기계학습의 원리와 응용 중간과제

2022021585 이재윤

Department of Computer Science and Engineering, Korea University

IDS508: 기계학습의원리와응용

1. LASSO를 파이썬 코딩으로 구현하기

LASSO regression이 Ridge regression과 다른 점은 최적해를 구할 때 L2 제약식이 아닌 L1 제약식을 적용하여 가중치 w를 업데이트할 때 w의 부호에 따라 λ값이 다르게 업데이트 되도록 하는 것이다.

LASSO를 적용하기 위해 사용된 데이터는 서울 시민의 동절기 코로나 추가 예방접종 데이터를 사용하였으며 X축으로는 접종일, Y축으로는 4차 접종률로 나타내었다. 시간에 따라 점점 접종률이 높아질 것이므로 regression을 통해 구하기에 적합하다고 판단하였다. 이에 작성된 코드와 데이터 차트는 다음과 같다. (모든 코드는 구글의 colaboratory 환경에서 작성되었다.)

```
1 import numpy as np
2 import pandas as pd
3 from google.colab import files
4 import matplotlib.pyplot as plt
5 %matplotlib inline
```

```
[ ]  1 uploaded = files.upload()
```

파일 선택 선택된 파일 없음 Upload widget is only

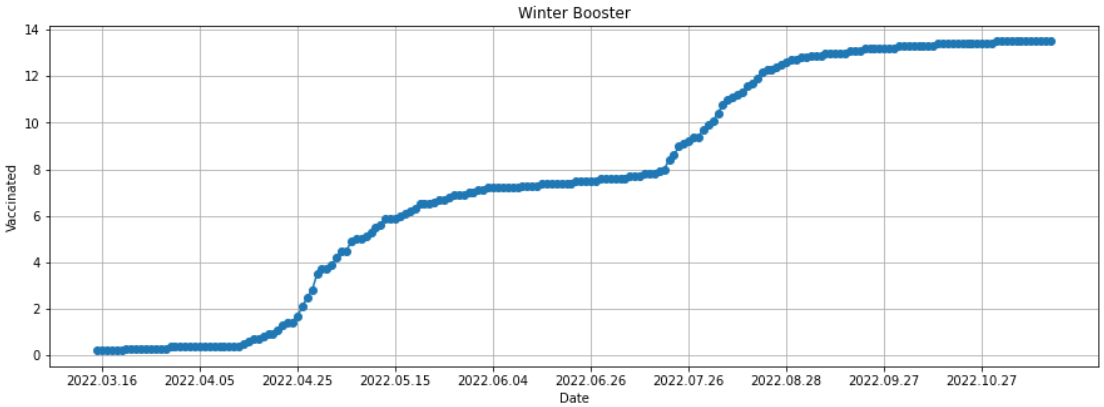
Saving vaccine.csv to vaccine (3).csv

```
[ ]  1 df = pd.read_csv("vaccine.csv", encoding='cp949')
```

```
[ ]  1 data = df.loc[0:196, ["접종일", "4차접종률(%)"]]
```

```
[ ]  1 x = data["접종일"][::-1]
     2 x = x.to_numpy()
     3 y = data["4차접종률(%)"][::-1]
     4 y = y.to_numpy()
```

```
[ ]  1 plt.figure(num=0,figsize=[15,5])
     2 plt.plot(x,y,marker="o")
     3 plt.title("Winter Booster")
     4 plt.xlabel("Date")
     5 plt.ylabel("Vaccinated")
     6 plt.xticks(np.arange(1, 201, 20))
     7 plt.grid()
     8 plt.show()
```



가중치 값이 폭발적으로 커지는 것을 방지하기 위해 최적의 α 값과 λ 값을 찾으려고 노력한 결과 α 값은 0.00000005, λ 값은 0.1 수준으로 설정하였을 때 적절한 것으로 나타났다.

