Day 2: Machine Learning Basics (Lab)

May 31, 2018

Prof. Jongwuk Lee

Contents

- **선형 회귀(Linear Regression) 구현**
- **로지스틱 회귀(Logistic Regression) 구현**
- **선형 회귀(Linear regression) 응용**
  - 세계 행복 지수 예측 (함께하기)
    - 데이터 읽기
    - 모델 학습
    - 모델 평가
  - 도요자 자동차 가격 예측 (직접 실습하기)
  - 집 가격 예측 (직접 실습하기)
- **분류 모델(classification) 응용**
  - 로지스틱 회귀(Logistic regression)
  - 결정 트리(Decision Tree)
  - 나이브 베이지안 분류(Naive Bayesian Classification)
- **평가 방법(Evaluation Metrics)**
  - 각 모델을 다양한 평가 방법으로 비교하기

선형 회귀(Linear Regression) 구현

템플릿 코드

- main 함수

```
X_train, Y_train = data_load('BloodPressure', 'train')
model = LinearRegression(X_train, Y_train)
model.train(0.0, 1) # learning rate, epoch

X_test, Y_test = data_load('BloodPressure', 'test')
prediction = model.predict(X_test)

rmse = eval(prediction, Y_test)
print("%.2f"%(rmse))
```

main 함수

➢ 학습 데이터를 이용하여 모델 학습

```
X_train, Y_train = data_load('BloodPressure', 'train')
model = LinearRegression(X_train, Y_train)
model.train(0.0, 1) # learning rate, epoch
```

- data_load(): 데이터를 읽어옴
- LinearRegression(): LinearRegression 객체 생성
- LinearRegression.train(): LinearRegression 모델을 학습함

main 함수

➢ 테스트 데이터를 이용하여 모델 테스트

```
X_test, Y_test = data_load('BloodPressure', 'test')
prediction = model.predict(X_test)
```

- data_load(): 데이터를 읽음
- LinearRegression.predict(data): 학습된 모델을 이용하여 data에 대해 선형 회귀를 수행

```
rmse = eval(prediction, Y_test)
print("%.2f"%(rmse))
```

- eval(): 예측값과 실제 값의 RMSE를 구함

템플릿 코드

➢ data_load 함수

```
def data_load(dana_name, tr_te):
    y = []
    x = []

    if dana_name == 'BloodPressure':
        if(tr_te == 'train'):
            file = './data/bloodPressure/train.csv'
        elif(tr_te == 'test'):
            file = './data/bloodPressure/test.csv'

        with open(file, 'r') as csvfile:
            csv_reader = csv.reader(csvfile)
            next(csv_reader) # head 건너 뜀
            for row in csv_reader:
                # row[0]: Age, row[1]: Blood Pressure
                x.append([1, float(row[0])]) # 편향(Bias)추가
                y.append([float(row[1])])

return np.array(x), np.array(y)
```

data_load 함수

➢ 데이터 경로 설정

```
if dana_name == 'BloodPressure':
        if(tr_te == 'train'):
            file = './data/bloodPressure/train.csv'
        elif(tr_te == 'test'):
            file = './data/bloodPressure/test.csv'
```

- 함수의 인자에 알맞은 데이터 셋을 읽음

data_load 함수

➢ csv 라이브러리를 이용한 파일 읽기

```
with open(file, 'r') as csvfile:
            csv_reader = csv.reader(csvfile)
            next(csv_reader)
            for row in csv_reader:
                x.append([1, float(row[0])])
                y.append(float(row[1]))
```

- csv.reader(): csv파일 읽기를 지원
- next(): csv파일의 첫 줄을 읽음
- float(): float 형 데이터로 형 변환

템플릿 코드

➢ LinearRegression 클래스

```
class LinearRegression():
    def __init__(self, x, y): # 생성자
        self.x = x # 변수 초기화
        self.y = y
        self.num_data = self.x.shape[0]
        self.num_feature = self.x.shape[1]
        self.w = np.zeros((self.num_feature, 1))

    def predict(self, x):
        ##################################################
        #  코드 작성
        ##################################################
        return result

    def train(self, lr, epoch):
        for i in range(0, epoch):
            ##################################################
            #  코드 작성
            ##################################################
```

LinearRegression 클래스

➢ 생성자를 이용한 변수 초기화

```
def __init__(self, x, y): # 생성자
        self.x = x # 변수 초기화
        self.y = y
        self.num_data = self.x.shape[0]
        self.num_feature = self.x.shape[1]
        self.w = np.zeros((self.num_feature, 1))
```

- def __init__(): 클래스의 생성자
- self: 해당 객체를 가리킴(C++의 this와 같음)
- x.shape: 데이터 x의 차원 정보
- np.zeros(): 0으로 된 array를 만듦
- np.zeros((x, y)): x행 y열의 array를 만듦

LinearRegression 클래스

➢ predict()를 이용한 예측과 train()을 이용한 모델 학습

```
def predict(self, x):
      #  코드 작성
      return result

def train(self, lr, epoch):
       for i in range(0, epoch):
           #  코드 작성
```

- predict(self, x): 입력 x에 대한 선형회귀 결과 result를 반환
- train(self, lr, epoch): 생성자에서 초기화된 훈련 데이터 self.x와 self.y를 이용하여 모델을 학습

템플릿 코드

➢ eval 함수

```
def eval(prediction, y):
    inner = np.power((prediction - y), 2)
    return np.sqrt(np.sum(inner, axis=0) / len(y))
```

- eval(prediction, y): 예측 값 prediction과 실제 값 y를 이용하여 RMSE를 계산
- np.power(x, 2): x^2을 계산
- np.sum(x, axis): axis(0: 세로, 1:가로)을 기준으로 값을 더함
- len(y): y의 길이

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(pred_k - true_k)^2}$$

n: 데이터의 수
$pred_k$: k번째 데이터의 예측 값
$true_k$: k번째 데이터의 실제 값

구현에 유용한 함수 설명

- **enumerate(): for문에서 사용되는 함수**

```
a=['일', '이', '삼', '사', '오']
for num in a:
    print('요소:', num)
    print('----------')
```

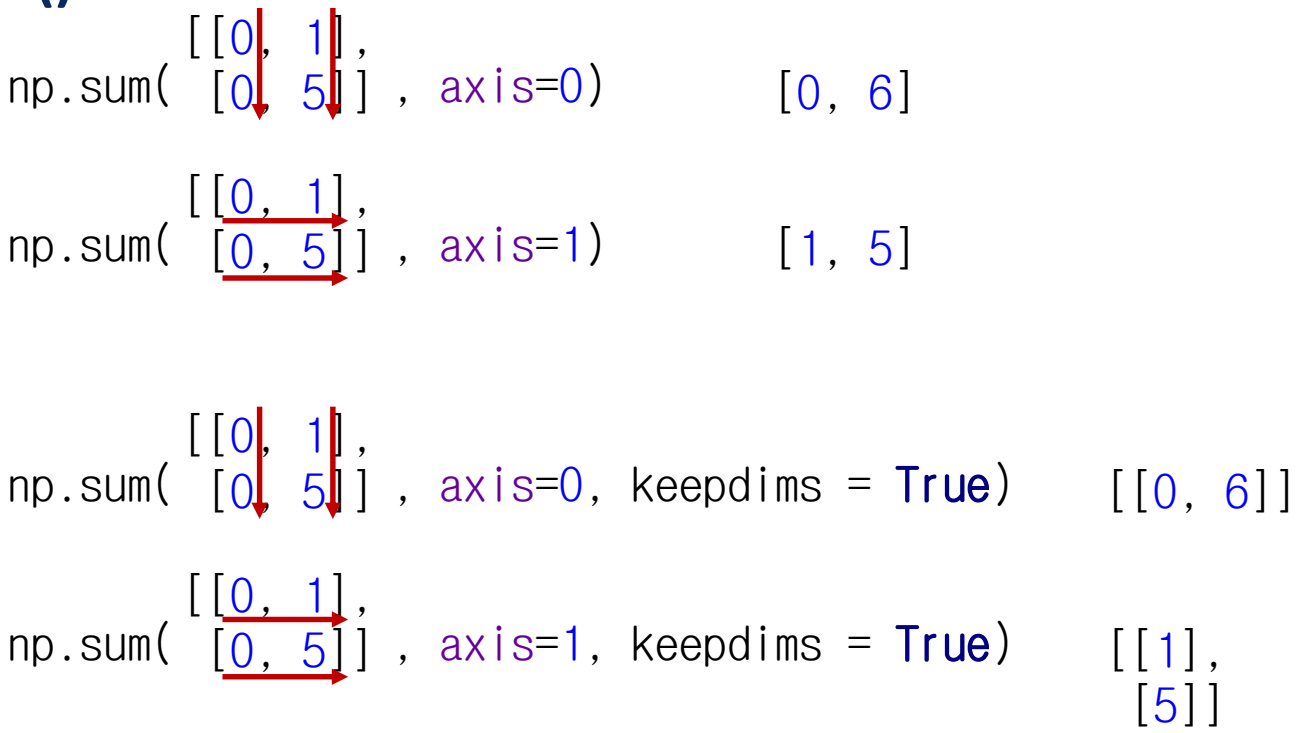
```
요소: 일
----------
요소: 이
----------
요소: 삼
----------
요소: 사
----------
요소: 오
----------
```

```
a=['일', '이', '삼', '사', '오']
for i, num in enumerate(a):
    print('인덱스:',i)
    print('요소:', num)
    print('----------')
```

```
인덱스: 0
요소: 일
----------
인덱스: 1
요소: 이
----------
인덱스: 2
요소: 삼
----------
인덱스: 3
요소: 사
----------
인덱스: 4
요소: 오
----------
```

