7.2 나이브베이즈 분류모형

조건부독립

확률변수 A, B가 독립이면 A, B의 결합확률은 주변확률의 곱과 같다.

$$P(A, B) = P(A)P(B)$$

조건부독립(conditional independence)은 일반적인 독립과 달리 조건이 되는 별개의 확률변수 C가 존재해야 한다. 조건이 되는 확률변수 C에 대한 A, B의 결합조건부확률이 C에 대한 A, B의 조건부확률의 곱과 같으면 A와 B가 C에 대해 조건부독립이라고 한다.

$$P(A, B|C) = P(A|C)P(B|C)$$

기호로는 다음과 같이 표기한다.

$$A \perp\!\!\!\perp B \mid C$$

조건부독립과 비교하여 일반적인 독립은 무조건부독립이라고 한다. 무조건부독립은 다음과 같이 표기하도 한다.

$$A \perp\!\!\!\perp B \mid \varnothing$$

A, B가 C에 대해 조건부독립이면 다음도 만족한다.

$$P(A|B, C) = P(A|C)$$

$$P(B|A, C) = P(B|C)$$

주의할 점은 조건부독립과 무조건부독립은 관계가 없다는 점이다. 즉, 두 확률변수가 독립이라고 항상 조건부독립이 되는 것도 아니고 조건부독립이라고 꼭 독립이 되는 것도 아니다.

$$P(A, B) = P(A)P(B) \;\; \bcancel{\Longrightarrow} \;\; P(A, B|C) = P(A|C)P(B|C)$$

$$P(A, B|C) = P(A|C)P(B|C) \;\; \bcancel{\Longrightarrow} \;\; P(A, B) = P(A)P(B)$$

예를 들어 어떤 동물의 어미의 몸무게가 $x$kg일 때 새끼의 몸무게는 x를 기댓값으로 하고 5kg 표준편차를 가지는 정규분포라고 가정하자. 이 동물의 새끼 중 2마리의 몸무게를 각각 A, B라고 하고 어미의 몸무게를 C라고 한다. 시뮬레이션을 통해 어미 표본과 각각의 어미에 대해 2마리의 새끼 표본을 만들자.

In [1]:

```
np.random.seed(0)
C = np.random.normal(100, 15, 2000)
A = C + np.random.normal(0, 5, 2000)
B = C + np.random.normal(0, 5, 2000)
```

스캐터플롯을 보면 B와 C 자체는 상관관계가 있음을 알 수 있다. 하지만 어미의 몸무게가 고정되어 있으면 두 새끼의 몸무게는 서로 독립인 것을 볼 수 있다.

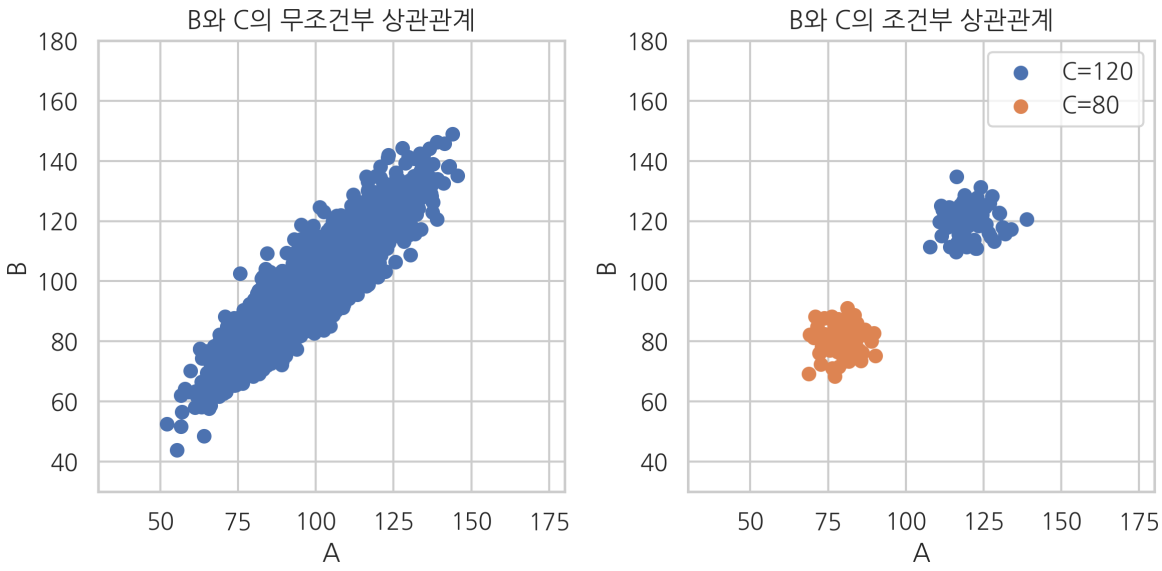
In [2]:

```
plt.figure(figsize=(8, 4))

plt.subplot(121)
plt.scatter(A, B)
plt.xlabel("A")
plt.ylabel("B")
plt.xlim(30, 180)
plt.ylim(30, 180)
plt.title("B와 C의 무조건부 상관관계")

plt.subplot(122)
idx1 = (118 < C) & (C < 122)
idx2 = (78 < C) & (C < 82)
plt.scatter(A[idx1], B[idx1], label="C=120")
plt.scatter(A[idx2], B[idx2], label="C=80")
plt.xlabel("A")
plt.ylabel("B")
plt.xlim(30, 180)
plt.ylim(30, 180)
plt.legend()
plt.title("B와 C의 조건부 상관관계")

plt.tight_layout()
plt.show()
```



나이브 가정

독립변수 x가 D차원이라고 가정하자.

$$x = (x_1, \ldots, x_D)$$

가능도함수는 $x_1, \ldots, x_D$의 결합확률이 된다.

$$P(x \mid y = k) = P(x_1, \ldots, x_D \mid y = k)$$

원리상으로는 $y=k$인 데이터만 모아서 이 가능도함수의 모양을 추정할 수 있다. 하지만 차원 D가 커지면 가능도함수의 추정이 현실적으로 어려워진다.

따라서 나이즈베이즈 분류모형(Naive Bayes classification model)에서는 모든 차원의 개별 독립변수가 서로 조건부독립(conditional independent)이라는 가정을 사용한다. 이러한 가정을 나이브 가정(naive assumption)이라고 한다.

나이브 가정으로 사용하면 벡터 x의 결합확률분포함수는 개별 스칼라 원소 x_d의 확률분포함수의 곱이 된다.

$$P(x_1, \ldots, x_D \mid y=k) = \prod_{d=1}^{D} P(x_d \mid y=k)$$

스칼라 원소 x_d의 확률분포함수는 결합확률분포함수보다 추정하기 훨씬 쉽다.

