AI를 활용한 데이터 과학 심화

- Scikit-learn 알고리즘과 예측모형에 관하여 -

2021.10.24

저작 : 조상구(趙 商 九)

내용

머릿말

빅데이터와 인공지능을 핵심으로 하는 지능정보 기술은 현재 우리 삶의 다양한 분야에 보편적으로 활용되어 새로운 가치를 창출하고 있다. 이 교재는 필자가 연구과제, 학교, 현장, 세미나 등에서 다루었던 연구 주제를 간추려 핵심적인 사항을 정리한 것이다. 파이썬의 기초문법, 넘파이와 판다에 대한 기초적인 지식을 가진 독자를 대상으로 하고 있다. 관련 코드는 깃허브 'https://github.com/ancestor9/AI_aSSIST_2022' 이다.

사이킷런(scikit-learn, scikit-learn.org)의 내용을 중심으로 지도학습 분야의 예측 알고리즘의 작동 방식을 설명하여 교차 검증, 하이퍼파라미터 무작위 서치, 파이프라인 등의 개념과 적용 방법을 쉽게 소개하였다. 또한 예측 모형의 학습 곡선, 혼동표, 분석보고서, ROC, PR곡선, 향상도 곡선, 보정 곡선, 로그 손실 등의 성능 지표의 소개와 모델의 특성에 맞는 하이퍼파라미터 튜닝 방법을 설명한다. 데이터 전처리과정으로 종속변수에 영향을 끼치는 특성 변수를 추출하거나 선택하기 위한 특성공학 분야도 간단하게 다루었다. 예측에 영향을 미치는 입력변수를 정제, 가공하거나 새로운 파생 변수를 생성하고 변형하는 파이썬 사이킷런(scikit-learn)의 다양한 모듈을 소개하고 있다.

지도 학습의 머신 러닝 분야의 K-최근접 이웃, 가우시안 나이브 베이지안, 회귀생성과 로지스틱회귀, 서포트벡터의 알고리즘을 직관적으로 이해할 수 있도록 정리하였다. 의사결정 트리, 베깅과 랜덤포레스트, 그레디언트 부스팅, 스태킹의 앙상블 모델 개념을 복잡한 수식 없이 쉽게 이해 할 수 있도록 예제 프로그램으로 설명하였다..

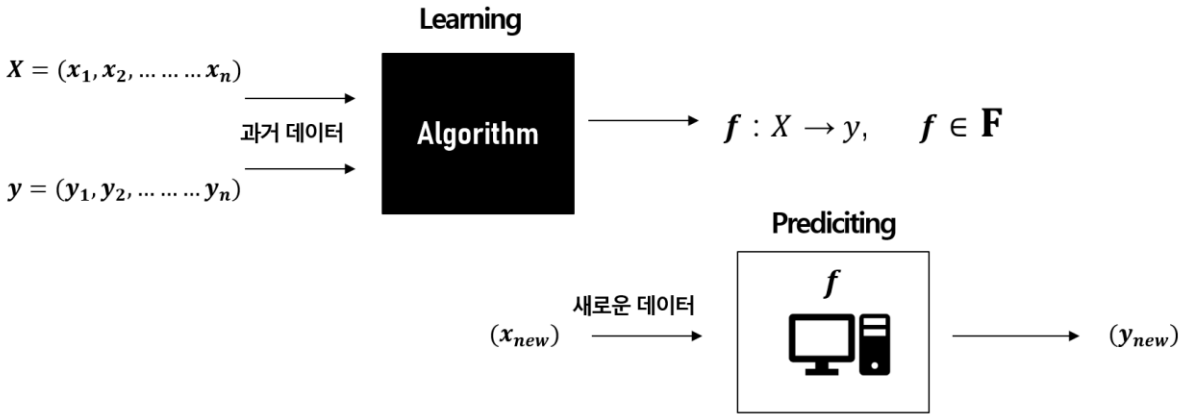
비지도 학습의 Kmeans, DBSCAN, GMM 등의 클러스터링 알고리즘을 학습한다. 지도학습과 비지도학습 방법을 혼합한 준지도 학습 방법과 이상치 탐지 알고리즘도 소개하였다. 소수 범주 유형을 포함하는 불균형데이터에서 예측 성능을 높이기 위한 손실함수 조정, 오버/언더/복합샘플링 기법을 학습한다. 마지막으로 레고를 조립하듯이 쉽게 코드를 만들 수 있는 로우 코드의 pycaret을 소개한다.

1. 머신러닝

■ 머신 러닝 이해

머신 러닝이란 과거 데이터를 컴퓨터에게 학습한 패턴인식(pattern recognition)으로 새로운 데이터를 컴퓨터가 스스로 예측하게 하는 컴퓨터 알고리즘(algorithm)의 연구이다.

컴퓨터는 과거 데이터($\mathbf{X}, \mathbf{y}$)를 다양한 알고리즘으로 학습(learning)하여 스스로 찾아낸 함수(f)로 한번도 보지 못한 데이터(x_{new})를 입력으로 미래 결과(y_{new})를 예측하게 된다(그림[]). 과거 데이터에 대한 학습 경험으로 형성된 일반화된 가설 함수(f)은 예측 모델이며, 데이터 특성과 알고리즘의 다양성에 따라 무수히 많은 예측 모형이 존재할 수 있다.



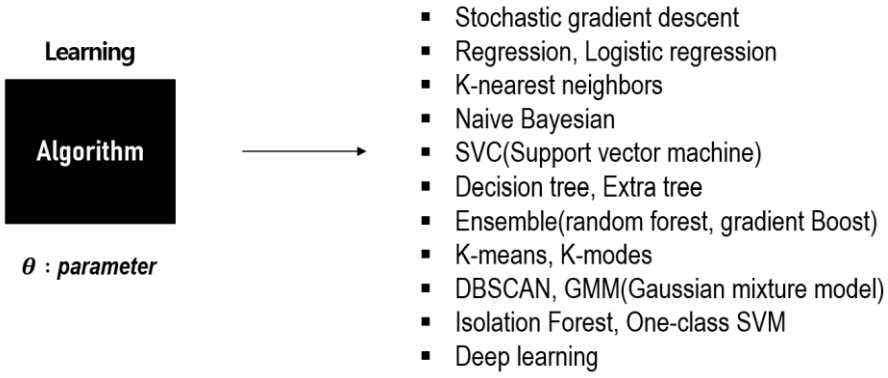
머신 러닝의 목적은 과거 데이터에 존재하는 규칙성(regularities)을 발견하여 새로운 데이터를 예측하는 것이다. 연구자는 예측 모형에 투입되는 데이터 전처리 과정(Data preprocessing) 수행과 다양한 알고리즘 적용은 Scikit-learn의 다양한 API를 이해하고 활용하면 된다.

■ 데이터, 알고리즘, 모델

개별데이터는 인스턴스(Instance)이며 입력 변수(Input), $\mathbf{X}$를 'Feature'라고 한다. 출력 변수(Output), $\mathbf{y}$를 'Target'이라고 한다. 머신 러닝은 과거 자료를 통해 일반화 모델을 만드는 귀납적 방식이다.

알고리즘은 입력 자료의 pattern을 인식하기 위한 가설 함수(f)로 데이터를 학습한다. 즉 모델, Model = Algorithm(Data)로 영어 표현은 'The Algorithm Fits(Learns) Data'이다..

사이킷런에서 제공하는 다양한 알고리즘(그림[])을 사용하여 데이터를 전달받아 예측 성능이 최대가 되는 파라미터(θ)를 가진 예측 모델($f(x;\ \theta)$)을 찾는다. 예측 성능 지표는 연구자가 연구 목적에 따라 설정하게 된다. 파라미터(θ)는 단순선형회귀 모델의 절편과 기울기 값이다.



■ 머신 러닝 종류

머신 러닝은 target의 유무에 따라 지도(Supervised)와 비지도(Unsupervised) 학습으로 구분한다. 지도학습(Supervised Learning)은 개별 인스턴스의 target이 레이블(Label)이 있으면 분류(Classification)라고 하며, 연속적인 값이면 회귀(Regression)라고 한다. 지도 학습의 분류 모델의 레이블이 우량과 불량, 적합과 부적합, 생존과 사망 등과 같이 두 가지 경우만 존재하면 이진 분류(Binary classification)로, 그 이상인 경우는 다중 분류(Multiclass classification)으로 구분한다. 비지도 학습(Unsupervised Learning)은 target이 없는 경우로 군집분석(Clustering)과 차원 축소(Dimensionality reduction)으로 구분된다.