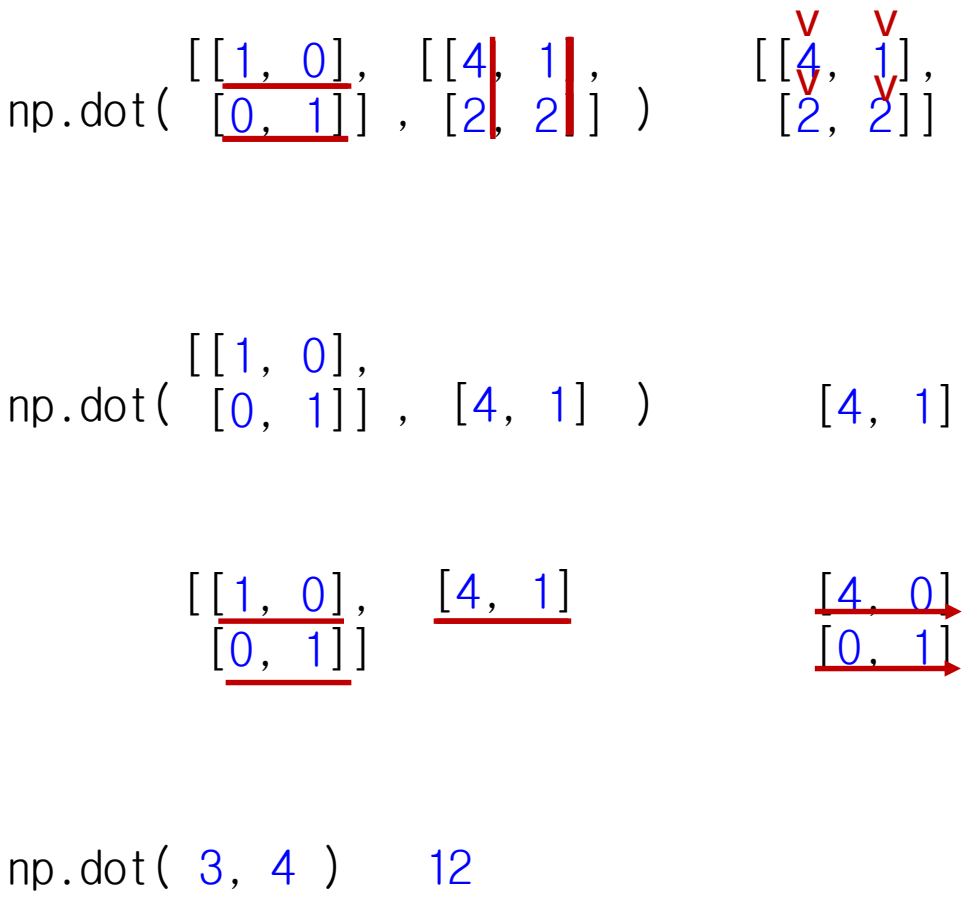
구현에 유용한 함수 설명

- **np.sum()**



구현에 유용한 함수 설명

- **np.dot()**



구현에 유용한 함수 설명

➢ np.where()

x > 1 을 만족하는 행 인덱스

```
x = [[0, 1],
     [2, 3]]

np.where( x > 1 )      (array([1, 1]), array([0, 1]))
```

x > 1 을 만족하는 열 인덱스

```
x = [0, 1]

np.where( x > 0 )      (array([1]))
```

x > 1 을 만족하는 인덱스

실습:선형 회귀(Linear Regression) 구현

- **템플릿 코드의 비어 있는 부분을 채워서 선형회귀가 잘 동작 하는 코드 구현**

1. 혈압 예측
   - 나이(Age)를 이용하여 혈압 예측
   - 잘 구현된 경우의 RMSE: 13.39

2. 물고기 크기 예측
   - 물고기의 나이(일), 물의 온도를 이용하여 물고기의 크기를 예측
   - 잘 구현된 경우 RMSE: 571.01

로지스틱 회귀(Logistic Regression) 구현

템플릿 코드

- main 함수

```
X_train, Y_train = data_load('Iris', 'train')
model = LogisticRegression(X_train, Y_train)
model.train(0.0, 1) # learning rate, epoch

X_test, Y_test = data_load('Iris', 'test')
prediction = model.predict(X_test)

accuracy = eval(prediction, Y_test)
print("%.2f" % (accuracy))
```

main 함수

➢ 학습 데이터를 이용하여 모델 학습

```
X_train, Y_train = data_load('Iris', 'train')
model = LogisticRegression(X_train, Y_train)
model.train(0.0, 1) # learning rate, epoch
```

- data_load(): 데이터를 읽음
- LogisticRegression(): LogisticRegression 객체 생성
- LogisticRegression.train(): LogisticRegression 모델을 학습함

main 함수

➢ 테스트 데이터를 이용하여 모델 테스트

```
X_test, Y_test = data_load('Iris', 'test')
prediction = model.predict(X_test)
```

- data_load(): 데이터를 읽음
- LogisticRegression.predict(data): 학습된 모델을 이용하여 data에 대해 로지스틱 회귀를 수행

```
accuracy = eval(prediction, Y_test)
print("%.2f" % (accuracy))
```

- eval(): 예측값과 실제 값의 accuracy를 구함

템플릿 코드

➢ data_load 함수

```
def data_load(dana_name, tr_te):
    y = []
    x = []
    if dana_name == 'Iris':
        if(tr_te == 'train'):
            file = './data/iris/train.csv'
        elif(tr_te == 'test'):
            file = './data/iris/test.csv'

        with open(file, 'r') as csvfile:
            csv_reader = csv.reader(csvfile)
            next(csv_reader) # head 건너 뜀
            for row in csv_reader:
                if (row[4] == 'Iris-setosa'):
                    y.append([float(0)])
                else:
                    y.append([float(1)])
                feature = [1] # 편향(Bias)추가
                feature.append(float(row[0])) # sepal length in cm
                feature.append(float(row[1]))  # sepal width in cm
                feature.append(float(row[2]))  # petal length in cm
                feature.append(float(row[3]))  # petal width in cm
                x.append(feature)
    return np.array(x), np.array(y)
```

data_load 함수

➢ 데이터 경로 설정

```
if dana_name == 'Iris':
        if(tr_te == 'train'):
            file = './data/iris/train.csv'
        elif(tr_te == 'test'):
            file = './data/iris/test.csv'
```

- 함수의 인자에 알맞은 데이터 셋을 읽음

data_load 함수

➢ csv 라이브러리를 이용한 파일 읽기

```
with open(file, 'r') as csvfile:
            csv_reader = csv.reader(csvfile)
            next(csv_reader) # head 건너 뜀
            for row in csv_reader:
                if (row[4] == 'Iris-setosa'):
                    y.append([float(0)])
                else:
                    y.append([float(1)])
                feature = [1] # 편향(Bias)추가
                feature.append(float(row[0])) # sepal length in cm
                feature.append(float(row[1]))  # sepal width in cm
                feature.append(float(row[2]))  # petal length in cm
                feature.append(float(row[3]))  # petal width in cm
                x.append(feature)
```

- csv.reader(): csv파일 읽기를 지원
- next(): csv파일의 첫 줄을 읽음
- float(): float 형 데이터로 형 변환

템플릿 코드

➢ LogisticRegression 클래스

```
class LogisticRegression():
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.num_data = self.x.shape[0]
        self.num_feature = self.x.shape[1]
        self.w = np.np.zeros((self.num_feature, 1))

    def _sigmoid(self, z):
        return  #  코드 작성

    def predict(self, x):
        #  코드 작성
        index = np.where(predic > 0.5)[0]
        result = np.zeros(predic.shape)
        result[index] = 1
        return result

    def train(self, lr, epoch):
        for i in range(0, epoch):
            #  코드 작성
```