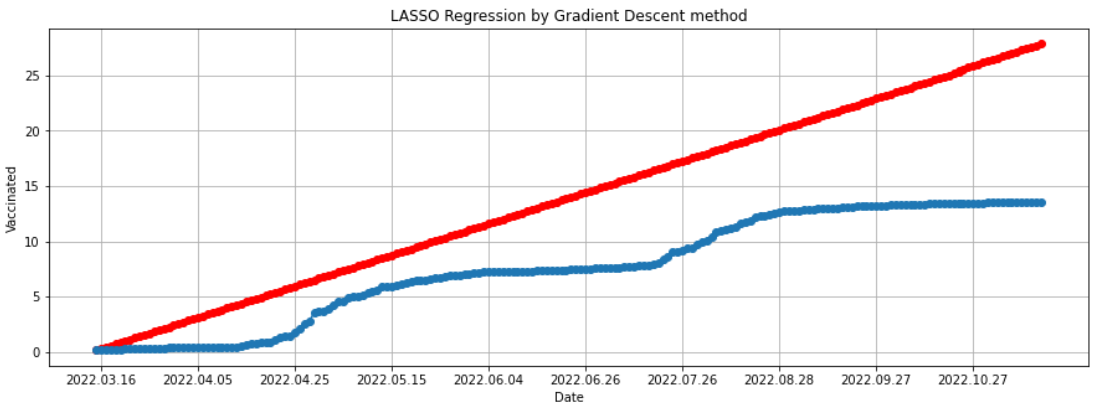
```
1 w = 1/6*np.random.rand(1)
2 b = 1/6*np.random.rand(1)
3
4 alpha = 0.00000005
5 lamb = 0.1
6
7 N = np.size(y)
8 x_idx = np.arange(0, np.size(x))
9
10 input_ = x_idx
11
```

Gradient descent를 적용하여 regression을 수행하였으며, 이 때 LASSO를 적용하여 w값의 부호에 따라 업데이트 되는 값을 다르게 설정해주었다.

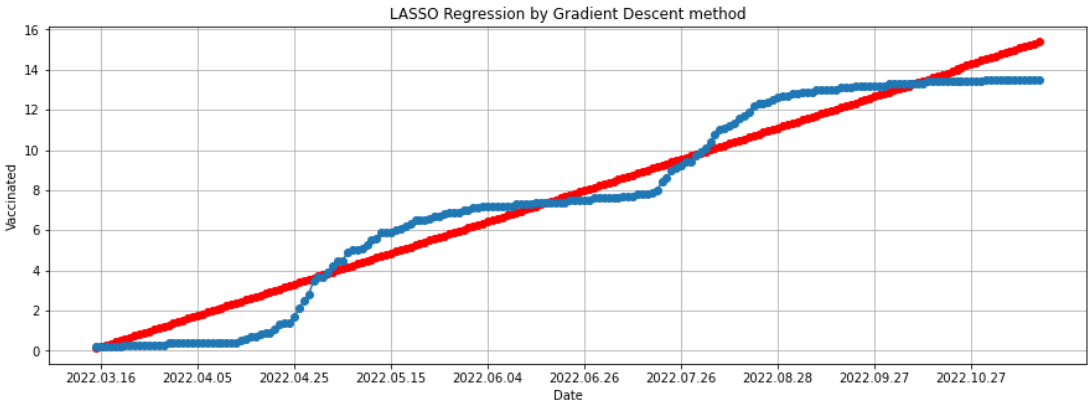
이에 대한 코드와 그 결과값은 다음과 같다.

```
for i in range(0,50):
    for k in range(0,N):
        output = np.dot(np.asarray([w,b]).T,np.asarray([input_[k],1]))

        dedw = input_[k]

        dLde = output - y[k]

        dedb = 1

        dLdw = dLde*dedw

        dLdb = dLde*dedb

        if w > 0:
            w = w - alpha*dLdw - alpha*lamb
        elif w < 0:
            w = w - alpha*dLdw + alpha*lamb

        b = b - alpha*dLdb

    plt.figure(num=0,figsize=[15,5])
    plt.plot(x,w*x_idx+b,'-r', x,y,marker="o")
    plt.title('LASSO Regression by Gradient Descent method')
    plt.xlabel("Date")
    plt.ylabel("Vaccinated")
    plt.xticks(np.arange(1, 201, 20))
    plt.grid()
    plt.show()

print("weight: ", w)
print("bias: ", b)
```

