R과 Python기반 머신러닝과 딥러닝 데이터 분석 모델 비교

2022.12.21

A5팀(임성구, 양정연, 김용현, 이해동)

목차

1. 머신러닝과 딥러닝의 비교

1.1 개념

머신러닝(Machine Learning)은 인공지능을 만들기 위해 기계를 학습시키는 다양한 방법에 대한 학문으로 '로봇공학', '제어계측공학'과 같이 하나의 학문이다.

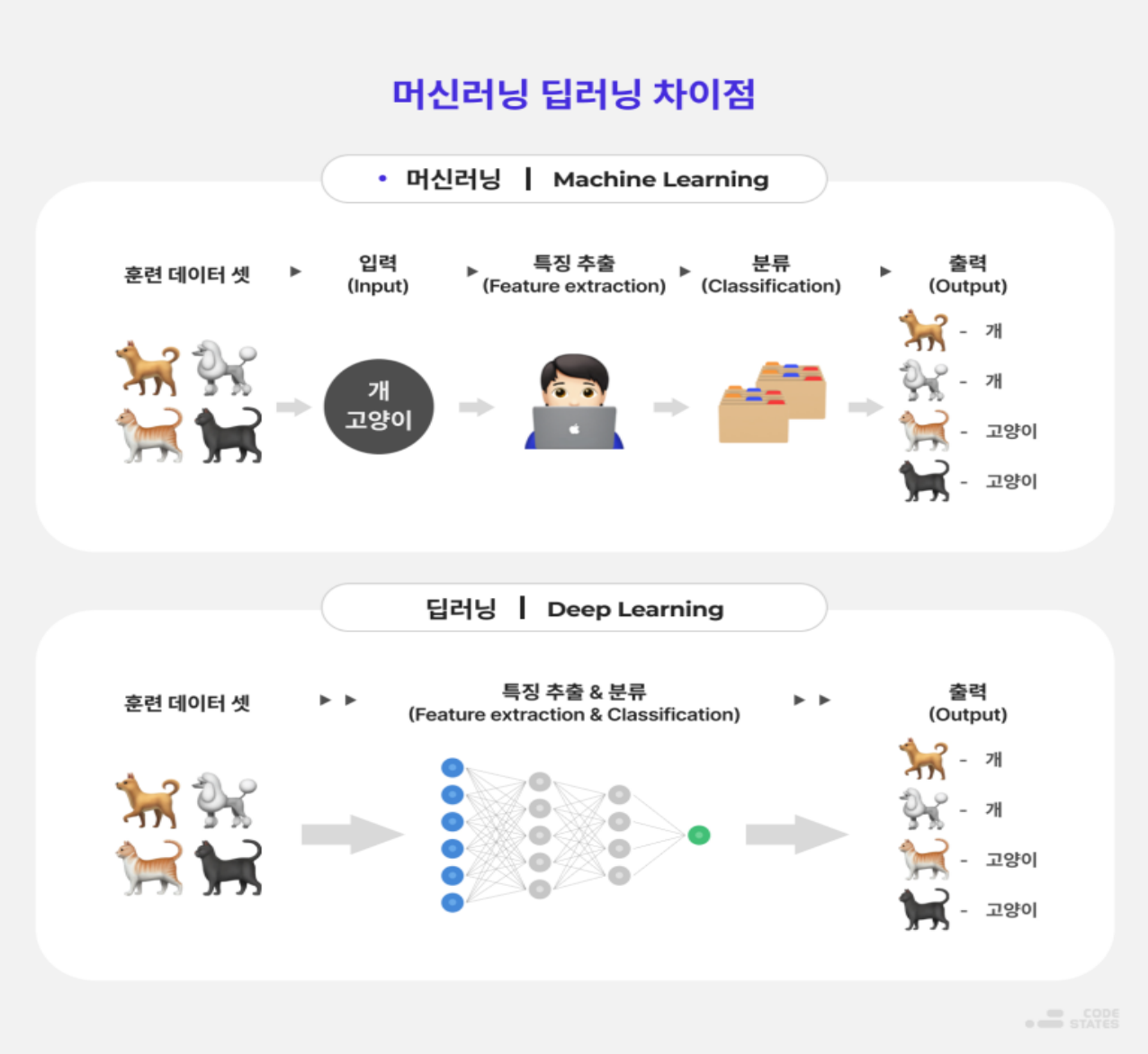
딥러닝(Deep Learning)이란 머신러닝의 한 종류이며 '신경망'을 여러 층 쌓아서 인공지능을 만드는 것이다. '층'이 깊다(Deep)고 해서, '심층 학습, 깊은 학습'으로 불리는 학습 방법이다. 함수의 합성처럼, 동물의 신경세포들의 합성인 '신경망(Neural Network)'을 따라 만든 '인공신경망(Artificial Neural Network)'에 여러 계층을 쌓아서 만든 깊은 신경망(Deep Neural Network), 즉 '딥러닝'이 만들어졌다.

또한 인공지능(Artificial intelligence)은 인간의 학습 능력과 추론 능력, 지각 능력 등을 컴퓨터 프로그램으로 실현한 기술을 말하는데, 그 연구 분야 중 하나가 바로 머신러닝이다. 위 그림처럼 딥러닝은 머신러닝에 속하며, 머신러닝은 인공지능에 속한다.

1.2 머신러닝과 딥러닝의 차이점

머신러닝은 주어진 데이터를 인간이 먼저 처리한다. 사람이 먼저 컴퓨터에 특정 패턴을 추출하는 방법을 지시하고, 그 이후 컴퓨터가 스스로 데이터의 특징을 분석하고 축적한다. 이렇게 축적된 데이터를 바탕으로 문제를 해결하도록 한다. 예를 들어 사람이 먼저 개와 고양이의 사진을 보고 개와 고양이의 특징을 추출한 후 많은 예시를 통해 컴퓨터를 학습시키고 식별하게 만든다.

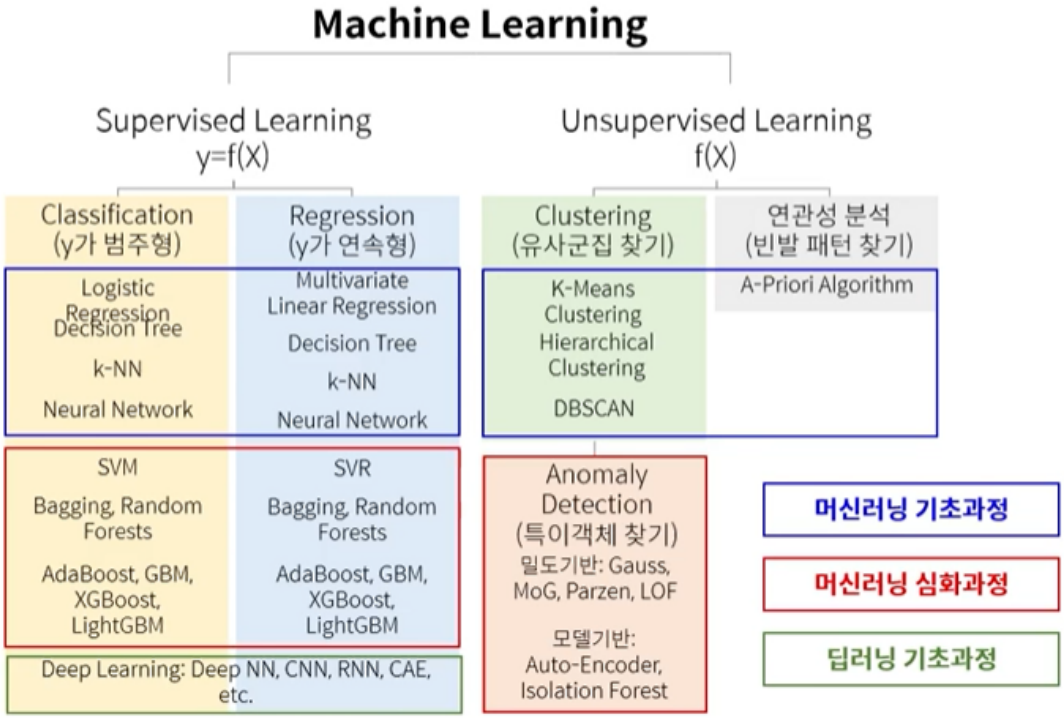
반면에 딥러닝은 머신러닝에서 사람이 하던 패턴 추출 작업이 생략된다. 컴퓨터가 스스로 데이터를 기반으로 학습할 수 있도록 정해진 신경망을 컴퓨터에게 주고, 어린아이가 학습하는 것처럼 경험 중심으로 학습을 수행한다.



▲ 인간이 개, 고양이의 특성을 추려 사전에 정의된 알고리즘과 규칙을 적용하는 머신러닝과 달리, 딥러닝에서는 심층 신경망을 통해 스스로 개, 고양이의 특성을 훈련하여 개와 고양이를 분류할 수 있다.

	머신러닝	딥러닝
데이터 요소 수	적은 양의 데이터를 사용하여 예측을 만들 수 있다.	예측을 수행하려면 많은 양의 학습 데이터를 사용해야 한다.
하드웨어 종속성	저사양 머신에서 작동할 수 있으며 컴퓨팅 능력이 많이 필요하지 않다.	고성능 머신에 의해 결정된다. 본질적으로 많은 수의 행렬 곱하기 연산을 수행한다. GPU는 이러한 작업을 효율적으로 최적화할 수 있다.
기능화 프로세스	사용자가 기능을 정확하게 식별하고 만들어야 한다.	데이터의 고급 기능을 학습하고 자체적으로 새 기능을 만든다.
학습 방법	학습 프로세스를 더 작은 단계로 나눈 후 각 단계의 결과들을 하나의 출력으로 결합한다.	엔드투엔드 단위로 문제를 해결하여 학습 프로세스를 진행한다.
실행 시간	학습하는 데 비교적 적은 시간이 소요된다.	알고리즘에 많은 레이어가 포함되어 있어 일반적으로 학습하는 데 시간이 오래 걸린다.
출력	일반적으로 점수 또는 분류와 같은 숫자 값이다.	텍스트, 점수 또는 사운드와 같은 여러 형식이 있을 수 있다.

▲ 머신러닝과 딥러닝의 특징

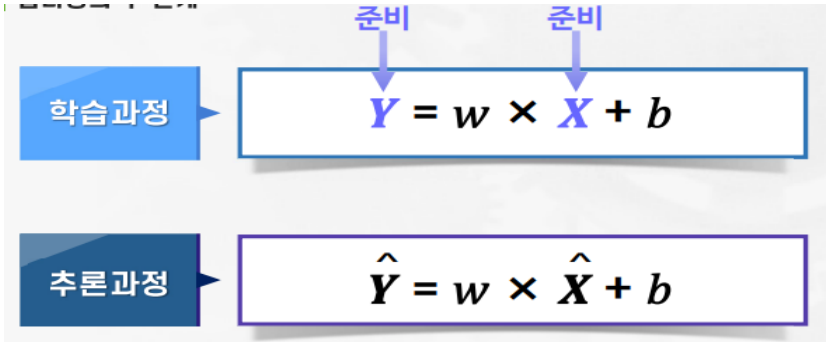
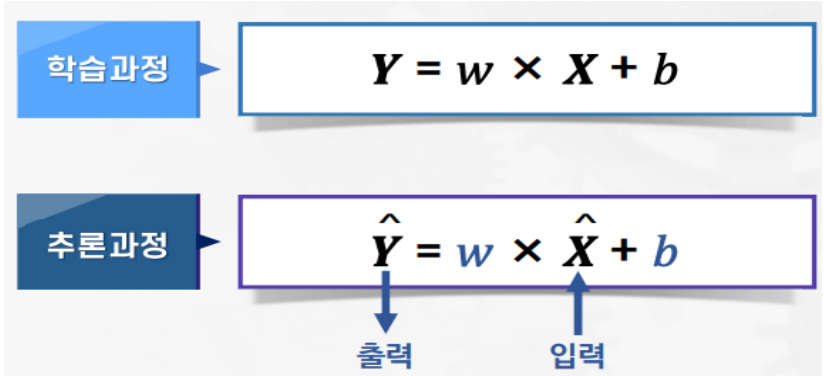


1.3 딥러닝 학습

딥러닝에도 머신러닝과 마찬가지로 분류와 회귀, 지도 학습과 비지도 학습이 있다. 유사 정답(pseudo label)을 만들고 정답 라벨 데이터를 기반으로 기존의 모델에 결합하는 준지도 학습(Semi-Supervised Learning)도 사용된다. 최근에는 레이블이 없는 데이터에서 특징표현(representation)을 선정하고, 학습한 특징 추출 모델을 활용하여 유사한 문제를 해결하는 자기 지도 학습(Self-Supervised Learning)방식도 활발히 연구되고 있다.

2. 선형 회귀 분석 비교

2.1 개요

	Y는 trainset의 label, X는 feature가 들어간다. Label의 클래스 개수와 Feature 개수에 따라 뉴런수가 정해진다.
	$\hat{Y}$의 오차가 점점 줄어들도록 w,b값을 계속 학습을 시키고 최적의 w,b를 찾아 식을 완성한다. 완성한 식 $\hat{X}$에 test셋을 입력해주면 예측값인 $\hat{Y}$값이 출력된다.

단순 선형 회귀 식[1] 에서 $\hat{Y}$의 오차를 줄이기 위해서는, 함수를 평가하는 방법중 하나인 평균 제곱 오차(Mean Squared Error)[2]를 최소화하면 된다.

예측값과 실제값의 차이를 구하는 기준인 손실함수(Loss function)로는 평가지표 MSE를 사용할 수 있으며, 손실 함수의 최솟값은 미분을 이용해 찾을 수 있다.[3]

[1] $\hat{y} = ax+b$

[2] $MSE = 1/n \sum(y-\hat{y})^2$

[3] 평균 절대값 오차인 MAE은 기울기가 일정해 최적화가 불편하기에 일반적으로 MSE를 사용한다.

	선형회귀 손실 함수의 미분
가중치를 미분	$\frac{\partial MSE}{\partial w} = -(y - \hat{y})x$
절편을 미분	$\frac{\partial MSE}{\partial b} = -(y - \hat{y})$

위 표를 경사하강법 적용된 식으로 표현하면 아래와 같다.

경사하강법 가중치 부분	$w - \frac{\partial MSE}{\partial w} = w + (y - \hat{y})x$
경사하강법 절편 부분	$b - \frac{\partial MSE}{\partial b} = b + (y - \hat{y})$

본 보고서에서는 경사하강법을 적용하여 선형 회귀 분석을 진행한다.

2.1.1 목표

제품적절성이 제품만족도에 미치는 영향 주제로 R을 이용한 단순 선형 회귀분석을 실시하고, 인공신경망을 이용한 선형 회귀분석을 python으로 실행하여 결과를 비교한다.

2.1.2 분석방법

SyncRNG패키지를 써서 R과 Python의 랜덤 데이터를 동일하게 맞춘다. R은 lm함수, Python은 인공신경망 경사하강법을 이용해서 모델링후 test셋으로 예측 결과를 비교한다.

