

Introduction to Machine Learning with Python

2. Supervised Learning(3)

Honedae Machine Learning Study Epoch #2

Contacts

Haesun Park

Email : haesunpark@gmail.com

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunpark>

Blog : <https://tensorflow.blog>

Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

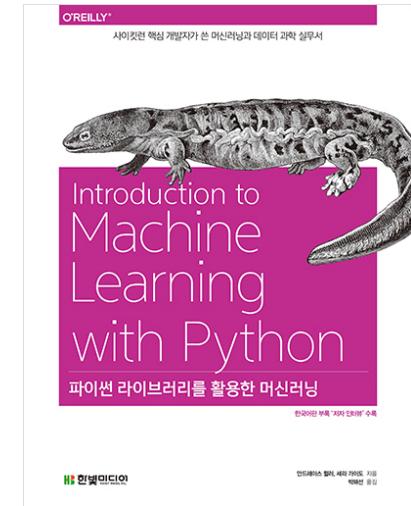
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

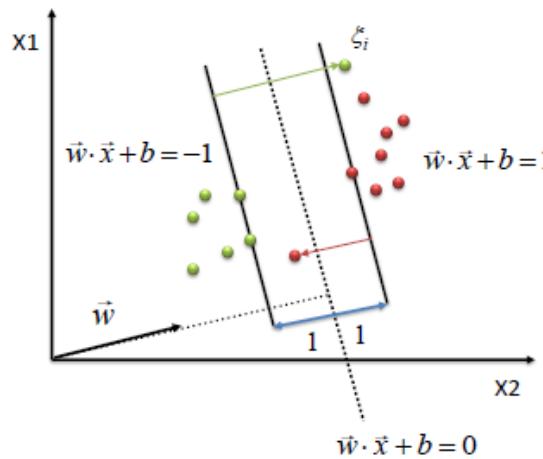
Github:

https://github.com/rickiepark/introduction_to_ml_with_python/



커널 서포트 벡터 머신

Linear SVM



결정 평면의 기울기를 줄여 마진을
최대화하기 위해 w 를 최소화

마진 위배를 허용하는 슬랙변수를 최소화

Constraint becomes :

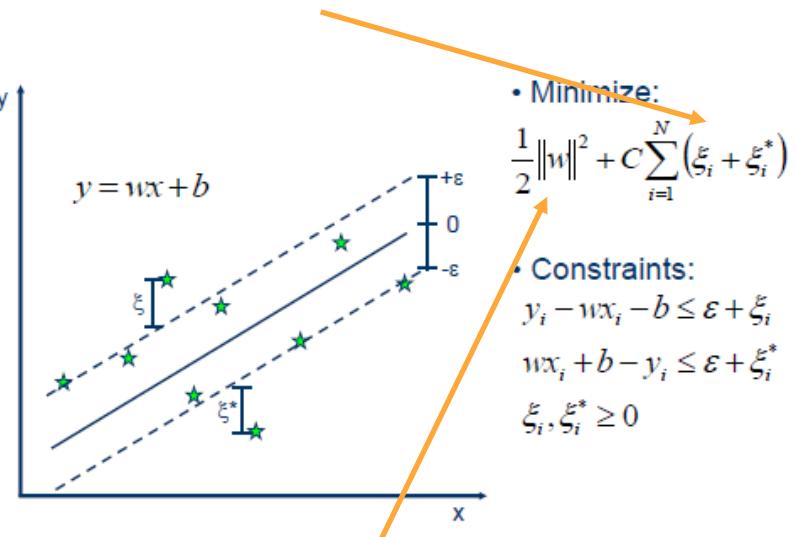
$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i, \forall x_i$$

$$\xi_i \geq 0$$

Objective function
penalizes for misclassified
instances and those within
the margin

$$\min \frac{1}{2} \|\vec{w}\|^2 + C \sum_i \xi_i$$

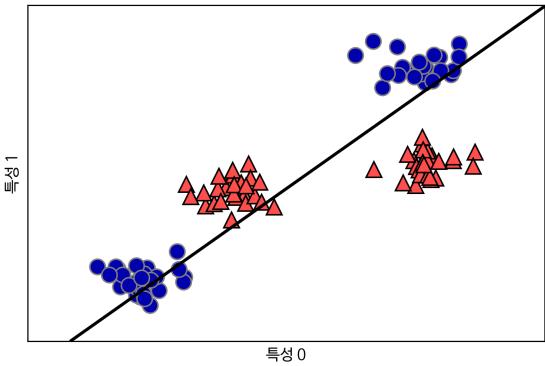
ξ trades-off margin width
and misclassifications



같은 epsilon 높이에서 마진을
최대화하기 위해 w 를 최소화

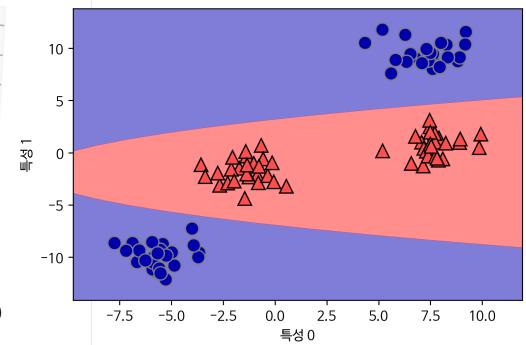
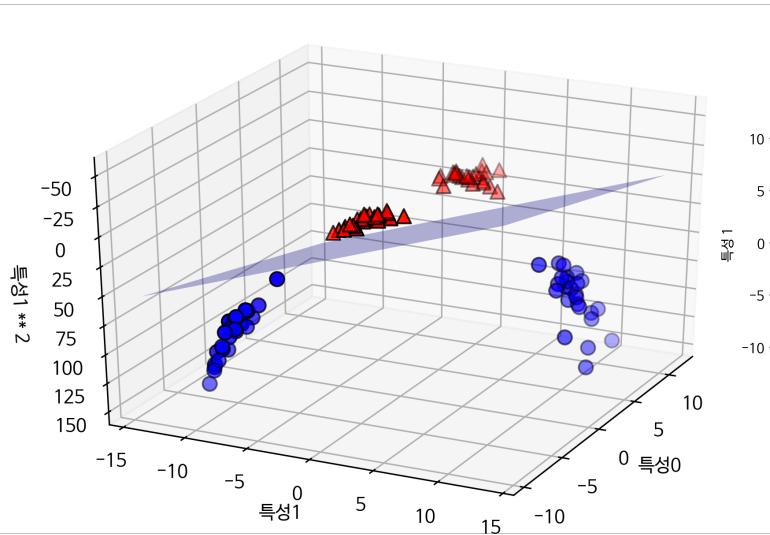
비선형 특성

```
In [78]: from sklearn.svm import LinearSVC  
linear_svm = LinearSVC().fit(X, y)
```



```
In [79]: # 두 번째 특성을 제곱하여 추가합니다  
X_new = np.hstack([X, X[:, 1:] ** 2])
```

```
In [80]: linear_svm_3d = LinearSVC().fit(X_new, y)
```



커널 기법 kernel trick

어떤 특성을 추가해야 할지 불분명하고 많은 특성을 추가하면 연산 비용 커짐

커널 함수라 부르는 특별한 함수를 사용하여 데이터 포인트 간의 거리를 계산하여 데이터 포인트의 특성이 고차원에 매핑된 것 같은 효과를 얻음

다항 커널

$$x_a = (a_1, a_2), x_b = (b_1, b_2) \text{ 일때 } (x_a \cdot x_b)^2 = (a_1 b_1 + a_2 b_2)^2 = \\ a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2 = (a_1^2, \sqrt{2}a_1 a_2, a_2^2) \cdot (b_1^2, \sqrt{2}b_1 b_2, b_2^2)$$

RBF radial basis function 커널

$$\exp(-\gamma \|x_a - x_b\|^2), e^x \text{의 테일러 급수 전개: } C \sum_{n=0}^{\infty} \frac{(x_a \cdot x_b)^n}{n!}$$

무한한 특성 공간으로 매핑하는 효과, 고차항일 수록 특성 중요성은 떨어짐

SVM 이해하기

커널 기법을 적용한 SVM을 ‘커널 SVM’ 혹은 그냥 ‘SVM’으로 부릅니다.

Scikit-Learn은 SVC, SVR 클래스를 제공합니다.

클래스 경계에 위치한 데이터 포인트를
서포트 벡터라 부릅니다.

RBF 커널을 가우시안 Gaussian 커널이라고도 부릅니다.

