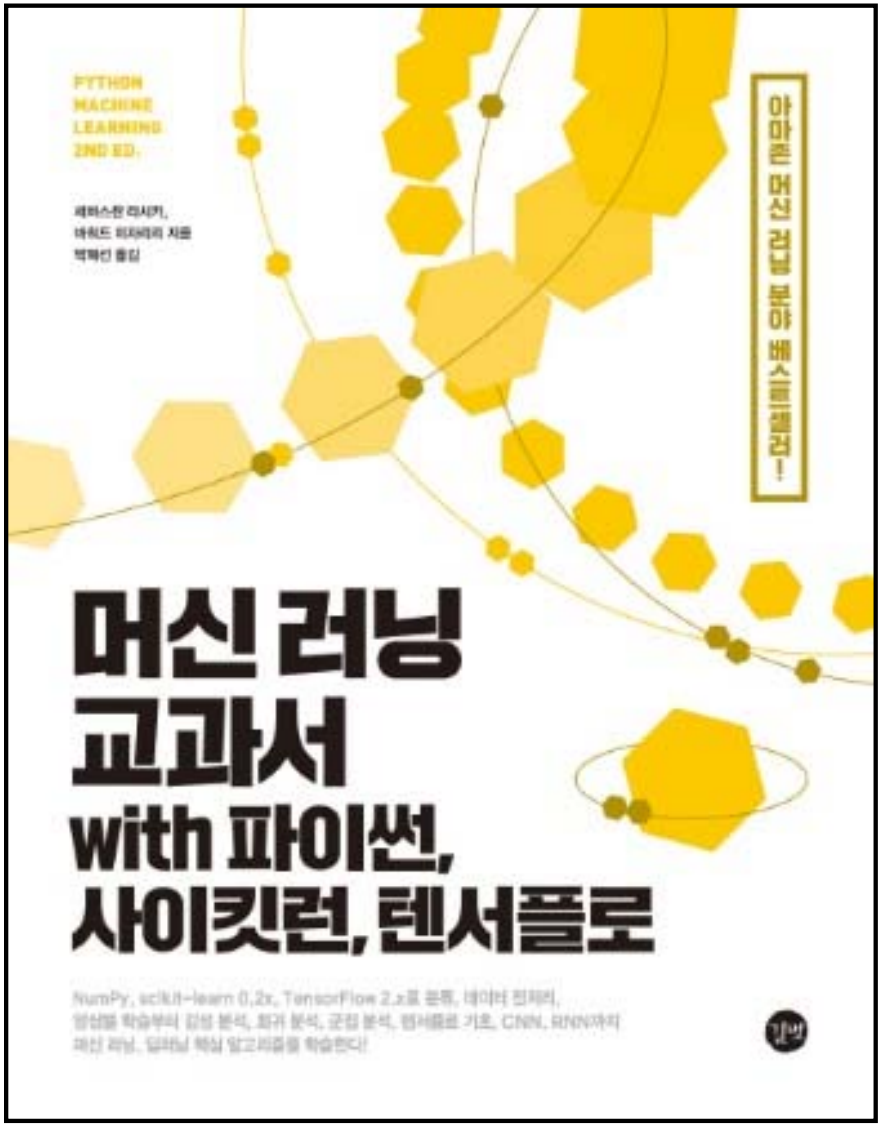
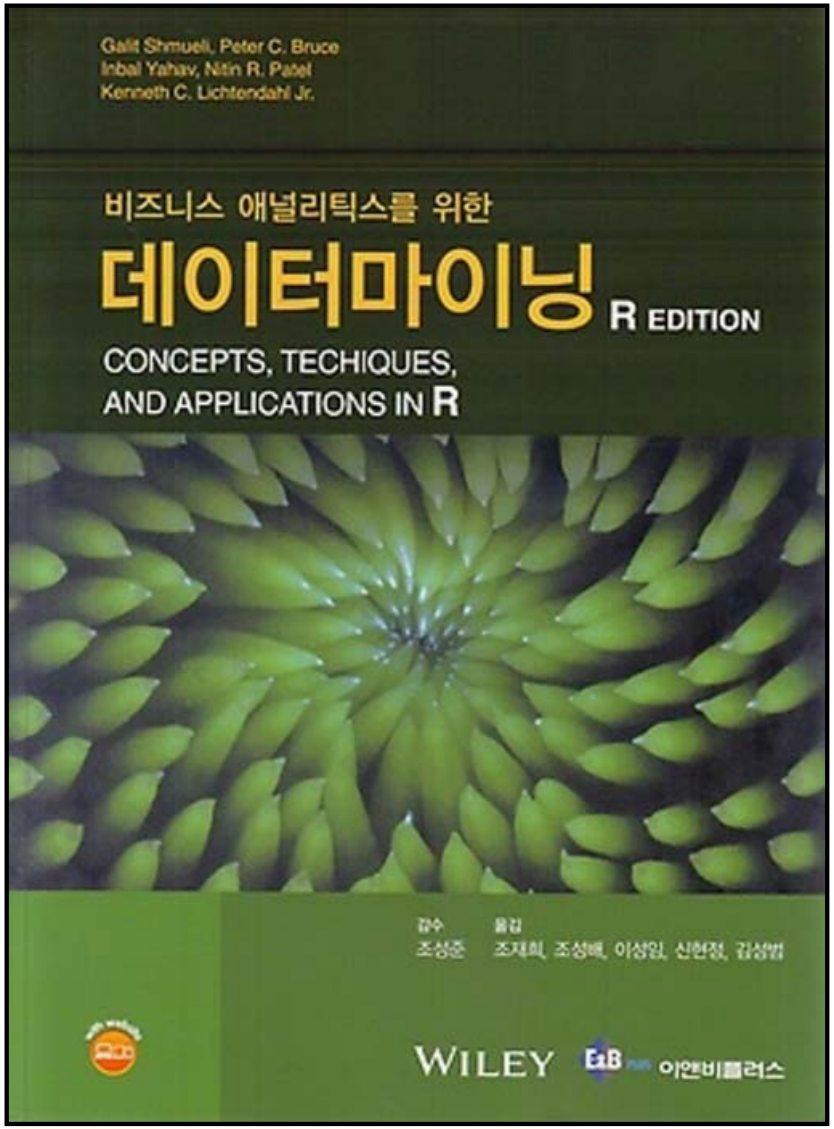
Ch08.LR과 SVM

교재


PYTHON
MACHINE
LEARNING
2ND ED.
아마존 머신 러닝 분야 베스트셀러!
머신 러닝
교과서
with 파이썬,
사이킷런, 텐서플로


Galit Shmueli, Peter C. Bruce
Inbal Yahav, Nitin R. Patel
Kenneth C. Lichtendahl Jr.
비즈니스 애널리틱스를 위한
데이터마이닝 R EDITION
CONCEPTS, TECHIQUES,
AND APPLICATIONS IN R
WILEY
이앤비플러스

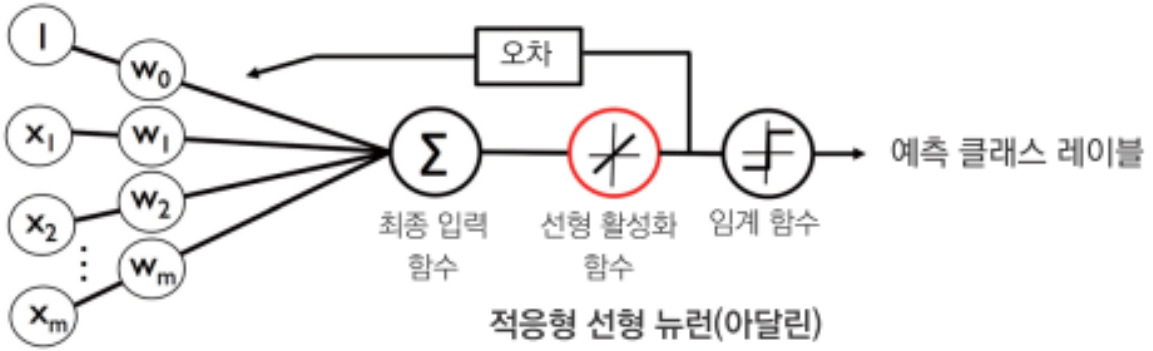
학습목표

- 로지스틱 회귀분석(Logistic Regression)
- 서포트 벡터 머신(SVM)

1.로지스틱 회귀분석

로지스틱 회귀분석

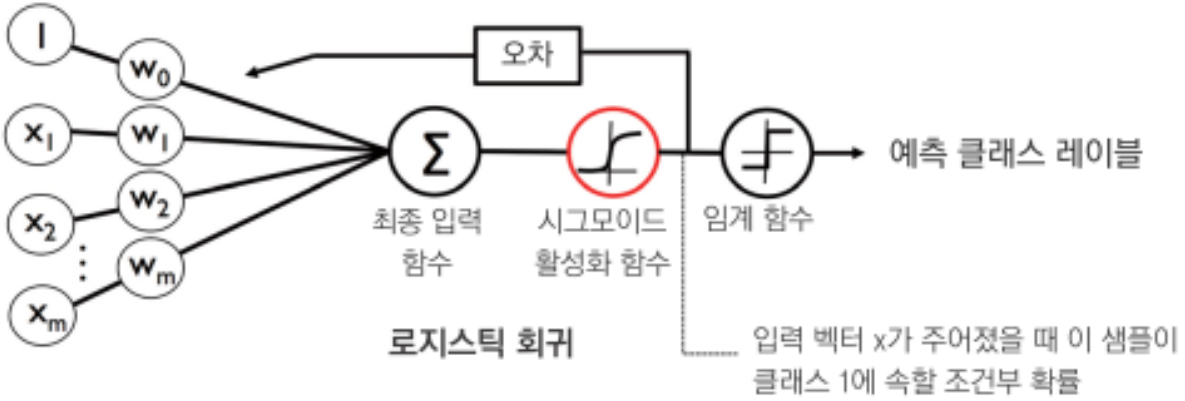
- 로지스틱 회귀분석
  - 클래스 소속 확률(odds)을 이용하여 분류
  - 활성함수 : Sigmoid 함수 이용



적응형 선형 뉴런(아달린)

$$h(z) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots$$

$$-\infty \leq a \leq \infty$$



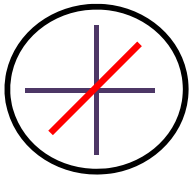
로지스틱 회귀

$$h(z) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots)}}$$

$$0 \leq a \leq 1$$

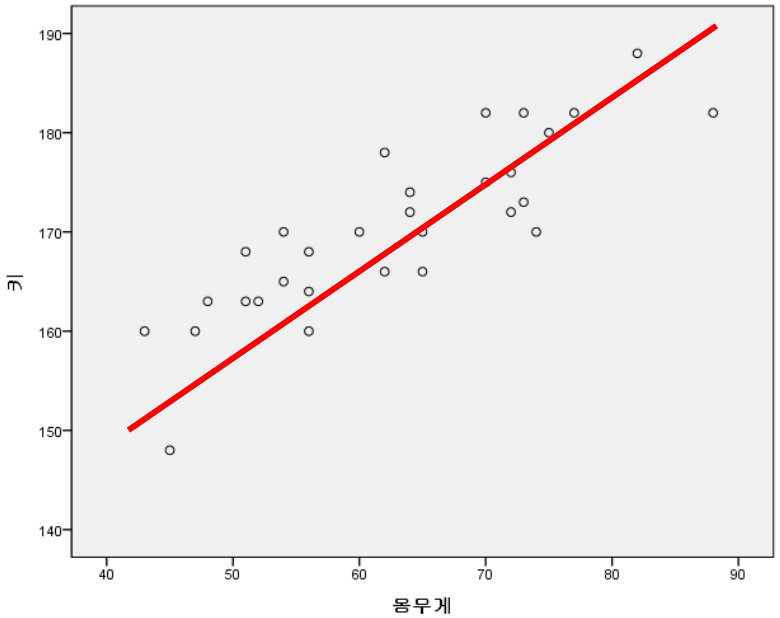
활성함수

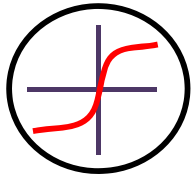
- 아달라인과 로지스틱 활성함수 차이



$$h(z) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots$$

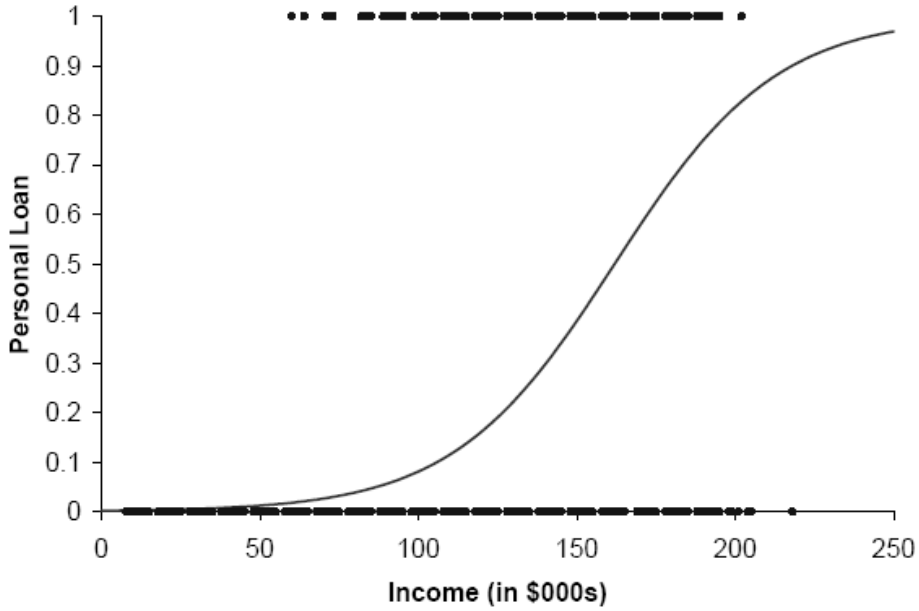
$$-\infty \leq a \leq \infty$$





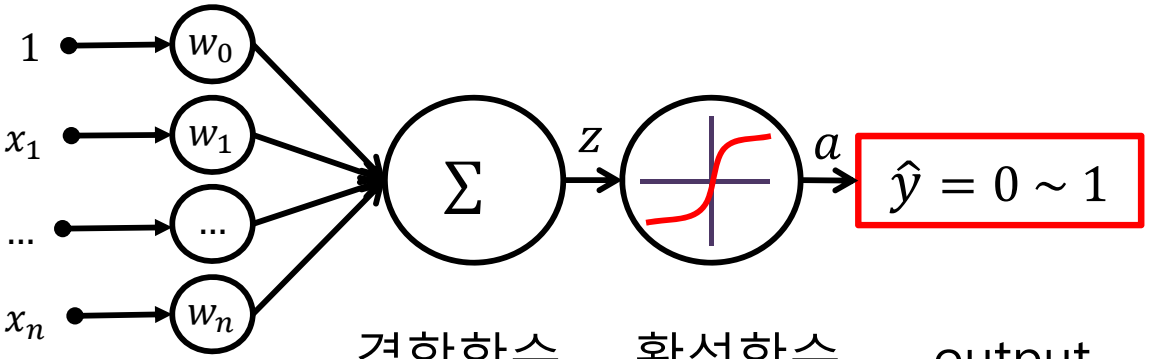
$$h(z) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots)}}$$

