R과 Python을 통한 딥러닝 분석

2022.12.21

C5팀(권호영, 신정훈, 정성수, 조아진)

목차

1. 분석 개요

1. 제품적절성이 제품만족도에 미치는 영향 주제로 R을 이용한 단순 선형 회귀분석을 실시하고, 딥러닝 교재 3장에서 사용된 인공신경망을 이용한 선형 회귀분석을 python으로 실행하여 결과를 비교하시오.

- R과 python에서 같은 product.csv 파일을 가지고 분석을 수행하였다. 해당 파일은 제품 친밀도, 적절성, 만족도 컬럼을 가지고 있는데, 분석에 필요한 적절성과 만족도 컬럼만을 추출하여 사용하였다. R에서는 lm()함수를 사용해서 회귀 모델을 만들었다면, python에서는 교재 내용을 바탕으로 경사 하강법 알고리즘 Neuron class를 생성하여 모델을 만들고 분석을 진행하였다. 이때 경사 하강법은 모델이 데이터를 잘 표현할 수 있도록 기울기를 사용하여 모델을 조금씩 조정하는 최적화 알고리즘이다.

2. 비 유무 예측 주제로 R을 이용한 로지스틱 회귀분석을 실시하고, 딥러닝 교재 4장에서 사용된 인공신경망을 이용한 로지스틱 회귀분석을 python으로 실행하여 결과를 비교하시오.

- 두 언어에서 같은 weather.csv 파일로 분석을 수행하고 ROC 그래프를 그렸다. 먼저 RainTomorrow라는 컬럼을 제외한 문자형 데이터를 가진 컬럼들과 결측값을 가지는 행을 삭제하였다. 그리고 올바른 모델의 성능 측정을 위해 훈련과 테스트 데이터로 나누어 분석한 후 정확도를 계산하였다. R에서는 glm() 함수를 사용해 로지스틱 회귀 모델을 생성하고, python에서는 단일층 신경망 SingleLayer() class를 생성하여 모델을 만들어냈다. 이때 에포크마다 훈련 세트를 무작위로 섞어 손실 함수의 값을 줄였다.

3. 다중분류를 위한 머신러닝 기법을 선택하고 R을 이용하여 다중분류를 실시하고, 딥러닝 교재 7장에서 사용된 텐서플로와 케라스를 이용한 인공신경망을 python으로 실행하여 결과를 비교하시오.

- 각 언어의 iris 데이터 패키지를 사용해 R에서는 랜덤포레스트 기법으로 다중분류를 실시했고, python에서는 텐서플로와 케라스를 활용해 분류를 실행하였다. 분류 문제에서 정확도를 직접 최적화할 수 없어 크로스 엔트로피 손실 함수를 최적화하였다. 손실함수를 최적화한다고 정확도 역시 높아지는 것은 아니지만 비교적 반비례하는 관계의 두 그래프를 얻을 수 있었다.

2. 선형 회귀분석 실행 및 결과 비교

2-1 데이터셋 설명

'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 ...

특정 제품에 대한 사람들의 친밀도, 적절도, 만족도를 숫자형태로 기입한 데이터이다.

메모 포함[1]: 데이터 셋 설명, 저처리 순서, 사용패키지 정리, 코드 정리, 개요 추가, 각문제 정리, references, 소스코드

메모 포함[2]: 각문제 정리는 결론을 뜻함

메모 포함[3]: 개요 추가, 결론1, 2, 3(조아진)
3코드마무리, 소스코드 (권호영)
패키지 모으기, 보고서 양식 최종 정리 (정성수)
전처리순서, 코드정리(신정훈)

2-2 R을 통한 선형 회귀분석

분석 순서
데이터 불러오기
-> 데이터 전처리
-> 단순선형 회귀 모델 생성
-> 선형 회귀모델 시각화

2-2-1 데이터 불러오기

```
product <- read.csv("dataset4/product.csv")
head(product)
```

```
> head(product)
제품_친밀도 제품_적절성 제품_만족도
1          3          4          3
2          3          3          2
3          4          4          4
4          2          2          2
5          2          2          2
```

```
6          3          3          3
```

2-2-2 데이터 전처리

```
x = product$제품_적절성
y = product$제품_만족도
df <- data.frame(x, y)
```

분석에 필요한 컬럼들을 x, y 변수에 담은 후, 구조를 데이터프레임으로 변경하였다.

2-2-3 단순 선형 회귀 모델 생성

```
lm <- lm(formula = y ~ x, data = product)
summary(lm)
```

```
Call:
lm(formula = y ~ x, data = product)

Residuals:
     Min       1Q   Median       3Q      Max
-1.99669 -0.25741  0.00331  0.26404  1.26404

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.77886    0.12416   6.273 1.45e-09 ***
x            0.73928    0.03823  19.340  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5329 on 262 degrees of freedom
Multiple R-squared:  0.5881,    Adjusted R-squared:  0.5865
F-statistic:   374 on 1 and 262 DF,  p-value: < 2.2e-16
```

회귀분석 결과, Adjusted R-squared값이 0.5865 이기 때문에 제품 적절성(x)이 제품 만족도(y)에 대하여 58.65%를 설명할 수 있다는 것을 알 수 있다. 결정계수는 어떤 분야인가에 따라 연구자의 판단에 따라 차이가 존재하므로 몇퍼센트 이상이 절대적으로 유용한지 말할 수 없다. 이 분석에서는 절반 이상 설명이 가능했기 때문에 어느정도 유용하다고 판단하였다.

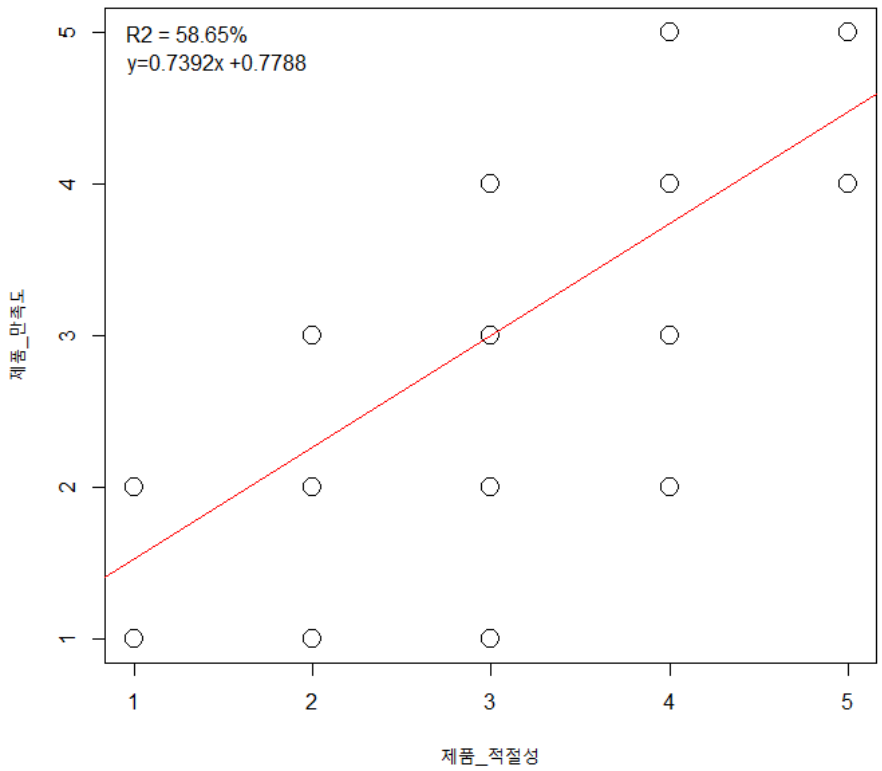
p-value가 2.2e-16로 유의수준 0.05보다 매우 작기 때문에 "제품 적절성은 제품 만족도에 영향을 미치지 않는다"라는 귀무가설을 기각할 수 있다. 따라서 제품

적절성은 제품 만족도에 영향을 미친다고 할 수 있다.
이때 회귀식의 절편은 0.77886 기울기는 0.73928임을 확인하였다.

