Chapter 7

# Hyper Parameter Tuning

Sejong Oh
Bio Information Technology Lab.

# Contents

- Bias-Variance trade off
- Hyper parameter tuning
- Model comparison
- Feature selection

# 1. Bias-Variance trade off

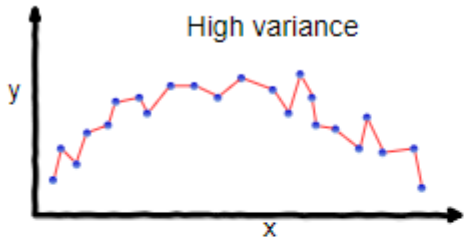
- Classification model error :
  - Noise + Bias (편향) + Variance (분산)
  - Noise - irreducible error

- Bias
  - 데이터 내에 있는 모든 정보를 고려하지 않음으로 인해, 지속적으로 잘못된 것들을 학습하는 경향
  - 예) 코끼리 모양을 학습하는데 다리 모양만 학습
  - Underfitting (과소적합)유발

- Variance
  - 데이터의 너무 세세한 부분까지 학습하여 모델을 만들다보니 새로운 데이터가 추가되면 모델이 쉽게 바뀜 → 모델 변동성이 커짐
  - 예) 옷 맞추기
  - Overfitting (과적합) 유발

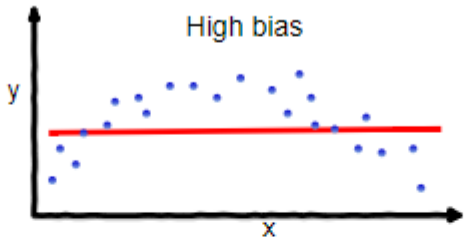
# 1. Bias-Variance trade off

- Bias-Variance trade off
  - Bias 를 줄이려고 하면 Variance 가 증가하고, Variance 를 줄이려고 하면 Bias 가 증가하는 현상
  - 결국은 둘이 적절히 균형을 이루는 지점에서 모델을 선택함



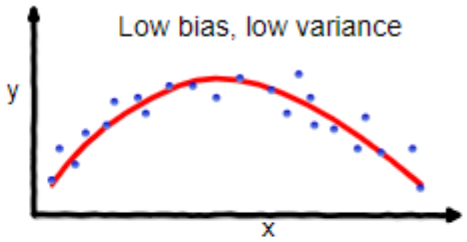

**overfitting**

너무 복잡한 모델




**underfitting**

너무 단순한 모델

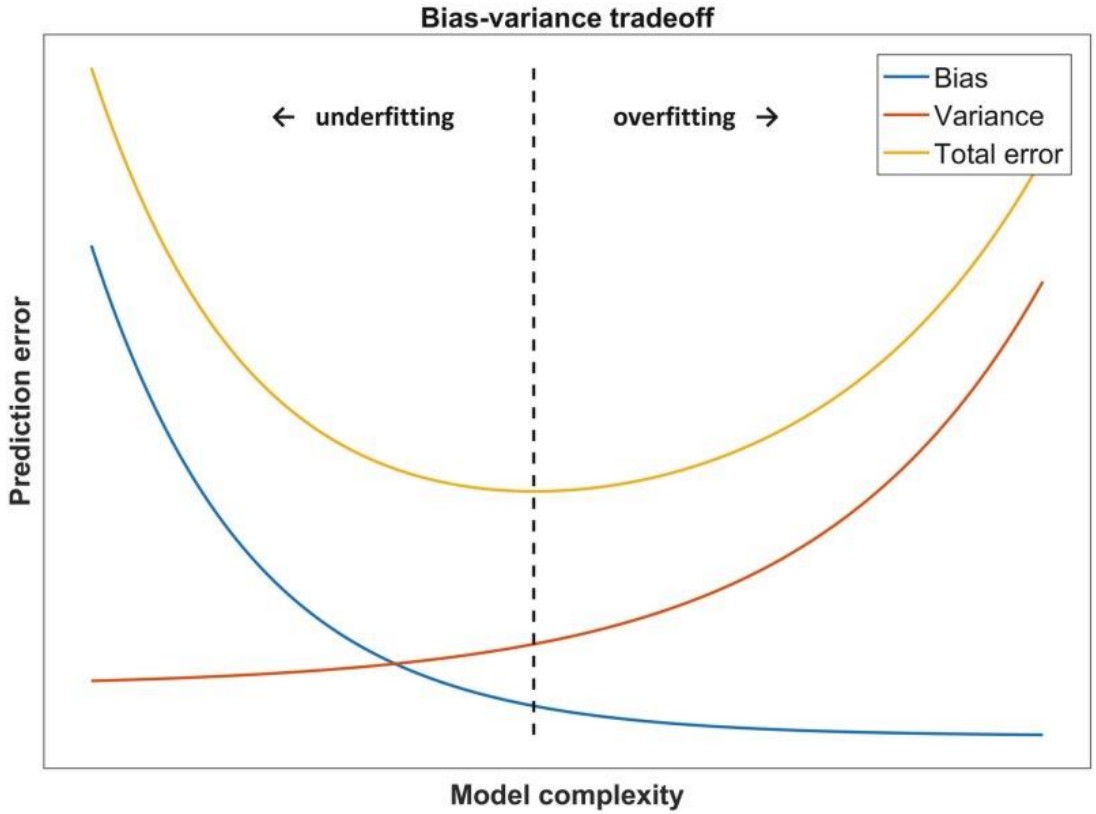



**Good balance**

http://storybydata.com/datacated-challenge/the-bias-and-variance-tradeoff/

# 1. Bias-Variance trade off

- Decision Tree
  - 너무 많은 가지 (복잡한 모델) : variance 증가
  - 너무 적은 가지 (단순한 모델) : bias 증가




