기계학습의원리와 응용
중간고사

2022011038
기계공학과 장대윤

1. LASSO 를 파이썬 코딩으로 구현하기

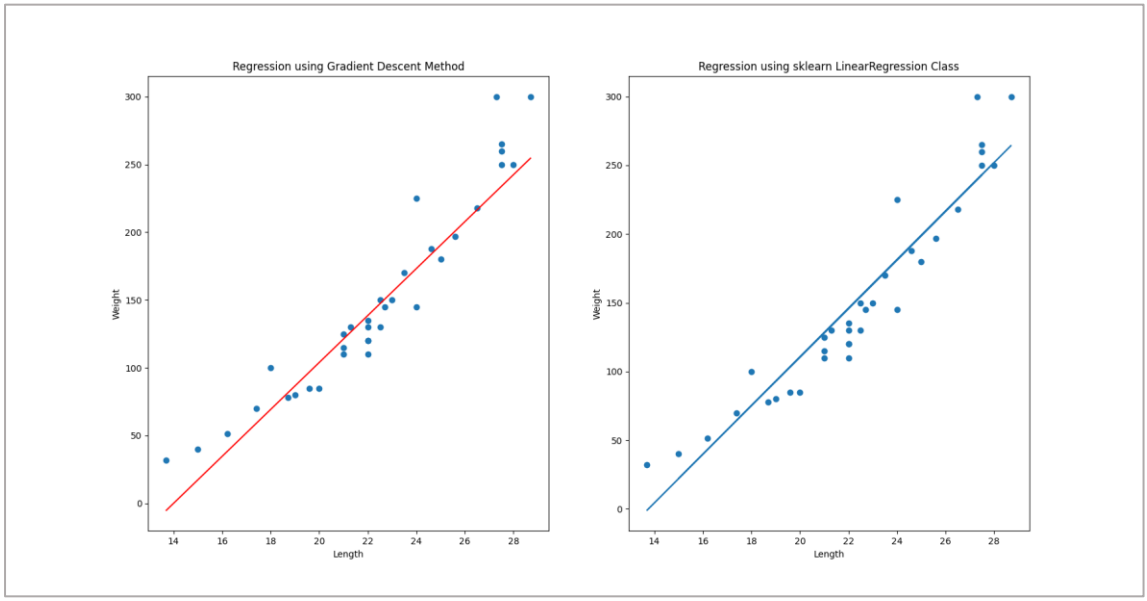
bit.ly/fish_csv_data 의 농어(perch) 데이터를 기반으로 Lasso Regression을 구현하였다.
직접구현한 결과값과 사이킷런에서 제공하는 Lasso() 함수를 사용한 결과를 비교하였다
[데이터 및 학습 파라미터 설정]
- Input : length (몸통 길이)
- output : weight (무게)
- epoch (iteration) = 5000 cycle
- lamda (learning-rate) = 1e-3 (0.001)

[결과 분석]
- 결과값

Coefficient	직접 구현	사이킷런
Weight	17.308329	17.691616
Bias	-242.202478	-243.335293

- Score
> train data set : 0.9300563
> test data set : 0.756351

점수평가값은 test 데이터가 부정확한 모습을 보였다. 이는 적은 데이터에는 선형회귀의 양상은 평가할 수 있으나, 그 정확도까지는 따라가기 어려움을 알 수 있다.
- 그래프



좌측이 weight, bias를 직접 iteration 하면서 생성한 선형그래프이며, 우측은 사이킷런에서 제공하는 라이브러리를 활용하여 그린 것이다. 비슷한 그래프이지만 우측 상단 부분을 보면 점을 지나는 위치가 약간 상이함을 알 수 있다. 이는 반복횟수를 증가시킬 경우 도달할 수 있다. 다만 다중 특성분류를 해야할 경우 과적합이 될 수 있기에 반복수는 늘리지 않는다.

2. **Logistic Regression 문제를 파이썬 코딩으로 구현하되 경사하강법을 적용하기.**

학습데이터는 NBA 2021 Rookie들의 평가 파라미터를 input으로, Draft 진출여부를 target으로 설정했다. 총 47명의 선수가 있으며, input은 FG%(필드골 성공률), FT%(프리드로우 성공률), 3PM(3점슛 성공률), PTS(득점수), TREB(트레블링), AST(어시스트), STL(스틸), BLK(블로킹), TO(턴오버, 실책) 총 9 종류의 파라미터로 설정했다. 또한 target은 Draft 진출 성공을 1, 실패를 0으로 설정했다.
위 데이터를 직접구현한 방식과 사이킷런의 내장함수 LogisticRegression()과 비교하였다.

[데이터 및 학습 파라미터 설정]
- epoch (iteration) = 1000 cycle
- lamda (learning-rate) = 1e-3 (0.001)
- 활성화함수 : ReLu