2-2-4 선형 회귀분석 모델 시각화

```
plot(formula = y ~ x, data = product,
xlab ="제품_적절성", ylab = "제품_만족도" ,col = 1, cex = 2)
abline(lm, col = "red")
text(1.3,5, labels="R2 = 58.65%")
text(1.47,4.8, labels="y=0.7392x + 0.7788")
```

제품의 적절성이 제품의 만족도에 미치는 영향에 대한 회귀분석을 시각화하였다.
산점도 데이터를 잘 표현하는 직선은 아래와 같다.



2-3 Python을 통한 선형 회귀분석

분석 순서
사용할 Package생성
-> 데이터 불러오기
-> 데이터 전처리
-> Neuron 생성
-> 가중치와 절편구하기
-> 회귀식에 대한 상관계수 구하기
-> 선형 회귀분석 모델 시각화

2-3-1 사용할 Package 생성

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from matplotlib import font_manager, rc
```

2-3-2 데이터 불러오기

```
product = pd.read_csv("product.csv", encoding='cp949')
product.head()
```

```
>> product.head()
```

	제품_친밀도	제품_적절성	제품_만족도
0	3	4	3
1	3	3	2
2	4	4	4
3	2	2	2
4	2	2	2

2-3-3 데이터 전처리

```
x = product["제품_적절성"]
y = product["제품_만족도"]
```

분석에 필요한 컬럼들을 x, y 변수에 담았다.

2-3-4 Neuron 생성

```
class Neuron:

    def __init__(self):
        self.w = 1.0     # 가중치를 초기화합니다
        self.b = 1.0     # 절편을 초기화합니다

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

    def backprop2(self, x, err2):
        w_grad = x * err2    # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err2    # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad


    def fit(self, x, y, lr = 0.3, epochs=100):
        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 = -(y_i - y_hat)      # 오차 계산
                err2 = -(2/n) *(y_i - y_hat)
                w_grad, b_grad = self.backprop2(x_i, err2)
```

```
            self.w -= w_grad *lr          # 가중치 업데이트
            self.b -= b_grad *lr          # 절편 업데이트


neuron = Neuron( )
```

전체 훈련 데이터를 모두 이용하여 한 단위의 작업을 진행하는 epoch 값을 100으로 하여 적절한 가중치와 절편을 구하는 뉴런을 생성하였다.

2-3-5 가중치와 절편 구하기

```
neuron.fit(x,y)
neuron.w # 0.7453346285634439
neuron.b # 0.7625095886771006
```

위에서 생성된 뉴런을 x 와 y 에 적용하여 가중치와 절편값을 구하였다. 가중치인 기울기는 0.7453 정도이고, 절편은 0.7625 정도이다.

2-3-6 회귀식에 대한 상관계수 구하기

```
est_y = np.array(x) * neuron.w +  neuron.b
r2 = r2_score(y, est_y) #0.5880131758128271
```

r2_score() 함수를 사용해 실제 데이터 내의 각 x 값을 회귀식에 대입하여 y 값의 추정치를 구하였다. 그 값을 est_y에 넣은 뒤, 실제 y 값과 est_y 값의 상관계수를 구해 r2에 저장하였다. 두 변수는 약 58.8%의 선형 관계의 정도를 가지는 것을 알 수 있다. 분야와 연구자의 판단에 따라 몇퍼센트 이상이 절대적으로 유용한지 말할 수 없지만, 이 분석에서는 과반수 이상 설명이 가능했기 때문에 어느정도 유용하다고 판단하였다.

2-3-7 선형 회귀분석 모델 시각화

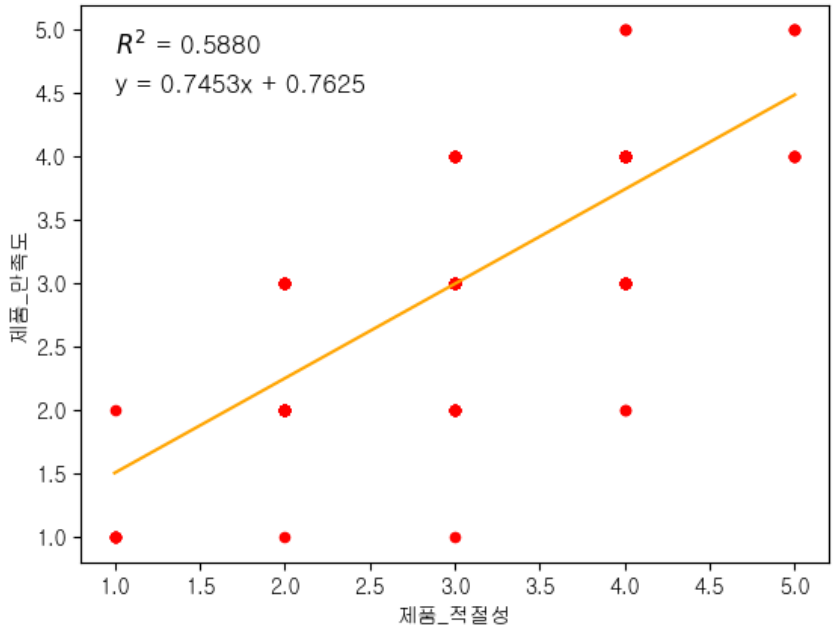
```
# 산점도 그래프 그리기
plt.scatter(x, y, color = 'r', s = 20)

# 회귀선 그래프 그리기
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')
```

```
# 텍스트 삽입
plt.text(1, 4.8, '$R^2$ = %.4f'%r2, size = 12) # (1, 4.8)의 위치에 크기 12로 R값 새김
plt.text(1, 4.5, 'y = %.4fx +  %.4f'%(neuron.w, neuron.b), size = 12)
# (1, 4.5)위치에 추세선 식 표현

