나이브 베이스 분류기(Naive Bayes Classification)

- 베이즈 정리를 적용한 확률적 분류 알고리즘
- 모든 특성들이 독립임을 가정 (naive 가정)
- 입력 특성에 따라 3개의 분류기 존재
  - 가우시안 나이브 베이즈 분류기
  - 베르누이 나이브 베이즈 분류기
  - 다항 나이브 베이즈 분류기

나이브 베이즈 분류기의 확률 모델

- 나이브 베이즈는 조건부 확률 모델
- *N*개의 특성을 나타내는 벡터 **x**를 입력 받아 k개의 가능한 확률적 결과를 출력

$$p(C_k|x_1, \dots, x_n)$$

- 위의 식에 베이즈 정리를 적용하면 다음과 같음

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

- 위의 식에서 분자만이 출력 값에 영향을 받기 때문에 분모 부분을 상수로 취급할 수 있음

$$\begin{aligned} p(C_k|\mathbf{x}) &\propto p(C_k)p(\mathbf{x}|C_k) \\ &\propto p(C_k, x_1, \dots, x_n) \end{aligned}$$

- 위의 식을 연쇄 법칙을 사용해 다음과 같이 쓸 수 있음

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(C_k)p(x_1, \dots, x_n|C_k) \\ &= p(C_k)p(x_1|C_k)p(x_2, \dots, x_n|C_k, x_1) \\ &= p(C_k)p(x_1|C_k)p(x_2|C_k, x_1)p(x_3, \dots, x_n|C_k, x_1, x_2) \\ &= p(C_k)p(x_1|C_k)p(x_2|C_k, x_1)\dots p(x_n|C_k, x_1, x_2, \dots, x_{n-1}) \end{aligned}$$

- 나이브 베이즈 분류기는 모든 특성이 독립이라고 가정하기 때문에 위의 식을 다음과 같이 쓸 수 있음

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &\propto p(C_k)p(x_1|C_k)p(x_2|C_k)\dots p(x_n|C_k) \\ &\propto p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \end{aligned}$$

- 위의 식을 통해 나온 값들 중 가장 큰 값을 갖는 클래스가 예측 결과

$$\hat{y} = \arg\max_k p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

```
import numpy as np
import pandas as pd
```

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.datasets import fetch_covtype, fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_extraction.text import CountVectorizer, HashingVectorizer, TfidfVectorizer
from sklearn import metrics
```

```
prior = [0.45, 0.3, 0.15, 0.1]
likelihood = [[0.3, 0.3, 0.4], [0.7, 0.2, 0.1], [0.15, 0.5, 0.35], [0.6, 0.2, 0.2]]

idx = 0
for c, xs in zip(prior, likelihood):
  result = 1.

  for x in xs:
    result *= x
  result *= c

  idx += 1
  print(f"{idx}번째 클래스의 가능성: {result}")
```

```
1번째 클래스의 가능성: 0.0162
2번째 클래스의 가능성: 0.0042
3번째 클래스의 가능성: 0.0039375
4번째 클래스의 가능성: 0.0024000000000000002
```

산림 토양 데이터

- 산림 지역 토양의 특징 데이터
- 토양이 어떤 종류에 속하는지 예측
- https://archive.ics.uci.edu/ml/datasets/Covertype 에서 데이터의 자세한 설명 확인 가능

```
covtype = fetch_covtype()
print(covtype.DESCR)
```

```
Downloading https://ndownloader.figshare.com/files/5976039
.. _covtype_dataset:

Forest covertypes
-----------------

The samples in this dataset correspond to 30×30m patches of forest in the US,
collected for the task of predicting each patch's cover type,
i.e. the dominant species of tree.
There are seven covertypes, making this a multiclass classification problem.
Each sample has 54 features, described on the
`dataset's homepage <https://archive.ics.uci.edu/ml/datasets/Covertype>`__.
Some of the features are boolean indicators,
while others are discrete or continuous measurements.

**Data Set Characteristics:**
```

```
    =================   ============
    Classes                        7
    Samples total             581012
    Dimensionality                54
    Features                     int
    =================   ============

:func:`sklearn.datasets.fetch_covtype` will load the covertype dataset;
it returns a dictionary-like object
with the feature matrix in the ``data`` member
and the target values in ``target``.
The dataset will be downloaded from the web if necessary.
```

```
pd.DataFrame(covtype.data)
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	2596.0	51.0	3.0	258.0	0.0	510.0	221.0	232.0	148.0	6279.0	1.0	0.0
1	2590.0	56.0	2.0	212.0	-6.0	390.0	220.0	235.0	151.0	6225.0	1.0	0.0
2	2804.0	139.0	9.0	268.0	65.0	3180.0	234.0	238.0	135.0	6121.0	1.0	0.0
3	2785.0	155.0	18.0	242.0	118.0	3090.0	238.0	238.0	122.0	6211.0	1.0	0.0
4	2595.0	45.0	2.0	153.0	-1.0	391.0	220.0	234.0	150.0	6172.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
581007	2396.0	153.0	20.0	85.0	17.0	108.0	240.0	237.0	118.0	837.0	0.0	0.0
581008	2391.0	152.0	19.0	67.0	12.0	95.0	240.0	237.0	119.0	845.0	0.0	0.0
581009	2386.0	159.0	17.0	60.0	7.0	90.0	236.0	241.0	130.0	854.0	0.0	0.0
581010	2384.0	170.0	15.0	60.0	5.0	90.0	230.0	245.0	143.0	864.0	0.0	0.0
581011	2383.0	165.0	13.0	60.0	4.0	67.0	231.0	244.0	141.0	875.0	0.0	0.0

