딥러닝/클라우드

Chapter 6

Cross validation

Sejong Oh

Bio Information Technology Lab.

Contents

- Bias-Variance trade off
- Cross validation
- Hyper parameter tuning
- Model comparison
- Performance metric

1. Bias-Variance trade off

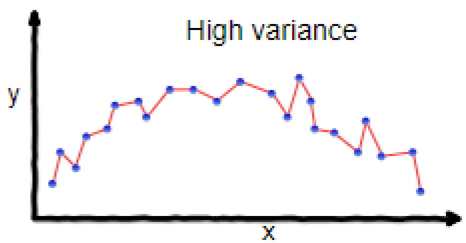
- Classification model error :
  - Noise + Bias (편향) + Variance (분산)
  - Noise : irreducible error

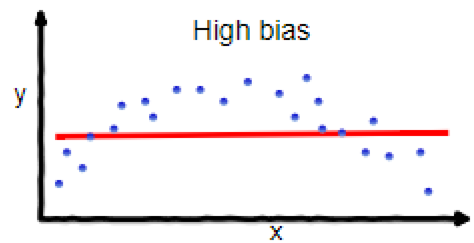
- Bias
  - 데이터 내에 있는 모든 정보를 고려하지 않음으로 인해, 지속적으로 잘못된 것들을 학습하는 경향
  - 예) 코끼리 모양을 학습하는데 다리 부분만 학습
  - Underfitting (과소적합)유발

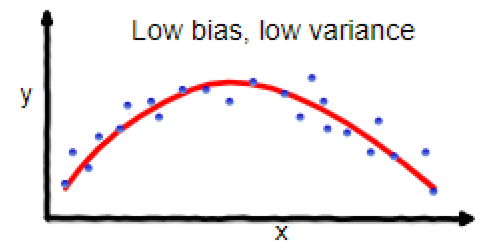
- Variance
  - 데이터의 너무 세세한 부분까지 학습하여 모델을 만들다보니 새로운 데이터가 추가되면 모델이 쉽게 바뀜 → 모델 변동성이 커짐
  - 예) 옷 맞추기
  - Overfitting (과적합) 유발

