Homework 3: Linear Model Selection and Regularization

Luying Jiang

In [1]:
```
import numpy as np
import random
import pandas as pd
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import os
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from mlxtend.plotting import plot_sequential_feature_selection as plot_s
fs
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
import itertools
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
```

Conceptual exercises

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

In [2]:
```
np.random.seed(2)
df1 = {}
beta = {}
df1 = pd.DataFrame(df1)
for i in range(1, 21):
    df1['x{}'.format(i)] = np.random.normal(0, 10, 1000)
    beta['x{}'.format(i)] = np.random.normal(0, 10, 1)
epsilon = np.random.normal(0, 10, 1000)
for i in np.random.choice(20,5):
    beta['x{}'.format(i)] = 0

y = np.zeros(1000)
for i in range(1, 21):
    y += df1['x{}'.format(i)] * beta['x{}'.format(i)]
y += epsilon
```

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

```
In [3]: X_train, X_test, y_train, y_test = train_test_split(df1, y, test_size=90
        0)
```

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?

```
In [4]: def get_best_subset (X_train, X_test, y_train, y_test, train=True):
            dict = {}
            best = 1000000000
            for k in range (1,5):
                for c in itertools.combinations(X_train, k):
                    lm = LinearRegression().fit(X_train[list(c)], y_train)
                    y_pred = lm.predict(X_train[list(c)])
                    mse = mean_squared_error(y_train, y_pred)
                    if mse < best:
                        best = mse
                        result = list(c)
                dict[str(result)] = best
            return dict
```

```
In [5]: result = get_best_subset(X_train, X_test, y_train, y_test)
        result
```

```
Out[5]: {"['x3']": 130268.68029551895,
         "['x3', 'x14']": 104831.270558862,
         "['x3', 'x8', 'x14']": 78860.047353813,
         "['x2', 'x3', 'x8', 'x14']": 60421.16536260763}
```

```
In [6]: sfs = SFS(LinearRegression(), k_features=5, forward=True, \
                  scoring ='neg_mean_squared_error', cv=0)
        sfs.fit(X_train, y_train)
        sfs.k_feature_names_
```

```
Out[6]: ('x2', 'x3', 'x8', 'x14', 'x16')
```

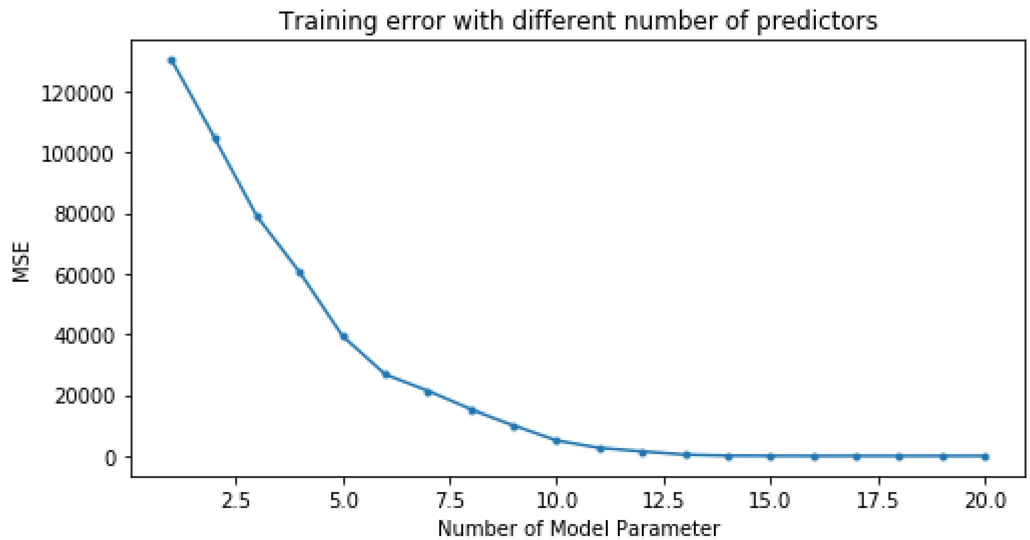
In [7]:
```python
result = []
model = []
for n in range (1,21):
    lr = LinearRegression()
    sfs = SFS(lr, k_features=n, forward=True, floating=False,
              scoring='neg_mean_squared_error', cv=0)
    sfs = sfs.fit(X_train, y_train)
    lm = lr.fit(X_train[list(sfs.k_feature_names_)], y_train)
    y_pred = lm.predict(X_test[list(sfs.k_feature_names_)])
    test_mse = mean_squared_error(y_test, y_pred)
    result.append([list(sfs.k_feature_names_), -sfs.k_score_, test_mse])
    model.append(lm)

best_models = pd.DataFrame(result, columns=['predictors', 'train_mse',
'test_mse'])
best_models['feature_num'] = range(1,21)
best_models.head(20)
```

