Naive Bayesian classifier design

Machine learning homework-2

과제기간: 2025.03.28~2025.04.17

담당 교수님: 박철수 교수님

학번: 2021202087

이름: 장현웅

목차

1. 과제 제목 및 목적

1.1. 과제 제목

Naive Bayesian classifier design

1.2. 과제 목적

본 과제의 목적은 펭귄 특징 데이터셋(penguins.csv)을 활용하여 Gaussian Naïve Bayesian Classifier를 설계하고 구현하는 것이다. 해당 데이터셋에는 Adelie, Chinstrap, Gentoo의 세 가지 Species label에 대해, 섬 위치, 부리 길이와 깊이, 날개 길이, 몸무게, 성별 등 총 6개의 feature가 제공된다. 구현된 분류기는 테스트 데이터의 feature 값을 입력 받아, 펭귄들이 어느 클래스에 속할지를 예측한다.

과제는 총 6개의 핵심 Python 함수를 구현하는 것으로 구성되며, 각각의 함수는 다음과 같은 역할을 수행한다.

feature_normalization(): feature 정규화 함수
get_normal_parameter(): 라벨 별 Gaussian Naive Bayes 파라미터 추정
get_prior_probability(): 사전확률 계산
Gaussian_PDF(): Gaussian PDF 값 계산
Gaussian_Log_PDF(): Gaussian Log PDF 값 계산
Gaussian_NB(): (log of) Gaussian Naive Bayes posterior probability 계산.

함수들은 HWJ.py에 저장되어 있으며, main_app.py에서 펭귄 데이터를 분류하고 모델의 성능(정확도)을 출력한다. 이번 과제를 통해 확률적 머신러닝과 베이즈 정리의 개념을 공부하고, 코드 수준에서 직접 구현하고 실험하여 내용을 깊게 이해할 수 있다.

2. Background

2.1. 베이지안 분류기의 핵심 수식과 구현 맥락

머신러닝 분류 문제는 관측된 입력 X = x로부터 감춰진 클래스 Y를 추론하는 작업이다. 베이지안 방법에서는 "관측값 x가 주어졌을 때 Y가 특정 클래스 Y_k일 조건부확률" $P(Y_k|x)$를 계산하고, 이 확률이 가장 큰 클래스를 예측으로 결정한다.

베이즈 정리에 의해:

$$P(Y_k \mid x) = \frac{P(x \mid Y_k)P(Y_k)}{P(x)} \rightarrow P(Y_k \mid x) \propto P(x \mid Y_k)P(Y_k)$$

$$posterior \propto likelihood \cdot prior$$

$P(Y_k)$: prior(사전 확률), $P(x|Y_k)$: likelihood(가능도), $P(Y_k|x)$: posterior(사후확률)

여기서 분포 P(x)는 evidence라 하며 모든 클래스에 대해 공통이므로 실제 계산시에는 likelihood와 prior값의 곱만 비교해도 된다.

2.2. Naïve Bayesian Classifier와 feature간 조건부 독립

Naïve Bayes는 모든 feature가 클래스 Y_k가 주어졌을 때 서로 조건부 독립임을 가정한다.

$$\boldsymbol{x} = (x_1, x_2, x_3, \dots, x_n)\ (feature\ vectors)$$
$$P(x_1, x_2, \dots, x_n|Y_k) = \prod_{i=1}^{n} P(x_i|Y_k)$$
$$multivariant\ likelihood$$
$$= product\ of\ likelihoods\ of\ single\ feature\ variant$$

이렇게 하면 각 feature의 likelihood 확률 분포(예: 가우시안)만 추정해 곱하면 되고, 두 개 이상의 feature 확률변수에 대해서 multivariant likelihood 확률분포를 따로 구할 필요가 없으므로 계산이 매우 단순해진다.