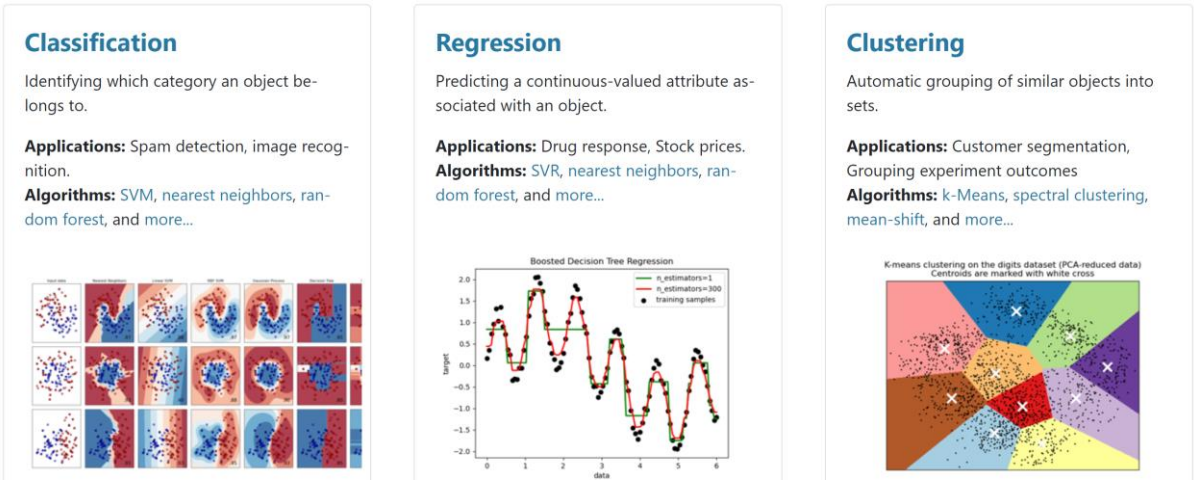
$$(x_1, x_2, \dots\dots\dots x_n)$$
$$(y_1, y_2, \dots\dots\dots y_n)$$

$$\begin{cases} y = \{0, 1\} & \rightarrow \text{Binary Classification} \\ y = \{0, 1, 2 \dots\dots . K\} & \rightarrow \text{MultiClass Classification} \\ y = Real\ Number & \rightarrow \text{Regression} \end{cases} \quad \rightarrow \textit{Supervised Learning}$$

$$(x_1, x_2, \dots\dots\dots x_n)$$

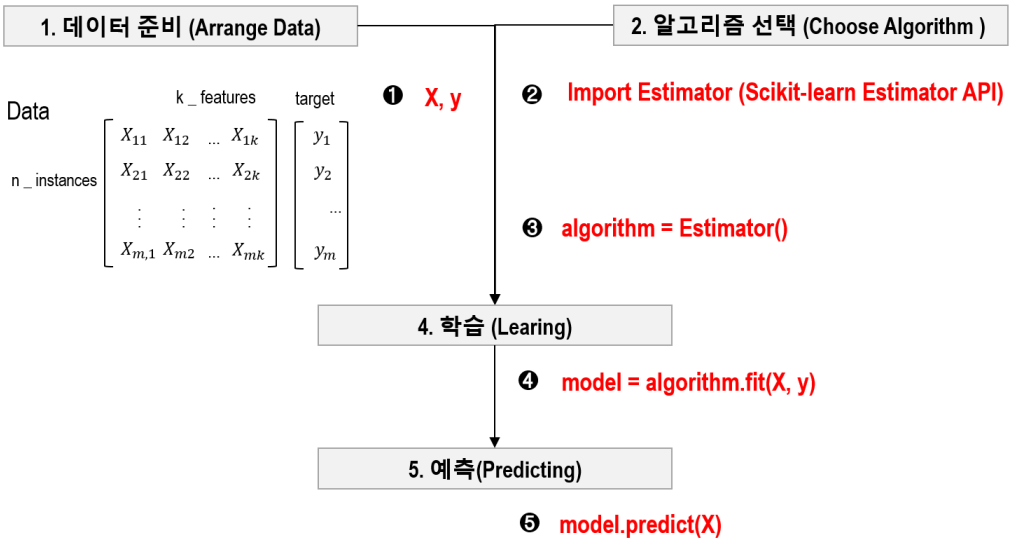
$$\begin{cases} (x_1, x_2, \dots\dots\dots x_n) & \rightarrow \text{Clustering} \\ (x_1, x_2, \dots x_k),\ k \leq n & \rightarrow \text{Dimensionality Reduction} \end{cases} \quad \rightarrow \textit{Unsupervised Learning}$$

사이킷런의 홈페이지에도 Classification, Regression, Clustering으로 구분하고 있다(그림[]).



■ 머신러닝 핵심 프로세스

지도 학습의 분류 모델을 예를 들면, 머신 러닝의 핵심 프로세스의 작동방식은 데이터와 알고리즘을 선택하여 컴퓨터에게 학습시켜 새로운 데이터를 예측하는 것이다(그림[]).



데이터 준비(Arrange data)단계에서 입력변수는 행렬(matrix), 종속변수는 벡터(vector)의 자료 형태로 파이썬의 넘파이(numpy), 판다(pandas)로 준비한다. 알고리즘 선택(Choose Algorithm)에서는 자료에 적용할 사이킷런 분류기(Classifier, 모형)를 선택하고 해당 모듈을 객체화하여 변수로 저장한다. 다음 단계는 모델 학습(Learning)과 예측(Predicting)으로 데이터에 알고리즘을 적용하여 최적의 파라미터를 찾아내어 학습한 모델로 새로운 데이터를 예측한다. *

강의 교재에서 모든 주피터 스크립트에서 동일하게 사용될 관련 모듈을 호출한다.

관련 모듈 호출

```
from IPython.core.interactiveshell import InteractiveShell # 모든 명령어 출력
InteractiveShell.ast_node_interactivity = 'all'
from IPython.core.display import display, HTML # 화면 좌우 크기 100%
display(HTML("<style>.container { width:100% !important; }</style>"))
import warnings # 파이썬 버전에 따른 오류 출력 방지
warnings.filterwarnings('ignore')

import pandas as pd # pandas
import numpy as np # numphy
import matplotlib.pyplot as plt # 시각화 matplob
import seaborn as sns; sns.set() # 시각화 seaborn
get_ipython().run_line_magic('matplotlib', 'inline') # 시각화 그림 자동 화면 출력
plt.rcParams['font.family'] = 'Malgun Gothic'  # 한글 표시
plt.rcParams['axes.unicode_minus'] = False  #한글 폰트 사용시 마이너스 폰트 깨짐 해결
```

데이터의 현황파악을 위해 데이터프레임과 사전형 자료, 리스트 축약문(list comprehension)으로 한눈에 일목요연하게 파악하는 script을 소개한다..

Boston 주택가격 데이터세트 I

```
from sklearn.datasets import load_boston
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)

pd.DataFrame({'자료 내용(contents)':{col:df[col].unique() for col in df},
              '데이터형태(dtypes)':{col:df[col].dtype for col in df},
              '고유값 수(nunique)':{col:len(df[col].unique()) for col in df},
              '결측치 비율(%)':{col:str(round(sum(df[col].isna())/len(df),2))+'%' for col in df},
              '중복건수':{col:sum(df[col].duplicated()) for col in df},
              'nan 비율':{col:int(df[df[col] == 'nan'].shape[0]/len(df)*100) for col in df}
             })
```