# 한글 깨짐 문제 해결
font_path = "C:/Windows/Fonts/NGULIM.TTF"
font = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font)

# 라벨 입력
plt.xlabel('제품_적절성')
plt.ylabel('제품_만족도')
plt.show()
```

먼저 산점도를 scatter() 함수를 이용해 그렸고 옵션 s=20을 설정해 점의 크기를 지정해주었다. 그리고 plot() 함수로 회귀선을 산점도와 같이 그려냈다. 이때 x, y축이 1부터 5까지 그려지는 그래프를 만들기 위해 pt1과 pt2를 위와 같이 활용해 그래프를 그렸다.

제품의 적절성이 제품의 만족도에 미치는 영향에 대한 회귀분석을 시각화하였다. 산점도 그래프를 잘 표현하는 직선은 아래와 같다.

회귀분석 결과 제품 적절성이 제품 만족도에 대하여 58.8%를 설명할 수 있다는 것을 알 수 있다. 이때 회귀식의 절편과 기울기는 위에서 구한대로 각각 0.7625, 0.7453이다.

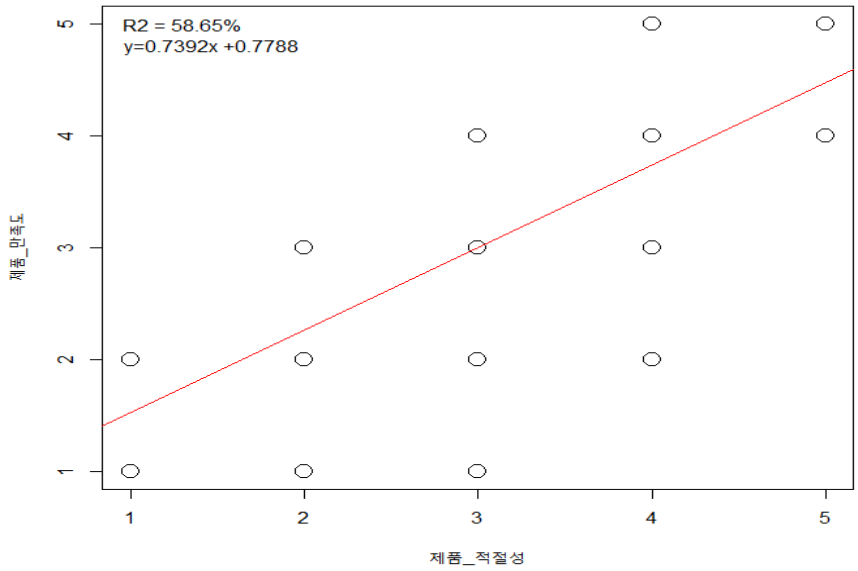


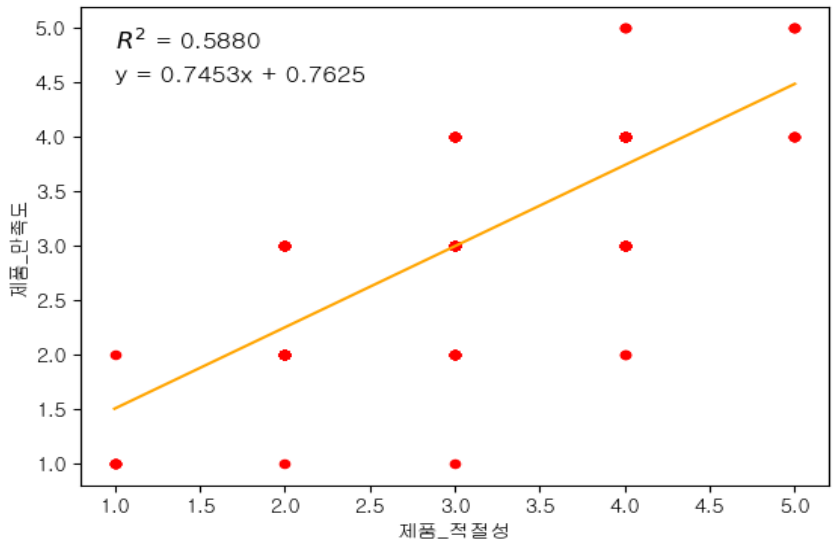
2-4 결과 비교

2-4-1 R과 Python 수치값 비교

	R	Python
절편(b)	0.7788	0.7625
가중치(w)	0.7392	0.7453
결정계수(R2)	0.5865	0.5880

2-4-2 R과 Python 시각화 그래프 비교





2-4-3 결론

R과 Python에서 절편과 가중치를 각각 구해본 결과 절편은 0.0163, 가중치는 0.0061 정도 수치적 차이가 있지만, 거의 같은 그래프임을 시각화 그림을 통해 알 수 있다.

이때 이 직선은 제품적절성과 제품만족도에 대한 산점도 데이터를 가장 잘 표현하는 직선이다.

그리고 회귀모델이 주어진 자료에 얼마나 적합한지를 평가하는 지표중 하나인 결정계수 값은 0.0015정도 차이를 가져 위와 마찬가지로 같은 결과를 얻어냈음을 알 수 있다. 이 분석에서는 두 결과 모두 0.5보다 큰 결정계수를 가지기 때문에 절반 이상 설명할 수 있다고 생각해 어느정도 유용하다고 판단하였다.

같은 데이터를 가지고 같은 방법으로 분석을 실행했기 때문에 두 언어에서 모두 동일한 결과를 얻어낼 수 있었다.

3. 로지스틱 회귀분석 실행 및 결과 비교

3-1 데이터셋 설명

```
'data.frame':  366 obs. of  15 variables:
 $ Date          : chr  "2014-11-01" "2014-11-02" "2014-11-03" "2014-11-04" ...
 $ MinTemp       : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
 $ MaxTemp       : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
 $ Rainfall      : num  0 3.6 3.6 39.8 2.8 0 0.2 0 0 16.2 ...
 $ Sunshine      : num  6.3 9.7 3.3 9.1 10.6 8.2 8.4 4.6 4.1 7.7 ...
 $ WindGustDir   : chr  "NW" "ENE" "NW" "NW" ...
 $ WindGustSpeed: int  30 39 85 54 50 44 43 41 48 31 ...
 $ WindDir       : chr  "NW" "W" "NNE" "W" ...
 $ WindSpeed     : int  20 17 6 24 28 24 26 24 17 6 ...
 $ Humidity      : int  29 36 69 56 49 57 47 57 48 32 ...
 $ Pressure      : num  1015 1008 1007 1007 1018 ...
 $ Cloud         : int  7 3 7 7 7 5 6 7 7 1 ...
 $ Temp          : num  23.6 25.7 20.2 14.1 15.4 14.8 17.3 15.5 18.9 21.7 ...
 $ RainToday     : chr  "No" "Yes" "Yes" "Yes" ...
 $ RainTomorrow : chr  "Yes" "Yes" "Yes" "Yes" ...
```

기후 상태에 따른 익일 강우 여부에 대한 데이터이다. RainTomorrow 컬럼을 제외한 문자형 데이터를 가지는 컬럼은 사용하지 않는다.

3-2 R을 통한 로지스틱 회귀분석

분석 순서
사용할 library 생성

-> 데이터 불러오기
-> 데이터 전처리
-> 학습데이터 와 검정데이터 생성
-> 로지스틱 회귀모델 생성
-> 정확도 측정
-> 시각화

3-2-1 사용할 library 생성

```
library(SyncRNG)
library(ROCR)
```

3-2-2 데이터 불러오기

```
weather = read.csv("weather.csv", stringsAsFactors = F)
head(weather, 3)
```

```
> head(weather, 3)
        Date MinTemp MaxTemp Rainfall Sunshine WindGustDir
1 2014-11-01     8.0    24.3      0.0      6.3          NW
2 2014-11-02    14.0    26.9      3.6      9.7         ENE
3 2014-11-03    13.7    23.4      3.6      3.3          NW
  WindGustSpeed WindDir WindSpeed Humidity Pressure Cloud Temp
1            30      NW        20       29   1015.0     7 23.6
2            39       W        17       36   1008.4     3 25.7
3            85     NNE         6       69   1007.2     7 20.2
  RainToday RainTomorrow
1        No          Yes
2       Yes          Yes
3       Yes          Yes
```

3-2-3 데이터 전처리

```
weather_df <- weather[ , c(-1, -6, -8, -14)]
weather_df$RainTomorrow[weather_df$RainTomorrow == 'Yes'] <- 1
weather_df$RainTomorrow[weather_df$RainTomorrow == 'No'] <- 0
weather_df$RainTomorrow <- as.numeric(weather_df$RainTomorrow)
```

불필요한 컬럼들을 제거하고 Y변수를 대상으로 더미 변수를 생성하였다.

3-2-4 학습데이터와 검정데이터 생성

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

head(weather_df[-idx[1:length(idx)],])
idx[1:length(idx)]
train <- weather_df[idx[1:length(idx)],]
test <- weather_df[-idx[1:length(idx)],]
```

python과 난수 통일을 위해 SyncRNG 함수를 사용하여 학습데이터와 검정데이터를 생성하였다. 학습과 검정데이터를 나누어 분석을 진행하면 올바른 모델의 성능을 측정할 수 있다.

3-2-5 로지스틱 회귀모델 생성

```
weather_model <- glm(RainTomorrow ~ ., data = train, family = 'binomial',
na.action=na.omit)
summary(weather_model)
```

```
> summary(weather_model)

Call:
glm(formula = RainTomorrow ~ ., family = "binomial", data = train,
    na.action = na.omit)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.70802  -0.44502  -0.20761  -0.07825   2.20034

Coefficients:
```

```
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   266.68491   73.55913   3.625 0.000288 ***
MinTemp        -0.20702    0.11454  -1.807 0.070713 .
MaxTemp        -1.00156    0.43681  -2.293 0.021854 *
Rainfall       -0.02069    0.05481  -0.377 0.705802
Sunshine       -0.24674    0.15687  -1.573 0.115739
WindGustSpeed   0.05494    0.02956   1.859 0.063094 .
WindSpeed      -0.10655    0.05102  -2.088 0.036773 *
Humidity        0.07052    0.04068   1.734 0.082999 .
Pressure       -0.26910    0.07144  -3.767 0.000165 ***
Cloud           0.16241    0.15995   1.015 0.309935
Temp            1.26419    0.45686   2.767 0.005656 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 168.483  on 179  degrees of freedom
Residual deviance:  96.862  on 169  degrees of freedom
  (5 observations deleted due to missingness)
AIC: 118.86

Number of Fisher Scoring iterations: 6
```

변수 "MaxTemp", "WindSpeed", "Pressure", "Temp"가 유의미함을 확인하였다.

