Classification

Boosting - XGBoost

- 뛰어난 성능
- GBM 대비 빠른 수행 시간(CPU 병렬 처리, GPU 지원)
- 다양한 성능 향상 기능

■ 규제 (Regularization) 기능 탑재

■ Tree Pruning(Node들을 다 만들었다가 leaf node부터 다시 검증하여 pruning) : leaf node가 overfitting 을 반영한다고 보고 penalty를 부여하는 등의 방식을 통해 가지치기

- 다양한 편의 기능

■ EarlyStopping : 수행 속도 감소 및 overfitting 방지

■ 내장된 교차 검증

■ 결측값 자체 처리

In [1]:
```
import numpy as np
import pandas as pd
import xgboost as xgb
from xgboost import plot_importance
from xgboost import XGBClassifier

import warnings

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics

from sklearn.tree import export_graphviz
warnings.filterwarnings('ignore')

dataset = load_wine()
X_features = dataset.data
y_label = dataset.target

wine_df = pd.DataFrame(data = X_features, columns = dataset.feature_names)
wine_df['target'] = y_label
wine_df.head(3)
```

Out[1]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_pher
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	

In [2]:
```
wine_df
```

Out[2]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_ph
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

178 rows × 14 columns

print(dataset.target_names) print(wine_df['target'].value_counts())

In [3]:
```
wine = load_wine()
wine.keys()

wine_pd = pd.DataFrame(data=np.c_[wine['data'], wine['target']],
                       columns=wine['feature_names'] + ['target'])
X_train, X_test, y_train, y_test = train_test_split(X_features, y_label, test_size=0.2, ran
dom_state=42) # 8 : 2
print(X_train.shape, X_test.shape)
```

```
(142, 13) (36, 13)
```

- XGBoost에서 Input 값으로 사용 위해 train, test set을 DMatrix로 변환

In [4]:
```
dtrain = xgb.DMatrix(data = X_train, label = y_train)
dtest = xgb.DMatrix(data = X_test, label = y_test)
```

- 하이퍼 파라미터 설정

In [5]:
```
params = { 'max_depth':3,
           'eta': 0.1, # Learning Rate
           'objective':'multi:softmax', # 다중 분류
           'num_class': 3,
           'eval_metric':'mlogloss',
           'early_stoppings':100
        }
num_rounds = 150
```