가능도함수를 추정한 후에는 베이즈정리를 사용하여 조건부확률을 계산할 수 있다.

$$\begin{aligned} P(y=k \mid x) &= \frac{P(x_1, \ldots, x_D \mid y=k)P(y=k)}{P(x)} \\ &= \frac{\left(\prod_{d=1}^{D} P(x_d \mid y=k)\right) P(y=k)}{P(x)} \end{aligned}$$

정규분포 가능도 모형

x 벡터의 원소가 모두 실수이고 클래스마다 특정한 값 주변에서 발생한다고 하면 가능도 분포로 정규분포를 사용한다. 각 독립변수 x_d마다, 그리고 클래스 k마다 정규 분포의 기댓값 $\mu_{d,k}$, 표준 편차 $\sigma_{d,k}^2$가 달라진다. QDA 모형과는 달리 모든 독립변수들이 서로 조건부독립이라고 가정한다.

$$P(x_d \mid y=k) = \frac{1}{\sqrt{2\pi\sigma_{d,k}^2}} \exp\left(-\frac{(x_d-\mu_{d,k})^2}{2\sigma_{d,k}^2}\right)$$

베르누이분포 가능도 모형

베르누이분포 가능도 모형에서는 각각의 $x=(x_1, \ldots, x_D)$의 각 원소 x_d가 0 또는 1이라는 값만을 가질 수 있다. 즉 독립변수는 D개의 독립적인 베르누이 확률변수, 즉 동전으로 구성된 동전 세트로 표현할 수 있다. 이 동전들의 모수 μ_d는 동전 d마다 다르다.

그런데 클래스 $y=k$ $(k=1, \ldots, K)$마다도 x_d가 1이 될 확률이 다르다. 즉, 동전의 모수 $\mu_{d,k}$는 동전 d마다 다르고 클래스 k마다도 다르다. 즉, 전체 $D \times K$의 동전이 존재하며 같은 클래스에 속하는 D개의 동전이 하나의 동전 세트를 구성하고 이러한 동전 세트가 K개 있다고 생각할 수 있다.

$$P(x_d \mid y=k) = \mu_{d,k}^{x_d}(1-\mu_{d,k})^{(1-x_d)}$$

$$P(x_1, \ldots, x_D \mid y=k) = \prod_{d=1}^{D} \mu_{d,k}^{x_d}(1-\mu_{d,k})^{(1-x_d)}$$

이러한 동전 세트마다 확률 특성이 다르므로 베르누이분포 가능도 모형을 기반으로 하는 나이브베이즈 모형은 동전 세트를 N번 던진 결과로부터 $1, \ldots, K$ 중 어느 동전 세트를 던졌는지를 찾아내는 모형이라고 할 수 있다.

다항분포 가능도 모형

다항분포 모형에서는 x 벡터가 다항분포의 표본이라고 가정한다. 즉, D개의 면을 가지는 주사위를 $\sum_{d=1}^{D} x_d$ 번 던져서 나온 결과로 본다. 예를 들어 x가 다음과 같다면,

$$x = (1, 4, 0, 5)$$

4면체 주사위를 10번 던져서 1인 면이 1번, 2인 면이 4번, 4인 면이 5번 나온 결과로 해석한다.

각 클래스마다 주사위가 다르다고 가정하므로 K개의 클래스를 구분하는 문제에서는 D개의 면을 가진 주사위가 K개 있다고 본다.

$$P(x_1, \ldots, x_D \mid y = k) \propto \prod_{d=1}^{D} \mu_{d,k}^{x_{d,k}}$$

$$\sum_{d=1}^{D} \mu_{d,k} = 1$$

따라서 다항분포 가능도 모형을 기반으로 하는 나이브베이즈 모형은 주사위를 던진 결과로부터 $1, \ldots, K$ 중 어느 주사위를 던졌는지를 찾아내는 모형이라고 할 수 있다.

사이킷런에서 제공하는 나이브베이즈 모형

사이킷런의 naive_bayes 서브패키지에서는 다음과 같은 세가지 나이브베이즈 모형 클래스를 제공한다.

- `GaussianNB` (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html): 정규분포 나이브베이즈
- `BernoulliNB` (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html): 베르누이분포 나이브베이즈
- `MultinomialNB` (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html): 다항분포 나이브베이즈

이 클래스들은 다양한 속성값 및 메서드를 가진다. 우선 사전 확률과 관련된 속성은 다음과 같다.

- `classes_`
  - 종속변수 Y의 클래스(라벨)
- `class_count_`
  - 종속변수 Y의 값이 특정한 클래스인 표본 데이터의 수
- `class_prior_`
  - 종속변수 Y의 무조건부 확률분포 $P(Y)$ (정규분포의 경우에만)
- `class_log_prior_`
  - 종속변수 Y의 무조건부 확률분포의 로그 $\log P(Y)$ (베르누이분포나 다항분포의 경우에만)

정규분포 나이브베이즈 모형

가우시안 나이브베이즈 모형 `GaussianNB` 은 가능도 추정과 관련하여 다음과 같은 속성을 가진다.

- theta_ : 정규분포의 기댓값 μ
- sigma_ : 정규분포의 분산 σ^2

예제

실수인 두 개의 독립변수 x_1, x_2와 두 종류의 클래스 $y = 0, 1$을 가지는 분류문제가 있다.

두 독립변수의 분포는 정규분포이고 y의 클래스에 따라 다음처럼 모수가 달라진다.

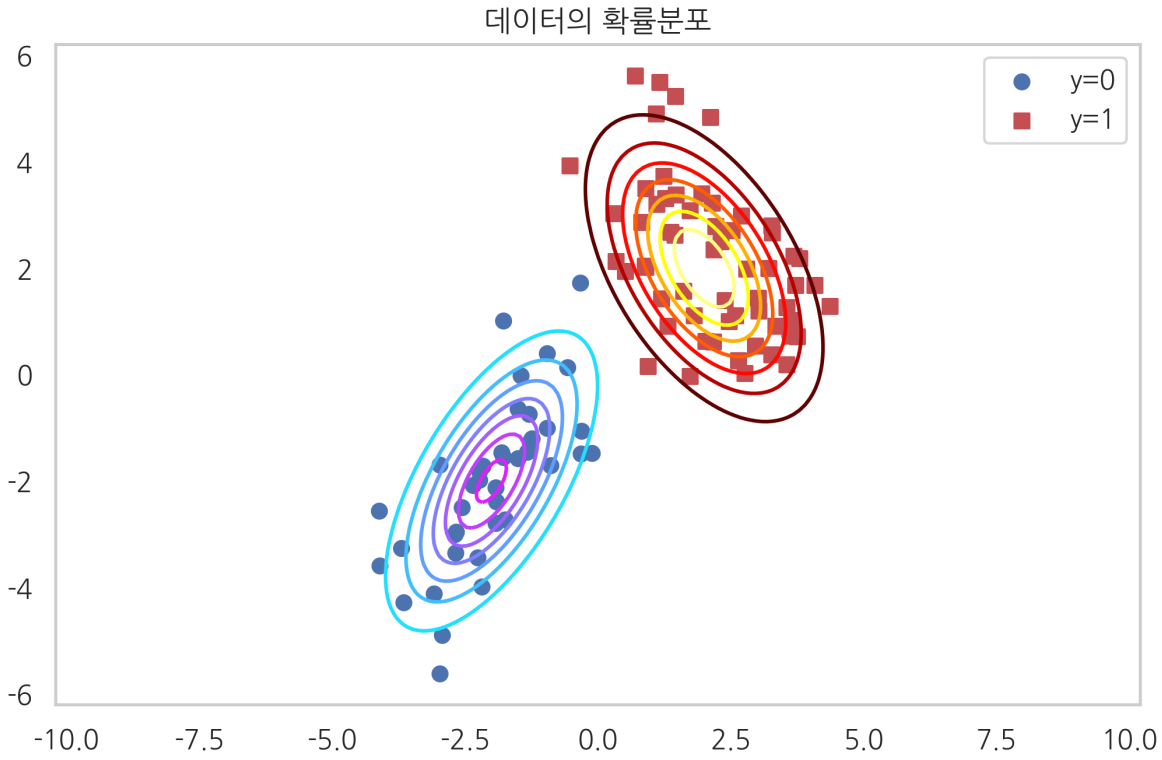
$$\mu_0 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \quad \Sigma_0 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 2 \end{bmatrix}$$
$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 1.2 & -0.8 \\ -0.8 & 2 \end{bmatrix}$$

데이터는 $y = 0$인 데이터가 40개, $y = 1$인 데이터가 60개 주어졌다. 이 데이터를 시각화하면 다음과 같다.