3-2-6 정확도 측정

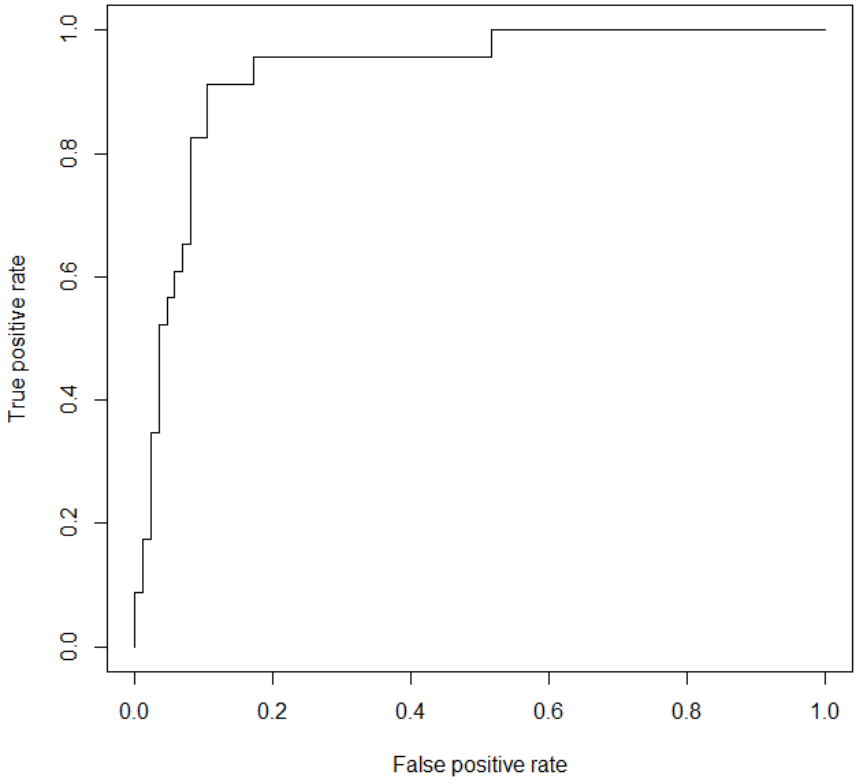
```
pred <- predict(weather_model, newdata = test, type = "response")

result_pred <- ifelse(pred >= 0.5, 1, 0)

table(result_pred, test$RainTomorrow)
sum(result_pred == test$RainTomorrow) / length(test$RainTomorrow)
```

0.5보다 크면 1을 아니면 0을 갖도록 하여 예측 결과를 저장하였다. 예측 결과와 test 데이터셋을 비교해 정확도를 구해냈다. 정확도: 0.8727273

3-2-7 ROC curve 시각화

```
pr <- prediction(pred, test$RainTomorrow)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```

y축은 민감도(tpr) x축은 1-특이도(fpr)를 갖는 ROC curve 생성하였다. ROC 커브는 이진 분류기의 성능을 표현하는 커브로 좌상단에 그래프가 닿을수록 성능이 좋기 때문에 아래 그래프를 보면 잘 분리된 것을 알 수 있다.


True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

3-3 Python을 통한 로지스틱 회귀분석

분석 순서
사용할 Package생성
-> 데이터 불러오기
-> 데이터 전처리
-> 학습데이터와 검정데이터 생성
-> 로지스틱 회귀뉴런 생성
-> 정확도 측정
-> 시각화

3-3-1 사용할 Package생성

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from SyncRNG import SyncRNG
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_curve
```

3-3-2 데이터 불러오기

```
weather = read.csv("weather.csv", stringsAsFactors = F)
head(weather, 3)
```

```
> head(weather, 3)
        Date MinTemp MaxTemp Rainfall Sunshine WindGustDir
1 2014-11-01     8.0    24.3      0.0      6.3          NW
2 2014-11-02    14.0    26.9      3.6      9.7         ENE
3 2014-11-03    13.7    23.4      3.6      3.3          NW
  WindGustSpeed WindDir WindSpeed Humidity Pressure Cloud Temp
```

```
1          30      NW        20       29   1015.0     7 23.6
2          39       W        17       36   1008.4     3 25.7
3          85     NNE         6       69   1007.2     7 20.2
  RainToday RainTomorrow
1        No          Yes
2       Yes          Yes
3       Yes          Yes
```

3-3-3 데이터 전처리

```
x_train=train.drop(["Date","WindGustDir","WindDir","RainToday","RainTomorrow"],axis = 1)
x_test=test.drop(["Date","WindGustDir","WindDir","RainToday","RainTomorrow"],axis = 1)
date['RainTomorrow'] = date['RainTomorrow'].apply(lambda x : 1 if x== "Yes" else 0)
```

불필요한 컬럼들을 제거하고 RainTomorrow의 문자형 데이터를 정수형으로 바꾸어주었다.

3-3-4 학습데이터와 검정데이터 생성

```
v=list(range(1,len(date)+ 1))
s=SyncRNG(seed=42)
ord=s.shuffle(v)
idx=ord[:round(len(date)*0.7)]
for i in range(0,len(idx)):
  idx[i]=idx[i]-1

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

y_train=train["RainTomorrow"]
x_train = np.array(x_train)
y_train = np.array(y_train)

y_test=test["RainTomorrow"]
```

```
x_test = np.array(x_test)
y_test = np.array(y_test)
```

R과 난수 통일을 위해 SyncRNG 함수를 사용하여 R에서와 동일한 학습데이터와 검정데이터를 생성하였다. R에서와 마찬가지로 학습과 검정데이터로 분리하면 올바른 모델 성능을 측정할 수 있다.

3-3-5 로지스틱 회귀뉴런 생성

```
class LogisticNeuron:

    def __init__(self, learning_rate=0.01, l1=0, l2=0):
        self.w = None
        self.b = None
        self.losses = []
        self.val_losses = []
        self.w_history = []
        self.lr = learning_rate
        self.l1 = l1
        self.l2 = l2

    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, epochs=300, x_val=None, y_val=None):
        self.w = np.ones(x.shape[1])  # 가중치를 초기화합니다.
        self.b = 0  # 절편을 초기화합니다.
        self.w_history.append(self.w.copy())  # 가중치를 기록합니다.
```

```
        np.random.seed(42)  # 랜덤 시드를 지정합니다.
        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
                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
                # 그래디언트에서 페널티 항의 미분 값을 더합니다
                w_grad += self.l1 * np.sign(self.w) +  self.l2 * self.w
                self.w -= self.lr * w_grad  # 가중치 업데이트
                self.b -= self.lr * b_grad  # 절편 업데이트
                # 가중치를 기록합니다.
                self.w_history.append(self.w.copy())
                # 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) +  self.reg_loss()/len(y))
            # 검증 세트에 대한 손실을 계산합니다

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

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

    def reg_loss(self):
        return self.l1 * np.sum(np.abs(self.w)) +  self.l2 / 2 * np.sum(self.w **
2)
```

로지스틱 회귀를 구현하기 위해 활성함수(activation)로 시그모이드 함수를, 임계함수로 계단함수를 적용하였다. 이때 로지스틱회귀는 경사 하강법을 사용해 손실 함수의 결과값을 최소화하는 방향으로 가중치를 업데이트한다.

또한 가중치 최적값을 제대로 찾기 위해 매 에포크마다 훈련 세트의 샘플 순서를 섞었다. 넘파이 배열의 인덱스를 섞은 후 인덱스 순서대로 샘플을 뽑게 되면 훈련

세트 자체를 섞는 것보다 효율적이고 빠르다.
시그모이드 함수의 출력값은 0~1 사이의 확률값이고 양성 클래스를 판단하는 기준은 0.5이상이다. z가 0보다 크면 출력값은 0.5보다 크고 아니면 0.5보다 작기 때문에 predict() 메서드에서 시그모이드 함수를 사용하지 않고 z 값의 크기만 비교해 결과를 반환하였다.