	자료 내용(contents)	데이터형태(dtypes)	고유값 수(nunique)	결측치 비율(%)	중복건수	nan 비율
CRIM	[0.00632, 0.02731, 0.02729, 0.03237, 0.06905, ...	float64	504	0.0%	2	0
ZN	[18.0, 0.0, 12.5, 75.0, 21.0, 90.0, 85.0, 100....	float64	26	0.0%	480	0
INDUS	[2.31, 7.07, 2.18, 7.87, 8.14, 5.96, 2.95, 6.9...	float64	76	0.0%	430	0
CHAS	[0.0, 1.0]	float64	2	0.0%	504	0
NOX	[0.538, 0.469, 0.458, 0.524, 0.499, 0.428, 0.4...	float64	81	0.0%	425	0
RM	[6.575, 6.421, 7.185, 6.998, 7.147, 6.43, 6.01...	float64	446	0.0%	60	0
AGE	[65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96....	float64	356	0.0%	150	0
DIS	[4.09, 4.9671, 6.0622, 5.5605, 5.9505, 6.0821,...	float64	412	0.0%	94	0
RAD	[1.0, 2.0, 3.0, 5.0, 4.0, 8.0, 6.0, 7.0, 24.0]	float64	9	0.0%	497	0
TAX	[296.0, 242.0, 222.0, 311.0, 307.0, 279.0, 252...	float64	66	0.0%	440	0
PTRATIO	[15.3, 17.8, 18.7, 15.2, 21.0, 19.2, 18.3, 17....	float64	46	0.0%	460	0
B	[396.9, 392.83, 394.63, 394.12, 395.6, 386.63,...	float64	357	0.0%	149	0
LSTAT	[4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19...	float64	455	0.0%	51	0

■ Scikit –learn dataset loading utilities

강의 교재에서 다룰 실습 데이터세트는 사이킷런에서 제공하는 dataset 유틸리티, fetch_**, load_**, make_** 3가지를 사용한다. 호출된 라이브러리의 객체 datasets에 '.'을 찍고 실행하면 필요한 데이터세트를 로딩할 수 있다(https://scikit-learn.org/stable/datasets.html).

```
from sklearn import datasets
datasets
```

```
<module 'sklearn.datasets' from 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages₩₩sklearn₩₩datasets₩₩__init__.py'>
```

```
datasets.
```

```
clear_data_home
dump_svmlight_file
fetch_20newsgroups
fetch_20newsgroups_vectorized
fetch_california_housing
fetch_covtype
fetch_kddcup99
fetch_lfw_pairs
fetch_lfw_people
fetch_olivetti_faces
```

1. Toy datasets

전체 데이터세트를 불러와 특정하든, 특정된 데이터세트를 불러오든 상관이 없다. Datasets 유틸리티의 데이터는 사전형 자료형태로 구성되어 있다.

Boston 주택가격 데이터세트 loading

```
from sklearn import datasets
boston= datasets.load_boston()
boston.keys()

from sklearn.datasets import load_boston
boston= load_boston()
boston.keys()
      dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
>>>   dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

데이터세트의 값은 numpy array로 구성되어 있으며 자료에 대한 설명을 확인한다.

Boston 주택가격 데이터세트 loading

```
boston.data[0:1]
boston.target[0:10]
print(boston.DESCR)
```

```
array([[6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
        6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
        4.980e+00]])

array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])

.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
>>>     - MEDV     Median value of owner-occupied homes in $1000's
```

파이썬 pandas의 데이터프레임으로 데이터세트를 구성한다.

Boston dataset to pandas

```
df= pd.DataFrame(boston.data, columns=boston.feature_names)
df['target']= boston.target
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

2. Real world datasets

여러 신문사로부터 20개 주제별로 수집된 뉴스 기사 데이터세트이다..

20newsgroups datasets

```
from sklearn.datasets import fetch_20newsgroups
newsgroups= fetch_20newsgroups()
newsgroups.keys()

df= pd.DataFrame(newsgroups.data, columns=['contents'])
df['target']= newsgroups.target
df.head()
```

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

	contents	target
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14

```
>>>
```

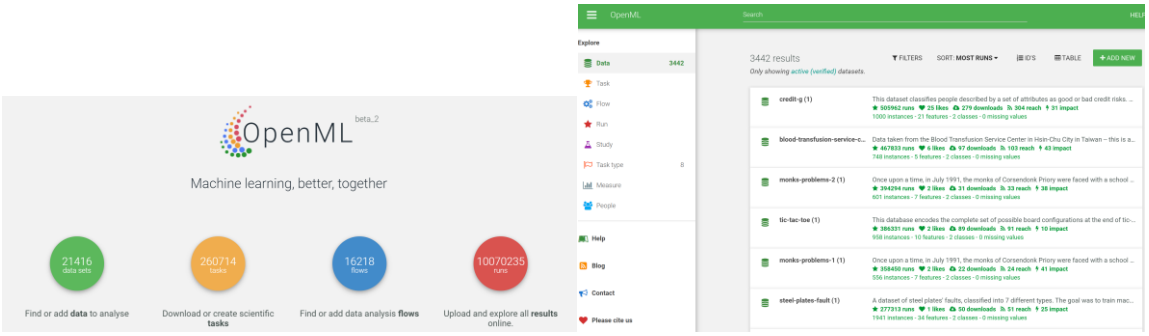
특정한 시점을 기준으로 훈련과 검증데이터로 구분된다.

20newsgroups train/test datasets

```
newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_train.keys()
newsgroups_train.target.shape
newsgroups_test = fetch_20newsgroups(subset='test')
newsgroups_test.target.shape
```

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

(11314,)

>>> (7532,)
```

외부 공개데이터 사이트 'https://www.openml.org/home'에서 datasets 의 이름으로 파이썬의 데이터로 불러올 수 있다.



UCI 데이터세트인 와인의 품질관련 데이터세트를 불러와 확인한다..

Open ML datasets

```
from sklearn.datasets import fetch_openml
data = fetch_openml(name='wine-quality-white')
data.url
data.keys()
df = pd.DataFrame(np.c_[data.data, data.target], columns= data.feature_names + ['target'])
df.head()
```

```
'https://www.openml.org/d/40498'
```

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

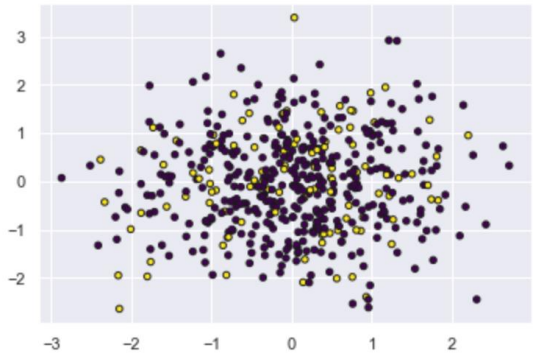
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	target
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.001	3.0	0.45	8.8	4
1	6.3	0.3	0.34	1.6	0.049	14.0	132.0	0.994	3.3	0.49	9.5	4
2	8.1	0.28	0.4	6.9	0.05	30.0	97.0	0.9951	3.26	0.44	10.1	4
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	4
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	4

3. generated datasets

분류, blobs, moon 형태의 가공 데이터 세트를 생성하여 확인한다.

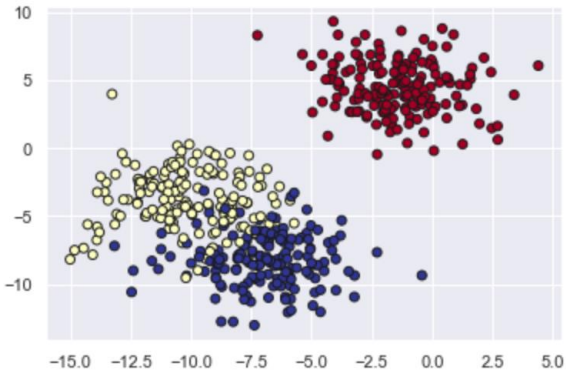
20newsgroups train/test datasets

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=500, n_classes=2, weights=[0.8, 0.2], random_state=1)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap='viridis', edgecolor='k')
```



20newsgroups train/test datasets

```
from sklearn.datasets import make_moons, make_circles, make_blobs
X, y = make_blobs(n_samples=500, centers=3,
                  n_features=2, cluster_std=2, random_state=1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='RdYlBu', edgecolor='k')
```



■ Scikit-learn getting started

사이킷런의 포도주 품질등급 데이터를 다운받아 확인하면 178개의 인스턴스(자료 크기)와 13개의 특성변수(색깔, 알콜농도키, 함유 화학물질 등)를 확인한다.