2.3. MLE를 사용한 Gaussian Naïve Bayesian Classifier

나이브 베이지안 분류기는 현실의 규칙 함수 f를 확률 함수 f'(x)로 근사하여,

$$f'(x) = argmax_{Y_k} P(Y_k|x)$$
$$= argmax_{Y_k}\{P(x|Y_k)P(Y_k)\}$$

로 분류 결과를 도출한다. 여기서 사전확률 $P(Y_k)$은 학습 데이터 또는 선험적 지식으로부터 그 분포가 이미 정확하게 결정되었다고 가정하면, 핵심 문제는 관측값 x에 대한 조건부 확률 $P(x|Y_k)$을 어떻게 모델링하느냐가 된다.

2.3.1. 각 feature의 분포 가정

Gaussain Naïve Bayesian Classifier는 독립인 각 feature x_i가 클래스 Y_k에서 가우시안 분포를 따른다고 가정한다. 여기서 parameter $\theta = \{\mu_{k,i}, \sigma_{k,i}^2\}$ 이다.

$$x_i|Y_k \sim N(\mu_{k,i}, \sigma_{k,i}^2)$$
$$P(x_i \mid Y_k, \theta) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} exp(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2})$$

2.3.2. MLE로 최적의 parameter $\hat{\theta} = \{\hat{\mu}_{k,i}, \hat{\sigma}_{k,i}^2\}$ 추정

MLE(maximum likelihood estimation)은 사전 확률 분포가 명확하게 정해져 있을 때, 관측한 dataset(모든 data sample들의 joint event)의 확률을 최대화하는 parameter가 모집단(실제 세계)에서의 분포를 가장 잘 나타내 준다고 생각하고, 이를 구하는 방법이다.

$$\hat{\theta} = \text{argmax}_\theta P(x_i \mid Y_k, \theta)$$

데이터 샘플 간에 서로 독립이고, 각 feature들은 클래스 Y_k 가 주어졌을 때 독립이면, MLE를 통해 구한 각 클래스의 likelihood $P(x_i \mid Y_k, \theta)$의 최적의 parameter를 계산한 것은 각 클래스의 해당 feature에 대한 샘플 평균과 샘플 분산으로 나타난다.

클래스 c에 속한 N_c개의 샘플들에 대해 다음과 같다. ($x^{(j}$는 j번째 샘플(행) 이다)

$$\hat{\mu}_{k,i} = \frac{1}{N_c}\sum_{j=1}^{N_c} x_i^{(j}, \hat{\sigma}_{k,i}^2 = \frac{1}{N_c}\sum_{j=1}^{N_c}\left(x_i^{(j} - \hat{\mu}_{k,i}\right)^2$$

2.3.3. Log posterior

테스트 데이터 x, 클래스 Y_k에 대해 likelihood 값은, x의 각 feature x_i에 대한 likelihood 값의 product이고, 각각은 최적화된 Gaussian PDF에 대입하여 얻어졌다. 하지만 실제 계산에서는 underflow를 방지하기 위해 로그를 취하여 계산할 수 있다.

$$\log(\text{likelihood}) = \log P(x|Y_k) = \sum_{feature\ i} \log P(x_i \mid \hat{\mu}_{k,i}, \hat{\sigma}_{k,i}^2)$$

여기에 사전확률의 로그를 더하면 다음과 같다.

$$\log(posterior) = \log(likelihood) + \log(prior)$$

$$\log P(Y_k|x) \propto \sum_{feature\ i} \log P(x_i \mid \hat{\mu}_{k,i}, \hat{\sigma}_{k,i}^2) + \log\ P(Y_k)$$

Posterior를 대신하는 값으로, 이 값들 간의 대소관계를 비교하여 정답을 판정할 수 있다.

2.3.4. Gaussian Naïve Bayes의 장단점

장점:

- 계산이 빠르고 구현이 간단하다.
- 적은 학습 데이터로도 비교적 잘 작동한다.
- 각 클래스의 분포를 명시적으로 모델링할 수 있다.

