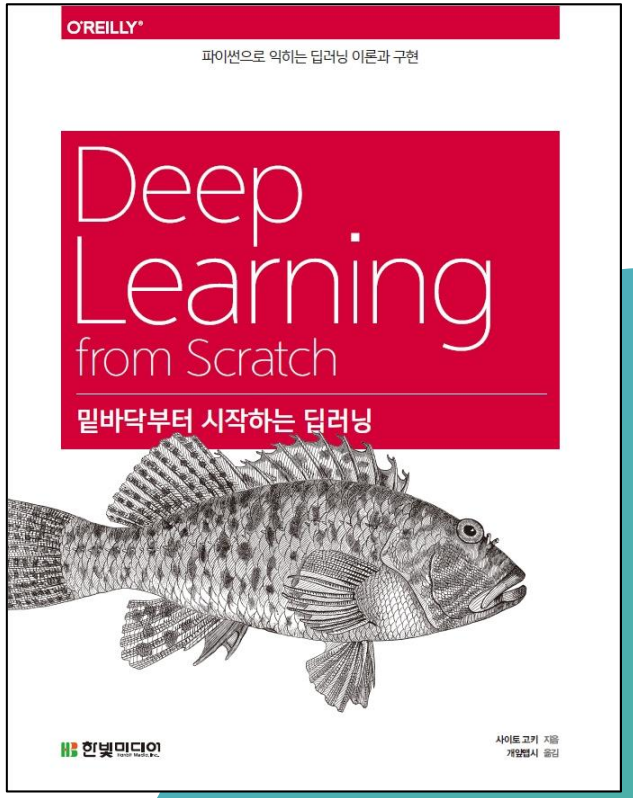
CHAPTER 2 퍼셉트론

밑바닥부터 시작하는 딥러닝



시작하기전에

- **이 책에서 사용한 프로그래밍 언어와 라이브러리**
  - 파이썬 3
  - 넘파이
  - matplotlib

- **참고할 사이트**
  - 아나콘다 배포판 https://www.anaconda.com/distribution
  - 깃허브 저장소 https://github.com/WegraLee/deep-learning-from-scratch

이 책의 학습 목표

- CHAPTER 1 파이썬에 대해 간략하게 살펴보고 사용법 익히기
- CHAPTER 2 퍼셉트론에 대해 알아보고 퍼셉트론을 써서 간단한 문제를 풀어보기
- CHAPTER 3 신경망의 개요, 입력 데이터가 무엇인지 신경망이 식별하는 처리 과정 알아보기
- CHAPTER 4 손실 함수의 값을 가급적 작게 만드는 경사법에 대해 알아보기
- CHAPTER 5 가중치 매개변수의 기울기를 효율적으로 계산하는 오차역전파법 배우기
- CHAPTER 6 신경망(딥러닝) 학습의 효율과 정확도를 높이기
- CHAPTER 7 CNN의 메커니즘을 자세히 설명하고 파이썬으로 구현하기
- CHAPTER 8 딥러닝의 특징과 과제, 가능성, 오늘날의 첨단 딥러닝에 대해 알아보기

Contents

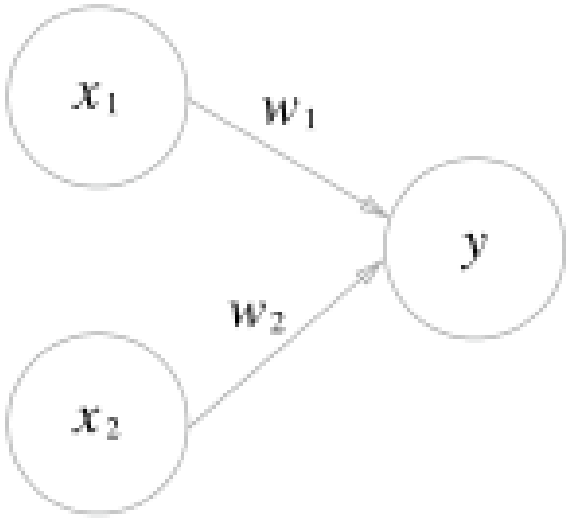
CHAPTER 2 퍼셉트론

퍼셉트론에 대해 알아보고 퍼셉트론을 써서 간단한 문제를 풀어보기

2.1 퍼셉트론이란?

퍼셉트론 * 은 다수의 신호를 입력으로 받아 하나의 신호를 출력.
퍼셉트론 신호도 흐름을 만들고 정보를 앞으로 전달.
다만, 실제 전류와 달리 퍼셉트론 신호는 '흐른다/안 흐른다(1 이나 0)'의 두 가지 값을 가진다.
이 책에서는 1 을 '신호가 흐른다', 0 을 '신호가 흐르지 않는다'라는 의미쓰인다.

그림 2-1 입력이 2개인 퍼셉트론



$$y = \begin{cases} 0 \ (w_1x_1 + w_2x_2 \leq \theta) \\ 1 \ (w_1x_1 + w_2x_2 > \theta) \end{cases} \quad [식 2.1]$$

2.2 단순한 논리 회로

[그림 2 - 2]와 같은 입력 신호와 출력 신호의 대응 표를 진리표 라고 한다.
이 그림은 AND 게이트의 진리표로, 두입력이 모두 1 일 때만 1 을 출력하고, 그 외에는 0 을 출력

그림 2-2 AND 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

2.2.2 NAND 게이트와 OR 게이트

그림 2-3 NAND 게이트의 진리표

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

그림 2-4 OR 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

NAND 게이트를 표현하려면 예를 들어 (w 1 , w 2 , θ) = (- 0 . 5 , - 0 . 5 , - 0 . 7) 조합이 있다.
사실 AND 게이트를 구현하는 매개변수의 부호를 모두 반전하기만 하면 NAND 게이트가 된다

2.3 퍼셉트론 구현하기

x1 과 x2 를 인수로 받는 AND 라는 함수.

```
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1
```

매개변수 w1 , w2 , theta 는 함수 안에서 초기화하고, 가중치를 곱한 입력의 총합이 임계값을 넘으면 1 을 반환하고 그 외에는 0 을 반환.
이 함수의 출력이 [그림 2 - 2]와 같은지 확인

```
AND(0, 0) # 0을 출력
AND(1, 0) # 0을 출력
AND(0, 1) # 0을 출력
AND(1, 1) # 1을 출력
```

2.3.2 가중치와 편향 도입

$$y = \begin{cases} 0 \ (b + w_1 x_1 + w_2 x_2 \leq 0) \\ 1 \ (b + w_1 x_1 + w_2 x_2 > 0) \end{cases} \quad [식 2.2]$$