$$0 \leq a \leq 1$$



활성함수

- 활성함수
  - 결합함수 ≠ 활성함수



$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

$$\hat{y} = a = h(z) = \frac{1}{1 + e^{-(z)}}$$

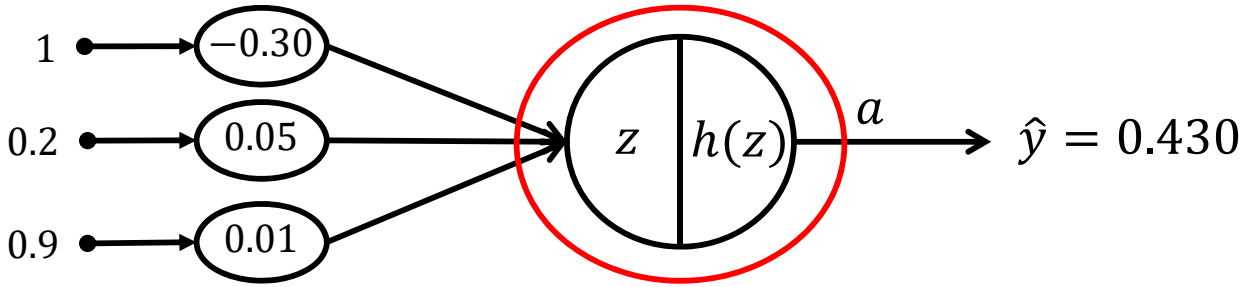
$$= \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n)}}$$

그림 13-5 다양한 활성화 함수 목록

활성화 함수	공식	사례	1차원 그래프
선형 함수	$\phi(z) = z$	아달린, 선형 회귀	
단위 계단 함수 (헤비사이드 함수)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	퍼셉트론 유형	
부호 함수	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	퍼셉트론 유형	
부분 선형 함수	$\phi(z) = \begin{cases} 0 & z \le -½ \\ z + ½ & -½ \le z \le ½ \\ 1 & z \ge ½ \end{cases}$	서포트 벡터 머신	
로지스틱 (시그모이드) 함수	$\phi(z) = \frac{1}{1 + e^{-z}}$	로지스틱 회귀, 다층 신경망	
하이퍼볼릭 탄젠트 (tanh) 함수	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	다층 신경망, RNN	
렐루 함수	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	다층 신경망, CNN	

활성함수

No	B1	Fat	Salt	w_0	w_1	w_2	z	$h(z)=\hat{y}$	y	$(y-\hat{y})$	$J(w)$	Δw_0	Δw_1	Δw_2
1	1	0.2	0.9	-0.300	0.050	0.010	-0.281	0.430	1	0.570	0.843			
2	1	0.1	0.1				-0.294	0.427	0	-0.427	0.557			
3	1	0.2	0.4				-0.286	0.429	0	-0.429	0.560			
4	1	0.2	0.5				-0.285	0.429	0	-0.429	0.561			
5	1	0.4	0.5				-0.275	0.432	1	0.568	0.840			
6	1	0.3	0.8				-0.277	0.431	1	0.569	0.841			
										cost$(J(w))$	4.203			



$$z = -0.30 \times 1 + 0.05 \times 0.2 + 0.01 \times 0.9 = -0.281$$

$$\hat{y} = a = h(z) = \frac{1}{1 + e^{-(z)}}$$

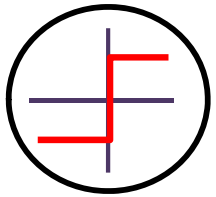
$$= \frac{1}{1 + e^{-(-0.281)}}$$

$$= 0.430$$

활성함수

- 활성함수
  - 퍼셉트론, 아달라인, 로지스틱 차이

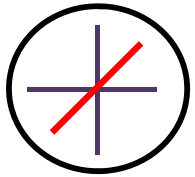
퍼셉트론



$$\hat{y} = a = h(z) = \begin{cases} -1 & if(z \leq 0) \\ 1 & if(z > 0) \end{cases}$$

$$\hat{y} = a = h(z) = h(-0.281) = -1$$

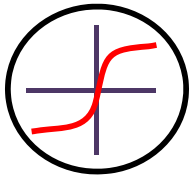
아달라인



$$\hat{y} = a = h(z) = z$$

$$\hat{y} = a = h(z) = z = h(-0.281) = -0.281$$

로지스틱

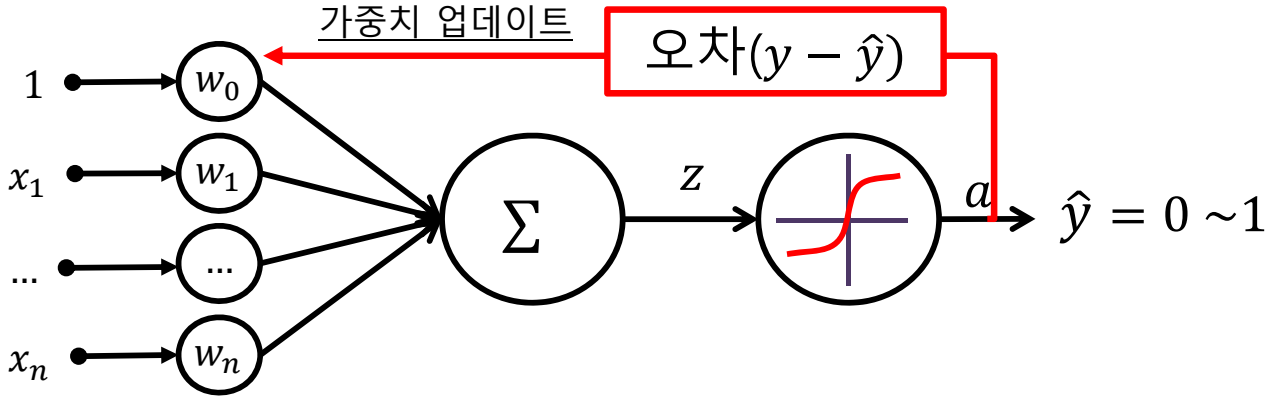


$$\hat{y} = a = h(z) = \frac{1}{1 + e^{-(z)}}$$

$$\hat{y} = a = h(z) = h(-0.281) = \frac{1}{1 + e^{-(-0.281)}} = 0.430$$

학습

- 학습
  - 비용함수 최소화하는 가중치

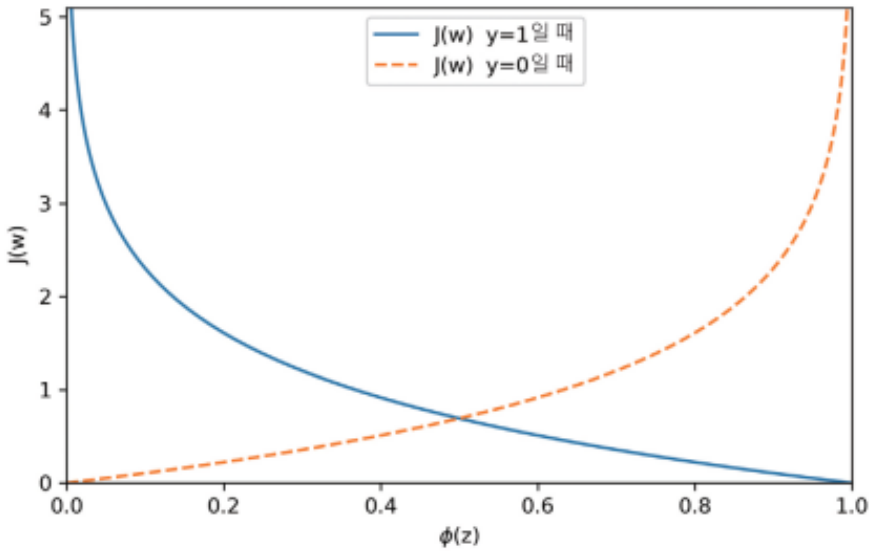


- 비용함수 $J(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

$$J(w) = \begin{cases} -\log\big(h(z)\big) & if\ y = 1 \\ -\log\big(1 - h(z)\big) & if\ y = 0 \end{cases}$$

하나로 결합

$$J(w) = \sum -y\log\big(h(z)\big) - (1-y)\log\big(1 - h(z)\big)$$

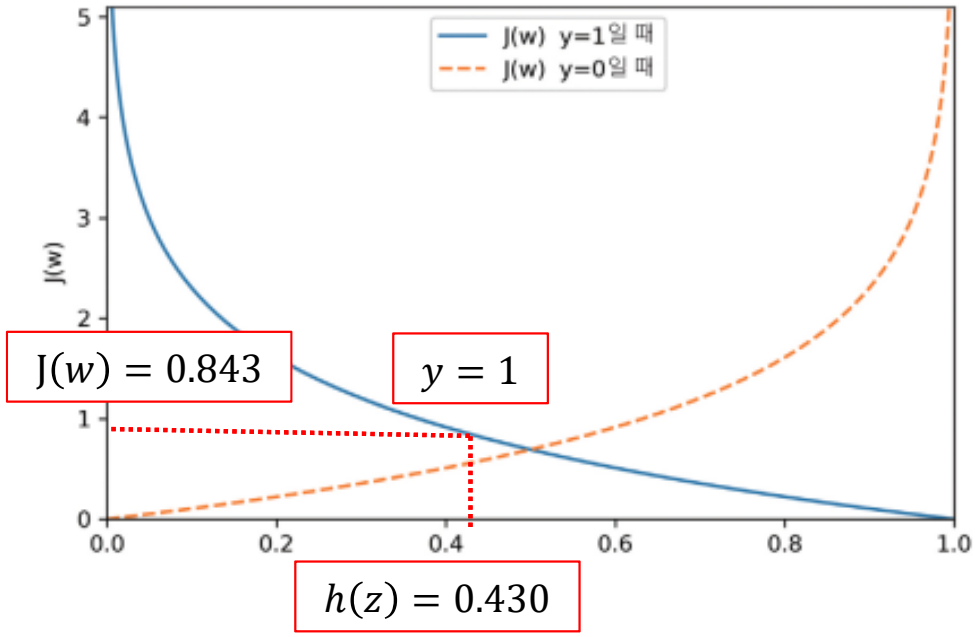


학습

No	B1	Fat	Salt	w_0	w_1	w_2	z	$h(z)=\hat{y}$	y	$(y-\hat{y})$	$J(w)$	Δw_0	Δw_1	Δw_2
1	1	0.2	0.9	-0.300	0.050	0.010	-0.281	0.430	1	0.570	0.843			
2	1	0.1	0.1				-0.294	0.427	0	-0.427	0.557			
3	1	0.2	0.4				-0.286	0.429	0	-0.429	0.560			
4	1	0.2	0.5				-0.285	0.429	0	-0.429	0.561			
5	1	0.4	0.5				-0.275	0.432	1	0.568	0.840			
6	1	0.3	0.8				-0.277	0.431	1	0.569	0.841			
										cost($J(w)$)	4.203			

$$J(w_1) = -y\log\big(h(z)\big) - (1-y)\log\big(1-h(z)\big)$$
$$= -1\log(0.430) - (0)\log(1-0.430)$$
$$= -1\log(0.430)$$
$$= 0.843$$

$$J(w) = \sum -y\log\big(h(z)\big) - (1-y)\log\big(1-h(z)\big)$$
$$= 0.843 + \cdots + 0.841 = 4.203$$



비용함수

- 비용함수

아달라인

$$J(w) = \frac{1}{2}\sum_{i=1}^{n=6}(y_i - \hat{y}_i)^2$$
$$= \frac{1}{2}(1.641) + \cdots + (1.631) = 3.208$$

로지스틱

$$J(w) = \sum -y\log\big(h(z)\big) - (1-y)\log\big(1-h(z)\big)$$
$$= 0.843 + \cdots + 0.841 = 4.203$$

가중치 업데이트

- 가중치 업데이트
  - Batch updating

$$w_j^{new} = w_j^{old} + \Delta w_j$$

  - 가중치 업데이트 : 비용함수를 최소화 – 비용함수를 미분

$$J(w) = \sum -y \log\big(h(z)\big) - (1-y)\log\big(1-h(z)\big)$$

$$J(w_1) = \frac{\partial J}{\partial w_1} = -\sum_{i=1}^{n}(y_i - \hat{y}_i)x_1$$

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta - \sum_{i=1}^{n}(y_i - \hat{y}_i)\, x_j = \eta \sum_{i=1}^{n}(y_i - \hat{y}_i)\, x_j$$

아달라인 함수 미분과 동일

$$J(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$J(w_1) = \frac{\partial J}{\partial w_1} = -\sum_{i=1}^{n}(y_i - \hat{y}_i)x_1$$