- 주어진 하이퍼 파라미터와 early stopping 파라미터를 train( ) 함수의 파라미터로 전달하고 학습

In [6]:
```
# train 데이터 셋은 'train' , evaluation(test) 데이터 셋은 'eval'
wlist = [ (dtrain, 'train'), (dtest, 'eval') ]
# 하이퍼 파라미터와 early stopping 파라미터를 train( ) 함수의 파라미터로 전달
xgb_model = xgb.train(params = params, dtrain = dtrain, num_boost_round = num_rounds, evals
= wlist )
```

```
[13:03:03] WARNING: C:WUsersWAdministratorWworkspaceWxgboost-win64_release_1.2.0WsrcWlearne
r.cc:516:
Parameters: { early_stoppings } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[0]     train-mlogloss:0.97301  eval-mlogloss:0.98561
[1]     train-mlogloss:0.86611  eval-mlogloss:0.88885
[2]     train-mlogloss:0.77496  eval-mlogloss:0.80796
[3]     train-mlogloss:0.69633  eval-mlogloss:0.73686
[4]     train-mlogloss:0.62601  eval-mlogloss:0.67307
[5]     train-mlogloss:0.56375  eval-mlogloss:0.61272
[6]     train-mlogloss:0.50942  eval-mlogloss:0.56015
[7]     train-mlogloss:0.46061  eval-mlogloss:0.51296
[8]     train-mlogloss:0.41674  eval-mlogloss:0.47134
[9]     train-mlogloss:0.37807  eval-mlogloss:0.43487
[10]    train-mlogloss:0.34355  eval-mlogloss:0.40217
[11]    train-mlogloss:0.31233  eval-mlogloss:0.37300
[12]    train-mlogloss:0.28474  eval-mlogloss:0.34749
[13]    train-mlogloss:0.25969  eval-mlogloss:0.32453
[14]    train-mlogloss:0.23738  eval-mlogloss:0.30201
[15]    train-mlogloss:0.21704  eval-mlogloss:0.28384
[16]    train-mlogloss:0.19899  eval-mlogloss:0.26530
[17]    train-mlogloss:0.18246  eval-mlogloss:0.24943
[18]    train-mlogloss:0.16753  eval-mlogloss:0.23623
[19]    train-mlogloss:0.15402  eval-mlogloss:0.22316
[20]    train-mlogloss:0.14161  eval-mlogloss:0.21332
[21]    train-mlogloss:0.13068  eval-mlogloss:0.20197
[22]    train-mlogloss:0.12032  eval-mlogloss:0.19309
[23]    train-mlogloss:0.11117  eval-mlogloss:0.18384
[24]    train-mlogloss:0.10294  eval-mlogloss:0.17550
[25]    train-mlogloss:0.09548  eval-mlogloss:0.16792
[26]    train-mlogloss:0.08890  eval-mlogloss:0.16209
[27]    train-mlogloss:0.08265  eval-mlogloss:0.15725
[28]    train-mlogloss:0.07695  eval-mlogloss:0.15009
[29]    train-mlogloss:0.07200  eval-mlogloss:0.14667
[30]    train-mlogloss:0.06720  eval-mlogloss:0.14079
[31]    train-mlogloss:0.06308  eval-mlogloss:0.13759
[32]    train-mlogloss:0.05936  eval-mlogloss:0.13385
[33]    train-mlogloss:0.05582  eval-mlogloss:0.13092
[34]    train-mlogloss:0.05272  eval-mlogloss:0.12799
[35]    train-mlogloss:0.04959  eval-mlogloss:0.12395
[36]    train-mlogloss:0.04685  eval-mlogloss:0.12076
[37]    train-mlogloss:0.04427  eval-mlogloss:0.11751
[38]    train-mlogloss:0.04207  eval-mlogloss:0.11635
[39]    train-mlogloss:0.04002  eval-mlogloss:0.11567
[40]    train-mlogloss:0.03805  eval-mlogloss:0.11206
[41]    train-mlogloss:0.03609  eval-mlogloss:0.11068
[42]    train-mlogloss:0.03452  eval-mlogloss:0.10991
[43]    train-mlogloss:0.03298  eval-mlogloss:0.10774
[44]    train-mlogloss:0.03149  eval-mlogloss:0.10693
[45]    train-mlogloss:0.03022  eval-mlogloss:0.10361
[46]    train-mlogloss:0.02881  eval-mlogloss:0.10182
[47]    train-mlogloss:0.02761  eval-mlogloss:0.10187
[48]    train-mlogloss:0.02645  eval-mlogloss:0.10000
[49]    train-mlogloss:0.02540  eval-mlogloss:0.09947
[50]    train-mlogloss:0.02450  eval-mlogloss:0.09943
[51]    train-mlogloss:0.02358  eval-mlogloss:0.09901
[52]    train-mlogloss:0.02278  eval-mlogloss:0.09878
[53]    train-mlogloss:0.02196  eval-mlogloss:0.09709
[54]    train-mlogloss:0.02119  eval-mlogloss:0.09585
[55]    train-mlogloss:0.02047  eval-mlogloss:0.09640
```

```
[56]    train-mlogloss:0.01976  eval-mlogloss:0.09445
[57]    train-mlogloss:0.01915  eval-mlogloss:0.09451
[58]    train-mlogloss:0.01857  eval-mlogloss:0.09451
[59]    train-mlogloss:0.01803  eval-mlogloss:0.09354
[60]    train-mlogloss:0.01755  eval-mlogloss:0.09335
[61]    train-mlogloss:0.01705  eval-mlogloss:0.09125
[62]    train-mlogloss:0.01659  eval-mlogloss:0.09157
[63]    train-mlogloss:0.01616  eval-mlogloss:0.09135
[64]    train-mlogloss:0.01579  eval-mlogloss:0.09117
[65]    train-mlogloss:0.01543  eval-mlogloss:0.09003
[66]    train-mlogloss:0.01509  eval-mlogloss:0.08920
[67]    train-mlogloss:0.01477  eval-mlogloss:0.08918
[68]    train-mlogloss:0.01446  eval-mlogloss:0.08746
[69]    train-mlogloss:0.01416  eval-mlogloss:0.08746
[70]    train-mlogloss:0.01384  eval-mlogloss:0.08604
[71]    train-mlogloss:0.01357  eval-mlogloss:0.08555
[72]    train-mlogloss:0.01331  eval-mlogloss:0.08458
[73]    train-mlogloss:0.01306  eval-mlogloss:0.08445
[74]    train-mlogloss:0.01282  eval-mlogloss:0.08384
[75]    train-mlogloss:0.01269  eval-mlogloss:0.08343
[76]    train-mlogloss:0.01256  eval-mlogloss:0.08303
[77]    train-mlogloss:0.01244  eval-mlogloss:0.08220
[78]    train-mlogloss:0.01232  eval-mlogloss:0.08182
[79]    train-mlogloss:0.01221  eval-mlogloss:0.08097
[80]    train-mlogloss:0.01209  eval-mlogloss:0.08062
[81]    train-mlogloss:0.01199  eval-mlogloss:0.07981
[82]    train-mlogloss:0.01188  eval-mlogloss:0.07948
[83]    train-mlogloss:0.01178  eval-mlogloss:0.07874
[84]    train-mlogloss:0.01169  eval-mlogloss:0.07798
[85]    train-mlogloss:0.01163  eval-mlogloss:0.07789
[86]    train-mlogloss:0.01158  eval-mlogloss:0.07737
[87]    train-mlogloss:0.01153  eval-mlogloss:0.07691
[88]    train-mlogloss:0.01147  eval-mlogloss:0.07682
[89]    train-mlogloss:0.01142  eval-mlogloss:0.07637
[90]    train-mlogloss:0.01137  eval-mlogloss:0.07588
[91]    train-mlogloss:0.01133  eval-mlogloss:0.07579
[92]    train-mlogloss:0.01128  eval-mlogloss:0.07536
[93]    train-mlogloss:0.01124  eval-mlogloss:0.07489
[94]    train-mlogloss:0.01119  eval-mlogloss:0.07481
[95]    train-mlogloss:0.01115  eval-mlogloss:0.07439
[96]    train-mlogloss:0.01111  eval-mlogloss:0.07398
[97]    train-mlogloss:0.01107  eval-mlogloss:0.07390
[98]    train-mlogloss:0.01103  eval-mlogloss:0.07351
[99]    train-mlogloss:0.01099  eval-mlogloss:0.07312
[100]   train-mlogloss:0.01095  eval-mlogloss:0.07304
[101]   train-mlogloss:0.01091  eval-mlogloss:0.07268
[102]   train-mlogloss:0.01088  eval-mlogloss:0.07231
[103]   train-mlogloss:0.01085  eval-mlogloss:0.07224
[104]   train-mlogloss:0.01081  eval-mlogloss:0.07187
[105]   train-mlogloss:0.01078  eval-mlogloss:0.07153
[106]   train-mlogloss:0.01075  eval-mlogloss:0.07147
[107]   train-mlogloss:0.01072  eval-mlogloss:0.07114
[108]   train-mlogloss:0.01069  eval-mlogloss:0.07108
[109]   train-mlogloss:0.01066  eval-mlogloss:0.07077
[110]   train-mlogloss:0.01063  eval-mlogloss:0.07101
[111]   train-mlogloss:0.01060  eval-mlogloss:0.07096
[112]   train-mlogloss:0.01057  eval-mlogloss:0.07119
[113]   train-mlogloss:0.01055  eval-mlogloss:0.07088
[114]   train-mlogloss:0.01052  eval-mlogloss:0.07112
[115]   train-mlogloss:0.01049  eval-mlogloss:0.07081
[116]   train-mlogloss:0.01047  eval-mlogloss:0.07104
[117]   train-mlogloss:0.01044  eval-mlogloss:0.07074
[118]   train-mlogloss:0.01042  eval-mlogloss:0.07097
[119]   train-mlogloss:0.01039  eval-mlogloss:0.07068
[120]   train-mlogloss:0.01037  eval-mlogloss:0.07090
[121]   train-mlogloss:0.01035  eval-mlogloss:0.07061
```

```
[122]   train-mlogloss:0.01032  eval-mlogloss:0.07083
[123]   train-mlogloss:0.01030  eval-mlogloss:0.07054
[124]   train-mlogloss:0.01028  eval-mlogloss:0.07076
[125]   train-mlogloss:0.01025  eval-mlogloss:0.07048
[126]   train-mlogloss:0.01023  eval-mlogloss:0.07069
[127]   train-mlogloss:0.01021  eval-mlogloss:0.07041
[128]   train-mlogloss:0.01019  eval-mlogloss:0.07062
[129]   train-mlogloss:0.01017  eval-mlogloss:0.07035
[130]   train-mlogloss:0.01015  eval-mlogloss:0.07056
[131]   train-mlogloss:0.01013  eval-mlogloss:0.07029
[132]   train-mlogloss:0.01011  eval-mlogloss:0.07049
[133]   train-mlogloss:0.01009  eval-mlogloss:0.07023
[134]   train-mlogloss:0.01007  eval-mlogloss:0.07043
[135]   train-mlogloss:0.01005  eval-mlogloss:0.07017
[136]   train-mlogloss:0.01004  eval-mlogloss:0.07037
[137]   train-mlogloss:0.01002  eval-mlogloss:0.07012
[138]   train-mlogloss:0.01000  eval-mlogloss:0.07031
[139]   train-mlogloss:0.00998  eval-mlogloss:0.07007
[140]   train-mlogloss:0.00996  eval-mlogloss:0.06982
[141]   train-mlogloss:0.00995  eval-mlogloss:0.07001
[142]   train-mlogloss:0.00993  eval-mlogloss:0.06978
[143]   train-mlogloss:0.00991  eval-mlogloss:0.06955
[144]   train-mlogloss:0.00990  eval-mlogloss:0.06932
[145]   train-mlogloss:0.00988  eval-mlogloss:0.06911
[146]   train-mlogloss:0.00987  eval-mlogloss:0.06929
[147]   train-mlogloss:0.00985  eval-mlogloss:0.06908
[148]   train-mlogloss:0.00984  eval-mlogloss:0.06927
[149]   train-mlogloss:0.00982  eval-mlogloss:0.06906
```

