Resampling Methods and Tree-based Methods

In [2]:

```
# %load ../standard_import.txt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as skl_lm
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor

%matplotlib inline
plt.style.use('seaborn-white')
```

Load dataset

Dataset available on http://www-bcf.usc.edu/~gareth/ISL/data.html (http://www-bcf.usc.edu/~gareth/ISL/data.html)

In [3]:

```
boston_df = pd.read_csv('../ISLR/Data/Boston.csv')
boston_df.info()

X_boston = boston_df.drop('medv', axis=1)
y_boston = boston_df.medv
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
crim       506 non-null float64
zn         506 non-null float64
indus      506 non-null float64
chas       506 non-null int64
nox        506 non-null float64
rm         506 non-null float64
age        506 non-null float64
dis        506 non-null float64
rad        506 non-null int64
tax        506 non-null int64
ptratio    506 non-null float64
black      506 non-null float64
lstat      506 non-null float64
medv       506 non-null float64
dtypes: float64(11), int64(3)
memory usage: 55.4 KB
```

5.1 Cross-Validation

1. Boston dataset 이용 figure 5.2, 5.4 재현

Figure 5.2 - Validation Set Approach

Using Polynomial feature generation in scikit-learn
http://scikit-learn.org/dev/modules/preprocessing.html#generating-polynomial-features (http://scikit-learn.org/dev/modules/preprocessing.html#generating-polynomial-features)

In [3]:

```
n_features = np.arange(1,11)
r_state = np.arange(0,10)

X, Y = np.meshgrid(n_features, r_state, indexing='ij')
Z = np.zeros((n_features.size,r_state.size))


# Generate 10 random splits of the dataset
for (i,j),v in np.ndenumerate(Z):

    X_train, X_test, y_train, y_test = train_test_split(X_boston, y_boston, test_size=0.5, random_st
    regr = RandomForestRegressor(n_estimators=100, max_features=X[i, j])

    regr.fit(X_train, y_train)
    pred = regr.predict(X_test)
    Z[i,j]= mean_squared_error(y_test, pred)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))

# Left plot (first split)
ax1.plot(X.T[0],Z.T[0], '-o')
ax1.set_title('Random split of the data set')

# Right plot (all splits)
ax2.plot(X,Z)
ax2.set_title('10 random splits of the data set')

for ax in fig.axes:
    ax.set_ylabel('Mean Squared Error')

    ax.set_xlabel('Number of Max Features')
    ax.set_xlim(0.5,10.5)
    ax.set_xticks(range(2,11,2))
```

