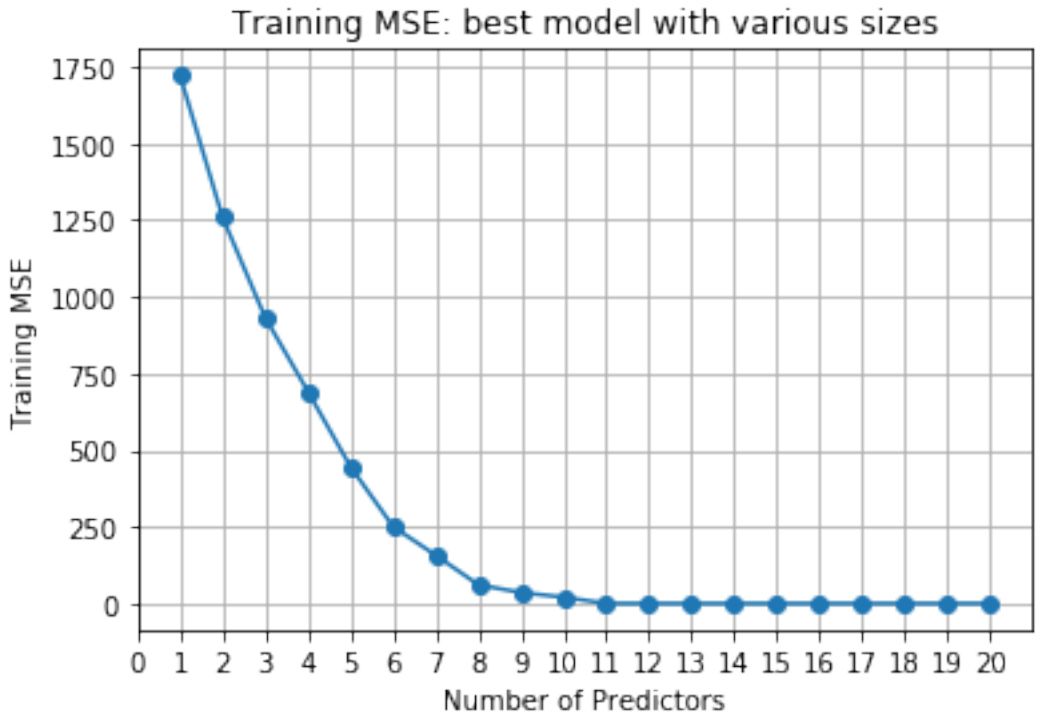
1.1.3 3. Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. For which model size does the training set MSE take on its minimum value?

[27]:
```
# Subset selection
#here is my original code, but jupyter could not handle  it, so I changed the␣
↪method
#idx = np.arange(20)
#best_models = []
#for i in range(1, 21):
    #models = []
    #for subset in combinations(idx, i):
         #res = sm.OLS(Y_train, X_train[:, susbet]).fit()
        #aic=res.aic
        #models.append((res,aic,subset))
    #best_models.append(sorted(models, key=lambda x:x[1])[0])

scores = []
for idx in range (1,21):
    lm= LinearRegression()
    subset = sfs(lm, k_features=idx, forward=True,
                     scoring ='neg_mean_squared_error', cv=0)
    subset.fit(X_train, Y_train)
    scores.append(-subset.k_score_)
```

[29]:
```
#plot the training MSE associated with the best model of each size
plt.plot(np.arange(1,21), scores, marker='o')
plt.xticks(np.arange(0, 21, 1))
```

```
plt.xlabel("Number of Predictors")
plt.ylabel("Training MSE")
plt.title("Training MSE: best model with various sizes")
plt.grid()
plt.show()
print('The training set MSE take on its minimum value at size {}'.format(np.
 ↪argmin(scores) + 1))
```




```
The training set MSE take on its minimum value at size 20
```

