Homework 3

YIMIN LI

Conceptual Exercises

1.

In [140]:
```
pip install mlxtend
```

```
Collecting mlxtend
  Downloading https://files.pythonhosted.org/packages/f1/1a/8ee48a7f448
063428d30080ee5afd9efcf8f01c119d3d473c27bd65b0e0e/mlxtend-0.17.1-py2.py
3-none-any.whl (1.3MB)
Requirement already satisfied: scipy>=1.2.1 in c:\users\qmun\anaconda3
\lib\site-packages (from mlxtend) (1.3.1)
Requirement already satisfied: setuptools in c:\users\qmun\anaconda3\li
b\site-packages (from mlxtend) (41.4.0)
Requirement already satisfied: numpy>=1.16.2 in c:\users\qmun\anaconda3
\lib\site-packages (from mlxtend) (1.16.5)
Requirement already satisfied: pandas>=0.24.2 in c:\users\qmun\anaconda
3\lib\site-packages (from mlxtend) (0.25.1)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\qmun\anaco
nda3\lib\site-packages (from mlxtend) (3.1.1)
Requirement already satisfied: scikit-learn>=0.20.3 in c:\users\qmun\an
aconda3\lib\site-packages (from mlxtend) (0.21.3)
Requirement already satisfied: joblib>=0.13.2 in c:\users\qmun\anaconda
3\lib\site-packages (from mlxtend) (0.13.2)
Requirement already satisfied: pytz>=2017.2 in c:\users\qmun\anaconda3
\lib\site-packages (from pandas>=0.24.2->mlxtend) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\qmun
\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2.8.0)
```

```
Requirement already satisfied: cycler>=0.10 in c:\users\qmun\anaconda3
\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\qmun\anaco
nda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in c:\users\qmun\anaconda3\lib\site-packages (from matplotlib>=3.0.0->m
lxtend) (2.4.2)
Requirement already satisfied: six>=1.5 in c:\users\qmun\anaconda3\lib
\site-packages (from python-dateutil>=2.6.1->pandas>=0.24.2->mlxtend)
(1.12.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.17.1
Note: you may need to restart the kernel to use updated packages.
```

In [156]:

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

1.1 Generate Dataset

In [161]:

```
np.random.seed(2020)
```

In [162]:

```
df = []
for _ in range(20):
    df.append(np.random.normal(0, 10, 1000))
df = np.array(df).T
df = pd.DataFrame(df)
df
```

Out[162]:

0	1	2	3	4	5	6	7

	0	1	2	3	4	5	6	7
0	-17.688457	6.578404	0.400779	9.694740	-7.074369	7.857836	-2.443279	-1.018334
1	0.755523	11.844630	-5.904629	18.493896	14.924363	20.030841	-8.676500	-11.031040
2	-11.306297	-20.439319	10.297282	-11.666558	13.679257	-5.435586	-7.975797	-10.020697
3	-6.514302	-7.712667	13.669068	-7.015749	1.039482	-16.053138	-28.182543	-5.651003
4	-8.931156	-17.644878	-0.951534	1.933064	13.248833	-7.942552	-8.799388	13.693893
...	...	...	...	...	...	...	...	...
995	2.968719	2.902783	1.304453	-4.013326	-7.508097	3.805827	-1.739481	6.527429
996	0.147629	9.267086	4.710981	5.749484	-5.212430	-6.900945	-11.242778	-0.485571
997	-4.203736	-3.840291	-11.669394	5.153094	-6.441710	-8.165011	1.347837	2.536647
998	1.611505	-8.291596	14.125256	-1.538833	6.381136	-5.781009	8.999482	-2.670113
999	1.473629	23.564267	7.319866	-1.117668	-1.579777	2.160247	-12.153794	-12.623845

1000 rows × 20 columns

In [163]:

```
beta = np.random.normal(0, 1, 20)
beta0 = set()
for _ in range(5):
    beta0.add(np.random.randint(19))
for i in beta0:
    beta[i] = 0
beta
```

Out[163]:

```
array([-0.54253274,  0.07265167, -0.17226138,  1.12925604, -0.54315653,
        0.        , -0.054445  ,  0.        , -2.04209635,  1.08410106,
       -0.90502631,  0.        ,  0.        , -0.72735377,  0.        ,
        1.68362229, -0.45129138, -0.62126452, -1.80681706,  0.3777818
8])
```

In [164]:

```
epis = np.random.normal(0, 10, 1000)
Y = np.sum(df * beta, axis=1)
```

```python
df['Y'] = Y
dict_name = {}
for i in range(20):
    dict_name[i] = 'x' + str(i)
df.rename(columns = dict_name, inplace=True)
df.head(5)
```

