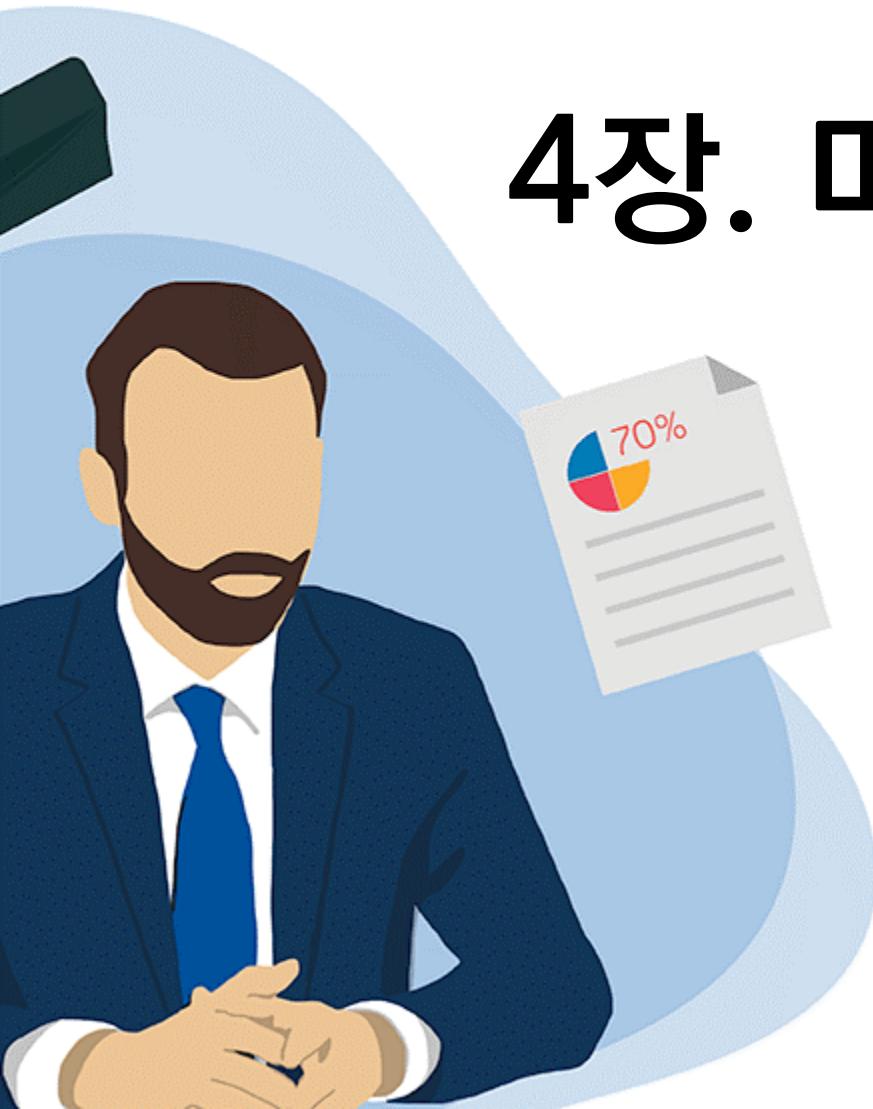


# 4장. 머신러닝 모형 최적화

- 
- 1절. 변수 선택과 차원 축소
  - 2절. 파라미터 탐색
  - 3절. 자료 불균형 처리
  - 4절. 최적 모형 탐색
  - 5절. 투표를 통한 최적의 양상을 모형 만들기

# Markdown cell

---

<font size="5" color="red"><b>ch4. 머신러닝 모형 최적화</b></font>  
# 1절. 변수 선택과 차원 축소

## ## 1-1 변수선택과 차원축소

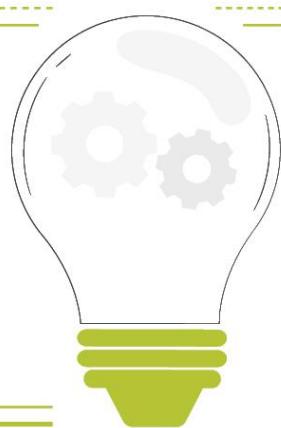
- 종속변수에 영향을 주는 변수들을 찾아 학습에 사용할 독립변수의 수를 줄임  
(어떻게 하면 score를 높일 수 있을지?)
- 과적합과 변수들 사이의 다중공선성(변수들간 강한 상관관계)을 줄일 수 있음
  - \* 회귀계수 해석이 어려워짐. 모델 예측력이 좋아도 해석력이 떨어짐(어떤 변수가 제일 큰 요인인지 잘), p값이나 유의성 검정이 왜곡될 수 있음
- 모형의 학습 시간을 줄일 수 있음
- 주성분분석, 상관분석, \*\*분류모형의 feature\_importance\_, 예측 모형의 coef\_\*\*
- SelectKBest : 가장 높은 score에 따라 K개의 특징을 선택

## ## 1-2 주성분분석(PCA, Principal Component Analysis)

- 주성분분석은 변수 선택 및 차원축소 방법(기존의 모든 변수를 조합하여 새로운 변수로 만들)으로 널리 사용
- 주성분 분석은 상관관계가 있는 변수들을 선형결합해서 \*\*분산이 극대화된 상관관계가 없는 새로운 변수(주성분)들로 축약\*\*하는 것
- 주성분 분석은 사실 선형대수학이라기보다는 선형대수학의 활용적인 측면이 강하며 영상인식, 통계 데이터분석(주성분 찾기), 데이터 압축, 노이즈제거 등 여러 분야에 사용
- 영상처리에서 많이 활용 : 여러개의 영상 중 대표 이미지를 찾을 때 활용



## 머신러닝을 이용한 데이터 분석



# 5장. 머신러닝 모형 최적화



## 1절. 변수 선택과 차원 축소

# 변수 선택과 차원 축소

## 1절. 변수 선택과 차원축소

종속변수에 영향을 주는 변수들을 찾아 학습에 사용할 독립변수의 수를 줄임

과적합과 변수들 사이의 다중공선성을 줄일 수 있음

모형의 학습 시간을 줄일 수 있음

주성분 분석, 상관 분석, 분류 모형의 `feature_importance_`, 예측 모형의 `coef_`

SelectKBest: 가장 높은 score에 따라 k개의 특징을 선택

# 주성분 분석(PCA, Principal Component Analysis)

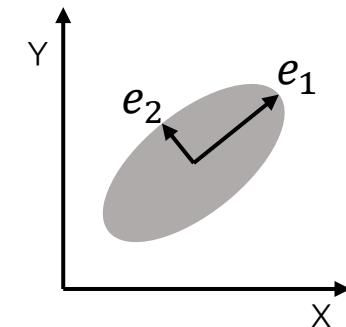
1절. 변수 선택과 차원축소

주성분 분석은 변수 선택 및 차원 축소 방법으로 널리 사용

주성분 분석은 상관관계가 있는 변수들을 선형결합(Linear combination)해서 분산이 극대화된 상관관계가 없는 새로운 변수(주성분)들로 축약하는 것

주성분 분석은 선형대수학이라기보다는 선형대수학의 활용적인 측면이 강하며 영상인식, 통계 데이터 분석(주성분 찾기), 데이터 압축, 노이즈 제거 등 여러 분야에 사용

$$\begin{aligned} c &= \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(x,y) & \text{cov}(y,y) \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{n} \sum (x_i - m_x)^2 & \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) \\ \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) & \frac{1}{n} \sum (y_i - m_y)^2 \end{pmatrix} \end{aligned}$$



# 주성분 분석(PCA, Principal Component Analysis)

## 1절. 변수 선택과 차원축소

```
sklearn.decomposition.PCA(n_components=None,  
                           copy=True, whiten=False,  
                           svd_solver='auto', tol=0.0,  
                           iterated_power='auto', random_state=None)
```

```
1 import seaborn as sns  
2 from sklearn.decomposition import PCA  
3 iris = sns.load_dataset("iris")  
4 iris_X, iris_y = iris.iloc[:, :-1], iris.species  
5  
6 pca = PCA(n_components=2)  
7 iris_pca = pca.fit_transform(iris_X)
```

# 주성분 분석

## 1절. 변수 선택과 차원축소

```
1 iris_pca[ :5, : ]
```

```
array([[-2.68412563,  0.31939725],  
      [-2.71414169, -0.17700123],  
      [-2.88899057, -0.14494943],  
      [-2.74534286, -0.31829898],  
      [-2.72871654,  0.32675451]])
```

```
1 pca.explained_variance_
```

각각의 주성분 벡터가 이루는 축에 투영(projection)한 결과의 분산

```
array([4.22824171, 0.24267075])
```

```
1 pca.components_
```

```
array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],  
      [ 0.65658877,  0.73016143, -0.17337266, -0.07548102]])
```

pca1 = 0.36138659\*x1 - 0.08452251\*x2 + 0.85667061\*x3 + 0.3582892\*x4

# 상관관계 확인

1절. 변수 선택과 차원축소

cmap의 종류 : <https://jrc-park.tistory.com/155>

<http://seaborn.pydata.org/generated/seaborn.heatmap.html#seaborn.heatmap>  
<http://seaborn.pydata.org/examples/many pairwise correlations.html>

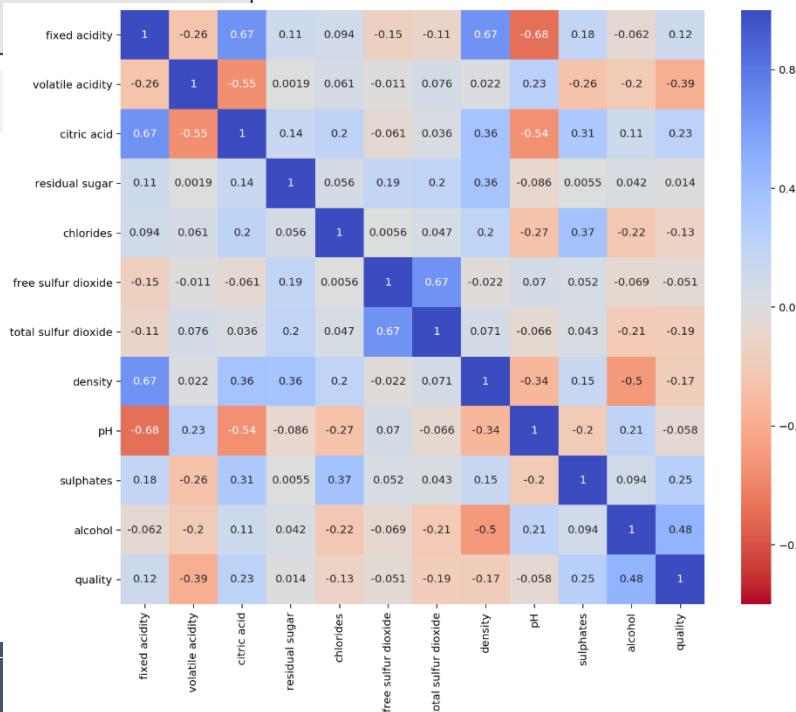
```
import pandas as pd  
redwine = pd.read_csv('winequality-red.csv', delimiter=';')  
redwine.head()
```

```
X = redwine.iloc[:, :-1]  
y = redwine.iloc[:, -1]  
from sklearn.model_selection import train_test_split  
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3)  
train_X.shape, test_X.shape, train_y.shape, test_y.shape  
((1119, 11), (480, 11), (1119,), (480,))
```

```
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
plt.figure(figsize=(12,10))  
sns.heatmap(redwine.corr(), annot=True,  
            vmin=-1, vmax=1, cmap="coolwarm_r")  
plt.show()
```