LogisticRegression 클래스

➢ 생성자를 이용한 변수 초기화

```
def __init__(self, x, y): # 생성자
        self.x = x # 변수 초기화
        self.y = y
        self.num_data = self.x.shape[0]
        self.num_feature = self.x.shape[1]
        self.w = np.np.zeros((self.num_feature, 1))
```

- def __init__(): 클래스의 생성자
- self: 해당 객체를 가리킴(C++의 this와 같음)
- x.shape: 데이터 x의 차원 정보
- np.zeros(): 0으로 된 array를 만듦
- np.zeros((x, y)): x행 y열의 array를 만듦

LogisticRegression 클래스

➢ predict()를 이용한 예측과 train()을 이용한 모델 학습

```
def _sigmoid(self, z):
    return  #  코드 작성

def predict(self, x):
    #  코드 작성
    index = np.where(predic > 0.5)[0]
    result = np.zeros(predic.shape)
    result[index] = 1
    return result

def train(self, lr, epoch):
    for i in range(0, epoch):
        #  코드 작성
```

- _sigmoid(self, z): x에 대한 sigmoid 결과를 반환
- predict(self, x): 입력 x에 대한 선형회귀 결과 result를 반환
- train(self, lr, epoch): 생성자에서 초기화된 훈련 데이터 self.x와 self.y를 이용하여 모델을 학습

템플릿 코드

➢ eval 함수

```
def eval(prediction, y):
    hit = 0
    for i, p in enumerate(prediction):
        if p == y[i]:
            hit += 1
    return hit/len(y)
```

- eval(prediction, y): 예측 값 prediction과 실제 값 y를 이용하여 accuracy를 계산
- enumerate(x): for문에서 index와 값을 반환
- len(y): y의 길이
- Accuracy: $\frac{hit}{n}$

hit: 예측과 정답이 같음
n: 데이터의 수

실습:로지스틱 회귀(Logistic Regression) 구현

- **템플릿 코드의 비어 있는 부분을 채워서 로지스틱 회귀가 잘 동작 하는 코드 구현**

1. 붓꽃 종류 분류
   - 꽃받침의 너비/길이와 꽃잎의 너비/길이를 이용하여 Iris-setosa와 Iris-setosa가 아닌 종류를 분류
   - 잘 구현된 경우의 Accuracy: 1.0

2. 타이타닉 생존자 분류
   - 타이타닉호 탑승자의 여객 등급(Pclass), 성별, 나이, 사촌 탑승여부, 가족 탑승여부, 표 값을 이용하여 탑승자의 생존 여부 분류
   - 잘 구현된 경우 Accuracy: 0.82

선형 회귀(Linear Regression) 응용

Happiness 데이터 소개

- 국가 별 각종 지표에 대한 정보 및 행복지수 포함

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Country	Region	Happiness	Happiness	Lower Cor	Upper Cor	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia
2	Ukraine	Central an	123	4.324	4.236	4.412	0.87287	1.01413	0.58628	0.12859	0.01829	0.20363	1.50066
3	Somaliland	Sub-Sahar	97	5.057	4.934	5.18	0.25558	0.75862	0.33108	0.3913	0.36794	0.51479	2.43801
4	Iran	Middle Ea	105	4.813	4.703	4.923	1.11758	0.38857	0.64232	0.22544	0.0557	0.38538	1.99817
5	Tanzania	Sub-Sahar	149	3.666	3.561	3.771	0.47155	0.77623	0.357	0.3176	0.05099	0.31472	1.37769
6	Zambia	Sub-Sahar	106	4.795	4.645	4.945	0.61202	0.6376	0.23573	0.42662	0.11479	0.17866	2.58991
7	Chile	Latin Ame	24	6.705	6.615	6.795	1.2167	0.90587	0.81883	0.37789	0.11451	0.31595	2.95505
8	Romania	Central an	71	5.528	5.427	5.629	1.1697	0.72803	0.67602	0.36712	0.00679	0.12889	2.45184
9	Taiwan	Eastern As	34	6.379	6.305	6.453	1.39729	0.92624	0.79565	0.32377	0.0663	0.25495	2.61523
10	North Cyp	Western E	62	5.771	5.67	5.872	1.31141	0.81826	0.84142	0.43596	0.16578	0.26322	1.93447
11	Chad	Sub-Sahar	144	3.763	3.672	3.854	0.42214	0.63178	0.03824	0.12807	0.04952	0.18667	2.30637

Happiness 데이터 소개

➢ 국가 별 각종 지표에 대한 정보 및 행복지수 포함

- 데이터 수 : 총 157개
- 변수(features)
  - 독립 변수: Economy (1인당 GDP), Family (가족 수), Health (기대 수명), Freedom (자유), Trust (정부 신뢰도), Generosity (관대함), Dystopia (미래 불안 지수)
  - 종속 변수: Happiness Score (행복지수)

➢ 실습 목표

- 각 국가 별로 관찰된 독립 변수를 바탕으로 그 국가의 행복지수를 예측하는 선형 회귀 모델 구현

데이터 읽어 오기

```
import pandas as pd
import numpy as np

train_input = pd.read_csv(“Happiness_train.csv”)
test_input = pd.read_csv(“Happiness_test.csv”)

features = ['Economy', 'Family', 'Health', 'Freedom',
'Trust', 'Generosity', 'Dystopia']

x_train = np.array(train_input[features])
y_train = np.array(train_input[‘Happiness Score’])
x_test = np.array(test_input[features])
y_test = np.array(test_input[‘Happiness Score’])
```

데이터 읽어 오기

➢ read_csv

- csv 파일을 읽어서 DataFrame 형태로 저장하는 pandas 함수

```
train_input = pd.read_csv("Happiness_train.csv")
test_input = pd.read_csv("Happiness_test.csv")
```

- 읽은 데이터가 어떤 구조를 이루는지 파악하기

```
print(train_input.head())
```

- DataFrame 변수의 전체 데이터 중 앞부분 일부만 출력

	Country	Region	Happiness Rank	Happiness Score	Lower Confidence Interval	Upper Confidence Interval	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia
0	Ukraine	Central and Eastern Europe	123	4.324	4.236	4.412	0.87287	1.01413	0.58628	0.12859	0.01829	0.20363	1.50066
1	Somaliland Region	Sub-Saharan Africa	97	5.057	4.934	5.180	0.25558	0.75862	0.33108	0.39130	0.36794	0.51479	2.43801
2	Iran	Middle East and Northern Africa	105	4.813	4.703	4.923	1.11758	0.38857	0.64232	0.22544	0.05570	0.38538	1.99817
3	Tanzania	Sub-Saharan Africa	149	3.666	3.561	3.771	0.47155	0.77623	0.35700	0.31760	0.05099	0.31472	1.37769
4	Zambia	Sub-Saharan Africa	106	4.795	4.645	4.945	0.61202	0.63760	0.23573	0.42662	0.11479	0.17866	2.58991

...

데이터 읽어 오기

➢ DataFrame 파싱

- DataFrame에서 특정 column에 해당하는 데이터를 호출

```
y_train = train_input['Happiness Score']
```

- 호출한 데이터를 np.array의 형태로 저장

```
y_train = np.array(train_input['Happiness Score'])
```

  - DataFrame 변수를 array로 형변환

	Country	Region	Happiness Rank	Happiness Score	Lower Confidence Interval	Upper Confidence Interval	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia
0	Ukraine	Central and Eastern Europe	123	4.324	4.236	4.412	0.87287	1.01413	0.58628	0.12859	0.01829	0.20363	1.50066
1	Somaliland Region	Sub-Saharan Africa	97	5.057	4.934	5.180	0.25558	0.75862	0.33108	0.39130	0.36794	0.51479	2.43801
2	Iran	Middle East and Northern Africa	105	4.813	4.703	4.923	1.11758	0.38857	0.64232	0.22544	0.05570	0.38538	1.99817
3	Tanzania	Sub-Saharan Africa	149	3.666	3.561	3.771	0.47155	0.77623	0.35700	0.31760	0.05099	0.31472	1.37769
4	Zambia	Sub-Saharan Africa	106	4.795	4.645	4.945	0.61202	0.63760	0.23573	0.42662	0.11479	0.17866	2.58991

...