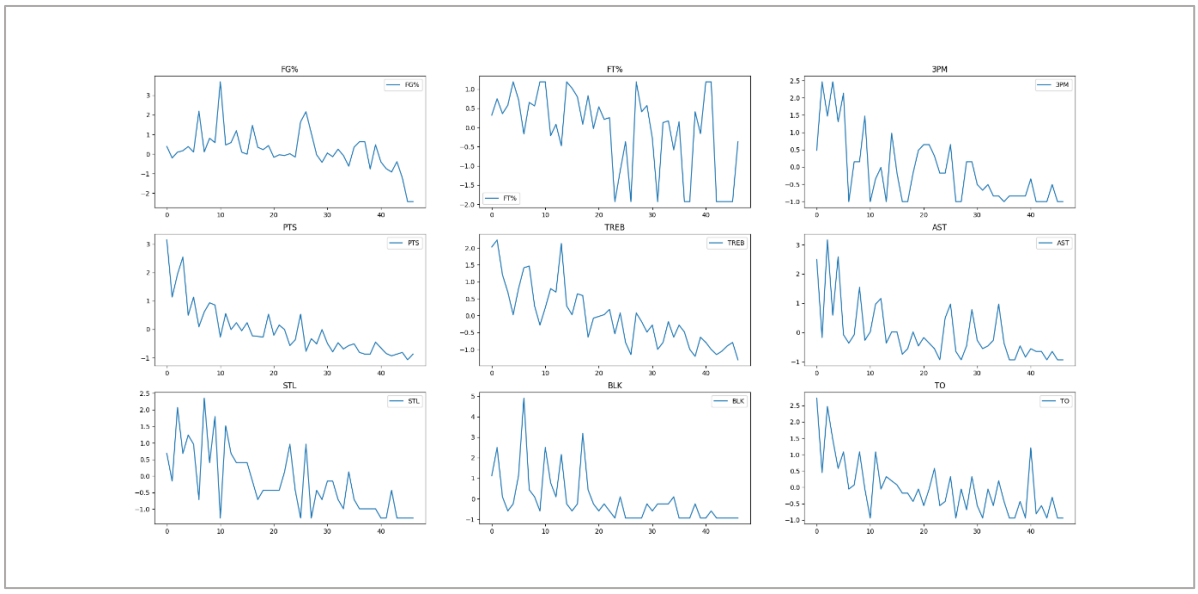
[결과 분석]
- 결과 값

Coef	W1	W2	W3	W4	W5	W6	W7	W8	W9	b
직접구현	-1.1268	-0.7607	-0.1259	0.2813	-0.0944	0.0476	-0.4722	1.3616	-0.1183	-4.42
사이킷런	0.3287	0.5091	-0.1522	0.6118	0.3457	0.6293	0.0277	0.6482	-0.2106	-4.9232

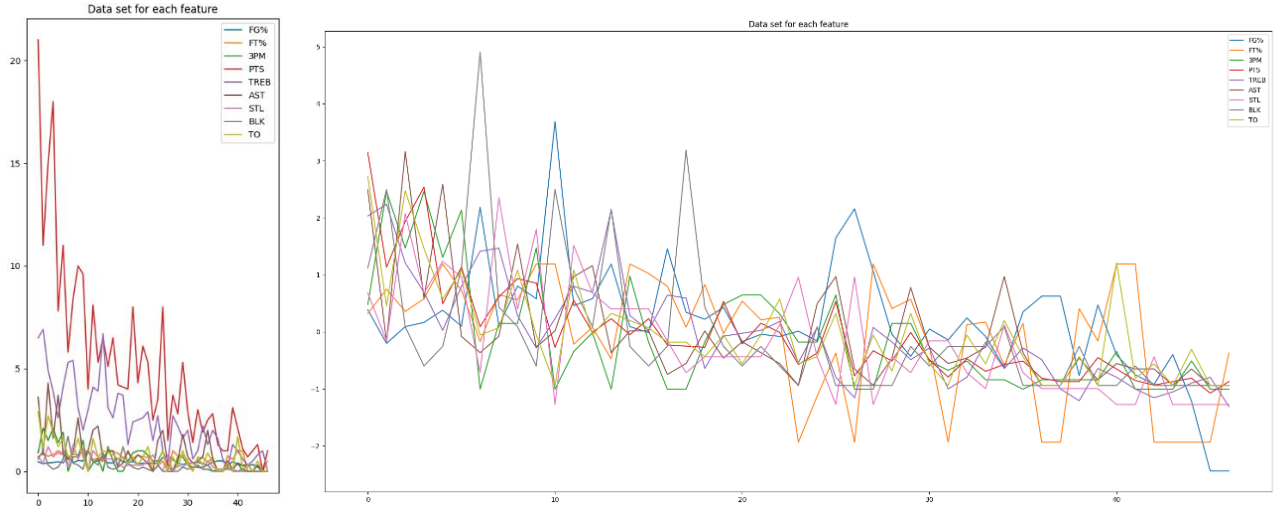
직접구현한 방식보다 사이킷런이 더 정확했다.
W1(필드골 성공률), W2(프리드로우 성공률), W4(득점수), W5(트레블링), W6(어시스트), W8(블로킹) 이 실제로도 중요한 지표이기 때문이다. 오히려 W3(3점슛), W9(실책)은 실제 경기에서 크게 작용하지 않음을 알 수 있다. 물론 포지션마다 다르긴 할 것이다.

-score
> train data set : 0.8696
> test data set : 0.8333

- 그래프
> Graph of 9 Features (independent)