각 변수들끼리의 상관관계를 확인하고 시각화 해서 종속변수와 상관관계가 높은 변수들만 선택



# 분류모형의 Feature Importance

1절. 변수 선택과 차원축소

- 분류 모형의 `feature_importances_` 속성은 각 독립변수들이 종속변수에 영향을 주는 정도를 저장

```
from sklearn.ensemble import RandomForestClassifier  
rf_model = RandomForestClassifier(n_estimators=10, random_state=10)  
rf_model.fit(X_train, y_train)
```

```
features = pd.DataFrame(data=np.c_[X.columns.values,  
                                     rf_model.feature_importances_],  
                         columns=["feature", "importance"])
```

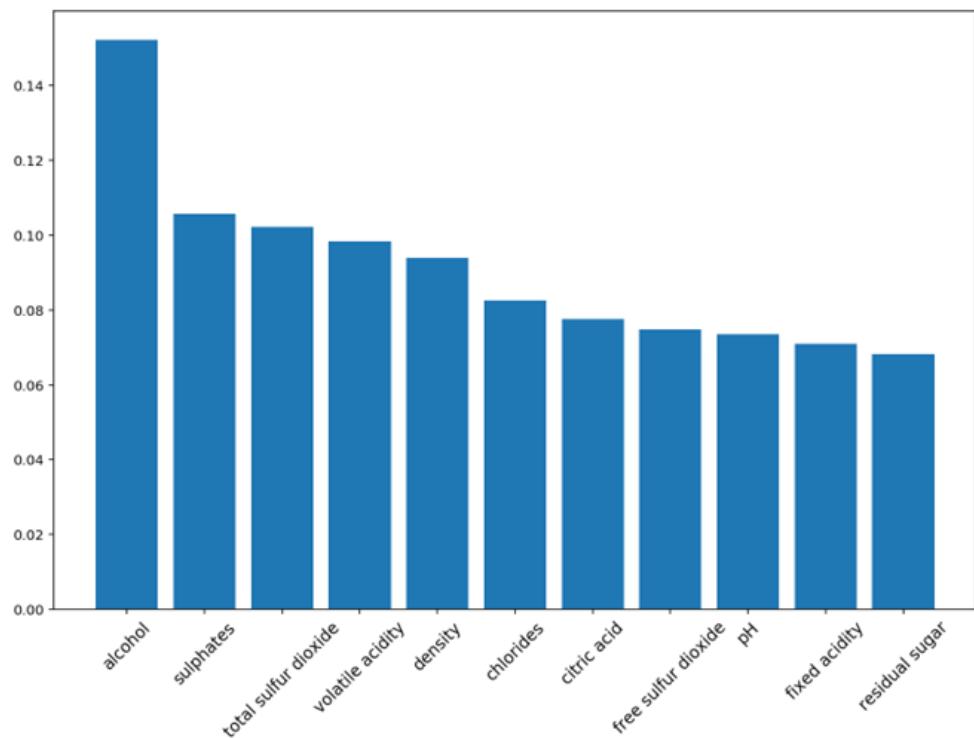
```
features.sort_values(by="importance", ascending=False, inplace=True)  
features.reset_index(drop=True, inplace=True)  
features
```

	feature	importance
0	alcohol	0.152202
1	sulphates	0.105698
2	total sulfur dioxide	0.102227
3	volatile acidity	0.0982629
4	density	0.0939209
5	chlorides	0.0825913
6	citric acid	0.077593
7	free sulfur dioxide	0.0749012
8	pH	0.0735724
9	fixed acidity	0.0708851
10	residual sugar	0.0681466

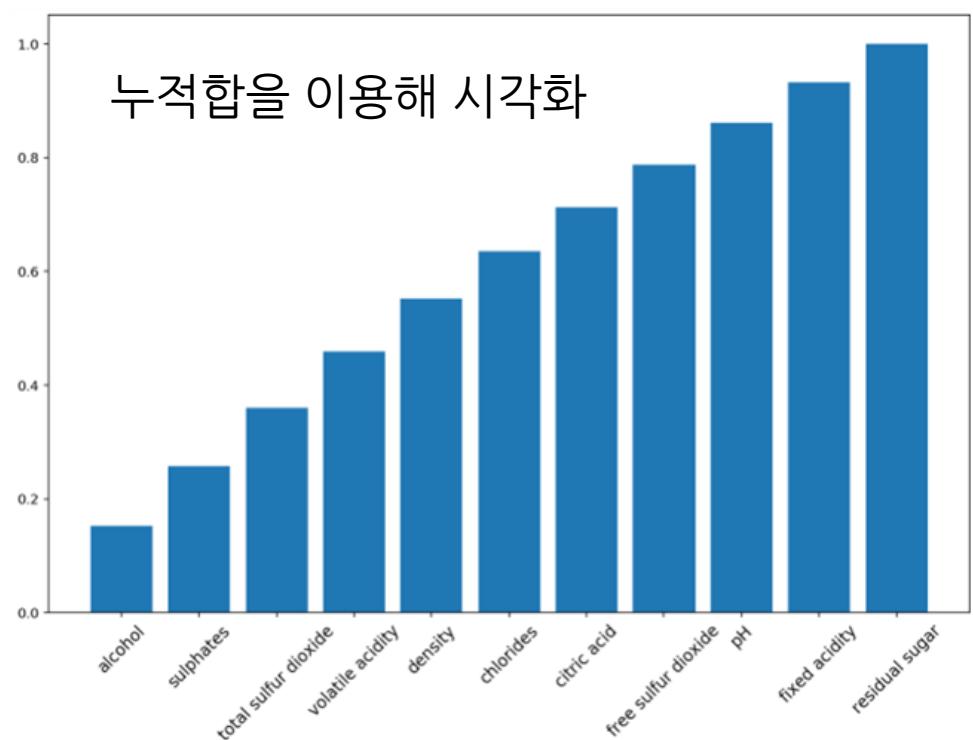
# feature\_importances\_를 이용한 변수 중요도 시각화

## 1절. 변수 선택과 차원축소

```
plt.figure(figsize=(12, 8))
plt.bar(features.feature, features.importance)
plt.xticks(features.feature, fontsize=12, rotation=45)
plt.show()
```



```
y_stack = np.cumsum(features.importance, axis=0)
plt.figure(figsize=(12, 8))
plt.bar(features.feature, y_stack)
plt.xticks(features.feature, fontsize=12, rotation=45)
plt.show()
```

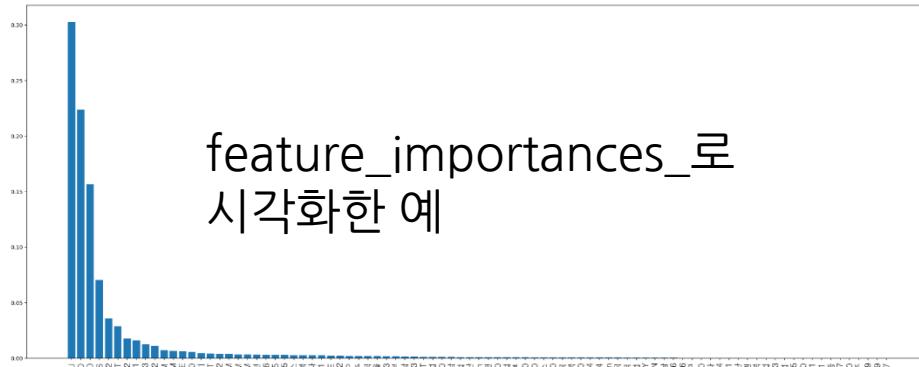


**변수를 선택하기 위해서…(부정탐지 사례에서)**

1절. 변수 선택과 차원축소

```
imp df.loc[imp_df.cum_sum <= 0.9]
```

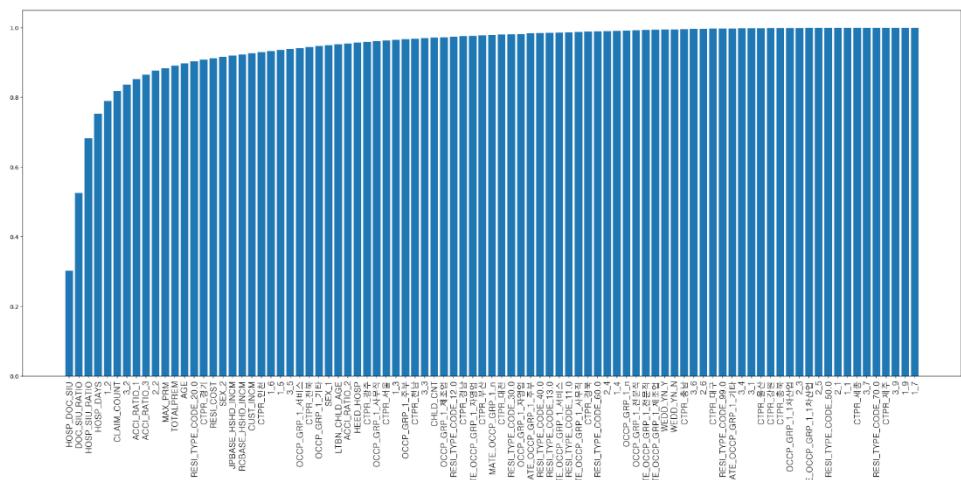
	variable	importance	cum_sum
57	HOSP_DOC_SIU	0.30264	0.30264
55	DOC_SIU_RATIO	0.223687	0.526327
56	HOSP_SIU_RATIO	0.156792	0.683118
52	HOSP_DAYS	0.0705903	0.753709
62	1_2	0.0359992	0.789708
54	CLAIM_COUNT	0.0287546	0.818462
76	3_2	0.0178182	0.836281
58	ACCI_RATIO_1	0.0162699	0.85255
60	ACCI_RATIO_3	0.0128517	0.865402
70	2_2	0.0112797	0.876682
85	MAX_PRM	0.0075201	0.884202
84	TOTALPRM	0.0068085	0.891011
0	AGE	0.00656627	0.897577



## feature\_importances\_로 시각화한 예

```
imp_df.loc[imp_df.cum_sum <= 0.98]
```

	variable	importance	cum_sum	
57	HOSP_DOC_SIU	0.30264	0.30264	79
55	DOC_SIU_RATIO	0.223687	0.526327	36 OCCP_GRP_1_서비스
56	HOSP_SIU_RATIO	0.156792	0.683118	28 CTPR_전부
52	HOSP_DAYS	0.0705903	0.753709	34 OCCP_GRP_1_?부
62	1_2	0.0359992	0.789708	3 SEX_1
54	CLAIM_COUNT	0.0287546	0.818462	2 LTBN_CHLD_AGE
76	3_2	0.0178182	0.836281	59 ACCL_RATIO_2
58	ACCL_RATIO_1	0.0162699	0.85255	53 HEED_HOSP
60	ACCL_RATIO_3	0.0128517	0.865402	19 CTPR_광주
70	2_2	0.0112797	0.876682	35 OCCP_GRP_1_사무직
85	MAX_PRM	0.0075201	0.884202	23 CTPR_서울
84	TOTALPREM	0.00668085	0.891011	63 1_3
0	AGE	0.00656627	0.897577	40 OCCP_GRP_1_주부
8	RESI_TYPE_CODE_20_0	0.00583556	0.90326	27 CTPR_경기
16	CTPR_경기	0.00473011	0.907991	77 3_3
83	RESI_COST	0.00413631	0.912127	1 CHLD_CNT
4	SEX_2	0.00404582	0.916173	39 OCCP_GRP_1_제조업
88	JPBASE_HSHD_INCM	0.00391716	0.92009	6 RESI_TYPE_CODE_12_0
87	RCBASE_HSHD_INCM	0.00343641	0.923526	17 CTPR_경남
86	CUST_INCM	0.0034145	0.926958	48 MATE_OCCP_GRP_1_자영업
26	CTPR_인천	0.00321198	0.93017	22 CTPR_부산
66	1_6	0.0030781	0.933248	44 MATE_OCCP_GRP_1_
65	1_5	0.00295263	0.9362	



