3. 앙상블 기법 (Ensemble)

3-1. 과잉적합 문제와 Regularization

3-2. Cross validation 시험

3-3. Confusion matrix (accuracy, precision, recall, f-measure)

3-4. 배깅, 부스팅

3-5. 랜덤 포레스트 (Random Forest)

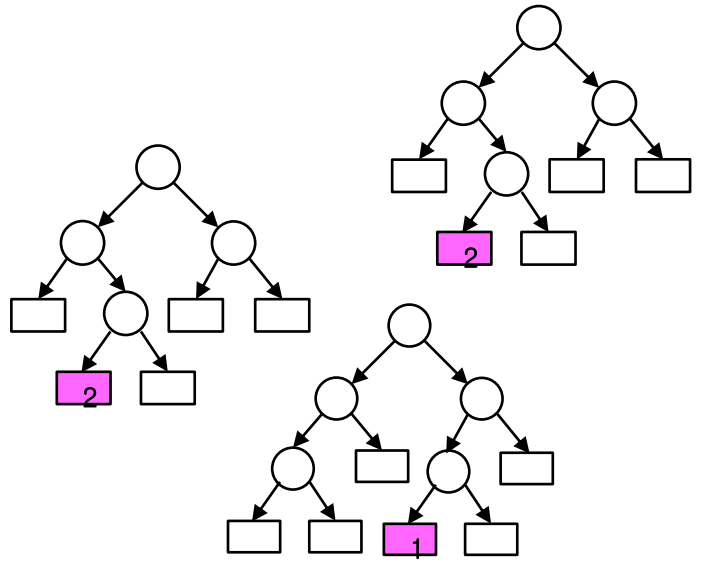
3-6. AdaBoot (Adaptive Boosting)

3-7. Gradient Boosting

3-8. Extreme Gradient Boosting (XGBoost)

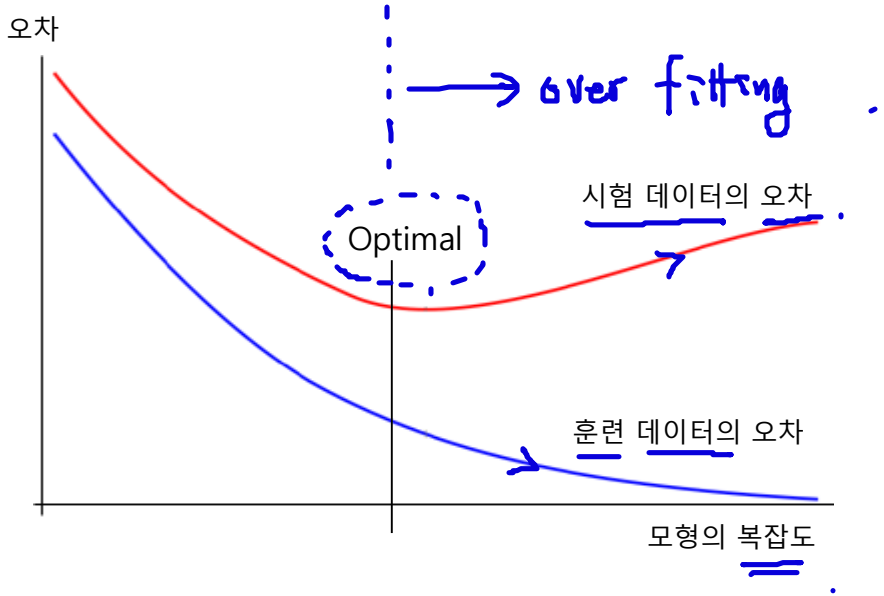
3-9. Light GBM

3-10. Isolation Forest (iForest)

