단어장

사이킷런(scikit-learn)

* 머신러닝 알고리즘과 도구들을 쉽게 사용할 수 있도록 제공하는 파이썬 라이브러리

분류(classification)

* 머신러닝에서 여러 개의 종류(혹은 클래스(class)라고 부름) 중 하나를 구별해 내는 문제

이진 분류(binary classification)

* 2개의 클래스 중 하나를 고르는 문제

특성(feature)

* 각 데이터의 특징(도미의 길이와 무게)

산점도(scatter plot)

* x, y축으로 이뤄진 좌표계에 두 변수(x, y)의 관계를 표현하는 방법

맷플롯립(matplotlib)

* 파이썬에서 과학계산용 그래프를 그리는 대표적인 패키지

임포트(import)

* 따로 만들어둔 파이썬 패키지(클래스와 함수의 묶음)를 사용하기 위해 불러오는 명령

선형(linear)

* 산점도 그래프가 일직선에 가까운 형태로 나타나는 경우

훈련(training)

* 모델에 데이터를 전달하여 규칙을 학습하는 과정

정확도(accuracy)

* score()를 통해서 0에서 1사이의 값 1에 가까울 수록 높은 정확도를 가짐

지도 학습(supervised learning)

* 데이터와 정답 (훈련데이터)을 이용하여 훈련하는 알고리즘
* 분류와 회귀로 나뉜다

입력(input)과 타깃(target)

* 데이터 = 입력, 정답 = 타깃

훈련 데이터(training data)

* 입력과 데이터를 합쳐 부르는 말

테스트 세트(test set)

* 평가에 사용하는 데이터, 보통 전체에 20 ~ 30%를 테스트 세트로 사용

훈련 세트(train set)

훈련에 사용되는 데이터

샘플(sample)

두 파이썬 리스트를 순회하면서 각 생선의 길이와 무게를 하나의 리스트로 담은 2차원 리스트를 만들고 이때 생성된 하나의 생성 데이터

샘플링 편향(sampling bias)

* 훈련 세트와 테스트 세트에 샘플이 골고루 섞여 있지 않으면 샘플링이 한쪽으로 치우쳤다는 의미

배열 인덱싱(array indexing)

* 1개의 인덱스가 아닌 여러 개의 인덱스로 한 번에 여러 개의 원소를 선택할 수 있습니다.

스케일(scale)

* 특정 기준에 따라서 크기를 조정하는 작업

데이터 전처리

* 머신러닝 모델에 훈련 데이터를 주입하기 전에 가공하는 단계
* 알고리즘들은 샘플 간의 거리에 영향을 많이 받으므로 제대로 사용하려면 특성값을 일정한 기준으로 맞춰는 것

표준편차

* 데이터들이 평균(산술평균)을 중심으로 얼마나 퍼져 있는지를 나타내는 값

표준점수

* 훈련 세트의 스케일을 바꾸는 대표적인 방법 중 하나
* 각 특성값이 평균에서 표준편차의 몇 배만큼 떨어져 있는지를 나타냄
* 표준 점수의 평균은 0이다

브로드캐스팅

* 크기가 다른 넘파이 배열에서 자동으로 사칙 연산을 모든 행이나 열로 확장하여 수행하는 기능

결정계수(coefficient of determination) 또는 R\*\*2

* 회귀 모델을 평가하는 값

과대적합(overfitting)

* 훈련 세트에서 점수가 굉장히 좋았는데 테스트 세트에서는 점수가 굉장히 나쁜 모델일 경우

과소적합(underfitting)

* 훈련 세트보다 테스트 세트의 점수가 높거나 두 점수가 모두 너무 낮은 경우

선형 회귀(linear regression)

* 대표적인 회귀 알고리즘
* 특성과 타깃 사이의 관계를 가장 잘 나타내는 선형 방정식
* 선형 회귀가 찾은 특성과 타깃 사이의 관계는 선형 방정식의 계수 또는 가중치에 저장됨

모델 파라미터

* 선형 회귀가 찾은 가중치처럼 머신러닝 모델이 특성에서 학습한 파라미터를 말합니다.

파라미터

* 모델이 학습을 통해 얻는 내부 값입니다.
* 데이터를 기반으로 훈련과정에서 자동으로 조정되는 값

다항 회귀

* 다항식을 사용하여 특성과 타깃 사이의 관계를 나타납니다.