## 누적합을 시각화

# RFE(Recursive Feature Elimination) 방식

1절. 변수 선택과 차원축소

- RandomForestClassifier 모형을 이용한 예
- 5개 특징이 남을 때까지 변수를 제거 함

```
rfe_model = RFE(RandomForestClassifier(n_estimators=10, random_state=10),  
                  n_features_to_select=5)  
rfe_model.fit(train_X, train_y)
```

```
import numpy as np  
import pandas as pd  
features_rfe = pd.DataFrame(data=np.c_[X.columns.values,  
                                         rfe_model.get_support()],  
                             columns=["feature", "selected"])
```

	feature	selected
1	volatile acidity	True
6	total sulfur dioxide	True
7	density	True
9	sulphates	True
10	alcohol	True
0	fixed acidity	False
2	citric acid	False
3	residual sugar	False
4	chlorides	False
5	free sulfur dioxide	False
8	pH	False

# SelectKBest

## 1절. 변수 선택과 차원축소

- 가장 높은 score에 따라 k개의 특징을 선택
- `sklearn.feature_selection.SelectKBest(score_func=<function f_classif>, k=10)`

인자	타입	기본값	설명
score_func	callable	f_classif	<p>함수는 두 개의 배열 X와 y를 취하고 한 쌍의 배열 (scores, pvalues) 또는 scores가 있는 단일 배열을 반환. 기본값은 f_classif. 기본 함수는 분류 작업에서만 동작.</p> <ul style="list-style-type: none"><li>• f_classif : 분류를 위한 label/feature 사이의 ANOVA F-value</li><li>• mutual_info_classif : 이산(discrete) 변수를 위한 상호정보(mutual information)</li><li>• chi2 : 분류를 위한 비 음수의 카이제곱 통계</li><li>• f_regression : 회귀를 위한 label/feature 사이의 F-value</li><li>• mutual_info_regression : 연속형 타겟을 위한 상호정보</li><li>• SelectPercentile : 가장 높은 점수의 백분위 수를 기반으로 특징을 선택</li><li>• SelectFpr : 오 험지율(false positive rate)을 기반으로 특징을 선택</li><li>• SelectFdr : 추정된 오류 발견율을 기반으로 특징을 선택</li><li>• SelectFwe : 제1종오류가 발생할 가능성(FWER; family-wise error rate)을 기반으로 특징을 선택</li><li>• GenericUnivariateSelect : 구성 가능한 모드가 있는 단변량 특징 선택자</li></ul>
k	정수 또는 "all"	10	선택할 상위 특징 개수를 지정. "all"로 지정하면 파라미터 검색을 위해 사용할 선택 항목을 무시함

# SelectKBest 예

## 1절. 변수 선택과 차원축소

- iris 데이터에서 score에 가장 큰 영향을 주는 변수 1개를 선택

```
1 from sklearn.datasets import load_iris  
2 from sklearn.feature_selection import SelectKBest, chi2  
3 X, y = load_iris(return_X_y=True)  
4 X.shape
```

(150, 4)

```
1 X_new = SelectKBest(chi2, k=1).fit_transform(X, y)
2 X_new.shape
```

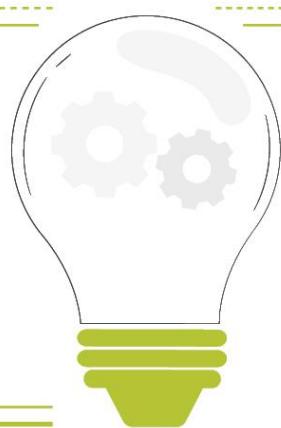
(150, 1)

1 | X new

```
array([[1.4],  
       [1.4],  
       [1.3],  
       [1.5],
```



## 머신러닝을 이용한 데이터 분석



### 5장. 머신러닝 모형 최적화



#### 2절. 파라미터 탐색

# 파라미터 탐색과 모형 최적화

## 2절. 파라미터 탐색

- 머신러닝 모형이 완성된 후에는 파라미터 탐색을 통해 모형을 최적화하고 예측 성능을 향상시켜야 함
- Scikit-learn 패키지는 하이퍼 파라미터 튜닝을 통해 모형의 최적화를 위한 여러 도구들을 제공
- validation\_curve() : 단일 하이퍼 파라미터 최적화 함수
- GridSearchCV : 그리드를 사용한 복수 하이퍼 파라미터 최적화 클래스

# validation\_curve

## 2절. 파라미터 탐색

- 최적화할 파라미터의 이름과 범위, 그리고 성능 측정 기준을 각각 param\_name, param\_range, scoring 인수로 받아 파라미터 범위의 모든 경우에 대해 성능을 계산

```
sklearn.model_selection.validation_curve(estimator, X, y,  
                                        param_name, param_range, groups=None, cv='warn',  
                                        scoring=None, n_jobs=None, pre_dispatch='all', verbose=0,  
                                        error_score='raise-deprecating')
```

# validation\_curve를 이용한 예

## 2절. 파라미터 탐색

```
1 # 데일리  
2 from sklearn.datasets import load_digits  
3 digits = load_digits()  
4 X, y = digits.data, digits.target
```

```
1 from sklearn.svm import SVC  
2 model = SVC().fit(X, y)  
3 model
```

SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0, decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rbf', max\_iter=-1, probability=False, random\_state=None, shrinking=True, tol=0.001, verbose=False)

```
1 import numpy as np  
2 param_range = np.logspace(-6, -1, 10)
```

gamma 값의 범위( $\frac{1}{10^6} \sim \frac{1}{10^1}$ )

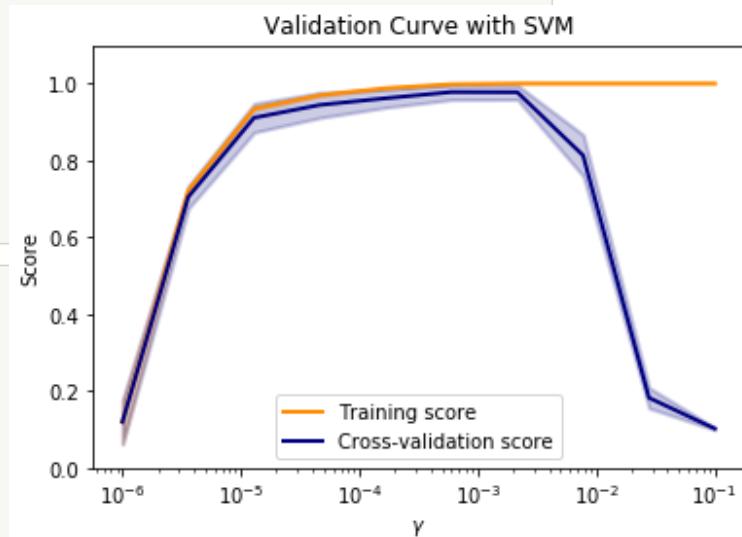
```
1 %%time  
2 from sklearn.svm import SVC  
3 from sklearn.model_selection import validation_curve  
4 train_scores, test_scores = validation_curve(  
    SVC(), X, y, param_name="gamma", param_range=param_range,  
    cv=10, scoring="accuracy", n_jobs=1)
```

# validation\_curve를 이용한 예

## 2절. 파라미터 탐색

```
1 import numpy as np  
2 train_scores_mean = np.mean(train_scores, axis=1)  
3 train_scores_std = np.std(train_scores, axis=1)  
4 test_scores_mean = np.mean(test_scores, axis=1)  
5 test_scores_std = np.std(test_scores, axis=1)
```

```
1 %matplotlib inline  
2 import matplotlib.pyplot as plt  
3  
4 plt.title("Validation Curve with SVM")  
5 plt.xlabel("$\gamma$")  
6 plt.ylabel("Score")  
7 plt.ylim(0.0, 1.1)  
8 plt.semilogx(param_range, train_scores_mean, label="Training score",  
9             color="darkorange", lw=2)  
10 plt.fill_between(param_range, train_scores_mean - train_scores_std,  
11                  train_scores_mean + train_scores_std, alpha=0.2,  
12                  color="darkorange", lw=2)  
13 plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",  
14              color="navy", lw=2)  
15 plt.fill_between(param_range, test_scores_mean - test_scores_std,  
16                  test_scores_mean + test_scores_std, alpha=0.2, color="navy", lw=2)  
17 plt.legend(loc="best")  
18 plt.show()
```



# GridSearchCV

## 2절. 파라미터 탐색

- validation\_curve 함수와 달리 모형 래퍼(Wrapper) 성격의 클래스임
- 중요 멤버는 fit(), predict()
- fit(), score(), predict(), predict\_proba(), decision\_function(), transform(), inverse\_transform() 메소드를 구현
- estimator의 파라미터들은 이들 메소드에 적용되어 파라미터 그리드를 통해 교차 검증된 그리드 검색에 의해 최적화

```
sklearn.model_selection.GridSearchCV(estimator, param_grid,  
    scoring=None, fit_params=None, n_jobs=None, iid='warn',  
    refit=True, cv='warn', verbose=0, pre_dispatch='2*n_jobs',  
    error_score='raise-deprecating', return_train_score='warn')
```

# GridSearchCV를 이용한 예

## 2절. 파라미터 탐색

```
1 from sklearn.pipeline import Pipeline
2 from sklearn.model_selection import GridSearchCV
3 from sklearn.svm import SVC
4 from sklearn.feature_selection import SelectKBest
5 import pandas as pd
6 redwine = pd.read_csv("winequality-red.csv", sep=";")
7 redwine_X, redwine_y = redwine.iloc[:, :-1], redwine.iloc[:, -1]
```

실행하는데 수분에서 수십분이 소요될 수 있음

```
1 selection = SelectKBest(k=1)
2 svm = SVC(kernel="linear")
3 pipeline = Pipeline([('univ_select', selection), ('svm', svm)])
4 param_grid = dict(univ_select_k=[4, 5, 6, 7, 8, 9, 10, 11], svm_C=[0.1, 1, 10])
5 grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=2, verbose=10)
6 grid_search.fit(redwine_X, redwine_y)
```

Fitting 2 folds for each of 24 candidates, totalling 48 fits

```
[CV] svm_C=0.1, univ_select_k=4 .....
[CV] ..... svm_C=0.1, univ_select_k=4, score=0.516, total= 0.0s
[CV] svm_C=0.1, univ_select_k=4 .....
[CV] ..... svm_C=0.1, univ_select_k=4, score=0.584, total= 0.1s
[CV] svm_C=0.1, univ_select_k=5 .....
[CV] ..... svm_C=0.1, univ_select_k=5, score=0.516, total= 0.0s
[CV] svm_C=0.1, univ_select_k=5 .....
```