In [7]:
```
pred_probs = xgb_model.predict(dtest) # 예측 확률 값
preds = np.round(pred_probs, 1)
```

In [8]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score

def get_eval(y_test, pred = None, pred_proba = None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred, average = 'micro')
    recall = recall_score(y_test, pred, average = 'micro')
    f1 = f1_score(y_test,pred, average = 'micro')

    print(confusion) # Confusion Matrix 출력
    print("정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, F1: {3:.4f}".format(accur
acy, precision, recall, f1))
```

In [9]:
```
get_eval(y_test, preds)
```

```
[[14  0  0]
 [ 0 14  0]
 [ 0  1  7]]
정확도: 0.9722, 정밀도: 0.9722, 재현율: 0.9722, F1: 0.9722
```

Boosting - LightGBM

- XGBoost 이후에 나와 XGBoost의 단점을 보완하고 성능을 높이는데 초점이 맞춰져 있다. XGBoost 대비 장점
- 더 빠른 학습과 예측 수행 시간(수행 시간은 설왕설래가 있다.)
- 더 작은 메모리 사용량
- 카테고리형 feature의 자동 변환과 최적 변환. One-hot Encoding을 사용하지 않고도 카테고리형 feature를 최적으로 변환하고 이에 따른 노드 변환 수행

In [10]:
```
from lightgbm import LGBMClassifier

dataset = load_wine()
X_features = dataset.data
y_label = dataset.target

wine_df = pd.DataFrame(data = X_features, columns = dataset.feature_names)
wine_df['target'] = y_label
```

In [11]:
```
lgbm_wrapper = LGBMClassifier(n_estimators = 150)

evals = [(X_test, y_test)]
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds = 100, eval_set = evals, verbose =
True)

preds = lgbm_wrapper.predict(X_test)
pred_proba = lgbm_wrapper.predict_proba(X_test)[:, 1]
```