581012 rows × 54 columns

```
covtype.target
```

```
array([5, 5, 2, ..., 3, 3, 3], dtype=int32)
```

학습, 평가 데이터 분류

```
covtype_X = covtype.data
covtype_y = covtype.target
```

```
covtype_X_train, covtype_X_test, covtype_y_train, covtype_y_test = train_test_split(covtype_X, covt
```

```
print('전체 데이터 크기: {}'.format(covtype_X.shape))
print('학습 데이터 크기: {}'.format(covtype_X_train.shape))
```

```
print('학습 데이터 크기: {}'.format(covtype_X_train.shape))
print('평가 데이터 크기: {}'.format(covtype_X_test.shape))
```

```
전체 데이터 크기: (581012, 54)
학습 데이터 크기: (464809, 54)
평가 데이터 크기: (116203, 54)
```

전처리

전처리 전 데이터

```
covtype_df = pd.DataFrame(data=covtype_X)
covtype_df.describe()
```

	0	1	2	3	4	
count	581012.000000	581012.000000	581012.000000	581012.000000	581012.000000	581(
mean	2959.365301	155.656807	14.103704	269.428217	46.418855	23
std	279.984734	111.913721	7.488242	212.549356	58.295232	15
min	1859.000000	0.000000	0.000000	0.000000	-173.000000	
25%	2809.000000	58.000000	9.000000	108.000000	7.000000	11
50%	2996.000000	127.000000	13.000000	218.000000	30.000000	19
75%	3163.000000	260.000000	18.000000	384.000000	69.000000	33
max	3858.000000	360.000000	66.000000	1397.000000	601.000000	71

```
covtype_train_df = pd.DataFrame(data=covtype_X_train)
covtype_train_df.describe()
```

	0	1	2	3	4	
count	464809.000000	464809.000000	464809.000000	464809.000000	464809.000000	4648
mean	2959.416212	155.724046	14.106001	269.457285	46.435568	23
std	279.901146	111.906199	7.488040	212.744292	58.375535	15
min	1859.000000	0.000000	0.000000	0.000000	-173.000000	
25%	2809.000000	58.000000	9.000000	108.000000	7.000000	11
50%	2996.000000	127.000000	13.000000	218.000000	30.000000	19
75%	3163.000000	261.000000	18.000000	384.000000	69.000000	33
max	3858.000000	360.000000	66.000000	1390.000000	599.000000	71

```
covtype_test_df = pd.DataFrame(data=covtype_X_test)
```

```
covtype_test_df = pd.DataFrame(data=covtype_X_test)
covtype_test_df.describe()
```

	0	1	2	3	4	
count	116203.000000	116203.000000	116203.000000	116203.000000	116203.000000	1162(
mean	2959.161657	155.387856	14.094516	269.311945	46.352005	23-
std	280.319947	111.943881	7.489073	211.768694	57.973111	15(
min	1867.000000	0.000000	0.000000	0.000000	-164.000000	
25%	2808.000000	58.000000	9.000000	108.000000	7.000000	11(
50%	2996.000000	127.000000	13.000000	218.000000	30.000000	19(
75%	3164.000000	260.000000	18.000000	384.000000	69.000000	33(
max	3849.000000	360.000000	65.000000	1397.000000	601.000000	70(

전처리 과정

```
scaler = StandardScaler()
covtype_X_train_scale = scaler.fit_transform(covtype_X_train)
covtype_X_test_scale = scaler.transform(covtype_X_test)
```

전처리 후 데이터

- 평균은 0에 가깝게, 표준편차는 1에 가깝게 정규화

```
covtype_train_df = pd.DataFrame(data=covtype_X_train_scale)
covtype_train_df.describe()
```

```
covtype_test_df = pd.DataFrame(data=covtype_X_test_scale)
covtype_test_df.describe()
```

	0	1	2	3	4	
count	116203.000000	116203.000000	116203.000000	116203.000000	116203.000000	1162(
mean	-0.000909	-0.003004	-0.001534	-0.000683	-0.001431	
std	1.001497	1.000338	1.000139	0.995415	0.993107	
min	-3.902869	-1.391560	-1.883806	-1.266580	-3.604863	
25%	-0.540964	-0.873268	-0.681888	-0.758927	-0.675550	
50%	0.130703	-0.256680	-0.147702	-0.241874	-0.281549	
75%	0.730915	0.931817	0.520030	0.538406	0.386540	
max	3.178210	1.825423	6.796712	5.299996	9.499956	

20 Newsgroup 데이터

- 뉴스 기사가 어느 그룹에 속하는지 분류
- 뉴스 기사는 텍스트 데이터이기 때문에 특별한 전처리 과정이 필요

```
newsgroup = fetch_20newsgroups()
print(newsgroup.DESCR)
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
.. _20newsgroups_dataset:

The 20 newsgroups text dataset
------------------------------

The 20 newsgroups dataset comprises around 18000 newsgroups posts on
20 topics split in two subsets: one for training (or development)
and the other one for testing (or for performance evaluation). The split
between the train and test set is based upon a messages posted before
and after a specific date.

This module contains two loaders. The first one,
:func:`sklearn.datasets.fetch_20newsgroups`,
returns a list of the raw texts that can be fed to text feature
extractors such as :class:`sklearn.feature_extraction.text.CountVectorizer`
with custom parameters so as to extract feature vectors.
The second one, :func:`sklearn.datasets.fetch_20newsgroups_vectorized`,
returns ready-to-use features, i.e., it is not necessary to use a feature
extractor.

**Data Set Characteristics:**

    =================   ==========
    Classes                     20
    Samples total            18846
    Dimensionality               1
    Features                  text
```

```
    ================    ==========

Usage
~~~~~