https://www.ncbi.nlm.nih.gov/books/NBK543534/figure/ch8.Fig3/

# 1. Bias-Variance trade off

- Note
  - 실제 training 에서는 bias 보다는 variance 가 커지는 경우 (overfitting) 을 더 많이 경험
  - Training accuracy 가 1에 가깝거나 training accuracy 와 test accuracy 의 차이가 크게 벌어지는 경우는 overfitting 을 의심해야 함.
  - 많은 classification algorithm 들이 overfitting을 방지하기 위한 기능을 가지고 있음
    - 예) tree 기반 algorithm : 가지치기 (pruning)
    - 예) SVM : regularization
    - 예) neural network : dropout

# 2. Hyper parameter tuning

- Most of classification algorithms have hyper parameters that influence model performance
- Hyper parameter tuning is a troublesome work and requires long time.

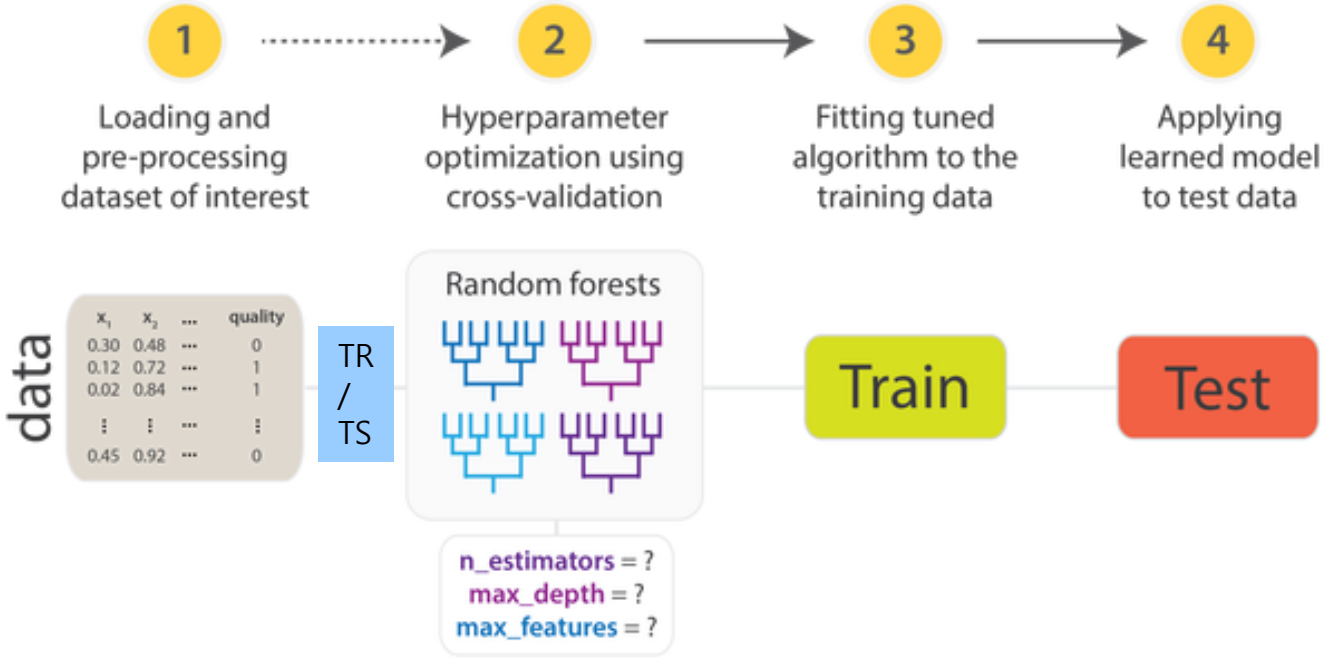
- example

| Parameter | Test value |
|---|---|
| P1 | 0.1, 0.3, 0.5 |
| P2 | 10, 15, 20, 15, 30 |
| P3 | 1,3,5,7 |

  - Number of combination : 3 x 5 x 4 = 60 cases ( 60 models are tested )
  - Models should be compared by k-fold cross validation

# 2. Hyper parameter tuning

- Model building process




https://cambridgecoding.wordpress.com/2016/04/03/scanning-hyperspace-how-to-tune-machine-learning-models/

# 2. Hyper parameter tuning

- Hyper parameter tuning with scikit-learn
- https://scikit-learn.org/stable/modules/grid_search.html#tips-for-parameter-search

**3.2. Tuning the hyper-parameters of an estimator**

3.2.1. Exhaustive Grid Search

3.2.2. Randomized Parameter Optimization

3.2.3. Tips for parameter search

3.2.4. Alternatives to brute force parameter search

# 2. Hyper parameter tuning

- Dataset : PimaIndiansDiabetes

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | pregnant | glucose | pressure | triceps | insulin | mass | pedigree | age | diabetes |
| 2 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | pos |
| 3 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | neg |
| 4 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | pos |
| 5 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | neg |
| 6 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | pos |
| 7 | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | neg |
| 8 | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | pos |
| 9 | 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | neg |
| 10 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | pos |
| 11 | 8 | 125 | 96 | 0 | 0 | 0 | 0.232 | 54 | pos |

