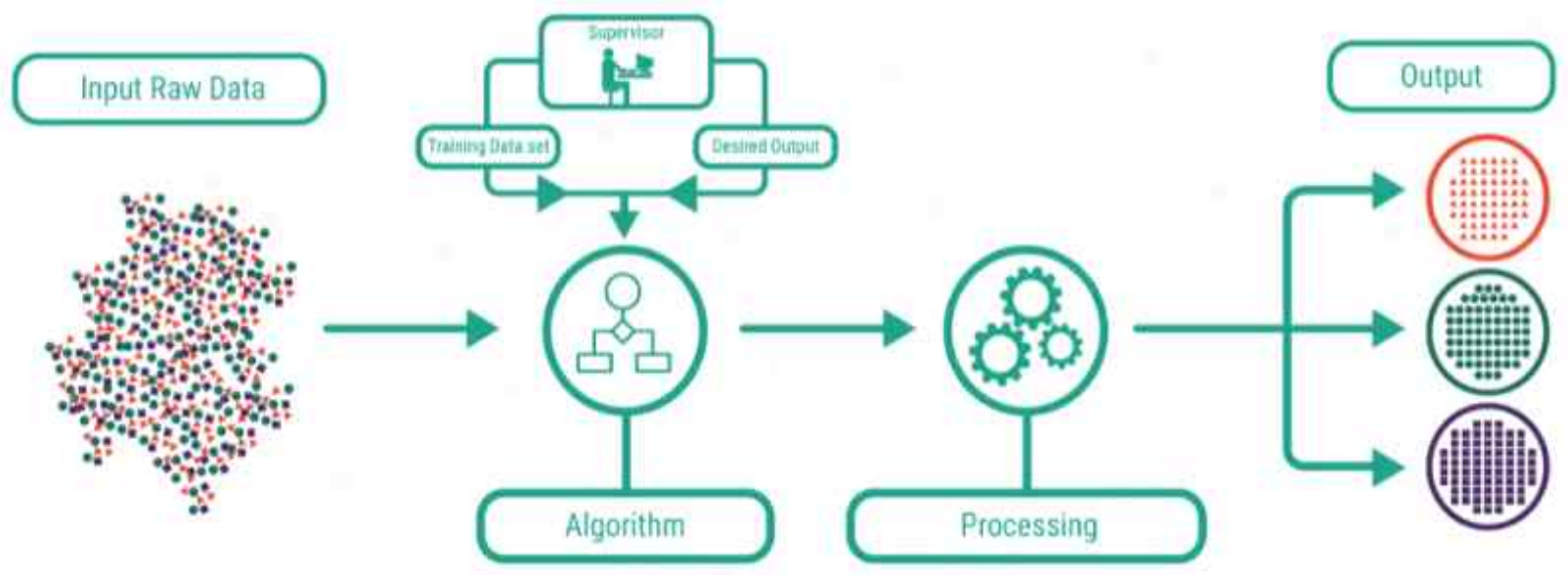
지도학습

지도학습



지도학습은 답이 있는 데이터를 CNN, SVM 같은 알고리즘을 통해 학습시키고 교차 검증을 수행하는 기법으로 적확도는 높으나 데이터 량이 많이 필요하고 학습하는데 오랜 시간이 걸립니다.

