# 7장 나이브 베이즈 분류 : 스팸 메일 데이터셋

#### 학습 목표

나이브 베이즈Naive Bayes 알고리즘를 파이썬으로 구현하여 스팸 문자를 필터링하는 예측 모델을 만들어봅시다. 그 후 수식으로 나이브 베이즈를 더 깊게 이해해봅시다.

#### 학습 순서



#### 나이브 베이즈 소개

나이브 베이즈는 베이즈 정리를 적용한 조건부 확률 기반의 분류 모델입니다. 여기서 조건부 확률은 A가 일어났을 때 B가 일어날 확률을 의미합니다. 예를 들어 ‘무료라는 단어가 들어 있을 때 해당 메일이 스팸일 확률’ 같은 겁니다. 이러한 특징으로 스팸 필터링을 위한 대표적인 모델로 꼽힙니다. 최근에는 딥러닝 같은 대안이 있어서 나이브 베이즈 모델을 쓰고자 하는 상황이 많지는 않습니다만, 그래도 스팸 메일 필터처럼 자연어 처리가 목적일 때는 여전히 나이브 베이즈 모델이 좋은 선택이 될 수 있습니다(딥러닝이 자연어 처리에 더 탁월한 모습을 보여주지만, 딥러닝보다 간단한 방법으로 자연어 처리를 원할 때).

<추가>



#### 장단점

| **장점** | **단점** |
| --- | --- |
| * 비교적 간단한 알고리즘에 속하며 속도 또한 빠릅니다. * 작은 훈련셋으로도 잘 예측합니다. | * 모든 독립변수가 각각 독립적임을 전제로 하는데 이는 장점이 되기도 하고 단점이 되기도 합니다. 실제로 독립변수들이 모두 독립적이라면 다른 알고리즘보다 우수할 수 있지만, 실제 데이터에서 그런 경우가 많지 않기 때문에 단점이기도 합니다. |

#### 유용한 곳

* 각 독립변수들이 모두 독립적이고 그 중요도가 비슷할 때 유용합니다.
* 자연어 처리(NLP)에서 간단하지만 좋은 성능을 보여줍니다.
* 카테고리 형태의 변수가 많을 때 적합하며, 숫자형 변수가 많은 때는 적합하지 않습니다.

#### TOP 10 선정 이유

* 범용성이 높지는 않지만 (앞서 설명한) 환경이 딱 들어맞는다면 충분히 경쟁력이 있는 알고리즘입니다. 특히나 딥러닝을 제외하고 자연어 처리에 가장 적합한 알고리즘입니다. 일반적인 데이터보다는 특수 상황을 고려해 배워두길 바랍니다.

## 7.1 문제 정의 : 한눈에 보는 분석 목표

<금토끼의 문제 정의> 금토끼는 최근 들어 스팸 문자가 너무 많아와서 업무에 집중하기가 힘들었습니다. 스팸 문자 여부를 판별하는 알고리즘을 만들어 활용하고 싶어졌습니다. 스팸 문자 데이터셋을 찾아서 확인해보니 “머신러닝은 데싸노트의 TOP 10 알고리즘으로 배우세요”처럼 사람이 쓰는 자연어로 되어 있었습니다. 단어가 열거되어 있어서 특정 카테고리로 구분할 수 없으며 더미 변수를 사용할 수도 없는 상황이었습니다. ‘문장에 사용된 단어를 사용 빈도로 구분하면 스팸 여부를 판단할 수 있지 않을까’라는 생각이 들었습니다.

| **난이도** | ⭐⭐☆ | | |
| --- | --- | --- | --- |
| **알고리즘** | 나이브 베이즈(Naive Bayes) | | |
| **데이터셋 파일명** | spam.csv | **종속 변수** | target(스팸 여부) |
| **데이터셋 소개** | 스팸 문자에 대한 데이터로, 독립변수는 text 하나밖에 없습니다. 그러나 이 하나의 변수에 긴 문장 형태의 데이터들이 들어있기 때문에 많은 전처리 작업이 필요합니다. 각 문장에 들어간 단어들을 활용하여 문자가 스팸인지 아닌지를 예측하게 됩니다. | | |
| **문제 유형** | 분류 | **평가지표** | 정확도, 혼동 행렬 |
| **사용한 모델** | MultinomialNB | | |
| **사용 라이브러리** | * numpy (numpy==1.19.5) * pandas (pandas==1.3.5) * seaborn (seaborn==0.11.2) * matplotlib (matplotlib==3.2.2) * sklearn (scikit-learn==1.0.2) * nltk (nltk==3.2.5) | | |
| **예제 코드 노트북** | 위치 : <https://github.com/musthave-ML10/notebooks>  파일 : 07\_Naive Bayes.ipynb | | |

## 7.2 라이브러리 및 데이터 불러오기 & 데이터 확인

우선 4개의 필수 모듈과 spam.csv 파일을 불러오겠습니다.

| import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns  file\_url = 'https://media.githubusercontent.com/media/musthave-ML10/data\_source/main/spam.csv' data = pd.read\_csv(file\_url) # 데이터셋 읽기 |
| --- |

head를 사용하여 데이터의 형태를 살펴봅시다.

| data.head() # 상위 5행 출력 |
| --- |

|  | **target** | **text** |
| --- | --- | --- |
| **0** | ham | Go until jurong point, crazy.. Available only ... |
| **1** | ham | Ok lar... Joking wif u oni... |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| **3** | ham | U dun say so early hor... U c already then say... |
| **4** | ham | Nah I don't think he goes to usf, he lives aro.. |



첫 번째 컬럼인 target이 목표 변수입니다. 현재까지는 spam과 ham 두 가지네요. 혹시 다른 값도 있는지 unique() 함수로 확실하게 확인하고 가겠습니다.

| data['target'].unique() # 목표 변수의 고윳값 확인 |
| --- |

array(['ham', 'spam'], dtype=object)

예상대로 ham과 spam뿐입니다. 여기서 spam은 스팸 문자를 뜻하고, ham은 스팸이 아닌 문자입니다(스팸과 햄으로 부르겠습니다).

text 컬럼을 살펴보면 말씀드린 것처럼 자연어 형태입니다.