Out[164]:

	x0	x1	x2	x3	x4	x5	x6	x7	
0	-17.688457	6.578404	0.400779	9.694740	-7.074369	7.857836	-2.443279	-1.018334	
1	0.755523	11.844630	-5.904629	18.493896	14.924363	20.030841	-8.676500	-11.031040	
2	-11.306297	-20.439319	10.297282	-11.666558	13.679257	-5.435586	-7.975797	-10.020697	-
3	-6.514302	-7.712667	13.669068	-7.015749	1.039482	-16.053138	-28.182543	-5.651003	-1
4	-8.931156	-17.644878	-0.951534	1.933064	13.248833	-7.942552	-8.799388	13.693893	-

5 rows × 21 columns

1.2

In [165]:
```python
X = df.drop(['Y'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size =
0.9)
x_test.shape
```

Out[165]: (900, 20)

1.3

In [166]:
```python
ls = []
for i in range(1, 21):
    sfs = SFS(LR(), k_features=i, forward=True, scoring='neg_mean_squar
ed_error', cv=0)
    sfs.fit(x_train, y_train)
```

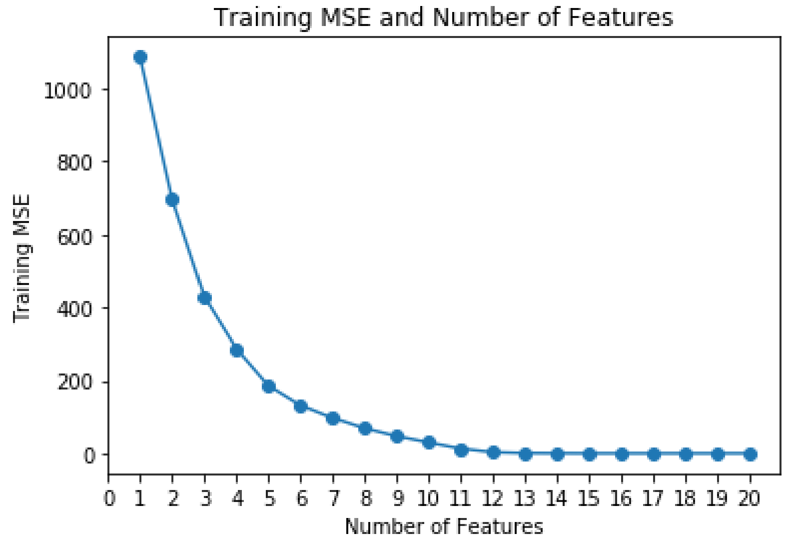
```
    ls.append(-sfs.k_score_)
ls
```

Out[166]:

```
[1086.4727512884353,
 697.8770615699992,
 430.10043519122615,
 287.4307624816783,
 186.09287850534628,
 132.1735642832613,
 97.62170780038525,
 69.00604719487217,
 47.35414438621481,
 30.08970309308089,
 13.116658926791292,
 3.178868662944707,
 0.7526327838293129,
 0.2596731938630361,
 1.4788584954225408e-27,
 1.5014739069059927e-27,
 1.2594981119483568e-27,
 1.1725958291448796e-27,
 1.4368300278781063e-27,
 2.0531702578747103e-27]
```

In [167]:

```
plt.plot(np.arange(1, 21), ls, marker = 'o', label=ls)
plt.xticks(np.arange(0, 21, 1))
plt.xlabel("Number of Features")
plt.ylabel("Training MSE")
plt.title("Training MSE and Number of Features")
plt.show()
```



As is shown on the list data and plot above, MSE decreases as its model size increases. Hence, the minimum value of MSE lies on the biggest training set at size 20.

1.4 & 1.5

In [168]:
```
ls_test = []
for i in range(1, 21):
    sfs = SFS(LR(), k_features=i, forward=True, scoring='neg_mean_squar
ed_error', cv=0)
    sfs.fit(x_train, y_train)
    lr = LR().fit(x_train[list(sfs.k_feature_names_)], y_train)
    test_mse = mean_squared_error(y_test, lr.predict(x_test[list(sfs.k_
feature_names_)]))
    ls_test.append([list(sfs.k_feature_names_), test_mse])
ls_test
```