초기에 설정된 regression 결과



반복문을 통해 얻어진 최종 regression 결과




```
weight:  [0.07786088]
bias:  [0.1232113]
```

2. Logistic Regression 문제를 파이썬 코딩으로 구현하되 경사하강법을 적용하기

Logistic regression을 통해 binary classification 문제를 해결하기에 적합하다고 판단하여 scikit-learn에서 기본적으로 제공하고 있는 데이터셋인 breast cancer 데이터셋을 가져왔다. 이에 대한 전처리 과정의 코드는 다음과 같다.

```
[ ]  1 from sklearn import datasets
     2 raw_cancer = datasets.load_breast_cancer()
```

```
[ ]  1 X = raw_cancer.data
     2 y = raw_cancer.target
```

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 X_tn, X_te, y_tn, y_te = train_test_split(X,y,random_state=0)
```

```
[ ]  1 from sklearn.preprocessing import StandardScaler
     2 std_scale = StandardScaler()
     3 std_scale.fit(X_tn)
     4 X_tn_std = std_scale.transform(X_tn)
     5 X_te_std = std_scale.transform(X_te)
```

이후 전처리된 데이터를 경사하강법을 적용한 logistic regression에 적용하기 위해 여러 parameter 값과 sigmoid 함수를 정의하였다.

```
import numpy as np
import math

W_new = np.zeros(30)
W0_new = 0
a = 1e-8
epochs = 60

MSE = np.array([])

def sigmoid(output):
    z = 1/(1+math.exp(-output))
    return z

```

여기서 설정된 최적의 epoch 값은 각 epoch를 시행하면서 얻어진 결과값의 정확도를 통해 과적합이 되지 않도록 적절한 epoch 값을 찾으면서 얻어냈다.

```
for epoch in range(epochs):

    p_preds = np.array([])
    p_pred_exps = np.array([])
    error = np.array([])
    error_x = np.zeros(30)
    p_class = np.array([])

    W = W_new
    W0 = W0_new

    for x in X_tn:

        p_pred = W0 + np.dot(W, x)
        p_preds = np.append(p_preds, p_pred)

        p_pred_exp = sigmoid(p_pred)
        p_pred_exps = np.append(p_pred_exps, p_pred_exp)

        if p_pred_exp > 0.5:
            p_class = np.append(p_class,1)
        else:
            p_class = np.append(p_class,0)

```

```
    error = p_pred_exps - y_tn
    error_x = np.dot(error, X_tn)

    MSE_val = (error).mean()
    MSE = np.append(MSE, MSE_val)

    W0_new = W0 - a*np.sum(error)

    for i in range(30):
        W_new[i] = W[i] - a*np.sum(error_x[i])

    # count = 0
    # if (epoch % 5 == 0):
    #     for i in range(len(p_class)):
    #         if ((p_class[i] - y_tn[i]) != 0):
    #             count += 1
    #     print(count, epoch)

```

위의 주석 처리한 부분이 최적의 epoch를 찾기 위해 정확도를 비교한 코드이다.

최종적으로 학습된 logistic regression 모델을 테스트 데이터셋에 대한 정확도를 평가하기 위해 testset_score 함수를 구현하였다.