데이터 읽어 오기

➢ DataFrame 파싱

- 참조하고자 하는 모든 변수 명을 취합한 리스트를 정의하여 데이터를 리스트 단위로 호출

```
features = ['Economy', 'Family', 'Health', 'Freedom',
'Trust', 'Generosity', 'Dystopia’]
x_train = np.array(train_input[features])
```

	Country	Region	Happiness Rank	Happiness Score	Lower Confidence Interval	Upper Confidence Interval	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia
0	Ukraine	Central and Eastern Europe	123	4.324	4.236	4.412	0.87287	1.01413	0.58628	0.12859	0.01829	0.20363	1.50066
1	Somaliland Region	Sub-Saharan Africa	97	5.057	4.934	5.180	0.25558	0.75862	0.33108	0.39130	0.36794	0.51479	2.43801
2	Iran	Middle East and Northern Africa	105	4.813	4.703	4.923	1.11758	0.38857	0.64232	0.22544	0.05570	0.38538	1.99817
3	Tanzania	Sub-Saharan Africa	149	3.666	3.561	3.771	0.47155	0.77623	0.35700	0.31760	0.05099	0.31472	1.37769
4	Zambia	Sub-Saharan Africa	106	4.795	4.645	4.945	0.61202	0.63760	0.23573	0.42662	0.11479	0.17866	2.58991

...

모델 학습

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()     # 모델 정의
model.fit(x_train, y_train)    # 모델 학습
pred = model.predict(x_test)   # 모델 예측
```

모델 학습

➢ 모델 정의

- 선형 회귀 학습을 진행하는 모델을 정의

```
model = LinearRegression()
```

➢ LinearRegression()의 주요 파라미터

- fit_intercept (Boolean)
  - 모델이 intercept(편향) 변수를 추가하여 학습할 것인지를 결정
  - default 값은 True

```
model = LinearRegression(fit_intercept=False)
# 파라미터 적용 예시
```

모델 학습

➢ 모델 학습 (Fit)

- Train data에서 추출된 데이터들을 모델에 제공하여 학습 진행

```
model.fit(x_train, y_train)
```

- 독립변수의 분포 x_train을 바탕으로 종속변수(행복 지수) y_train을 예측하는 학습을 진행

➢ 모델 예측 (Prediction)

- Train data로 주어지지 않은 test 데이터에 대해 학습 완료된 모델을 이용하여 예측

```
pred = model.predict(x_test)
```

정확도 평가 방법

➢ 제곱 평균 제곱근 (Root Mean Square Error, RMSE)

- 각 데이터의 오차의 제곱의 평균의 제곱근으로서, 실제 값($true$)과 추정 값($pred$)의 평균적 오차를 확인하는 데 사용하는 지표

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(pred_k - true_k)^2}$$

정확도 평가

➢ 모델 평가 (Evaluation)

- 모델이 예측한 값 pred와 실제 값인 y_test 사이의 오차를 계산

```
from sklearn.metrics import mean_squared_error
import math

rmse = math.sqrt(mean_squared_error(pred, y_test))
print("%.6f" % rmse)
```

- Result: 0.000266

모델 평가

➢ 학습된 가중치 확인

- 학습이 완료된 모델이 각 변수 별로 측정한 가중치 값과 편향 값을 확인하기

```
print("weights:", model.coef_)          # 변수 별 가중치
print("intercept:", model.intercept_) # 편향
```

```
weights: [ 0.99994962  0.99996256  0.99997291  1.0002023   0.99973521  1.00011789
  0.99995951]
intercept: 0.000113448913716
```

모델 분석

➢ 학습된 가중치 확인

- 학습된 모델로 테스트 진행 시 값을 어떻게 예측하는지 확인하기

```
example = x_test[0]
print("example:", example)
print("calculation:", np.dot(example, model.coef_) +
model.intercept_)                          # 예측값 직접 계산
print("prediction:", pred[0])              # 예측값 추출
```

```
example: [ 1.24585  1.04685  0.69058  0.4519   0.055    0.14443  2.20035]
calculation: 5.83495757186
prediction: 5.83495757186
```

- np.dot(U,V): 동일한 크기의 1차원 배열 U, V의 각 요소의 곱을 전부 더한 값
  - $\sum_{i=1}^{n} U_i V_i$ (n: 배열의 크기)

모델 평가

➢ 학습된 가중치 확인

- 학습이 완료된 모델이 각 변수 별로 측정한 가중치 값과 편향 값을 확인하기

	Economy	Family	Health	Freedom	Trust	Generosity	Dystopia
테스트 케이스 x	1.24585	1.04685	0.69058	0.4519	0.055	0.14443	2.20035
				×			
가중치 w	0.999950	0.999963	0.999973	1.000202	0.999735	1.000118	0.999960
				=			
wx	1.245787	1.046811	0.690561	0.451991	0.054985	0.144447	2.2200261
				+			
편향 b	0.000113						
				=			
최종 예측값	5.834958						

모델 분석

➢ 독립변수 선택

- 입력으로 사용할 독립변수의 종류 및 수를 조절하여 성능 변화를 관찰
  - 단일 변수를 선택 시, 각 변수 별 영향력의 차이를 확인 가능

```
features = ['Economy']
```

Case	Feature	Result: RMSE
#1	Economy	0.6835
#2	Family	0.8284
#3	Health	0.6918
#4	Freedom	0.8391
#5	Trust	0.8967
#6	Generosity	1.0045
#7	Dystopia	0.8789

모델 분석

➢ 독립변수 선택

- 입력으로 사용할 변수의 종류 및 수를 조절하여 성능 변화를 관찰
  - 복수 변수를 선택 시, 예측에 도움이 되는 변수의 조합을 판단 가능

```
features = ['Economy', 'Health']
```

Case	Feature	Result: RMSE
#1	Economy, Health	0.6269
#2	Freedom, Dystopia	0.7594
#3	Freedom, Trust, Generosity, Dystopia	0.6658
#4	Economy, Family, Generosity, Dystopia	0.1783

모델 분석

➢ 그래프 확인

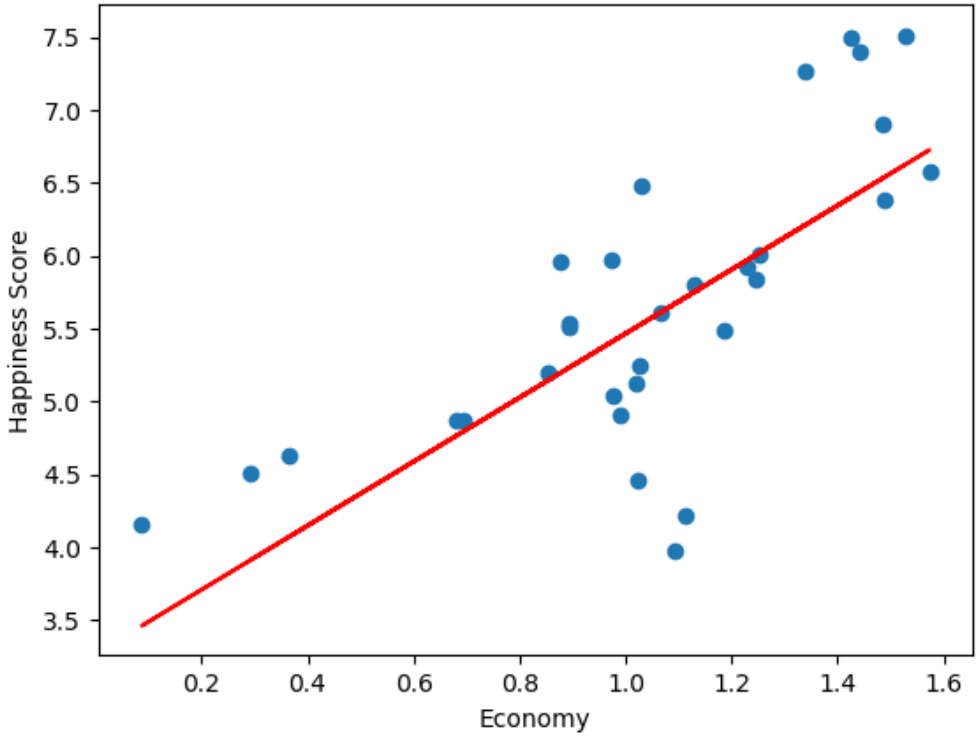
- 특정 1개의 변수로 학습 시, 모델이 예측한 1차함수를 그래프로 그려 확인