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

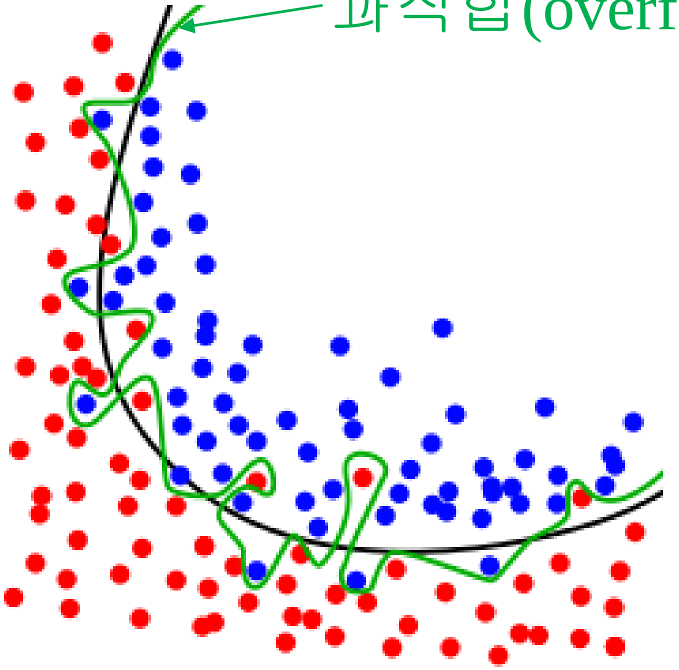
underfitting

너무 단순한 모델

Good balance

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

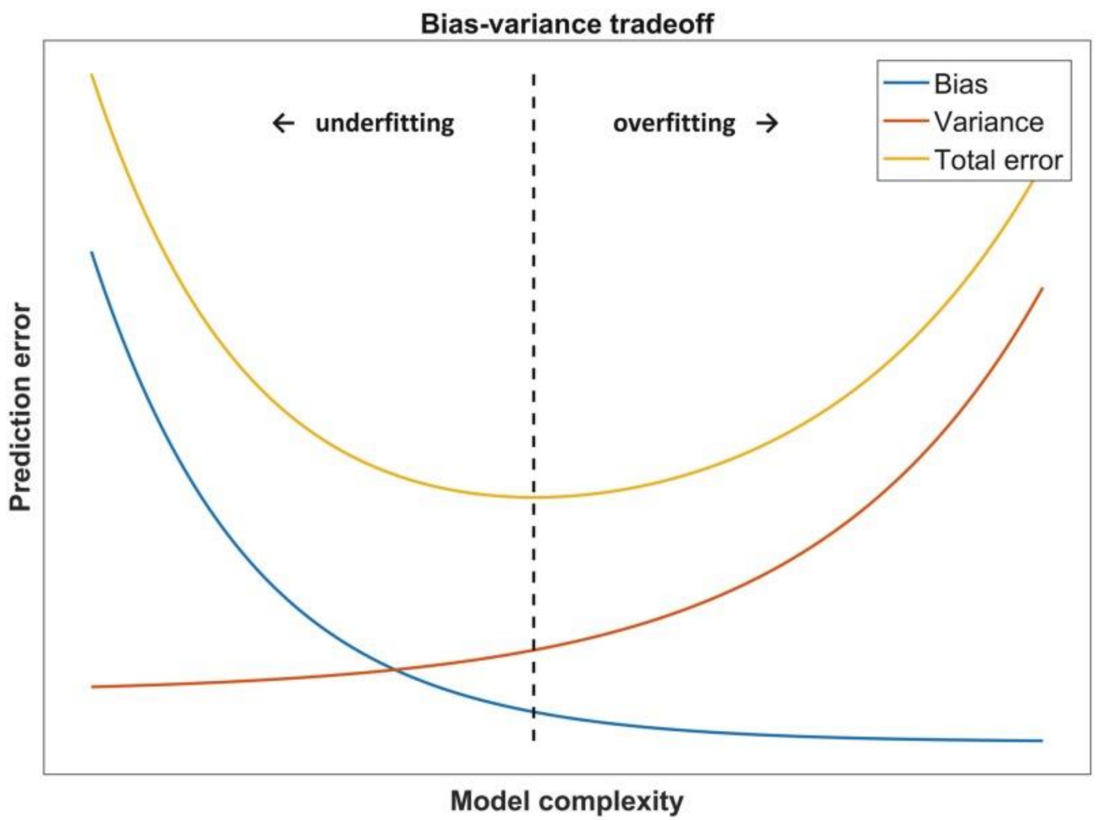
1. Bias-Variance trade off



https://ko.wikipedia.org/wiki/%EA%B3%BC%EC%A0%81%ED%95%A9

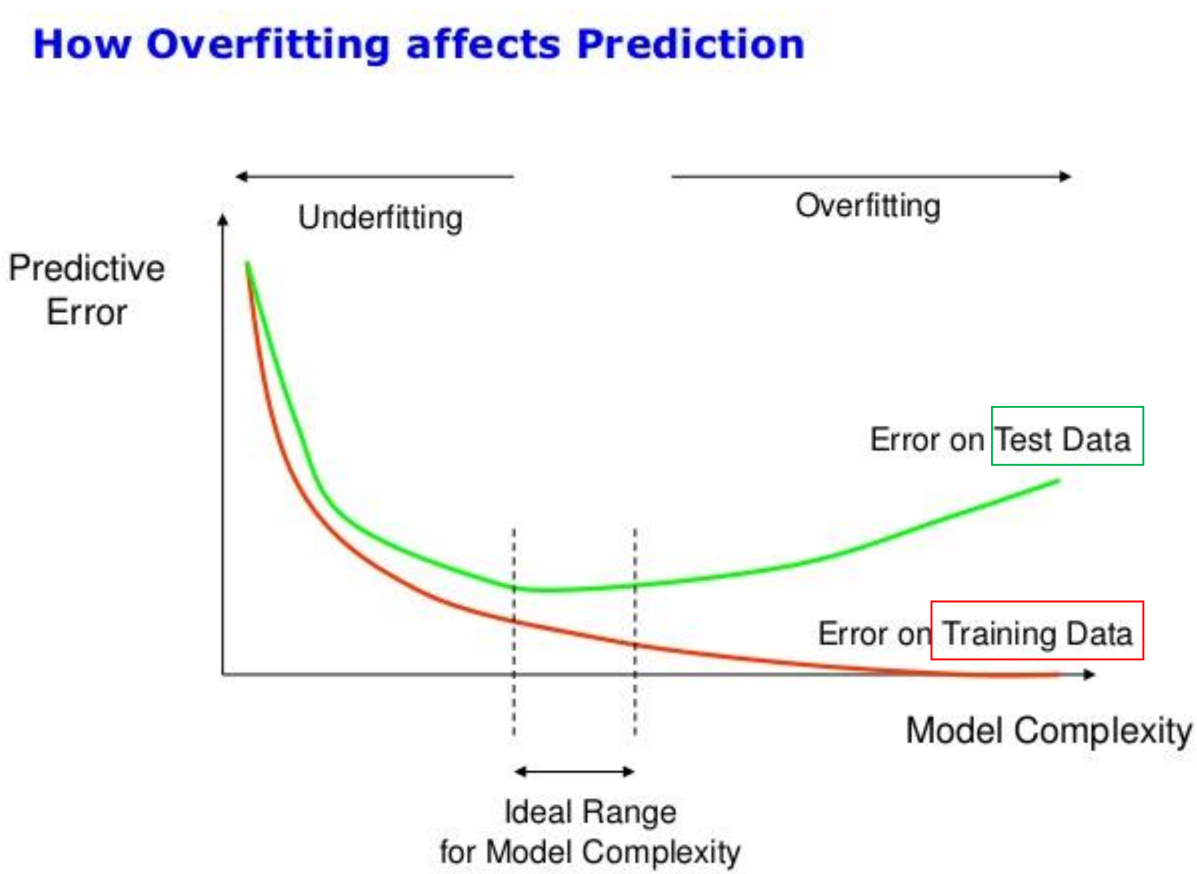
1. Bias-Variance trade off

- 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



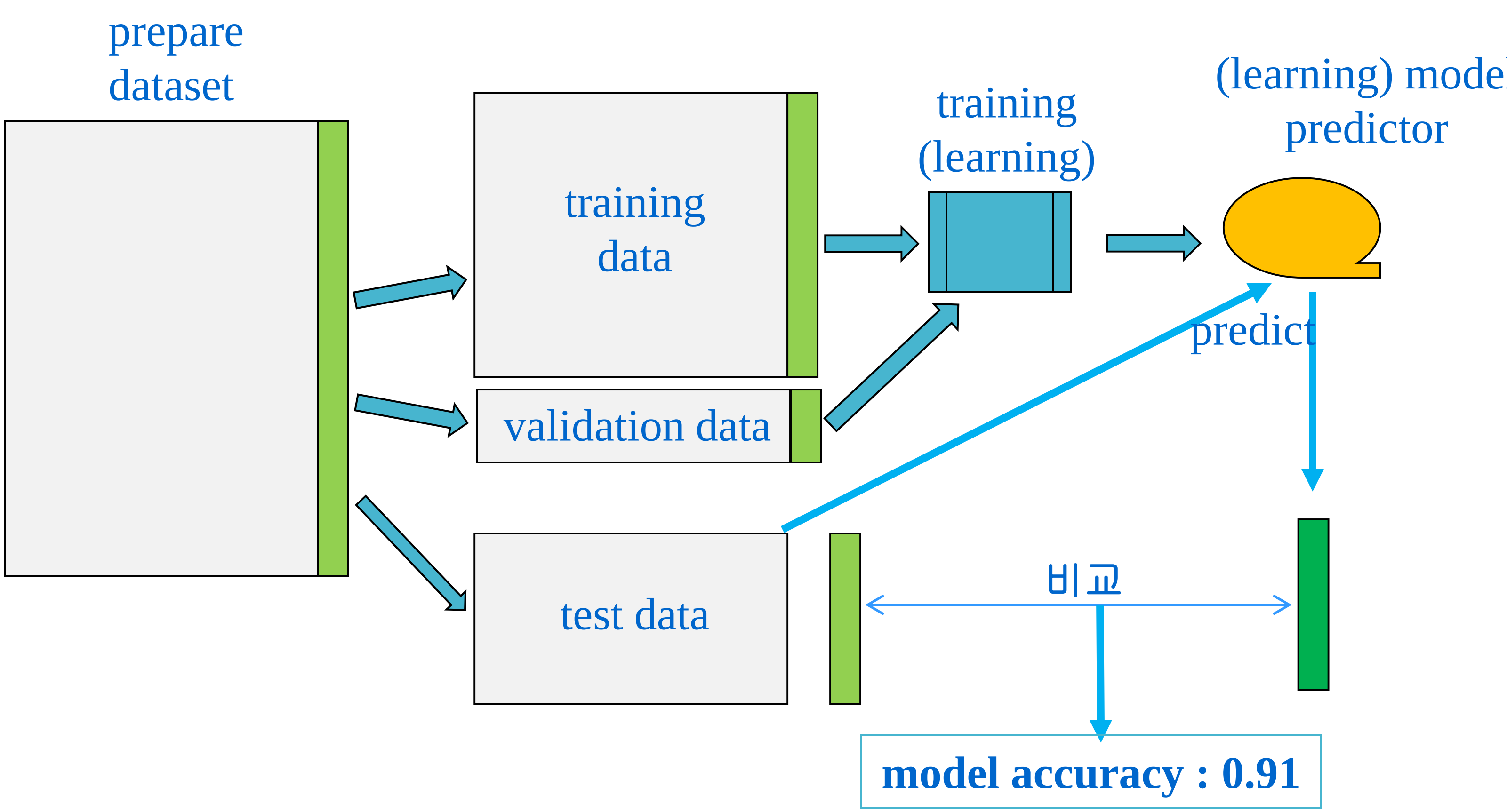
https://stats.stackexchange.com/questions/292283/general-question-regarding-over-fitting-vs-complexity-of-models