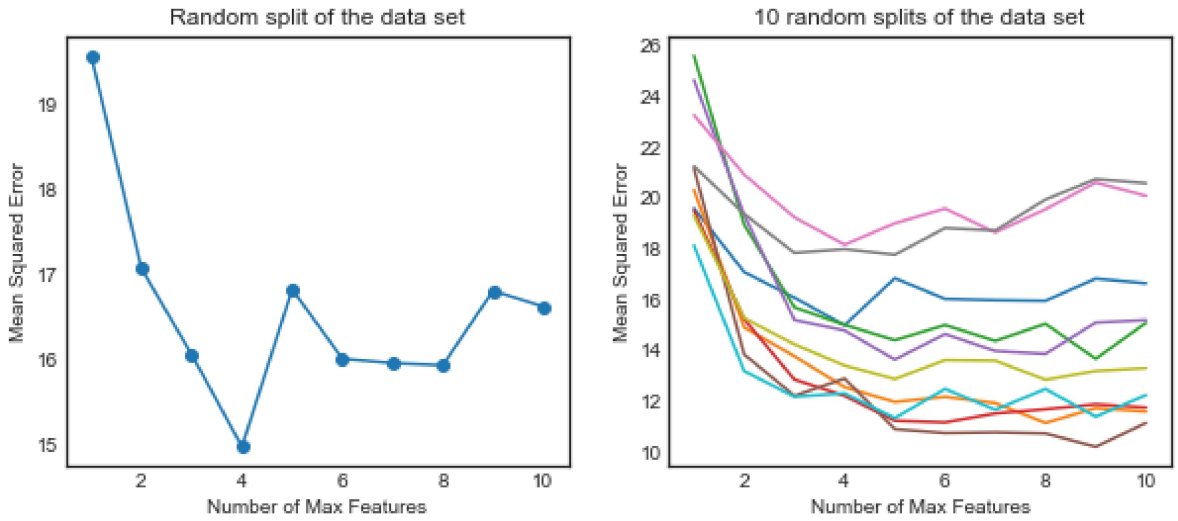


Figure 5.4

In [4]:

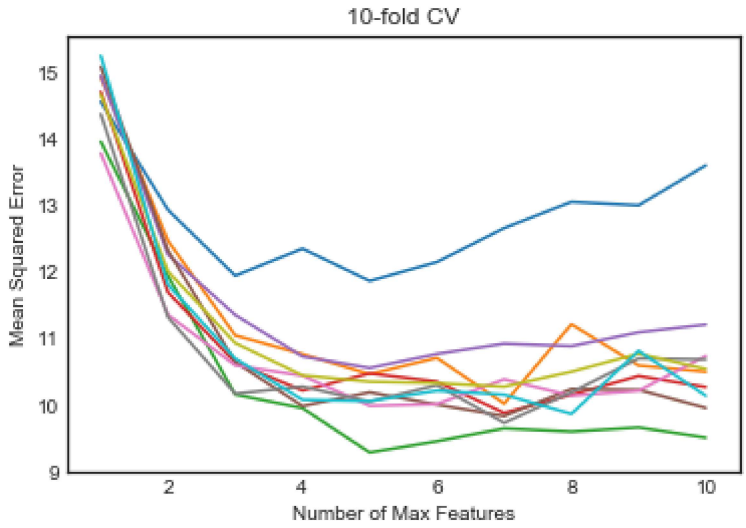
```
n_features = np.arange(1,11)
r_state = np.arange(0,10)
```

In [5]:

```
# k-fold CV
folds = 10

X, Y = np.meshgrid(n_features, r_state, indexing='ij')
Z = np.zeros((n_features.size,r_state.size))

for (i,j),v in np.ndenumerate(Z):
    regr = RandomForestRegressor(n_estimators=100, max_features=X[i, j])
    kf_10 = KFold(n_splits=folds, random_state=Y[i,j], shuffle=True)
    Z[i,j] = cross_val_score(regr, X_boston, y_boston, cv=kf_10, scoring='neg_mean_squared_error').m
```

In [6]:

```
# Right plot
plt.plot(X,Z*-1)
plt.title('10-fold CV')

plt.ylabel('Mean Squared Error')
plt.xlabel('Number of Max Features')
plt.xlim(0.5,10.5)
plt.xticks(range(2,11,2))
plt.show()
```



2. 1번 결과 설명

사용하지 않은 데이터로부터 구한 Validation Error를 통해 Test Error를 추정할 수 있다. 이때 사용할 수 있는 방법 중 두 가지가 Validation Set Approach와 K-fold Cross-validation Approach다. Validation set approach는 데이터를 무작위로 나눈 뒤(training set, validation set) Training Set에 대해 학습하고, Validation set으로 Validation Error를 구하는 방법이다. 그런데 Validation Set Approach는 아래 두 가지 단점이 있다.

1. 위 그림(5.2)에서 볼 수 있듯이 Validation Set에 대한 error는 random seed를 어떻게 두느냐에 따라 많이 달라진다. 단 한번만 나누기 때문에 training set/validation set 각각에 어떤 데이터가 포함되었는지에 따라 결과가 차이가 많이 난다.
2. 데이터의 일부만 사용하기 때문에, 성능을 안 좋게 예측할 수 있다. statistical method는 일반적으로 적은 양의 데이터에 대해 더 안 좋은 성능을 보이기 때문이다. 위에서는 0.5/0.5 비율로 나누었기 때문에 절반 정도의 데이터를 썼을 때 성능과 유사한 결과가 나왔을 것이다.