```
import matplotlib.pyplot as plt

features = ['Economy']                    # 단일 변수
...                                       # 모델 학습 및 예측

plt.scatter(x_test, y_test)               # 실제 데이터 산포도
plt.plot(x_test, pred, color='red')       # 예측 데이터 선형그래프
plt.xlabel('Economy')                     # x축 명
plt.ylabel('Happiness Score')             # y축 명
plt.show()                                # 그래프 출력
```

모델 분석

➢ 그래프 확인

- 특정 1개의 변수로 학습 시, 모델이 예측한 1차함수를 그래프로 그려 확인



실습1: 도요타 자동차 가격 예측

- 자동차들의 각 주어진 특징을 바탕으로 자동차의 가격을 예측하는 선형 회귀 모델을 작성

	A	B	C	D	E	F	G	H	I	J	K	L
1	Id	Model	Price	Age_08_04	Mfg_Mont	Mfg_Year	KM	Fuel_Type	HP	Met_Color	Automatic	cc
2	914	TOYOTA C	8950	60	9	1999	58269	Petrol	110	1	0	1600
3	1135	TOYOTA C	7750	71	10	1998	107516	Petrol	110	1	0	1600
4	1313	TOYOTA C	7500	76	5	1998	70039	Petrol	110	1	0	1600
5	1379	TOYOTA C	6750	76	5	1998	57263	Petrol	110	1	0	1600
6	1380	TOYOTA C	7950	75	6	1998	57144	Petrol	110	1	0	1600
7	610	TOYOTA C	7500	59	10	1999	190900	Diesel	72	1	0	2000
8	1257	TOYOTA C	8950	77	4	1998	78435	Petrol	86	0	0	1300
9	447	TOYOTA C	11750	48	9	2000	75045	Petrol	110	1	0	1600
10	573	TOYOTA C	13000	49	8	2000	36000	Petrol	110	0	0	1600
11	1397	TOYOTA C	8500	73	8	1998	52000	Petrol	110	1	1	1600
12	1004	TOYOTA C	10950	57	12	1999	40214	Petrol	86	0	0	1300
13	922	TOYOTA C	8950	65	4	1999	57374	Petrol	110	1	0	1600
14	1334	TOYOTA C	8950	78	3	1998	65500	Petrol	86	1	0	1300
15	924	TOYOTA C	9995	61	8	1999	57169	Petrol	110	1	0	1600
16	1069	TOYOTA C	5740	74	7	1998	159908	Petrol	110	0	0	1600
17	1076	TOYOTA C	7750	70	11	1998	151300	Diesel	72	1	0	2000
18	631	TOYOTA C	8950	65	4	1999	132807	Diesel	72	1	0	2000
19	517	TOYOTA C	10500	56	1	2000	52448	Petrol	97	0	0	1400
20	467	TOYOTA C	8750	49	8	2000	68565	Petrol	110	1	0	1600
21	1152	TOYOTA C	6900	74	7	1998	101773	Petrol	110	0	0	1600

...

...

실습1: 도요타 자동차 가격 예측

➢ 참고 사항

- DataFrame에서 열(column) 명을 구체적으로 언급하는 대신 번호로 연속적인 범위를 지정함으로써 데이터 호출 가능

```
x_train = np.array(train_input.iloc[:, 3:])
```

- 변수 중 'Fuel_Type' 변수(H열)는 정수가 아닌 문자열이다. 이 변수는 'Diesel' 또는 'Petrol'을 값으로 포함하고 있으며, 이를 0과 1의 정수 값으로 치환한 후 모델을 학습시켜야 한다.

실습2: 집 가격 예측

- 집들의 각 주어진 특징을 바탕으로 집의 가격을 예측하는 선형 회귀 모델을 작성

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	date	price	bedrooms	bathroom	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
2	3.65E+09	201405141	543000	3	2.5	2090	7640	1	0	0	3	7
3	7.93E+08	201412181	360000	3	1.25	2350	6200	1	0	0	4	7
4	2.47E+09	201409041	350000	3	2.25	2470	10290	2	0	0	3	8
5	7.15E+09	201502111	235000	4	1.75	1720	10137	2	0	0	3	7
6	8.86E+09	201406261	350000	3	2.25	1780	16290	2	0	0	4	8
7	6.84E+09	201408291	402000	3	2.5	1520	3425	2	0	0	3	7
8	2.04E+08	201503241	595000	3	2.75	2150	31238	2	0	0	3	9
9	6.79E+09	201407141	476000	3	1.75	1650	9600	1	0	0	4	7
10	1.33E+09	201504271	864000	4	2.5	3720	105850	2	0	0	4	10
11	3.24E+09	201408221	600000	3	2.5	4930	77536	2	0	0	3	9
12	9.24E+09	201503191	699000	5	2.75	2970	36817	2	0	0	4	8
13	4.06E+09	201410071	750000	2	2	2180	21392	2	0	0	3	8
14	9.25E+08	201408191	604950	3	2.5	2110	5608	1	0	0	3	8
15	9.23E+09	201501231	418500	2	1	790	5800	1	0	0	3	6
16	9.2E+09	201405071	618080	3	2.5	2030	6500	2	0	0	3	8
17	9.55E+09	201504141	750000	3	2.5	2560	9182	2	0	0	3	9
18	3.4E+09	201407111	250000	2	1	1030	8786	1	0	0	3	6
19	1.1E+09	201410261	755000	3	2.5	2420	8856	1	0	3	3	9
20	2.02E+09	201504281	978000	4	2.75	2890	7821	2	0	0	3	9
21	8.86E+09	201405211	332500	3	2.25	1800	10500	2	0	0	3	7

...

...

로지스틱 회귀(Logistic Regression) 응용

WineQuality 데이터 소개

- **각 와인의 성분 및 그에 따른 등급 평가에 대한 정보를 포함**
  - 데이터 수 : 총 1599개
  - 변수(features)
    - 독립 변수: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol
    - 종속 변수: quality (등급)
      - 값: [3, 4, 5, 6, 7, 8]

- **실습 목표**
  - 각 와인 별로 관찰된 성분을 바탕으로 와인의 등급을 분류하는 로지스틱 회귀 모델 구현

데이터 읽어 오기

```
import pandas as pd
import numpy as np

train_input = pd.read_csv("WineQuality_train.csv")
test_input = pd.read_csv("WineQuality_test.csv")

x_train = np.array(train_input.iloc[:, 0:11])
y_train = np.array(train_input['quality'])
x_test = np.array(test_input.iloc[:, 0:11])
y_test = np.array(test_input['quality'])
```

데이터 읽어 오기

➢ DataFrame 파싱

- DataFrame에서 열(column) 명을 구체적으로 언급하는 대신 번호로 연속적인 범위를 지정함으로써 데이터 호출 가능

```
x_train = np.array(train_input.iloc[:, 0:11])
```

- iloc[]: DataFrame 내 특정 위치(범위)의 데이터를 찾는 함수

	0	1	2	3	4	5	6	7	8	9	10	11
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	6.0	0.33	0.32	12.9	0.054	6.0	113.0	0.99572	3.30	0.56	11.5	4
1	9.9	0.25	0.46	1.7	0.062	26.0	42.0	0.99590	3.18	0.83	10.6	6
2	9.3	0.48	0.29	2.1	0.127	6.0	16.0	0.99680	3.22	0.72	11.2	5
3	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.99460	3.39	0.47	10.0	7
4	7.2	0.63	0.00	1.9	0.097	14.0	38.0	0.99675	3.37	0.58	9.0	6
...												

모델 학습

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(C=1000.0, solver='newton-cg',
multi_class='multinomial', max_iter=500)   # 모델 정의
model.fit(x_train, y_train)                # 모델 학습
pred = model.predict(x_test)               # 모델 예측
```

모델 학습

➢ 모델 정의

- 로지스틱 회귀 학습을 진행하는 모델을 정의

```
model = LogisticRegression()
```

➢ LogisticRegression()의 주요 파라미터

- solver: {'liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga'}
  - 학습에 사용할 알고리즘 선택
  - default 값은 'liblinear'이며, 주로 클래스가 2개인 이분법 학습에 사용됨
  - 클래스가 3개 이상인 다중클래스 학습을 진행하려면 다른 알고리즘을 선택하는 것을 권장
  - 참고자료: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

모델 학습

➢ LogisticRegression()의 주요 파라미터

- C (Float)
  - 모델의 정규화(regularization) 비중의 역수 값으로, 0에 가깝게 작을수록 정규화의 비중이 커짐
  - default 값은 1.0
- multi_class: {'ovr', 'multinomial'}
  - 'ovr': default 값, 클래스가 2개인 이분법 학습을 적용
  - 'multinomial': 클래스가 3개 이상인 다중클래스 학습을 적용
    - solver를 'liblinear'로 설정했다면, multi_class를 'multinomial'로 설정할 수 없음
- max_iter (int)
  - 모델의 최대 학습 회수에 대한 값으로, default 값은 100
  - 만약 모델이 주어진 max_iter 회수 내에 학습을 완료하지 못한다면, 이 값을 증가시켜줘야 함

```
model = LogisticRegression(C=1000.0, solver='newton-cg',
multi_class='multinomial', max_iter=500)
# 파라미터 적용 예시
```