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) regression, SVM : regularization
    - 예) neural network : dropout

2. K-fold Cross Validation

- motivation



Is it correct ?

2. K-fold Cross Validation

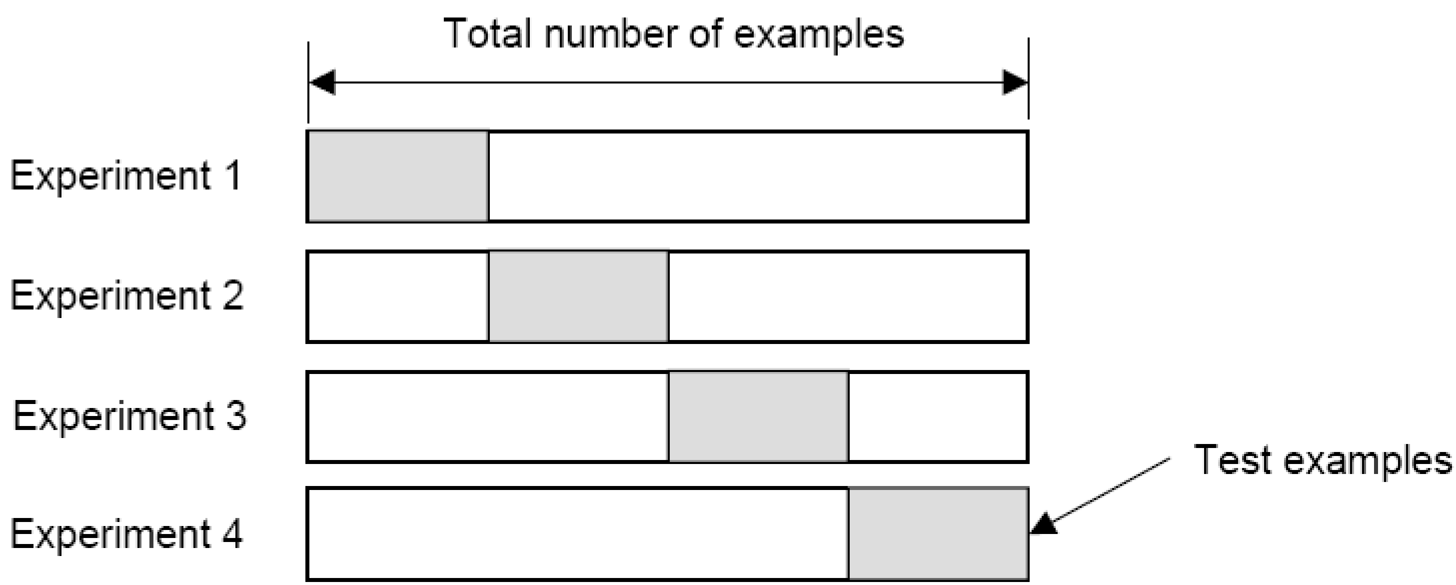
- Only one classification experiment is enough ?

  - Classification accuracy = 0.91 (???)

- 위의 예에서 Test 데이터셋을 다르게 만들면 accuracy 가 달라질 것이다
- Test 데이터셋이 어떻게 구성되었는가에 따라 accuracy 가 원래 성능보다 높거나 낮게 나올 수도 있다.
- 그렇다면 어떻게 해야 분류 모델 또는 분류 알고리즘의 성능 (미래 데이터에 대한)을 보다 정확히 알 수 있을까?