## 7.3 전처리 : 특수 기호 제거하기

한 가지 처리할 게 있습니다. 바로 쉼표, 마침표 등과 같은 특수 기호들입니다. 자연어를 다룰 때 데이터의 기준은 단어입니다. 단어를 처리할 때 특수 기호는 노이즈가 되므로 제거해야 합니다. 우선은 제거할 특수 기호를 목록으로 마련해야 합니다.

특수 기호 목록은 파이썬에 내장된 string에서 얻을 수 있습니다.

| import string # 임포트 |
| --- |

string에 내재된 punctuation를 실행하면 특수 기호 목록을 확인할 수 있습니다.

| string.punctuation # 특수 기호 목록 출력 |
| --- |

!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

이제부터 ❶ 문자열에서 문자를 하나씩 꺼내 특수 기호인지 판단한 뒤, 특수 기호가 아닌 문자들만으로 리스트에 저장합니다. ❷ 각 문자를 문장으로 합칩니다. ❸ 이 문자열이 문장별로 행에 저장되게 변환해봅시다.



먼저 문자열 하나하나를 가져와서 특수문자를 없애는 작업을 해보겠습니다. 우선 문자열을 하나 가져오겠습니다.

| sample\_string = data['text'].loc[0] # 문자열 선택 sample\_string |
| --- |

'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...'

이 문자열의 문자를 하나하나 불러와 출력해보겠습니다.

| for i in sample\_string: # 문자열의 문자를 하나씩 출력  print(i) |
| --- |

G

o

u

n

t

i

l

... 생략 ...

그럼 한 줄에 한 문자가 출력되는데 너무 길어서 아랫부분을 생략했습니다. 이제 이렇게 불러온 문자에 특수 기호가 보이네요.

특수 기호에 속하는지 아닌지 판단하는 코드를 추가해봅시다.

| for i in sample\_string: # 문자열의 문자를 하나씩 출력  if i not in string.punctuation: # ❶ 특수 기호가 아니면  print(i) |
| --- |

G

o

u

n

t

i

l

... 생략 ...

기대한 바대로 쉼표나 마침표 같은 특수 기호가 안 보입니다. ❶ string.punctuation에 있는 문자이면 특수 기호라는 뜻입니다. 특수 기호가 아닌 문자만 필요하므로 not in을 사용했습니다(반대로 특수 기호인지 판단하려면 in만 써주세요).

<글상자/>

**in 살펴보기**

문자열에 속하는지 아닌지 알아보는 쉬운 코드를 제시해봅니다.

| 'a' in 'apple' # 문자열에 ‘a’가 있는지 확인 |
| --- |

True

| 'b' in 'apple' # 문자열에 ‘b’가 있는지 확인 |
| --- |

False

‘apple’이라는 문자열 안에 ‘a’가 있으므로 True, ‘b’가 없으므로 False가 됩니다. if절은 주어진 조건에서 True인 것들에만 작동하게 됩니다.

</>

이렇게 특수 기호를 제외한 문자들을 리스트 형태로 모아주겠습니다.

| new\_string =[] # ❶ 빈 리스트 생성 for i in sample\_string: # 문자열 순회  if i not in string.punctuation: # 특수 기호가 아니면  new\_string.append(i) # ❷ 리스트에 문자 추가 |
| --- |

❶ 빈 리스트(new\_string)를 하나 만들어 ❷ append() 함수를 사용하여 특수 기호가 아닌 문자를 하나씩 추가합니다.

new\_string을 출력하면 리스트 형태로 문자를 확인할 수 있습니다.

| new\_string |
| --- |

['G',

'o',

' ',

'u',

'n',

't',

'i',

'l',

... 생략 ...

문장을 분석해야 하므로 리스트 안의 문자들을 문자열로 만들어야 합니다. join() 함수를 쓰면 리스트 안의 값들을 하나로 합칠 수 있습니다.

<함수 소개/>

| **함수** | **설명** |
| --- | --- |
| join() | 리스트 안에 있는 문자들을 합쳐서 문자열로 만듭니다.  sample = ['a','p','p','l','e'] # ❶  '\_'.join(sample) # ❷  'a\_p\_p\_l\_e'  ❶ sample 리스트에 문자 5개가 있습니다. ❷ join() 함수 앞에 ‘\_’를 붙였습니다. 각 문자 사이에 \_를 넣으라는 뜻입니다. 아무것도 지정하지 않아도 됩니다. |

</>

문장 형태로 만들어봅시다.

| new\_string =[] # 빈 리스트 생성 for i in sample\_string: # 문자열 순회  if i not in string.punctuation: # 특수 기호가 아니면  new\_string.append(i) # 리스트에 문자 추가 new\_string = ''.join(new\_string) # ❶ 리스트를 문자열 형태로 변환 |
| --- |

❶ join() 코드는 for문이 모두 끝난 뒤에 실행되어야 하니 들여쓰지 않도록 주의하시기 바랍니다.

그럼 이 코드를 별도의 함수로 만들겠습니다.

| def remove\_punc(x): # ❶ 함수 정의  new\_string =[] # 빈 리스트  for i in x: # 순회  if i not in string.punctuation: # 특수 기호가 아니면  new\_string.append(i) # 리스트에 문자 추가  new\_string = ''.join(new\_string) # 리스트를 문자열 형태로 변환   return new\_string # ❷ 반환 |
| --- |

❶ def를 사용하여 remove\_punc() 함수를 지정했습니다. ❷ 최종적으로 합쳐진 new\_string값을 반환합니다.

함수가 잘 작동하는지 sample\_string으로 테스트하겠습니다.

| remove\_punc(sample\_string) # 특수 기호 삭제 함수 호출 |
| --- |

'Go until jurong point crazy Available only in bugis n great world la e buffet Cine there got amore wat'

우리가 기대한 결과를 확인할 수 있습니다. 그러면 데이터(data)에 적용할 차례입니다. 과연 제대로 동작할지 궁금하군요.

| remove\_punc(data['text']) # 특수 기호 삭제 함수 호출 |
| --- |

'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...Ok lar... Joking wif u oni...Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C\'s apply 08452810075over18\'sU dun say so early hor... U c already then say...Nah I don\'t think he goes to usf, he lives around here thoughFreeMsg Hey there darling it\'s been 3 week\'s now and no word back! I\'d like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcvEven my brother is not like to speak with me. They treat me like aids patent.As per your ... 생략 ...