# GridSearchCV를 이용한 예

2절. 파라미터 탐색

**best\_estimator\_** : 가장 높은 점수를 낸 파라미터의 모형, **params\_** : 최적 파라미터 정보

```
1 print(grid_search.best_estimator_)
```

```
Pipeline(memory=None,
         steps=[('univ_select',
                  SelectKBest(k=9,
                               score_func=<function f_classif at 0x000002339E555E58>)),
                ('svm',
                 SVC(C=1, break_ties=False, cache_size=200, class_weight=None,
                      coef0=0.0, decision_function_shape='ovr', degree=3,
                      gamma='scale', kernel='linear', max_iter=-1,
                      probability=False, random_state=None, shrinking=True,
                      tol=0.001, verbose=False))],
         verbose=False)
```

```
1 grid_search.best_params_
```

```
{'svm__C': 1, 'univ_select__k': 9}
```

# 병렬처리

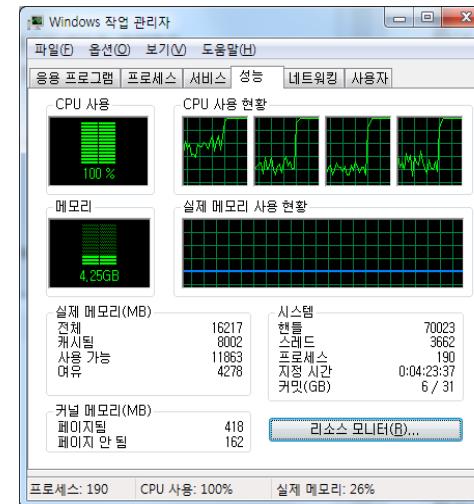
## 2절. 파라미터 탐색

사이킷런의 일부 클래스의 n\_jobs 파라미터는 연산에 사용할 코어의 수를 지정할 수 있음

```
1 %%time
2 from sklearn.svm import SVC
3 from sklearn.model_selection import validation_curve
4 train_scores, test_scores = validation_curve(
5     SVC(), X, y, param_name="gamma", param_range=param_range,
6     cv=10, scoring="accuracy", n_jobs=-1)
```



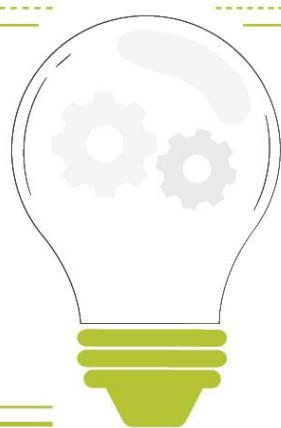
n<sub>jobs</sub>=1



n<sub>jobs</sub>=-1



## **머신러닝을 이용한 데이터 분석**



### **5장. 머신러닝 모형 최적화**

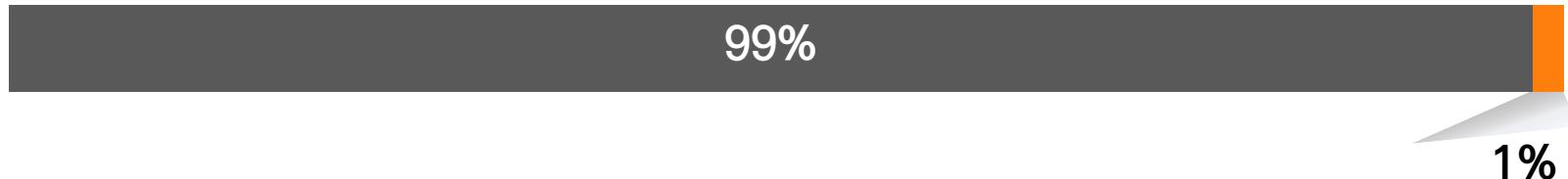


#### **3절. 자료 불균형 처리**

# 불균형 데이터

## 3절. 자료 불균형 처리

- 데이터셋이 정상비정상=99:1로 매우 불균형



- 불균형 데이터의 문제점

[case 1] 학습용-검증용 세트 분리 시, 학습용 세트에 비정상 데이터가 포함되지 않을 가능성 존재



→ 그 결과, 단일클래스로 학습되어 비정상 거래를 감지할 수 없음

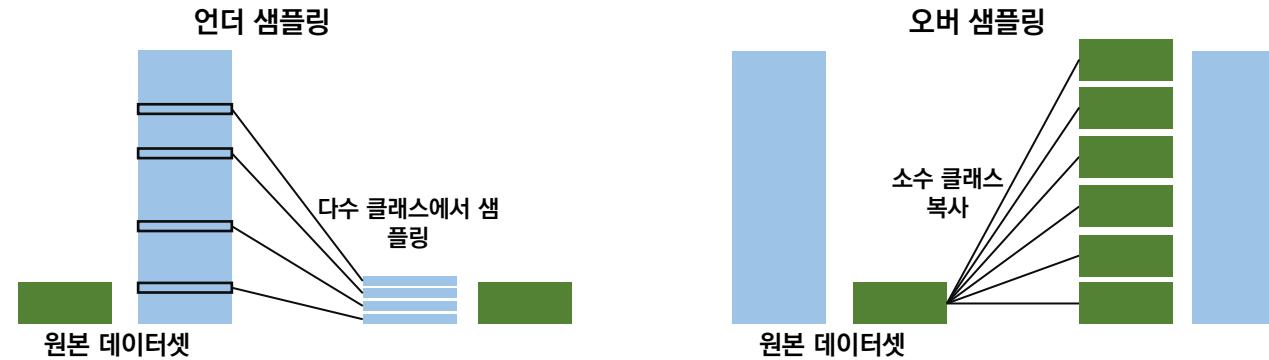
[case 2] 모델 학습 시, 비정상데이터의 절대적인 양이 적어 제대로 된 학습이 일어나지 않음



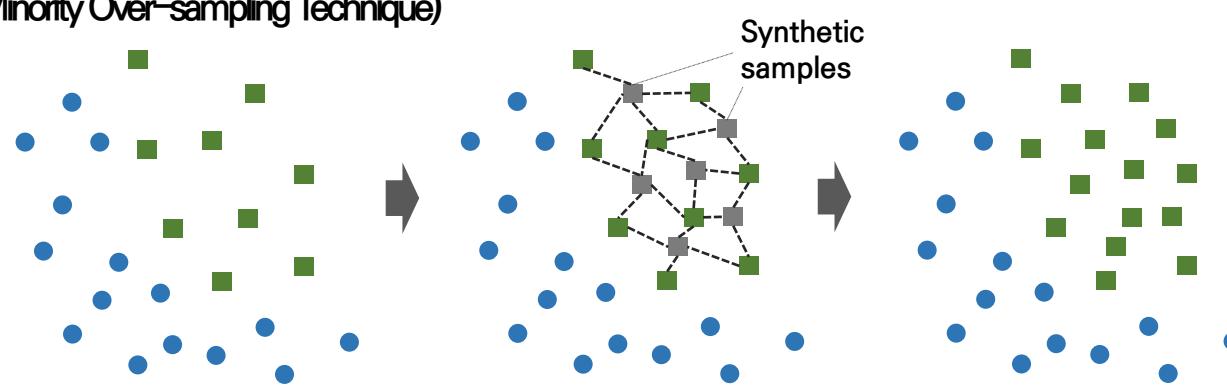
# 언더 샘플링과 오버 샘플링

## 3절. 자료 불균형 처리

### ✓ 단순 언더/오버 샘플링



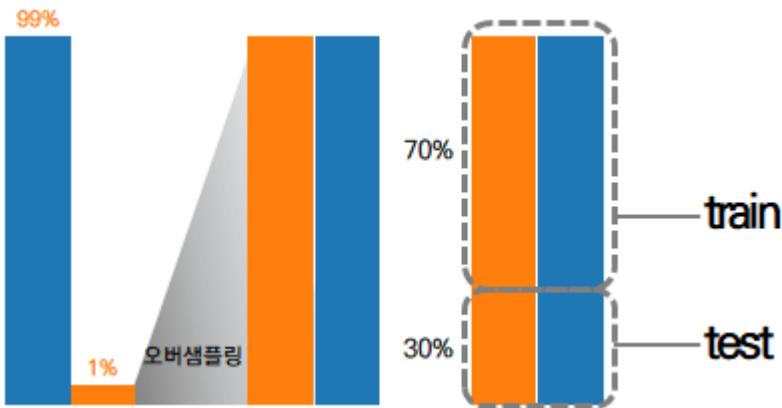
### ✓ SMOTE(Synthetic Minority Over-sampling Technique)



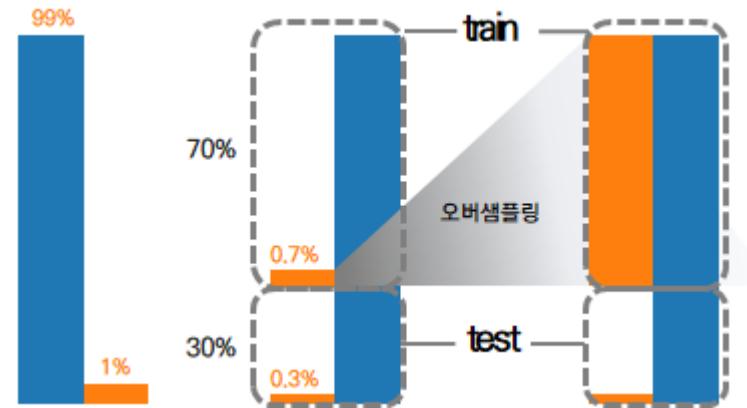
# 계층적 샘플링후 오버샘플링

## 3절. 자료 불균형 처리

전체를 오버샘플링 하고 train-test set 분리



계층적 샘플링 후 train 데이터만 오버샘플링



- test set에도 오버샘플링이 들어가 정확한 판단이 되지 않음
- 현실을 반영하지 못함

- 계층적 샘플링 사용:  
기존 클래스비율과 동일하게 train-test셋을 분리
- 유사값을 생성해 Class(1)에 대한 학습을 강화하고  
test에서는 원래 표본으로 예측해 정확도 판단

# SMOTE를 이용한 오버샘플링

3절. 자료 불균형 처리

pip install imbalanced-learn==0.10.1 명령으로 패키지를 설치해야 함

```
imblearn.over_sampling.SMOTE(sampling_strategy='auto',
                               random_state=None, k_neighbors=5,
                               n_jobs=1, ratio=None)
```

```
1 from imblearn.over_sampling import SMOTE
2 sm = SMOTE()
3 X_resampled, y_resampled = sm.fit_sample(X, y)
4
5 X_resampled.shape, y_resampled.shape
((19720, 10), (19720,))
```

# 가중치 제어

3절. 자료 불균형 처리

가중치 제어(Weight Control)는 데이터가 불균형(imbalanced)일 경우에 모델 및 데이터에 따라 가중치(weight)를 부여하는 방법

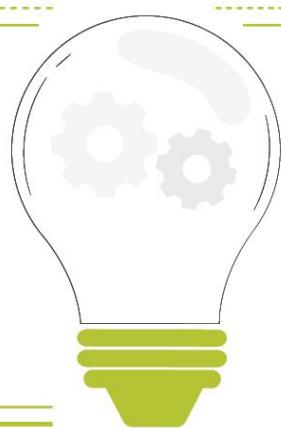
Scikit-learn의 예측모형에 class\_weight 매개변수를 이용해 설정

```
1 from sklearn.ensemble import RandomForestClassifier  
2 rf_model = RandomForestClassifier(n_estimators=100, max_features=2,  
3                                     class_weight={0:1, 1:1.4},  
4                                     random_state=42)  
5 rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(class_weight={0: 1, 1: 1.4}, max_features=2,  
random_state=42)
```