```
def testset_score(X, Y):

    p_preds = np.array([])
    p_pred_exps = np.array([])
    p_class = np.array([])

    W = W_new
    W0 = W0_new

    for x in X:

        p_pred = W0 + np.dot(W, x)
        p_preds = np.append(p_preds, p_pred)

        p_pred_exp = sigmoid(p_pred)
        p_pred_exps = np.append(p_pred_exps, p_pred_exp)

        if p_pred_exp > 0.5:
            p_class = np.append(p_class,1)
        else:
            p_class = np.append(p_class,0)

```

```
79      TP = 0
80      FP = 0
81      TN = 0
82      FN = 0
83
84      for i in range(len(p_class)):
85          if ((p_class[i] == 1) and (Y[i] == 1)):
86              TP += 1
87          elif ((p_class[i] == 1) and (Y[i] == 0)):
88              FP += 1
89          elif ((p_class[i] == 0) and (Y[i] == 0)):
90              TN += 1
91          elif ((p_class[i] == 0) and (Y[i] == 1)):
92              FN += 1
93
94      print(f'TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}')
95
```

학습된 모델을 통해 얻어진 분류값과 실제값을 비교하여 true positive, false positive, true negative, false negative 값을 출력하도록 정의하였다. 그 결과는 다음과 같다.

```
96 testset_score(X_te, y_te)

TP: 82, FP: 10, TN: 43, FN: 8
```

올바르게 분류한 결과값이 125개, 틀리게 분류한 결과값이 18개로 준수한 정확도를 보여주었다. Precision은 TP/(TP+FP)=82/(82+10)로 약 0.8913 수준이었으며 Recall은 TP/(TP+FN)=82/(82+8)로 약 0.9111 수준으로 나타났다.

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

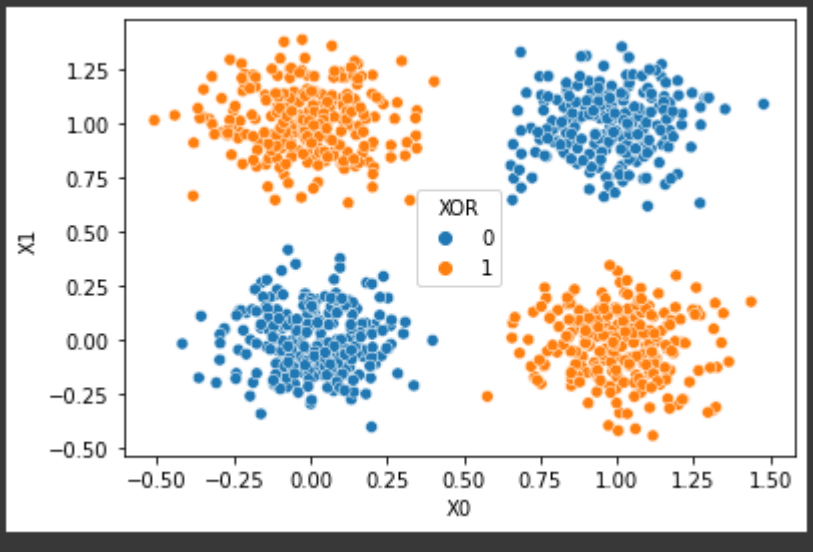
XOR 문제를 해결하기 위해서는 모델이 (1, 0), (0, 1)에 대해서는 1로, (0, 0), (1, 1)에 대해서는 0으로 올바르게 분류할 수 있어야 한다. 이는 선형 회귀로는 해결할 수 없으므로 인공신경망을 통해서 해결해야 하는데, 이를 위해 XOR로 분류되어 있는 noisy한 데이터를 가져와 인공신경망을 학습시켰다. 사용된 데이터의 형태와 코드는 다음과 같다.