지도 학습의 기법

분류
-이진 분류 : 두 종류로 나누어 분류하는기법
-다중 분류 : 여러 종류로 나누어 분류하는 기법

회귀
-독립변수 기반 분석 : 입력값의 갯수에 따른 분석
-종속변수 기반분석 : 종속변수의 갯수에 따른 분석

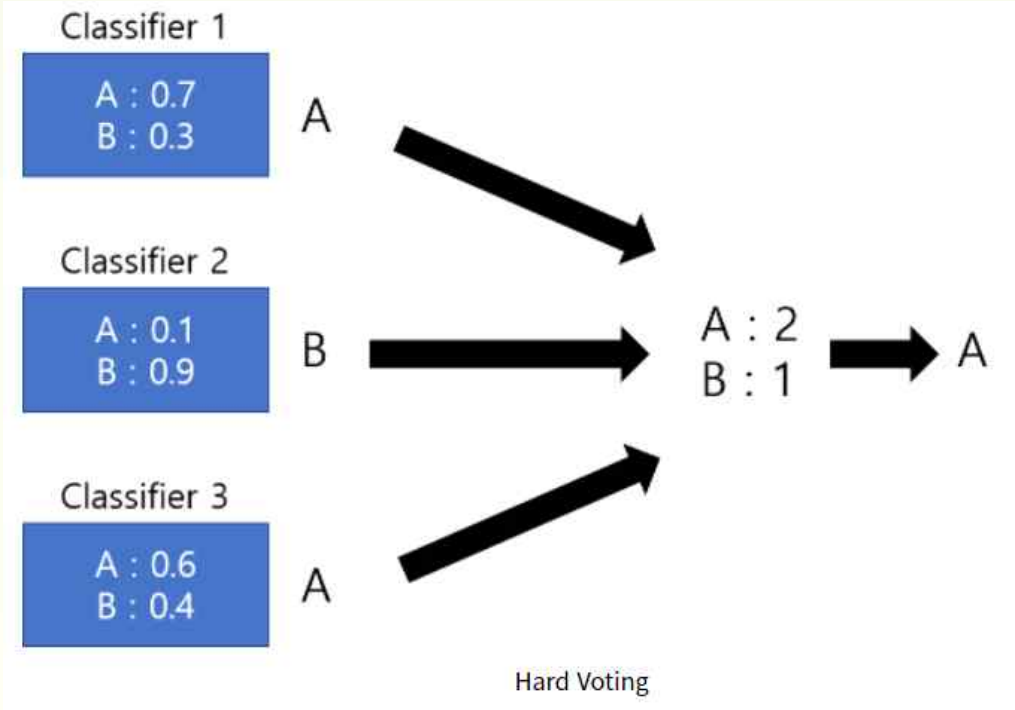
앙상블 기법

앙상블 기법은 머신러닝을 위한 다양한 학습 모델을 결합하여 학습시키는 것으로, 하나의 학습알고리즘 으로 학습시킨것보다 더 좋은 예측성능을 얻을 수 있는 기법으로 Voting, Bagging, Boosting, Stacking 등이 있습니다.

Voting	Bagging	Boosting	Stacking
여러 모델이 투표를 통해 최종 예측 결과를 결정하는 방식	중복을 허용하여 데이터를 분할 후 같은유형의 모델을 사용해 학습시켜 결과를 집계하는 방식	여러 학습모델을 순차적으로 사용하며 이전 모델에서 예측이 틀린 데이터를 다음 모델에서 올바르게 예측하도록 가중치를 부여하며 학습시키는 방식	여러 모델들을 활용해 각각의 예측 결과를 도출한 뒤 그 예측 결과를 결합해 최종 예측 결과를 만들어내는 방식