- pregnant Number of times pregnant
- glucose Plasma glucose concentration (glucose tolerance test)
- pressure Diastolic blood pressure (mm Hg)
- triceps Triceps skin fold thickness (mm)
- insulin 2-Hour serum insulin (mu U/ml)
- mass Body mass index (weight in kg/(height in m)₩^2)
- pedigree Diabetes pedigree function
- age Age (years)
- diabetes Class variable (test for diabetes)

# 2. Hyper parameter tuning

- (1) Greed search cross validation
  - The grid search provided by GridSearchCV exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter

  - param_grid Example for svm

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
```

# 2. Hyper parameter tuning

07.RF_tuning_grid.py

```
# Random Forest tuning Example
# using: GridSearchCV

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import pprint

pp = pprint.PrettyPrinter(width=80, indent=4)

# prepare the credit dataset
df = pd.read_csv('D:/data/PimaIndiansDiabetes.csv.csv')
print(df.head())
print(df.columns)   # column names
```

# 2. Hyper parameter tuning

```
In [28]: print(df.head())
   pregnant  glucose  pressure  triceps  insulin  mass  pedigree  age diabetes
0         6      148        72       35        0  33.6     0.627   50      pos
1         1       85        66       29        0  26.6     0.351   31      neg
2         8      183        64        0        0  23.3     0.672   32      pos
3         1       89        66       23       94  28.1     0.167   21      neg
4         0      137        40       35      168  43.1     2.288   33      pos

In [29]: print(df.columns)   # column names
Index(['pregnant', 'glucose', 'pressure', 'triceps', 'insulin', 'mass',
       'pedigree', 'age', 'diabetes'],
      dtype='object')
```

# 2. Hyper parameter tuning

```
df_X = df.loc[:, df.columns != 'diabetes']
df_y = df['diabetes']

# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
    train_test_split(df_X, df_y, test_size=0.3,\
                     random_state=1234)

# base model
base_model = RandomForestClassifier(random_state=1234)
base_model.fit(train_X, train_y)
base_accuracy = base_model.score(test_X, test_y)
```

```
In [31]: base_accuracy
Out[31]: 0.7532467532467533
```

# 2. Hyper parameter tuning

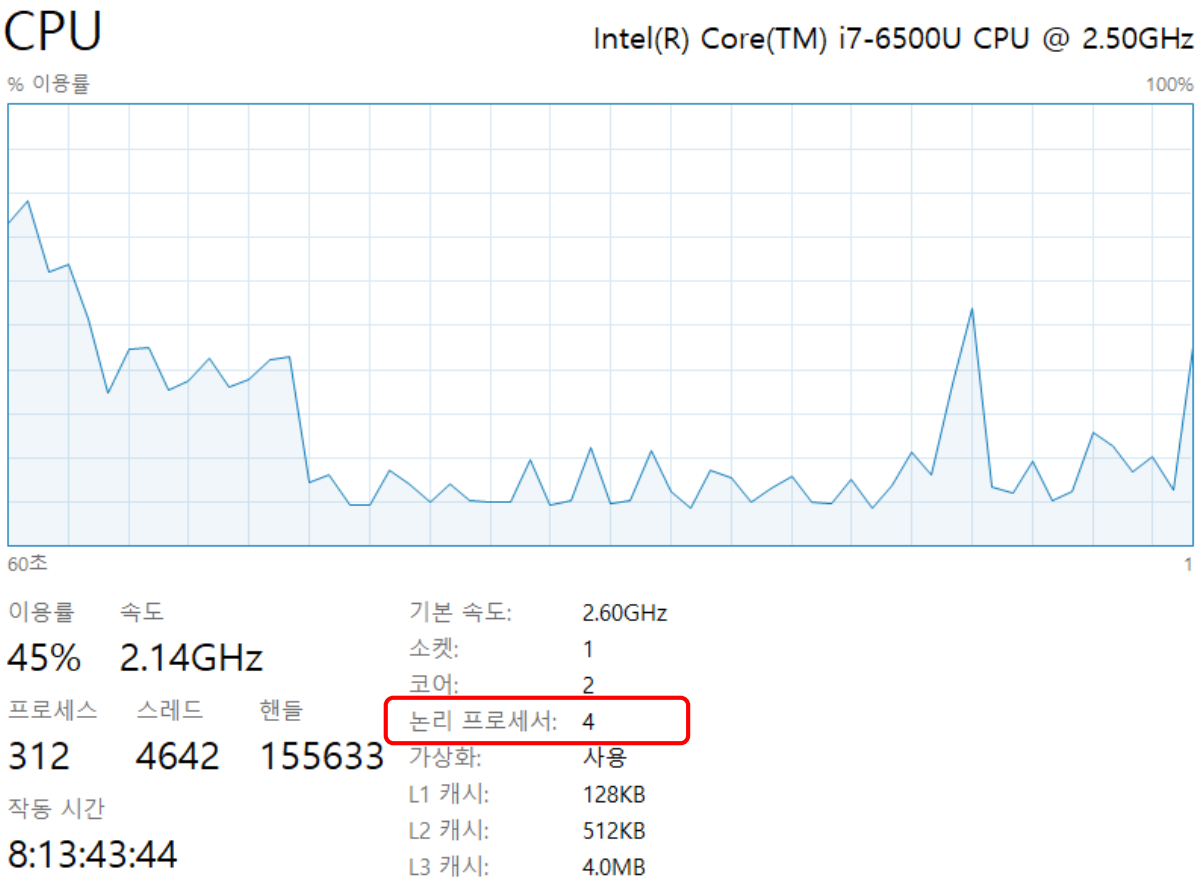
```
## GridSearchCV ##############################################

