분류(Classification)

안 화 수

사이킷런(Scikit-Learn)

- 사이킷런

- 사이킷런 모듈 설치

1. python 에 설치

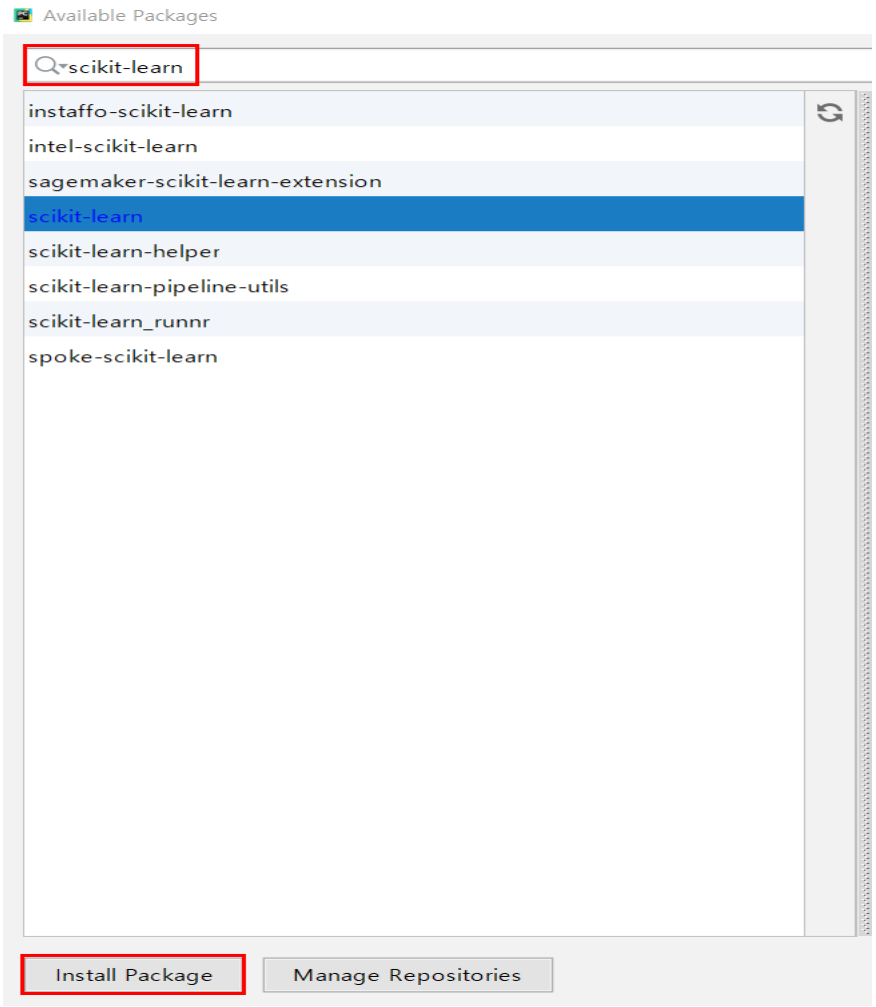
```
c:₩> pip install scikit-learn
```

2. anaconda 에 설치

```
c:₩> conda install scikit-learn
```

사이킷런(Scikit-Learn)

- PyCharm에 scikit-learn 설치



Scikit-Learn DataSet

사이킷런에 내장된 예제 데이터

사이킷런에는 별도의 외부 웹사이트에서 데이터 세트를 다운로드 받을 필요 없이 예제로 활용할 수 있는 간단하면서도 좋은 데이터 세트가 내장되어 있다. 이 데이터는 datasets 모듈에 있는 여러 API를 호출해 만들 수 있다.

<분류나 회귀 연습용 예제 데이터>

API명	설명
datasets.load_digits()	분류용, 0에서 9까지 숫자 이미지 픽셀 데이터셋
datasets.load_iris()	분류용, 붓꽃에 대한 피처를 가진 데이터셋
datasets.load_breast_cancer()	분류용, 위스콘신 유방암 피처들과 악성/음성 레이블
datasets.load_boston()	회귀용, 미국 보스톤 집 피처들과 가격 데이터셋
datasets.load_diabetes()	회귀용, 당뇨 데이터셋

분류(Classification)

- **분류**

분류(Classification)는 지도 학습의 한 영역으로, 학습 데이터를 학습한 후에 미지의 데이터를 분류하는 것을 의미한다.

< 분류 알고리즘 >

Types	Tasks	Algorithms
지도 학습 (Supervised Learning)	분류 (Classification)	KNN : K Nearest Neighbor
		SVM : Support Vector Machine
		Decision Tree (의사결정 나무)
		Logistic Regression

분류(Classification)

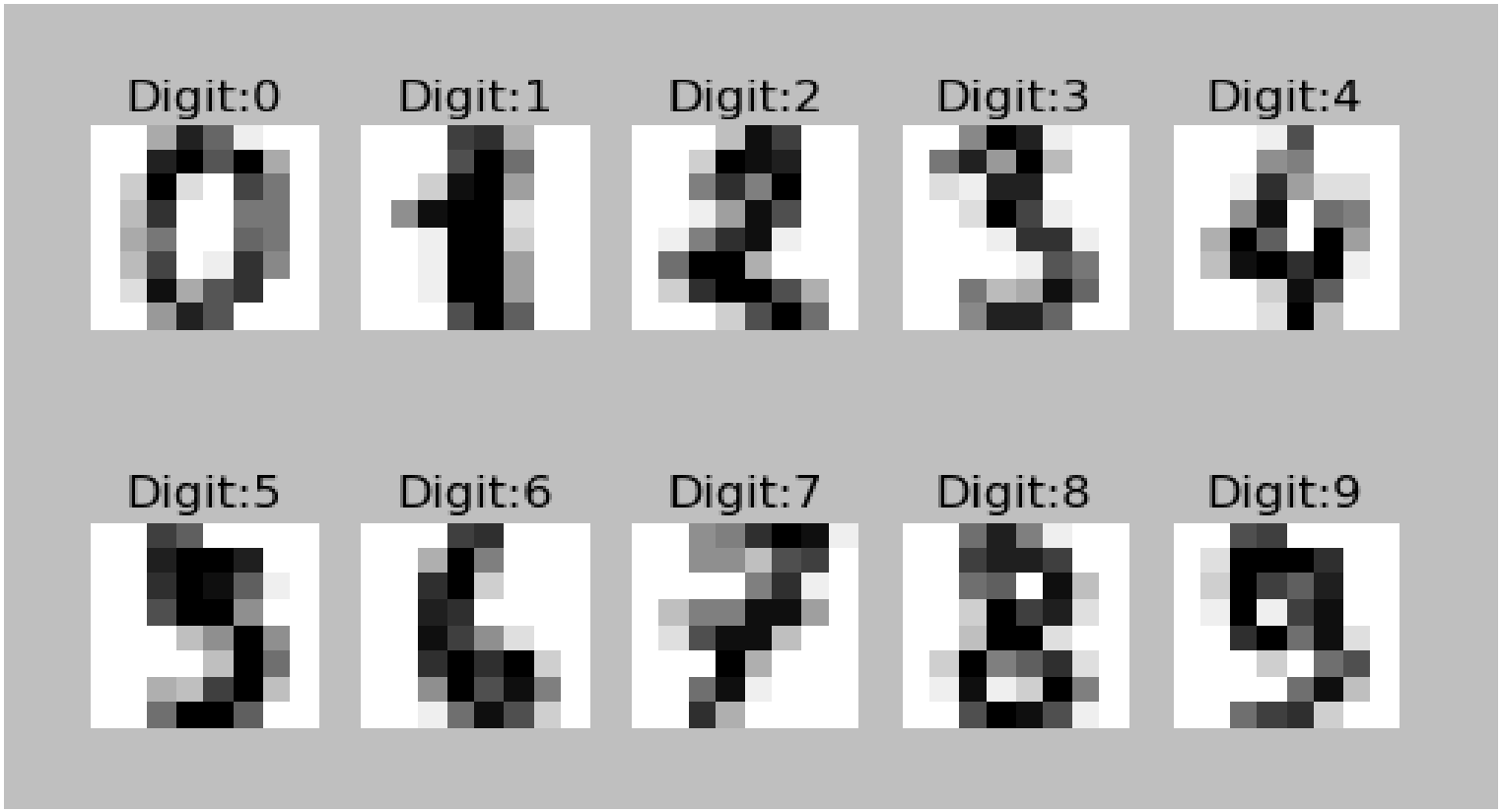
- **digits 데이터셋**
  digits 데이터셋은 0부터 9까지 손으로 쓴 숫자 이미지 데이터로 구성되어 있다.
  이미지 데이터는 8 x 8픽셀 흑백 이미지로, 1797장이 들어 있다.

digits.py

```
from sklearn import datasets
import matplotlib.pyplot as plt

# digits 데이터 로드
digits = datasets.load_digits()
print(digits)

# 0~9 이미지를 2행 5열로 출력
for label, img in zip(digits.target[:10], digits.images[:10]):
    plt.subplot(2,5, label+1)                    # 2행 5열로 이미지 배치
    plt.axis('off')
    plt.imshow(img, cmap=plt.cm.gray)    #그레이스케일(gray scale) 이미지(흑백 이미지)
    plt.title('Digit:{0}'.format(label))

plt.show()
```

분류(Classification)

- digits 데이터셋



분류(Classification)

❖ 분류기를 만들어 정답률 평가

scikit-learn 을 사용해 3과 8 이미지 데이터를 분류하는 분류기를 만든 후에 분류기의 성능을 테스트 해보자.

digits 데이터셋은 0부터 9까지 손으로 쓴 숫자 이미지는 8 x 8픽셀 흑백 이미지로, 1797장이 들어 있다. 이 중에서 숫자 3과 8 이미지는 357개 이미지로 되어 있다.

- 학습에 사용할 데이터
  - ✓ digits 데이터셋의 전체 이미지 개수 : 1797 개
  - ✓ 숫자 3과 8 이미지 갯수 : 357 개
  - ✓ 학습 데이터 개수 : 214 개 (3과 8 전체 이미지의 60%)
- 분류기 종류
  - ✓ 의사결정 나무 분류기(Decision Tree Classifier)

```
classifier = tree.DecisionTreeClassifier()
```

  - ✓ 분류기를 이용해서 학습

```
classifier.fit()
```

분류(Classification)

- **분류기를 만들어 정답률 평가**

classification.py (1 / 4)

```
import numpy as np
from sklearn import datasets

# 난수 시드 : 동일한 결과를 출력하기 위해서 설정
np.random.seed(0)

# 손으로 쓴 숫자 데이터 읽기
digits=datasets.load_digits()

# 3과 8의 데이터 위치를 구하기
flag_3_8=(digits.target==3)+(digits.target==8)
print(flag_3_8)                    # [False False False ...  True False  True]
```

분류(Classification)

- **분류기를 만들어 정답률 평가**

classification.py (2 / 4)

```
# 3과 8 이미지와 레이블을 구해서 변수에 저장
images = digits.images[flag_3_8]                # 이미지
labels = digits.target[flag_3_8]                # 레이블
print(images.shape)                             # (357, 8, 8)
print(labels.shape)                             # (357,)

# 3과 8의 이미지 데이터를 2차원에서 1차원으로 변환
# reshape(images.shape[0],-1) : 열에 -1은 가변적이라는 의미
images = images.reshape(images.shape[0],-1)
print('reshape:',images.shape)                  # reshape: (357, 64)
```

분류(Classification)

- **분류기를 만들어 정답률 평가**

classification.py (3 / 4)

```
# 결정 트리(decision tree)알고리즘을 사용해 분류기를 만들어서 학습
from sklearn import tree

# 학습 데이터와 테스트 데이터를 분할 ( 6 : 4 비율 )
n_samples = len(flag_3_8[flag_3_8])          # 3과 8 전체 데이터 갯수
print(n_samples)                             # 357개
train_size = int(n_samples*3/5)              # 학습 데이터 갯수
print(train_size)                            # 214개

# 결정 트리 분류기 모델 생성
classifier = tree.DecisionTreeClassifier()

# 모델 학습
# 학습 데이터는 손으로 쓴 숫자의 전체 이미지 데이터 중 60%를 사용해서 학습
classifier.fit(images[:train_size], labels[:train_size])
```

분류(Classification)

- **분류기를 만들어 정답률 평가**

classification.py (4 / 4)

```
# 분류기의 성능 평가
from sklearn import metrics                    # 성능 평가 모듈 import

test_label=labels[train_size:]                 # 테스트 데이터 레이블을 구함
print('test_label:', test_label)

# 테스트 데이터 이용해서 예측 레이블을 구함
predict_label=classifier.predict(images[train_size:])
print('predict_label:', predict_label)

# 정답률 계산
# scikit-learn 에서는 accuracy_score() 함수로 정답률을 계산함.
print('정답률(Accuracy):', metrics.accuracy_score(test_label, predict_label))
```

 # 정답률(Accuracy): 0.8741258741258742

Confusion matrix

- **혼동행렬(confusion matrix)**

분류기의 성능 지표를 확인하는 방법하는 방법을 알아보자.
Positive와 Negtive 중 하나를 반환하는 분류기를 살펴보자. 이때 Positive와 Negtive 각각에서 정답(True)과 오답(False)이 있기 때문에 조합이 4개가 생긴다. 이 경우의 수 로 만들어진 행렬을 **혼동행렬(confusion matrix)** 이라고 하며, 분류기 평가에서 자주 사용한다.

