3장. 분류분석

1절. 분류분석 개요

2절. 분류 모형

3절. 인공신경망

4절. 분류 모형 성능평가

3장. 분류분석

1절. 분류분석 개요

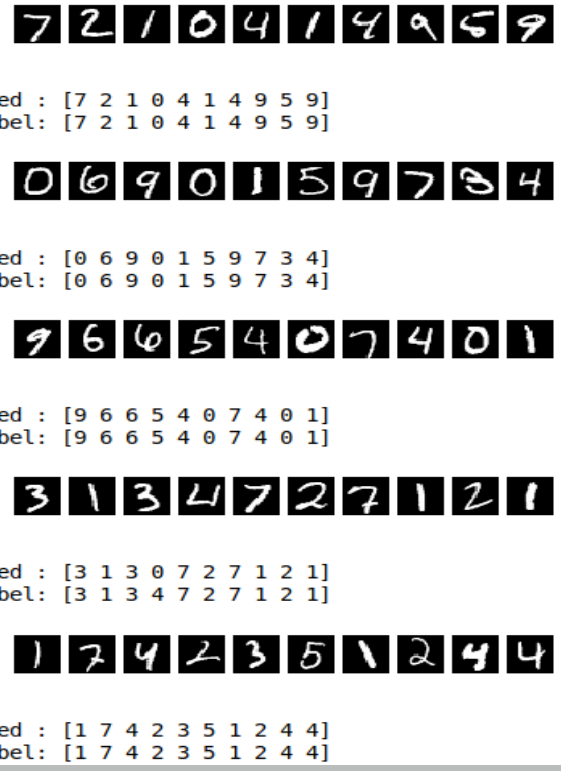
분류 분석(Classification analysis) 개요

데이터의 속성을 활용하여 데이터에 대한 분류 기준을 수립하는 과정

분류 분석은 지도 학습(Supervised learning, 또는 감독 학습)에 해당함

분류 학습의 예

- MNIST 필기체 숫자 데이터 분류
- iris 데이터의 종 분류
- 와인데이터 등급 분류



Scikit-learn 패키지

다양한 머신러닝 알고리즘을 하나의 패키지 안에서 모두 제공해줌

예제 데이터셋, 전처리 등의 기능도 제공해 줌

Scikit-learn 패키지

- 공식 사이트 : http://scikit-learn.org
- 도큐먼트 : http://scikit-learn.org/stable/
- 설치 : pip install sklearn

Tensorflow와 sklearn.datasets는 실습을 위한 샘플용 데이터셋을 제공

- 샘플용 데이터셋의 접근 방법
  - 기본적으로 Scikit-learn 패키지 안에 내장되어 있는 형태(load명령으로 import)
  - 인터넷에서 다운로드하여 사용하는 형태(fetch명령으로 import),
  - 새로운 데이터셋을 생성시켜 사용하는 형태(make 명령으로 생성)
- load 계열
  - **load_boston() : 보스턴 집값 데이터**
  - **load_diabetes() : 당뇨병 관련 데이터**
  - **load_iris() : iris 데이터**
- fetch 계열
  - fetch_openml() : mnist, iris, 보스톤 집값데이터
  - fetch_20newsgroups() : 뉴스텍스트 데이터
  - fetch_rcv1() : 로이터뉴스 말뭉치
  - fetch_california_housing : 주택 데이터
- make 계열
  - make_regression() : regression용 데이터 생성
  - make_classification() : classification용 데이터 생성
  - make_blobs() : clustering용 데이터 생성

3장. 분류분석

2절. 분류분석 모형

분류분석 모형의 종류

확률적 모형

주어진 데이터에 대해(conditionally) 각 클래스가 정답일 조건부확률(conditional probability)을 계산하는 모형,

확률적 모형은 조건부확률을 계산하는 방법에 따라 직접 조건부확률 함수를 추정하는 확률적 판별(discriminative) 모형과 베이즈 정리를 사용하는 확률적 생성(generative) 모형으로 나누어 짐

판별함수 모형

주어진 데이터를 클래스에 따라 서로 다른 영역으로 나누는 경계면(decision boundary)을 찾은 후 이 경계면으로부터 주어진 데이터가 어느 위치에 있는지를 계산하는 판별함수를 이용하는 모형

분류분석 모형의 종류

모형	방법론 / 클래스
Quadratic Discriminant Analysis	**확률적** 생성(generative) 모형 sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis
나이브 베이지안 (Naive Bayes)	**확률적** 생성(generative) 모형 sklearn.naive_bayes.MultinomialNB
로지스틱 회귀 (Logistic Regression)	**확률적** 판별(discriminative) 모형 sklearn.linear_model.LogisticRegression
의사결정나무 (Decision Tree)	**확률적** 판별(discriminative) 모형 sklearn.tree.DecisionTreeClassifier
퍼셉트론 (Perceptron)	**판별함수**(discriminant function) 모형 sklearn.linear_model.Perceptron
서포트 벡터 머신 (Support Vector Machine)	**판별함수**(discriminant function) 모형 sklearn.svm.SVC
신경망 (Neural Network)	**판별함수**(discriminant function) 모형 sklearn.neural_network.MLPClassifier

주어진 데이터에 대해 각 클래스가 정답일 조건부확률을 계산하는 모형

조건부확률을 계산하는 방법

- 생성 모형(generative model) : 베이즈 정리 이용
- 판별 모형(discriminative model) : 조건부확률 이용

Scikit-Learn에서 조건부확률을 사용하는 분류 모형들

- 모두 독립변수 x가 주어지면 종속변수 y의 모든 카테고리 값에 대해 다음 함수를 지원

- predict_proba() : 조건부확률을 계산하는 함수
  predict_log_proba() : 조건부확률의 로그값을 계산하는 함수

각 클래스별 특징 데이터의 확률분포를 추정한 다음 베이즈 정리를 사용하여 확률 P를 계산

$$P(y=k|x) = \frac{P(x|y=k)\,P(y=k)}{P(x)}$$

전체 확률의 법칙을 이용하여 특징 데이터 x의 무조건부 확률분포 P(x)를 구할 수 있음

$$P(x) = \sum_{k=1}^{K} P(x|y=k)\,P(y=k)$$

베이즈 정리(Bayes' theorem)

- 두 확률 변수의 사전 확률과 사후 확률 사이의 관계를 나타내는 정리
- 베이즈 확률론 해석에 따르면 베이즈 정리는 사전확률로부터 사후확률을 구할 수 있음
- 베이즈 정리는 불확실성 하에서 의사결정문제를 수학적으로 다룰 때 중요하게 이용함
- 특히, 정보와 같이 눈에 보이지 않는 무형자산이 지닌 가치를 계산할 때 유용하게 사용됨
- 전통적인 확률이 연역적 추론에 기반을 두고 있다면 베이즈 정리는 확률임에도 귀납적, 경험적인 추론을 사용함

QDA

이차판별분석법(quadratic discriminant analysis, QDA)는 대표적인 확률론적 생성모형(generative model)

가능도 즉, y의 클래스값에 따른 x의 분포에 대한 정보를 먼저 알아낸 후, 베이즈 정리를 사용하여 주어진 x에 대한 y의 확률분포를 찾아냄

이차판별분석법에서는 독립변수 x가 실수이고 확률분포가 다변수 정규분포라고 가정함

확률분포들을 알고 있으면 독립변수 x에 대한 y클래스의 조건부확률분포는 다음과 같이 베이즈 정리와 전체 확률 법칙으로 구할 수 있음

$$P(y=k|x) = \frac{p(x|y=k)P(y=k)}{p(x)} = \frac{p(x|y=k)P(y=k)}{\sum_l p(x|y=l)P(y=l)}$$

```
sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis(*,
    priors=None, reg_param=0.0, store_covariance=False,
    tol=0.0001)
```

QuadraticDiscriminantAnalysis

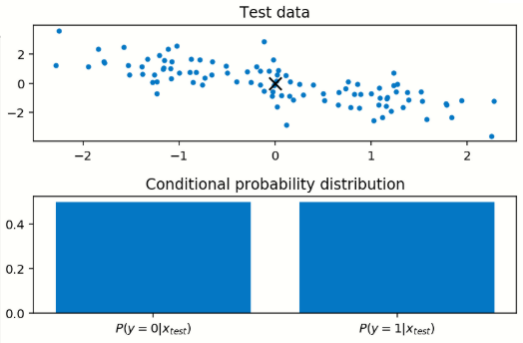
- priors_ : ndarray, 각 클래스 k의 사전확률. None이면 훈련 데이터로부터 유추됨
- reg_param : float, 기본값 0.0, S2를 다음과 같이 변환하여 클래스별 공분산 추정을 정규화 함. S2는 주어진 클래스의 scaling_ 속성.
  S2 = (1 − reg_param) * S2 + reg_param * np.eye(n_features)
- store_covariance : bool, 기본값 False, True인 경우 클래스 공분산 행렬이 계산되어 self.covariance_ 속성에 저장됨
- tol : float, 기본값 1.0e−4, 특이값으로 간주되는 절대 임계값. 절대 임계값은 Xk의 등급을 추정하는데 사용되며, 여기서 Xk는 k등급의 샘플 중심 행렬임. 이 매개변수는 예측에 영향을 미치지 않음. 피쳐가 동일 선상에 있다고 간주 될 때 발생하는 경고만 제어함

QDA


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

```
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=False, tol=0.0001)
```

```
1  x = [[0, 0]]
2  p = model.predict_proba(x)[0]
3  plt.subplot(211)
4  plt.scatter(X.T[0], X.T[1], s=10)
5  plt.scatter(x[0][0], x[0][1], c='k', s=100, marker='x')
6  plt.title("Test data")
7
8  plt.subplot(212)
9  plt.bar(model.classes_, p)
10 plt.title("Conditional probability distribution")
11 plt.xticks(model.classes_, ["$P(y=0|x_{test})$", "$P(y=1|x_{test})$"])
12 plt.tight_layout()
13 plt.show()
```



나이브 가정(naïve assumption)

- 독립변수 x가 D차원이라고 가정했을 경우 x=(x1,⋯, xD)
- 가능도함수는 x1,⋯,xD의 결합확률이 됨
  P(x | y=k) = P(x1, ⋯, xD | y=k)
- 원리상으로는 y=k인 데이터만 모아서 이 가능도함수의 모양을 추정할 수 있음
- 그러나 차원 D가 커지면 가능도 함수의 추정이 현실적으로 어려워지기 때문에 나이즈베이즈 분류모형에서는 모든 차원의 개별 독립변수가 서로 조건부독립(conditional independent) 이라는 가정을 사용