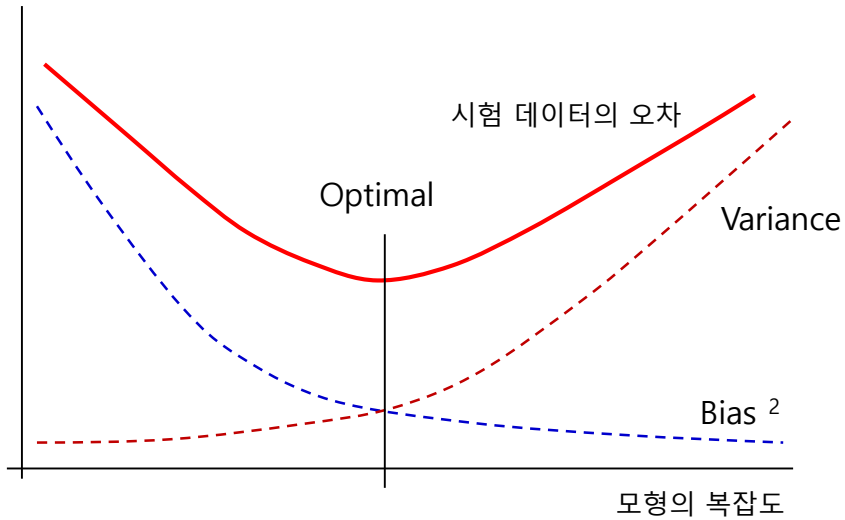


과잉적합 (Overfitting) 문제와 Regularization

- 모형이 복잡하거나 학습이 과도하면 훈련 데이터 자체는 정확히 설명할 수 있으나, 학습에 사용되지 않은 시험 데이터에 대해서는 설명력이 떨어질 수 있음.
  → 과잉적합 문제 (Overfitting) → 일반화 특성이 좋지않음.
- 데이터 마이닝에서 과잉적합은 일반화 성능을 떨어뜨리는 주요한 요인이므로 대단히 중요한 문제임.
- 과잉적합을 방지하기 위해서는 Validation 데이터를 이용하여 오류를 평가하고 최적의 학습 중단 지점을 설정해야 함.
- 오차를 최소화하는 함수에 (Cost function) 모형의 복잡도에 대한 Penelty 항을 추가함. Penelty 항으로 인해 복잡도가 증가할수록 오차는 증가하므로, 오차를 최소화하려는 알고리즘이 복잡도가 증가하지 않는 방향으로 진행함. → Regularization



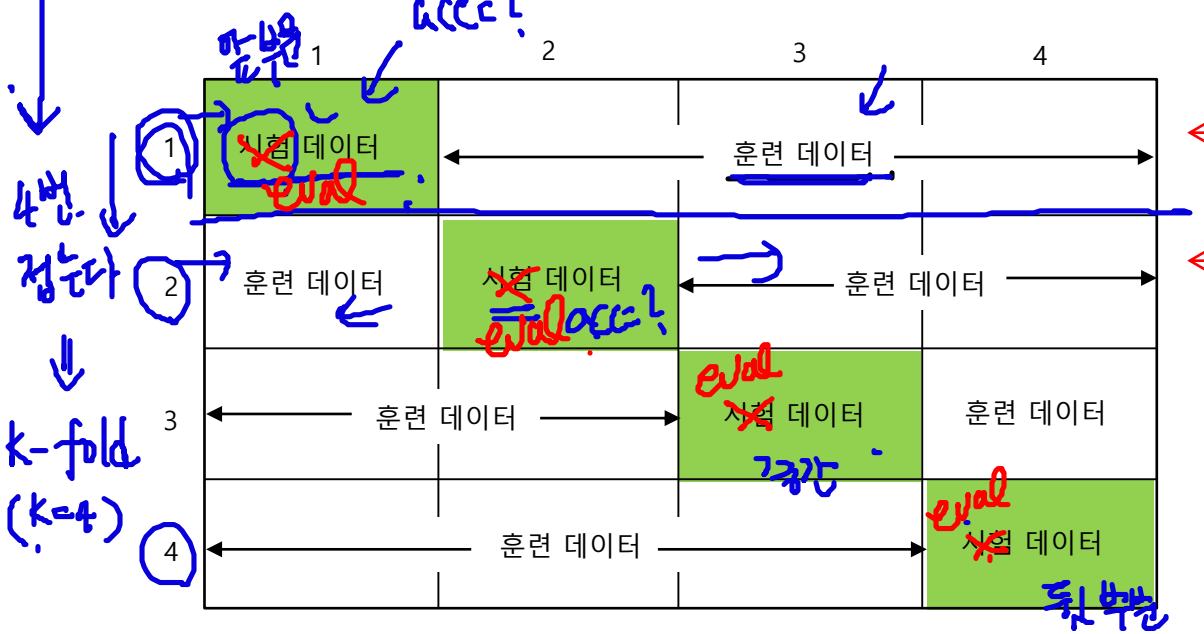
- 모형이 복잡해질수록 훈련 데이터에 대한 설명력은 증가함. (훈련 데이터의 오차 감소). 그러나, 학습 결과를 학습에 사용되지 않은 시험 데이터에 적용하면 오차가 감소하다가 다시 증가함. → Overfitting
- 시험 데이터의 오차가 최소가 되는 수준에서 모형의 복잡도를 결정함.



- 모형이 복잡해질수록 훈련 데이터에 대한 설명력은 증가함.
  → 학습 오차 감소 (Bias ^{2} 감소)
- Overfitting 되면, 데이터에 따라 결과의 편차가 커짐 (Variance 증가)
- 시험 데이터의 오차가 최소가 되는 수준에서 모형의 복잡도를 결정함.

