3. Scikit-learn

Scikit-learn 개요

Sciket-learn 개요

- 파이썬으로 구현된 가장 유명한 기계학습 오픈소스 라이브러리이다.
- 기계학습의 여러 가지 알고 리즘 및 데이터 처리 기법을 쉽고 빠르게 적용해보고 최상의 결과를 얻을 수 있다.
- 특징
  - 파사드 디자인 패턴을 적용하여 라이브러리 인터페이스 통일
  - 다양한 기계학습 알고리즘, 모델 선택 및 데이터 전처리 기능 탑재
  - Numpy를 기반으로 개발되어 속도 최적화
  - 다른 라이브러리와의 호환성이 좋음
  - GPU는 지원하지 않음
- scikit-learn은 고수준의 API로 직관적이고 사용하기 쉬운 인터페이스가 특징이다.
- estimator, fit, predict, transform 4가지 개념만 익히면 금방 적응이 가능하다.

Scikit-learn 데이터셋

Scikit-learn 데이터셋

- Scikit-learn의 서브패키지인 sklearn.datasets는 실습을 위한 샘플용 데이터셋을 제공한다.
- 샘플용 데이터셋의 접근 방법
  - 기본적으로 Scikit-learn 패키지 안에 내장되어 있는 형태(load명령으로 import)
  - 인터넷에서 다운로드하여 사용하는 형태(fetch명령으로 import),
  - 새로운 데이터셋을 생성시켜 사용하는 형태(make 명령으로 생성)
- load 계열
  - load_boston() : 보스턴 집값 데이터
  - load_diabetes() : 당뇨병 관련 데이터
  - load_iris() : iris 데이터
- fetch 계열
  - fetch_covtype() : 토지조사 데이터
  - fetch_20newsgroups() : 뉴스텍스트 데이터
  - fetch_rcv1() : 로이터뉴스 말뭉치
  - fetch_california_housing : 주택 데이터

Scikit-learn 데이터셋

- make 계열
  - make_regression() : regression용 데이터 생성
  - make_classification() : classification용 데이터 생성
  - make_blobs() : clustering용 데이터 생성

sklearn 데이터셋 객체

- data : 독립 변수의 ndarray 배열 형태
- target : 종속 변수의 ndarray 배열 형태
- feature_names : 독립 변수 이름의 리스트 형태
- target_names : 종속 변수 이름의 리스트 형태
- DESCR : 데이터에 대한 설명
- filename : 데이터가 저장된 로컬 주소

estimator, fit, predict, transform

estimator

- 학습데이터에 기반해 모델을 적합시키고 새로운 데이터의 어떤 특성을 추론할 수 있는 객체 를 Estimator라고 지칭한다.
- Estimator는 fit 메서드를 가지고 있으며 scikit-learn이 제공하는 모든 기계학습 알고리즘은 Estimator이며 클래스로 구현되어 있다.
- 모델의 성능을 검증하는 cross_val_score 함수나 하이퍼 파라 미터 튜닝을 지원하는 GridSearchCV 같은 경우 이 estimator를 인자로 받아 내부에서 fit을 실행한다.
- Ex)

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data = load_breast_cancer() # 연습용 데이터 셋 로드
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, random_state=42)

# Estimator 인스터화 및 하이퍼 파라미터 설정
model = DecisionTreeClassifier(criterion='entropy')
model
```

estimator, fit, predict, transform

fit

- 인스턴스화한 estimator에서 fit 메서드를 이용해서 학습시킨다.
- 지도학습 알고리즘은 학습데이터와 라벨데이터를 함께 인자로 전달하고, 비지도학습 알고리즘은 학습데이터만 인자로 전달한다.

predict

- 입력데이터 에 대한 모델의 예측 결과를 반환한다.
- Ex)

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred
```

estimator, fit, predict, transform

transform

- 피처를 처리하는 기능은 predict 대신 에 transfrom 메서드로 실행되고 결과를 반환한다.
- Ex)

```
from sklearn.preprocessing import StandardScaler
# 전처리하기 전 출력
print(X_train)

# 전처리 - 스케일링 적용
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
# 전처리후 결과 확인
X_train
```

estimator, fit, predict, transform

fit_transfrom

- fit과 transform을 하나로 결합한 fit_transform을 제공한다.
- Ex)

```
# 전처리하기 전 출력
print(X_test)

# 전처리 - 스케일링 적용
scaler = StandardScaler()
scaler.fit(X_train)
X_test = scaler.transform(X_test)
# 전처리후 결과 확인
X_test
```