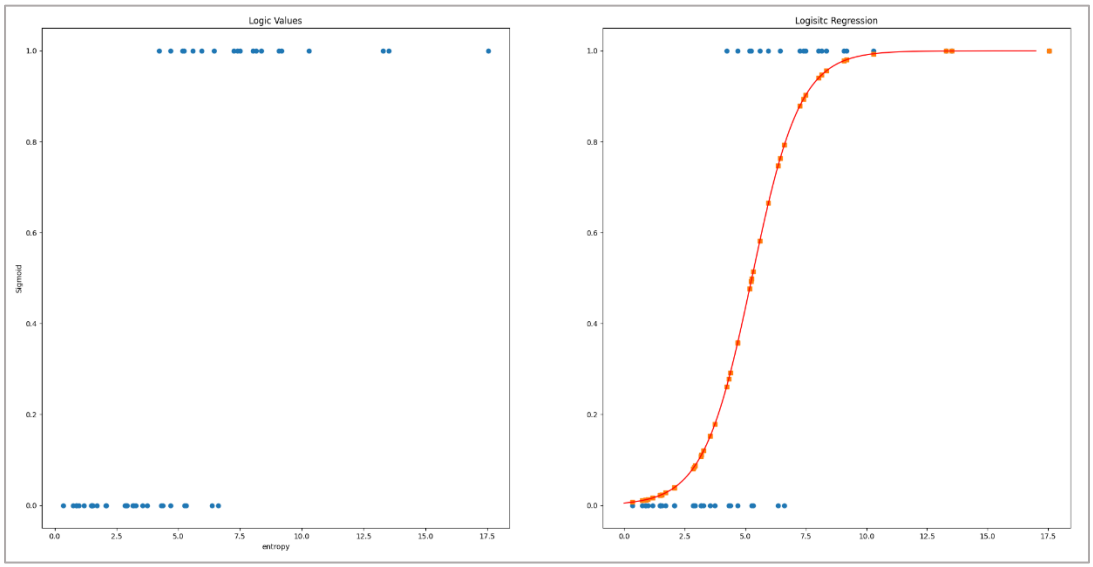


> Graph of all Features line graph



총 9개의 특성을 독립적으로 특성표현을 해보았다. 양상을 보이는 특성과, 그렇지 않은 특성이 있으며, 양상이라고 해도 우하향하는 양상일뿐 offset과 noise가 존재하기 때문에 Logistic Regression을 진행해야했다. 또한 그래프의 스케일이 달라 표준화 작업을 진행한 후 학습을 진행했다.

> Feature 을 Logistic 하여 찍은 점과 그 점을 토대로 학습시킨 Logistic Graph이다.



실제 데이터에 근접한 회귀 곡선이 나왔다.

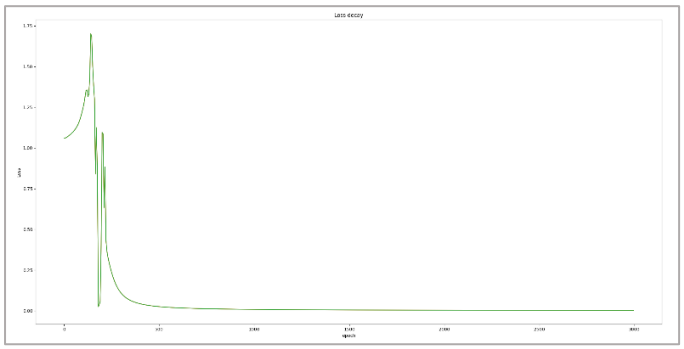
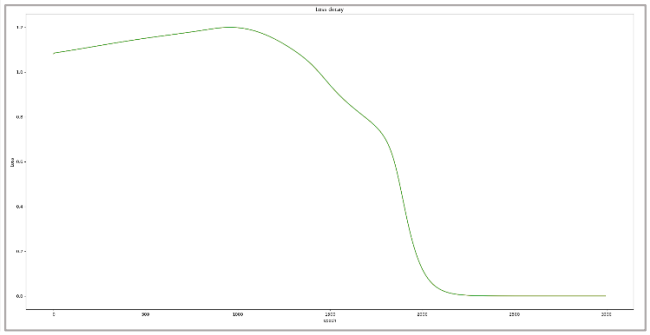
3. XOR 분류 문제를 인공신경망을 사용하여 해결하되 ReLu 활성화 함수를 적용하여 파이썬으로 코딩하기

XOR문제를 풀기 위해 활성화함수로 ReLu를 사용하여 Loss와 3D mesh plot을 통해 결과를 분석해보았다. 비교를 위해 활성화함수를 Sigmoid로 하여 비교분석도 진행했다.

[데이터 및 학습 파라미터 설정]

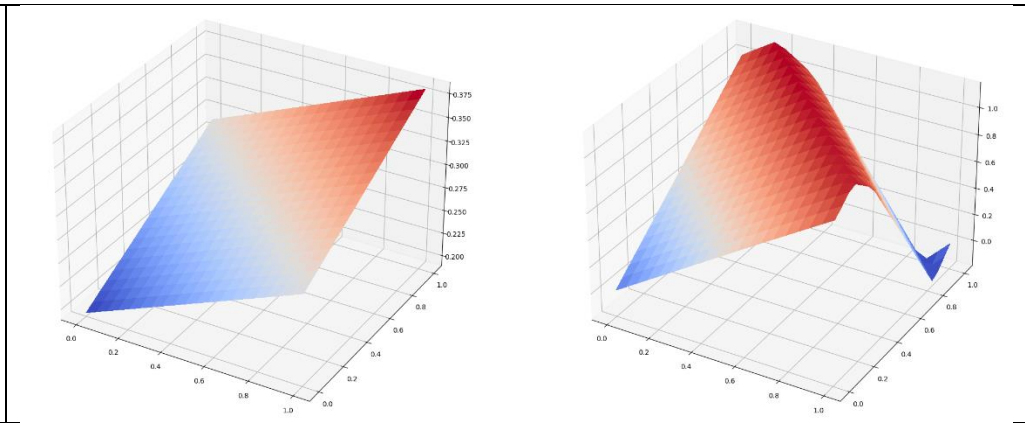
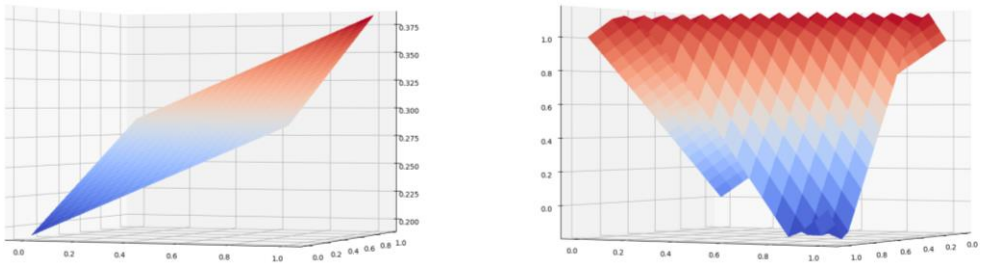
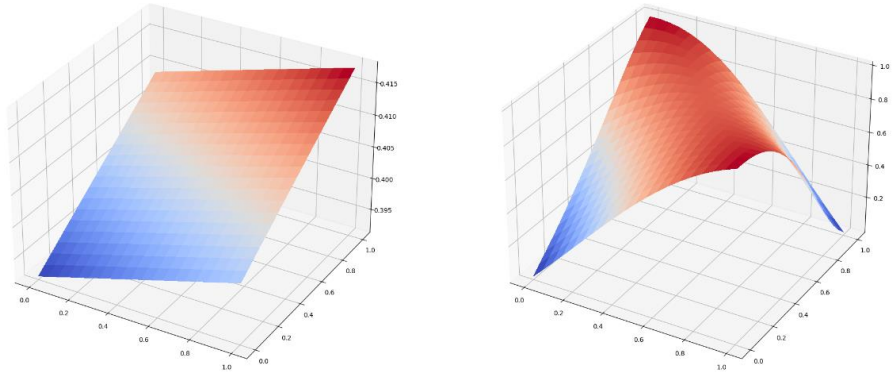
- XOR : [0,0], [0,1], [1,0], [1,1]
- hidden unit : 5
- hidden layer 1,
- learning_rate = 0.1
- epoch (iteration) = 3000