The :func:`sklearn.datasets.fetch_20newsgroups` function is a data
fetching / caching functions that downloads the data archive from
the original `20 newsgroups website`_, extracts the archive contents
in the ``~/scikit_learn_data/20news_home`` folder and calls the
:func:`sklearn.datasets.load_files` on either the training or
testing set folder, or both of them::

  >>> from sklearn.datasets import fetch_20newsgroups
  >>> newsgroups_train = fetch_20newsgroups(subset='train')

  >>> from pprint import pprint
  >>> pprint(list(newsgroups_train.target_names))
  ['alt.atheism',
   'comp.graphics',
   'comp.os.ms-windows.misc',
   'comp.sys.ibm.pc.hardware',
   'comp.sys.mac.hardware',
   'comp.windows.x',
   'misc.forsale',
   'rec.autos',
   'rec.motorcycles',
   'rec.sport.baseball',
   'rec.sport.hockey',
   'sci.crypt'
```

newsgroup.data

```
["From: lerxst@wam.umd.edu (where's my thing)₩nSubject: WHAT car is this!?₩nNntp-Posting-Ho
 "From: guykuo@carson.u.washington.edu (Guy Kuo)₩nSubject: SI Clock Poll - Final Call₩nSumm
 'From: twillis@ec.ecn.purdue.edu (Thomas E Willis)₩nSubject: PB questions...₩nOrganizatio
 'From: jgreen@amber (Joe Green)₩nSubject: Re: Weitek P9000 ?₩nOrganization: Harris Compute
 'From: jcm@head-cfa.harvard.edu (Jonathan McDowell)₩nSubject: Re: Shuttle Launch Question
 'From: dfo@vttoulu.tko.vtt.fi (Foxvog Douglas)₩nSubject: Re: Rewording the Second Amendmer
 'From: bmdelane@quads.uchicago.edu (brian manning delaney)₩nSubject: Brain Tumor Treatmen
 'From: bgrubb@dante.nmsu.edu (GRUBB)₩nSubject: Re: IDE vs SCSI₩nOrganization: New Mexico S
 'From: holmes7000@iscsvax.uni.edu₩nSubject: Win 3.0 ICON HELP PLEASE!₩nOrganization: Unive
 "From: kerr@ux1.cso.uiuc.edu (Stan Kerr)₩nSubject: Re: Sigma Designs Double up??₩nArticle-
 'From: irwin@cmptrc.lonestar.org (Irwin Arnstein)₩nSubject: Re: Recommendation on Duc₩nSu
 'From: david@terminus.ericsson.se (David Bold)₩nSubject: Re: Question for those with popu
 'From: rodc@fc.hp.com (Rod Cerkoney)₩nSubject: *$G4qxF,fekVH6₩nNntp-Posting-Host: hpfcmrc
 'From: dbm0000@tm0006.lerc.nasa.gov (David B. Mckissock)₩nSubject: Re: Space Station Redes
 "From: jllee@acsu.buffalo.edu (Johnny L Lee)₩nSubject: RE:  == MOVING SALE ===₩nSummary: F
 'From: mathew <mathew@mantis.co.uk>₩nSubject: Re: <Political Atheists?₩nOrganization: Man
 'From: ab@nova.cc.purdue.edu (Allen B)₩nSubject: Re: TIFF: philosophical significance of 4
 'From: CPKJP@vm.cc.latech.edu (Kevin Parker)₩nSubject: Insurance Rates on Performance Cars
 'From: ritley@uimrl7.mrl.uiuc.edu ()₩nSubject: SEEKING THERMOCOUPLE AMPLIFIER CIRCUIT₩nRep
 'From: abarden@tybse1.uucp (Ann Marie Barden)₩nSubject: X-Terminal Config. file question₩n
 'From: keith@cco.caltech.edu (Keith Allan Schneider)₩nSubject: Re: <<Pompous ass₩nOrganiza
 'From: leunggm@odin.control.utoronto.ca (Gary Leung)₩nSubject: Re: NHL Team Captains₩nOrga
 'From: rpwhite@cs.nps.navy.mil (rpwhite)₩nSubject: Re: Catalog of Hard-to-Find PC Enhancer
 'From: csyphers@uafhp..uark.edu (Chris Syphers)₩nSubject: Re: ?? DOS font size in windows'
 'From: nodine@lcs.mit.edu (Mark H. Nodine)₩nSubject: Re: Quadra SCSI Problems???₩nKeywords
 "From: kph2q@onyx.cs.Virginia.EDU (Kenneth Hinckley)₩nSubject:  VOICE INPUT -- vendor in
 'From: nagle@netcom.com (John Nagle)₩nSubject: Re: What do Nuclear Site₩'s Cooling Towers
 "From: r4938585@joplin.biosci.arizona.edu (Doug Roberts)₩nSubject: Re: NL vs. AL?₩nOrganiz
```

