



Week 2

파이썬 머신러닝 완벽가이드 2장, 3장 필사 & 개념정리(2.6장, 3.6장 제외)

Chapter 02: 사이킷런으로 시작하는 머신러닝

01 사이킷런 (scikit-learn) 소개와 특징

- 사이킷런은 가장 많이 사용되는 파이썬 머신러닝 라이브러리



특징:

- 가장 파이썬스러운 API를 제공
- 다양한 알고리즘과 개발을 위한 편리한 프레임워크와 API 제공
- 검증됐고, 많은 환경에서 사용됨

02 첫 번째 머신러닝 만들어 보기 - 붓꽃 품종 예측하기

- 머신러닝 모델: 붓꽃의 품종을 분류(Classification)하기
 - 분류(Classification)는 대표적인 지도학습 (Supervised Learning) 방법의 하나
 - 지도학습이란? 명확한 정답이 주어진 데이터를 먼저 학습한 뒤 미지의 정답을 예측하는 방식
- 꽃잎의 길이, 너비, 꽃받침의 길이, 너비 피쳐 (Feature) 기반으로 꽃의 품종 예측하기
- 사이킷런 패키지 내 모듈명 알아보기!

sklearn.datasets	사이킷런에서 자체적으로 제공하는 데이터 세트 생성
sklearn.tree	트리 기반 ML 알고리즘 구현
sklearn.model_selection	학습 데이터와 검증 데이터, 예측 데이터로 데이터 분리하거나 최적의 하이퍼 파라미터로 평가하기 위해 사용

▼ 하이퍼 파라미터란?

머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터들을 통칭
하이퍼 파라미터를 통해 머신러닝 알고리즘 성능 튜닝 가능

```
from sklearn.datasets import load_iris #붓꽃 데이터 세트 생성
from sklearn.tree import DecisionTreeClassifier #ML 알고리즘: 의사결정 트리
from sklearn.model_selection import train_test_split #학습 데이터와 테스트 데이터 분리
```

머신러닝 모델 만들 때 따를 순서

1. 데이터 세트 로딩

- 데이터 세트에서 피쳐(feature)만으로 된 데이터 로딩
- 데이터 세트에서 레이블(결정 값) 데이터 불러오기
- 데이터 세트를 DataFrame으로 변환

2. 학습용 데이터와 테스트 데이터 분리

- train_test_split() API 이용
- 예제에서는 피쳐 데이터 세트와 레이블 데이터 세트 각각 학습용과 테스트용으로 나눴다
- 예시:

```
test_size = 0.2 #전체 데이터 중 테스트 데이터 20%, 학습 데이터 80%
```

3. 의사결정 트리 이용해 학습과 예측 수행

- DecisionTreeClassifier 객체로 생성
- 학습 수행과 예측 수행 코드:

```
변수명_fit(X_train, y_train) #학습 수행
변수명_predict(X_test)
```

예측은 반드시 학습 데이터가 아닌 다른 데이터 이용해야 한다 (보통 테스트 데이터 세트 이용)

4. 예측 성능 평가

- 여기서는 정확도를 측정함 (예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지)
- accuracy_score() 이용

```
from sklearn.metrics import accuracy_score
accuracy_score(실제 레이블 데이터 세트, 예측 레이블 데이터 세트)
```

사이킷런의 기반 프레임워크 익히기

Estimator 이해 및 fit(), predict() 메서드

- 분류(Classification)와 회귀(Regression)에서 학습과 예측 결과 반환 가능
- **Classifier**: 분류 알고리즘을 구현한 클래스 + **Regressor**: 회귀 알고리즘을 구현한 클래스 = **Estimator** 클래스

