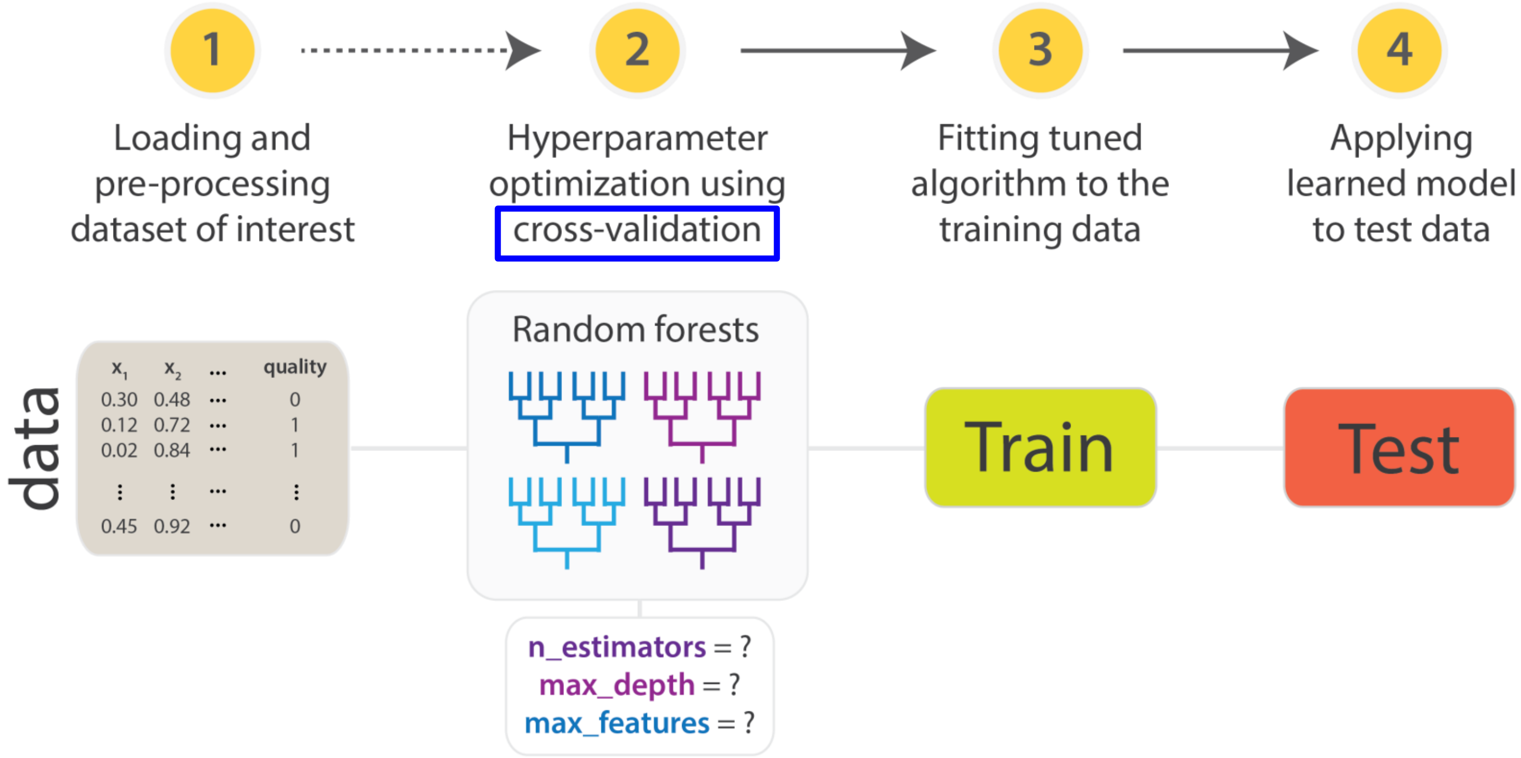
4. Cross Validation

KUBIG 학술부

KU-BIG

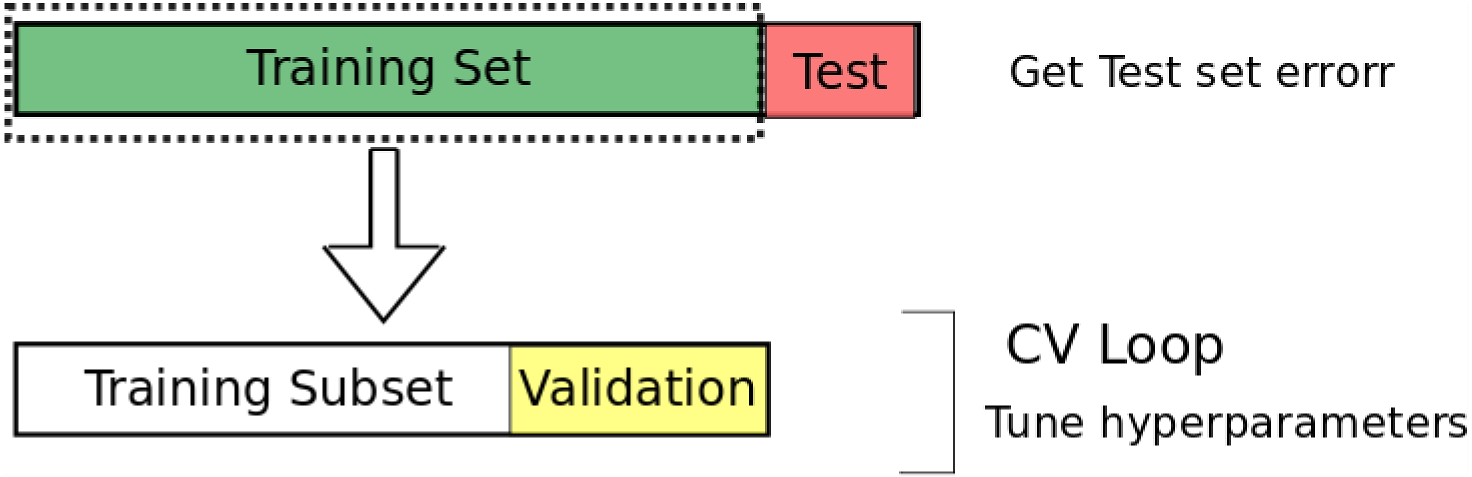
Hyperparameter tuning



Index

1. Hold-out Cross Validation
2. LOOCV (leave-one-out-cross-validation)
3. K-fold Cross Validation
4. Nested Cross Validation

1. Hold-out Cross Validation



0. 데이터를 Training set과 Test set으로 나눈다.

1. Training set을 다시 Training Subset과 Validation set으로 한번 나눈다.

2. Training Subset으로 학습한 모델이 Validation set에서 가장 좋은 성능을 내는 hyperparameter를 선택

In [1]:
```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,ShuffleSplit,GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

In [13]:
```python
rsf=rs.split(train_set_X,train_set_y)

for iteration, data in enumerate(rsf, start=1):
    print('\n{} {:^61}'.format('Iteration', 'Training subset observations'))
    print('\n{:^9} {}'.format(iteration, data[0]))
    print('\n{:^61}'.format('Validation set observations'))
    print('\n{:^9}'.format(str(data[1])))
```

```
Iteration                    Training subset observations

    1       [ 35  34  84  43  46 102  49  39  96  33  87  12  19 106 115  17   7  95
 107   0   1  18  69  92  61 111  22 116  79  66  23  13  86  58  53  45
 113  77 112  31 109  16  60  29  90  38  91  72  51  55  24  81  41  40
  59   9 103  83  99  11  30  26  47  94  57  56  65   2  15  21  37 108
  62 118  80  78  10   8  50   5   4  28 104  14  98 117  42  74  67  54
  48  52 110  82  75  89]

                  Validation set observations

[ 25  68 101  44 105  64 100 119  93  85   6  73  27  97  32  20  71  88
  63   3 114  36  70  76]
```

Iris data에는 150개의 관측치가 있다.

150*0.8*0.8=96개

150*0.8*0.2=24개