입력데이터 확인

```
from sklearn import datasets
X, y = datasets.load_wine(return_X_y=True, as_frame=True)
df = X.merge(y, left_index=True, right_index=True)
df.head()
df.shape
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od28
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

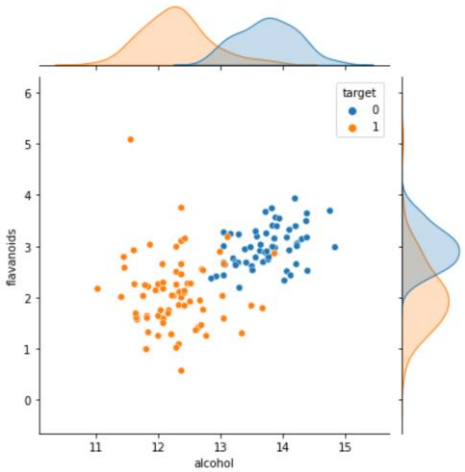
(178, 14)

>>> (178, 14)

품질등급 '0'과 '1'인 이진분류 경우만 분석하기 위해 품질등급 '2'의 자료는 제거하여 시각화하면 알코올농도와 플라바노이드의 값에 따라 와인 품질 등급이 차이가 나는 것을 확인한다.

알코올농도와 화학물질에 따른 품질등급 산포도

```
sns.jointplot(data=df.loc[df.target < 2, :], x='alcohol',y='flavanoids', hue='target')
```



와인 품질의 구분(classification)을 위해 2개의 특성의 평균 데이터로 자료를 준비한다.

알콜농도와 플라보노이드 특성변수 데이터 준비(Arrange data)

```
data = df.loc[df.target < 2, :]
```

```
data_mean = data.groupby('target')[['alcohol','flavanoids']].mean().round(2).reset_index()
data_mean
X = data_mean.drop(['target'], axis=1)
y = data_mean['target']
```

	target	alcohol	flavanoids
0	0	13.74	2.98
1	1	12.28	2.08

사이킷런의 랜덤포레스트 알고리즘을 호출하여 데이터에(데이터의 크기는 2개)에 훈련시켜 알코올 '14', 화학물질'1'인 경우는 '와인 품질등급 0'로 예측한다. 필자와 독자의 예측 결과를 동일하게 재현하려면 동일한 'random_state'을 지정하면 된다.

사이킷런의 랜덤포레스트 분류기를 호출하여 객체화

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
clf.fit(X,y)
clf.predict([[14, 1]])
```

```
>>> array([0])
```

와인 품질 등급별 평균값 2 개로 학습한 예측모형으로 전체 178 자료를 예측하여 품질등급을 실제와 비교해 보면 정확도(accuracy)가 0.81(81.51.7%)로 확인된다.

평균 데이터로 학습하여 예측(Prediction)

```
X = data[['alcohol','flavanoids']]
y = data['target']
y_pred = clf.predict(X)
(y_pred == y).mean()
from sklearn.metrics import accuracy_score #  사이킷런의 정확도 산출 모듈
accuracy_score(y_pred, y)
```

```
        0.9230769230769231
>>>     0.9230769230769231
```

평균데이터가 아닌 전체 데이터로 학습하여 예측한 결과의 정확도는 0.992로 남녀 평균 2개의 자료로 학습한 결과보다 훨씬 높아진 것을 확인 할 수 있다.

전체 데이터로 학습하여 예측(Prediction)

```
clf = RandomForestClassifier(random_state=0)
clf.fit(X,y)
y_pred = clf.fit(X,y).predict(X)
accuracy_score(y_pred, y)
```

```
>>> 1.0
```

다양한 알고리즘으로 학습하기 위해 관련 모듈을 불러온다.

다양한 알고리즘 API 호출

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from mlxtend.plotting import plot_decision_regions # 2차원 특성변수의 분류경계선을 시각
clfs = [RandomForestClassifier(), LogisticRegression(), SVC(), GaussianNB(), KNeighborsClassifier()]
```

모형의 정확도와 의사결정 경계선은 알고리즘에 따라 달라지며 모델은 성능 지표를 기준으로 비교 평가하여야 한다.

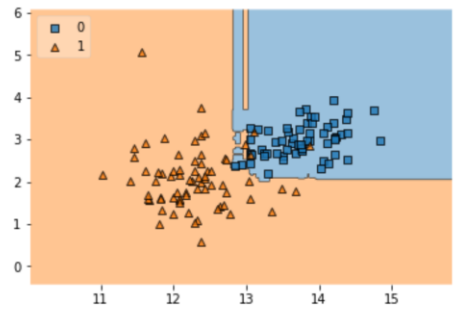
학습 및 예측의 분류경계선 시각화

```
for i, clf in enumerate(clfs) :
    y_pred = clf.fit(X,y).predict(X)
    print(f'{clf}알고리즘의 예측 모형정확도 {accuracy_score(y_pred, y).round(2)}')#정확도(Accuracy)
    plot_decision_regions(X.values, y.values, clf=clf, legend=2); plt.show()
```

```
RandomForestClassifier()알고리즘의 예측 모형정확도 0.9920324949226683
LogisticRegression()알고리즘의 예측 모형정확도 0.8864239962505859
SVC()알고리즘의 예측 모형정확도 0.8826745820965474
GaussianNB()알고리즘의 예측 모형정확도 0.884393063583815
>>> KNeighborsClassifier()알고리즘의 예측 모형정확도 0.9056397437900328
```

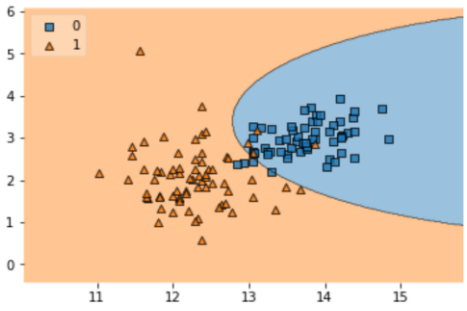
RandomForestClassifier()알고리즘의 예측 모형정확도 1.0

<AxesSubplot:>



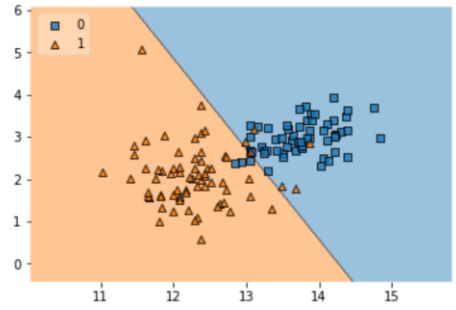
GaussianNB()알고리즘의 예측 모형정확도 0.94

<AxesSubplot:>



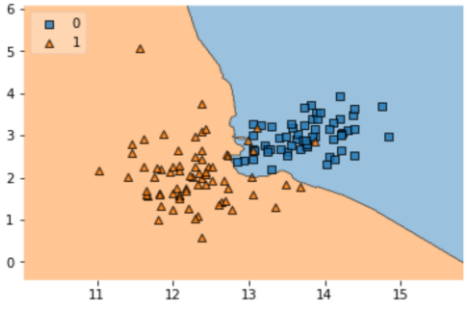
LogisticRegression()알고리즘의 예측 모형정확도 0.93

<AxesSubplot:>



KNeighborsClassifier()알고리즘의 예측 모형정확도 0.96

<AxesSubplot:>



훈련과 검증데이터가 동일하여 과적합이 발생하는 것을 방지하기 위해 훈련과 검증데이터를 각각 짝수와 홀수 인덱스를 기준으로 나눈 후에 예측하여 정확도를 확인하면 '0.95'이다.

훈련과검증데이터 분리후 교차검증

```
X_train = X[1::2]; y_train = y[1::2]
X_test = X[::2]; y_test = y[::2]
clf = LogisticRegression()
y_pred = clf.fit(X_train, y_train).predict(X_test)
accuracy_score(y_pred, y_test)
 >>> 0.9538461538461539
```

데이터를 훈련과 검증데이터로 구분하는 모듈을 사이킷런에서 불러와 훈련과 검증데이터를 각각 20%, 80%로 무작위 구분하여 예측을 하면 정확도는 '0.872'이다..

train_test_split API

```
from sklearn.model_selection import train_test_split #  훈련과 검증데이터 분류
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
y_pred = clf.fit(X_train, y_train).predict(X_test)
accuracy_score(y_pred, y_test)
  >>> 0.9615384615384616
```

특성변수의 평균, 분산을 각각 '0'과 '1'로 표준화하여 예측 하면 성능이 향상되기도 한다.

StandardScaler와 정확도