Scikit-learn 주요 모듈

Scikit-learn 주요 모듈

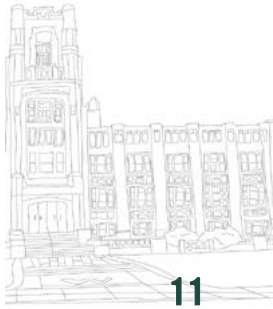
분류	모듈명	내장 기능
예제 데이터	sklearn.datasets	연습용 데이터셋
피처 처리	sklearn.preprocessing	전처리 관련 기법 (원핫 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	모델에 중요한 영향을 미치는 피처를 탐색 및 선택하는 기법
	sklearn.feature_extraction	원시 데이터로부터 피처를 추출하는 기능 이미지에 대한 피처 추출은 하위 모듈 image에, 텍스트 데이터의 피처 추출은 하위 모듈 text에 지원 API가 있음.
차원 축소	sklearn.decomposition	차원 축소 관련 알고리즘 계열 (PCA, NMF, Truncated SVD 등)

Scikit-learn 주요 모듈

검증, 하이퍼 파라미터 튜닝, 데이터 분리	sklearn.model_selection	검증, 하이퍼 파라미터 튜닝, 데이터 분리 등 (cross_validate, GridSearchCV, train_test _split, learning_curve 등)
모델 평가	sklearn.metrics	모델의 성능을 측정 및 평가하는 기법 (accuray, precision, recall, ROC curve 등)
기계학습 알고리즘	sklearn.ensemble	앙상블 알고리즘 계열 (랜덤 포레스트, 에이다 부스트, 배깅 등)
	sklearn.linear_model	선형 알고리즘 계열 (선형회귀, 로지스틱 회귀, SGD 등)
	sklearn.naive_bayes	나이브 베이즈 알고리즘 계열 (베르누이 NB, 가우시안 NB, 다항분포 NB 등)
	sklearn.neighbors	최근접 이웃 알고리즘 계열 (K-NN 등)

Scikit-learn 주요 모듈

	sklearn.svm	Support Vector Machine 계열 알고리즘
	sklearn.tree	의사 결정 나무 계열 알고리즘
	sklearn.cluster	비지도 학습(클러스터링) 알고리즘 (KMeans, , DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 기계학습 알고리즘 등을 연쇄적으로 실행하는 기능



Scikit-learn: Iris 데이터셋으로 Machine Learning

Iris 데이터셋으로 기계학습 맛보기

- iris 데이터 불러오기

```
from sklearn.datasets import load_iris
data = load_iris()
data.keys()
```

•data.keys()의 내용은 다음과 같다.

- data : 독립 변수 ndarray 배열
- target : 종속 변수 ndarray 배열
- target_names : 종속 변수 이름 리스트
- DESCR 데이터에 : 대한 설명
- feature_names : 독립 변수 이름 리스트
- filename : 데이터가 저장된 로컬 주소

Scikit-learn: Iris 데이터셋으로 Machine Learning

- **DESCR을 통해 아이리스 데이터에 대해 알아보자**

  print(data.DESCR)

  •통계적 정보를 제외하고 중요한 도메인 지식을 요약하면 다음과 같다.
  - **데이터 명 : IRIS (아이리스, 붓꽃 데이터)**
  - **데이터 수 : 150개**
  - **변수 개수 : 5개**
  - **변수의 이해**

Sepal Length	꽃받침의 길이 정보
Sepal Width	꽃받침의 너비 정보
Petal Length	꽃잎의 길이 정보
Petal Width	꽃잎의 너비 정보
Species	꽃의 종류 정보. setosa / versicolor / virginica 의 3종류로 구분

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 전처리
  - 실습에 필요한 라이브러리를 임포트한다.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

  - 현재 변수 데이터는 numpy의 ndarray이다. 분석의 용이성을 위해 pandas의 DataFrame으로 변환한다.

```
iris = pd.DataFrame(data=data.data, columns=data.feature_names)
iris
```

  - 타겟 데이터도 DataFrame 구조로 변환한다.

```
target = pd.Series(data.target, dtype="category")
target
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 타겟값을 숫자에서 문자열로 수정한다.