사이킷런의 주요 모듈

분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위로 선택 작업 수행하는 다양한 기능 제공

피처 처리	<code>sklearn.feature_extraction</code>	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는 데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tfidf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 <code>sklearn.feature_extraction.text</code> 모듈에, 이미지 데이터의 피처 추출은 <code>sklearn.feature_extraction.image</code> 모듈에 지원 API가 있음.
피처 처리 & 차원 축소	<code>sklearn.decomposition</code>	차원 축소와 관련한 알고리즘을 지원하는 모듈임. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있음
데이터 분리, 검증 & 파라미터 튜닝	<code>sklearn.model_selection</code>	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	<code>sklearn.metrics</code>	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	<code>sklearn.ensemble</code>	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스팅, 그래디언트 부스팅 등을 제공
	<code>sklearn.linear_model</code>	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	<code>sklearn.naive_bayes</code>	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등.
	<code>sklearn.neighbors</code>	최근접 이웃 알고리즘 제공. K-NN 등
	<code>sklearn.svm</code>	서포트 벡터 머신 알고리즘 제공
	<code>sklearn.tree</code>	의사 결정 트리 알고리즘 제공
유틸리티	<code>sklearn.cluster</code>	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
	<code>sklearn.pipeline</code>	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

머신러닝 모델 구축하는 주요 프로세스:

- **처리** (feature processing): 피처의 가공, 변경, 추출 수행
- ML 알고리즘 **학습/예측** 수행
- 모델 **평가**의 단계 반복적으로 수행

내장된 예제 데이터 세트

- 사이킷런에는 예제로 활용할 수 있는 데이터 세트들이 있다
- 내장된 데이터 세트들은 회귀 용도, 분류 용도로 나뉜다

- **fetch 계열의 명령**: 데이터의 크기가 커서 패키지에 저장돼 있지 않고 인터넷-홈 디렉터리-scikit_learn_data에 저장-불러들이는 데이터다
- 내장된 데이터 세트는 일반적으로 딕셔너리 형태로 돼 있다
 - 키 예제들:

data	피처의 데이터 세트	넘파이 배열(ndarray)
target	분류 시 레이블 값, 회귀일 때는 숫자 결괏값 데이터 세트	넘파이 배열(ndarray)
target_names	개별 레이블의 이름	넘파이 배열 또는 파이썬 리스트
feature_names	피처의 이름	넘파이 배열 또는 파이썬 리스트
DESCR	데이터 세트에 대한 설명과 각 피처의 설명	스트링

- 코드 예시:

```
keys = iris_data.keys()
print('붓꽃 데이터 세트의 키들:', keys)

붓꽃 데이터 세트의 키들: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

04 Model Selection 모듈 소개

사이킷런의 **model_selection 모듈**은 학습 데이터와 테스트 데이터 세트를 분리하거나 교차 검증 분할 및 평가, 그리고 Estimator의 하이퍼 파라미터를 튜닝하기 위한 함수와 클래스 제공

학습/테스트 데이터 세트 분리 - train_test_split()

- 먼저 학습 데이터 세트로만 학습하고 예측해서 무엇이 문제인지 살펴본다

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
dt_clf = DecisionTreeClassifier()
train_data = iris.data
train_label = iris.target
dt_clf.fit(train_data, train_label)
```

```
#학습 데이터 세트로 예측 수행
pred = dt_clf.predict(train_data)
print('예측 정확도:', accuracy_score(train_label, pred))
```

-예측 정확도: 1.0 🤖

- 예측 결과가 100% 정확한 이유는 이미 학습한 학습 데이터 세트를 기반으로 예측했기 때문이다.
- 따라서 예측을 수행하는 데이터 세트는 학습을 수행한 학습용 데이터 세트가 아니라 **전용의 테스트 데이터 세트여야 한다**

