나이브 베이지안 분류기

- **01** 베이즈 정리의 이해
- **02** 베이즈 분류기 구현하기
- **03** 나이브 베이지안 분류기 구현하기
- **04** 20newsgroup으로 분류 연습하기

학습목표

1. 베이즈 정리에 대해 이해한다.
2. 베이즈 분류기를 만들고 코드로 나타낸다.
3. 나이브 베이지안 분류기를 구현한다.
4. 20newsgroup 데이터셋으로 분류를 실습한다

01
베이즈 정리의 이해

01 베이즈 정리의 이해

1. 확률의 표현

- 이산형 값의 확률

$$P(X) = \frac{count(Event_x)}{count(Event_{allevent})}$$

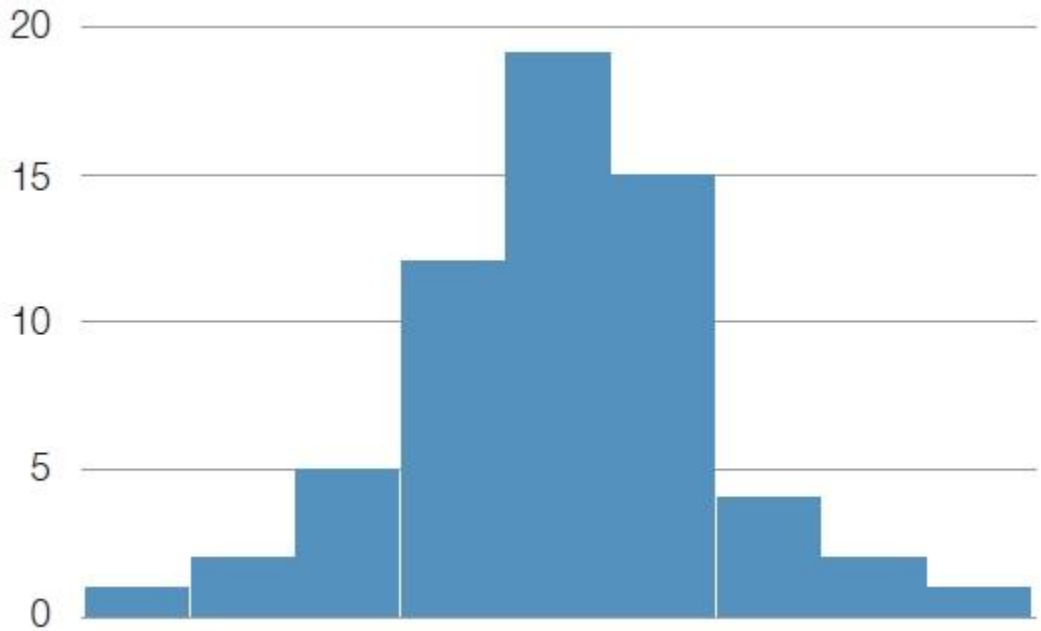

그림 11-1 이산형 값에 대한 분포를 표현한 도수분포표

01 베이즈 정리의 이해

- 연속형 값의 확률 : 해당 데이터를 적절히 표현하는 함수를 생성한 후 해당 함수의 적분 값을 취한다
  - 특정 위치의 값은 없고 적분 값을 취해 구간의 확률을 계산

$$P(-\infty < x < \infty) \int_{-\infty}^{\infty} f(x)dx = 1$$

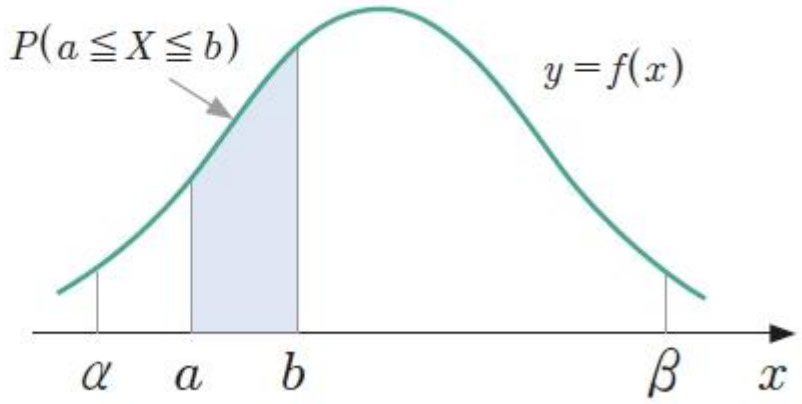

그림 11-2 연속형 값에 대한 확률 분포

01 베이즈 정리의 이해

1.1 확률의 기본 성질

- 확률은 모든 사건에 대해 반드시 0에서 1사이의 값을 가짐

$$0 \le P(X) \le 1$$

- 각 사건들이 서로 관계가 없는 경우, 즉 각 사건들이 일어날 확률이 다른 사건이 일어날 확률에 영향을 미치지 않을 때 각 사건들이 '독립'되었다고 정의

$$P(S) = \sum_{i=1}^{N} P(E_i) = 1$$

01 베이즈 정리의 이해

1.2 조건부 확률

- 조건부 확률(conditional probability) : 어떤 사건이 일어난다고 가정했을 때 다른 사건이 일어날 확률
- P(A|B) : B라는 사건이 발생했을 때 A와 B 사건의 교집합이 발생할 확률