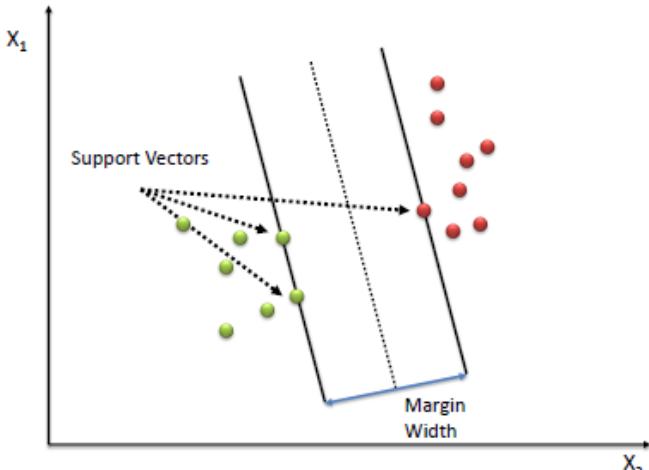
커널의 폭(gamma), $e^0 \sim e^{-\infty} = 1 \sim 0$

γ 가 작을 수록 샘플의 영향 범위 커짐

$\gamma = \frac{1}{2\sigma^2}$ 일 때 σ 를 커널의 폭이라 부름

$$\exp(-\gamma \|x_a - x_b\|^2)$$

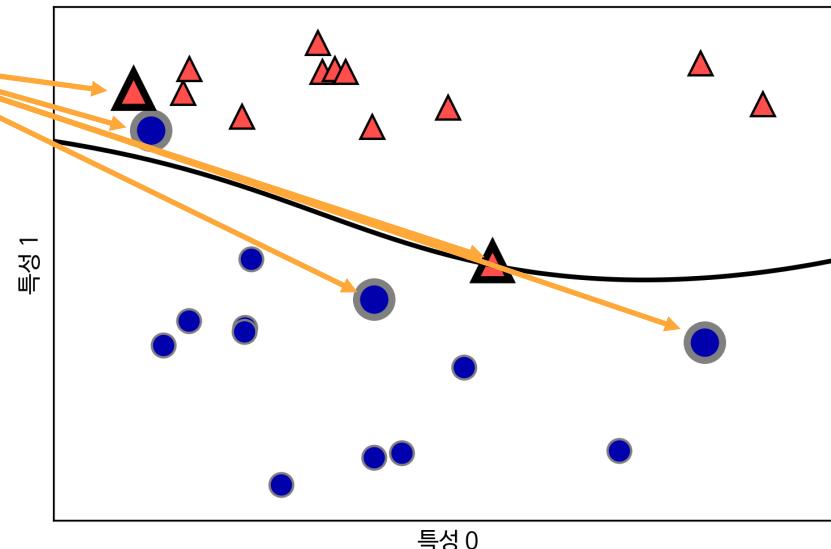
유클리디안 거리



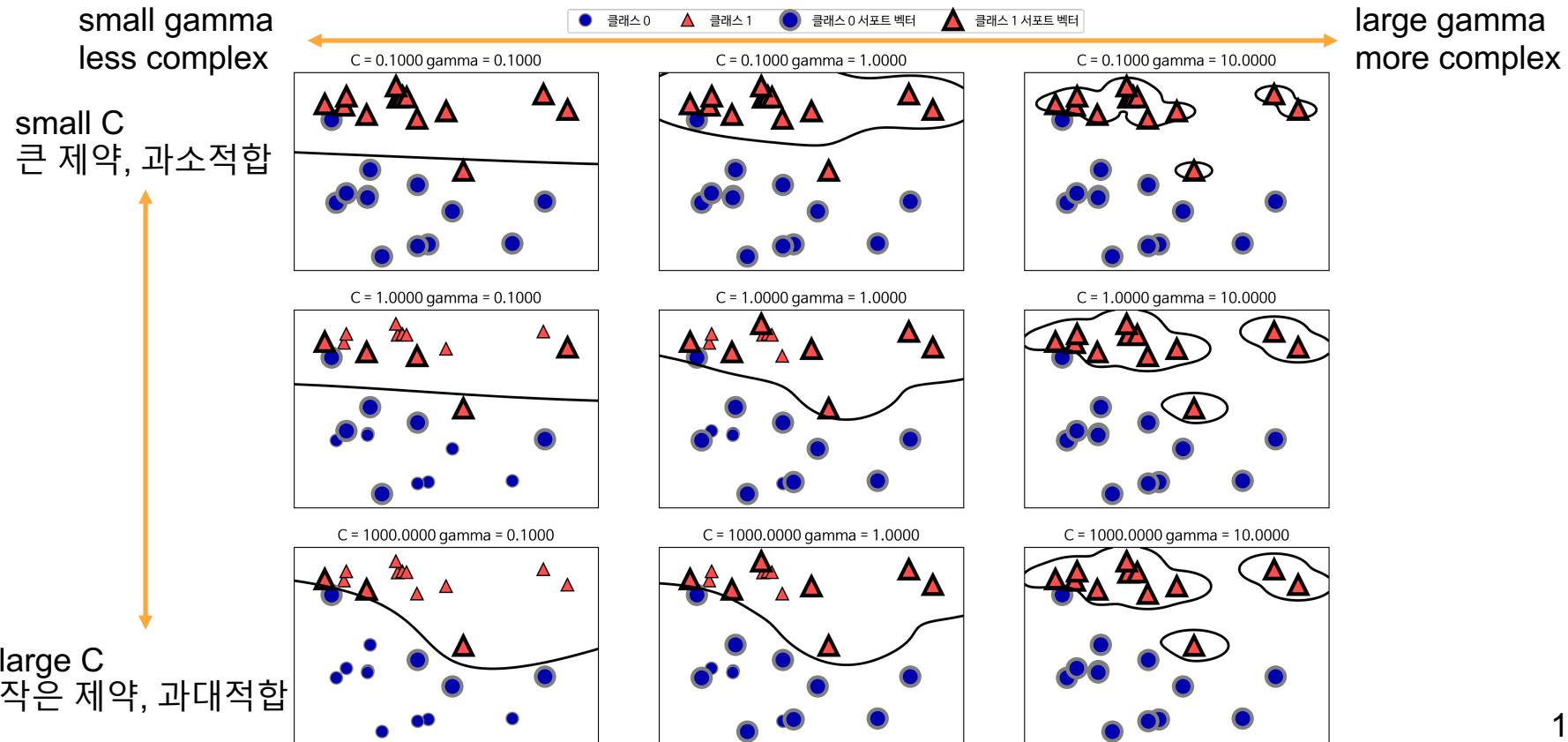
SVC + forge dataset

```
In [82]: from sklearn.svm import SVC  
  
X, y = mglearn.tools.make_handcrafted_dataset()  
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)  
# 서포트 벡터  
sv = svm.support_vectors_  
# dual_coef_의 부호에 의해 서포트 벡터의 클래스 레이블이 결정됩니다  
sv_labels = svm.dual_coef_.ravel() > 0
```

서포트 벡터



매개변수 튜닝

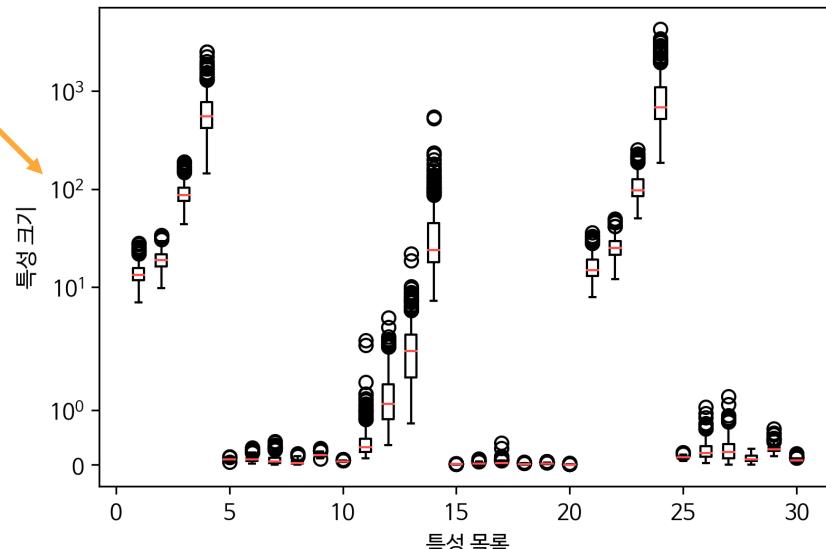


SVC + cancer dataset

```
In [84]:  
X_train, X_test, y_train, y_test = train_test_split(  
    cancer.data, cancer.target, random_state=0)  
  
svc = SVC() ← C=1, gamma=1/n_features  
svc.fit(X_train, y_train)  
  
print("훈련 세트 정확도: {:.2f}".format(svc.score(X_train, y_train)))  
print("테스트 세트 정확도: {:.2f}".format(svc.score(X_test, y_test)))
```