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

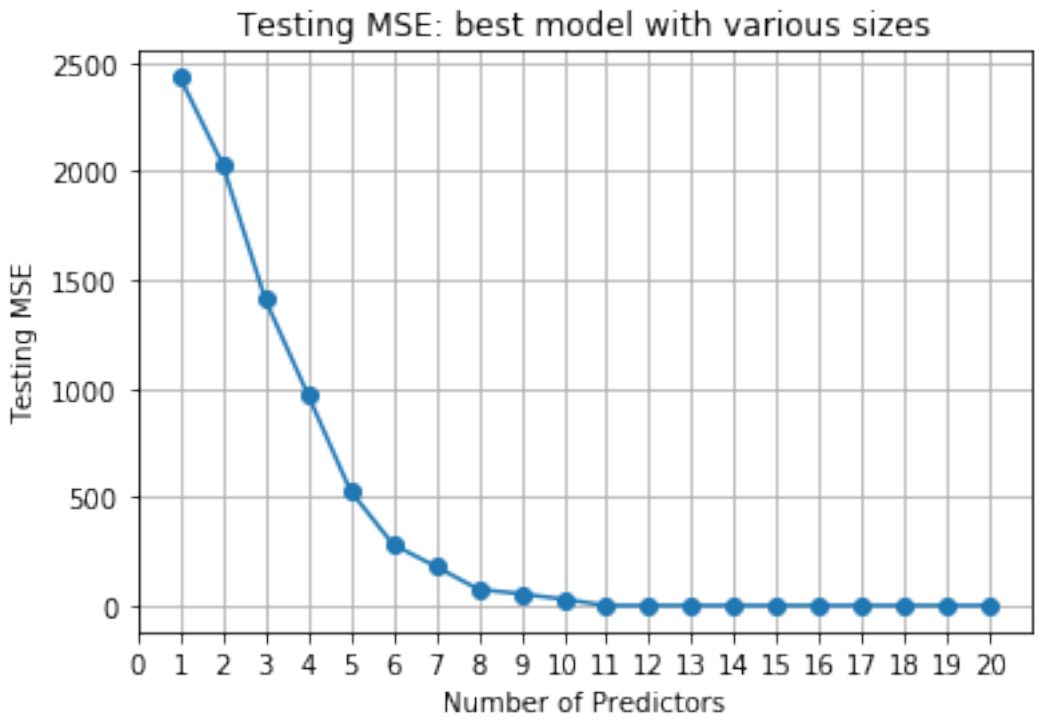
[38]:
```
test_mses = []
feature_idx = []
for idx in range (1,21):
    lm= LinearRegression()
    subset = sfs(lm, k_features=idx, forward=True,
                    scoring ='neg_mean_squared_error', cv=0)
    subset.fit(X_train, Y_train)
    lm = lm.fit(X_train[:, subset.k_feature_idx_], Y_train)
    test_mses.append(mean_squared_error(lm.predict(X_test[:, subset.
 ↪k_feature_idx_]),
                                        Y_test))
    feature_idx.append(list(subset.k_feature_idx_))
```

```
testdf = pd.DataFrame({"model_size": np.arange(1,21),"MSE": test_mses,
                       "feature_index": feature_idx})
testdf
```

[38]:

	model_size	MSE	feature_index
0	1	2432.920501	[3]
1	2	2032.068742	[3, 15]
2	3	1410.077805	[3, 8, 15]
3	4	972.006839	[3, 8, 9, 15]
4	5	528.043940	[0, 3, 8, 9, 15]
5	6	281.899818	[0, 3, 8, 9, 14, 15]
6	7	180.668302	[0, 3, 7, 8, 9, 14, 15]
7	8	75.435523	[0, 3, 7, 8, 9, 12, 14, 15]
8	9	54.811640	[0, 2, 3, 7, 8, 9, 12, 14, 15]
9	10	29.766675	[0, 2, 3, 4, 7, 8, 9, 12, 14, 15]
10	11	1.116165	[0, 2, 3, 4, 6, 7, 8, 9, 12, 14, 15]
11	12	1.118329	[0, 2, 3, 4, 6, 7, 8, 9, 12, 14, 15, 18]
12	13	1.129041	[0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 14, 15, 18]
13	14	1.129712	[0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 14, 15, 17, 18]
14	15	1.138547	[0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 14, 15, 16, 17…
15	16	1.137117	[0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 14, 15, 16…
16	17	1.142900	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15,…
17	18	1.152769	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14,…
18	19	1.152492	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…
19	20	1.157323	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,…