> Loss 분석

좌측이 ReLu를 사용한 경우이며, 우측이 Sigmoid를 사용한 경우이다. ReLu는 초기에 Loss의 변화가 크지만 빠르게 수렴한다는 장점이 있다. Sigmoid는 느리지만 튀지 않는다는 특징이 있다.

> 3d plot의 초기 평면(좌측)과 학습 후 바뀐 지점을 볼 수 있는 3차원 곡면(우측)을 볼 수 있다.
ReLu와 Sigmoid 모두 **convex**한 구조로 바뀌어감을 볼 수 있다. 간헐적으로 **convcave**한 구조로 바뀌는 경우도 있다.

ReLu	
ReLu (side view)	
Sigmoid	

- 결과 예측

두 활성화 함수 모델 모두 정확하게 예측함을 볼 수 있었다.

> ReLu

```
100%|██████████| 3000/3000 [00:01<00:00, 2694.70it/s]
======================================================================
Input :  [0, 0, 1] predict :  [0.00414003]
Input :  [0, 1, 1] predict :  [0.99998937]
Input :  [1, 0, 1] predict :  [0.99998745]
Input :  [1, 1, 1] predict :  [5.09789547e-07]
```

> Sigmoid

```
100%|██████████| 3000/3000 [00:00<00:00, 3008.26it/s]
======================================================================
Input :  [0, 0, 1] predict :  [2.85443558e-10]
Input :  [0, 1, 1] predict :  [1.]
Input :  [1, 0, 1] predict :  [1.]
Input :  [1, 1, 1] predict :  [4.8610449e-11]
```

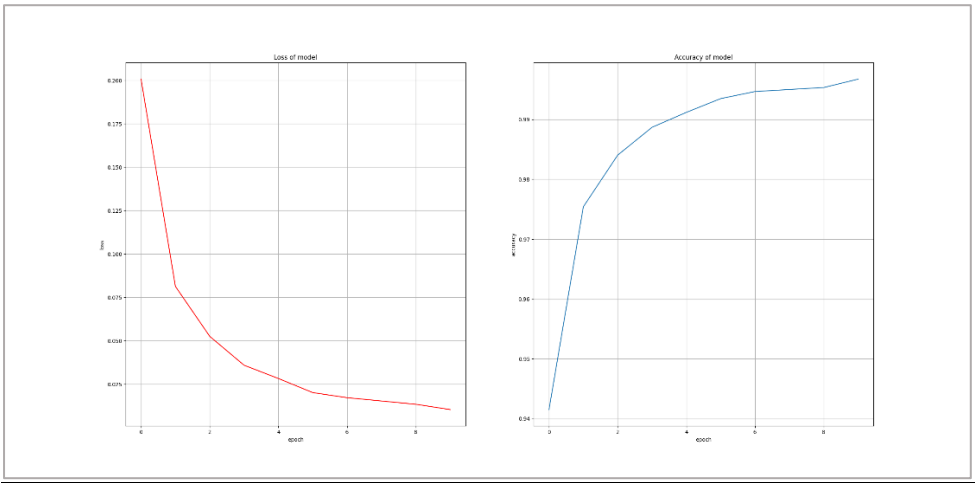
학습 모델의 Hidden Unit과 Layer를 증가시키면 더 정확할 것이다. 추가적으로 XOR처럼 데이터의 종류가 적은 모델이 아닌 다량의 데이터가 있고, NN도 개수가 많을 경우 중간중간 일부 뉴런을 비활성화하면서 학습하는 학습방법도 적용하면 더욱 좋은 모델을 만들 수 있을 것이다.

4. **MNIST 숫자 손글씨 데이터를 파이썬으로 사용하여 인공신경망 기법으로 학습하고, 자신이 직접 쓴 숫자 손글씨를 테스트 해보기**

MNIST 데이터는 keras에서 제공하는 데이터 샘플을 사용했다. 데이터의 개수는 60000개로써 각기 다른 필기체를 구분할 수 있는 data set이다. 28x28 픽셀의 데이터는 one-hot-encoding을 통해서 데이터를 전처리해주어야한다. (발산 방지를 위해 0인 데이터는 0.0001으로 설정해주었다.)