훈련 세트 정확도: 1.00 ← 과대적합
테스트 세트 정확도: 0.63

cancer 데이터셋의
데이터 스케일



SVM은 특성의 범위에
크게 민감함

데이터 전처리

MinMaxScaler : $\frac{X - \min(X)}{\max(X) - \min(X)}$, 0 ~ 1 사이로 조정

```
In [86]: # 훈련 세트에서 특성별 최솟값 계산
min_on_training = X_train.min(axis=0)
# 훈련 세트에서 특성별 (최댓값 - 최솟값) 범위 계산
range_on_training = (X_train - min_on_training).max(axis=0)

# 훈련 데이터에 최솟값을 빼고 범위로 나누면
# 각 특성에 대해 최솟값은 0 최댓값은 1 임
X_train_scaled = (X_train - min_on_training) / range_on_training
print("특성별 최솟값\n{}".format(X_train_scaled.min(axis=0)))
print("특성별 최댓값\n{}".format(X_train_scaled.max(axis=0)))
```

특성별 최소 값

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

특성별 최대 값

```
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
```

```
In [87]: # 테스트 세트에도 같은 작업을 적용하지만
# 훈련 세트에서 계산한 최솟값과 범위를 사용합니다(자세한 내용은 3장에 있습니다)
X_test_scaled = (X_test - min_on_training) / range_on_training
```

SVC + 전처리 데이터

```
In [88]: svc = SVC()
svc.fit(X_train_scaled, y_train)

print("훈련 세트 정확도: {:.3f}".format(svc.score(X_train_scaled, y_train)))
print("테스트 세트 정확도: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

훈련 세트 정확도: 0.948
테스트 세트 정확도: 0.951 ← 전처리된 후에는 오히려 과소적합 됩니다.

```
In [89]: svc = SVC(C=1000) ← 제약완화
svc.fit(X_train_scaled, y_train)

print("훈련 세트 정확도: {:.3f}".format(svc.score(X_train_scaled, y_train)))
print("테스트 세트 정확도: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

훈련 세트 정확도: 0.988
테스트 세트 정확도: 0.972

장단점과 매개변수

장점

강력하며 여러 종류의 데이터셋에 적용 가능합니다.

특성이 적을 때에도 복잡한 결정 경계 만들니다(커널 트릭).

특성이 많을 때에도 잘 작동합니다.

SVC/SVR(libsvm), LinearSVC/LinearSVR(liblinear)

단점

샘플이 많을 경우 훈련 속도가 느리고 메모리를 많이 사용합니다(>100,000).

데이터 전처리와 매개변수에 민감합니다.(→랜덤 포레스트, 그래디언트 부스팅)

분석하기 어렵고 비전문가에게 설명하기 어렵습니다.

매개변수

C, gamma, coef0(다항, 시그모이드), degree(다항)

linear: $x_1 \cdot x_2$, *poly*: $(\gamma(x_1 \cdot x_2) + c)^d$, *sigmoid*: $\tanh(\gamma(x_1 \cdot x_2) + c)$

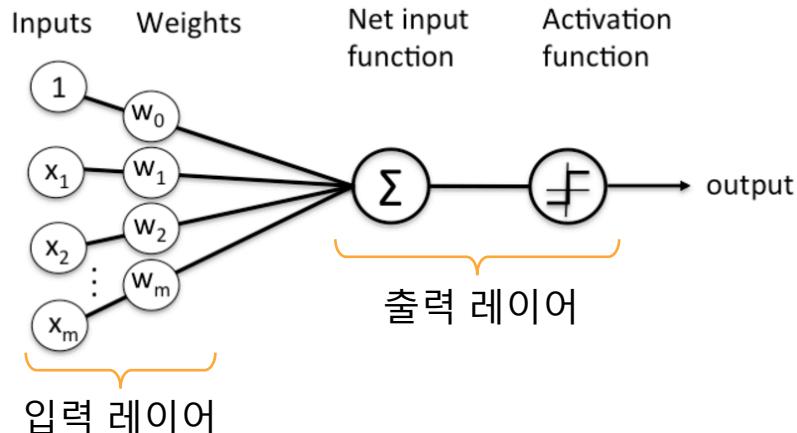
신경망 neural network

퍼셉트론 perceptron

1957년에 프랑크 로젠틀라트가 발표하였습니다.

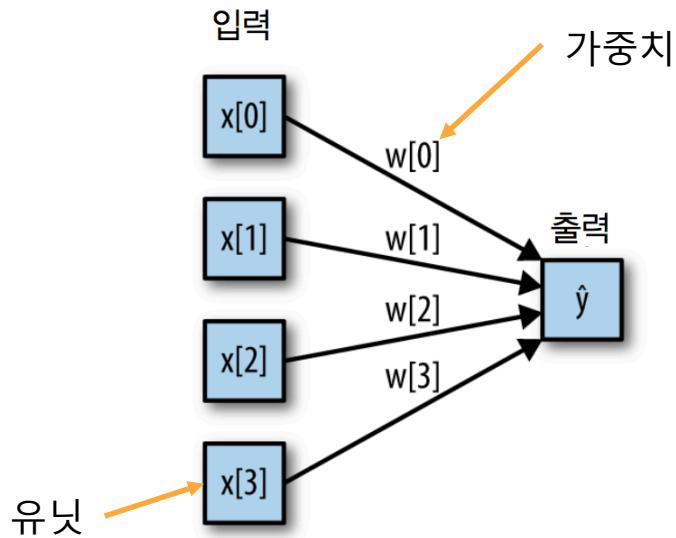
종종 뉴럴 네트워크를 멀티레이어 퍼셉트론 multilayer perceptron 으로도 부릅니다.

사이킷런은 `sklearn.linear_model.Perceptron` 클래스를 제공하다가 0.18 버전에서 `MLPClassifier`, `MLPRegressor` 추가되었습니다.



선형 모델의 일반화

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b$$



신경망 그림에서 편향이 표현되지 않는 경우가 종종 있습니다.

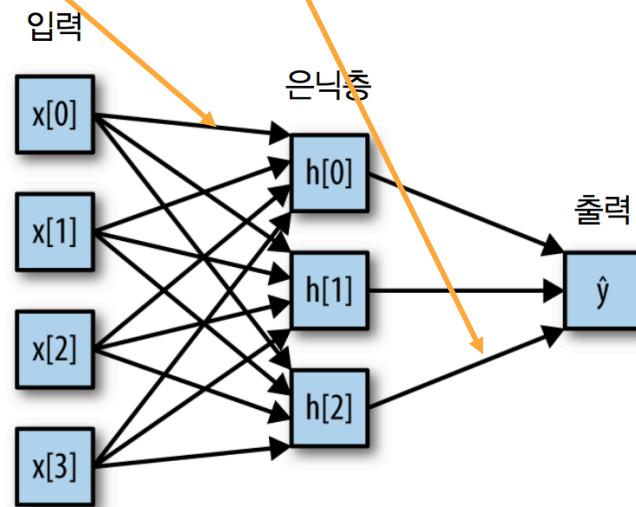
다층 퍼셉트론

$$h[0] = w[0,0] \times x[0] + w[1,0] \times x[1] + w[2,0] \times x[2] + w[3,0] \times x[3] + b[0]$$

$$h[0] = w[0,1] \times x[0] + w[1,1] \times x[1] + w[2,1] \times x[2] + w[3,1] \times x[3] + b[1]$$

$$h[0] = w[0,2] \times x[0] + w[1,2] \times x[1] + w[2,2] \times x[2] + w[3,2] \times x[3] + b[2]$$

$$\hat{y} = v[0] \times h[0] + v[1] \times h[1] + v[2] \times h[2] + b$$



Naming

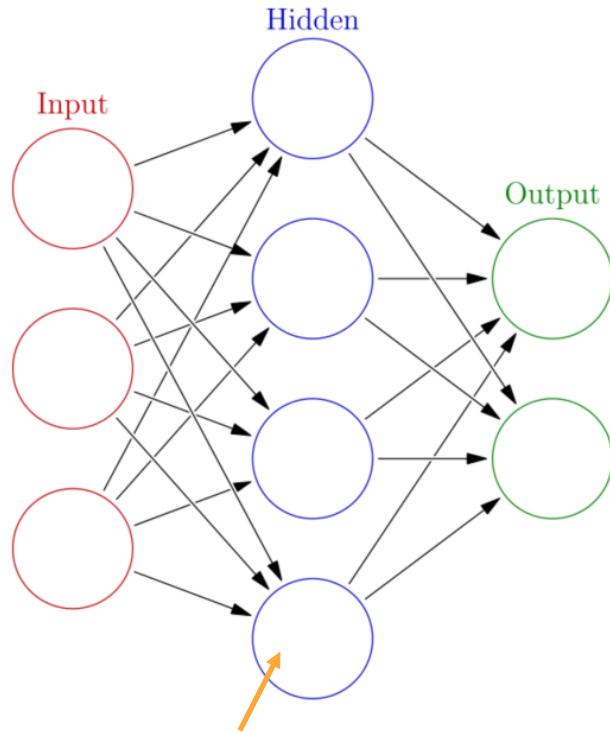
완전 연결 뉴럴 네트워크
(Fully Connected Neural Network)