가중치 업데이트

No	B1	Fat	Salt	w_0	w_1	w_2	z	$h(z)=\hat{y}$	y	$(y-\hat{y})$	$J(w)$	Δw_0	Δw_1	Δw_2
1	1	0.2	0.9	-0.300	0.050	0.010	-0.281	0.430	1	0.570	0.843	0.570	0.114	0.513
2	1	0.1	0.1				-0.294	0.427	0	-0.427	0.557	-0.427	-0.043	-0.043
3	1	0.2	0.4				-0.286	0.429	0	-0.429	0.560	-0.429	-0.086	-0.172
4	1	0.2	0.5				-0.285	0.429	0	-0.429	0.561	-0.429	-0.086	-0.215
5	1	0.4	0.5				-0.275	0.432	1	0.568	0.840	0.568	0.227	0.284
6	1	0.3	0.8				-0.277	0.431	1	0.569	0.841	0.569	0.171	0.455
										cost($J(w)$)	4.203	0.422	0.298	0.823
												0.084	0.060	0.165

$$\Delta w_j = -\eta \times J(w_1) = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_{i=1}^{n} (y_i - \hat{y}_i)\, x_j$$

w_1일 때 , 가중치$= J(w_1) = \big(1-(0.430)\big) \times 0.2 + \cdots + \big(1-(431)\big) \times 0.2$

$$= (0.570) \times 0.2 + \cdots + (0.569) \times 0.2$$

$$= 0.114 + \cdots + 0.171 = 0.298$$

$$\Delta w_1 = \eta \times J(w_1) = 0.2 \times 0.298 = 0.060$$

가중치 업데이트

No	B1	Fat	Salt	w_0	w_1	w_2	z	$h(z)=\hat{y}$	y	$(y-\hat{y})$	$J(w)$	Δw_0	Δw_1	Δw_2
1	1	0.2	0.9	-0.300	0.050	0.010	-0.281	0.430	1	0.570	0.843	0.570	0.114	0.513
2	1	0.1	0.1				-0.294	0.427	0	-0.427	0.557	-0.427	-0.043	-0.043
3	1	0.2	0.4				-0.286	0.429	0	-0.429	0.560	-0.429	-0.086	-0.172
4	1	0.2	0.5				-0.285	0.429	0	-0.429	0.561	-0.429	-0.086	-0.215
5	1	0.4	0.5				-0.275	0.432	1	0.568	0.840	0.568	0.227	0.284
6	1	0.3	0.8				-0.277	0.431	1	0.569	0.841	0.569	0.171	0.455
										cost($J(w)$)	4.203	0.422	0.298	0.823
			new_w	-0.216	0.110	0.175						0.084	0.060	0.165

$$w_j^{new} = w_j^{old} + \Delta w_j$$

$$w_1^{new} = 0.05 + 0.060$$

$$= 0.110$$

3개 방법론 비교

퍼셉트론	Output	$\hat{y} = a = h(z) = h(-0.281) = -1$
	오차(error)	$(y - \hat{y}) = (1 - (-1)) = 2$
	가중치 변화율	$\Delta w_j = \eta(y - \hat{y})x_j = 0.2\big(1 - (-1)\big)0.2 = 0.4$
아달라인	Output	$\hat{y} = a = h(z) = z = h(-0.281) = -0.281$
	오차(error)	$(y - \hat{y}) = (1 - (-0.281)) = 1.281$
	가중치 변화율	$\Delta w_j = \eta \sum_{i=1}^{n} (y_i - \hat{y}_i)\, x_j = 0.2\{(1 - (-0.281)) \times 0.2 + \cdots\} = 0.159$
로지스틱	Output	$\hat{y} = a = h(z) = \frac{1}{1 + e^{-(-0.281)}} = 0.430$
	오차(error)	$(y - \hat{y}) = (1 - (0.430)) = 0.570$
	가중치 변화율	$\Delta w_j = \eta \sum_{i=1}^{n} (y_i - \hat{y}_i)\, x_j = 0.2\{(1 - (0.430)) \times 0.2 + \cdots\} = 0.159$

로지스틱 회귀 함수 만들기 실습

```
            cost = -y.dot(np.log(h)) - ((1 - y).dot(np.log(1 - h)))
            print('== 비용함수(J)')
            print('cost: ', cost)
            print('')

            self.cost_.append(cost)

        return self

    def net_input(self, X):
        """최종 입력 계산"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, z):
        """선형 활성화 계산"""
        ## np.clip(값, 최소값, 최대값): 최소, 최대값을 벗어나면 최소, 최대값으로 대체, 261 -> 250
        return 1. / (1. + np.exp(-np.clip(z, -250, 250)))

    def predict(self, X):
        """단위 계단 함수를 사용하여 클래스 레이블을 반환합니다"""
        return np.where(self.activation(self.net_input(X)) >= 0.0, 1, 0)
```

로지스틱 회귀 함수 만들기 실습

```
logistic = LogisticGD(n_iter = 1, eta = 0.2)
logistic.fit(X, y)
```

```
== 에포크
n_iter 0

== 초기 가중치(w)
self.w_:  [-0.3, 0.05, 0.01]

== 결합함수(z)
z:  [-0.281 -0.294 -0.286 -0.285 -0.275 -0.277]

== 활성함수(h)
h:  [0.43020863 0.42702488 0.42898341 0.42922839 0.43168002 0.43118942]

== 에러
errors:
0    0.569791
1   -0.427025
2   -0.428983
3   -0.429228
4    0.568320
5    0.568811
Name: Acceptance, dtype: float64

== 가중치 업데이트(X)
X.T.dot(errors):  Fat     0.297585
Salt    0.823111
dtype: float64
self.eta * X.T.dot(errors):  Fat     0.059517
Salt    0.164622
dtype: float64
self.w_[1:]:  [0.10951691857915814, 0.17462212832990065]

== 가중치 업데이트(b)
errors.sum():  0.42168524844191835
self.eta * errors.sum():  0.08433704968838368
self.w_[0]:  -0.2156629503116163

== 최종 가중치(전체)
self.w:  [-0.2156629503116163, 0.10951691857915814, 0.17462212832990065]

== 비용함수(J)
cost:  4.202779610365861
```

2.로지스틱 회귀분석 실습

유니버설 은행

- 사례)유니버설 은행
  - target marketing을 활용한 캠페인
  - 개인대출 제안에 대한 수락여부
  - 총데이터: 5,000개
  - 성공율: 9.6%(480명)
  - 분리: 3,000개(학습), 2,000개(검증)

TABLE 2.2 SAMPLE OF BANK CUSTOMERS OF UNIVERSAL BANK

ID	Age (나이)	Professional Experience (경력)	Income (소득)	Family Size (가족수)	CC Avg (월별 신용카드 평균 사용액)	Education (교육)	Mortgage (담보부채권)	Loan (개인대출)	Account (증권계좌)	CD Account (CD계좌)	Online Banking (온라인뱅킹)	Credit Card (신용카드)
1	25	1	49	4	1.60	UG	0	No	Yes	No	No	No
2	45	19	34	3	1.50	UG	0	No	Yes	No	No	No
3	39	15	11	1	1.00	UG	0	No	No	No	No	No
4	35	9	100	1	2.70	Grad	0	No	No	No	No	No
5	35	8	45	4	1.00	Grad	0	No	No	No	No	Yes
6	37	13	29	4	0.40	Grad	155	No	No	No	Yes	No
7	53							No	No	No	Yes	No
8	50							No	No	No	No	Yes
9	35							No	No	No	Yes	No
10	34							Yes	No	No	No	No
11	65	39	105	4	2.40	Prof	0	No	No	No	No	No
12	29	5	45	3	0.10	Grad	0	No	No	No	Yes	No
13	48	23	114	2	3.80	Prof	0	No	Yes	No	No	No
14	59	32	40	4	2.50	Grad	0	No	No	No	Yes	No

의사결정나무 모델 copy

1.기본 package 설정

```
## 5.분류모델구축 (3장.p.83~130)
# from sklearn.tree import DecisionTreeClassifier # 결정 트리
# from sklearn.naive_bayes import GaussianNB # 나이브 베이즈
# from sklearn.neighbors import KNeighborsClassifier # K-최근접 이웃
# from sklearn.ensemble import RandomForestClassifier # 랜덤 포레스트
# from sklearn.ensemble import BaggingClassifier # 앙상블
# from sklearn.linear_model import Perceptron # 퍼셉트론
from sklearn.linear_model import LogisticRegression # 로지스틱 회귀 모델
# from sklearn.svm import SVC # 서포트 벡터 머신(SVM)
# from sklearn.neural_network import MLPClassifier # 다층인공신경망
```

3.데이터 전처리

3.데이터 전처리

- 문자형 자료를 숫자(범주형)로 인코딩 -> 범주형 변수를 가변수로 처리 : One Hot Encording
- 숫자형 자료를 표준화
- 단, 결정나무, 랜덤 포레스트, 나이브 베이즈 분류 : 원본데이터 그대로 유지

3.1 data(X) 수치형 데이터 표준화

- X.keys()에서 index 키를 가져옴 ['Age', 'Experience', 'Income', 'Family', 'CCAvg']

```
stdsc = StandardScaler()
X.iloc[:,[0,1,2,3,4,6]]= stdsc.fit_transform(X.iloc[:,[0,1,2,3,4,6]])
```

```
X.head()
```

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	SecuritiesAccount	CDAccount	Online	CreditCard
0	-1.774417	-1.666078	-0.538229	1.397414	-0.193385	1	-0.555524	1	0	0	0
1	-0.029524	-0.096330	-0.864109	0.525991	-0.250611	1	-0.555524	1	0	0	0
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.536736	1	-0.555524	0	0	0	0
3	-0.901970	-0.968413	0.569765	-1.216855	0.436091	2	-0.555524	0	0	0	0
4	-0.901970	-1.055621	-0.625130	1.397414	-0.536736	2	-0.555524	0	0	0	1

3.데이터 전처리

3.2 data(X) 레이블 인코딩

- 질변변수 가변수화
- 가변수 처리시 문자로 처리를 해야 변수명 구분이 쉬움

```
X['Education'] = X['Education'].replace ([1,2,3], ['Under','Grad','Prof'])
```

```
X.head()
```

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	SecuritiesAccount	CDAccount	Online	CreditCard
0	-1.774417	-1.666078	-0.538229	1.397414	-0.193385	Under	-0.555524	1	0	0	0
1	-0.029524	-0.096330	-0.864109	0.525991	-0.250611	Under	-0.555524	1	0	0	0
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.536736	Under	-0.555524	0	0	0	0
3	-0.901970	-0.968413	0.569765	-1.216855	0.436091	Grad	-0.555524	0	0	0	0
4	-0.901970	-1.055621	-0.625130	1.397414	-0.536736	Grad	-0.555524	0	0	0	1

```
X.keys()
```

```
Index(['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
       'Mortgage', 'SecuritiesAccount', 'CDAccount', 'Online', 'CreditCard'],
      dtype='object')
```

```
X = pd.get_dummies(X[['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
                      'Mortgage', 'SecuritiesAccount', 'CDAccount', 'Online', 'CreditCard']],
                   columns=['Education'],
                   drop_first = True)
```

```
X.head()
```

	Age	Experience	Income	Family	CCAvg	Mortgage	SecuritiesAccount	CDAccount	Online	CreditCard	Education_Prof	Education_Under
0	-1.774417	-1.666078	-0.538229	1.397414	-0.193385	-0.555524	1	0	0	0	0	1
1	-0.029524	-0.096330	-0.864109	0.525991	-0.250611	-0.555524	1	0	0	0	0	1
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.536736	-0.555524	0	0	0	0	0	1
3	-0.901970	-0.968413	0.569765	-1.216855	0.436091	-0.555524	0	0	0	0	0	0
4	-0.901970	-1.055621	-0.625130	1.397414	-0.536736	-0.555524	0	0	0	1	0	0

5.모델구축

```
logistic = LogisticRegression(solver='liblinear',
                              penalty='l2',
                              C=0.001,
                              random_state=1)

logistic.fit(X_train, y_train)
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=1, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

6.모델검정

6.2 정오분류표로 검정

```
confmat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['True[0]','True[1]'],
                       columns=['Predict[0]', 'Predict[1]'])
confmat
```

	Predict[0]	Predict[1]
True[0]	1349	7
True[1]	117	27

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.99      0.96      1356
           1       0.79      0.19      0.30       144

    accuracy                           0.92      1500
   macro avg       0.86      0.59      0.63      1500
weighted avg       0.91      0.92      0.89      1500
```

6.3 정확도, 민감도 확인

- 클래스가 2개일 경우에만 실행

```
print('잘못 분류된 샘플 개수: %d' % (y_test != y_pred).sum())
print('정확도: %.3f' % accuracy_score(y_test, y_pred))
print('정밀도: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('재현율: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
잘못 분류된 샘플 개수: 124
정확도: 0.917
정밀도: 0.794
재현율: 0.188
F1: 0.303
```

부록.Logistic Regress 회귀계수

부록. Logistic Regress 회귀계수

- 전체변수 투입: sm.Logit(y, X)
- 특정변수만 넣고 싶을 때
- Logistic_ml = sm.Logit.from_formula("PersonalLoan ~ Age + Experience + Income + Family + CCAvg", bank_df)

```
import statsmodels.api as sm
logistic_ml = sm.Logit(y, X) #로지스틱 회귀분석 시행
logistic_coef = logistic_ml.fit()
logistic_coef.summary2()
```

부록.Logistic Regress 회귀계수

Model:	Logit	Pseudo R-squared:	0.528
Dependent Variable:	PersonalLoan	AIC:	1516.1369
Date:	2019-11-04 14:12	BIC:	1594.3432
No. Observations:	5000	Log-Likelihood:	-746.07
Df Model:	11	LL-Null:	-1581.0
Df Residuals:	4988	LLR p-value:	0.0000
Converged:	1.0000	Scale:	1.0000
No. Iterations:	9.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Age	-0.7512	0.6853	-1.0960	0.2731	-2.0944	0.5921
Experience	0.8612	0.6827	1.2615	0.2071	-0.4768	2.1992
Income	2.4375	0.1145	21.2843	0.0000	2.2130	2.6620
Family	0.3616	0.0714	5.0629	0.0000	0.2216	0.5016
CCAvg	0.3730	0.0748	4.9860	0.0000	0.2264	0.5196
Mortgage	0.0915	0.0576	1.5881	0.1123	-0.0214	0.2045
SecuritiesAccount	-1.7463	0.2888	-6.0472	0.0000	-2.3123	-1.1803
CDAccount	4.9464	0.3341	14.8030	0.0000	4.2915	5.6013
Online	-2.1804	0.1350	-16.1499	0.0000	-2.4450	-1.9158
CreditCard	-2.1050	0.2029	-10.3758	0.0000	-2.5026	-1.7074
Education_Prof	-1.6077	0.1422	-11.3079	0.0000	-1.8864	-1.3290
Education_Under	-5.5799	0.2466	-22.6258	0.0000	-6.0633	-5.0965