In [3]:

```
np.random.seed(0)
rv0 = sp.stats.multivariate_normal([-2, -2], [[1, 0.9], [0.9, 2]])
rv1 = sp.stats.multivariate_normal([2, 2], [[1.2, -0.8], [-0.8, 2]])
X0 = rv0.rvs(40)
X1 = rv1.rvs(60)
X = np.vstack([X0, X1])
y = np.hstack([np.zeros(40), np.ones(60)])

xx1 = np.linspace(-5, 5, 100)
xx2 = np.linspace(-5, 5, 100)
XX1, XX2 = np.meshgrid(xx1, xx2)
plt.grid(False)
plt.contour(XX1, XX2, rv0.pdf(np.dstack([XX1, XX2])), cmap=mpl.cm.cool)
plt.contour(XX1, XX2, rv1.pdf(np.dstack([XX1, XX2])), cmap=mpl.cm.hot)
plt.scatter(X0[:, 0], X0[:, 1], marker="o", c='b', label="y=0")
plt.scatter(X1[:, 0], X1[:, 1], marker="s", c='r', label="y=1")
plt.legend()
plt.title("데이터의 확률분포")
plt.axis("equal")
plt.show()
```



이 데이터를 가우시안 나이브베이즈 모형으로 다음처럼 풀 수 있다.

In [4]:

```
from sklearn.naive_bayes import GaussianNB
model_norm = GaussianNB().fit(X, y)
```

y 클래스의 종류와 각 클래스에 속하는 표본의 수, 그리고 그 값으로부터 구한 사전 확률

$$p(y=0), p(y=1)$$

의 값은 다음과 같다.

In [5]:

```
model_norm.classes_
```

Out[5]:

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

In [6]:

```
model_norm.class_count_
```

Out[6]:

```
array([40., 60.])
```

In [7]:

```
model_norm.class_prior_
```

Out[7]:

```
array([0.4, 0.6])
```

각 클래스에 따라 x가 이루는 확률분포의 모수를 계산하면 다음과 같다. 나이브 가정에 따라 x_1, x_2는 독립이므로 상관관계를 구하지 않았다.

In [8]:

```
model_norm.theta_[0], model_norm.sigma_[0]
```

Out[8]:

```
(array([-1.96197643, -2.00597903]), array([1.02398854, 2.31390497]))
```

In [9]:

```
model_norm.theta_[1], model_norm.sigma_[1]
```

Out[9]:

```
(array([2.19130701, 2.12626716]), array([1.25429371, 1.93742544]))
```

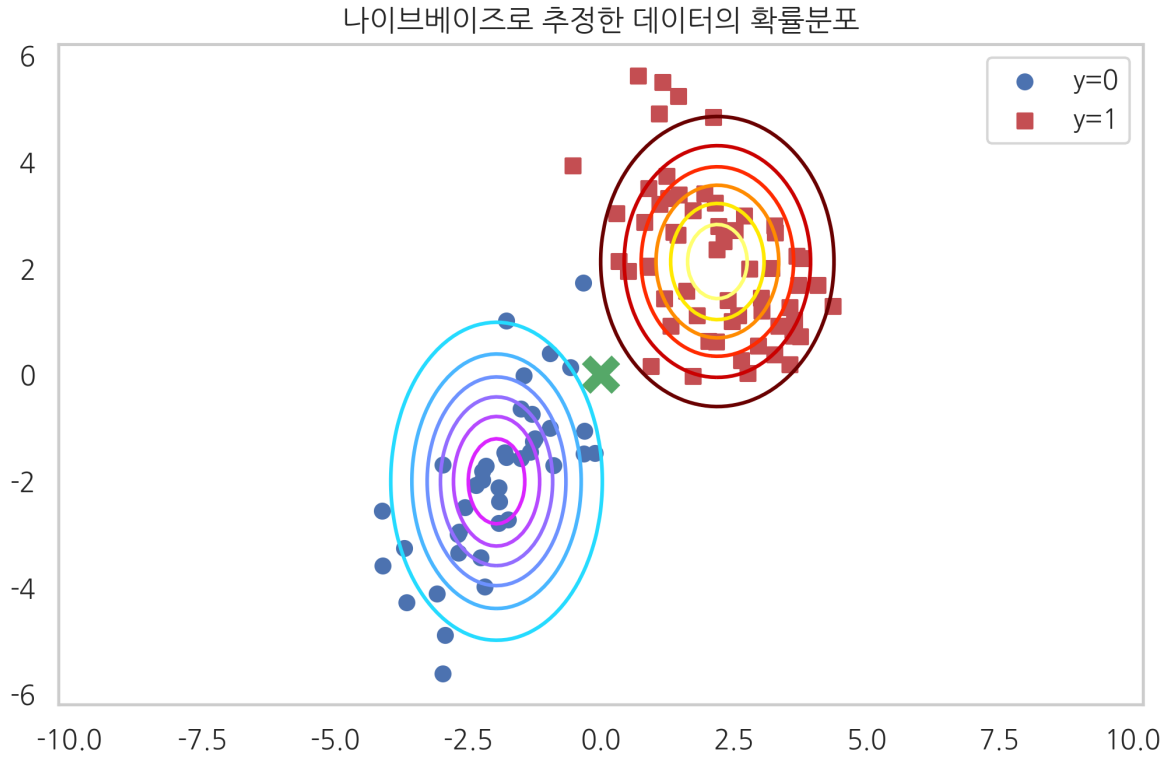
이렇게 구한 데이터의 확률분포를 시각화하면 다음과 같다.

In [10]:

```
rv0 = sp.stats.multivariate_normal(model_norm.theta_[0], model_norm.sigma_[0])
rv1 = sp.stats.multivariate_normal(model_norm.theta_[1], model_norm.sigma_[1])

xx1 = np.linspace(-5, 5, 100)
xx2 = np.linspace(-5, 5, 100)
XX1, XX2 = np.meshgrid(xx1, xx2)
plt.grid(False)
plt.contour(XX1, XX2, rv0.pdf(np.dstack([XX1, XX2])), cmap=mpl.cm.cool)
plt.contour(XX1, XX2, rv1.pdf(np.dstack([XX1, XX2])), cmap=mpl.cm.hot)
plt.scatter(X0[:, 0], X0[:, 1], marker="o", c='b', label="y=0")
plt.scatter(X1[:, 0], X1[:, 1], marker="s", c='r', label="y=1")

x_new = [0, 0]
plt.scatter(x_new[0], x_new[1], c="g", marker="x", s=150, linewidth=5)
plt.legend()
plt.title("나이브베이즈로 추정한 데이터의 확률분포")
plt.axis("equal")
plt.show()
```



이 모형을 사용하여 $x_{\text{new}} = (0, 0)$인 데이터의 y값을 예측하자. 각 클래스값이 나올 확률은 `predict_proba` 메서드로 구할 수 있다. 결과는 y=0일 확률이 0.48, y=1일 확률이 0.52이다.