[식 2 . 1]과 [식 2 . 2]는 기호 표기만 바꿨을 뿐, 그 의미는 같다.
여기에서 b 를 편향 bias 이라하며 w 1 과 w 2 는 그대로 가중치 weight .
[식 2 . 2] 관점에서 해석해보자면, 퍼셉트론은 입력신호에 가중치를 곱한 값과 편향을 합하여,
그 값이 0 을 넘으면 1 을 출력하고 그렇지 않으면 0을 출력.
그럼 넘파이를 사용해서 [식 2 . 2] 방식으로 구현해보자.

```
>>> import numpy as np
>>> x = np.array([0, 1])      # 입력
>>> w = np.array([0.5, 0.5]) # 가중치
>>> b = -0.7                  # 편향
>>> w*x
array([ 0. ,  0.5])
>>> np.sum(w*x)
0.5
>>> np.sum(w*x) + b
-0.19999999999999996  # 대략 -0.2 ( 부동소수점 수에 의한 연산 오차 )
```

2.3.3 가중치와 편향 구현하기

```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

여기에서 - θ 가 편향 b 로 치환(2 . 3 . 1 절에서 구현한 AND 의 theta 가 - b 가 되었다).
그리고 편향은 가중치 w 1 , w 2 와 기능이 다르다는 사실에 주의.
w 1 과 w 2 는 각 입력 신호가 결과에 주는 영향력(중요도)을 조절하는 매개변수고, 편향은 뉴런이 얼마나 쉽게 활성화(결과로 1 을 출력)하느냐를 조정하는 매개변수.

2.4 퍼셉트론의 한계

그림 2-5 XOR 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

$$y = \begin{cases} 0 \ (-0.5 + x_1 + x_2 \leq 0) \\ 1 \ (-0.5 + x_1 + x_2 > 0) \end{cases}$$

[식 2.3]

우선 OR 게이트의 동작을 시각적으로 생각해보자.
OR 게이트는, 예를 들어 가중치 매개변수가 (b , w 1 , w 2) = (- 0 . 5 , 1 . 0 , 1 . 0)일 때 [그림 2 - 4]의 진리표를 만족합니다. 이때의 퍼셉트론은 [식 2 . 3]으로 표현.

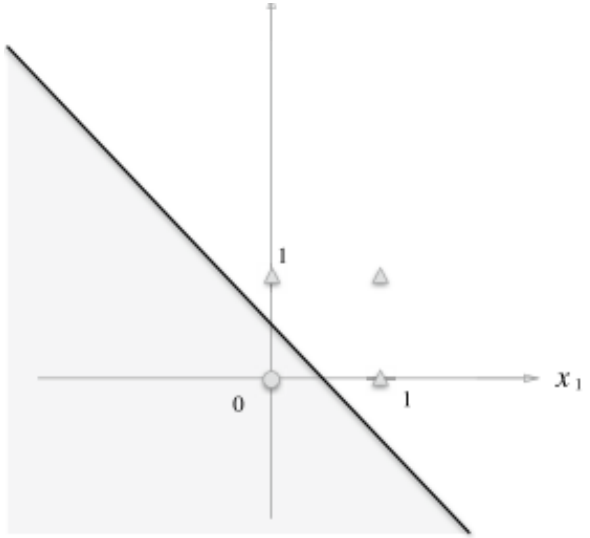


그림 2-6 퍼셉트론의 시각화 : 회색 영역은 0 을 출력하는 영역이며, 전체 영역은 OR 게이트의 성질을 만족한다.

2.4.2 선형과 비선형

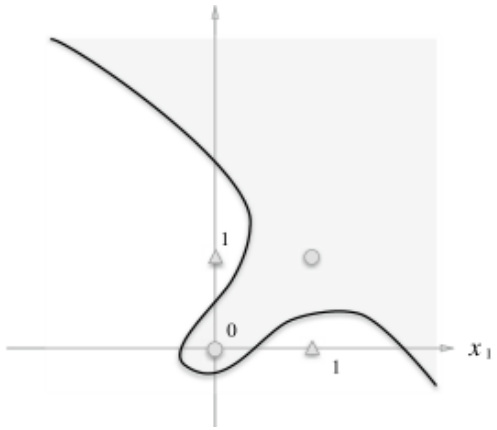


그림 2-8 곡선이라면 ○과 △을 나눌 수 있다.

퍼셉트론은 직선 하나로 나눈 영역만 표현할 수 있다는 한계가있다.
[그림 2 – 8] 같은 곡선은 표현할 수 없다는 것이다.
덧붙여서 [그림 2 – 8]과 같은 곡선의 영역을 비선형 영역, 직선의 영역을 선형 영역이라고 한다.
선형, 비선형이라는 말은 기계학습 분야에서 자주 쓰이는용어로, [그림 2 - 6]과 [그림 2 - 8] 같은 이미지를 떠올리시면 된다

2.5.1 기존 게이트 조합하기

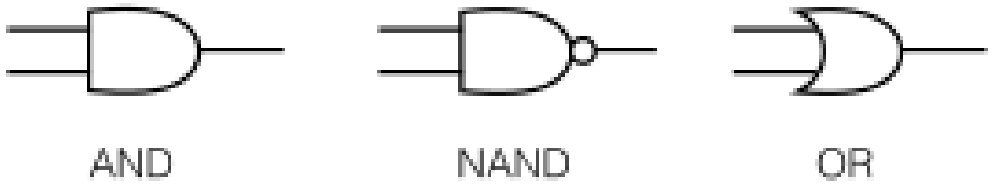


그림 2-9 AND , NAND , OR 게이트 기호

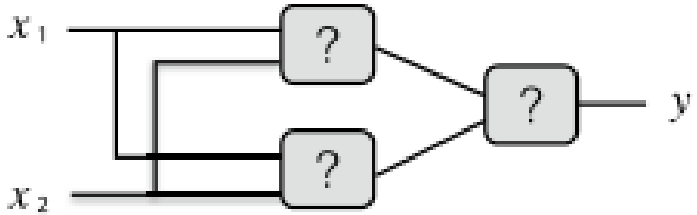


그림 2-10 AND , NAND , OR 게이트 하나씩을 '?'에 대입해 XOR 를 완성하자

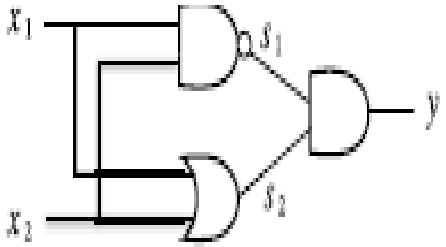


그림 **2-11**

x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

그림 **2-12**

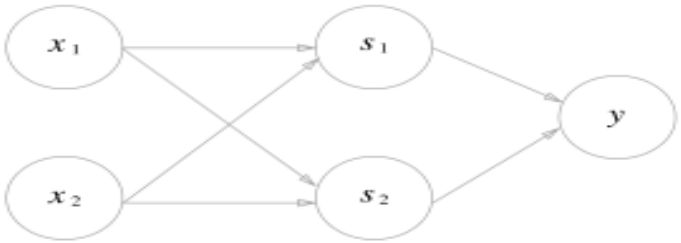
NAND 의 출력을 s 1 , OR의 출력을 s 2 로 해서 진리표를 만들면 [그림 2 - 12]처럼 된다.
X 1 , x 2 , y 에 주목하면 분명히턖 의 출력과 같다.

2.5.2 XOR 게이트 구현하기

```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

이 XOR 함수는 기대한 대로의 결과를 출력.