150-(96+24)=150*0.2=30 , Test data

KU-BIG

In [2]:

```
iris=load_iris()
train_set_X, test_set_X,train_set_y, test_set_y=train_test_split(iris.data,
                                                                  iris.target,
                                                                 test_size=0.2,
                                                                 random_state=1
```

전체 data를 Training set과 Test set으로 분리

```
model=DecisionTreeClassifier()
param={
    'max_depth':[1,2,3],
    'min_samples_leaf':[2,3]
}
```

Tuning할 hyperparameter

Training set을 training subset과 validation set으로 분리해줄 CV splitter

```
rs=ShuffleSplit(n_splits=1,test_size=0.2)
Dtr=GridSearchCV(cv=rs,estimator=model,param_grid=param,scoring='accuracy')
Dtr.fit(train_set_X,train_set_y)
print('Model performance : ',
      accuracy_score(test_set_y, Dtr.predict(test_set_X)))
```

```
Model performance :  0.8666666666666667
```

KU-BIG

In [17]:

```python
iris=load_iris()

model=DecisionTreeClassifier()
param={
    'max_depth':[1,2,3],
    'min_samples_leaf':[2,3]
}

for i in range(10):
    train_set_X, test_set_X,train_set_y, test_set_y=train_test_split(iris.data
                                                                      iris.target,
                                                                     test_size=0.2)
    rs=ShuffleSplit(n_splits=1,test_size=0.2)
    Dtr=GridSearchCV(cv=rs,estimator=model,param_grid=param,scoring='accuracy'
    Dtr.fit(train_set_X,train_set_y)
    print('Model performance : ',
      accuracy_score(test_set_y, Dtr.predict(test_set_X)),
      'Best parameter : ',
       Dtr.best_params_)
```