In [11]:

```
model_norm.predict_proba([x_new])
```

Out[11]:

```
array([[0.48475244, 0.51524756]])
```

이 값이 나오게 된 중간 과정을 살펴보자. 우선 추정된 독립변수의 모수와 정규 분포의 확률 밀도 함수를 사용하여 가능도를 구할 수 있다. 나이브베이즈 가정에 따라 두 입력 변수의 곱을 결합 확률로 계산한다.

$$p(x_1, x_2|y) \propto p(x_1)p(x_2)$$

In [12]:

```
likelihood = [
    (sp.stats.norm(model_norm.theta_[0][0], np.sqrt(model_norm.sigma_[0][0])).pdf(x_new[0]) * ₩
     sp.stats.norm(model_norm.theta_[0][1], np.sqrt(model_norm.sigma_[0][1])).pdf(x_new[1])),
    (sp.stats.norm(model_norm.theta_[1][0], np.sqrt(model_norm.sigma_[1][0])).pdf(x_new[0]) * ₩
     sp.stats.norm(model_norm.theta_[1][1], np.sqrt(model_norm.sigma_[1][1])).pdf(x_new[1])),
]
likelihood
```

Out[12]:

```
[0.006615760017637298, 0.00468796559514829]
```

여기에 사전 확률을 곱하면 사후 확률에 비례하는 값이 나온다.

$$p(y|x_1, x_2) \propto p(x_1, x_2|y)p(y)$$

아직 정규화 상수 $p(x)$로 나누어주지 않았으므로 두 값의 합이 1이 아니다. 즉, 확률이라고 부를 수는 없다. 하지만 크기를 비교하면 y=0일 확률이 y=1일 확률보다 훨씬 크다는 것을 알 수 있다.

In [13]:

```
posterior = likelihood * model_norm.class_prior_
posterior
```

Out[13]:

```
array([0.0026463 , 0.00281278])
```

이 값을 정규화하면 `predict_proba` 메서드로 구한 것과 같은 값이 나온다.

In [14]:

```
posterior / posterior.sum()
```

Out[14]:

```
array([0.48475244, 0.51524756])
```

연습 문제 1

붓꽃 분류문제를 가우시안 나이브베이즈 모형을 사용하여 풀어보자.

(1) 각각의 종이 선택될 사전확률을 구하라.

(2) 각각의 종에 대해 꽃받침의 길이, 꽃받침의 폭, 꽃잎의 길이, 꽃잎의 폭의 평균과 분산을 구하라.

(3) 학습용 데이터를 사용하여 분류문제를 풀고 다음을 계산하라.

- 분류결과표
- 분류보고서

In [15]:

```
from sklearn.datasets import load_iris

iris = load_iris()
X1 = iris.data
y1 = iris.target

from sklearn.naive_bayes import GaussianNB

model1 = GaussianNB().fit(X1, y1)
model1.class_prior_
```

Out[15]:

```
array([0.33333333, 0.33333333, 0.33333333])
```

In [16]:

```
model1.theta_
```

Out[16]:

```
array([[5.006, 3.428, 1.462, 0.246],
       [5.936, 2.77 , 4.26 , 1.326],
       [6.588, 2.974, 5.552, 2.026]])
```

In [17]:

```
model1.sigma_
```

Out[17]:

```
array([[0.121764, 0.140816, 0.029556, 0.010884],
       [0.261104, 0.0965  , 0.2164  , 0.038324],
       [0.396256, 0.101924, 0.298496, 0.073924]])
```

In [18]:

```
y1_pred = model1.predict(X1)

from sklearn.metrics import confusion_matrix

confusion_matrix(y1, y1_pred)
```

Out[18]:

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  3, 47]])
```

In [19]:

```
from sklearn.metrics import classification_report

print(classification_report(y1, y1_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.94      0.94      0.94        50
           2       0.94      0.94      0.94        50

    accuracy                           0.96       150
   macro avg       0.96      0.96      0.96       150
weighted avg       0.96      0.96      0.96       150
```

베르누이분포 나이브베이즈 모형

전자우편과 같은 문서 내에 특정한 단어가 포함되어 있는지의 여부는 베르누이 확률변수로 모형화할 수 있다. 이렇게 독립변수가 0 또는 1의 값을 가지면 베르누이 나이브베이즈 모형을 사용한다.

베르누이분포 나이브베이즈 모형 클래스 `BernoulliNB` 는 가능도 추정과 관련하여 다음 속성을 가진다.

- `feature_count_` : 각 클래스 k에 대해 d번째 동전이 앞면이 나온 횟수 $N_{d,k}$
- `feature_log_prob_` : 베르누이분포 모수의 로그

$$\log \mu_k = (\log \mu_{1,k}, \ldots, \log \mu_{D,k}) = \left(\log \frac{N_{1,k}}{N_k}, \ldots, \log \frac{N_{D,k}}{N_k} \right)$$

여기에서 N_k은 클래스 k에 대해 동전을 던진 총 횟수이다. 표본 데이터의 수가 적은 경우에는 모수에 대해 다음처럼 스무딩(smoothing)을 할 수도 있다.

스무딩

표본 데이터의 수가 적은 경우에는 베르누이 모수가 0 또는 1이라는 극단적인 모수 추정값이 나올 수도 있다. 하지만 현실적으로는 실제 모수값이 이런 극단적인 값이 나올 가능성이 적다. 따라서 베르누이 모수가 0.5인 가장 일반적인 경우를 가정하여 0이 나오는 경우와 1이 나오는 경우, 두 개의 가상 표본 데이터를 추가한다. 그러면 0이나 1과 같은 극단적인 추정값이 0.5에 가까운 다음과 값으로 변한다. 이를 라플라스 스무딩(Laplace smoothing) 또는 애드원(Add-One) 스무딩이라고 한다.

$$\hat{\mu}_{d,k} = \frac{N_{d,k} + \alpha}{N_k + 2\alpha}$$

가중치 α를 사용하여 스무딩의 정도를 조절할 수도 있다. 가중치 α는 정수가 아니라도 괜찮다. 가중치가 1인 경우는 무정보 사전확률을 사용한 베이즈 모수추정의 결과와 같다.

In [20]:

```
X = np.array([
    [0, 1, 1, 0],
    [1, 1, 1, 1],
    [1, 1, 1, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 1],
    [0, 1, 1, 0],
    [0, 1, 1, 1],
    [1, 0, 1, 0],
    [1, 0, 1, 1],
    [0, 1, 1, 0]])
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

이 데이터는 4개의 키워드를 사용하여 정상 메일 4개와 스팸 메일 6개를 BOW 인코딩한 행렬이다. 예를 들어 첫 번째 메일은 정상 메일이고 1번, 4번 키워드는 포함하지 않지만 2번, 3번 키워드를 포함한다고 볼 수 있다.

이 데이터를 베르누이 나이브베이즈 모형으로 예측해 보자.

In [21]:

```
from sklearn.naive_bayes import BernoulliNB
model_bern = BernoulliNB().fit(X, y)
```

y 클래스의 종류와 각 클래스에 속하는 표본의 수, 그리고 그 값으로부터 구한 사전 확률의 값은 다음과 같다.