사이킷런은 3가지 나이브베이즈 모형을 클래스를 제공

- GaussianNB: 정규분포 나이브베이즈 ; 연속형 실수
- BernoulliNB: 베르누이분포 나이브베이즈 : 0또는 1 이진데이터
- MultinomialNB: 다항분포 나이브베이즈 : 이산형 데이터

```
sklearn.naive_bayes.MultinomialNB(*,
        alpha=1.0, fit_prior=True, class_prior=None)
```

MultinomialNB

- 다항분포 가능도 함수를 사용하는 나이브베이즈 모형
- 다항분포 가능도 모형을 기반으로 하는 나이브베이즈 모형은 주사위를 던진 결과로부터 1,…,K 중 어느 주사위를 던졌는지를 찾아내는 모형
- alpha : float, 기본값 1.0, Additive(Laplace/Lidstone) smoothing 파라미터
- fit_prior : bool, 기본값 True, 클래스 사전확률을 배울지 여부. False 이면 균일한 prior
- class_prior : (n_classes,) 모양 배열, 기본값 None, 수업의 사전확률. 지정된 경우 사전(prior) 데이터는 데이터에 따라 조정되지 않음

나이브베이즈

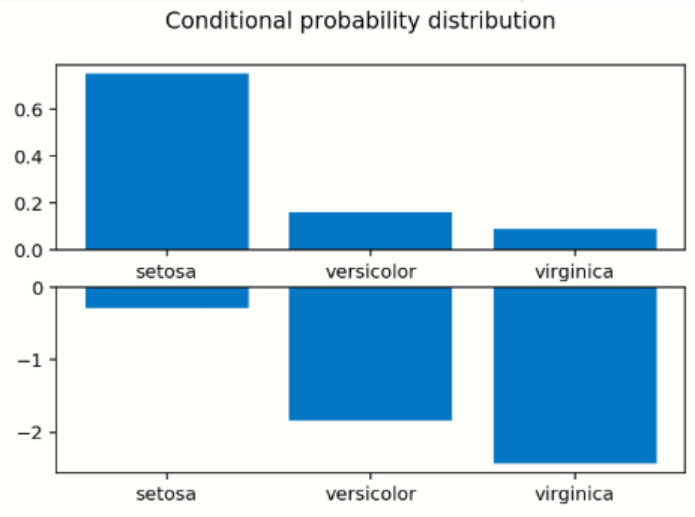

```
1 from sklearn.naive_bayes import MultinomialNB
2 model = MultinomialNB()
3 model.fit(X, y)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
1 test_X = [[5.0, 3.4, 1.2, 0.25]]
2 model.predict(test_X)
```

```
array(['setosa'], dtype='<U10')
```

```
1 %matplotlib inline
2 import matplotlib.pyplot as plt
3 plt.subplot(211)
4 plt.bar(model.classes_, model.predict_proba(test_X)[0])
5 plt.xticks(model.classes_)
6 plt.subplot(212)
7 plt.bar(model.classes_, model.predict_log_proba(test_X)[0])
8 plt.xticks(model.classes_)
9 plt.suptitle("Conditional probability distribution")
10 plt.show()
11
```



조건부확률 p가 x에 대한 함수 f(x)로 표시될 수 있다고 가정하고 그 함수를 찾아내는 방법

$$p(y = k|x) = f(x)$$

로지스틱 회귀(Logistic Regression)와 의사결정나무(Decision Tree)가 있음

로지스틱 회귀 모형

종속변수가 이항분포를 따르고 그 모수 μ가 독립변수 x에 의존한다고 가정

$$p(y|x) = Bin(y; \mu(x), N)$$

종속변수 y가 0 또는 1인 분류 문제를 풀 때는 x값을 이용하여 μ(x)를 예측한 후 다음 기준에 따라 $\hat{y}$ 값을 출력

$$\hat{y} = \begin{cases} 1 & if\ \mu(x) \geq 0.5 \\ 0 & if\ \mu(x) < 0.5 \end{cases}$$

```
sklearn.linear_model.LogisticRegression(penalty='l2', *,
        dual=False, tol=0.0001, C=1.0,
        fit_intercept=True, intercept_scaling=1,
        class_weight=None, random_state=None,
        solver='lbfgs', max_iter=100,
        multi_class='auto', verbose=0, warm_start=False,
        n_jobs=None, l1_ratio=None)
```

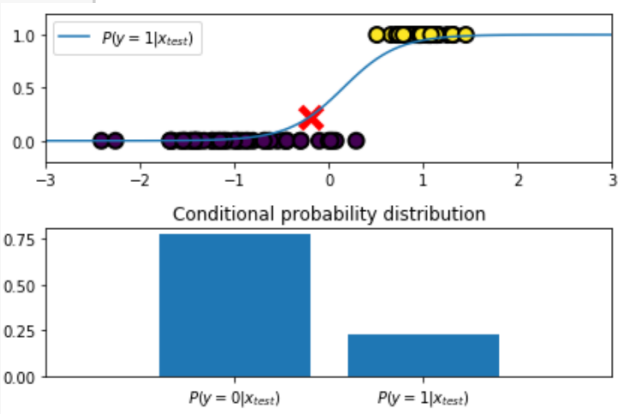
로지스틱 회귀 모형

분류 예측

```
import numpy as np
xx = np.linspace(-3, 3, 100)
XX = xx[:, np.newaxis]
prob = model.predict_proba(XX)[:, 1]
```

시각화

```
import matplotlib.pyplot as plt
%matplotlib inline
x_test = [[-0.2]]
plt.subplot(211)
plt.plot(xx, prob)
plt.scatter(X, y, marker='o', c=y, s=100, edgecolor='k', linewidth=2)
plt.scatter(x_test[0], model.predict_proba(x_test)[0][1:], marker='x',
            s=200, c='r', lw=5)
plt.xlim(-3, 3)
plt.ylim(-.2, 1.2)
plt.legend(["$P(y=1|x_{test})$"])
plt.subplot(212)
plt.bar(model.classes_, model.predict_proba(x_test)[0])
plt.xlim(-1, 2)
plt.gca().xaxis.grid(False)
plt.xticks(model.classes_, ["$P(y=0|x_{test})$", "$P(y=1|x_{test})$"])
plt.title("Conditional probability distribution")
plt.tight_layout()
plt.show()
```



여러 가지 규칙을 순차적으로 적용하면서 독립변수 공간을 분할하는 분류 모형

분류(classification)와 회귀분석(regression)에 모두 사용될 수 있으므로 CART(Classification And Regression Tree)라고도 함

의사결정나무를 이용한 분류법

- 여러가지 독립변수 중 하나의 독립변수를 선택하고 그 독립변수에 대한 기준값(threshold)을 정함(최적의 분류 규칙을 찾는 방법이 있어야함)
- 전체 학습 데이터 집합(부모 노드)을 해당 독립변수의 값이 기준값보다 작은 데이터 그룹(자식 노드 1)과 해당 독립변수의 값이 기준값보다 큰 데이터 그룹(자식 노드 2)으로 나눔
- 각각의 자식 노드에 대해 1~2의 단계를 반복하여 하위의 자식 노드를 만듬. 단, 자식 노드에 한가지 클래스의 데이터만 존재한다면 더 이상 자식 노드를 나누지 않고 중지함
- 자식 노드 나누기를 연속적으로 적용하면 노드가 계속 증가하는 나무(tree)와 같은 형태로 표현할 수 있음

분류 규칙을 정하는 방법

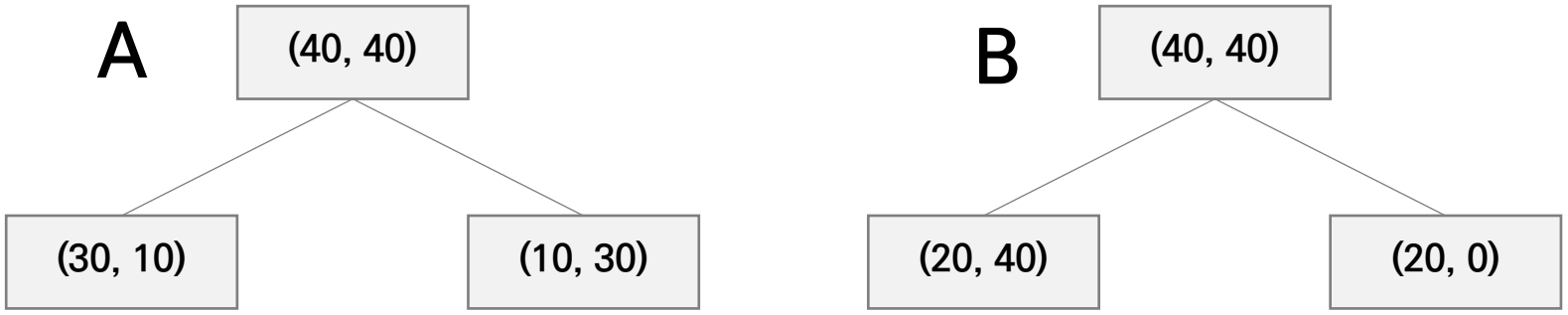
- 부모 노드와 자식 노드 간의 엔트로피를 가장 낮게 만드는 최상의 독립변수와 기준값을 찾는 것
- 이러한 기준을 정량화한 것이 정보획득량(information gain)
- 정보획득량(information gain)은 X라는 조건에 의해 확률변수 Y의 엔트로피가 얼마나 감소하였는가를 나타내는 값

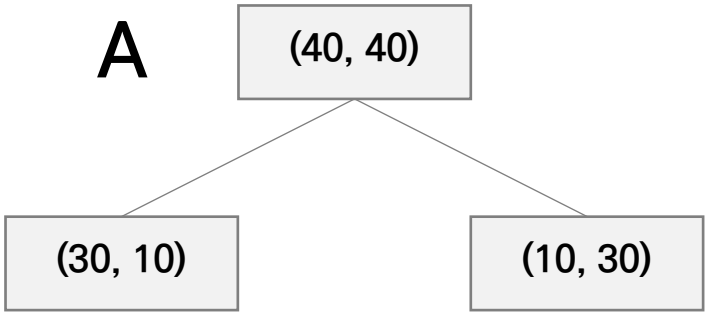
$$H[Y] = -\sum_i y_i' \log(y_i)$$

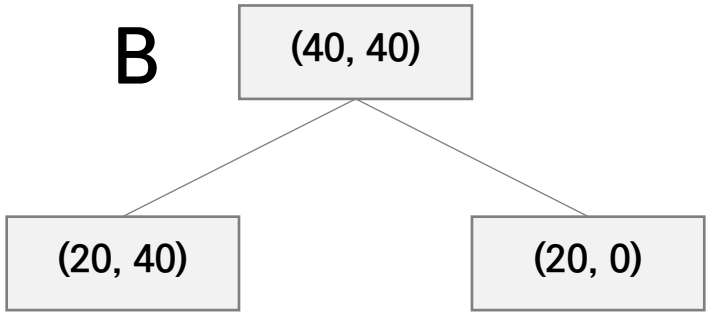
- Y의 엔트로피에서 X에 대한 Y의 조건부 엔트로피를 뺀 값으로 정의

$$IG[Y,X] = H[Y] - H[Y|X]$$

아래 두 분류 중 더 좋은 방법은?