2.1.3 데이터 정의 및 탐색적 데이터 분석(EDA)

1) 데이터 정의

사용 데이터 : product.csv

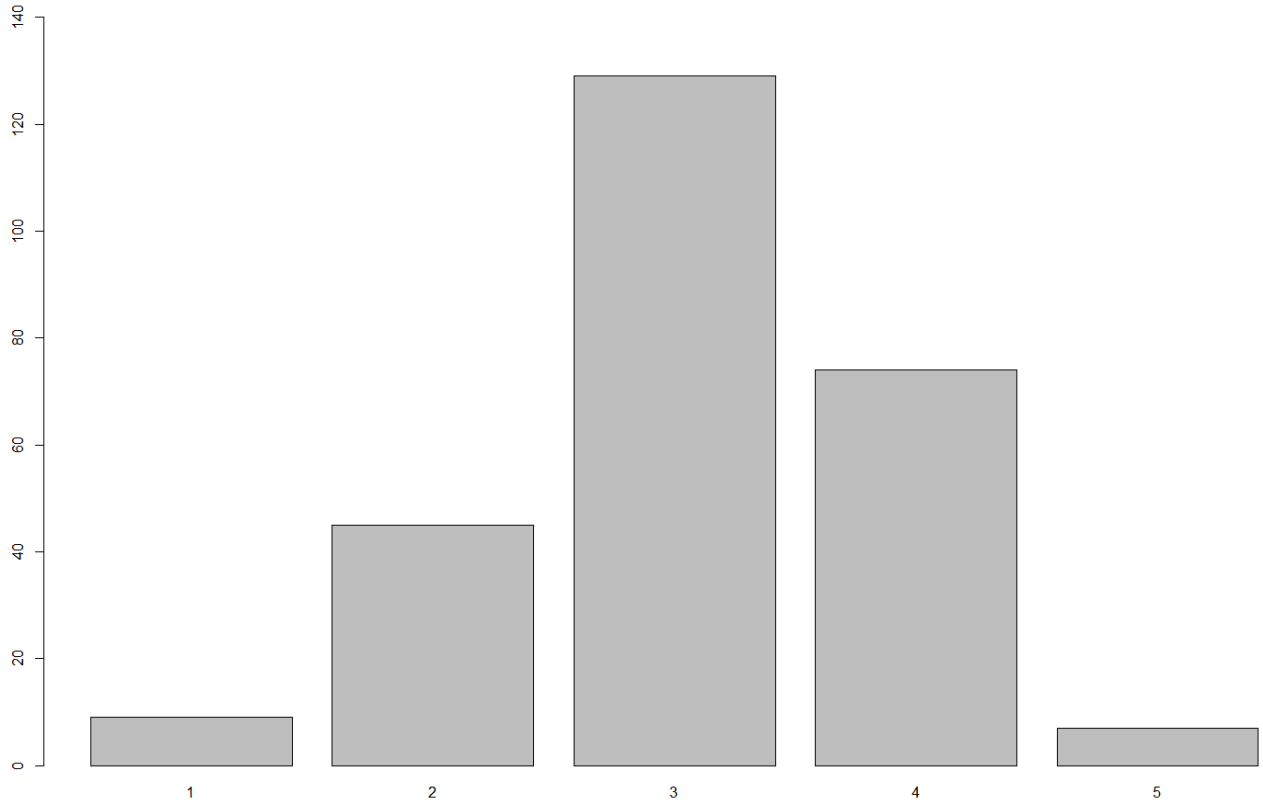
264개 제품의 점수를 "제품_친밀도, 제품_적절성, 제품_만족도"의 세 항목으로 나누어 살펴볼 수 있는 데이터이다. 각 항목은 최소1에서 최대5까지의 점수 값을 포함하고 있다. 본 보고서에서는 "제품_적절성"이 "제품_만족도"에 미치는 영향을 분석해 보고자 한다. 따라서 "제품_친밀도"는 제외하고 "제품_적절성", "제품_만족도" 두 개 열을 이용하여 분석을 수행한다.

2) 탐색적 데이터 분석(EDA)

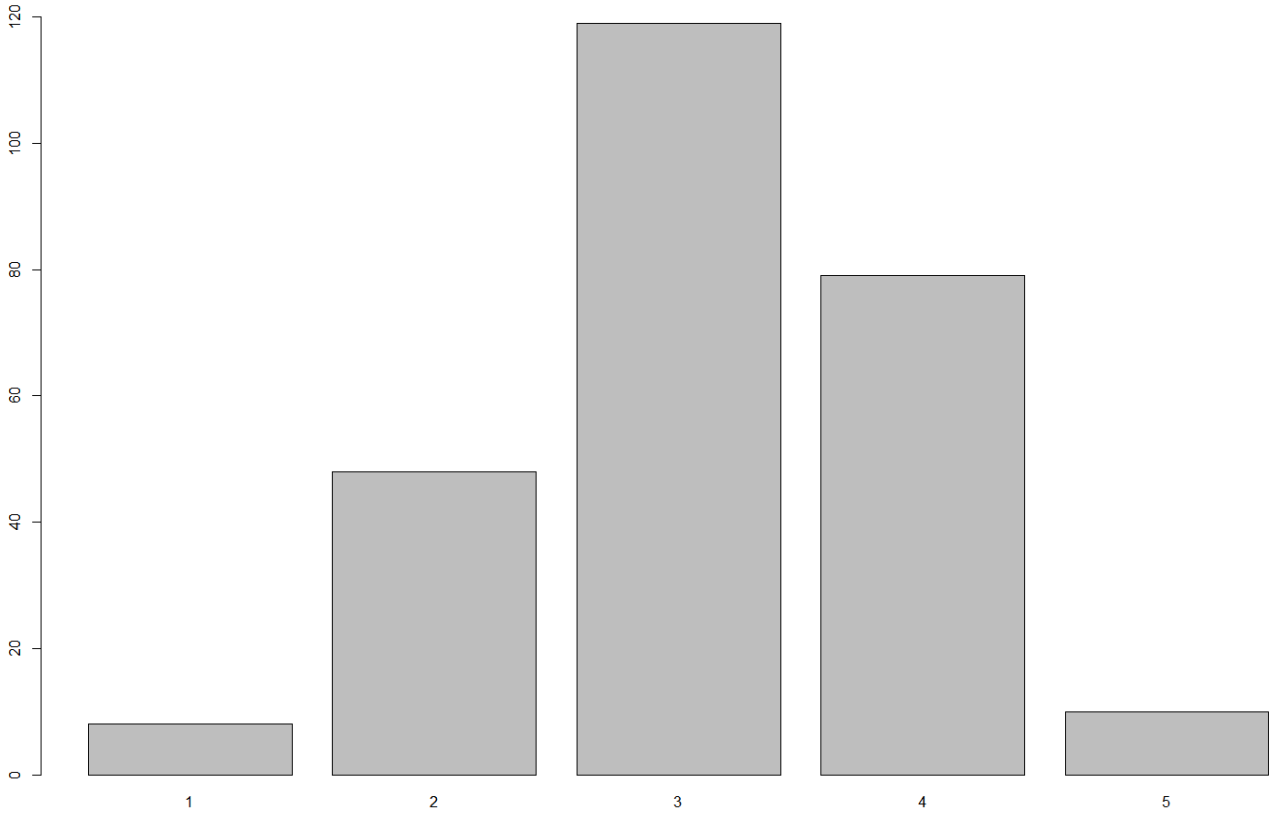
데이터셋의 구조와 데이터 요약

```
> summary(df)
 제품_친밀도      제품_적절성      제품_만족도
Min.   :1.000   Min.   :1.000   Min.   :1.000
1st Qu.:2.000   1st Qu.:3.000   1st Qu.:3.000
Median :3.000   Median :3.000   Median :3.000
Mean   :2.928   Mean   :3.133   Mean   :3.095
3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000
Max.   :5.000   Max.   :5.000   Max.   :5.000
```

```
> str(df)
'data.frame':    264 obs. of  3 variables:
 $ 제품_친밀도: int  3 3 4 2 2 3 4 2 3 4 ...
 $ 제품_적절성: int  4 3 4 2 2 3 4 2 2 2 ...
 $ 제품_만족도: int  3 2 4 2 2 3 4 2 3 3 ...
```



▲ 제품만족도 분포를 막대그래프으로 시각화



▲ 제품적절성 분포를 막대그래프으로 시각화

히스토그램을 통해 독립변수의 분포가 정규분포와 근사하다 볼 수 있다.

탐색적 데이터 분석 결과, 결측치나 이상치를 발견할 수 없었고 전처리가 필요한 변수도 없음을 확인했다.

2.2 R 단순 선형회귀 분석

2.2.1 데이터 호출

```
library(ggplot2)
library(caret)
df<-read.csv("dataset2/product.csv")
```

2.2.2 모델 생성

```
library(SyncRNG)
v <- 1:nrow(df)
s <- SyncRNG(seed=42)
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]

idx[1:length(idx)]
train <- df[idx,]
test <- df[-idx,]

m_lm<- lm(제품_만족도~제품_적절성, train)
```

```
> m_lm

Call:
lm(formula = 제품_만족도 ~ 제품_적절성, data = train)

Coefficients:
(Intercept)  제품_적절성
     0.7171       0.7607
```

파이썬과 R에서 동일한 데이터로 홀드아웃 교차검증을 진행하기 위해 SyncRNG패키지를 이용하여 데이터를 분할하였다.

문제에서 독립변수(적절성), 종속변수(만족도)를 정해주었기 때문에 해당 변수로 단순 선형회귀 분석을 실시한다.

```
> summary(m_lm)
Call:
lm(formula = 제품_만족도 ~ 제품_적절성, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.75974 -0.23841  0.00092  0.24026  1.24026

Coefficients:
             Estimate   Std. Error  t value     Pr(>|t|)
(Intercept)   0.71708    0.14118    5.079      9.29e-07 ***
제품_적절성   0.76067    0.04402   17.281      < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.522 on 183 degrees of freedom
Multiple R-squared:  0.62,    Adjusted R-squared:  0.618
F-statistic: 298.6 on 1 and 183 DF,  p-value: < 2.2e-16
```

p-value<2.2e-16이므로 모델이 통계적으로 유의하고, 결정계수는 0.62이다.

회귀식은 제품_만족도=0.76067*제품_적절성 + 0.71708 이다.

2.2.3 모델 평가

```
library(Metrics)
library(caret)
p_lm<-predict(m_lm,test[,-1])
RMSE(p_lm,test[,3])
R2(p_lm,test[,3])
```

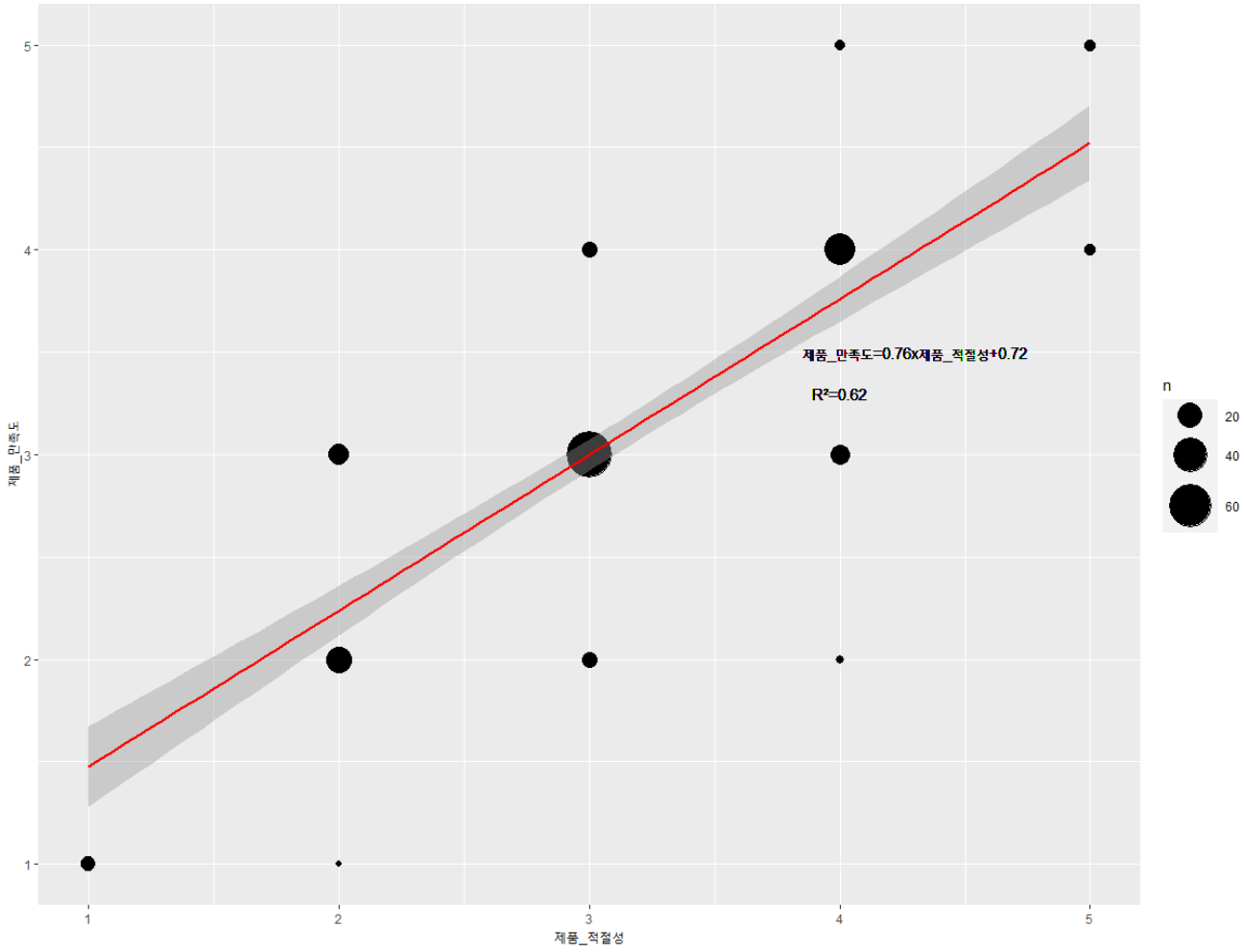
```
> RMSE(p_lm,test[,3])
0.5585512
> R2(p_lm,test[,3])
0.5023452
```

train셋으로 학습한 모델로 test셋을 예측한 결과 RMSE는 0.559, R2는 0.502가 나왔다

2.2.4 시각화

```
ggplot(train,aes(x=제품_적절성,y=제품_만족도))+
  geom_count()+
  scale_size_area(max_size = 15)+
  stat_smooth(method='lm',color='red')+
  geom_text(x=4.3,y=3.5,label="제품_만족도=0.76x제품_적절성+0.72")+
  geom_text(x=4,y=3.3,label="R²=0.62")
```

제품적절성에 대한 제품만족도의 변화를 점 그래프로 시각화 하였다. 중첩되는 데이터가 많을수록 점의 크기가 커지도록 했으며 회귀선을 추가해 모델링 결과를 대략적으로 파악 할 수 있도록 했다.