```
Model performance :  0.8666666666666667 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 2}
Model performance :  0.9 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9666666666666667 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9666666666666667 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 3}
Model performance :  1.0 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 3}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
```

KU-BIG

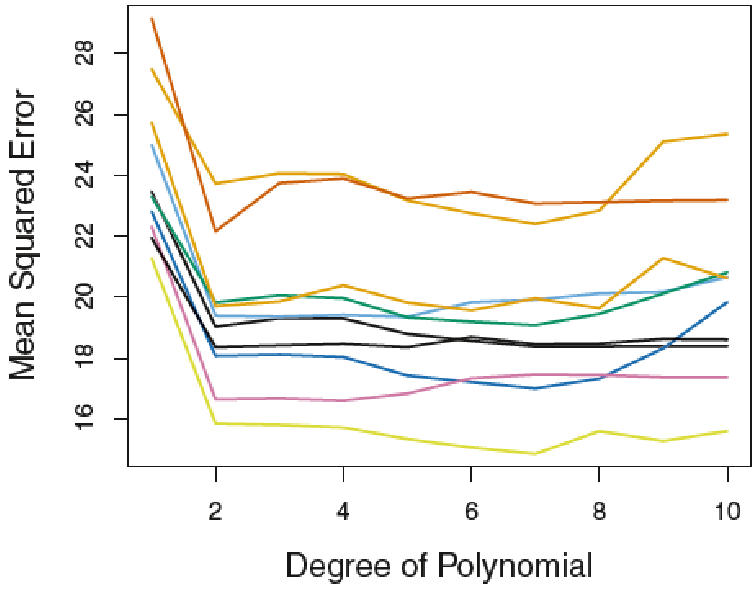
In [17]:

```python
iris=load_iris()

model=DecisionTreeClassifier()
param={
    'max_depth':[1,2,3],
    'min_samples_leaf':[2,3]
}

for i in range(10):
    train_set_X, test_set_X,train_set_y, test_set_y=train_test_split(iris.data
                                                              iris.target,
                                                             test_size=0.2)

    rs=ShuffleSplit(n_splits=1,test_size=0.2)
    Dtr=GridSearchCV(cv=rs,estimator=model,param_grid=param,scoring='accuracy'
    Dtr.fit(train_set_X,train_set_y)
    print('Model performance : ',
      accuracy_score(test_set_y, Dtr.predict(test_set_X)),
      'Best parameter :
       Dtr.best_params_
```

Random한 split이 어떻게 이뤄지느냐에 따라 추정값이 크게 달라짐.

```
Model performance :  0.8666666666666667 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 2}
Model performance :  0.9 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9666666666666667 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9666666666666667 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 3}
Model performance :  1.0 Best parameter :  {'max_depth': 3, 'min_samples_leaf': 3}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
Model performance :  0.9333333333333333 Best parameter :  {'max_depth': 2, 'min_samples_leaf': 2}
```

Hold-out Cross Validation의 단점



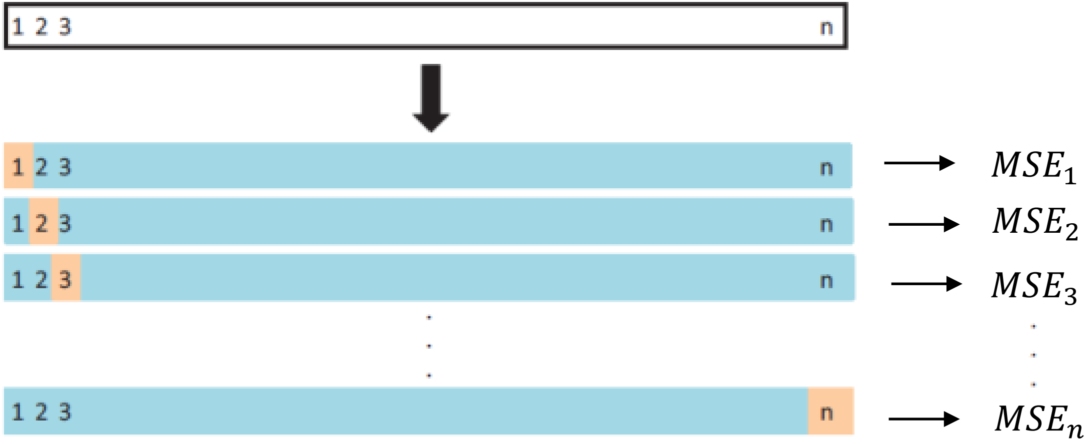
'Auto' dataset으로 Hold-out Cross Validation을 반복적으로 적용해 봄

동일한 데이터로 10번 해본 결과 매번 결과가 다름을 알 수 있음.

Split에 randomness가 있고 random한 split을 한번만 하기 때문

출처 : An introduction to statistical learning with Applications in R

2. LOOCV(leave-one-out-cross validation)



$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MSE}_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

1. 관측치가 n개인 training set을 관측치가 1개인 validation set과 관측치가 (n-1)개인 training subset으로 나눈다.
2. 각각의 관측치들이 한번씩 validation set이 되도록 1번 과정을 n번 반복하며 metric을 구한다.
3. 이 n개 metric의 평균을 기준으로 가장 좋은 성능을 보여주는 hyper parameter를 선택한다.