단점:

- feature 간 독립 가정이 현실에서는 성립하지 않는 경우가 많다.
- Gaussian 분포를 따르지 않는 feature에 대해 성능 저하 가능성이 있다.

3. 소스 코드에 대한 설명

3.1. Main_app.py

본 프로그램은 Gaussian Naïve Bayesian Classifier를 직접 구현한 것으로, 전체 흐름은 main_app.py를 중심으로 구성되어 있다. 각 단계는 다음과 같은 순서로 작동한다.

(1) 데이터 로딩 및 전처리

- sns.load_dataset('penguins')를 통해 펭귄 데이터를 불러온다.
- 결측값이 포함된 행은 dropna()로 제거한다.
- 'island', 'sex', 'species'는 문자열이므로 .astype('category').cat.codes를 통해 숫자로 인코딩한다.

(2) 데이터 셔플 및 분할

- 데이터셋을 sample(frac=1)로 섞은 후 index를 재설정한다.
- 전체 데이터 중 앞의 250개를 학습 데이터, 나머지 50개를 테스트 데이터로 분리한다 (split_data).

(3) feature 정규화 (Normalization)

- feature_normalization() 함수를 이용하여 각 feature의 평균과 표준편차를 계산하고, 정규화된 데이터를 생성한다.
- 정규화된 데이터는 data = normalled_data로 교체된다.

(4) 학습 파라미터 추정 (Training Phase)

- get_normal_parameter()를 통해 클래스별 평균(mu)과 표준편차(sigma)를 구한다.
- get_prior_probability()를 통해 학습 데이터의 클래스 비율을 계산하여 prior 확률을 얻는다.

(5) Test

- Gaussian_NB() 함수를 통해 log posterior 확률을 계산한다. 이때 Gaussian_Log_PDF를 이용하여 각 feature의 log likelihood를 구하고, log prior를 더해서 log posterior를 구한다. (대소관계가 유지되므로, 그대로 구현)
- classifier()를 통해 log posterior 중 가장 확률이 높은 클래스를 예측값으로 설정한다.

(6) 정확도 측정 및 출력

- accuracy() 함수로 예측값과 정답을 비교하여 정확도와 맞춘 개수를 계산한다.
- print() 문으로 결과를 출력한다. 모델 성능이 확인되는 핵심 지표다.

각 함수는 HWJ.py에 정의되어 있으며, main_app.py에서는 이 함수들을 순차적으로 호출한다.

3.2. HWJ.py

3.2.1. 구현 함수

Main_app.py에서 쓰이는 각 함수에 대해 다음과 같이 구현하였다.

feature_normalization(): 입력된 데이터의 각 feature가 평균이 0, 표준편차가 1이 되도록 정규화한다. 먼저 각 feature의 평균과 표준편차를 계산한 뒤, 각 값에서 평균을 빼고 표준편차로 나눈다. 표준편차가 너무 작은 경우는 계산이 불안정해지므로, 최소값 1e-3으로 보정하였다.

get_normal_parameter(): MLE(Maximum Likelihood Estimation)에 따라 각 클래스에 대해 feature별 평균(mu)과 표준편차(sigma)를 계산한다. 각 클래스에 해당하는 데이터 인덱스를 numpy.where()로 찾은 뒤, 그에 해당하는 데이터를 모아 np.mean(), np.std()로 계산하였다. 추정된 mu와 sigma는 Gaussian_Log_PDF() 계산에 사용된다.

get_prior_probability(): 학습 데이터에서 각 클래스가 등장하는 비율을 계산하여 prior 확률을 추정한다. 전체 데이터 개수 대비 각 클래스의 샘플 수 비율을 계산함으로써, P(Y=k)의 값을 얻는다. 이는 나중의 posterior 계산에서 로그를 취해 더해진다.