▲ 점은 제품_적절성에 따른 제품_만족도의 누적 크기로 적절성이 3일때 만족도 3의 갯수가 가장 많았고 선은 제품_만족도=0.76067*제품_적절성 + 0.71708을 설명하고 있다

2.3 Python 단순 선형회귀 분석

2.3.1 데이터 호출

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from SyncRNG import SyncRNG

raw_data = pd.read_csv('E:/GoogleDrive/A5팀 프로젝트 자료(12월22일)/dataset/product.csv',encoding='cp949')

v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=42)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]

for i in range(0,len(idx)):
    idx[i]=idx[i]-1

train=raw_data.loc[idx]
test=raw_data.drop(idx)

x_train = train.제품_적절성
y_train = train.제품_만족도
x_test = test.제품_적절성
y_test = test.제품_만족도
```

컬럼명에 한글이 있어서 encoding='cp949' 옵션을 추가 했다. 파이썬과 R에서 동일한 데이터로 홀드아웃 교차검증을 진행하기 위해 SyncRNG패키지를 이용하여 데이터를 분할하였다.

2.3.2 모델 생성

```
class Neuron:
    def __init__(self):
        self.w = 1  # 가중치를 초기화합니다
        self.b = 1 # 절편을 초기화합니다

    def forpass(self, x):
        y_hat = x * self.w + self.b  # 직선 방정식을 계산합니다
        return y_hat

    def backprop(self, x, err):
        w_grad = x * err  # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err  # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad

    def fit(self, x, y,lr, epochs=400):
        for i in range(epochs):  # 에포크만큼 반복합니다
            for x_i, y_i in zip(x, y):  # 모든 샘플에 대해 반복합니다
                n=len(x)
                y_hat = self.forpass(x_i)  # 정방향 계산
                err = -(2/n)*(y_i - y_hat)  # 오차 계산
                w_grad, b_grad = self.backprop(x_i, err)  # 역방향 계산
                self.w -= w_grad*lr  # 가중치 업데이트
                self.b -= b_grad*lr # 절편 업데이트

neuron = Neuron()
neuron.fit(x_train, y_train,0.1)
print(neuron.w)
print(neuron.b)
```

뉴런 클래스를 생성한다. 클래스 내에 가중치(w), 절편(b)를 1로 초기화 하고 정방향, 역방향 계산하는 메서드와 모델링하는 메서드를 생성한다. 에포크수는 손실값이 수렴하기 시작할때의 값으로 설정해주었다.

실행결과 회귀식은 제품_만족도=0.7551*제품_적절성 + 0.71541 이다.

2.3.3 예측 및 평가

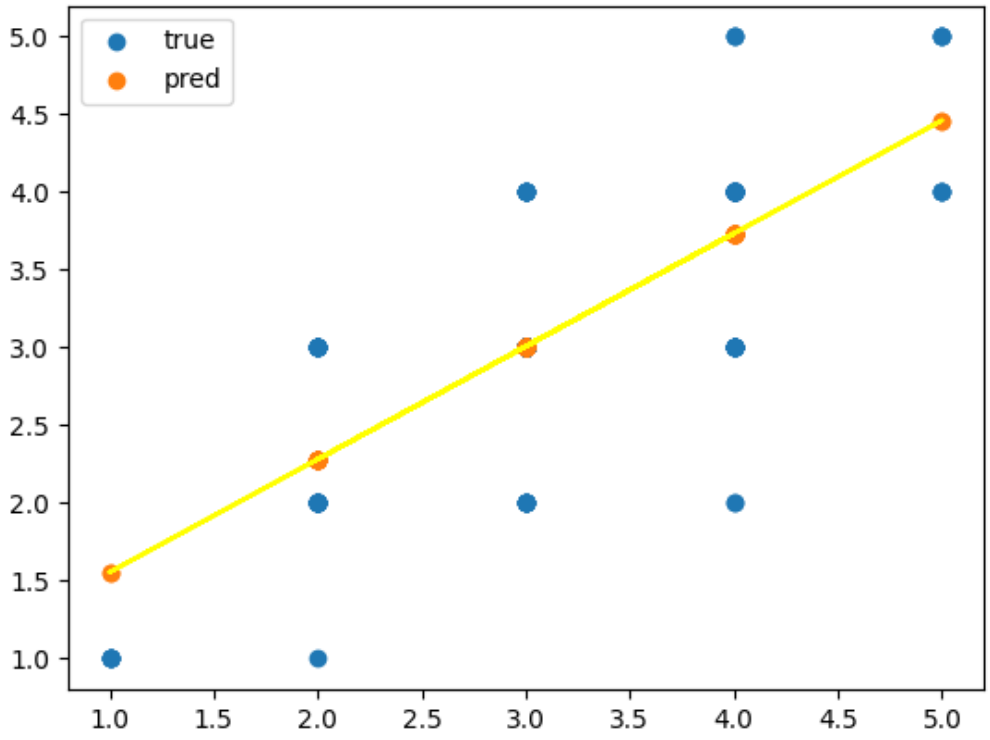
```
predict=[]
predict = x_test * neuron.w + neuron.b

from sklearn.metrics import mean_squared_error, r2_score
mse=mean_squared_error(predict, y_test)
import math
math.sqrt(mse)
r2_score(y_test, predict)
```

손실함수가 최소일때 W,b값으로 식을 새로 세워 test셋을 대입하여 예측값을 구한다. 예측값과 실제값의 차이를 제곱해서 평균(MSE)으로 회귀 모델을 평가한다. RMSE 값은 0.557795이 R2는 0.4969가 나왔다.

2.3.4 시각화

```
plt.scatter(x_train, y_train, label='true')
plt.scatter(x_test, predict, label='pred')
pt1 = (1, 1 * neuron.w + neuron.b)
pt2 = (5, 5 * neuron.w + neuron.b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]],color='orange')
plt.legend()
plt.show
```

제품적절성에 대한 제품만족도의 변화를 점 그래프로 시각화 하였다. 회귀선을 추가해 모델링 결과를 대략적으로 파악 할 수 있도록 했다.



▲ 파란점은 train셋을 표현한 것이고 노란 선은 제품_만족도=0.7551*제품_적절성 + 0.71541을 설명하고 있다.

2.4 결론(비교)

	R	Python
직선의 기울기	0.76067	0.75512
절편	0.71708	0.71541
RMSE	0.55855	0.55780
R2	0.50235	0.49692

위 표는 제품적절성이 제품만족도에 미치는 영향을 각각 R을 이용한 단순 선형 회귀분석과 인공신경망을 이용한 선형 회귀분석을 python으로 진행한 결과를 비교한 것이다. R은 lm함수, Python은 인공신경망 경사하강법을 이용해서 모델링후 test셋으로 예측하였다. 회귀식은 비슷한 결과를 보여주고 있지만, RMSE와 R2에서 R이 조금 더 높은 결과를 보여주고 있다.
따라서, R이 조금 더 좋은 회귀식을 제시했다고 할 수 있다.

3. 로지스틱 회귀 분석 비교

3.1 개요

로지스틱 손실 함수[4]의 값을 최소로 하는 가중치와 절편을 찾기 위해 미분을 이용한다.

	로지스틱 손실 함수의 미분
가중치를 미분	$\frac{\partial}{\partial w_i} L = -(y - a)x_i$
절편을 미분	$\frac{\partial}{\partial b} L = -(y - a)$

위 표를 경사하강법 적용된 식으로 표현하면 아래와 같다.

경사하강법 가중치 부분	$w_i - \frac{\partial}{\partial w_i} L = w_i + (y - a)x_i$
경사하강법 절편 부분	$b - \frac{\partial L}{\partial b} = b + (y - a)$

본 보고서에서는 경사하강법을 적용하여 로지스틱 회귀 분석을 진행한다.

3.1.1 목표

비가 오는지를 예측하기 위해 R을 이용한 로지스틱 회귀분석을 실시하고, 인공신경망을 이용한 로지스틱 회귀분석을 python으로 실행하여 결과를 비교한다.

3.1.2 분석방법

R에서는 EDA를 실시하여 전처리 내용 확인 후, 목적에 맞지 않거나 다중 공선성이 발생하는 변수를 제거 하고 스케일링과 인코딩을 실시한다. 제거된 변수에 한해서 후진제거법을 이용해 로지스틱 모델을 재정의 후 최종 모델을 평가하고 시각화한다.

Python에서는 전처리 실시 후 인공신경망을 적용한 로지스틱 회귀분석을 수행하는 사용자 정의 클래스를 만들고 평가 후 시각화한다.

---

[4] L=-(y log(a)+(1-y)log(1-a))

(a는 활성화 함수가 출력한 값(1/1+(e)^-z)이고 y는 타깃이다.)

3.1.3 데이터 정의 및 탐색적 데이터 분석(EDA)

1) 데이터 정의

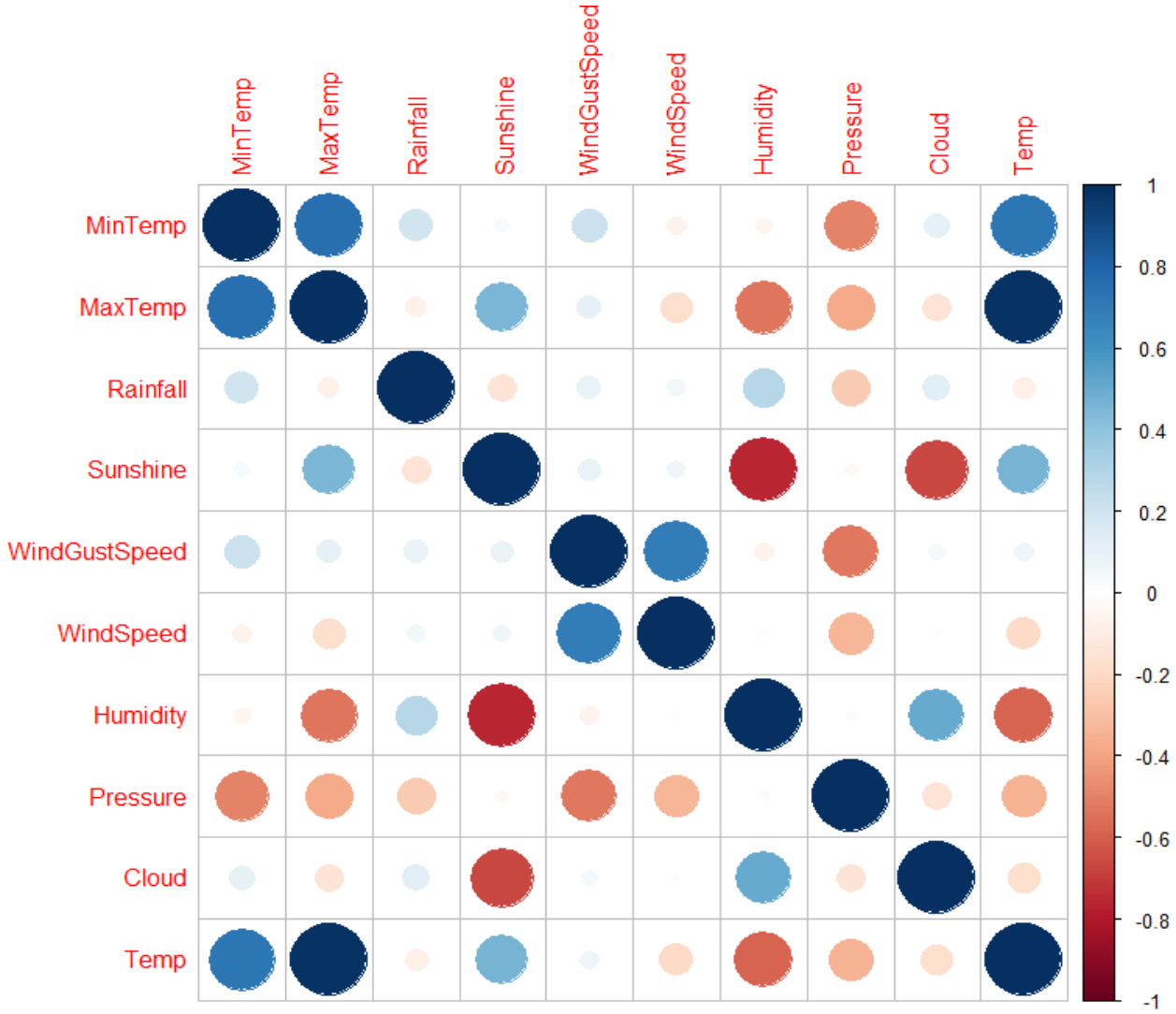
사용 데이터: weather.csv

기상 관측 자료를 날짜별, 항목별로 기록한 자료이다. 각 항목에 따라 다른 자료형과 단위를 포함한다. 독립변수는 RainTomorrow 이며, 종속변수는 Date부터 RainToday 까지 12개이다.

변수명	설명
Date	The date of observation / 관측일
MaxTemp	The maximum temperature in degrees celsius. / 최고 온도 (섭씨)
MinTemp	The minimum temperature in degrees celsius. / 최저 온도 (섭씨)
Rainfall	The amount of rainfall recorded for the day in mm. / 일일 강수량 (mm)
Sunshine	The number of hours of bright sunshine in the day. / 일조 시간 (hour)
WindGustDir	The direction of the strongest wind gust in the 24 hours to midnight. / 가장 센 wind gust (순간적으로 강해지는 바람) 의 방향
WindSpeed	Wind speed (km/hr) averaged over 10 minutes. / 평균 풍속 (10분)
Humidity	The concentration of water vapor present in the air. / 습도
Pressure	Atmospheric pressure (hpa) reduced to mean sea level. / 대기압
Cloud	Fraction of sky obscured by cloud. / 구름이 하늘을 덮은 정도
Temp	Temperature (degrees C) / 기온
RainToday	Yes: if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise No / 09시까지 1mm이상 강수량이면 Yes, 아니면 No
RainTomorrow	The target variable. Did it rain tomorrow? YES/NO / 타겟 변수. 내일(다음날) 비가 올지를 예측

2) 탐색적 데이터 분석 (EDA)

```
> summary(df)
        Date         MinTemp          MaxTemp         Rainfall      
 2014-11-01:  1   Min.   :-5.300   Min.   : 7.60   Min.   : 0.000  
 2014-11-02:  1   1st Qu.: 2.300   1st Qu.:15.03   1st Qu.: 0.000  
 2014-11-03:  1   Median : 7.450   Median :19.65   Median : 0.000  
 2014-11-04:  1   Mean   : 7.266   Mean   :20.55   Mean   : 1.428  
 2014-11-05:  1   3rd Qu.:12.500   3rd Qu.:25.50   3rd Qu.: 0.200  
 2014-11-06:  1   Max.   :20.900   Max.   :35.80   Max.   :39.800  
 (Other)   :360                                                    

    Sunshine       WindGustDir   WindGustSpeed      WindDir      WindSpeed     
 Min.   : 0.000   NW     : 73   Min.   :13.00   NW     : 61   Min.   : 0.00  
 1st Qu.: 5.950   NNW    : 44   1st Qu.:31.00   WNW    : 61   1st Qu.:11.00  
 Median : 8.600   E      : 37   Median :39.00   NNW    : 47   Median :17.00  
 Mean   : 7.909   WNW    : 35   Mean   :39.84   N      : 30   Mean   :17.99  
 3rd Qu.:10.500   ENE    : 30   3rd Qu.:46.00   ESE    : 27   3rd Qu.:24.00  
 Max.   :13.600   (Other):144   Max.   :98.00   (Other):139   Max.   :52.00  
 NA's   :3        NA's   : 3    NA's   :2       NA's   : 1                   

    Humidity        Pressure         Cloud            Temp          RainToday 
 Min.   :13.00   Min.   : 996.8   Min.   :0.000   Min.   : 5.10   No :300   
 1st Qu.:32.25   1st Qu.:1012.8   1st Qu.:1.000   1st Qu.:14.15   Yes: 66   
 Median :43.00   Median :1017.4   Median :4.000   Median :18.55             
 Mean   :44.52   Mean   :1016.8   Mean   :4.025   Mean   :19.23             
 3rd Qu.:55.00   3rd Qu.:1021.5   3rd Qu.:7.000   3rd Qu.:24.00             
 Max.   :96.00   Max.   :1033.2   Max.   :8.000   Max.   :34.50             