Cross Validation (교차 검증) 시험

- 훈련 데이터로 학습한 후 동일한 훈련 데이터를 평가하면 당연히 좋은 성과가 나옴. 훈련 데이터로 학습한 결과를 학습에 전혀 사용되지 않은 시험 데이터에 적용해서 적용한 모형의 성과를 측정해야 함.
- 전체 데이터를 훈련 데이터 세트와 시험 데이터 세트로 구분함. 훈련 데이터와 시험 데이터가 골고루 분산되도록 구분함.
- K-fold Cross Validation : 전체 데이터를 K 구간으로 나눈 후 각 구간에서 시험 데이터 하나씩 할당함. 시험 데이터를 제외한 데이터 (훈련 데이터)로 학습하고, 제외했던 시험 데이터에 결과를 적용해서 정확도, 오류 율 등을 확인함.
- 아래 예시는 전체 데이터를 4 구간으로 나누어 Cross Validation을 적용한 것임.
- Cross Validation은 알고리즘 간 성능 비교에 사용될 수 있고, 동일 알고리즘 내에서도 정확도 평가를 위해 사용됨.



1번 구간에서는 첫 번째 부분을 시험 데이터로 사용하고 나머지 부분은 훈련 데이터로 사용함.

2번 구간에서는 두 번째 부분을 시험 데이터로 사용하고 나머지 부분은 훈련 데이터로 사용함.

Confusion Matrix 방식에 의한 분류 알고리즘의 성능 측정

- 훈련 데이터로 학습한 후 시험 데이터의 목적 패턴을 얼마나 정확하게 맞히는가를 계량화함.
- Confusion Matrix는 시험 데이터에 포함된 실제 목적 패턴 (실측 값)과 학습을 통해 예측된 패턴 (추정 값)으로 구성됨.
- TP (True Positive)는 실제 클래스가 Positive (+) 인데 Positive로 올바르게 분류한 개수임. TN (True Negative) 도 (-)를 (-)로 올바르게 분류한 개수임.
- FP (False Positive)와 FN (False Negative)은 잘못 분류한 개수임.
- 정확도 (Accuracy)는 전체 개수 중, 올바르게 분류한 비율을 측정한 것임.
- 정밀도 (Precision)와 리콜 (Recall)은 Positive 클래스가 Negative 클래스보다 중요하게 취급되는 경우의 측정 기준임.
- 매수 전략을 사용하는 경우 주가가 오를 것으로 예측했는데 떨어지는 경우가 (손실 발생), 떨어질 것으로 예측했는데 오른 경우보다 (거래없음) 더 좋지 않음. 이런 경우 Accuracy 보다 Precision이나 Recall을 이용할 수 있음.
- F-Measure는 Precision과 Recall을 종합하여 측정함 (가중조화평균). b=1 이면 Precision과 Recall을 동등하게 취급하고, b=0.5 이면 Precision을 더 강조함.

		Actual Value	
		Positive	Negative
Estimated Value	Positive	TP 4	FP 2
	Negative	FN 1	TN 3

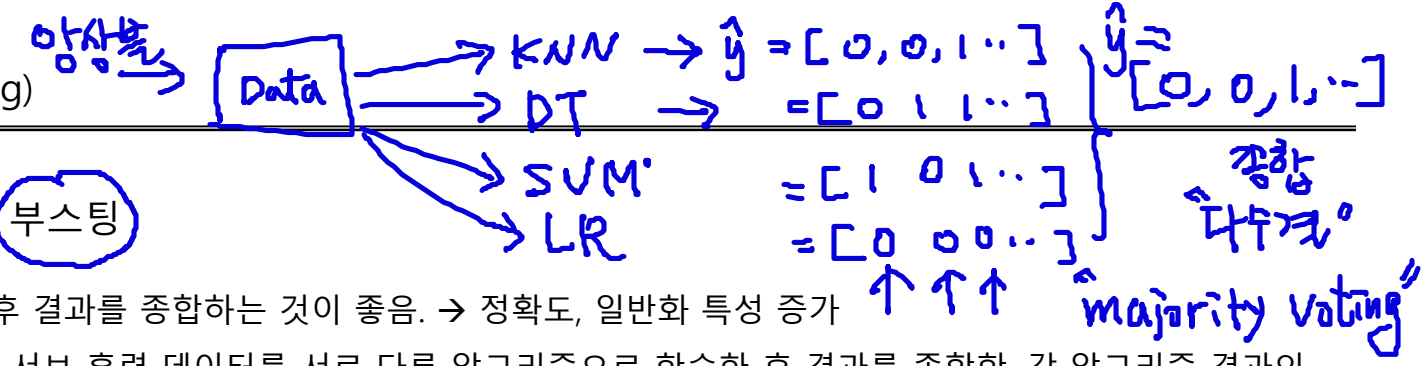
Precision

Recall

- 정확도 (Accuracy) = 정확히 예측한 개수 / 총 예측 개수
  = (TP + TN) / (TP + FP + TN + FN)
- 정밀도 (Precision) = TP / (TP + FP)
- 리콜 (Recall) = TP / (TP + FN)
- F_b-Measure = $(1+b^2)$ * (Precision * Recall) / (b^2 * Precision + Recall)