따라서 train_test_split()을 통해 원본 데이터에 학습 및 테스트 데이터를 분리

- 코드: train_test_split([피쳐 데이터 세트], [레이블 데이터 세트])
- 선택적 파라미터 입력

test_size	전체 데이터에서 테스트 데이터 크기 결정 (디폴트: 0.25)
train_size	전체 데이터에서 학습용 데이터 크기 결정 (잘 사용하지 않음)
shuffle	데이터 분리하기 전에 미리 섞을지 결정 (디폴트: True)
random_state	- 호출할 때마다 동일함 학습/테스트용 데이터 세트를 생성하기 위해 주어지는 난수 값 - 지정하지 않으면 수행할 때마다 다른 학습/테스트 용 데이터 생성

train_test_split()의 반환값은 튜플 형태, 학습 피쳐, 테스트 피쳐, 학습 레이블, 테스트 레이블 순으로 반환됨

교차 검증

- 학습 데이터와 예측 성능 평가용의 테스트용 데이터의 약점: 과적합(Overfitting)
 - ▼ 과적합이란? 모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터 수행할 경우에는 예측 성능이 과도하게 떨어지는 것

모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터 수행할 경우에는 예측 성능이 과도하게 떨어지는 것
- **교차 검증**: 데이터 편향을 막기 위해 별도의 여러 세트로 구성된 학습 데이터 세트와 검증 데이터에서 학습과 평가를 수행
- 대부분 ML 모델의 성능 평가는 [교차 검증 기반으로 1차 평가] → [최종적으로 테스트 데이터 세트에 적용해 평가]으로 진행
- 데이터 세트 **학습, 검증, 테스트** 로 나뉨

K 폴드 교차 검증

- 가장 보편적으로 사용되는 교차 검증 기법
- K개의 데이터 폴드 세트를 만들어 K번만큼 각 폴드 세트에 학습과 검증 평가 반복적으로 수행

예시:



- K 폴드 교차 검증 위해 KFold와 StratifiedKFold 클래스 사용
- 코드

```
#n개의 폴드 세트로 분리하는 KFold 객체와 폴드 세트별 정확도를 담은 리스트  
kfold = KFold(n_splits=n)
```

교차 검증을 보다 간편하게 - cross_val_score()

- 선언 형태: `cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')`
 - 주요 파라미터 [estimator] (분류 알고리즘 클래스), [X], (피쳐 데이터 세트) [y] (레이블 데이터 세트), [scoring] (예측 성능 평가 지표), [cv] (교차 검증 폴드 수)
 - 코드

```
cross_val_score(dt_clf, data, label, scoring='accuracy', cv=3)
```

GridSearchCV - 교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번에

- 하이퍼 파라미터는 머신러닝 알고리즘을 구성하는 주요 구성 요소이며, 이 값을 조정해 알고리즘의 예측 성능을 개선할 수 있다
- GridSearchCV는 최적의 파라미터를 도출할 수 있는 방안 제공
- 예:
 - max_depth = [1, 2, 3]와 min_samples_split = [2, 3]에서 총 6회에 걸쳐 파라미터를 바꿔 실행
 - 6개의 파라미터 조합이라면 총 CV 3회 x 6회 파라미터 조합 = 18회의 학습/평가가 이뤄짐
- 여기서 rank_test_score()가 1이면 성능이 1위라는 뜻이다
- 테스트 이후 마지막에 GridSearchCV 객체의 fit()을 수행해서 최고 성능을 나타낸 하이퍼 파라미터의 값을 알아본다
- 최적의 하이퍼 파라미터를 알아본 후 Estimator를 학습해서 예측한다

05 데이터 전처리

- 결손값 (NaN, Null 값)은 허용되지 않는다
 - 방법: 1) 얼마 되지 않는다면 → 피처의 평균값으로 대체 2) 대부분 → 해당 피처 드롭 3) 해당 피처 중요한 높은 피처 & 평균값으로 대체안됨 → 업무 로직으로 정밀한 대체 값 선정
- 문자열 피처:
 - 카테고리형 피처와 텍스트형 피처
 - 카테고리형: 코드 값으로 표현
 - 텍스트형: 피처 벡터화 등의 기법으로 벡터화 OR 불필요한 피처인 경우 삭제 (예: 주민번호)