```
target =target.cat.rename_categories(data.target_names)
target

iris["species"] = target
iris
```

- 열이름 수정

```
iris.rename({"sepal length (cm)":"sepal_length", "sepal width (cm)": "sepal_width", "petal length (cm)":"petal_length", "petal width (cm)":"petal_width"}, axis=1, inplace=True)
iris
```

- 결측값 확인
  - isna로 결측값을 True로 변환한 후 sum 하면 결측값의 개수를 알 수 있다.
  - 결과를 확인해 보면 피쳐(feature)의 결측값 개수는 0개이고, 타겟(target)도 결측값 개수가 0임을 알 수 있다.

```
iris.isna().sum(axis=0)
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 결측값 전처리
  - 아이리스는 다행히 결측값이 없으므로 결측값 전처리를 생략한다.
  - 결측값이 있었더라면, 나머지 데이터의 평균, 중앙값 을 사용하는 방법 등이 있다.
  - 결측값의 수가 너무 많을 경우, 데이터가 대표성을 갖지 못하므로 삭제할 수도 있다.

- info 메서드를 이용하여 DataFrame에 대한 정보를 확인한다.

```
iris.info()
```

- describe 메서드를 이용하여 기초 통계량을 확인한다.

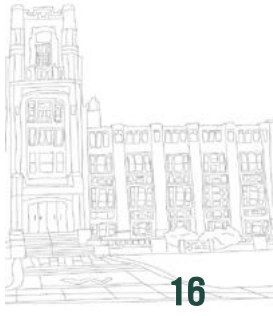
```
iris.describe()
```

- corr 메서드를 이용하여 피처끼리의 상관관계를 분석한다.

```
iris.corr()
```

- groupby 메서드를 이용하여 타겟별 기술 통계 분석한다.

```
iris.groupby("species").size()
```



Scikit-learn: Iris 데이터셋으로 Machine Learning

- 데이터 시각화
  - boxplot을 그려 기초 통계량을 시각화하고 이상치를 탐지한다.

```
def boxplot_iris(feature_names, dataset):
    i = 1
    plt.figure(figsize=(11, 9))
    for col in feature_names:
        plt.subplot(2,2,i)
        plt.axis('on')
        plt.tick_params(axis="both", left=True, top=False, right=False,
                        bottom=True, labelleft=False, labeltop=False,
                        labelright=False, labelbottom=False)
        dataset[col].plot(kind="box", subplots=True, sharex=False, sharey=False)
        plt.title(col)
        i += 1
    plt.show()

boxplot_iris(iris.columns[:-1], iris)
```

- sepal_length와 sepal_width의 박스 길이가 짧은 것으로 보아 이 둘의 데이터 중심화 경향이 높다고 할 수 있다. sepal_width에서는 이상치가 탐지되었다. petal_length와 petal_width은 박스의 길이가 긴 것으로 보아 데이터 가 넓게 분포되어 있음을 알 수 있다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- pairplot을 이용하여 피처 간의 상관관계 및 데이터 분포를 시각화한다.

```
import seaborn as sns
sns.pairplot(iris, hue="species")
plt.show()
```