 RainTomorrow
 No :300     
 Yes: 66     
```



▲ numeric 변수간 상관관계 시각화

로지스틱 회귀분석을 하는 데 있어 데이터의 날짜나 시간의 흐름이 필수가 아니라고 판단하여, 날짜 변수(DATE)는 삭제하고 분석을 진행하기로 했다.

또한 summary 결과 Sunshine, WindGustDir, WindGustSpeed, WindDir에서 결측치가 확인되므로 전처리 과정에서 결측치를 제거할 것이다. 그리고 숫자형 변수들은 범위가 다르므로 스케일링을 실시하고, 범주형 변수들은 인코딩을 실시하여 데이터를 목적에 맞게 전처리 한다. 위 <numeric 변수간 상관관계 시각화> 그림을 보면, 온도변수간(~Temp) 상관관계가 매우 높다는 것을 확인할 수 있었다.

3.2 R 로지스틱 회귀 분석

3.2.1 데이터 호출 및 전처리

```
df<-read.csv("weather.csv")
df<-df[,-1]
df<-na.omit(df)
p<-preProcess(df,"range")
df<-predict(p, df)
```

Date열을 삭제하고 Sunshine, WindGustDir, WindGustSpeed, WindDir의 결측치를 삭제했다. 숫자형 변수의 범위를 전부 0~1로 정규화 했다.

3.2.2 모델 생성

1) 로지스틱 회귀분석 모델 생성

```
library(SyncRNG)
v <- 1:nrow(df)
s <- SyncRNG(seed=42)
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]

idx[1:length(idx)]
train <- df[idx,]
test <- df[-idx,]

m_glm<-glm(RainTomorrow~.,train,family = 'binomial')
```

```
> vif(m_glm)
                  GVIF  Df GVIF^(1/(2*Df))
MinTemp       9.987353   1        3.160277
MaxTemp     103.489052   1       10.172957
Rainfall      4.618540   1        2.149079
Sunshine      5.706559   1        2.388841
WindGustDir 149.878842  15        1.181747
WindGustSpeed 3.621736   1        1.903086
WindDir     250.865501  15        1.202213
WindSpeed     4.018268   1        2.004562
Humidity     11.525174   1        3.394875
```

Pressure	3.262373	1	1.806204
Cloud	2.816897	1	1.678361
Temp	112.272939	1	10.595893
RainToday	4.104661	1	2.025996

df<-df[,-c(1,2,5,7)]

> vif(m_glm) #(변수 제거 후)

Rainfall	Sunshine	WindGustSpeed	WindSpeed	Humidity	Pressure	Cloud
2.383619	3.124489	2.253798	2.114723	3.763055	1.901527	1.725216

Temp	RainToday
2.938106	2.283285

모델 생성 결과 <glm.fit: 적합된 확률값들이 0또는 1입니다> 라는 경고 메세지를 출력하는데, 이는 다중공선성 때문이다. 다중공선성을 제거하기 위해 GVIF^(1/(2*Df)) 계수가 10이상이면서 EDA에서 상관성이 높았던 변수(MinTemp, MaxTemp)를 삭제했다. 이 때, MinTemp는 계수는 유의하나 MaxTemp와 연관이 있기 때문에 삭제했다. 그리고 WindGustDir과 WindDir은 인코딩 시 차원이 지나치게 많아지는 문제가 발생하여 삭제처리했다.
변수(MinTemp, MaxTemp, WindGustDir, WindDir) 제거 후 다중공선성 문제가 해결되었다.

> summary(m_glm)

Call:

glm(formula = RainTomorrow ~ ., family = "binomial", data = train)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.1641	-0.4185	-0.2097	-0.0936	2.9673

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-2.2881	2.5532	-0.896	0.37017	
Rainfall	-0.8210	1.9925	-0.412	0.68030	
Sunshine	-2.4072	1.4188	-1.697	0.08975	.
WindGustSpeed	7.7124	2.3636	3.263	0.00110	**
WindSpeed	-4.3129	1.9332	-2.231	0.02569	*
Humidity	4.0970	1.9471	2.104	0.03537	*

```
Pressure          -5.5289     1.7796   -3.107   0.00189 **
Cloud              1.4177     1.0200    1.390   0.16453
Temp               1.9017     1.4710    1.293   0.19610
RainTodayYes      -0.3432     0.6819   -0.503   0.61477
---
 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 236.49  on 251  degrees of freedom
Residual deviance: 135.65  on 242  degrees of freedom
AIC: 155.65

Number of Fisher Scoring iterations: 6
```

2) 후진제거법으로 모델 재정의

```
m_glm=step(m_glm, direction = "backward")
summary(m_glm)
 Call:
   glm(formula = RainTomorrow ~ Sunshine + WindGustSpeed + WindSpeed +
         Humidity + Pressure + Cloud, family = "binomial", data = train)

 Deviance Residuals:
   Min       1Q   Median       3Q      Max
-2.2317  -0.4318  -0.2244  -0.1064   2.9461

 Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)         -0.1487     1.8355  -0.081 0.935433
Sunshine            -2.6014     1.3771  -1.889 0.058886 .
WindGustSpeed        8.1376     2.3437   3.472 0.000516 ***
WindSpeed           -5.2871     1.7763  -2.976 0.002916 **
Humidity             2.1781     1.4704   1.481 0.138538
Pressure            -5.9720     1.5375  -3.884 0.000103 ***
Cloud                1.4371     0.9835   1.461 0.143933
---
  Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 236.49  on 251  degrees of freedom
Residual deviance: 138.43  on 245  degrees of freedom
AIC: 152.43
Number of Fisher Scoring iterations: 6
```

통계적으로 유의하지 않은 변수가 여전히 존재하기 때문에, 후진제거법을 이용했다. 유의하지 않은 변수 Rainfall, RainToday, Temp를 하나씩 제거하며 로지스틱 회귀 모델을 새롭게 정의했다. 새로 정의한 모델의 AIC값은 152.43이다.

3.2.4 모델 평가

```
p_glm<-predict(m_glm,test,type='response')
p.glm<-ifelse(p_glm>=0.5,'Yes','No')

table(test$RainTomorrow)
table(p.glm)

caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)
caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)$byClass
caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)$overall
```

Confusion Matrix and Statistics

```
Reference
Prediction No Yes
No  84  10
Yes  4   9

Accuracy : 0.8692
95% CI : (0.7902, 0.9266)
No Information Rate : 0.8224
P-Value [Acc > NIR] : 0.1253

Kappa : 0.4887
```

Mcnemar's Test P-Value : 0.1814

Sensitivity : 0.9545
Specificity : 0.4737
Pos Pred Value : 0.8936
Neg Pred Value : 0.6923
Prevalence : 0.8224
Detection Rate : 0.7850
Detection Prevalence : 0.8785
Balanced Accuracy : 0.7141

'Positive' Class : No

Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision
0.9545455	0.4736842	0.8936170	0.6923077	0.8936170

Recall	**F1**	Prevalence	Detection Rate	Detection Prevalence
0.9545455	**0.9230769**	0.8224299	0.7850467	0.8785047

Balanced Accuracy
0.7141148

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.8691589	0.4887372	0.7902236	0.9265834	0.8224299	0.1253417

McnemarPValue
0.1814492

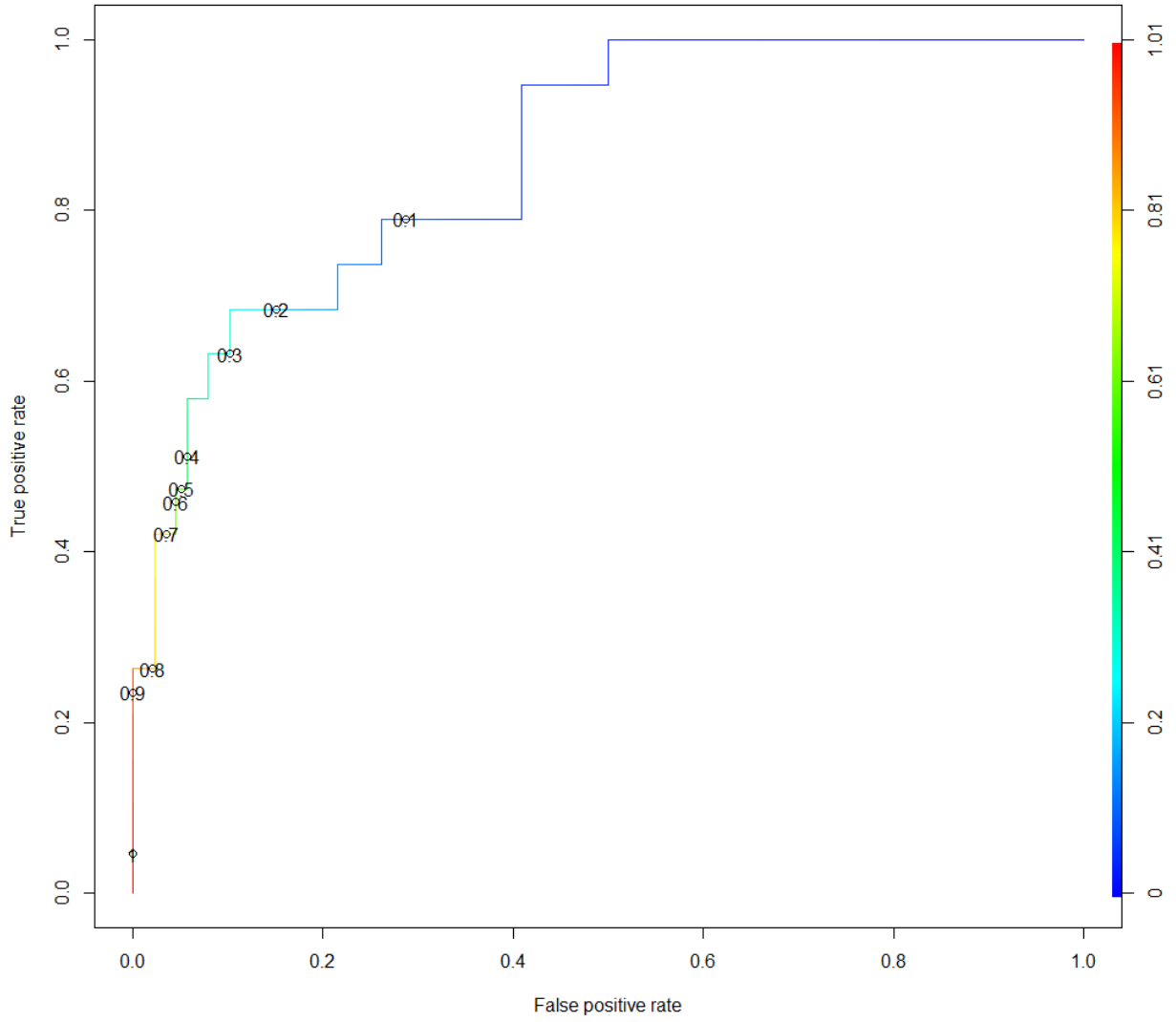
auc(as.factor(p.glm),test$RainTomorrow)
0.7929624

모델 생성 시 나눴던 train data와 test data를 이용한 홀드아웃 교차검증으로 모델 평가를 실시했다. 분류모델 평가를 위해 caret 패키지의 confusionmatrix를 사용한 결과 정확도(Accuracy) 0.87, 민감도(Sensitivity) 0.95, F1-Score 0.92, AUC 0.79 이다.

3.2.5 시각화

> ROC 곡선

```
library(ROCR)[5]
ROCR_p_glm <- prediction(p_glm,list(test$RainTomorrow))
ROCR_pf_glm <- performance(ROCR_p_glm,'tpr','fpr')
plot(ROCR_pf_glm,colorize=TRUE,print.cutoffs.at=seq(0.1,by=0.1))
```



▲ 로지스틱 회귀 분석 ROC curve. 위 그래프를 통해 TPR(True positive rate)을 높이고 FPR(False positive rate)을 줄이는 임계값을 찾아 선택할 수 있다. AUC값이 0.79로 성능이 좋은 모델이라는 것을 확인할 수 있다.

[5] https://cran.r-project.org/web/packages/ROCR/ROCR.pdf

3.3 Python 로지스틱 회귀 분석

3.3.1 데이터 호출 및 전처리

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

raw_data = pd.read_csv('E:/GoogleDrive/A5팀 프로젝트
자료(12월22일)/dataset/weather.csv',encoding='cp949')

raw_data.drop(['Date'],axis=1,inplace=True)

le = LabelEncoder()
cat_cols = ['WindGustDir', 'WindDir', 'RainToday','RainTomorrow']
raw_data[cat_cols] = raw_data[cat_cols].apply(le.fit_transform)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

cols = ['MinTemp', 'MaxTemp', 'Rainfall', 'Sunshine', 'WindGustDir',
       'WindGustSpeed', 'WindDir', 'WindSpeed', 'Humidity', 'Pressure',
       'Cloud', 'Temp', 'RainToday', 'RainTomorrow']
raw_data[cols] = scaler.fit_transform(raw_data[cols])

v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=42)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]

for i in range(0,len(idx)):
   idx[i]=idx[i]-1

train=raw_data.loc[idx] # 70%
test=raw_data.drop(idx) # 30%
```

```
train=raw_data.dropna()
test=raw_data.dropna()

x_train = np.array(train.iloc[:,0:13], dtype=np.float32)
y_train = np.array(train.iloc[:,-1], dtype=np.float32)
x_test = np.array(test.iloc[:,0:13], dtype=np.float32)
y_test = np.array(test.iloc[:,-1], dtype=np.float32)
```

R에서와 마찬가지로 DATE 변수를 제거했으며, 범주형 변수는 인코딩하고 나머지 변수들은 최소-최대 정규화를 실시했다. syncRNG 함수를 사용하여 R과 같은 seed값으로 정확한 비교를 할 수 있게끔 설정했다. 추가로 로지스틱 회귀모델을 수행하는 클래스에서는 numpy 연산이 사용되므로 학습데이터와 테스트데이터를 array로 변경했다.

3.3.2 모델 생성

```
class SingleLayer:


    def __init__(self):
        self.w = None
        self.b = None
        self.losses = []


    def forpass(self, x):
        z = np.sum(x * self.w) + self.b  # 직선 방정식을 계산합니다
        return z


    def backprop(self, x, err):
        w_grad = x * err  # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err  # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad
```

__init()__메서드는 추후 입력 데이터의 특성 개수에 맞게 결정하기 위해 가중치를 미리 초기화하지 않았다. 그리고 손실 함수의 결과값이 줄어드는지를 관찰하기 위해 self.losses

리스트를 만들었다.