```
from sklearn.preprocessing import StandardScaler
X_train_std = StandardScaler().fit(X_train).transform(X_train)
X_test_std = StandardScaler().fit(X_test).transform(X_test)
y_pred= clf.fit(X_train_std, y_train).predict(X_test_std)
accuracy_score(y_pred, y_test)
 >>> 0.9230769230769231
```

Pipeline을 사용하여 데이터 변환과 fitting을 한꺼번에 수행한 정확도의 평균을 확인한다. 2번의 교차검증에도 y의 범주 유형이 똑같은 비중을 가질 수 있도록 'stratify= y'로 설정한다.

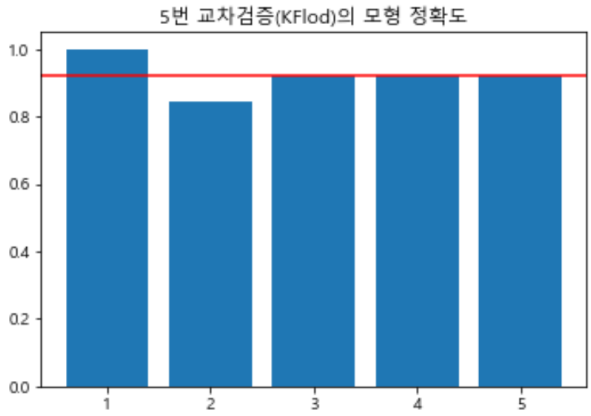
pipeline

```
from sklearn.pipeline import make_pipeline
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)
pipe = make_pipeline(
    StandardScaler(),
    LogisticRegression()
)
y_pred = pipe.fit(X_train, y_train).predict(X_test)
accuracy_score(y_pred, y_test)
 >>> 0.9393939393939394
```

KFold함수를 사용하여 교차검증을 5번 수행한 모형의 정확도와 평균을 확인한다.

K-fold cross validation (교차검증)

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=True, random_state=11)
accuracy_list = []
for train_index, test_index in kf.split(X):
    X_train, y_train = X.iloc[train_index, :], y[train_index]
    X_test, y_test = X.iloc[test_index], y[test_index]
    y_predict = pipe.fit(X_train, y_train).predict(X_test)
    accuracy_list.append(accuracy_score(y_test, y_predict))
plt.rcParams['font.family'] = 'Malgun Gothic'  # 한글 표시
plt.rcParams['axes.unicode_minus'] = False  #한글 폰트 사용시 마이너스 폰트 깨짐 해결
plt.bar(np.arange(1, 6), accuracy_list)
plt.axhline(y=np.array(accuracy_list).mean(), color='r', linestyle='-')
plt.title('5번 교차검증(KFlod)의 모형 정확도')
```



5겹 Kfold 평균 정확도는 cross_validate 함수를 사용하여 쉽게 확인할 수 있다.

cross_validate 기능

```
from sklearn.model_selection import cross_validate
np.mean(accuracy_list)
result = cross_validate(pipe, X, y, cv=kf)  # defaults to 5-fold CV
result['test_score'].mean()
```

```
0.9230769230769231

>>> 0.9230769230769231
```

Pipeline을 확인하면 자료표준화(standardscaler), 랜덤포레스트(randomforest)로 순차적으로 데이터전처리와 알고리즘으로 구성되어 있다. 랜덤포레스트의 파라미터는 clf.get_params()로 확인할 수 있다. 데이터의 특성에 적합한 최적의 파라마팅 튜닝은 다음에 자세히 살펴보도록 한다.

Default parameters

```
pipe
```

```
clf
clf.get_params()
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

```
LogisticRegression()
```

```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

2개의 랜덤포레스트 파라미터, max_depth와 n_estimators에 대한 하이퍼파라미터 튜닝을 RandomizedSearchCV(cross validation) 방법으로 최적화한다. 5차 교차검증으로 20회(n_iters=20)반복하여 총 100번의 fiiting을 수행한다.

Hyperparamerization with cross validation

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
param_distributions = {'randomforestclassifier__max_depth': np.random.choice(100, 50),
                       'randomforestclassifier__n_estimators' : np.random.choice(100, 50)
                      }
rs = RandomizedSearchCV(estimator=pipe,
                        n_iter=20,
                        param_distributions=param_distributions,
                        random_state=0, verbose=1)
rs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:   16.6s finished
```

```
RandomizedSearchCV(estimator=Pipeline(steps=[('standardscaler',
                                              StandardScaler()),
                                             ('randomforestclassifier',
                                              RandomForestClassifier(random_state=0))]),
                   n_iter=20,
                   param_distributions={'randomforestclassifier__max_depth': array([38, 50, 69, 49,  9, 82, 58, 55, 87, 51,  6,  5,
5,
       60, 57, 87, 12, 22, 16, 49,  9, 45, 37, 83, 73, 30, 90, 46, 48, 70,
       18, 66, 69,  4, 44, 52, 22, 88, 61, 35, 75, 66, 78, 88, 79, 82]),
                                        'randomforestclassifier__n_estimators': array([77,  4, 69, 91, 85, 19, 95, 82, 34, 52, 45,
96, 90,
       69, 96, 46, 67, 10, 76, 41, 97, 18, 96, 43, 39, 51, 11, 41, 29, 88,
        0,  6, 79, 95, 94,  8, 72,  8, 13, 23, 16,  4, 75,  8, 44, 42])},
                   random_state=0, verbose=1)
```

최적화된 모형의 훈련과 검증데이터 정확도는 각각 '0.963'과 '0.965'이며 최적 모델의 파라미터 max_depth와 n_estimators는 각각 12개와 88개 이다.

The best estimator

```
print('optimal train score: {:.3f}'.format(rs.best_score_))
print('test score: {:.3f}'.format(rs.score(X_test, y_test)))
print('optimal parameter: {}'.format(rs.best_params_))
```

```
>>>
optimal train score: 0.933
test score: 0.962
optimal parameter: {'randomforestclassifier__n_estimators': 77, 'randomforestclassifier__max_depth': 35}
```

하이퍼파라미터를 무작위그리트서치한 총 20번 검증데이터의 평균 정확도(KFold=5)가 높은 5개의 파라미터를 확인한다.

The best estimator의 훈련과 검증과정 진행 결과

```
results= pd.DataFrame(rs.cv_results_)[['params','mean_test_score', 'rank_test_score']]
results.sort_values('rank_test_score').round(4).head(10)
```

	params	mean_test_score	rank_test_score
9	{'randomforestclassifier__n_estimators': 38, 'randomforestclassifier__max_depth': 41}	0.9329	1
1	{'randomforestclassifier__n_estimators': 77, 'randomforestclassifier__max_depth': 35}	0.9329	1
17	{'randomforestclassifier__n_estimators': 68, 'randomforestclassifier__max_depth': 27}	0.9329	1
16	{'randomforestclassifier__n_estimators': 85, 'randomforestclassifier__max_depth': 28}	0.9329	1
4	{'randomforestclassifier__n_estimators': 56, 'randomforestclassifier__max_depth': 84}	0.9329	1
5	{'randomforestclassifier__n_estimators': 61, 'randomforestclassifier__max_depth': 51}	0.9329	1
15	{'randomforestclassifier__n_estimators': 35, 'randomforestclassifier__max_depth': 50}	0.9329	1
14	{'randomforestclassifier__n_estimators': 41, 'randomforestclassifier__max_depth': 23}	0.9329	1
13	{'randomforestclassifier__n_estimators': 77, 'randomforestclassifier__max_depth': 8}	0.9329	1
10	{'randomforestclassifier__n_estimators': 77, 'randomforestclassifier__max_depth': 23}	0.9233	10

연구 목적에 맞는 모델간 성능을 비교하는 성능지표는 정확도(Accuracy)이외 Recall, Precision, f1-score 등 다양하게 사용 가능하다. 하이퍼파라미터 작업은 오랜 시간이 걸릴 수 있기 때문에 최적화된 모델은 저장 할 수 있다..

model persistency

```
from joblib import dump, load
dump(rs.best_estimator_, '../working/randoforest_randomized.joblib')
```

■ 모형 평가 지표_Threshold Metrics

지도학습의 분류문제에서 예측과 실제가 얼마나 차이가 있는지 의사결정임계값에 따라 변동하는 정확도(accuracy), 검측율(Recall), 정밀도(Precision), 검측율과 정밀도의 조화평균(f1-score)을 살펴본다. 연구 목적에 따라 최적의 의사결정임계값을 선정할 수 있다.

569명 환자(인스턴스)가 특성변수(31개)에 따라 흉부암 발생시 생존여부(target)를 나타내는 이진분류(binary classification) 사이킷런에서 데이터를 확보한다

흉부암 데이터 확보

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
cancer = pd.DataFrame(np.c_[cancer['target'], cancer['data']],
                      columns = np.append('target', cancer['feature_names']))
cancer['target'] = cancer['target'].map({0.0: 1, 1.0:0}).astype('int64')
cancer.shape
cancer.head()
cancer.target.value_counts(normalize=True)
#cancer.target.value_counts().plot(kind='bar', rot='360')
```

```
(569, 31)
```

	target	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	...	worst radius
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54

5 rows × 31 columns

```
0    0.627417
1    0.372583
Name: target, dtype: float64
```

SVC(support vector classification)알고리즘으로 예측 확률을 구하기 위해 'probability=True' 파라미터를 설정하였다.