Confusion matrix

혼동행렬(confusion matrix)

		예측	
		Positive	Negtive
실제	Positive	True Positive (TP)	False Negtive (FN)
	Negtive	False Positive (FP)	True Negtive (TN)

'confusion matrix'에서
'True/False'는 실제 값이 1이냐 0이냐를 맞췄는지를 나타냄
즉, 'True'는 실제와 예측이 일치하는경우 이며, 'False'는 실제와 예측이 불일치하는 경우를 말함
'Positive/Negative'는 예측한 값이 1이냐 0이냐를 의미함

TP의 경우 1라고 예측했는데, 실제로도 1인 경우 **(정답)**
TN의 경우 0라고 예측했고, 실제값 또한 0인 경우 **(정답)**

FP의 경우 1라고 예측했으나, 실제값은 0인 경우,
FN의 경우 0라고 예측했는데, 실제로는 1인 경우,

Confusion matrix

- 알고리즘 성능 평가에 사용.

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

- TP : True로 예측하고 실제 값도 True
- TN : False로 예측하고 실제 값도 False
- FP : True로 예측하고 실제는 False
- FN : False로 예측하고 실제는 True

분류기의 성능평가

- **분류기의 성능평가 지표**

  분류기의 성능을 평가하는 지표로는 **정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F 값** 등이 있다.

분류기의 성능평가 지표

- **1. 정확도(Accuracy) :** 전체 예측에서 정답이 있는 비율(전체 중에서 올바르게 예측한 것이 몇 개인가)
  정확도 (Accuracy) = (TP + TN) / (TP + FP + FN + TN)

- **2. 정밀도(Precision) :** 분류기가 Positive로 예측했을 때 진짜로 Positive한 비율
  Positive로 예측하는 동안 어느 정도 맞았는지, 정확도가 높은지를
  나타내는 지표 (내가 푼 문제 중에서 맞춘 정답 개수)
  정밀도 (Precision) = TP / (TP + FP)

- **3. 재현율(Recall) :** 진짜로 Positive인 것을 분류기가 얼마나 Positive라고 예측했는지 나타내는 비율
  (전체 중에서 내가 몇 개를 맞췄는가)
  실제로 Positive인 것 중에서 어느 정도 검출할 수 있었는지 가늠하는 지표
  재현율 (Recall) = TP / (TP + FN)

- **4. F값(F-measure) :** 적합률과 재현율의 조화 평균. 지표 2개를 종합적으로 볼 때 사용
  F값이 높을수록 분류 모형의 예측력이 좋다고 할 수 있다.
  F값(F-measure) = 2 x Precision x Recall / (Precision + Recall)

일반적으로 분류기의 성능을 이야기 할 때, 정확도(Accuracy)를 보지만 그것만으로 충분하지 않을 경우에 다른 성능평가 지표를 같이 살펴봐야 된다.

분류기의 성능평가

- **분류기를 만들어 성능 평가 : 정답률, 혼동행렬, 적합률, 재현율, F값**

classification_validate.py (1 / 4)

```
import numpy as np
from sklearn import datasets

# 손으로 쓴 숫자 데이터 읽기
digits=datasets.load_digits()

# 3과 8의 데이터 위치를 구하기
flag_3_8=(digits.target==3)+(digits.target==8)
print(flag_3_8)

# 3과 8의 데이터를 구하기
# 3과 8 이미지와 레이블을 꺼내 변수에 저장
images = digits.images[flag_3_8]          # 이미지
labels = digits.target[flag_3_8]          # 레이블

# 3과 8의 이미지 데이터를 2차원에서 1차원으로 변환
# reshape(images.shape[0],-1) : 열에 -1은 가변적이라는 의미
images = images.reshape(images.shape[0],-1)
```

분류기의 성능평가

- **분류기를 만들어 성능 평가 : 정답률, 혼동행렬, 적합률, 재현율, F값**

classification_validate.py (2 / 4)

```
# 결정 트리(decision tree)알고리즘을 사용해 분류기를 만들어서 학습
from sklearn import tree

# 분류기 생성
n_samples = len(flag_3_8[flag_3_8])          # 3과 8 전체 데이터 갯수
print(n_samples)                             # 357개
train_size = int(n_samples*3/5)              # 학습 데이터 갯수
print(train_size)                            # 214개
classifier = tree.DecisionTreeClassifier()

# classifier.fit()함수로 분류기에 학습 데이터를 학습시킨다.
# 학습 데이터는 손으로 쓴 숫자의 전체 이미지 데이터 중 60%를 사용해서
# 학습 시킨다.
classifier.fit(images[:train_size], labels[:train_size])
```

분류기의 성능평가

- **분류기를 만들어 성능 평가 : 정답률, 혼동행렬, 적합률, 재현율, F값**

classification_validate.py (3 / 4)

```
# 분류기의 성능 평가
from sklearn import metrics              # 성능 평가 모듈 import

expected=labels[train_size:]             # 테스트 데이터의 레이블을 구함
print('expected:', expected)

# 테스트 데이터 이용해서 예측 레이블을 구함
predicted=classifier.predict(images[train_size:])
print('predicted:', predicted)


# 정답률을 계산하고 출력
# scikit-learn 에서는 accuracy_score() 함수로 정답률을 계산함.
print('정답률(Accuracy):', metrics.accuracy_score(expected, predicted))
```

분류기의 성능평가

- **분류기를 만들어 성능 평가 : 정답률, 혼동행렬, 적합률, 재현율, F값**

classification_validate.py (4 / 4)

```
# 정답률, 혼동행렬, 적합률, 재현율, F값을 계산하고 출력
# accuracy_score() 함수로 정답률을 계산함.
# confusion_matrix() 함수로 혼동행렬을 계산함.
# precision_score() 함수로 적합률을 계산함.
# recall_score() 함수로 재현율을 계산함.
# f1_score() 함수로 F값을 계산함.

print('정답률(Accuracy):', metrics.accuracy_score(expected, predicted))

print('혼동행렬(Confusion matrix):', metrics.confusion_matrix(expected, predicted))

print('적합률(Precision):', metrics.precision_score(expected, predicted, pos_label=3))

print('재현율(Recall):', metrics.recall_score(expected, predicted, pos_label=3))

print('F값(F-measure):', metrics.f1_score(expected, predicted, pos_label=3))
```

분류기의 성능평가 지표

혼동행렬(confusion matrix) : metrics.confusion_matrix()

정답률 (Accuracy) : metrics.accuracy_score()

적합률 (Precision) : metrics.precision_score()

재현율 (Recall) : metrics.recall_score()

F값(F-measure) : metrics.f1_score()

모든 평가지표 : metrics.classification_report()

분류기(Classifier)

- 분류기의 종류
  - 결정 트리 (Decision Tree)
  - 랜덤 포레스트 (Random Forest)
  - 에이다부스트 (AdaBoost)
  - 서포트 벡터 머신 (Support Vector Machine)

결정 트리(Decision Tree)

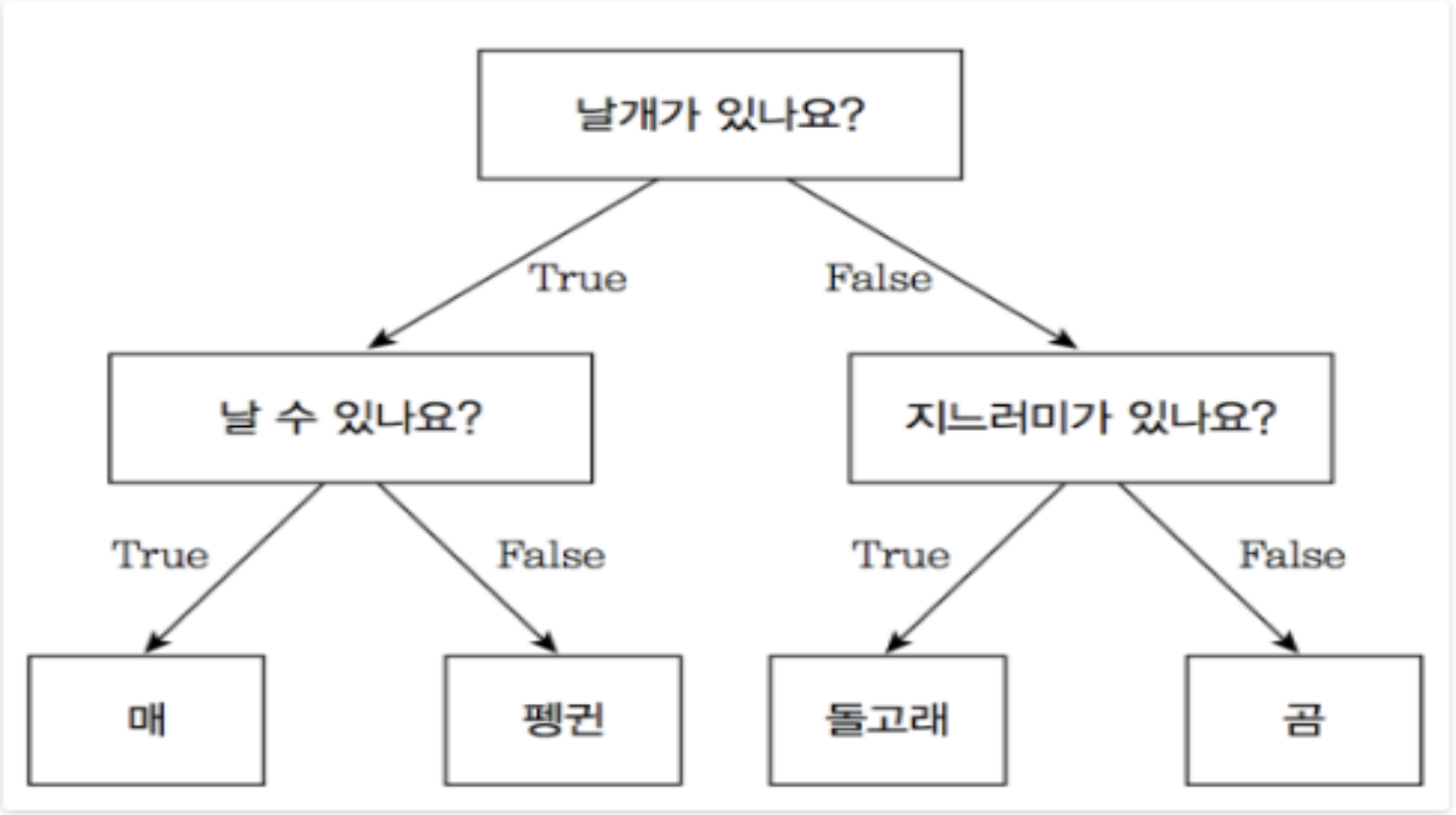
결정 트리(Decision Tree)

1. 결정 트리는 데이터를 여러 등급으로 분류하는 지도 학습 중의 하나로, 트리 구조를 이용한 분류 알고리즘이다.
2. 결정 트리 학습에서는 학습 데이터에서 트리 모델을 생성한다. 무엇을 기준으로 분기할 지에 따라 결정 트리는 몇 가지 방법으로 분류할 수 있다.
3. 결정 트리의 장점은 분류 규칙을 트리 모델로 가시화할 수 있어, 분류 결과의 해석이 비교적 용이하다는 점이다.
4. 생성한 분류 규칙도 편집할 수 있으며, 학습을 위한 계산 비용이 낮다는 점도 장점이다.
5. 결정 트리는 과적합 하는 경향이 있고, 취급하는 데이터의 특성에 따라 트리 모델을 생성하기 어렵다는 단점도 있다.
6. 결정 트리는 과적합 하는 경향이 있어 결정 트리 단독으로 사용하지 않고, 앙상블 학습을 조합해서 사용하는 경우가 많다.