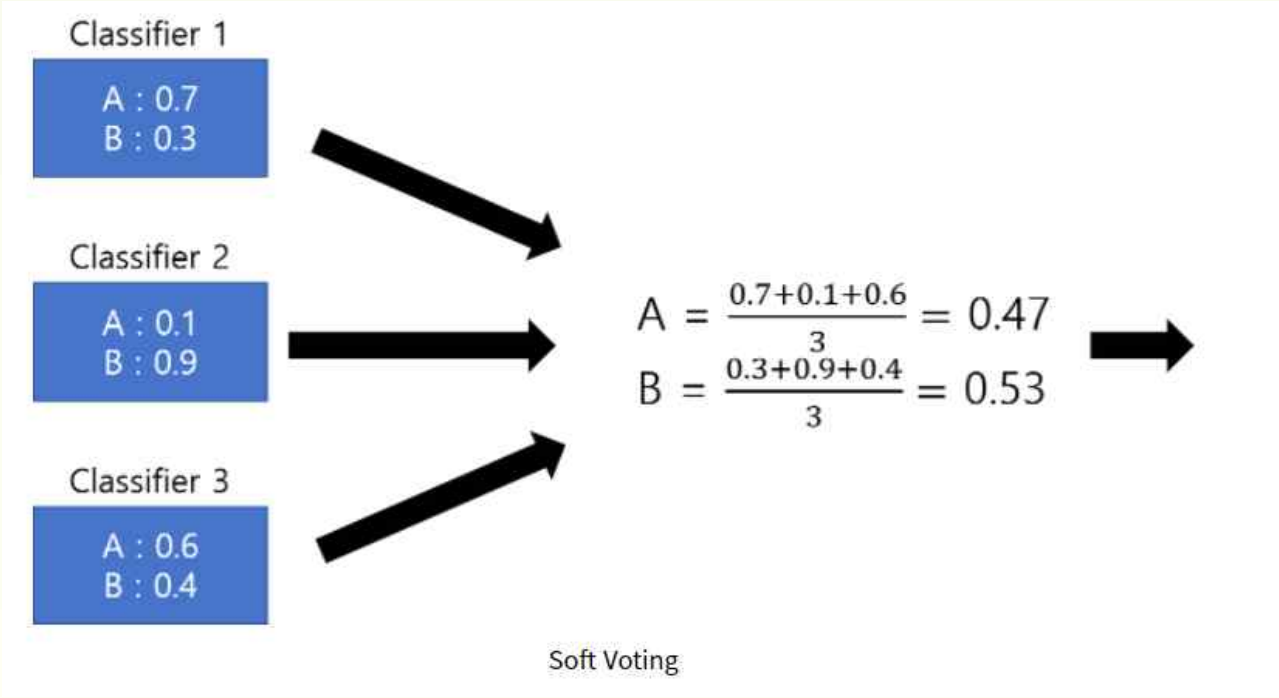
Hard Voting이 동작하는 방식

각각의 모델들이 결과를 예측하면 단순하게 가장 많은 표를 얻은 결과를 선택하는 것.



Soft Voting이 동작하는 방식

각 class별로 모델들이 예측한 probability를 합산해서 가장 높은 class를 선택



In [2]:
```
# 필요한 모듈과 데이터 불러오기
import pandas as pd

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from warnings import filterwarnings
filterwarnings('ignore')

cancer = load_breast_cancer()

data_df = pd.DataFrame(cancer.data, columns = cancer.feature_names)
data_df.head(3)
```

Out[2]:

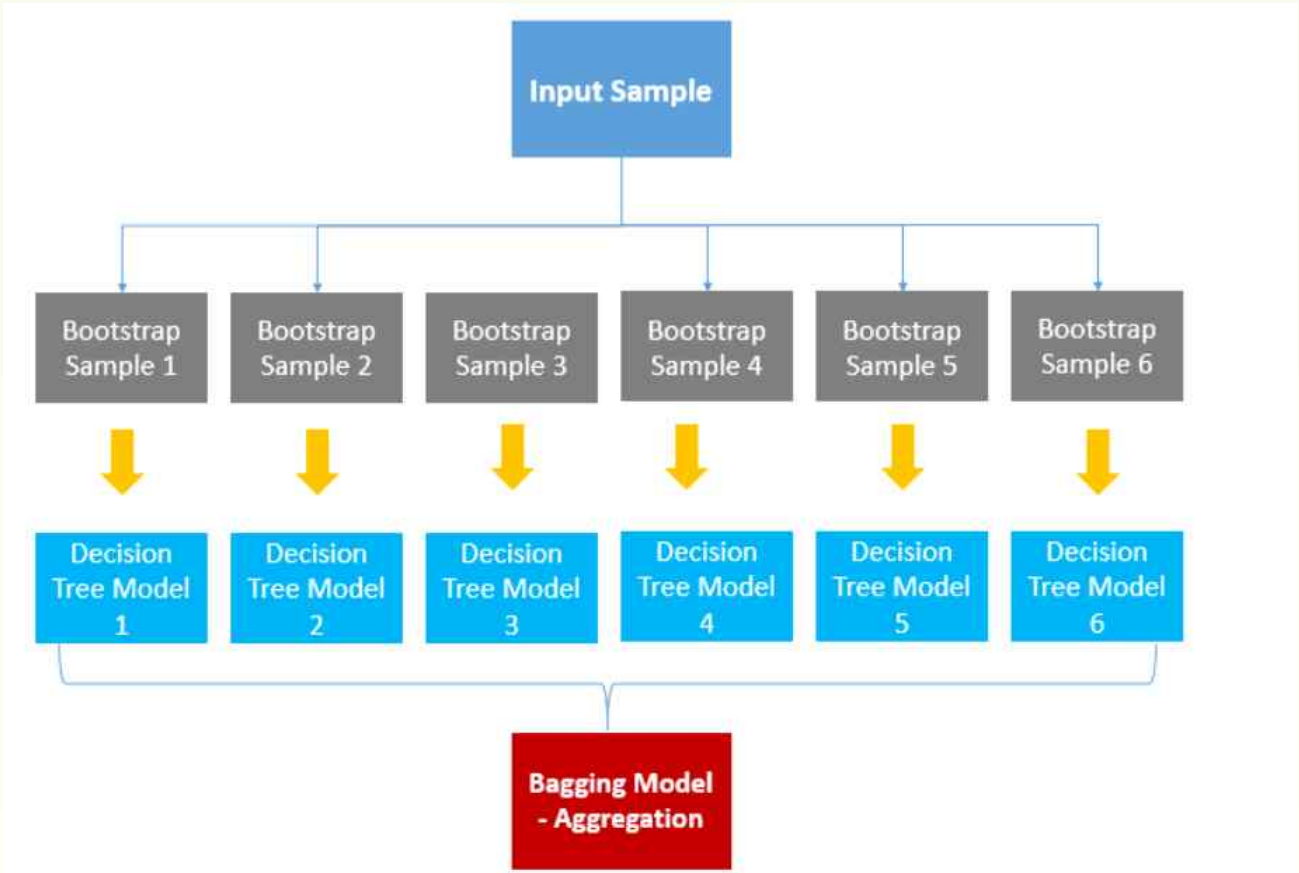
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.6	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.8	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.5	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758