```
[1]	valid_0's multi_logloss: 0.93247
Training until validation scores don't improve for 100 rounds
[2]	valid_0's multi_logloss: 0.826273
[3]	valid_0's multi_logloss: 0.737772
[4]	valid_0's multi_logloss: 0.660022
[5]	valid_0's multi_logloss: 0.589707
[6]	valid_0's multi_logloss: 0.537431
[7]	valid_0's multi_logloss: 0.488267
[8]	valid_0's multi_logloss: 0.442401
[9]	valid_0's multi_logloss: 0.407765
[10]	valid_0's multi_logloss: 0.372497
[11]	valid_0's multi_logloss: 0.343244
[12]	valid_0's multi_logloss: 0.319435
[13]	valid_0's multi_logloss: 0.297106
[14]	valid_0's multi_logloss: 0.276194
[15]	valid_0's multi_logloss: 0.256467
[16]	valid_0's multi_logloss: 0.239311
[17]	valid_0's multi_logloss: 0.222908
[18]	valid_0's multi_logloss: 0.209475
[19]	valid_0's multi_logloss: 0.197891
[20]	valid_0's multi_logloss: 0.185007
[21]	valid_0's multi_logloss: 0.175826
[22]	valid_0's multi_logloss: 0.163459
[23]	valid_0's multi_logloss: 0.153675
[24]	valid_0's multi_logloss: 0.143801
[25]	valid_0's multi_logloss: 0.135417
[26]	valid_0's multi_logloss: 0.128361
[27]	valid_0's multi_logloss: 0.122586
[28]	valid_0's multi_logloss: 0.113909
[29]	valid_0's multi_logloss: 0.107625
[30]	valid_0's multi_logloss: 0.101607
[31]	valid_0's multi_logloss: 0.0950204
[32]	valid_0's multi_logloss: 0.0914064
[33]	valid_0's multi_logloss: 0.0869924
[34]	valid_0's multi_logloss: 0.0818646
[35]	valid_0's multi_logloss: 0.0767855
[36]	valid_0's multi_logloss: 0.0743221
[37]	valid_0's multi_logloss: 0.0720034
[38]	valid_0's multi_logloss: 0.0701426
[39]	valid_0's multi_logloss: 0.0687016
[40]	valid_0's multi_logloss: 0.0651532
[41]	valid_0's multi_logloss: 0.0637023
[42]	valid_0's multi_logloss: 0.0625905
[43]	valid_0's multi_logloss: 0.0600529
[44]	valid_0's multi_logloss: 0.0590979
[45]	valid_0's multi_logloss: 0.0583626
[46]	valid_0's multi_logloss: 0.0584329
[47]	valid_0's multi_logloss: 0.0572606
[48]	valid_0's multi_logloss: 0.0540785
[49]	valid_0's multi_logloss: 0.0546949
[50]	valid_0's multi_logloss: 0.0527401
[51]	valid_0's multi_logloss: 0.0507414
[52]	valid_0's multi_logloss: 0.0479783
[53]	valid_0's multi_logloss: 0.0487248
[54]	valid_0's multi_logloss: 0.0463448
[55]	valid_0's multi_logloss: 0.0466639
[56]	valid_0's multi_logloss: 0.0439232
[57]	valid_0's multi_logloss: 0.044244
[58]	valid_0's multi_logloss: 0.0432298
[59]	valid_0's multi_logloss: 0.0431848
[60]	valid_0's multi_logloss: 0.0407052
[61]	valid_0's multi_logloss: 0.0416899
[62]	valid_0's multi_logloss: 0.0398434
[63]	valid_0's multi_logloss: 0.0398797
[64]	valid_0's multi_logloss: 0.0373701
```

```
[65]    valid_0's multi_logloss: 0.0354254
[66]    valid_0's multi_logloss: 0.0342176
[67]    valid_0's multi_logloss: 0.034285
[68]    valid_0's multi_logloss: 0.0325448
[69]    valid_0's multi_logloss: 0.0329103
[70]    valid_0's multi_logloss: 0.0326795
[71]    valid_0's multi_logloss: 0.0304978
[72]    valid_0's multi_logloss: 0.0310014
[73]    valid_0's multi_logloss: 0.0298282
[74]    valid_0's multi_logloss: 0.0284437
[75]    valid_0's multi_logloss: 0.0284083
[76]    valid_0's multi_logloss: 0.0276243
[77]    valid_0's multi_logloss: 0.0269481
[78]    valid_0's multi_logloss: 0.0270356
[79]    valid_0's multi_logloss: 0.025382
[80]    valid_0's multi_logloss: 0.0249337
[81]    valid_0's multi_logloss: 0.0248124
[82]    valid_0's multi_logloss: 0.0243542
[83]    valid_0's multi_logloss: 0.0235941
[84]    valid_0's multi_logloss: 0.0227857
[85]    valid_0's multi_logloss: 0.0211054
[86]    valid_0's multi_logloss: 0.0204188
[87]    valid_0's multi_logloss: 0.0208618
[88]    valid_0's multi_logloss: 0.0195575
[89]    valid_0's multi_logloss: 0.0186794
[90]    valid_0's multi_logloss: 0.0174409
[91]    valid_0's multi_logloss: 0.0170504
[92]    valid_0's multi_logloss: 0.0167476
[93]    valid_0's multi_logloss: 0.015453
[94]    valid_0's multi_logloss: 0.0142442
[95]    valid_0's multi_logloss: 0.013407
[96]    valid_0's multi_logloss: 0.0129285
[97]    valid_0's multi_logloss: 0.0123832
[98]    valid_0's multi_logloss: 0.0114736
[99]    valid_0's multi_logloss: 0.0108068
[100]   valid_0's multi_logloss: 0.0110696
[101]   valid_0's multi_logloss: 0.0102442
[102]   valid_0's multi_logloss: 0.010459
[103]   valid_0's multi_logloss: 0.0100306
[104]   valid_0's multi_logloss: 0.00976545
[105]   valid_0's multi_logloss: 0.00950385
[106]   valid_0's multi_logloss: 0.00950191
[107]   valid_0's multi_logloss: 0.00926028
[108]   valid_0's multi_logloss: 0.00874029
[109]   valid_0's multi_logloss: 0.00906543
[110]   valid_0's multi_logloss: 0.00912147
[111]   valid_0's multi_logloss: 0.00926517
[112]   valid_0's multi_logloss: 0.00881004
[113]   valid_0's multi_logloss: 0.00864635
[114]   valid_0's multi_logloss: 0.00865149
[115]   valid_0's multi_logloss: 0.00827492
[116]   valid_0's multi_logloss: 0.00832687
[117]   valid_0's multi_logloss: 0.00835935
[118]   valid_0's multi_logloss: 0.00819979
[119]   valid_0's multi_logloss: 0.00813502
[120]   valid_0's multi_logloss: 0.00781386
[121]   valid_0's multi_logloss: 0.00757848
[122]   valid_0's multi_logloss: 0.00769003
[123]   valid_0's multi_logloss: 0.00752223
[124]   valid_0's multi_logloss: 0.007359
[125]   valid_0's multi_logloss: 0.00729009
[126]   valid_0's multi_logloss: 0.00717366
[127]   valid_0's multi_logloss: 0.00709924
[128]   valid_0's multi_logloss: 0.00705838
[129]   valid_0's multi_logloss: 0.00693292
[130]   valid_0's multi_logloss: 0.00686772
```