2. K-fold Cross Validation

- Create a K-fold partition of the dataset
  - For each of K experiments, use K-1 folds for training and the remaining one for testing (일반적으로 k=10 을 많이 사용)



  - 모델의 정확도는 각 fold 의 정확도들의 평균으로 계산

$$Acc = \frac{1}{K}\sum_{i=1}^{K} Acc_i$$

2. K-fold Cross Validation

- K-CV 직접구현

06.svm_kfold.py

```
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np

# Load the iris dataset
wine_X, wine_y = datasets.load_wine(return_X_y=True)

# Define fold
kf = KFold(n_splits=5, random_state=123, shuffle=True)  # 5 fold

# Define learning model
model =  svm.SVC()

acc = np.zeros(5)      # accuracy for 5 fold
i = 0                  # fold no
```

Classes	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

```
for train_index, test_index in kf.split(wine_X):
    print("fold:", i)

    train_X, test_X = wine_X[train_index], wine_X[test_index]
    train_y, test_y =  wine_y[train_index], wine_y[test_index]

    # Train the model using the training sets
    model.fit(train_X, train_y)

    # Make predictions using the testing set
    pred_y = model.predict(test_X)

    # model evaluation: accuracy #############
    acc[i] = accuracy_score(test_y, pred_y)
    print('Accuracy : {0:3f}'.format(acc[i]))
    i += 1

print("5 fold :", acc)
print("mean accuracy :", np.mean(acc))
```

2. K-fold Cross Validation

```
fold: 0
Accuracy : 0.500000
fold: 1
Accuracy : 0.694444
fold: 2
Accuracy : 0.722222
fold: 3
Accuracy : 0.685714
fold: 4
Accuracy : 0.714286

In [2]: print("5 fold :", acc)
5 fold : [0.5        0.69444444 0.72222222 0.68571429 0.71428571]

In [3]: print("mean accuracy :", np.mean(acc))
mean accuracy : 0.6633333333333333
```

2. K-fold Cross Validation

- K-fold Cross Validation (simple way)

06.svm_cross_val_score.py

```
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import cross_val_score
import numpy as np

# Load the iris dataset
wine_X, wine_y = datasets.load_wine(return_X_y=True)

# Define learning model
model =  svm.SVC()

scores = cross_val_score(model, wine_X, wine_y, cv=5,
                         scoring='accuracy')

print("fold acc", scores)
print("mean acc", np.mean(scores))
```

2. K-fold Cross Validation

```
fold acc [0.63888889 0.61111111 0.63888889 0.68571429 0.74285714]
mean acc 0.6634920634920635
```

여러 개의 평가 척도를 동시에 적용하려면

```
..
from sklearn.model_selection import cross_validate
..
scores = cross_validate(model, wine_X, wine_y, cv=5,
                        scoring=['accuracy', 'balanced_accuracy'])
print("fold acc", scores)
print("mean acc", np.mean(scores['test_accuracy']))
print("mean balanced-acc", np.mean(scores['test_balanced_accuracy']))
```

```
>>> print("fold acc", scores)
fold acc {'fit_time': array([0.00099778, 0.0009973 , 0.0009973 , 0.00099707, 0.00099754]), 'score_time': array([0.00299191, 0.00199413, 0.00
09973 , 0.00199461, 0.0009973 ]), 'test_accuracy': array([0.63888889, 0.61111111, 0.63888889, 0.68571429, 0.74285714]), 'test_balanced_accur
acy': array([0.62936508, 0.59206349, 0.63492063, 0.62037037, 0.66666667])}
>>> print("mean acc", np.mean( scores['test_accuracy']))
mean acc 0.6634920634920635
>>> print("mean balanced-acc", np.mean( scores['test_balanced_accuracy']))
mean balanced-acc 0.6286772486772486
```

2. K-fold Cross Validation

- scoring

Scoring
Classification
'accuracy'
'balanced_accuracy'
'top_k_accuracy'
'average_precision'
'neg_brier_score'
'f1'
'f1_micro'
'f1_macro'
'f1_weighted'
'f1_samples'
'neg_log_loss'
'precision' etc.
'recall' etc.
'jaccard' etc.
'roc_auc'
'roc_auc_ovr'
'roc_auc_ovo'
'roc_auc_ovr_weighted'
'roc_auc_ovo_weighted'

Regression
'explained_variance'
'max_error'
'neg_mean_absolute_error'
'neg_mean_squared_error'
'neg_root_mean_squared_error'
'neg_mean_squared_log_error'
'neg_median_absolute_error'
'r2'
'neg_mean_poisson_deviance'
'neg_mean_gamma_deviance'
'neg_mean_absolute_percentage_error'
'd2_absolute_error_score'
'd2_pinball_score'
'd2_tweedie_score'

https://scikit-learn.org/stable/modules/model_evaluation.html

2. K-fold Cross Validation

- Note. K-fold cross validation 의 용도
  - K-fold cross validation이 원하는 모델을 도출하지는 않음
  - 주어진 데이터셋으로 모델 개발시 '미래의 정확도'를 추정
  - 최종 모델 개발을 위한 hyper parameter 튜닝에 사용 ☆
  - 전처리시 feature selection 에 사용 ☆

  - K-fold cross validation 에 의해 최적의 hyper parameter 값을 확정하면 전체 데이터를 활용하여 최종 모델을 완성함.