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

  - $A = \{x | x\text{는 홀수}\}, \quad B = \{x | x\text{는 4보다 작은 수}\}$
    해당 사건의 (A|B)는, B의 상황에서 A를 만족하는 값을 찾으면 되므로 {1, 3}

01 베이즈 정리의 이해

2. 베이즈 정리

- 베이즈 정리(Bayes' theorem) : 두 확률 변수의 사전확률과 사후확률 사이의 관계를 나타내는 정리
- 베이즈 정리의 전제 : 객관적인 확률이 존재하지 않고 이전 사건으로 인해 확률이 지속적으로 업데이트된다

그림 11-3 3장의 트럼프카드

01 베이즈 정리의 이해

- 베이즈주의자는 실제로 뒤집어 카드가 나온 확률을 기반으로 실행할 때마다 계속하여 확률들을 업데이트
- 빈도주의자는 다음 확률을 계속하여 실행할 경우 $\frac{1}{3}$에 수렴할 것이므로 $\frac{1}{3}$로 간주

$\frac{3}{30}$ $\frac{8}{30}$ $\frac{19}{30}$ ← 카드뒤집기 게임의 실제 확률

그림 11-4 카드뒤집기 게임의 실제 확률(베이즈주의자)

01 베이즈 정리의 이해

- H는 가설, D는 데이터일 때 P(H|D)는 사후확률

$$P(H|D) = \frac{P(H)P(D|H)}{P(D)}$$

- 사후확률 : 데이터가 주어졌을 때 해당 가설이 맞는지에 대한 확률
- 가능도(likelihood) : 어떤 사건이 발생했을 때 다음 사건이 발생할 수 있는 모든 확률의 발생가능한 정도를 확률로 나타냄
  - P(D|H) : 가설이 주어졌을 때 해당 데이터가 존재할 확률

01 베이즈 정리의 이해

베이스 정리 문제

크기와 색깔이 다른 13개의 공 중 1개의 공을 무작위로 뽑을 때, 뽑은 공이 큰 공이었다면 이 공이 검은색일 확률은 얼마일까?

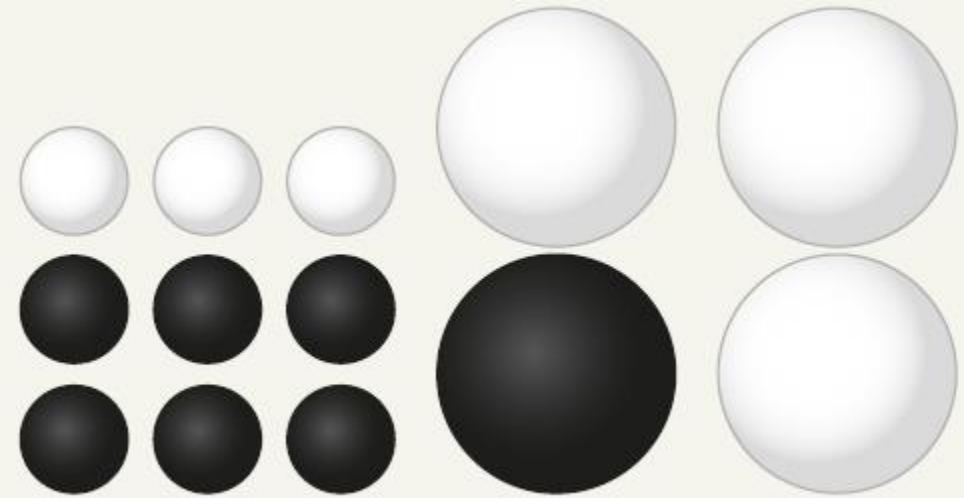

그림 11-5 공 뽑기 문제

01 베이즈 정리의 이해

- 큰 공이 나올 확률은 P(BIG), 작은 공이 나올 확률은 P(SMALL), 검은색 공이 나올 확률은 P(BLACK), 흰색 공이 나올 확률은 P(WHITE)
- 큰 공을 뽑을 때 검은색일 확률은 P(BLACK | BIG)

$$P(BLACK \mid BIG) = \frac{P(BLACK)P(BIG \mid BLACK)}{P(BIG)}$$

$$\frac{\frac{7}{13} \times \frac{1}{4}}{\frac{4}{13}} = \frac{7}{16}$$

02
베이즈 분류기 구현하기

02 베이즈 분류기 구현하기

1. 베이즈 분류기 만들기

- 메일에 비아그라(viagra)라는 단어가 들어가면 어느 정도의 확률로 스팸메일인지 판단하는 베이즈 분류기

$$P(spam \mid viagra) = \frac{P(spam)P(viagra \mid spam)}{P(viagra)}$$

02 베이즈 분류기 구현하기

표 11-1 비아그라와 스팸메일을 나타내는 임의의 데이터

number	viagra	spam	number	viagra	spam
1	1	1	11	1	0
2	0	0	12	0	0
3	0	0	13	0	0
4	0	0	14	1	0
5	0	0	15	0	0
6	0	0	16	0	0
7	0	1	17	0	0
8	0	0	18	0	1
9	1	1	19	0	1
10	1	0	20	1	1