```
'From: jonh@david.wheaton.edu (Jonathan Hayward)₩nSubject: Re: Pantheism & Environmentalis
'From: jimf@centerline.com (Jim Frost)₩nSubject: Re: Is car saftey important?₩nOrganizatio
"From: mrh@iastate.edu (Michael R Hartman)₩nSubject: Re: Car Stereo Stolen?₩nOrganization
"Subject: Teenage acne₩nFrom: pchurch@swell.actrix.gen.nz (Pat Churchill)₩nOrganization: A
'From: xandor@unixg.ubc.ca (John Gilbert )₩nSubject: Re: Exploding TV!₩nOrganization: The
"From: ayr1@cunixa.cc.columbia.edu (Amir Y Rosenblatt)₩nSubject: Re: Israeli Expansion-lus
'From: joec@hilbert.cyprs.rain.com ( Joe Cipale)₩nSubject: Re: Clayton Need not Retract₩n(
'From: dchhabra@stpl.ists.ca (Deepak Chhabra)₩nSubject: Re: Goalie masks₩nNntp-Posting-Hos
'From: static@iat.holonet.net (Joe Ehrlich)₩nSubject: Re: BMW MOA members read this!₩nOrga
'From: ebrandt@jarthur.claremont.edu (Eli Brandt)₩nSubject: Re: Do we need the clipper for
'From: behanna@syl.nj.nec.com (Chris BeHanna)₩nSubject: Re: Should liability insurance be
'From: bressler@iftccu.ca.boeing.com (Rick Bressler)₩nSubject: Re: Gun Lovers (was Re: My
'From:  (Sean Garrison)₩nSubject: Re: Bonilla₩nNntp-Posting-Host: berkeley-kstar-node.net
"From: root@ncube.com (Operator)₩nSubject: Re: Which fax modem is the best?₩nNntp-Posting-
"From: ab245@cleveland.Freenet.Edu (Sam Latonia)₩nSubject: Re: Need phone number for Weste
"From: paul@csd4.csd.uwm.edu (Paul R Krueger)₩nSubject: Brewer bullpen rocked again...₩nOr
"From: cmeyer@bloch.Stanford.EDU (Craig Meyer)₩nSubject: Re: Jack Morris₩nOrganization: DS
"From: Robert Everett Brunskill <rb6t+@andrew.cmu.edu>₩nSubject: Re: $$$ to fix TRACKBALL₩
'From: vng@iscs.nus.sg₩nSubject: Wyse 60 Terminal Emulator₩nReply-To: VNG@ISCS.NUS.SG₩nOrg
"From: speedy@engr.latech.edu (Speedy Mercer)₩nSubject: Re: Looking for MOVIES w/ BIKES₩n(
'From: tg@cs.toronto.edu (Tom Glinos)₩nSubject: 12V to 3V and 48V at 3A₩nOrganization: Dep
'From: 18084TM@msu.edu (Tom)₩nSubject: Golden & Space ages₩nX-Added: Forwarded by Space D
'From: johnc@crsa.bu.edu (John Collins)₩nSubject: Problem with MIT-SHM₩nOrganization: Bos
"From: dlecoint@garnet.acns.fsu.edu (Darius_Lecointe)₩nSubject: Re: Sabbath Admissions 5o
'From: yuting@Xenon.Stanford.EDU (Eugene Y. Kuo)₩nSubject: Any updated Canon BJ-200 drive
'From: km@cs.pitt.edu (Ken Mitchum)₩nSubject: Re: Patient-Physician Diplomacy₩nArticle-I.D
'From: sera@zuma.UUCP (Serdar Argic)₩nSubject: Day and night Armenians were rounding up ma
"From: cme@ellisun.sw.stratus.com (Carl Ellison)₩nSubject: Re: Clipper Crypto₩nOrganizatio
'From: eliot@lanmola.engr.washington.edu (eliot)₩nSubject: Re: Improvements in Automatic T
"From: d_jaracz@oz.plymouth.edu (David R. Jaracz)₩nSubject: Re: Octopus in Detroit?₩nOrgan
```

```
newsgroup.target_names
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 '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']
```

학습, 평가 데이터 분류

```
newsgroup_train = fetch_20newsgroups(subset='train')
```

```
newsgroup_test = fetch_20newsgroups(subset='test')
```

```
X_train, y_train = newsgroup_train.data, newsgroup_train.target
X_test, y_test = newsgroup_test.data, newsgroup_test.target
```

벡터화

- 텍스트 데이터는 기계학습 모델에 입력 할 수 없음
- 벡터화는 텍스트 데이터를 실수 벡터로 변환해 기계학습 모델에 입력 할 수 있도록 하는 전처리 과정
- Scikit-learn에서는 Count, Tf-idf, Hashing 세가지 방법을 지원

CountVectorizer

- 문서에 나온 단어의 수를 세서 벡터 생성

```
count_vectorizer = CountVectorizer()
```

```
X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)
```

데이터를 희소 행렬 형태로 표현

```
X_train_count
```

```
<11314x130107 sparse matrix of type '<class 'numpy.int64'>'
        with 1787565 stored elements in Compressed Sparse Row format>
```