[40]:
```
plt.plot(np.arange(1,21), test_mses, marker='o')
plt.xticks(np.arange(0, 21, 1))
plt.xlabel("Number of Predictors")
plt.ylabel("Testing MSE")
plt.title("Testing MSE: best model with various sizes")
plt.grid()
plt.show()
print('The test set MSE take on its minimum value at model size  {}'.format(np.
 ↪argmin(test_mses) + 1))
```




```
The test set MSE take on its minimum value at model size  11
```

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

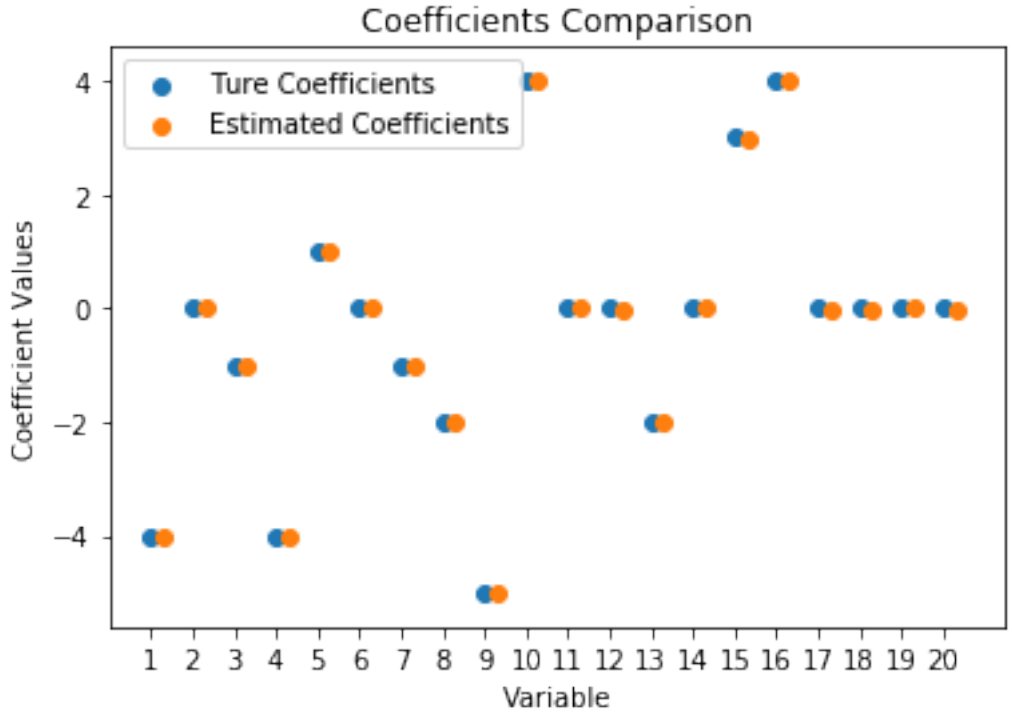
The test set MSE takes on its minimum value at size 11. As there are exactly 11 non-zero coefficients in our model, it could be concluded that the best model selects all the true predictors.

1.1.6 6. 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.

[58]:
```
# calculate the coefficient for the estimated variables
estimates = np.zeros(20)
best_feats = testdf["feature_index"][10]  #since we know the least MSE model is␣
 ↪size 11
best_model = lm.fit(X[:,best_feats], Y)
for i, coef in zip(best_feats,best_model.coef_):
    estimates[i]=coef
```

[94]:
```
plt.scatter(np.arange(1,21), beta, label='Ture Coefficients')
plt.scatter(np.arange(1,21)+0.3, estimates, label='Estimated Coefficients')
plt.xticks(np.arange(1,21))
plt.xlabel('Variable')
plt.ylabel('Coefficient Values')
```

```
plt.title("Coefficients Comparison")
plt.legend()
plt.show()
```



According to the graph above, the best model's estimation of coefficients is very close to the true model.