# hyper parameter tuning
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3, 'auto', 'sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
```

# 2. Hyper parameter tuning

```
# Create a based model
rf = RandomForestClassifier(random_state=1234)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2)
```

| | |
|---|---|
| estimator | Classification algorithm |
| param_grid | Param grid |
| cv | 모델 평가시 cross validation 수 |
| n_jobs | 작업에 사용할 processor수. (-1 은 모든 processor 사용) |
| verbose | Tuning 과정에서 발생하는 메시지 표시 정도 (숫자 클수록 상세정보 표시) |

# 2. Hyper parameter tuning



# 2. Hyper parameter tuning

```
# Create a based model
rf = RandomForestClassifier(random_state=1234)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2)
```

# 2. Hyper parameter tuning

```
# Fit the grid search to the data
grid_search.fit(train_X, train_y)

# best parameters
pp.pprint(grid_search.best_params_)
```

```
In [52]: grid_search.fit(train_X, train_y)
Fitting 5 folds for each of 576 candidates, totalling 2880 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  6.3min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed: 13.8min
[Parallel(n_jobs=-1)]: Done 1977 tasks      | elapsed: 18.5min
[Parallel(n_jobs=-1)]: Done 2584 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done 2880 out of 2880 | elapsed: 26.7min finished
```

```
In [53]: pp.pprint(grid_search.best_params_)
{   'bootstrap': True,
    'max_depth': 80,
    'max_features': 2,
    'min_samples_leaf': 3,
    'min_samples_split': 10,
    'n_estimators': 100}
```

# 2. Hyper parameter tuning

```
# best model
best_model = grid_search.best_estimator_
best_accuracy = best_model.score(test_X, test_y)

print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( \
       base_accuracy, best_accuracy))
print('Improvement of {:0.2f}%.'.format( \
       100 * (best_accuracy - base_accuracy) / base_accuracy))
```

```
In [38]: print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( base_accuracy,
best_accuracy))
base acc: 0.75. best acc : 0.74

In [39]: print('Improvement of {:0.2f}%.'.format( 100 * (best_accuracy - base_accuracy) /
base_accuracy))
Improvement of -2.30%.
```

# 2. Hyper parameter tuning

- (2) Random search cross validation
  - randomized search over parameters, where each setting is sampled from a distribution over possible parameter values.
  - two main benefits
    - A budget can be chosen independent of the number of parameters and possible values.
    - Adding parameters that do not influence the performance does not decrease efficiency.
  - Function : RandomizedSearchCV

# 2. Hyper parameter tuning

07.RF_tuning_random.py

```
# Random Forest tuning Example
# using: RandomizedSearchCV

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import pprint

(생략. GreedSearch 와 동일)
```

# 2. Hyper parameter tuning

```
## RandomizedSearchCV ##############################################

# define range of parameter values
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
                                            num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

# 2. Hyper parameter tuning

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pp.pprint(random_grid)
```

```
In [25]: pp.pprint(random_grid)
{   'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

# 2. Hyper parameter tuning

```
# Use the random grid to search for best hyperparameters

rf = RandomForestClassifier(random_state=1234)
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = random_grid,
                               n_iter = 100, cv = 5, verbose=2,
                               random_state=42, n_jobs = -1)
```

| | |
|---|---|
| estimator | Classification algorithm |
| param_distributions | Param grid |
| n_iter | Param combination 에서 선택할 조합의 갯수 |
| cv | 모델 평가시 cross validation 수 |
| verbose | Tuning 과정에서 발생하는 메시지 표시 정도 (숫자 클수록 상세정보 표시) |
| random_state | Random seed |
| n_jobs | 작업에 사용할 processor수. (-1 은 모든 processor 사용) |

# 2. Hyper parameter tuning

```
# Fit the random search model
rf_random.fit(train_X, train_y)

# best parameters
pp.pprint(rf_random.best_params_)
```

```
In [19]: rf_random.fit(train_X, train_y)
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   51.9s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  9.7min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 14.3min finished
```

```
In [42]: pp.pprint(rf_random.best_params_)
{   'bootstrap': True,
    'max_depth': None,
    'max_features': 'auto',
    'min_samples_leaf': 2,
    'min_samples_split': 5,
    'n_estimators': 800}
```