```
[131]   valid_0's multi_logloss: 0.00683431
[132]   valid_0's multi_logloss: 0.00671904
[133]   valid_0's multi_logloss: 0.00666497
[134]   valid_0's multi_logloss: 0.00670171
[135]   valid_0's multi_logloss: 0.00659455
[136]   valid_0's multi_logloss: 0.00654126
[137]   valid_0's multi_logloss: 0.006579
[138]   valid_0's multi_logloss: 0.00647849
[139]   valid_0's multi_logloss: 0.00651936
[140]   valid_0's multi_logloss: 0.00646354
[141]   valid_0's multi_logloss: 0.00636922
[142]   valid_0's multi_logloss: 0.00641081
[143]   valid_0's multi_logloss: 0.00635555
[144]   valid_0's multi_logloss: 0.00630391
[145]   valid_0's multi_logloss: 0.00621506
[146]   valid_0's multi_logloss: 0.00616724
[147]   valid_0's multi_logloss: 0.00606498
[148]   valid_0's multi_logloss: 0.00602134
[149]   valid_0's multi_logloss: 0.00597986
[150]   valid_0's multi_logloss: 0.00600745
Did not meet early stopping. Best iteration is:
[149]   valid_0's multi_logloss: 0.00597986
```

In [12]:
```
get_eval(y_test, preds, pred_proba)
```

```
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
정확도: 1.0000, 정밀도: 1.0000, 재현율: 1.0000, F1: 1.0000
```

Bagging - Random Forest

- 랜덤 포레스트는 다재 다능한 알고리즘이다. 빠른 수행 속도, 높은 예측 성능을 보인다.
- 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을 한다.
- 여러 개의 데이터 세트를 중첩되게 분리하는 것을 부트스트래핑(bootstrapping) 분할 방식이라고한다. (원본 데이터의 건수가 10개인 학습 데이터 세트에 랜덤 포레스트를 3개의 결정 트리 기반으로 학습하려고 n_estimators = 3으로 하이퍼 파라미터를 부여하면 다음과 같이 데이터 서브 세트가 만들어진다.)

In [13]:
```
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
```

In [14]:
```
dataset = load_wine()
X_features = dataset.data
y_label = dataset.target
```

In [15]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X_features, y_label, te
st_size=0.2, random_state = 42)

estimator = RandomForestClassifier()


estimator.fit(X_train, y_train)
preds = estimator.predict(X_test)
pred_probs = estimator.predict(X_test)[:1]
get_eval(y_test, preds, pred_probs)
```

```
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
정확도: 1.0000, 정밀도: 1.0000, 재현율: 1.0000, F1: 1.0000
```

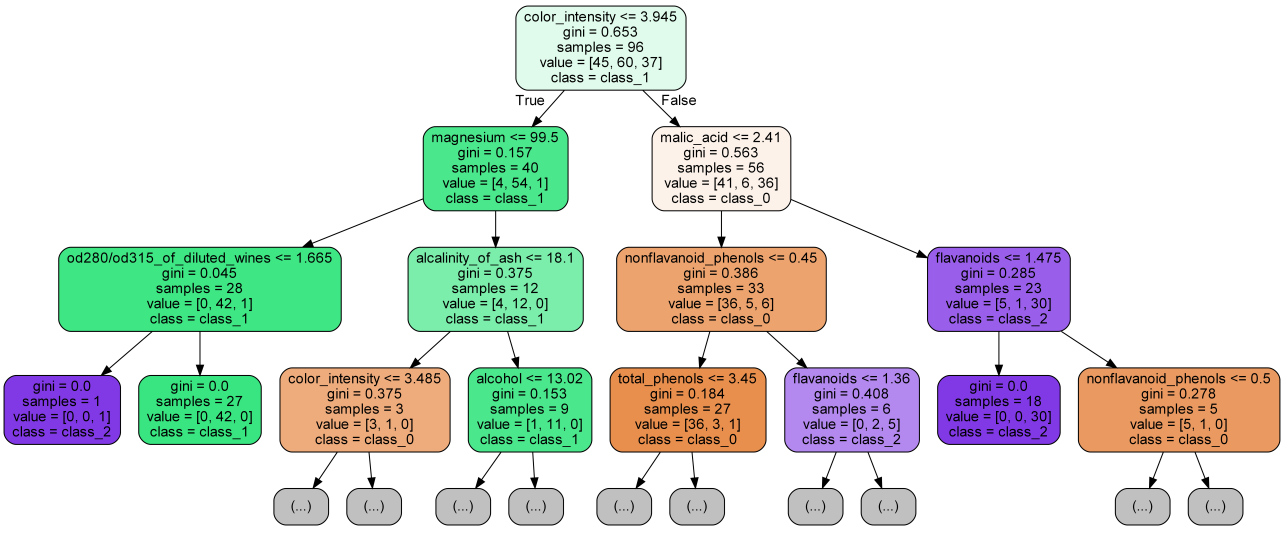
지니 계수 : 다양성이 낮을수록(불평등 할수록) 균일도가 높다.