결정 트리(Decision Tree)

결정 트리(Decision Tree)

<몇가지 동물들을 구분하기 위한 결정트리>



결정 트리(Decision Tree)

- **결정 트리(Decision Tree)**
  DecisionTree.py (1 / 3)

```
import matplotlib.pyplot as plt
from sklearn import datasets, tree
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# 손으로 쓴 숫자 데이터 읽기
digits = datasets.load_digits()

# 이미지를 2행 5열로 표시
for label, img in zip(digits.target[:10], digits.images[:10]):
    plt.subplot(2, 5, label + 1)
    plt.axis('off')
    plt.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Digit: {0}'.format(label))
plt.show()
```

결정 트리(Decision Tree)

- **결정 트리(Decision Tree)**
  DecisionTree.py (2 / 3)

```
# 3과 8의 데이터 위치를 구하기
flag_3_8 = (digits.target == 3) + (digits.target == 8)

# 3과 8의 데이터를 구하기
images = digits.images[flag_3_8]
labels = digits.target[flag_3_8]

# 3과 8의 이미지 데이터를 1차원으로 변환
images = images.reshape(images.shape[0], -1)

# 분류기 생성
n_samples = len(flag_3_8[flag_3_8])
train_size = int(n_samples * 3 / 5)
# 트리 모델의 최대깊이를 max_depth=3로 설정
classifier = tree.DecisionTreeClassifier(max_depth=3)
classifier.fit(images[:train_size], labels[:train_size])
```

결정 트리(Decision Tree)

- **결정 트리(Decision Tree)**

  DecisionTree.py (3 / 3)

```
# 분류기 성능을 확인
expected = labels[train_size:]
predicted = classifier.predict(images[train_size:])

print('정답률(Accuracy):', accuracy_score(expected, predicted))
print('혼돈행렬(Confusion matrix):', confusion_matrix(expected, predicted))
print('적합률(Precision):', precision_score(expected, predicted, pos_label=3))
print('재현율(Recall):', recall_score(expected, predicted, pos_label=3))
print('F값(F-measure):', f1_score(expected, predicted, pos_label=3))
```

랜덤 포레스트(Random Forest)

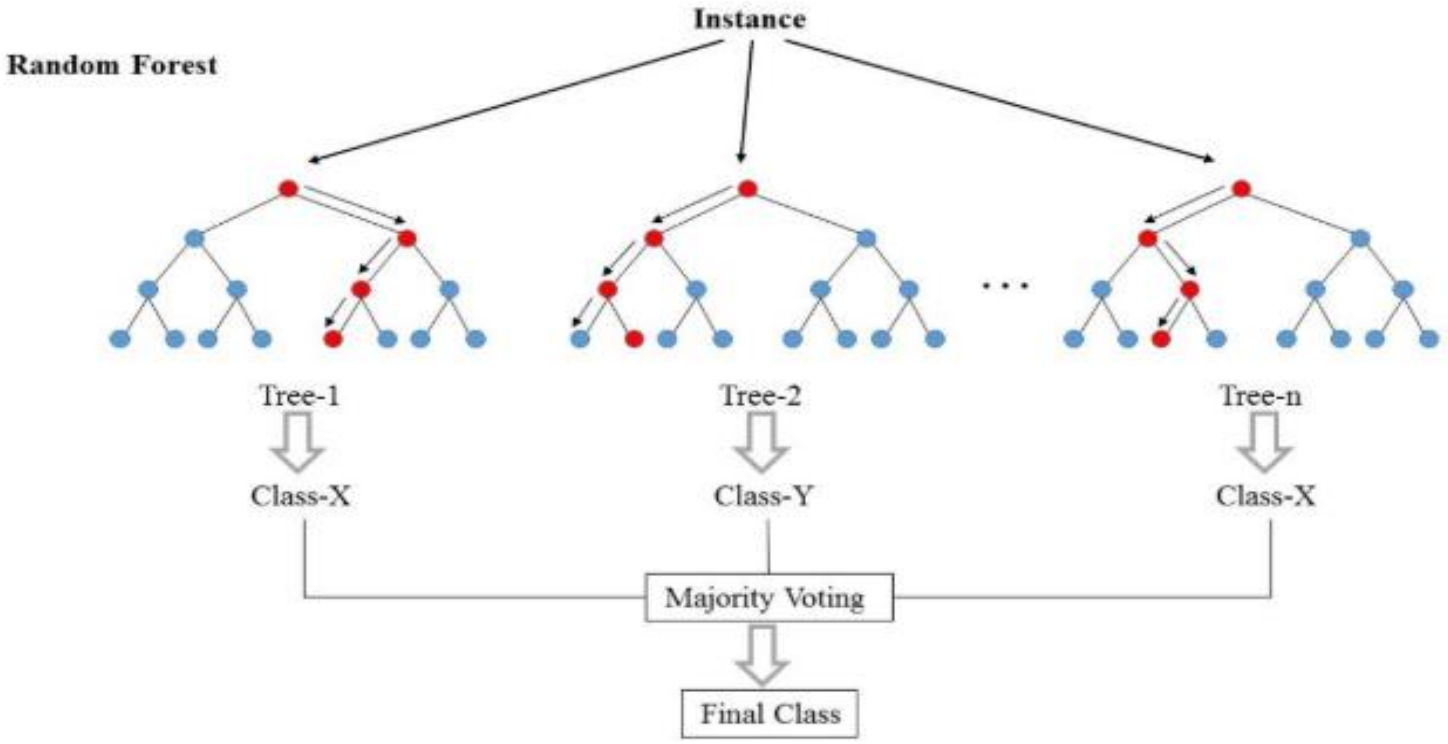
- **랜덤 포레스트(Random Forest)**

1. 랜덤 포레스트(Random Forest)는 앙상블 학습의 배깅(bagging)으로 분류되는 알고리즘이다.
2. 전체 학습 데이터 중에서 중복이나 누락을 허용해 학습 데이터셋을 여러 개 추출하여, 그 일부의 속성을 사용해 결정 트리(약한 학습기)를 생성한다.
3. 랜덤 포레스트는 학습과 판별을 빠르게 처리하고, 학습 데이터의 노이즈에도 강하다는 장점이 있다.
4. 랜덤 포레스트는 분류 외에도 회귀나 클러스터링에도 사용 할 수 있다.
5. 학습 데이터의 개수가 적을 경우에는 과적합이 발생하기 때문에, 학습 데이터 적은 경우에는 사용하지 않는 것이 좋다.

랜덤 포레스트(Random Forest)

- **랜덤 포레스트(Random Forest)**

  간단히 설명하면 train data를 무작위로 sampling해서 만든 다수의 Decision tree를 기반으로 다수결로 결과를 추출한다.



앙상블 학습

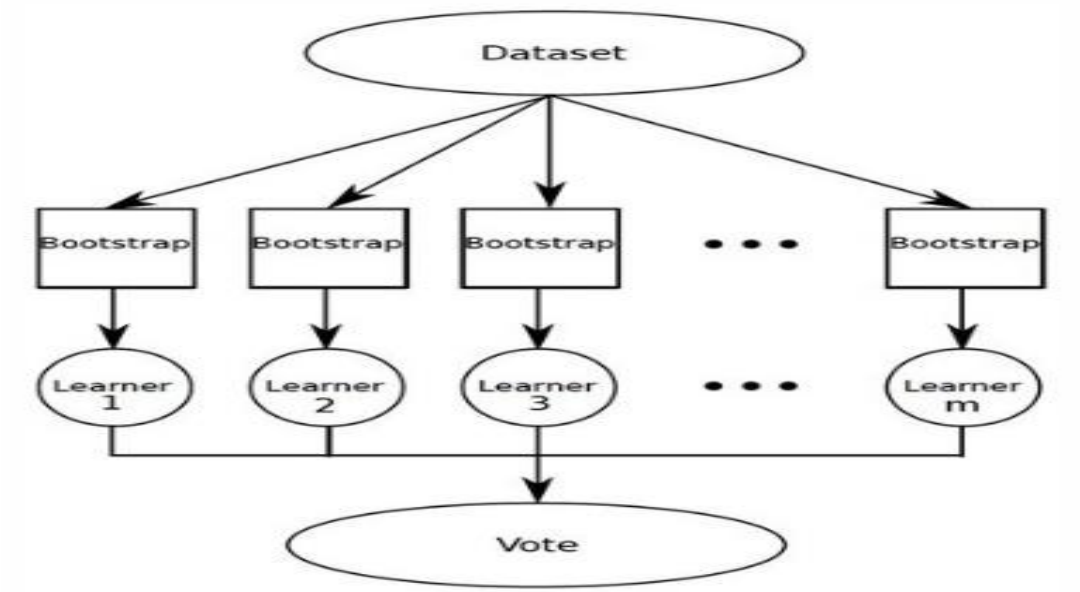
- **앙상블 학습(Ensenbles)**

1. 앙상블 학습은 몇 가지 성능이 낮은 분류기(약한 학습기)를 조합해 성능이 높은 분류기를 만드는 방법이다.
2. 약한 학습기 알고리즘은 정해진 것이 없으므로 적절히 선택해서 사용해야 한다.
3. 앙상블 학습 결과는 약한 학습기의 결과 값 중 다수결로 결정한다.
4. 앙상블 학습은 약한 학습기 생성 방법에 따라 배깅(bagging)과 부스팅(boosting)으로 나눌 수 있다.

앙상블 학습

배깅(bagging)

1. 학습 데이터를 빼고 중복을 허용 해 그룹 여러 개로 분할하고 학습 데이터의 그룹마다 약한 학습기를 생성하는 방법이다.
2. 배깅으로 여러 그룹으로 분할한 학습 데이터 그룹에서 약한 학습기를 각각 생성하고, 약한 학습기를 조합해 성능 좋은 분류기를 만들 수 있다.



앙상블 학습

- **부스팅(boosting)**

  약한 학습기를 여러 개 준비하고 가중치가 있는 다수결로 분류하는 방법이다.
  그 가중치도 학습에 따라 결정한다.
  난이도가 높은 학습 데이터를 올바르게 분류할 수 있는 약한 학습기의 판별 결과를 중시하도록 가중치를 업데이트해 나간다.