```
for v in X_train_count[0]:
  print(v)
```

```
  (0, 56979)    3
  (0, 75358)    2
  (0, 123162)   2
  (0, 118280)   2
  (0, 50527)    2
  (0, 124031)   2
  (0, 85354)    1
  (0, 114688)   1
  (0, 111322)   1
  (0, 123984)   1
  (0, 37780)    5
  (0, 68532)    3
  (0, 114731)   5
  (0, 87620)    1
  (0, 95162)    1
  (0, 64095)    1
  (0, 98949)    1
  (0, 90379)    1
  (0, 118983)   1
```

```
  (0, 89362)    3
  (0, 79666)    1
  (0, 40998)    1
  (0, 92081)    1
  (0, 76032)    1
  (0, 4605)     1
  :     :
  (0, 37565)    1
  (0, 113986)   1
  (0, 83256)    1
  (0, 86001)    1
  (0, 51730)    1
  (0, 109271)   1
  (0, 128026)   1
  (0, 96144)    1
  (0, 78784)    1
  (0, 63363)    1
  (0, 90252)    1
  (0, 123989)   1
  (0, 67156)    1
  (0, 128402)   2
  (0, 62221)    1
  (0, 57308)    1
  (0, 76722)    1
  (0, 94362)    1
  (0, 78955)    1
  (0, 114428)   1
  (0, 66098)    1
  (0, 35187)    1
  (0, 35983)    1
  (0, 128420)   1
  (0, 86580)    1
```

HashingVectorizer

- 각 단어를 해쉬 값으로 표현
- 미리 정해진 크기의 벡터로 표현

```
hash_vectorizer = HashingVectorizer(n_features=1000)
```

```
X_train_hash = hash_vectorizer.fit_transform(X_train)
X_test_hash = hash_vectorizer.transform(X_test)
```

```
X_train_hash
```

```
<11314x1000 sparse matrix of type '<class 'numpy.float64'>'
        with 1550687 stored elements in Compressed Sparse Row format>
```

```
print(X_train_hash[0])
```

```
  (0, 80)       -0.0642824346533225
  (0, 108)      0.0642824346533225
  (0, 111)      -0.128564869306645
  (0, 145)      0.0642824346533225
  (0, 158)      0.0642824346533225
```

```
  (0, 159)	-0.0642824346533225
  (0, 161)	0.0642824346533225
  (0, 165)	-0.0642824346533225
  (0, 171)	0.0642824346533225
  (0, 182)	0.0642824346533225
  (0, 195)	-0.0642824346533225
  (0, 196)	0.19284730395996752
  (0, 205)	-0.0642824346533225
  (0, 209)	0.0642824346533225
  (0, 234)	0.0642824346533225
  (0, 237)	0.0642824346533225
  (0, 248)	0.0642824346533225
  (0, 265)	0.19284730395996752
  (0, 274)	0.0642824346533225
  (0, 277)	0.19284730395996752
  (0, 284)	-0.0642824346533225
  (0, 286)	-0.0642824346533225
  (0, 296)	0.0642824346533225
  (0, 362)	-0.0642824346533225
  (0, 364)	-0.0642824346533225
  :	:
  (0, 739)	0.0
  (0, 761)	-0.0642824346533225
  (0, 766)	0.0642824346533225
  (0, 800)	-0.0642824346533225
  (0, 812)	-0.0642824346533225
  (0, 842)	0.0642824346533225
  (0, 848)	-0.0642824346533225
  (0, 851)	0.0642824346533225
  (0, 863)	-0.0642824346533225
  (0, 881)	0.0642824346533225
  (0, 892)	0.0642824346533225
  (0, 897)	0.0
  (0, 899)	0.0642824346533225
  (0, 906)	-0.0642824346533225
  (0, 918)	0.128564869306645
  (0, 926)	-0.0642824346533225
  (0, 935)	0.0642824346533225
  (0, 939)	-0.0642824346533225
  (0, 951)	-0.0642824346533225
  (0, 958)	-0.38569460791993504
  (0, 960)	0.0642824346533225
  (0, 968)	-0.0642824346533225
  (0, 987)	-0.0642824346533225
  (0, 996)	0.0642824346533225
  (0, 997)	-0.128564869306645
```

TfidfVectorizer

- 문서에 나온 단어 빈도(term frequency)와 역문서 빈도(inverse document frequency)를 곱해서 구함
- 각 빈도는 일반적으로 로그 스케일링 후 사용
- $tf(t,d) = log(f(t,d) + 1)$
- $idf(t,D) = \frac{|D|}{|d \in D : t \in d| + 1}$
- $tfidf(t,d,D) = tf(t,d) \times idf(t,D)$