# 머신러닝을 이용한 데이터 분석



## 5장. 머신러닝 모형 최적화



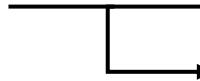
### 4절. 양상블 모형

# 모델 양상블이란?

4.1 부트스트래핑과 0.632규칙

Combine different classifiers into a meta-classifier that has a better generalization performance than each individual classifier alone

여러가지 분류기를 메타 분류기로 결합하여  
개별 분류기 하나보다 더 나을 일반화 성능을 갖게 하는 것



주로 의사결정나무 사용

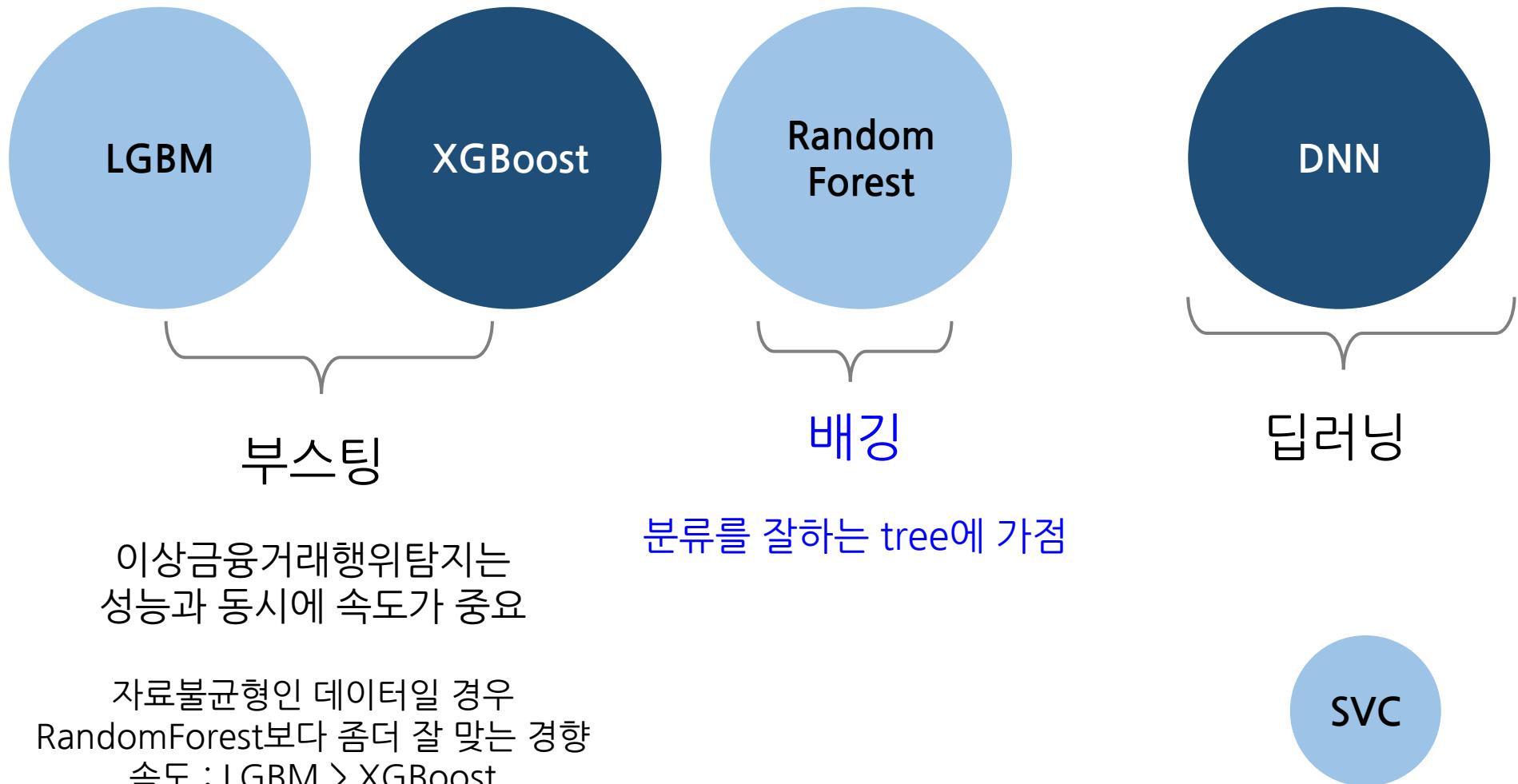
# 앙상블 학습이란?

4.1 부트스트래핑과 0.632규칙

- 목적 : 여러 분류기(분류모델)를 하나의 메타 분류기(통합분류모델)로 연결하여 개별분류기보다 더 좋은 일반화 성능을 달성
- 방법
  - 여러 분류 알고리즘 사용 : 다수결 투표(Voting)
  - 하나의 분류 알고리즘 이용 : 배깅(bagging), 부스팅(Boosting)
- 종류
  - 투표(Majority Voting) : 동일한 훈련세트
  - 배깅(Bagging) : 훈련샘플에서 알고리즘마다 별도의 훈련세트 추출(복원추출, bootstrapping)해서 모델 구축 후 투표하면서 학습
    - ex. Random Forest : 의사결정나무 알고리즘 이용
  - 부스팅(Boosting) : 샘플을 뽑을 때 잘못된 분류 data 50%를 가중치 적용

# 주요 알고리즘

최적 모형 탐색



# 앙상블

4.1 부트스트래핑과 0.632규칙

- 어떤 데이터의 값을 예측한다고 할 때, **하나의 모델을 활용하지만 여러 개의 모델을 조화롭게 학습시켜 그 모델들의 예측 결과들을 이용한다**면 더 정확한 예측값을 구할 수 있음
- **앙상블 학습은 여러 개의 결정 트리(Decision Tree)를 결합하여 하나의 결정 트리보다 더 좋은 성능을 내는 머신러닝 기법**
- **앙상블 학습의 핵심은 여러 개의 약 분류기(Weak Classifier)를 결합하여 강 분류기(Strong Classifier)를 만들어 모델의 정확성을 향상시킴**
  - **약 분류기를 병렬적으로 사용하면 배깅**
  - **Variance(데이터 밀집도)를 감소시키고 싶다면 배깅**
  - **약 분류기를 순차적으로 사용하면 부스팅**
  - **Bias(중심과의 거리)를 감소시키고 싶다면 부스팅**

# 부트스트래핑

## 4.1 부트스트래핑과 0.632규칙

부트스트래핑은 n개의 관찰로 구성된 모집단을 대체하여 무작위 샘플링을 수행하는 프로세스



복원추출을 허락했으므로 중복되는 데이터가 존재할 수 있음

복원추출의 예를 들어 모집단이 (2,3,4,5,6)이고 대체 가능한 크기로 크기가 4인 두 개의 랜덤 샘플링 했을 경우 샘플 1은 (2,3,3,6)이고 샘플 2는 (4,4,6,2)가 되어 샘플 내에 중복된 데이터가 존재할 수 있음

# 부트스트래핑의 0.632 규칙(.632+rule)

## 4.1 부트스트래핑과 0.632규칙

훈련 오차는…

$$TrainingError = \frac{1}{n} \sum L(y_i, \hat{y}_i)$$

교차 검증은 표본 오류의 예상 출력을 추정하는 방법

$$Error = E[L(y_i, f(x_i))]$$

K-폴드 교차 검증의 경우라면…

$$Error of CV = \sum L(y_i, f(x_i))$$

Training 데이터는  $X = (x_1, x_2, \dots, x_n)$ 이고, 각  $z_i$ 가  $n$ 개의 샘플 집합인 이 집합  $(z_1, \dots, z_b)$ 에서 부트스트랩 샘플링을 하면, 샘플 외 오류(OOSE, out-of-sample error)율은

$$OOSE = L(y_i, f_b(x_i))$$

$$Error = 0.368 * TrainingError + 0.632 * OOSE$$

OOSE : 앞서 봄한 63.2%가 아닌 원 데이터의 남은 37.8%의 데이터에 결정적인 정보가 존재할 경우 이를 반영할 수 없다는 에러

출처 : Efron, B. and R. Tibshirani (1997), "Improvements on Cross-Validation: The .632+ Bootstrap Method,"  
*Journal of the American Statistical Association* Vol. 92, No. 438. (Jun), pp. 548–560 <https://www.jstor.org/stable/2965703>

# 0.368은 어디에서?

## 4.1 부트스트래핑과 0.632규칙

$$\lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right)^n = e^{-1} \approx 0.368$$

$n=6$ 일 경우  $1/3$ ,  $n=11$ 일 경우  $0.35$ ,  $n=99$ 일 경우  $0.366$ ,  
 $N=10000$  이상일 경우  $0.368$ 에 수렴

$0.368$  : 어떤 물건이 선택되지 않을 확률  
 $1-0.368 = 0.632$  : 선택한 항목의 확률

```
1 import numpy as np
2
3 N = 1000000 #
4 bootstrap = np.random.choice(N, N, replace=True)
5 np.round(len(set(bootstrap))/N, 3)
```

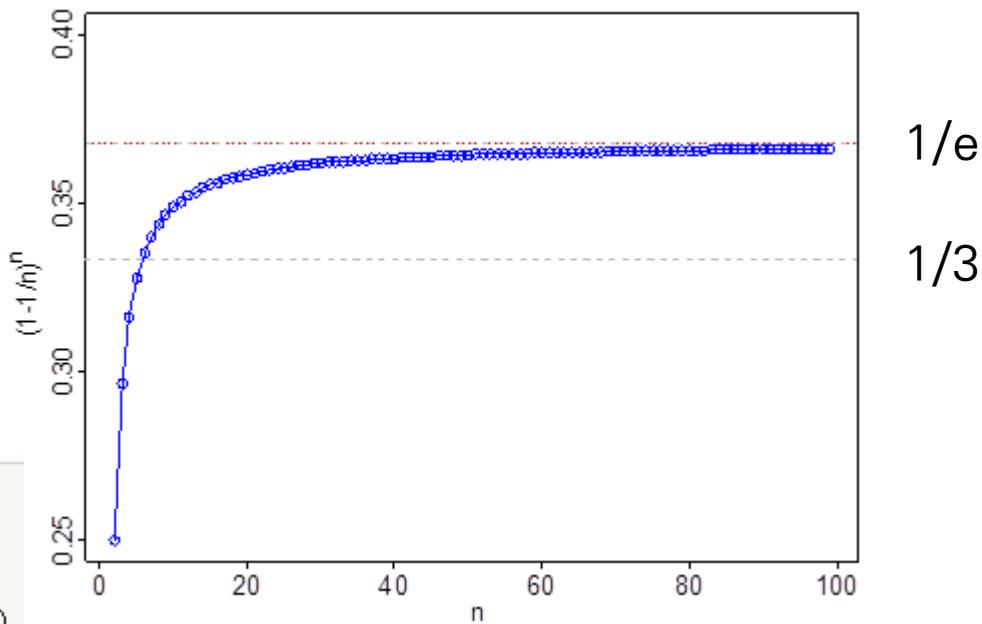
executed in 307ms, finished 16:55:33 2021-06-24