Out[7]:

	predictors	train_mse	test_mse	feature_num
0	[x3]	130268.680296	124211.006842	1
1	[x3, x14]	104831.270559	103478.765617	2
2	[x3, x8, x14]	78860.047354	74406.126524	3
3	[x2, x3, x8, x14]	60421.165363	56416.614948	4
4	[x2, x3, x8, x14, x16]	39447.367666	43932.745864	5
5	[x2, x3, x8, x14, x15, x16]	26771.309786	35481.282530	6
6	[x2, x3, x4, x8, x14, x15, x16]	21364.329809	26646.408100	7
7	[x2, x3, x4, x8, x14, x15, x16, x17]	15344.592737	18707.731001	8
8	[x2, x3, x4, x8, x14, x15, x16, x17, x19]	10083.771613	11081.163344	9
9	[x2, x3, x4, x8, x14, x15, x16, x17, x19, x20]	5133.793472	5879.447794	10
10	[x2, x3, x4, x8, x12, x14, x15, x16, x17, x19,...	2718.890888	3122.631642	11
11	[x2, x3, x4, x8, x12, x14, x15, x16, x17, x18,...	1551.050808	1969.011692	12
12	[x2, x3, x4, x7, x8, x12, x14, x15, x16, x17, ...	490.823603	604.120009	13
13	[x1, x2, x3, x4, x7, x8, x12, x14, x15, x16, x...	145.564074	171.624498	14
14	[x1, x2, x3, x4, x5, x7, x8, x12, x14, x15, x1...	81.147105	119.025969	15
15	[x1, x2, x3, x4, x5, x6, x7, x8, x12, x14, x15...	63.755699	109.800173	16
16	[x1, x2, x3, x4, x5, x6, x7, x8, x10, x12, x14...	63.069694	110.644822	17
17	[x1, x2, x3, x4, x5, x6, x7, x8, x10, x11, x12...	62.832081	110.006844	18
18	[x1, x2, x3, x4, x5, x6, x7, x8, x10, x11, x12...	62.764931	110.100536	19
19	[x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11,...	62.699046	110.408001	20

In [8]:
```
plt.figure(figsize=(8,4))
plt.plot(best_models['feature_num'], best_models['train_mse'], marker=
'.')
plt.title('Training error with different number of predictors')
plt.xlabel('Number of Model Parameter')
plt.ylabel('MSE')
```