```
In [1]:
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,LeaveOneOut,GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
In [13]:
loof=loo.split(train_set_X,train_set_y)

for iteration, data in enumerate(loof, start=1):
    print('\n{} {:^61}'.format('Iteration', 'Training subset observations'))
    print('\n{:^9} {}'.format(iteration, data[0]))
    print('\n{:^70}'.format('Validation set observations'))
    print('\n{:^70}'.format(str(data[1])))
```

```
Iteration                    Training subset observations

    1      [  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119]

                      Validation set observations

                                 [0]
```

Iris data에는 150개의 관측치가 있다.

(150*0.8)-1=119개

1개

150-119-1=150*0.2=30 , Test data

```
Iteration                    Training subset observations

  2      [  0   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119]

                     Validation set observations

                              [1]
```

```
Iteration                    Training subset observations

  3      [  0   1   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119]

                     Validation set observations

                              [2]
```

```
Iteration                    Training subset observations

 120     [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118]

                     Validation set observations

                              [119]
```

In [3]:

```
iris=load_iris()
train_set_X, test_set_X,train_set_y, test_set_y=train_test_split(iris.data,
                                                                  iris.target,
                                                                 test_size=0.2,
                                                                 random_state=4
```

전체 data를 Training set과 Test set으로 분리

```
model=DecisionTreeClassifier()
param={
    'max_depth':[1,2,3],
    'min_samples_leaf':[2,3]
}
```

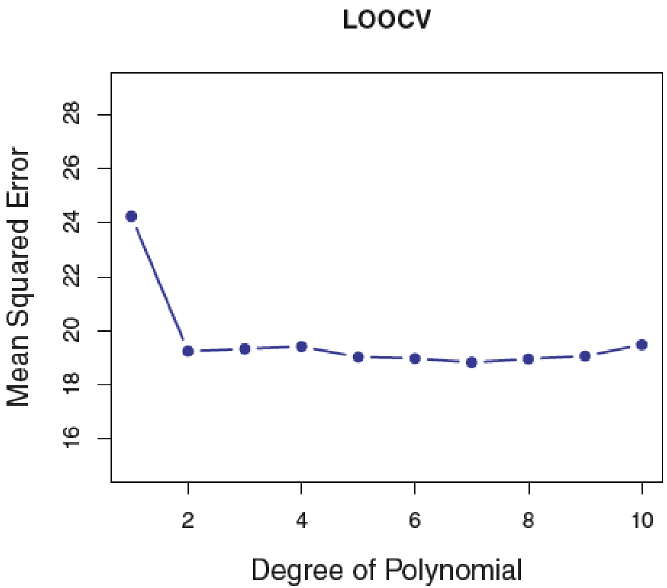
Tuning할 hyperparameter

Training set을 training subset과 validation set으로 분리해줄 CV splitter

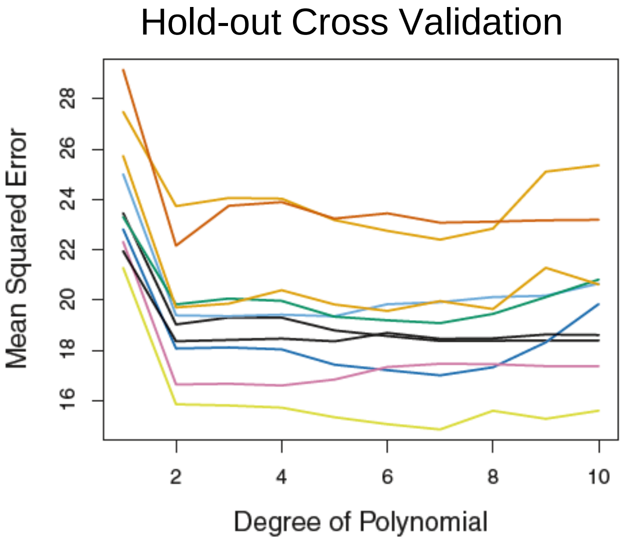
```
loo=LeaveOneOut()
Dtr=GridSearchCV(cv=loo,estimator=model,param_grid=param,scoring='accuracy')
Dtr.fit(train_set_X,train_set_y)
print('Model performance : ',
      accuracy_score(test_set_y, Dtr.predict(test_set_X)))
```

