In [1]:
```
import numpy as np
import pandas as pd
import itertools
import time

import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

import statsmodels.api as sm
```

Conceptual exercise

Training/test error for subset selection

1

In [206]:
```
from sklearn.datasets import make_blobs
import random

# generate coefficients
random.seed = 1918
coes = []
zero_index = np.random.choice(20, 10, replace=False)
for i in range(20):
    if i in zero_index:
        coes.append(0)
    else:
        coes.append(random.uniform(-5, 5))
coes = np.array(coes)

# generate observations and responses
X, y = make_blobs(n_samples=1000, n_features=20, random_state=0)
y = []
for j in range(1000):
    y.append(sum(X[j] * coes) + 0.2 * np.random.randn())

y = np.array(y)
```

2

In [207]:
```
data = np.c_[X, y]
train, test = train_test_split(data, test_size=0.9, train_size=0.1, rand
om_state=1227)
X_train = train[:, :20]
y_train = train[:, 20]
X_test = test[:, :20]
y_test = test[:, 20]
```

3

Reference:
https://github.com/JWarmenhoven/ISLR-python (https://github.com/JWarmenhoven/ISLR-python)

In [199]:
```
def fit_linear_reg(X_train, y_train, X_test, y_test):
    lr = linear_model.LinearRegression(fit_intercept=True)
    lr.fit(X_train, y_train)
    MSE_tr = mean_squared_error(y_train, lr.predict(X_train))
    MSE_test = mean_squared_error(y_test, lr.predict(X_test))

    return MSE_tr, MSE_test, lr.coef_
```

note

Coen D. Needell：

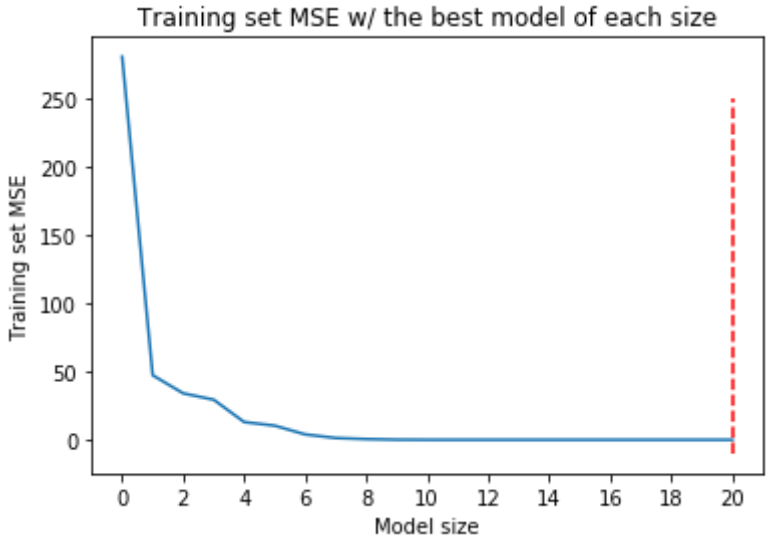
I'd guess you're checking all possible models, which runs like 2^n, since the model is linear you can assume independent variables. This means that every model of size k will include all parameters included in the best models of size p < k. Instead of looping over all n choose k models, just loop over all n-k parameters not currently included in the smaller model, then add the one which reduces error the most to your list of parameters.

In [208]:
```
co_ls = list(range(20))
best_combo = []
temp = []
MSE_tr_ls = [np.var(train[:,20])]
MSE_test_ls = [np.var(test[:,20])]
coef_hat_ls = []
for times in range(20):
    MSE_tr = 99999
    for i in co_ls:
        temp.append(i)
        result = fit_linear_reg(X_train[:, temp], y_train, X_test[:, tem
p], y_test)
        if result[0] < MSE_tr:
            MSE_tr = result[0]
            MSE_test = result[1]
            coef_hat = result[2]
            best_factor = i
        temp.remove(i)

    temp.append(best_factor)
    coef_hat_ls.append(coef_hat)
    co_ls.remove(best_factor)
    MSE_tr_ls.append(MSE_tr)
    MSE_test_ls.append(MSE_test)
    best_combo.append(tuple(temp))
```

In [209]:
```
best_combo
```

Out[209]:
```
[(16,),
 (16, 6),
 (16, 6, 15),
 (16, 6, 15, 18),
 (16, 6, 15, 18, 7),
 (16, 6, 15, 18, 7, 12),
 (16, 6, 15, 18, 7, 12, 2),
 (16, 6, 15, 18, 7, 12, 2, 9),
 (16, 6, 15, 18, 7, 12, 2, 9, 3),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17, 1),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17, 1, 14),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17, 1, 14, 5),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17, 1, 14, 5, 4),
 (16, 6, 15, 18, 7, 12, 2, 9, 3, 8, 13, 19, 0, 10, 17, 1, 14, 5, 4, 1
1)]
```