$$P(spam \mid viagra) = \frac{\frac{6}{20} \times \frac{3}{6}}{\frac{6}{20}}$$

02 베이즈 분류기 구현하기

2. 코드로 표현하기

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

viagra_spam = {'viagra':
           [1,0,0,0,0,0,0,0,1,1,1,0,0,1,0,0,0,0,0,1],
               'spam':
           [1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,1,1]}
df = pd.DataFrame(viagra_spam, columns = ['viagra',
'spam'])
np_data = df.values
```

- $P(viagra \cap spam)$

In [2]:
```
sum((np_data[:, 0] == 1) & (np_data[:, 1] == 1)) / 20
```

Out [2]:
```
0.15
```

02 베이즈 분류기 구현하기

- $P(viagra)$, $P(spam)$, $P(viagra \cap spam)$, $P(\sim viagra \cap spam)$

In [3]:	p_viagra = sum(np_data[:, 0] == 1) / len(np_data) p_viagra
Out [3]:	0.3
In [4]:	p_spam = sum(np_data[:, 1] == 1) / len(np_data) p_spam
Out [4]:	0.3
In [5]:	p_v_cap_s = sum((np_data[:, 0] == 1) & (np_data[:, 1] == 1)) / len(np_data) p_v_cap_s
Out [5]:	0.15
In [6]:	p_n_v_cap_s = sum((np_data[:, 0] == 0) & (np_data[:, 1] == 1)) / len(np_data) p_n_v_cap_s
Out [6]:	0.15

02 베이즈 분류기 구현하기

- $P(spam \mid viagra)$

```
In [7]: p_spam * (p_v_cap_s / p_spam ) / p_viagra
Out [7]: 0.5
```

- $P(spam \mid \sim viagra)$

```
In [8]: p_spam * (p_n_v_cap_s / p_spam ) / (1-p_viagra)
Out [8]: 0.21428571428571143
```

02 베이즈 분류기 구현하기

- ‘viagra’라는 단어가 포함되었을 때 스팸메일일 확률(=0.5)은 ‘viagra’라는 단어가 포함되지 않았을 때 스팸메일일 확률(=0.2142857142857143)보다 높음
- ‘viagra’라는 단어가 있으면 스팸메일로 분류하는 것이 합리적
  - ‘viagra’라는 단어 외에 영향을 주는 단어가 있을 수 있다.
  - 오히려 스팸에서 제외되는 메일에 ‘viagra’라는 단어가 있을 수 있다.
- 나이브 베이지안 분류기가 위 문제점을 해결

03
나이브 베이지안 분류기 구현하기

1. 나이브 베이지안 분류기 만들기

- 나이브 베이지안 분류기(Naive Bayesian Classifier) : 여러 개의 열을 사용하여 분류기를 구성

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

data_url = "c:/source/ch11/fraud.csv"
df= pd.read_csv(data_url, sep=',')
df.head()
```

Out [1]:

	ID	History	CoApplicant	Accommodation	Fraud
0	1	current	none	own	True
1	2	paid	none	own	False
2	3	paid	none	own	False
3	4	paid	guarantor	rent	True
4	5	arrears	none	own	False

03 나이브 베이지안 분류기 구현하기

- Y 값을 따로 빼내고 X 데이터들을 원핫인코딩으로 처리

In [2]:
```
del df["ID"]
Y_data = df.pop("Fraud")
Y_data = Y_data.values
x_df = pd.get_dummies(df)
x_df.head(10).T
```

Out [2]:

	0	1	2	3	4	5	6	7	8	9
History_arrears	0	0	0	0	1	1	0	1	0	0
History_current	1	0	0	0	0	0	1	0	1	0
History_none	0	0	0	0	0	0	0	0	0	1
History_paid	0	1	1	1	0	0	0	0	0	0
CoApplicant_coapplicant	0	0	0	0	0	0	0	0	0	0
CoApplicant_guarantor	0	0	0	1	0	0	0	0	0	0
CoApplicant_none	1	1	1	0	1	1	1	1	1	1
Accommodation_free	0	0	0	0	0	0	0	0	0	0
Accommodation_own	1	1	1	0	1	1	1	1	0	1
Accommodation_rent	0	0	0	1	0	0	0	0	1	0

03 나이브 베이지안 분류기 구현하기

In [3]:
```
x_data = x_df.values
x_data
```

Out [3]:
```
array([[0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 1, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 0, 1],
       [0, 0, 1, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 0, 1],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
       [1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 1, 0, 1, 0]], dtype=uint8)
```

03 나이브 베이지안 분류기 구현하기

$$P(Y_c|X_1,\cdots,X_n)=\frac{P(Y_c)\prod_{i=1}^{n}P(X_i|Y_c)}{\prod_{i=1}^{n}P(X_i)} \qquad Y_c \; is \; a \; label$$

- Y 값이 True인 경우와 False인 경우

In [4]:	P_Y_True = sum(Y_data==True) / len(Y_data) P_Y_False = 1 - P_Y_True P_Y_True,P_Y_False
Out [4]:	(0.3, 0.7)

03 나이브 베이지안 분류기 구현하기

- $P(Y_{True})$ 와 $P(Y_{False})$ 의 인덱스 값 정리

In [5]:	`np.where(Y_data)`
Out [5]:	`(array([ 0, 3, 5, 9, 11, 12], dtype=int64),)`

In [6]:	`ix_Y_True = np.where(Y_data)`
	`ix_Y_False = np.where(Y_data==False)`
	
	`ix_Y_True, ix_Y_False`

Out [6]:	`((array([ 0, 3, 5, 9, 11, 12], dtype=int64),),`
	` (array([ 1, 2, 4, 6, 7, 8, 10, 13, 14, 15, 16, 17, 18, 19],`
	`dtype=int64),))`