$$H[Y] = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = \frac{1}{2} + \frac{1}{2} = 1$$

부모 노드의 엔트로피

A

$$H[Y|X = X_1] = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) = 0.81$$

$$H[Y|X = X_2] = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.81$$

$$H[Y|X] = \frac{1}{2}H[Y|X = X_1] + \frac{1}{2}H[Y|X = X_2] = 0.81$$

$$IG = H[Y] - H[Y|X] = 0.19$$

B

$$H[Y|X = X_1] = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right) = 0.92$$

$$H[Y|X = X_2] = 0$$

$$H[Y|X] = \frac{3}{4}H[Y|X = X_1] + \frac{1}{4}H[Y|X = X_2] = 0.69$$

$$IG = H[Y] - H[Y|X] = 0.31$$

```
sklearn.tree.DecisionTreeClassifier(*, criterion='gini',
        splitter='best', max_depth=None,
        min_samples_split=2, min_samples_leaf=1,
        min_weight_fraction_leaf=0.0, max_features=None,
        random_state=None, max_leaf_nodes=None,
        min_impurity_decrease=0.0,
        min_impurity_split=None, class_weight=None,
        presort='deprecated', ccp_alpha=0.0)
```

- 의사결정모형을 만듬, Depth는 1로 설정

```
from sklearn.datasets import load_iris
data = load_iris()
X = data.data[:, 2:]
y = data.target
feature_names = data.feature_names[2:]
```

```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(criterion='entropy', max_depth=1, random_state=0)
dt_model.fit(X, y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=1,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

- 의사결정모형의 분류 과정을 트리로 시각화

```
import io
from sklearn.tree import export_graphviz
# conda install graphviz
import pydot # pip install pydot
from IPython.core.display import Image
```

```
def draw_decision_tree(model, feature_names):
    dot_buf = io.StringIO()
    export_graphviz(model, out_file=dot_buf, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dot_buf.getvalue())[0]
    image = graph.create_png()
    return Image(image)

```

의사결정나무

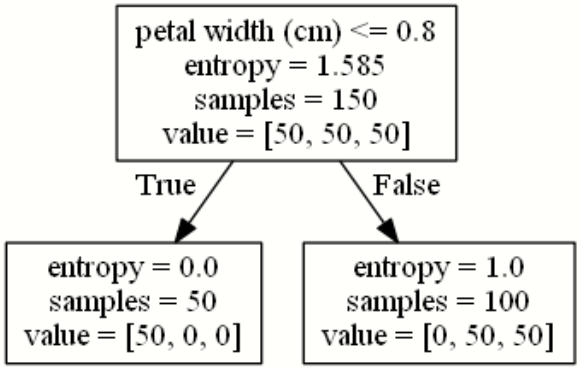
2절. 분류분석 모형

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

 def plot_decision_regions(X, y, model, title):
    species = ['setosa', 'versicolor', 'virginica']
    resolution = 0.01
    markers = ('s', '^', 'o')
    colors = ('red', 'blue', 'lightgreen')
    cmap = mpl.colors.ListedColormap(colors)

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = model.predict(
       np.array([xx1.ravel(), xx2.ravel()]).T).reshape(xx1.shape)

    plt.contour(xx1, xx2, Z, cmap=mpl.colors.ListedColormap(['k']))
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
       plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
                   c=[cmap(idx)], marker=markers[idx], s=80, label=species[cl])

    plt.xlabel(data.feature_names[2])
    plt.ylabel(data.feature_names[3])
    plt.legend(loc='upper left')
    plt.title(title)

    return Z
```

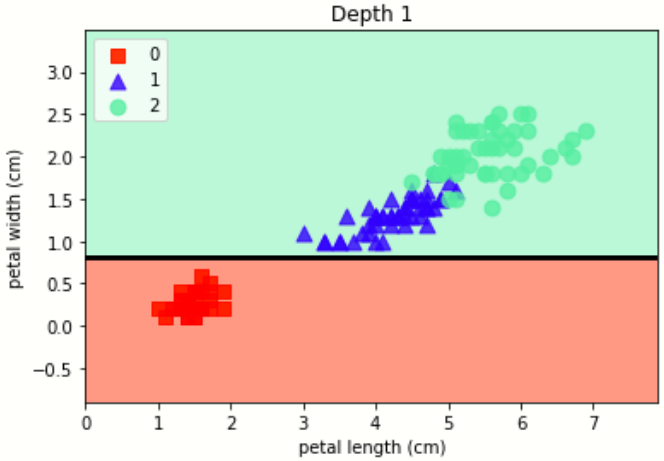
- 의사결정모형에 의해 데이터의 영역이 어떻게 나눠졌는지 시각화

의사결정나무


```
1 draw_decision_tree(dt_model, feature_names=data.feature_names[2:])
```



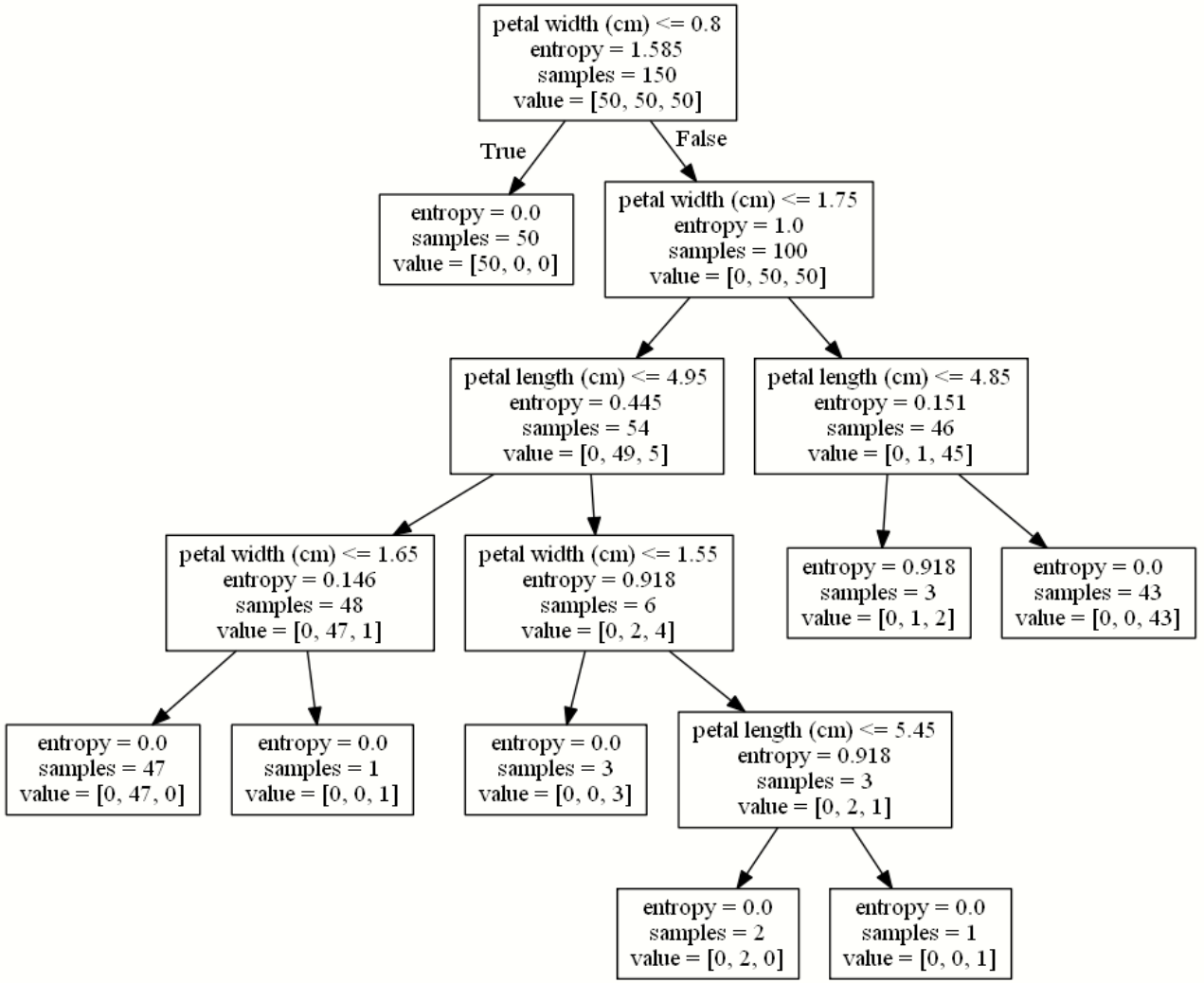

```
1 plot_decision_regions(X, y, dt_model, "Depth 1")
2 plt.show()
```



- max_depth=1인 모형

의사결정나무

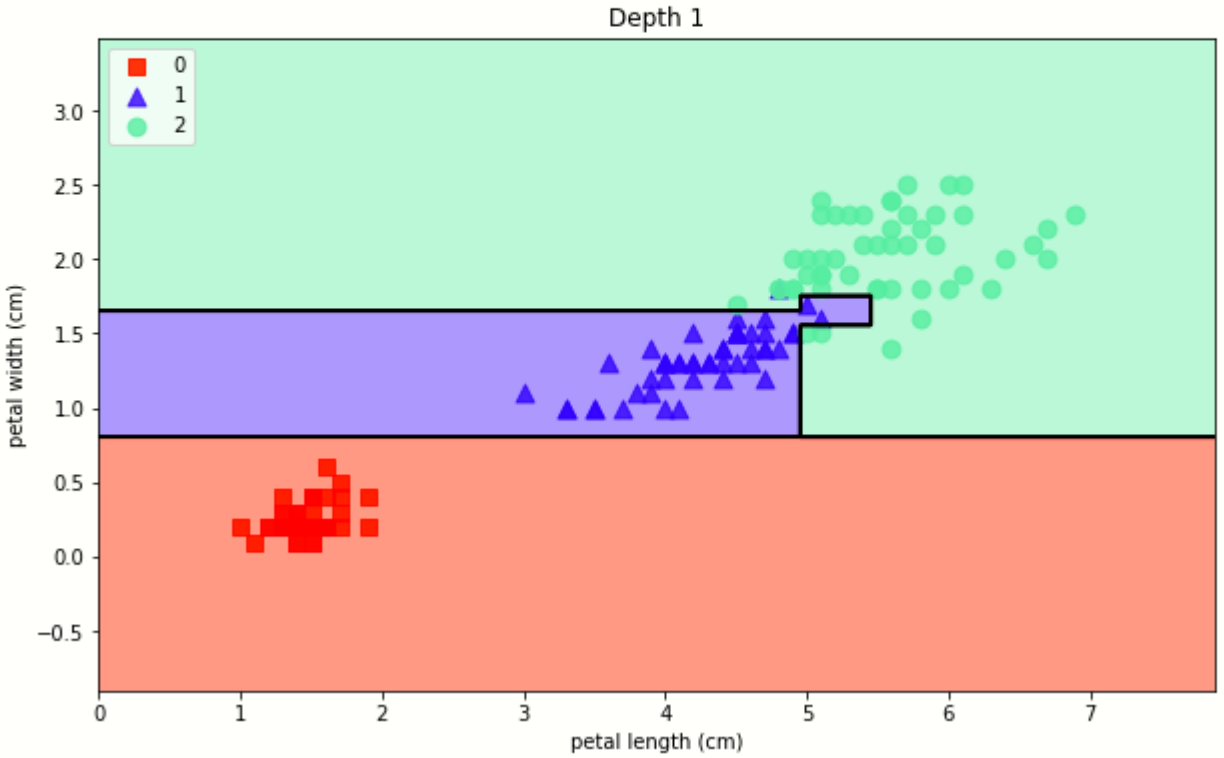

```
1 draw_decision_tree(dt_model5, feature_names=data.feature_names[2:])
```