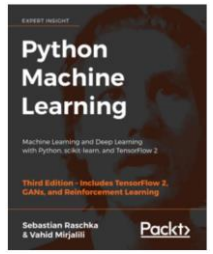
```
XOR(0, 0) # 0을 출력
XOR(1, 0) # 1을 출력
XOR(0, 1) # 1을 출력
XOR(1, 1) # 0을 출력
```



이상으로 2 층 구조를 사용해 퍼셉트론으로 XOR 게이트를 구현할 수 있게 되었다. 이처럼 퍼셉트론은 층을 쌓아(깊게 하여) 더 다양한 것을 표현할 수 있다.

Implementing a perceptron learning algorithm in Python

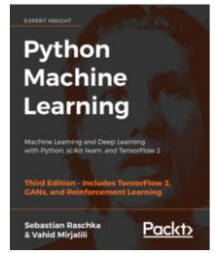
```python
import numpy as np


class Perceptron(object):
    """Perceptron classifier.

    Parameters
    ------------
    eta : float
      Learning rate (between 0.0 and 1.0)
    n_iter : int
      Passes over the training dataset.
    random_state : int
      Random number generator seed for random weight
      initialization.

    Attributes
    -----------
    w_ : 1d-array
      Weights after fitting.
    errors_ : list
      Number of misclassifications (updates) in each epoch.

    """
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state
```



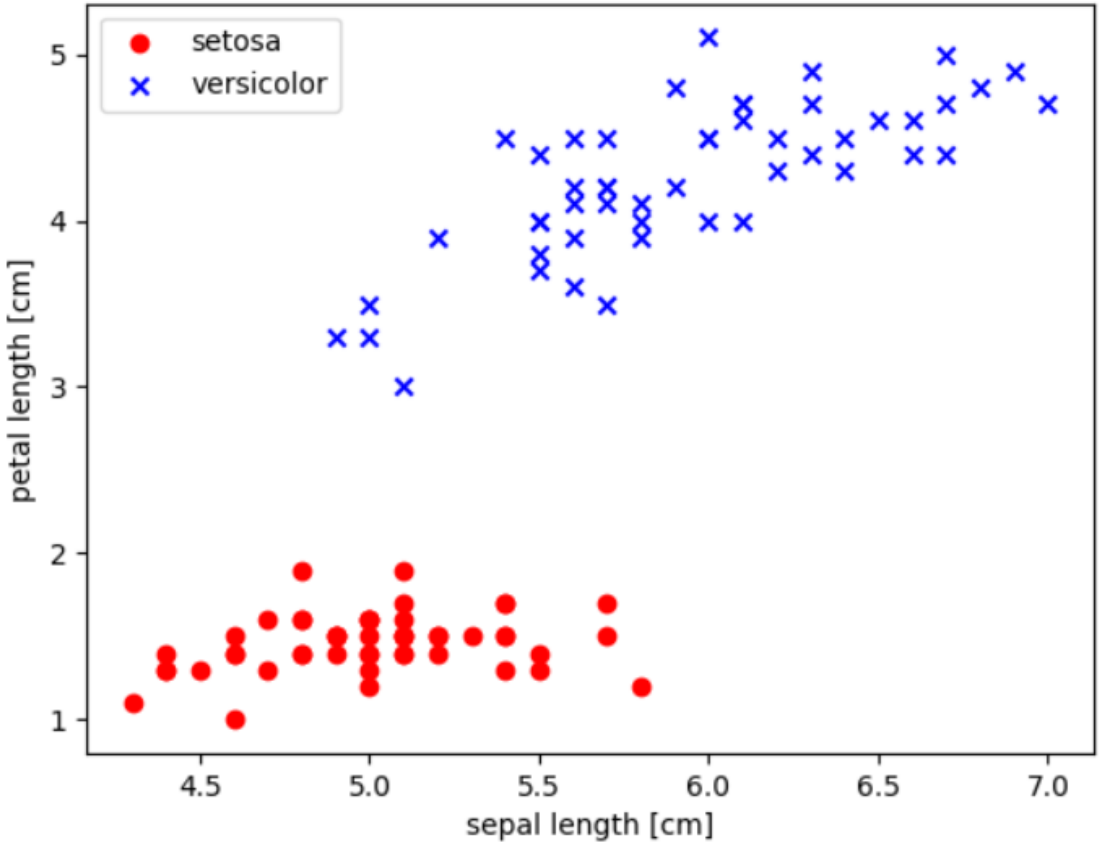
Chap 02

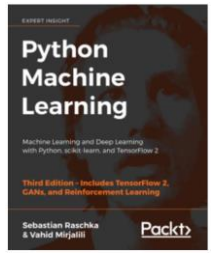
```python
    def fit(self, X, y):
        """Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
          Training vectors, where n_samples is the number of samples and
          n_features is the number of features.
        y : array-like, shape = [n_samples]
          Target values.

        Returns
        -------
        self : object

        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

Implementing a perceptron learning algorithm in Python



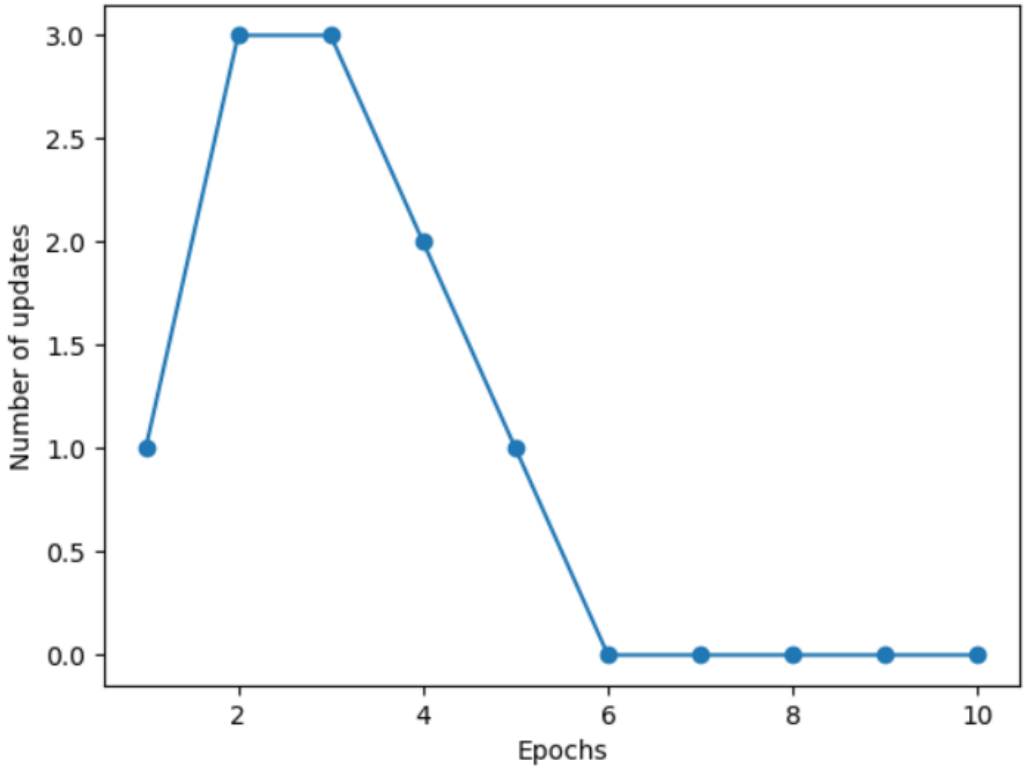