3-3-6 정확도 측정

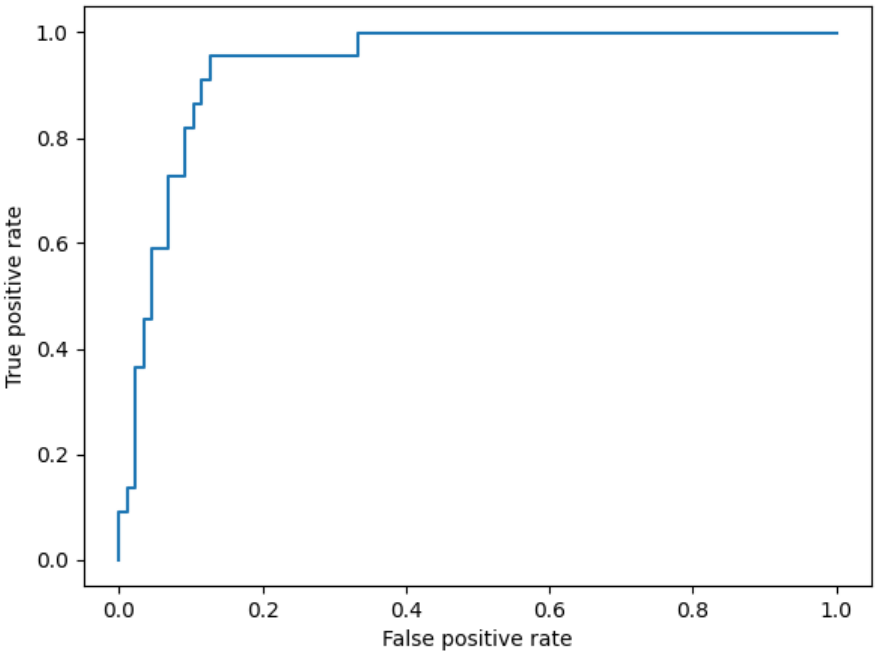
```
neuron = SingleLayer()
neuron.fit(x_train, y_train)
neuron.score(x_test, y_test
```

에포크마다 훈련세트를 무작위로 섞어 손실 함수의 값을 줄였기 때문에 수치적으로 꽤 높은 정확도(=0.8807339449541285)를 얻어낼 수 있었다.

3-3-7 ROC curve 시각화

```
pred_positive_label = neuron.proba(x_test)
fprs, tprs, thresholds = roc_curve(y_test,  pred_positive_label, pos_label=1)
precisions, recalls, thresholds = roc_curve(y_test, pred_positive_label,
pos_label=1)
plt.figure(figsize=(15,5))
plt.plot(fprs,tprs,label='ROC')
plt.ylable(“True positive rate”)
plt.xlable(“False positive rate”)
```

R에서와 마찬가지로 ROC curve 생성하였다. 아래 그래프가 좌상단에 가깝게 그려졌기 때문에 잘 분리된 것을 알 수 있다.


True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

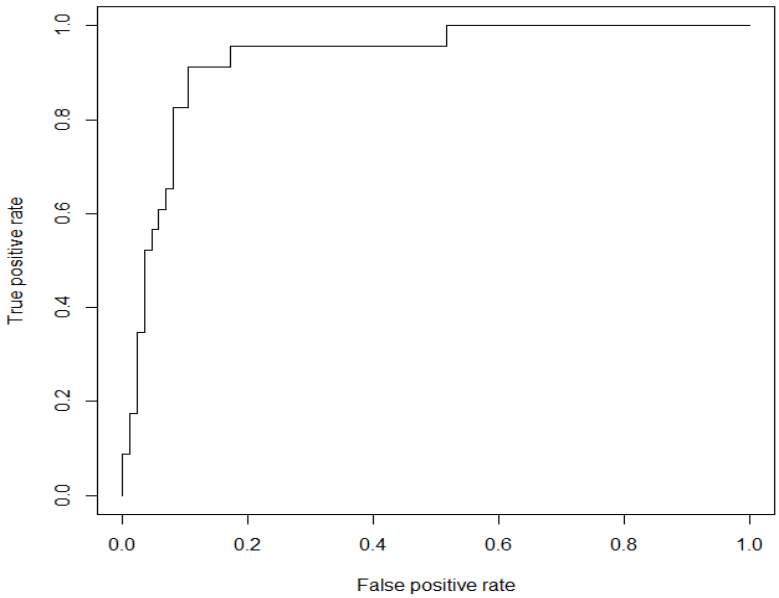
3-4 결과 비교

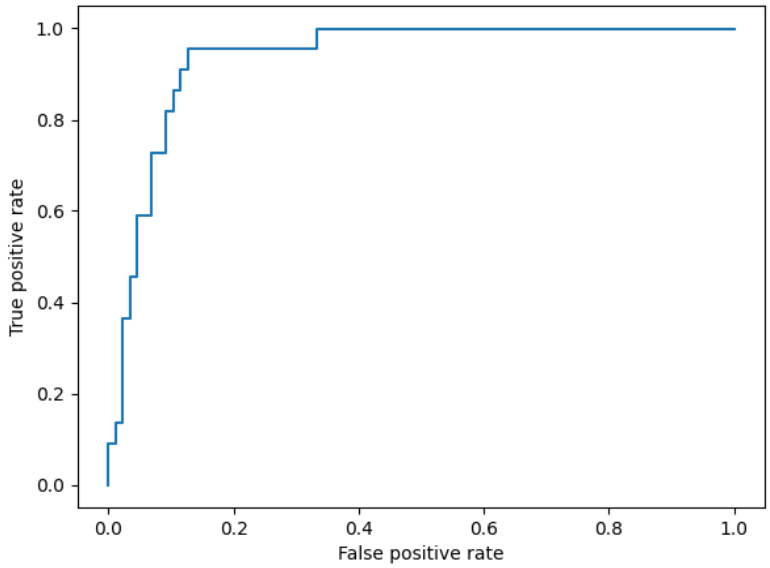
메모 포함[4]: 결과 추가

3-4-1 R과 Python 수치값 비교

	R	Python
정확도(Accuracy)	0.8727	0.8807

3-4-2 R과 Python 시각화 그래프 비교





3-4-3 결론

R과 Python을 통해 로지스틱회귀 분석을 실시한 결과 정확도가 87.27%, 88.07%로 거의 같은 결과를 얻어냈다는 것을 알 수 있다. 같은 데이터셋에서 학습과 검정 데이터를 동일하게 나누었기 때문에 거의 같은 결과를 얻어낼 수 있었다.

ROC 그래프에서도 같은 그래프를 그려낸 것을 알 수 있고, 좌상단에 가깝기 때문에 이진 분류가 잘 된 것을 알 수 있다. 즉, 예측 값이 정분류 된 값이 많다는 것이다.

4. 다중분류 실행 및 결과 비교

4-1 데이터셋 설명

'data.frame': 150 obs. of 6 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 ... $ label : num 0 0 0 0 0 0 0 0 0 0 ...
붓꽃의 3가지 종(setosa, versicolor, virginica)에 대해 꽃받침 sepal과 꽃잎 petal의 길이를 정리한 데이터이다.

4-2 R을 통한 다중 분류(랜덤포레스트)

분석 순서
사용할 library생성
-> 데이터 불러오기
-> 데이터 전처리
-> 랜덤포레스트 실행
-> 오차 시각화
-> 정확도 시각화

4-2-1 사용할 library생성

```
library(randomForest)
```

4-2-2 데이터 불러오기

```
data(iris)
head(iris, 3)
```

```
  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
```

4-2-3 데이터 전처리

```
set.seed(42)
idx <- sample(2, nrow(iris), replace=T, prob=c(0.7, 0.3))
trData <- iris[idx == 1, ]
nrow(trData)
teData <- iris[idx == 2, ]
nrow(teData)
```

올바른 모델 성능 측정을 위해 sample() 함수를 사용해 prob 옵션을 0.7, 0.3으로 지정하여 학습데이터와 검정데이터로 분리하였다.