Gaussian_Log_PDF(): 주어진 feature값이 특정 클래스에 속할 확률(=likelihood)을 계산하기 위한 정규분포 확률밀도함수의 로그값을 구한다. 로그로 계산하는 이유는, 여러 feature의 likelihood를 곱하게 되면 underflow가 발생할 수 있기 때문에, 로그를 취해서 합의 형태를 취함으로써 안정적인 연산을 할 수 있기 때문이다. 수식은 $\log(\text{pdf}) = -0.5 \log(2\pi\sigma^2) - (x - \mu)^2 / (2\sigma^2)$ 이다.

Gaussian_NB(): 각 테스트 데이터에 대해 클래스별 log posterior 확률을 계산한다. 이때, Gaussian_Log_PDF를 사용하여 각 feature의 log-likelihood를 모두 더하고, 여기에 log-prior를 더하여 log-posterior를 얻는다. 이를 데이터 개수 x 클래스 수 만큼 반복하여 2차원 log posterior 배열을 반환한다.

3.2.2. 미리 구현되어있는 함수

미리 구현되어있는 함수에 대해서는 다음과 같이 구조를 설명하였다.

split_data(): 전체 데이터를 학습용(training)과 평가용(test)으로 분리한다. split_factor 값을 기준으로 데이터를 나누며, feature와 label을 동일한 순서로 분할하여 각각 반환한다.

classifier(): posterior 배열에서 가장 큰 확률값을 가진 클래스의 인덱스를 argmax()로 찾아 예측값으로 설정한다. 전체 test 데이터에 대해 예측을 수행하여 numpy array 형태로 반환한다.

accuracy(): 예측값(prediction)과 실제 정답 라벨(test_label)을 비교하여 맞춘 개수(hit_num)를 구하고, 이를 전체 데이터 수로 나눈 비율을 정확도(accuracy)로 계산한다. 최종 결과는 정확도(%)와 맞춘 개수로 출력된다.

4. Sudo code

4.1. main_app.py

```
Load penguins dataset
Drop missing values
Encode categorical features as numerical
Shuffle and split dataset (train/test)
Normalize features
Estimate mu, sigma, prior from training set
For each test sample:
    - For each class:
        - Compute log likelihood from Gaussian_Log_PDF
        - Add log prior
        - Get posterior
    - Choose class with highest posterior
Compare predictions with ground truth
Print accuracy and hit count
```

4.2. HWJ.py (utill.py)

구현 대상 함수에 대해서만 sudo code를 작성한다.

```
feature_normalization(data):
    # data: N×D numpy array
    mu = array of length D
    std = array of length D
    normal_feature = N×D array
    for i in 0 to D-1:
        mu[i] = mean(data[:, i])
        std[i] = max(std(data[:, i]), 1e-3)
        for j in 0 to N-1:
            normal_feature[j, i] = (data[j, i] - mu[i]) / std[i]
    return normal_feature

get_normal_parameter(data, label, label_num):
    # data: N×D, label: N, label_num: number of classes K
    mu = K×D array
    sigma = K×D array
    for k in 0 to K-1:
        indices = positions where label == k
        class_data = data[indices, :]
        mu[k, :] = mean(class_data, axis=0)
        sigma[k, :] = std(class_data, axis=0)
    return mu, sigma
```

```
get_prior_probability(label, label_num):
    # label: N, label_num: K
    prior = array of length K
    N = length(label)
    for k in 0 to K-1:
        prior[k] = count(label == k) / N
    return prior

Gaussian_PDF(x, mu, sigma):
    # x, mu, sigma: arrays of length D
    sigma = max(sigma, 1e-3)
    pdf = 1 / (sqrt(2*pi) * sigma) * exp(-(x - mu)^2 / (2 * sigma^2))
    return pdf

Gaussian_Log_PDF(x, mu, sigma):
    # x, mu, sigma: arrays of length D
    sigma = max(sigma, 1e-3)
    log_pdf = -0.5 * log(2*pi) - log(sigma) - (x - mu)^2 / (2 * sigma^2)
    return log_pdf

Gaussian_NB(mu, sigma, prior, data):
    # mu, sigma: K×D, prior: length K, data: N×D
    N = number of rows in data
    K = number of classes
    posterior = N×K array
    for i in 0 to N-1:
        for k in 0 to K-1:
            # sum log-likelihoods over features
            ll = sum(Gaussian_Log_PDF(data[i], mu[k], sigma[k]))
            posterior[i, k] = ll + log(prior[k])
    return posterior
```