```
import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/'
        'machine-learning-databases/iris/iris.data', header=None)
df.tail()

# ### Plotting the Iris data

# In[11]:


import matplotlib.pyplot as plt
import numpy as np

# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

# extract sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')

plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')

# plt.savefig('images/02_06.png', dpi=300)
plt.show()
```





Implementing a perceptron learning algorithm in Python

```
# ### Training the perceptron model

# In[12]:


ppn = Perceptron(eta=0.1, n_iter=10)

ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')

# plt.savefig('images/02_07.png', dpi=300)
plt.show()
```



4.6 퍼셉트론 프로그래밍

- Scikit-learn 라이브러리 활용

프로그램 4-1 OR 데이터에 퍼셉트론 적용

```
01  from sklearn.linear_model import Perceptron
02
03  # 훈련 집합 구축
04  X=[[0,0],[0,1],[1,0],[1,1]]
05  y=[-1,1,1,1]
06
07  # fit 함수로 Perceptron 학습
08  p=Perceptron()
09  p.fit(X,y)
10
11  print("학습된 퍼셉트론의 매개변수: ",p.coef_,p.intercept_)
12  print("훈련집합에 대한 예측: ",p.predict(X))
13  print("정확률 측정: ",p.score(X,y)*100,"%")
```

```
학습된 퍼셉트론의 매개변수:  [[2. 2.]] [-1.]
훈련집합에 대한 예측:  [-1  1  1  1]
정확률 측정:  100.0%
```

- Scikit-learn 라이브러리 활용

프로그램 4-2 필기 숫자 데이터에 퍼셉트론 적용

```
01  from sklearn import datasets
02  from sklearn.linear_model import Perceptron
03  from sklearn.model_selection import train_test_split
04  import numpy as np
05
06  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
07  digit=datasets.load_digits()
08  x_train,x_test,y_train,y_test=train_test_split
    (digit.data,digit.target,train_size=0.6)
09
10  # fit 함수로 Perceptron 학습
11  p=Perceptron(max_iter=100,eta0=0.001,verbose=0)
12  p.fit(x_train,y_train) # digit 데이터로 모델링
13
14  res=p.predict(x_test)  # 테스트 집합으로 예측
15
```

데이터 읽기

모델 객체 생성

모델 학습

학습된 모델로 예측

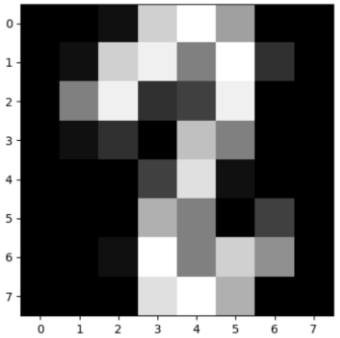
- Scikit-learn 라이브러리 활용