0.632

```
1 np.exp(-1), 1-np.exp(-1)
```

executed in 13ms, finished 17:13:03 2021-06-24

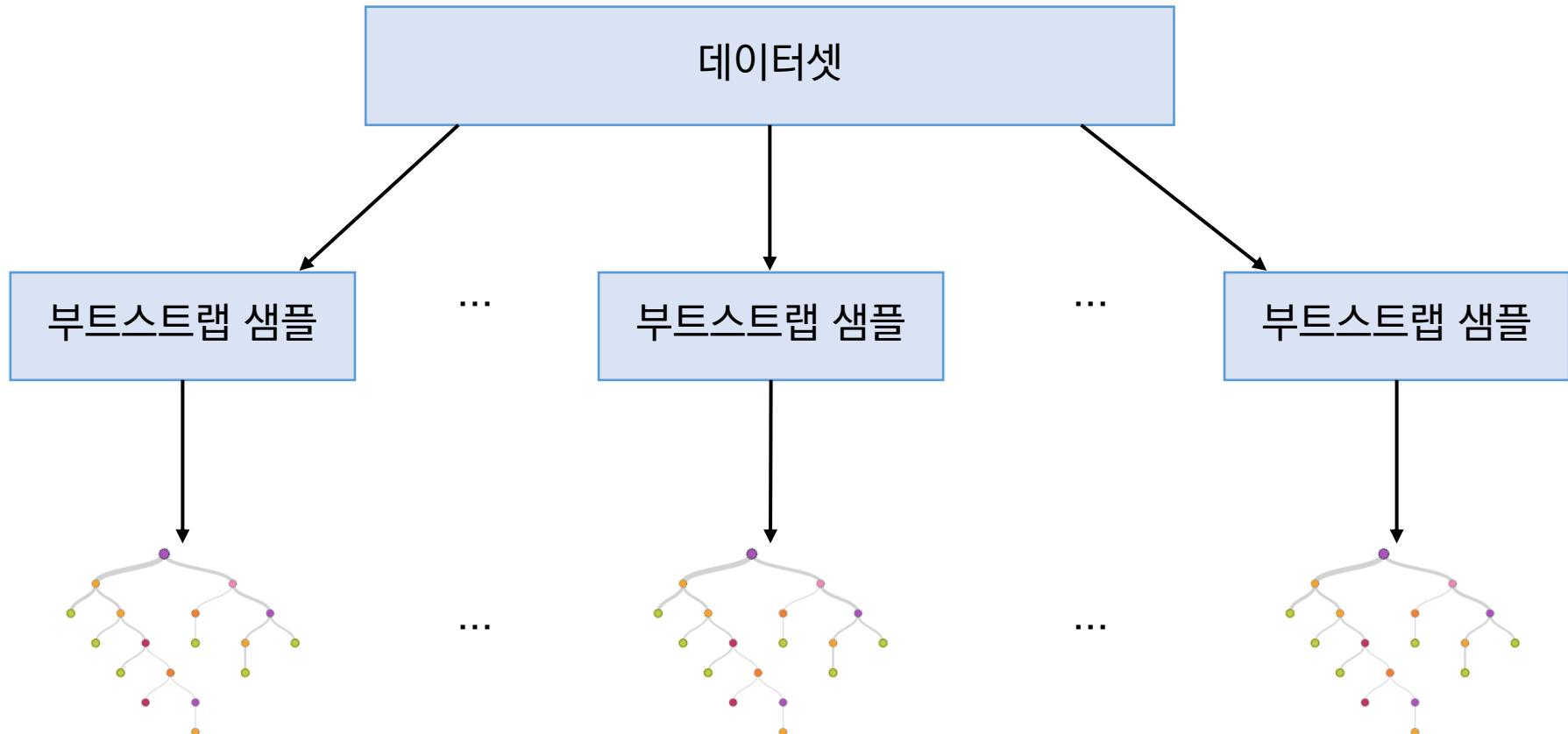
(0.36787944117144233, 0.6321205588285577)



# 베깅(Bagging)

## 4.2. 배깅

원데이터에서 Bootstrap으로 샘플링 후, 각 샘플에 대한 의사결정나무를 만들고,  
그 나무들을 모아 최종 의사결정을 하는 모델



부트스트랩을 이용하여 데이터를 샘플링하면, 그 데이터를 바탕으로 의사결정나무 모델을 선정한다.

# 베깅(Bagging)

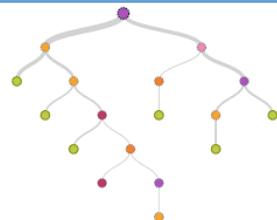
## 4.2. 배깅

원데이터에서 Bootstrap으로 샘플링 후, 각 샘플에 대한 의사결정나무를 만들고,  
그 나무들을 모아 최종 의사결정을 하는 모델

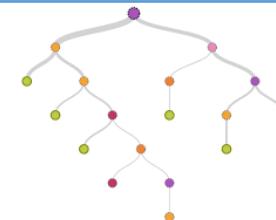
데이터셋

이 의사결정나무들은 모두 같은 의사결정나무일 수 없다.

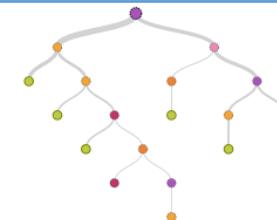
부트스트랩으로 샘플링 했으므로, .632+rule에 의해 어느정도의 중복과  
함께 각기 다른 샘플로 의사결정나무 모델을 만들었다고 할 수 있다.



...



...



부트스트랩을 이용하여 데이터를 샘플링하면, 그 데이터를 바탕으로 의사결정나무 모델을 선정한다.

# Sample 데이터

## 4.2. 배경

<https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

변수명	의미
Class label	등급(1,2,3)
Alcohol	알코올
Malic acid	능금산
Ash	회분
Alcalinity of ash	회분의 알칼리도
Magnesium	마그네슘
Total phenols	총 페놀
Flavanoids	플라보노이드
Nonflavanoid phenols	비플라보노이드 페놀
Proanthocyanins	프로안토시아닌
Color intensity	색상 강도
Hue	색조
OD280/OD315 of diluted wines	희석 된 와인의 OD280 / OD315
Proline	프롤린

# 데이터 불러오기

## 4.2. 배경

### 와인데이터 불러오기

```
1 import pandas as pd  
2  
3 wine_df = pd.read_csv("https://archive.ics.uci.edu/ml/  
4                         "machine-learning-databases/wine/wine.data", header=None)
```

executed in 743ms, finished 19:34:24 2021-06-24

```
1 wine_df.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',  
2                     'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',  
3                     'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',  
4                     'Proline']
```

executed in 6ms, finished 19:34:35 2021-06-24

```
1 wine_df.head()
```

executed in 18ms, finished 19:34:40 2021-06-24

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

# 독립변수와 종속변수 분리

## 4.2. 배깅

### 독립변수(X)와 종속변수(y) 분리하기

```
1 wine_df = wine_df[wine_df['Class_label'] != 1]  
2 wine_df.head()
```

executed in 261ms, finished 19:35:40 2021-06-24

Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline	
59	2	12.37	0.94	1.36	10.6	88	1.98	0.57	0.28	0.42	1.95	1.05	1.82	520
60	2	12.33	1.10	2.28	16.0	101	2.05	1.09	0.63	0.41	3.27	1.25	1.67	680
61	2	12.64	1.36	2.02	16.8	100	2.02	1.41	0.53	0.62	5.75	0.98	1.59	450
62	2	13.67	1.25	1.92	18.0	94	2.10	1.79	0.32	0.73	3.80	1.23	2.46	630
63	2	12.37	1.13	2.16	19.0	87	3.50	3.10	0.19	1.87	4.45	1.22	2.87	420

```
1 X = wine_df[['Alcohol', 'Hue']].values  
2 y = wine_df['Class_label'].values
```

executed in 11ms, finished 19:38:38 2021-06-24

```
1 X.shape, y.shape
```

executed in 9ms, finished 19:38:38 2021-06-24

((119, 2), (119,))

# 레이블인코딩 및 데이터 샘플링

## 4.2. 배깅

### 레이블인코딩 후 샘플링하기

```
1 from sklearn.preprocessing import LabelEncoder  
2 from sklearn.model_selection import train_test_split  
3  
4 le = LabelEncoder()  
5 y = le.fit_transform(y)  
6  
7 train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.4, random_state=1)
```

executed in 11ms, finished 19:38:38 2021-06-24

```
1 print(train_y)  
2 print(test_y)
```

executed in 7ms, finished 19:38:40 2021-06-24

```
[0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0  
0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0]  
[1 0 0 1 1 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 0  
1 0 0 1 0 0 1 0 1 1 1]
```

# 분류 모형

## 4.2. 배깅

### 의사결정나무 모형과 배깅 모형

```
1 from sklearn.ensemble import BaggingClassifier  
2 from sklearn.tree import DecisionTreeClassifier  
3  
4 tree = DecisionTreeClassifier(criterion='entropy', max_depth=None, random_state=1)  
5 bag = BaggingClassifier(base_estimator=tree, n_estimators=500, bootstrap=True, bootstrap_features=False, random_state=1)
```

executed in 16ms, finished 20:02:19 2021-06-24

```
1 tree.fit(train_X, train_y)
```

executed in 19ms, finished 20:02:19 2021-06-24

DecisionTreeClassifier(criterion='entropy', random\_state=1)

```
1 tree.score(test_X, test_y)
```

executed in 9ms, finished 20:02:20 2021-06-24

0.8333333333333334

```
1 bag.fit(train_X, train_y)
```

executed in 657ms, finished 20:02:21 2021-06-24

BaggingClassifier(base\_estimator=DecisionTreeClassifier(criterion='entropy',  
random\_state=1),  
n\_estimators=500, random\_state=1)

```
1 bag.score(test_X, test_y)
```

executed in 59ms, finished 20:02:21 2021-06-24

0.8958333333333334

DT모형의 결과와 Bagging  
모형의 결과를 비교하세요.

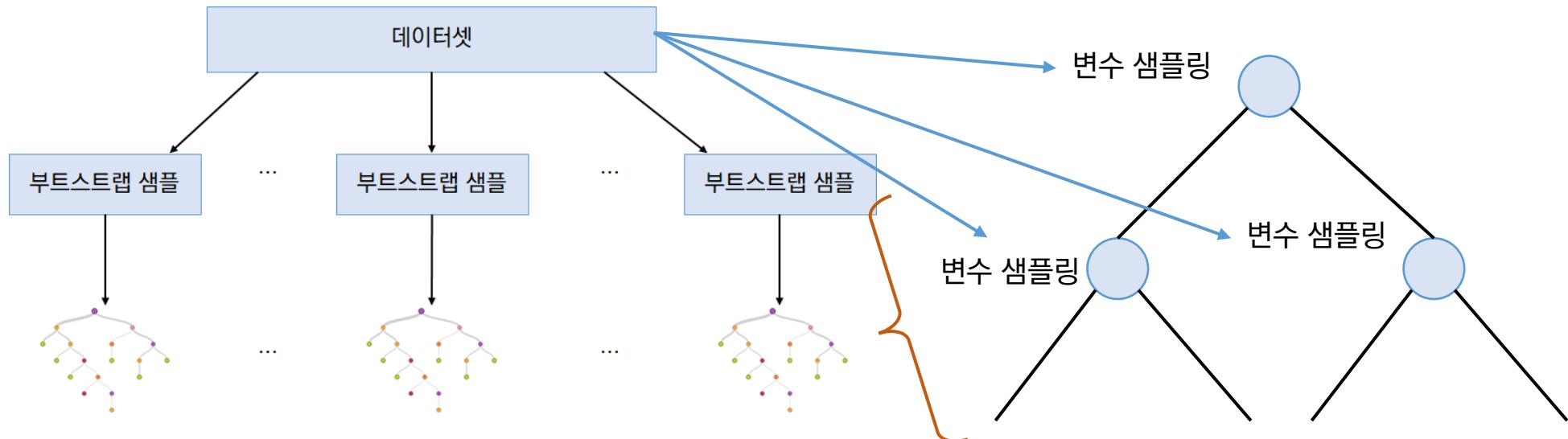
# RandomForest

## 4.2. 배깅

부트스트랩을 이용하여 데이터를 샘플링하고 그 데이터를 바탕으로 의사결정나무 모델을 설정  
+

의사결정나무 모델을 설정하는 과정에서 각 노드를 분류할 **변수들**도 샘플링

$\sqrt{p}$  ( $p =$  변수개수) 만큼 선택



매번 다른 변수들로 노드에서 분류가 되므로 각각의 DT들에는 큰 차이가 있음

→ 더 나은 일반화 성능을 만족시킴

# RandomForest

## 4.2. 배깅

```
1 from sklearn.ensemble import RandomForestClassifier  
2 rf = RandomForestClassifier()
```

executed in 13ms, finished 00:05:00 2021-07-14

```
1 rf.fit(train_X, train_y)
```

executed in 141ms, finished 00:05:06 2021-07-14

RandomForestClassifier()

```
1 rf.score(test_X, test_y)
```

executed in 20ms, finished 00:05:13 2021-07-14

0.9166666666666666

# 부스팅(Boosting)

## 4.3. 부스팅

여러 개의 알고리즘이 순차적으로 학습/예측을 하면서 이전에 학습한 알고리즘의 예측이 틀린 데이터를 올바르게 예측할 수 있도록, 다음 알고리즘에, 가중치를 부여하여 학습과 예측을 진행하는 방식