랜덤 포레스트(Random Forest)

- **랜덤 포레스트(Random Forest)**
  RandomForest.py (1 / 3)

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import ensemble
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# 손으로 쓴 숫자 데이터 읽기
digits = datasets.load_digits()

# 이미지를 2행 5열로 표시
for label, img in zip(digits.target[:10], digits.images[:10]):
    plt.subplot(2, 5, label + 1)
    plt.axis('off')
    plt.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Digit: {0}'.format(label))
plt.show()
```

랜덤 포레스트(Random Forest)

- **랜덤 포레스트(Random Forest)**
  RandomForest.py (2 / 3)

```
# 3과 8의 데이터 위치를 구하기
flag_3_8 = (digits.target == 3) + (digits.target == 8)

# 3과 8의 데이터를 구하기
images = digits.images[flag_3_8]
labels = digits.target[flag_3_8]

# 3과 8의 이미지 데이터를 1차원으로 변환
images = images.reshape(images.shape[0], -1)

# 분류기 생성
n_samples = len(flag_3_8[flag_3_8])
train_size = int(n_samples * 3 / 5)

# n_estimators는 약한 학습기 갯수로 20을 지정, max_depth는 트리모델의 최대깊이,
classifier = ensemble.RandomForestClassifier(n_estimators=20, max_depth=3)
classifier.fit(images[:train_size], labels[:train_size])
```

랜덤 포레스트(Random Forest)

- **랜덤 포레스트(Random Forest)**

  RandomForest.py (3 / 3)

```
# 분류기 성능을 확인
expected = labels[train_size:]
predicted = classifier.predict(images[train_size:])

print('Accuracy:', accuracy_score(expected, predicted))
print('Confusion matrix:', confusion_matrix(expected, predicted))
print('Precision:', precision_score(expected, predicted, pos_label=3))
print('Recall:', recall_score(expected, predicted, pos_label=3))
print('F-measure:', f1_score(expected, predicted, pos_label=3))
```

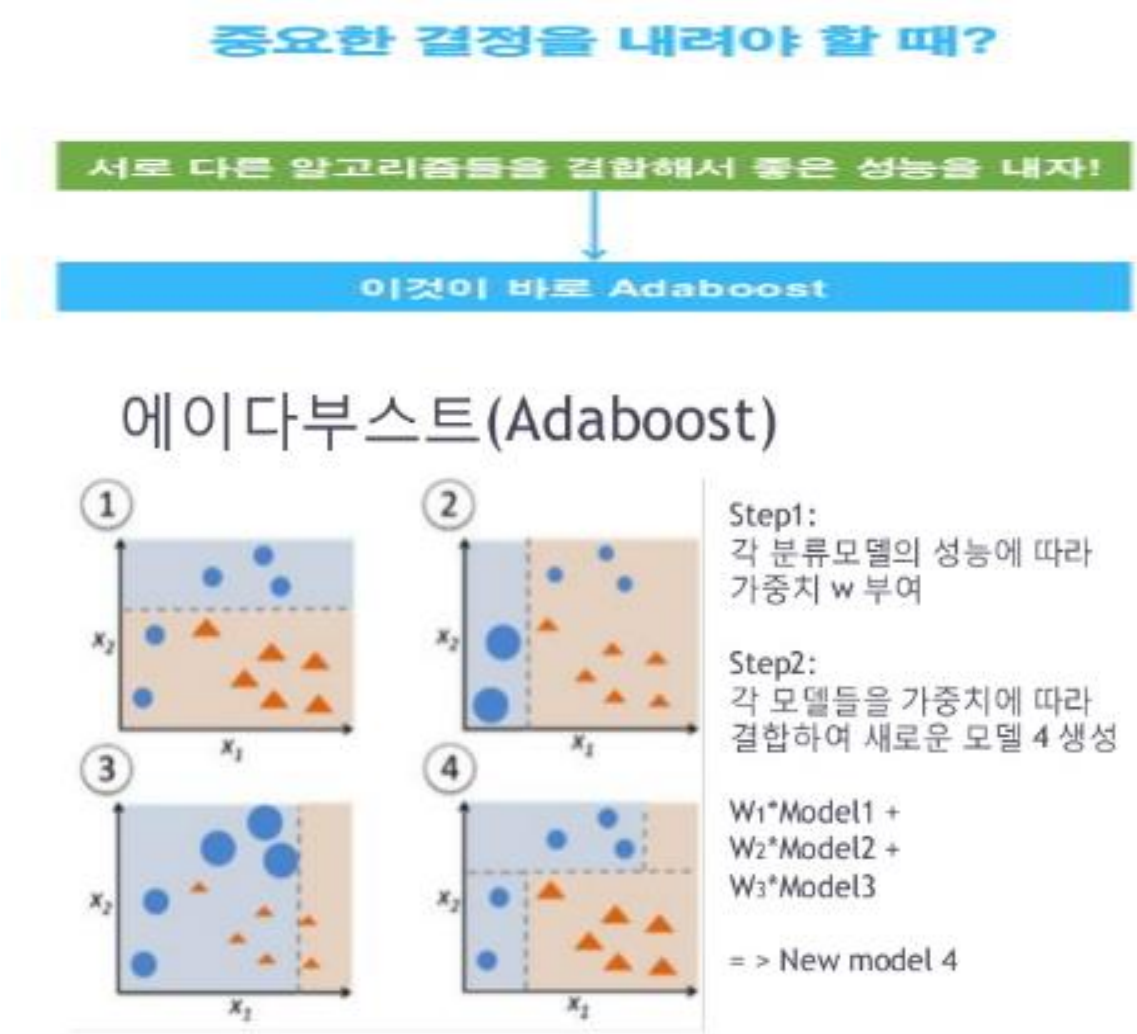
에이다부스트(AdaBoost)

- **에이다부스트(AdaBoost)**

1. 에이다부스트는 앙상블 학습의 부스팅(boosting)으로 분류하는 알고리즘이다.
2. 에이다부스트에서는 난이도가 높은 데이터를 제대로 분류할 수 있는 약한 학습기(weak learner)의 분류 결과를 중시하므로 약한 학습기에 가중치를 준다.
3. 난이도가 높은 학습 데이터와 성능이 높은 약한 학습기에 가중치를 주어서 정확도를 높인다.
4. 에이다부스트는 분류 정밀도가 높지만, 학습 데이터의 노이즈에 쉽게 영향을 받는다.

에이다부스트(AdaBoost)

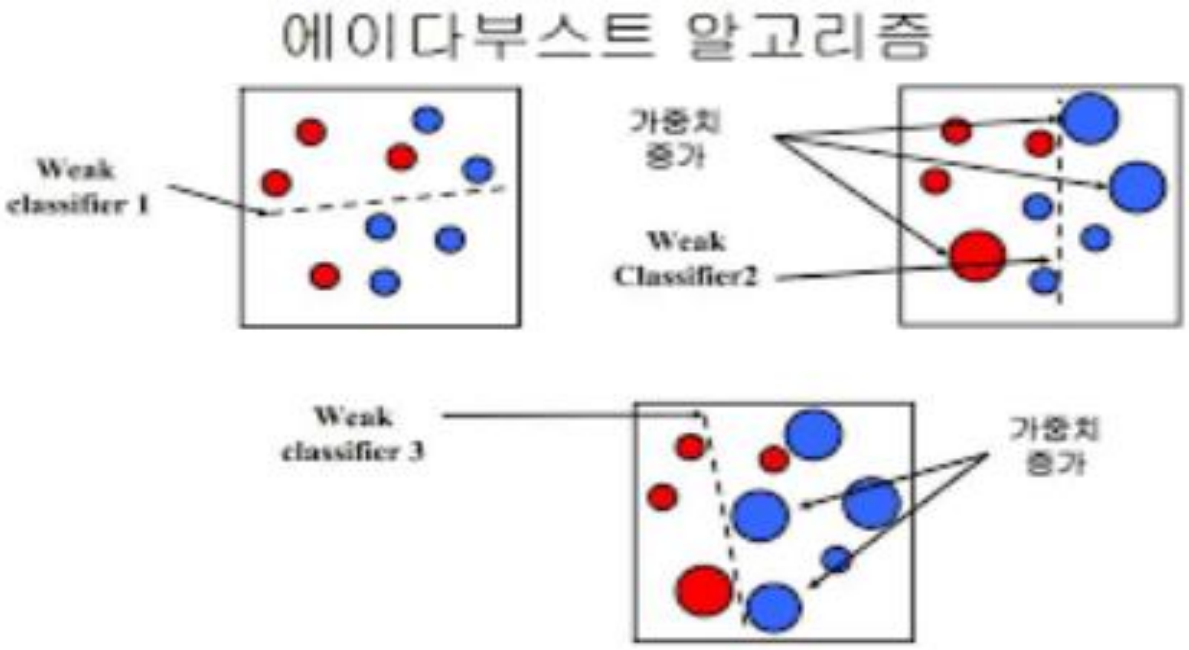
- 에이다부스트(AdaBoost)



에이다부스트(AdaBoost)

에이다부스트(AdaBoost)

- 각기 다른 데이터에 강점을 가지는 학습기를 여러개 구축
- 이전 학습기의 에러에 따라 데이터에 부여되는 가중치 조정



에이다부스트(AdaBoost)

- **에이다부스트(AdaBoost)**
  AdaBoost.py (1 / 3)

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import tree, ensemble
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# 손으로 쓴 숫자 데이터 읽기
digits = datasets.load_digits()

# 이미지를 2행 5열로 표시
for label, img in zip(digits.target[:10], digits.images[:10]):
    plt.subplot(2, 5, label + 1)
    plt.axis('off')
    plt.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Digit: {0}'.format(label))
plt.show()
```

에이다부스트(AdaBoost)

- **에이다부스트(AdaBoost)**
  AdaBoost.py (2 / 3)

```
# 3과 8의 데이터 위치를 구하기
flag_3_8 = (digits.target == 3) + (digits.target == 8)

# 3과 8의 데이터를 구하기
images = digits.images[flag_3_8]
labels = digits.target[flag_3_8]

# 3과 8의 이미지 데이터를 1차원으로 변환
images = images.reshape(images.shape[0], -1)

# 분류기 생성
n_samples = len(flag_3_8[flag_3_8])
train_size = int(n_samples * 3 / 5)

# base_estimator는 약한 학습기를 지정하는 파라미터로 여기서는 결정트리를 지정,
# n_estimators는 약한 학습기 갯수를 지정
classifier = ensemble.AdaBoostClassifier(base_estimator=
                        tree.DecisionTreeClassifier(max_depth=3), n_estimators=20)
classifier.fit(images[:train_size], labels[:train_size])
```

에이다부스트(AdaBoost)

- **에이다부스트(AdaBoost)**

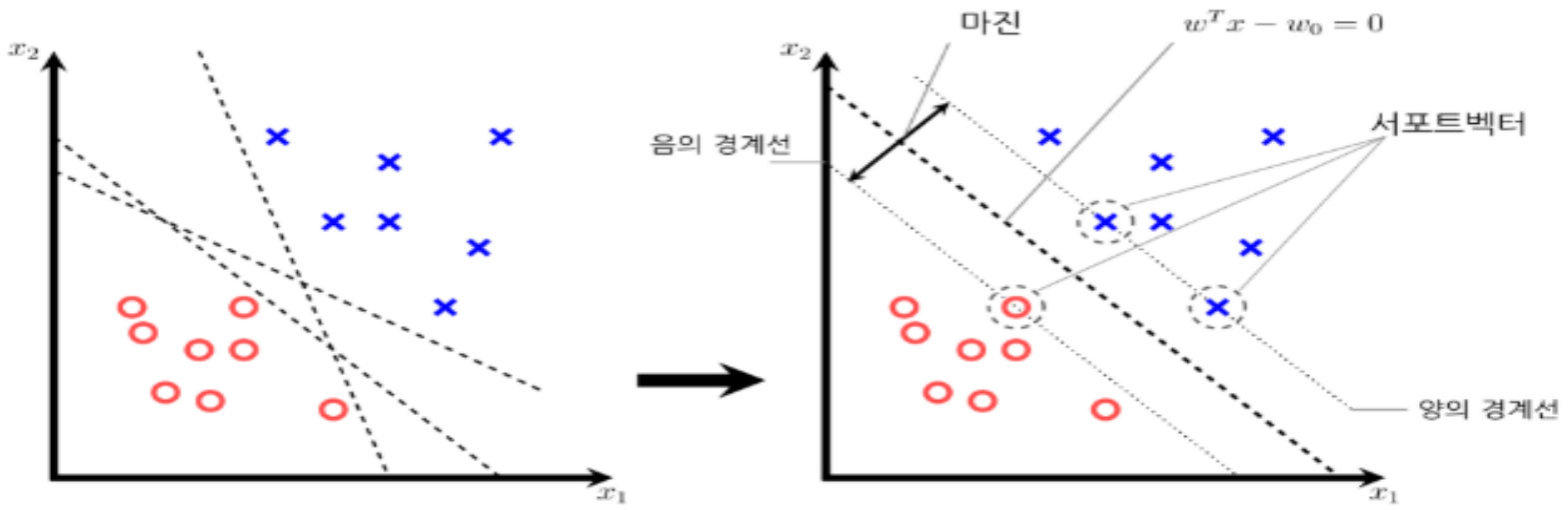
  AdaBoost.py (3 / 3)

```
# 분류기 성능을 확인
expected = labels[train_size:]
predicted = classifier.predict(images[train_size:])

print('Accuracy:', accuracy_score(expected, predicted))
print('Confusion matrix:', confusion_matrix(expected, predicted))
print('Precision:', precision_score(expected, predicted, pos_label=3))
print('Recall:', recall_score(expected, predicted, pos_label=3))
print('F-measure:', f1_score(expected, predicted, pos_label=3))
```