1.1.7 7. Create a plot displaying

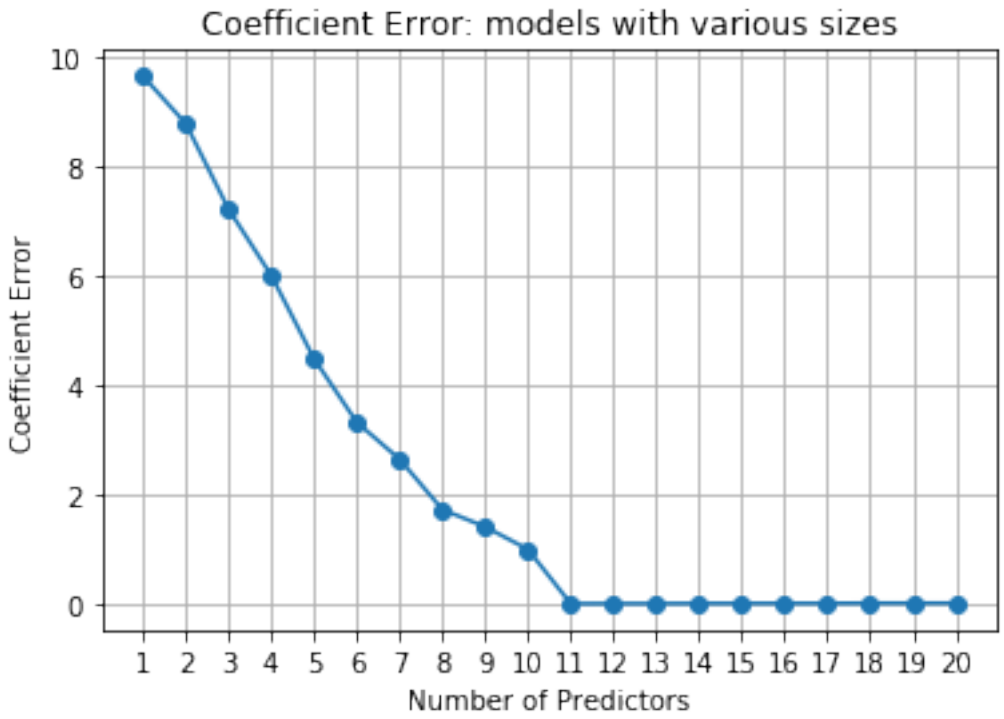
$$\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$$

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?

[71]:
```
# caculate the coefficient errors for different best models
#based on the number of preditors
coef_errors = []
for r in range(20):
    estimates = np.zeros(20)
    feats = testdf["feature_index"][r]
    model = lm.fit(X[:,feats], Y)
    for i, coef in zip(feats,model.coef_):
```

```
        estimates[i]=coef
    coef_errors.append(np.sqrt(np.sum((beta - estimates) ** 2)))
```

[72]:
```
# plot the coefficient errors
plt.plot(np.arange(1,21), coef_errors, marker='o')
plt.xlabel('Number of Predictors')
plt.xticks(np.arange(1,21))
plt.ylabel('Coefficient Error')
plt.title('Coefficient Error: models with various sizes')
plt.grid()
plt.show()
```



The coefficient error plot is very similar to the test MSE plot, reaching minimum at size 11.

2 Application exercises

[73]:
```
#data loading
gss_train = pd.read_csv("gss_train.csv")
gss_test = pd.read_csv("gss_test.csv")
```

[74]:
```
#data spliting
x_train = gss_train.drop(['egalit_scale'], axis=1)
y_train = gss_train['egalit_scale']
```

```
x_test = gss_test.drop(['egalit_scale'], axis=1)
y_test = gss_test['egalit_scale']
```

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

[75]:
```
ols = LinearRegression().fit(x_train, y_train)
pred= ols.predict(x_test)
mse_ols =mean_squared_error(y_test, pred)
print('The test MSE of least squares linear model is {:.3f}'.format(mse_ols))
```

```
The test MSE of least squares linear model is 63.214
```

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

[92]:
```
ridge = RidgeCV(alphas= np.logspace(-5, 10, 45),cv=10).fit(x_train, y_train)
pred=ridge.predict(x_test)
mse_ridge = mean_squared_error(y_test, pred)
print('The test MSE of ridge regression model is {:.3f}'.format(mse_ridge))
```

```
The test MSE of ridge regression model is 62.143
```

2.0.3 3. 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.

[97]:
```
lasso = LassoCV(alphas=np.logspace(-5, 3, 45), cv=10).fit(x_train, y_train)
pred=lasso.predict(x_test)
mse_lasso = mean_squared_error(y_test, pred)
non_zero =(lasso.coef_ != 0).sum()
print("The test MSE of lasso regression model is {:.3f}".format(mse_lasso))
print("Number of nonzero coefficients: {}".format(non_zero))
```

```
The test MSE of lasso regression model is 62.778
Number of nonzero coefficients: 24
```

2.0.4 4. 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.