결과물에서 뭔가 이상한 점이 느껴지시나요? 행마다 한 개 문자열이 보이길 기대했으나 한 행에 모든 문자가 합쳐진 형태입니다. 이대로 data['text']를 업데이트해주면 큰일 납니다. 언제 어떤 문제가 발생할지 모르니 데이터를 다룰 때는 반드시 하나하나 확인해주세요.

이런 상황이 발생한 이유는 remove\_punc() 함수가 한 줄의 문자열에만 작동하도록 설계되었기 때문입니다. 따라서 시리즈의 한 줄 한 줄 따로 적용되게 하는 코드가 구현해 해결하면 됩니다. apply() 함수를 사용하면 데이터프레임의 한 줄 한 줄을 따로 함수에 적용시킬 수 있습니다.

<함수/>

| **함수** | **설명** |
| --- | --- |
| apply() | 함수가 데이터의 한 행마다 별도로 적용되어야 할 때 사용하면 각 행마다 함수를 적용할 수 있습니다.  data = pd.Series([[1,2], [3,4,5]]) # ❶  def check\_len(x):  return len(x)  data.apply(check\_len) # ❷  0 2  1 3  dtype: int64  ❶ data의 각 행에 각각 숫자 두 개 [1,2]와 세 개 [3,4,5]를 가진 리스트[[1,2],[3,4,5]]가 들어있습니다. ❷ data 뒤에 apply()를 붙이고 사용하려는 함수의 이름을 넣습니다. |

</>

apply() 함수를 사용해 데이터프레임 전체가 아닌 text 변수에, 즉 판다스 시리즈 형태에 적용해보겠습니다.

| data['text'].apply(remove\_punc) # 함수에 한 행씩 적용되도록 실행 |
| --- |

0 Go until jurong point crazy Available only in ...

1 Ok lar Joking wif u oni

2 Free entry in 2 a wkly comp to win FA Cup fina...

3 U dun say so early hor U c already then say

4 Nah I dont think he goes to usf he lives aroun...

...

5569 This is the 2nd time we have tried 2 contact u...

5570 Will ü b going to esplanade fr home

5571 Pity was in mood for that Soany other suggest...

5572 The guy did some bitching but I acted like id ...

5573 Rofl Its true to its name

Name: text, Length: 5574, dtype: object

우리가 기대한 대로 출력됐습니다. 결과를 data['text']에 업데이트해주겠습니다.

| data['text'] = data['text'].apply(remove\_punc) # 데이터셋 업데이트 |
| --- |

## 7.4 전처리 : 불용어 제거하기

특수 기호를 모두 처리했으니 이번에는 불용어stopword를 제거하겠습니다. 불용어는 자연어 분석에 큰 도움이 안 되는 단어를 의미합니다. 이러한 단어를 제거해주면 데이터를 조금이나마 더 가볍게 만들 수 있습니다. 자연어 처리에서는 각 단어가 하나의 독립변수처럼 작용하기 때문에, 지금은 컬럼이 2개뿐인데도 분석 과정에서 데이터를 방대하게 펼치게 됩니다. 그래서 불용어를 제거해 조금이라도 부담을 더는 겁니다.

그러한 이유로 불용어를 제거해보겠습니다. ❶ 판다스 시리즈에 저장된 문자열 하나를 단어 단위로 리스트로 변환하고 ➝ ❷ 불용어가 아니면 소문자로 저장한 뒤 ➝ ❸ 문자를 문자열로 합칩니다. ➝ ➍ 이 과정을 반복하며 모든 문자열에 적용합니다.



분석의 목적에 따라 불용어가 달라질 수 있습니다. 예를 들어 지금처럼 스팸 문자를 예측할 때와, 상품 리뷰의 긍정/부정을 예측할 때 의미있는 단어의 성격이 다를 수 있습니다. 불용어를 목적에 맞게 작성해도 되지만, 여기에서는 nltk 라이브러리에서 제공되는 불용어 목록을 사용합니다. 우선 nltk를 통해 ‘stopwords’를 다운로드합니다.

| import nltk # 임포트 nltk.download('stopwords') # 불용어 목록 가져오기 |
| --- |

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

True

이제 stopwords를 임포트하겠습니다.

| from nltk.corpus import stopwords # 불용어 목록 임포트 |
| --- |

stopwords에서 제공되는 리스트를 확인해봅시다.

| stopwords.words('english') # 영어 불용어 선택 |
| --- |

['i',

'me',

'my',

'myself',

'we',

'our',

'ours',

'ourselves',

'you',

"you're",

"you've",

"you'll",

"you'd",

'your',

'yours',

'yourself',

'yourselves',

... 생략 ...

<글상자/>

**stopwords.words('english')에서** **'english'**

‘english’는 영문 불용어를 지정하는 속성입니다. 한국어 불용어를 지정하려면 ‘korea’를 넣으면 될 것 같지만 안 됩니다. NLTK 라이브러리는 한국에 불용어가 제공되지 않습니다. 한국어 텍스트를 다룰려면 불용어를 별도로 리스트 형태로 만들어서 지정하시거나, 구글에서 검색하여 몇 가지 준비된 한국어 불용어를 활용하면 됩니다. 또한 한국어의 경우 영어와는 달리 단어에 조사가 붙기 때문에 특정 단어를 포착하기 위해 이러한 부분도 처리해야 하는데, 이 책에서는 NLP가 주목적이 아니므로 해당 내용까지는 다루지 않습니다. 한국어 불용어는 [www.ranks.nl](about:blank) 등에서 받을 수 있습니다.

* 한국어 불용어 : https://www.ranks.nl/stopwords/korean

현재 24개 언어의 불용어를 제공합니다.

from nltk.corpus import stopwords

print(stopwords.fileids())

['arabic',

'azerbaijani',

'bengali',

'danish',

'dutch',

'english',

'finnish',

'french',

'german',

'greek',

'hungarian',

'indonesian',

'italian',

'kazakh',

'nepali',

'norwegian',

'portuguese',

'romanian',

'russian',

'slovene',

'spanish',

'swedish',

'tajik',

'turkish']

</>

리스트 형태로 제공되며, 분석에 큰 영향이 없을 것 같은 단어(i, you, he, she 등)가 들어 있습니다. 불용어를 제거하는 함수를 만들겠습니다.

우선 for문을 통해서 문장의 단어를 하나하나씩 불러와야 합니다. 특수 기호는 문자 하나하나를 불러와서 확인했던 반면, 이번에는 단어 하나하나를 불러와야 합니다. 그러나 문장을 그대로 for문에 적용하면 아까처럼 문자 하나씩 불러오므로 문장을 단어 단위로 쪼개는 작업이 필요합니다. split() 함수를 사용해 쪼갤 수 있습니다.