- max_depth=5인 모형

의사결정나무

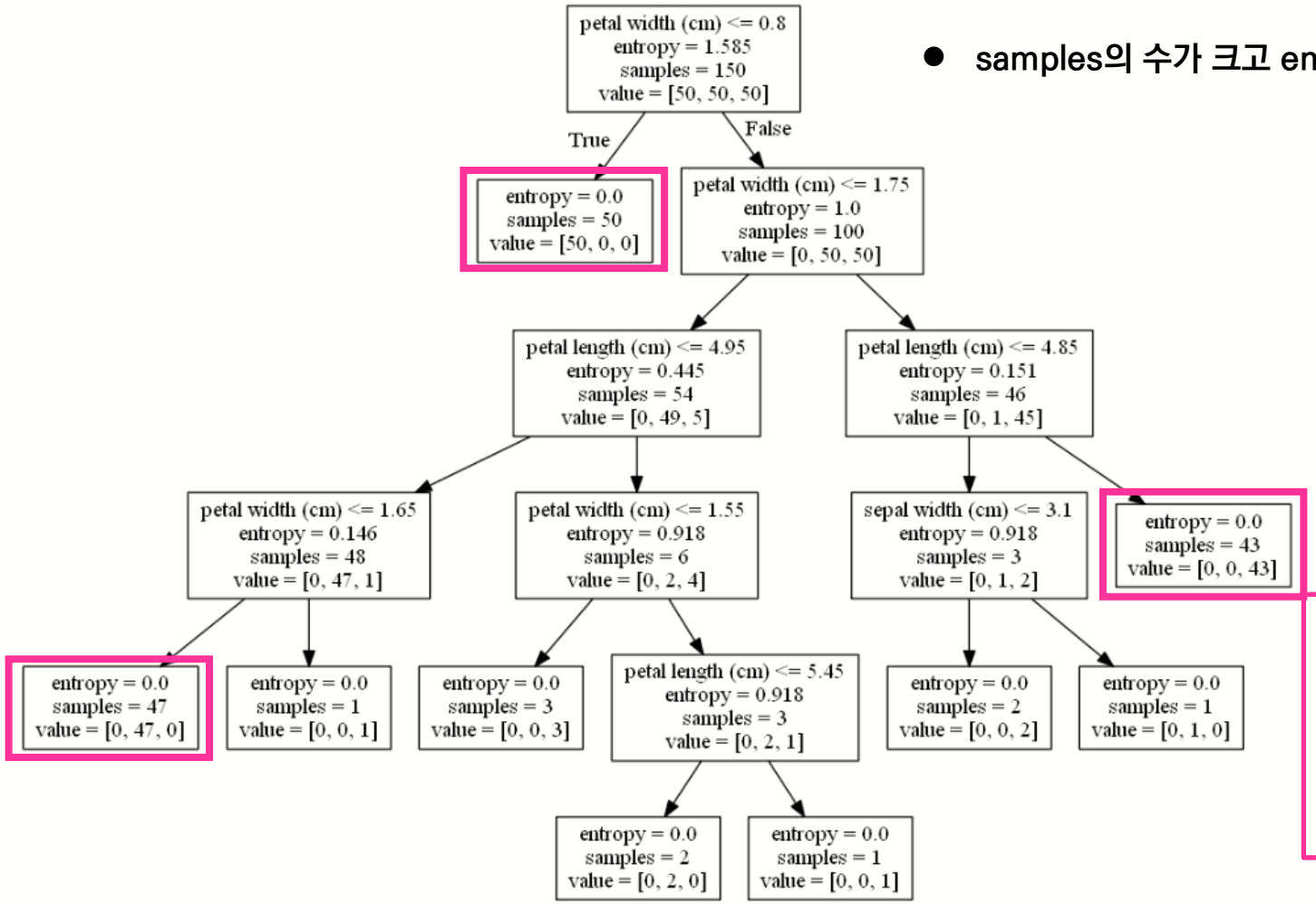

```
1 plt.figure(figsize=(10,6))
2 plot_decision_regions(X, y, dt_model5, "Depth 1")
3 plt.show()
```



- max_depth=5인 모형

의사결정나무


```
1 draw_decision_tree(dt_model6, feature_names=data.feature_names)
```



- samples의 수가 크고 entropy가 작은 노드를 찾으면…

Sample의 수가 크고, **entropy**가 작은 노드를 찾고 그 노드까지 내려오는 상위 노드들의 변수의 값 범위를 통해 분류에 영향을 주는 변수와 값의 변위를 알 수 있음

판별함수 기반 모형

- 동일한 클래스가 모여 있는 영역과 그 영역을 나누는 경계면(boundary plane)을 정의
- 이 경계면은 경계면으로부터의 거리를 계산하는 형태의 함수인 판별함수로 정의됨
- 클래스는 판별함수값의 부호에 따라 나뉨(경계선 f(x)=0, 클래스 1 f(x)>0, 클래스 0 f(x)<0
- Scikit-learn의 판별함수 모형은 판별함수 값을 출력하는 decision_function()을 제공
- 퍼셉트론, 커널 SVM, 인공신경망 모형 등이 있음

퍼셉트론

- 퍼셉트론(Perceptron)은 가장 단순한 판별함수 모형
- 판별함수(discriminant function) 기반 모형은 동일한 클래스가 모여 있는 영역과 그 영역을 나누는 경계면(boundary plane)을 정의
- 이 경계면은 경계면으로부터의 거리를 계산하는 형태의 함수인 판별함수로 정의
- Scikit-learn에서 판별함수 기반 모형은 판별함수 값을 출력하는 decision_function() 함수를 제공

```
1  from sklearn.datasets import load_iris
2  import numpy as np
3  iris = load_iris()
4  idx = np.in1d(iris.target, [0, 2])
5  X = iris.data[idx, 0:2]
6  y = iris.target[idx]
```

분류를 위한 데이터셋

```
1  from sklearn.linear_model import Perceptron
2  model = Perceptron(max_iter=100, eta0=0.1, random_state=1).fit(X, y)
```

퍼셉트론 모형

3장. 분류분석

3절. 인공신경망

인공지능의 역사 및 딥러닝의 혁신

~1990년 : 이론 정립 → ~2000년 : 구현 시도 → ~2010년 : 본격 시도 → 2010년~ : 혁신의 시작 (딥러닝 기반의 인공지능)

시대별 한계 :
- ~1990년 : 컴퓨팅의 한계로 제안된 이론 구현의 어려움
- ~2000년 : 데이터의 한계로 현실 문제 해결하지 못함
- ~2010년 : 알고리즘의 한계로 완성도 부족

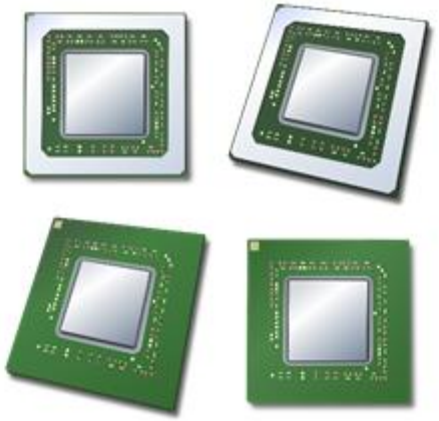
혁신적 알고리즘 제안

Geoffrey Hinton (Univ. of Toronto, Google)

- 혁신적 딥러닝 이론 제안(2006년)
- 실제 구현으로 혁신적 성능 증명
- 이미지 인식 대회인 ImageNet Challenge에서 압도적 성능으로 우승(2012년)

+

컴퓨팅 파워 발전



- CPU는 고성능화와 함께 저가격화

+

데이터 폭증

- 2020년 데이터의 크기는 40ZB (40조 GB) 크기

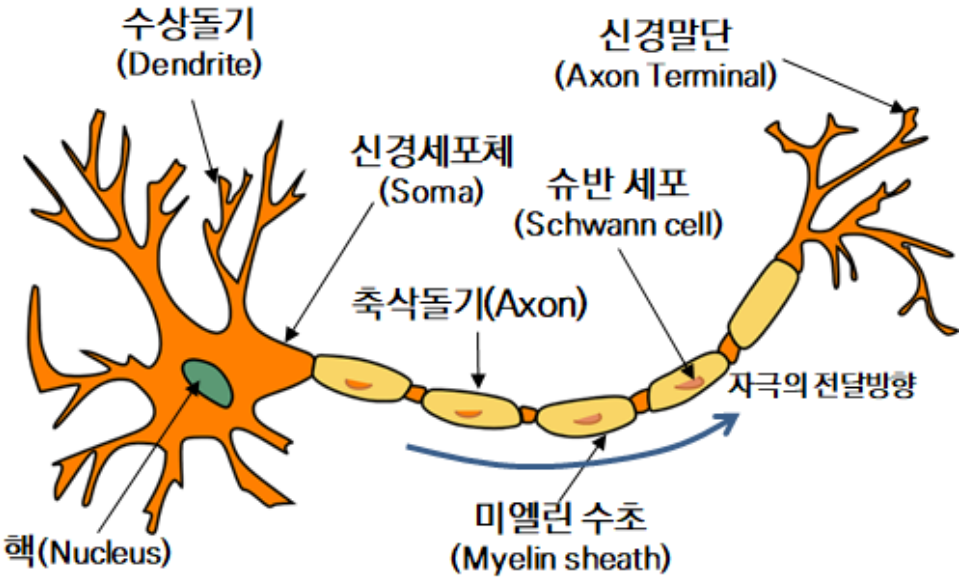
인간의 뉴런(neuron)