4-2-4 랜덤포레스트 실행

```
RFmodel <- randomForest(Species~., data=trData, ntree=100, proximity=T)
RFmodel
```

```
Call:
 randomForest(formula = Species ~ ., data = trData, ntree = 100,       proximity =
T)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 1.04%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         28          0         0  0.00000000
versicolor      0         37         1  0.02631579
virginica       0          0        30  0.00000000
```

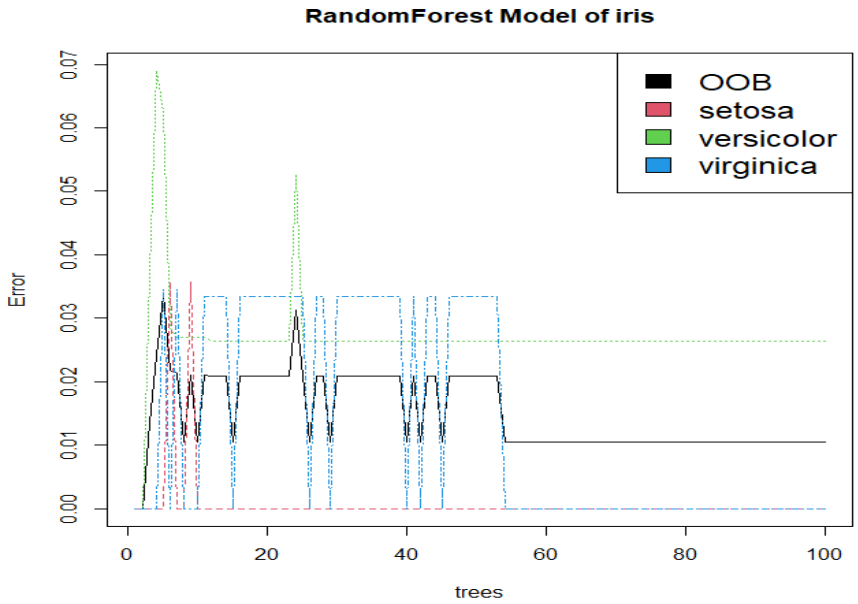
개체들 간의 근접도 행렬을 얻기 위해 proximity = T를 사용하였다.

4-2-5 손실 그래프 시각화

```
plot(RFmodel, main="RandomForest Model of iris")
legend("topright", colnames(RFmodel$err.rate),col=1:4,cex=1.5,fill=1:4)
```

RFmodel에 plot() 함수를 대입하여 손실함수 그래프를 얻어냈다. 그 결과 tree 값이 일정 수준이 지나면서 Error가 특정 값으로 유지되는 것을 확인할 수 있다. 특히 두 종류의 Error 값은 0이 되는 것을 통해 tree 값을 키울수록 손실 함수를 최소화 할 수 있음을 알 수 있다.

그리고 OOB(Out of Bag)란 train 데이터 중 선택되지 않은 데이터를 의미한다. OOB error는 이러한 샘플에서 얻은 오류를 뜻하고, 다른 검증 기술과 비교했을 때 신뢰도가 높다. 이 값이 0.01이므로 손실 함수 최적화가 적절히 이루어졌음을 확인할 수 있다.



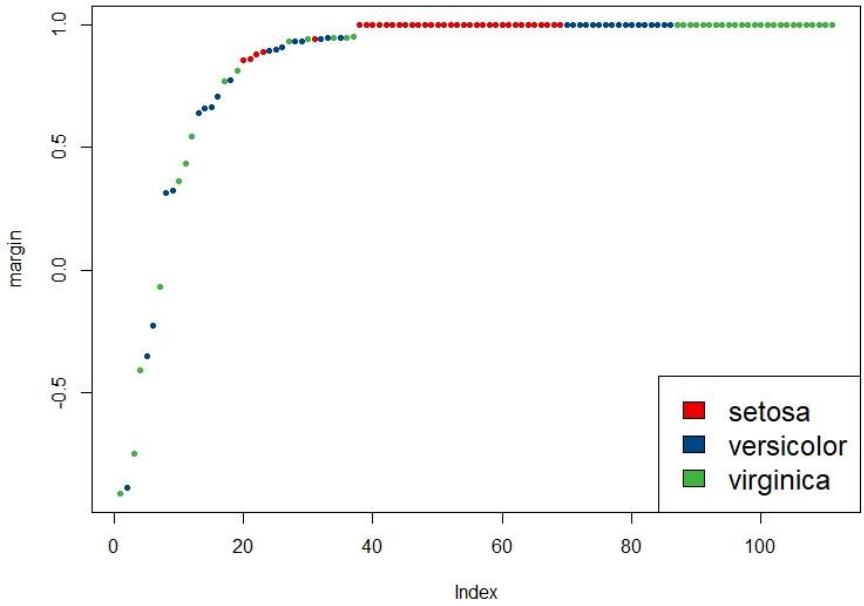
4-2-6 정확도 그래프 시각화

```
# 테스트데이터로 예측
pred <- predict(RFmodel, newdata=teData)

# 실제값과 예측값 비교
table(teData$Species, pred)

# 시각화
COLS = c("#ED0000FF","#00468BFF","#42B540FF")
names(COLS) =unique(teData$Species)
MAR = sort(margin(RFmodel,teData$Species))
plot(as.numeric(MAR),col=COLS[names(MAR)],pch=20,ylab="margin")
legend("bottomright",fill=COLS,names(COLS), cex=1.5)
```

```
> margin(RFmodel, teData$Species)
 setosa      setosa      setosa      setosa      setosa
1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
 setosa      setosa      setosa      setosa      setosa
1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
 setosa      setosa      setosa      setosa      setosa
1.00000000 0.87878788 0.86206897 1.00000000 1.00000000
 setosa      setosa      setosa      setosa      setosa
1.00000000 1.00000000 1.00000000 1.00000000 0.88888889
 setosa      setosa      setosa      setosa      setosa
1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
 setosa      setosa      setosa      setosa      setosa
1.00000000 1.00000000 1.00000000 0.94285714 1.00000000
 setosa      setosa      setosa      setosa      setosa
0.85714286 1.00000000 1.00000000 1.00000000 1.00000000
 setosa      setosa  versicolor  versicolor  versicolor
1.00000000 1.00000000 0.94736842 0.93548387 1.00000000
versicolor  versicolor  versicolor  versicolor  versicolor
0.77419355 0.32352941 1.00000000 0.90000000 0.64285714
versicolor  versicolor  versicolor  versicolor  versicolor
1.00000000 0.94594595 1.00000000 0.65957447 -0.88888889
versicolor  versicolor  versicolor  versicolor  versicolor
1.00000000 0.31707317 0.66666667 1.00000000 0.70588235
versicolor  versicolor  versicolor  versicolor  versicolor
-0.22580645 1.00000000 1.00000000 1.00000000 -0.35135135
versicolor  versicolor  versicolor  versicolor  versicolor
0.93548387 1.00000000 0.91111111 1.00000000 1.00000000
versicolor  versicolor  versicolor  versicolor  versicolor
0.89473684 1.00000000 1.00000000 1.00000000 1.00000000
versicolor  versicolor   virginica   virginica   virginica
```

```
0.94444444  1.00000000  1.00000000  1.00000000  1.00000000
virginica       virginica       virginica       virginica       virginica
1.00000000  1.00000000 -0.91176471  0.94871795  0.94117647
virginica       virginica       virginica       virginica       virginica
1.00000000  0.94594595  1.00000000  0.95238095  1.00000000
virginica       virginica       virginica       virginica       virginica
1.00000000  1.00000000 -0.75000000  1.00000000  0.43589744
virginica       virginica       virginica       virginica       virginica
1.00000000  1.00000000  0.36363636  1.00000000  1.00000000
virginica       virginica       virginica       virginica       virginica
1.00000000 -0.41176471  1.00000000  1.00000000  1.00000000
virginica       virginica       virginica       virginica       virginica
-0.06976744  1.00000000  1.00000000  0.81395349  0.76923077
virginica       virginica       virginica       virginica       virginica
1.00000000  1.00000000  1.00000000  0.93333333  1.00000000
virginica
0.54285714
```

아래의 그림은 훈련용 자료 값(총 109개)의 마진을 나타낸다. 마진(margin)은 랜덤포리스트의 분류기(classifiers) 가운데 정분류를 수행한 비율에서 다른 클래스로 분류한 비율의 최대치를 뺀 값을 나타낸다. x값이 1에 가까울수록 정확히 분류되고 -1에 가까울수록 제대로 분류되지 않음을 뜻한다. 위의 margin 함수의 결과를 통해 음수 값을 가지지 않는 'setosa' 종이 가장 잘 분류가 된 것을 아래 그래프에서 확인할 수 있다.