```
tfidf_vectorizer = TfidfVectorizer()
```

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
X_train_tfidf
```

```
<11314x130107 sparse matrix of type '<class 'numpy.float64'>'
        with 1787565 stored elements in Compressed Sparse Row format>
```

```
for v in X_train_tfidf[0]:
  print(v)
```

```
  (0, 86580)    0.13157118714240987
  (0, 128420)   0.04278499079283093
  (0, 35983)    0.03770448563619875
  (0, 35187)    0.09353930598317124
  (0, 66098)    0.09785515708314481
  (0, 114428)   0.05511105154696676
  (0, 78955)    0.05989856888061599
  (0, 94362)    0.055457031390147224
  (0, 76722)    0.06908779999621749
  (0, 57308)    0.1558717009157704
  (0, 62221)    0.02921527992427867
  (0, 128402)   0.05922294083277842
  (0, 67156)    0.07313443922740179
  (0, 123989)   0.08207027465330353
  (0, 90252)    0.031889368795417566
  (0, 63363)    0.08342748387969037
  (0, 78784)    0.0633940918806495
  (0, 96144)    0.10826904490745741
  (0, 128026)   0.060622095889758885
  (0, 109271)   0.10844724822064673
  (0, 51730)    0.09714744057976722
  (0, 86001)    0.07000411445838192
  (0, 83256)    0.08844382496462173
  (0, 113986)   0.17691750674853082
  (0, 37565)    0.03431760442478462
  :     :
  (0, 4605)     0.06332603952480323
  (0, 76032)    0.019219463052223086
  (0, 92081)    0.09913274493911223
  (0, 40998)    0.0780136819691811
  (0, 79666)    0.10936401252414274
  (0, 89362)    0.06521174306303763
  (0, 118983)   0.037085978050619146
  (0, 90379)    0.019928859956645867
  (0, 98949)    0.16068606055394932
  (0, 64095)    0.03542092427131355
  (0, 95162)    0.03447138409326312
  (0, 87620)    0.035671863140815795
  (0, 114731)   0.14447275512784058
  (0, 68532)    0.07325812342131596
  (0, 37780)    0.38133891259493113
  (0, 123984)   0.036854292634593756
  (0, 111322)   0.01915671802495043
  (0, 114688)   0.06214070986309586
  (0, 85354)    0.03696978508816316
```

```
  (0, 124031)   0.10798795154169122
  (0, 50527)    0.054614286588587246
  (0, 118280)   0.2118680720828169
  (0, 123162)   0.2597090245735688
  (0, 75358)    0.35383501349706165
  (0, 56979)    0.057470154074851294
```

가우시안 나이브 베이즈

- 입력 특성이 가우시안(정규) 분포를 갖는다고 가정

```
model = GaussianNB()
model.fit(covtype_X_train_scale, covtype_y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
predict = model.predict(covtype_X_train_scale)
acc = metrics.accuracy_score(covtype_y_train, predict)
f1 = metrics.f1_score(covtype_y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 score: {}'.format(f1))
```

```
Train Accuracy: 0.08834596576228085
Train F1 score: [0.04090544 0.0180553  0.33522871 0.14011136 0.04358473 0.0689769
 0.23654715]
```

```
predict = model.predict(covtype_X_test_scale)
acc = metrics.accuracy_score(covtype_y_test, predict)
f1 = metrics.f1_score(covtype_y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 score: {}'.format(f1))
```

```
Test Accuracy: 0.08661566396736745
Test F1 score: [0.03855795 0.01753252 0.33402699 0.13126341 0.04248857 0.06550218
 0.23386621]
```

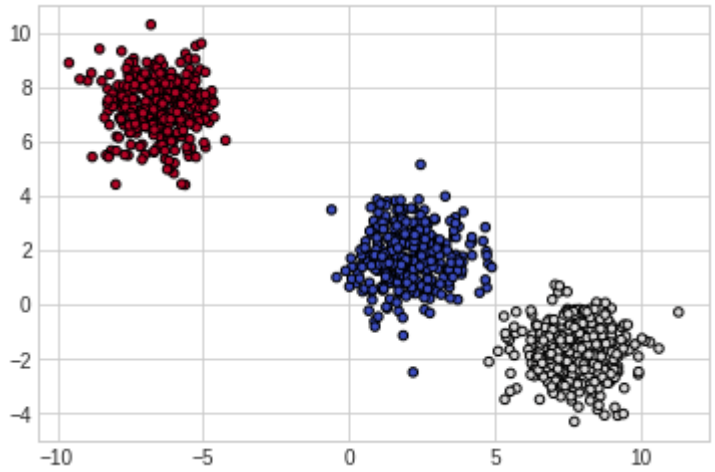
```
import matplotlib.pyplot as plt
plt.style.use(['seaborn-whitegrid'])
from sklearn.datasets import make_blobs
```

```
def make_meshgrid(x, y, h=.02):
  x_min, x_max = x.min()-1, x.max()+1
  y_min, y_max = y.min()-1, y.max()+1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                       np.arange(y_min, y_max, h))

  return xx, yy

def plot_contours(clf, xx, yy, **params):
  Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
  Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)
  out = plt.contourf(xx, yy, Z, **params)

  return out
```

```
X, y = make_blobs(n_samples=1000)
```

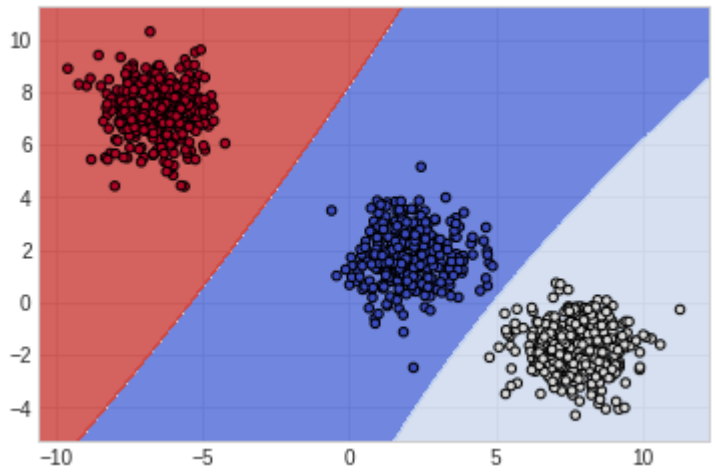
```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



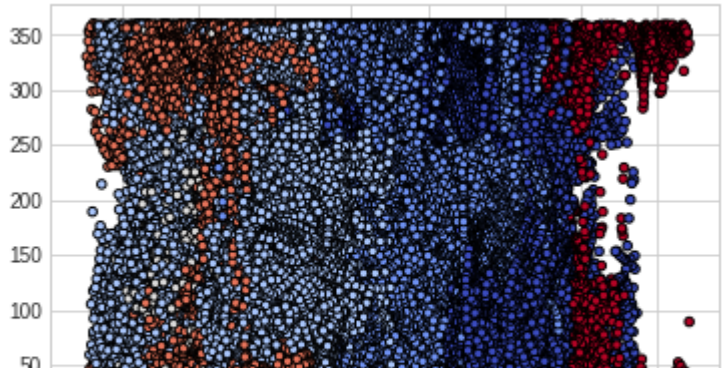