모델 학습

➢ 모델 학습 (Fit)

- Train data에서 추출된 데이터들을 모델에 제공하여 학습 진행

```
model.fit(x_train, y_train)
```

- 모델은 x_train으로 제공된 와인 별 독립변수의 분포를 바탕으로 y_train으로 제공되는 종속변수(와인 등급)를 예측하는 학습을 진행

➢ 모델 예측 (Prediction)

- Train data로 주어지지 않은 test 데이터에 대해 학습 완료된 모델을 이용하여 예측

```
pred = model.predict(x_test)
```

정확도 평가 방법

➢ Accuracy_score

- 각 주어진 실제 클래스에 대해 모델이 성공적인 클래스 예측을 진행한 데이터의 비율을 평가

```
from sklearn.metrics import accuracy_score

p = [0, 2, 2, 1, 1, 2]
t = [0, 1, 2, 2, 1, 2]
print("%.6f" % accuracy_score(y_true=t,
y_pred=p))
```

예측 (pred)	실제 (true)	정답 여부
0	0	O
2	1	X
2	2	O
1	2	X
1	1	O
2	2	O

- Result: 0.666667

정확도 평가

➢ 모델 평가 (Evaluation)

- 모델이 예측한 값 pred와 실제 값인 y_test 사이의 오차를 계산

```
from sklearn.metrics import accuracy_score

print("%.6f" % accuracy_score(y_true=y_test,
y_pred=pred))
```

- Result: 0.626959

모델 평가

➢ 파라미터 변형

- LogisicRegression()의 파라미터를 다양하게 적용시켜 성능 변화를 관찰
  - 명시되지 않은 파라미터들은 각 파라미터 별 default 값으로 설정됨
    - LogisicRegression(): 모든 파라미터를 default로 설정

Case	모델 정의	Result: Accuracy
#1	LogisticRegression()	0.5925
#2	LogisticRegression(C=1000.0)	0.6144
#3	LogisticRegression(solver='newton-cg', multi_class='multinomial')	0.6113
#4	LogisticRegression(solver='lbfgs', multi_class='multinomial')	0.6019
#5	LogisticRegression(solver='saga', multi_class='multinomial', max_iter=5000)	0.6082

실습: 대출 가능 여부 판단

- 고객들의 각 주어진 특징을 바탕으로 해당 고객이 대출 가능한 고객인지 판단하는 로지스틱 회귀 모델을 작성

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal L	Securities	CD Accou	Online	CreditCard
2	2470	43	18	89	92780	1	0.1	2	307	0	0	0	0	1
3	968	55	30	73	92675	4	3.8	2	0	0	0	0	1	0
4	729	45	20	114	94720	2	4.4	2	0	1	0	0	0	0
5	327	52	27	80	95616	1	1.3	3	0	0	0	1	1	1
6	679	52	27	61	92101	4	1.8	3	207	0	0	0	0	0
7	1197	37	13	71	94609	2	2.7	1	94	0	0	0	1	0
8	1396	47	23	190	92831	4	0.3	3	305	1	0	0	0	0
9	1288	42	18	54	94010	4	2.2	2	0	0	0	0	0	0
10	2276	40	16	115	94305	1	3.4	1	0	0	0	0	1	0
11	513	39	14	54	95035	3	3	1	108	0	0	0	0	1
12	840	39	15	79	92646	4	2.4	1	0	0	0	0	0	0
13	1650	29	4	73	95039	1	0.8	2	0	0	0	0	1	0
14	197	48	24	165	93407	1	5	1	0	0	0	0	0	1
15	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
16	31	59	35	35	93106	1	1.2	3	122	0	0	0	1	0
17	2197	51	24	189	95211	4	4.75	2	0	1	0	0	1	0
18	682	34	9	164	94720	1	6	3	0	1	0	0	1	0
19	2379	30	5	61	95605	1	0.8	2	251	0	1	1	1	0
20	490	53	28	43	91380	2	2.1	3	0	0	0	1	1	1
21	463	29	4	183	91423	3	8.3	3	0	1	0	0	1	0

...

실습: 대출 가능 여부 판단

➢ 참고 사항

- 대출 가능 여부를 표시해주는 'Personal Loan'은 0 또는 1로 이루어진 이분법 클래스이다.
- 타겟인 'Personal Loan' 열이 표의 처음이나 끝이 아닌 중간(J열)에 존재하다. 추출해야 하는 독립 변수가 한 범위에 있지 않고 분할 되어있다.
- 추출하고자 하는 변수의 열 번호의 목록을 배열로 작성함으로써, 연속적이지 않은 범위의 변수를 호출할 수 있다.

```
x_train = np.array(train_input.iloc[:, [1, 2, 3, 4, 5, 6,
7, 8, 10, 11, 12, 13]])
```

결정 트리(Decision Tree) 응용

모델 학습

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(max_depth=30,
max_features=6, random_state=0)              # 모델 정의
model.fit(x_train, y_train)                  # 모델 학습
pred = model.predict(x_test)                 # 모델 예측
```

모델 학습

➢ 모델 정의

- 결정 트리 학습을 진행하는 모델을 정의

```
model = DecisionTreeClassifier()
```

➢ DecisionTreeClassifier()의 주요 파라미터

- max_depth (int or None)
  - 트리의 최대 깊이
  - default 값은 None이며, 이 경우에는 트리 생성에 깊이 제한이 없음
- max_features (int or None)
  - 한번의 트리 분할에 고려하는 최대 변수(feature) 수
  - default 값은 None이며, 이 경우에는 입력된 변수의 가짓수와 동일

모델 학습

➢ DecisionTreeClassifier()의 주요 파라미터

- max_leaf_nodes (int or None)
  - 트리 생성 시 잎 노드 수의 개수 제한
  - default 값은 None이며, 이 경우에는 트리 생성에 잎 노드 제한이 없음
- random_state (int or None)
  - 무작위 난수 생성에 사용할 seed
  - default 값은 None이며, 트리 생성을 매번 무작위로 진행
  - 특정 정수 값(0 등)을 입력할 경우, 같은 값을 입력한 경우에 대해 반복적인 테스트를 진행해도 동일한 결과를 얻을 수 있음

```
model = DecisionTreeClassifier(max_depth=30,
max_features=6, random_state=0)    # 파라미터 적용 예시
```

정확도 평가

➢ 모델 평가 (Evaluation)

- 모델이 예측한 값 pred와 실제 값인 y_test 사이의 오차를 계산