<함수/>

| **함수** | **설명** |
| --- | --- |
| split() | 기본적으로 띄어쓰기 기반으로 단어를 분리합니다. 만약 띄어쓰기가 아니라 특정 문자 기준으로 분리하고 싶다면 괄호 안에 해당 문자를 입력하면 됩니다. 알파벳, 특수 기호 등 어떤 것도 가능합니다. 예시를 위하여 임의의 스트링을 하나 만들겠습니다.   | sample\_string2 = 'This is not - SPAM' | | --- |   이 문장을 ‘-’ 문자를 기준으로 나눠보겠습니다.   | sample\_string2.split('-') # 특정 문자를 기준으로 문자열 분할 | | --- |   ['This is not ', ' SPAM'] |

</>

| sample\_string = data['text'].loc[0]  sample\_string.split() # 단어 단위로 문장 분할 |
| --- |

['Go',

'until',

'jurong',

'point',

'crazy',

'Available',

'only',

'in',

'bugis',

'n',

'great',

'world',

'la',

'e',

'buffet',

'Cine',

'there',

'got',

'amore',

'wat']

단어 단위 리스트로 변경되었습니다.

각 단어가 불용어에 속하는지 아닌지 판독하는 코드를 구현하겠습니다.

| for i in sample\_string.split(): # 순회  if i not in stopwords.words('english'): # 불용어가 아니면  print(i) # 출력 |
| --- |

Go

jurong

point

crazy

Available

bugis

n

great

world

la

e

buffet

Cine

got

amore

wat

여기서 하나 손봐주어야 할 부분이 있는데요, 보면 Go나 Cine 같은 단어는 첫 글자가 대문자입니다. 파이썬은 대소문자를 구분합니다. stopwords에 들어 있는 단어들은 모두 소문자로 되어 있습니다. 정확하게 하려면 확인할 단어가 모두 소문자이어야 합니다. 따라서 string의 대소문자 변환에 관련된 코드를 살펴보겠습니다.

<함수/>

| **함수** | **설명** |
| --- | --- |
| lower() | 소문자로 바꿉니다.  sample\_word = 'nAive\_Bayes'  print(sample\_word.lower())  naive\_bayes |
| upper() | 대문자로 바꿉니다.  print(sample\_word.upper())  NAIVE\_BAYES |
| capitalize() | 단어의 첫 문자만 대문자로, 나머지는 소문자로 바꿉니다.  print(sample\_word.capitalize())  Naive\_bayes |

</>

lower() 함수를 사용해 모두 소문자로 바꿉니다.

| for i in sample\_string.split(): # 순회  if i.lower() not in stopwords.words('english'): # ❶ 불용어가 아니면 소문자로 변환   print(i.lower()) # 소문자로 출력 |
| --- |

go

jurong

point

crazy

available

bugis

n

great

world

la

e

buffet

cine

got

amore

wat

❶ 단어를 받아주는 역할을 i가 하고 있기 때문에 i에 lower를 적용했습니다.

각 단어를 문장으로 다시 합칩니다.

| new\_string=[] # ❶ 빈 리스트 생성 for i in sample\_string.split(): # 순회  if i.lower() not in stopwords.words('english'): # 소문자로 변환한 단어가 불용어가 아니면  new\_string.append(i.lower()) # ❷ 문자 단위로 추가  new\_string = ' '.join(new\_string) # ❸ 공백 단위로 묶기   new\_string |
| --- |

'go jurong point crazy available bugis n great world la e buffet cine got amore wat'

❶ 빈 문자열을 만들어 ❷ 문자 단위로 추가해 ❸ 공백을 기준으로 단어를 합쳐줍니다.

join() 앞 따옴표 사이에 공란을 빼먹지 마세요! 우리가 특수 기호를 제거할 때는 이곳에 빈칸이 없었는데, 이때는 문자 하나씩을 모아서 문장처럼 이어붙였기 때문입니다. 반면 지금은 단어 단위로 모아졌기 때문에 단어 사이의 빈칸을 만들어주어야 합니다. 그래야 제대로 된 문장 형태로 변경됩니다.

샘플 문장에서 stopwords를 제거하는 코드를 작성해보았습니다. text 컬럼 전체에 적용해야 하므로 별도의 함수로 만들고, apply()를 각 데이터 라인에 적용시키겠습니다.

| def stop\_words(x):  new\_string=[] # 새 리스트 생성  for i in x.split(): # 순회  if i.lower() not in stopwords.words('english'): # 소문자로 변환한 단어가 불용어가 아니면  new\_string.append(i.lower()) # 문자 단위로 추가  new\_string = ' '.join(new\_string) # 공백 단위로 묶기   return new\_string # ❶ 반환  data['text'] = data['text'].apply(stop\_words) # ❷ 텍스트에 stop\_words 함수 적용 data['text'] |
| --- |

0 go jurong point crazy available bugis n great ...

1 ok lar joking wif u oni

2 free entry 2 wkly comp win fa cup final tkts 2...

3 u dun say early hor u c already say

4 nah dont think goes usf lives around though

...

5569 2nd time tried 2 contact u u £750 pound prize ...

5570 ü b going esplanade fr home

5571 pity mood soany suggestions

5572 guy bitching acted like id interested buying s...

5573 rofl true name

Name: text, Length: 5574, dtype: object

원하는 대로 출력되었군요.

sample\_string을 x로 변경하고, ❶ 최종 변경된 new\_string을 반환합니다. 그리고 ❷ data['text']에 반영합니다.

## 7.5 전처리 : 목표 컬럼 형태 변경하기

이번 예제에서 목표 컬럼은 문자(spam과 ham)입니다. 문자 형식의 데이터도 모델링에 에러를 유발하지는 않지만, 때에 따라 해석에 문제가 생길 수도 있기 때문에 숫자로 변환하겠습니다. 변환에는 map() 함수를 사용합니다.