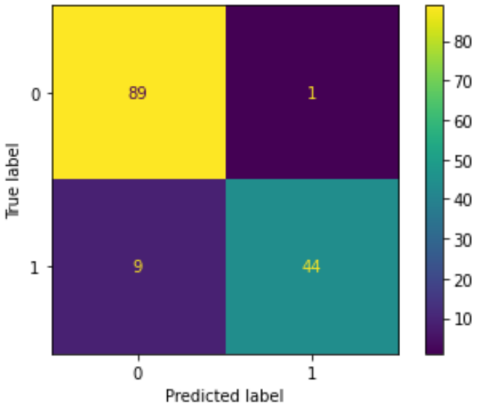
훈련과 검증데이터 분리 및 SVC알고리즘으로 학습

```
X = cancer.drop('target', axis = 1)
y = cancer['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)
clf = SVC(random_state=0, probability=True)
clf.fit(X_train, y_train)
```

예측 모형은 '0.5'이상인 예측 양성확률(의사결정임계값, thresholds)을 모두 양성으로 판정하며 혼동표도 의사결정임계값 '0.5'를 기준으로 양성과 음성의 예측 결과를 보여준다.

양성 예측 확률 '0.5' 기준 혼동표(Confusion matrix)

```
from sklearn.metrics import plot_confusion_matrix, confusion_matrix
plot_confusion_matrix(clf, X_test, y_test);plt.show()
y_pred = clf.predict(X_test)
confusion_matrix(y_test, y_pred)
```




```
array([[89,  1],
       [ 9, 44]], dtype=int64)
```

혼동표를 그리드 없이 보려면 sns.set()이후 'sns.set_style("whitegrid", {'axes.grid' : False})' 명령어를 입력한다.

양성과 음성의 정밀도, 검측율, f1-score는 의사결정임계값 '0.5'를 기준으로 보여준다.

양성 예측 확률 '0.5' 기준 분류 보고서(Classification report)

```
from sklearn.metrics import classification_report
target_names = ['생존(0)', '사망(1)']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

      생존(0)       0.91      0.99      0.95        90
      사망(1)       0.98      0.83      0.90        53

    accuracy                           0.93       143
   macro avg       0.94      0.91      0.92       143
weighted avg       0.93      0.93      0.93       143
```

Threshold ‘0.5’를 기준으로 검측율, 정밀도, f1-score를 계산하여 확인한다.

다양한 알고리즘으로 예측 모형 비교

```
cm = confusion_matrix(y_test, y_pred)
print(f'정확도(accuracy): {(cm * np.eye(2)).sum() / cm.sum():.3f}')
print(f'정밀도(precision): {((cm * np.eye(2)).sum(axis=0) / cm.sum(axis=0)).round(3)}')
precision = (cm * np.eye(2)).sum(axis=0) / cm.sum(axis=0)
print(f'검측율(recall): {((cm * np.eye(2)).sum(axis=1) / cm.sum(axis=1)).round(3)}')
recall = (cm * np.eye(2)).sum(axis=1) / cm.sum(axis=1)
print(f'f1-score(Harmonic mean): {((2 / (1 / precision + 1 / recall))).round(3)}')
```

```
정확도(accuracy): 0.930
정밀도(precision): [0.908 0.978]
검측율(recall): [0.989 0.83 ]
f1-score(Harmonic mean): [0.947 0.898]

array([[89,  1],
       [ 9, 44]], dtype=int64)
```

Threshold를 ‘0’에서 ‘1’을 기준으로 양성 예측확률(y_probas[:,1]를 확인한다. Threshold가 ‘0’과 ‘1’의 극단적인 경우 양성판정 여부를 예로 나타낸다. ‘0.5’보다 낮은 Threshold기준 양성으로 판정하면 양성판정이 많아져서 검측율은 상승하나 정밀도는 하락한다.

Threshold에 따른 양성판정 결과

```
y_probas = clf.predict_proba(X_test)
n = len(y_probas)
threshhold = np.linspace(0,1,n)
y_pred = y_probas[:,1].reshape(-1,1) > threshhold.reshape(1,-1) # numpy broadcasting
y_pred.shape
y_pred[:, 0]
y_pred[:, 142]
```

```
>>> (143, 143)
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True])
```

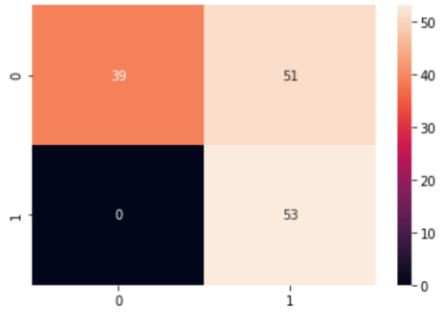
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False])
```

Threshold에 따른 혼동표와 분석보고서를 확인한다. 낮은 threshold는 양성판정에 대해 낮은 기준을 설정으로 양성판정이 많아지게 되지만 엄격한 threshhold는 그 반대이다.

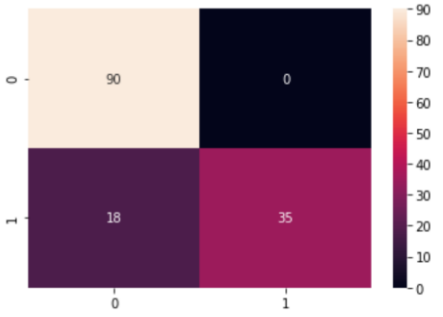
Threshold에 따른 혼동표와 분석보고서

```
def threshhold_matrix(y_test, y_pred):
    target_names = ['생존(0)', '사망(1)']
    cm = confusion_matrix(y_test, y_pred)
    print(classification_report(y_test, y_pred, target_names=target_names))
    sns.heatmap(cm, annot = True, fmt="d"); plt.show()
    print('*'*100)

threshhold_matrix(y_test, y_pred[:,4])
threshhold_matrix(y_test, y_pred[:,140])
```

```
              precision    recall  f1-score   support

      생존(0)       1.00      0.43      0.60        90
      사망(1)       0.51      1.00      0.68        53

    accuracy                           0.64       143
   macro avg       0.75      0.72      0.64       143
weighted avg       0.82      0.64      0.63       143
```




```
              precision    recall  f1-score   support

      생존(0)       0.83      1.00      0.91        90
      사망(1)       1.00      0.66      0.80        53

    accuracy                           0.87       143
   macro avg       0.92      0.83      0.85       143
weighted avg       0.90      0.87      0.87       143
```



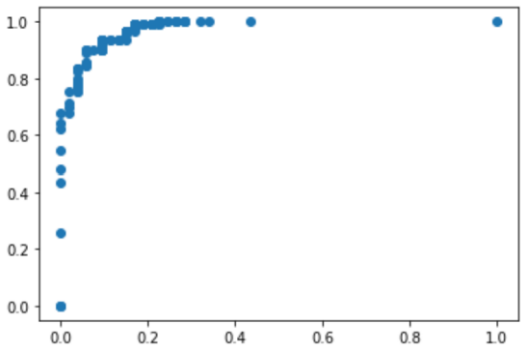
■ 모형 평가 지표_Ranking Metrics

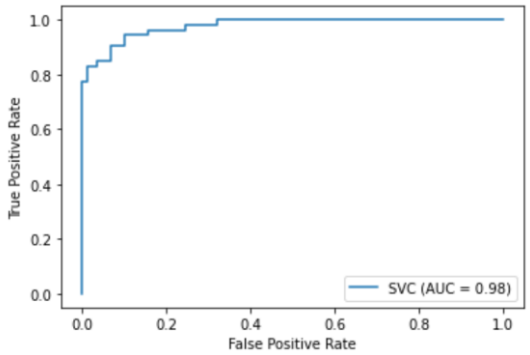
다양한 지도학습의 분류모형에 대한 평가와 선택은 가변적인 threshold를 고려하여 결정하여야 한다. 예측 모형의 성능을 객관적인 지표로 비교 평가하여 우선순위를 결정할 수 있는 수신자운영특성곡선(Receiver Operating Characteristic Curve, ROC)과 정밀도·검측율 그래프(Prececion-Recall Graph, PR)를 이해한다.

특정 예측 모형의 허위양성율(false positive rate, FPR)과 참양성율(true positive rate, TPR)을 가변적인 threshold에 따라 2차원의 그래프로 나타내면 ROC가 만들어 진다.

ROC 곡선