- 최종적으로 하나의 종류만 남아 있을 때가 균일도가 높다,

- petal length
- gini 지니계수
- samples 데이터 건수
- value = []
- 색깔이 진할수록 단일화 분류 값을 가지고 있다. 연하면 불확실성이 높다

In [16]:
```
model = estimator.estimators_[3]
export_graphviz(model, out_file='tree.dot',
                feature_names = dataset.feature_names,
                class_names = dataset.target_names,
                max_depth = 3, # 표현하고 싶은 최대 depth
                precision = 3, # 소수점 표기 자릿수
                filled = True, # class별 color 채우기
                rounded=True, # 박스의 모양을 둥글게
               )

# 생성된 .dot 파일을 .png로 변환
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'RandomForest-tree.png', '-Gdpi=600'])

# jupyter notebook에서 .png 직접 출력
from IPython.display import Image
Image(filename = 'RandomForest-tree.png')
```

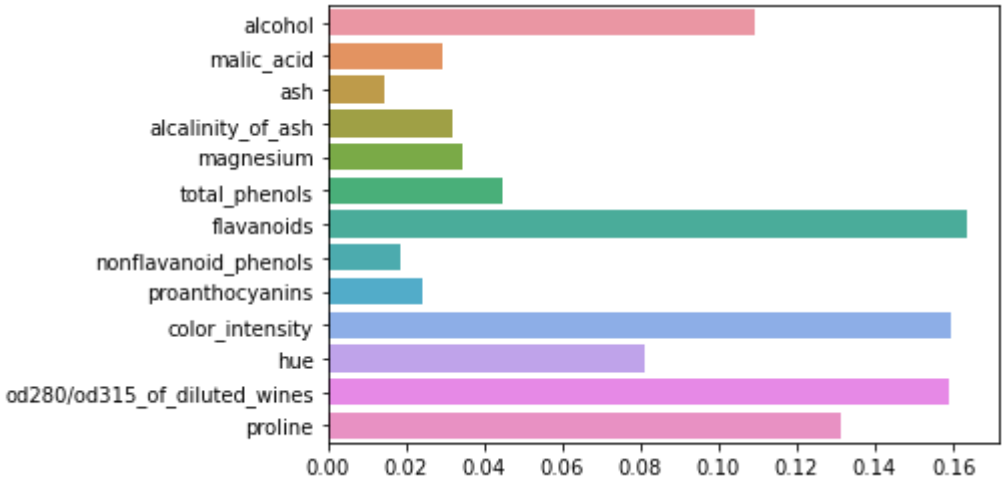
Out[16]:



In [17]:
```
import seaborn as sns
%matplotlib inline

# feature importance
print("Feature importances:Wn{0}".format(np.round(estimator.feature_importances_, 3)))

# feature별 importance 매핑
for name, value in zip(dataset.feature_names , estimator.feature_importances_):
    print('{0} : {1:.3f}'.format(name, value))

# feature importance를 column 별로 시각화 하기
sns.barplot(x=estimator.feature_importances_ , y=dataset.feature_names)
```

```
Feature importances:
[0.109 0.03  0.014 0.032 0.034 0.045 0.163 0.018 0.024 0.159 0.081 0.159
 0.131]
alcohol : 0.109
malic_acid : 0.030
ash : 0.014
alcalinity_of_ash : 0.032
magnesium : 0.034
total_phenols : 0.045
flavanoids : 0.163
nonflavanoid_phenols : 0.018
proanthocyanins : 0.024
color_intensity : 0.159
hue : 0.081
od280/od315_of_diluted_wines : 0.159
proline : 0.131
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24e46b34df0>
```



Decision Tree

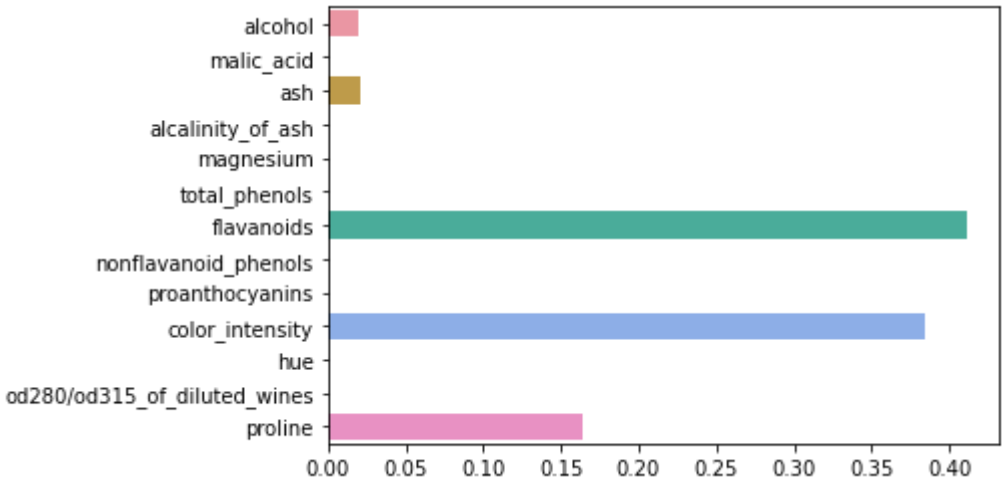
- 쉽고 유연하게 적용 가능한 알고리즘이며, Scaling 이나 정규화 등의 사전 가공의 영향이 매우 적다. 하지만, 예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하며, 이로 인한 과적합(Overfitting)이 발생해 반대로 예측 성능이 저하될 수 있다.
- 이런 단점이 앙상블 기법에서는 오히려 장점으로 작용하는데, 앙상블은 매우 많은 여러 개의 약한 학습기(Weak Learner)를 결합해 확률적 보완과 오류가 발생한 부분에 대한 가중치를 계속 업데이트하면서 예측 성능을 향상시키는데, 결정 트리가 좋은 약한 학습기가 되기 때문이다.