In [22]:

```
model_bern.classes_
```

Out[22]:

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

In [23]:

```
model_bern.class_count_
```

Out[23]:

```
array([4., 6.])
```

In [24]:

```
np.exp(model_bern.class_log_prior_)
```

Out[24]:

```
array([0.4, 0.6])
```

각 클래스 k 별로, 그리고 각 독립변수 d 별로, 각각 다른 베르누이 확률변수라고 가정하여 모두 8개의 베르누이 확률변수의 모수를 구하면 다음과 같다.

In [25]:

```
fc = model_bern.feature_count_
fc
```

Out[25]:

```
array([[2., 4., 3., 1.],
       [2., 3., 5., 3.]])
```

In [26]:

```
fc / np.repeat(model_bern.class_count_[:, np.newaxis], 4, axis=1)
```

Out[26]:

```
array([[0.5       , 1.        , 0.75      , 0.25      ],
       [0.33333333, 0.5       , 0.83333333, 0.5       ]])
```

그런데 이 값은 모형 내에서 구한 값과 다르다. 모형 내에서 스무딩(smoothing)이 이루어지기 때문이다. 스무딩은 동전의 각 면 즉, 0과 1이 나오는 가상의 데이터를 추가함으로서 추정한 모수의 값이 좀 더 0.5에 가까워지도록 하는 방법이다. 이 때 사용한 스무딩 가중치 값은 다음처럼 확인할 수 있다.

In [27]:

```
model_bern.alpha
```

Out[27]:

```
1.0
```

스무딩이 적용된 베르누이 모수값은 다음과 같다.

In [28]:

```
theta = np.exp(model_bern.feature_log_prob_)
theta
```

Out[28]:

```
array([[0.5       , 0.83333333, 0.66666667, 0.33333333],
       [0.375     , 0.5       , 0.75      , 0.5       ]])
```

이에 모형이 완성되었으니 테스트 데이터를 사용하여 예측을 해 본다. 예를 들어 1번, 2번 키워드를 포함한 메일이 정상 메일인지 스팸 메일인지 알아보자.

In [29]:

```
x_new = np.array([0, 1, 1, 1])
```

In [30]:

```
model_bern.predict_proba([x_new])
```

Out[30]:

```
array([[0.34501348, 0.65498652]])
```

위 결과에서 정상 메일일 가능성이 약 3배임을 알 수 있다. 이 값은 다음처럼 구할 수도 있다.

In [31]:

```
p = ((theta ** x_new) * (1 - theta) ** (1 - x_new)).prod(axis=1)₩
    * np.exp(model_bern.class_log_prior_)
p / p.sum()
```

Out[31]:

```
array([0.34501348, 0.65498652])
```

반대로 3번, 4번 키워드가 포함된 메일은 스팸일 가능성이 약 90%이다.

In [32]:

```
x_new = np.array([0, 0, 1, 1])
```

In [33]:

```
model_bern.predict_proba([x_new])
```

Out[33]:

```
array([[0.09530901, 0.90469099]])
```

In [34]:

```
p = ((theta ** x_new) * (1 - theta) ** (1 - x_new)).prod(axis=1)₩
    * np.exp(model_bern.class_log_prior_)
p / p.sum()
```

Out[34]:

```
array([0.09530901, 0.90469099])
```

연습 문제 2

(1) MNIST 숫자 분류문제에서 sklearn.preprocessing.Binarizer 로 x값을 0, 1로 바꾼다(값이 8 이상이면 1, 8 미만이면 0). 즉 흰색과 검은색 픽셀로만 구성된 이미지로 만든다(다음 코드 참조)

```
from sklearn.datasets import load_digits
digits = load_digits()
X2 = digits.data
y2 = digits.target
from sklearn.preprocessing import Binarizer
X2 = Binarizer(7).fit_transform(X2)
```

이 이미지에 대해 베르누이 나이브베이즈 모형을 적용하자. 분류 결과를 분류보고서 형식으로 나타내라.

(2) BernoulliNB 클래스의 binarize 인수를 사용하여 같은 문제를 풀어본다.

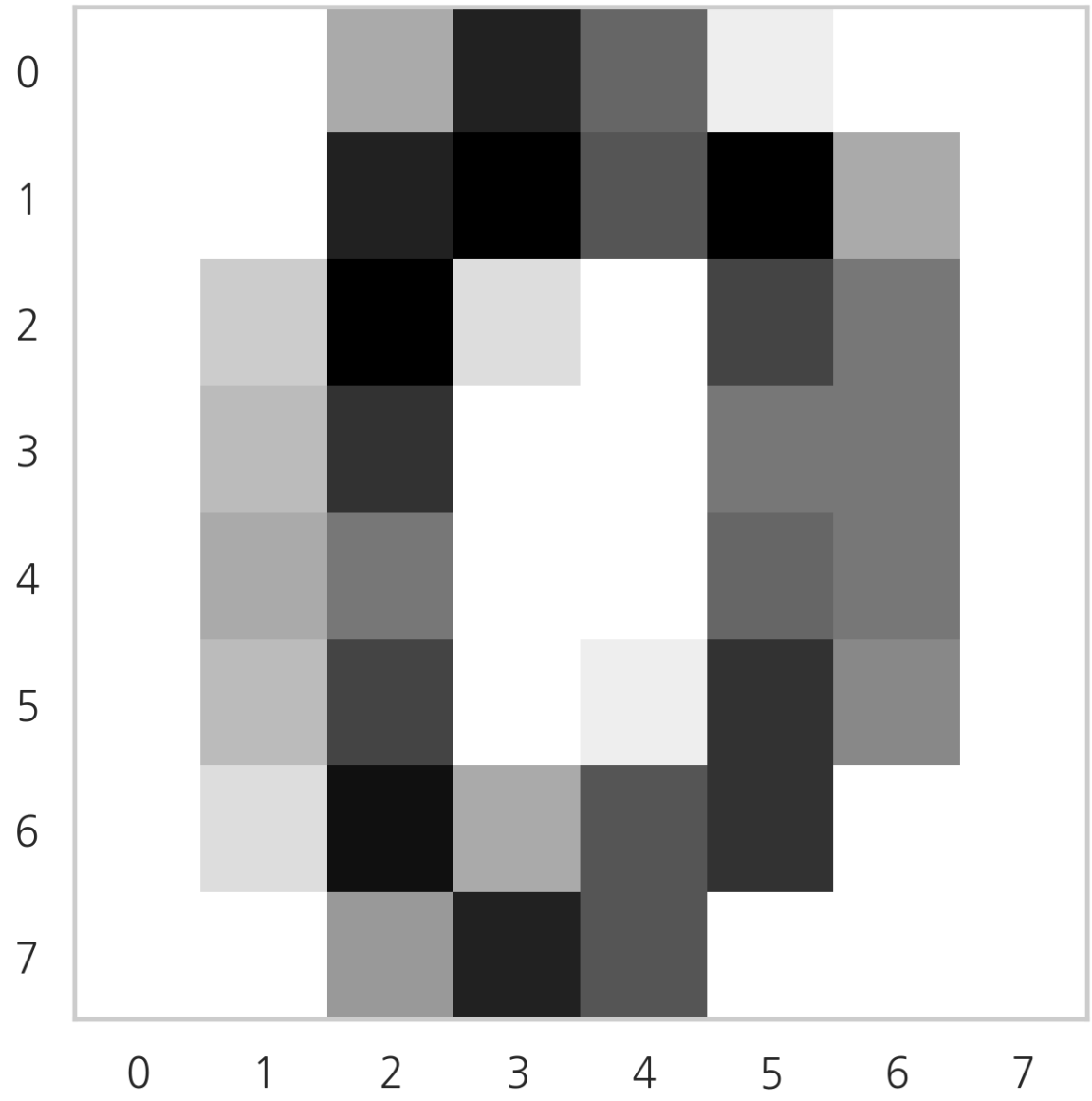
(3) 계산된 모형의 모수 벡터 값을 각 클래스별로 8x8 이미지의 형태로 나타내라. 이 이미지는 무엇을 뜻하는가?