서포트 벡터 머신 (Support Vector Machine)

서포트 벡터 머신 (Support Vector Machine)

1. 서포트 벡터 머신은 분류 및 회귀 모두 사용 가능한 지도 학습 알고리즘이다.
2. 서포트 벡터 머신에서는 분할선부터 근접 샘플 데이터까지 마진을 최대화하는 직선을 가장 좋은 분할선으로 생각한다.
3. 서포트 벡터 머신은 학습 데이터의 노이즈에 강하고 분류 성능이 매우 좋다는 특징이 있다.
4. 다른 알고리즘에 비교하면 학습 데이터 개수도 많이 필요하지 않는다.



서포트 벡터 머신 (Support Vector Machine)

- **서포트 벡터 머신 (Support Vector Machine)**
  SupportVectorMachine.py (1 / 3)

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# 손으로 쓴 숫자 데이터 읽기
digits = datasets.load_digits()

# 이미지를 2행 5열로 표시
for label, img in zip(digits.target[:10], digits.images[:10]):
    plt.subplot(2, 5, label + 1)
    plt.axis('off')
    plt.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Digit: {0}'.format(label))
plt.show()
```

서포트 벡터 머신 (Support Vector Machine)

- **서포트 벡터 머신 (Support Vector Machine)**

SupportVectorMachine.py (2 / 3)

```
# 3과 8의 데이터를 구하기
flag_3_8 = (digits.target == 3) + (digits.target == 8)

# 3과 8의 데이터를 구하기
images = digits.images[flag_3_8]
labels = digits.target[flag_3_8]

# 3과 8의 이미지 데이터를 1차원으로 변환
images = images.reshape(images.shape[0], -1)

# 분류기 생성
n_samples = len(flag_3_8[flag_3_8])
train_size = int(n_samples * 3 / 5)
# SVC(Support Vector Classifier)
# C는 패널티 파라미터로 어느 정도 오류 분류를 허용하는지 나타낸다.
# 알고리즘에 사용될 커널 유형을 지정. 'linear', 'poly', 'rbf'(기본값), 'sigmoid', 'precomputed
classifier = svm.SVC(kernel = 'rbf')
classifier.fit(images[:train_size], labels[:train_size])
```

서포트 벡터 머신 (Support Vector Machine)

- **서포트 벡터 머신 (Support Vector Machine)**

  SupportVectorMachine.py (3 / 3)

```
# 분류기 성능을 확인
expected = labels[train_size:]
predicted = classifier.predict(images[train_size:])

print('Accuracy:', accuracy_score(expected, predicted))
print('Confusion matrix:', confusion_matrix(expected, predicted))
print('Precision:', precision_score(expected, predicted, pos_label=3))
print('Recall:', recall_score(expected, predicted, pos_label=3))
print('F-measure:',  f1_score(expected, predicted, pos_label=3))
```

분류(classification) 예제

- 분류(classification) 예제

분류(classification) 알고리즘은 예측하려는 대상의 속성(변수)을 입력받고, 목표 변수가 가지고 있는 카테고리(범주형) 값 중에서 어느 한 값으로 분류하여 예측하는 것이다.
훈련 데이터에 목표 변수(0 또는 1)을 함께 입력하기 때문에 지도 학습에 속하는 알고리즘이다.

분류 알고리즘은 고객 분류, 질병 진단, 스팸 메일 필터링, 음성 인식 등 목표 변수가 카테고리 값을 갖는 경우에 사용한다.

분류 알고리즘에는 KNN, SVM, Decision Tree, Logistic Regression 등 다양한 알고리즘이 있다.

KNN(K Nearest Neighbor)

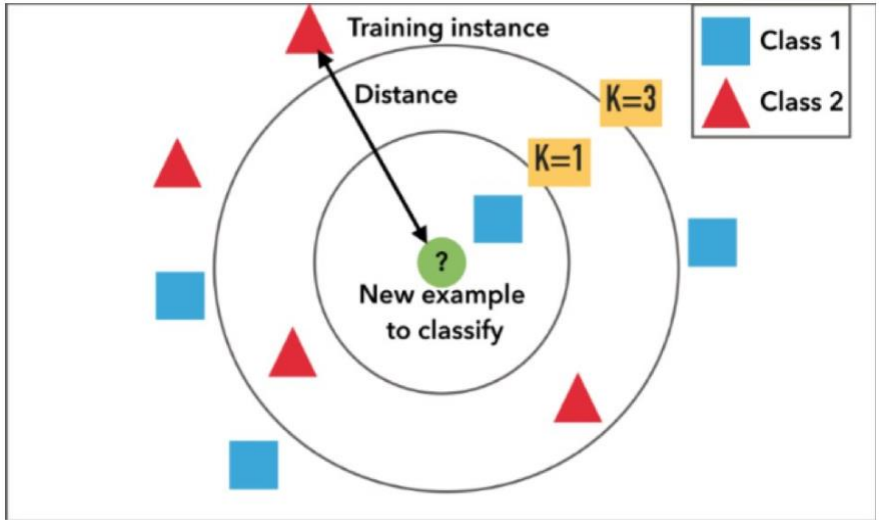
- K 최근접 이웃(KNN) 알고리즘

KNN은 K Nearest Neighbor 의 약칭이다. K개의 가까운 이웃이라는 뜻이다. 새로운 관측값이 주어지며 기존 데이터 중에서 가장 속성이 비슷한 K개의 이웃을 먼저 찾는다. 그리고 가까운 이웃들이 가지고 있는 목표값과 같은 값으로 분류하여 예측한다.

K값에 따라 예측의 정확도가 달라지므로, 적절한 K값을 찾는 것이 매우 중요하다.

K=1이면 가장 가까운 이웃인 Class1으로 분류된다.

K=3이면 가장 가까운 이웃중 가장 많은 Class2로 분류된다.



KNN(K Nearest Neighbor)

- K 최근접 이웃(KNN) 알고리즘

seaborn 라이브러리에서 제공되는 titanic 데이터셋의 탑승객의 생존여부를 KNN 알고리즘으로 분류해보자.

Step1. 데이터 준비

Step2. 데이터 탐색

Step3. 분석에 활용할 속성(feature 또는 variable) 선택

Step4. 훈련 데이터 / 검증 데이터 분할

Step5. KNN모델 학습 및 모델 성능 평가

KNN(K Nearest Neighbor)

- Step1. 데이터 준비
  Seaborn에서 제공하는 titanic 데이터셋 가져오기

knn_classification.py (1)

```
#기본 라이브러리 불러오기
import pandas as pd
import seaborn as sns

# load_dataset 함수로 titanic 데이터를 읽어와서 데이터프레임으로 변환
df = sns.load_dataset('titanic')
print(df)                          # [891 rows x 15 columns]

# 데이터 살펴보기
print(df.head())                   # 앞에서 5개의 데이터 불러오기
print('₩n')

#  IPython 디스플레이 설정 - 출력할 열의 개수를 15개로 늘리기
pd.set_option('display.max_columns', 15)
print(df.head())
print('₩n')
```

KNN(K Nearest Neighbor)

- Step1. 데이터 준비 : 결과

```
#  IPython 디스플레이 설정 - 출력할 열의 개수를 15개로 늘리기
pd.set_option('display.max_columns', 15)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	W
0	0	3	male	22.0	1	0	7.2500	S	Third	
1	1	1	female	38.0	1	0	71.2833	C	First	
2	1	3	female	26.0	0	0	7.9250	S	Third	
3	1	1	female	35.0	1	0	53.1000	S	First	
4	0	3	male	35.0	0	0	8.0500	S	Third	

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

KNN(K Nearest Neighbor)

- Step2. 데이터 탐색

knn_classification.py (2)

```
# 데이터 자료형 확인 : 데이터를 확인하고 NaN이 많은 열 삭제
print(df.info())
print('₩n')

# NaN값이 많은 deck(배의 갑판)열을 삭제 : deck 열은 유효한 값이 203개
# embarked(승선한)와 내용이 겹치는 embark_town(승선 도시) 열을 삭제
# 전체 15개의 열에서  deck, embark_town 2개의 열이 삭제되어서
# 13개의 열이름만 출력
rdf = df.drop(['deck', 'embark_town'], axis=1)
print(rdf.columns.values)
print('₩n')
# ['survived' 'pclass' 'sex' 'age' 'sibsp' 'parch' 'fare' 'embarked' 'class'
#  'who' 'adult_male' 'alive' 'alone']
```

KNN(K Nearest Neighbor)

- Step2. 데이터 탐색

knn_classification.py (3)

```
# 승객의 나이를 나타내는 age 열에 누락 데이터가 177개 포함되어 있다.
# 누락 데이터를 평균 나이로 치환하는 방법도 가능하지만, 누락 데이터가 있는 행을 모두 삭제
# 즉, 177명의 승객 데이터를 포기하고 나이 데이터가 있는 714명의 승객만을 분석 대상
# age 열에 나이 데이터가 없는 모든 행을 삭제 - age 열(891개 중 177개의 NaN 값)
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
print(len(rdf))                          # 714  (891개 중 177개 데이터 삭제)

# embarked열에는 승객들이 타이타닉호에 탑승한 도시명의 첫 글자가 들어있다.
# embarked열에는 누락데이터(NaN)가 2개에 있는데, 누락데이터를 가장많은 도시명(S)으로치환
# embarked 열의 NaN값을 승선도시 중에서 가장 많이 출현한 값으로 치환하기
# value_counts()함수와 idxmax()함수를 사용하여 승객이 가장 많이 탑승한 도시명의 첫글자는 S
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()
print(most_freq)                         # S : Southampton
```

KNN(K Nearest Neighbor)

- Step2. 데이터 탐색

  knn_classification.py (4)

```
# embarked 열의 최빈값(top)을 확인하면 S 로 출력됨
print(rdf.describe(include='all'))
print('₩n')

# embarked 열에 fillna() 함수를 사용하여 누락 데이터(NaN)를 S로 치환한다.
rdf['embarked'].fillna(most_freq, inplace=True)
```

KNN(K Nearest Neighbor)

- Step2. 데이터 탐색 : 결과

# 데이터 자료형 확인 : 데이터를 확인하고 NaN이 많은 열 삭제
print(df.info())

```
survived       891 non-null  int64
pclass         891 non-null  int64
sex            891 non-null  object
age            714 non-null  float64
sibsp          891 non-null  int64
parch          891 non-null  int64
fare           891 non-null  float64
embarked       889 non-null  object
class          891 non-null  category
who            891 non-null  object
adult_male     891 non-null  bool
deck           203 non-null  category
embark_town    889 non-null  object
alive          891 non-null  object
alone          891 non-null  bool
```

KNN(K Nearest Neighbor)