```
16  # 혼동 행렬
17  conf=np.zeros((10,10))
18  for i in range(len(res)):
19      conf[res[i]][y_test[i]]+=1
20  print(conf)
21
22  # 정확률 계산
23  no_correct=0
24  for i in range(10):
25      no_correct+=conf[i][i]
26  accuracy=no_correct/len(res)
27  print("테스트 집합에 대한 정확률은 ", accuracy*100, "%입니다.")
```

성능 측정

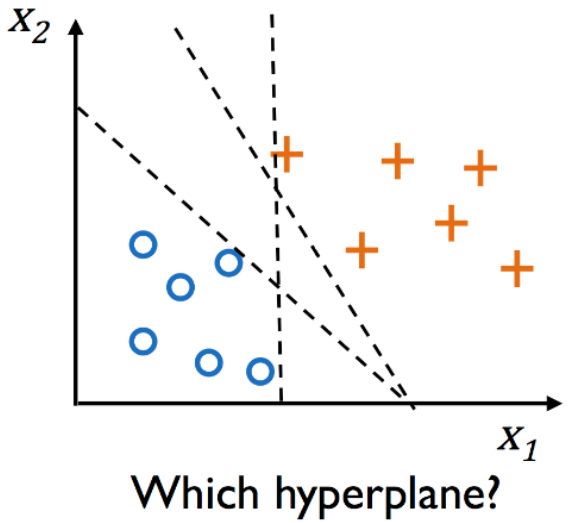
```
[[66.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0. 65.  0.  0.  0.  0.  1.  0.  6.  2.]
 [ 0.  2. 58.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0. 62.  0.  0.  0.  0.  1.  0.]
 [ 0.  1.  0.  0. 60.  0.  0.  2.  1.  0.]
 [ 0.  0.  0.  0.  0. 71.  0.  0.  1.  1.]
 [ 0.  0.  0.  0.  0.  0. 80.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 75.  0.  0.]
 [ 0.  3.  0.  2.  0.  0.  1.  1. 66.  0.]
 [ 0.  1.  0.  1.  0.  9.  0.  3.  2. 72.]]
테스트 집합에 대한 정확률은 93.88038942976355%입니다.
```

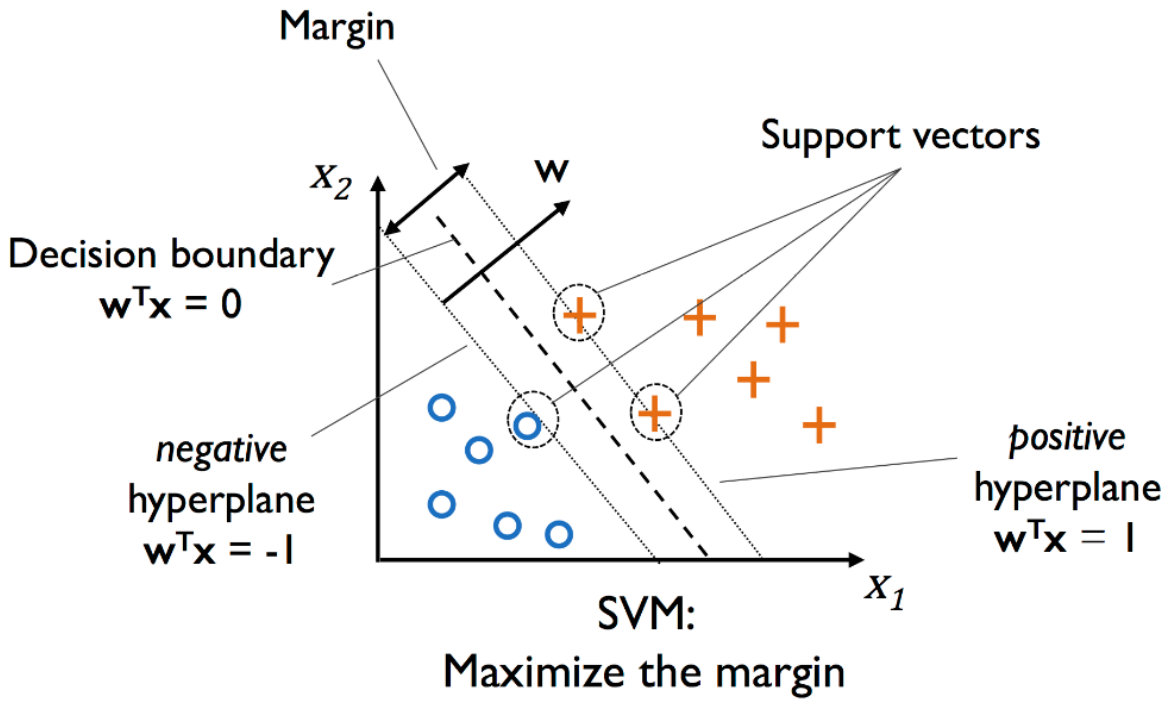
```
>>> y=x_train[0,:].reshape(8,8)
>>> import matplotlib.pyplot as plt
>>> import pickle
>>> plt.imshow(y,'gray')
<matplotlib.image.AxesImage object at 0x000001EB4A702BB0>
>>> plt.show()
```

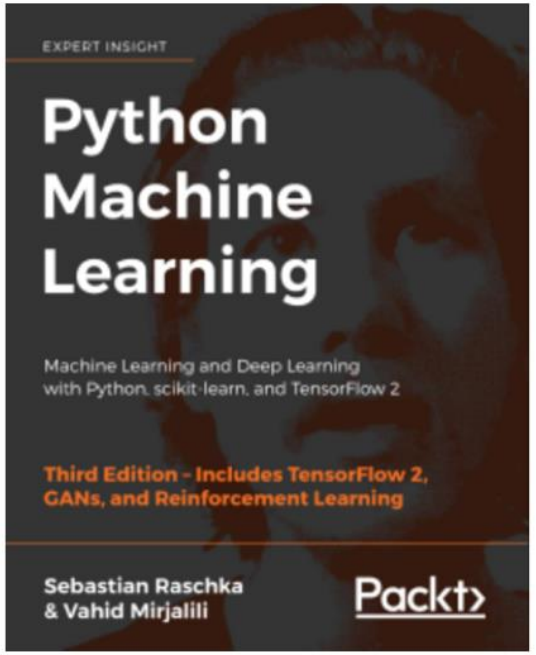


Maximum margin classification with support vector machines (chapter 3)

- Margin: the distance between the separating hyperplane (decision boundary)



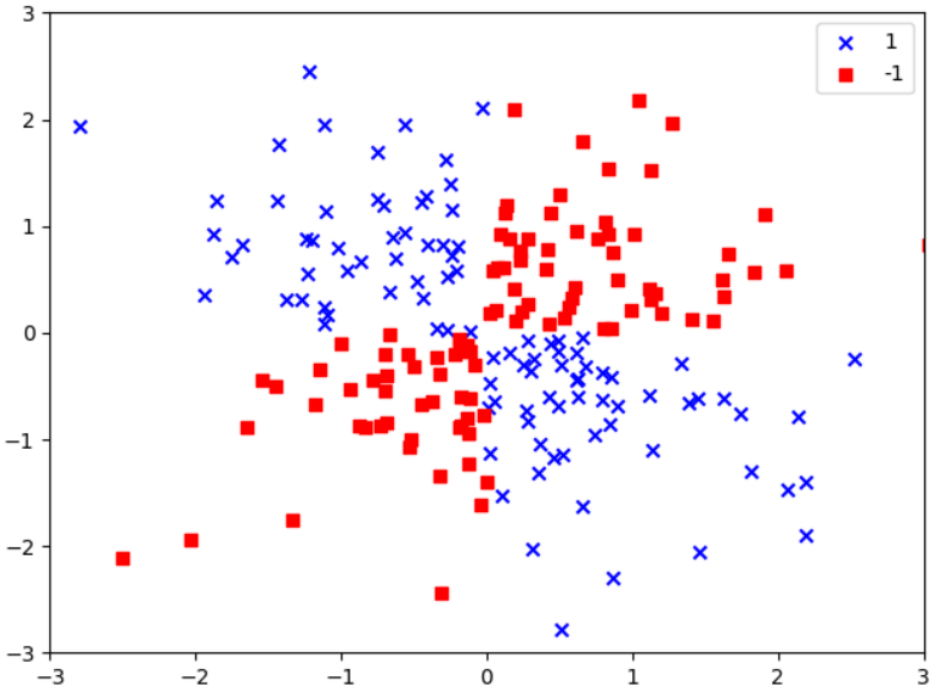




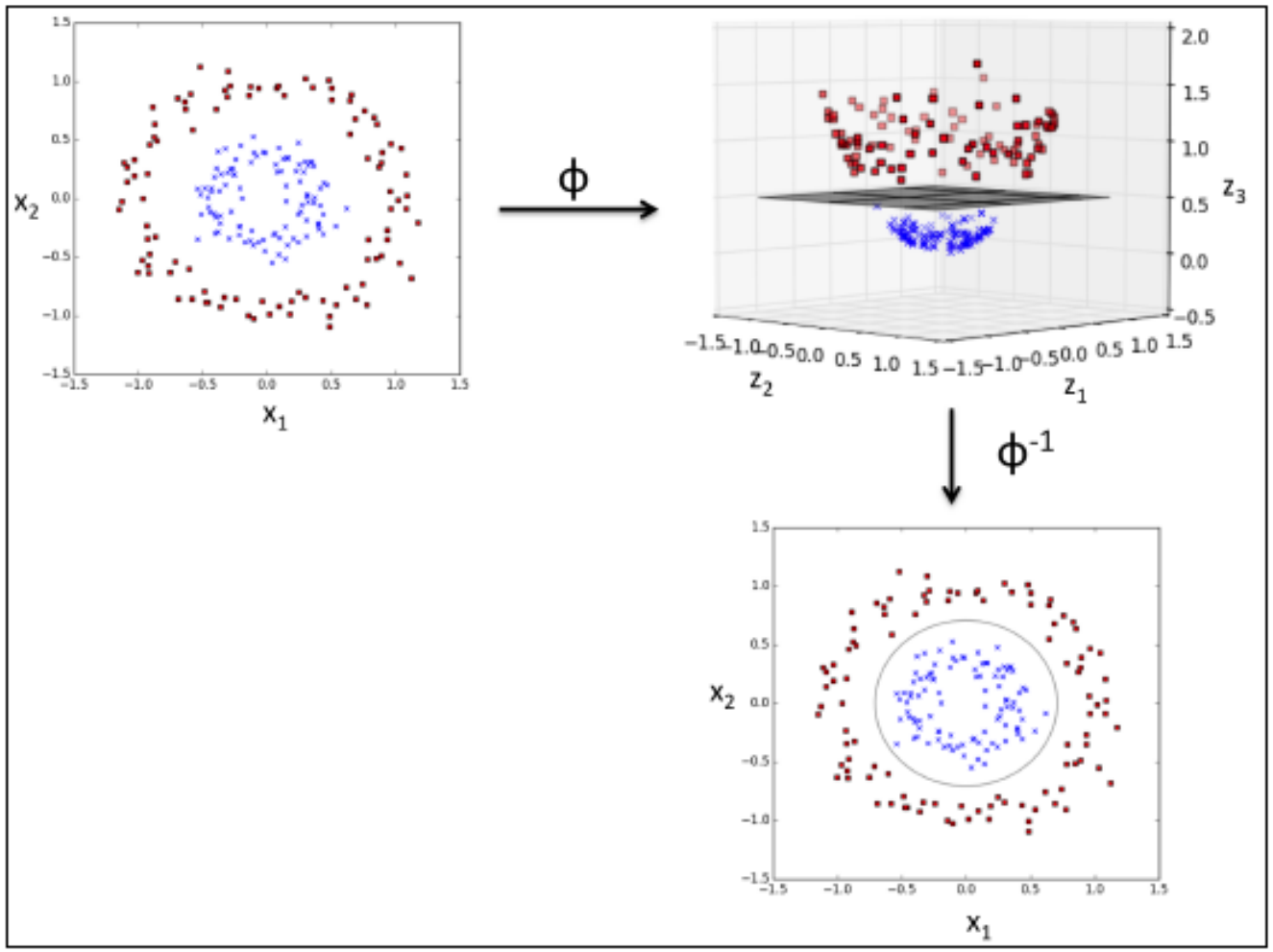
Solving nonlinear problems using a kernel SVM

- Easily kernelized to solve nonlinear classification problems
- Create a simple dataset that has the form of an XOR gate
  - ✓ Separate samples using a linear hyperplane(?)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap

np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,
                       X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)

plt.scatter(X_xor[y_xor == 1, 0],
            X_xor[y_xor == 1, 1],
            c='b', marker='x',
            label='1')
plt.scatter(X_xor[y_xor == -1, 0],
            X_xor[y_xor == -1, 1],
            c='r',
            marker='s',
            label='-1')

plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
#plt.savefig('images/03_12.png', dpi=300)
plt.show()
```



```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    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 = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, 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=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test samples
    if test_idx:
        # plot all samples
        X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    edgecolor='black',
                    alpha=1.0,
                    linewidth=1,
                    marker='o',
                    s=100,
                    label='test set')
```