부록.Logistic Regress 회귀계수

```
np.exp(logistic_coef.params)
```

```
Age                    0.471820
Experience             2.366013
Income                11.444403
Family                 1.435643
CCAvg                  1.452095
Mortgage               1.095844
SecuritiesAccount      0.174414
CDAccount            140.667639
Online                 0.112999
CreditCard             0.121845
Education_Prof         0.200348
Education_Under        0.003773
dtype: float64
```

7.최적화

7.1 파이프라인 모델 만들기

- 파이프라인을 이용하여 최적 모델 만들기
- 기본모형은 아무 옵션이 없는 모델로 부터 시작
- 파라미터 옵션 확인: pipe_tree.get_params().keys()

```
pipe_logistic = make_pipeline(LogisticRegression(random_state=1))
```

```
pipe_logistic.get_params().keys()
```

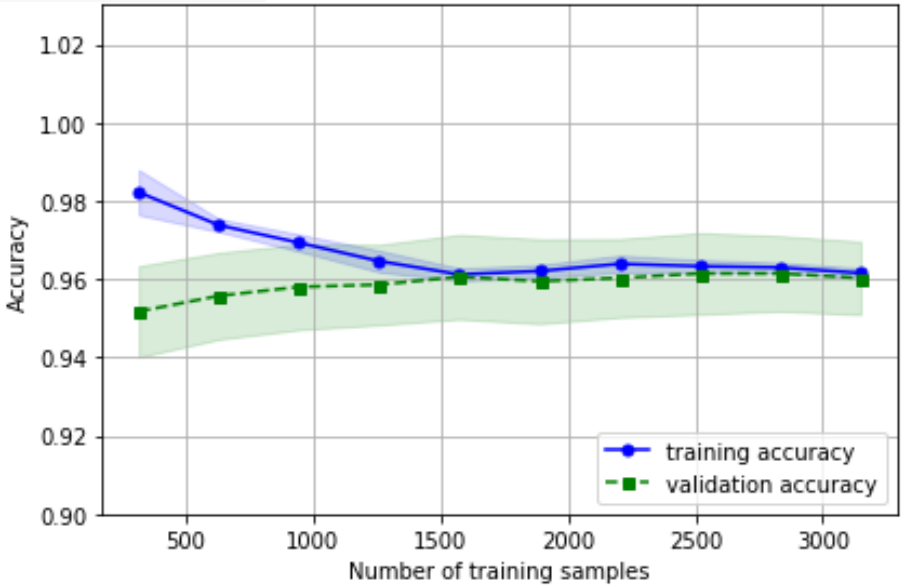
```
dict_keys(['memory', 'steps', 'verbose', 'logisticregression', 'logisticregress
sion__dual', 'logisticregression__fit_intercept', 'logisticregression__intercep
ion__max_iter', 'logisticregression__multi_class', 'logisticregression__n_jobs
_state', 'logisticregression__solver', 'logisticregression__tol', 'logisticregr
```

7.최적화

7.2 학습 곡선으로 편향과 분산 문제 분석하기

- 훈련 샘플링 수를 이용하여 편향과 분산 검정
- 편향: 정확도가 높은지 검정
- 분산: 훈련/검정 데이터의 정확도의 차이가 적은지

```
train_sizes, train_scores, test_scores =₩
                learning_curve(estimator=pipe_logistic, # 수정
                               X=X_train,
                               y=y_train,
                               train_sizes=np.linspace(0.1, 1.0, 10),
                               cv=10,
                               n_jobs=1)
```



7.최적화

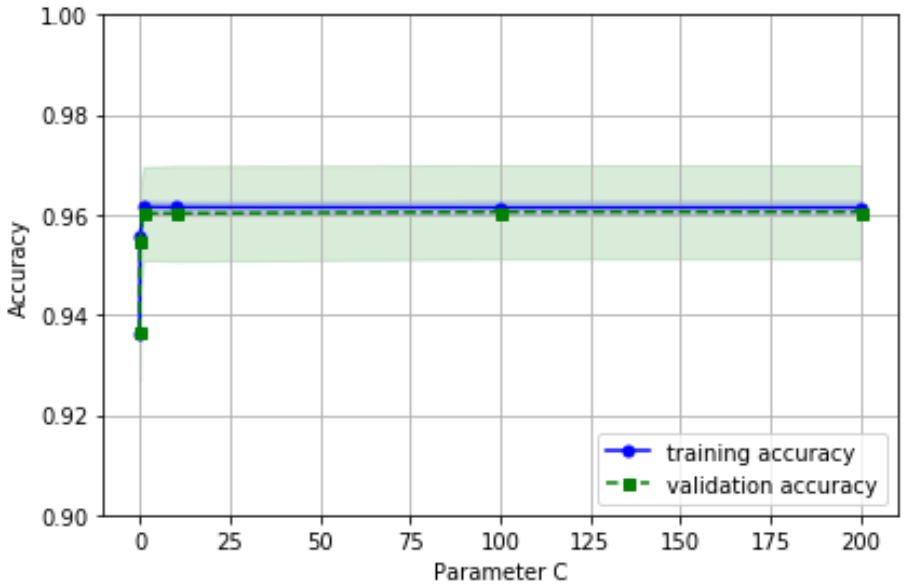
7.3 검증 곡선으로 과대적합과 과소적합 조사하기

- 과대적합 : 파라미터가 많음 -> 파라미터 축소
- 과소적합 : 파라미터가 적음 -> 파라미터 추가

```
param_range = [0.01, 0.1, 1.0, 10, 100, 200]  # 수정

train_scores, test_scores = validation_curve(
                estimator=pipe_logistic, # 수정
                X=X_train,
                y=y_train,
                param_name='logisticregression__C', ## 수정
                param_range=param_range,
                cv=10)
```

```
plt.grid()
plt.xlabel('Number of C') # 수정
plt.legend(loc='lower right')
plt.xlabel('Parameter C') # 수정
plt.ylabel('Accuracy')
plt.ylim([0.9, 1.00])  # 수정
plt.tight_layout()
plt.show()
```



7.최적화

7.4 하이퍼파라미터 튜닝

- 그리드 서치를 사용한 머신 러닝 모델 세부 튜닝
- 기계학습 모델의 성능을 결정하는 하이퍼 파라미터 튜닝

```
param_range = [0.01, 0.1, 1.0, 10, 100, 200]  # 수정

param_grid = [{'logisticregression__C': param_range}] # 수정

gs = GridSearchCV(estimator=pipe_logistic, # 수정
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)

gs = gs.fit(X_train, y_train)

print(gs.best_score_)
print(gs.best_params_)
```

```
0.9605714285714285
{'logisticregression__C': 100}
```

8.최적화 모델 검정

8.최적화 모델 검정

- 최적모델을 이용해 검정 데이터(full data) 최종 확인
- best_tree 로 모델명 변경

```
best_logistic = gs.best_estimator_
best_logistic.fit(X_train, y_train)
```

```
C:₩Users₩leecho₩Anaconda3₩lib₩site-packages₩sklearn₩linear_model₩logistic.py:432: FutureWarn
in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Pipeline(memory=None,
         steps=[('logisticregression',
                 LogisticRegression(C=100, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='warn', n_jobs=None,
                                    penalty='l2', random_state=1, solver='warn',
                                    tol=0.0001, verbose=0, warm_start=False))],
         verbose=False)
```

8.최적화 모델 검정

8.최적화 모델 검정

- 최적모델을 이용해 검정 데이터(full data) 최종 확인
- best_tree 로 모델명 변경

```
best_logistic = gs.best_estimator_
best_logistic.fit(X_train, y_train)
```

```
C:₩Users₩leecho₩Anaconda3₩lib₩site-packages₩sklearn₩linear_model₩logistic.py:432: Futur
in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Pipeline(memory=None,
         steps=[('logisticregression',
                 LogisticRegression(C=100, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='warn', n_jobs=None,
                                    penalty='l2', random_state=1, solver='warn',
                                    tol=0.0001, verbose=0, warm_start=False))],
         verbose=False)
```

- 검증용 데이터로 예측

```
y_pred = best_logistic.predict(X_test)
```

8.최적화 모델 검정

- 정오분류표로 검정

수정모델

```
confmat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['True[0]','True[1]'],
                       columns=['Predict[0]', 'Predict[1]'])
confmat
```

	Predict[0]	Predict[1]
True[0]	1341	15
True[1]	50	94

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1356
           1       0.86      0.65      0.74       144

    accuracy                           0.96      1500
   macro avg       0.91      0.82      0.86      1500
weighted avg       0.95      0.96      0.95      1500
```

초기모델

6.2 정오분류표로 검정

```
confmat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['True[0]','True[1]'],
                       columns=['Predict[0]', 'Predict[1]'])
confmat
```

	Predict[0]	Predict[1]
True[0]	1349	7
True[1]	117	27

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.99      0.96      1356
           1       0.79      0.19      0.30       144

    accuracy                           0.92      1500
   macro avg       0.86      0.59      0.63      1500
weighted avg       0.91      0.92      0.89      1500
```

8.최적화 모델 검정

- 정확도, 민감도 확인
- 초기 모델
- 정확도: 0.917
- 정밀도: 0.794
- 재현율: 0.188
- F1: 0.303

```
print('잘못 분류된 샘플 개수: %d' % (y_test != y_p
print('정확도: %.3f' % accuracy_score(y_test, y_p
print('정밀도: %.3f' % precision_score(y_true=y_t
print('재현율: %.3f' % recall_score(y_true=y_test
print('F1: %.3f' % f1_score(y_true=y_test, y_pred
```

```
잘못 분류된 샘플 개수: 65
정확도: 0.957
정밀도: 0.862
재현율: 0.653
F1: 0.743
```

6.3 정확도, 민감도 확인

- 클래스가 2개일 경우에만 실행

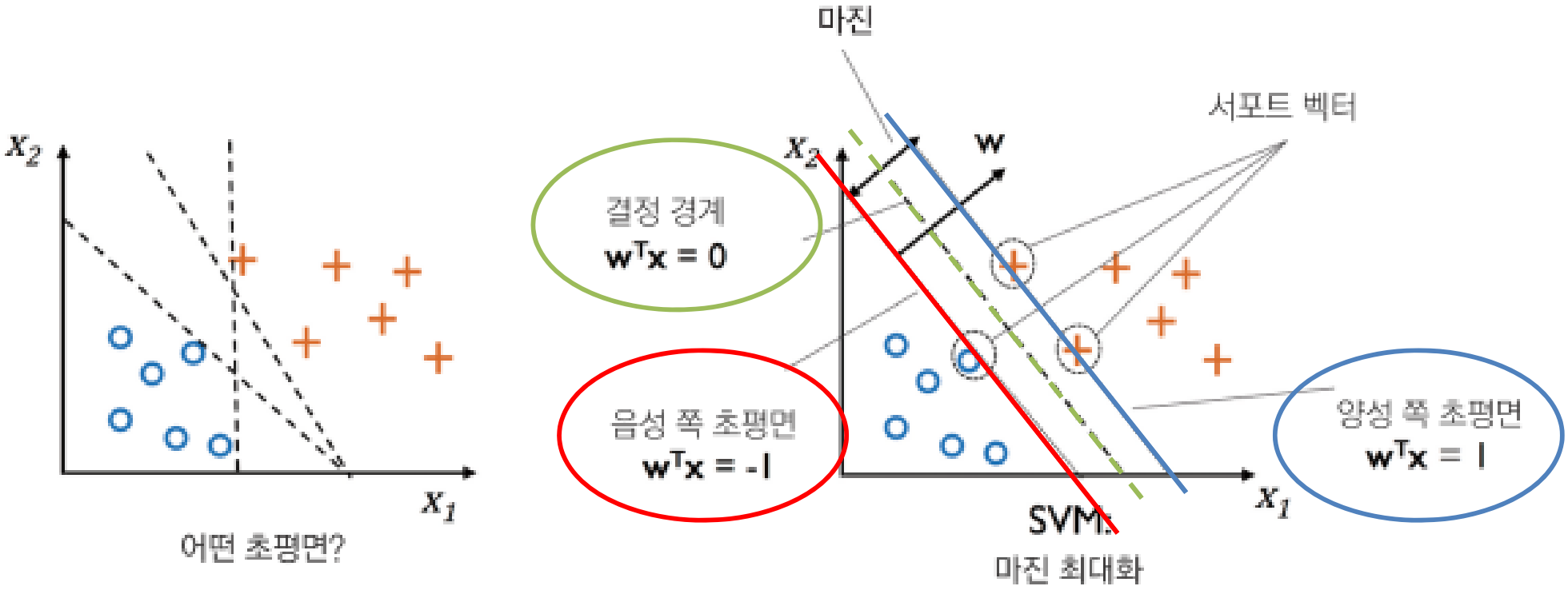
```
print('잘못 분류된 샘플 개수: %d' % (
print('정확도: %.3f' % accuracy_score
print('정밀도: %.3f' % precision_scor
print('재현율: %.3f' % recall_score(y
print('F1: %.3f' % f1_score(y_true=y_
```

```
잘못 분류된 샘플 개수: 124
정확도: 0.917
정밀도: 0.794
재현율: 0.188
F1: 0.303
```

3.서포트 벡터 머신(SVM)