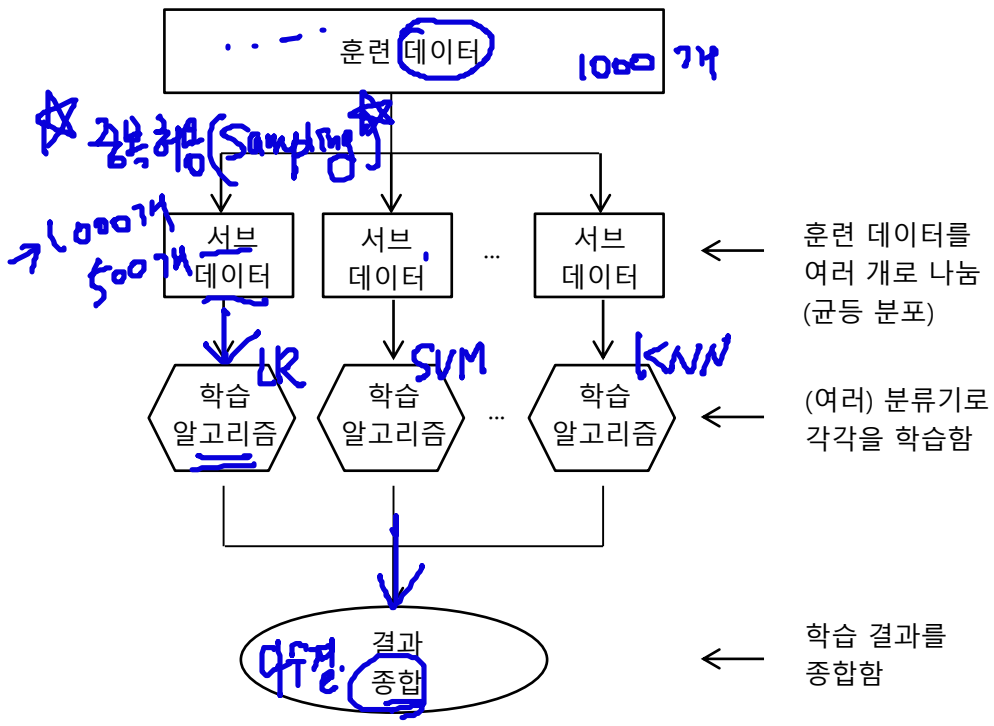
		Actual Value	
		Positive	Negative
Estimated Value	Positive	2,043	1,043
	Negative	1,676	2,247

- Accuracy = 정확히 예측한 개수 / 총 예측 개수
  = (2043 + 2247) / (2043 + 1043 + 1676 + 2247)
  = 61.29%
- Precision = 2,043 / (2,043 + 1,043) = 66.2 %
- Recall = 2,043 / (2,043 + 1,676) = 54.9%
- F_1-Measure = 2 * 0.662 * 0.549 / (0.662 + 0.549) = 60.02%



앙상블 기법 (Ensemble or Classifier Combination) : 배깅, 부스팅

- 단일 분류 알고리즘을 사용하는 것보다 여러 알고리즘을 사용한 후 결과를 종합하는 것이 좋음. → 정확도, 일반화 특성 증가
- 훈련 데이터를 여러 개의 서브 훈련 데이터로 나눈 후 (샘플링) 각 서브 훈련 데이터를 서로 다른 알고리즘으로 학습한 후 결과를 종합함. 각 알고리즘 결과의 다수결로 결정함.



배깅 (Bagging or Bootstrap Aggregation)