4-2-7 정확도 계산

sum(teData$Species == pred) / length(pred)
예측값과 검정데이터 두 변수를 비교해 정확도(=0.8703704)를 구해냈다. 약 87%로 예측이 잘 되었음을 알 수 있다.

4-3 Python을 통한 다중 분류(인공신경망)

분석 순서
사용할 Package생성
-> 데이터 전처리
-> 인공신경망 실행
-> 오차 시각화

-> 정확도 시각화

4-3-1 사용할 Package생성

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

4-3-2 데이터 전처리

```
# 데이터 로드
iris2 = load_iris()

X = iris2['data']
y = iris2['target']

# 원-핫 인코딩 변환
Y = tf.keras.utils.to_categorical(y)

# 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                                        test_size=0.3,
                                        random_state=42)
```

올바른 모델 성능 측정을 위해 전체 데이터 중 70%는 학습데이터로, 30%는 검정데이터로 분리하였다.

4-3-3 인공신경망 실행

```
print(X_train.shape, y_train.shape)
# (105, 4) (105, 3)

# 모델 생성
model = Sequential()

# 은닉층과 출력층에 모델 추가
```

```
model.add(Dense(100,activation='sigmoid',input_shape=(4,)))
model.add(Dense(100,activation='sigmoid'))
# 합성곱층을 여러 개 추가해도 학습할 모델 파라미터의 개수가 크게 늘지 않기 때문에 계산 효율성 좋음
model.add(Dense(3,activation='softmax'))
model.summary()

# 최적화 알고리즘과 손실 함수 지정
model.compile(loss='categorical_crossentropy',
              optimizer='Adam', , # 최적화 알고리즘으로 적응적 학습률 알고리즘 아담 사용
              metrics=['accuracy'])
# 아담은 손실 함수의 값이 최적값에 가까워질수록 학습률을 낮춰 손실 함수의 값이 안정적으로 수렴되게 함

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200)
```

```
Model: "sequential"
_________________________________________________________________
Layer(type)     Output Shape       Param  #
=================================================================
dense(Dense)    (None, 100)        500

dense_1(Dense)  (None, 100)        10100

dense_2(Dense)  (None, 3)          303
100*3+3
=================================================================
Total params: 10,903
Trainable params: 10,903
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/200
4/4 [==============================] - 1s 57ms/step - loss: 1.1177 - accuracy: 0.3048 - val_loss: 1.1282 - val_accuracy: 0.2889
Epoch 2/200
4/4 [==============================] - 0s 5ms/step - loss: 1.1118 - accuracy: 0.3524 - val_loss: 1.1435 - val_accuracy: 0.2889
...
...
...
Epoch 199/200
4/4 [==============================] - 0s 5ms/step - loss: 0.0832 - accuracy: 0.9810 - val_loss: 0.0481 - val_accuracy: 1.0000
Epoch 200/200
```

```
4/4 [==============================] - 0s 5ms/step - loss: 0.0843 - accuracy: 0.9619 - val_loss: 0.0489 - val_accuracy: 1.0000
```

입력데이터의 개수는 4개이고 출력층의 유닛 개수는 3개이므로 input_shape 매개변수에 4를, 마지막 Dense의 매개변수에 3을 입력하였다. 또한 합성곱층을 여러 개 추가하여 성능을 향상시켰는데 이렇게 해도 학습할 모델 파라미터의 개수가 크게 늘지 않기 때문에 계산 효율성이 좋다.

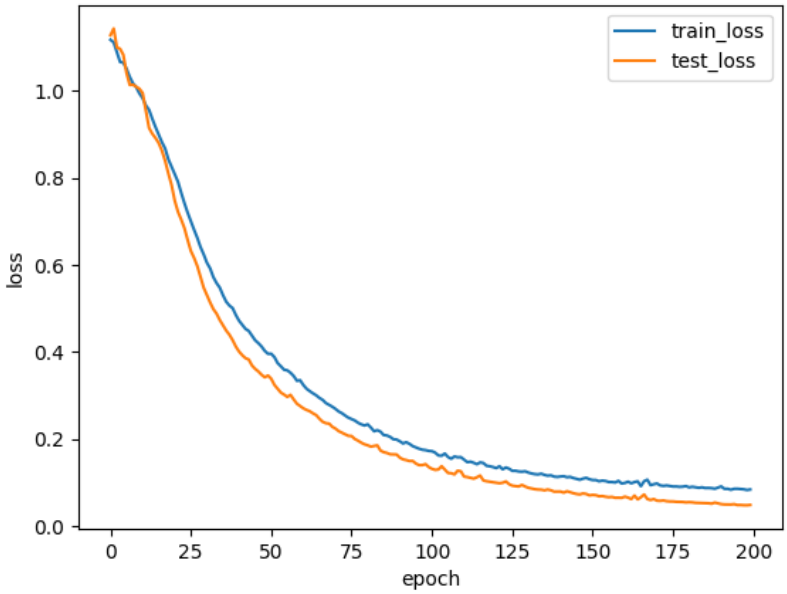
이때 첫 번째 완전 연결층의 가중치 개수는 4개의 입력이 100개의 뉴런에 연결되고, 가중치는 뉴런마다 있으므로 4*100+100=500개가 된다. 두 번째 완전 연결층의 가중치 개수도 100개의 입력이 100개의 뉴런에 연결되고 뉴런마다 가중치가 있으므로 100*100+100=10100개가 된다. 마지막 연결층 역시 같은 원리로 100개의 입력이 3개의 뉴런에 연결되고 뉴런마다 있는 가중치를 더해 100*3+3=303개가 된다.

최적화 알고리즘으로 adam을 지정하였는데, 이 알고리즘은 손실 함수의 값이 최적값에 가까워질수록 학습률을 낮춰 손실 함수의 값이 안정적으로 수렴되게 한다. 결과적으로 200번째 에포크에서 정확도 0.9619가 나오는 것을 확인하였다.

4-3-4 손실 그래프 시각화

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss', 'test_loss'])
plt.show()
```

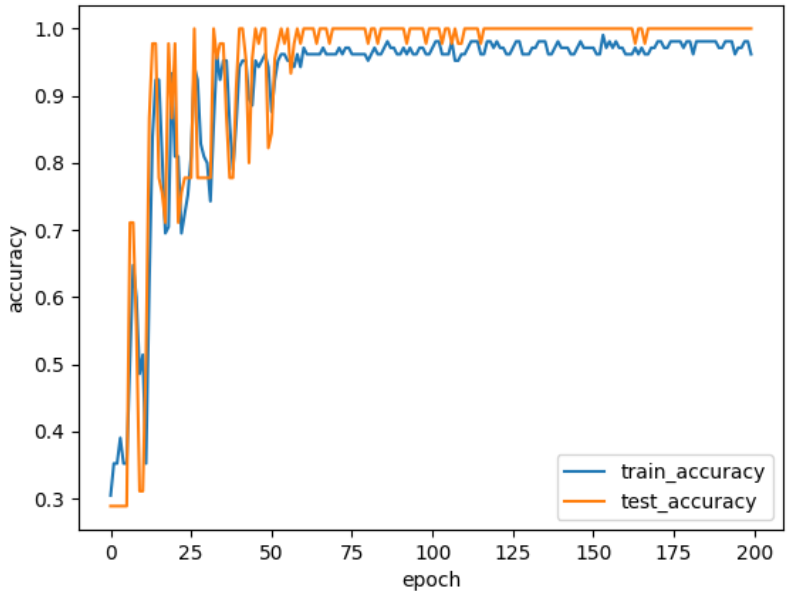
200번의 에포크를 거친 손실 함수 그래프를 시각화하여 0에 수렴하는 그래프를 얻어냈다. 분류 문제에서 정확도를 직접 최적화 할 수 없어 크로스 엔트로피 손실 함수를 최적화한 것이다. 하지만 손실함수를 최적화한다고 정확도 역시 항상 높아지는 것은 아니다.



4-3-5 정확도 그래프 시각화

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train_accuracy', 'test_accuracy'])
plt.show()
```

학습데이터의 정확도와 검정데이터의 정확도를 그려낸 결과 어느 순간부터 accuracy 값이 1에 가까이 유지되는 것을 알 수 있다. 위에서 언급한 것처럼 손실함수를 최적화하더라도 항상 정확도가 향상되는 것은 아니지만 충분히 에포크를 늘리고 지켜본 결과 loss 값이 줄어들수록 accuracy 값이 증가했다고 할 수 있는 결과를 얻을 수 있었다.