[데이터 및 학습파라미터 설정]
- epoch (iteration) : 10 (each data set, 60000 x 10 = 60만)
- 손 글씨 직접 종이에 써서 추출 (0 ~ 9)

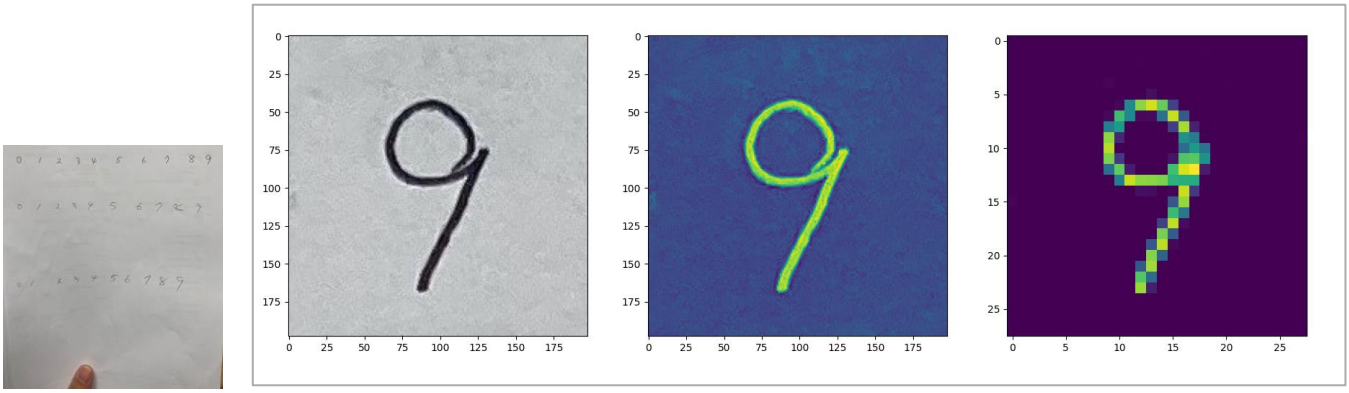
[결과 분석]
- Loss 및 Accuracy
> epoch에 따라 Loss는 감소, Accuracy는 증가하는 것을 그래프를 통해 확인할 수 있다.



	Train Data set	Test Data set
Loss	0.00312	0.99927
Accuracy	0.07290	0.98220

비교데이터(test data)의 정확도는 0.98로 충분하다. 또한 학습데이터(train data)보다는 약간 낮은 값이므로 오버피팅 되지 않은 것이라고 볼 수 있다. 따라서 이 모델은 잘 학습되었음을 알 수 있었다.
함수는 ReLu 활성화 함수를 사용했다.

> 손글씨 데이터의 이미지 처리 과정



실제 손으로 쓴 글씨를 카메라로 촬영 후 이를 28x28 사이즈의 이미지로 사용했다. 여기서 중요한 것은 데이터셋은 배경이 검정색이고, 글씨가 흰색이라는 것이다. 물론 가장 우측 이미지는 이미지 출력방식에 따라 보라색으로 표현되었지만 실제로는 검정색 배경에 흰 글씨 데이터이다.
이를 위해 실제 데이터를 흑백전환 하였으나 이는 배경이 회색이었기에 평균적으로 약 40~60 정도의 픽셀값을 가지고 있었다. 따라서 이를 대비 강조를 통해 조절 후 추출하여 사용하는 과정을 거쳤다. 이 때 28x28로 픽셀의 크기를 줄이는 과정은 모든 흑백과 대비 처리가 진행된 이후에 진행해야 한다. 만약 먼저 pixel의 크리를 줄이고 이미지 처리를 할 경우 무늬가 생길 수 있다.

이미지 처리된 데이터를 예측한 결과는 아래와 같다.

```
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0104 - accuracy: 0.9967
Loss (Train) :  0.003120852867141366  Accuracy (Train) :  0.9992666840553284
Loss (Test) :  0.07289579510688782  Accuracy (Test) :  0.982200026512146
1/1 [==============================] - 0s 37ms/step
예측된 숫자 :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

실제 숫자와 같은 값으로 예측하였다.

Source Code

1.

```
import numpy as np

from matplotlib import pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

# 1. 데이터 준비
perch_length = np.array(
    [13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0,
     21.0, 21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5,
     22.5, 22.7, 23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5,
     27.3, 27.5, 27.5, 27.5, 28.0, 28.7]
     )
perch_weight = np.array(
    [32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
     110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,
     130.0, 150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0,
     197.0, 218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0]
     )

# 2. 학습 파라미터 설정
w = np.random.rand(1)
b = np.random.rand(1)
print('init w : ', w)
print('init b : ', b)
lamda = 1e-3
iteration = 5000

# 3. 데이터 전처리
N = np.size(perch_weight)
train_input, test_input, train_target, test_target = train_test_split(perch_length,
perch_weight, random_state=42)
train_input = train_input.reshape(-1,1)
train_target = train_target.reshape(-1,1)
test_input = test_input.reshape(-1,1)
test_target = test_target.reshape(-1,1)

''' Main Algorithm'''
''' ---- START ---'''
# 4. 모델 훈련
for i in tqdm(range(0,iteration)) :
    for j in range(0, len(train_input)):
        output = np.dot(np.asarray([w,b]).T, np.asarray([train_input[j], 1]))

        dedw = train_input[j]
        dLde = output - train_target[j]
        dedb = 1
        dLdw = dLde * dedw
```

```
        dLdb = dLde * dedb

        # w = w - lamda * dLdw
        if w >= 0:
            w = w - lamda*dLdw - lamda*0.1*w
            b = b - lamda * dLdb
        else:
            w = w - lamda*dLdw + lamda*0.1*w
            b = b - lamda * dLdb

# 4.1 사이킷런 학습
lr = Lasso()
lr.fit(train_input, train_target)

''' Main Algorithm'''
''' ----- END ----'''

# 5. 결과 확인
print('Weight(w) - iteration : ', w[0][0])
print("          - sklearn   : ", lr.coef_[0])
print('Bais(b)   - iteration : ', b[0][0])
print('          - sklearn   : ', lr.intercept_[0])
print('Score (train data set) : ', lr.score(train_input, train_target))
print('Score (test data set) : ', lr.score(test_input, test_target))

# ------------------- graph --------------------------------
plt.subplot(1,2,1)
plt.title('Regression using Gradient Descent Method')
plt.xlabel('Length')
plt.ylabel('Weight')
plt.scatter(perch_length, perch_weight, marker='o')
plt.plot(perch_length, w*perch_length+b, '-r')

plt.subplot(1,2,2)
plt.title('Regression using sklearn LinearRegression Class')
plt.xlabel('Length')
plt.ylabel('Weight')
plt.scatter(perch_length, perch_weight)
plt.plot(train_input, lr.coef_ * train_input+lr.intercept_)

plt.show()
```

2.