또한 로지스틱 회귀는 정방향 계산과 가중치를 업데이트하기 위한 역방향 계산(오차 역전파)이 필요하다. 이를 위해 forpass, backprop 메서드를 구현했다. backprop 메서드를 통해 가중치와 절편이 업데이트되어 점차 최적화된 가중치와 절편을 얻을 것이다.

```
    def activation(self, z):
        z = np.clip(z, -100, None)  # 안전한 np.exp() 계산을 위해
        a = 1 / (1 + np.exp(-z))  # 시그모이드 계산
        return a

    def fit(self, x, y, epochs=500):
        self.w = np.ones(x.shape[1])  # 가중치를 초기화합니다.
        self.b = 0  # 절편을 초기화합니다.
        for i in range(epochs):  # epochs만큼 반복합니다
            loss = 0
            # 인덱스를 섞습니다
            indexes = np.random.permutation(np.arange(len(x)))
            for i in indexes:  # 모든 샘플에 대해 반복합니다
                n = len(x)
                z = self.forpass(x[i])  # 정방향 계산
                a = self.activation(z)  # 활성화 함수 적용
                err = -(y[i] - a)  # 오차 계산
                w_grad, b_grad = self.backprop(x[i], err)  # 역방향 계산
                self.w -= w_grad  # 가중치 업데이트
                self.b -= b_grad  # 절편 업데이트
                # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다
                a = np.clip(a, 1e-10, 1 - 1e-10)
                loss += -(y[i] * np.log(a) + (1 - y[i]) * np.log(1 - a))
            # 에포크마다 평균 손실을 저장합니다
            self.losses.append(loss / len(y))
```

activation 메서드는 z를 0~1사이의 확률값으로 변환시키기 위한 시그모이드 함수, 즉 활성화

함수 역할을 한다. 시그모이드 함수는 오즈비를 기반으로 하므로 이에 따라 계산식을 만들었다. fit은 훈련을 수행하는 메서드로, 먼저 에포크마다 인덱스를 무작위로 섞음으로써 손실 함수를 줄여 성능 향상을 했다. 그리고 로지스틱 회귀분석의 과정에 따라 정방향 계산과 역방향 계산, 가중치와 절편 업데이트, 마지막으로 손실 누적까지의 과정을 모든 샘플에 대해 반복한다.

```
    def predict(self, x):
        z = [self.forpass(x_i) for x_i in x]  # 정방향 계산
        return np.array(z) > 0  # 스텝 함수 적용

    def proba(self,x):   #ROC 커브용 확률출력 메서드
        z = [self.forpass(x_i) for x_i in x]
        return np.array(z)

    def score(self, x, y):
        return np.mean(self.predict(x) == y)

layer = SingleLayer()
layer.fit(x_train, y_train)
layer.score(x_test, y_test)
layer.predict(x_test)
```

여러 개의 샘플(활성화 함수, 임계 함수)을 적용한 예측값을 얻기 위해 predict 메서드를 만들었다. 선형 함수(z)가 0보다 크면 시그모이드 함수의 출력값이 0.5보다 크므로 시그모이드 함수를 사용하는 대신 선형 함수(z)가 0보다 큰지, 작은지를 비교한다.

ROC curve 시각화를 위해 확률을 출력해주는 porba 메서드를 정의한 후, 정확도를 계산해주는 score 메서드를 정의했다. 이는 올바르게 예측한 샘플의 비율이 된다.

이렇게 정의한 SingleLayer 클래스를 layer객체로 만들고 훈련 세트로 훈련시켰다. 모델의 정확도는 아래 <3.3.3 모델 평가>에서 확인할 수 있다.

3.3.3 모델 평가

```
from sklearn.metrics import classification_report
print(classification_report(layer.predict(x_test) , y_test))
```

```
              precision    recall  f1-score   support
```

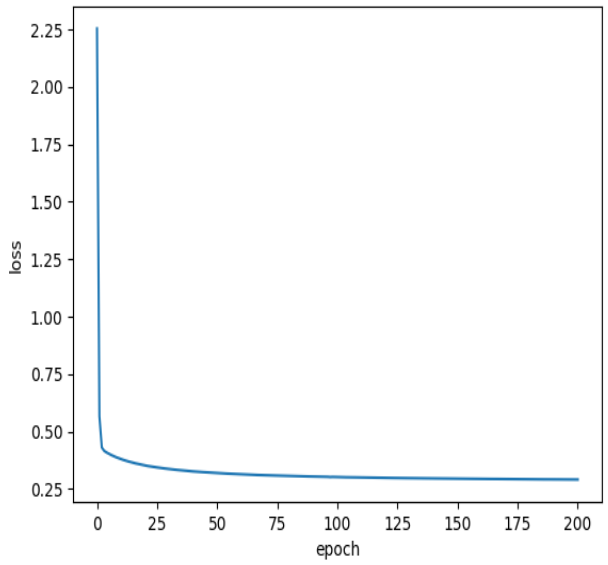
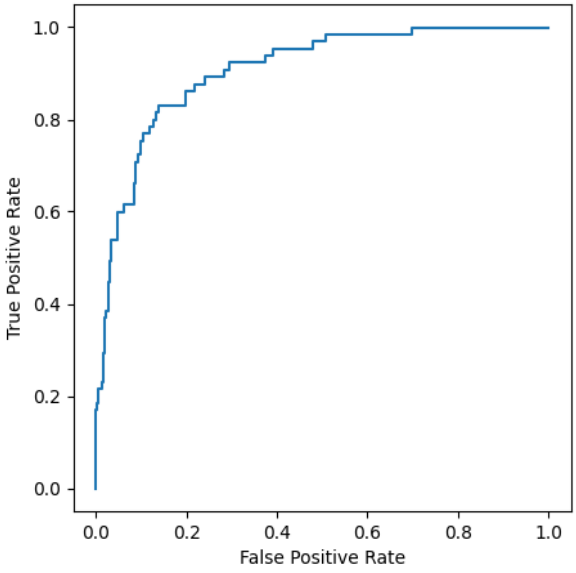
False	0.97	0.91	0.93	316
True	0.54	0.78	0.64	45
accuracy			0.89	361
macro avg	0.75	0.84	0.79	361
weighted avg	0.91	0.89	0.90	36

```
from sklearn.metrics import roc_auc_score
print('ROC auc value : {}'.format(roc_auc_score(y_test,layer.predict(x_test))))
#ROC auc value : 0.7523388773388773
```

분류 모델을 리포트한 결과 정확도는 0.89로 출력됐다. 또한 ROC auc value는 0.75이다.

3.3.4 시각화

```
# 손실 함수 누적값
plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()

# ROC curve
pro=layer.proba(x_test)
from sklearn.metrics import roc_curve
fprs, tprs, thresholds = roc_curve(y_test, pro,pos_label=1)
precisions, recalls, thresholds = roc_curve(y_test, pro,pos_label=1)

plt.figure(figsize=(5,5))
plt.plot(fprs,tprs,label='ROC')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

손실 함수 누적값 그래프	ROC curve
	

왼쪽 손실 누적 그래프를 보면, 로지스틱 손실 함수의 값(loss)이 에포크가 진행됨에 따라 감소하고 있음을 확인할 수 있다. 25 에포크 전에서 급격히 완만해지므로 25 에포크 이상을 진행하면 과적합이 발생할 수 있으므로 유의해야 한다. 오른쪽 ROC curve 의 모양을 살펴보았을 때 아래 영역의 넓이가 넓은 것을 확인할 수 있으며 AUC 값이 0.75로 모델의 성능이 우수하다고 볼 수 있다.

3.4 결론(비교)

R

	precision	recall	f1-score	support
False	0.96	0.90	0.93	318
True	0.49	0.74	0.59	43
accuracy			**0.88**	361
macro avg	0.73	0.82	0.76	361
weighted avg	0.91	0.88	0.89	361

python

	precision	recall	f1-score	support
False	0.97	0.91	0.93	316
True	0.54	0.78	0.64	45
accuracy			**0.89**	361
macro avg	0.75	0.84	0.79	361
weighted avg	0.91	0.89	0.90	361

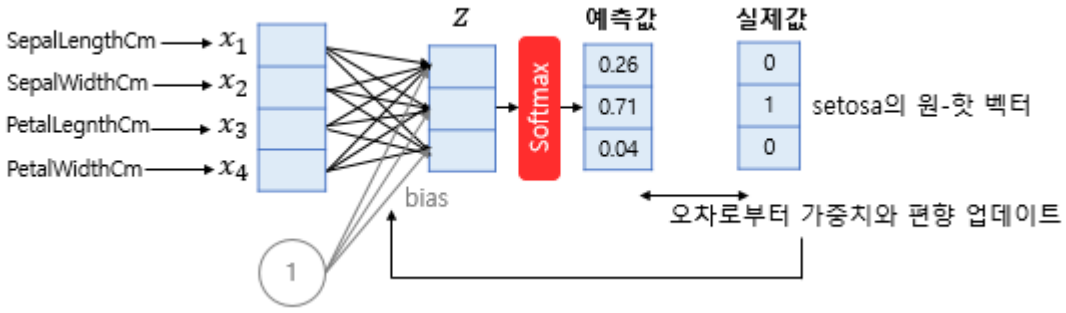
위 표는 비 예측을 위한 로지스틱 회귀분석을 각각 R(머신러닝)과 python(딥러닝)으로 진행한 결과를 비교한 것이다. 두 방법의 가장 큰 차이점은 feature의 추출을 누가 진행하느냐였다. R에서의 로지스틱 회귀분석은 변수 제거와 후진 제거법 등 feature를 추출하는 과정을 직접 지시함으로써 머신러닝의 특성에 따라 분석을 진행했다. 반면에 python에서는 딥러닝의 특성에 따랐는데, 로지스틱 회귀분석의 연산 과정에 따라 적절한 객체를 구성해 주면 feature 선정은 컴퓨터가 처리했다.

그리고 모델 평가에 따른 차이점을 보면, 정확도(accuracy)는 0.88과 0.89로 딥러닝에서 약 0.01% 정도 근소하게 향상된 모습을 보인다. 또한 precision, recall, f1-score, support 등 다른 값에서도 유사한 결과를 관찰할 수 있다.

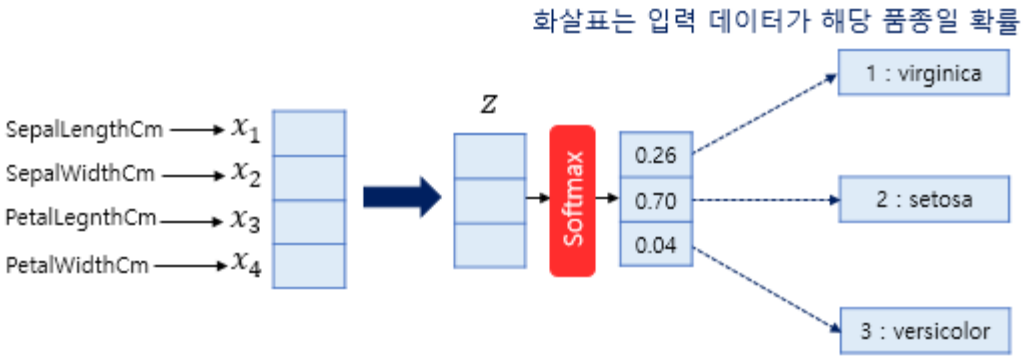
4. 다중 분류 분석 비교

4.1 개요

소프트맥스 회귀는 로지스틱 회귀 중에서도 다중 분류를 위한 회귀이다. 로지스틱 회귀가 0 또는 1, 성공 또는 실패 등의 두 개의 라벨을 분류한다면, 소프트맥스 회귀는 A 또는 B 또는 C처럼, 3개 이상의 라벨을 분류하는데 쓰일 수 있다.



소프트맥스 함수의 입력 벡터 z 의 차원수만큼 결과값이 나오도록(차원 축소) 가중치 곱을 진행한다. 위의 그림에서 화살표는 총 (4 × 3 = 12) 12개이며 전부 다른 가중치를 가지고, 학습 과정에서 점차적으로 오차를 최소화하는 가중치로 값이 변경된다.

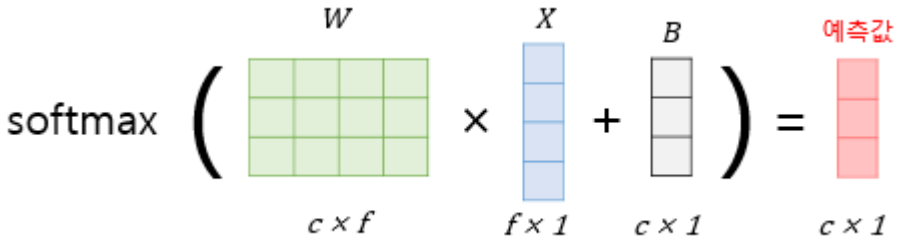


Z값은 합이 1이 넘기때문에 공정하게 비교하기 어렵다. 따라서 소프트맥스 함수를 적용하여 출력값을 정규화 하여 분류될 확률(예측값)을 구한다. 예측값과 실제값의 오차를 계산하기 위해서 소프트맥스 회귀는 비용 함수로 크로스 엔트로피 함수[6]를 사용한다.

소프트맥스 회귀를 벡터와 행렬 연산으로 생각해보면, 입력을 특성(feature)의 수만큼의 차원을 가진 입력 벡터 x라고 하고, 가중치 행렬을 W, 편향을 b라고 하였을 때, 소프트맥스 회귀에서 예측값(Z)을 구하는 과정을 벡터와 행렬 연산으로 표현하면 아래와 같다. 예측값(Z)의 차원수는

[6] L= ∑y log(a)=-(y1 log(a1)+y2 log(a2)+...+ y_n log(a)) =-log(a))
손실함수를 L을 미분하면 최종적으로 =-(y-a) 식을 얻을 수 있다. 이 식은 로지스틱 손실 함수의 미분과 결과가 같다.

타겟의 클래스 수와 일치해야 한다. softmax 함수의 f는 특성(feature)의 수이며 c은 클래스의 개수이다.



4.1.1 목표

다중분류를 위한 R을 이용하여 머신러닝 다중분류를 실시하고, 텐서플로와 케라스를 이용한 인공신경망을 python으로 실행하여 결과를 비교한다.

4.1.2 분석방법

정확한 비교를 위해 R과 Python에서 SyncRNG 함수를 이용해 서로 동일한 데이터로 나누었다. R에서는 랜덤포레스트 함수로 trees 값을 100으로 진행했고, Python에서는 'setosa', 'versicolor', 'virginica'를 각각 0, 1, 2로 변환하고 레이블인코딩 후 원-핫인코딩을 한다. 그 후 텐서플로와 케라스를 이용한 딥러닝을 진행하였다.

4.1.2 데이터 정의 및 탐색적 데이터 분석(EDA)

iris 데이터셋은 꽃잎의 각 부분의 너비와 길이들을 측정한 데이터이며 150개의 레코드로 구성되어 있다.

변수명	변수 설명
Sepal.Length	꽃받침의 길이
Sepal.Width	꽃받침의 너비
Petal.Length	꽃잎의 길이
Petal.Width	꽃잎의 너비
Species	꽃의 종류

> head(iris)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

> str(iris)

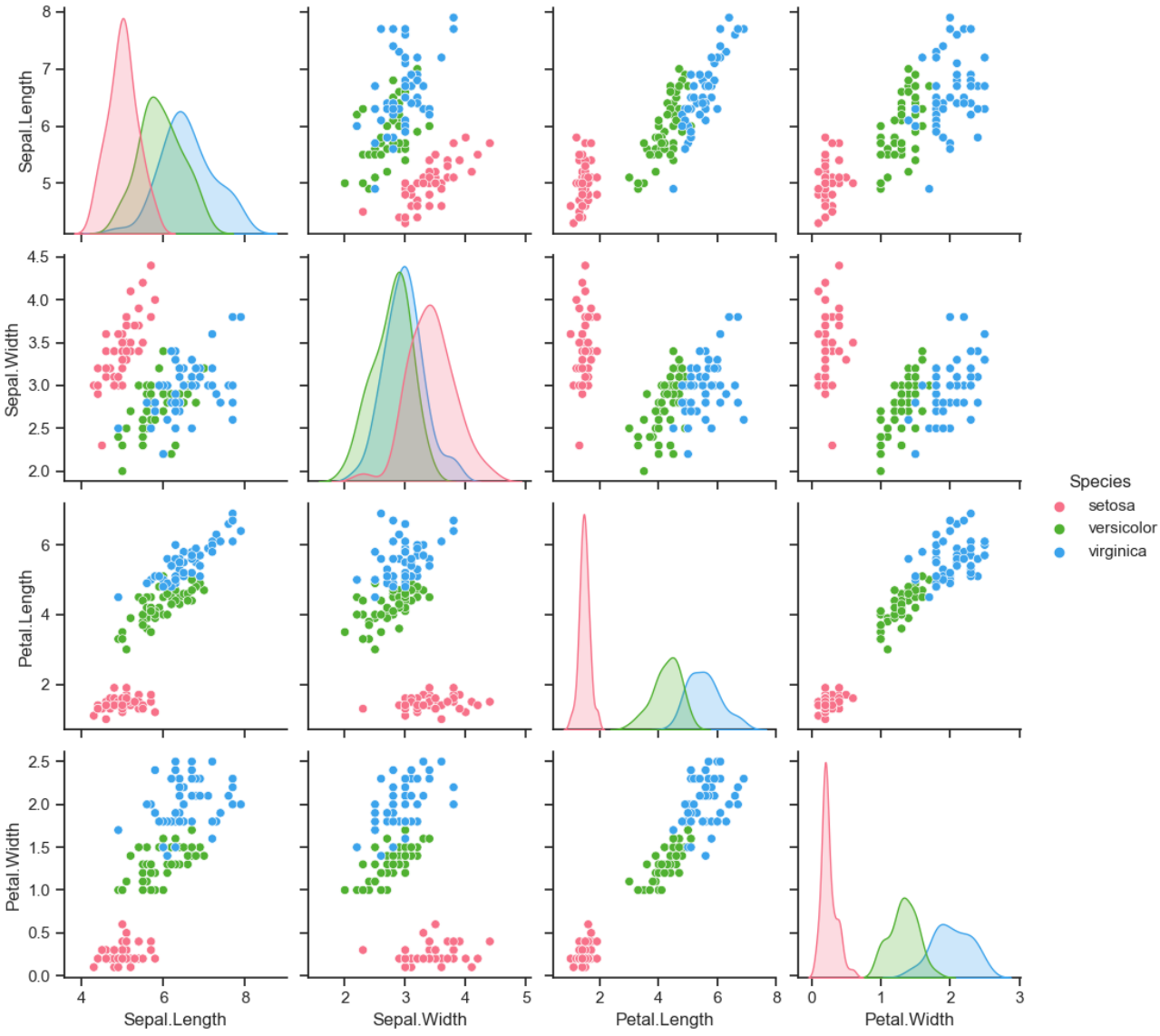
```
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