```
tpr = []; fpr = []
for i in range(n):
    cm = confusion_matrix(y_test, y_pred[:,i])
    tpr.append(cm[0,0]/cm.sum(axis=1)[0])
    fpr.append(cm[1,0]/cm.sum(axis=1)[1])
plt.scatter(fpr, tpr); plt.show()
from sklearn.metrics import plot_roc_curve
plot_roc_curve(clf, X_test, y_test)
```





Threshold가 '0'이면 모든 데이터를 '1'로 판단하기 때문에 TPR과 FPR은 모두 '1'이다. Threshol를 '1'에서 '0'으로 감소시키면 서 해당하는 TPR과 FPR을 그리면 우상향의 ROC가 된다. AUROC(Area Under ROC)는 ROC의 아래 면적으로 이 값이 클수록 우수한 모델이다. 면적의 크기는 음성을 양성으로 골라내는 비중(FPR)보다 양성을 양성으로 판정하는 비중이(TPR) 얼마나 큰가를 나타낸다.

AUROC

```
from sklearn.metrics import roc_auc_score
y_score = clf.predict_proba(X_test)[:, 1]
roc_auc_score(y_test, y_score)
 >>> 0.9775681341719078
```

분류임계값(threshold)에 따른 정확도의 변화

```
((clf.predict_proba(X_test)[:,1] >= 0.3) == y_test).mean()
((clf.predict_proba(X_test)[:,1] >= 0.5) == y_test).mean()
((clf.predict_proba(X_test)[:,1] >= 0.8) == y_test).mean()
```

```
0.9090909090909091

0.9230769230769231

>>> 0.916083916083916
```

Precision과 Recall을 2차원 좌표에 시각화하면 PR 그래프를 확인할 수 있으며 곡선의 아래 면적(average_precision_score)를 구할 수 있다.

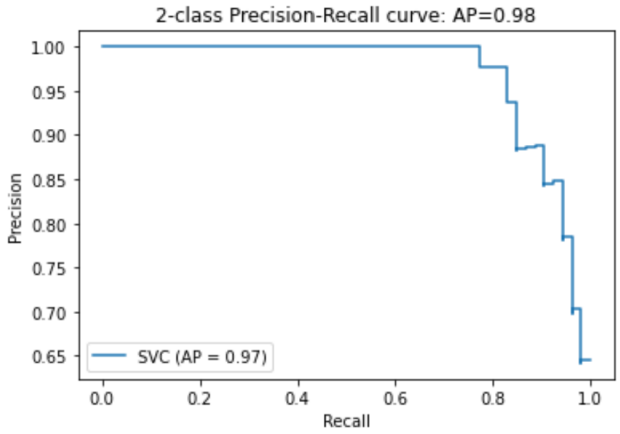
AP

```
from sklearn.metrics import average_precision_score
average_precision = average_precision_score(y_test, y_score)
print(f'Average precision-recall score: {average_precision:0.2f}')
```

```
>>> Average precision-recall score: 0.97
```

PR 곡선

```
from sklearn.metrics import plot_precision_recall_curve
disp = plot_precision_recall_curve(clf, X_test, y_test)
disp.ax_.set_title(f'2-class Precision-Recall curve: AP={average_precision:0.2f}')
```

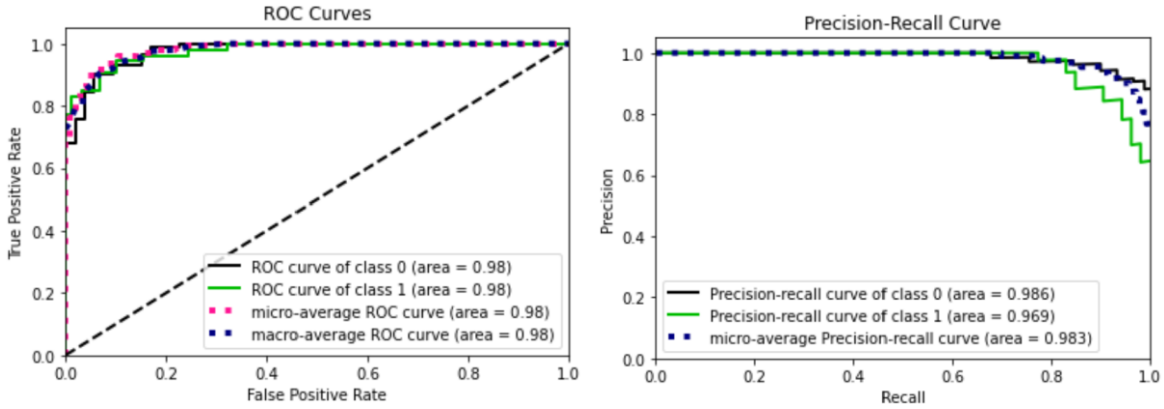


양성의 비중이 10% 미만인 불균형 데이터의 경우는 검측율(Recall)의 성능지표가 중요하기 때문에 모형의 정확도(accuracy), ROC보다는 PR그래프가 성능 지표로 더 적합하다.

사이킷런의 scikit plot 모듈로 ROC와 PR 그래프를 확인한다. 내부적으로 5겹 교차검증을 디폴트로 양성과 음성을 기준으로 'micro'와 'macro'의 성능지표를 각각 계산하여 보여주고 있다.

Scikit plot 모듈로 AUC와 PR그래프 시각화

```
import scikitplot as skplt
skplt.metrics.plot_roc(y_test, y_probas)
skplt.metrics.plot_precision_recall(y_test, y_probas)
```



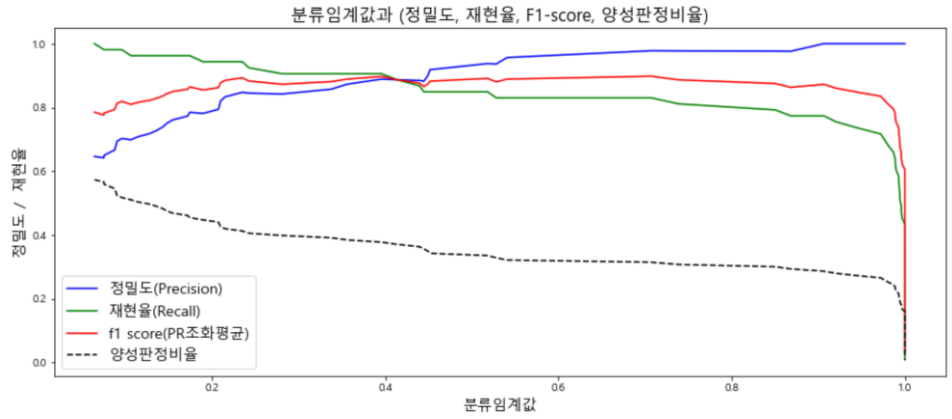
검측율, 정밀도, f1-score를 threshold에 따라 나타내기 위한 사용자정의함수(to_lables)를 만들고 시각화한다. 전체 검증데이터개수에서 양성판정을 내린 개수는 'scores'로 나타내어 모형의 양성 판정을 받은 비율을 확인하여 단속활동이나 불량 검사 작업에 실제 적용이 가능하다.

분류임계값에 따른 recall, precesion,양성판정 개수 구하기