부스팅은 기본적으로 앙상블(Ensemble) 아이디어에서 Sequential이 추가된 형태

부스팅 알고리즘 종류

- AdaBoost
- GBM(Gradient Boosting Machine)
- XGBoost
- LightGBM
- CatBoost

# AdaBoost

## 4.3. 부스팅

AdaBoost는 Adaptive Boosting를 줄여서 부르는 말로 관측치들에 가중치를 더하면서 동작을 함

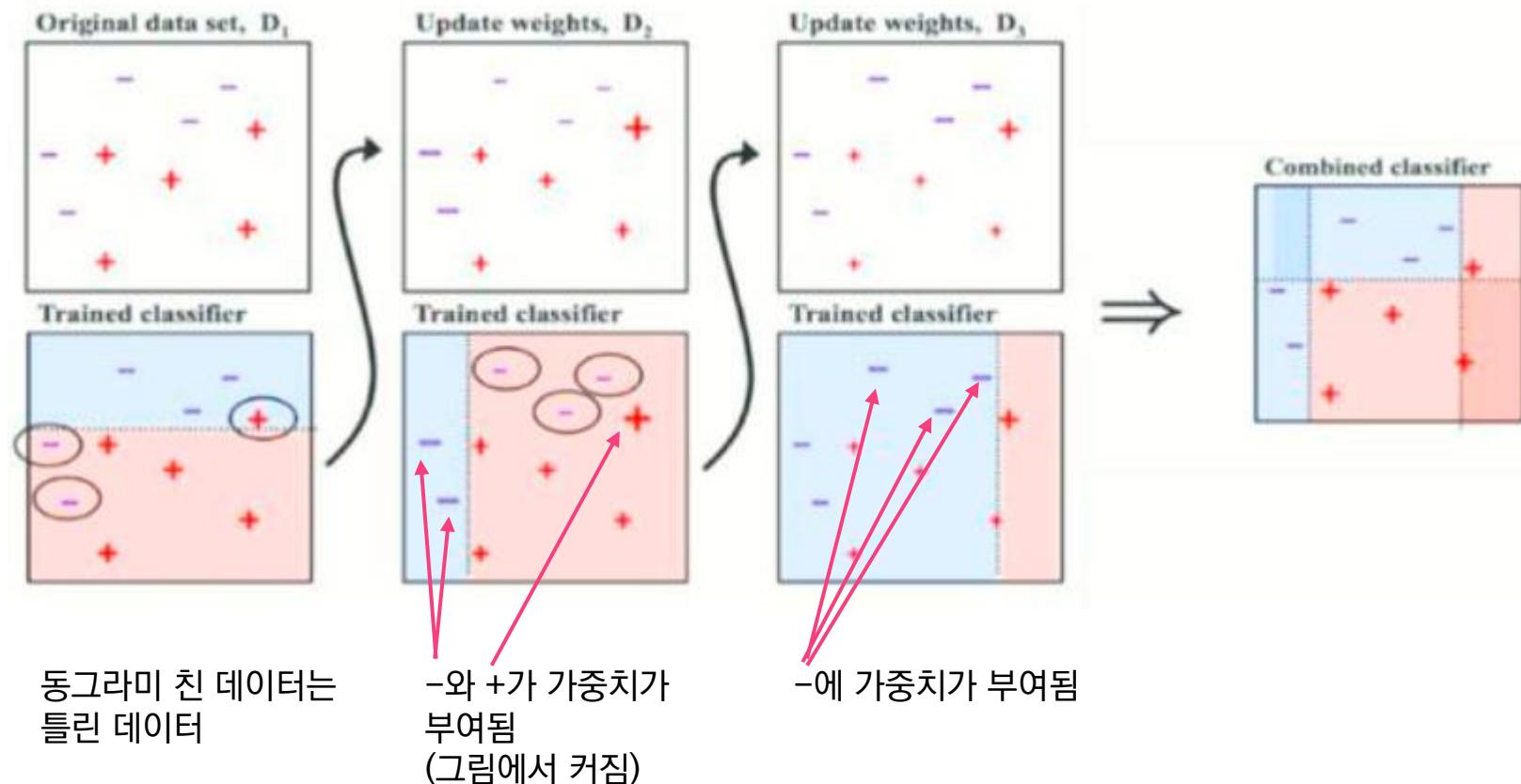
분류하기 어려운 Instances에는 가중치를 더하고 이미 잘 분류되어진 (다루어진) Instances는 가중치를 덜 함

- 즉, 약한 학습기(weak learner)의 오류에 가중치를 더하면서 부스팅을 수행하는 알고리즘
- 약한 학습기(weak learner)로 의사 결정 트리(Decision Tree)를 사용

# AdaBoost

## 4.3. 부스팅

Adaptive Boosting : 이진 분류기가 틀린 부분을 적응적으로(Adaptive) 바꿔가며 잘못 분류되는 데이터에 집중하도록 하는 것



# Gradient Boosting Machine(GBM)

## 4.3. 부스팅

GBM은 AdaBoost처럼 양상블에 이전까지의 오차를 보정하도록 예측기를 순차적(Sequential)으로 추가함

AdaBoost처럼 매 반복마다 샘플의 가중치를 조정하는 대신에 이전 예측기가 만든 잔여 오차(Residual Error)에 새로운 예측기를 학습시킴

- 가중치 업데이트를 경사하강법(Gradient Descent) 기법을 사용하여 최적화된 결과를 얻는 알고리즘
- Sequential + Additive Model
- 이전 모델의 Residual를 가지고 weak learn를 강화함  
즉, Residual를 예측하는 형태의 모델임
- 과적합(Overfitting) 이슈가 있음

# AdaBoost 분류 모형

## 4.3. 부스팅

```
1 from sklearn.ensemble import AdaBoostClassifier  
2 abm = AdaBoostClassifier()
```

executed in 18ms, finished 23:05:48 2021-07-13

```
1 abm.fit(train_X, train_y)
```

executed in 83ms, finished 23:05:53 2021-07-13

AdaBoostClassifier()

```
1 abm.score(test_X, test_y)
```

executed in 22ms, finished 23:06:00 2021-07-13

0.8958333333333334

# GradientBoosting

## 4.3. 부스팅

Gradient가 현재까지 학습된 분류기의 약점(weak)을 알려주고, 이후 모델이 그것을 중점으로 해서 보완을 하는 방식

```
1 from sklearn.ensemble import GradientBoostingClassifier  
2 gbm = GradientBoostingClassifier()
```

executed in 13ms, finished 23:03:19 2021-07-13

```
1 gbm.fit(train_X, train_y)
```

executed in 54ms, finished 23:03:22 2021-07-13

GradientBoostingClassifier()

```
1 gbm.score(test_X, test_y)
```

executed in 14ms, finished 23:03:37 2021-07-13

0.8958333333333334

# XGBoost

## 4.3. 부스팅

### Gradient Boosting의 문제점

- 느리다.
- 과적합(overfitting) 우려가 있다.

### eXtreme Gradient Boosting

- gbm보다 빠르다
- 과적합 방지가 가능한 규제가 포함되어 있다.
- CART(Classification And Regression Tree)를 기반으로 하므로 분류와 회귀 둘 다 가능하다.
- 조기종료(early stopping)을 제공한다.
- 앙상블 부스팅의 특징인 가중치 부여를 경사하강법(gradient descent)으로 한다.

# LightGBM

## 4.3. 부스팅

LightGBM은 기존의 Tree 기반 알고리즘과는 다르게 동작함

Tree 기반 알고리즘인 XGBoost의 경우 **균형 트리 분할(Level Wise)** 방식을 사용했다면, **LightGBM은 리프 중심 트리 분할(Leaf Wise)** 방식을 사용함

- **Level Wise** 트리 분석은 균형을 잡아주어야 하기 때문에 Tree의 깊이(depth)가 줄어들고 연산이 추가되는 것이 단점이면,
- **Leaf Wise**은 트리의 균형을 맞추지 않고 **최대 손실 값(Max data loss)**를 가지는 leaf 노드를 지속적으로 분할하면서 Tree의 깊이(depth)가 깊어지고 비대칭적인 트리가 생성

최대 손실값을 가지는 leaf node를 반복할 수록 **균형 트리 분할(Level wise)** 방식보다 예측 오류 손실을 최소화 할 수 있음

# XGBoost

## 4.3. 부스팅

$$\hat{y} = \alpha * tree_A + \beta * tree_B + \gamma * tree_C \dots$$

$K$ 개의 트리를 가지고 있음

$$\hat{y} = \sum_{k=1}^K f_k(x_i) \quad f_k \in F$$

$\hat{y}$  : 예측 값

$f_k$  :  $F$  공간 안에서  $k$ 번째 *decision tree*

$$Obj = \boxed{\sum_{t=1}^n l(y_i, \hat{y}_i)} + \boxed{\sum_{k=1}^K \Omega(f_k)}$$

Training loss    Regularization term

# XGBoost 설치

## 4.3. 부스팅

```
(base) C:\Users\JK>pip install xgboost
Collecting xgboost
  Downloading xgboost-1.4.2-py3-none-win_amd64.whl (97.8 MB)
    █████████████████████████████████████████ | 97.8 MB 37 kB/s
Requirement already satisfied: numpy in c:\users\jk\anaconda3\lib\site-packages (from xgboost) (1.19.5)
Requirement already satisfied: scipy in c:\users\jk\anaconda3\lib\site-packages (from xgboost) (1.6.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.4.2

(base) C:\Users\JK>
```

xgboost 오류 발생할 경우 <https://www.lfd.uci.edu/~gohlke/pythonlibs/#xgboost>에서  
파이썬 버전에 맞는 whl 파일 다운받아 설치  
pip install xgboost-1.4.2-cp38-cp38-win\_amd64.whl

# XGBoost 분류 모형

## 4.3. 부스팅

```
1 from xgboost import XGBClassifier  
2 xgb = XGBClassifier()
```

executed in 13ms, finished 23:18:08 2021-07-13

```
1 xgb.fit(train_X, train_y, eval_metric='logloss')
```

executed in 62ms, finished 23:22:28 2021-07-13

...

```
1 xgb.score(test_X, test_y)
```

executed in 11ms, finished 23:22:33 2021-07-13

0.8958333333333334

전체 코드는 실습파일을 참고하세요.