3. 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

sklearn.svm.SVC

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

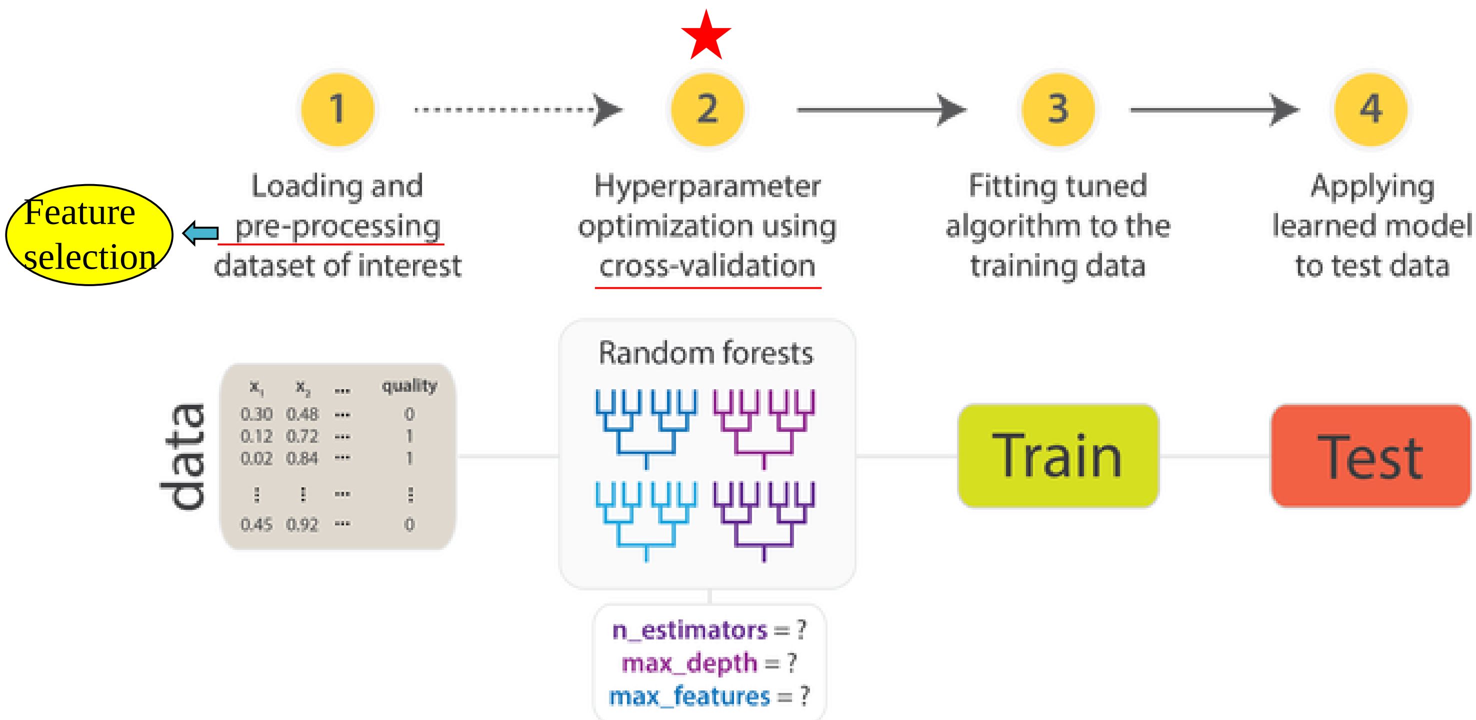
- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

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

- Model building process



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

3. 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. 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)

3. 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']},
 ]
```

3. Hyper parameter tuning

06.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
from sklearn.model_selection import cross_val_score
import pandas as pd
import pprint
import numpy as np

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

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

3. 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')
```

3. Hyper parameter tuning

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

# base model
base_model = RandomForestClassifier(random_state=1234)
scores = cross_val_score(base_model, df_X, df_y, cv=5)
base_accuracy = np.mean(scores)
base_accuracy
```

```
In [22]: base_accuracy
Out[22]: 0.7721840251252017
```

3. Hyper parameter tuning

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

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

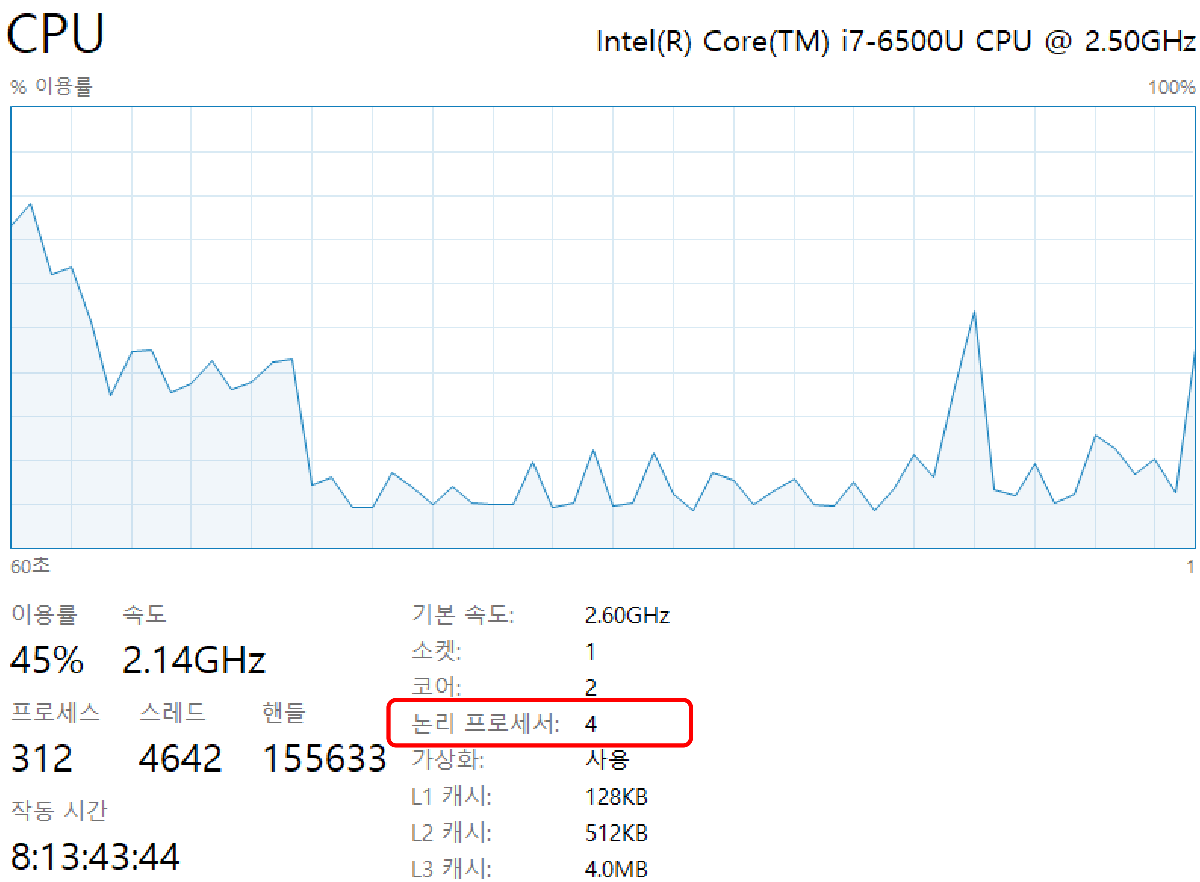
3. Hyper parameter tuning

