Modellling HW3

In [89]:

```
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sb
from sklearn.model_selection import train_test_split, GridSearch
CV
from sklearn.linear_model import LinearRegression, Ridge, LassoC
V, ElasticNetCV
from sklearn.metrics import mean_squared_error
import itertools
import statsmodels.api as sm
from mlxtend.feature_selection import SequentialFeatureSelector
as sfs
```

Conceptual Exercise (1-7)

Exercise1

In [90]:

```
np.random.seed(0)
X = np.array([np.random.normal(0, 1, 1000) for n in range(20)])
beta = np.array([np.random.randint(0, 5) for n in range(20)]).re
shape(-1, 1)
beta
```

Out[90]:

```
array([[4],
       [1],
       [3],
       [1],
       [3],
       [4],
       [0],
       [2],
       [0],
       [2],
       [4],
       [4],
       [0],
       [3],
       [2],
       [1],
       [1],
       [4],
       [1],
       [4]])
```

In [91]:

```
error = np.random.normal(0, 1, 1000)
y = np.sum(X*beta, axis=0) + error
```

Exercise2

In [92]:

```
X = X.transpose()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
ize=900)
```

Exercise3-6

In [93]:

```
model = LinearRegression()
sfs_sub = sfs(model, k_features=5, forward=True,
              scoring='neg_mean_squared_error', cv=5)
sfs_sub.fit(X_train, y_train)
sfs_sub.k_feature_idx_
```

Out[93]:

```
(7, 10, 11, 17, 19)
```

In [94]:

```
count = []
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)
    count.append(i)
    score.append(-sfs_fit.k_score_)

feature_mse = pd.DataFrame({"feature count": count, "mse": score
})
feature_mse
```

Out[94]:

	feature count	mse
0	1	115.313212
1	2	97.031543
2	3	82.950805
3	4	72.536992
4	5	61.217282
5	6	48.477466
6	7	38.026497
7	8	29.027582
8	9	18.151328
9	10	13.384845
10	11	10.565883
11	12	7.331903
12	13	5.011386
13	14	3.736031
14	15	2.673806
15	16	1.747950
16	17	1.161935
17	18	1.145502
18	19	1.179278
19	20	1.272666

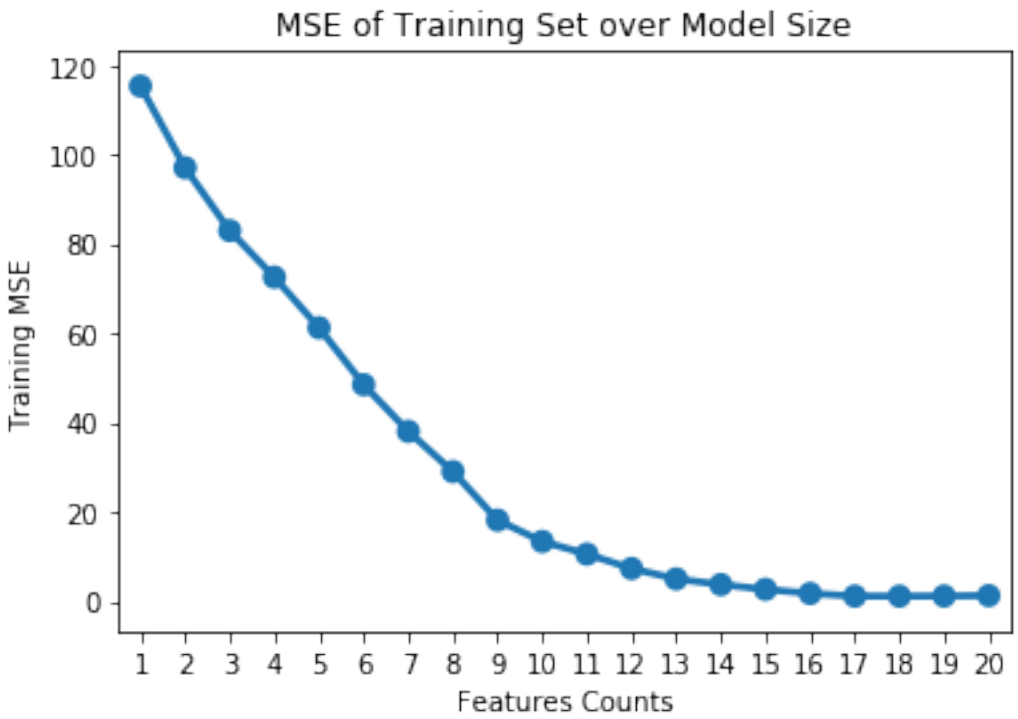
```
The training set MSE take on its minimum value at model si
ze equals 17.
```