$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

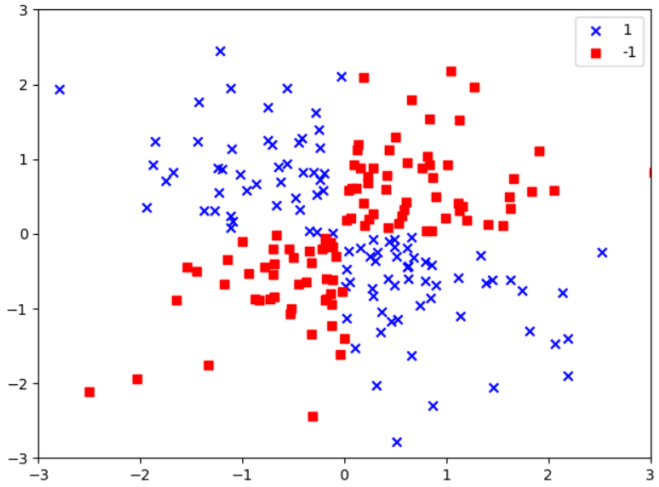
Solving nonlinear problems using a kernel SVM

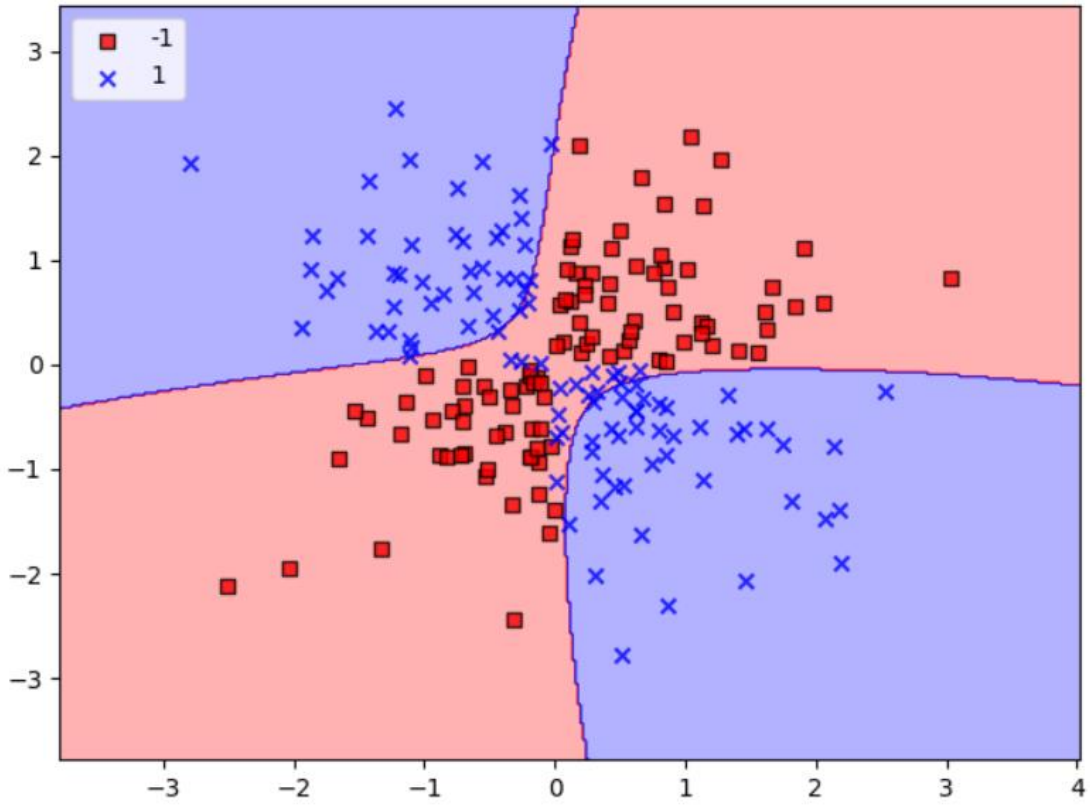
One of the most widely used kernels is the **Radial Basis Function (RBF)** or simply called the **Gaussian Kernel** The term kernel can be interpreted as a similarity function between a pair of samples

kernel='linear' vs kernel='rbf '

$$K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\frac{\| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \|^2}{2\sigma^2}\right) = \exp\left(-\gamma \| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \|^2\right)$$