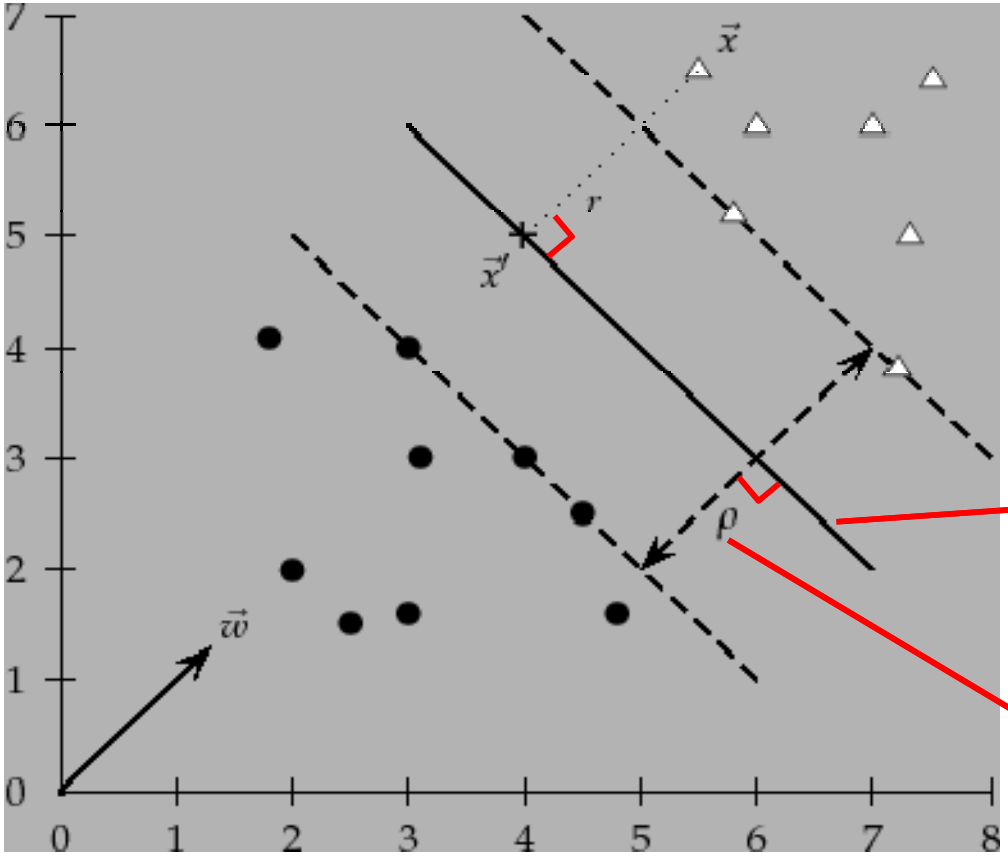
서포트 벡터 머신(SVM)

- 서포트 벡터 머신(Support Vector Machine)
  - 퍼셉트론의 확장
  - 퍼셉트론 학습: 분류오차의 최소화($y - \hat{y}$)
  - SVM 학습: 마진(초평면=결정경계)의 최대화
  - 서포트 벡터



서포트 벡터 머신(SVM)

- 비용함수
  - 마진(초평면=결정경계)의 최대화
  - 마진 = 서포트 벡터와 직교하는 직선(w)과의 거리



$$w_1 x_1 + w_2 x_2 + b = 0$$

$$\rho = \frac{2}{\|w\|}$$

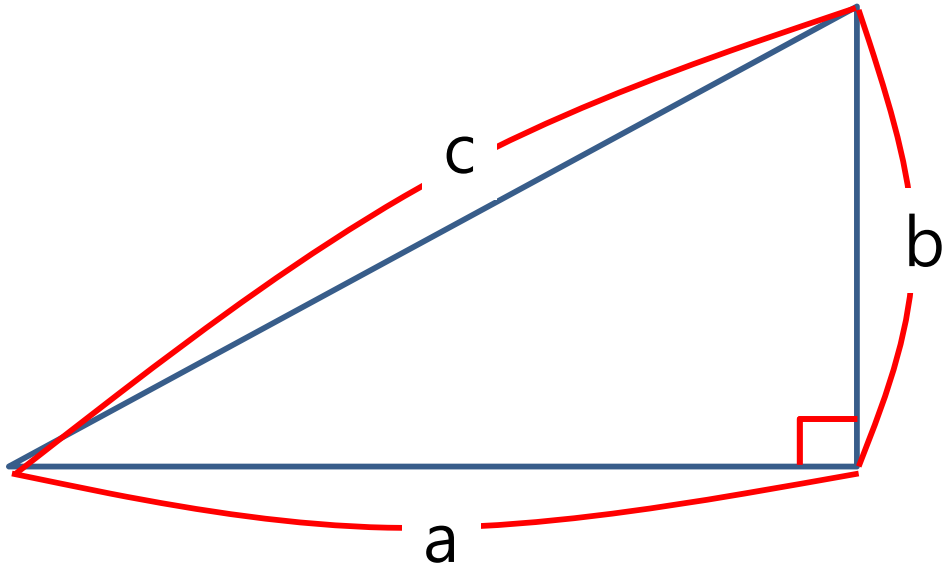
https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html

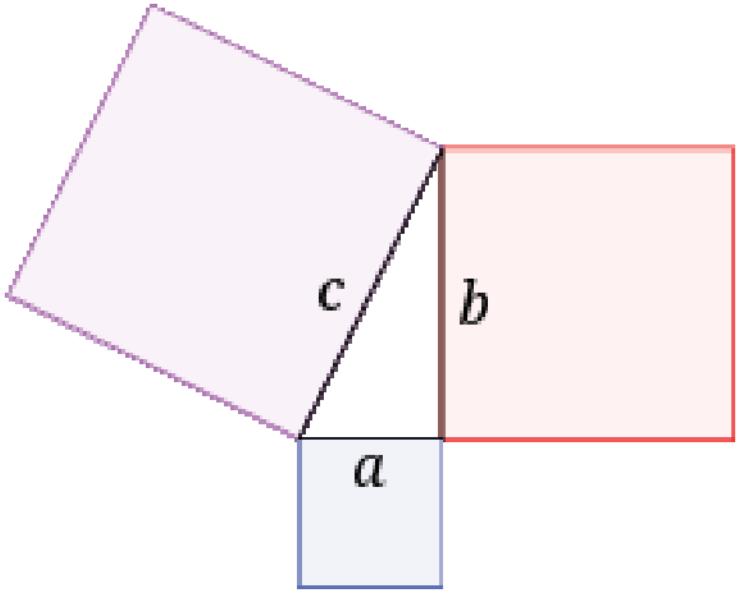
수학적 지식: 선형대수

- 피타고라스의 정리
  - 직각 삼각형의 두 직각변 a, b를 각각 한 변으로 하는 정사각형 면적의 합은 빗변 c를 한 변으로 하는 정사각형의 면적과 같음

$$c^2 = a^2 + b^2$$

$$c = \sqrt{a^2 + b^2} \qquad c = \sqrt{2^2 + 1^2} = \sqrt{4 + 1} = \sqrt{5}$$





https://ko.wikipedia.org/wiki/피타고라스의 정리

수학적 지식: 선형대수

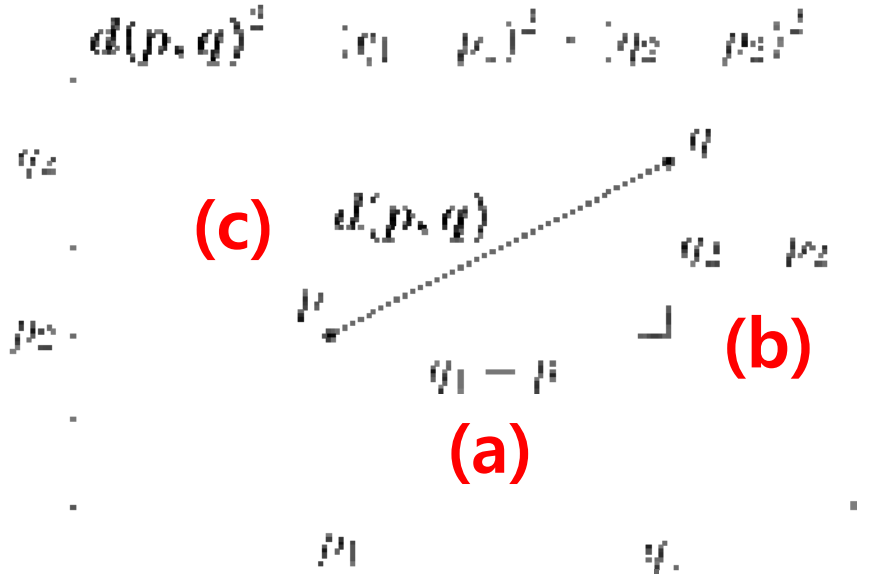
- 유클리드 거리(Euclidean distance)
  - 공간(좌표)에서 두 점 사이의 거리를 계산
  - 파타고라스 정리 활용

$$a = (q_1 - p_1)$$

$$b = (q_2 - p_2)$$

$$c = \sqrt{a^2 + b^2}$$

$$c = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$



https://ko.wikipedia.org/wiki/피타고라스의 정리

수학적 지식: 선형대수

- 유클리드 거리(Euclidean distance)
  - 두 점 사이의 거리공간에서 두 점 사이의 거리를 계산

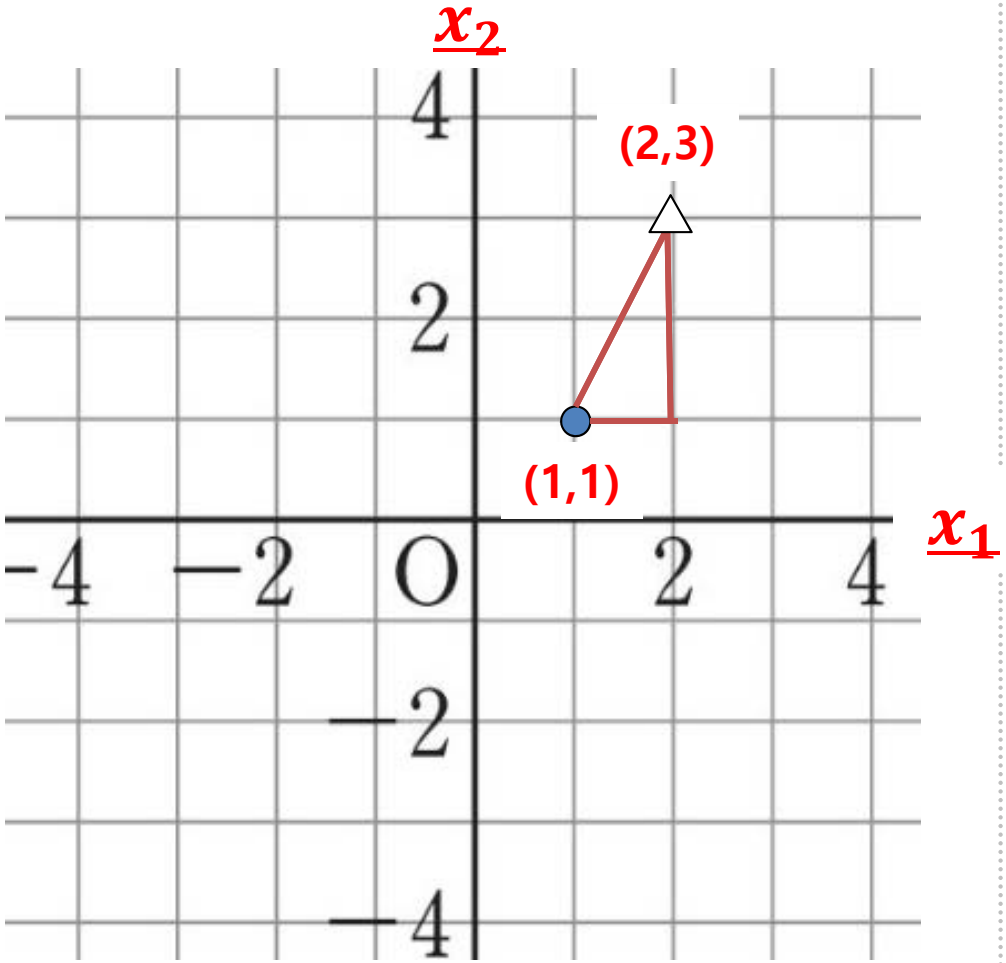
$p = (1,1) \qquad q = (2,3)$

$a = (2 - 1) = 1$

$b = (3 - 1) = 2$

$c = \sqrt{a^2 + b^2}$

$d = c = \sqrt{(1)^2 + (2)^2} = \sqrt{5}$



https://ko.wikipedia.org/wiki/피타고라스의 정리

수학적 지식: 선형대수

- 노름(norm, length, magnitude)
  - 벡터공간의 벡터들에 대한 '길이' 혹은 '크기'를 부여하는 함수
  - 원점에서 점 $\mathrm{P}(x_1, x_2, \ldots, x_i)$에 이르는 거리

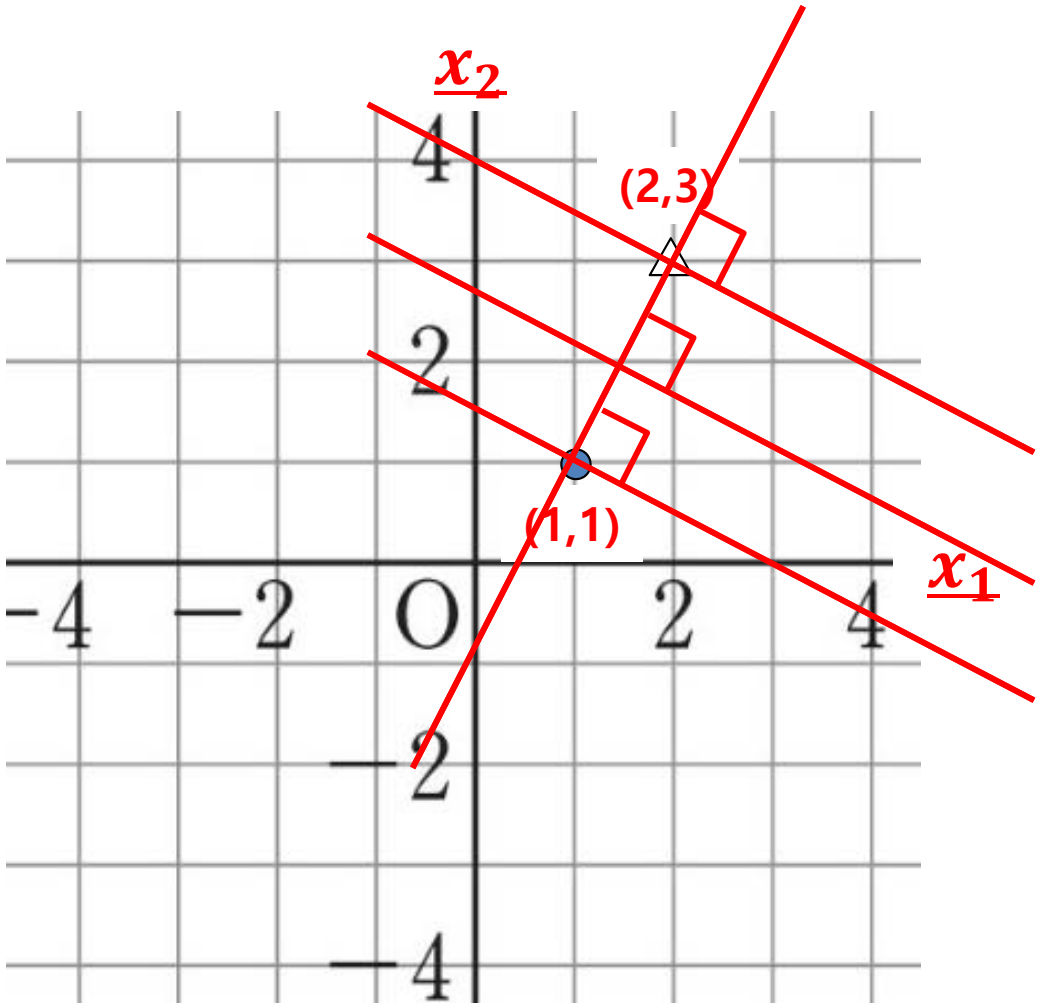
$$\|x\| = \sqrt{(x_1)^2+(x_2)^2+\cdots+(x_i)^2}$$