```
Model performance :  0.9666666666666667
```

LOOCV의 장점



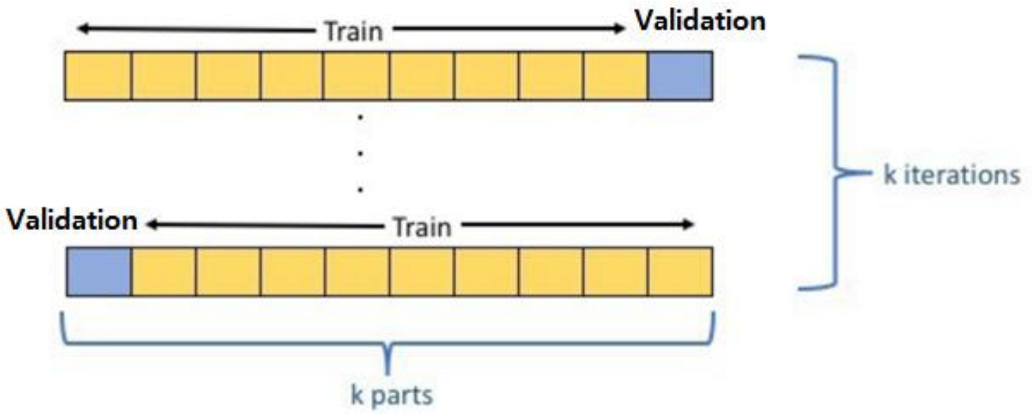
VS



LOOCV는 Training set이 어떻게 Training subset과 validation set으로 나눠지는지에 randomness가 없음.

따라서, Training set이 동일하다면 항상 같은 결과를 반환함.

3. k-fold cross validation



$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i.$$

1. Training set을 동일한 크기의 k개의 그룹으로 나눈다.

2. K개의 그룹 중 (k-1)개의 그룹에 해당하는 data들을 training subset으로 하고 나머지 1개의 그룹에 해당하는 data들을 validation set으로 한다.

3. 모든 그룹이 한번씩 validation set이 되도록 2번 과정을 k번 반복하며 K개의 metric을 구한 후 평균을 내서 가장 좋은 성능을 보여주는 hyperparameter를 선택한다.

In [1]:

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.metrics import accuracy_score
```

In [65]:

```python
kf = KFold(n_splits=5, shuffle=False).split(train_set_X,train_set_y)

for iteration, data in enumerate(kf, start=1):
    print('\n{} {:^61}'.format('Iteration', 'Training subset observations'))
    print('\n{:^9} {}'.format(iteration, data[0]))
    print('\n{:^61}'.format('Validation set observations'))
    print('\n{:^9}'.format(str(data[1])))
    print('\n{:^61}'.format("Test set observations"))
    print('\n',np.arange(iris.data.shape[0]-test_set_X.shape[0],
                         iris.data.shape[0],1))
```