```
import numpy as np

import matplotlib.pyplot as plt
from tqdm import tqdm
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler


# Global macro
G_USE_DATA_FROM_CSV = 0

# user Functions
def sigmoid(x):
    return 1/(1+np.exp(-x))

def loss_function(predict, y):  # cross-entropy method
    loss = -y * np.log(predict) - (1-y)*np.log(1-predict)
    return loss

# 1. 데이터 준비
feature = ('FG%','FT%','3PM','PTS','TREB','AST','STL','BLK','TO')
if G_USE_DATA_FROM_CSV:
    data = pd.read_csv('../docs/NBA Rookie DRAFT Rank.csv')
    print(data.head())

    # Split data to input & target
    data_input = data[['FG%','FT%','3PM','PTS','TREB','AST','STL','BLK','TO']].to_numpy()
    data_target = data[['Entry of draft']].to_numpy()
    train_input, test_input, train_target, test_target = train_test_split(data_input,
data_target, test_size=0.4, random_state=42)
    sc = StandardScaler()
    sc.fit(train_input)
    train_input_std = sc.transform(train_input)
    train_test_std = sc.transform(test_input)

    print(".csv is used.\n")
else:
    input = np.array([[0.461, 0.722, 0.9, 21., 6.5, 3.6, 0.7, 0.6, 2.9],
                [0.365, 0.859, 2.1, 11., 6.9, 0.8, 0.4, 1.0, 1.1],
                [0.413, 0.734, 1.5, 15., 4.9, 4.3, 1.2, 0.3, 2.7],
                [0.425, 0.805, 2.1, 18., 3.9, 1.6, 0.7, 0.1, 1.9],
                [0.460, 1.000, 1.4, 7.8, 2.6, 3.7, 0.9, 0.2, 1.2],
                [0.414, 0.852, 1.9, 11., 4.1, 0.9, 0.8, 0.6, 1.6],
                [0.755, 0.565, 0.0, 5.8, 5.3, 0.6, 0.2, 1.7, 0.7],
                [0.416, 0.828, 0.7, 8.4, 5.4, 0.9, 1.3, 0.4, 0.8],
```

```
                  [0.529, 0.799, 0.7, 10., 3.1, 2.6, 0.6, 0.3, 1.6],
                  [0.493, 1.000, 1.5, 9.6, 2.0, 0.7, 1.1, 0.1, 0.7],
                  [1.000, 1.000, 0.0, 4.0, 3.0, 1.0, 0.0, 1.0, 0.0],
                  [0.473, 0.550, 0.4, 8.1, 4.1, 2.0, 1.0, 0.5, 1.6],
                  [0.493, 0.645, 0.6, 5.3, 3.9, 2.2, 0.7, 0.3, 0.7],
                  [0.592, 0.467, 0.0, 6.5, 6.7, 0.6, 0.6, 0.9, 1.0],
                  [0.413, 1.000, 1.2, 5.1, 3.1, 1.0, 0.6, 0.2, 0.9],
                  [0.396, 0.947, 0.5, 6.5, 2.6, 1.0, 0.6, 0.1, 0.8],
                  [0.636, 0.875, 0.0, 4.2, 3.8, 0.2, 0.4, 0.2, 0.6],
                  [0.454, 0.646, 0.0, 4.1, 3.7, 0.4, 0.2, 1.2, 0.6],
                  [0.434, 0.885, 0.5, 4.0, 1.3, 1.0, 0.3, 0.4, 0.4],
                  [0.468, 0.610, 0.9, 8.0, 2.4, 0.5, 0.3, 0.2, 0.7],
                  [0.370, 0.792, 1.0, 4.3, 2.5, 0.8, 0.3, 0.1, 0.3],
                  [0.391, 0.687, 1.0, 6.1, 2.6, 0.6, 0.3, 0.2, 0.7],
                  [0.385, 0.701, 0.8, 5.3, 2.9, 0.4, 0.5, 0.1, 1.2],
                  [0.400, 0.000, 0.5, 2.5, 1.5, 0.0, 0.8, 0.0, 0.3],
                  [0.372, 0.258, 0.5, 3.5, 2.7, 1.5, 0.3, 0.3, 0.4],
                  [0.667, 0.500, 1.0, 8.0, 1.0, 2.0, 0.0, 0.0, 1.0],
                  [0.750, 0.000, 0.0, 1.5, 0.3, 0.3, 0.8, 0.0, 0.0],
                  [0.571, 1.000, 0.0, 3.7, 2.7, 0.0, 0.0, 0.0, 0.7],
                  [0.391, 0.750, 0.7, 2.8, 2.2, 0.5, 0.3, 0.0, 0.2],
                  [0.328, 0.802, 0.7, 5.3, 1.6, 1.8, 0.2, 0.2, 1.0],
                  [0.406, 0.532, 0.3, 2.9, 2.0, 0.7, 0.4, 0.1, 0.3],
                  [0.375, 0.000, 0.2, 1.4, 0.6, 0.4, 0.4, 0.2, 0.0],
                  [0.438, 0.660, 0.3, 3.0, 1.0, 0.5, 0.2, 0.2, 0.7],
                  [0.385, 0.674, 0.1, 1.9, 2.2, 0.7, 0.1, 0.2, 0.3],
                  [0.296, 0.432, 0.1, 2.5, 1.3, 2.0, 0.5, 0.3, 0.9],
                  [0.455, 0.667, 0.0, 2.8, 2.0, 0.6, 0.2, 0.0, 0.4],
                  [0.500, 0.000, 0.1, 1.3, 1.6, 0.0, 0.1, 0.0, 0.0],
                  [0.500, 0.000, 0.1, 1.0, 0.6, 0.0, 0.1, 0.0, 0.0],
                  [0.273, 0.750, 0.1, 1.0, 0.2, 0.5, 0.1, 0.2, 0.4],
                  [0.475, 0.568, 0.1, 3.1, 1.3, 0.1, 0.1, 0.0, 0.0],
                  [0.332, 1.000, 0.4, 2.1, 1.0, 0.4, 0.0, 0.0, 1.7],
                  [0.275, 1.000, 0.0, 1.1, 0.6, 0.3, 0.0, 0.1, 0.1],
                  [0.248, 0.000, 0.0, 0.7, 0.3, 0.3, 0.3, 0.0, 0.3],
                  [0.333, 0.000, 0.0, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0],
                  [0.200, 0.000, 0.3, 1.3, 0.8, 0.3, 0.0, 0.0, 0.5],
                  [0.000, 0.000, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
                  [0.000, 0.500, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
                  ])
    target = 
np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0]).T.reshape(-1,1)
    train_input, test_input, train_target, test_target = train_test_split(input, target, 
test_size=0.5, random_state=42)
    sc = StandardScaler()
    sc.fit(train_input)
    input_std = sc.transform(input)
    train_input_std = sc.transform(train_input)
    train_test_std = sc.transform(test_input)
    print("manual data is used\n")