- 유클리디안 노름
  - 두 점에 이르는 거리

$$\|q-p\| = \sqrt{(q_1-p_1)^{\ 2}+(q_2-p_2)^2+\cdots}$$

서포트 벡터 머신(SVM)

- 초평면을 구하는 방법
  - 두 점 $p = (1,1)$, $q = (2,3)$ 를 지나는 선(벡터(w))를 이용
  - 두 점을 지나면서 직교(직각)인 선(초평면)을 구함
  - 두 점의 중앙을 지나면서 직교(직각)인 선(초평면)을 구함

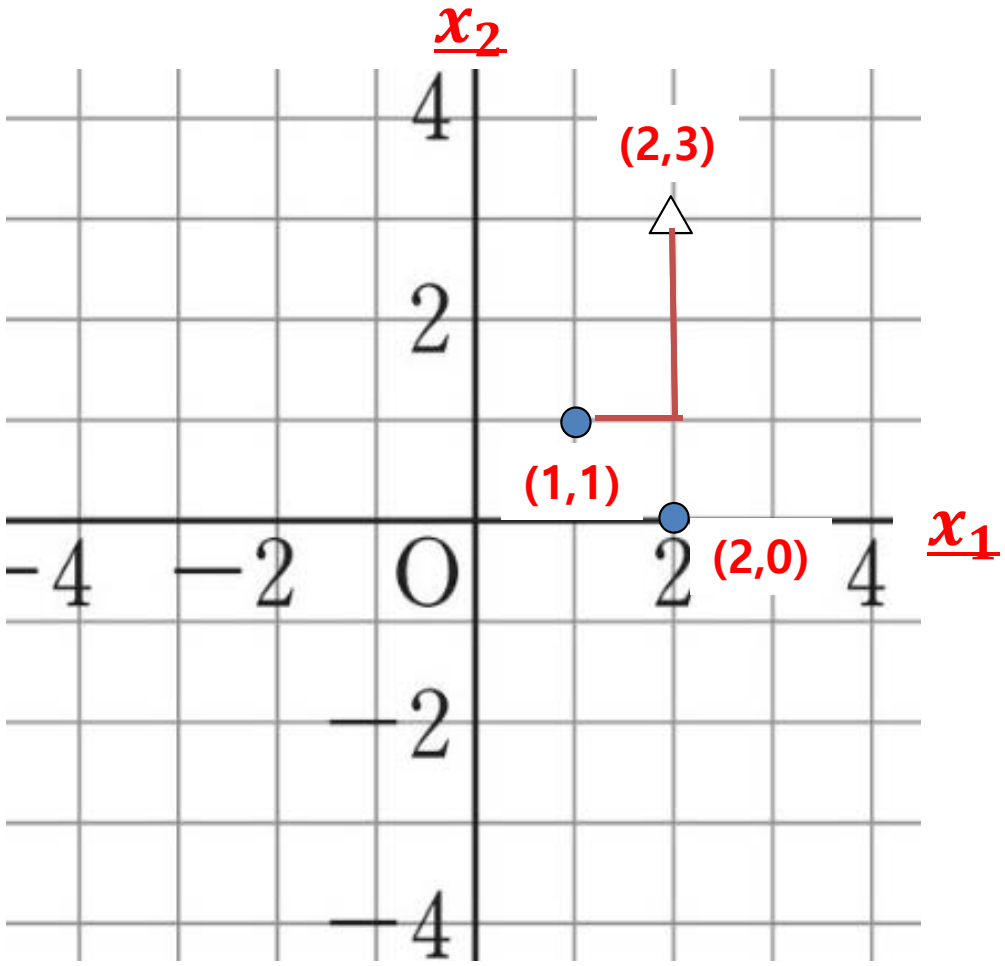


https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html

서포트 벡터 머신(SVM)

- 예제)
  - +1 : (2,3)
  - -1 : (1,1), (2,0)
  - +1과 -1사이의 가장 가까운 두 점 찾기
    - +1 : (2,3)
    - -1 : (1,1)
  - 두 점의 중앙을 지나면서 직교(직각)인 선(초평면)을 구함

$$w = (a, 2a)$$



서포트 벡터 머신(SVM)

- 두 점을 지나면서 직교(직각)인 선(초평면)을 구함

$w_1x_1 + w_2x_2 + b = 1$

$w_1x_1 + w_2x_2 + b = -1$

- $+1 : (2,3)$

$ax_1 + 2ax_2 + b = -1$

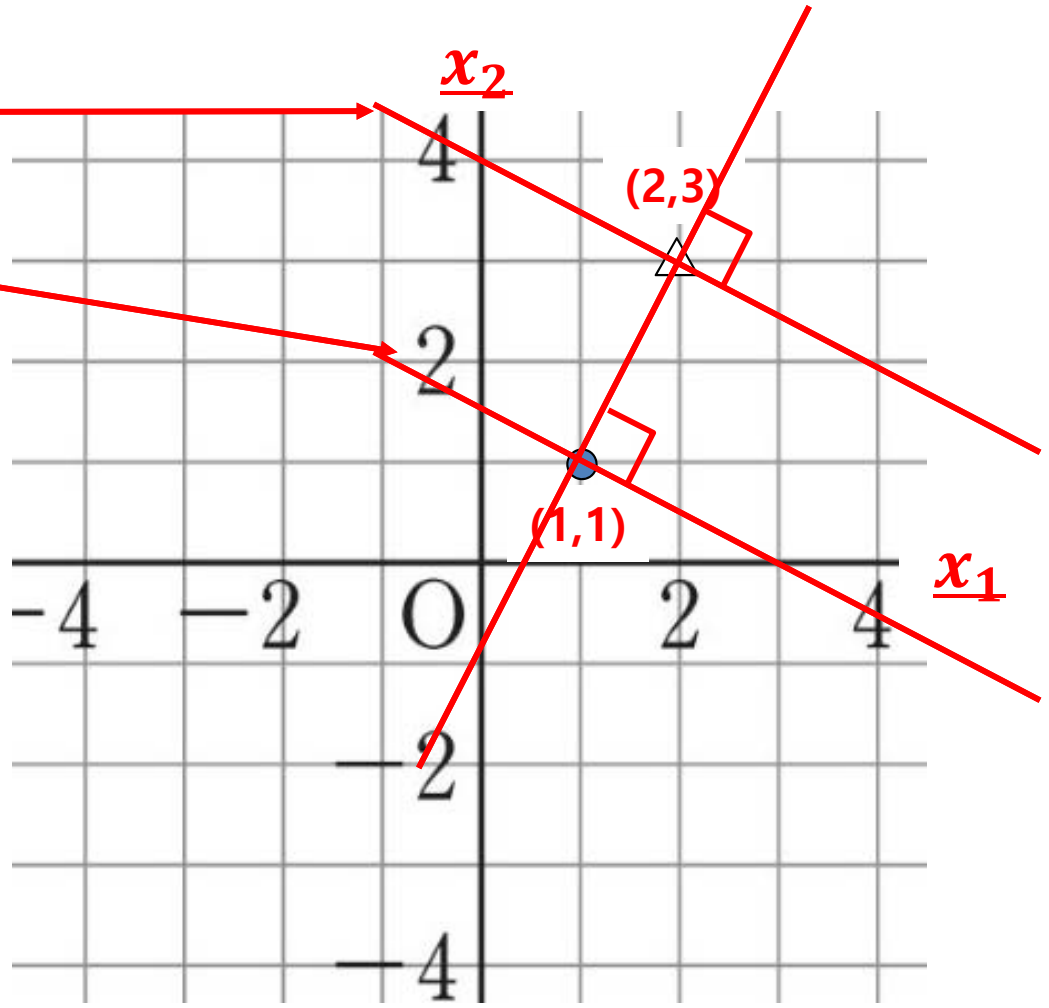
$1a + 2a + b = -1$

- $-1 : (1,1)$

$ax_1 + 2ax_2 + b = 1$

$2a + 6(2 * 3)a + b = 1$

$a = \frac{2}{5}, b = -\frac{11}{5}$



서포트 벡터 머신(SVM)

- 두 점의 중앙을 지나면서 직교(직각)인 선(초평면)을 구함

$$a = \frac{2}{5}, b = -\frac{11}{5}$$

$$w = (a, 2a) = (\frac{2}{5}, \frac{4}{5})$$

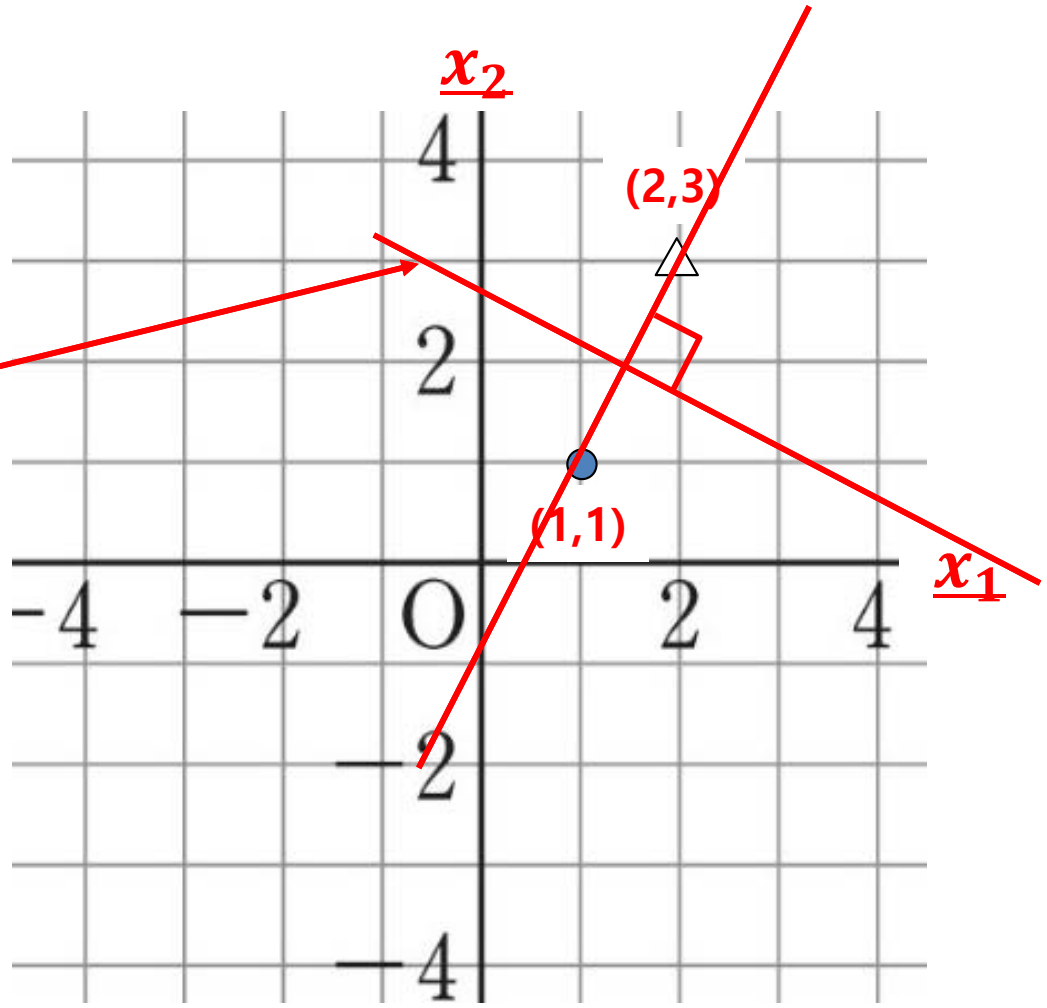
$$w_1 x_1 + w_2 x_2 + b = 0$$

$$\frac{2}{5} x_1 + \frac{2}{5} x_2 - \frac{11}{5} = 0$$

$$x_1 + 2x_2 - 5.5 = 0$$

- margin

$$\frac{2}{\|w\|} = \sqrt{\left(\frac{2}{5}\right)^2 + \left(\frac{4}{5}\right)^2} = \sqrt{\frac{20}{25}} = \sqrt{5}$$