03 나이브 베이지안 분류기 구현하기

$$\log\left\{P(Y_c)\prod_{i=1}^{n}P(X_i|Y_c)\right\} = \log P(Y_c) + \sum_{i=1}^{n}\log P(X_i|Y_c)$$

- $P(X_i|Y_{True})$

In [7]:
```
p_x_y_true = (x_data[ix_Y_True].sum(axis=0)) /
sum(Y_data==True)
p_x_y_false = (x_data[ix_Y_False].sum(axis=0)) /
sum(Y_data==False)

p_x_y_true, p_x_y_false
```

Out [7]:
```
(array([0.16666667, 0.5       , 0.16666667,
        0.16666667, 0.        , 0.16666667,
        0.83333333, 0.        , 0.66666667,
        0.33333333]),
 array([0.42857143, 0.28571429, 0.        ,
        0.28571429, 0.14285714, 0.        ,
        0.85714286, 0.07142857, 0.78571429,
        0.14285714]))
```

03 나이브 베이지안 분류기 구현하기

In [8]:	x_test = [0,1,0,0,0,1,0, 0,1,0] p_y_true_test = P_Y_True + p_x_y_true.dot(x_test) p_y_false_test = P_Y_False + p_x_y_false.dot(x_test) p_y_true_test , p_y_false_test
Out [8]:	(1.6333333333333333, 1.7714285714285714)
In [9]:	p_y_true_test < p_y_false_test
Out [9]:	True

2. 사이킷런을 활용한 나이브 베이지안 분류기

- 하나의 문장이 있을 때 이 문장을 sports와 not sports로 나누는 분류기 만들기
  - 사이킷런의 클래스를 사용

In [10]:
```
y_example_text = ["Sports","Not sports","Sports",
                  "Sports","Not sports"]
y_example = [1 if c=="Sports" else 0 for c in
             y_example_text ]
text_example = ["A great game game",
                "The The election was over",
                "Very clean game match",
                "A clean but forgettable game game",
                "It was a close election", ]
```

- BoW(Bag of Words) : 단어별로 인덱스가 부여되어 있을 때 한 문장 또는 한 문서에 대한 벡터를 표현하는 기법
  - 하나의 단어를 벡터화시킬 때는 원핫인코딩 기법을 사용

그림 11-6 원핫인코딩 예시(BoW)

03 나이브 베이지안 분류기 구현하기

- 전체 문서에 있는 모든 단어들에 이미 인덱스가 부여되어 있고 출현한 단어에 대해서만 단어의 개수를 벡터로 표현

표 11-2 BoW 기법으로 표현된 데이터

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	1	0	0	0	0	0	1	0	1

03 나이브 베이지안 분류기 구현하기

In [11]:
```
from sklearn.feature_extraction.text import
CountVectorizer

countvect_example = CountVectorizer()
X_example = countvect_example.fit_transform(text_example)
countvect_example.get_feature_names()
```

Out [11]:
```
['but',
 'clean',
 'close',
 'election',
 'forgettable',
 'game',
 'great',
 'it',
 'match',
 'over',
 'the',
 'very',
 'was']
```

03 나이브 베이지안 분류기 구현하기

In [12]:
```
countvect_example.transform(text_example).toarray()
```

Out [12]:
```
array([[0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 2, 0, 1],
       [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
       [1, 1, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]],
      dtype=int64)
```

In [13]:
```
countvect_example.vocabulary_
```

Out [13]:
```
{'great': 6,
 'game': 5,
 'the': 10,
 'election': 3,
 'was': 12,
 'over': 9,
 'very': 11,
 'clean': 1,
 'match': 8,
 'but': 0,
 'forgettable': 4,
 'it': 7,
 'close': 2}
```

2.1 베르누이 나이브 베이지안 분류기

- 베르누이 나이브 베이지안 분류기(BernoulliNB) : 다루고자 하는 모든 데이터가 불린 피쳐
- 사용되는 데이터 타입은 이산형 데이터인데, 이러한 데이터를 모두 불린 타입으로 변경하여 학습
  - 정수 타입 숫자라면 임계값 기준으로 True 또는 False로 변환

In [14]:	`from sklearn.naive_bayes import BernoulliNB`
	
	`clf = BernoulliNB(alpha=1, binarize=0)`
	`clf.fit(X_example, y_example)`

Out [14]:	`BernoulliNB(alpha=1, binarize=0)`

- class_log_prior_ 는 각 클래스마다 prior의 값에 log를 붙여서 값을 출력

In [15]:	clf.class_log_prior_
Out [15]:	array([-0.91629073, -0.51082562])

2.2 다항 나이브 베이지안 분류기

- 다항 나이브 베이지안 분류기(MultinomialNB) : 베르누이 분류기와 달리 각 피쳐들이 이산형이지만, 이진값이 아닌 여러 개의 값을 가질 수 있다

- 나이브 베이지안 식을 변형하여 사용

$$P(Y_c|X_1,\cdots,X_n)=\frac{P(Y_c)\prod_{i=1}^{n}P(X_i|Y_c)}{\prod_{i=1}^{n}P(X_i)} \qquad Y_c \; is \; a \; label$$

- 가능도

$$P(X_i|Y_c)=\frac{\sum tf(x_i,d\in Y_c)+\alpha}{\sum N_d\in Y_c+\alpha V}$$

03 나이브 베이지안 분류기 구현하기

표 11-3 다항 분류기를 이용한 확률 연산을 위한 데이터

데이터	문서	단어	클래스
훈련 데이터	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
테스트 데이터	5	Chinese Chinese Chinese Tokyo Japan	?