3 rows × 30 columns

In [3]:
```
# 보팅 적용을 위한 개별 모델은 로지스틱 회귀와 KNN입니다.
logistic_regression = LogisticRegression()
knn = KNeighborsClassifier(n_neighbors=8)

# 개별모델을 소프트보팅 기반의 앙상블 모델로 구현한 분류기
voting_model = VotingClassifier(estimators=[ ('LogisticRegression', logistic_regression), ('KNN', k
nn)], voting='soft')

# 데이터를 훈련셋과 테스트셋으로 나누기
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.2, rand
om_state=156)

# 보팅 분류기의 학습/예측/평가
voting_model.fit(X_train, y_train)
pred = voting_model.predict(X_test)
print('보팅 분류기의 정확도: {0: .4f}'.format(accuracy_score(y_test, pred)))

# 개별 모델의 학습/예측/평가
classifiers = [logistic_regression, knn]
for classifier in classifiers:
    classifier.fit(X_train, y_train)
    pred = classifier.predict(X_test)
    class_name = classifier.__class__.__name__
    print('{0} 정확도: {1:.4f}'.format(class_name, accuracy_score(y_test, pred)))
```

```
보팅 분류기의 정확도:  0.9561
LogisticRegression 정확도: 0.9474
KNeighborsClassifier 정확도: 0.9386
```