Validation set appoach의 단점을 보완하기 위해 k-Fold Cross validation을 대신 사용할 수 있다. 이 방법은 데이터를 k 그룹으로 나눈 뒤, k-1개의 그룹으로 학습하고 학습에 사용하지 않은 그룹 하나로 Validation Error를 구하는 과정을 각기 다른 k번에 대해 수행한 뒤 평균낸다. 이때 장점은 아래와 같다.

1. 10번 반복해 평균을 내므로 Validation Error의 variation이 Validation set approach보다 작다. 5.4 그림의 error 값들이 5.2 그림보다 더 모여있는 것을 확인할 수 있다.
2. 10-fold 예측 시 전체의 0.9만큼 학습에 사용한다. Validation set approach에서는 전체에서 0.5를 사용함으로써 test error를 과추정하는 문제가 있었는데, 0.9를 사용함으로써 test error 과추정 문제를 완화하게 된다. 만약 validation set approach에서 학습 데이터 비율을 0.9를 사용한다면 10-fold cross validation과 유사하게 사이즈 감소에 따른 test error 과추정 문제를 완화할 수 있겠지만 이때는 validation error의 variance가 더 커질 것이다.

3. Feature Importance 그림

In [8]:

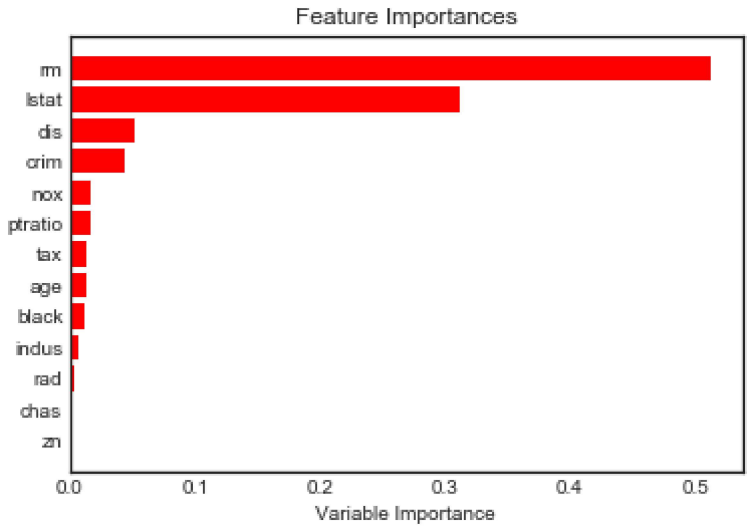
```
regressor = RandomForestRegressor(n_estimators=500)
x_train, x_test, y_train, y_test = train_test_split(X_boston, y_boston)
regressor.fit(x_train, y_train)
```

Out[8]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [9]:

```
importances = regressor.feature_importances_
feature_names = list(X_boston.columns)

z = zip(feature_names, importances)
z = sorted(z, key=lambda pair: pair[1], reverse=False)
feature_names = [x for x, _ in z]
importances = [y for _, y in z]

plt.figure()
plt.title("Feature Importances")
plt.barh(range(X_boston.shape[1]), importances,
       color="r", align="center")
# If you want to define your own labels,
# change indices to a list of labels on the following line.
plt.yticks(range(X_boston.shape[1]), feature_names)
plt.xlabel('Variable Importance')

plt.show()
```