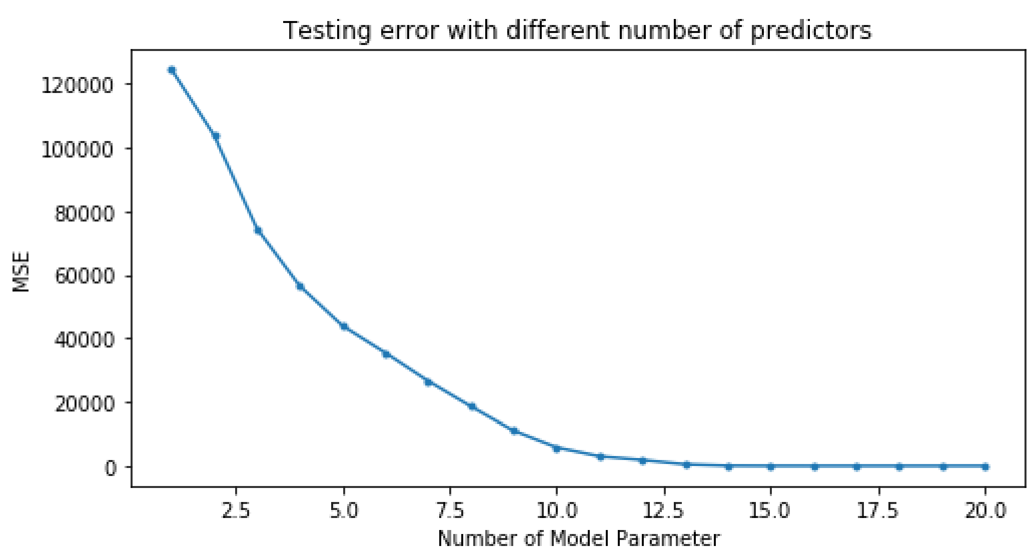
Out[8]:
```
Text(0, 0.5, 'MSE')
```



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

In [9]:
```
plt.figure(figsize=(8,4))
plt.plot(best_models['feature_num'], best_models['test_mse'], marker='.'
)
plt.title('Testing error with different number of predictors')
plt.xlabel('Number of Model Parameter')
plt.ylabel('MSE')
```

Out[9]:
```
Text(0, 0.5, 'MSE')
```



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

If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you generate the data previously until you create a data generating process in which the test set MSE is minimized for an intermediate model size.

In [10]:
```python
best_models['test_mse'].min()
```

Out[10]:
```
109.8001732639053
```

In [11]:
```python
best_models.query('test_mse==109.8001732639053')
```

Out[11]:

	predictors	train_mse	test_mse	feature_num
15	[x1, x2, x3, x4, x5, x6, x7, x8, x12, x14, x15...	63.755699	109.800173	16

In [12]:
```python
print(best_models['predictors'][15])
```

```
['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x12', 'x14', 'x15',
'x16', 'x17', 'x18', 'x19', 'x20']
```

In [13]:
```python
for key, value in beta.items():
    if value == 0:
        print(key)
```

```
x9
x10
x11
x13
```

Test mse is lowest for the model with ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x12', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20']

This is very reasonable because as we can see the beta for x9, x10, x11, x13 were zero. This best model filtered out these four predictors.

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.

In [14]:
```
min_mse_model = model[15]
min_lst = list(min_mse_model.coef_)
min_lst
```

Out[14]:
```
[1.8280706256311576,
 14.77127178673719,
 20.390694558078607,
 -9.152853880904809,
 0.7953814716867165,
 0.48196614186240994,
 -3.7248470106838463,
 -14.423420459372034,
 -5.034947722055537,
 -12.866717627018499,
 9.624494713501795,
 -11.203478466459838,
 9.004664466474496,
 3.286636196502318,
 8.559956012571163,
 6.511452800264059]
```

In [15]:
```
lst = []
for value in beta.values():
    if value:
        lst.append(value[0])
lst
```

Out[15]:
```
[1.9774991367911983,
 14.8084904924909,
 20.357540841172572,
 -9.143790010975474,
 0.6692490981071317,
 0.3886642224103053,
 -3.607692700773831,
 -14.507080041604558,
 -5.070750888610807,
 -12.929710944132786,
 9.751853160826798,
 -11.175148764317184,
 8.90347181016492,
 3.2327635788443683,
 8.602370328063184,
 6.327728833594861]
```

In [16]:
```
df2 = pd.DataFrame({"Model_coefficient": min_lst, 'Beta':lst })
df2
```

Out[16]:

	Model_coefficient	Beta
0	1.828071	1.977499
1	14.771272	14.808490
2	20.390695	20.357541
3	-9.152854	-9.143790
4	0.795381	0.669249
5	0.481966	0.388664
6	-3.724847	-3.607693
7	-14.423420	-14.507080
8	-5.034948	-5.070751
9	-12.866718	-12.929711
10	9.624495	9.751853
11	-11.203478	-11.175149
12	9.004664	8.903472
13	3.286636	3.232764
14	8.559956	8.602370
15	6.511453	6.327729