- 시냅스(synapse)를 통해 뉴런간 신호를 전달
- 각 뉴런은 수상돌기(dendrite)를 통해 입력 신호를 받음
- 입력 신호가 특정크기(threshold) 이상인 경우에만 활성화 되어 축삭돌기(axon)을 통해 다음 뉴런으로 전달

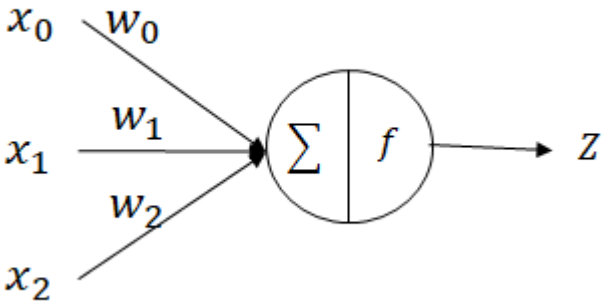
인공 뉴런(노드; node)

- 각 노드는 가중치가 있는 입력 신호를 받음
- 입력신호는 모두 더한 후, 활성화 함수(activation function)을 적용함
- 활성화 함수의 값이 특정 값 이상인 경우에만 다음 노드의 입력값으로 전달

인간의 뉴런 구조

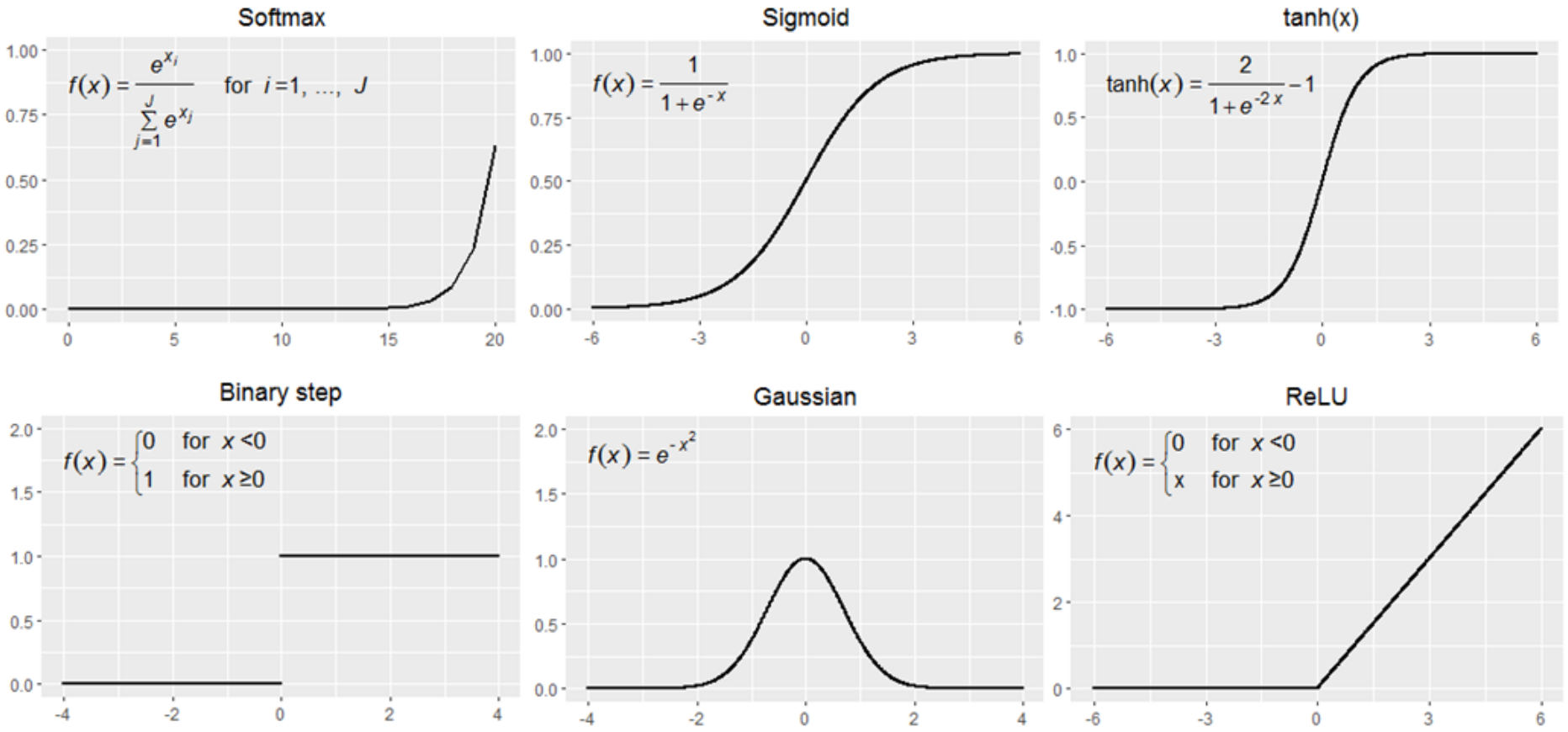


인공 뉴런의구조

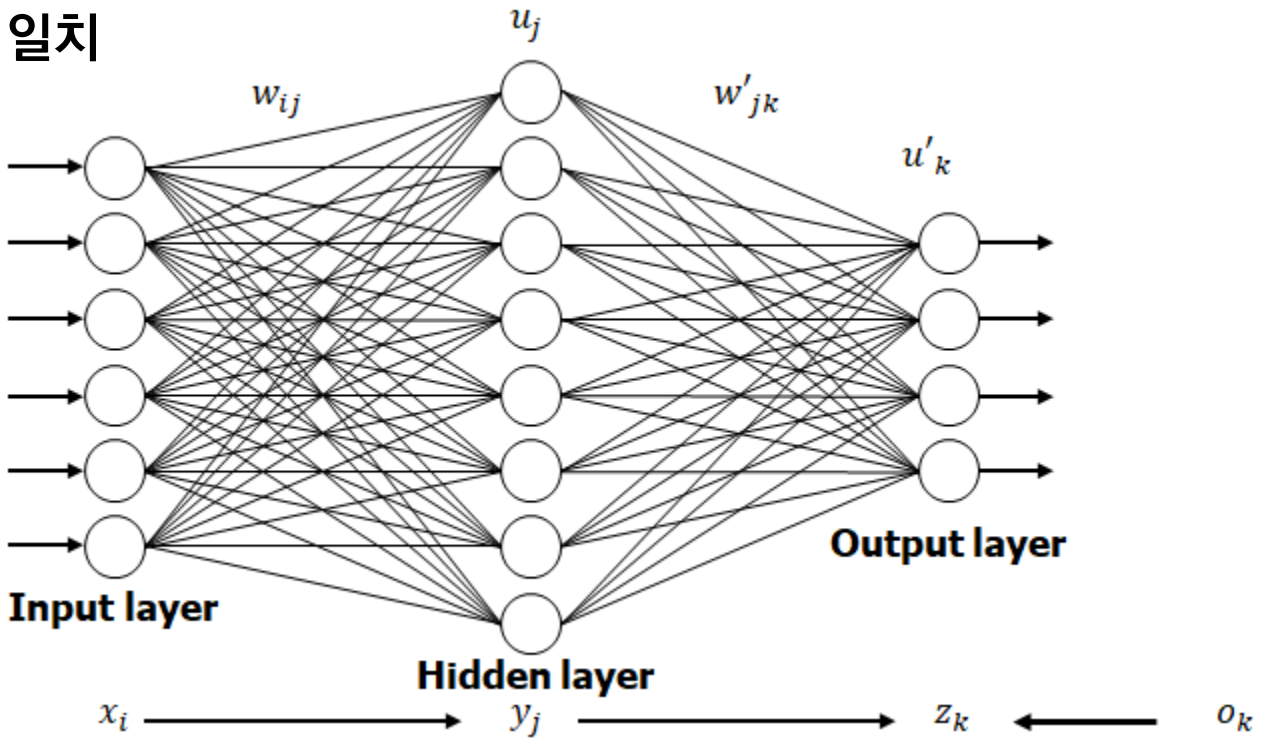


활성화 함수(activation function)

생물학적 뉴런(neuron)에서 입력 신호가 일정 크기 이상일 때만 신호를 전달하는 메커니즘을 모방한 함수



- 인공신경망은 입력층(Input layer), 은닉층(Hidden layer), 출력층(Output layer)으로 구성
- 각 층의 뉴런들을 퍼셉트론(Perceptron)이라고 부르기도 함
  - 퍼셉트론(Perceptron)은 인공 뉴런의 한 종류
- 입력층의 뉴런의 수는 입력 데이터의 수이며, 출력층은 분류 문제를 해결할 경우에는 분류의 수와 일치