Bagging이 동작하는 방식



```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

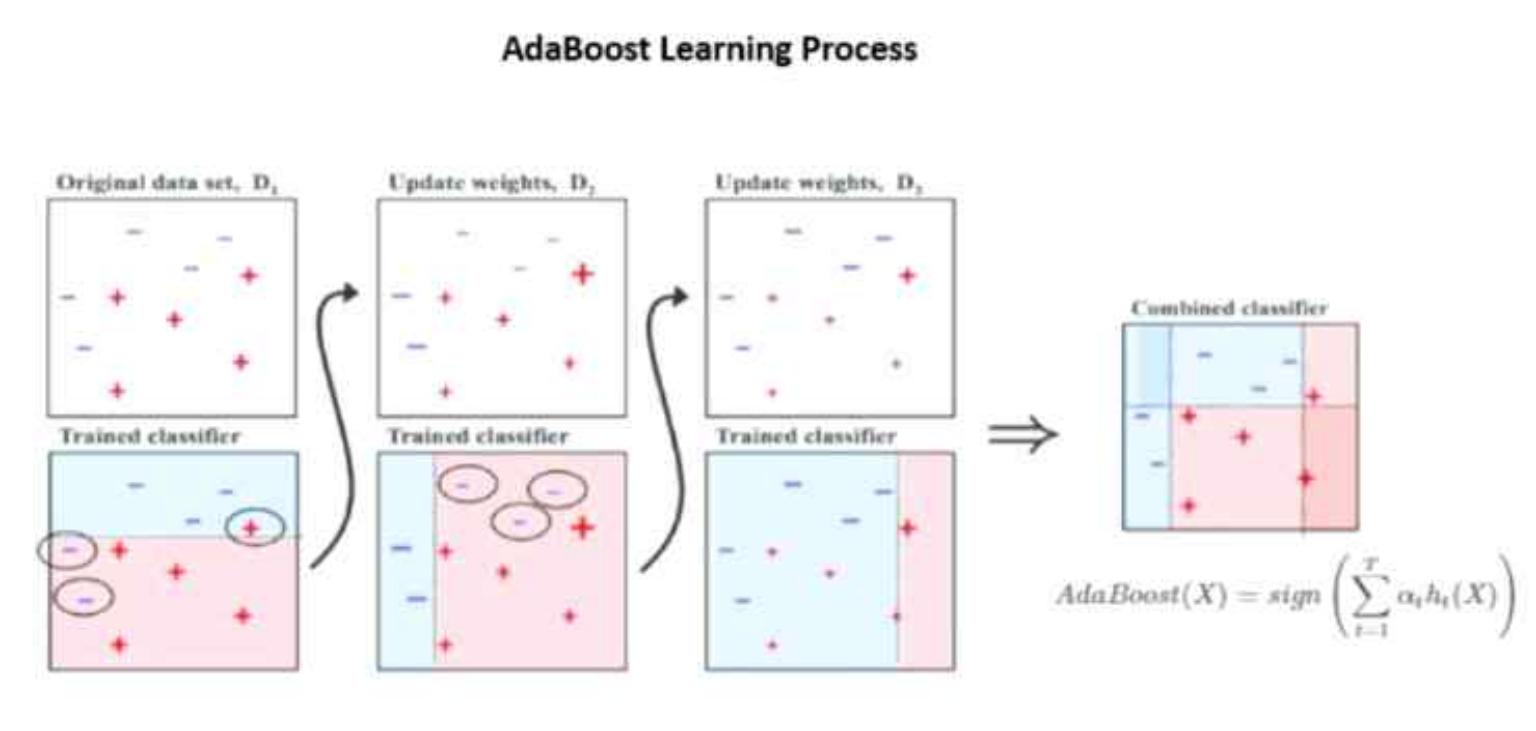
Bagging

base_estimator : 앙상블 학습을 위한 베이스 예측기 (Decision Tree, SVM,LogisticRegression 등)
n_estimators : 앙상블에 사용할 분류기의 수
bootstrap : True는 배깅, False는 페이스팅
n_jobs : fit, predict에 사용할 CPU 코어 수 지정 (None(=1)이 기본값, -1로 설정하면 모든 프로세서를 다 사용)
oob_score : True로 설정하면 자동으로 oob평가 수행

```
bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)

bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)

accuracy_score(y_test, y_pred)
>>>0.912
```

```
bag_clf2 = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1, oob_score=T
rue)

bag_clf2.fit(X_train, y_train)
bag_clf2.oob_score_
>>>0.9253333333333333
```

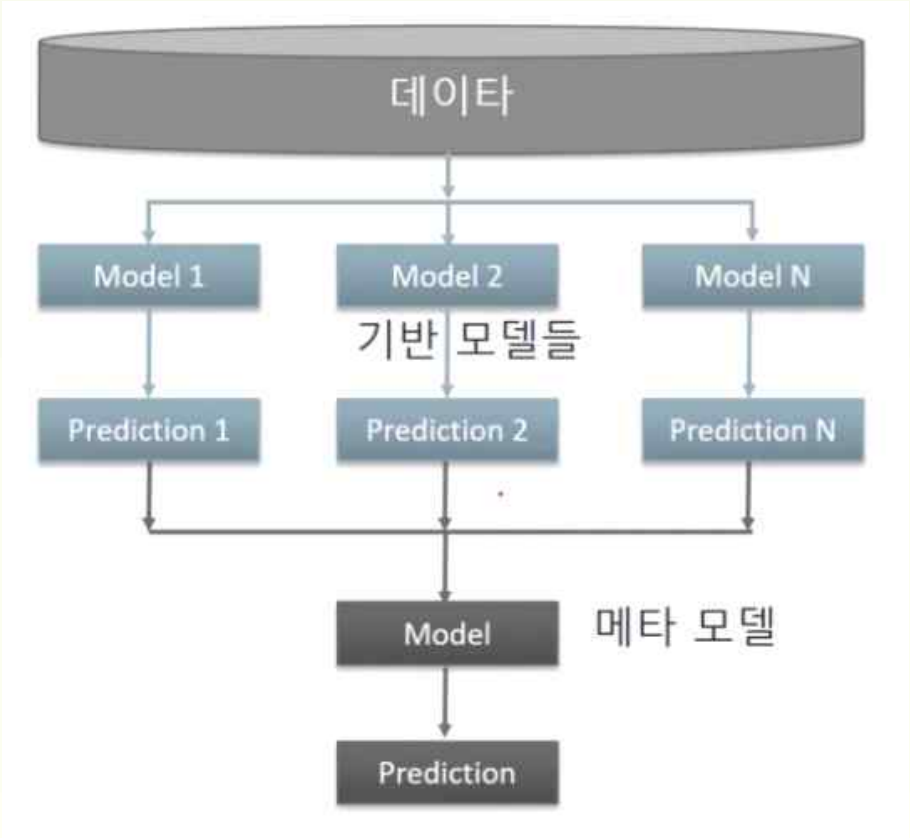
Boosting이 동작하는 방식



Boosting