5. 실행 결과

5.1. 실행 결과 캡쳐 첨부

20번의 실행결과를 캡쳐했으며, mean 데이터는 동일하게 나오기에 10번 이후 생략했다.

```
(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [-1.36694126e-17  3.68740740e-16  5.09393995e-16  1.82703369e-16
 -2.80056258e-17  2.16043399e-16]
accuracy is 85.54216867469879% !
the number of correct prediction is 71 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 2.33380215e-17  3.66073538e-16  5.57632852e-16  1.56031344e-16
 -1.13356105e-16  8.26832763e-17]
accuracy is 91.56626506024097% !
the number of correct prediction is 76 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [-5.00100462e-19  3.74741946e-16  6.04788158e-16  1.79106292e-16
 -1.28734194e-16  1.85703971e-16]
accuracy is 92.77108433734940% !
the number of correct prediction is 77 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 1.92205277e-16  3.76075547e-16  5.19396004e-16  1.61365749e-16
 -5.36774495e-17 -1.33360123e-17]
accuracy is 89.15662650602410% !
the number of correct prediction is 74 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 8.40168775e-17  3.56071529e-16  5.35461732e-16  2.01373786e-16
 -1.00020092e-16 -2.08041792e-16]
accuracy is 93.97590361445783% !
the number of correct prediction is 78 of 83 !
```

```
명령 프롬프트

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 8.26832763e-17  3.64739937e-16  5.76865882e-16  1.71367758e-16
 -7.60152702e-17  1.16690108e-16]
accuracy is 90.36144578313254% !
the number of correct prediction is 75 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [-2.17377001e-16  3.73408345e-16  5.29439689e-16  1.77368964e-16
 -6.60132609e-17  2.13376197e-16]
accuracy is 87.95180722891565% !
the number of correct prediction is 73 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 9.56858883e-17  3.48069921e-16  5.47610005e-16  1.56698145e-16
 -1.09688701e-16 -1.52030540e-16]
accuracy is 85.54216867469879% !
the number of correct prediction is 71 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [-4.30086397e-17  3.74075145e-16 -1.22657973e-15  1.71367758e-16
 -7.16810662e-17  2.28712611e-16]
accuracy is 89.15662650602410% !
the number of correct prediction is 74 of 83 !

(GaussianNB) C:\Users\prosp\GaussianNB>main_app.py
Mean: [6.51651652e-01 4.39927928e+01 1.71648649e+01 2.00966967e+02
 4.20705706e+03 5.04504505e-01]
normalled_Mean: [ 4.50090415e-17  3.47403121e-16 -1.26208686e-15  1.47696336e-16
 -8.26832763e-17  8.80176812e-17]
accuracy is 87.95180722891565% !
the number of correct prediction is 73 of 83 !
```

```
accuracy is 97.590361445783l3% !
the number of correct prediction is 81 of 83 !
```

```
accuracy is 87.95180722891565% !
the number of correct prediction is 73 of 83 !
```

```
accuracy is 90.361445783132540% !
the number of correct prediction is 75 of 83 !
```

```
accuracy is 90.361445783132540% !
the number of correct prediction is 75 of 83 !
```

```
accuracy is 90.361445783132540% !
the number of correct prediction is 75 of 83 !
```

```
accuracy is 91.566265060240970% !
the number of correct prediction is 76 of 83 !
```

```
accuracy is 90.361445783132540% !
the number of correct prediction is 75 of 83 !
```

```
accuracy is 90.361445783132540% !
the number of correct prediction is 75 of 83 !
```

```
accuracy is 84.337349397590370% !
the number of correct prediction is 70 of 83 !
```

```
accuracy is 89.156626506024100% !
the number of correct prediction is 74 of 83 !
```

5.2. 실험 결과 정리

위의 실험 결과를 포함한 20번의 수치를 정리했다. 소수점 4자리에서 반올림했으며 전체 평균 정확도는 89.82%, 평균 맞춘 개수는 74.6 / 83 이다.