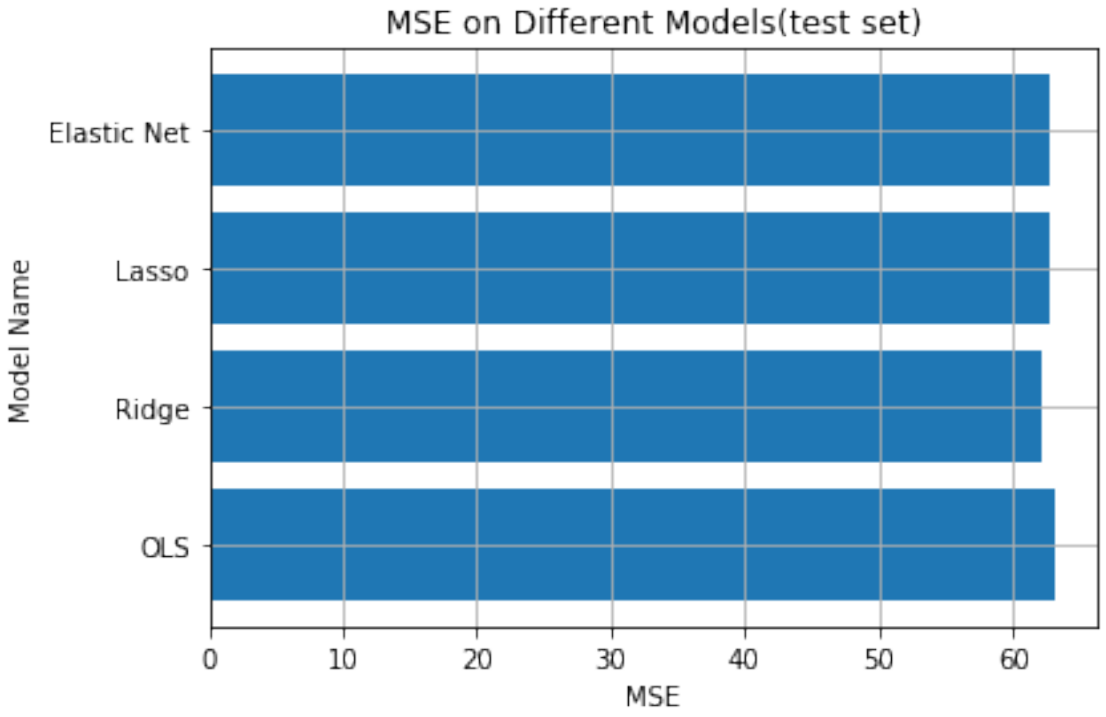
[106]:
```
alphas=np.arange(0.1, 1.1, 0.1)
elastic = ElasticNetCV(alphas=alphas,cv=10).fit(x_train, y_train)
pred=elastic.predict(x_test)
mse_elastic = mean_squared_error(y_test, pred)
non_zero = np.sum(elastic.coef_ != 0)
```

```
print('alpha:', elastic.l1_ratio_)
print('lambda:', elastic.alpha_)
print("The test MSE of elastic net regression model is {:.3f}".
 ↪format(mse_elastic))
print("Number of nonzero coefficients: {}".format(non_zero))
```

```
alpha: 0.5
lambda: 0.1
The test MSE of elastic net regression model is 62.507
Number of nonzero coefficients: 40
```

2.0.5 5. 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?

[93]:
```
names = ["OLS", "Ridge", "Lasso", "Elastic Net"]
mses = [mse_ols, mse_ridge, mse_lasso, mse_elastic]
fig, ax = plt.subplots()
ax.grid()
ax.barh(range(len(names)), mses)
ax.set_yticks(range(len(names)))
ax.set_yticklabels(names)
ax.set_xlabel('MSE')
ax.set_ylabel('Model Name')
ax.set_title('MSE on Different Models(test set)')
plt.show()
```



The most accurate model is Ridge model, which has the samllest test MSE of 62.143. However, ElasticNet, Lasso, and OLS actually share the similar MSE with Ridge, all in the range of 62-63, meaning the accuracy of individual's egalitarianism prediction is low (less than 0.4) for all the four models.To obtain a higher accuracy, we should probably refer to other models.