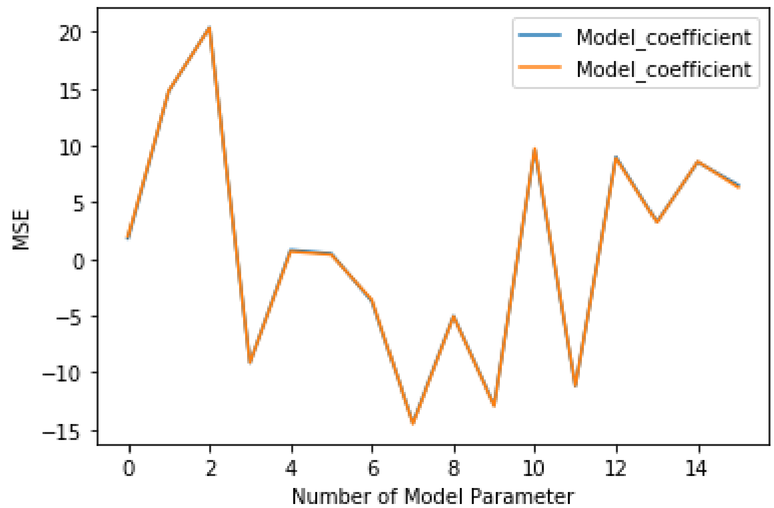
In [17]:
```
df2.describe()
```

Out[17]:

	Model_coefficient	Beta
count	16.000000	16.000000
mean	1.178020	1.161591
std	10.126757	10.133632
min	-14.423420	-14.507080
25%	-6.064424	-6.089011
50%	1.311726	1.323374
75%	8.671133	8.677646
max	20.390695	20.357541

In [18]:
```python
plt.plot(df2['Model_coefficient'], label = 'Model_coefficient')
plt.plot(df2['Beta'], label = 'Model_coefficient')
plt.legend()
plt.xlabel('Number of Model Parameter')
plt.ylabel('MSE')
```

Out[18]:
```
Text(0, 0.5, 'MSE')
```



As we can see from the dataframe and the graph above, the best model's coefficient is very close to true betas.

7.Create a plot displaying for a range of values of r. Comment on what you observe. How does this compare to the test MSE plot?

In [19]:
```python
beta
```

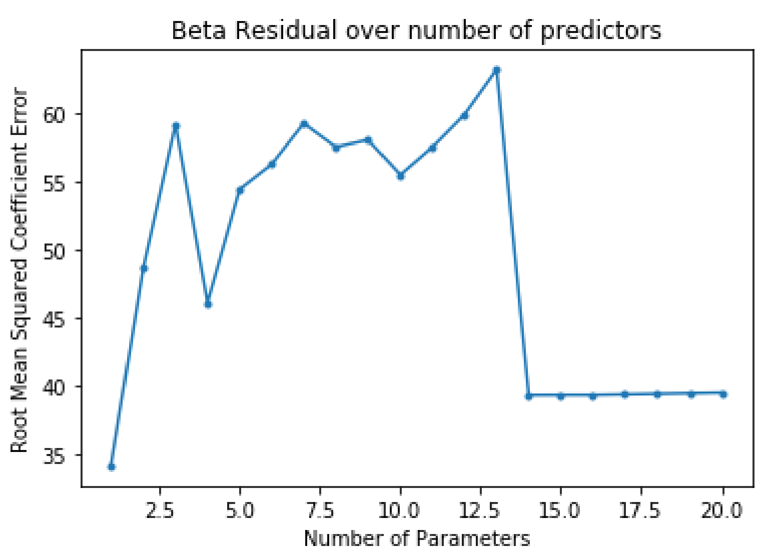
Out[19]:
```
{'x1': array([1.97749914]),
 'x2': array([14.80849049]),
 'x3': array([20.35754084]),
 'x4': array([-9.14379001]),
 'x5': array([0.6692491]),
 'x6': array([0.38866422]),
 'x7': array([-3.6076927]),
 'x8': array([-14.50708004]),
 'x9': 0,
 'x10': 0,
 'x11': 0,
 'x12': array([-5.07075089]),
 'x13': 0,
 'x14': array([-12.92971094]),
 'x15': array([9.75185316]),
 'x16': array([-11.17514876]),
 'x17': array([8.90347181]),
 'x18': array([3.23276358]),
 'x19': array([8.60237033]),
 'x20': array([6.32772883])}
```