> summary(iris)

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

탐색적 데이터 분석 결과 결측치와 이상치를 발견할 수 없고 전처리가 필요한 변수도 없다.

```
import seaborn as sns
sns.pairplot(raw_data, hue="Species", size=3, diag_kind="kde")
```



▲ 4개의 특성에 해당하는 Sepal.Length, Sepal.Width, Petal.Length, Petal.Width에 대해서 모든 쌍(pair)의 조합인 16개의 경우에 대해서 산점도. 만약 동일한 특성의 쌍일 경우에는 히스토그램으로 나타냄.

4.2 R 다중분류 분석

4.2.1 모델 생성 및 예측

```
library(SyncRNG)
v <- 1:nrow(df)
s <- SyncRNG(seed=38)
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]
idx[1:length(idx)]
train <- df[idx,]
test <- df[-idx,]

m_rf<-randomForest(Species~.,train,ntree=100)
p_rf<-predict(m_rf, test[,-5])
```

R과 Python의 iris 데이터를 동일하게 나누기 위해 SyncRNG 함수를 사용하여 train,test 데이터셋을 만들고 트리수를 100개로 지정해 랜덤 포레스트 모델을 생성했다.

4.2.2 모델 평가

```
Confusion Matrix and Statistics

                Reference
Prediction    setosa versicolor virginica
  setosa          15          0         0
  versicolor       0         13         2
  virginica        0          2        13

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.8173, 0.986)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA
```

Statistics by Class:

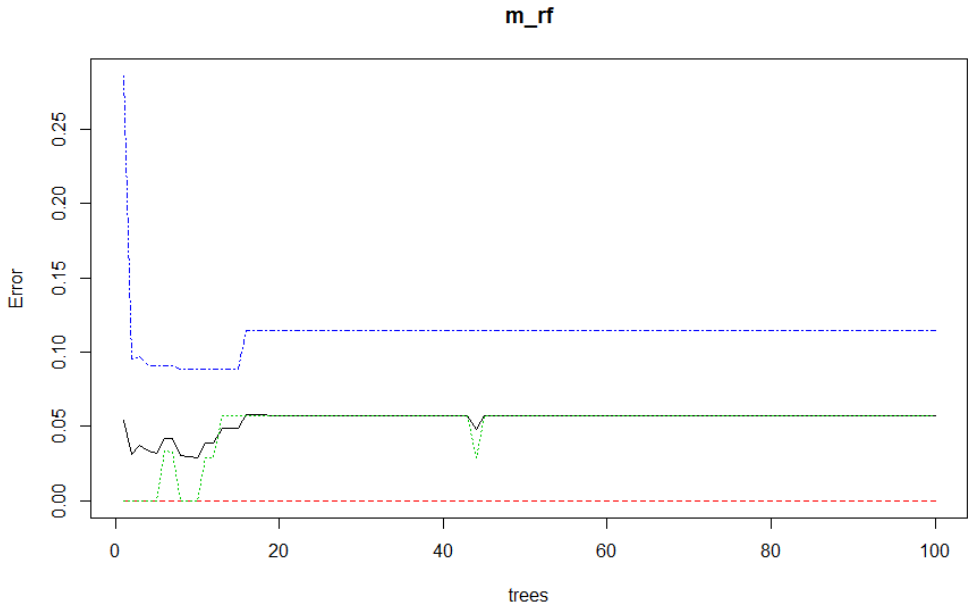
	Class: setosa	Class: versicolor	Class: virginica
Sensitivity	1.0000	0.8667	0.9333
Specificity	1.0000	0.9667	0.9333
Pos Pred Value	1.0000	0.9286	0.8750
Neg Pred Value	1.0000	0.9355	0.9655
Prevalence	0.3333	0.3333	0.3333
Detection Rate	0.3333	0.2889	0.3111
Detection Prevalence	0.3333	0.3111	0.3556
Balanced Accuracy	1.0000	0.9167	0.9333

predict 함수로 랜덤포레스트를 진행한 m_rf 변수과 test set을 이용하여 모델평가를 진행하였는데 여기서 정확도가 93%로 예측력이 매우 높은 모델임을 알 수 있다

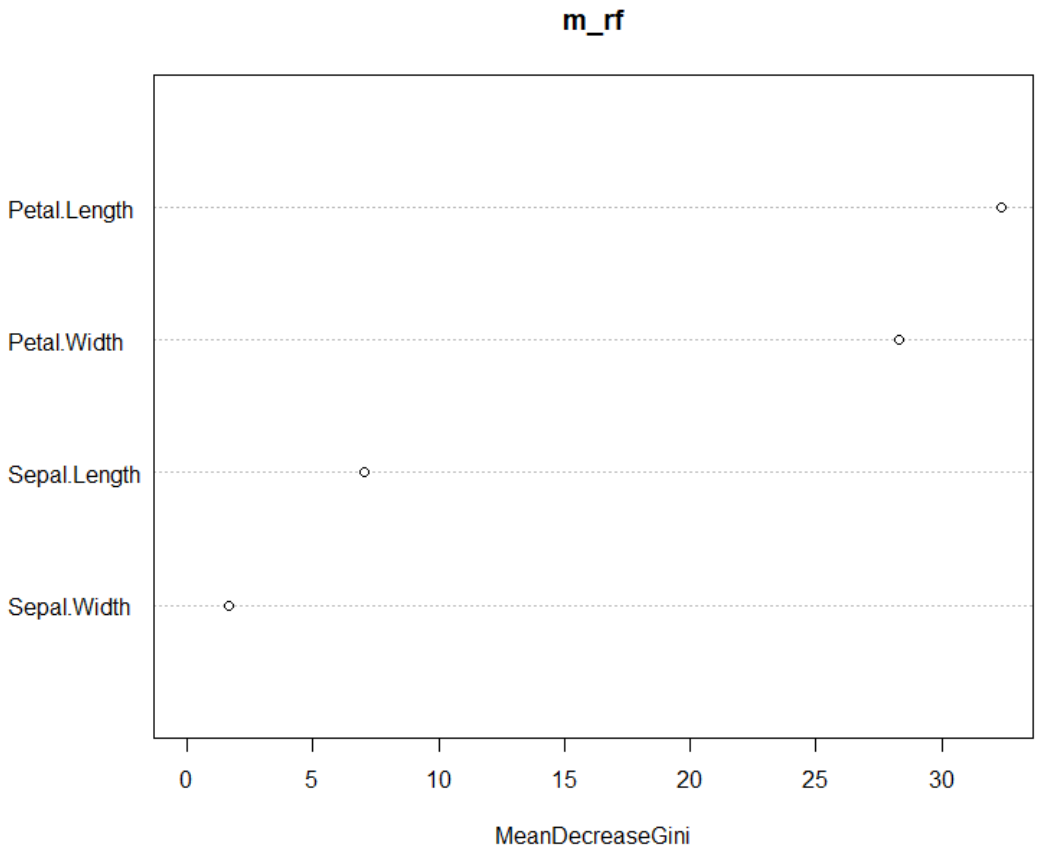
4.2.3 시각화

```
plot(m_rf)
randomForest::varImpPlot(m_rf)
```

plot 함수를 이용해서 모델을 시각화 했고 밑의 randomForest::varImpPlot() 함수를 이용해 랜덤포레스트 모형을 사용한 변수 중요도를 시각화 하였다



▲ 각각의 컬럼들의 트리수가 많아질수록 에러율이 적어지고, 트리수가 20개 이후로는 에러율이 일정한 것을 알 수 있다.



▲ 각 tree에서 해당 feature를 기준으로 분류하는 지점에서의 불순도의 총합을 계산후 모든 tree의 값들의 평균을 낸것을 표로 나타낸 것이다. 여기서 Petal.Length, Petal.Width가 모델이 분류를 하는데에 중요하게 작용되는 것들이다.

4.3 Python 다중분류 분석

4.3.1 데이터 호출

```
from SyncRNG import SyncRNG
import pandas as pd
from tensorflow import keras
import matplotlib.pyplot as plt
import tensorflow.compat.v1 as tf
import numpy as np
from sklearn.preprocessing import LabelEncoder


raw_data = pd.read_csv('E:/GoogleDrive/A5팀 프로젝트 자료(12월22일)/dataset/iris.csv')
```

4.3.2 데이터 전처리 및 샘플링

```
v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=38)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]


for i in range(0,len(idx)):
    idx[i]=idx[i]-1


train=raw_data.loc[idx] # 70%
test=raw_data.drop(idx) # 30%


x_train = np.array(train.iloc[:,0:4], dtype=np.float32)
y_train = np.array(train.Species.replace(['setosa','versicolor','virginica'],[0,1,2]))
x_test = np.array(test.iloc[:,0:4], dtype=np.float32)
y_test = np.array(test.Species.replace(['setosa','versicolor','virginica'],[0,1,2]))


y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_test_encoded = tf.keras.utils.to_categorical(y_test)
```

```
y_train_encoded.shape, y_test_encoded.shape

class_name=['Setosa','Versicolor','Virginica']

raw_data.Species.unique()
np.bincount(y_test)
np.bincount(y_train)

x_train.shape
x_test.shape
```

R과 Python의 난수 생성을 동일하게 하기 위해 SyncRNG 함수를 이용해서 교차검증용 train, test셋 7:3비율, Species기준으로 분할해서 각 종의 빈도수는 같다. 타겟의 명목형 변수를 숫자로 바꿔 의도하지 않은 의미가 부여되지 않도록 하고 의도한 범위 내에서만 값을 얻을 수 있도록 하기 위해 레이블인코딩을 진행 후 원-핫 인코딩을 실시해서 차원을 재구성 했다. 그래서 R에서의 Setosa, Versicolor, Virginica가 파이썬에서는 0, 1, 2로 표현되어 있음을 유의해야 한다.

4.3.3 케라스를 이용하여 신경망 생성

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(3, input_dim=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
history = model.fit(x_train, y_train_encoded, epochs=200, batch_size=5, validation_data=(x_test,
y_test_encoded))

print("₩n 테스트 loss: %.4f" % (model.evaluate(x_test, y_test_encoded)[0]))
print("₩n 테스트 accuracy: %.4f" % (model.evaluate(x_test, y_test_encoded)[1]))
predictions=model.predict(x_test)
```

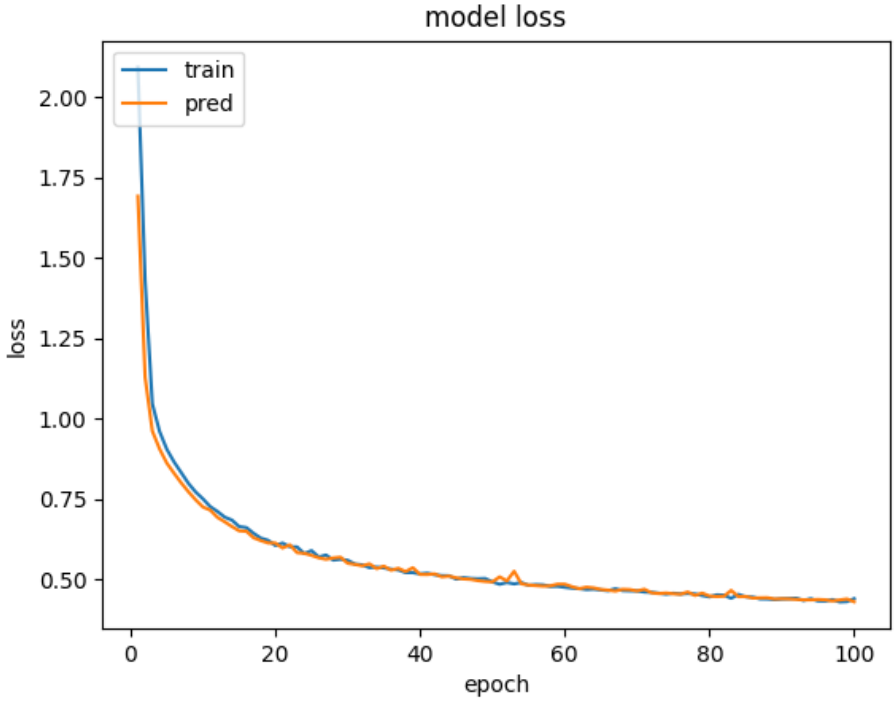
케라스를 이용해서 모델을 생성한다. 활성화 함수는 소프트맥스 함수를 이용하고 은닉층 갯수는 따로 설정하지는 않았다. 출력층 뉴런 갯수는 타겟의 클래스 갯수인 3개로 설정했고 소프트맥스 함수로 출력값을 분류한다. 오차 함수로는 크로스 엔트로피 함수를 사용하고 옵티마이저로는 경사 하강법의 일종인 SGD를 사용 했다.