덴스 네트워크
(Dense Network)

멀티 레이어 퍼셉트론
(Multi-Layer Perceptron)

피드 포워드 뉴럴 네트워크
(Feed Forward Neural Network)

딥 러닝
(Deep Learning)



뉴런 혹은 유닛 unit으로 불립니다.
(동물의 뇌와 전혀 관련이 없습니다)

다층 퍼셉트론

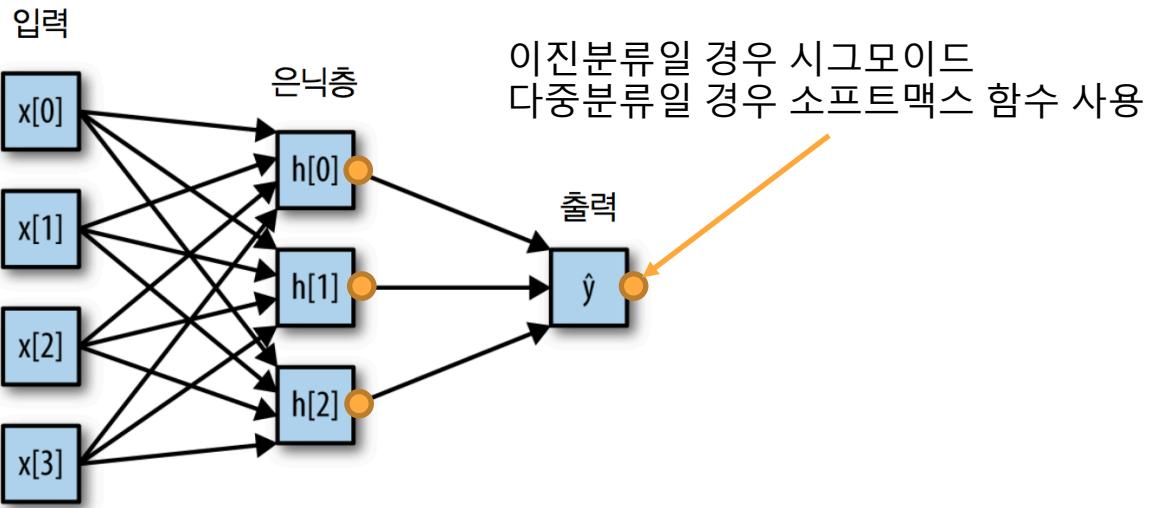
$$h[0] = \tanh(w[0,0] \times x[0] + w[1,0] \times x[1] + w[2,0] \times x[2] + w[3,0] \times x[3] + b[0])$$

$$h[0] = \tanh(w[0,1] \times x[0] + w[1,1] \times x[1] + w[2,1] \times x[2] + w[3,1] \times x[3] + b[1])$$

$$h[0] = \tanh(w[0,2] \times x[0] + w[1,2] \times x[1] + w[2,2] \times x[2] + w[3,2] \times x[3] + b[2])$$

$$\hat{y} = v[0] \times h[0] + v[1] \times h[1] + v[2] \times h[2] + b$$

비선형 활성화 함수



비용 함수

로지스틱 회귀와 마찬가지로 로지스틱 비용 함수(이진분류)나 크로스엔트로피 비용 함수(다중분류)를 사용합니다.

$$-\sum_{i=1}^n y \log(\hat{y})$$

모델 함수가 비선형이므로 최적값을 해석적으로 계산하지 못해 경사 하강법^{gradient descent} 계열의 알고리즘을 즐겨 사용합니다.

각 가중치 w 에 대해 비용 함수를 미분하여 기울기 아래쪽으로 조금씩(learning_rate) 이동합니다.

비용 함수를 직접 w 에 대해 미분하는 대신 체인룰을 이용해 미분값(그래디언트)을 누적하여 곱해갑니다(역전파backpropagation)

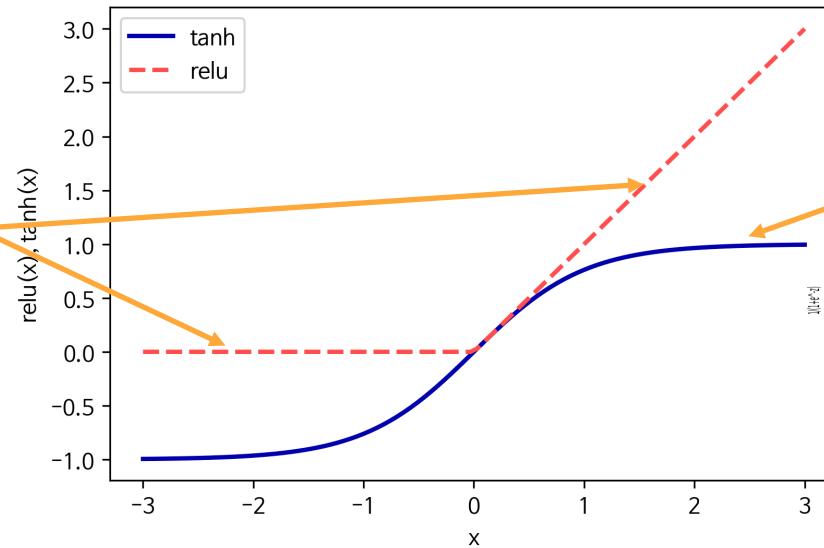
$$\frac{d\hat{y}}{dw} = \frac{d\hat{y}}{du} \cdot \frac{du}{dw}$$

활성화 함수 activation function

은닉 유닛의 가중치 합의 결과에 비선형성을 주입

렐루 ReLU, 하이퍼볼릭 탄젠트 \tanh , 시그모이드 sigmoid

$x > 0$ 그래디언트는 1
 $x = 0$ (sub)그래디언트는 0
 $x < 0$ 그래디언트는 0
(여러 변종 등장)



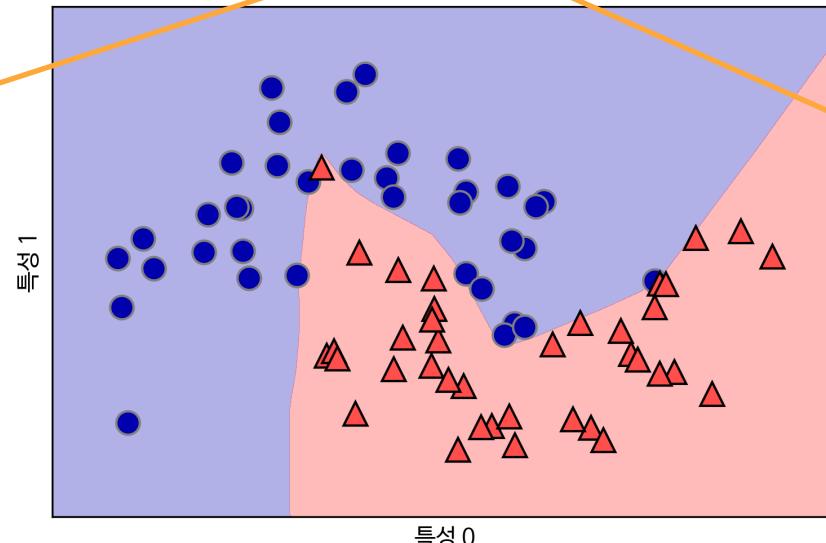
기울기가 0에 가까워집니다.

$$\frac{dy}{dw} = \frac{dy}{du} \cdot \frac{du}{dw}$$

MLPClassifier + two_moons

```
In [94]: from sklearn.neural_network import MLPClassifier  
from sklearn.datasets import make_moons  
  
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,  
random_state=42)  
  
mlp = MLPClassifier(solver='lbfgs', random_state=0).fit(X_train, y_train)
```