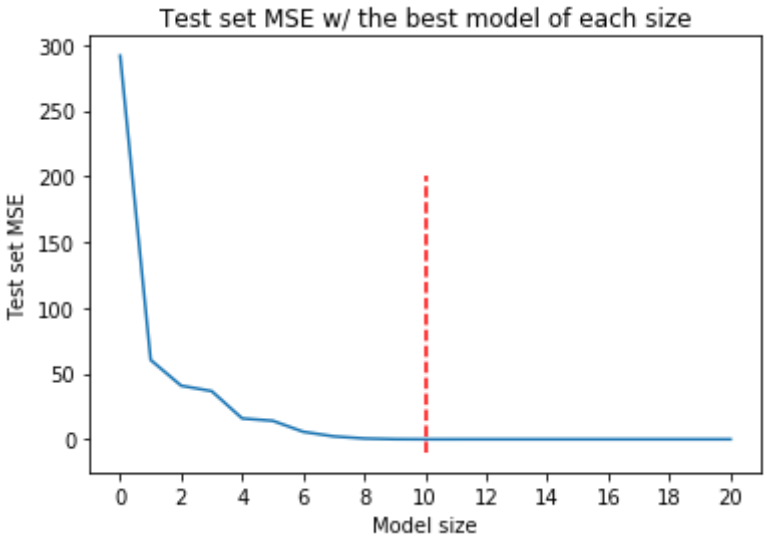
In [210]:
```
p1=plt.plot(list(range(21)), MSE_tr_ls, label='')

p1 = MSE_tr_ls.index(min(MSE_tr_ls))
plt.vlines(p1, -10, 250, colors = "red", ls='--')
plt.xlabel('Model size')
plt.ylabel('Training set MSE');
plt.title('Training set MSE w/ the best model of each size')
plt.xticks(np.linspace(0, 20, 11))
plt.show();
```



Based on the graph above, when model size=20, the training set MSE takes on its minimum value.

4 & 5

In [211]:
```
p2 = MSE_test_ls.index(min(MSE_test_ls))

p1=plt.plot(list(range(21)), MSE_test_ls, label='')

plt.vlines(p2, -10, 200, colors = "red", ls='--')
plt.xlabel('Model size')
plt.ylabel('Test set MSE');
plt.title('Test set MSE w/ the best model of each size')
plt.xticks(np.linspace(0, 20, 11))
plt.show();
```



Based on the graph above, when model size=11, the test set MSE takes on its minimum value. However, the MSE has only slight change around 11.

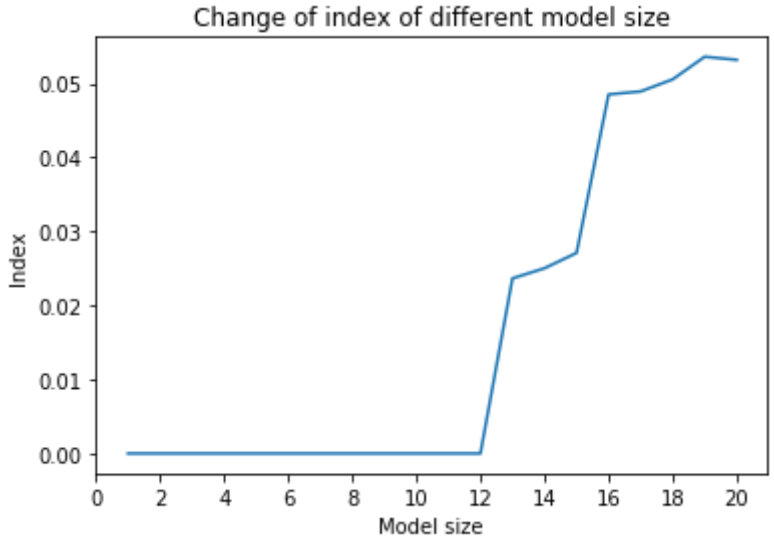
6

For training set MSE, it is reasonable to see such result since the more predictors a model contains, the more flexible it would be. And as the number of predictors increases, the model fits the training data better. The optimal model size based on test data, on the other hand, is closer to the real number of coefficients.