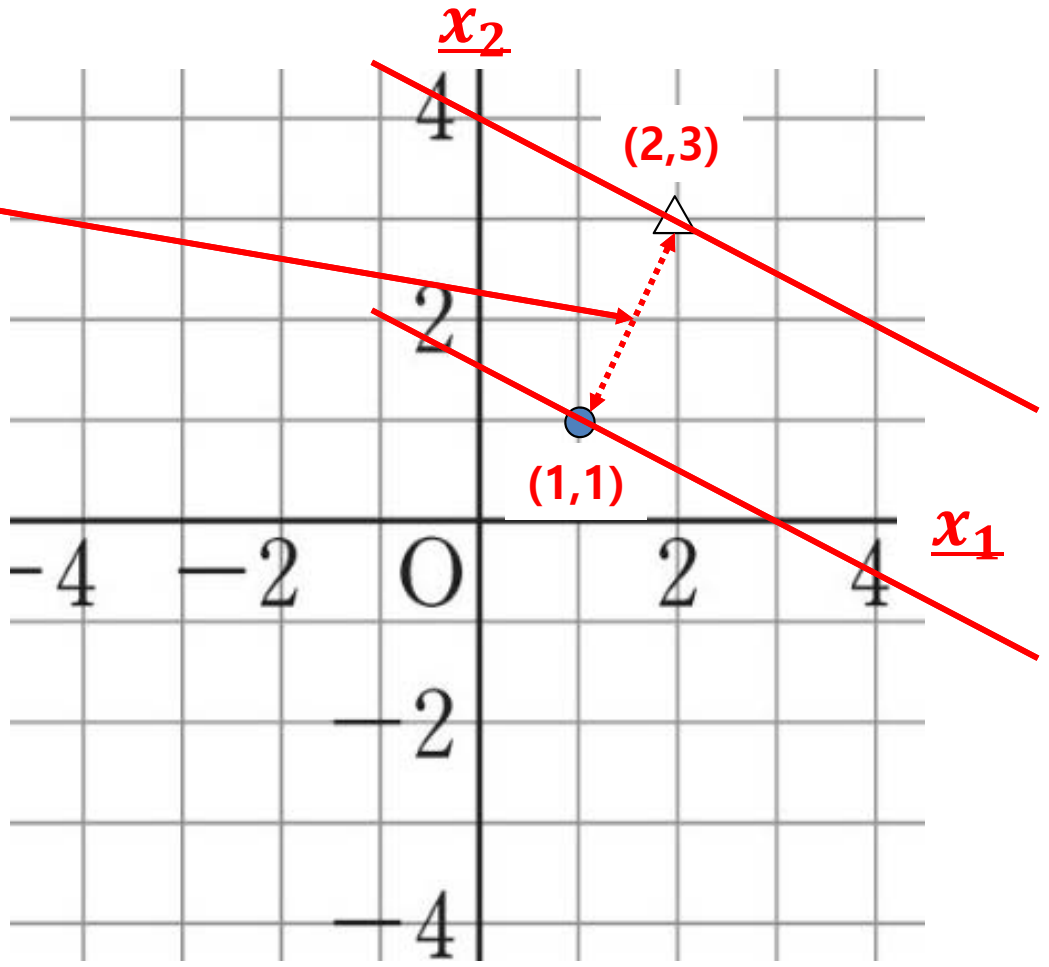
서포트 벡터 머신(SVM)

- margin

$$\rho = \frac{2}{\|w\|}$$

$$\|w\| = \sqrt{\left(\frac{2}{5}\right)^2 + \left(\frac{4}{5}\right)^2} = \sqrt{\frac{20}{25}} = \sqrt{5}$$

$$\frac{2}{\|w\|} = \frac{2}{\sqrt{\frac{20}{25}}} = \frac{2}{2\frac{1}{\sqrt{5}}} = \sqrt{5}$$

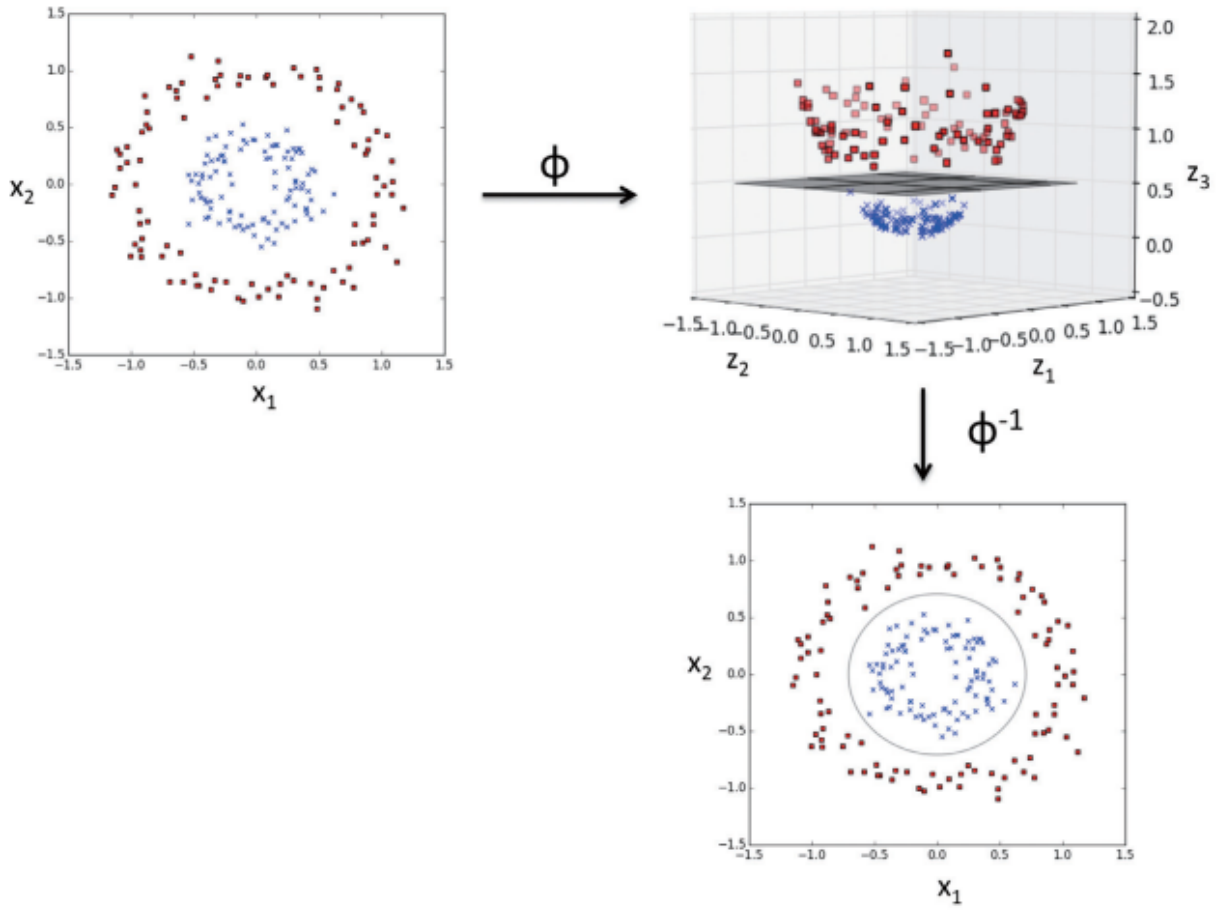


서포트 벡터 머신(SVM)

- SVM을 이용한 비선형 문제
  - 직선으로 구분이 안될 경우
  - 커널을 이용

- SVM Kernel
  - rbf(radial basis function)
  - Gaussian kernel
  - Polynomial kernel

그림 3-13 고차원 공간에서 찾은 결정 경계의 예



서포트 벡터 머신(SVM)

- 규제
  - L2규제: 개별 가중치 값을 제한하여 모델 복잡도 축소
  - 규제증가(C(↑)) → 가중치감소(w(↓)) → 훈련데이터의 의존성을 축소

그림 3-10 C 값에 따라 달라지는 SVM의 결정 경계와 마진

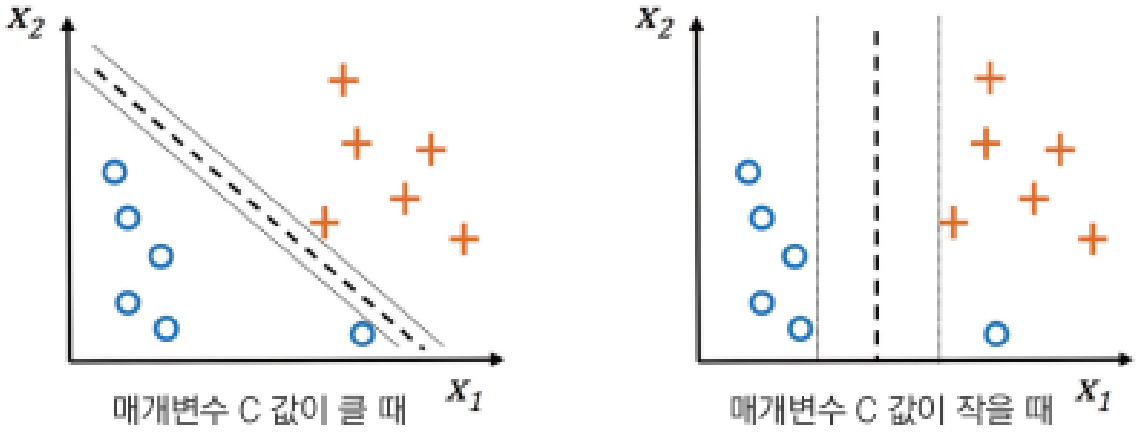
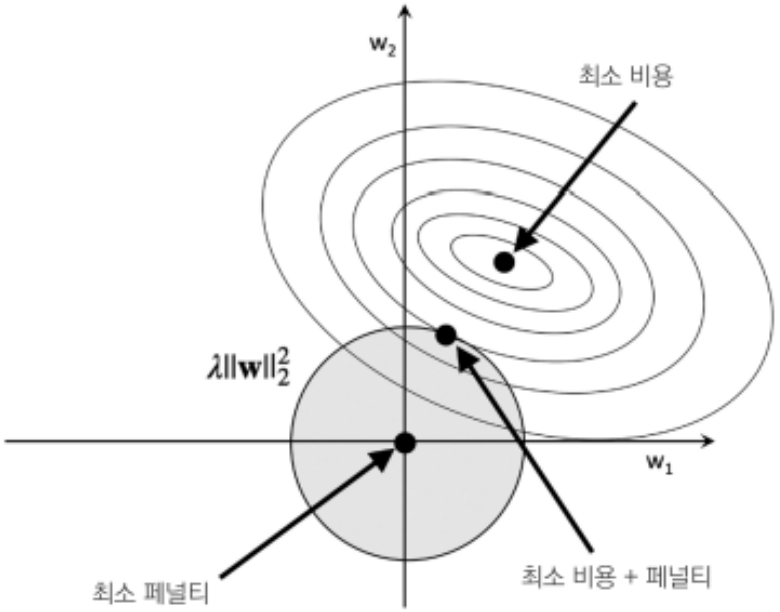


그림 4-5 L2 규제와 비용 함수

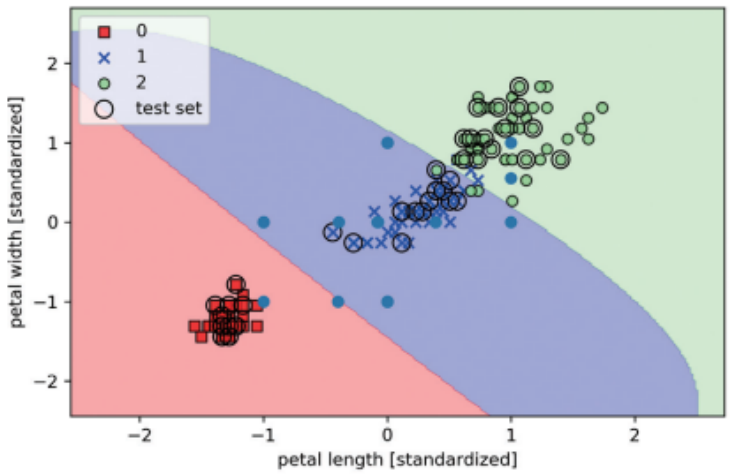


서포트 벡터 머신(SVM)

- Python SVM 옵션
  - Kernel : rbf(radial basis function), linear, poly(polynomial)
  - C : 규제 함수값 제한
  - gamma : 가우시안 구(γ)의 크기 제한
    - $\gamma(\uparrow)$ → 서포트 벡터의 영향(↓)
    - $\gamma(\uparrow)$ → 경계가 샘플에 가까워짐

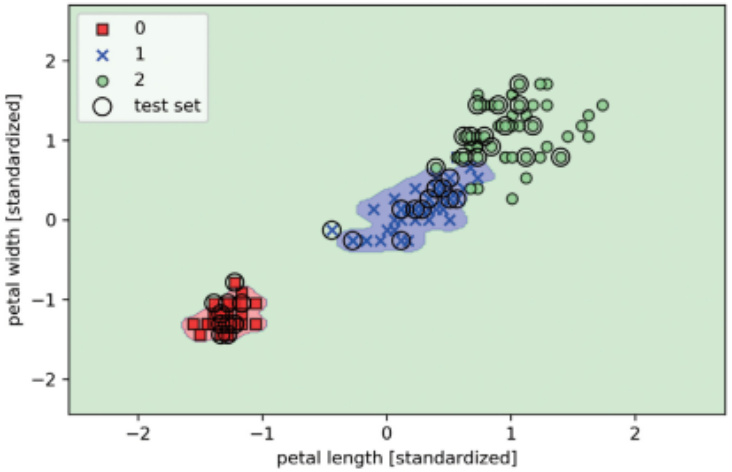
$\gamma(\downarrow)$

그림 3-15 RBF 커널 SVM 모델이 학습한 붓꽃 데이터셋의 결정 경계



$\gamma(\uparrow)$

그림 3-16 감마 매개변수를 크게 한 RBF 커널 SVM의 결정 경계



https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html

4.SVM 실습

예제 데이터

- 사례) 자동차 사고

변수명	설명
ALCHL_I	음주 여부: 1=있음, 2=없음
PROFIL_I_R	도로 정보: 0=기타, 1=level1
SUR_CONM	도로의 노면상태: 1=건조, 2=젖음, 3=눈/진흙, 9=모름
VEH_INVL	관련된 차량의 수
MAX_SEV_IR	상해/치명 여부: 0=무상해, 1=상해, 2=치명

TABLE 11.4 SUBSET FROM THE ACCIDENTS DATA, FOR A HIGH-FATALITY REGION

Obs.	ALCHL_I	PROFIL_I_R	SUR_COND	VEH_INVL	MAX_SEV_IR
1	1	1	1	1	1
2	2	1	1	1	0
3	2	1	1	1	1
4	1	1	1	1	0
5	2	1	1	1	2
6	2	0	1	1	1
7	2	0	1	3	1
8	2	0	1	4	1
9	2	0	1	2	0
10	2	0	1	2	0