Out[168]:
```
[[['x8'], 1082.692923233422],
 [['x8', 'x15'], 849.397392713374],
 [['x8', 'x15', 'x18'], 539.2165419662747],
 [['x3', 'x8', 'x15', 'x18'], 392.061337906979],
 [['x3', 'x8', 'x9', 'x15', 'x18'], 282.9744435124488],
```

```
 [['x3', 'x8', 'x9', 'x10', 'x15', 'x18'], 195.77543238597374],

 [['x3', 'x8', 'x9', 'x10', 'x13', 'x15', 'x18'], 138.9575903050876],
 [['x3', 'x8', 'x9', 'x10', 'x13', 'x15', 'x16', 'x18'], 122.3137529669
4294],
 [['x3', 'x8', 'x9', 'x10', 'x13', 'x15', 'x16', 'x17', 'x18'],
  86.56039514711821],
 [['x3', 'x4', 'x8', 'x9', 'x10', 'x13', 'x15', 'x16', 'x17', 'x18'],
  52.624598174759605],
 [['x0', 'x3', 'x4', 'x8', 'x9', 'x10', 'x13', 'x15', 'x16', 'x17', 'x1
8'],
  20.334774632013787],
 [['x0',
   'x3',
   'x4',
   'x8',
   'x9',
   'x10',
   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  4.38094173798977],
 [['x0',
   'x2',
   'x3',
   'x4',
   'x8',
   'x9',
   'x10',
   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  1.037393385902454],
 [['x0',
```

```
   'x1',

   'x2',
   'x3',
   'x4',
   'x8',
   'x9',
   'x10',
   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  0.3754051555517683],
 [['x0',
   'x1',
   'x2',
   'x3',
   'x4',
   'x6',
   'x8',
   'x9',
   'x10',
   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  1.5776323208232044e-27],
 [['x0',
   'x1',
   'x2',
   'x3',
   'x4',
   'x6',
   'x7',
   'x8',
   'x9',
```

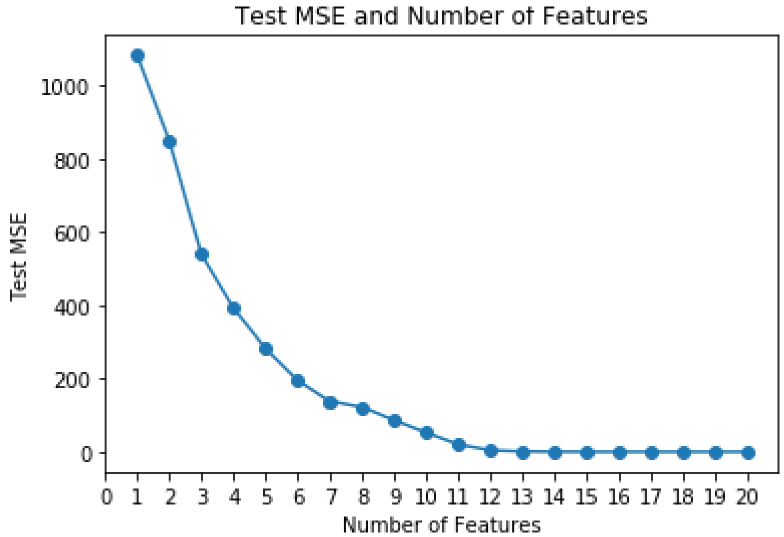
```
   'x10',

   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  1.6481833745138853e-27],
 [['x0',
   'x1',
   'x2',
   'x3',
   'x4',
   'x6',
   'x7',
   'x8',
   'x9',
   'x10',
   'x12',
   'x13',
   'x15',
   'x16',
   'x17',
   'x18',
   'x19'],
  1.5509956357188947e-27],
 [['x0',
   'x1',
   'x2',
   'x3',
   'x4',
   'x6',
   'x7',
   'x8',
   'x9',
   'x10',
   'x11',
   'x12',
   'x13',
```

```
'x15',

  'x16',
  'x17',
  'x18',
  'x19'],
 1.3980496499585618e-27],
[['x0',
  'x1',
  'x2',
  'x3',
  'x4',
  'x5',
  'x6',
  'x7',
  'x8',
  'x9',
  'x10',
  'x11',
  'x12',
  'x13',
  'x15',
  'x16',
  'x17',
  'x18',
  'x19'],
 1.5527292583484516e-27],
[['x0',
  'x1',
  'x2',
  'x3',
  'x4',
  'x5',
  'x6',
  'x7',
  'x8',
  'x9',
  'x10',
  'x11',
  'x12',
```

```
  'x13',

  'x14',
  'x15',
  'x16',
  'x17',
  'x18',
  'x19'],
 2.4542220151030666e-27]]
```