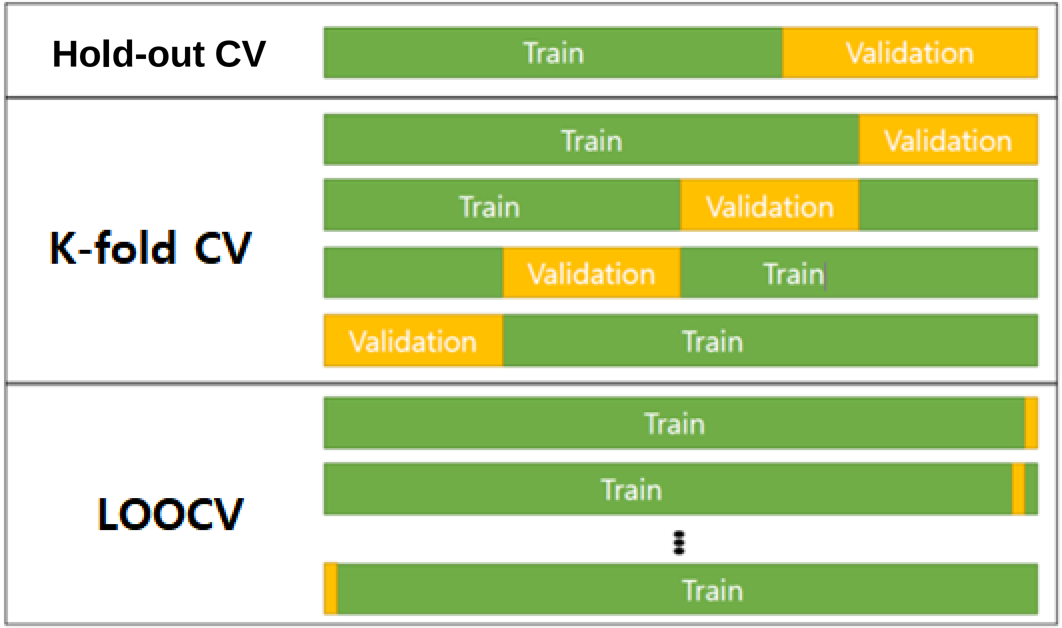

KU-BIG

```
Iteration                    Training subset observations

   1     [ 24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41
  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119]

                   Validation set observations

[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]

                     Test set observations

 [120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
 138 139 140 141 142 143 144 145 146 147 148 149]
```

```
Iteration                    Training subset observations

   2     [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  48  49  50  51  52  53  54  55  56  57  58  59
  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119]

                   Validation set observations

[24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47]

                     Test set observations

 [120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
 138 139 140 141 142 143 144 145 146 147 148 149]
```

```
Iteration                    Training subset observations

   5      [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95]

                   Validation set observations

[ 96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
 114 115 116 117 118 119]

                     Test set observations

 [120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
 138 139 140 141 142 143 144 145 146 147 148 149]
```

Hold-out CV, K-fold CV, LOOCV



Time series data를 위한 Cross Validation

1. Temporal dependence

-시간 순서가 있는 time series data를 random하게 training subset, validation set으로 나눌 수 없다.

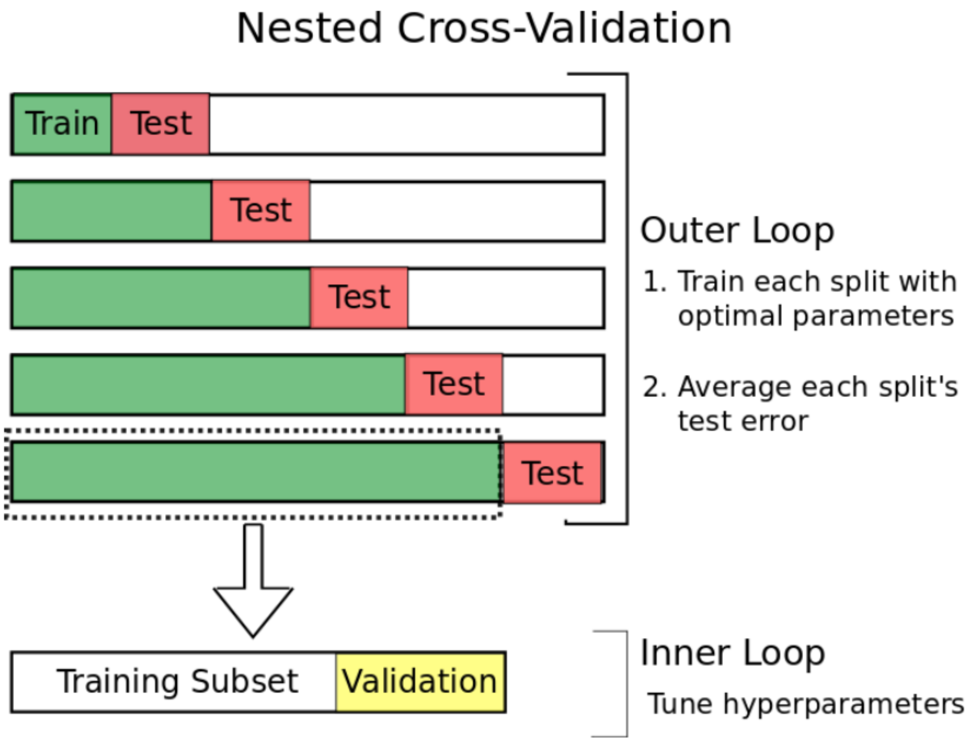
- 어느 시점부터를 test data로 나눌지가 자의적임. Biased estimator가 될 가능성이 높음

2. 과거의 데이터를 이용해 미래를 예측해야 함.