```python
# 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)

multi-core가 작동하는 경우는 사용하지 않는다.

3. Hyper parameter tuning



3. Hyper parameter tuning

```
# Fit the grid search to the data
grid_search.fit(df_X, df_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
```

multi-core가
작동하는 경우는
이 메시지가
출력되지 않음

(verbose 제외 권장)

```
In [70]: pp.pprint(grid_search.best_params_)
{   'bootstrap': True,
    'max_depth': 80,
    'max_features': 3,
    'min_samples_leaf': 4,
    'min_samples_split': 8,
    'n_estimators': 100}
```

3. Hyper parameter tuning

```
# best model
best_model = grid_search.best_estimator_
best_scores = cross_val_score(best_model, df_X, df_y, cv=5)
best_accuracy = np.mean(best_scores)

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 [68]: print('base acc: {0:0.2f}. best acc :
{1:0.2f}'.format( \
    ...:        base_accuracy, best_accuracy))
base acc: 0.77. best acc : 0.78

In [69]: print('Improvement of {:0.2f}%.'.format( \
    ...:        100 * (best_accuracy - base_accuracy) /
base_accuracy))
Improvement of 1.18%.
```

3. 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

3. Hyper parameter tuning

06.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)
```

3. 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 = [2, 3, 5, '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]
```

3. Hyper parameter tuning

```python
# 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]}
```

3. 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)

3. Hyper parameter tuning

```
# Fit the random search model
rf_random.fit(df_X, df_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 [79]: pp.pprint(rf_random.best_params_)
{   'bootstrap': True,
    'max_depth': 60,
    'max_features': 'sqrt',
    'min_samples_leaf': 2,
    'min_samples_split': 2,
    'n_estimators': 1000}
```

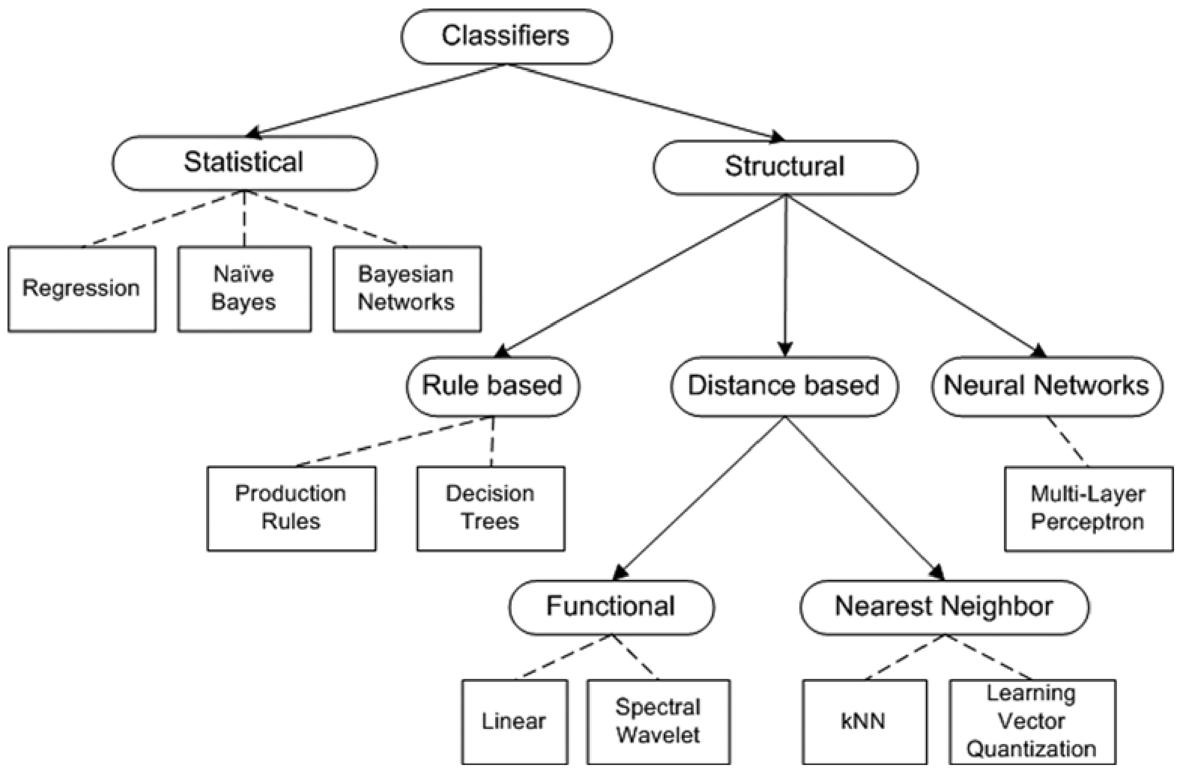
3. Hyper parameter tuning

```
# best model
best_random_model = rf_random.best_estimator_
best_random_scores = cross_val_score(best_random_model, df_X, df_y,
                                     cv=5)
best_random_accuracy = np.mean(best_random_scores)

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 [81]: print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( \
    ...:        base_accuracy, best_random_accuracy))
base acc: 0.77. best acc : 0.78

In [82]: print('Improvement of {:0.2f}%.'.format( 100 *  \
    ...:       (best_random_accuracy - base_accuracy) / base_accuracy))
Improvement of 0.85%.
```