In [174]:

```python
ls_value = []
for i in ls_test:
    ls_value.append(i[1])

plt.plot(np.arange(1, 21), ls_value, marker = 'o', label=ls_value)
plt.xticks(np.arange(0, 21, 1))
plt.xlabel("Number of Features")
plt.ylabel("Test MSE")
plt.title("Test MSE and Number of Features")
plt.show()

pair_min = ls_test[0]
for i in ls_test:
    if i[1] < pair_min[1]:
        pair_min = i
pair_min
```




```
Out[174]: [['x0',
            'x1',
            'x2',
            'x3',
            'x4',
            'x6',
            'x7',
            'x8',
            'x9',
            'x10',
            'x11',
            'x12',
            'x13',
            'x15',
            'x16',
            'x17',
            'x18',
            'x19'],
           1.3980496499585618e-27]
```

The test set size MSE is 18 with predictors ['x0','x1','x2','x3','x4','x6','x7','x8','x9','x10', 'x11','x12','x13','x15','x16','x17','x18','x19']

This model filters out x5, x14 which beta is 0 but includes x7, x10, x11 which beta is also 0 where might have some errors.

1.6

In [197]:
```
ls_beta = list(beta)
beta_adj = ls_beta[:5] + ls_beta[6:14] + ls_beta[15:]
beta_adj
```

Out[197]:
```
[-0.5425327395498198,
 0.07265167160956879,
 -0.17226137799031946,
 1.129256035657263 8,
 -0.5431565349607967,
 -0.054444997679736665,
 0.0,
 -2.0420963499964357,
 1.0841010648344358,
 -0.9050263118832786,
 0.0,
 0.0,
 -0.7273537660373857,
 1.683622286407236,
 -0.4512913802904245,
 -0.6212645196656981,
 -1.8068170634275635,
 0.3777818823249585]
```

In [198]:
```
estimated = LinearRegression().fit(X[pair_min[0]],Y)
list(estimated.coef_)
```

Out[198]:
```
[-0.5425327395498195,
 0.07265167160956587,
 -0.17226137799032645,
 1.129256035657255 6,
 -0.5431565349607991,
 -0.05444499767974047,
```

```
 -7.595865228848664e-15,
 -2.0420963499964313,
 1.0841010648344367,
 -0.9050263118832831,
 1.8732417977253785e-15,
 3.696062513560934e-15,
 -0.7273537660373872,
 1.683622286407242,
 -0.451291380290418,
 -0.6212645196656966,
 -1.8068170634275573,
 0.37778188232496035]
```

In [199]:
```
ls_difference = []
for i in range(len(beta_adj)):
    ls_difference.append(beta_adj[i] - list(estimated.coef_)[i])
ls_difference
```

Out[199]:
```
[-3.3306690738754696e-16,
 2.914335439641036e-15,
 6.994405055138486e-15,
 8.215650382226158e-15,
 2.3314683517128287e-15,
 3.802513859341161e-15,
 7.595865228848664e-15,
 -4.440892098500626e-15,
 -8.881784197001252e-16,
 4.440892098500626e-15,
 -1.8732417977253785e-15,
 -3.696062513560934e-15,
 1.5543122344752192e-15,
 -5.995204332975845e-15,
 -6.494804694057166e-15,
 -1.4432899320127035e-15,
 -6.217248937900877e-15,
 -1.8318679906315083e-15]
```

As shown in the above difference list, we would find that the difference between true value and predicted value is very small. Even for those beta = 0 predictor, the difference is also very small.