```
테스트 loss: 0.1904
테스트 accuracy: 0.9778
array([[9.67251122e-01, 3.27430032e-02, 5.91728667e-06],.....
       [2.61039147e-03, 6.79958761e-01, 3.17430854e-01],.....
       [1.08743525e-05, 1.76265225e-01, 8.23723912e-01],..... ]]
```

출력층 뉴런 결과 값의 일부이다. 출력층 뉴런은 3개이고 출력층 뉴런 수는 타겟의 클래스 수와 일치한다. 첫 번째 행 [9.67251122e-01, 3.27430032e-02, 5.91728667e-06]의 순서대로 Setosa, Versicolor, Virginica 의 분류 확률이며 이는 모든 행에 동일하게 부여된다. 셋 중 가장 높은 확률값으로 분류를 하는데, 첫 번째 행에서는 9.67251122e-01 이 가장 높으므로 Setosa로 분류되고, 두 번째 행은 6.79958761e-01 이 가장 높으므로 Versicolor로 분류되고, 세 번째 행은 8.23723912e-01 이 가장 높으므로 Virginica로 분류된다.

4.3.6 시각화

```
print(history.history.keys())
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss','val_loss'])
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['accuracy','val_accuracy'])
plt.show()
```



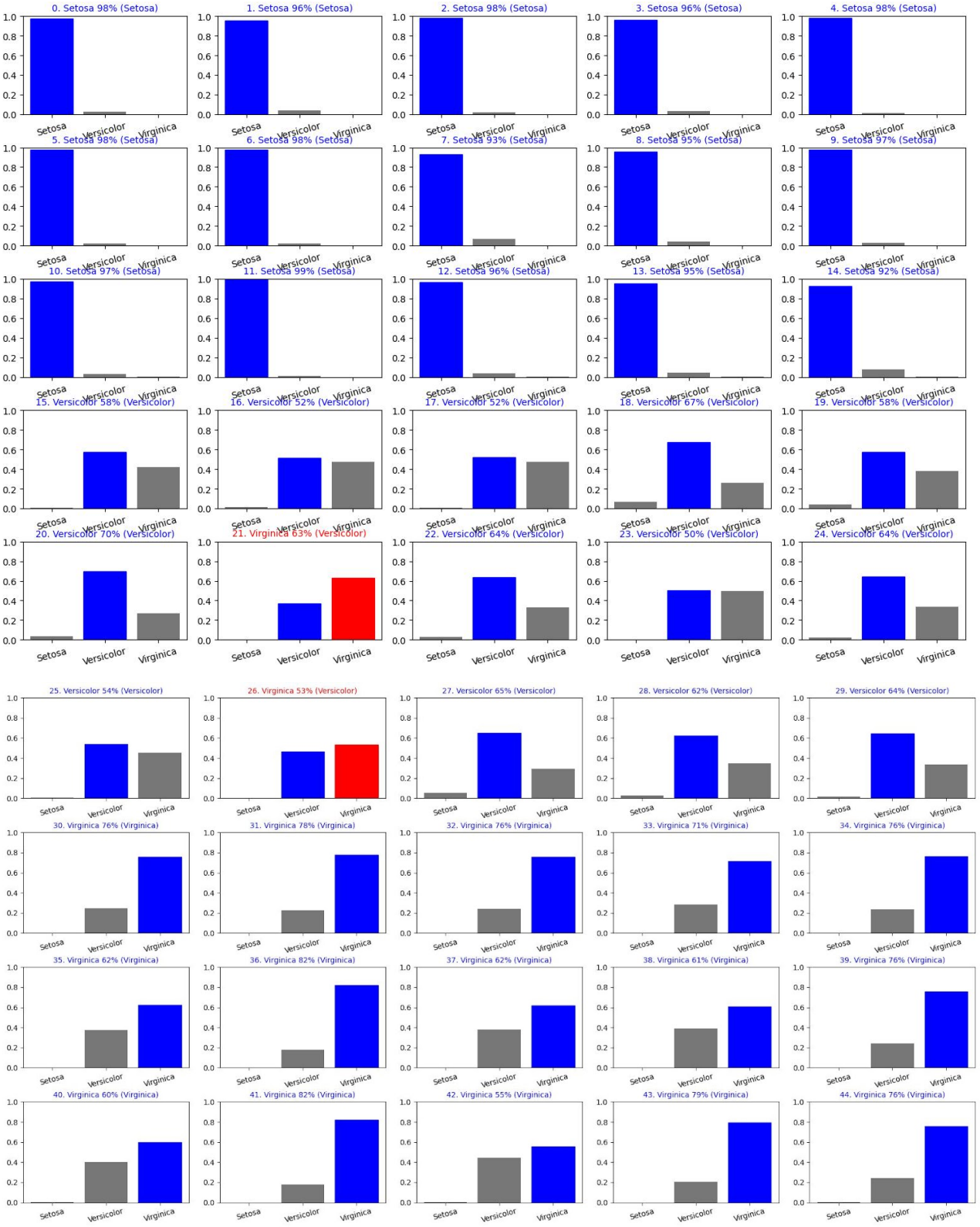
▲ **Epochs가 증가할수록 손실률은 감소하는 것을 볼 수 있다.**

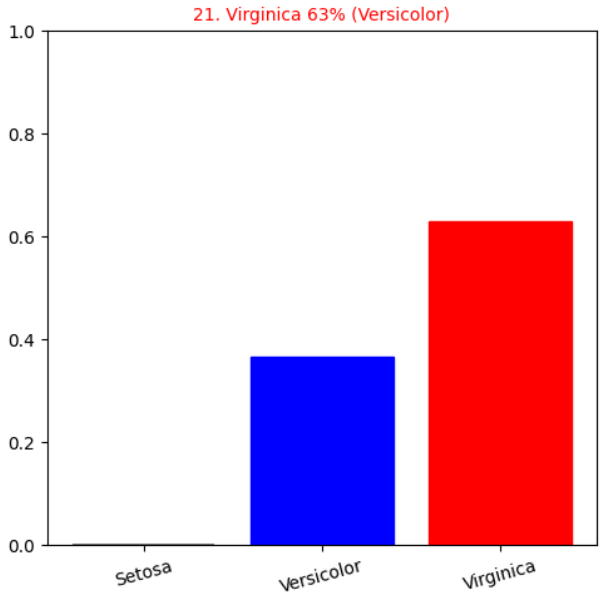
▲ Epochs가 증가할수록 정확도는 증가하는 것을 볼 수 있다. 하지만 약 50번째 Epochs 부터 더이상 증가하지 못하고 일정하게 유지되는 것을 볼 수 있는데 50개가 최적점으로 판단된다. 50개 이상부터는 과적합이 발생 할 수 있으니 유의해야 한다.

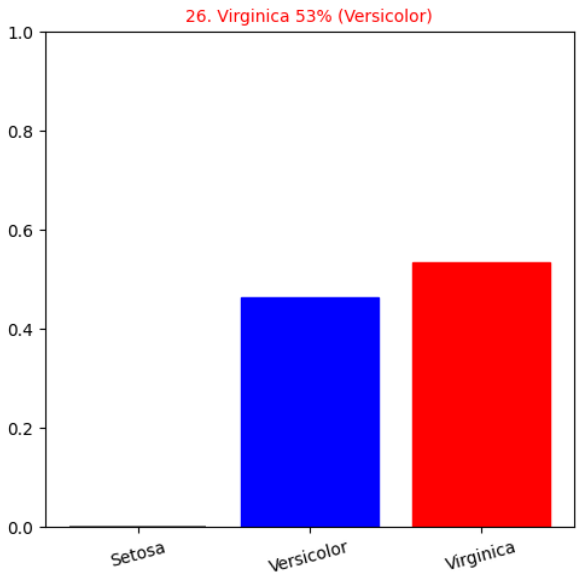
```
def plot_value_array(i, predictions_array, true_label):
 true_label = true_label[i]
 plt.grid(False)
 plt.xticks(range(3),class_names,rotation=15)
 thisplot = plt.bar(range(3), predictions_array, color="#777777")
 plt.ylim([0, 1])
 predicted_label = np.argmax(predictions_array)
 title_font = {
     'fontsize': 10,
 }
 if predicted_label == true_label:
   color = 'blue'
 else:
   color = 'red'
 plt.title("{}. {} {:2.0f}% ({})".format(i,class_names[predicted_label],
                                 100*np.max(predictions_array),
```

```
                                class_names[true_label]),
                                color=color,fontdict=title_font)
 thisplot[predicted_label].set_color('red')
 thisplot[true_label].set_color('blue')

predictions=model.predict(x_test)

#0~24
num_rows = 5
num_cols = 5
num_table = num_rows*num_cols
plt.figure(figsize=(2*num_cols, 2*num_rows))
for i in range(num_table):
    plt.subplot(num_rows, num_cols,i+1)
    plot_value_array(i, predictions[i], y_test)
plt.tight_layout()
plt.show()

#25~44
num_rows = 4
num_cols = 5
num_table = num_rows*num_cols
plt.figure(figsize=(2*num_cols, 2*num_rows))
for i in range(num_table):
    plt.subplot(num_rows, num_cols,i+1)
    plot_value_array(i+25, predictions[i+25], y_test)
    if i>=19:
        break
plt.tight_layout()
plt.show()
```

사용자 정의 클래스를 생성하여 테스트 셋의 예측 결과를 시각화했다. 가독성을 위해 25개씩 출력하였다.


0. Setosa 98% (Setosa)
1. Setosa 96% (Setosa)
2. Setosa 98% (Setosa)
3. Setosa 96% (Setosa)
4. Setosa 98% (Setosa)
5. Setosa 98% (Setosa)
6. Setosa 98% (Setosa)
7. Setosa 93% (Setosa)
8. Setosa 95% (Setosa)
9. Setosa 97% (Setosa)
10. Setosa 97% (Setosa)
11. Setosa 99% (Setosa)
12. Setosa 96% (Setosa)
13. Setosa 95% (Setosa)
14. Setosa 92% (Setosa)
15. Versicolor 58% (Versicolor)
16. Versicolor 52% (Versicolor)
17. Versicolor 52% (Versicolor)
18. Versicolor 67% (Versicolor)
19. Versicolor 58% (Versicolor)
20. Versicolor 70% (Versicolor)
21. Virginica 63% (Versicolor)
22. Versicolor 64% (Versicolor)
23. Versicolor 50% (Versicolor)
24. Versicolor 64% (Versicolor)
25. Versicolor 54% (Versicolor)
26. Virginica 53% (Versicolor)
27. Versicolor 65% (Versicolor)
28. Versicolor 62% (Versicolor)
29. Versicolor 64% (Versicolor)
30. Virginica 76% (Virginica)
31. Virginica 78% (Virginica)
32. Virginica 76% (Virginica)
33. Virginica 71% (Virginica)
34. Virginica 76% (Virginica)
35. Virginica 62% (Virginica)
36. Virginica 82% (Virginica)
37. Virginica 62% (Virginica)
38. Virginica 61% (Virginica)
39. Virginica 76% (Virginica)
40. Virginica 60% (Virginica)
41. Virginica 82% (Virginica)
42. Virginica 55% (Virginica)
43. Virginica 79% (Virginica)
44. Virginica 76% (Virginica)
Setosa
Versicolor
Virginica





테스트 셋의 예측 결과를 막대그래프로 시각화했다. X축은 종속변수의 클래스, Y축은 확률 값이다. 즉, 총 45개의 막대 그래프는 테스트 셋이 Setosa, Versicolor, Virginica 중 하나로 분류될 확률을 나타낸다. 이때 정확하게 분류된 것은 파란색으로, 잘못 분류된 것은 빨간색으로 표시했다. 45개의 테스트 셋 중 43개가 정답을 맞추고 2개가 오답이었는데, 각각 63%, 53%의 확률로 Virginica로 잘못 분류된 것을 확인할 수 있다.

4.4 결론(비교)

R

	Sensitivity	Specificity	Pos	Pred Value	Neg Pred Value
Class: setosa	1.000	1.000	1.000	1.000	1.000
Class: versicolor	0.866	1.000	1.000	0.937	1.000
Class: virginica	1.000	0.933	0.882	1.000	0.882

	Precision	Recall	F1	Prevalence
Class: setosa	1.000	1.000	1.000	0.333
Class: versicolor	1.000	0.866	0.928	0.333
Class: virginica	0.882	1.000	0.937	0.333

	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: setosa	0.333	0.333	1.000
Class: versicolor	0.288	0.288	0.933
Class: virginica	0.333	0.377	0.966

Python

	precision	recall	f1-score	support
0	1.00	1.00	1.00	15
1	0.93	1.00	0.97	14
2	1.00	0.94	0.97	16
accuracy		0.98	45	
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

위 표는 R과 Python에서 다중 분류 분석을 진행한 결과를 나타낸 것이다. R과 Python의 F1 score는 Setosa 1.0, 1.0, versicolor 0.92, 0.97 virginica 0.93, 0.97로 파이썬이 조금더 좋은 점수를 얻었다. 정확성(accuracy)은 R은 각각 1.0, 0.93, 0.96이고 Python은 0.98이므로 전체적으로 Python이 좋은 결과를 냈다.

5. 출처 및 참고자료

5.1 머신러닝과 딥러닝 비교

1	인공지능·머신러닝·딥러닝 차이점	https://www.codestates.com/blog/content/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%EB%94%A5%EB%9F%AC%EB%8B%9D%EA%B0%9C%EB%85%90
2	딥 러닝 대 기계 학습	https://learn.microsoft.com/ko-kr/azure/machine-learning/concept-deep-learning-vs-machine-learning
3	머신러닝 데이터 모델 이해	https://velog.io/@raffier/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EB%AA%A8%EB%8D%B8

5.2 선형 회귀 분석 비교

1	신경망과 회귀분석 비교	https://statisticsplaybook.tistory.com/14
2	텐서플로 회귀모형	https://gooopy.tistory.com/101
3	회귀 네트워크 만들기	https://davinci-ai.tistory.com/27
4	ggplot회귀식 표시하기	https://kuduz.tistory.com/1118
5	손실함수 미분	https://hayden-archive.tistory.com/303
6	머신러닝 선형회귀	https://velog.io/@supremo7/%EB%A8%B8%EC%8B%A0-%EB%9F%AC%EB%8B%9D-%EC%84%A0%ED%98%95-%ED%9A%8C%EA%B7%80Linear-Regression
7	선형회귀 손실함수	https://brunch.co.kr/@cookery/99
8	실무에 활용하는 딥러닝	http://www.kocw.net/home/cview.do?mty=p&kemId=1380203

5.3 로지스틱 회귀 분석 비교

1	딥러닝 로지스틱 회귀 분석	https://m.blog.naver.com/view-view/221774416395
2	파이썬 로지스틱회귀 분석	https://returnclass.tistory.com/m/29
3	결과 시각화	https://bioinformaticsandme.tistory.com/296
4	EDA on R with GLM	https://www.kaggle.com/code/mbkinaci/comprehensive-eda-on-r-with-logistic-regression/notebook

5.4 다중 분류 분석 비교

1	다중 분류 신경망	https://returnclass.tistory.com/m/35
2	Do it여러개의 이미지를 분류하는 다층 신경망	https://soohwan-justin.tistory.com/59
3	Stacked Models	https://bradleyboehmke.github.io/HOML/stacking.html
4	How To Build Stacked Ensemble Models	https://predictivehacks.com/how-to-build-stacked-ensemble-models-in-r/
5	케라스 모델 저장 및 로드	https://www.tensorflow.org/guide/keras/save_and_serialize#registering_the_custom_object
6	소프트맥스 회귀	https://wikidocs.net/35476
7	텐서플로 이미지분류	https://www.tensorflow.org/tutorials/keras/classification

6. 소스코드

6.1 선형 회귀 분석

1) R