In [35]:

```
from sklearn.datasets import load_digits

digits = load_digits()
X2 = digits.data
y2 = digits.target


plt.imshow(X2[0, :].reshape((8, 8)), cmap=plt.cm.binary)
plt.grid(False)
plt.show()
```

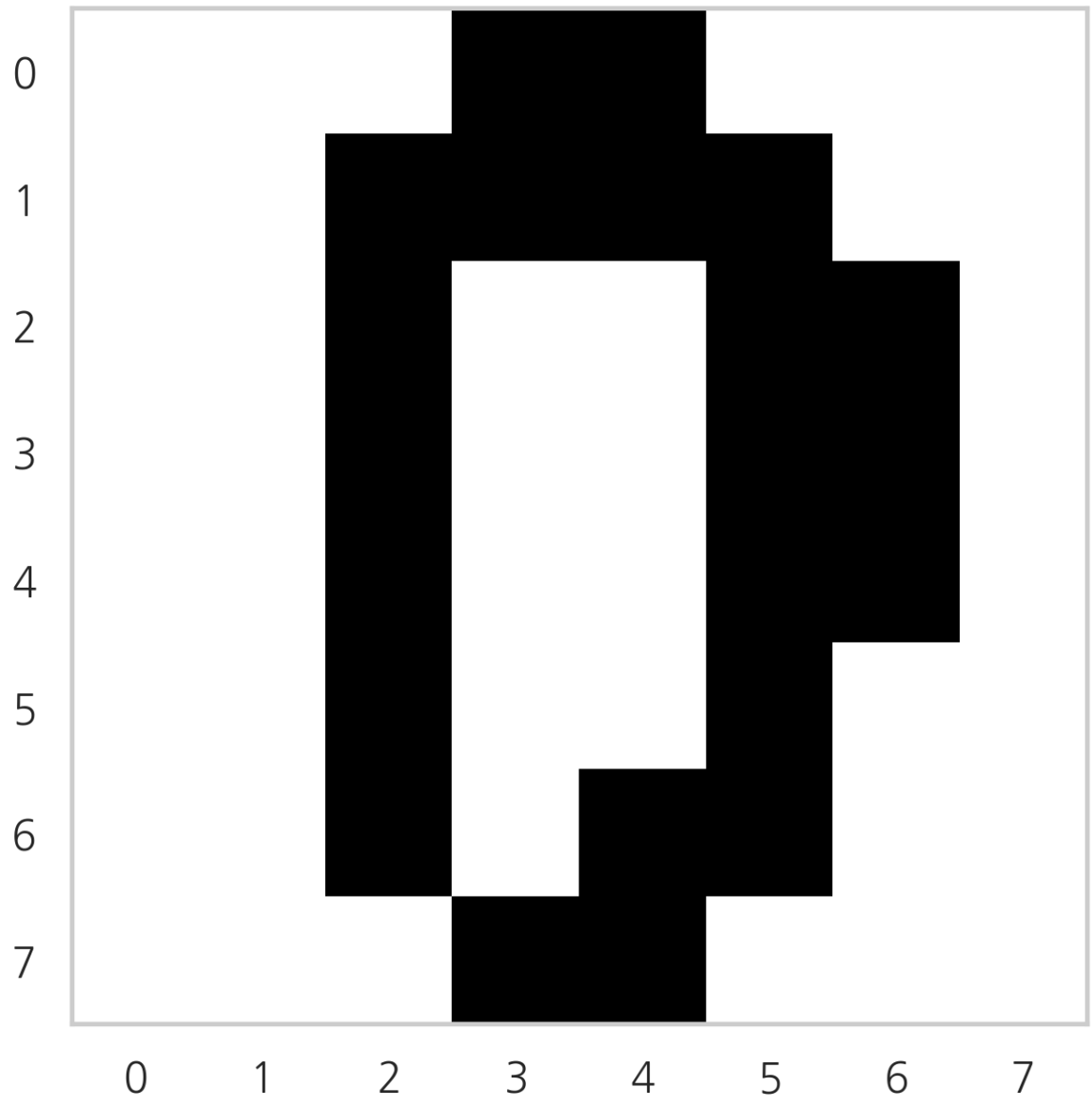


In [36]:

```
from sklearn.preprocessing import Binarizer

X2 = Binarizer(7).fit_transform(X2)

plt.imshow(X2[0, :].reshape((8, 8)), cmap=plt.cm.binary)
plt.grid(False)
plt.show()
```



In [37]:

```
from sklearn.naive_bayes import BernoulliNB

model = BernoulliNB().fit(X2, y2)
y2_pred = model.predict(X2)

from sklearn.metrics import classification_report

print(classification_report(y2, y2_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       178
           1       0.80      0.80      0.80       182
           2       0.91      0.90      0.91       177
           3       0.93      0.85      0.89       183
           4       0.96      0.94      0.95       181
           5       0.92      0.88      0.90       182
           6       0.97      0.96      0.97       181
           7       0.91      0.99      0.95       179
           8       0.80      0.82      0.81       174
           9       0.80      0.87      0.83       180

    accuracy                           0.90      1797
   macro avg       0.90      0.90      0.90      1797
weighted avg       0.90      0.90      0.90      1797
```

In [38]:

```
X2 = digits.data
y2 = digits.target

model = BernoulliNB(binarize=7).fit(X2, y2)
y2_pred = model.predict(X2)

from sklearn.metrics import classification_report

print(classification_report(y2, y2_pred))
```