# 2. 학습 파라미터 설정
learning_rate = 1e-3
```

```
iteration = 1000
W = np.random.rand(train_input.shape[1],1)  # shape[0] : rows, shape[1] : columns
b = np.random.rand(1,1)
print('init w : ', W.T)
print('init b : ', b)

# 3. 모델 학습
for i in tqdm(range(0,iteration)):
    output = np.dot(train_input,W)
    predict = sigmoid(output)

    Loss = loss_function(predict,train_target)
    Loss = np.sum(Loss)

    # get dL/dW & dL/db
    dLdsig = -1 * ( train_target/predict - (1-train_target)/(1-
predict) )    #dLoss()/dsigmoid()
    dsigdy = predict * (1-predict)                                    #dsigmoid()/dy
    dLdW = np.dot(train_input.T, dLdsig*dsigdy)
    dLdb = np.sum(dLdsig * dsigdy)

    # Update W, b
    W = W - learning_rate * dLdW
    b = b - learning_rate * dLdb

# 3.1 사이킷런 학습 Using sklearn Class
lr = LogisticRegression(C=1, max_iter=iteration)
lr.fit(train_input, train_target)
pred_lr = lr.predict(test_input)
pred_proba = lr.predict_proba(test_input)
precision = precision_score(test_target, pred_lr)
conf_matrix = confusion_matrix(test_target, pred_lr)
class_report = classification_report(test_target, pred_lr)

# 4. 결과 확인
print(feature)
print('Weight(w) - iteration : ', W.T)
print("          - sklearn   : ", lr.coef_[0])
print('Bais(b)   - iteration : ', b[0][0])
print('          - sklearn   : ', lr.intercept_[0])
print('Score (train data set) : ', lr.score(train_input, train_target))
print('Score (test data set) : ', lr.score(test_input, test_target))

print('learning finish')

# ------------------- graph ---------------------------------

y = np.dot(input, lr.coef_.T)
# print(y)
columns, rows = input.shape
k= input[:,0]
c = np.arange(0,columns,1)

plt.figure("NBA Rookie Draft Ranking")
```

```
plt.title("Data set for each feature")
for i in range(0,rows):
    # plt.scatter(c, input[:,i], label=feature[i])
    plt.plot(c, input_std[:, i], label=feature[i])
plt.legend()

plt.figure("Feature Detail")
plt.subplot(3,3,1)
# plt.ylim(-2, 2)
for i in range(0,rows):
    # plt.scatter(c, input[:,i], label=feature[i])
    plt.subplot(3,3,i+1)
    plt.title(feature[i])
    plt.plot(c, input_std[:, i], label=feature[i])
    plt.legend()

plt.figure("Logistic Regression Analysis")
plt.subplot(1,2,1)
plt.title("Logic Values")
plt.scatter(y, target, marker="o")
plt.xlabel('entropy')
plt.ylabel('Sigmoid')

plt.subplot(1,2,2)
xn = np.arange(0, 17, 0.01)
yn = np.exp(-xn-b[0])/(1+np.exp(-xn-b[0]))
plt.title("Logisitc Regression")
plt.scatter(y, target, marker="o")
plt.scatter(y, 1/(1+np.exp(-y-lr.intercept_)), marker= 's')
plt.plot(xn, 1/(1+np.exp(-xn-lr.intercept_)), '-r')
# plt.plot(xn,yn)

# sample_data = np.array([[0.3, 0.7, 0.8, 2.7, 1.2, 0.3, 0.4, 0.1, 0.8]])
sample_data = np.array([[0.460, 1.000, 1.4, 7.8, 2.6, 3.7, 0.9, 0.2, 1.2]])

predict_sample = np.dot(sample_data, W.ravel())
# print( np.exp(predict_sample+b[0])/(1+np.exp(predict_sample+b[0])))
# print(1/(1+np.exp(-predict_sample-b[0])))
# print(lr.predict(sample_data))



plt.show()
print("process finished")
```

3.