  - 실행 결과를 보면 setosa는 다른 클래스들과 확연하게 떨어져서 군집을 이루고 있다. 가상의 선을 그어서 분류할 수 있으므로 setosa는 선형모델로 잘 분류될 것이다. versicolor와 virginica는 sepal_width와 sepal_length 피처로 그린 그래프에서는 서로 뒤섞여서 선을 그어서 분류하기 어려워 보인다. 그러나 다른 그래프를 보면 약간 애매한 부분이 있지만, 그래도 분류가 가능할 것으로 보인다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- piechart를 그려 타겟의 클래스 비율을 시각화한다. 아래 실행 결과를 보며뉴타겟 클래스별로 데이터가 균등하게 구성되어있음을 한 눈에 알 수 있다.

```
def piechart_iris(feature_names, tartget, dataset):
    i = 1
    plt.figure(figsize=(11, 9))
    for colName in [target]:
        labels = []
        sizes = []
        df = dataset.groupby(colName).size()
        for key in df.keys():
            labels.append(key)
            sizes.append(df[key])
        plt.subplot(2,2,i)
        plt.axis('on')
        plt.tick_params(axis="both", left=False, top=False, right=False, bottom=True,
            labelleft=True, labeltop=True, labelright=False, labelbottom=False)
        plt.pie(sizes, labels=labels, autopct="%1.1f%%", shadow=True, startangle=140)
        plt.axis('equal')
        i += 1
    plt.show()
piechart_iris(iris.columns[:-1], iris.species, iris)
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 학습
  - sklearn의 train_test_split 함수로 훈련용 데이터셋과 성능평가용 데이터셋을 나눈다.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.iloc[:, :-1], iris.iloc[:, -1],
test_size=0.33, random_state=42)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

  - 데이터 분석 결과를 토대로 적합한 알고리즘을 선택한다.

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=None,
                min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                max_features=None, random_state=42, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                class_weight=None)
model.fit(X_train, y_train) # 학습한다.
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 성능평가용 데이터셋을 이용해 성능을 평가한다. scikit-learn에서 score는 accuray(정확도)를 의미한다.

```
model.score(X_test, y_test)
```

모델 일반화 전략

- 기계학습은 데이터로부터 모델의 성능이 결정되는 특징(data-driven)이 있으므로, 데이터가 충분히 많아야 성능을 발휘할 수 있다. 데이터가 부족한 경우에는 과적합(overfitting) 문제가 발생할 수 있다. 모델이 학습 데이터의 특징에만 적합하여 새로운 데이터(unseen data)에 대한 예측 능력이 떨어지게 되는 것이다. 테스트 데이터는 학습에서 모델에 간접적으로라도 영향을 미치면 안 되기 때문에 테스트 데이터로 검증을 해서는 안된다. 그래서 검증(validation) 데이터셋을 따로 두어 매 훈련마다 검증 데이터셋에 대해 평가하여 모델을 튜닝해야 한다.K-겹 교차 검증(K-fold cross-validation)을 사용하여 검증한다.
- K-겹 교차 검증(K-fold cross-validation)

```
from sklearn.model_selection import cross_val_score, KFold
cv = KFold(n_splits=10, shuffle=True, random_state=42)
results = cross_val_score(model, X_train, y_train, cv=cv)
fin_result = np.mean(results)
for i, _ in enumerate(results):
    print("{}번째 교차검증 정확도: {}".format(i, _))
print("교차검증 최종 정확도: {}".format(fin_result))
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- **Learning Curve**
  - 데이터의 양이 충분히 많을 경우는 트레인셋과 밸리데이션셋을 무작위로 나누더라도 동일한 데이터 분포를 유 지한다. 교차검증 기법은 데이터가 충분하지 않을 때 필요한 기법이다. 데이터의 양이 충분한지 판단하기 위해 서 학습곡선(Learning Curve)을 그린다. 학습 곡선은 x축을 학습 데이터의 개수, y축을 성능 점수로 하여 학습 데이터의 양을 조금씩 늘릴 때마다 성능이 어떻게 변화하는지를 보여주는 곡선이다. 성능 점수는 내부적으로 교 차 검증하여 산출한다.
  - pip install scikit-plot # 추가 설치

```
import scikitplot as skplt
model = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=None,
                min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                max_features=None, random_state=42, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None)
skplt.estimators.plot_learning_curve(model, X_train, y_train, figsize=(6,6))
plt.show()
```