In [95]:

```
ax = sb.pointplot(x="feature count", y="mse", data=feature_mse)
ax.set(xlabel='Features Counts', ylabel='Training MSE',
       title='MSE of Training Set over Model Size')
```

Out[95]:

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



In [96]:

```
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 = mean_squared_error(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": fea
ture_idx})
test_mse
```

Out[96]:

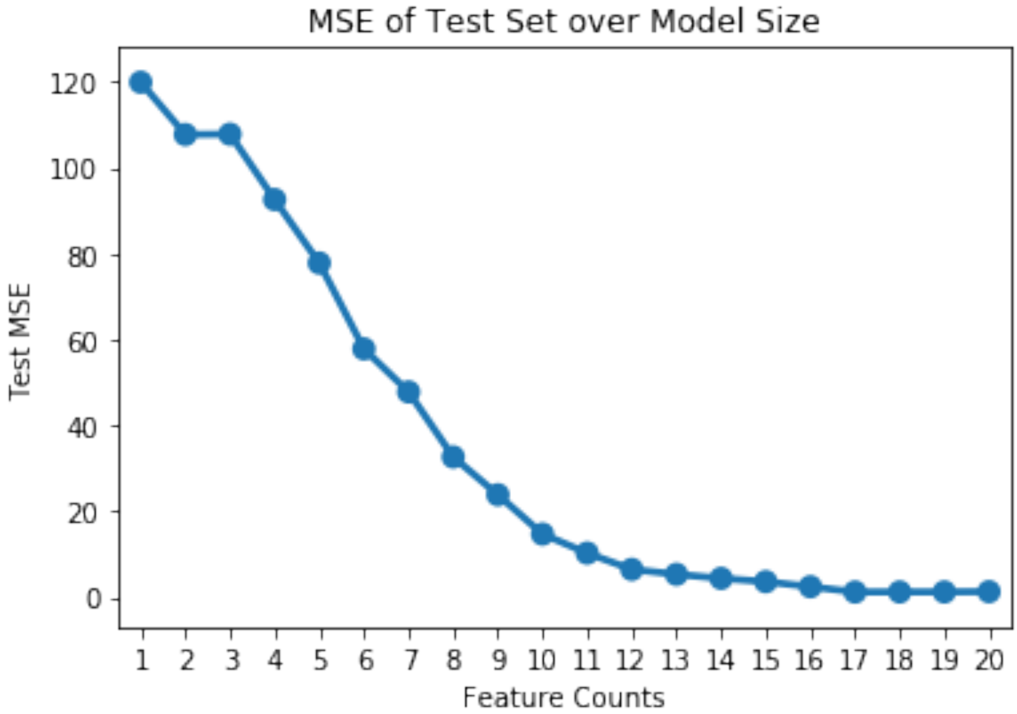
	feature_num	mse	feature index
0	1	119.955104	[17]
1	2	107.762531	[11, 17]
2	3	107.852190	[7, 11, 17]
3	4	92.617523	[7, 11, 17, 19]
4	5	77.836823	[7, 10, 11, 17, 19]
5	6	57.827346	[5, 7, 10, 11, 17, 19]
6	7	47.802566	[4, 5, 7, 10, 11, 17, 19]
7	8	32.649114	[0, 4, 5, 7, 10, 11, 17, 19]
8	9	23.841517	[0, 2, 4, 5, 7, 10, 11, 17, 19]
9	10	14.680115	[0, 2, 4, 5, 7, 10, 11, 13, 17, 19]
10	11	10.250804	[0, 2, 4, 5, 7, 9, 10, 11, 13, 17, 19]
11	12	6.449264	[0, 2, 4, 5, 7, 9, 10, 11, 13, 14, 17, 19]
12	13	5.302897	[0, 2, 4, 5, 7, 9, 10, 11, 13, 14, 15, 17, 19]
13	14	4.356484	[0, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 15, 17, 19]
14	15	3.596682	[0, 1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 15, 1...
15	16	2.417552	[0, 1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 15, 1...
16	17	1.181086	[0, 1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 15, 1...
17	18	1.196760	[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 15...
18	19	1.201580	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14,...
19	20	1.261861	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...