```
from sklearn.metrics import precision_recall_curve
def to_labels(pos_probs, threshold):
    return (pos_probs >= threshold).astype('int')
p, r, thresholds = precision_recall_curve(y_test, y_probas[:, 1])
scores = [np.sum(to_labels(y_probas[:, 1], t))/len(y_probas[:, 1]) for t in thresholds]
```

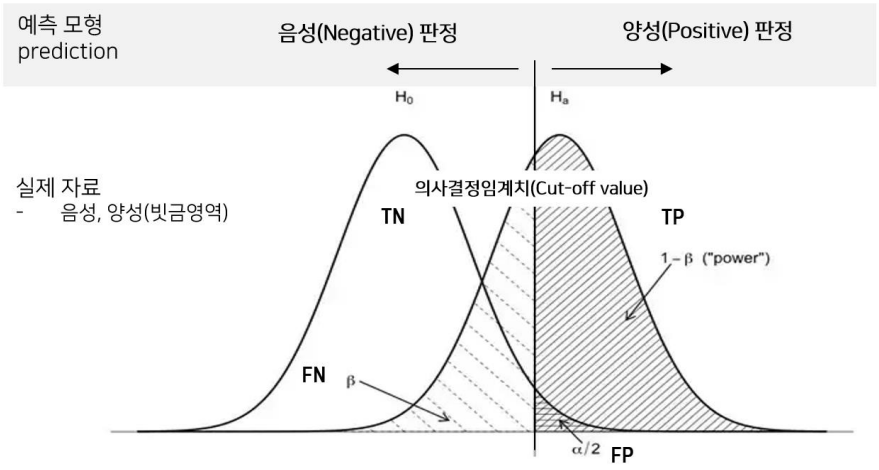
분류임계값과 성능지표

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.rcParams["figure.figsize"] = (15,6)
    plt.rcParams['font.family']='Malgun Gothic' #  한글폰트
    plt.title('분류임계값과  (정밀도,  재현율, F1-score,  양성판정비율)', fontsize='16')
    plt.plot(thresholds, precisions[:-1], 'b-', label='정밀도(Precision)') #  판사
    plt.plot(thresholds, recalls[:-1], 'g-', label='재현율(Recall)') #  검사
    plt.plot(thresholds, (2*precisions[:-1]*recalls[:-1])/(precisions[:-1] +recalls[:-1]),
             'r-', label='f1 score(PR조화평균)') #  변호사
    plt.plot(thresholds, scores, 'k--', label='양성판정비율') # q-ratio
    plt.ylabel('정밀도 /   재현율', fontsize='14')
    plt.xlabel('분류임계값', fontsize='14')
    plt.legend(loc='best', fontsize='14')
plot_precision_recall_vs_threshold(p, r, thresholds)
```



분류임계값(threshold)가 증가하면 검측율(재현율)은 감소하고 정밀도는 증가하며 최대 f1-score는 '0.898'이며 이때 threshold는 '0.707'이다.

음성(귀무가설, H_0)과 양성(대립가설, H_1)의 실제 자료를 머신러닝의 예측 모형이 의사결정임계치(threshold)에 따라 양성과 음성으로 각각 판정할 경우 허위음성(FN, False Negative)과 허위양성(False Positive)이 발생하는 것과 같다. 카드 사기, 부도업체 적발, 불량 제품 예측 등에는 허위음성(False Negative)를 줄이는 것이 중요하고 양성판정을 엄격하게 하기 위해 Cut-off 값을 증가시키면 참양성 개수(TP)는 감소한다.



Precision, f1-score 등 다양하게 사용 가능하다. 각 성능지표의 의미는 다음 절에서

F1-csore가 최대인 threshold

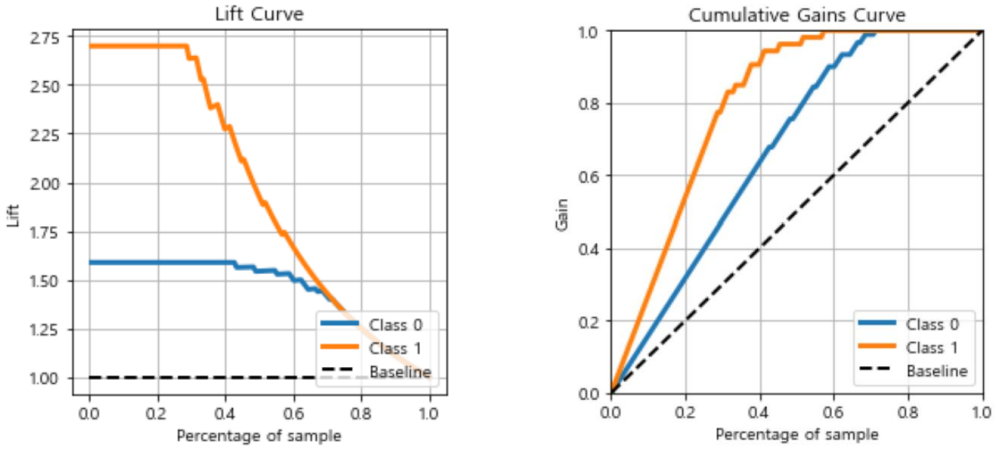
```
from sklearn.metrics import f1_score
scores = [f1_score(y_test, to_labels(y_probas[:, 1], t)) for t in thresholds]
ix = np.argmax(scores)
print(f'Threshold(분류임계값)={thresholds[ix]:.3f}, at Maximum F-measure={scores[ix]:.3f}')
```

```
>>> Threshold(분류임계값)=0.707, at Maximum F-measure=0.898
```

Lift curve와 Cumulative gain curve도 확인할 수 있다.

Lift curve, Cumulative gain curve

```
skplt.metrics.plot_lift_curve(y_test, y_probas); plt.show()
skplt.metrics.plot_cumulative_gain(y_test, y_probas)
```



마케팅분야에서 잠재고객중에서 충성도(양성 판정 확률)가 높은 고객을 우선 대상으로 인센티브를 제공하는 캠페인을 실행할 때 Lift곡선과 누적이익곡선을 기준으로 자원배분을 하면 된다,

■ 모형 평가 지표_Probalistic Metrics

실제로 새로운 양성 데이터를 임의의 두 개의 모형이 각각 '0.96'과 '0.54'의 확률로 양성 판정을 한다면 '0.96'의 확률로 판정한 모형이 더 우수하다.

Brier-score 통계량은 실제 값(y_test)에서 예측 확률(y_probas)을 차감하여 제곱한 총합으로 예측의 확률이 얼마나 정확한지를 보여주는 수치이다. 예를 들어 실제 값인 [0, 1]에 대해 'A'와 'B'가 각각 [0.1, 0.9]와 [0.5, 0.5]의 확률로 예측하였다고 가정한다. 확신을 가진 'A'에 비해 'B'는 반신반의하고 있다. 이런 경우 A와 B의 brie-score는 각각 '0', '0.25'로 'A'가 'B' 보다 예측을 잘한다. 예측 모형의 Brier score를 확인하면 0.25이하로 '0'에 근접하여 모형의 신뢰성이 높다.

Brier score

```
from sklearn.metrics import brier_score_loss
brier_score_loss(y_test, y_score) #brier_score_loss(y_test, y_probas[:, 1])
```

```
>>> 0.05393556118435772
```

상관계수와 유사한 지표인 MCC(Matthews correlation coefficient)는 실제와 예측과의 상관관계를[16] 나타낸다. '-1'부터 '1'까지의 값을 가지며 '1'은 100% 정확한 예측, '0'은 무작위 예측, '-1'은 100% 부정확한 예측을 의미한다.

Matthews correlation coefficient

```
from sklearn.metrics import matthews_corrcoef
y_pred = clf.predict(X_test)
matthews_corrcoef(y_test, y_pred)
```

```
>>> 0.8518535291230603
```

MCC와 유사한 지표인 Cohen kappa score를 확인한다.

Cohen_kappa_score

```
from sklearn.metrics import cohen_kappa_score
cohen_kappa_score(y_test, y_pred)
```

```
>>> 0.8453050627434011
```

보정 곡선은 모델의 예측값을 기준으로 [0~10%], (10~20%], … (90~100%]의 구간으로 예측확률과 실제 비율의 관계를 나타낸다. 45도선에 예측 확률이 근접하면 신뢰성이 높다.

Calibration curve

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomForestClassifier, GradientBoostingClassifier)
from sklearn.preprocessing import StandardScaler

classifiers = {}
```

```
classifiers.update({'GNB': GaussianNB()})
classifiers.update({'LGR': LogisticRegression(class_weight='balanced')})
classifiers.update({'SVC': SVC(probability=True)})
classifiers.update({'RF': RandomForestClassifier(n_estimators=100)})
from tqdm import tqdm_notebook
classifier_labels = []
probas_list = []
for classifier_label, classifier in tqdm_notebook(classifiers.items()):
    y_proba = classifier.fit(X_train, y_train).predict_proba(X_test)
    probas_list.append(y_proba)
    classifier_labels.append(classifier_label)

skplt.metrics.plot_calibration_curve(y_test, probas_list, classifier_labels, figsize=(15,8))
leg = plt.legend(title='Model', fontsize = 14)
leg._legend_box.align = 'left'
```