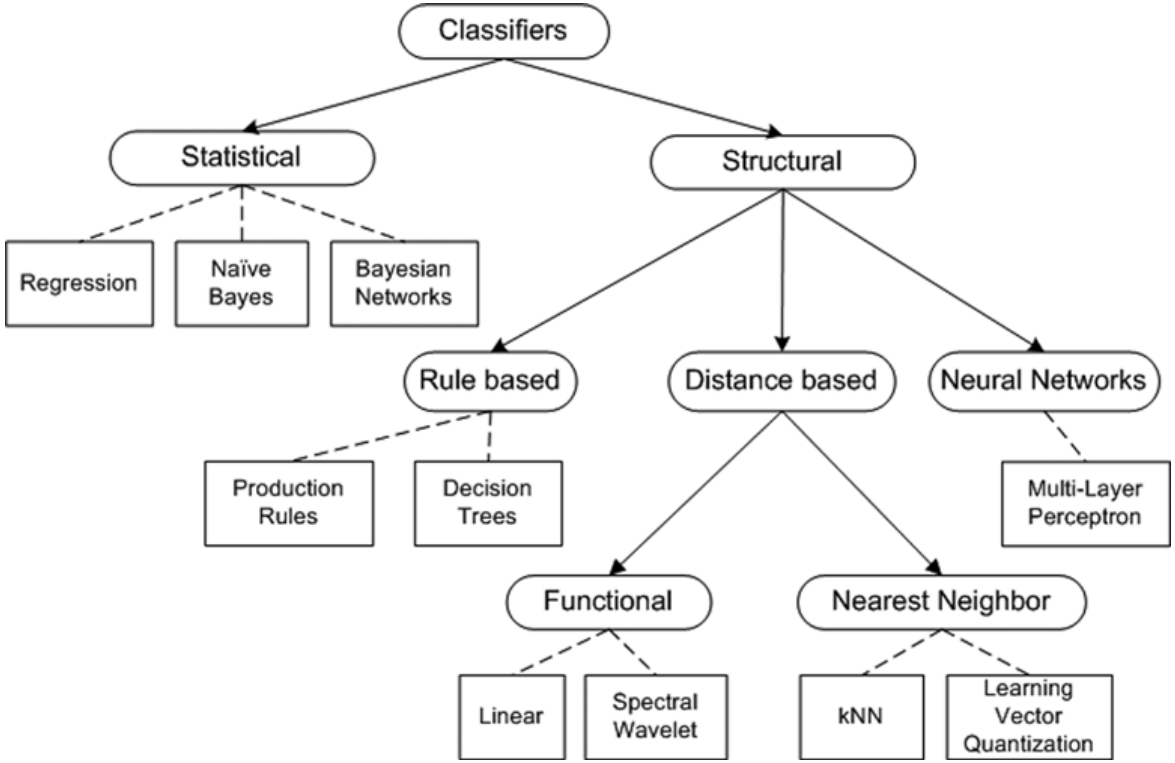
# 2. Hyper parameter tuning

```
# best model
best_random_model = rf_random.best_estimator_
best_random_accuracy = best_random_model.score(test_X, test_y)

print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( \
       base_accuracy, best_random_accuracy))
print('Improvement of {:0.2f}%.'.format( 100 *  \
      (best_random_accuracy - base_accuracy) / base_accuracy))
```

```
In [26]: print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( base_accuracy,
best_random_accuracy))
base acc: 0.75. best acc : 0.76

In [27]: print('Improvement of {:0.2f}%.'.format( 100 * (best_random_accuracy - base_accuracy)
/ base_accuracy))
Improvement of 1.15%.
```

# 3. Model Comparison

- There is no “super classification classifier” for every dataset.
- We need to test various classifiers (predictors,, models) as much as possible
- Scikit-learn supports easy to model comparison




https://mariuszprzydatek.com/2014/05/26/machine-learning/

# 3. Model Comparison

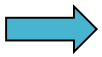
07.model_comparison.py

```
# Model comparison Example

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder
```

# 3. Model Comparison

```
# prepare the credit dataset
df = pd.read_csv('D:/data/PimaIndiansDiabetes.csv.csv')
print(df.head())
print(df.columns)   # column names

df_X = df.loc[:, df.columns != 'diabetes']
df_y = df['diabetes']

# change string label to integer for Logistic regression
encoder = LabelEncoder()
encoder.fit(df_y)
df_y = encoder.transform(df_y)
```

```
In [93]: df_y
Out[93]:
0      pos
1      neg
2      pos
3      neg
4      pos
      ...
763    neg
764    neg
765    neg
766    pos
767    neg
```



```
In [95]: df_y
Out[95]:
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1
```

# 3. Model Comparison

```
# prepare configuration for cross validation test harness
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression(max_iter=500)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC()))
```

```
In [103]: models
Out[103]:
[('LR', LogisticRegression(max_iter=500)),
 ('KNN', KNeighborsClassifier()),
 ('DT', DecisionTreeClassifier()),
 ('RF', RandomForestClassifier()),
 ('SVM', SVC())]
```

# 3. Model Comparison

```
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed,
                shuffle=True)
        cv_results = model_selection.cross_val_score(model,
                        df_X, df_y, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(),
                cv_results.std())
        print(msg)
```

```
LR: 0.772163 (0.049684)
KNN: 0.710988 (0.050792)
DT: 0.688927 (0.043638)
RF: 0.760390 (0.050851)
SVM: 0.760458 (0.034712)
```