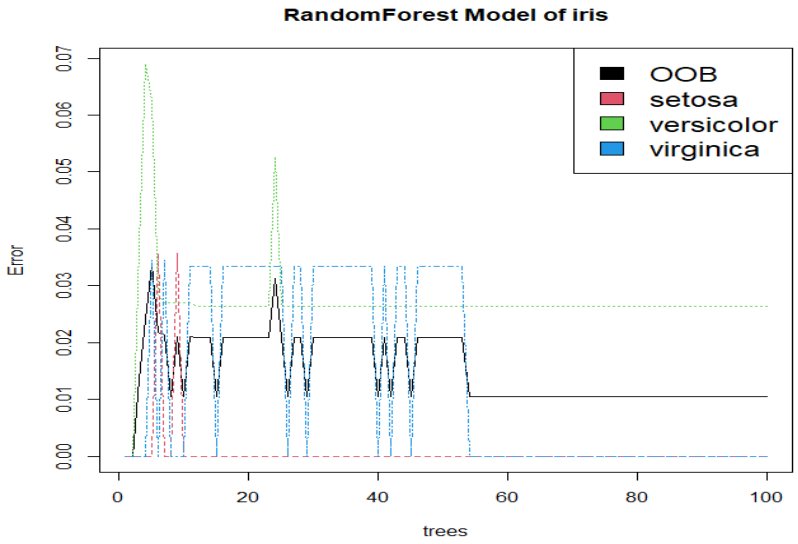
4-4 결과 비교

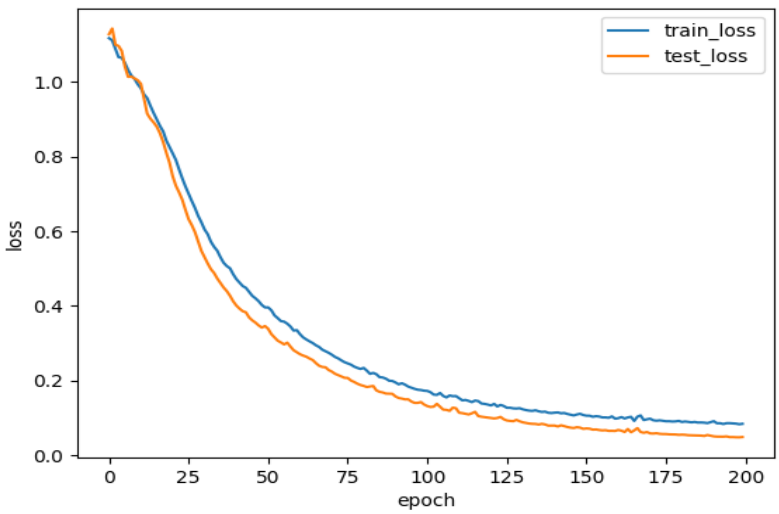
4-4-1 R과 Python 수치값 비교

	R	Python
정확도(Accuracy)	0.8703	0.9619

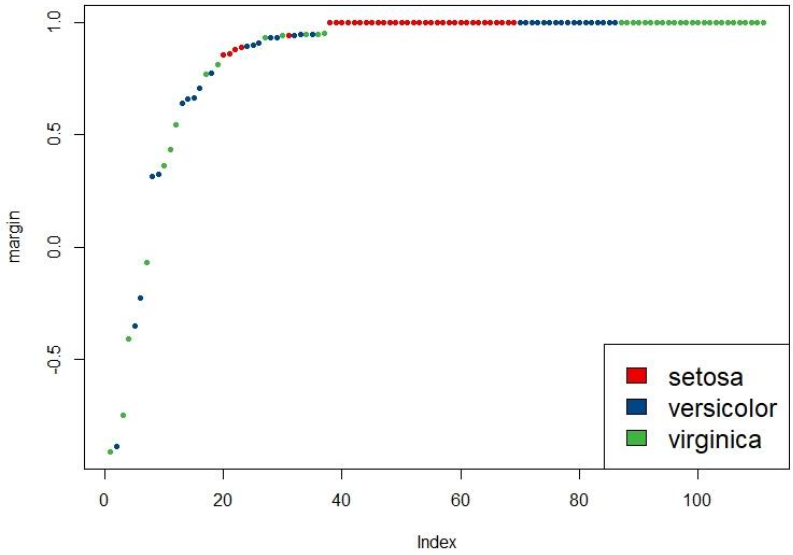
4-4-2 R과 Python 시각화 그래프 비교

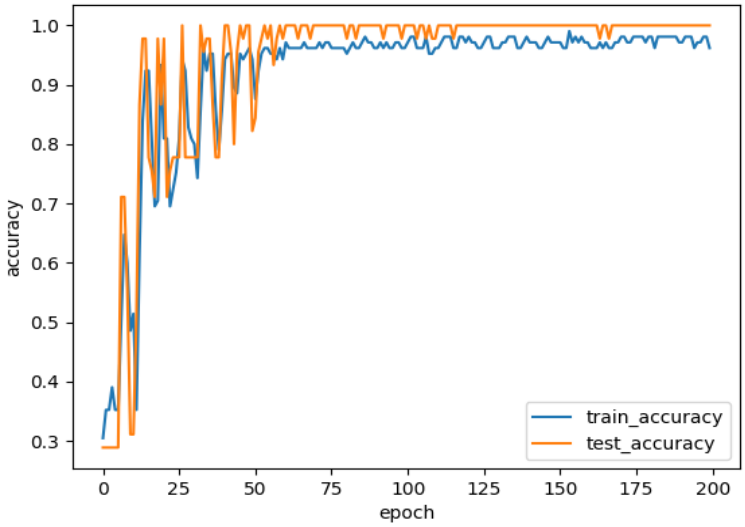
- R, Python 손실 그래프 비교





- R, Python 정확도 그래프 비교





4-4-3 결론

R과 Python을 통해 로지스틱회귀 분석을 실시한 결과 정확도가 각각 87.03%, 96.19%로 얻어졌다. R에서는 ntree 수를 100, python에서는 에포크 수를 200으로 지정했기 때문에 차이가 생겼다.

하지만 두 손실 그래프는 비슷한 모양을 가지는 것을 알 수 있다. 두 언어 모두 tree와 epoch 숫자가 커질수록 손실값이 0에 가까워지는 것을 확인할 수 있다.

R 정확도 그래프는 margin() 함수를 통해 각 항목마다 (잘 분류된 비율 - 잘못 분류된 비율 최대값) 계산을 통해 각각의 값을 계산하고, 그 값이 오름차순으로 나열된 그래프이다. 그에 비해 python 정확도 그래프는 epoch 값에 따라 변화되는 학습데이터와 검정데이터의 정확도가 나타난 그래프이다. R 정확도 그래프를 통해 과반수 이상의 항목이 잘 분류가 되었음을 파악할 수 있고, python 그래프를 통해 에포크 값을 키울수록 정확도가 향상되는 것을 알 수 있다.