1.기본 package 설정

```
## 5.분류모델구축 (3장.p.83~130)
# from sklearn.tree import DecisionTreeClassifier # 결정 트리
# from sklearn.naive_bayes import GaussianNB # 나이브 베이즈
# from sklearn.neighbors import KNeighborsClassifier # K-최근접 이웃
# from sklearn.ensemble import RandomForestClassifier # 랜덤 포레스트
# from sklearn.ensemble import BaggingClassifier # 앙상블
# from sklearn.linear_model import Perceptron # 퍼셉트론
# from sklearn.linear_model import LogisticRegression # 로지스틱 회귀 모델
from sklearn.svm import SVC # 서포트 벡터 머신(SVM)
# from sklearn.neural_network import MLPClassifier # 다층인공신경망
```

2.데이터 가져오기

2.1 데이터프레임으로 저장

- 원본데이터(csv)를 dataframe 형태로 가져오기(pandas)

```
accidents_df = pd.read_csv('accidentsnn.csv')
accidents_df.head()
```

	ALCHL_I	PROFIL_I_R	SUR_COND	VEH_INVL	MAX_SEV_IR
0	2	0	1	1	0
1	2	1	1	1	2
2	1	0	1	1	0
3	2	0	2	2	1
4	2	1	1	2	1

- 자료구조 살펴보기

```
accidents_df.shape
```

```
(999, 5)
```

```
# 자료구조 살펴보기
accidents_df.keys()
```

```
Index(['ALCHL_I', 'PROFIL_I_R', 'SUR_COND', 'VEH_INVL', 'MAX_SEV_IR'], dtype='object')
```

2.데이터 가져오기

2.2 data와 target으로 분리

- 필요한 데이터만 추출
- data: X, target: y 로 분리

```
X = accidents_df.drop (['MAX_SEV_IR'], axis=1)
X.head()
```

	ALCHL_I	PROFIL_I_R	SUR_COND	VEH_INVL
0	2	0	1	1
1	2	1	1	1
2	1	0	1	1
3	2	0	2	2
4	2	1	1	2

```
y = accidents_df['MAX_SEV_IR']
np.bincount(y)
```

```
array([551, 299, 149], dtype=int64)
```

3.데이터 전처리

3.1 data(X) 수치형 데이터 표준화

- X.keys()에서 index 키를 가져옴

```
stdsc = StandardScaler()
X.iloc[:,[3]] = stdsc.fit_transform(X.iloc[:,[3]])
```

```
X.head()
```

	ALCHL_I	PROFIL_I_R	SUR_COND	VEH_INVL
0	2	0	1	-0.517878
1	2	1	1	-0.517878
2	1	0	1	-0.517878
3	2	0	2	1.206655
4	2	1	1	1.206655

3.데이터 전처리

3.2 data(X) 레이블 인코딩

- 질변변수 가변수화
- 가변수 처리시 문자로 처리를 해야 변수명 구분이 쉬움

```
X['ALCHL_I'] = X['ALCHL_I'].replace ([1,2], ['Yes','No'])
```

```
X['PROFIL_I_R'] = X['PROFIL_I_R'].replace ([0,1], ['etc','level1'])
```

```
X['SUR_COND'] = X['SUR_COND'].replace ([1,2,3,9], ['dry','wet','snow','non'])
```

```
X.head()
```

	ALCHL_I	PROFIL_I_R	SUR_COND	VEH_INVL
0	No	etc	dry	-0.517878
1	No	level1	dry	-0.517878
2	Yes	etc	dry	-0.517878
3	No	etc	wet	1.206655
4	No	level1	dry	1.206655

3.데이터 전처리

```
X.keys()
```

```
Index(['ALCHL_I', 'PROFIL_I_R', 'SUR_COND', 'VEH_INVL'], dtype='object')
```

```
X = pd.get_dummies(X[['ALCHL_I', 'PROFIL_I_R', 'VEH_INVL', 'SUR_COND']],
                   columns=['ALCHL_I', 'PROFIL_I_R', 'SUR_COND'],
                   drop_first=True)
```

```
X.head()
```

	VEH_INVL	ALCHL_I_Yes	PROFIL_I_R_level1	SUR_COND_dry	SUR_COND_non	SUR_COND_snow	SUR_COND_wet
0	-0.517878	0	0	1	0	0	0
1	-0.517878	0	1	1	0	0	0
2	-0.517878	1	0	1	0	0	0
3	1.206655	0	0	0	0	0	1
4	1.206655	0	1	1	0	0	0

5.모델구축

5.모델구축

- kernel = linear 일때는 gamma는 사용 못함
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
svm = SVC(kernel='rbf',
          random_state=1,
          gamma=0.2,
          C=1.0)


svm.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.2, kernel='rbf',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

6.모델검정

6.2 정오분류표로 검정

- class label이 3개 이므로 추가

```
confmat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['True[0]','True[1]','True[2]'],
                       columns=['Predict[0]', 'Predict[1]','Predict[2]'])
confmat
```

	Predict[0]	Predict[1]	Predict[2]
True[0]	160	0	5
True[1]	0	90	0
True[2]	14	23	8

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       165
           1       0.80      1.00      0.89        90
           2       0.62      0.18      0.28        45

    accuracy                           0.86       300
   macro avg       0.78      0.72      0.70       300
weighted avg       0.84      0.86      0.83       300
```

6.모델검정

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       165
           1       0.80      1.00      0.89        90
           2       0.62      0.18      0.28        45

    accuracy                           0.86       300
   macro avg       0.78      0.72      0.70       300
weighted avg       0.84      0.86      0.83       300
```

6.3 정확도, 민감도 확인

- 클래스가 2개일 경우에만 실행

```
print('잘못 분류된 샘플 개수: %d' % (y_test != y_pred).sum())
print('정확도: %.3f' % accuracy_score(y_test, y_pred))
# print('정밀도: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
# print('재현율: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
# print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
잘못 분류된 샘플 개수: 42
정확도: 0.860
```

7.최적화

7.1 파이프라인 모델 만들기

- 파이프라인을 이용하여 최적 모델 만들기
- 기본모형은 아무 옵션이 없는 모델로 부터 시작
- 파라미터 옵션 확인: pipe_tree.get_params().keys()

```
pipe_svm = make_pipeline(SVC(random_state=1))
```

```
pipe_svm.get_params().keys()
```

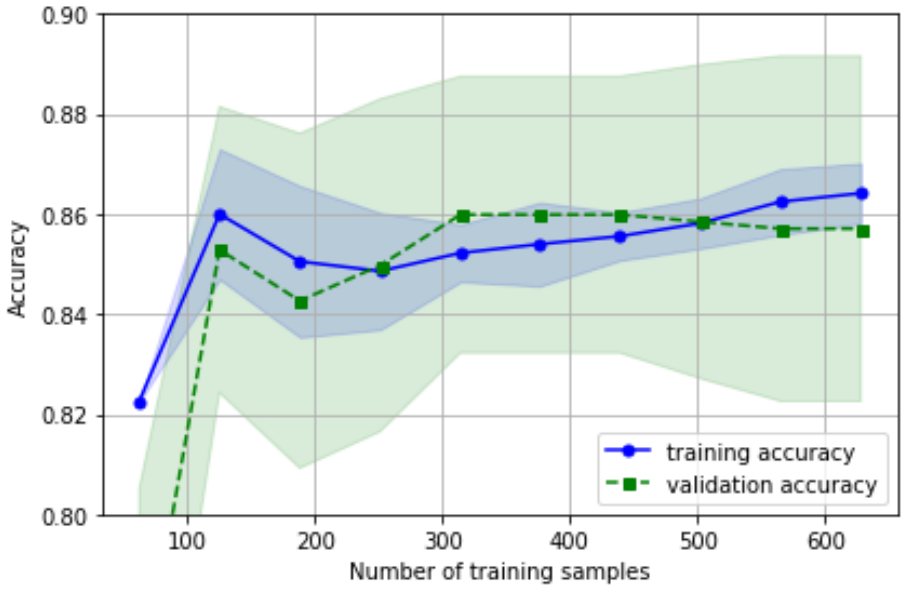
```
dict_keys(['memory', 'steps', 'verbose', 'svc', 'svc__C', 'svc__cache_size', 'svc__class_weigh
pe', 'svc__degree', 'svc__gamma', 'svc__kernel', 'svc__max_iter', 'svc__probability', 'svc__ra
vc__verbose'])
```

7.최적화

7.2 학습 곡선으로 편향과 분산 문제 분석하기

- 훈련 샘플링 수를 이용하여 편향과 분산 검정
- 편향: 정확도가 높은지 검정
- 분산: 훈련/검정 데이터의 정확도의 차이가 적은지

```
train_sizes, train_scores, test_scores =₩
                learning_curve(estimator=pipe_svm, # 수정
                               X=X_train,
                               y=y_train,
                               train_sizes=np.linspace(0.1, 1.0, 10),
                               cv=10,
                               n_jobs=1)
```



7.최적화

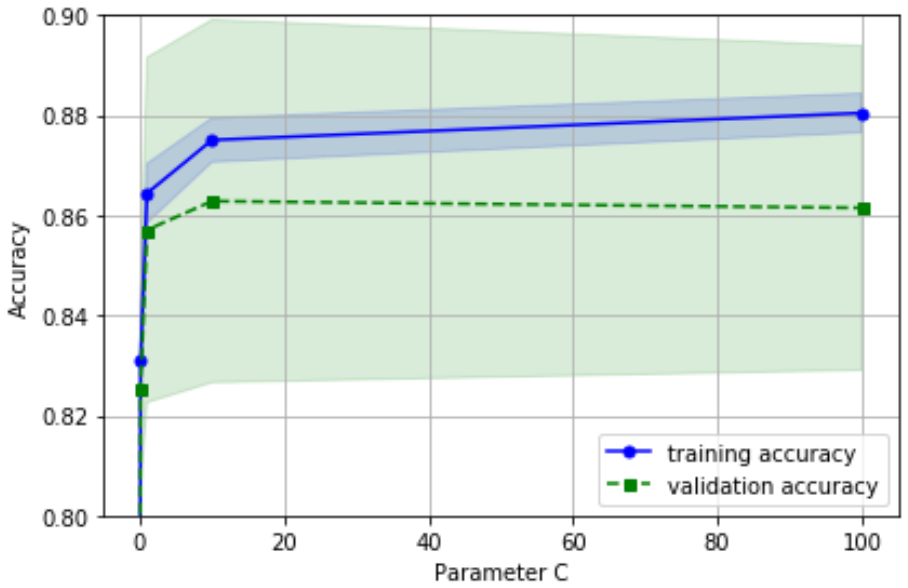
7.3 검증 곡선으로 과대적합과 과소적합 조사하기

- 과대적합 : 파라미터가 많음 -> 파라미터 축소
- 과소적합 : 파라미터가 적음 -> 파라미터 추가

```
param_range = [0.01, 0.1, 1.0, 10, 100]  # 수정

train_scores, test_scores = validation_curve(
                estimator=pipe_svm, # 수정
                X=X_train,
                y=y_train,
                param_name='svc__C', ## 수정
                param_range=param_range,
                cv=10)
```

```
plt.grid()
plt.xlabel('Number of C') # 수정
plt.legend(loc='lower right')
plt.xlabel('Parameter C') # 수정
plt.ylabel('Accuracy')
plt.ylim([0.9, 1.00])  # 수정
plt.tight_layout()
plt.show()
```



7.최적화

7.4 하이퍼파라미터 튜닝

- 그리드 서치를 사용한 머신 러닝 모델 세부 튜닝
- 기계학습 모델의 성능을 결정하는 하이퍼 파라미터 튜닝

```
param_range = [0.01, 0.1, 1.0, 10, 100]  # 수정

param_grid = [{'svc__C': param_range, # 수정
               'svc__gamma': param_range, # 수정
               'svc__kernel': ['rbf']}] # 수정

gs = GridSearchCV(estimator=pipe_svm, # 수정
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)

gs = gs.fit(X_train, y_train)

print(gs.best_score_)
print(gs.best_params_)
```

```
0.8683834048640916
{'svc__C': 10, 'svc__gamma': 0.1, 'svc__kernel': 'rbf'}
```

8.최적화 모델 검정

8.최적화 모델 검정

- 최적모델을 이용해 검정 데이터(full data) 최종 확인
- best_tree 로 모델명 변경

```
best_svm = gs.best_estimator_
best_svm.fit(X_train, y_train)
```

```
Pipeline(memory=None,
         steps=[('svc',
                 SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
                     decision_function_shape='ovr', degree=3, gamma=0.1,
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=1, shrinking=True, tol=0.001,
                     verbose=False))],
         verbose=False)
```

8.최적화 모델 검정

- 정오분류표로 검정

```
confmat = pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['True[0]','True[1]','True[2]'],
                       columns=['Predict[0]', 'Predict[1]','Predict[2]'])
confmat
```

	Predict[0]	Predict[1]	Predict[2]
True[0]	160	0	5
True[1]	0	88	2
True[2]	14	19	12

```
print('Classification Report')
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       165
           1       0.82      0.98      0.89        90
           2       0.63      0.27      0.38        45

    accuracy                           0.87       300
   macro avg       0.79      0.74      0.74       300
weighted avg       0.85      0.87      0.84       300
```

- 정확도, 민감도 확인

```
print('잘못 분류된 샘플 개수: %d' % (y_test != y_pred).sum())
print('정확도: %.3f' % accuracy_score(y_test, y_pred))
# print('정밀도: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
# print('재현율: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
# print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
잘못 분류된 샘플 개수: 40
정확도: 0.867
```

참고자료

동영상 및 참고교재

- International
  - CS231n: Convolutional Neural Networks for Visual Recognition, http://cs231n.stanford.edu/

- Domestic
  - **머신러닝 교과서 with 파이썬, 사이킷런, 텐서플로**, 세바스찬 라시카, 바히드 미자리리 지음, 박해선 옮김, 길벗, 2019
  - **케라스 창시자에게 배우는 딥러닝**, 프랑소와 숄레 지음, 박해선 옮김, 길벗, 2019
  - K-MOOC
    - 파이썬 프로그래밍, 김경미, 한동대학교
    - 파이썬을 이용한 빅데이터 분석, 유성준, 세종대학교
    - 딥러닝 개론, 김희철, 대구대학교
    - 파이썬으로 배우는 기계학습 입문, 김영섭, 한동대학교