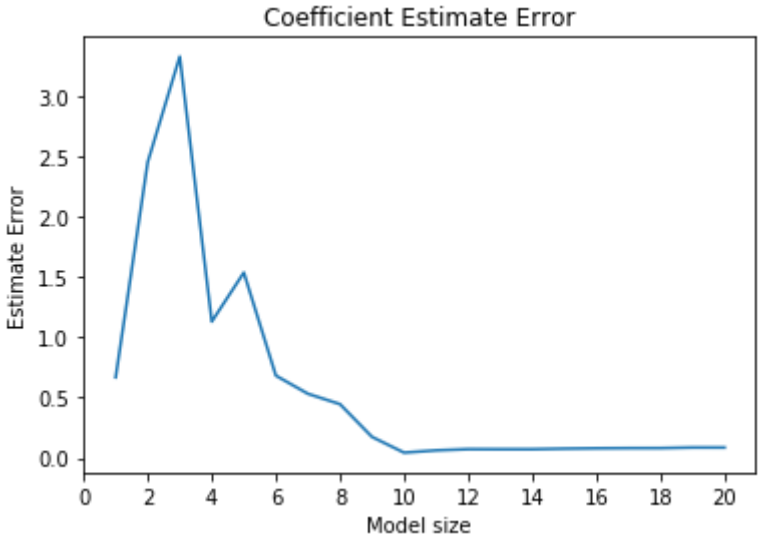
7

In [212]:

```
import math

# keep all coefficients

index = []
for i in range(20):
    ls = np.c_[coes, np.zeros(20)]
    for j in range(i + 1):
        ls[best_combo[i][j], 1] = coef_hat_ls[i][j]
    temp = math.sqrt(sum((ls[0] - ls[1]) ** 2))
    index.append(temp)

p1=plt.plot(list(range(1, 21)), index, label='')
plt.xlabel('Model size')
plt.ylabel('Index');
plt.title('Change of index of different model size')
plt.xticks(np.linspace(0, 20, 11))
plt.show();
```



In [214]:
```
# only keep chosen coefficients

index = []
for i in range(20):
    temp = 0
    ls = np.c_[coes, np.zeros(20)]
    for j in range(i + 1):
        temp += (ls[best_combo[i][j], 0] - coef_hat_ls[i][j]) ** 2

    index.append(math.sqrt(temp))


p1=plt.plot(list(range(1, 21)), index, label='')
plt.xlabel('Model size')
plt.ylabel('Estimate Error');
plt.title('Coefficient Estimate Error')
plt.xticks(np.linspace(0, 20, 11))
plt.show();

# This plot is similar with the test MSE plot,
# that as the model size increases, the coefficient error decreases
# and has the minimum value when p=10
# Both suggest the optimal subset selection.
```



Application exercises

1

In [237]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')

X_train, y_train = train.drop('egalit_scale', axis=1), train['egalit_sca
le']
X_test, y_test = test.drop('egalit_scale', axis=1), test['egalit_scale']
```

In [238]:
```
lr = linear_model.LinearRegression(fit_intercept=True)
lr.fit(X_train, y_train)

MSE_test = mean_squared_error(y_test, lr.predict(X_test))
print('The test MSE is ' + str(MSE_test))
```

```
The test MSE is 63.21362962301499
```

2

In [239]:
```
from sklearn.model_selection import cross_val_score
import glmnet as gln
```

In [240]:
```
ridge = gln.ElasticNet(alpha=0, scoring='mean_squared_error')
ridge.fit(X_train, y_train)

# Lambda with best CV performance
lambda_best = ridge.lambda_max_

pred = ridge.predict(X_test, lamb=lambda_best)
MSE_test = mean_squared_error(y_test, pred)

print('The best lambda is ' + str(lambda_best))
print('The test MSE is ' + str(MSE_test))
```

```
The best lambda is 2.6850362757695585
The test MSE is 60.89784231422841
```

3 Lasso

In [241]:
```
from sklearn.linear_model import Lasso, LassoCV

lasso = Lasso(max_iter=10000)
lassocv = LassoCV(alphas=None, cv=10, random_state=4024)
lassocv.fit(X_train, y_train)
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
MSE_test = mean_squared_error(y_test, lasso.predict(X_test))

print('The best lambda is ' + str(lassocv.alpha_))
print('The test MSE is ' + str(MSE_test))
```

```
The best lambda is 0.09991906177130906
The test MSE is 62.7780157899344
```

In [242]:
```
a = pd.Series(lasso.coef_, index=X_train.columns)
non_zero = sum((a != 0).astype(int))
print('The number of non-zero coefficient estimates is ' + str(non_zero
))
```

```
The number of non-zero coefficient estimates is 24
```

4 Elastic net

In [243]:
```
from sklearn.linear_model import ElasticNetCV
cv_values = np.arange(0.1, 1.1, 0.1)

regr = ElasticNetCV(l1_ratio=cv_values, cv=10, random_state=1917)
regr.fit(X_train, y_train)
```

Out[243]:
```
ElasticNetCV(alphas=None, copy_X=True, cv=10, eps=0.001, fit_intercept=
True,
             l1_ratio=array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.
9, 1. ]),
             max_iter=1000, n_alphas=100, n_jobs=None, normalize=False,
             positive=False, precompute='auto', random_state=1917,
             selection='cyclic', tol=0.0001, verbose=0)
```

In [244]:
```
print(str(regr.alpha_))
print(str(regr.l1_ratio_))
```

```
0.09991906177130906
1.0
```

In [247]:
```
MSE_test = mean_squared_error(y_test, regr.predict(X_test))
a = pd.Series(lasso.coef_, index=X_train.columns)
non_zero = sum((a != 0).astype(int))

print('The test MSE is ' + str(MSE_test))
print('The number of non-zero coefficient estimates is ' + str(non_zero
))
```

```
The test MSE is 62.7780157899344
The number of non-zero coefficient estimates is 24
```

5

Among all methods, the ridge regression model yields the lowest MSE.However, all test MSEs perform similarly and obtained by the methods above are >60. it may be hard to predict an individual's egalitarianism accurately.