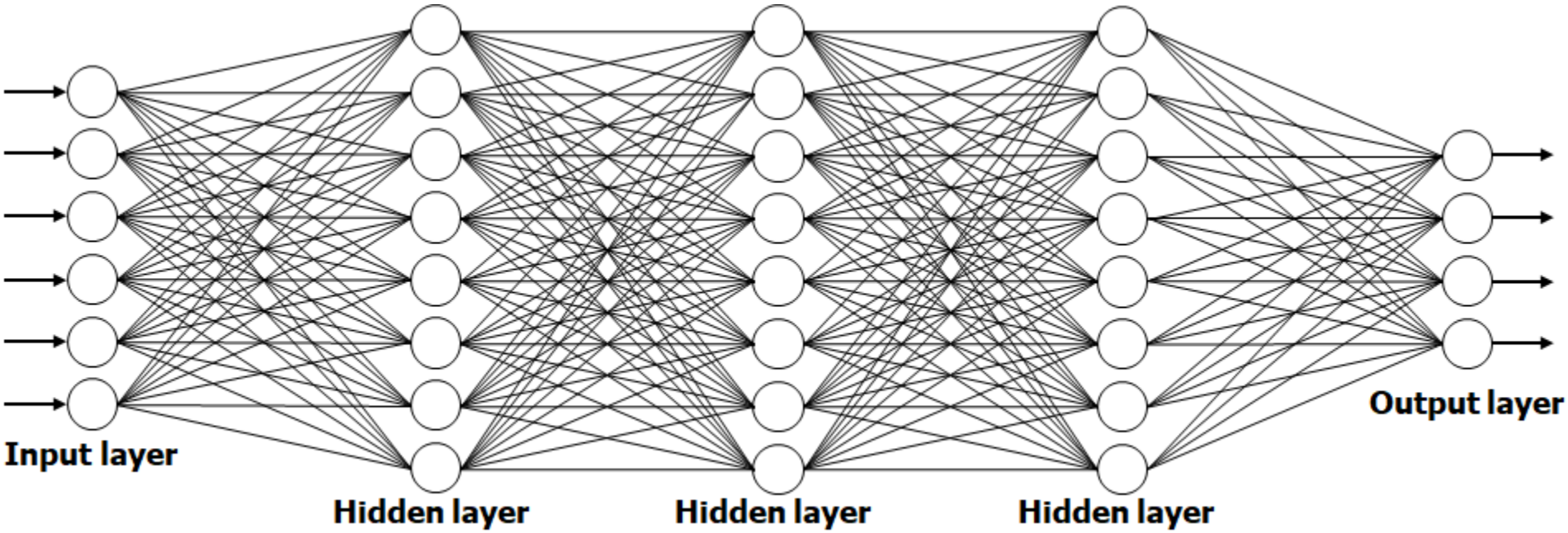


인공신경망 구성요소

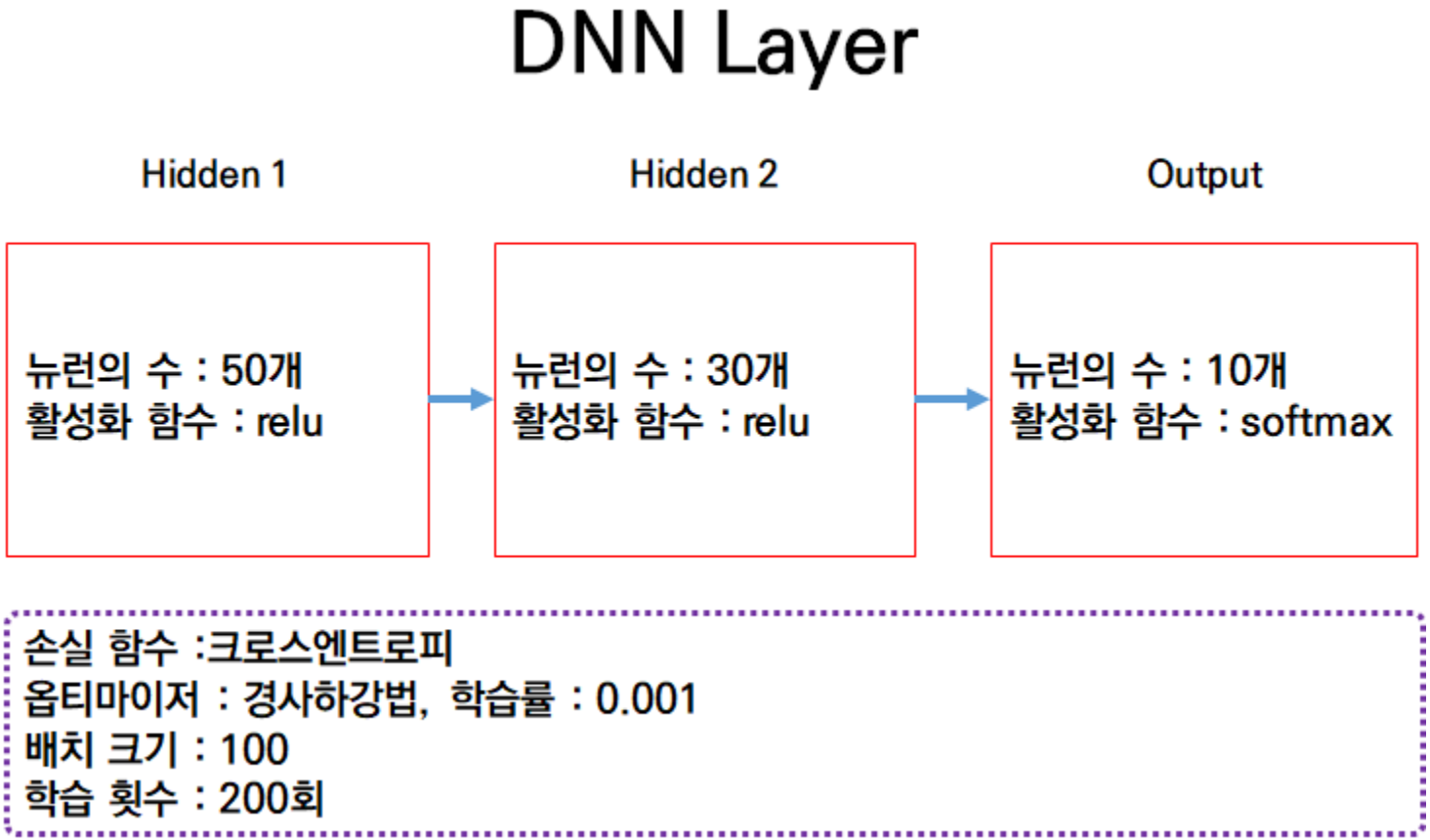
- 입력층(input layer)
  - 입력 값으로 구성된 레이어
  - 학습 데이터셋의 “입력 변수의 개수 노드 + 1”만큼의 노드로 구성
    - 0번째 원소는 바이어스(bias)로 값은 항상 1을 할당
- 출력층(output layer)
  - 모델의 출력값을 만들어 내는 레이어
  - 예) IRIS 품종 분류 문제
    - 멀티 클래스 분류 문제의 경우 소프트맥스(softmax) 함수를 출력함수로 사용
- 은닉층(hidden layer)
  - 입력층과 출력층 사이의 레이어
  - 뉴런(neuron)과 시냅스(synapse)로 구성된 인간의 두뇌를 모방하는 레이어
  - 은닉층으로 들어오는 입력값의 합을 계산한 후 활성화 함수를 적용
  - 활성화 함수 출력이 임계치를 넘지 않을 경우 다음 노드로 0을 전달(신호를 전달하지 않음)
  - 은닉층의 개수와 은닉 노드 개수
  - 너무 적으면 입력 데이터를 제대로 표현하지 못해 모델을 제대로 학습하지 못함
  - 너무 많으면 과적합(overfitting)이 발생하며, 학습 시간도 많이 소모

은닉 계층이 3개인 인공신경망

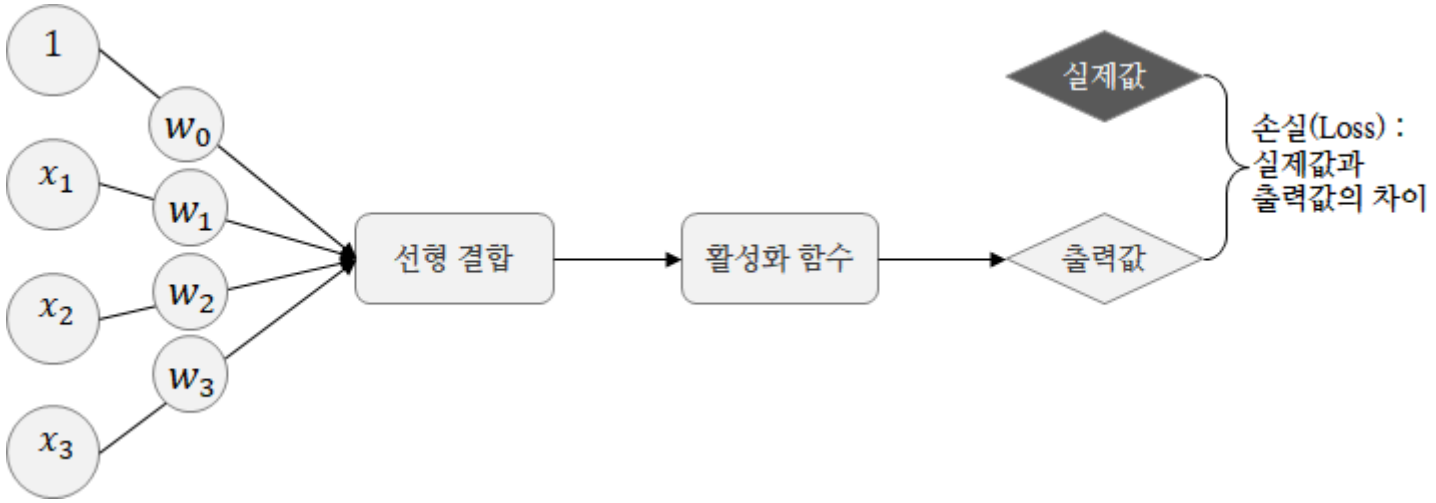
- 일반적으로 은닉 계층이 3개 이상 들어간 인공 신경망을 DNN
- 실제로 구현된 DNN의 경우에는 이렇게 간단하지 않음
- 은닉 계층이 60~90개 정도로 이뤄진 신경망도 많이 있음



딥러닝 인공신경망 네트워크의 간단한 표현

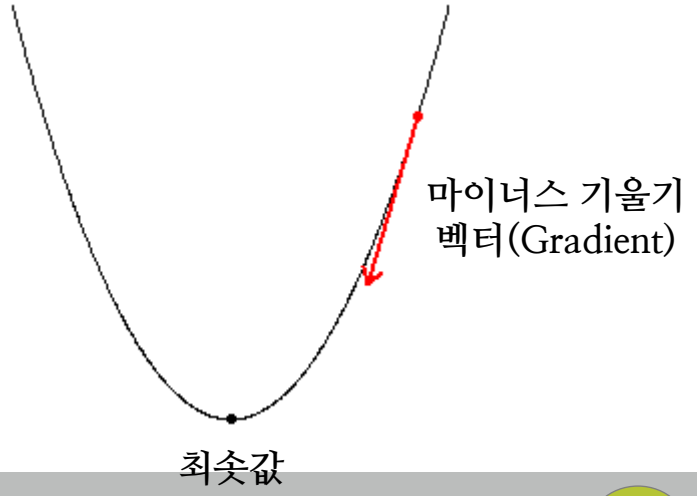


- 인공신경망에서 가중치(Weight)는 학습을 통해 손실(Loss)을 최소화 하는 방향으로 추정
- 추론한 값과 실제 값의 차이를 손실(Loss)로 정의하고 이러한 차이를 표현한 함수를 손실 함수(Loss Function)이라고 함



경사하강법

- 손실을 최소화 하는 방향은 기울기 벡터(Gradient)를 통해 정해짐
  - 마이너스 기울기 벡터(Gradient)는 현재 위치에서 함수의 최솟값으로 가는 가장 빠른 방향을 정해 줌
  - 이와 같은 과정을 반복하여 마이너스 기울기 벡터를 따라가게 되면 결국은 해당 함수의 최솟값으로 수렴할 수 있음
  - 이러한 방법을 경사하강법(Gradient Descent)이라고 함
- 인공신경망의 관점에서 설명하며, 학습을 통해 현재 가중치 값에 해당하는 손실함수의 마이너스 기울기 벡터를 구하게 되고 이런 과정을 반복해 손실을 최소화 하는 가중치 값을 추정하게 됨
- 이렇게 추정된 값을 기반으로 모델이 결정 됨