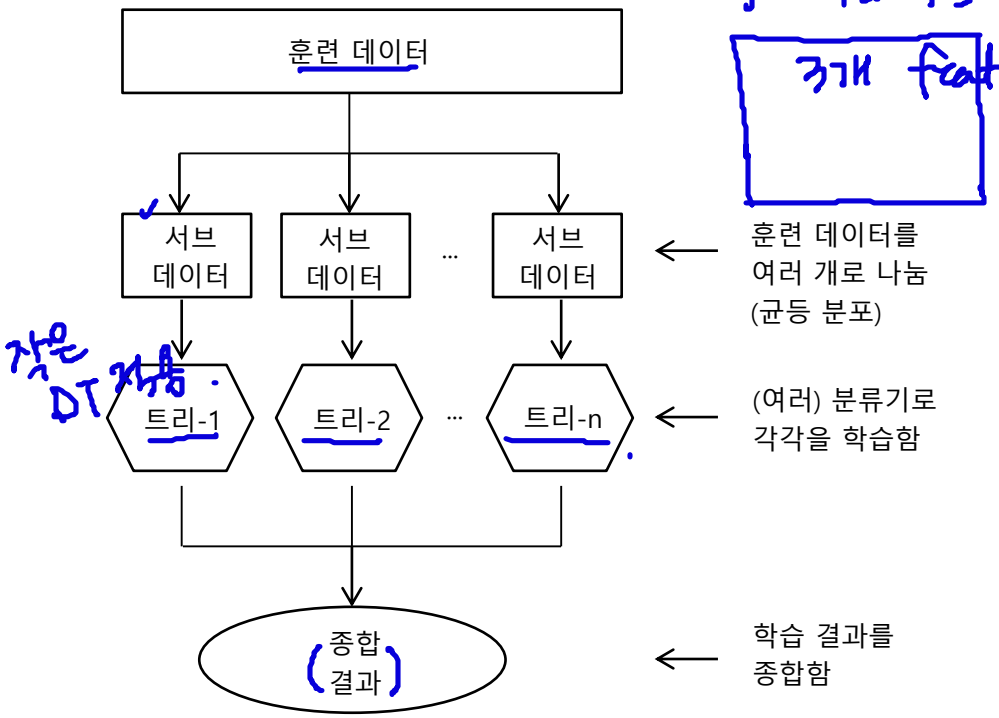
- (균일한 확률분포로) 훈련 데이터를 반복적으로 샘플링하여 서브 훈련 데이터를 만듦. (BootStrap : 단순복원 임의추출)
- 각각의 서브 훈련 데이터는 원본 훈련 데이터와 같은 크기를 갖음.
  → 데이터 중복을 허용함.
- 결과의 분산 (변동)이 감소하고 과잉적합 (overfit)이 방지된다.

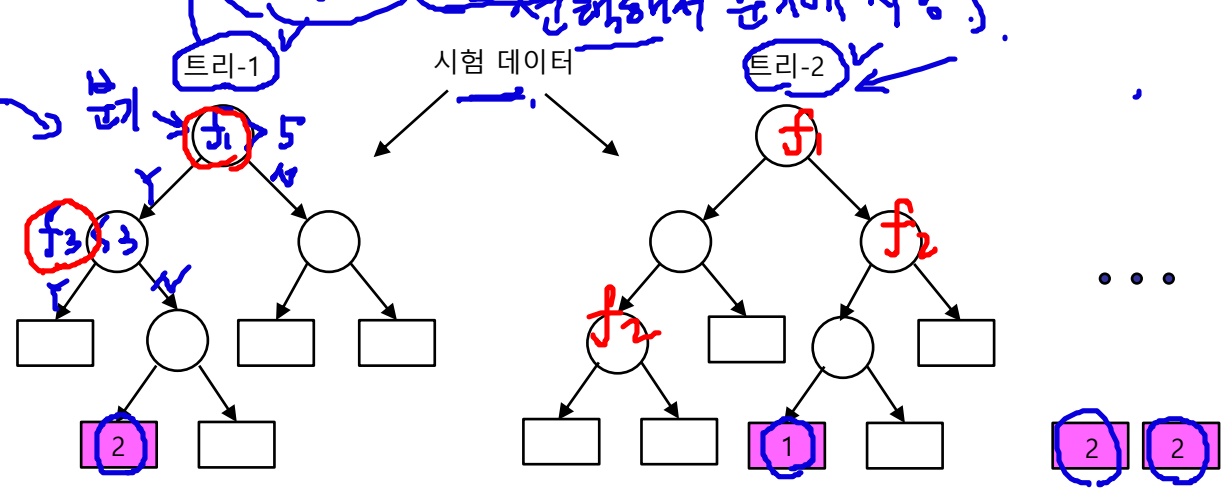
부스팅 (Boosting)

- 배깅과 마찬가지로 서브 훈련 데이터 샘플을 만듦.
- 처음 샘플링은 모든 데이터에 동일 가중치를 두어 샘플링함.
- 샘플링된 서브 훈련 데이터로 학습함.
- 분류가 잘못된 데이터의 가중치를 높이고 다시 샘플링함.
  → 잘못 분류된 패턴은 선택될 확률이 높아짐.
- 새로운 샘플링으로 다시 학습함. 이를 반복하면 점차 분류하기 어려운 패턴들이 많이 선택됨.
- 분류가 어려운 패턴에 더욱 집중하여 정확도를 높이는 방법.

앙상블 기법 (Ensemble or Classifier Combination) : 랜덤 포레스트 (Random Forest)

- 랜덤 포레스트는 Decision Tree를 위해 특별히 만들어진 앙상블 기법임.
- 랜덤 포레스트는 다수의 Decision Tree의 예측을 종합하여 분류함.
- 배깅과 동일하게 훈련 데이터에서 n-개의 서브 데이터를 샘플링하고 (중복 허용, 균등 분포), 각 서브 데이터마다 Decision Tree를 구축함.
- 서브 데이터는 훈련 데이터로부터 랜덤하게 추출되므로, 서브 트리들은 약간씩 다르게 구성됨.
- 약간씩 다른 트리의 결론을 종합하므로 분산이 감소하고 일반화 특성이 향상됨.