  - 실행결과를 보면 초록 선은 교차 검증 결과이다. 초록선이 우상향하는데 어느 순간 떨어지기 시작하면, 과적합되는 시점이다. 빨간선은 학습에 사용했던 데이터로 검증한 결과이다. 빨간선은 데이터가 많아지면 일시적으로 떨어질 수도 있다. 이는 일시적으로 발생할 수 있는 현상이며 장기적으로 그래프는 수렴한다.
  - cv 옵션을 지정하지 않았으므로 기본값인 3fold가 적용되었다. 트레인셋에는 100개의 데이터가 들어있고 교차검증으로 33%를 사용하였으므로 x축의 최대값은 66이다. 학습 곡선을 보면 초록선이 올라가는 모양에서 끊겼으므로, 데이터가 더 많이 있을 때 어떻게 될지 알 수 없다. 따라서 이 학습곡선으로 알수 있는 것은, 현재 의사결정나무 모델은 데이터가 더 있다면 성능이 더 좋아질 가능성이 있다는 것뿐이다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 모델 최적화 전략을 위한 GridSearchCV를 이용한 하이퍼 파라미터 탐색
  - 하이퍼 파라미터는 기계가 스스로 찾을 수 없어서 사람이 직접 지정해야 하는 파라미터를 일컫는다.
  - 하이퍼 파라미터를 찾는 방법도 분석가가 경험적으로 얻은 노하우로 찾는 것이 일반적이다.
  - scikit-learn은 하이퍼 파라미터를 찾는 GridSearchCV 기능을 제공한다. 하이퍼 파라미터 조합에 대한 경우의 수를 모두 격자(grid)에 나열하고 모든 조합을 일일히 학습 및 성능 측정하는 기능이다.

```
from sklearn.model_selection import GridSearchCV
estimator = DecisionTreeClassifier()
cv = KFold(n_splits=10, random_state=42, shuffle=True)
parameters = {'max_depth' : [4,5,6,10],
              'criterion' : ['gini', 'entropy'],
              'splitter' : ['best', 'random'],
              'min_weight_fraction_leaf' : [0.0, 0.1, 0.2, 0.3],
              'random_state' : [7, 23, 42, 78],
              'min_impurity_decrease': [0.0, 0.05, 0.1, 0.2]}
model = GridSearchCV(estimator = estimator,
                     param_grid = parameters,
                     cv = cv, verbose = -1,
                     n_jobs = -1, refit = True)
model.fit(X_train, y_train)
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- GridSearCV로 찾은 최적의 파라미터와 최적화된 성능은 best_params_와 best_score_ 속성에 기록된다. refit 옵션을 True로 했을 경우, 찾아낸 최적의 하이퍼 파라미터로 모델을 학습시켜 best_estimator_ 속성에 기록한다.

```
print("Best Estimator:₩n", model.best_estimator_);print()
print("Best Params:₩n", model.best_params_);print()
print("Best Score:₩n", model.best_score_);print()
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 평가 지표 및 모델 평가
  - Accuracy 의 함정
    - 정확도(accuracy)만으로는 모델을 올바르게 평가할 수 없는 한계가 있다. 만약 테스트셋에 48개의 setosa와 1개의 versicolor 와 1개의 virginica가 있다고 해보자. 이 테스트셋을 이용해 평가할 경우 정확도가 96%가 되는 문제가 있다. 하지만 이 모델이 성능이 좋아서 라고는 결코 말할 수 없을 것이다.

```
from sklearn.metrics import accuracy_score
pred = model.predict(X_test)
acc = accuracy_score(y_test, pred)
print("Accuracy : ", acc)
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- **혼동 행렬(Confusion Matrix)**
  - 예측 결과 맞은 것과 틀린 것을 모두 분석하기 위해 오분류표(confusion matrix)를 이용한다.
  - confusion matrix는 예측한 것 중에서 얼마나 잘 맞았는지, 또는 실제 타겟에 대해서 얼마나 잘 맞았는지 다각도로 성능을 검증할 수 있다.
  - 이진분류 : 네 가지 개념(TP, FP, TN, FN)를 기반으로 precision, recall, f1-score 등의 평가 점수를 만들 수 있다.

	양성 예측	음성 예측
실제 양성	TP (True Positive)	FN (False Negative)
실제 음성	FP (False Positive)	TN (True Negative)

  - 다중분류 : 아이리스 데이터는 다중 분류(multi label classification) 과제이기 때문에 위처럼 4가지 개념으로 표현할 수가 없다. setosa, versicolor, virginica를 각각 이진분류 과제로 생각하고 3개씩 지표를 만들 것이다. setosa를 예로 들면 다음과 같다.

	setosa 예측	versicolor 예측	virginica 예측
실제 setosa	실제 setosa 이고 setosa로 예측한 경우	실제 setosa 인데 versicolor로 예측한 경우	실제 setosa 인데 virginica로 예측한 경우
실제 versicolor	실제 verisicolor 인데 setosa로 예측한 경우	실제 verisicolor 이고 versicolor로 예측한 경우	실제 verisicolor 인데 virginica로 예측한 경우
실제 virginica	실제 virginica 인데 setosa로 예측한 경우	실제 virginica 인데 versicolor로 예측한 경우	실제 virginica 인데 virginica로 예측한 경우

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 아이리스 데이터셋의 Confusion Matrix