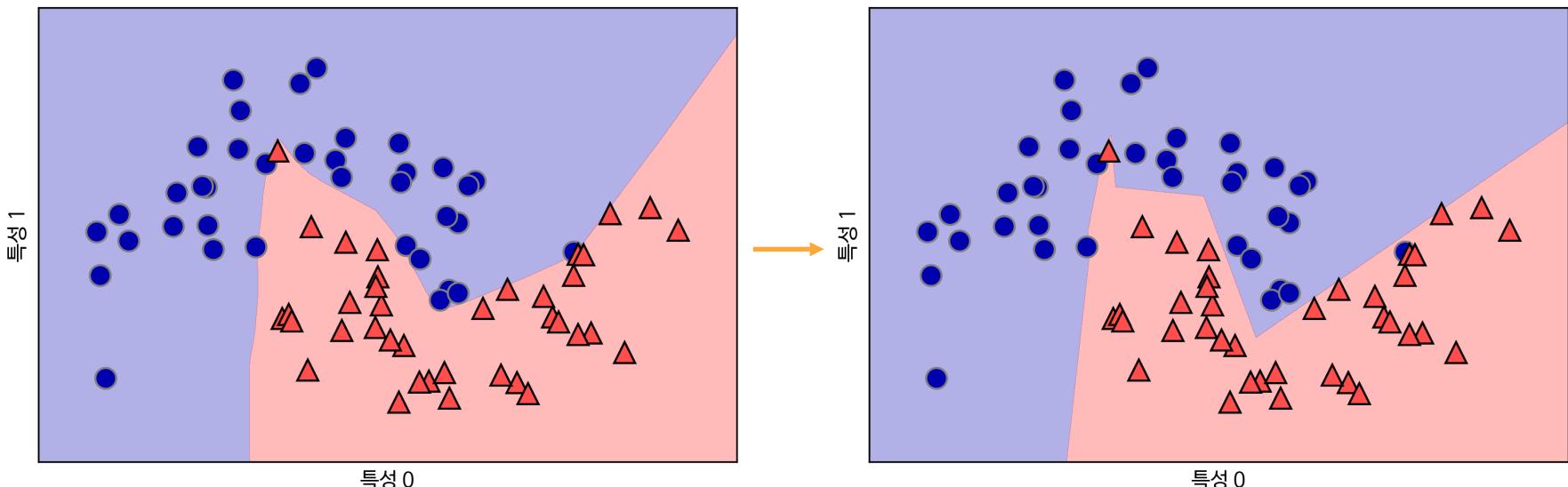
Limited BFGS
의사 뉴턴 방법



hidden_layer_sizes=[100],
activation='relu'

은닉 유닛 개수 = 10

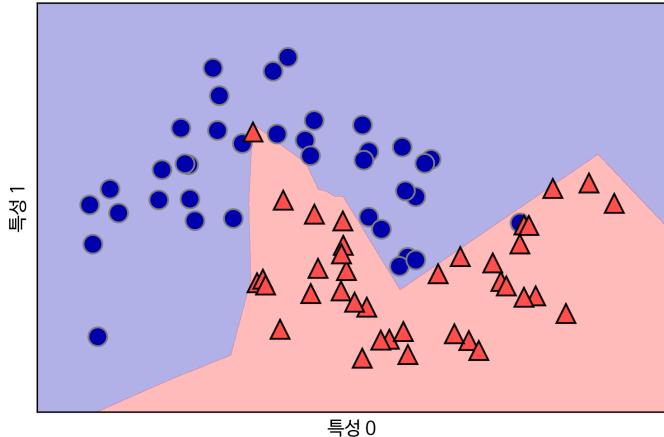
```
In [95]: mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
```



레이어 추가, tanh 활성화 함수

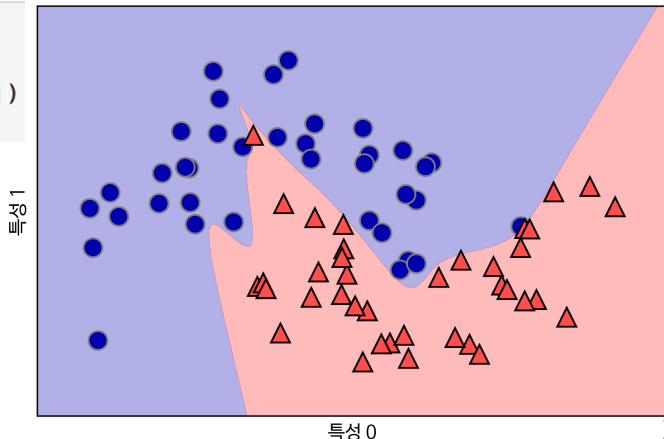
In [96]: # 10개의 유닛으로 된 두 개의 은닉층

```
mlp = MLPClassifier(solver='lbfgs', random_state=0,  
                     hidden_layer_sizes=[10, 10])  
mlp.fit(X_train, y_train)
```

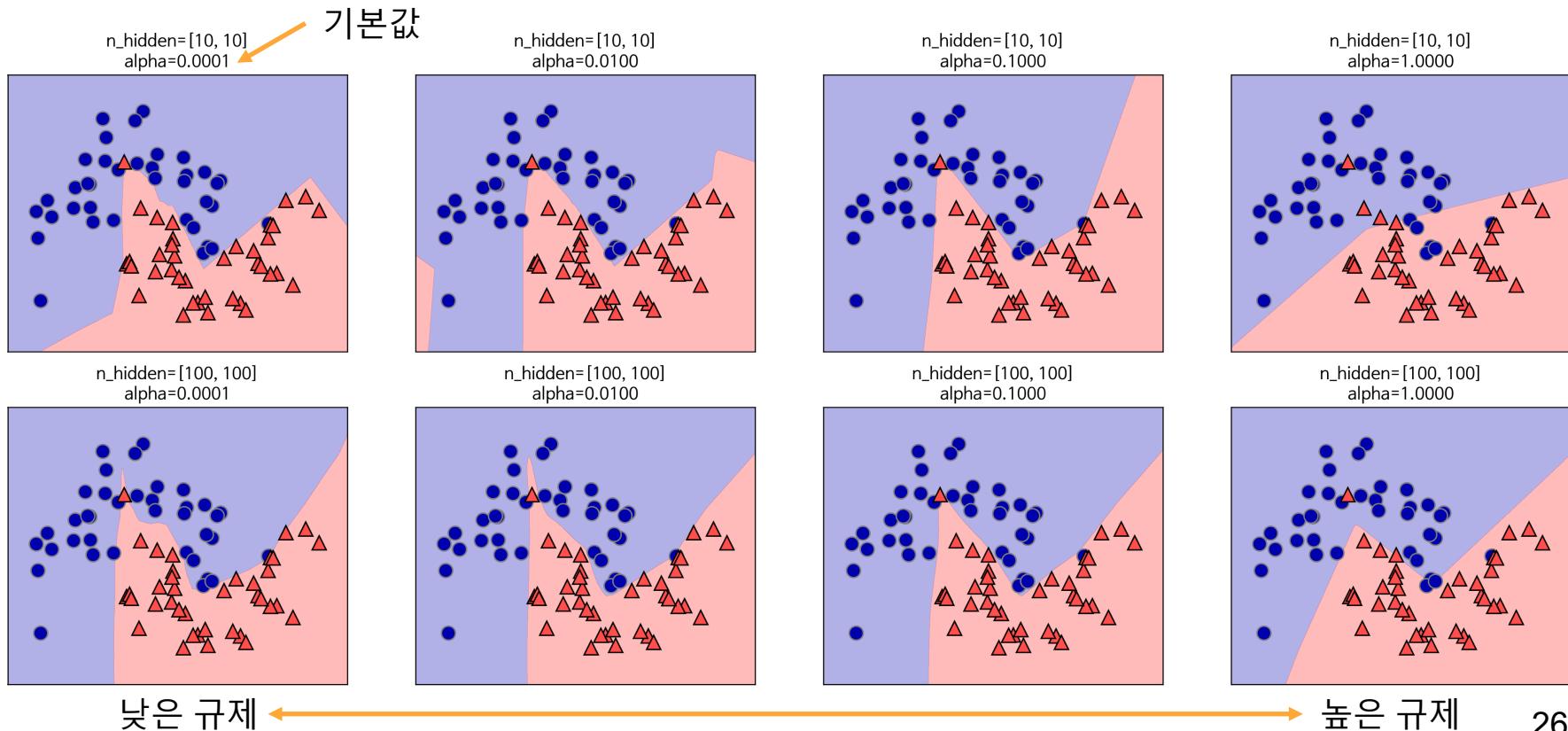


In [97]: # tanh 활성화 함수가 적용된 10개의 유닛으로 된 두 개의 은닉층

```
mlp = MLPClassifier(solver='lbfgs', activation='tanh',  
                     random_state=0, hidden_layer_sizes=[10, 10])  
mlp.fit(X_train, y_train)
```