```
model = GaussianNB()
model.fit(X, y)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
xx, yy = make_meshgrid(X[:, ], X[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
plt.scatter(covtype_X[:, 0], covtype_X[:, 1], c=covtype_y, cmap=plt.cm.coolwarm, s=20, edgecolors='
```



베르누이 나이브 베이즈

- 입력 특성이 베르누이 분포에 의해 생성된 이진 값을 갖는 다고 가정

학습 및 평가 (Count)

```
model = BernoulliNB()
model.fit(X_train_count, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
predict = model.predict(X_train_count)
acc = metrics.accuracy_score(y_train, predict)
f1 = metrics.f1_score(y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 Score: {}'.format(f1))
```

```
Train Accuracy: 0.7821283365741559
Train F1 Score: [0.80096502 0.8538398  0.13858268 0.70686337 0.85220126 0.87944493
 0.51627694 0.84532672 0.89064976 0.87179487 0.94561404 0.91331546
 0.84627832 0.89825848 0.9047619  0.79242424 0.84693878 0.84489796
 0.67329545 0.14742015]
```

```
predict = model.predict(X_test_count)
acc = metrics.accuracy_score(y_test, predict)
f1 = metrics.f1_score(y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 Score: {}'.format(f1))
```

```
Test Accuracy: 0.6307753584705258
Test F1 Score: [0.47086247 0.60643564 0.01       0.56014047 0.6953405  0.70381232
 0.44829721 0.71878646 0.81797753 0.81893491 0.90287278 0.74794521
 0.61647059 0.64174455 0.76967096 0.63555114 0.64285714 0.77971474
 0.31382979 0.00793651]
```

학습 및 평가 (Hash)

```
model = BernoulliNB()
model.fit(X_train_hash, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
predict = model.predict(X_train_hash)
acc = metrics.accuracy_score(y_train, predict)
f1 = metrics.f1_score(y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 Score: {}'.format(f1))
```

```
Train Accuracy: 0.5951917977726711
Train F1 Score: [0.74226804 0.49415205 0.45039019 0.59878155 0.57327935 0.63929619
 0.35390947 0.59851301 0.72695347 0.68123862 0.79809524 0.70532319
 0.54703833 0.66862745 0.61889927 0.74707471 0.6518668  0.60485269
 0.5324165  0.54576271]
```

```
predict = model.predict(X_test_hash)
acc = metrics.accuracy_score(y_test, predict)
f1 = metrics.f1_score(y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 Score: {}'.format(f1))
```

```
Test Accuracy: 0.4430430164630908
Test F1 Score: [0.46678636 0.33826638 0.29391892 0.45743329 0.41939121 0.46540881
 0.34440068 0.46464646 0.62849873 0.53038674 0.63782051 0.55251799
 0.32635983 0.34266886 0.46105919 0.61780105 0.46197991 0.54591837
 0.27513228 0.3307888 ]
```

학습 및 평가 (Tf-idf)

```
model = BernoulliNB()
model.fit(X_train_tfidf, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
predict = model.predict(X_train_tfidf)
acc = metrics.accuracy_score(y_train, predict)
f1 = metrics.f1_score(y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 Score: {}'.format(f1))
```

```
Train Accuracy: 0.7821283365741559
Train F1 Score: [0.80096502 0.8538398  0.13858268 0.70686337 0.85220126 0.87944493
 0.51627694 0.84532672 0.89064976 0.87179487 0.94561404 0.91331546
 0.84627832 0.89825848 0.9047619  0.79242424 0.84693878 0.84489796
 0.67329545 0.14742015]
```

```
predict = model.predict(X_test_tfidf)
```

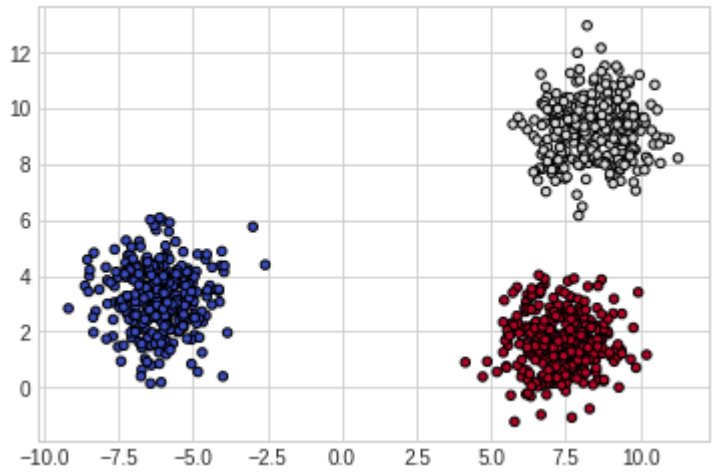
```
acc = metrics.accuracy_score(y_test, predict)
f1 = metrics.f1_score(y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 Score: {}'.format(f1))
```