- $P(Chinese|c)$ $\frac{5+1}{8+6} = \frac{6}{14} = \frac{3}{7}$

```
In [16]: from sklearn.naive_bayes import MultinomialNB

         clf = MultinomialNB(alpha=1)
         clf.fit(X_example, y_example)
```

```
Out [16]: MultinomialNB(alpha=1)
```

2.3 가우시안 나이브 베이지안 분류기

- 가우시안 나이브 베이지안 분류기(GaussianNB) : 연속형 값을 피쳐로 가진 데이터의 확률을 구하기 위해 y의 분포를 정규분포(gaussian)로 가정
- 확률밀도 함수 상의 해당 값 x가 나올 확률로 나이브 베이지안(NB)을 구현
- 가능도

$$P(x_i \mid Y_c) = \frac{1}{\sqrt{2\pi\sigma_{Y_c}^2}} exp\left(-\frac{(x_i - \mu Y_e)^2}{2\sigma_{Y_c}^2}\right)$$

03 나이브 베이지안 분류기 구현하기

표 11-4 키, 몸무게, 발 사이즈의 연속형 데이터

성별(gender)	키(height)	몸무게(weight)	발 사이즈(foot size)
남자(male)	6 ft	180 lbs	12 inches
남자(male)	5.92 ft	190 lbs	11 inches
남자(male)	5.58 ft	170 lbs	12 inches
남자(male)	5.92 ft	165 lbs	10 inches
여자(female)	5 ft	100 lbs	6 inches
여자(female)	5.5 ft	150 lbs	8 inches
여자(female)	5.42 ft	130 lbs	7 inches
여자(female)	5.75 ft	150 lbs	9 inches

03 나이브 베이지안 분류기 구현하기

표 11-5 [표 11-4] 데이터의 평균과 분산

성별 (gender)	평균 (height)	분산 (height)	평균 (weight)	분산 (weight)	평균 (foot size)	분산 (foot size)
male	5.855	3.5033×10-02	176.25	1.2292×10+02	11.25	9.1667×10-01
female	5.4175	9.7225×10-02	132.5	5.5833×10+02	7.5	1.6667

- 새로운 값이 입력되면 다음 수식과 같이 성별을 구분

$$postertior(male) = \frac{P(male)p(height|male)p(weight|male)p(foot\ size|male)}{evidence}$$

$$postertior(female) = \frac{P(female)p(height|female)p(weight|female)p(foot\ size|female)}{evidence}$$

In [17]:
```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(X_example.toarray(), y_example)
```

Out [17]:
```
GaussianNB()
```

04
20newsgroup으로 분류 연습하기

1. 20newsgroup 데이터셋 소개

- 20개의 뉴스 텍스트 데이터를 주제별로 분류하는 문제
- 사이킷런에서 제공하는 20newsgroup 데이터셋과 나이브 베이지안 분류기를 사용

2. 20newsgroup 데이터셋 불러오기

- 모듈을 호출하고 20newsgroup 데이터셋을 다운로드

In [1]:	`import pandas as pd`
	`import numpy as np`
	`import matplotlib.pyplot as plt`
	`import seaborn as sns`
	`import re`
	`%matplotlib inline`

In [2]:	`from sklearn.datasets import fetch_20newsgroups`
	`news = fetch_20newsgroups(subset='all')`
	`news.keys()`

Out [2]:	`dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])`

딕셔너리 타입(dict type)의 의미

일반적으로 모듈에서 제공되는 데이터셋은 모두 비슷한 형태의 딕셔너리 타입을 제공한다. 출력된 결과값의 의미는 다음과 같다.

- data : 실제 데이터
- filenames : 다운로드된 데이터의 파일 위치
- target_names : 데이터 y 값의 이름
- target : 데이터 y 값의 인덱스
- DESCR : 데이터에 대한 설명

04 20newsgroup으로 분류 연습하기

In [3]:
```
print(news.data[0])
```

Out [3]:
```
From: Mamatha Devineni Ratnam <mr47+@andrew.cmu.edu>
Subject: Pens fans reactions
Organization: Post Office, Carnegie Mellon, Pittsburgh,
PA
Lines: 12
NNTP-Posting-Host: po4.andrew.cmu.edu



I am sure some bashers of Pens fans are pretty confused
about the lack of any kind of posts about the recent Pens
massacre of the Devils.
Actually,
I am bit puzzled too and a bit relieved. However, I am
going to put an end
to non-PIttsburghers' relief with a bit of praise for the
Pens. Man,
they
are killing those Devils worse than I thought. Jagr just
showed you why he is much better than his
```

Out [3]:	season stats. He is also a lot fo fun to watch in the playoffs. Bowman should let JAgr have a lot of fun in the next couple of games since the Pens are going to beat the pulp out of Jersey anyway. I was very disappointed not to see the Islanders lose the final regular season game. PENS RULE!!!