<함수/>

| **이름** | **설명** |
| --- | --- |
| map() | map() 함수는 딕셔너리 타입의 데이터를 사용하여 매칭되는 값을 불러오도록 사용할 수 있습니다.  <코드/>  sample1 = pd.Series(['a','b','c']) # ❶ 시리즈 생성  sample1.map({'a': 'apple', 'b':'banana','c':'cherry'}) # ❷ 키의 값 출력  0 apple  1 banana  2 cherry  </>  <출력 결과/>  dtype: object  </>  ❶ a,b,c를 판다스 시리즈 형태로 저장합니다.  ❷ map() 함수가 딕셔너리 {'a': 'apple', 'b':'banana','c':'cherry'}에서 a,b,c 각각의 키에 매칭되는 값들을 불러옵니다.  또한 apply() 함수처럼 다른 함수를 적용시키는데에 사용할 수도 있습니다.  <코드/>  sample2 = pd.Series(['a','b','c']) # ❸ 시리즈 생성  def add\_i(x): # ❹ 사용자 지정 함수  return x+'i' # ‘i’ 추가  sample2.map(add\_i) # ❺ ‘i’를 추가하는 함수 호출  0 ai  1 bi  2 ci  <코드/>  <출력결과/>  dtype: object  </>  ❸ a,b,c를 판다스 시리즈 형태로 저장합니다.  ❹ 각 문자에 i가 추가되는 함수를 만들었습니다.  ❺ map을 사용하여 판다스 시리즈에 add\_i 함수를 적용시켰습니다. |

</>

스팸이면 1, 아니면 0으로 변환합시다.

| data['target'] = data['target'].map({'spam':1, 'ham':0}) # ❶ 텍스트를 숫자로 변환  data['target'] # 출력 |
| --- |



❶ map() 함수 인수로 딕셔너리 형태로 매핑할 값을 정의해 넣어주었습니다. ❷ 출력을 보니 0과 1로 제대로 변경되었습니다.

## 7.6 전처리 : 카운트 기반으로 벡터화하기

카운트 기반 벡터화[[1]](#footnote-0)는 말 그대로 문자를 개수 기반으로 벡터화하는 방식입니다. 데이터 전체에 존재하는 모든 단어들을 사전처럼 모은 뒤에 인덱스를 부여하고, 문장마다 속한 단어가 있는 인덱스를 카운트하는 겁니다.

예를 들어 다음과 같은 데이터가 있다고 합시다.

<그림>

▽ 카운트 기반 벡터화 과정



</>

이제부터 카운트 기반 벡터화 기법을 코딩로 구현해봅시다. text와 target 컬럼을 x, y라는 이름으로 저장합니다.

| x = data['text'] # 독립변수 y = data['target'] # 종속변수 |
| --- |

벡터화에 사용할 CountVectorizer를 임포트합니다.

| from sklearn.feature\_extraction.text import CountVectorizer |
| --- |

CountVectorizer 사용 방법은 scaling과 상당히 유사합니다. 임의의 이름으로 해당 모듈 속성을 부여하고, fit()으로 학습하며 transform()으로 변환합니다.

우선 fit()으로 학습을 진행합시다.

| cv = CountVectorizer() # 객체 생성 cv.fit(x) # ❶ 학습하기  cv.vocabulary\_ # ❷ 단어와 인덱스 출력 |
| --- |

{'go': 3791,

'jurong': 4687,

'point': 6433,

'crazy': 2497,

'available': 1414,

'bugis': 1881,

'great': 3888,

'world': 9184,

'la': 4847,

'buffet': 1879,

'cine': 2214,

'got': 3848,

'amore': 1181,

'wat': 8947,

'ok': 5995,

'lar': 4886,

'joking': 4655,

'wif': 9079,

'oni': 6027,

'free': 3577,

... 생략 ...

❶ fit()으로 x를 학습한 결과를 cv 객체(모델)에 저장합니다. ❷ cv 뒤에 vocabulary\_를 쓰면 객체에 들어 있는 모든 단어에 대한 인덱스를 볼 수 있습니다. 여기서 인덱스는 큰 의미가 없는 일종의 id와 같은 역할입니다.

그럼 이제 학습된 cv의 transform() 함수로 데이터 x를 변환하고 결과를 확인하겠습니다.

| x = cv.transform(x) # 트랜스폼 print(x) # 이 데이터는 print를 사용해야 아래와 같은 결과물이 보입니다. |
| --- |

(0, 1181) 1

(0, 1414) 1

(0, 1879) 1

(0, 1881) 1

(0, 2214) 1

(0, 2497) 1

(0, 3791) 1

(0, 3848) 1

(0, 3888) 1

(0, 4687) 1

(0, 4847) 1

(0, 6433) 1

(0, 8947) 1

(0, 9184) 1

(1, 4655) 1

(1, 4886) 1

(1, 5995) 1

(1, 6027) 1

(1, 9079) 1

(2, 71) 1

(2, 441) 1

(2, 454) 1

(2, 875) 1

(2, 1267) 1

(2, 2330) 1

... 생략 ...

다소 이해하기 힘든 출력 형태네요, 첫 번째 줄의 데이터를 예로 들어 구체적으로 알아보겠습니다.



❶ 첫 번째 숫자인 0은 우리가 변환시킨 데이터의 행(row) 번호입니다. 0번째 줄 데이터라는 의미입니다. ❷ 뒤에 있는 숫자는 단어의 인덱스값이고, ❸ 마지막 1은 해당 단어가 1번 등장했다는 의미입니다. 따라서 ‘0번째 행 데이터에는 인덱스가 1181인 단어가 한 번 등장한다’로 해석하면 됩니다. 실제로 0번째 행의 데이터를 확인해봅시다.

| data.loc[0]['text'] # 0번째 행의 ‘text’ 열의 값 선택 |
| --- |

'go jurong point crazy available bugis n great world la e buffet cine got amore wat'

정말 그런지 앞의 세 단어 go, jurong, point만 확인해볼게요. cv.vocabulary\_는 딕셔너리 형태였기 때문에, 다음과 같은 코드로 인덱스값을 출력하면 됩니다.

| print(cv.vocabulary\_['go']) # 인덱스 출력 print(cv.vocabulary\_['jurong']) # 인덱스 출력 print(cv.vocabulary\_['point']) # 인덱스 출력 |
| --- |

3791

4687

6433

<note/>

앞의 코드에서 굳이 print() 함수를 사용하지 않아도 됩니다. 그럼에도 사용한 이유는 이 책의 실습을 주피터 노트북에서 진행하기 때문입니다. 주피터 노트북에서 한 셀에 여러 줄을 출력하려면 print()를 써주셔야 합니다. 그러지 않으면 마지막 한 줄에 대한 결괏값만 출력합니다. print() 함수를 쓰지 않으려면 출력 단위(앞의 코드에서는 한 줄)로 실행해야 합니다.