```
Test Accuracy: 0.6307753584705258
Test F1 Score: [0.47086247 0.60643564 0.01       0.56014047 0.6953405  0.70381232
 0.44829721 0.71878646 0.81797753 0.81893491 0.90287278 0.74794521
 0.61647059 0.64174455 0.76967096 0.63555114 0.64285714 0.77971474
 0.31382979 0.00793651]
```

시각화

```
X, y = make_blobs(n_samples=1000)
```

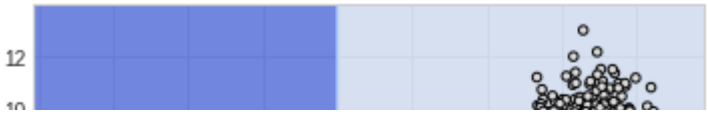
```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = BernoulliNB()
model.fit(X, y)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

```
xx, yy = make_meshgrid(X[:, 0], X[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```



다항 나이브 베이즈

- 입력 특성이 다항분포에 의해 생성된 빈도수 값을 갖는 다고 가정

학습 및 평가 (Count)

```
model = MultinomialNB()
model.fit(X_train_count, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
predict = model.predict(X_train_count)
acc = metrics.accuracy_score(y_train, predict)
f1 = metrics.f1_score(y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 Score: {}'.format(f1))
```

```
Train Accuracy: 0.9245182959165635
Train F1 Score: [0.95228426 0.904      0.25073746 0.81402003 0.96669513 0.88350983
 0.90710383 0.97014925 0.98567818 0.99325464 0.98423237 0.95399516
 0.95703454 0.98319328 0.98584513 0.95352564 0.97307002 0.97467249
 0.95157895 0.86526946]
```

```
predict = model.predict(X_test_count)
acc = metrics.accuracy_score(y_test, predict)
f1 = metrics.f1_score(y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 Score: {}'.format(f1))
```

```
Test Accuracy: 0.7728359001593202
Test F1 Score: [0.77901431 0.7008547  0.00501253 0.64516129 0.79178082 0.73370166
 0.76550681 0.88779285 0.93951094 0.91390728 0.94594595 0.78459938
 0.72299169 0.84635417 0.86029412 0.80846561 0.78665077 0.89281211
 0.60465116 0.48695652]
```

학습 및 평가 (Tf-idf)

```
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
predict = model.predict(X_train_tfidf)
```

```
acc = metrics.accuracy_score(y_train, predict)
f1 = metrics.f1_score(y_train, predict, average=None)

print('Train Accuracy: {}'.format(acc))
print('Train F1 Score: {}'.format(f1))
```

```
Train Accuracy: 0.9326498143892522
Train F1 Score: [0.87404162 0.95414462 0.95726496 0.92863002 0.97812773 0.97440273
 0.91090909 0.97261411 0.98659966 0.98575021 0.98026316 0.94033413
 0.9594478  0.98032506 0.97755611 0.77411003 0.93506494 0.97453907
 0.90163934 0.45081967]
```

```
predict = model.predict(X_test_tfidf)
acc = metrics.accuracy_score(y_test, predict)
f1 = metrics.f1_score(y_test, predict, average=None)

print('Test Accuracy: {}'.format(acc))
print('Test F1 Score: {}'.format(f1))
```

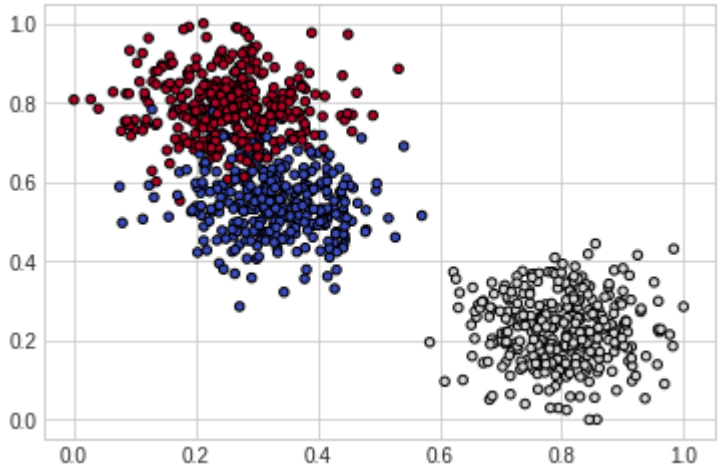
```
Test Accuracy: 0.7738980350504514
Test F1 Score: [0.63117871 0.72       0.72778561 0.72104019 0.81309686 0.81643836
 0.7958884  0.88135593 0.93450882 0.91071429 0.92917167 0.73583093
 0.69732938 0.81907433 0.86559803 0.60728118 0.76286353 0.92225201
 0.57977528 0.24390244]
```

시각화

```
X, y = make_blobs(n_samples=1000)
```

```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```




```
model = MultinomialNB()
model.fit(X, y)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
xx, yy = make_meshgrid(X[:, 0], X[:, 1])
plot_contours(model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k');
```