L2 규제 매개변수 alpha



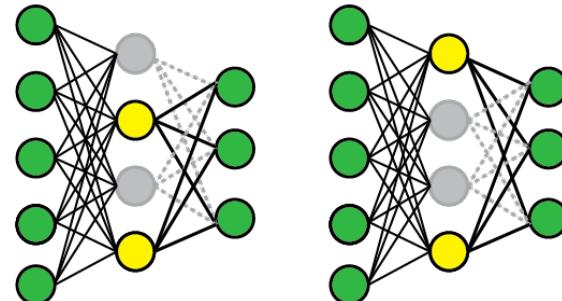
신경망의 복잡도

은닉층의 수가 많을 수록,

은닉층의 유닛 개수가 많을 수록,

규제가 낮을 수록 복잡도가 증가됨

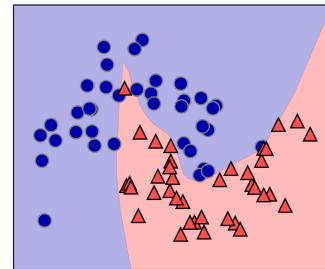
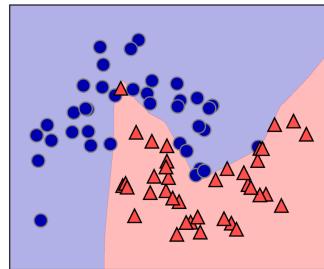
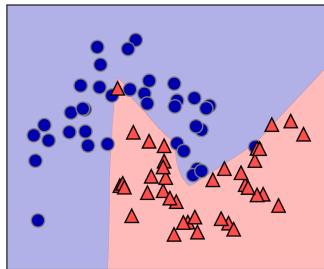
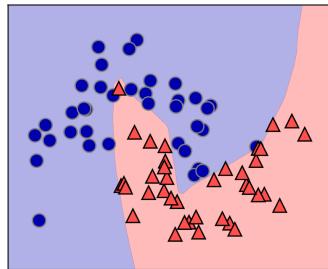
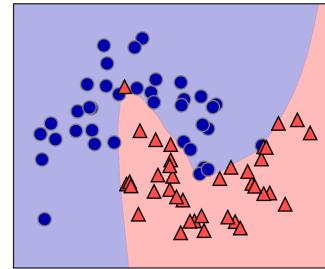
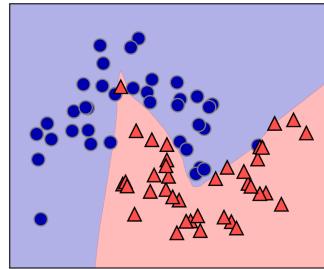
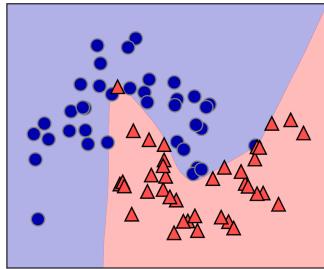
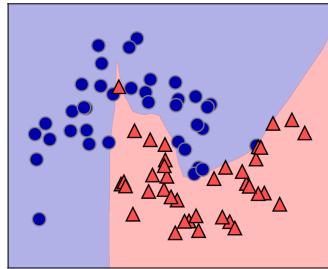
모델 훈련시 **훈련 샘플마다** 은닉층의 유닛의 일부를 랜덤하게 작동시키지 않아
마치 여러개의 신경망을 앙상블하는 것 같은 드롭아웃(dropout)이 신경망에서 과대적합
방지하는 대표적인 방법 → scikit-learn에 추가될 예정



랜덤 초기화

모델을 훈련할 때 가중치를 무작위로 초기화

모델의 크기와 복잡도가 낮으면 영향을 미칠 수 있음



가중치 초기화 방법

Scikit-Learn에서는 Glorot 초기화 방식을 사용하여 가중치와 편향을 모두 초기화합니다.

시그모이드 함수: $-\sqrt{\frac{2}{n_{inputs}+n_{outputs}}} \sim +\sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$ 의 균등 분포

tanh, relu 함수: $-\sqrt{\frac{6}{n_{inputs}+n_{outputs}}} \sim +\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$ 의 균등 분포

* Xavier 초기화

시그모이드: $\pm \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$ tanh: $\pm 4 \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$ relu: $\pm \sqrt{2} \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$

MLPClassifier + cancer dataset

신경망에서도 데이터의 범주에 민감함

```
In [101]: X_train, X_test, y_train, y_test = train_test_split(  
            cancer.data, cancer.target, random_state=0)  
  
mlp = MLPClassifier(random_state=42)  
mlp.fit(X_train, y_train)  
  
print("훈련 세트 정확도: {:.2f}".format(mlp.score(X_train, y_train)))  
print("테스트 세트 정확도: {:.2f}".format(mlp.score(X_test, y_test)))
```

훈련 세트 정확도: 0.91
테스트 세트 정확도: 0.88

```
In [102]: # 훈련 세트 각 특성의 평균을 계산합니다  
mean_on_train = X_train.mean(axis=0)  
# 훈련 세트 각 특성의 표준 편차를 계산합니다  
std_on_train = X_train.std(axis=0)  
  
# 데이터에서 평균을 빼고 표준 편차로 나누면  
# 평균 0, 표준 편차 1 인 데이터로 변환됩니다.  
X_train_scaled = (X_train - mean_on_train) / std_on_train  
# (훈련 데이터의 평균과 표준 편차를 이용해) 같은 변화를 테스트 세트에도 합니다  
X_test_scaled = (X_test - mean_on_train) / std_on_train
```

평균이 0이고, 분산이 1인 표준정규분포
표준 점수 또는 z 점수
StandardScaler

MLPClassifier + adam

최대 반복횟수 도달
(기본값 200)

```
mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)

print("훈련 세트 정확도: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
print("테스트 세트 정확도: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

훈련 세트 정확도: 0.991
테스트 세트 정확도: 0.965

```
/Users/rickypark/anaconda/envs/introduction_to_ml_with_python/lib/python3.!
% (), ConvergenceWarning)
```

In [103]:

```
mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)

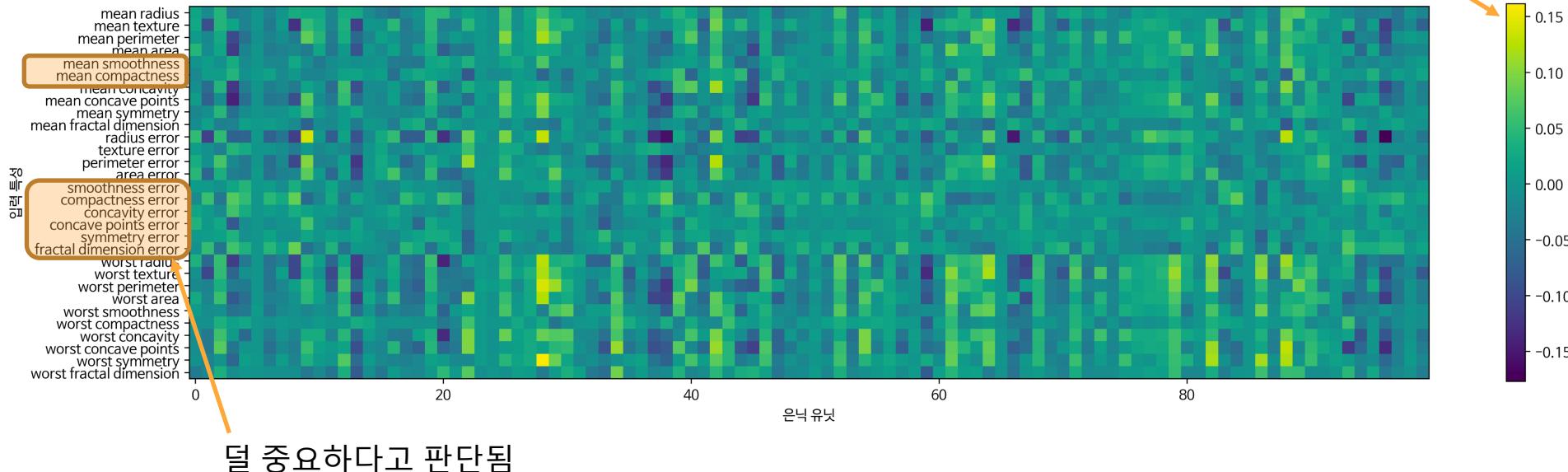
print("훈련 세트 정확도: {:.3f}".format(mlp.score(X_train_scaled, y_train)))
print("테스트 세트 정확도: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

훈련 세트 정확도: 0.993
테스트 세트 정확도: 0.972

신경망의 가중치 검사

입력층과 은닉층 사이의 가중치

밝을수록 큰 값



은닉층과 출력층 사이의 가중치는 해석하기 더 어렵습니다.

DL framework landscape



	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	+
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	
	Python							

PyTorch
Caffe2
(Python)

장단점과 매개변수

장점

충분한 시간과 데이터가 있으면 매우 복잡한 모델을 만들 수 있습니다.
종종 다른 알고리즘을 압도하는 성능을 발휘합니다(음성인식, 이미지분류, 번역 등)

단점

데이터 전처리에 민감합니다(배치 정규화).
이종의 데이터 타입일 경우 트리모델이 더 나을 수 있습니다.
매개변수 튜닝이 매우 어렵습니다(학습된 모델 재사용).
Scikit-Learn은 콘볼루션 convolution이나 리커런트 recurrent 신경망을 제공하지 않습니다.