다중 회귀(multiple regression)

* 여러 개의 특성을 사용한 선형 회귀

특성 공학(feature engineering)

* 기존의 특성을 사용해 새로운 특성을 뽑아내는 작업

판다스(pandas)

* 유명한 데이터 분석 라이브러리

데이터프레임(dataframe)

* 판다스의 핵심 데이터 구조

변환기(transformer)

* [사이킷런](#kix.nwiulsbuibg7)은 특성을 만들거나 전처리하기 위한 다양한 클래스

matplotlib

> import matplotlib.pyplot as plt

* matplotlib의 pyplot 모듈을 plt 이름으로 불러옵니다. 그래프를 그릴 때 사용합니다.

> plt.scatter( x축 값, y축 값 [, marker= ‘’])

* x축과 y축 데이터를 이용해 **산점도(Scatter plot)**를 그립니다. (여러개 사용가능 / 색상이 다름)
* marker : 그래프에서 모양을 바꿈(기본 : 원형, ‘^’ : 삼각형, ‘D’ : 네모)

> plt.plot( [ 시작, 끝 ] , [ 시작 \* [lr.coef\_](#nrr5itzfhqi1) + [lr.intercept](#rjjotbe7o71m)\_, 끝 \* [lr.coef\_](#nrr5itzfhqi1) +

[lr.intercept\_](#rjjotbe7o71m) ] )

* 1차 방정식 그래프
* 두 숫자를 각각의 식에 넣어서 나온 값을 연결하는 것

> plt.xlim(( 시작, 끝 ))

* 그래프의 x축의 크기를 지정

> plt.xlabel( “x축 이름” )

* x축에 이름을 붙입니다.

> plt.ylabel( “y축 이름” )

* y축 이름 붙이기

> plt.show()

* plt.에 정해진 값 출력

이웃 분류 알고리즘

> from sklearn.neighbors import KNeighborsClassifier

> kn = KNeighborsClassifier()

* k-최근접 이웃 알고리즘을 구현한 클래스

> kn.\_fit\_X

* kn에 fit를 통해 넣은 입력값

> kn.\_y

* kn에 fit를 통해 넣은 타깃값

> kn[개수] = KNeighborsClassifier(n\_neighbors = 참고할 가까운 데이터 개수)

* 참고 데이터를 지정할 수 있음 (클래스 기본값 5)

이웃 알고리즘 공통

공통사용?

> zip(리스트 데이터)

* 나열된 리스트 각각에서 하나씩 원소를 꺼내 반환(for)

> \_\_.fit( [ [ 데이터, 정답 ] ])

* 주어진 데이터로 알고리즘을 훈련

> \_\_.score( 데이터, 정답 )

* 모델을 평가함, 0에서 1사이의 값을 반환

> \_\_.predict( [ [ 예측하고 싶은 데이터 값 ] ] )

* 예측할 데이터는 반드시 2차원 배열로 넣어야 합니다. (예: [[25, 150]])

거리, 위치 = kn.kneighbors( [ [ x, y ] ] )

* 주어진 샘플에서 가장 가까운 이웃을 찾아 주는 메서드 ( (x,y)에서 이웃 까지의 거리, 데이터에서 어느 위치에 있는지를 출력 | 기본 5개)

> from sklearn.model\_selection import train\_test\_split

> 훈련\_input, 테스트\_input, 훈련\_target, 테스트\_target = train\_test\_split(

> fish\_data, fish\_target [, random\_state=42])

* train\_test\_split() : 전달되는 리스트나 배열을 비율에 맞게 훈련 세트와

테스트 세트로 나누어 줌

* random\_state : 랜덤 seed를 고정시켜 줌

> from sklearn.metrics import mean\_absolute\_error

> mean\_absolute\_error( 정답 데이터 , 모델이 예측한 값 )

* 회귀모델의 예측 정확도를 평가할 때 사용하는 오차 측정 지표 중 하나
* | 정답 데이터 - 모델이 예측한 값 | 의 평균값을 출력

> from sklearn.preprocessing import PolynomialFeatures ([변환기](#iktyfw8mk56))

> poly = PolynomialFeatures([include\_bias = False])