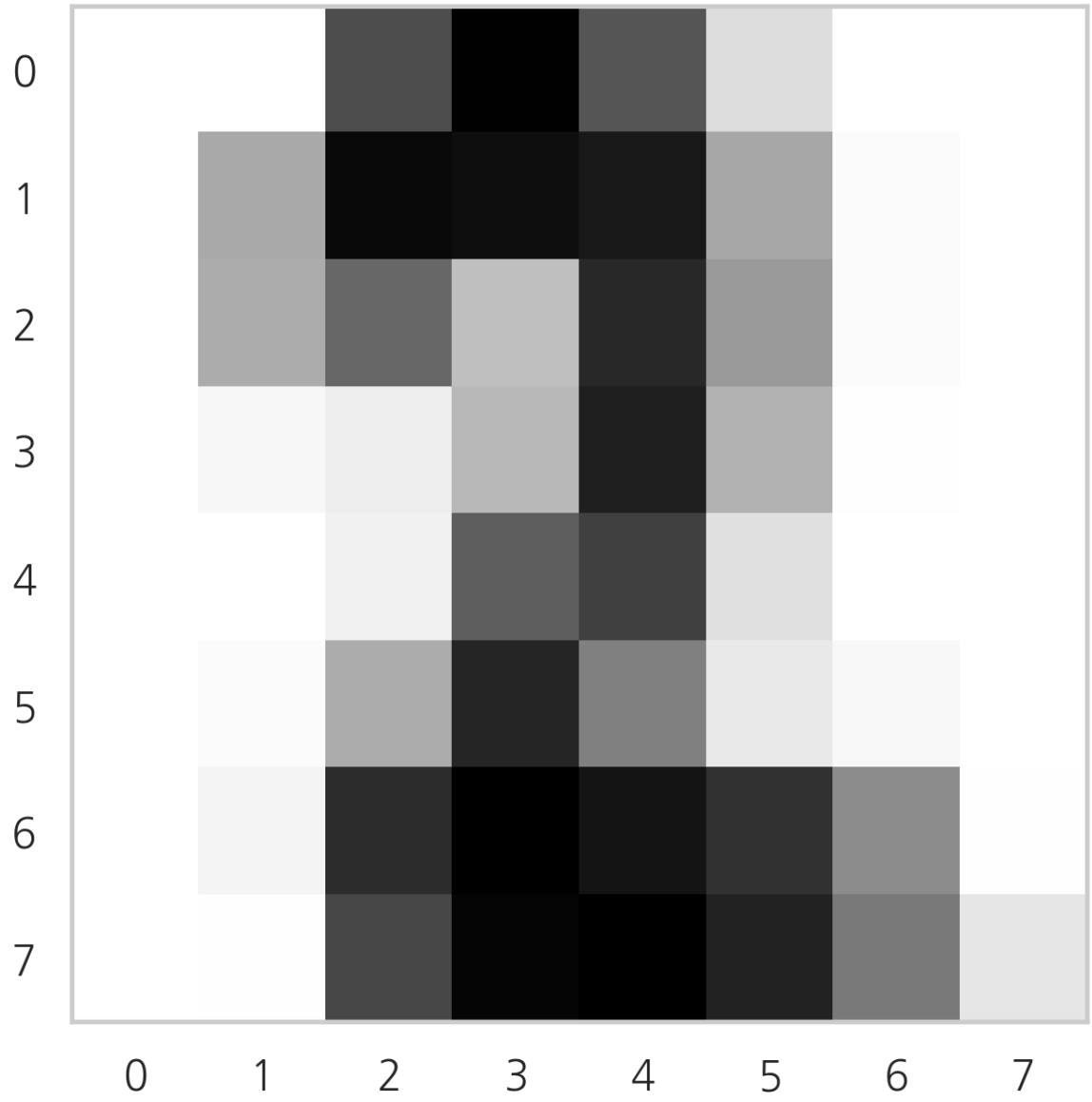
```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       178
           1       0.80      0.80      0.80       182
           2       0.91      0.90      0.91       177
           3       0.93      0.85      0.89       183
           4       0.96      0.94      0.95       181
           5       0.92      0.88      0.90       182
           6       0.97      0.96      0.97       181
           7       0.91      0.99      0.95       179
           8       0.80      0.82      0.81       174
           9       0.80      0.87      0.83       180

    accuracy                           0.90      1797
   macro avg       0.90      0.90      0.90      1797
weighted avg       0.90      0.90      0.90      1797
```

In [39]:

```
plt.imshow(np.exp(model.feature_log_prob_)[2].reshape((8, 8)), cmap=plt.cm.binary)
plt.grid(False)
plt.show()
```



다항분포 나이브베이즈 모형

다항분포 나이브베이즈 모형 클래스 `MultinomialNB` 는 가능도 추정과 관련하여 다음 속성을 가진다.

- `feature_count_` : 각 클래스 k에서 d번째 면이 나온 횟수 $N_{d,k}$
- `feature_log_prob_` : 다항분포의 모수의 로그

$$\log \mu_k = (\log \mu_{1,k}, \ldots, \log \mu_{D,k}) = \left(\log \frac{N_{1,k}}{N_k}, \ldots, \log \frac{N_{D,k}}{N_k} \right)$$

여기에서 N_k은 클래스 k에 대해 주사위를 던진 총 횟수를 뜻한다.

스무딩 공식은

$$\hat{\mu}_{d,k} = \frac{N_{d,k} + \alpha}{N_k + D\alpha}$$

이다.

이번에도 스팸 메일 필터링을 예로 들어보다. 다만 BOW 인코딩을 할 때, 각 키워드가 출현한 빈도를 직접 입력 변수로 사용한다.

In [40]:

```
X = np.array([
    [3, 4, 1, 2],
    [3, 5, 1, 1],
    [3, 3, 0, 4],
    [3, 4, 1, 2],
    [1, 2, 1, 4],
    [0, 0, 5, 3],
    [1, 2, 4, 1],
    [1, 1, 4, 2],
    [0, 1, 2, 5],
    [2, 1, 2, 3]])
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

In [41]:

```
from sklearn.naive_bayes import MultinomialNB
model_mult = MultinomialNB().fit(X, y)
```

사전 확률은 다음과 같이 구한다.

In [42]:

```
model_mult.classes_
```

Out[42]:

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

In [43]:

```
model_mult.class_count_
```

Out[43]:

```
array([4., 6.])
```

In [44]:

```
np.exp(model_mult.class_log_prior_)
```

Out[44]:

```
array([0.4, 0.6])
```

다음으로 각 클래스에 대한 가능도 확률분포를 구한다. 다항분포 모형을 사용하므로 각 클래스틀 4개의 면을 가진 주사위로 생각할 수 있다. 그리고 각 면이 나올 확률은 각 면이 나온 횟수를 주사위를 던진 전체 횟수로 나누면 된다. 우선 각 클래스 별로 각각의 면이 나온 횟수는 다음과 같다.

In [45]:

```
fc = model_mult.feature_count_
fc
```

Out[45]:

```
array([[12., 16.,  3.,  9.],
       [ 5.,  7., 18., 18.]])
```

이 데이터에서 클래스 Y=0인 주사위를 던진 횟수는 첫번째 행의 값의 합인 40이므로 클래스 Y=0인 주사위를 던져 1이라는 면이 나올 확률은 다음처럼 계산할 수 있다.

$$\mu_{1,Y=0} = \frac{12}{40} = 0.3$$

In [46]:

```
fc / np.repeat(fc.sum(axis=1)[:, np.newaxis], 4, axis=1)
```

Out[46]:

```
array([[0.3       , 0.4       , 0.075     , 0.225     ],
       [0.10416667, 0.14583333, 0.375     , 0.375     ]])
```

실제로는 극단적인 추정을 피하기 위해 이 값을 가중치 1인 스무딩을 한 추정값을 사용한다.

In [47]:

```
model_mult.alpha
```

Out[47]:

```
1.0
```

In [48]:

```
(fc + model_mult.alpha) / ₩
    (np.repeat(fc.sum(axis=1)[:, np.newaxis],
               4, axis=1) + model_mult.alpha * X.shape[1])
```

Out[48]:

```
array([[0.29545455, 0.38636364, 0.09090909, 0.22727273],
       [0.11538462, 0.15384615, 0.36538462, 0.36538462]])
```

이렇게 구한 모수 추정치는 다음과 같다.

In [49]:

```
theta = np.exp(model_mult.feature_log_prob_)
theta
```

Out[49]:

```
array([[0.29545455, 0.38636364, 0.09090909, 0.22727273],
       [0.11538462, 0.15384615, 0.36538462, 0.36538462]])
```

이제 이 값을 사용하여 예측을 해 보자. 만약 어떤 메일에 1번부터 4번까지의 키워드가 각각 10번씩 나왔다면 다음처럼 확률을 구할 수 있다. 구해진 확률로부터 이 메일이 스팸임을 알 수 있다.

In [50]:

```
x_new = np.array([10, 10, 10, 10])
model_mult.predict_proba([x_new])
```

Out[50]:

```
array([[0.38848858, 0.61151142]])
```

다항분포의 확률질량함수을 사용하면 다음처럼 직접 확률을 구할 수도 있다.

In [51]:

```
p = (theta ** x_new).prod(axis=1)*np.exp(model_bern.class_log_prior_)
p / p.sum()
```

Out[51]:

```
array([0.38848858, 0.61151142])
```

연습 문제 3

MNIST 숫자 분류문제를 다항분포 나이브베이즈 모형을 사용하여 풀고 이진화(Binarizing)를 하여 베르누이 나이브베이즈 모형을 적용했을 경우와 성능을 비교하라.

In [52]:

```
from sklearn.datasets import load_digits

digits = load_digits()
X2 = digits.data
y2 = digits.target

from sklearn.naive_bayes import GaussianNB

model = GaussianNB().fit(X2, y2)
y2_pred = model.predict(X2)

from sklearn.metrics import classification_report

print(classification_report(y2, y2_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       178
           1       0.83      0.85      0.84       182
           2       0.98      0.64      0.77       177
           3       0.94      0.79      0.86       183
           4       0.98      0.84      0.90       181
           5       0.91      0.93      0.92       182
           6       0.96      0.99      0.98       181
           7       0.72      0.99      0.83       179
           8       0.58      0.86      0.69       174
           9       0.94      0.71      0.81       180

    accuracy                           0.86      1797
   macro avg       0.88      0.86      0.86      1797
weighted avg       0.89      0.86      0.86      1797
```

연습 문제 4

텍스트 분석에서 TF-IDF 인코딩을 하면 단어의 빈도수가 정수가 아닌 실수값이 된다. 이런 경우에도 다항분포 모형을 적용할 수 있는가?

In [53]:

```
X4 = np.array([
    [3.1, 4.0, 1.0, 2.0],
    [3.0, 5.0, 1.0, 1.0],
    [3.0, 3.5, 0.0, 4.0],
    [3.0, 4.0, 1.0, 2.8],
    [1.0, 2.1, 1.0, 4.0],
    [0.0, 0.0, 5.0, 3.0],
    [1.0, 2.0, 4.0, 1.0],
    [1.0, 1.0, 4.1, 2.0],
    [0.0, 1.0, 2.0, 5.0],
    [2.9, 1.0, 2.0, 3.0]])
y4 = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

from sklearn.naive_bayes import MultinomialNB

model_mult4 = MultinomialNB().fit(X4, y4)
y4_pred = model_mult4.predict(X4)

from sklearn.metrics import classification_report

print(classification_report(y4, y4_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

뉴스그룹 분류

다음은 뉴스그룹 데이터에 대해 나이브베이즈 분류모형을 적용한 결과이다.

In [54]:

```
from sklearn.datasets import fetch_20newsgroups

news = fetch_20newsgroups(subset="all")
X = news.data
y = news.target

from sklearn.feature_extraction.text import TfidfVectorizer, HashingVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

model1 = Pipeline([
    ('vect', CountVectorizer()),
    ('model', MultinomialNB()),
])
model2 = Pipeline([
    ('vect', TfidfVectorizer()),
    ('model', MultinomialNB()),
])
model3 = Pipeline([
    ('vect', TfidfVectorizer(stop_words="english")),
    ('model', MultinomialNB()),
])
model4 = Pipeline([
    ('vect', TfidfVectorizer(stop_words="english",
                             token_pattern=r"₩b[a-z0-9_₩-₩.]+[a-z][a-z0-9_₩-₩.]+₩b")),
    ('model', MultinomialNB()),
])
```

In [55]:

```
%%time
from sklearn.model_selection import cross_val_score, KFold

for i, model in enumerate([model1, model2, model3, model4]):
    scores = cross_val_score(model, X, y, cv=5)
    print(("Model{0:d}: Mean score: {1:.3f}").format(i + 1, np.mean(scores)))
```

```
Model1: Mean score: 0.855
Model2: Mean score: 0.856
Model3: Mean score: 0.883
Model4: Mean score: 0.888
CPU times: user 5min 52s, sys: 8.16 s, total: 6min
Wall time: 6min 5s
```

연습 문제 5

(1) 만약 독립변수로 실수 변수, 0 또는 1 값을 가지는 변수, 자연수 값을 가지는 변수가 섞여있다면 사이킷런에서 제공하는 나이브베이즈 클래스를 사용하여 풀 수 있는가?

(2) 사이킷런에서 제공하는 분류문제 예제 중 숲의 수종을 예측하는 covtype 분류문제는 연속확률분포 특징과 베르누이확률분포 특징이 섞여있다. 이 문제를 사이킷런에서 제공하는 나이브베이즈 클래스를 사용하여 풀어라.

In [56]:

```
from sklearn.datasets import fetch_covtype

covtype = fetch_covtype()
X = covtype.data
y = covtype.target
X1 = X[:, :10]
X2 = X[:, 10:]
X1[0], X2[0]
```

Out[56]:

```
(array([2.596e+03, 5.100e+01, 3.000e+00, 2.580e+02, 0.000e+00, 5.100e+02,
        2.210e+02, 2.320e+02, 1.480e+02, 6.279e+03]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]))
```

In [57]:

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB

model1 = GaussianNB().fit(X1, y)
model2 = BernoulliNB().fit(X2, y)

prob1 = model1.predict_proba(X1)
prob2 = model2.predict_proba(X2)

likeli1 = prob1 / model1.class_prior_
likeli2 = prob2 / model1.class_prior_
prob = likeli1 * likeli2 * model1.class_prior_
y_pred = np.argmax(prob, axis=1) + 1
```

In [58]:

```
confusion_matrix(y, y_pred)
```

Out[58]:

```
array([[140348,  53648,    157,      0,   2071,    881,  14735],
       [ 65031, 188638,   5849,     19,  12541,   7752,   3471],
       [     0,   2307,  23675,   3094,   1319,   5359,      0],
       [     0,      0,    551,   1941,      0,    255,      0],
       [   245,   5134,    275,      0,   3513,    326,      0],
       [     0,   1910,   6230,    621,    464,   8142,      0],
       [  6371,    183,      0,      0,     22,      0,  13934]])
```

In [59]:

```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.66      0.66      0.66    211840
           2       0.75      0.67      0.71    283301
           3       0.64      0.66      0.65     35754
           4       0.34      0.71      0.46      2747
           5       0.18      0.37      0.24      9493
           6       0.36      0.47      0.41     17367
           7       0.43      0.68      0.53     20510

    accuracy                           0.65    581012
   macro avg       0.48      0.60      0.52    581012
weighted avg       0.68      0.65      0.66    581012
```

In [60]:

```
model = BernoulliNB().fit(X, y)
y_pred = model.predict(X)
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.65      0.48      0.55    211840
           2       0.65      0.76      0.70    283301
           3       0.60      0.87      0.71     35754
           4       0.55      0.43      0.48      2747
           5       0.22      0.06      0.10      9493
           6       0.24      0.23      0.23     17367
           7       0.63      0.61      0.62     20510

    accuracy                           0.63    581012
   macro avg       0.51      0.49      0.49    581012
weighted avg       0.63      0.63      0.62    581012
```