```
from sklearn.metrics import confusion_matrix
pred = model.predict(X_test)
confMatrix = confusion_matrix(y_test, pred)
print("Confusion Matrix : ₩n : ", confMatrix)
```

- scikit-plot을 이용하면 confusion matrix를 좀 더 직관적인 heatmap으로 시각화한다. scikit-learn에서는 가 로축과 세로축의 라벨이 없었던 반면에 scikit-plot은 축 라벨이 있어서 결과를 해석하기가 좀 더 용이하다.

```
import scikitplot as skplt
pred = model.predict(X_test)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- **accuracy / precision / recall / fall-out / f-score**
  - **confusion matrix의 TP, TN, FP, FN을 기반으로 평가 결과를 점수화할 수 있다. 이 4가지 개념은 이진 분류 과제에서만 가능하다.**
  - **아이리스 데이터와 같이 타겟 클래스가 N개인 다중 분류 과제는 각각의 타겟 클래스를 이진분류로 생각하고 N개의 confusion matrix를 구한다.**
  - **아이리스 데이터를 예로 들면 setosa, versicolor, virginica를 각각 이진분류 과제로 생각하고 3개의 confusion matrix를 만든다. setosa의 confusion matrix를 그린다면 다음과 같다.**

	setosa가 맞다고 예측	setosa가 아니라고 예측 (virsicolor 또는 virginica로 예측)
실제 setosa일 때	TP (True Positive)	FN (False Negative)
실제 setosa 아닐 때 (virsicolor 또는 virginica)	FP (False Positive)	TN (True Negative)

Scikit-learn: Iris 데이터셋으로 Machine Learning

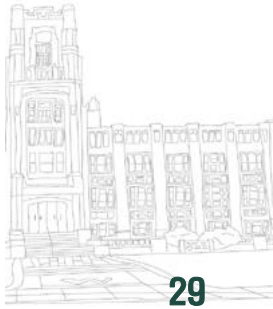
- 정확도(accuracy)
  - 전체 샘플 중 맞게 예측한 샘플 수의 비율을 뜻한다. 높을수록 좋은 모형이다.
  - 일반적으로 학습에서 최적화 목적함수로 사용된다.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- 정밀도(precision)
  - 정밀도(precision)는 예측한 클래스 중에 실제로 맞은 비율이다.

$$\text{precision} = \frac{TP}{TP + FP}$$

```
from sklearn.metrics import precision_score
# average 기본값은 binary이며 다중 분류일때는 binary가 아니어야 함
precisions = precision_score(y_test, model.predict(X_test), average=None)
for target, score in zip(data.target_names, precisions):
    print(f"{target}의 정밀도: {score}")
```



Scikit-learn: Iris 데이터셋으로 Machine Learning

- 재현율(recall)
  - 재현율(recall)은 실제 타겟 클래스 중에 예측이 맞은 비율이다. 민감도(sensitivity) 라고 부르기도 한다.