- 서브 데이터로 여래 개의 트리를 생성함. (서브 트리)
- 시험 데이터를 각 서브 트리에 적용하여 분류하고 다수의 트리가 분류하는 결과를 따름.
- 예) 트리-1 은 2, 트리-2는 1 이라고 분류하고, 트리-3,4 는 2라고 분류하면 최종으로 2라고 분류함. (다수결)

* 실습 파일 : 3-1.voteClassifier.py

Majority Voting 알고리즘 예시

- Majority Voting 알고리즘을 이용하여 iris 데이터를 분류한다.

```
# Majority voting의 앙상블 방법을 연습한다.
# ----------------------------------------
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier

# iris 데이터를 읽어온다.
iris = load_iris()

# Train 데이터 세트와 Test 데이터 세트를 구성한다
trainX, testX, trainY, testY = \
    train_test_split(iris['data'], iris['target'], test_size = 0.2)

# 4가지 모델을 생성한다 (KNN, Decision tree, SVM, Logistic Regression).
# 각 모델은 최적 조건으로 생성한다. (knn의 k개수, dtree의 max_depth 등)
# sklearn 문서 : Recommended for an ensemble of well-calibrated classifiers.
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
dtree = DecisionTreeClassifier(criterion='gini', max_depth=8)
svm = SVC(kernel='rbf', gamma=0.1, C=1.0, probability=True)
lreg = LogisticRegression(max_iter=500)

# 4가지 모델로 앙상블을 구성한다.
base_model = [('knn', knn), ('dtree', dtree), ('svm', svm), ('lr', lreg)]
ensemble = VotingClassifier(estimators=base_model, voting='soft')
```

```
# 4가지 모델을 각각 학습하고, 결과를 종합한다.
# VotingClassifier()의 voting = ('hard' or 'soft')에 따라 아래와 같이 종합한다.
# hard (default) : 4가지 모델에서 추정한 class = (0,1,2)중 가장 많은 것으로 판정.
# soft : 4가지 모델에서 추정한 각 class의 확률값의 평균 (혹은 합)을 계산한 후,
#        확률이 가장 높은 (argmax(P)) class로 판정한다.
# sklearn 문서 : If 'hard', uses predicted class labels for majority rule voting.
# Else if 'soft', predicts the class label based on the argmax of the sums of
# the predicted probabilities, which is recommended for an ensemble of
# well-calibrated classifiers.
ensemble.fit(trainX, trainY)

# 학습데이터와 시험데이터의 정확도를 측정한다.
print('\n학습데이터의 정확도 = %.2f' % ensemble.score(trainX, trainY))
print('시험데이터의 정확도 = %.2f' % ensemble.score(testX, testY))

# 시험데이터의 confusion matrix를 작성한다 (row : actual, col : predict)
predY = ensemble.predict(testX)
print('\nConfusion matrix :')
print(confusion_matrix(testY, predY))
```

결과 :

```
학습데이터의 정확도 = 0.98
시험데이터의 정확도 = 0.97

Confusion matrix :
[[11  0  0]
 [ 0  7  1]
 [ 0  0 11]]
```

* 실습 파일 : 3-2.Bagging.py

Bagging 알고리즘 예시

- Bagging 알고리즘을 이용하여 iris 데이터를 분류한다.

```
# Bagging에 의한 앙상블 방법을 연습한다.
# ------------------------------------
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
import numpy as np

# iris 데이터를 읽어온다.
iris = load_iris()

# Train 데이터 세트와 Test 데이터 세트를 구성한다
trainX, testX, trainY, testY = \
    train_test_split(iris['data'], iris['target'], test_size = 0.2)

# 4가지 모델을 생성한다 (KNN, Decision tree, SVM, Logistic Regression).
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
dtree = DecisionTreeClassifier(criterion='gini', max_depth=8)
svm = SVC(kernel='rbf', gamma=0.1, C=1.0, probability=True)
lreg = LogisticRegression(max_iter=500)

# 4가지 모델로 Bagging을 구성하고, 각 모델의 추정 확률을 누적한다.
prob = np.zeros((testY.shape[0], iris.target_names.shape[0]))
base_model = [knn, dtree, svm, lreg]
```

```
for m in base_model:
    bag = BaggingClassifier(base_estimator=m, n_estimators=100, bootstrap=True)
    bag.fit(trainX, trainY)

    prob += bag.predict_proba(testX)

# 확률의 누적합이 가장 큰 class를 찾고, 정확도를 측정한다.
predY = np.argmax(prob, axis=1)
accuracy = (testY == predY).mean()
print()
print("\n시험데이터의 정확도 = %.2f" % accuracy)

# 시험데이터의 confusion matrix를 작성한다 (row : actual, col : predict)
print('\nConfusion matrix :')
print(confusion_matrix(testY, predY))
```