</>

go의 인덱스는 3791, jurong은 4687, point는 6433입니다. 이 숫자이 문장(X 출력 결과)에서 한 번씩만 출연해야겠죠? 정말 그런지 확인해볼까요?

(0, 1181) 1

(0, 1414) 1

(0, 1879) 1

(0, 1881) 1

(0, 2214) 1

(0, 2497) 1

(0, 3791) 1 ← go

(0, 3848) 1

(0, 3888) 1

(0, 4687) 1 ← jurong

(0, 4847) 1

(0, 6433) 1 ← point

(0, 8947) 1

(0, 9184) 1

(1, 4655) 1

(1, 4886) 1

(1, 5995) 1

(1, 6027) 1

(1, 9079) 1

(2, 71) 1

(2, 441) 1

(2, 454) 1

(2, 875) 1

(2, 1267) 1

(2, 2330) 1

... 생략 ...

샘플로 3개 단어만 확인했습니다. 그외 단어를 확인해도 결과는 일치할 겁니다. 문자를 이런 식의 형태로 변환시키는 방법을 카운트 기반 벡터화(CounterVectorize)라고 합니다.

## 7.7 모델링 및 예측/평가하기

모델링에 앞서 훈련셋과 시험셋으로 나누어주겠습니다. 카운트 기반 벡터화 과정에서 X와 y로 데이터를 나누어주었으니 이를 활용하여 train\_test\_split() 함수로 독립변수/종속변수에 대한 훈련셋/시험셋 분할을 진행하겠습니다.

| from sklearn.model\_selection import train\_test\_split # 임포트 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 100) # 학습셋, 시험셋 분할 |
| --- |

나이브 베이즈 알고리즘으로 MultinomialNB 모듈을 사용합니다. 역시 sklearn 라이브러리에서 불러올 수 있습니다.

<note/>

**MultinomialNB 모듈**

다항 분포에 대한 Naive Bayes 알고리즘입니다. 다항 분포Multinomial 외에 정규 분포Gaussian, 베르누이분포Bernoulli에 따른 NB 모듈이 있으며, 데이터 분포의 특성을 명확히 파악하기 어렵다면 이 세 가지 모듈을 모두 사용하여 가장 결과가 좋은 모델을 선택하시면 됩니다.

</>

| from sklearn.naive\_bayes import MultinomialNB |
| --- |

사용 방법은 다른 모델링 방법과 같습니다. fit()으로 학습하고 predict()로 예측하면 됩니다.

| model = MultinomialNB() # 모델 객체 생성 model.fit(x\_train, y\_train) # 학습 pred = model.predict(x\_test) # 예측 |
| --- |

예측 결과를 평가에 accuracy\_score와 함께, 이번에는 confusion\_matrix 모듈도 사용하겠습니다. confusion\_matrix는 실제값과 예측값을 비교하여 매트릭스 형태로 표현하는 모듈입니다.

<note/>

**confusion\_matrix 모듈**

행을 실제값 0과1, 열을 예측값 0과 1로 두어 2 X 2 매트릭스를 만듭니다. 이를 통해 실젯값이 0인데 예측값고 0인 경우, 실젯값은 0인데 예측값은 1인 경우, 실젯값도 1이고 예측값고 1인 경우, 실젯값은 1이나 예측값은 0인 4가지 경우를 한눈에 파악할 수 있습니다.

accuracy\_score()와 같이 실젯값, 예측값을 순서대로 넣어주면 됩니다.

<코드/>

confusion\_matrix(실제값, 예측값)

</>

</>

둘다 같은 라이브러리에 있으므로 아래 코드로 한 번에 임포트합니다.

| from sklearn.metrics import accuracy\_score, confusion\_matrix |
| --- |

정확도부터 확인하겠습니다.

| accuracy\_score(y\_test, pred) # 정확도 계산 |
| --- |

0.9856502242152466

약 98.9%로 아주 높은 예측률을 보여줍니다. 다음으로 confusion\_matrix를 사용하겠습니다.

| print(confusion\_matrix(y\_test, pred)) # 혼동행렬 출력 |
| --- |

[[965 12]

[ 4 134]]

더 깔끔한 결과물을 얻고자 print()를 함께 사용했습니다.

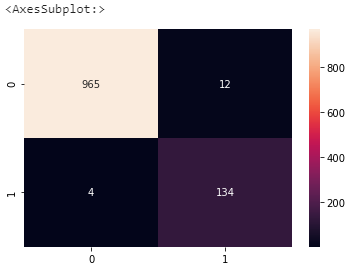
confusion\_matrix는 실제값과 예측값들이 각각 어떻게 분포되었는지를 행렬로 나타냅니다. 이를 혼동 행렬confusion matrix이라고 부릅니다. 각 셀은 다음과 같은 기준으로 분류됩니다.



❶ 실제값이 0이고 예측값도 0인 경우는 965건, ➍ 실제값이 1이고 예측값도 1인 경우는 134건입니다. 배경에 음영이 있는 영역이 정확한 예측에 해당합니다. 그리고 ❸ 1인데 0으로 잘못 예측된 경우는 4건, 반대로 ❷ 실제값은 0인데 1로 잘못 예측된 경우는 12건입니다. 흰색 배경 영역이 잘못된 예측입니다.

confusion\_matrix의 출력 결과에 어느 쪽이 0이고 1인지 별도의 레이블이 달려있지 않아 헷갈린다면, heatmap을 사용함으로써 살펴볼 수 있습니다.

| sns.heatmap(confusion\_matrix(y\_test, pred), annot=True, fmt='.0f') # ❶ 혼동행렬 히트맵 |
| --- |



❶ fmt 매개변수는 히트맵 안의 숫자가 표시되는 형식을 정의합니다. .0f는 소수점 이하 없이, .2f를 쓰면 소수점 둘째 자리까지 보여줍니다. 경우에 따라 해당 매개변수가 없을 시 과학적 표기법으로 나타날 수 있으니 필요할 때 설정해주면 되겠습니다.

이 매트릭스의 값들을 단순 산술해도 정확도를 구할 수 있습니다. 정확도 수식은 다음과 같습니다.



이수식을 적용해보겠습니다.