# XGBoost Parameters

## 4.3. 부스팅

- 학습 하이퍼파라미터
  - objective [기본설정값=reg:linear]: 지도학습 손실 최소화 함수를 정의
    - binary:logistic: 이항 분류 문제 로직스틱 회귀모형으로 반환값이 클래스가 아니라 예측 확률.
    - multi:softmax: 다항 분류 문제의 경우 소프트맥스(Softmax)를 사용해서 분류하는데 반환되는 값이 예측확률이 아니라 클래스임. 또한 num\_class도 지정해야함.
    - multi:softprob: 각 클래스 범주에 속하는 예측확률을 반환함.
  - eval\_metric: 설정한 objective별로 기본설정값이 지정되어 있음.
    - rmse: root mean square error
    - mae: mean absolute error
    - logloss: negative log-likelihood
    - error: Binary classification error rate (0.5 threshold)
    - merror: Multiclass classification error rate
    - mlogloss: Multiclass logloss
    - auc: Area under the curve
  - seed [기본설정값: 0]: 재현가능하도록 난수를 고정시킴.

참고 : <https://statklee.github.io/model/model-python-xgboost-hyper.html>

<https://xgboost.readthedocs.io/en/latest/parameter.html>

<https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>

# XGBoost Parameters

## 4.3. 부스팅

- 일반 하이퍼 파라미터
  - booster: 의사결정 기반 모형(gbtree), 선형 모형(linear)
  - mthread: 병렬처리에 사용되는 코어수, 특정값을 지정하지 않는 경우 자동으로 시스템 코어수를 탐지하여 병렬처리에 동원함.
- 부스팅 하이퍼 파라미터
  - eta [기본설정값: 0.3]: GBM에 학습율과 유사하고 일반적으로 0.01 ~ 0.2 값이 사용됨
  - min\_child\_weight [기본설정값: 1]: 과적합(overfitting)을 방지할 목적으로 사용되는데, 너무 높은 값은 과소적합(underfitting)을 야기하기 때문에 CV를 사용해서 적절한 값이 제시되어야 한다.
  - max\_depth [기본설정값: 6]: 과적합 방지를 위해서 사용되는데 역시 CV를 사용해서 적절한 값이 제시되어야 하고 보통 3-10 사이 값이 적용된다.
  - max\_leaf\_nodes: max\_leaf\_nodes 값이 설정되면 max\_depth는 무시된다. 따라서 두값 중 하나를 사용한다.
  - max\_delta\_step [기본설정값: 0]: 일반적으로 잘 사용되지 않음.
  - subsample [기본설정값: 1]: 개별 의사결정나무 모형에 사용되는 임의 표본수를 지정. 보통 0.5 ~ 1 사용됨.
  - colsample\_bytree [기본설정값: 1]: 개별 의사결정나무 모형에 사용될 변수갯수를 지정. 보통 0.5 ~ 1 사용됨.
  - lambda [기본설정값: 1]: 능선 회귀(Ridge Regression)의 L2 정규화(regularization) 하이퍼 파라미터. 그다지 많이 사용되고 있지는 않음.
  - alpha [기본설정값: 0]: 라쏘 회귀(Lasso Regression)의 L1 정규화(regularization) 하이퍼 파라미터로 차원이 높은 경우 알고리즘 속도를 높일 수 있음.
  - scale\_pos\_weight [기본설정값: 1]: 클래스 불균형이 심한 경우 0보다 큰 값을 지정하여 효과를 볼 수 있음.

# 하이퍼 파라미터 탐색

## 4.3. 부스팅

```
1 from sklearn.model_selection import GridSearchCV  
2 xgb_param_grid = {'max_depth': [3,5,7,9],  
3                   'subsample': [0.4, 0.6, 0.8, 1.0]}  
4 grid = GridSearchCV(estimator=xgb, param_grid=xgb_param_grid,  
5                      scoring='roc_auc', n_jobs=-1, cv=5,  
6                      refit=True, return_train_score=True)
```

executed in 7ms, finished 23:32:50 2021-07-13

```
1 grid.fit(train_X, train_y)
```

executed in 5.53s, finished 23:33:07 2021-07-13

```
1 grid_df.loc[:, ['mean_test_score', 'params']]  
executed in 25ms, finished 23:34:12 2021-07-13
```

	mean_test_score	params
0	0.962667	{'max_depth': 3, 'subsample': 0.4}
1	0.963556	{'max_depth': 3, 'subsample': 0.6}
2	0.959111	{'max_depth': 3, 'subsample': 0.8}
3	0.958222	{'max_depth': 3, 'subsample': 1.0}

```
1 grid_df = pd.DataFrame(grid.cv_results_)
```

executed in 16ms, finished 23:33:35 2021-07-13

```
1 grid_df.head()
```

executed in 37ms, finished 23:33:38 2021-07-13

```
1 grid_df[grid_df['rank_test_score'] == 1]
```

executed in 25ms, finished 23:34:51 2021-07-13

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	p
1	0.048584	0.003665	0.005398	7.997037e-04	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_subsample	params	split0_te
0	0.136756	0.057495	0.008997	0.007507	3	0.4	{'max_depth': 3, 'subsample': 0.4}	

# Voting에 의한 앙상블의 앙상블

## 4.5. 투표를 이용한 앙상블

여러 모형을 사용해서 다수의 규칙에 투표해서 예측된 클래스의 레이블을 사용할 수 있음

	Model A	Model B	Model C	분류 결과
0/1 이진분류	1	0	1	1
	0	0	1	0
	1	1	0	1
	1	1	1	1

# VotingClassifier

## 4.5. 투표를 이용한 앙상블

- VotingClassifier를 사용하면 여러 모형을 사용해서 다수의 규칙에 투표해서 예측된 클래스의 레이블을 사용할 수 있음

```
sklearn.ensemble.VotingClassifier(estimators,  
                                 voting='hard', weights=None, n_jobs=None,  
                                 flatten_transform=True)
```

*voting* : 문자열, ‘hard’ 또는 ‘soft’, 기본값 ‘hard’, ‘hard’인 경우 다수의 규칙 투표에 예측된 클래스 레이블을 사용합니다. ‘soft’인 경우 예측된 확률의 합 argmax를 기반으로 클래스 레이블을 예측하며 이것은 잘 보정된 분류모형의 앙상블에 권장됩니다. 예를 들면 어떤 데이터를 분류 예측할 경우 ‘hard’인 경우 분류모형 A, B, C가 이진분류(0 또는 1) 결과가 각각 0, 0, 1인 경우 그 결과는 0으로 분류되지만 ‘soft’인 경우 분류모형 A, B, C의 결과가 각각 (0.6, 0.4), (0.55, 0.45), (0.2, 0.8)이라면 각 결과의 합은 (1.35, 1.65)가 되어 1로 분류됩니다.

# hard voting

## 4.5. 투표를 이용한 앙상블

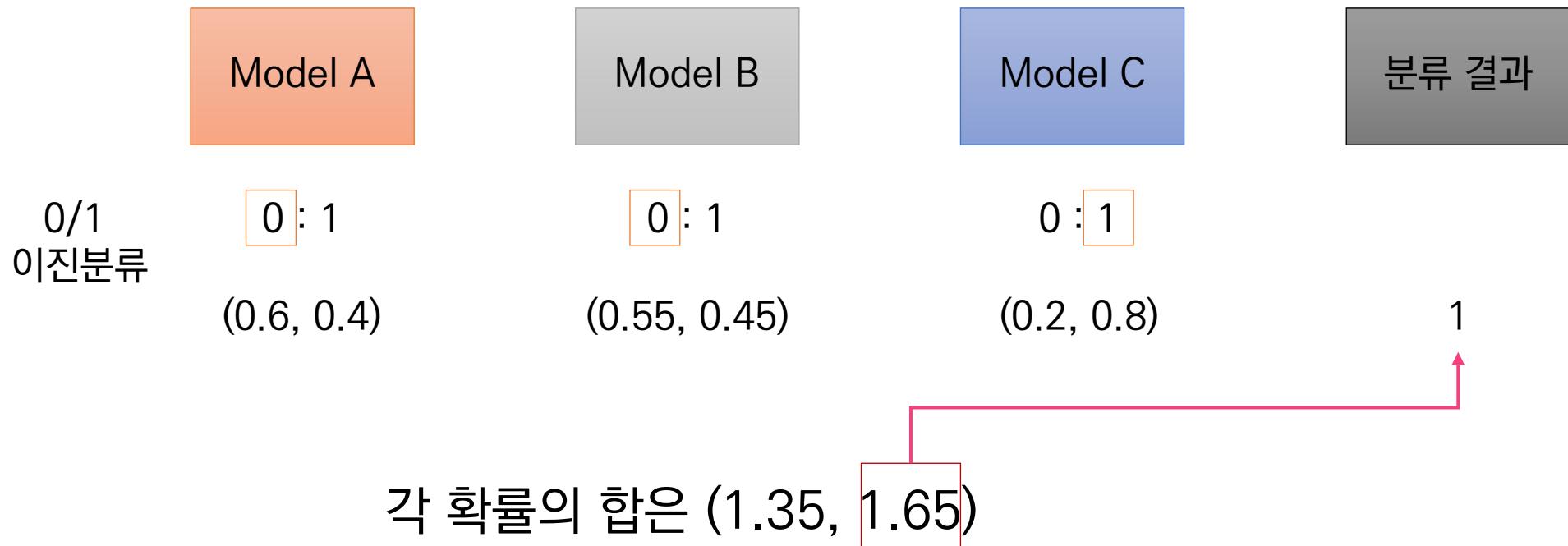
다수의 규칙 투표에 예측된 클래스 레이블을 사용

	Model A	Model B	Model C	분류 결과
0/1 이진분류	1	0	1	1
	0	0	1	0
	1	1	0	1
	1	1	1	1

# soft voting

## 4.5. 투표를 이용한 앙상블

분류할 확률의 총 합을 이용해서 결정함



# VotingClassifier – hard

## 4.5. 투표를 이용한 앙상블

```
1 from sklearn.ensemble import VotingClassifier  
2 voting_model = VotingClassifier(estimators=[("bagging", bag),  
3                                         ("random forest", rf),  
4                                         ("xgboost", xgb)],  
5                                         voting="hard")
```

executed in 4ms, finished 00:19:27 2021-07-14

```
1 voting_model.fit(train_X, train_y)
```

executed in 778ms, finished 00:21:39 2021-07-14

...

```
1 voting_model.score(test_X, test_y)
```

executed in 68ms, finished 00:21:41 2021-07-14

0.9166666666666666

# VotingClassifier – soft

## 4.5. 투표를 이용한 앙상블

```
1 voting_model = VotingClassifier(estimators=[("bagging", bag),  
2                               ("random forest", rf),  
3                               ("xgboost", xgb)],  
4                               voting="soft")
```

executed in 8ms, finished 00:23:17 2021-07-14

```
1 voting_model.fit(train_X, train_y)
```

executed in 771ms, finished 00:23:19 2021-07-14

...

```
1 voting_model.score(test_X, test_y)
```

executed in 73ms, finished 00:23:24 2021-07-14

0.875