MLPClassifier

- Scikit-learn 패키지에는 다층신경망을 구현한 클래스
- LBFGS 또는 확률적 경사 하강법(SGD)을 사용하여 로그 손실 함수를 최적화

```
sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100, ),
                activation='relu', solver='adam', alpha=0.0001,
                batch_size='auto', learning_rate='constant',
                learning_rate_init=0.001, power_t=0.5, max_iter=200,
                shuffle=True, random_state=None, tol=0.0001,
                verbose=False, warm_start=False, momentum=0.9,
                nesterovs_momentum=True, early_stopping=False,
                validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
                epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

MLPClassifier로 분류하기

iris 데이터 분류하기

```
import seaborn as sns
```

```
iris = sns.load_dataset("iris")
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
le.fit(iris.species)
```

```
LabelEncoder()
```

```
iris.species = le.transform(iris.species)
```

```
from sklearn.model_selection import train_test_split
```

```
iris_X = iris.iloc[:,:-1]
iris_y = iris.species # iris.iloc[:,-1]
```

```
train_X, test_X, train_y, test_y = \
train_test_split(iris_X, iris_y, test_size=0.3,
                 random_state=1)
```

```
train_X.shape, test_X.shape
```

```
((105, 4), (45, 4))
```

```
mlp = MLPClassifier(hidden_layer_sizes=(50,50,30),
                    max_iter=500)
```

```
mlp.fit(train_X, train_y)
```

```
pred = mlp.predict(test_X)
```

```
pred
```

```
array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1,
2, 0, 0, 1, 2,
       2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1,
2, 2, 0, 2, 2,
       1])
```

```
mlp.score(test_X, test_y)
```

```
0.9333333333333333
```

인공신경만 모형의 파라미터

인공신경망 모형을 정의할 때에 연구자가 고려해야 할 사항

- 레이어의 수 : 인공신경망 모형에서 은닉층의 수를 지정하는 것
- 뉴런의 수 : 입력층의 뉴런의 수는 입력데이터의 변수의 수와 동일해야 하며, 출력층의 뉴런의 수는 분류분석의 경우 분류 레이블의 수이며, 회귀분석의 경우 1개.
- 활성화 함수 : 각 계층에서 사용할 활성화 함수를 지정하는 것
- 손실함수 : 예측한 값과 실제 값과의 차이를 계산하는 함수를 지정
- 옵티마이저 : 손실함수를 최소화 하도록 가중치를 갱신시키기 위한 옵티마이저
- 학습률 : 옵티마이저가 사용할 학습률
- 학습 횟수(epoch) : 모든 데이터가 입력되어 가중치가 업데이트 되는 학습 횟수
- 배치 크기 : 1회 epoch가 학습될 동안 가중치가 업데이트 되어야 하는 입력데이터의 크기(batch_size)

Scikit-learn MLPClassifier vs. Tensorflow DNNClassifier

가장 단순한 피드 포워드 신경망의 구현, 따라서 원칙적으로 동일

Scikit-learn MLPClassifier

- Scikit-learn의 개발자들은 보다 전통적인 머신러닝 영역에 초점을 맞추고 딥러닝 영역으로 너무 많이 확장하지 않도록 신중하게 선택했음
- 작은 크기의 데이터를 이용해서 분류 모형을 만들 수 있지만 그 이상은 쉽지 않음

Tensorflow DNNClassifier

- Tensorflow는 딥러닝에 전념하므로 매우 복잡한 딥러닝 모델을 구성 할 수 있음
- Tensorflow는 LeakyReLU, ELU 등의 보다 복잡한 활성화 함수를 사용하거나 배치 정규화, 드롭아웃 등을 추가

3장. 분류분석

4절. 분류모형 성능평가

Scikit-learn의 모형 평가 방법

예측 모형의 score 메소드

- 예측 모형들은 score() 메소드를 통해 예측 모형을 평가할 수 있는 기본 기준을 제공합니다.

metrics 함수

- 메트릭(sklearn.metrics) 모듈은 분류, 회귀 그리고 군집모형 등 예측 모형의 평가를 위한 함수들을 제공합니다.

scoring 매개변수

- sklearn.model_selection 모듈의 cross_val_score() 함수 또는 GridSearchCV 클래스 등 교차 검증(cross-validation)을 사용하는 모형 평가 도구들은 내부적으로 scoring 매개변수를 이용해 모형 평가 규칙을 정의합니다.
- scoring 매개변수의 가능한 값은 sklearn.metrics.SCORES.keys() 함수를 통해 알 수 있습니다.

사이킷런의 분류모형 성능 평가 함수

이진 분류 모델에서 사용

함수	설명
precision_recall_curve(y_true, probas_pred)	다른 확률 임계값에 대한 **정밀도와 재현율** 쌍을 계산
roc_curve(y_true, y_score[, pos_label, …])	ROC(Receiver operating characteristic)를 계산
balanced_accuracy_score(y_true, y_pred[, …])	균형 잡힌 정확도를 계산

멀티 클래스에서 사용

함수	설명
cohen_kappa_score(y1, y2[, labels, weights, …])	두 평가자의 평가가 얼마나 일치하는지 평가하는 코헨의 카파(Cohen's kappa)를 계산
confusion_matrix(y_true, y_pred[, labels, …])	분류 정확도를 평가하기 위한 혼동 행렬을 계산
hinge_loss(y_true, pred_decision[, labels, …])	평균 힌지 손실(Hinge loss)을 계산
matthews_corrcoef(y_true, y_pred[, …])	MCC(Matthews correlation coefficient)를 계산

멀티 라벨에서 사용

함수	설명
accuracy_score(y_true, y_pred[, normalize, …])	분류 정확도를 계산합니다.
classification_report(y_true, y_pred[, …])	분류 평가 보고서를 만듭니다.
f1_score(y_true, y_pred[, labels, …])	F1 점수를 계산합니다.(balanced F-score 또는 F-measure라고 부릅니다.
fbeta_score(y_true, y_pred, beta[, labels, …])	F-beta를 계산합니다.
hamming_loss(y_true, y_pred[, labels, …])	평균 해밍 손실(Hamming loss)을 계산합니다. 해밍 손실은 멀티 클래스 분류에서 가장 널리 사용되는 손실함수입니다.
jaccard_similarity_score(y_true, y_pred[, …])	자카드 유사도(Jaccard similarity)를 계산합니다.
log_loss(y_true, y_pred[, eps, normalize, …])	로그 손실(로지스틱 손실 또는 크로스 엔트로피 손실)을 계산합니다.
precision_recall_fscore_support(y_true, y_pred)	각 클래스에 대해 precision, recall, F-measure를 계산합니다.
precision_score(y_true, y_pred[, labels, …])	정밀도를 계산합니다.
recall_score(y_true, y_pred[, labels, …])	재현율을 계산합니다.
zero_one_loss(y_true, y_pred[, normalize, …])	Zero-one 분류 손실을 계산합니다.

분류표(혼동 행렬)

- 클래스가 0과 1 두 종류만 있을 경우에 클래스 이름을 "Positive"와 "Negative"로 표시
- 분류 모형의 예측 결과가 맞은 경우, 즉 Positive를 Positive라고 예측하거나 Negative를 Negative라고 예측한 경우에는 "True"
- 예측 결과가 틀린 경우, 즉 Positive를 Negative라고 예측하거나 Negative를 Positive라고 예측한 경우에는 "False"

		예측 클래스 N	예측 클래스 Y
실제 클래스	N	True/Negative 실제 N. 예측 Y	False/Positive 실제 N, 예측 Y
실제 클래스	Y	False/Negative 실제 Y. 예측 Y	True/Positive 실제 Y, 예측 Y

분류표 API

Scikit-learn 패키지의 confusion_matrix() 함수

Pandas 패키지의 crosstab() 함수

```
sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None,
                                 sample_weight=None)
```

```
pandas.crosstab(index, columns, values=None, rownames=None,
                colnames=None, aggfunc=None, margins=False,
                margins_name='All', dropna=True, normalize=False)
```

혼동행렬


```
1 y_true = [1, 1, 0, 0, 2, 1, 0, 2, 2]
2 y_pred = [1, 1, 0, 1, 1, 0, 0, 2, 1]
```

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_true, y_pred)
```

```
array([[2, 1, 0],
       [1, 2, 0],
       [0, 2, 1]], dtype=int64)
```

```
1 import pandas as pd
2 df = pd.DataFrame({"y_true":y_true, "y_pred":y_pred})
3 pd.crosstab(df.y_true, df.y_pred, margins=True)
```

y_pred	0	1	2	All
y_true				
0	2	1	0	3
1	1	2	0	3
2	0	2	1	3
All	3	5	1	9

혼동행렬 예

		Predicted Target		
		N(일반 고객)	Y(보험 사기자)	All
Actual Target	일반 고객 (Negative)	1613 True/Negative 실제 N. 예측 N	22 False/Positive 실제 N, 예측 Y	1635(91.19%)
	보험 사기자 (Positive)	81 False/Negative 실제 Y. 예측 N	77 True/Positive 실제 Y, 예측 Y	158(8.81%)
	All	1694	99	1793

True : 1690

- 실제 일반 고객을 일반 고객(Negative)으로 바르게 분류(True)한 고객의 수가 1613명
- 실제 일반 고객을 보험사기자(Positive)로 틀리게 분류(False)한 고객의 수가 22명
- 실제 보험사기자를 일반 고객(Negative)으로 틀리게 분류(False)한 고객의 수가 81명
- 실제 보험사기자를 보험사기자(Positive)로 바르게 분류(True)한 고객의 수가 77명

혼동행렬을 이용한 분류모형 평가

메트릭	계산식	의미
정확도 (Accuracy)	(TP+TN)/(TP+FP+FN+TN)	전체 예측에서 옳은 예측의 비율
정밀도 (Precision)	TP/(FP+TP)	Y으로 예측된 것 중 실제로도 Y인 경우의 비율
민감도 (Recall)	TP/(FN+TP)	실제로 Y인 것들 중 예측이 Y로 된 경우 비율(=Sensitivity)
특이도 (Specificity)	TN/(TN+FP)	실제로 N인 것들 중에서 예측이 N으로 된 경우의 비율(N의 Recall값)
오류율 (FP Rate, fallout)	FP/(TN+FP)	Y가 아닌데 Y로 예측된 비율.(=errorrate) 1-Specificity와 같은 값
F-measure	$2 * \frac{Precision * Recall}{Precision + Recall}$	Precision과 Recall의 조화 평균. 시스템의 성능을 하나의 수치로 표현하기 위해 사용하는 점수. 0~1사이의 값을 가짐. Precision과 Recall 두 값이 골고루 클 때 큰 값을 가짐.