결과 :

```
시험데이터의 정확도 = 0.97

Confusion matrix :
[[ 9  0  0]
 [ 0  8  0]
 [ 0  1 12]]
```

* 실습 파일 : 3-3.RandomForest.py

Random Forest 알고리즘 예시

- Random Forest 알고리즘을 이용하여 iris 데이터를 분류한다.

```
# RandomForest에 의한 앙상블 방법을 연습한다.
# ----------------------------------------
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# iris 데이터를 읽어온다.
iris = load_iris()

# Train 데이터 세트와 Test 데이터 세트를 구성한다
trainX, testX, trainY, testY = \
    train_test_split(iris['data'], iris['target'], test_size = 0.2)

rf = RandomForestClassifier(max_depth=5, n_estimators=100)
rf.fit(trainX, trainY)

# 학습데이터와 시험데이터의 정확도를 측정한다.
print('\n학습데이터의 정확도 = %.2f' % rf.score(trainX, trainY))
print('시험데이터의 정확도 = %.2f' % rf.score(testX, testY))

# 시험데이터의 confusion matrix를 작성한다 (row : actual, col : predict)
predY = rf.predict(testX)
print('\nConfusion matrix :')
print(confusion_matrix(testY, predY))
print()
print(classification_report(testY, predY, target_names=iris.target_names))
```

```
# Sub tree별 시험데이터의 정확도를 확인한다.
print('\nSubtree별 시험데이터 정확도 :')
for i in range(10):
    subTree = rf.estimators_[i]
    print('subtree (%d) = %.2f' % (i, subTree.score(testX, testY)))
```

```
학습데이터의 정확도 = 1.00
시험데이터의 정확도 = 0.97

Confusion matrix :
[[ 9  0  0]
 [ 0 11  0]
 [ 0  1  9]]

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         9
  versicolor       0.92      1.00      0.96        11
   virginica       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30


Subtree별 시험데이터 정확도 :
subtree (0) = 0.97
subtree (1) = 0.90
subtree (2) = 0.90
subtree (3) = 1.00
```