# 3. Model Comparison

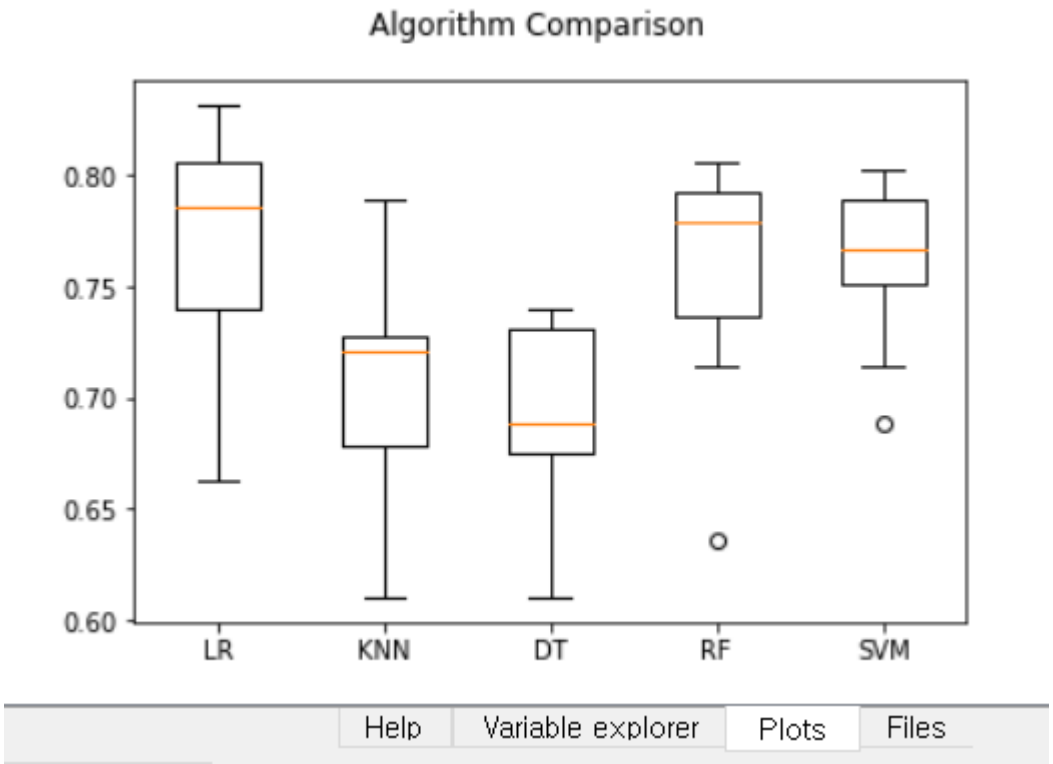
```
print(results)
# average accuracy of classifiers
for i in range(0,len(results)):
    print(names[i] + "\t" + str(round(np.mean(results[i]),4)))
```

```
In [105]: print(results)
[array([0.83116883, 0.74025974, 0.74025974, 0.80519481, 0.79220779,
       0.77922078, 0.66233766, 0.80519481, 0.82894737, 0.73684211]),
array([0.72727273, 0.71428571, 0.61038961, 0.72727273, 0.7012987 ,
       0.72727273, 0.66233766, 0.77922078, 0.78947368, 0.67105263]),
array([0.7012987 , 0.67532468, 0.62337662, 0.67532468, 0.71428571,
       0.67532468, 0.61038961, 0.74025974, 0.73684211, 0.73684211]),
array([0.79220779, 0.77922078, 0.71428571, 0.77922078, 0.80519481,
       0.79220779, 0.63636364, 0.80519481, 0.77631579, 0.72368421]),
array([0.79220779, 0.75324675, 0.71428571, 0.79220779, 0.77922078,
       0.77922078, 0.68831169, 0.75324675, 0.80263158, 0.75      ])]
```

```
In [106]: for i in range(0,len(results)):
     ...:     print(names[i] + "\t" + str(round(np.mean(results[i]),4)))
LR  0.7722
KNN 0.711
DT  0.6889
RF  0.7604
SVM 0.7605
```

# 3. Model Comparison

```
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
LR  0.7722
KNN 0.711
DT  0.6889
RF  0.7604
SVM 0.7605
```

# 4. Feature selection

- Feature = variable = column in datasets
- More information leads better classification performance ?

당뇨병진단

| gender | age | height | weight | f_color | Label |
|---|---|---|---|---|---|
| | | | | | Pos |
| | | | | | Neg |
| | | | | | |
| | | | | | |

- 1000 features ?
  - Requires selection of good features

# 4. Feature selection

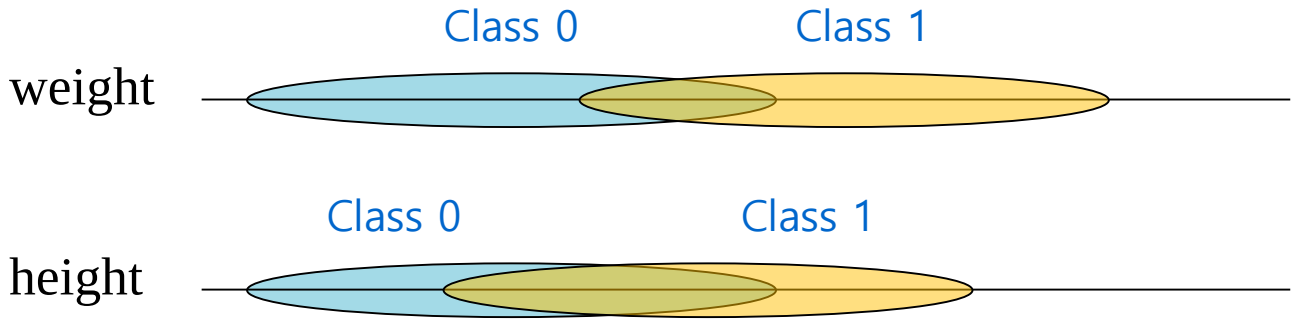
- Feature selection
  - the process where you automatically or manually select
  - those features which contribute most
  - to your prediction variable or output in which you are interested in.