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor,
                      classifier=svm)

plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_14.png', dpi=300)
plt.show()
```





Gamma can be understood as a cut-off parameter for the Gaussian sphere. If we increase the gamma value, we increase the influence or reach of the training samples, which leads to a tighter and bumpier decision boundary.

Apply an RBF kernel SVM to Iris flower dataset

```
from sklearn import datasets

iris=datasets.load_iris()
print(iris.DESCR)


for  i in range(0,len(iris.data)):
    print(i+1,iris.data[i],iris.target[i])
```

```
1 [5.1 3.5 1.4 0.2] 0
2 [4.9 3.  1.4 0.2] 0
3 [4.7 3.2 1.3 0.2] 0
4 [4.6 3.1 1.5 0.2] 0
5 [5.  3.6 1.4 0.2] 0
6 [5.4 3.9 1.7 0.4] 0
7 [4.6 3.4 1.4 0.3] 0
8 [5.  3.4 1.5 0.2] 0
9 [4.4 2.9 1.4 0.2] 0
10 [4.9 3.1 1.5 0.1] 0
                    .
                    .
                    .
139 [6.  3.  4.8 1.8] 2
140 [6.9 3.1 5.4 2.1] 2
141 [6.7 3.1 5.6 2.4] 2
142 [6.9 3.1 5.1 2.3] 2
143 [5.8 2.7 5.1 1.9] 2
144 [6.8 3.2 5.9 2.3] 2
145 [6.7 3.3 5.7 2.5] 2
146 [6.7 3.  5.2 2.3] 2
147 [6.3 2.5 5.  1.9] 2
148 [6.5 3.  5.2 2. ] 2
149 [6.2 3.4 5.4 2.3] 2
150 [5.9 3.  5.1 1.8] 2
```

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
```

Iris flower dataset

```
The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.
```

그림 3-2 iris의 세 가지 품종(왼쪽부터 Setosa, Versicolor, Virginica)

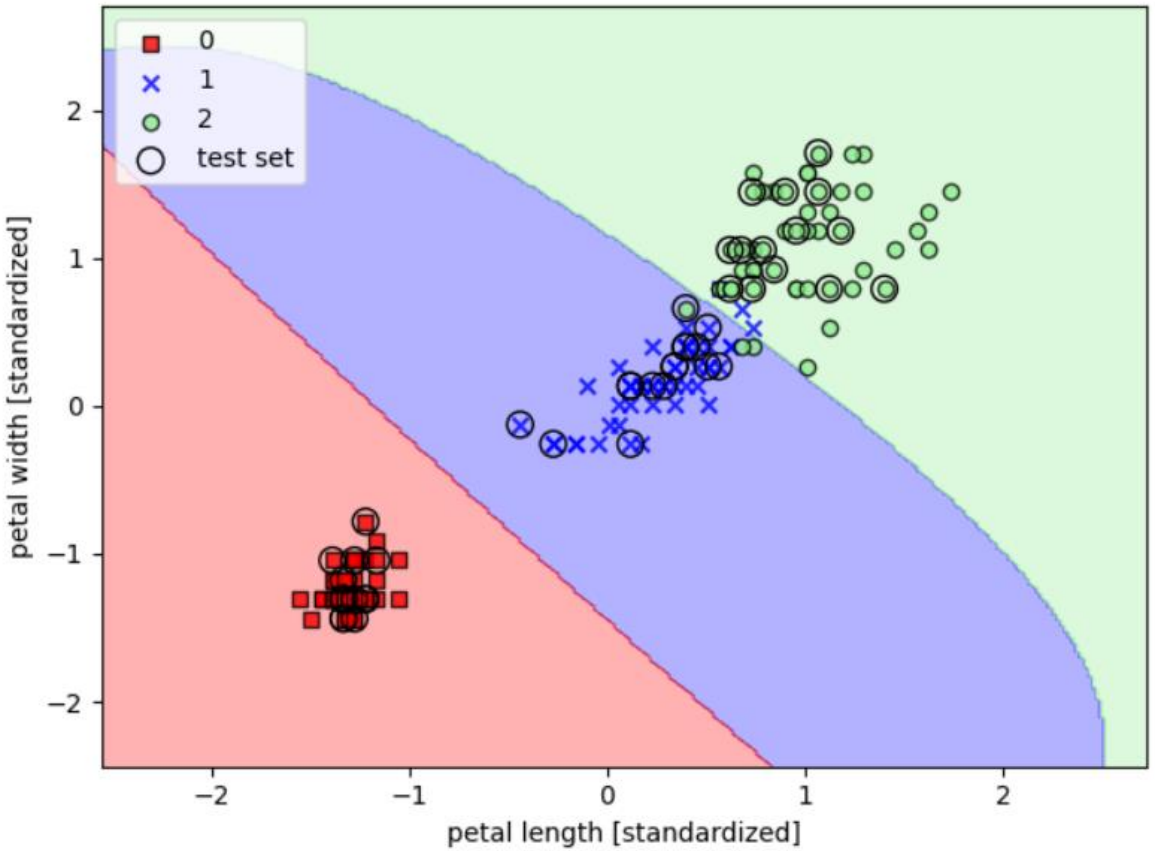
Apply an RBF kernel SVM to Iris flower dataset

```
# In[34]: IRIS flower data

from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

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=1, stratify=y)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

Apply an RBF kernel SVM to Iris flower dataset

Since we choose a relatively small value for gamma, the resulting decision boundary of the RBF kernel SVM model will be relatively soft.

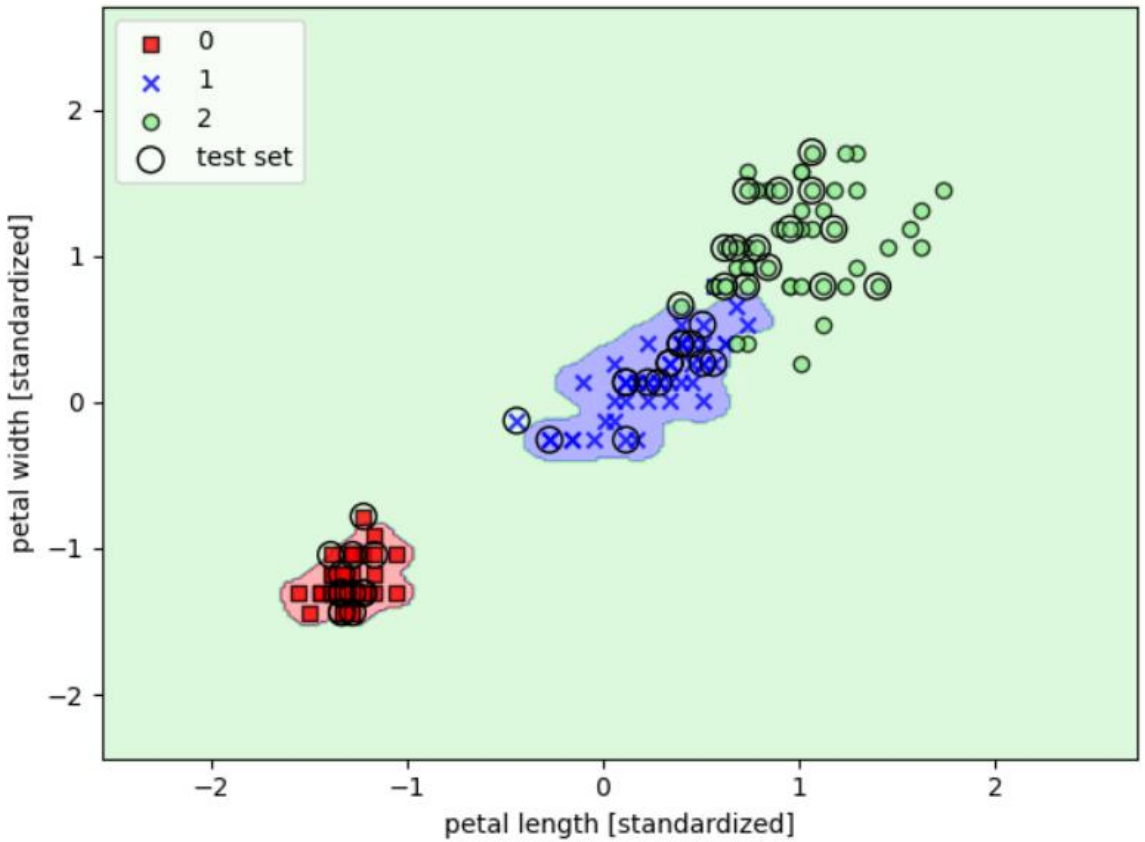
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_15.png', dpi=300)
plt.show()
```