```r
library(SyncRNG)
library(ggplot2)
library(caret)
library(Metrics)
#데이터 호출
df <- read.csv("product.csv")
summary(df)
str(df)
#분포 막대그래프
barplot(table(df$제품_만족도))
barplot(table(df$제품_적절성))
#모델생성
v <- 1:nrow(df)
s <- SyncRNG(seed=42)
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]
idx[1:length(idx)]
train <- df[idx,]
test <- df[-idx,]
m_lm<- lm(제품_만족도~제품_적절성, train)
m_lm
#모델 요약확인
summary(m_lm)
#모델평가
p_lm<-predict(m_lm,test[,-1])
RMSE(p_lm,test[,3])
R2(p_lm,test[,3])
#시각화
ggplot(train,aes(x=제품_적절성,y=제품_만족도))+
  geom_count()+
  scale_size_area(max_size = 15)+
  stat_smooth(method='lm',color='red')+
  geom_text(x=4.3,y=3.5,label="제품_만족도=0.76x제품_적절성+0.72")+
  geom_text(x=4,y=3.3,label="R²=0.62")
```

2) Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from SyncRNG import SyncRNG
import math
from sklearn.metrics import mean_squared_error, r2_score

#데이터 호출
raw_data = pd.read_csv('E:/GoogleDrive/DATASET/dataset/product.csv',encoding='cp949')

v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=42)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]

for i in range(0,len(idx)):
    idx[i]=idx[i]-1

train=raw_data.loc[idx]
test=raw_data.drop(idx)

x_train = train.제품_적절성
y_train = train.제품_만족도
x_test = test.제품_적절성
y_test = test.제품_만족도
#모델생성
class Neuron:
    def __init__(self):
        self.w = 1  # 가중치를 초기화합니다
        self.b = 1 # 절편을 초기화합니다

    def forpass(self, x):
        y_hat = x * self.w + self.b  # 직선 방정식을 계산합니다
        return y_hat
```

```
    def backprop(self, x, err):
        w_grad = x * err  # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err  # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad

    def fit(self, x, y,lr, epochs=400):
        for i in range(epochs):  # 에포크만큼 반복합니다
            for x_i, y_i in zip(x, y):  # 모든 샘플에 대해 반복합니다
                n=len(x)
                y_hat = self.forpass(x_i)  # 정방향 계산
                err = -(2/n)*(y_i - y_hat)  # 오차 계산
                w_grad, b_grad = self.backprop(x_i, err)  # 역방향 계산
                self.w -= w_grad*lr  # 가중치 업데이트
                self.b -= b_grad*lr # 절편 업데이트

neuron = Neuron()
neuron.fit(x_train, y_train,0.1)
print(neuron.w)
print(neuron.b)
#모델 예측 및 평가
predict=[]
predict = x_test * neuron.w + neuron.b
mse=mean_squared_error(predict, y_test)
math.sqrt(mse)
r2_score(y_test, predict)
#시각화
plt.scatter(x_train, y_train)
plt.scatter(x_test, predict)
pt1 =(1, 1*neuron.w+neuron.b)
pt2 =(5, 5*neuron.w+neuron.b)
plt.plot([pt1[0],pt2[0]],[pt1[1],pt2[1]],color='orange')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

6.2 로지스틱 회귀분석

1) R 소스코드

```r
#2번
library(car)
library(caret)
library(ModelMetrics)
library(corrplot)
df<-read.csv("E:/GoogleDrive/DATASET/dataset4/weather.csv")

#EDA
head(df)
str(df)
summary(df)

df<-na.omit(df) #결측치 제거
cordf<-df[sapply(df,is.numeric)]  #numeric컬럼만 cordf에 넣기
M <- cor(cordf)
corrplot(M, method="circle")
#Temp와minTemp, maxTemp 상관관계가 매우 크다. 다중공선성 가능성 큼

#결측치 제거, 전처리
df<-df[,-1] #날짜 제거
df<-na.omit(df) #결측치 제거
p<-preProcess(df,"range") #모든 숫자형변수 범위0~1 최소-최대 정규화
df<-predict(p,df)

#모델 생성후 다중 공선성 문제로 다시 전처리 부분
df<-df[,-c(1,2,5,7)]
##전처리 후 다중공선성
# Rainfall      Sunshine  WindGustSpeed     WindSpeed      Humidity      Pressure         Cloud
# FALSE         FALSE         FALSE         FALSE         FALSE         FALSE         FALSE
# Temp     RainToday
# FALSE         FALSE

#모델생성
v <- 1:nrow(df)
s <- SyncRNG(seed=42)
```

```
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]

train<-df[idx,]
test<-df[-idx,]
m_glm<-glm(RainTomorrow~.,train,family = 'binomial')
# Warning message:
# glm.fit: 적합된 확률값들이 0 또는 1 입니다
# 원인 다중공선성
# 다중공선성 변수 제거해야함
vif(m_glm)
summary(m_glm)


##유의한 변수 선택
#후진제거법으로 로지스틱회귀모형을 새롭게 정의
m_glm=step(m_glm, direction = "backward")
summary(m_glm)

p_glm<-predict(m_glm,test,type='response')
p.glm<-ifelse(p_glm>=0.5,'Yes','No')

#모델평가
table(test$RainTomorrow)
table(p.glm)

caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)
caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)$byClass
caret::confusionMatrix(as.factor(p.glm),test$RainTomorrow)$overall
auc(as.factor(p.glm),test$RainTomorrow)
```

2) Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from SyncRNG import SyncRNG

raw_data = pd.read_csv('E:/GoogleDrive/DATASET/dataset/weather.csv',encoding='cp949')
raw_data.head()
#날짜 제거
raw_data.drop(['Date'],axis=1,inplace=True)

#범주형변수 레이블인코더
le = LabelEncoder()
cat_cols = ['WindGustDir', 'WindDir', 'RainToday','RainTomorrow']
raw_data[cat_cols] = raw_data[cat_cols].apply(le.fit_transform)

#각 변수간 범위가 전부 다르기 때문에 최소-최대 정규화로 스케일링
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

cols = ['MinTemp', 'MaxTemp', 'Rainfall', 'Sunshine', 'WindGustDir',
       'WindGustSpeed', 'WindDir', 'WindSpeed', 'Humidity', 'Pressure',
       'Cloud', 'Temp', 'RainToday', 'RainTomorrow']
raw_data[cols] = scaler.fit_transform(raw_data[cols])

# 데이터 셋  7:3 으로 분할
v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=42)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]

for i in range(0,len(idx)):
    idx[i]=idx[i]-1

# 학습데이터, 테스트데이터 생성
train=raw_data.loc[idx] # 70%
```

```
test=raw_data.drop(idx) # 30%

#결측치 제거
train=raw_data.dropna()
test=raw_data.dropna()
#np.sum(train.isnull(),axis=0)

x_train = np.array(train.iloc[:,0:13], dtype=np.float32)
y_train = np.array(train.iloc[:,-1], dtype=np.float32)
x_test = np.array(test.iloc[:,0:13], dtype=np.float32)
y_test = np.array(test.iloc[:,-1], dtype=np.float32)

x_train.shape, x_test.shape, y_train.shape,y_test.shape


class SingleLayer:

    def __init__(self):
        self.w = None
        self.b = None
        self.losses = []

    def forpass(self, x):
        z = np.sum(x * self.w) + self.b  # 직선 방정식을 계산합니다
        return z

    def backprop(self, x, err):
        w_grad = x * err  # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err  # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad

    def activation(self, z):
        z = np.clip(z, -100, None)  # 안전한 np.exp() 계산을 위해
        a = 1 / (1 + np.exp(-z))  # 시그모이드 계산
        return a

    def fit(self, x, y,lr, epochs=401):
```

```
        self.w = np.ones(x.shape[1])  # 가중치를 초기화합니다.
        self.b = 0  # 절편을 초기화합니다.

        for j in range(epochs):  # epochs만큼 반복합니다
            loss = 0
            # 인덱스를 섞습니다
            indexes = np.random.permutation(np.arange(len(x)))
            for i in indexes:  # 모든 샘플에 대해 반복합니다
                n=len(x)
                z = self.forpass(x[i])  # 정방향 계산
                a = self.activation(z)  # 활성화 함수 적용
                err = -(2/n)*(y[i] - a)  # 오차 계산
                w_grad, b_grad = self.backprop(x[i], err)  # 역방향 계산

                self.w -= w_grad*lr  # 가중치 업데이트
                self.b -= b_grad*lr # 절편 업데이트
                # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다
                a = np.clip(a, 1e-10, 1 - 1e-10)
                loss += -(y[i] * np.log(a) + (1 - y[i]) * np.log(1 - a))
            # 에포크마다 평균 손실을 저장합니다
            self.losses.append(loss / len(y))
            if j % 100 == 0:
             print('[',j,']',err,self.b,self.w)

    def predict(self, x):
        z = [self.forpass(x_i) for x_i in x]  # 정방향 계산
        return np.array(z) > 0  # 스텝 함수 적용

    def proba(self,x):  #ROC 커브용 확률출력 메서드
        z = [self.forpass(x_i) for x_i in x]
        return np.array(z)

    def score(self, x, y):
        return np.mean(self.predict(x) == y)

layer = SingleLayer()
layer.fit(x_train, y_train, 1)
```

```
print(layer.score(x_test, y_test))
layer.predict(x_test)
from sklearn.linear_model import LogisticRegression
#print(inspect.getsource(LogisticRegression))
log=LogisticRegression()
log.fit(x_train, y_train)
print(log.score(x_test, y_test))

plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()

from sklearn.metrics import roc_auc_score
print('ROC auc value : {}'.format(roc_auc_score(y_test,layer.predict(x_test)))) # auc에 대한 면적
#ROC auc value : 0.7523388773388773

from sklearn.metrics import classification_report
print(classification_report(layer.predict(x_test) , y_test))
#               precision    recall  f1-score   support
#        False       0.97      0.91      0.93       316
#         True       0.54      0.78      0.64        45
#     accuracy                           0.89       361
#    macro avg       0.75      0.84      0.79       361
# weighted avg       0.91      0.89      0.90       361

#시각화
##ROC커브 시각화
pro=layer.proba(x_test)
from sklearn.metrics import roc_curve
import inspect

fprs, tprs, thresholds = roc_curve(y_test, pro,pos_label=1)
precisions, recalls, thresholds = roc_curve(y_test, pro,pos_label=1)

# ROC
plt.figure(figsize=(5,5))
```

```
plt.plot(fprs,tprs,label='ROC')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

6.3 다중 분류 분석

1) R

```
library(randomForest)
library(SyncRNG)
library(ggplot2)
library(caret)
library(ModelMetrics)
data(iris)
head(iris)
str(iris)
summary(iris)
df<-iris
table(df$Species)

v <- 1:nrow(df)
s <- SyncRNG(seed=38)
idx <- s$shuffle(v)[1:round(nrow(df)*0.7)]
train<-df[idx,]
test<-df[-idx,]
m_rf<-randomForest(Species~.,train,ntree=100,random_state=0)
plot(m_rf)

randomForest::varImpPlot(m_rf)
p_rf<-predict(m_rf, test[,-5])
caret::confusionMatrix(p_rf,test$Species)
caret::confusionMatrix(p_rf,test$Species)$byClass
```

2) Python

```
import tensorflow.compat.v1 as tf
from SyncRNG import SyncRNG
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 데이터 불러오기
raw_data = pd.read_csv('E:/GoogleDrive/DATASET/dataset/iris.csv')
class_names = ['Setosa','Versicolor','Virginica']

#데이터 시각화
import seaborn as sns
sns.pairplot(raw_data, hue="Species", size=2, diag_kind="kde")

# 데이터 셋  7:3 으로 분할
v=list(range(1,len(raw_data)+1))
s=SyncRNG(seed=38)
ord=s.shuffle(v)
idx=ord[:round(len(raw_data)*0.7)]

for i in range(0,len(idx)):
    idx[i]=idx[i]-1

# 학습데이터, 테스트데이터 생성
train=raw_data.loc[idx] # 70%
#train=train.sort_index(ascending=True)
test=raw_data.drop(idx) # 30%

x_train = np.array(train.iloc[:,0:4], dtype=np.float32)
y_train = np.array(train.Species.replace(['setosa','versicolor','virginica'],[0,1,2]))
x_test = np.array(test.iloc[:,0:4], dtype=np.float32)
y_test = np.array(test.Species.replace(['setosa','versicolor','virginica'],[0,1,2]))

y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_test_encoded = tf.keras.utils.to_categorical(y_test)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


model = Sequential()
model.add(Dense(3, input_dim=4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train_encoded, epochs=100, batch_size=1, validation_data=(x_test,
y_test_encoded))

epochs = range(1, len(history.history['accuracy']) + 1)
plt.plot(epochs, history.history['loss'])
plt.plot(epochs, history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'pred'], loc='upper left')
plt.show()
print("₩n 테스트 loss: %.4f" % (model.evaluate(x_test, y_test_encoded)[0]))
print("₩n 테스트 accuracy: %.4f" % (model.evaluate(x_test, y_test_encoded)[1]))


import matplotlib.pyplot as plt
history_dict = history.history
history_dict.keys()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

# "bo"는 "파란색 점"입니다
plt.plot(epochs, loss, 'ro', label='Training loss')
# b는 "파란 실선"입니다
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.clf()   # 그림을 초기화합니다
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'y', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

###예측 시각화
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(3),class_names,rotation=15)

  thisplot = plt.bar(range(3), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
  title_font = {
      'fontsize': 10,
  }
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'


  plt.title("{}. {} {:2.0f}% ({})".format(i,class_names[predicted_label],
                                  100*np.max(predictions_array),
                                  class_names[true_label]),
                                  color=color,fontdict=title_font)
  thisplot[predicted_label].set_color('red')
```

```
  thisplot[true_label].set_color('blue')

predictions=model.predict(x_test)

####0~24
num_rows = 5
num_cols = 5
num_table = num_rows*num_cols
plt.figure(figsize=(3*num_cols, 3*num_rows))
for i in range(num_table):
    plt.subplot(num_rows, num_cols,i+1)
    plot_value_array(i, predictions[i], y_test)
plt.tight_layout()
plt.show()


####25~44
num_rows = 4
num_cols = 5
num_table = num_rows*num_cols
plt.figure(figsize=(3*num_cols, 3*num_rows))
for i in range(num_table):
    plt.subplot(num_rows, num_cols,i+1)
    plot_value_array(i+25, predictions[i+25], y_test)
    if i>=19:
        break
plt.tight_layout()
plt.show()

from sklearn.metrics import classification_report
result=[np.argmax(x) for x in predictions]
print(classification_report( result, y_test))
```