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |

  - Evaluate features and choose good feature subset

# 4. Feature selection

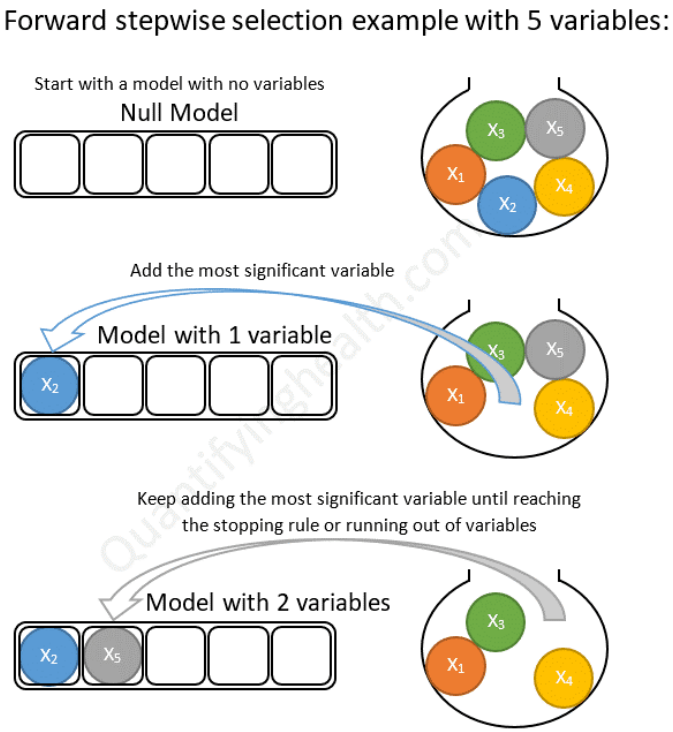
- Which is a good feature ?
  - Good features has clear class boundary

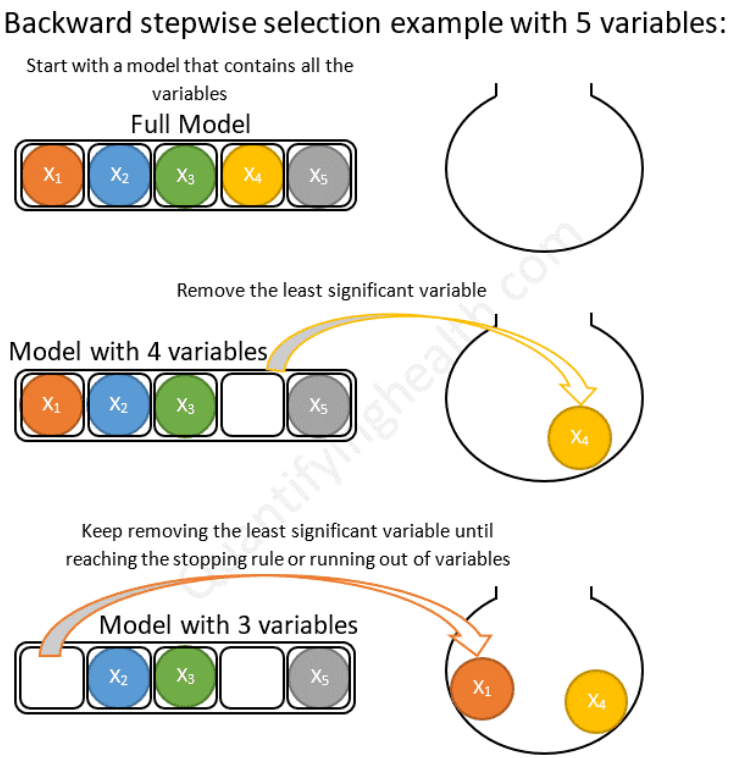



- Filter method
  - Evaluate each feature and select best *n* features
  - Easy and fast
  - Problem : does not consider feature interaction
    - Best{ 1,2,5 } can be better than Best{1,2,3}

# 4. Feature selection

- Feature subset selection
  - Forward selection
  - Backward elimination




https://quantifyinghealth.com/stepwise-selection/

# 4. Feature selection

- Scikit-learn
  - The classes in the sklearn.feature_selection module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.
  - https://scikit-learn.org/stable/modules/feature_selection.html

**1.13. Feature selection**

1.13.1. Removing features with low variance

1.13.2. Univariate feature selection

1.13.3. Recursive feature elimination

1.13.4. Feature selection using SelectFromModel

1.13.5. Feature selection as part of a pipeline

# 4. Feature selection

07.feature_selection.py

```
# Feature selection Example

import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# prepare the dataset
df = pd.read_csv('D:/data/PimaIndiansDiabetes.csv.csv')
print(df.head())
print(df.columns)   # column names

df_X = df.loc[:, df.columns != 'diabetes']
df_y = df['diabetes']

# whole features
model = LogisticRegression(solver='lbfgs', max_iter=500)
scores = cross_val_score(model, df_X, df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

```
In [314]: print("Acc: "+str(scores.mean()))
Acc: 0.7721925133689839
```

# 4. Feature selection

```
#####################################################################
# feature selection by filter method
#####################################################################
# feature evaluation method : chi-square
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test = SelectKBest(score_func=chi2, k=df_X.shape[1])
fit = test.fit(df_X, df_y)
# summarize evaluation scores
print(np.round(fit.scores_, 3))

f_order = np.argsort(-fit.scores_)  # sort index by decreasing order
sorted_columns = df.columns[f_order]
```

```
In [281]: print(np.round(fit.scores_, 3))
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
```

```
In [289]: print(sorted_columns.tolist())
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure', 'pedigree']
```

# 4. Feature selection

```
# test classification accuracy by selected features (RF)
model = LogisticRegression(solver='lbfgs', max_iter=500)
for i in range(1, df_X.shape[1]+1):
    fs = sorted_columns[0:i]
    df_X_selected = df_X[fs]
    scores = cross_val_score(model, df_X_selected, df_y, cv=5)
    print(fs.tolist())
    print(np.round(scores.mean(), 4))
```