Apply an RBF kernel SVM to Iris flower dataset

The decision boundary around the classes 0 and 1 is much tighter using a relatively large value of gamma

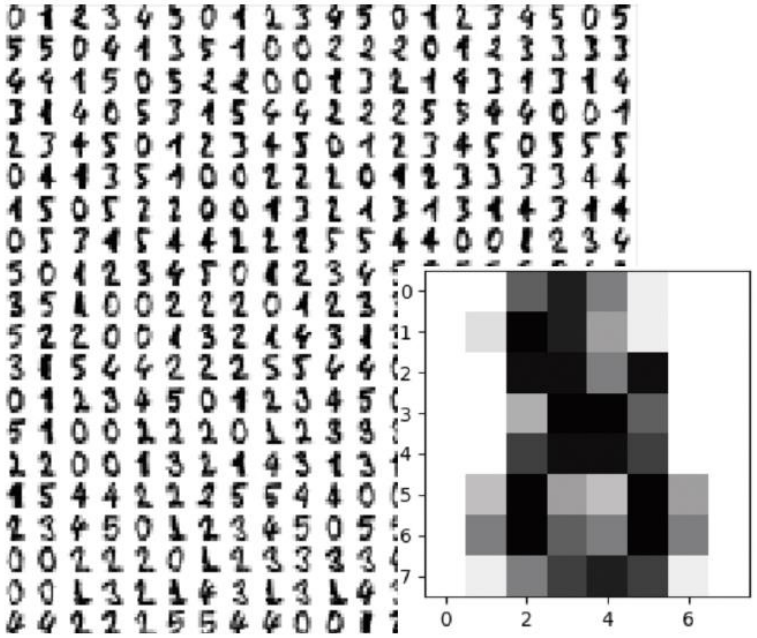
```
svm = SVC(kernel='rbf', random_state=1, gamma=100.0, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=svm, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_16.png', dpi=300)
plt.show()
```



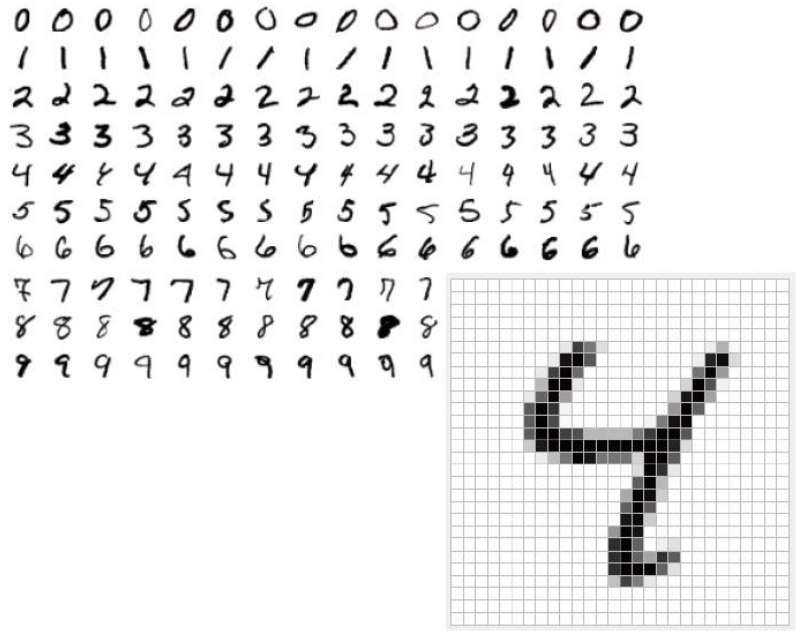
두 가지 필기 숫자 데이터셋

sklearn 데이터셋: 8*8 맵(64개 화소), 1797개 샘플, [0,16] 명암값

MNIST 데이터셋: 28*28맵(784개 화소), 7만개 샘플, [0,255] 명암값



(a) sklearn에서 제공하는 데이터셋



(b) MNIST 데이터셋

그림 3-6 필기 숫자 데이터셋

```
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import train_test_split
import numpy as np

# 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
digit=datasets.load_digits()
x_train,x_test,y_train,y_test=train_test_split(digit.data,digit.target,train_size=0.6)

# svm의 분류 모델 SVC를 학습
s=svm.SVC(gamma=0.001)
s.fit(x_train,y_train)

res=s.predict(x_test)

# 혼동 행렬 구함
conf=np.zeros((10,10))
for i in range(len(res)):
    conf[res[i]][y_test[i]]+=1
print(conf)

# 정확률 측정하고 출력
no_correct=0
for i in range(10):
    no_correct+=conf[i][i]
accuracy=no_correct/len(res)
print("테스트 집합에 대한 정확률은", accuracy*100, "%입니다.")
```

target
y_test[i]

predict
res[i]

```
[[72.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 79.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0. 84.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 66.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0. 65.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 80.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. 59.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0. 67.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0. 70.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 71.]]
테스트 집합에 대한 정확률은 99.16550764951322 %입니다.
```

프로그램3-5: 퍼셉트론(프로그램 4-2) 보다 우수한 성능