4. GradientBoostingRegressor function

Figure 5.2 재현

In [4]:

```
from sklearn.ensemble import GradientBoostingRegressor

n_estimators = np.arange(10, 221, 30)
r_state = np.arange(0,10)

X, Y = np.meshgrid(n_estimators, r_state, indexing='ij')
Z = np.zeros((n_estimators.size,r_state.size))


# Generate 10 random splits of the dataset
for (i,j),v in np.ndenumerate(Z):

    X_train, X_test, y_train, y_test = train_test_split(X_boston, y_boston, test_size=0.5, random_st
    regr = GradientBoostingRegressor(n_estimators=X[i, j])

    regr.fit(X_train, y_train)
    pred = regr.predict(X_test)
    Z[i,j]= mean_squared_error(y_test, pred)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))

# Left plot (first split)
ax1.plot(X.T[0],Z.T[0], '-o')
ax1.set_title('Random split of the data set')

# Right plot (all splits)
ax2.plot(X,Z)
ax2.set_title('10 random splits of the data set')

for ax in fig.axes:
    ax.set_ylabel('Mean Squared Error')
    ax.set_xlabel('Number of Trees')
    ax.set_xticks(list(n_estimators))
```

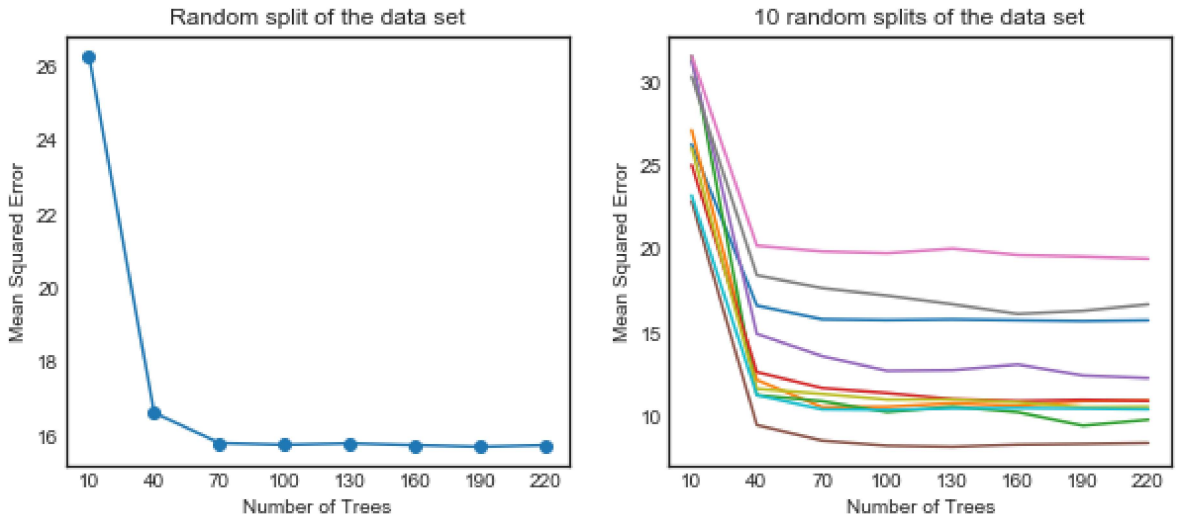


Figure 5.4 재현

In [5]:

```
# k-fold CV
folds = 10

X, Y = np.meshgrid(n_estimators, r_state, indexing='ij')
Z = np.zeros((n_estimators.size,r_state.size))

for (i,j),v in np.ndenumerate(Z):
    regr = GradientBoostingRegressor(n_estimators=X[i, j])
    kf_10 = KFold(n_splits=folds, random_state=Y[i,j], shuffle=True)
    Z[i,j] = cross_val_score(regr, X_boston, y_boston, cv=kf_10, scoring='neg_mean_squared_error').m
```

In [6]:

```
# Right plot
plt.plot(X,Z*-1)
plt.title('10-fold CV')

plt.ylabel('Mean Squared Error')
plt.xlabel('Number of Trees')
plt.xticks(list(n_estimators))
plt.show()
```