* 선형 회귀에 다항식을 적용할 수 있도록, 기존 입력 특성을 제곱하거나 조합해서 확장하는 도구
* include\_bias : 1을 포함 할건지 안할건지

> poly.fit([[ 숫자, 숫자 ]])

* 입력 데이터의 특성 개수를 기준으로, 변환에 사용할 다항식 항의 구조(열의 수)를 미리 계산하여 저장함.
* [[ 숫자, 숫자 ]] : 1행 2열 → 샘플 1개, 특성 2개
* 모델이 상용하는 fit와 다름

> poly.transform([[ 숫자, 숫자]])

* fit()에서 정한 규칙(다항식 항의 구조)에 따라, 실제 데이터를 변환함
* 결과 [1, x1, x2, x1², x1·x2, x2²] 형식으로 출력
* 기본적으로 2차항 까지 생성함(기본적으로 degree = 2)
* 모든 특성이 0일 때도, 출력에 영향을 줄 수 있는 향을 남기기 위해 1을 넣는다(기본값 : inlude\_bias = True)

numpy

> import numpy as np

* numpy라이브러리를 가지고 옴

> np.array( 데이터 )

* 데이터를 하나의 데이터 안에 넣음

> 배열.shape

* 배열의 크기를 알려줌( (3,) = 1차원, 원소3개 | (2, 3) = 2행 3열, 2차원 | (2, 2, 2) = (깊이 2, 행 2, 열 2), 3차원 )

> np.random.seed( 숫자 )

* 랜덤값을 고정함

> np.arange( 숫자 )

* 0부터 “숫자” - 1까지의 숫자를 가진 리스트를 출력

> np.random.shuffle( 배열 )

* 안에 순서를 섞음

> arr[[ 범위 ]]

* 여러 개의 인덱스를 한 번에 넘겨서, 해당 위치의 값을 선택함
* 예: arr[[1, 3]] → 인덱스 1과 3의 값

> arr1[arr2[범위]]

* arr2의 0~9까지 값들을 인덱스로 사용해서 arr1에서 해당 위치 값을 꺼냄
* 예: arr2 = [3, 0, 2, ...]이면 → arr1[3], arr1[0], arr1[2], ...

> arr[ 숫자1 , 숫자2 ] | arr[ 숫자1 , 숫자2 ]

* 2차원 배열에서 값을 찾음 숫자1 = 행, 숫자2 = 열을 나타냄

> np.column\_stack((리스트1 , 리스트2 … ))

* 행과 열을 바꾸어줌(리스트들을 묶을 때 튜플로 묶음) return은 배열로 받음

> np.ones( 숫자 ) | np.zeros( 숫자 )

* “숫자”만큼 채워진 리스트를 생성(np.ones(3) = [1. 1. 1.], np.zeros(3) = [0. 0. 0.])

> np.mean( 데이터 , axis = 0 )

> np.std( 데이터 , axis = 0 )

* mean = 평균을 계산
* std = 표준편차를 계산

> (데이터 - mean) / std

* 표준점수를 구하는 식

> 배열.reshape(행, 열)

* 배열을 원하는 행과 열로 나누어 준다(배열의 요소 = 행 \* 열이여야 한다.)
* -1 입력시 다른 차원을 채우고 남은 만큼 해줌( 전체:42 (-1, 1) = (42, 1)

이웃 회귀 알고리즘

> from sklearn.neighbors import KNeighborsRegressor

> knr = KNeighborsRegressor(n\_neighbors = 개수)

* 이웃 회귀 알고리즘을 구현할 클래스
* n\_neighbors : 참고할 이웃의 수를 결정

이웃 선형 회귀 알고리즘

> from sklearn.linear\_model import LinearRegression

> lr = LInearregression()

* 이웃 선형 회귀 알고리즘

> lr.coef\_

* 입력 변수(특성)마다 곱해지는 값(기울기, weight값)

> lr.intercept\_

* x = 0일대 y값, 즉 그래프에서 y축과 만나는 지점(절편, bias)

pandas

> import pandas as pd

* 판다스 라이브러리 가지고 옴

> pd.read\_csv(‘주소’)

* 주소에 있는 데이터를 데이터프레임으로 만듬

> 데이터프레임.head()

* 처음 5개의 행을 보여준다