혼동 행렬을 이용한 분류모형 평가

	CUST_ID	y_true	y_pred
0	37	0	0
1	51	0	0
2	60	0	0
3	65	0	0
4	73	0	0

```
1 from sklearn.metrics import accuracy_score
2 accuracy_score(result.y_true, result.y_pred)
```

0.9425543781372002

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

```
1 from sklearn.metrics import precision_score
2 precision_score(result.y_true, result.y_pred)
```

0.7777777777777778

$$precision = \frac{TP}{TP + FP}$$

혼동행렬을 이용한 분류모형 평가


```
1 from sklearn.metrics import recall_score
2 recall_score(result.y_true, result.y_pred)
```

$$recall = \frac{TP}{TP+FN}$$

0.4873417721518987

$$specificity = \frac{TN}{TN+FP}$$

```
1 recall_score(result.y_true, result.y_pred, pos_label=0)
```

0.9865443425076452

```
1 specificity = recall_score(result.y_true, result.y_pred, pos_label=0)
2 fallout = 1-specificity
3 fallout
```

$$fallout = \frac{FP}{TN+FP}$$

0.013455657492354778

```
1 from sklearn.metrics import f1_score
2 f1_score(result.y_true, result.y_pred)
```

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

0.5992217898832685

$$F_\beta = (1+\beta^2)(precision \times recall)/(\beta^2 precision + recall)$$

beta값이 2일 경우 Recall의 가중치가 Precision보다 높으며(False Negative를 더 강조), 0.5일 경우 Precision의 가중치가 Recall보다 더 높음. F1 score는 beta=1일 경우

ROC 커브를 이용한 성능 비교

ROC 커브는 클래스 판별 기준값의 변화에 따른 위양성률(fall-out)과 재현율(recall)의 변화를 시각화한 것

ROC 그래프는 가로축을 False Positive Rate(1-True Negative Rate(Specificity, TN/(TN+FP)) 값의 비율로 하고 세로축을 True Positive Rate(Sensitive(Recall), TP/P)로 하여 시각화한 그래프

분류 모형과 decision_function()

- 모든 이진 분류모형은 판별 평면으로부터의 거리에 해당하는 판별함수(discriminant function)를 가지며 판별함수 값이 음수이면 0인 클래스, 양수이면 1인 클래스에 해당
- 즉 0 이 클래스 판별 기준값이 되는데, ROC 커브는 이 클래스 판별 기준값이 달라진다면 판별 결과가 어떻게 달라지는지를 표현한 것
- Scikit-Learn의 분류모형을 위한 클래스들은 판별함수 값을 계산하는 decision_function() 함수를 제공

roc_curve()


```
sklearn.metrics.roc_curve(y_true, y_score,
        pos_label=None, sample_weight=None,
        drop_intermediate=True)
```

구문에서…

- y_true : array, shape=[n_samples], 실제 이진 레이블. 레이블이 {-1, 1} 또는 {0, 1}이 아닌 경우 pos_label을 명시적으로 제공해야 함
- y_score : array, shape=[n_samples], 목표 점수, 긍정(Positive) 클래스의 확률 추정치, 신뢰도 값 또는 임계값이 없는 결정값(일부 분류 모델은 "decision_function"에 의해 반환) 일 수 있음
- pos_label : int 또는 str, 기본값=None, 긍정(Positive) 클래스의 레이블. pos_label=None인 경우 y_true가 {-1, 1} 또는 {0, 1}에 있으면 pos_label이 1로 설정되고 그렇지 않으면 오류가 발생함
- sample_weight : (n_samples,) 모양 배열 형식, 기본값=None, 샘플 가중치
- drop_intermediate : boolean, 기본값=True, ROC 곡선에 나타나지 않는 일부 차선의 임계값을 제거할지 여부. 더 가벼운 ROC 곡선을 만드는 데 유용함. 0.17에 추가.
- 이 함수의 반환 값
  - fpr : 증가하는 False Positive Rate, tpr : 증가하는 True Positive Rate
  - threshold : fpr과 tpr을 계산하는데 사용되는 의사결정(decision_function) 함수의 감소하는 임계값. threshold[0]은 예측되는 인스턴스가 없음을 나타내며 임의로 max(y_score) + 1로 설정됨

두 모형의 혼동 행렬이 같을 경우…


```
from sklearn.datasets import make_classification
```

```
X, y = make_classification(n_samples=1000, weights=[0.95,0.05],
                           random_state=5)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```
model1 = LogisticRegression().fit(X, y)
```

```
model2 = SVC(gamma=0.0001, C=3000, probability=True).fit(X,y)
```

```
pred1 = model1.predict(X)
pred2 = model2.predict(X)
```

```
pd.crosstab(y, pred1)
```

col_0	0	1
row_0		
0	940	3
1	30	27

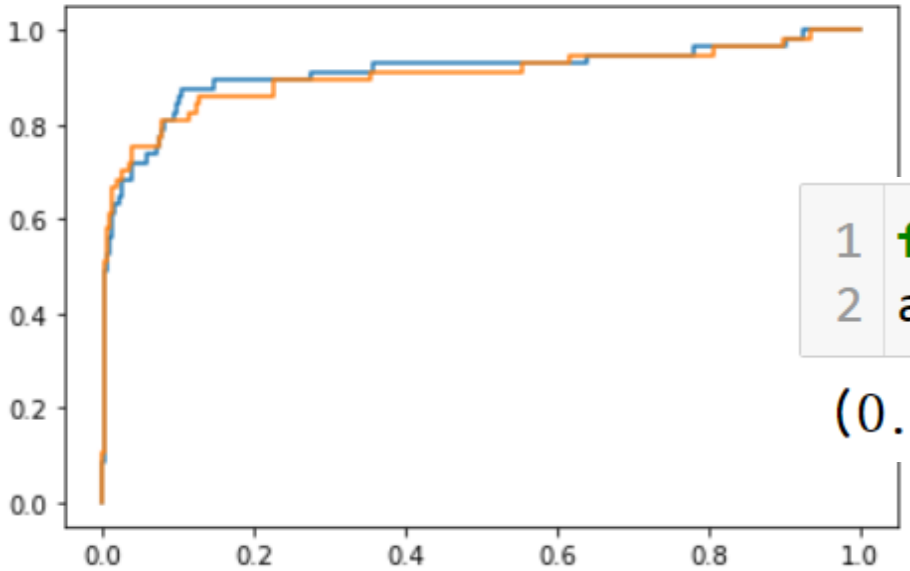
```
pd.crosstab(y, pred2)
```

col_0	0	1
row_0		
0	940	3
1	30	27

ROC 커브를 이용한 성능 비교


```
from sklearn.metrics import roc_curve
fpr1, tpr1, thr1 = roc_curve(y, model1.decision_function(X))
fpr2, tpr2, thr2 = roc_curve(y, model2.decision_function(X))
```

```
plt.plot(fpr1, tpr1)
plt.plot(fpr2, tpr2)
plt.show()
```




```
from sklearn.metrics import auc
auc(fpr1, tpr1), auc(fpr2, tpr2)
```

```
(0.9112202563673234, 0.9037227214377407)
```

다중 클래스의 ROC 커브 – 레이블 이진화


```
1 from sklearn.datasets import load_iris
2 iris = load_iris()
```

```
1 from sklearn.preprocessing import label_binarize
```

```
1 iris_X = iris.data
```

```
1 iris_y = label_binarize(iris.target, [0,1,2])
```

```
1 iris_y
```

```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
```

다중 클래스의 ROC 커브


```
from sklearn.naive_bayes import GaussianNB
```

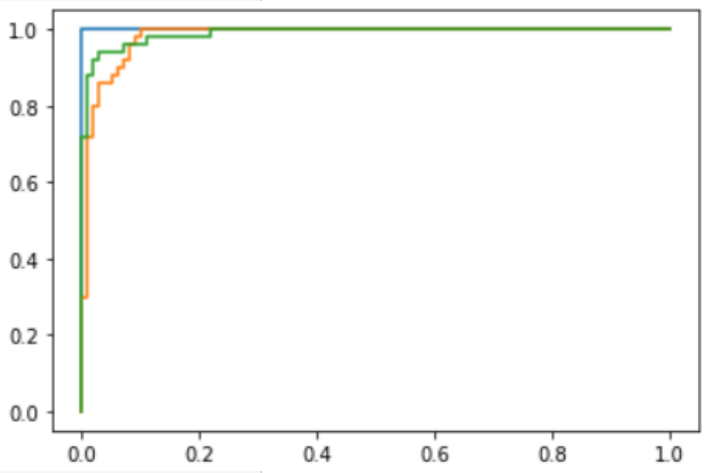
```
gnb_model0 = GaussianNB().fit(iris_X, iris_y[:,0])
gnb_model1 = GaussianNB().fit(iris_X, iris_y[:,1])
gnb_model2 = GaussianNB().fit(iris_X, iris_y[:,2])
```

```
fpr0, tpr0, thr0 = roc_curve(iris_y[:,0],
        gnb_model.predict_proba(iris_X)[:,1])
```

```
fpr1, tpr1, thr1 = roc_curve(iris_y[:,1],
        gnb_model1.predict_proba(iris_X)[:,1])
```

```
fpr2, tpr2, thr2 = roc_curve(iris_y[:,2],
        gnb_model2.predict_proba(iris_X)[:,1])
```

```
plt.plot(fpr0, tpr0)
plt.plot(fpr1, tpr1)
plt.plot(fpr2, tpr2)
plt.show()
```



연습문제

1. **winequality-red** 데이터의 와인 등급을 분류하는 모델을 구현하고 가장 좋은 모델을 판단하세요 (**https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv**)

- 확률적 모형(**QuadraticDiscriminantAnalysis, MultinormialNB, DecisionTreeClassifier, LogisticRegression**) 클래스와 판별함수 모형(**SVC, MLPClassifier, Perceptron**) 클래스를 한개씩 선택하여 구현하세요.
- 모델 학습 후 교차분류표와 **accuracy, precision, recall,** 특이도, 위양성률, **f1 score**를 출력하세요. **ROC** 커브의 **auc(Area Under Curve)** 값을 포함해서 가장 성능이 좋은 분류모형을 판단하세요

다음 표는 **Wine Quality** 데이터 셋 변수에 대한 설명입니다

No	변수명	변수설명
1	fixed acidity	결합산도
2	volatile acidity	휘발성 산도
3	citric acid	구연산
4	residual sugar	발효 후 와인 속에 남아있는 당분
5	chlorides	염화물
6	free sulfur dioxide	유리 이산화황
7	total sulfur dioxide	총 이산화황
8	density	밀도
9	pH	산도
10	sulphates	황산염
11	alcohol	알코올
12	quality	와인 등급(0~10사이의 점수)