```
['insulin']
0.6563
['insulin', 'glucose']
0.7475
['insulin', 'glucose', 'age']
0.7371
['insulin', 'glucose', 'age', 'mass']
0.7683
['insulin', 'glucose', 'age', 'mass', 'pregnant']
0.7709
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps']
0.7696
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure']
0.7748
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure', 'pedigree']
0.7722
```

# 4. Feature selection

```
##################################################################
# Backward elimination (Recursive Feature Elimination)
##################################################################
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=4)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()   # selected features
print("Selected Features: %s" % fs)

scores = cross_val_score(model, df_X[fs], df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

```
In [299]: print("Num Features: %d" % fit.n_features_)
Num Features: 4
```

```
In [301]: print("Selected Features: %s" % fs)
Selected Features: ['pregnant', 'glucose', 'mass', 'pedigree']

In [304]: print("Acc: "+str(scores.mean()))
Acc: 0.7695526695526695
```

# 4. Feature selection

- RFE

**Parameters:**

**estimator : *object***

A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

**n_features_to_select : *int or None (default=None)***

The number of features to select. If `None`, half of the features are selected.

**step : *int or float, optional (default=1)***

If greater than or equal to 1, then `step` corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then `step` corresponds to the percentage (rounded down) of features to remove at each iteration.

**verbose : *int, (default=0)***

Controls verbosity of output.

# 4. Feature selection

```
#####################################################################
# Forward selection
#####################################################################
# please install 'mlxtend' moudle

from mlxtend.feature_selection import SequentialFeatureSelector as SFS

model = LogisticRegression(solver='lbfgs', max_iter=500)
sfs1 = SFS(model,
           k_features=5,          # # of features
           verbose=2,
           scoring='accuracy',
           cv=5)

sfs1 = sfs1.fit(df_X, df_y, custom_feature_names=df_X.columns)
sfs1.subsets_              # selection process
sfs1.k_feature_idx_        # selected feature index
sfs1.k_feature_names_      # selected feature name
```

# 4. Feature selection

```
In [352]: sfs1.subsets_                # selection process
Out[352]:
{1: {'feature_idx': (1,),
  'cv_scores': array([0.708, 0.708, 0.766, 0.771, 0.784]),
  'avg_score': 0.7474747474747474,
  'feature_names': ('glucose',)},
 2: {'feature_idx': (1, 5),
  'cv_scores': array([0.773, 0.734, 0.766, 0.784, 0.739]),
  'avg_score': 0.7591206179441474,
  'feature_names': ('glucose', 'mass')},
 3: {'feature_idx': (1, 5, 7),
  'cv_scores': array([0.773, 0.734, 0.74 , 0.804, 0.791]),
  'avg_score': 0.7683048977166624,
  'feature_names': ('glucose', 'mass', 'age')},
 4: {'feature_idx': (1, 4, 5, 7),
  'cv_scores': array([0.766, 0.734, 0.753, 0.804, 0.784]),
  'avg_score': 0.7682964094728801,
  'feature_names': ('glucose', 'insulin', 'mass', 'age')},
 5: {'feature_idx': (0, 1, 4, 5, 7),
  'cv_scores': array([0.753, 0.74 , 0.786, 0.791, 0.784]),
  'avg_score': 0.7708768355827178,
  'feature_names': ('pregnant', 'glucose', 'insulin', 'mass', 'age')}}

In [353]: sfs1.k_feature_idx_         # selected feature index
Out[353]: (0, 1, 4, 5, 7)

In [354]: sfs1.k_feature_names_       # selected feature name
Out[354]: ('pregnant', 'glucose', 'insulin', 'mass', 'age')
```

# 4. Feature selection

- SequentialFeatureSelector (SFS)
  - http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.feature_selection/#sequentialfeatureselector

**Parameters**

- `estimator` : scikit-learn classifier or regressor
- `k_features` : int or tuple or str (default: 1)
- `forward` : bool (default: True)

  Forward selection if True, backward selection otherwise
- `floating` : bool (default: False)

  Adds a conditional exclusion/inclusion if True.
- `verbose` : int (default: 0), level of verbosity to use in logging.

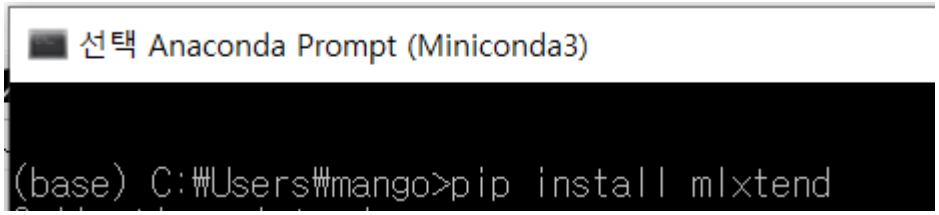
  If 0, no output, if 1 number of features in current set, if 2 detailed logging i ncluding timestamp and cv scores at step.
- `scoring` : str, callable, or None (default: None)

# 4. Feature selection

```
scores = cross_val_score(model, df_X[list(sfs1.k_feature_names_)], df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

```
In [356]: print("Acc: "+str(scores.mean()))
Acc: 0.7708768355827178
```

- Note. mlxtend 설치



# 4. Feature selection

- Summary

| | # of feature | accuracy |
|---|---|---|
| Whole features | 8 | 0.772 |
| Filter method | 7 | 0.775 |
| Forward selection | 5 | 0.771 |
| Backward elimination | 4 | 0.770 |

- Scikit-learn RFE use 'importance of each feature'
  - Applied models should have coef_ attribute or feature_importances_
  - KNN cannot use RFE

# Conclusion

- Practical model development process