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

tree_clf = DecisionTreeClassifier(random_state=156)

dataset = load_wine()
X_train , X_test , y_train , y_test = train_test_split(X_features, y_label, test_size=0.2,
random_state=42)

tree_clf.fit(X_train , y_train)

from sklearn.tree import export_graphviz

# export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함.
export_graphviz(tree_clf, out_file="tree.dot", class_names=dataset.target_names , W
feature_names = dataset.feature_names, impurity=True, filled=True)
```

In [19]:
```
import seaborn as sns
import numpy as np
%matplotlib inline

print("Feature importances:Wn{0}".format(np.round(tree_clf.feature_importances_, 3)))

for name, value in zip(dataset.feature_names , tree_clf.feature_importances_):
    print('{0} : {1:.3f}'.format(name, value))

sns.barplot(x = tree_clf.feature_importances_ , y = dataset.feature_names)
```

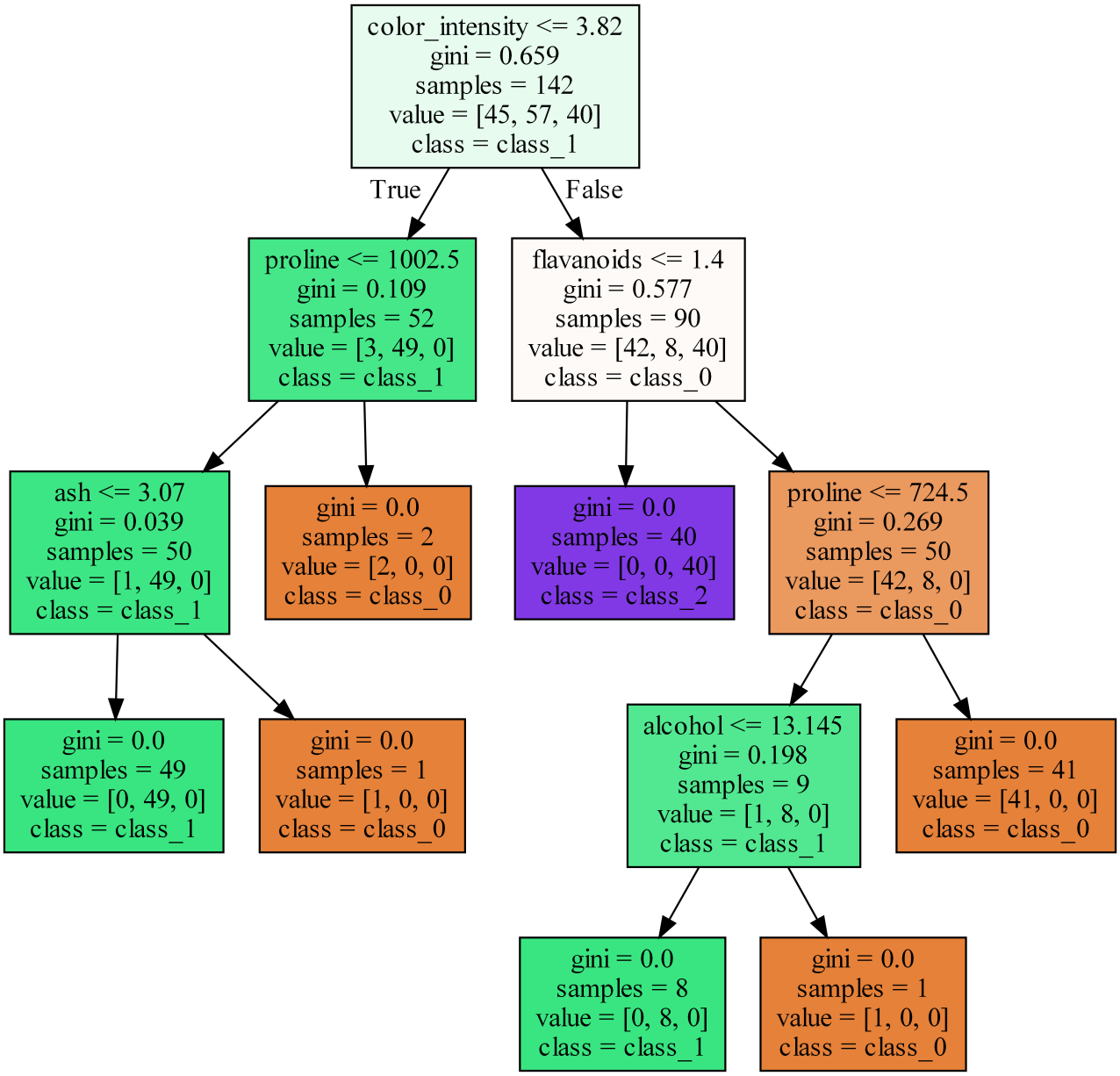
```
Feature importances:
[0.019 0.    0.021 0.    0.    0.    0.411 0.    0.    0.385 0.    0.
 0.164]
alcohol : 0.019
malic_acid : 0.000
ash : 0.021
alcalinity_of_ash : 0.000
magnesium : 0.000
total_phenols : 0.000
flavanoids : 0.411
nonflavanoid_phenols : 0.000
proanthocyanins : 0.000
color_intensity : 0.385
hue : 0.000
od280/od315_of_diluted_wines : 0.000
proline : 0.164
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24e47305850>
```



In [20]:
```
# .dot 파일로 export 해줍니다
export_graphviz(tree_clf, out_file='DecisionTree.dot',
                feature_names = dataset.feature_names,
                class_names = dataset.target_names,
                max_depth = 3, # 표현하고 싶은 최대 depth
                precision = 3, # 소수점 표기 자릿수
                filled = True, # class별 color 채우기
                rounded = True # 박스의 모양을 둥글게
               )