```
import numpy as np

import matplotlib.pyplot as plt
from tqdm import tqdm

# 1. 데이터 준비
input = [[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]]
output = [0, 1, 1, 0]

N = np.size(input, 0)  # number of samples
Ni = np.size(input, 1)  # dimension of the samples of input
No = 1  # dimension of the sample of output
Nh = 5  # number of hidden units
Ws = 1 / 4 * np.random.rand(Nh, Ni + 1)
# print(Ws)
Wo = 1 / 4 * np.random.rand(No, Nh)

# 2. 학습 파라미터 설정
alpha = 0.1  # Learning rate
iteration = 3000

t_ = []
loss_ = []

def ReLu(x):
    return np.maximum(0,x)

# 3. 모델 훈련
for epoch in tqdm(range(0, iteration)):
    loss = 0
    for id_ in range(0, N):
        dWs = 0 * Ws
        dWo = 0 * Wo

        x = np.append(input[id_], 1)
        S = np.dot(Ws, x)
        y = np.dot(Wo, ReLu(S))
        d = output[id_]

        for j in range(0, Nh):
            for i in range(0, No):
                dWo[i, j] = dWo[i, j] + ReLu(S[j]) * (y[i] - d)

        Wo = Wo - alpha * dWo

        for k in range(0, Ni + 1):
            for j in range(0, Nh):
                for i in range(0, No):
                    dWs[j, k] = dWs[j, k] + x[k] * Wo[i, j] * ReLu(S[j]) * (1 - ReLu(S[j]))
* (y[i] - d)
```

```
        Ws = Ws - alpha * dWs

        loss = loss + 1 / 2 * np.linalg.norm(y - d)


##------------------------- for display graph ---------------------------------------
    if np.mod(epoch, 5) == 0:
        # print(epoch, "-th epoch trained")
        t_ = np.append(t_, epoch)
        loss_ = np.append(loss_, loss)

    if (epoch == 0) or (epoch == iteration-1) :

        plt.figure(num=0, figsize=[10, 5])
        plt.plot(t_, loss_, marker="")
        plt.title('Loss decay')
        plt.xlabel('epoch')
        plt.ylabel('Loss')

        ## figure out the function shape the
model=========================================
        xn = np.linspace(0, 1, 20)
        yn = np.linspace(0, 1, 20)
        xm, ym = np.meshgrid(xn, yn)
        xx = np.reshape(xm, np.size(xm, 0) * np.size(xm, 1))
        yy = np.reshape(ym, np.size(xm, 0) * np.size(xm, 1))
        Z = []

        for id__ in range(0, np.size(xm)):
            x = np.append([xx[id__], yy[id__]], [1, 1])
            S = np.dot(Ws, x)
            y_ = np.dot(Wo, ReLu(S))
            Z = np.append(Z, y_)

        if epoch==0:
            fig = plt.figure(num=1, figsize=[10, 5])
            ax = fig.add_subplot(121, projection='3d')
            surf = ax.plot_surface(xm, ym, np.reshape(Z, (np.size(xm, 0), np.size(xm, 1))),
cmap='coolwarm', linewidth=0,
                                   antialiased=False)
        if epoch==iteration-1:
            ax = fig.add_subplot(122, projection='3d')
            surf = ax.plot_surface(xm, ym, np.reshape(Z, (np.size(xm, 0), np.size(xm, 1))),
cmap='coolwarm', linewidth=0,
                                   antialiased=False)
print("======================================================================")

# 4. 결과 확인
for id_ in range(0, N):
    x = np.append(input[id_], 1)
    S = np.dot(Ws, x)
    y = np.dot(Wo, ReLu(S))
    print("Input : ", input[id_], "predict : ", y)
```

```
# -------------------- graph --------------------------------
plt.figure(num=0, figsize=[10, 5])
plt.plot(t_, loss_, marker="")
plt.title('Loss decay')
plt.xlabel('epoch')
plt.ylabel('Loss')

plt.show()

print('end')
```

4.

```
import matplotlib.pyplot as plt

import tensorflow as tf
import numpy as np
from os.path import exists
import pandas as pd


# 1. 데이터 준비
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

handwrite_data = np.loadtxt('../docs/HandwriteNumber/csv/data_total_grayscale_C2.csv',
delimiter=',')
print("handwrite data is opened")
x_handwrite_data = handwrite_data[:, 1:].reshape(10,28,28)
y_handwrite_data = handwrite_data[:, 0]

# 2. 데이터 전처리
x_train, x_test = (x_train/255.0 + 0.001), (x_test/255.0 + 0.001)

# 3. 모델 구성
file_exist_flag = exists("../src_model/XOR_ANN_epoch1.h5")
if file_exist_flag :
    model = tf.keras.models.load_model('../src_model/XOR_ANN_epoch10.h5')
else:
    model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                        tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
                                        ])
# 4. 모델 컴파일
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
# 5. 모델 훈련
    history = model.fit(x_train, y_train, epochs=10, verbose=1)

# 6. 정확도 평가
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=0)
print("Loss (Train) : ", train_loss, " Accuracy (Train) : ", train_acc)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("Loss (Test) : ", test_loss, " Accuracy (Test) : ", test_acc)

# 모델 저장
model.save("../src_model/XOR_ANN_epoch10.h5")

# 7. 손글씨 데이터 예측
predict = model.predict(x_handwrite_data)
pr = []
for i in range(0,10):
    pr.append(np.argmax(predict[i]))
```

```
print("예측된 숫자 : ", pr)

history_loss = history.history['loss']
history_acc = history.history['accuracy']

# ------------------- graph ---------------------------------
plt.subplot(1,2,1)
plt.title("Loss of model")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.plot(range(len(history_loss)), history_loss, 'r')
plt.grid()
plt.subplot(1,2,2)
plt.title("Accuracy of model")
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.plot(range(len(history_acc)), history_acc)
plt.grid()

plt.show()

print("end")
```

.