Step2. 데이터 탐색 : 결과

# embarked 열의 최빈값(top)을 확인하면 S 로 출력됨
print(rdf.describe(include='all'))

	fare	embarked	class	who	adult_male	alive	alone
count	714.000000	712	714	714	714	714	714
unique	NaN	3	3	3	2	2	2
top	NaN	S	Third	man	True	no	True
freq	NaN	554	355	413	413	424	404
mean	34.694514	NaN	NaN	NaN	NaN	NaN	NaN
std	52.918930	NaN	NaN	NaN	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
25%	8.050000	NaN	NaN	NaN	NaN	NaN	NaN
50%	15.741700	NaN	NaN	NaN	NaN	NaN	NaN
75%	33.375000	NaN	NaN	NaN	NaN	NaN	NaN
max	512.329200	NaN	NaN	NaN	NaN	NaN	NaN

KNN(K Nearest Neighbor)

- Step3. 분석에 사용할 열(속성) 선택

knn_classification.py (5)

```
# 분석에 활용할 열(속성)을 선택
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
print(ndf.head())
print('₩n')

# KNN모델을 적용하기 위해 sex열과embarked열의 범주형 데이터를 숫자형으로 변환
# 이 과정을 더미 변수를 만든다고 하고, 원핫인코딩(one-hot-encoding)이라고 부른다.
# 원핫인코딩 - 범주형 데이터를 모델이 인식할 수 있도록 숫자형으로 변환 하는것
# sex 열은 male과 female값을 열 이름으로 갖는 2개의 더미 변수 열이 생성된다.
# concat()함수로 생성된 더미 변수를 기존 데이터프레임에 연결한다.
onehot_sex = pd.get_dummies(ndf['sex'])
ndf = pd.concat([ndf, onehot_sex], axis=1)
```

KNN(K Nearest Neighbor)

- Step3. 분석에 사용할 열(속성) 선택

knn_classification.py (6)

```
# embarked 열은 3개의 더미 변수 열이 만들어지는데, prefix='town' 옵션을
# 사용하여 열 이름에 접두어 town을 붙인다. ( town_C, town_Q, town_S)
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')
ndf = pd.concat([ndf, onehot_embarked], axis=1)

# 기존 sex열과 embarked열 삭제
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
print(ndf.head())                    # 더미 변수로 데이터 출력
print('₩n')
```

KNN(K Nearest Neighbor)

- Step3. 분석에 사용할 열(속성) 선택 : 결과

```
# 분석에 활용할 열(속성)을 선택
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
print(ndf.head())
```

	survived	pclass	sex	age	sibsp	parch	embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S

KNN(K Nearest Neighbor)

- Step3. 분석에 사용할 열(속성) 선택 : 결과

```
# 기존 sex열과 embarked열 삭제
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
print(ndf.head())                    # 더미 변수로 데이터 출력
```

	survived	pclass	age	sibsp	parch	female	male	town_C	town_Q	town_S
0	0	3	22.0	1	0	0	1	0	0	1
1	1	1	38.0	1	0	1	0	1	0	0
2	1	3	26.0	0	0	1	0	0	0	1
3	1	1	35.0	1	0	1	0	0	0	1
4	0	3	35.0	0	0	0	1	0	0	1

KNN(K Nearest Neighbor)

- Step4. 훈련 데이터 / 검증 데이터 분할

knn_classification.py (7)

```
# 변수 정의
x=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male',
       'town_C', 'town_Q', 'town_S']]          # 독립 변수(x)
y=ndf['survived']                              # 종속 변수(y)

# 독립 변수 데이터를 정규화(normalization)
# 독립 변수 열들이 갖는 데이터의 상대적 크기 차이를 없애기 위하여
# 정규화를 한다.
from sklearn import preprocessing
x = preprocessing.StandardScaler().fit(x).transform(x)
```

KNN(K Nearest Neighbor)

- Step4. 훈련 데이터 / 검증 데이터 분할

knn_classification.py (8)

```
# train data 와 test data로 분할(7:3 비율)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=10)

print('train data 개수: ', x_train.shape)      # train data 개수:  (499, 9)
print('test data 개수: ', x_test.shape)        # test data 개수:  (215, 9)
```

KNN(K Nearest Neighbor)

- Step5. KNN모델 학습 및 모델 성능 평가

knn_classification.py (9)

```
# sklearn 라이브러리에서 KNN 분류 모델 가져오기
from sklearn.neighbors import KNeighborsClassifier

# KNN 모델 객체 생성 (k=5로 설정)
knn = KNeighborsClassifier(n_neighbors=5)

# train data를 가지고 모델 학습
knn.fit(x_train, y_train)

# test data를 가지고 y_hat을 예측 (분류)
y_hat = knn.predict(x_test)                    # 예측값 구하기

# 첫 10개의 예측값(y_hat)과 실제값(y_test) 비교 : 10개 모두 일치함 (0:사망자, 1:생존자)
print(y_hat[0:10])                             # [0 0 1 0 0 1 1 1 0 0]
print(y_test.values[0:10])                     # [0 0 1 0 0 1 1 1 0 0]
```

KNN(K Nearest Neighbor)

- Step5. KNN모델 학습 및 모델 성능 평가

  knn_classification.py (10)

```
# KNN모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
knn_matrix = metrics.confusion_matrix(y_test, y_hat)
print(knn_matrix)
# [[109   16]
#  [ 25   65]]

# KNN모델 성능 평가 - 평가지표 계산
knn_report = metrics.classification_report(y_test, y_hat)
print(knn_report)
```

KNN(K Nearest Neighbor)

- Step5. KNN모델 학습 및 모델 성능 평가 : 결과

```
# KNN모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
knn_matrix = metrics.confusion_matrix(y_test, y_hat)
print(knn_matrix)

# [[109  16]
#  [ 25  65]]
```

TP(True Positive) : 215명의 승객 중에서 사망자를 정확히 분류한 것이 109명
FP(False Positive) : 생존자를 사망자로 잘못 분류한 것이 25명
FN(False Negative) : 사망자를 생존자로 잘못 분류한 것이 16명
TN(True Negative) : 생존자를 정확하게 분류한 것이 65명

KNN(K Nearest Neighbor)

- Step5. KNN모델 학습 및 모델 성능 평가 : 결과

```
# KNN모델 성능 평가 - 평가지표 계산
knn_report = metrics.classification_report(y_test, y_hat)
print(knn_report)
```

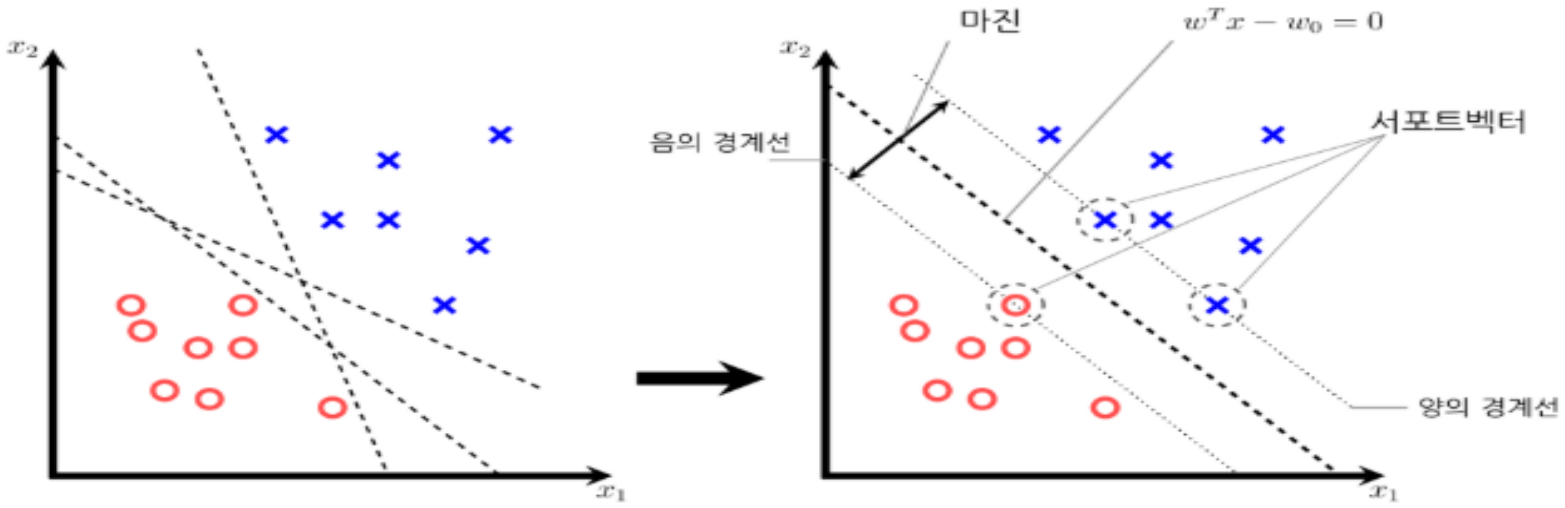
f1지표(f1-score)는 모델의 예측력을 종합적으로 평가하는 지표이다.
f1-score 지표를 보면 사망자(0) 예측의 정확도가 0.84이고, 생존자(1) 예측의 정확도는 0.76으로 예측 능력에 차이가 있다. 평균적으로 0.81 정확도를 갖는다.

	precision	recall	f1-score	support
0	0.81	0.87	0.84	125
1	0.80	0.72	0.76	90
accuracy			0.81	215
macro avg	0.81	0.80	0.80	215
weighted avg	0.81	0.81	0.81	215

SVM(Support Vector Machine)

서포트 벡터 머신 (Support Vector Machine)

1. 서포트 벡터 머신은 분류 및 회귀 모두 사용 가능한 지도 학습 알고리즘이다.
2. 서포트 벡터 머신에서는 분할선부터 근접 샘플 데이터까지 마진을 최대화하는 직선을 가장 좋은 분할선으로 생각한다.
3. 서포트 벡터 머신은 학습 데이터의 노이즈에 강하고 분류 성능이 매우 좋다는 특징이 있다.
4. 다른 알고리즘에 비교하면 학습 데이터 개수도 많이 필요하지 않는다.



SVM(Support Vector Machine)

- SVM 알고리즘

seaborn 라이브러리에서 제공되는 titanic 데이터셋의 탑승객의 생존여부를 SVM 알고리즘으로 분류해보자.

Step1. 데이터 준비

Step2. 데이터 탐색

Step3. 분석에 활용할 속성(feature 또는 variable) 선택

Step4. 훈련 데이터 / 검증 데이터 분할

Step5. SVM모델 학습 및 모델 성능 평가

SVM(Support Vector Machine)

- Step1. 데이터 준비
  Seaborn에서 제공하는 titanic 데이터셋 가져오기

svm_classification.py (1)

```
#기본 라이브러리 불러오기
import pandas as pd
import seaborn as sns

# load_dataset 함수로 titanic 데이터를 읽어와서 데이터프레임으로 변환
df = sns.load_dataset('titanic')
print(df)                          # [891 rows x 15 columns]

# 데이터 살펴보기
print(df.head())                   # 앞에서 5개의 데이터 불러오기
print('₩n')

#  IPython 디스플레이 설정 - 출력할 열의 개수를 15개로 늘리기
pd.set_option('display.max_columns', 15)
print(df.head())
print('₩n')
```

SVM(Support Vector Machine)

- Step1. 데이터 준비 : 결과

```
#  IPython 디스플레이 설정 - 출력할 열의 개수를 15개로 늘리기
pd.set_option('display.max_columns', 15)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	W
0	0	3	male	22.0	1	0	7.2500	S	Third	
1	1	1	female	38.0	1	0	71.2833	C	First	
2	1	3	female	26.0	0	0	7.9250	S	Third	
3	1	1	female	35.0	1	0	53.1000	S	First	
4	0	3	male	35.0	0	0	8.0500	S	Third	

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

SVM(Support Vector Machine)

- Step2. 데이터 탐색

svm_classification.py (2)

```
# 데이터 자료형 확인 : 데이터를 확인하고 NaN이 많은 열 삭제
print(df.info())
print('₩n')