# 생성된 .dot 파일을 .png로 변환
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'decistion-tree.png', '-Gdpi=600'])

# jupyter notebook에서 .png 직접 출력
from IPython.display import Image
Image(filename = 'decistion-tree.png')
```

Out[20]:



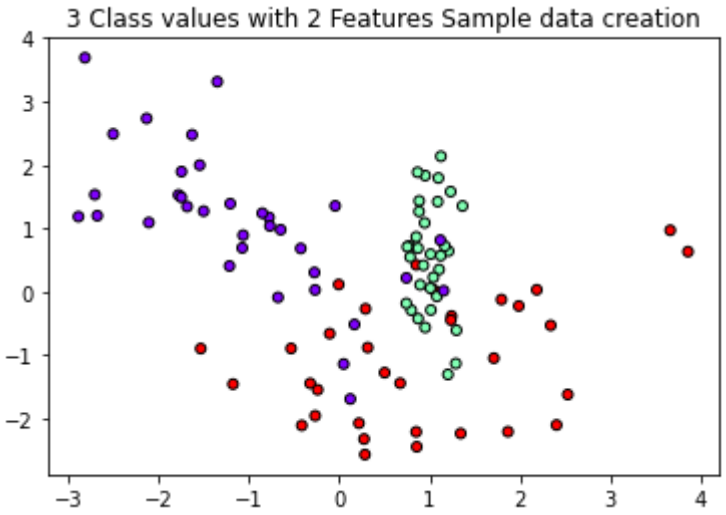
In [21]:
```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
%matplotlib inline
```

In [22]:
```
def get_dataset() :
    feature_names = dataset.feature_names
```

Sample Data를 통한 2차원 시각화.

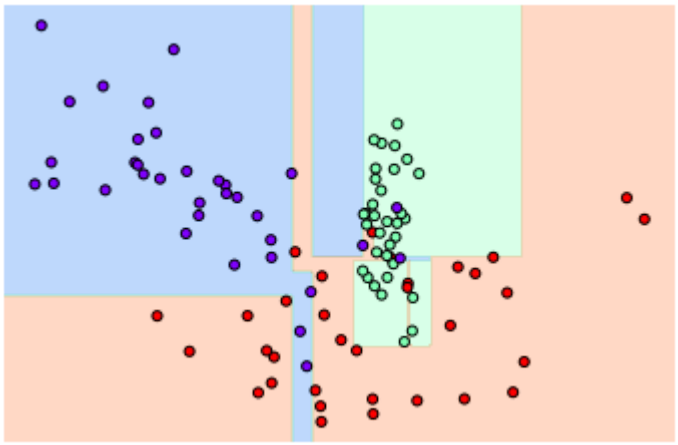
In [23]:
```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
%matplotlib inline

plt.title("3 Class values with 2 Features Sample data creation")

# 2차원 시각화를 위해서 feature는 2개, 결정값 클래스는 3가지 유형의 classification 샘플 데이터 생성.
X_features, y_labels = make_classification(n_features=2, n_redundant=0, n_informative=2,
                                           n_classes=3, n_clusters_per_class=1,random_state=0)

# plot 형태로 2개의 feature로 2차원 좌표 시각화, 각 클래스값은 다른 색깔로 표시됨.
plt.scatter(X_features[:, 0], X_features[:, 1], marker='o', c=y_labels, s=25, cmap='rainbow', edgecolor='k')
```

Out[23]: <matplotlib.collections.PathCollection at 0x24e4839ff70>



In [24]:
```
# Classifier의 Decision Boundary를 시각화 하는 함수
def visualize_boundary(model, X, y):
    fig,ax = plt.subplots()

    ax.scatter(X[:, 0], X[:, 1], c=y, s=25, cmap='rainbow', edgecolor='k',
               clim=(y.min(), y.max()), zorder=3)
    ax.axis('tight')
    ax.axis('off')
    xlim_start , xlim_end = ax.get_xlim()
    ylim_start , ylim_end = ax.get_ylim()

    model.fit(X, y)
    xx, yy = np.meshgrid(np.linspace(xlim_start,xlim_end, num=200),np.linspace(ylim_start,ylim_end, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels = np.arange(n_classes + 1) - 0.5,
                           cmap='rainbow', clim=(y.min(), y.max()),
                           zorder=1)
```

In [26]:
```
from sklearn.tree import DecisionTreeClassifier

# 특정한 트리 생성 제약없는 결정 트리의 Decsion Boundary 시각화.
dt_clf = DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```



앙상블 학습

여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

앙상블의 유형

- 일반적으로는 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)으로 구분할 수 있으며, 이외에 스태킹(Stacking) 등의 기법이 있다.
- 대표적인 배깅은 랜덤 포레스트(Random Forest) 알고리즘이 있으며, 부스팅은 에이다 부스팅, 그래디언트 부스팅, XGBoost, LightGBM 등이 있다. 정형 데이터의 분류나 회귀에서는 GBM 부스팅계열의 앙상블이 전반적으로 높은 예측 성능을 나타낸다.
- 넓은 의미로는 서로 다른 모델을 결합한 것을 앙상블로 지칭하기도 한다.

앙상블의 특징

- 단일 모델의 약점을 다수의 모델을 결합하여 보완
- 뛰어난 성능을 가진 모델들로만 구성하는 것보다 성능이 떨어지더라도 서로 다른 유형의 모델을섞는 것이 오히려 전체 성능에 도움이 될 수 있다.
- 랜덤 포레스트 및 뛰어난 부스팅 알고리즘은 모두 결정 트리 알고리즘을 기반 알고리즘으로 적용한다. (결정 트리의 단점인 Overfitting을 수십~수천 개의 많은 분류기를 결합해 보완하고 장점인직관적인 분류기준은 강화하는 방식으로 이용한다.)

In [ ]:
```
```