- y 값은 target과 target_names

In [4]:	news.target
Out [4]:	array([10, 3, 17, ..., 3, 1, 7])
In [5]:	news.target_names
Out [5]:	['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale',

```
Out [5]:  'rec.autos',
          'rec.motorcycles',
          'rec.sport.baseball',
          'rec.sport.hockey',
          'sci.crypt',
          'sci.electronics',
          'sci.med',
          'sci.space',
          'soc.religion.christian',
          'talk.politics.guns',
          'talk.politics.mideast',
          'talk.politics.misc',
          'talk.religion.misc']
```

3. 뉴스 분류기 모델 개발의 전략

그림 11-7 뉴스 분류기 모델 개발 과정

- 벡터화 : BoW를 생성하는 CountVectorizer를 약간 변형한 TF-idfVectorizer를 생성하여 텍스트를 벡터화
- 교차 검증 : 모델 성능을 여러 번 측정하여 평균치를 측정
- 파이프라인 : 데이터 전처리부터 성능 측정까지 연결된 코드로 나타냄

4. 데이터 전처리

In [6]:
```
news_df = pd.DataFrame({'News' : news.data, 'Target' :
news.target})
news_df.head()
```

Out [6]:

	News	Target
0	From: Mamatha Devineni Ratnam <mr47+@andrew.cm...	10
1	From: mblawson@midway.ecn.uoknor.edu (Matthew ...	3
2	From: hilmi-er@dsv.su.se (Hilmi Eren)\nSubject...	17
3	From: guyd@austin.ibm.com (Guy Dawson)\nSubjec...	3
4	From: Alexander Samuel McDiarmid <am2o+@andrew...	4

In [7]:
```
target_dict = {idx:name for idx, name in
enumerate(news.target_names)}
news_df["Target"] = news_df["Target"].replace(target_dict)
```

04 20newsgroup으로 분류 연습하기

In [8]:

```
def data_cleansing(df):
    delete_email = re.sub(r'\b[\w\+]+@[\w]+.[\w]+
                          .[\w]+.[\w]+\b', ' ', df)
    delete_number = re.sub(r'\b|\d+|\b', ' ',
                           delete_email)
    delete_non_word = re.sub(r'\b[\W]+\b', ' ',
                             delete_number)
    cleaning_result =' '.join(delete_non_word.split())
    return cleaning_result

news_df.loc[:, 'News'] =
news_df['News'].apply(data_cleansing)
news_df.head()
```

Out [8]:

	News	Target
0	From Mamatha Devineni Ratnam Subject Pens fans...	rec.sport.hockey
1	From Matthew B Lawson Subject Which high perfo...	comp.sys.ibm.pc.hardware
2	From hilmi Hilmi Eren Subject Re ARMENIA SAYS ...	talk.politics.mideast
3	From Guy Dawson Subject Re IDE vs SCSI DMA and...	comp.sys.ibm.pc.hardware
4	From Alexander Samuel McDiarmid Subject driver...	comp.sys.mac.hardware

5. 벡터화하기

- BoW에 해당하는 CountVectorizer 외의 벡터화 기법들
- tfidf : 전체 문서에서 많이 나오는 단어의 중요도는 줄이고 해당 문서에만 많이 나오는 단어의 중요도를 올리는 기법
  - TF(Term Frequency) : 문서에서 해당 단어가 얼마나 나왔는지 나타내주는 빈도 수
  - DF(Document Frequency) : 해당 단어가 있는 문서의 수
  - IDF(Inverse Document Frequency) : 해당 단어가 있는 문서의 수가 높아질수록 가중치를 축소하기 위해 역수를 취함

$$\log(N \div (1 + DF))$$

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

- 여러 문서에서 단어가 많이 나오면 밑 수식에서 로그 값이 작아지면서 중요도를 떨어뜨림
- 사이킷런에서는 TfidfVectorizer 클래스 사용

- 토큰(token) : 인덱스를 지정해야 하는 단어들의 리스트를 정리하는 기법
  - 어간 추출(stemming) : 띄어쓰기 기준이 아닌 의미나 역하링 다른 단어들을 기준으로 분리
  - 문법적 기준을 기반으로 어근이나 어미를 토큰으로 사용

In [9]:	!pip install nltk
Out [9]:	Requirement already satisfied: nltk in c:\miniconda3\envs\ml\lib\site-packages (3.6.3) Requirement already satisfied: click in c:\miniconda3\envs\ml\lib\site-packages (from nltk) (8.0.1) Requirement already satisfied: regex in c:\miniconda3\envs\ml\lib\site-packages (from nltk) (2021.8.28)

	Requirement already satisfied: joblib in c:\miniconda3\envs\ml\lib\site-packages (from nltk) (1.0.1) Requirement already satisfied: tqdm in c:\miniconda3\envs\ml\lib\site-packages (from nltk) (4.62.3) Requirement already satisfied: colorama in c:\miniconda3\envs\ml\lib\site-packages (from click->nltk) (0.4.4)
In [10]:	from nltk import stem stmmer = stem.SnowballStemmer("english") sentence = 'looking looks looked‘ [stmmer.stem(word) for word in sentence.split()]
Out [10]:	['look', 'look', 'look']
In [11]:	stmmer.stem("images"), stmmer.stem("imaging"), stmmer.stem("imagination“)
Out [11]:	('imag', 'imag', 'imagin')