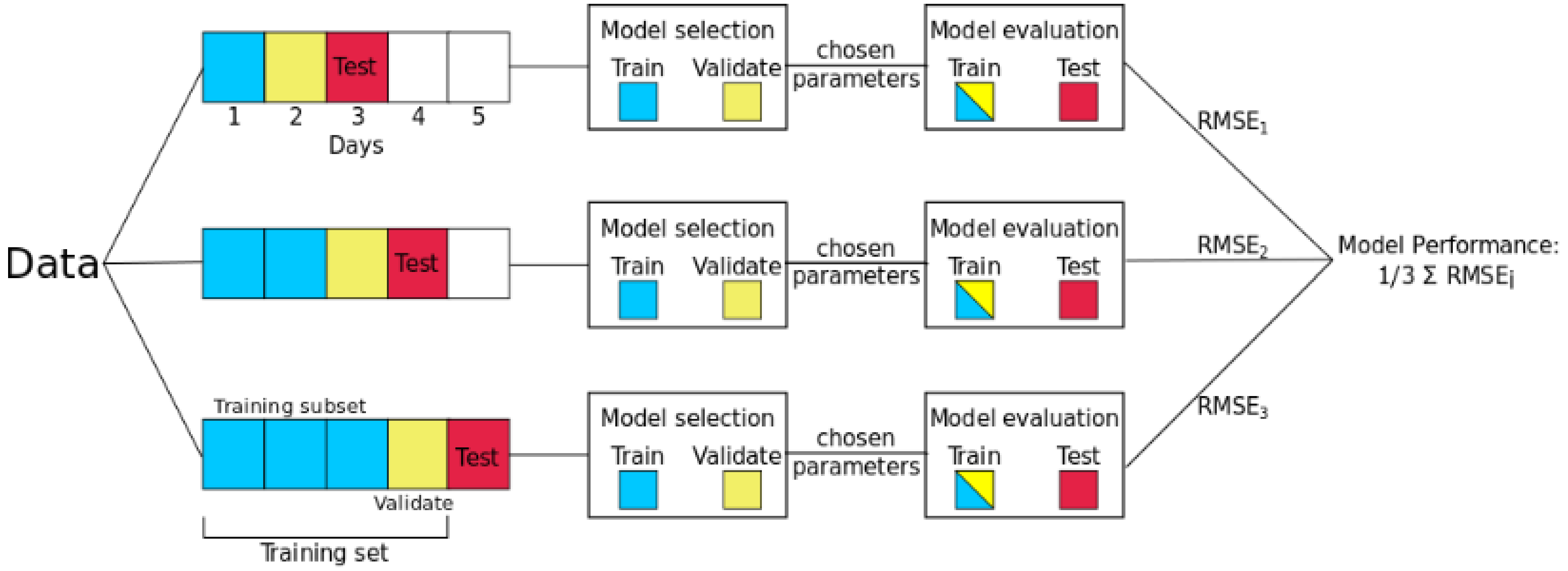
- 미래의 데이터로 과거의 데이터를 예측하는 상황을 원하지 않음.

따라서, k-fold cross validation 사용이 곤란

4. Nested Cross Validation



Day forward chaining



In [1]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import TimeSeriesSplit,GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_absolute_error

df=pd.read_csv("C:/Users/jjw11/Desktop/고려대학교/학회/data/Gemini_ETHUSD_d.csv'
df.head()
```

Out[1]:

	Date	Symbol	Open	High	Low	Close	Volume ETH	Volume USD
0	2019-09-14	ETHUSD	181.55	181.55	181.43	181.43	0.00	0.00
1	2019-09-13	ETHUSD	180.88	181.80	177.70	181.55	4447.19	798113.88
2	2019-09-12	ETHUSD	178.47	182.60	176.45	180.88	2749.15	491636.33
3	2019-09-11	ETHUSD	179.61	182.70	173.70	178.47	12741.56	2270428.10
4	2019-09-10	ETHUSD	181.04	184.35	176.68	179.61	8576.85	1544625.31

In [2]:
```python
df=df.drop(['Symbol'],axis=1)
df=df.sort_values(by='Date').set_index('Date')
df2=df.iloc[:100,:]
df2.head()
```

Out[2]:

	Open	High	Low	Close	Volume ETH	Volume USD
Date						
2016-05-09	12.00	12.00	9.36	9.98	1317.90	12885.06
2016-05-10	9.98	9.98	9.36	9.68	672.06	6578.20
2016-05-11	9.68	10.47	9.68	10.43	3052.51	30978.11
2016-05-12	10.43	12.00	9.92	10.20	2072.56	22183.39
2016-05-13	10.20	11.59	10.20	10.69	1769.71	18923.55

KU-BIG

In [32]:

```python
model=ElasticNet()
param={'alpha':np.arange(0.1,10,0.1)}
tscv=TimeSeriesSplit(n_splits=95)
```

In [33]:

```python
mae=[]

X=df2[['Open','High','Low','Volume ETH','Volume USD','Close']]
X=X.iloc[:-1,]
y=df2['Close'].shift(periods=-1)
y=y.iloc[:-1,]