데이터 인코딩

- 레이블 인코딩 (Label encoding)과 원-핫 인코딩 (One Hot encoding)이 대표적
- 레이블 인코딩:
 - 카테고리 피처를 코드형 숫자 값으로 변환
 - 예: TV, 냉장고, 믹서 → 1, 2, 3로 각각 변환
 - '01', '02'도 문자열이므로 숫자형으로 변환돼야 함

- LabelEncoder 클래스로 구현 → fit()과 transform() 호출

```
encoder = LabelEncoder()
encoder.fit(변수명)
labels = encoder.transform(변수명)
```

이렇게 하면 labels에서 변수명이 각각 숫자형 값으로 변환된 걸 알 수 있다

- encoder.classes_를 이용해서 속성값을 확인하면 된다

주의: 가중치 → 중요도로 인식될 가능성 큼 ($1 > 2 \rightarrow 1$ 더 중요? 라고 인식) 레이블 인코딩은 선형회귀와 같은 ML 알고리즘에는 적용하지 않아야 함

- 원-핫 인코딩 (One-Hot Encoding)

- 피쳐 값의 유허에 따라 새로운 피쳐 추가해 고유 값에 해당하는 칼럼에만 1을 표시, 나머지 칼럼 0 표시하는 방식
 - 예: 고유 값 피쳐들은 'TV를 위한 상품 분류', '냉장고를 위한 상품 분류', 등으로 나뉘고 기존 열을 행으로 바꿔서 해당하는 컬럼에는 1, 나머지는 0을 표시
 - OneHotEncoder로 변환 가능

- 주의: 1) 변환하기 전에 모든 문자열 값이 숫자형 값으로 변환돼야 한다는 것 2) 입력 값으로 2차원 데이터 필요

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

#먼저 숫자 값으로 변환을 위해 LabelEncoder로 변환
encoder = LabelEncoder()
encoder.fit(변수명)
labels = encoder.transform(변수명)
#2차원 데이터로 변환
labels = labels.reshape(-1, 1)

#원-핫 인코딩 적용
oh_encoder = OneHotEncoder()
oh_encoder.fit(labels)
oh_labels = oh_encoder.transform(labels)
```

- 판다스에서 get_dummies()로 원-핫 인코딩 가능

- 사이킷런의 OneHotEncoder와 다르게 문자열 카테고리 숫자 형으로 변환 필요 x

```
import pandas as pd
pd.get_dummies(pd로 데이터 프레임화 시킨 변수명)
```

피처 스케일링과 정규화

- 피처 스케일링 (feature scaling): 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
 - 표준화와 정규화가 대표적
- StandardScaler와 MinMaxScaler - 사이킷런 대표 피처 스케일링 클래스
 - StandardScaler
 - 표준화를 쉽게 지원하는 클래스; 개별 피처를 평균이 0이고 분산이 1인 값으로 변환
 - 순서: StandardScaler 생성 → fit() → transform()

MinMaxScaler

- 데이터 값을 0과 1사이의 범위 값으로 변환합니다 (음수 값이 있으면 -1에서 1값으로 변환)
- 데이터의 분포가 가우시안 분포가 아닐 경우 적용

!학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점!

- Scaler 객체를 이용해 **학습 데이터 세트**로 fit()과 transform()을 적용하면 **테스트 데이터 세트**로는 다시 fit()을 수행하지 않고 **학습 데이터 세트**로 fit()을 수행한 결과를 이용해 transform() 변환을 적용

요약:

1. 가능하다면 전체 데이터의 스케일링 변환을 적용한 뒤 학습과 테스트 데이터로 분리
2. 1이 여의치 않다면 테스트 데이터 변환 시에는 fit()이나 fit_transform()을 적용하지 않고 학습 데이터로 이미 fit()된 Scaler 객체를 이용해 transform()으로 변화

Chapter 03: 평가

01 정확도(Accuracy)

- 정확도: 모델 예측 성능을 나타내는 평가 지표

- 사이킷런의 BaseEstimator 클래스를 상속받아 아무런 학습 하지 않고, 성별에 따라 생존자 예측하는 단순한 Classifier 생성
- 정확도는 불균형한 (imbalanced) 레이블 값 분포에서 ML 모델의 성능을 판단할 경우, 적합한 평가 지표가 아님

02 오차 행렬

- 오차행렬(confusion matrix, 혼동행렬): 학습된 분류 모델이 예측을 수행하면서 얼마나 헛갈리고 있는지 보여주는 지표
 - 이진 분류에서 활용
 - 4분면 행렬에서 실제 레이블 클래스 값과 예측 레이블 클래스 값이 어떠한 유형을 가지고 매핑되는지 나타냄
 - True Negative, False Negative, False Positive, True Positive
- 코드

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, fakepred)
```