매개변수

solver=['adam', 'sgd', 'lbfgs'],
sgd일 경우 momentum + nesterovs_momentum
alpha(L2 규제) $\alpha \uparrow \rightarrow$ 규제 $\uparrow w \downarrow$
 $\alpha \downarrow \rightarrow$ 규제 $\downarrow w \uparrow$

신경망의 복잡도와 권장 설정

100개의 특성과 100개의 은닉 유닛, 1개의 출력 유닛 = 10,100개의 가중치

+ 100개의 유닛을 가진 은닉층 추가 → 10,000개의 가중치 증가

만약 1000개 유닛을 가진 은닉층이라면 $101,000 + 1,000,000 = 1,101,000$ 개

solver의 기본값은 ‘adam’으로 데이터 스케일에 조금 민감합니다.

("The Marginal Value of Adaptive Gradient Methods in Machine Learning," A. C. Wilson et al. (2017),에서 Adam, RMSProp 등이 보여준 회의적인 결과로 모멘텀 방식을 함께 고려해야 합니다.)

‘lbfgs’는 안정적이지만 규모가 큰 모델이나 대량의 데이터셋에서는 시간이 오래 걸립니다.

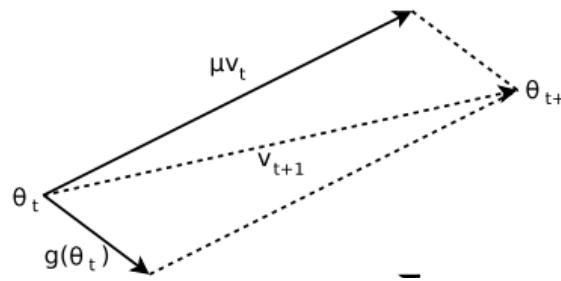
‘sgd’는 ‘momentum’과 ‘nesterov_momentum’ 옵션과 함께 사용합니다.

모멘텀

모멘텀 알고리즘은 이전 그레디언트 값을 가속도와 같이 생각하여 전역 최적값의 방향으로 더 빠르게 수렴하게 만듭니다.

momentum 매개변수는 일종의 댐퍼와 같은 역할을 합니다.

$$\begin{aligned} &\text{momentum} \quad \text{learning_rate} \\ &v_{t+1} = \mu v_t - \epsilon g(\theta_t) \\ &\theta_{t+1} = \theta_t + v_{t+1} \end{aligned}$$



(a) SGD without momentum



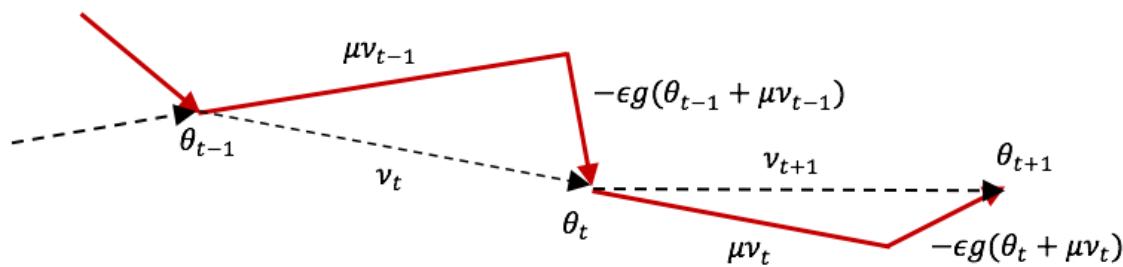
(b) SGD with momentum

네스테로프 모멘텀

네스테로프 모멘텀은 모멘텀 방식으로 진행한 후의 그래디언트를 계산하여 현재에 적용합니다.

$$\begin{aligned}v_{t+1} &= \mu v_t - \epsilon g(\theta_t + \mu v_t) \\ \theta_{t+1} &= \theta_t + v_{t+1}\end{aligned}$$

실제 구현에서는 앞선 그래디언트를 계산하는 대신 모멘텀 방식을 두 번 적용하여 네스테로프의 근사값을 구합니다. (<https://tensorflow.blog/2017/03/22/momentum-nesterov-momentum/> 참조)



분류의 불확실성 추정

분류의 불확실성

어떤 테스트 포인트에 대해 예측 클래스 뿐만 아니라 얼마나 그 클래스임을 확신하는지가 중요할 때가 있음

대부분 decision_function 과 predict_proba 메서드 둘 중 하나는 제공함

```
In [107]: from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_circles
X, y = make_circles(noise=0.25, factor=0.5, random_state=1)

# 예제를 위해 클래스의 이름을 "blue" 와 "red" 로 바꿉니다
y_named = np.array(["blue", "red"])[y]

# 여러개의 배열을 한꺼번에 train_test_split 에 넣을 수 있습니다
# 훈련 세트와 테스트 세트로 나누는 방식은 모두 같습니다.
X_train, X_test, y_train_named, y_test_named, y_train, y_test = \
    train_test_split(X, y_named, y, random_state=0)

# 그래디언트 부스팅 모델을 만듭니다
gbdt = GradientBoostingClassifier(random_state=0)
gbdt.fit(X_train, y_train_named)
```

결정 함수

이진 분류에서 decision_function() 반환값의 크기 (n_samples,)

```
In [108]: print("X_test.shape: {}".format(X_test.shape))
print("결정 함수 결과 형태: {}".format(gbrt.decision_function(X_test).shape))
```

```
X_test.shape: (25, 2)
결정 함수 결과 형태: (25, )
```

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b$$

```
In [109]: # 결정 함수 결과 중 앞부분 일부를 확인합니다
print("결정 함수:\n{}".format(gbrt.decision_function(X_test)[:6]))
```

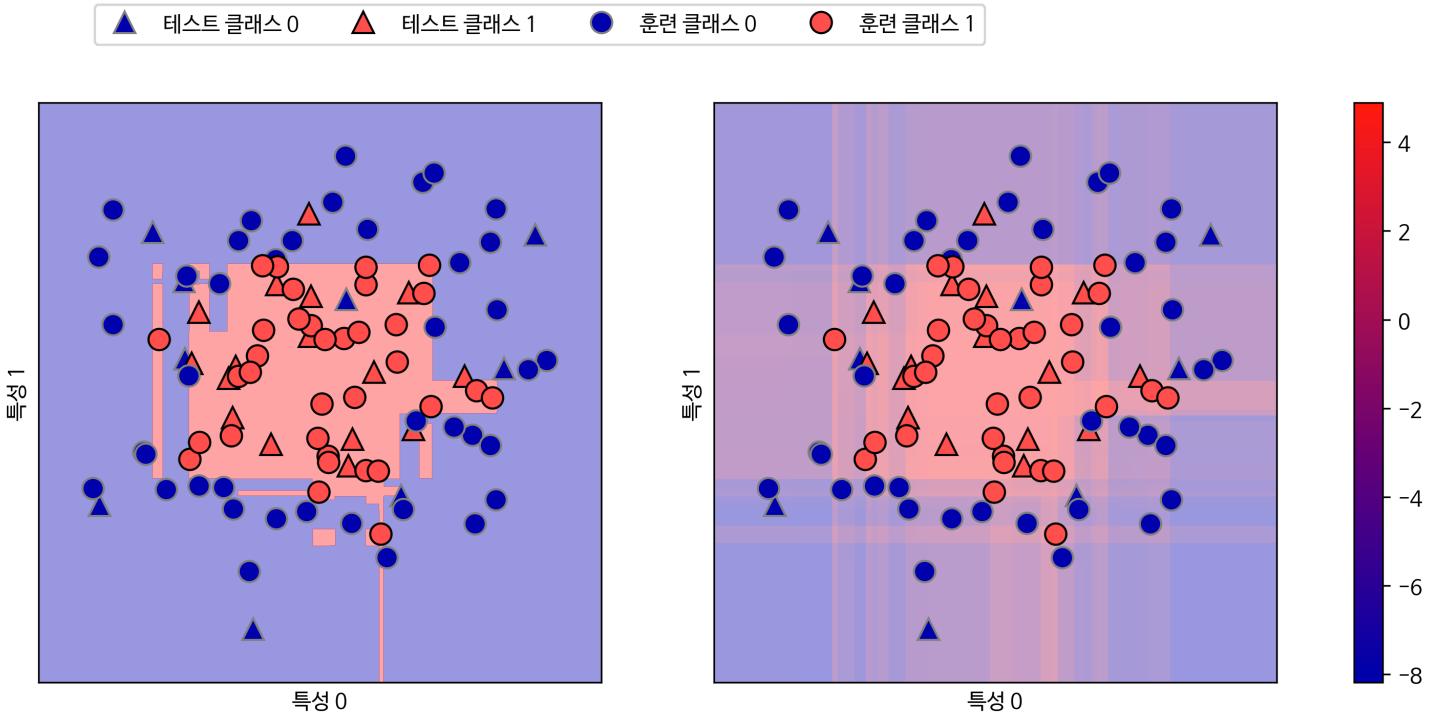
```
결정 함수:
[ 4.136 -1.702 -3.951 -3.626  4.29   3.662]
```

```
In [110]: print("임계치와 결정 함수 결과 비교:\n{}".format(
    gbrt.decision_function(X_test) > 0))
print("예측:\n{}".format(gbrt.predict(X_test)))
```

```
임계치와 결정 함수 결과 비교:
[ True False False False  True  True False  True  True False  True
  True False  True False False  True  True  True  True False
 False]
예측:
```

```
['red' 'blue' 'blue' 'blue' 'red' 'red' 'blue' 'red' 'red' 'red' 'blue'
 'red' 'red' 'blue' 'red' 'blue' 'blue' 'blue' 'red' 'red' 'red' 'red'
 'red' 'blue' 'blue']
```