```
import xgboost as xgb
import time
start = time.time() # 시작 시간 지정
xgb_dtrain = xgb.DMatrix(data = train_x, label = train_y) # 학습 데이터를 XGBoost 모델에 맞게 변환
xgb_dtest = xgb.DMatrix(data = test_x) # 평가 데이터를 XGBoost 모델에 맞게 변환
xgb_param = {'max_depth': 10, # 트리 깊이, default = 6
        'learning_rate': 0.01, # Step Size
        'n_estimators': 100, # Number of trees, 트리 생성 개수
        'objective': 'multi:softmax', # 목적 함수    # 현재 분류 / eval_metric과 objective는 범주형/연속형 경우 나누어 작성필요
        'num_class': len(set(train_y)) + 1} # 파라미터 추가, Label must be in [0, num_class) -> num_class보다 1 커야한다.
xgb_model = xgb.train(params = xgb_param, dtrain = xgb_dtrain) # 학습 진행
xgb_model_predict = xgb_model.predict(xgb_dtest) # 평가 데이터 예측
print("Accuracy: %.2f" % (accuracy_score(test_y, xgb_model_predict) * 100), "%") # 정확도 % 계산
print("Time: %.2f" % (time.time() - start), "seconds") # 코드 실행 시간 계산
```

Stacking이 동작하는 방식



```python
import numpy as np

# 스태킹 모델에 사용할 알고리즘
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# 위스콘신 유방암 예제 데이터 로드
# metrics로 accuracy를 사용
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer_data = load_breast_cancer()

X_data = cancer_data.data
y_label = cancer_data.target

X_train, X_test, y_train, y_test = train_test_split(X_data, y_label,
test_size=0.2)
```

```python
# 개별 ML 모델 객체 생성 (기반모델)
knn_clf = KNeighborsClassifier(n_neighbors=4)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=30)
dt_clf = DecisionTreeClassifier()
ada_clf = AdaBoostClassifier(n_estimators=100)

# 메타모델(스태킹으로 만들어진 데이터 학습 및 예측)
lr_final = LogisticRegression(C=10)
```

```python
# 개별 모델 학습

knn_clf.fit(X_train, y_train)
rf_clf.fit(X_train, y_train)
dt_clf.fit(X_train, y_train)
ada_clf.fit(X_train, y_train)
```

```
AdaBoostClassifier(n_estimators=100)
```

```python
# 기반 모델 예측 세트와 정확도 확인

knn_pred = knn_clf.predict(X_test)
rf_pred = rf_clf.predict(X_test)
dt_pred = dt_clf.predict(X_test)
ada_pred = ada_clf.predict(X_test)

print('KNN 정확도 :',accuracy_score(y_test, knn_pred))
print('RF 정확도 :',accuracy_score(y_test, rf_pred))
print('DT 정확도 :',accuracy_score(y_test, dt_pred))
print('ADA부스트 정확도 :',accuracy_score(y_test, ada_pred))
```

```
KNN 정확도 : 0.9385964912280702
RF 정확도 : 0.9649122807017544
DT 정확도 : 0.9736842105263158
ADA부스트 정확도 : 0.9473684210526315
```