```
from sklearn.metrics import accuracy_score

print("%.6f" % accuracy_score(y_true=y_test,
y_pred=pred))
```

- Result: 0.658307

모델 평가

➢ 파라미터 변형

- DecisionTreeClassifier()의 파라미터를 다양하게 적용시켜 성능 변화를 관찰

Case	모델 정의	Result: Accuracy
#1	DecisionTreeClassifier(random_state=0)	0.6426
#2	DecisionTreeClassifier(max_depth=20, random_state=0)	0.6395
#3	DecisionTreeClassifier(max_features=6, random_state=0)	0.6583
#4	DecisionTreeClassifier(max_leaf_nodes=200, random_state=0)	0.6614
#5	DecisionTreeClassifier(max_depth=20, max_leaf_nodes=500, random_state=0)	0.6677

모델 분석

➢ 생성된 트리를 그래프로 관찰

- graphviz 설치

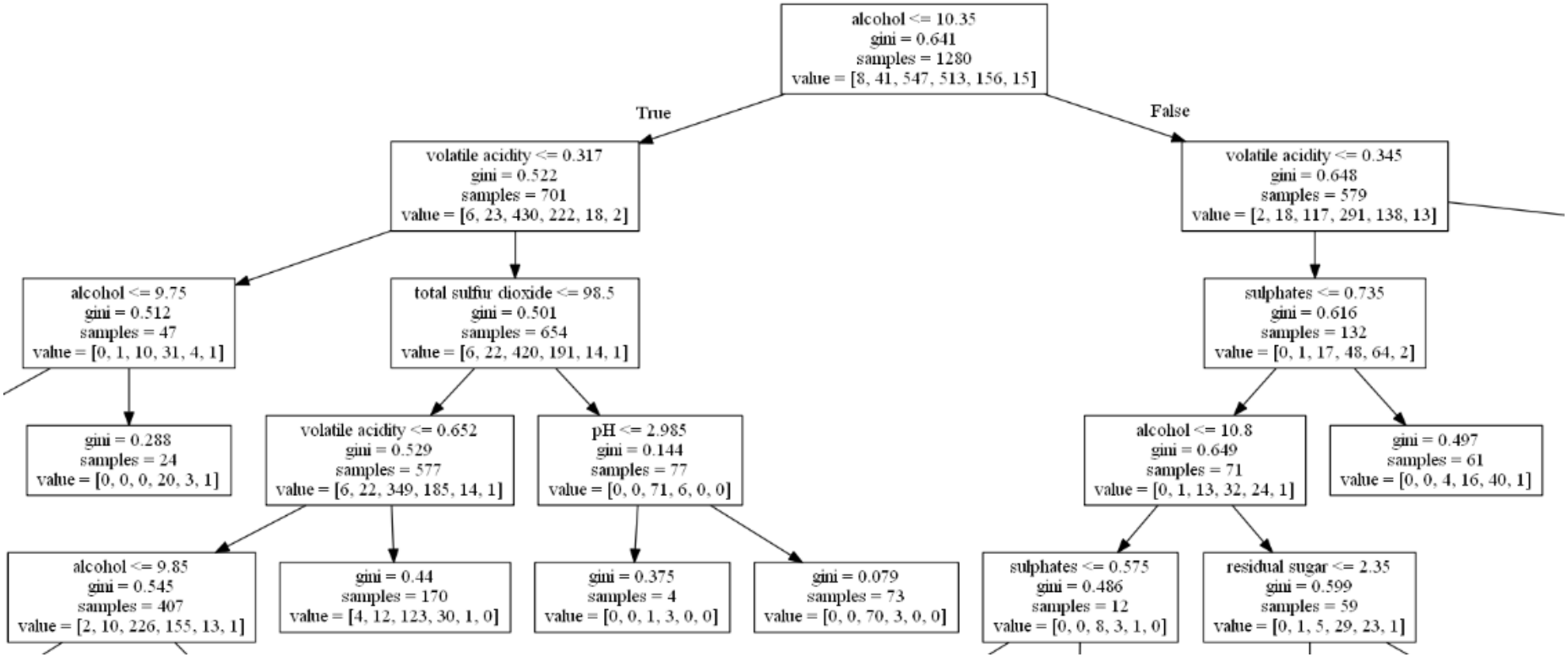
```
pip install python-graphviz
```

```
from sklearn.tree import export_graphviz
import graphviz

model = DecisionTreeClassifier(max_depth=10,
max_leaf_nodes=50, random_state=0)       # accuracy = 0.5925
model.fit(x_train, y_train)

dot_data = export_graphviz(model, out_file=None,
feature_names=train_input.columns[0:11]) # dot 데이터 생성
graph = graphviz.Source(dot_data)        # 그래프 데이터 생성
graph.format = 'png'           # 데이터 저장 포맷을 png로 지정
graph.render(filename='tree') # 그래프 데이터 저장
```

모델 분석

➢ 생성된 트리를 그래프로 관찰

- tree.png



...

실습: 스팸메일 판단

- 각 메일의 단어 분포를 바탕으로 해당 메일이 스팸 메일인지 아닌지를 판단하는 결정 트리 모델을 작성

	A	B	C	D	E	F	G	H	I	...	BF
1	word_freq.	word_freq.	word_freq.	word_freq.	word_freq.	word_freq.	word_freq.	word_freq.	word_freq.		spam
2	0	0	0.45	0	0.22	0.22	0	0	0.67		1
3	0	0	0	0	0	0	0	0	0		0
4	0	0	0	0	0	0	0	0	0		0
5	0	0.19	0.59	0	0.19	0	0	0	0		0
6	0	0	0.32	0	0.64	0.64	0.64	0.32	0.32		1
7	0.56	0	0.32	0	1.13	0.08	0	0	0.16		1
8	0	0.53	0.53	0	0.53	0	0	0	0		1
9	0	0	0	0	0	0	0	0	0		0
10	0	0	0	0	0	0	0	0	0.75		0
11	0	0	0	0	0	0	1.25	0	0		1
12	0.72	0	0	0	0	0	0	0	0		0
13	0	0	0.06	0	0	0	0	0.06	0.13		0
14	0	0	0	0	0	0	0	0	0		0
15	0.1	0.72	0.62	0	0.62	0.1	0.2	0.2	0		0
16	0	0	0	0	0	0.57	0	0	0		0
17	0	0	0	0	0	0	0	0	0		0
18	0	0	1.2	0	0	0	0	0	0		0
19	0	0	0.68	0	0	0	0	1.36	0		1
20	0.2	0.1	0.7	0	1.1	0.2	0	0.3	0		1
21	0.02	0.02	0	0	0	0	0	0	0.05		0

...

실습: 스팸메일 판단

➢ 참고 사항

- 각 데이터 당 58개의 변수가 존재하는 방대한 데이터이다. 마지막 변수인 'spam'을 제외하고는 모두 독립변수인 점을 감안하여, 'spam'을 제외한 모든 열을 검색하는 코드를 작성하는 것이 더 간편하다.

```
x_train = np.array(train_input.loc[:,
train_input.columns != 'spam']])
```

- loc[]: DataFrame 내 특정 열(column) 이름을 가진 데이터를 찾는 함수
  - iloc[]은 열 번호를 입력한다는 점에서 차이를 보임

나이브 베이지안 분류 응용
(Naive Bayesian Classification)

모델 학습

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()                          # 모델 정의
model.fit(x_train, y_train)                   # 모델 학습
pred = model.predict(x_test)                  # 모델 예측
```

모델 학습

➢ 모델 정의

- 나이브 베이지안 분류 학습을 진행하는 모델을 정의
- 확률 계산 알고리즘에 따라 GaussianNB, MultinomialNB, BernoulliNB가 존재

```
from sklearn.naive_bayes import GaussianNB,
MultinomialNB, BernoulliNB

model1 = GaussianNB()
model2 = MultinomialNB()
model3 = BernoulliNB()
```

모델 학습

➢ 모델 정의

- 나이브 베이지안 분류는 클래스(y) 별 각 독립변수 $x_i (i = 1 \dots n)$의 발현 비율 $P(x_i|y)$를 구하는 것을 목적으로 한다.

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(y) \cdot P(x_1|y) \cdot P(x_2|y) \cdot \cdots \cdot P(x_n|y)}{P(x_1, x_2, \dots, x_n)}$$

- GaussianNB()
  - 주어진 변수가 정규분포(평균 $\mu_{i,y}$, 표준편차 $\sigma_{i,y}$)를 이룬다고 가정하며, 연속적인 값을 지닌 독립변수를 처리할 때 사용
  - $P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_{i,y}^2}} \exp\left(-\frac{(x_i - \mu_{i,y})^2}{2\sigma_{i,y}^2}\right)$

모델 학습

➢ 모델 정의

- MultinomialNB()
  - 전체 데이터 대비 단순 출현 비율을 계산하는 모델이며, 이산적인 값을 지닌 독립변수를 처리할 때 사용
  - $P(x_i|y) = \frac{x_i \text{ 값이 존재하는 데이터 수}}{\text{전체 데이터 수}}$
- BernoulliNB()
  - 주어진 변수가 베르누이분포(1일 확률 $p_{i,y}$)를 이룬다고 가정하며, 값이 0 혹은 1만 존재하는 이분법 독립변수를 처리할 때 사용
  - $P(x_i|y) = \begin{cases} p_{i,y} & (x_i = 1) \\ 1 - p_{i,y} & (x_i = 0) \end{cases}$
  - 입력되는 변수가 이분법 데이터가 아닐 경우, 내부 파라미터 설정에 따라 강제로 이분법 데이터로 치환된다.
- 참고자료: http://scikit-learn.org/stable/modules/naive_bayes.html

정확도 평가

➢ 모델 평가 (Evaluation)

- 모델이 예측한 값 pred와 실제 값인 y_test 사이의 오차를 계산

```
from sklearn.metrics import accuracy_score

print("%.6f" % accuracy_score(y_true=y_test,
y_pred=pred))
```

Model	Accuracy Score
GaussianNB()	0.5737
MultinomialNB()	0.4859
BernoulliNB()	0.4201

모델 분석

➢ 확률 예측값 출력

- 주어진 변수에 따라, 각 클래스 별 확률이 어떻게 분포되는지 관찰
- 예제: x_test[0]

fixed acidity	7.50000
volatile acidity	0.58000
citric acid	0.20000
residual sugar	2.00000
chlorides	0.07300
free sulfur dioxide	34.00000
total sulfur dioxide	44.00000
density	0.99494
pH	3.10000
sulphates	0.43000
alcohol	9.30000
quality	5.00000

```
print(model.predict_proba(x_test[0].
reshape(1,-1)))
```

	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8
확률	4.2e-08	3.2e-03	**7.3e-01**	2.6e-01	1.2e-03	1.8e-04

- predict_proba()에 들어가는 변수는 반드시 2차원 배열이어야 한다. 1차원 배열을 사용하고 싶을 경우, reshape(1,-1)을 사용함으로써 2차원 배열로 만들 수 있다.