Els=GridSearchCV(cv=tscv, estimator=model, param_grid=param,
                 scoring='neg_mean_absolute_error')
Els.fit(X,y)


for train_index, Validation_index in tscv.split(df2):
    if Validation_index==df2.shape[0]-2:
        break

    print("TRAIN subset:", train_index, "\nValidation set:",
          Validation_index,"\ntest set:",Validation_index+1)


    X_test=X.iloc[Validation_index+1,:]
    y_test=y.iloc[Validation_index+1]

    mae.append(mean_absolute_error(y_test,Els.predict(X_test)))

print(' Model performance : ',np.mean(mae))
```

```
TRAIN subset: [0 1 2 3 4]
Validation set: [5]
test set: [6]
TRAIN subset: [0 1 2 3 4 5]
Validation set: [6]
test set: [7]
TRAIN subset: [0 1 2 3 4 5 6]
Validation set: [7]
test set: [8]
TRAIN subset: [0 1 2 3 4 5 6 7]
Validation set: [8]
test set: [9]
```

```
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95]
Validation set: [96]
test set: [97]
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96]
Validation set: [97]
test set: [98]
 Model performance :  0.6953183352916578
```

In [32]:

```python
model=ElasticNet()
param={'alpha':np.arange(0.1,10,0.1)}
tscv=TimeSeriesSplit(n_splits=95)
```

In [33]:

```python
mae=[]

X=df2[['Open','High','Low','Volume ETH','Volume USD','Close']]
X=X.iloc[:-1,]
y=df2['Close'].shift(periods=-1)
y=y.iloc[:-1,]

Els=GridSearchCV(cv=tscv, estimator=model, param_grid=param,
                 scoring='neg_mean_absolute_error')
Els.fit(X,y)
```

Inner Loop??

```python
for train_index, Validation_index in tscv.split(df2):
    if Validation_index==df2.shape[0]-2:
        break

    print("TRAIN subset:", train_index, "\nValidation set:",
          Validation_index,"\ntest set:",Validation_index+1)


    X_test=X.iloc[Validation_index+1,:]
    y_test=y.iloc[Validation_index+1]

    mae.append(mean_absolute_error(y_test,Els.predict(X_test)))

print(' Model performance : ',np.mean(mae))
```

```
TRAIN subset: [0 1 2 3 4]
Validation set: [5]
test set: [6]
TRAIN subset: [0 1 2 3 4 5]
Validation set: [6]
test set: [7]
TRAIN subset: [0 1 2 3 4 5 6]
Validation set: [7]
test set: [8]
TRAIN subset: [0 1 2 3 4 5 6 7]
Validation set: [8]
test set: [9]
```

```
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95]
Validation set: [96]
test set: [97]
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96]
Validation set: [97]
test set: [98]
 Model performance :  0.6953183352916578
```

In [50]:
```
model=ElasticNet()
tscv=TimeSeriesSplit(n_splits=95)
```

In [9]:
```
model_performance_mae=[]
for train_index, Validation_index in tscv.split(df2):
    if Validation_index==df2.shape[0]-2:
        break
    mae=[]
    model_set={}
    print("TRAIN subset:", train_index, "\nValidation set:", Validation_index,
    df2_train, df2_validation,df2_test = df2.iloc[train_index,:],df2.iloc[Vali
    X_train=df2_train[['Open','High','Low','Volume ETH','Volume USD','Close']]
    X_train=X_train.iloc[:-1,]
    y_train=df2_train['Close'].shift(periods=-1)
    y_train=y_train.iloc[:-1,]

    X_validation=df2_validation[['Open','High','Low','Volume ETH','Volume USD'
    y_validation=df2.iloc[Validation_index+1,3]
    X_test=df2_test[['Open','High','Low','Volume ETH','Volume USD','Close']]
    y_test=df2.iloc[Validation_index+2,3]


    for i in np.arange(0.1,10,0.1):
        model_set[round((i/0.1)-1)]=ElasticNet(alpha=i)
        mae.append(mean_absolute_error(y_validation,model_set[round((i/0.1)-1)
    idx_least_mae=np.argmin(np.array(mae))
    model_to_fit=model_set[idx_least_mae]

    df2_final_train=df2.iloc[:int(Validation_index+1),:]
    X_final_train=df2_final_train[['Open','High','Low','Volume ETH','Volume US

    X_final_train=X_train.append(X_validation)

    y_final_train=y_train.append(y_validation)
    model_to_fit.fit(X_final_train,y_final_train)
    model_performance_mae.append(mean_absolute_error(y_test,
                                                     model_to_fit.predict(X_te


print(' Model performance : ',np.mean(model_performance_mae))
```

Inner loop,
hyperparameter tuning

```
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95]
Validation set: [96]
test set: [97]
TRAIN subset: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96]
Validation set: [97]
test set: [98]
 Model performance :  1.064419068751438
```

Thank you!

- Q&A Time!