04 20newsgroup으로 분류 연습하기

In [12]:

```
from sklearn.feature_extraction.text import
CountVectorizer
import nltk

enlish_stemmer = nltk.stem.SnowballStemmer("english")
class StemmedCountVectorizer(CountVectorizer):
    def build_analyzer(self):
        analyzer =
super(StemmedCountVectorizer,self).build_analyzer()
        return lambda doc: (enlish_stemmer.stem(w)
                            for w in analyzer(doc))

from sklearn.feature_extraction.text import
TfidfVectorizer

enlish_stemmer = nltk.stem.SnowballStemmer("english")
class StemmedTfidfVectorizer(TfidfVectorizer):
    def build_analyzer(self):
        analyzer =
super(StemmedTfidfVectorizer,self).build_analyzer()
        return lambda doc: (enlish_stemmer.stem(w)
                            for w in analyzer(doc))
```

6. 모델링하기

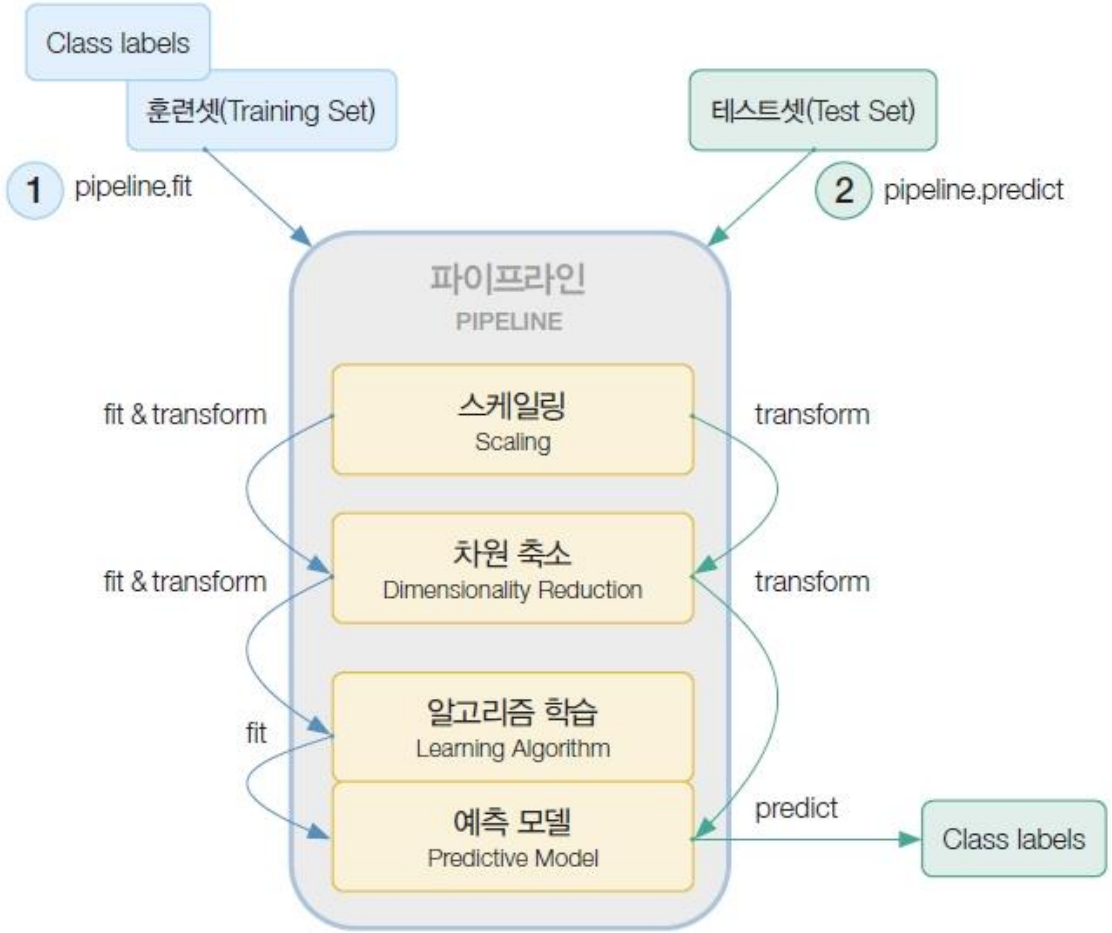

그림 11-8 파이프라인을 구성하여 데이터 처리

- 경우의 수 생성하고 학습 파이프라인에 넣어 학습 수행

In [13]:
```
from sklearn.naive_bayes import MultinomialNB,
BernoulliNB,GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline

vectorizer = [CountVectorizer(), TfidfVectorizer(),
StemmedCountVectorizer(), StemmedTfidfVectorizer()]
# algorithms = [BernoulliNB(), MultinomialNB(),
GaussianNB(), LogisticRegression()]
algorithms = [MultinomialNB(), LogisticRegression()]

pipelines = []
```

In [14]:
```
import itertools
for case in list(itertools.product(vectorizer,
algorithms)):
 pipelines.append(make_pipeline(*case))
```