$$\text{recall} = \frac{TP}{TP + FN}$$

```
from sklearn.metrics import recall_score
# average 기본값은 binary이며 다중 분류일때는 binary가 아니어야 함
recalls = recall_score(y_test, model.predict(X_test), average=None)
for target, score in zip(data.target_names, recalls):
    print(f"{target}의 정밀도: {score}")
```

- fall-out
  - 위양성률(fall-out)은 타겟이 아닌 실제 클래스 중에서 틀린 비율이다. 1-특이도(specificity)로 표현하기도 한다.

$$\text{fallout} = \frac{FP}{FP + TN}$$

- fall-out 점수를 계산하는 기능은 scikit-learn에서 제공하지 않는다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- f-score
  - 정밀도와 재현율의 가중조화평균(weight harmonic average)을 F점수(F-score)라고 한다. 정밀도에 주어지는 가중치를 베타(beta)라고 한다.

$$F_\beta = (1 + \beta^2)\,(\text{precision} \times \text{recall}) \,/\, (\beta^2\,\text{precision} + \text{recall})$$

  - precision과 recall의 가중치를 균등하게 부여하기 위해서 를 1로 설정하는 경우가 많으며 이를 특별히 f1-score라고 지칭한다.

$$F_1 = 2 \cdot \text{precision} \cdot \text{recall} \,/\, (\text{precision} + \text{recall})$$

```
from sklearn.metrics import fbeta_score, f1_score
# average 기본값은 binary이며 다중 분류일때는 binary가 아니어야 함
fbetas = fbeta_score(y_test, model.predict(X_test), average=None, beta=2)
for target, score in zip(data.target_names, fbetas):
    print(f"{target}의 정밀도: {score}")

f1s = f1_score(y_test, model.predict(X_test), average=None)
for target, score in zip(data.target_names, f1s):
    print(f"{target}의 정밀도: {score}")
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 혼동 행렬 예제

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
```

– 실행 결과

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

–첫 행은 실제로 0인 두개의 데이터가 둘 다 정확하게 0으로 예측되었다는 뜻이다. 두번째 행은 실제로 1인 하나의 데이터가 2로 분류되었다는 뜻이다. 마지막 행은 실제로 2인 데이터 3개 중 2개만 정확하게 3으로 분류되었고 하나는 0으로 분류되었다는 뜻이다.

–열 기준으로 보면 첫번째 열은 분류 모형이 0이라고 예측한 3개의 데이터 중 2개만 원래 0이었고 하나는 원래 2였다는 뜻이다. 마지막 열은 분류 모형이 2라고 예측한 3개의 데이터 중 2개만 2였고 하나는 1이었다는 의미이다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- classification_report
  - 각각의 클래스를 양성(positive) 클래스로 보았을 때의 정밀도, 재현율, F1점수를 각각 구하고 그 평균값으로 전체 모형의 성능을 평가한다.
  - support : 실제 타겟 클래스의 개수
  - macro: 단순평균
  - weighted: 각 클래스에 속하는 표본의 갯수로 가중평균
  - accuracy: 정확도. 전체 학습데이터의 개수에서 각 클래스에서 자신의 클래스를 정확하게 맞춘 개수의 비율.
- Ex1)

```
# 간단 테스트 예제
from sklearn.metrics import classification_report
y_true = [0, 0, 0, 1, 1, 0, 0]
y_pred = [0, 0, 0, 0, 1, 1, 1]
print(classification_report(y_true, y_pred, target_names=['class 0', 'class 1']))
```