```
# 기반 모델의 예측 결과를 스태킹

stacked_pred = np.array([knn_pred, rf_pred, dt_pred, ada_pred])
print(stacked_pred.shape)

# transpose를 이용, 행과 열의 위치를 교환, 칼럼 레벨로 각 모델의 예측 결과를 피처로 사용

stacked_pred = np.transpose(stacked_pred)
print(stacked_pred.shape)
```

```
(4, 114)
(114, 4)
```

```
# 메타 모델은 기반모델의 예측결과를 기반으로 학습

lr_final.fit(stacked_pred, y_test)
final_pred = lr_final.predict(stacked_pred)

print('최종 메타 모델 정확도 : ',accuracy_score(y_test, final_pred))
```

```
최종 메타 모델 정확도 :  0.9824561403508771
```

과적합

학습이 반복되며 정확도가 올라갑니다. 이상적인 학습이란 데이터가 계속 들어와 학습이 반복되면 될수록 정확도가 높아지는 것입니다.
하지만 이 과정에서 학습 모델이 주어진 데이터에 너무 과하게 맞춰져서**(overfit)** 조금이라도 다른 데이터만 들어와도 다른 결과로 예측하여 결과적으로 정확도가 낮아지는 현상을 '과적합' 이라고 합니다.

과적합 방지

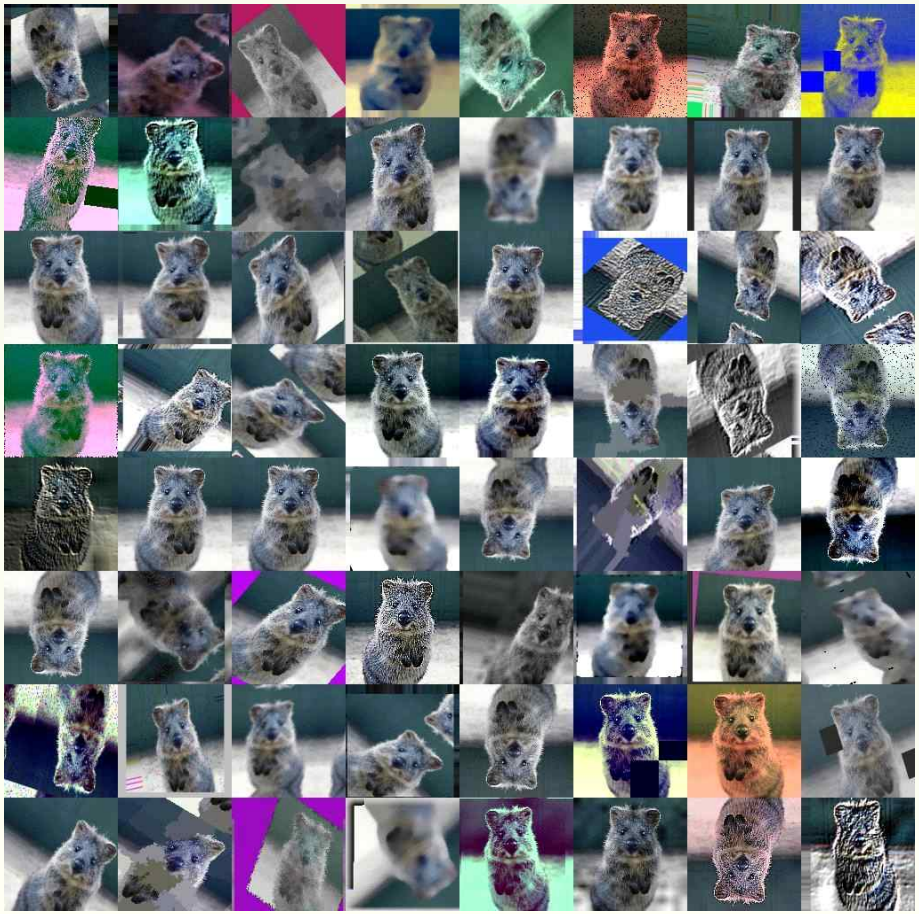
과적합 방지 방법

1. 학습 모델 단순화
2. 입력값 정규화
3. 가중치 규제 적용
4. 데이터의 양을 늘리기
5. 상황에 맞게 앙상블 기법 적용

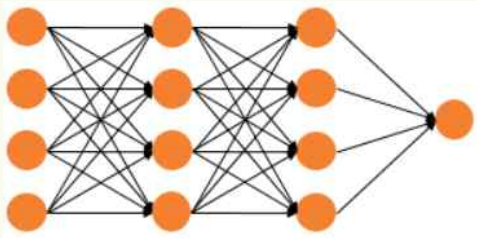
등 그 외에도 어떤 데이터를 어느방식 으로 학습 시키느냐에 따라 여러 과적합 방지 방법이 있습니다.

Data Augmentation

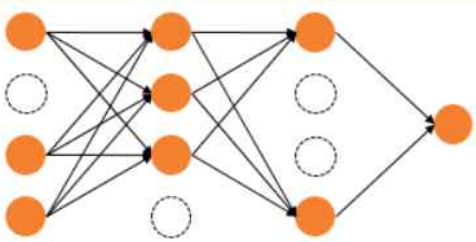
데이터양을 늘리는 방법으로
예를 들어 이미지 데이터
하나를 색 명암, 또는 각도를
회전시켜 데이터를
증강시킵니다.



드롭아웃(Dropout)



드롭아웃 적용 전



드롭아웃 적용 후

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense

max_words = 10000
num_classes = 46

model = Sequential()
model.add(Dense(256, input_shape=(max_words,), activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(num_classes, activation='softmax'))
```