- 정확도 = 예측 결과와 실제 값이 동일한 건수/전체 데이터 수 = $(TN + TP) / (TN + FP + FN + TP)$
- Positive = 1, Negative = 0, 로 부여한느 경우 많음

03 정밀도와 재현율

- 정밀도와 재현율: Positive 데이터 세트의 예측 성능에 좀 더 초점을 맞춘 평가 지표
- 정밀도 = $TP / (FP + TP)$
- 재현율 = $TP / (FN + TP)$
- 재현율이 중요 지표인 경우는 실제 Positive 양성 데이터를 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우

정밀도/재현율 트레이드오프

- 정밀도와 재현율은 상호 보완적인 평가 지표이기 때문에 어느 한쪽을 강제로 높이면 다른 하나의 수치는 떨어짐
 - 이를 정밀도/재현율의 트레이드오프(Trade-off)라 부름
- 사이킷런의 predict_proba() 이용해서 예측 확률 반환

- 사이킷런의 Binarizer 클래스를 이용해서 predict_proba()가 반환하는 확률값을 가진 ndarray의 칼럼 위치를 파악한다

```
from sklearn.preprocessing import Binarizer

X = [[1, -1, 2],
      [2, 0, 0],
      [0, 1.1, 1.2]]

#X의 개별 원소들이 threshold값보다 같거나 작으면 0을, 크면 1을 반환
binarizer = Binarizer(threshold=1.1)
print(binarizer.fit_transform(X))
```

임계값을 이후 낮추면 재현율 값이 올라가로 정밀도가 떨어진다

- Positive 예측값이 많아지면 상대적으로 재현율 값이 높아진다.
 - 양성 예측을 많이 한다 → 실제 양성을 음성으로 예측하는 횟수 줄어든다
- 또한 시각화를 통해 정밀도와 재현율의 임계값에 따른 값 변화를 표현할 수 있다

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline
```

정밀도와 재현율의 맹점

- 정밀도가 100% 되는 방법
 - 확실한 기준이 되는 경우만 Positive, 나머지 모두 Negative로 예측하기
- 재현율이 100% 되는 방법
 - 모든 환자를 Positive로 예측하기

04 F1 스코어

- F1 스코어: 정밀도와 재현율을 결합한 지표
 - 높은 값: 어느 한쪽으로 치우치지 않음
 - f1_score() 이용

```
f1 = f1_score(y_test, pred)
```

05 ROC 곡선과 AUC

- ROC 곡선과 AUC 스코어: 이진 분류의 예측 성능 측정에서 중요하게 사용되는 지표
- 사이킷런에서 ROC 곡선 구하기 위해 `roc_curve()` 이용
- 코드: 레이블 값이 1일 때의 예측 확률을 추출하고 반환된 임계값 배열 로우 수를 파악한 후 단위로 추출된 임계값에 FPR, TPR 값을 가져온다
- 코드를 통해 FPR의 변화에 따른 TPR의 변화를 ROC 곡선으로 시각화할 수 있다