4. 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/

4. Model Comparison

07.model_comparison.py

```
# Model comparison Example

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

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
```

4. Model Comparison

```
# prepare the credit dataset
df = pd.read_csv('D:/data/PimaIndiansDiabetes.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)
```

이 부분 생략해도 정상실행됨

```
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, 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,
```

4. Model Comparison

```
# 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())]
```

4. Model Comparison

```
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        cv_results = cross_val_score(model,
                     df_X, df_y, cv=10, 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)
```

4. Model Comparison

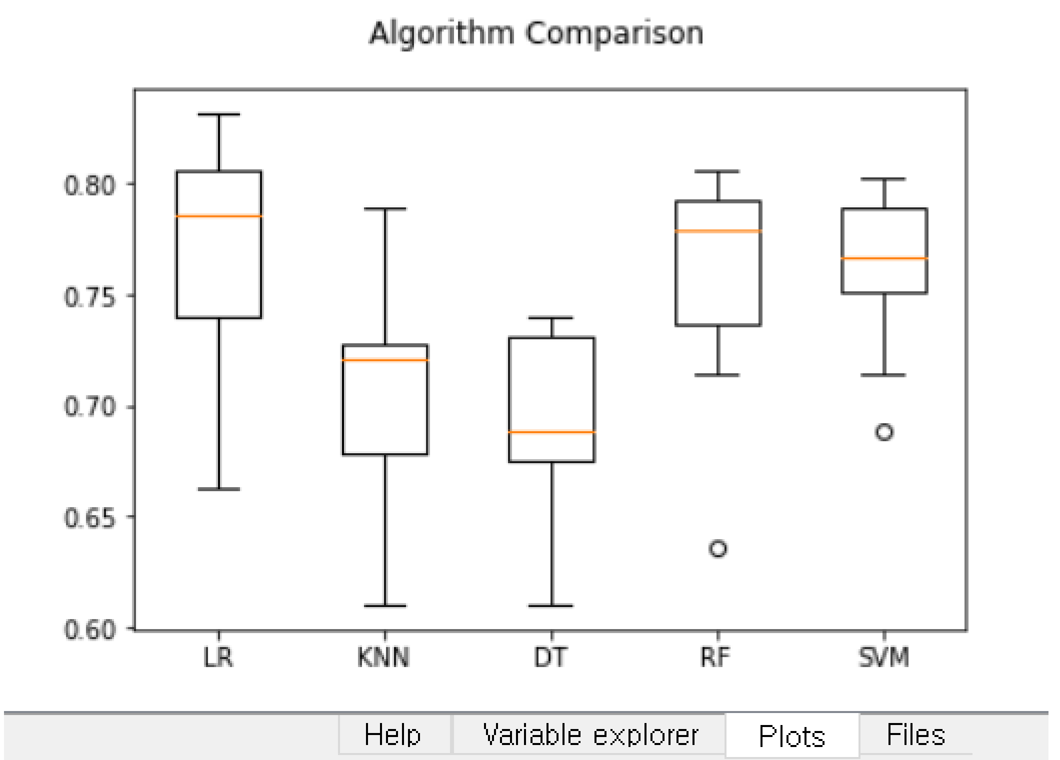
```
print(results)
# average accuracy of classifiers
for i in range(0,len(results)):
    print(names[i] + "\t" + str(round(np.mean(results[i]),4)))
```

```
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
```