04 20newsgroup으로 분류 연습하기

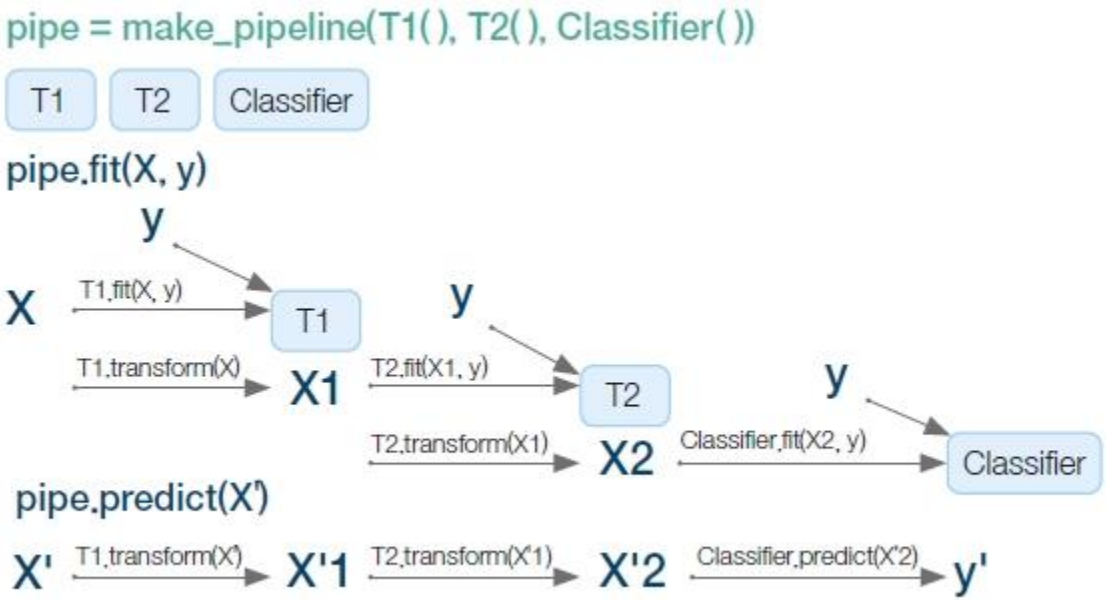

그림 11-9 파이프라인에 저장하여 학습하기

In [15]:

```
ngrams_params = [(1,1),(1,3)]
stopword_params = ["english"]
lowercase_params = [True, False]
max_df_params = np.linspace(0.4, 0.6, num=6)
min_df_params = np.linspace(0.0, 0.0, num=1)

attributes = {"ngram_range":ngrams_params,
"max_df":max_df_params,"min_df":min_df_params,

"lowercase":lowercase_params,"stop_words":stopword_params}
vectorizer_names = ["countvectorizer","tfidfvectorizer",
"stemmedcount vectorizer","stemmedtfidfvectorizer"]
vectorizer_params_dict = {}

for vect_name in vectorizer_names:
    vectorizer_params_dict[vect_name] = {}
    for key, value in attributes.items():
        param_name = vect_name + "__" + key
        vectorizer_params_dict[vect_name][param_name] =
value
```

04 20newsgroup으로 분류 연습하기

In [16]:

```
algorithm_names = ["multinomialnb", "logisticregression"]

algorithm_params_dict = {}
alpha_params = np.linspace(1.0, 1.0, num=1)
for i in range(1):
    algorithm_params_dict[algorithm_names[i]] = {
        algorithm_names[i]+ "__alpha" : alpha_params
    }
c_params = [0.1, 5.0, 7.0, 10.0, 15.0, 20.0, 100.0]

algorithm_params_dict[algorithm_names[1]] = [{
    "logisticregression__multi_class" : ["multinomial"],
    "logisticregression__solver" : ["saga"],
    "logisticregression__penalty" : ["l1"],
    "logisticregression__C" : c_params
    },{
    "logisticregression__multi_class" : ["ovr"],
    "logisticregression__solver" : ['liblinear'],
    "logisticregression__penalty" : ["l2"],
    "logisticregression__C" : c_params
   }
    ]
```

In [17]:
```
pipeline_params= []
for case in list(itertools.product(vectorizer_names,
algorithm_names)):
    vect_params = vectorizer_params_dict[case[0]].copy()
    algo_params = algorithm_params_dict[case[1]]

    if isinstance(algo_params, dict):
        vect_params.update(algo_params)
        pipeline_params.append(vect_params)
    else:
        temp = []
        for param in algo_params:
            vect_params.update(param)
            temp.append(vect_params)
        pipeline_params.append(temp)
```

7. 학습 수행하기

- 구성된 파이프라인을 순서대로 호출

In [18]:
```
from sklearn.preprocessing import LabelEncoder

X_data = news_df.loc[:, 'News'].tolist()
y_data = news_df['Target'].tolist()
y = LabelEncoder().fit_transform(y_data)
```

In [19]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report,
accuracy_score

scoring = ['accuracy']
estimator_results = []
```

```
for i, (estimator, params) in
enumerate(zip(pipelines,pipeline_params)):
    n_jobs = -1
    gs_estimator = GridSearchCV(refit="accuracy",
                estimator=estimator,param_grid=params,
                scoring=scoring, cv=5, verbose=1,
                n_jobs=n_jobs)
    print(gs_estimator)

    gs_estimator.fit(X_data, y)
    estimator_results.append(gs_estimator)
```

[TIP] 다음과 같은 결과가 화면에 출력된다. 현재 4개의 하이퍼 매개변수 실험이 동시에 이루어지고 있다는 것을 의미하는데, 컴퓨터 CPU가 많으면 많을수록 좀 더 많은 학습을 수행할 수 있다

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4
concurrent workers.
```