예상대로 accuracy\_score 계산 결과와 같은 수치입니다. 즉, 혼동 행렬은 정확도보다 더 많은 정보를 제공합니다. 각 셀에 부여된 명칭이 있을 정도로 중요한 지표입니다.

▽ 혼동 행렬

|  |  | 예측값 | |
| --- | --- | --- | --- |
|  |  | 0 | 1 |
| 실제값 | 0 | True Negative (TN)  음성을 음성으로 판단 | False Positive  (FP)  음성을 양성으로 판단 |
| 1 | False Negative  (FN)  양성을 음성으로 판단 | True Positive  (TP)  양성을 양성으로 판단 |

명칭은 참/거짓True/False와 양성/음성Positive/Negative의 조합으로 이루어졌는데, 우선 참/거짓은 정확한 예측 영익인지 잘못된 예측 영역인지에 따라 나뉩니다. 정확한 예측은 참, 잘못된 예측은 거짓입니다. 양성과 음성은 예측값을 기준으로 나뉩니다. 예측값이 1이면 양성, 0이면 음성입니다. 이 프로젝트에서는 Spam이면 양성인 1이 됩니다. 양성/음성은 예측값 기준으로 결정됩니다. 혼동하지 않도록 합시다.

False는 하나의 별칭이 더 있습니다.

| 구분 | | 예측 결과 | |
| --- | --- | --- | --- |
| 음성 | 양성 |
| 실제 정답 | 음성 | 정확함  (Correct) | 1종 오류  (Type 1 Error) |
| 양성 | 2종 오류  (Type 2 Error) | 정확함  (Correct) |

잘못된 예측된 두 영역을 오류Error라고 부릅니다. 거짓 양성은 1종 오류Type1 Error, 거짓 음성은 2종 오류Type2 Error입니다. ‘양성 1종, 음성 2종’으로 외우시면 편합니다.

<용어/>

**1종 오류(Type 1 Error)**

실제 음성인 것을 양성으로 예측하는 오류. False Positive

</>

<용어/>

**2종 오류(Type 2 Error)**

실제 양성인 것을 음성으로 예측하는 오류. False Negative

</>

1종, 2종 오류는 성격이 다른 오류이고, 때에 따라서는 둘 중 한쪽이 더 중요합니다. 예를 들어서 암 진단 예측 모델을 만들었다고 가정해봅시다. 진단에서 1종 오류는 ‘실제 암이 아닌데 암에 걸렸다고 진단한 경우’입니다. 2종 에러는 ‘실제 암에 걸렸는데 아니라고 진단한 경우’입니다. 1종 오류에서는 추가 진단이나 치료 과정으로 곧 암이 아님을 알게 될 테니 큰 문제는 아닙니다. 반면 2종 오류는 환자가 암에 걸린 사실을 모르고 지내면서 병이 악화될 겁니다. 심지어 치료 시기까지 놓칠 수도 있습니다. 이런 경우는 2종 오류가 훨씬 중요합니다.



그렇다면 스팸 문자에서는 어떨까요? 기획 의도에 따라 결정됩니다. 1종 오류에서는 스팸이 아닌데 스팸으로 분류된 경우입니다. 스팸이 아닌 중요 문자가 필터링될 수 있는 거죠. 2종 오류는 스팸인데 스팸이 아니라고 잘못 분류한 경우입니다. 스팸을 받게는 되겠지만 중요한 문자가 차단될 확률은 낮아질 겁니다.

여기에서 한걸음 더 나아가면 재현율Recall, 정밀도Precision, F-1 점수F1 score와 AUCArea Under Curve(곡선 아래 면적) 등의 개념이 등장하는데, 이는 10장(재현율, 정밀도, F-1 점수), 11장(AUC)에서 다루겠습니다.

## 7.8 이해하기 : 나이브 베이즈 모델

나이브 베이즈 알고리즘의 근간은 베이즈 정리입니다. 따라서 나이브 베이즈 알고리즘을 이해하려면 조건부확률을 구하는 베이즈 정리를 이해해야만 합니다. 수식부터 알아보겠습니다.

P는 확률을 뜻합니다. 즉, P(A)는 A가 발생할 확률입니다. |는 조건부 확률 기호입니다. P(A|B)는 B가 발생했을 때 A가 발생할 확률을 말합니다. 이제 각 항목에 대해 자세한 설명을 하겠습니다.

* : 사후확률입니다. B가 발생했을 때, A가 발생할 확률인데, 스팸문자의 예를 적용하면 B라는 특정 단어가 등장했을 때 A가 스팸일 확률입니다.
* : 사전확률입니다. B의 발생유무와 관련 없이 기본적으로 A가 발생할 확률로서, 여기서는 전체 문자 중 스팸문자의 비율이 되겠습니다.
* : 우도Likelihood 혹은 가능도라고도 부릅니다. A가 발생했을 때, B가 발생할 확률로, 여기서는 스팸 메일인 경우 B라는 특정 단어가 들어 있을 확률입니다.
* : 전체에서 B가 발생할 확률로, 전체 문자에서 B라는 특정 단어가 들어 있을 확률입니다.

<용어/>

**사후확률**

사건 A와 B가 있을 때, 사건 A가 발생한 상황에서 사건 B가 발생할 확률

</>

<용어/>

**사전확률**

위의 예에서, 사건 A와 상관없이 사건 B가 발생할 확률

</>

<용어/>

**베이즈 정리**

두 확률 변수의 사전확률과 사후확률 사이의 관계를 나타내는 정리로, 사후확률을 구할 때 쓰임

</>



혹시나 위의 설명이 잘 이해가 되지 않는다고 해도 괜찮습니다. 이번에는 그림으로 다시 한 번 살펴보겠습니다.



이 예시에서는 전체 문자 중 스팸이 70%입니다. 파란색 영역이 특정 단어 X가 포함된 경우입니다(❶+❷). 스팸에 X가 포함된 경우는 50%, 햄에 X가 포함된 경우는 10%입니다. 최종적으로 ‘특정 단어 X가 있을 때 스팸일 확률’을 구해야 합니다. 바로 파란 영역 전체 중 빗금친 파란 영역의 비율인 거죠. 그림으로 보니 굉장히 단순하군요.