# NaN값이 많은 deck(배의 갑판)열을 삭제 : deck 열은 유효한 값이 203개
# embarked(승선한)와 내용이 겹치는 embark_town(승선 도시) 열을 삭제
# 전체 15개의 열에서  deck, embark_town 2개의 열이 삭제되어서
# 13개의 열이름만 출력
rdf = df.drop(['deck', 'embark_town'], axis=1)
print(rdf.columns.values)
print('₩n')
# ['survived' 'pclass' 'sex' 'age' 'sibsp' 'parch' 'fare' 'embarked' 'class'
#  'who' 'adult_male' 'alive' 'alone']
```

SVM(Support Vector Machine)

- Step2. 데이터 탐색

svm_classification.py (3)

```
# 승객의 나이를 나타내는 age 열에 누락 데이터가 177개 포함되어 있다.
# 누락 데이터를 평균 나이로 치환하는 방법도 가능하지만, 누락 데이터가 있는 행을 모두 삭제
# 즉, 177명의 승객 데이터를 포기하고 나이 데이터가 있는 714명의 승객만을 분석 대상
# age 열에 나이 데이터가 없는 모든 행을 삭제 - age 열(891개 중 177개의 NaN 값)
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
print(len(rdf))                          # 714  (891개 중 177개 데이터 삭제)
print('₩n')

# embarked열에는 승객들이 타이타닉호에 탑승한 도시명의 첫 글자가 들어있다.
# embarked열에는 누락데이터(NaN)가 2개에 있는데, 누락데이터를 가장많은 도시명(S)으로치환
# embarked 열의 NaN값을 승선도시 중에서 가장 많이 출현한 값으로 치환하기
# value_counts()함수와 idxmax()함수를 사용하여 승객이 가장 많이 탑승한 도시명의 첫글자는 S
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()
print(most_freq)                         # S : Southampton
print('₩n')
```

SVM(Support Vector Machine))

- Step2. 데이터 탐색

  svm_classification.py (4)

```
# embarked 열의 최빈값(top)을 확인하면 S 로 출력됨
print(rdf.describe(include='all'))
print('₩n')

# embarked 열에 fillna() 함수를 사용하여 누락 데이터(NaN)를 S로 치환한다.
rdf['embarked'].fillna(most_freq, inplace=True)
```

SVM(Support Vector Machine)

- Step2. 데이터 탐색 : 결과

# 데이터 자료형 확인 : 데이터를 확인하고 NaN이 많은 열 삭제
print(df.info())

```
survived       891 non-null  int64
pclass         891 non-null  int64
sex            891 non-null  object
age            714 non-null  float64
sibsp          891 non-null  int64
parch          891 non-null  int64
fare           891 non-null  float64
embarked       889 non-null  object
class          891 non-null  category
who            891 non-null  object
adult_male     891 non-null  bool
deck           203 non-null  category
embark_town    889 non-null  object
alive          891 non-null  object
alone          891 non-null  bool
```

SVM(Support Vector Machine)

- Step2. 데이터 탐색 : 결과

# embarked 열의 최빈값(top)을 확인하면 S 로 출력됨
print(rdf.describe(include='all'))

	fare	embarked	class	who	adult_male	alive	alone
count	714.000000	712	714	714	714	714	714
unique	NaN	3	3	3	2	2	2
top	NaN	S	Third	man	True	no	True
freq	NaN	554	355	413	413	424	404
mean	34.694514	NaN	NaN	NaN	NaN	NaN	NaN
std	52.918930	NaN	NaN	NaN	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
25%	8.050000	NaN	NaN	NaN	NaN	NaN	NaN
50%	15.741700	NaN	NaN	NaN	NaN	NaN	NaN
75%	33.375000	NaN	NaN	NaN	NaN	NaN	NaN
max	512.329200	NaN	NaN	NaN	NaN	NaN	NaN

SVM(Support Vector Machine)

- Step3. 분석에 사용할 열(속성) 선택

svm_classification.py (5)

```
# 분석에 사용할 열(속성)을 선택
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
print(ndf.head())
print('₩n')

# KNN모델을 적용하기 위해 sex열과embarked열의 범주형 데이터를 숫자형으로 변환
# 이 과정을 더미 변수를 만든다고 하고, 원핫인코딩(one-hot-encoding)이라고 부른다.
# 원핫인코딩 - 범주형 데이터를 모델이 인식할 수 있도록 숫자형으로 변환 하는것
# sex 열은 male과 female값을 열 이름으로 갖는 2개의 더미 변수 열이 생성된다.
# concat()함수로 생성된 더미 변수를 기존 데이터프레임에 연결한다.
onehot_sex = pd.get_dummies(ndf['sex'])
ndf = pd.concat([ndf, onehot_sex], axis=1)
```

SVM(Support Vector Machine)

- Step3. 분석에 사용할 열(속성) 선택

svm_classification.py (6)

```
# embarked 열은 3개의 더미 변수 열이 만들어지는데, prefix='town' 옵션을
# 사용하여 열 이름에 접두어 town을 붙인다. ( town_C, town_Q, town_S)
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')
ndf = pd.concat([ndf, onehot_embarked], axis=1)

# 기존 sex열과 embarked열 삭제
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
print(ndf.head())                    # 더미 변수로 데이터 출력
print('₩n')
```

SVM(Support Vector Machine)

- Step3. 분석에 사용할 열(속성) 선택 : 결과

```
# 분석에 사용할 열(속성)을 선택
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
print(ndf.head())
```

	survived	pclass	sex	age	sibsp	parch	embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S

SVM(Support Vector Machine)

- Step3. 분석에 사용할 열(속성) 선택 : 결과

```
# 기존 sex열과 embarked열 삭제
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
print(ndf.head())                    # 더미 변수로 데이터 출력
```

	survived	pclass	age	sibsp	parch	female	male	town_C	town_Q	town_S
0	0	3	22.0	1	0	0	1	0	0	1
1	1	1	38.0	1	0	1	0	1	0	0
2	1	3	26.0	0	0	1	0	0	0	1
3	1	1	35.0	1	0	1	0	0	0	1
4	0	3	35.0	0	0	0	1	0	0	1

SVM(Support Vector Machine)

- Step4. 훈련 데이터 / 검증 데이터 분할

svm_classification.py (7)

```
# 변수 정의
x=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male',
        'town_C', 'town_Q', 'town_S']]          # 독립 변수 X
y=ndf['survived']                               # 종속 변수 Y

# 독립 변수 데이터를 정규화(normalization)
# 독립 변수 열들이 갖는 데이터의 상대적 크기 차이를 없애기 위하여
# 정규화를 한다.
from sklearn import preprocessing
x = preprocessing.StandardScaler().fit(x).transform(x)
```

SVM(Support Vector Machine)

- Step4. 훈련 데이터 / 검증 데이터 분할

svm_classification.py (8)

```
# train data 와 test data로 분할(7:3 비율)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=10)

print('train data 개수: ', x_train.shape)      # train data 개수:  (499, 9)
print('test data 개수: ', x_test.shape)        # test data 개수:  (215, 9)
```

SVM(Support Vector Machine)

❖ Step5. SVM모델 학습 및 모델 성능 평가

svm_classification.py (9)

```
# sklearn 라이브러리에서 SVM 분류 모델 가져오기
from sklearn import svm

# SVC 모델 객체 생성 (kernel='rbf' 적용)
svm_model = svm.SVC(kernel='rbf')

# train data를 가지고 모델 학습
svm_model.fit(x_train, y_train)

# test data를 가지고 y_hat을 예측 (분류)
y_hat = svm_model.predict(x_test)                    # 예측값 구하기

# 첫 10개의 예측값(y_hat)과 실제값(y_test) 비교 : 8개 일치함( 0:사망자, 1:생존자)
print(y_hat[0:10])                                   # [0 0 1 0 0 0 1 0 0 0]
print(y_test.values[0:10])                           # [0 0 1 0 0 1 1 1 0 0]
```

SVM(Support Vector Machine)

- Step5. SVM모델 학습 및 모델 성능 평가

svm_classification.py (10)

```
# SVM모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
svm_matrix = metrics.confusion_matrix(y_test, y_hat)
print(svm_matrix)
# [[120   5]
#  [ 35  55]]

# SVM모델 성능 평가 - 평가지표 계산
svm_report = metrics.classification_report(y_test, y_hat)
print(svm_report)
```

SVM(Support Vector Machine)

- Step5. SVM모델 학습 및 모델 성능 평가 : 결과

```
# SVM모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
svm_matrix = metrics.confusion_matrix(y_test, y_hat)
print(svm_matrix)
# [[120   5]
#  [ 35  55]]
```

TP(True Positive) : 215명의 승객 중에서 사망자를 정확히 분류한 것이 120명
FP(False Positive) : 생존자를 사망자로 잘못 분류한 것이 35명
FN(False Negative) : 사망자를 생존자로 잘못 분류한 것이 5명
TN(True Negative) : 생존자를 정확하게 분류한 것이 55명

SVM(Support Vector Machine)

- Step5. SVM모델 학습 및 모델 성능 평가 : 결과