4. 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
```

5. Performance metric

- Performance metric
  - Performance evaluation of learning model (classification)

For binary classification model only

- Sensitivity (recall)
- Specificity
- precision
- F1 score
- ROC, AUC

For all classification model

- Accuracy

5. Performance metric

- Binary classification metric

positive (1) : 양성
negative (0) : 음성

Actual (실제값)

Predict (예측값)

	Fact is positive	Fact is negative
Predict as positive	TP	FP
Predict as negative	FN	TN

actual	predict
1	1
0	1
0	0
1	1
0	1
1	0

TP : true positive FP : false positive
FN : false negative TN : true negative

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

(정확도)

5. Performance metric

	Fact is True	Fact is False
Predict as True	TP	FP
Predict as False	FN	TN

- Binary Classification Error

Sensitivity = TP/(TP+FN)
(민감도, recall(재현율))

Specificity = TN/(TN+FP)
(특이도)

- Sensitivity
  - Fraction of all Class1 (True) that we correctly predicted at Class 1
  - *How good are we at finding what we are looking for*
- Specificity
  - Fraction of all Class 2 (False) called Class 2
  - *How many of the Class 2 do we filter out of our Class 1 predictions*

5. Performance metric

	Fact is True	Fact is False
Predict as True	TP	FP
Predict as False	FN	TN

- Binary Classification Error

Precision = TP/(TP+FP)
(정밀도)

환자(Positive), 정상인(Negative)

Sensitivity : 환자를 환자라고 예측한 비율
Specificity : 정상인을 정상인이라고 예측한 비율
Precision : 환자라고 예측한 것 중에서 실제 환자의 비율

어떤 평가기준이라도 값이 클수록 좋다!

5. Performance metric

- Binary classification metric
  - F1 score : harmonic mean of precision and recall
    - precision 과 recall 의 불균형에 대해 감점이 이루어짐

$$F1\ score = \frac{2 \times recall \times precision}{recall + precision}$$

recall	precision	F1 score
1	1	1
1	0	0
0.8	0.8	0.8
0.8	0.5	0.62

5. Performance metric

- How to calculate sensitivity, specificity,.. for multi-class model ?

class A, class B, class C

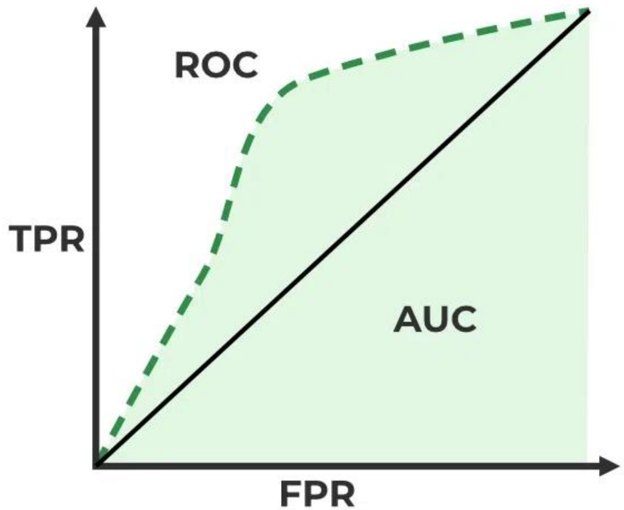
For class A:
positive : class A, negative: class B,C

For class B:
positive : class B, negative: class A,C

For class C:
positive : class C, negative: class A,B

5. Performance metric

- Binary classification metric
  - roc-auc



https://www.geeksforgeeks.org/auc-roc-curve/

- Accuracy는 imbalance dataset에 대해 올바로 평가하지 못함
- AUC는 imbalance의 정도에 자유로움

- ROC curve 해석 설명 : https://nittaku.tistory.com/297

5. Performance metric

- Python metric
  - https://scikit-learn.org/stable/modules/model_evaluation.html

Scoring	Function	Comment
Classification		
'accuracy'	`metrics.accuracy_score`	
'balanced_accuracy'	`metrics.balanced_accuracy_score`	for imbalanced datasets.
'average_precision'	`metrics.average_precision_score`	
'neg_brier_score'	`metrics.brier_score_loss`	
'f1'	`metrics.f1_score`	for binary targets
'f1_micro'	`metrics.f1_score`	micro-averaged
'f1_macro'	`metrics.f1_score`	macro-averaged
'f1_weighted'	`metrics.f1_score`	weighted average
'f1_samples'	`metrics.f1_score`	by multilabel sample
'neg_log_loss'	`metrics.log_loss`	requires `predict_proba` support
'precision' etc.	`metrics.precision_score`	suffixes apply as with 'f1'
'recall' etc.	`metrics.recall_score`	suffixes apply as with 'f1'
'jaccard' etc.	`metrics.jaccard_score`	suffixes apply as with 'f1'
'roc_auc'	`metrics.roc_auc_score`	
'roc_auc_ovr'	`metrics.roc_auc_score`	
'roc_auc_ovo'	`metrics.roc_auc_score`	
'roc_auc_ovr_weighted'	`metrics.roc_auc_score`	
'roc_auc_ovo_weighted'	`metrics.roc_auc_score`	

5. Performance metric

Clustering	
'adjusted_mutual_info_score'	`metrics.adjusted_mutual_info_score`
'adjusted_rand_score'	`metrics.adjusted_rand_score`
'completeness_score'	`metrics.completeness_score`
'fowlkes_mallows_score'	`metrics.fowlkes_mallows_score`
'homogeneity_score'	`metrics.homogeneity_score`
'mutual_info_score'	`metrics.mutual_info_score`
'normalized_mutual_info_score'	`metrics.normalized_mutual_info_score`
'v_measure_score'	`metrics.v_measure_score`
Regression	
'explained_variance'	`metrics.explained_variance_score`
'max_error'	`metrics.max_error`
'neg_mean_absolute_error'	`metrics.mean_absolute_error`
'neg_mean_squared_error'	`metrics.mean_squared_error`
'neg_root_mean_squared_error'	`metrics.mean_squared_error`
'neg_mean_squared_log_error'	`metrics.mean_squared_log_error`
'neg_median_absolute_error'	`metrics.median_absolute_error`
'r2'	`metrics.r2_score`
'neg_mean_poisson_deviance'	`metrics.mean_poisson_deviance`
'neg_mean_gamma_deviance'	`metrics.mean_gamma_deviance`

5. Performance metric

- example

```
from sklearn.metrics import accuracy_score

test_y = [2, 0, 2, 2, 0, 1]
pred_y = [0, 0, 2, 2, 0, 2]

acc = accuracy_score(test_y, pred_y)
print(acc)
```

```
In [250]: print(acc)
0.6666666666666666
```

5. Performance metric

- Confusion matrix

```
from sklearn.metrics import confusion_matrix
test_y = [2, 0, 2, 2, 0, 1]
pred_y = [0, 0, 2, 2, 0, 2]
confusion_matrix(test_y, pred_y)
```

```
In [224]: confusion_matrix(test_y, pred_y)
Out[224]:
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]], dtype=int64)
```

$$
\begin{array}{ccc}
 & & \text{pred\_y} \\
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
\text{test\_y} & \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \begin{array}{c} [2,\ 0,\ 0] \\ [0,\ 0,\ 1] \\ [1,\ 0,\ 2] \end{array}
\end{array}
$$

```
# binary classification
test_y = [1, 0, 0, 1, 0, 1]
pred_y = [0, 0, 0, 1, 0, 1]
tn, fp, fn, tp = confusion_matrix(test_y, pred_y).ravel()
(tn, fp, fn, tp)
```

```
In [243]: (tn, fp, fn, tp)
Out[243]: (3, 0, 1, 2)
```