- Ex2)

```
from sklearn.metrics import classification_report
print(classification_report(y_test, model.predict(X_test), target_names=['setosa', 'versicolor',
    'virginica']))
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- **ROC curve**
  - ROC(Receiver Operating Characteristic) Curve는 클래스 판별 기준값의 변화에 따른 재현율과 위양성율의 변화를 시각화한 것이다.
  - ROC curve는 TPR(True Positive Rate)를 y축, FPR(False Positive Rate)를 x축으로 하는 그래프이다. TPR은 recall, FPR은 fall-out과 같다.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

  - 클래스 판별 기준값이 낮을수록 Positive로 예측하는 수가 증가한다. 예를 들어 setosa일 확률이 0.6일 때, 기준값을 0.55라고 하면 positive(양성)으로 예측하지만 기준값을 0.65로 높이면 negative(음성)으로 예측이 바뀐다. TPR의 분모(TP+FN)는 실제 positive인 표본 개수이므로 항상 고정이며 FPR의 분모(FP+TN)도 실제 negative인 표본 개수이므로 항상 고정이다. positive 예측 = TP + FP 이므로 분모가 고정인 상 태에서 positive 예측이 증가하면 일반적으로 TP와 FP도 동시에 증가한다. 재현율과 위양성율은 양의 상관관계 가 있다.

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 분류문제를 풀고 decision_function 메서드를 이용하여 모든 표본 데이터에 대해 판별함수값을 계산한 다음 계산된 판별함수값이 가장 큰 데이터부터 가장 작은 데이터 순서로 정렬한 것이다.

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np

X, y = make_classification(n_samples=16, n_features=2, n_informative=2, n_redundant=0,
                           random_state=0)

model = LogisticRegression().fit(X, y)
y_hat = model.predict(X)
f_value = model.decision_function(X)

df = pd.DataFrame(np.vstack([f_value, y_hat, y]).T, columns=["f", "y_hat", "y"])
df.sort_values("f", ascending=False).reset_index(drop=True)
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

- 기준값 0을 사용하여 이진 분류결과표, 재현율, 위양성율을 계산하면 다음과 같다.

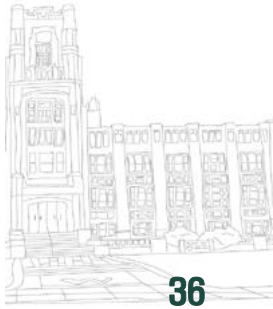
```
# 이진 분류 결과 표
from sklearn.metrics import confusion_matrix
confusion_matrix(y, y_hat, labels=[1, 0])
```

–실행 결과는 다음과 같다.

```
    array([[7, 1],
           [1, 7]], dtype=int64)
```

```
# 재현율, 위양성율을 계산

recall = 7 / (7 + 1)  # TPR
fallout = 1 / (1 + 7) # FPR
print("recall =", recall)
print("fallout =", fallout)
```



Scikit-learn: Iris 데이터셋으로 Machine Learning

- 사이킷런 패키지에서 만들어진 모형은 기본적으로 정확도(accuracy)를 최대화하는 모형이다. 하지만 정확도, 정밀도, 재현도 등의 성능이 동일한 모형도 ROC 커브에서 살펴보면 성능이 달라지는 것을 볼 수 있다.

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

X, y = make_classification(n_samples=1000, weights=[0.95, 0.05], random_state=5)

model1 = LogisticRegression().fit(X, y)
y_hat1 = model1.predict(X)

model2 = SVC(gamma=0.0001, C=3000, probability=True).fit(X, y)
y_hat2 = model2.predict(X)

print(confusion_matrix(y, y_hat1))
print(confusion_matrix(y, y_hat2))

from sklearn.metrics import classification_report
print(classification_report(y, model1.predict(X)))
print(classification_report(y, model2.predict(X)))
```

Scikit-learn: Iris 데이터셋으로 Machine Learning

```
fpr1, tpr1, thresholds1 = roc_curve(y, model1.decision_function(X))
fpr2, tpr2, thresholds1 = roc_curve(y, model2.decision_function(X))

import matplotlib.pyplot as plt
plt.rcParams["font.family"] = 'Malgun Gothic'
plt.plot(fpr1, tpr1, 'o-', ms=2, label="Logistic Regression")
plt.plot(fpr2, tpr2, 'o-', ms=2, label="Kernel SVM")
plt.legend()
plt.plot([0, 1], [0, 1], 'k--', label="random guess")
plt.xlabel('위양성률(Fall-Out)')
plt.ylabel('재현률(Recall)')
plt.title('ROC 커브')
plt.show()
```