파란 영역은 전체에서 특정 단어 X가 들어 있는 경우이므로, 수식에서 P(B)에 해당합니다(❶+❷). 그래서 수식의 분모 부분에 들어 있습니다. 이제 수식의 분자 부분인 빗금친 파란 영역의 크기(❶)를 계산해야 하는데, 이 또한 매우 단순합니다. 전체에서 스팸인 경우가 30%이고, 그중 B가 들어 있는 경우가 50%이므로, 0.3ｘ0.5로 구할 수 있습니다. 여기에서 0.3이 바로 P(A)가 되며, 0.5가 P(B|A)가 됩니다.

정리하면

* 스팸인 경우 : P(A) = = 0.3
* 스팸 중 특정 단어 X가 들어있는 경우 : P(B|A) = = 0.5
* 전체 중 특정 단어 X가 들어있는 경우 : P(B) = = (0.3 X 0.5) + (0.7 X 0.1) = 0.22

입니다. 앞의 그림을 이해했다면, 앞서 설명한 조건부확률을 구하는 베이즈 정리 수식이 쉽게 이해될 겁니다.

<수식에 따른 계산 결과>



## 학습 마무리

#### 되짚어보기

7.1 스팸 문자를 구분하는 모델을 만듭니다.

7.2 판다스, 넘파이, 맷플롯립, 시본 라이브러리를 임포트했습니다. 프로젝트에 쓸 예제 데이터셋을 불러옵니다.

7.3 문자 메시지에 있는 마침표, 느낌표 등의 특수 기호를 제거했습니다.

7.4 예측에 변별력이 없을 것으로 보이는 불용어를 제거했습니다.

7.5 문자형(spam과 ham) 종속변수를 숫자(1과 0)로 변경했습니다.

7.6 컴퓨터가 쉽게 이해할 수 있도록, 문장을 각 단어의 카운트 기반으로 벡터화했습니다.

7.7 나이브 베이즈 분류기를 사용하여 모델링한 결과, 98.9%의 높은 정확도를 얻었습니다.



#### 과제

스팸 메일 데이터셋을 나이브 베이즈 분류로 학습해 스팸을 예측해보았습니다. 베이즈 정리의 확실한 이해를 위하여 위의 설명 내용을 직접 손으로 풀어봅시다. MultinomialNB() 대신 GaussianNB(), BernoulliNB(binarize=True)를 사용하여 모델링해보고 결과를 비교해봅시다.

#### 핵심 용어 정리

1. **나이브 베이즈 분류기** : 조건부확률을 기반으로 하는 모델로, 자연어와 같이 변수의 개수가 많을 때 유용합니다.
2. **1종 오류**(Type 1 Error, False Positive) : 실제 음성인 것을 양성으로 예측하는 오류입니다.
3. **2종 오류**(Type 2 Error, False Negative) : 실제 양성인 것을 음성으로 예측하는 오류입니다.
4. **사후확률** : 사건 A와 B가 있을 때, 사건 A가 발생한 상황에서 사건 B가 발생할 확률입니다.
5. **사전확률** : 위의 예에서, 사건 A와 상관없이 사건 B가 발생할 확률입니다.
6. **베이즈 정리** : 두 확률 변수의 사전확률과 사후확률 사이의 관계를 나타내는 정리로, 사후확률을 구할 때 쓰입니다.

#### 새로운 함수와 라이브러리

* **string.punctuation** : 특수기호 목록 출력
* **sklearn.naive\_bayes.MultinomialNB()** : 다항분포 나이브베이즈 알고리즘
* **nltk.corpus.stopwords** : 불용어 목록
* **sklearn.feature\_extraction.text.CountVectorizer()** : 카운트 기반 벡터화 알고리즘
* **sklearn.metrics.confusion\_matrix()** : 혼동 행렬
* **CountVectorizer객체.vocabulary**\_ : 벡터화 객체에서 단어와 인덱스 확인

## 연습문제

1. 자연어 처리에서 의미 없이 빈번하게 발생하는 단어를 의미하는 것은?

① 불용어

② 특수기호

③ 벡터

④ 인덱스

2. 자연어로 된 문장을 머신러닝 알고리즘이 받아들일 수 있도록, 각 단어와 출현 빈도로 정리하는 함수는?

① get\_dummies()

② CountVectorizer()

③ StandardScaler()

④ value\_counts()

3. 다음 혼동행렬에 대한 해석으로 옳지 않은 것은?

|  |  | 예측값 | |
| --- | --- | --- | --- |
|  |  | 0 | 1 |
| 실제값 | 0 | 64 | 3 |
| 1 | 16 | 17 |

① 정확도는 81%이다.

② 1종 오류에 해당하는 것은 총 3건이다.

③ 양성을 양성으로 예측한 경우는 총 17건이다.

④ False Negative는 총 64건이다.

4. 다음 중 나이브 베이즈 알고리즘을 가장 잘 설명한 것은?

① 조건부 확률을 기반으로 한 알고리즘으로, 사전확률과 사후확률을 활용한다.

② 독립변수와 종속변수의 선형관계를 전제로 한다.

③ 독립변수의 개수가 적을 때 잘 작동하는 알고리즘이다.

④ 독립변수 간의 상관관계가 강하게 나타날 때도 잘 작동한다.

#### 정답 및 해설

1. 1

① 불용어

2. 2

① get\_dummies() ← 더미 변수로 변환하는 함수입니다.

② CountVectorizer() ← 단어를 인덱스와 출현 빈도로 벡터화하는 함수입니다.

③ StandardScaler() ← 표준화 스케일링을 위한 함수입니다.

④ value\_counts() ← 변수 내 고윳값 별 출현 횟수를 확인하는 함수입니다.

3. 4

④ False Negative는 총 64건이다. ← False Negative는 실제값 1, 예측값 0 에 해당하는 부분으로 총 16건입니다.

4. 1

① 조건부 확률을 기반으로 한 알고리즘으로 사전확률과 사후확률을 활용한다.

② 독립변수와 종속변수의 선형관계를 전제로 한다. ← 선형회귀, 로지스틱 회귀에 해당하는 설명입니다.

③ 독립변수의 개수가 적을 때 잘 작동하는 알고리즘이다. ← 독립변수의 개수가 많을 때 상대적으로 더 잘 작동합니다.

④ 독립변수 간의 상관관계가 강하게 나타날 때도 잘 작동한다. ← 독립변수 간의 상관관계가 없음을 전제로 하는 알고리즘입니다.

1. CountVectorizer. Bag of words(BOW)라고도 합니다. [↑](#footnote-ref-0)