드롭아웃은 신경망 학습 시에만 사용하고, 예측 시에는 사용하지 않는 것이 일반적입니다. 학습 시에 인공 신경망이 특정 뉴런 또는 특정 조합에 너무 의존적이게 되는 것을 방지해주고, 매번 랜덤 선택으로 뉴런들을 사용하지 않으므로 서로 다른 신경망들을 앙상블하여 사용하는 것 같은 효과를 내어 과적합을 방지합니다.

학습 모델 단순화

```
import tensorflow as tf


model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

학습 모델 깊이를 줄여 단순화 시킵니다.

가중치 규제 적용

.규제 (Regularization)
학습이 과대적합 되는 것을 방지하고자 일종의 penalty를 부여하는 것

L2 규제 (L2 Regularization)

각 가중치 제곱의 합에 규제 강도(Regularization Strength) λ를 곱한다.
λ를 크게 하면 가중치가 더 많이 감소되고(규제를 중요시함), λ를 작게 하면 가중치가 증가합니다(규제를 중요시하지 않음).

L1 규제 (L1 Regularization)

가중치의 제곱의 합이 아닌 가중치의 합을 더한 값에 규제 강도 (Regularization Strength) λ를 곱하여 오차에 더합니다.
어떤 가중치(w)는 실제로 0이 됩니다. 즉, 모델에서 완전히 제외되는 특성이 생기는 것입니다.

L1 Regularization

Lasso (L1 Regularization)

Lasso(Least Absolute Shrinkage and Selection Operator)

- 선형 회귀에 L1 규제 계수를 적용합니다.
- 가중치(weight)의 절대 값의 합을 최소화 하는 계수를 추가 합니다.
- 불필요한 회귀 계수를 급격히 감소, 0으로 만들어 제거합니다.
- 특성(Feature) 선택에 유리합니다.

주요 hyperparameter

- `alpha` : L1 규제 계수

수식

$$Error = MSE + \alpha|w|$$

```python
from sklearn.linear_model import Lasso
```

```python
# 값이 커질 수록 큰 규제입니다.
alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]
```

```python
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(x_train, y_train)
    pred = lasso.predict(x_test)
    add_model('Lasso(alpha={})'.format(alpha), pred, y_test)
plot_all()
```

L2 Regularization

Ridge (L2 Regularization)

- L2 규제 계수를 적용합니다.
- 선형회귀에 가중치 (weight)들의 제곱합에 대한 최소화를 추가합니다.

주요 hyperparameter

- `alpha` : 규제 계수

수식

$$Error = MSE + \alpha w^2$$

```
# 값이 커질 수록 큰 규제입니다.
alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]
```

```
for alpha in alphas:
    ridge = Ridge(alpha=alpha, random_state=SEED)
    ridge.fit(x_train, y_train)
    pred = ridge.predict(x_test)
    add_model('Ridge(alpha={})'.format(alpha), pred, y_test)
plot_all()
```

감사합니다.