* 실습 파일 : 3-3.RandomForest.py

Random Forest 알고리즘 예시

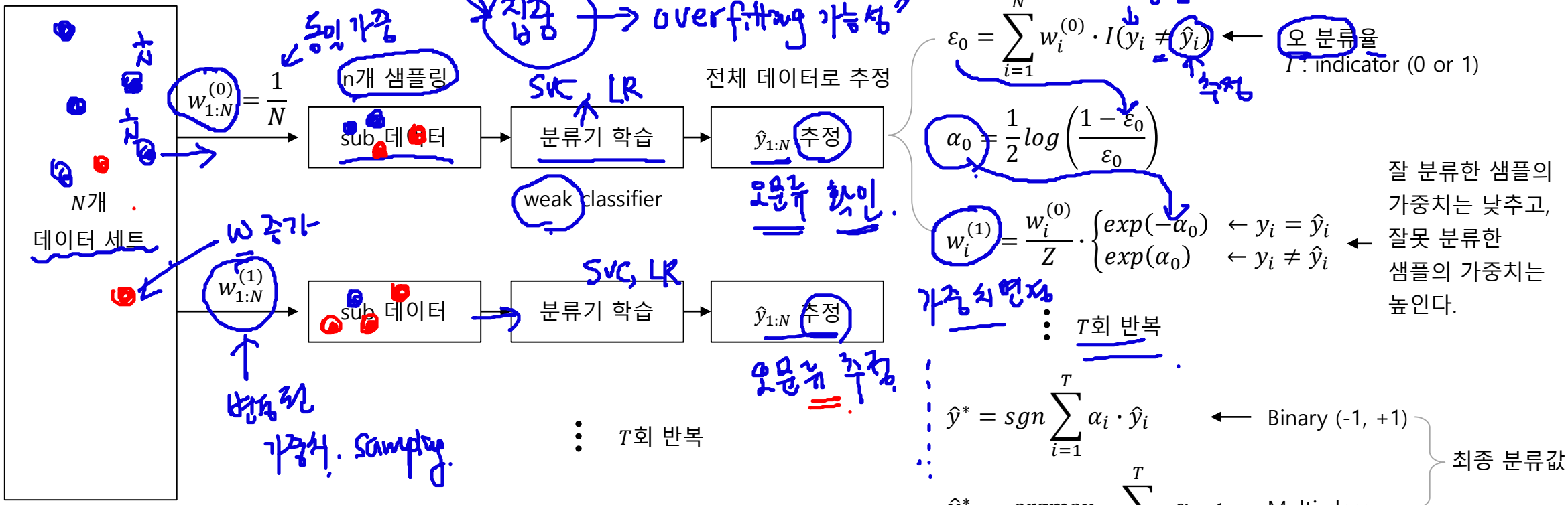
- Random Forest 알고리즘을 이용하여 iris 데이터를 분류한다.

```
# classification_report()를 해석해 본다.
import numpy as np
label = np.vstack([testY, predY]).T

# precision : class = n이라고 예측한 것 중 실제 classe=n인 비율
def precision(n):
    y = label[label[:, 1] == n]
    match = y[y[:, 0] == y[:, 1]]
    return match.shape[0] / y.shape[0]

print()
print('class-0 precision : %.2f' % precision(0))
print('class-1 precision : %.2f' % precision(1))
print('class-2 precision : %.2f' % precision(2))

# recall : 실제 class = n인 것중 classe=n으로 예측한 비율
def recall(n):
    y = label[label[:, 0] == n]
    match = y[y[:, 0] == y[:, 1]]
    return match.shape[0] / y.shape[0]

print()
print('class-0 recall : %.2f' % recall(0))
print('class-1 recall : %.2f' % recall(1))
print('class-2 recall : %.2f' % recall(2))

# F1-score (b=1) : precision과 recall의 가중조화평균
def f1_score(n):
    p = precision(n)
    r = recall(n)
    return 2 * p * r / (p + r)
```

```
print()
print('class-0 f1-score : %.2f' % f1_score(0))
print('class-1 f1-score : %.2f' % f1_score(1))
print('class-2 f1-score : %.2f' % f1_score(2))

결과 :

class-0 precision : 1.00
class-1 precision : 0.92
class-2 precision : 1.00

class-0 recall : 1.00
class-1 recall : 1.00
class-2 recall : 0.90

class-0 f1-score : 1.00
class-1 f1-score : 0.96
class-2 f1-score : 0.95

# classification_report() 결과

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         9
  versicolor       0.92      1.00      0.96        11
   virginica       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

AdaBoost (Adaptive Boosting) 알고리즘

- AdaBoost는 여러 개의 약한 분류기를 사용하고, 잘못 분류한 데이터 샘플을 더 많이 샘플링해서 정확도를 높이려는 알고리즘이다.
- 처음에는 모든 데이터를 대상으로 1/N 비율로 동등하게 샘플링해서 서브-데이터를 만든 후 특정 분류기 (ex : DT, SVM 등)를 학습한다.
- 학습이 완료된 분류기로 전체 데이터를 분류해 보고 오 분류율을 계산해서 샘플링 비율을 변경한다 (1/N → 새로운 비율). 이 때 잘 분류한 샘플의 비율은 낮추고, 잘못 분류한 샘플의 비율을 높인다. ← key point
- 다시 새로운 비율로 서브-데이터를 구성하고, 동일한 과정을 반복한다. 완료된 후에는 중간 결과의 가중합으로 최종 결과를 결정한다.



* 실습 파일 : 3-4.AdaBoost.py

AdaBoost 알고리즘 예시

- AdaBoost 알고리즘을 이용하여 iris 데이터를 분류한다.

```
# AdaBoost에 의한 앙상블 방법을 연습한다.
# ------------------------------------
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report

# iris 데이터를 읽어온다.
iris = load_iris()

# Train 데이터 세트와 Test 데이터 세트를 구성한다
trainX, testX, trainY, testY = \
    train_test_split(iris['data'], iris['target'], test_size = 0.2)

svm = SVC(kernel='rbf', gamma=0.1, C=1.0, probability=True)
aboost = AdaBoostClassifier(base_estimator=svm, n_estimators=100)
aboost.fit(trainX, trainY)

# 학습데이터와 시험데이터의 정확도를 측정한다.
print('\n학습데이터의 정확도 = %.2f' % aboost.score(trainX, trainY))
print('시험데이터의 정확도 = %.2f' % aboost.score(testX, testY))

# 시험데이터의 confusion matrix를 작성한다 (row : actual, col : predict)
predY = aboost.predict(testX)
print('\nConfusion matrix :')
print(confusion_matrix(testY, predY))
print()
print(classification_report(testY, predY, target_names=iris.target_names))
```

```
결과 :

학습데이터의 정확도 = 0.97
시험데이터의 정확도 = 0.93

Confusion matrix :
[[13  0  0]
 [ 0  9  0]
 [ 0  2  6]]

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.82      1.00      0.90         9
   virginica       1.00      0.75      0.86         8

    accuracy                           0.93        30
   macro avg       0.94      0.92      0.92        30
weighted avg       0.95      0.93      0.93        30
```