Acc	Hit	Acc	Hit
85.542	71	97.590	81
91.566	76	87.952	73
92.771	77	90.361	75
89.157	74	90.361	75
93.976	78	90.361	75
90.361	75	91.566	76
87.952	73	90.361	75
85.542	71	90.361	75
89.157	74	84.337	70
87.952	73	89.157	74

5.3. 실험 결과 해석

총 20회에 걸쳐 Gaussian Naïve Bayesian Classifier를 실행한 결과, 평균 정확도는 약 89.82%, 평균 맞춘 개수는 약 74.6개 / 83개로 나타났다. 정확도는 매 시행마다 달랐지만 84.34%에서 최대 97.59%까지의 분포를 보이며 안정적이었다.

데이터가 무작위로 shuffle되고 일정 수(83개)의 테스트 데이터를 기반으로 예측했기 때문에 각 실행마다 결과에 약간의 변동이 있었던 것으로 해석할 수 있다

평균적으로 정확도가 90% 정도를 보인다는 점은 데이터가 Gaussian 분포를 따른다는 가정이 어느 정도 현실을 잘 반영하고 있으며, feature 간 조건부 독립이라는 Naïve 가정이 실제 데이터에서 큰 문제없이 적용될 수 있었음을 보여준다.

그러나, 정확도가 84% 수준까지 내려간 경우도 있었으므로, 추후 모델 개선을 위해 분포 가정 수정, 또는 MAP 방식과 같은 추가 기법을 활용할 수 있다.

전체적으로, 본 과제에서 구현한 Gaussian Naïve Bayesian Classifier는 단순한 구조임에도 불구하고 실용적인 분류 성능을 보였다.

6. 더 알아볼 것

6.1. 다른 Distribution에 대해서 확장하기

지금까지 Gaussian Naïve Bayes Classification을 진행해보았으니, Parametric Estimation 시 Likelihood의 개형이 Gaussian distribution이 아니라 Poisson distribution 등인 경우를 생각해볼 수 있다.

6.2. Non Parametric 방법으로 Likelihood Estimation하기

혹은, 히스토그램으로부터 직접 Likelihood의 개형을 찾거나 K-NNR 등을 사용하면, 정해진 개형 없이 각 Class의 Likelihood function을 추정해볼 수 있다. Implement 후 성능을 비교해보면서, 강의시간에 배우지 않은 내용들에 대해 이해해볼 수 있을 것이다.

7. 참고 문헌

7.1.1. 강의 및 교육자료

- 박철수 교수님, *Bio Computing & Machine Learning Lab*, 강의 슬라이드
  - **Ch2. Basic Probability Theory**
  - **Ch3. Random Process**
  - **Ch4. Likelihood Ratio Test**

7.1.2. 이론 및 알고리즘 참고

- Wikipedia contributors, "Bayesian probability", *Wikipedia, The Free Encyclopedia*
- IBM Cloud Education, "What is a Naive Bayes classifier?"
- IBM Documentation, "Parameter estimates in parametric models"

7.1.3. 블로그 및 응용 해설

- Simon's Research Center, "Bayes' Theorem: 베이즈 정리의 기초 개념"
- 공돌이의 수학정리노트, "나이브 베이즈 분류기"
- 귀퉁이 서재, "머신러닝 -1. 나이브 베이즈 분류"
- 초짜공대생의 공부노트, "조건부 독립"

7.1.4. 실습 코드 구현 참고

- Scikit-learn Documentation, "Naive Bayes classifiers"
- Wikidocs, "Python pandas DataFrame 기초"