In [97]:

```
ax = sb.pointplot(x="feature_num", y="mse", data=test_mse)
ax.set(xlabel='Feature Counts', ylabel='Test MSE',
       title='MSE of Test Set over Model Size')
```

Out[97]:

```
[Text(0, 0.5, 'Test MSE'),
 Text(0.5, 0, 'Feature Counts'),
 Text(0.5, 1.0, 'MSE of Test Set over Model Size')]
```



In [98]:

```
best_fea = test_mse["feature index"][17]
best_fea
```

Out[98]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 15, 16,
17, 18, 19]
```

The test set MSE take on its minimum value at model size 17.

In [102]:

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

Out[102]:

```
array([4.03466628, 0.98505953, 2.9757184 , 1.0099782
5, 2.97940753,
       3.98473857, 0.01114542, 1.97324169, 1.9312721
, 3.97571264,
       4.01008082, 3.02123082, 2.06192281, 1.0113238
9, 0.99443939,
       3.94037051, 1.02869798, 4.010057  ])
```

In [103]:

```
true_model = beta[best_fea]
true_coef = []

for i in range(len(true_model)):
    true_coef.append(true_model[i][0])

comparison = pd.DataFrame({"feature": best_fea,
                           "best model": best_model.coef_,
                           "true model": true_coef})
comparison
```

Out[103]:

	feature	best model	true model
0	0	4.034666	4
1	1	0.985060	1
2	2	2.975718	3
3	3	1.009978	1
4	4	2.979408	3
5	5	3.984739	4
6	6	0.011145	0
7	7	1.973242	2
8	9	1.931272	2
9	10	3.975713	4
10	11	4.010081	4
11	13	3.021231	3
12	14	2.061923	2
13	15	1.011324	1
14	16	0.994439	1
15	17	3.940371	4
16	18	1.028698	1
17	19	4.010057	4

```
The list above shows that the coefficients of our best mod
els are somehow very close to the true betas.
```

Exercise7

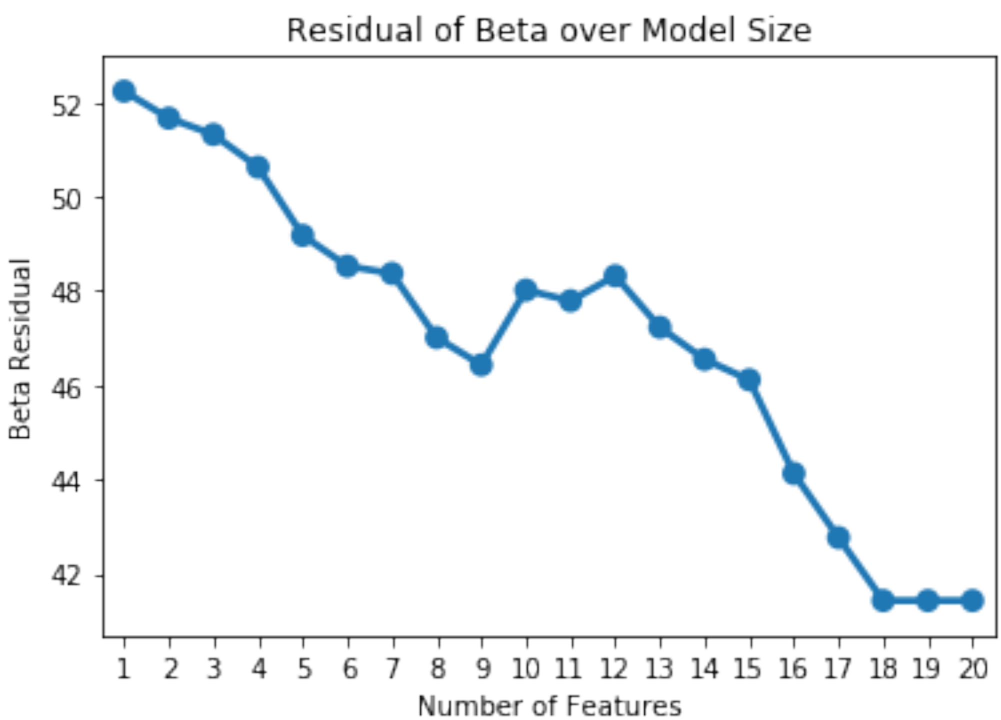
In [88]:

```
feature_num = list(i for i in range(1,21))
beta_res = []
for res in range(1,21):
    fea = best_fea[:res]
    lm = model.fit(X_train[:,fea], y_train)
    coef_hat = lm.coef_
    b = np.zeros(20)
    for i ,f in enumerate(fea):
        b[fea] = coef_hat[i]
    calc = (((beta - b)**2).sum())**.5
    beta_res.append(calc)

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

Out[88]:

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



The graph above demonstrates that Beta residuals are overall decreasing, however, low beta residual does not necessarily mean a low test mse. We can also see that 18 features yield the lowest Beta residual, showing a consistent pattern with our previous mse analysis.

In [ ]:

```

```

Application exercises(1-5)

In [38]:

```
gss_train = pd.read_csv('data/gss_train.csv')
gss_test = pd.read_csv('data/gss_test.csv')
gss_train.head()
```

Out[38]:

	age	attend	authoritarianism	black	born	childs	colath	colrac	co
0	21	0	4	0	0	0	1	1	
1	42	0	4	0	0	2	0	1	
2	70	1	1	1	0	3	0	1	
3	35	3	2	0	0	2	0	1	
4	24	3	6	0	1	3	1	1	

5 rows × 78 columns

In [87]:

```
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']

def model_mse(model):
    fit_model = model().fit(X_train, y_train)
    MSE = mean_squared_error(fit_model.predict(X_test), y_test)
    mse = "The test MSE of this model is: " + str(MSE)
    return mse, fit_model
```

In [88]:

```
#Least square linear model
model_mse(LinearRegression)[0]
```

Out[88]:

```
'The test MSE of this model is: 63.213629623014995'
```

In [143]:

```
#Ridge model
para_ridge = {'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0
.9, 1.0]}
ridge_regressor = GridSearchCV(Ridge(), para_ridge, scoring='neg
_mean_squared_error', cv=10)
ridge_regressor.fit(X_train, y_train)
mse_ridge = mean_squared_error(ridge_regressor.predict(X_test),
y_test)
print('The test MSE for ridge regression is', mse_ridge)
```

```
The test MSE for ridge regression is 63.05236636947819
```

In [144]:

```
#Lasso model
para_lasso = {'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0
.9, 1.0]}
lasso_regressor = GridSearchCV(Lasso(), para_lasso, scoring='neg
_mean_squared_error', cv=10)
lasso_regressor.fit(X_train, y_train)
mse_lasso = mean_squared_error(lasso_regressor.predict(X_test),
y_test)
print('The test MSE for ridge regression is', mse_lasso)
Lasso(alpha=0.01).fit(X_train, y_train)
print('There are ' + str(((lasso.coef_ != 0).sum())) + ' non-zer
o coefficient estimates.')
```

```
The test MSE for ridge regression is 62.778415554773
89
There are 65 non-zero coefficient estimates.
```

In [139]:

```
#Elasstic net regression
alpha_ela = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
ela_regressor = ElasticNetCV(alpha_ela, cv=10).fit(X_train, y_tr
ain)
mse_ela = mean_squared_error(ela_regressor.predict(X_test), y_te
st)
print('The test MSE for ela regression is', mse_ela)
print('There are ' + str(((ela_regressor.coef_ != 0).sum())) + '
non-zero coefficient estimates.')
```

```
The test MSE for ela regression is 62.7780157899344
There are 24 non-zero coefficient estimates.
```

From the MSE results of models above, it seems that there is no big difference among these four and we don't have very good prediction on egalitarianism. However, the Lasso and the Elassitic net model are a bit better than the rest two, with test MSE being around 62.78. We also discover that regularization improves the result, for the reason that it reduces the overfitting problem. We might need to consider using non-linear models to fit the data and see the MSE results. Or we could use some validation set.

In [ ]:

```

```