In [20]:
```
beta_dict = {}
for key, value in beta.items():
    if value:
        beta_dict[key] = value[0]
    else:
        beta_dict[key] = value
```

In [21]:
```
predictors = []
for i in range(1, 21):
    predictors.append('x{}'.format(i))
```

In [22]:
```
res_lst = []
count = 0
for m in model:
    coefficient = m.coef_
    j = 0
    for p in predictors:
        count_pred = 0
        beta_j = beta_dict[p]
        if p not in best_models['predictors'][count]:
            beta_jr = 0
        else:
            beta_jr = coefficient[count_pred]
            count_pred += 1
        j += (beta_j - beta_jr) ** 2
    count += 1
    res_lst.append(j ** (1/2))
best_models['beta_diff'] = res_lst
plt.plot(best_models['feature_num'],best_models['beta_diff'], marker='.'
)
plt.title('Beta Residual over number of predictors')
plt.ylabel('Root Mean Squared Coefficient Error')
plt.xlabel('Number of Parameters')
```

Out[22]:
```
Text(0.5, 0, 'Number of Parameters')
```



From the above graph we can see there is a big jump from p = 13. It might due to that linear regression algorithm is using the available parameters to spuriously explain other parameters. Compared with the MSE plot, we can see that error declines steadily as we add in more parameters. we get into the more accurate models about p = 15.

Application exercises

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

In [23]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
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
```

In [24]:
```
model = LinearRegression().fit(X_train, y_train)
err = mean_squared_error(model.predict(X_test), y_test)
model.score(X_train, y_train)
```

Out[24]:
```
0.4041427253127764
```

In [25]:
```
print("Test MSE for Linaer Regression is:", err)
```

```
Test MSE for Linaer Regression is: 63.213629623014995
```

2.Fit a ridge regression model on the training set. Report the test MSE.

In [26]:
```
model = RidgeCV(alphas=(.1,1.0,10), cv=10).\
                fit(X_train, y_train)
err = mean_squared_error(model.predict(X_test), y_test)
model.score(X_train, y_train)
```

Out[26]:
```
0.40251855279853743
```

In [27]:
```
print("Test MSE for Ridge is:", err)
```

```
Test MSE for Ridge is: 62.49920243957809
```

3.Fit a lasso regression on the training set. Report the test MSE, along with the number of non-zero coefficient estimates.

In [28]:
```
model = LassoCV(alphas=(.1,1.0,10),cv=10).fit(X_train, y_train)
err = mean_squared_error(model.predict(X_test), y_test)
model.score(X_train, y_train)
```

Out[28]: 0.38161120285798467

In [29]:
```
print("Test MSE for Lasso is:", err)
print('Non-zero coefficent estimates:', (model.coef_ != 0).sum())
```

```
Test MSE for Lasso is: 62.77841555477389
Non-zero coefficent estimates: 24
```

4.Fit an elastic net regression model on the training set

In [30]:
```
model = ElasticNetCV(l1_ratio = np.linspace(.1,1,11), cv=10).\
                    fit(X_train, y_train).fit(X_train, y_train)
err = mean_squared_error(model.predict(X_test), y_test)
model.score(X_train, y_train)
```

Out[30]: 0.3816218146011686

In [31]:
```
print("Test MSE for Elastic Net: ", err)
print("Non-zero coefficient estimates: ", (model.coef_ != 0).sum())
```

```
Test MSE for Elastic Net:  62.7780157899344
Non-zero coefficient estimates:  24
```

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?

The test MSE generated from these models are almost the same (around 63). The number of non-zero coefficient estimates is the same for Lasso Regression and Elastic Net Regression. We are not predictiong the individual's egalitarianism very well since the training accuracies are very low for all models. So, in order to get a higher accuracy we would need to change the model entirely.