결정 경계 + decision_function



예측 확률

`predict_proba()` 반환값의 크기 (`n_samples, n_classes`)

가장 큰 확률 값을 가진 클래스를 예측 클래스로 선택함

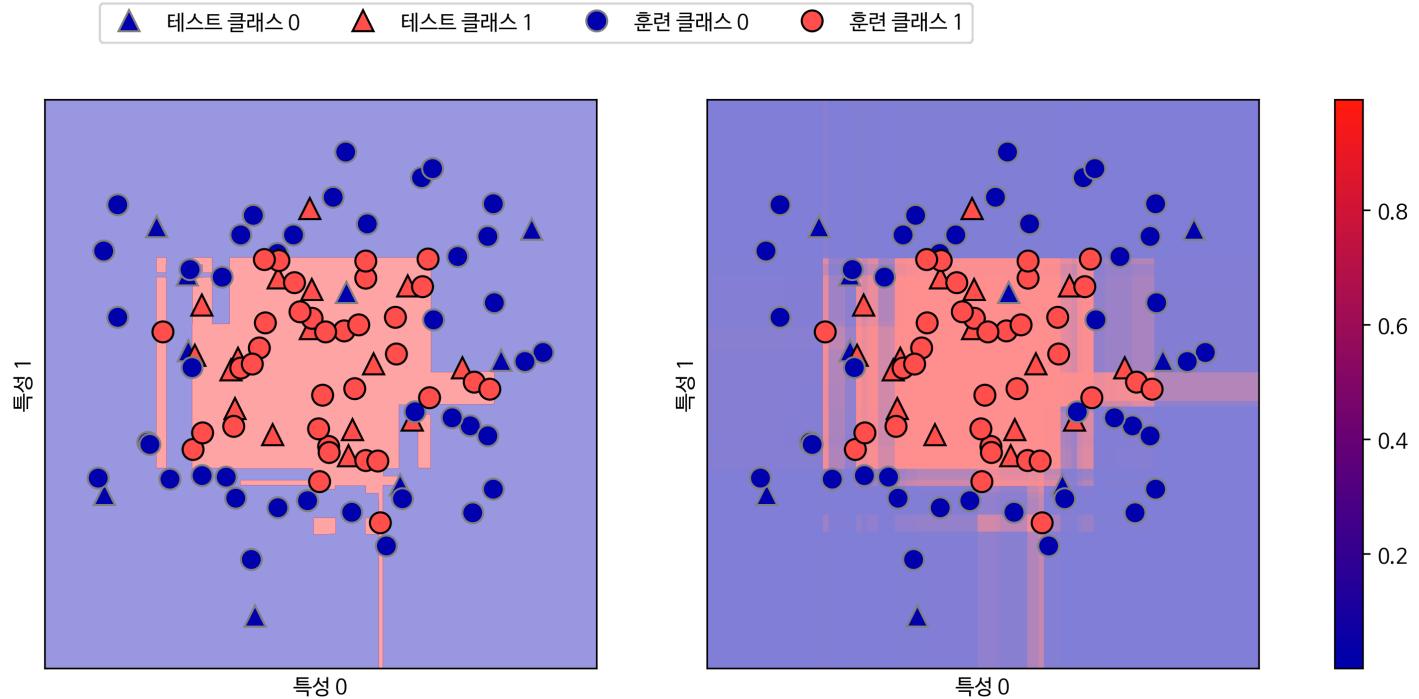
```
In [114]: print("확률 값의 형태: {}".format(gbrt.predict_proba(X_test).shape))
```

확률 값의 형태: (25, 2)

```
In [115]: # predict_proba 결과 중 앞부분 일부를 확인합니다
print("Predicted probabilities:\n{}".format(
    gbrt.predict_proba(X_test[:6])))
```

```
Predicted probabilities:
[[ 0.016  0.984]
 [ 0.846  0.154]
 [ 0.981  0.019]
 [ 0.974  0.026]
 [ 0.014  0.986]
 [ 0.025  0.975]]
```

결정 경계 + predict_proba



다중 분류 + decision_function

반환값의 크기는 (n_samples, n_classes)

가장 큰 값이 예측 클래스가 됨

```
In [117]: from sklearn.datasets import load_iris  
  
iris = load_iris()  
X_train, X_test, y_train, y_test = train_test_split(  
    iris.data, iris.target, random_state=42)  
  
gbrt = GradientBoostingClassifier(learning_rate=0.01, random_state=0)  
gbrt.fit(X_train, y_train)
```

```
In [118]: print("결정 함수의 결과 형태: {}".format(gbrt.decision_function(X_test).shape))  
# plot the first few entries of the decision function  
print("결정 함수 결과:\n{}".format(gbrt.decision_function(X_test)[:6, :]))
```

결정 함수의 결과 형태: (38, 3)

결정 함수 결과:

```
[[ -0.529  1.466 -0.504]  
 [ 1.512 -0.496 -0.503]  
 [-0.524 -0.468  1.52 ]  
 [-0.529  1.466 -0.504]  
 [-0.531  1.282  0.215]  
 [ 1.512 -0.496 -0.503]]
```

다중 분류 + predict_proba

반환값의 크기는 (n_samples, n_classes)

```
In [120]: # predict_proba 결과 중 앞부분 일부를 확인합니다
print("예측 확률:\n{}".format(gbdt.predict_proba(X_test)[:6]))
# 행 방향으로 확률을 더하면 1 이 됩니다
print("합: {}".format(gbdt.predict_proba(X_test)[:6].sum(axis=1)))
```

```
예측 확률:
[[ 0.107  0.784  0.109]
 [ 0.789  0.106  0.105]
 [ 0.102  0.108  0.789]
 [ 0.107  0.784  0.109]
 [ 0.108  0.663  0.228]
 [ 0.789  0.106  0.105]]
합: [ 1.  1.  1.  1.  1.]
```

```
In [121]: print("가장 큰 예측 확률의 인덱스:\n{}".format(
    np.argmax(gbdt.predict_proba(X_test), axis=1)))
print("예측:\n{}".format(gbdt.predict(X_test)))
```

```
가장 큰 예측 확률의 인덱스:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0]
예측:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0]
```

요약 및 정리

알고리즘 정리

최근접 이웃

작은 데이터셋일 경우, 기본 모델로서 좋고 설명하기 쉬움.

선형 모델

첫 번째로 시도할 알고리즘. 대용량 데이터셋 가능. 고차원 데이터에 가능.

나이브 베이즈

분류만 가능. 선형 모델보다 훨씬 빠름. 대용량 데이터셋과 고차원 데이터에 가능.

선형 모델보다 덜 정확함.

결정 트리

매우 빠름. 데이터 스케일 조정이 필요 없음. 시각화하기 좋고 설명하기 쉬움.

랜덤 포레스트

결정 트리 하나보다 거의 항상 좋은 성능을 냄. 매우 안정적이고 강력함.
데이터 스케일 조정 필요 없음. 고차원 희소 데이터에는 잘 안 맞음.

그래디언트 부스팅

랜덤 포레스트보다 조금 더 성능이 좋음. 랜덤 포레스트보다 학습은 느리나 예측은
빠르고 메모리를 조금 사용. 랜덤 포레스트보다 매개변수 튜닝이 많이 필요함.

서포트 벡터 머신

비슷한 의미의 특성으로 이뤄진 중간 규모 데이터셋에 잘 맞음.
데이터 스케일 조정 필요. 매개변수에 민감.

신경망

특별히 대용량 데이터셋에서 매우 복잡한 모델을 만들 수 있음.
매개변수 선택과 데이터 스케일에 민감. 큰 모델은 학습이 오래 걸림.