실습: 문장 긍정/부정 분석

- 각 문장의 단어 분포를 바탕으로 해당 문장이 긍정 문장인지 부정 문장인지 분석하는 나이브 베이지안 모델을 작성

	A	B
1	sentence	label
2	don't waste your time here	0
3	no i'm going to eat the potato that i found some strangers hair in it	0
4	and considering the two of us left there very full and happy for about $20 you just can't go wrong	1
5	an extensive menu provides lots of options for breakfast	1
6	the jalapeno bacon is soooo good	1
7	i just wanted to leave	0
8	all in all ha long bay was a bit of a flop	0
9	we waited for forty five minutes in vain	0
10	the food is very good for your typical bar food	1
11	i'll definitely be in soon again	1
12	this place is good	1
13	did not like at all	0
14	5 stars for the brick oven bread app	1
15	always a great time at dos gringos	1
16	i do love sushi but i found kabuki to be over priced over hip and under services	0
17	the block was amazing	1
18	worst thai ever	0
19	the vegetables are so fresh and the sauce feels like authentic thai	1
20	thoroughly disappointed	0
21	we had fantastic service and were pleased by the atmosphere	1

...

실습: 문장 긍정/부정 분석

➢ 참고 사항

- 문장(string)으로 주어진 데이터를 분석하기 위해 tf-idf 분류를 선행하는 것을 권장한다.

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

train_input = pd.read_csv("SentimentSentence_train.csv")
test_input = pd.read_csv("SentimentSentence_test.csv")

x_train_input = np.array(train_input['sentence'])
x_test_input = np.array(test_input['sentence'])

tfidf = TfidfVectorizer()                    # Tf-idf 계산 함수
tfidf.fit(x_train_input)
x_train = tfidf.transform(x_train_input).toarray()
x_test = tfidf.transform(x_test_input).toarray()
```

평가 방법(Evaluation Metrics)

정밀도(Precision)

```
from sklearn.metrics import precision_score

model = ...
model.fit(x_train, y_train)
pred = model.predict(x_test)

print(precision_score(y_true=y_test, y_pred=pred,
average=None))
```

정밀도(Precision)

➢ 정밀도 측정 예시

pred	0	0	1	1	1	2	2	2	2	2
true	0	2	1	1	0	0	0	1	1	2

- average='binary'
  - 클래스가 2개일 때 이분법으로 정밀도를 계산
  - default 값이기 때문에, 다중클래스 문제를 다룰 경우 이 부분을 생략하면 안 된다.
- average=None
  - 각 클래스 별 정밀도의 계산 결과를 반환
  - Result: [0.5 0.6667 0.2]
    - 0으로 예측한 데이터 중 50%가 정답
    - 1로 예측한 데이터 중 66.7%가 정답
    - 2로 예측한 데이터 중 20%가 정답

정밀도(Precision)

➢ 정밀도 측정 예시

pred	0	0	1	1	1	2	2	2	2	2
true	0	2	1	1	0	0	0	1	1	2

- average='micro'
  - 클래스 구분 없이 모든 데이터에 대해 정밀도를 반환
  - Result: 0.4
    - 전체 10개 데이터 중 40%가 정답
  - accuracy_score()와 동일한 기능을 수행함
- average='macro'
  - 각 클래스 별 정밀도의 평균을 반환
  - Result: 0.4556
    - 각 클래스에 대해 [0.5 0.6667 0.2] 의 정밀도가 관찰되었을 때, 이들의 평균값은 0.4556

정밀도(Precision)

➢ 각 모델 별 정밀도 측정

- WineQuality 데이터를 학습한 각 모델에 대해 각 클래스 별 정밀도 측정
  - average=None: 각 클래스 별 정밀도
  - average='macro': 평균 정밀도

Model	Accuracy Score	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	평균 정밀도
LogisticRegression (solver='newton-cg', multiclass='multinomial')	0.6113	0	0	0.6772	0.5548	0.4667	0	0.2682
LogisticRegression (solver='newton-cg', multiclass='multinomial', class_weight='balanced')	0.4577	0.05	0.1515	0.6780	0.5652	0.3462	0.1111	0.3170
DecisionTreeClassifier (max_depth=30, max_features=6, random_state=0)	0.6583	0	0.1429	0.7113	0.6585	0.6047	0.3333	0.4084
GaussianNB()	0.5737	0	0.3	0.6746	0.5390	0.5429	0	0.3427

재현율(Recall)

```
from sklearn.metrics import recall_score

model = ...
model.fit(x_train, y_train)
pred = model.predict(x_test)

print(recall_score(y_true=y_test, y_pred=pred, average=None))
```

재현율(Recall)

➢ 재현율 측정 예시

pred	0	0	1	1	1	2	2	2	2	2
true	0	2	1	1	0	0	0	1	1	2

- average='binary'
  - 클래스가 2개일 때 이분법으로 재현율 계산
  - default 값이기 때문에, 다중클래스 문제를 다룰 경우 이 부분을 생략하면 안 된다.
- average=None
  - 각 클래스 별 재현율의 계산 결과를 반환
  - Result: [0.25 0.5 0.5]
    - 0 클래스의 데이터 중 25%를 정확히 예측함
    - 1 클래스의 데이터 중 50%를 정확히 예측함
    - 2 클래스의 데이터 중 50%를 정확히 예측함

재현율(Recall)

➢ 재현율 측정 예시

pred	0	0	1	1	1	2	2	2	2	2
true	0	2	1	1	0	0	0	1	1	2

- average='micro'
  - 클래스 구분 없이 모든 데이터에 대해 재현율을 반환
  - Result: 0.4
    - 전체 10개 데이터 중 40%를 정확히 예측
  - accuracy_score()와 동일한 기능을 수행함
- average='macro'
  - 각 클래스 별 재현율의 평균을 반환
  - Result: 0.4167
    - 각 클래스에 대해 [0.25 0.5 0.5] 의 재현율이 관찰되었을 때, 이들의 평균값은 0.4167

재현율(Recall)

➢ 각 모델 별 재현율 측정

- WineQuality 데이터를 학습한 각 모델에 대해 각 클래스 별 재현율 측정
  - average=None: 각 클래스 별 재현율
  - average='macro': 평균 재현율

Model	Accuracy Score	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	평균 재현율
LogisticRegression (solver='newton-cg', multiclass='multinomial')	0.6113	0	0	0.7985	0.6480	0.1628	0	0.2682
LogisticRegression (solver='newton-cg', multiclass='multinomial', class_weight='balanced')	0.4577	0.5	0.4167	0.5970	0.3120	0.4186	1	0.5407
DecisionTreeClassifier (max_depth=30, max_features=6, random_state=0)	0.6583	0	0.0833	0.7537	0.6480	0.6047	0.3333	0.4038
GaussianNB()	0.5737	0	0.25	0.6343	0.6080	0.4419	0	0.3224

F1 점수(F1 Score)

```
from sklearn.metrics import f1_score

model = ...
model.fit(x_train, y_train)
pred = model.predict(x_test)

print(f1_score(y_true=y_test, y_pred=pred, average=None))
```

- average 파라미터 옵션은 앞의 정밀도 함수, 재현율 함수와 동일하다.

F1 점수(F1 Score)

➢ 각 모델 별 F1 점수 측정

- WineQuality 데이터를 학습한 각 모델에 대해 각 클래스 별 F1 점수를 측정
  - average=None: 각 클래스 별 F1 점수
  - average='macro': 평균 F1 점수

Model	Accuracy Score	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	평균 F1
LogisticRegression (solver='newton-cg', multiclass='multinomial')	0.6113	0	0	0.7329	0.5978	0.2414	0	0.2620
LogisticRegression (solver='newton-cg', multiclass='multinomial', class_weight='balanced')	0.4577	0.0909	0.2222	0.6349	0.4021	0.3789	0.2	0.3215
DecisionTreeClassifier (max_depth=30, max_features=6, random_state=0)	0.6583	0	0.1053	0.7319	0.6532	0.6047	0.3333	0.4047
GaussianNB()	0.5737	0	0.2727	0.6538	0.5714	0.4872	0	0.3309

실습: 각 데이터 분석

- 앞서 실습한 3개의 데이터(대출 가능 여부 데이터, 스팸 메일 데이터, 문장 긍정/부정 데이터)를 분류하는 각 모델을 작성
- 각 모델의 정밀도, 재현율, F1 점수를 비교분석


it's
Q&A
TIME!