```
# SVM모델 성능 평가 - 평가지표 계산
svm_report = metrics.classification_report(y_test, y_hat)
print(svm_report)
```

f1지표(f1-score)는 모델의 예측력을 종합적으로 평가하는 지표이다.
f1-score 지표를 보면 사망자(0) 예측의 정확도가 0.86이고, 생존자(1) 예측의 정확도는 0.73으로 예측 능력에 차이가 있다.
전반적으로 KNN모델의 예측 능력과 큰 차이가 없다.

	precision	recall	f1-score	support
0	0.77	0.96	0.86	125
1	0.92	0.61	0.73	90
accuracy			0.81	215
macro avg	0.85	0.79	0.80	215
weighted avg	0.83	0.81	0.81	215

Decision Tree

- 결정 트리(Decision Tree) 알고리즘

Decision Tree 는 의사결정 나무라는 의미를 가지고 있다.
컴퓨터 알고리즘에서 즐겨 사용하는 트리(tree) 구조를 사용하고,
각 분기점(node)에는 분석 대상의 속성(변수)들이 위치한다.
각 분기점마다 목표값을 가장 잘 분류할 수 있는 속성을 찾아서 배치하고,
해당 속성이 갖는 값을 이용하여 새로운 가지(branch)를 만든다.
각 분기점에서 최적의 속성을 선택할 때는 해당 속성을 기준으로 분류한 값들이 구분되는 정도를 측정한다.
다른 종류의 값들이 섞여있는 정도를 나타내는 Entropy를 주로 활용하는데,
Entropy가 낮을수록 분류가 잘 된 것이다.
Entropy가 일정 수준 이하로 낮아질 때까지 앞의 과정을 반복한다.

Decision Tree

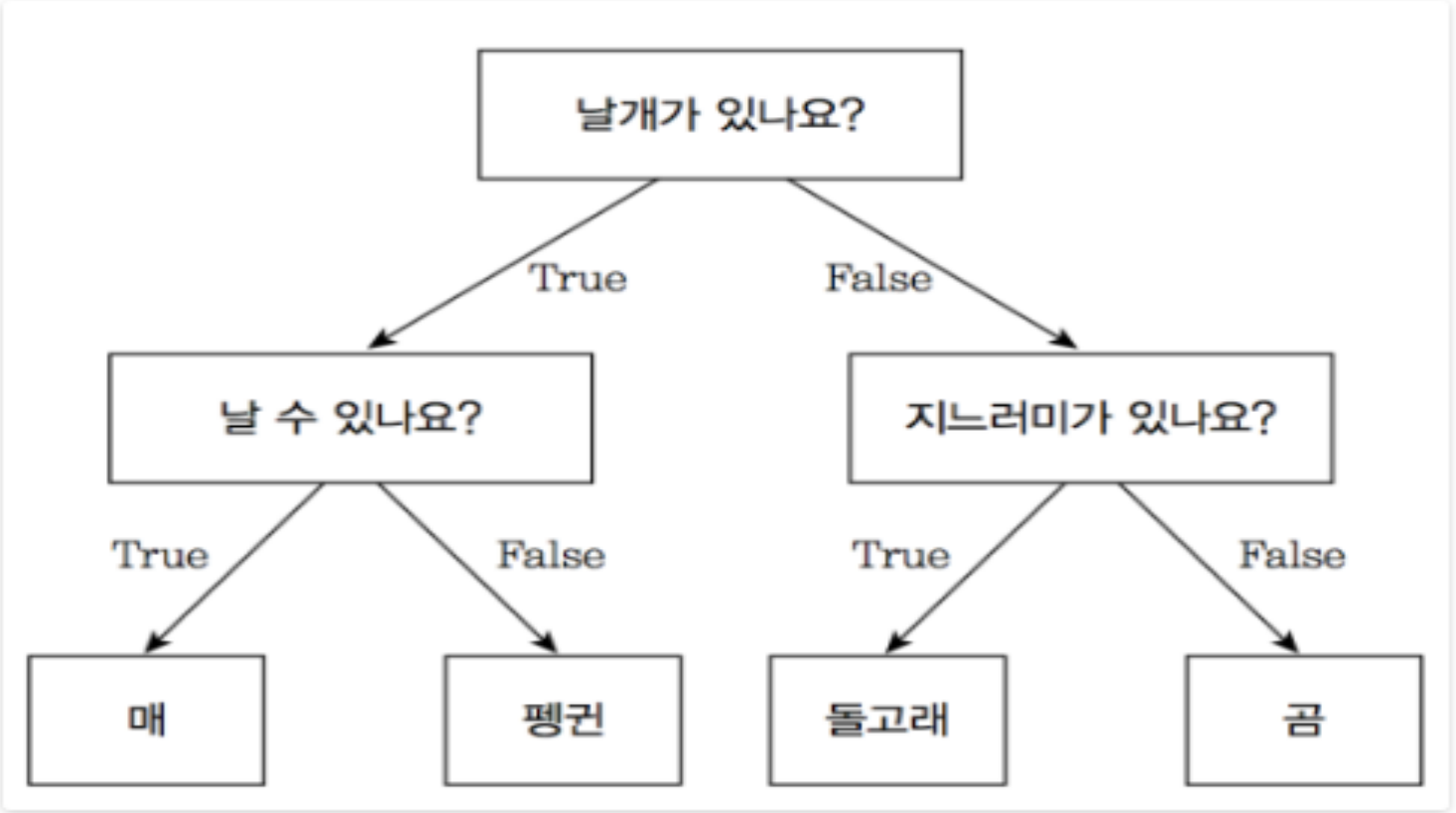
결정 트리(Decision Tree) 알고리즘

1. 결정 트리는 데이터를 여러 등급으로 분류하는 지도 학습 중의 하나로, 트리 구조를 이용한 분류 알고리즘이다.
2. 결정 트리 학습에서는 학습 데이터에서 트리 모델을 생성한다. 무엇을 기준으로 분기할 지에 따라 결정 트리는 몇 가지 방법으로 분류할 수 있다.
3. 결정 트리의 장점은 분류 규칙을 트리 모델로 가시화할 수 있어, 분류 결과의 해석이 비교적 용이하다는 점이다.
4. 생성한 분류 규칙도 편집할 수 있으며, 학습을 위한 계산 비용이 낮다는 점도 장점이다.
5. 결정 트리는 과적합 하는 경향이 있고, 취급하는 데이터의 특성에 따라 트리 모델을 생성하기 어렵다는 단점도 있다.
6. 결정 트리는 과적합 하는 경향이 있어 결정 트리 단독으로 사용하지 않고, 앙상블 학습을 조합해서 사용하는 경우가 많다.

Decision Tree

- 결정 트리(Decision Tree) 알고리즘

<몇가지 동물들을 구분하기 위한 결정트리>



Decision Tree

- 결정 트리(Decision Tree) 알고리즘

UCI(University of California, Irvine) 머신러닝 저장소에서 제공하는 암세포 진단(breast cancer) 데이터셋을 이용하여 악성종양 여부를 Decision Tree 알고리즘으로 분류해보자.

Step1. 데이터 준비

Step2. 데이터 탐색

Step3. 훈련 데이터 / 검증 데이터 분할

Step4. Decision Tree모델 학습 및 모델 성능 평가

Decision Tree

- Step1. 데이터 준비

UCI(University of California, Irvine) 머신러닝 저장소에서 제공하는 암세포 진단(breast cancer) 데이터셋을 사용한다. 샘플 ID, 암세포 조직의 크기와 모양 등 종양 특성을 나타내는 열 9개, 악성 종양 여부(2: 양성, 4: 악성)를 나타내는 열로 총 11개의 열로 구성된다.

tree_classification.py (1)

```
import pandas as pd
import numpy as np

# UCI 저장소에서 암세포 진단(Breast Cancer) 데이터셋 가져오기
uci_path = 'https://archive.ics.uci.edu/ml/machine-learning-databases/₩
breast-cancer-wisconsin/breast-cancer-wisconsin.data'
df = pd.read_csv(uci_path, header=None)
print(df)                          # [699 rows x 11 columns]

# 11개의 열 이름 지정
df.columns = ['id','clump','cell_size','cell_shape', 'adhesion','epithlial',
              'bare_nuclei','chromatin','normal_nucleoli', 'mitoses', 'class']

#  IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기
pd.set_option('display.max_columns', 15)
print(df.head())                   # 데이터 살펴보기 : 앞에서부터  5개의 데이터 출력
```

Decision Tree

- Step1. 데이터 준비 : 결과

```
#  IPython 디스플레이 설정 - 출력할 열의 개수를 15개로 늘리기
pd.set_option('display.max_columns', 15)
```

	id	clump	cell_size	cell_shape	adhesion	epithlial	bare_nuclei	₩
0	1000025	5	1	1	1	2	1	
1	1002945	5	4	4	5	7	10	
2	1015425	3	1	1	1	2	2	
3	1016277	6	8	8	1	3	4	
4	1017023	4	1	1	3	2	1	

	chromatin	normal_nucleoli	mitoses	class
0	3	1	1	2
1	3	2	1	2
2	3	1	1	2
3	3	7	1	2
4	3	1	1	2

Decision Tree

- Step2. 데이터 탐색

tree_classification.py (2)

```
# 데이터 자료형 확인 : bare_nuclei 열만 object(문자형)이고 나머지 열은 숫자형
print(df.info())
print('₩n')

# 데이터 통계 요약정보 확인  : bare_nuclei 열은 출력안됨 (10개의 열만 출력)
print(df.describe())
print('₩n')

# bare_nuclei 열의 고유값 확인 : bare_nuclei 열은 ? 데이터가 포함되어 있음
print(df['bare_nuclei'].unique())
# ['1' '10' '2' '4' '3' '9' '7' '?' '5' '8' '6']

# bare_nuclei 열의 자료형 변경 (문자열 -> 숫자)
# bare_nuclei 열의 '?' 를  누락데이터(NaN)으로 변경
df['bare_nuclei'].replace('?', np.nan, inplace=True)          # '?'을 np.nan으로 변경
df.dropna(subset=['bare_nuclei'], axis=0, inplace=True)       # 누락데이터 행을 삭제
df['bare_nuclei'] = df['bare_nuclei'].astype('int')           # 문자열을 정수형으로 변환

print(df.describe())                                          # 데이터 통계 요약정보 확인
print('₩n')                                                  # 11개의 열 모두 출력 : bare_nuclei 열  출력
```

Decision Tree

Step2. 데이터 탐색 : 결과

# 데이터 자료형 확인 : bare_nuclei 열만 object(문자형)이고 나머지 열은 숫자형
print(df.info())

```
Data columns (total 11 columns):
id                   699 non-null int64
clump                699 non-null int64
cell_size            699 non-null int64
cell_shape           699 non-null int64
adhesion             699 non-null int64
epithlial            699 non-null int64
bare_nuclei          699 non-null object
chromatin            699 non-null int64
normal_nucleoli      699 non-null int64
mitoses              699 non-null int64
class                699 non-null int64
```

Decision Tree

- Step3. 훈련 데이터 / 검증 데이터 분할

tree_classification.py (3)

```
# 분석에 사용할 속성(변수) 선택
x=df[['clump','cell_size','cell_shape', 'adhesion','epithlial',
        'bare_nuclei','chromatin','normal_nucleoli', 'mitoses']]     #독립(설명) 변수 X
y=df['class']                                                        #종속(예측) 변수 Y
# class (2: benign(양성), 4: malignant(악성) )

# 설명 변수 데이터를 정규화
from sklearn import preprocessing
x = preprocessing.StandardScaler().fit(x).transform(x)

# train data 와 test data로 구분(7:3 비율)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=10)

print('train data 개수: ', x_train.shape)                  # train data 개수:  (478, 9)
print('test data 개수: ', x_test.shape)                    # test data 개수:  (205, 9)
```

Decision Tree

- Step4. Decision Tree모델 학습 및 모델 성능 평가

tree_classification.py (4)

```
# sklearn 라이브러리에서 Decision Tree 분류 모델 가져오기
from sklearn import tree

# Decision Tree 모델 객체 생성 (criterion='entropy' 적용)
# 각 분기점에서 최적의 속성을 찾기 위해 분류 정도를 평가하는 기준으로 entropy 값을 사용
# 트리 레벨로 5로 지정하는데, 5단계 까지 가지를 확장할 수 있다는 의미
# 레벨이 많아 질수록 모델 학습에 사용하는 훈련 데이터에 대한 예측은 정확해진다.
tree_model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5)

# train data를 가지고 모델 학습
tree_model.fit(x_train, y_train)

# test data를 가지고 y_hat을 예측 (분류)
y_hat = tree_model.predict(x_test)                    # 2: benign(양성), 4: malignant(악성)

# 첫 10개의 예측값(y_hat)과 실제값(y_test) 비교 : 10개 모두 일치함
print(y_hat[0:10])                                    # [4 4 4 4 4 4 2 2 4 4]
print(y_test.values[0:10])                            # [4 4 4 4 4 4 2 2 4 4]
```

Decision Tree

- Step4. Decision Tree모델 학습 및 모델 성능 평가

  tree_classification.py (5)

```
# Decision Tree 모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
tree_matrix = metrics.confusion_matrix(y_test, y_hat)
print(tree_matrix)
# [[127   4]
#  [  2  72]]

# Decision Tree 모델 성능 평가 - 평가지표 계산
tree_report = metrics.classification_report(y_test, y_hat)
print(tree_report)
```

Decision Tree

❖ Step4. Decision Tree모델 학습 및 모델 성능 평가 : 결과

```
# Decision Tree 모델 성능 평가 - Confusion Matrix(혼동 행렬) 계산
from sklearn import metrics
tree_matrix = metrics.confusion_matrix(y_test, y_hat)
print(tree_matrix)
# [[127   4]
#  [  2  72]]
```

양성 종양의 목표값은 2, 악성 종양은 4
TP(True Positive) : 양성 종양을 정확하게 분류한 것이 127개
FP(False Positive) : 악성 종양을 양성 종양으로 잘못 분류한 것이 2개
FN(False Negative) : 양성 종양을 악성 종양으로 잘못 분류한 것이 4개
TN(True Negative) : 악성 종양을 정확하게 분류한 것이 72개

Decision Tree

Step4. Decision Tree모델 학습 및 모델 성능 평가 : 결과

```
# Decision Tree 모델 성능 평가 - 평가지표 계산
tree_report = metrics.classification_report(y_test, y_hat)
print(tree_report)
```

f1지표(f1-score)는 모델의 예측력을 종합적으로 평가하는 지표이다.
f1-score 지표를 보면 양성종양(2) 예측의 정확도가 0.98이고, 악성종양(4) 예측의 정확도는 0.96으로 예측 능력에 큰 차이가 없다. 평균적으로는 0.97의 정확도를 갖는다.

	precision	recall	f1-score	support
2	0.98	0.97	0.98	131
4	0.95	0.97	0.96	74
accuracy			0.97	205
macro avg	0.97	0.97	0.97	205
weighted avg	0.97	0.97	0.97	205