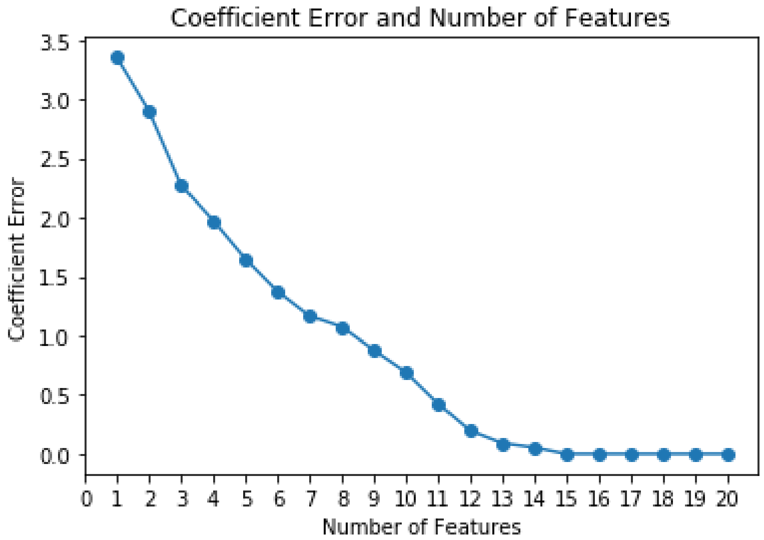
1.7

In [241]:
```
ls_array_hat = []
for k in range(len(ls_test)):
    array_hat = np.zeros(20)
    estimated = LinearRegression().fit(X[ls_test[k][0]],Y)
    ls_estimated = list(estimated.coef_)

    ls_num = []
    for i in ls_test[k][0]:
        ls_num.append(int(i[1:]))

    for j in range(len(ls_num)):
        array_hat[ls_num[j]] = ls_estimated[j]
    ls_array_hat.append(array_hat)

coef_error = []
for each_array_hat in ls_array_hat:
    coef_error.append(np.sqrt(np.sum((beta - each_array_hat) ** 2)))
coef_error
```

Out[241]:
```
[3.355301000636694,
 2.902798907254069,
 2.273867603614476,
 1.9744765922983611,
 1.650972068060046,
 1.3758446970166607,
 1.167205128597094,
 1.0763910748788779,
 0.8788763558912417,
 0.6927534261158053,
 0.42712103128811,
 0.19664697625825375,
 0.09166771975588732,
 0.05479864502792337,
 4.325981602464918e-15,
```

```
 3.9898632940070615e-15,
 4.595609446404584e-15,
 1.9612511362660355e-14,
 3.936681421010751e-15,
 4.027548099429476e-15]
```

In [242]:
```
plt.plot(np.arange(1, 21), coef_error, marker = 'o', label=coef_error)
plt.xticks(np.arange(0, 21, 1))
plt.xlabel("Number of Features")
plt.ylabel("Coefficient Error")
plt.title("Coefficient Error and Number of Features")
plt.show()
```



The coefficient error graph is similar to the test MSE graph and it shows that when the number of features increase, the coefficient error decreases, which is similar to the test/training MSE graph. Also, it is worth noting that the graph first drops quickly than reaches its minimum when n = 19, which is the same as the forementioned graph.

Application Exercises

2.1

In [243]:
```
url_train = "https://raw.githubusercontent.com/macss-model20/problem-se
t-3/master/data/gss_train.csv"
url_test = "https://raw.githubusercontent.com/macss-model20/problem-set
-3/master/data/gss_test.csv"
gss_train = pd.read_csv(url_train)
gss_test = pd.read_csv(url_test)
gss_train.head(5)
```

Out[243]:

	age	attend	authoritarianism	black	born	childs	colath	colrac	colcom	colmil	...	zodiac_(
0	21	0	4	0	0	0	1	1	0	1	...	
1	42	0	4	0	0	2	0	1	1	0	...	
2	70	1	1	1	0	3	0	1	1	0	...	
3	35	3	2	0	0	2	0	1	0	1	...	
4	24	3	6	0	1	3	1	1	0	0	...	

5 rows × 78 columns

In [244]:
```
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)
lr = LinearRegression().fit(x_train, y_train)
mse = mean_squared_error(lr.predict(x_test), y_test)
print("the test MSE: " + str(mse))
```

```
the test MSE: 63.213629623014995
```

2.2

In [245]:
```
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:' + str(ridge_mse))
```

```
the test MSE of ridge:62.4992024395781
```

2.3

In [246]:

```
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:' + str(lasso_mse))
lasso_num = (lasso.coef_ != 0).sum()
print('Number of non-zero for lasso:' + str(lasso_num))
```

```
the test MSE of lasso:62.7780157899344
Number of non-zero for lasso:24
```

2.4

In [249]:

```
import warnings
warnings.filterwarnings('ignore')
elastic = ElasticNetCV(cv=10, alphas = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
, 0.7, 0.8, 0.9, 1]).fit(x_train, y_train)
elastic_mse = mean_squared_error(elastic.predict(x_test), y_test)
print('the test MSE of elastic: ' + str(elastic_mse))
elastic_num = (elastic.coef_ != 0).sum()
print('Number of non-zero for elastic: ' + str(elastic_num))
print('l1 ratio: ' + str(elastic.l1_ratio_))
print('alpha: ' + str(elastic.alpha_))
```

```
the test MSE of elastic: 62.5070860872212
Number of non-zero for elastic: 40
l1 ratio: 0.5
alpha: 0.1
```

5.

All four models have a similar MSE around 62-63, which means there are no significant differences in MSE between different approaches. Generally speaking, we don't predict the egalitaianism well, and we might have another model other than these models, since egalitarianism is range from (1,35).