```
[ ]  1 import numpy as np
     2 import pandas as pd
     3 import seaborn as sns
     4 from google.colab import files
```

```
[ ]  1 uploaded = files.upload()
```

```
파일 선택  선택된 파일 없음        Upload widget is only available w
Saving XOR.csv to XOR (2).csv
```

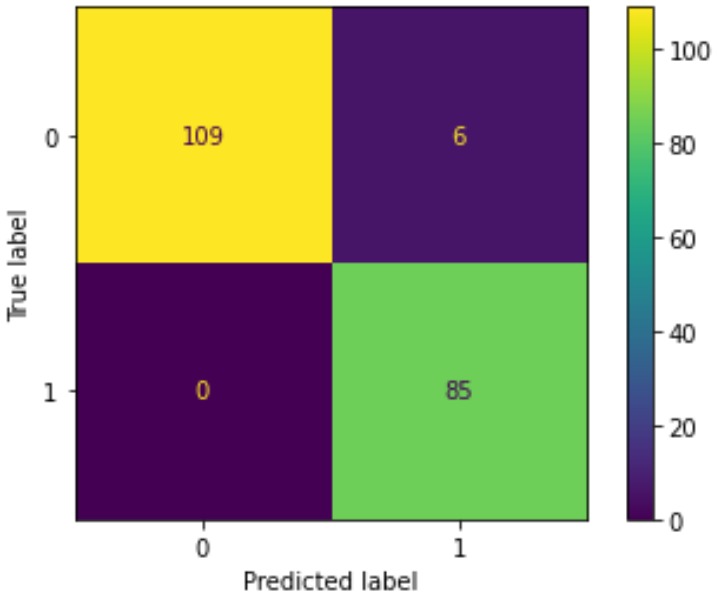
```
1 df = pd.read_csv('XOR.csv')
2 sns.scatterplot(x = df['X0'], y = df['X1'], hue = df['XOR']);
```



해당 데이터를 학습시키기 위해 scikit-learn의 인공신경망 모델인 MLPClassifier를 사용하였다. 이 때, 모델의 activation function으로는 ReLU 함수를 적용시켜주었다. 테스트 데이터셋은 전체의 0.2로 설정하였으며 이에 대한 코드와 해당 모델의 confusion matrix로 나타낸 결과는 다음과 같다.

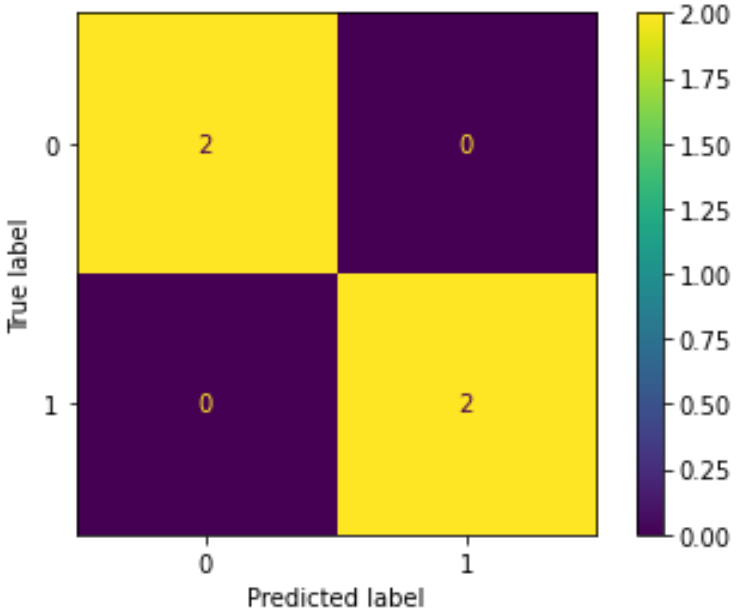
```
1 from sklearn.neural_network import MLPClassifier
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import plot_confusion_matrix
4
5 mlp = MLPClassifier(hidden_layer_sizes=8, activation='relu')
6
7 X = df[['X0','X1']]
8 y = df['XOR']
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
10
```

```
mlp.fit(X_train, y_train)

plot_confusion_matrix(mlp, X_test, y_test)

```



추가로 단순한 XOR 데이터에 대해서도 confusion matrix로 나타내보았다.

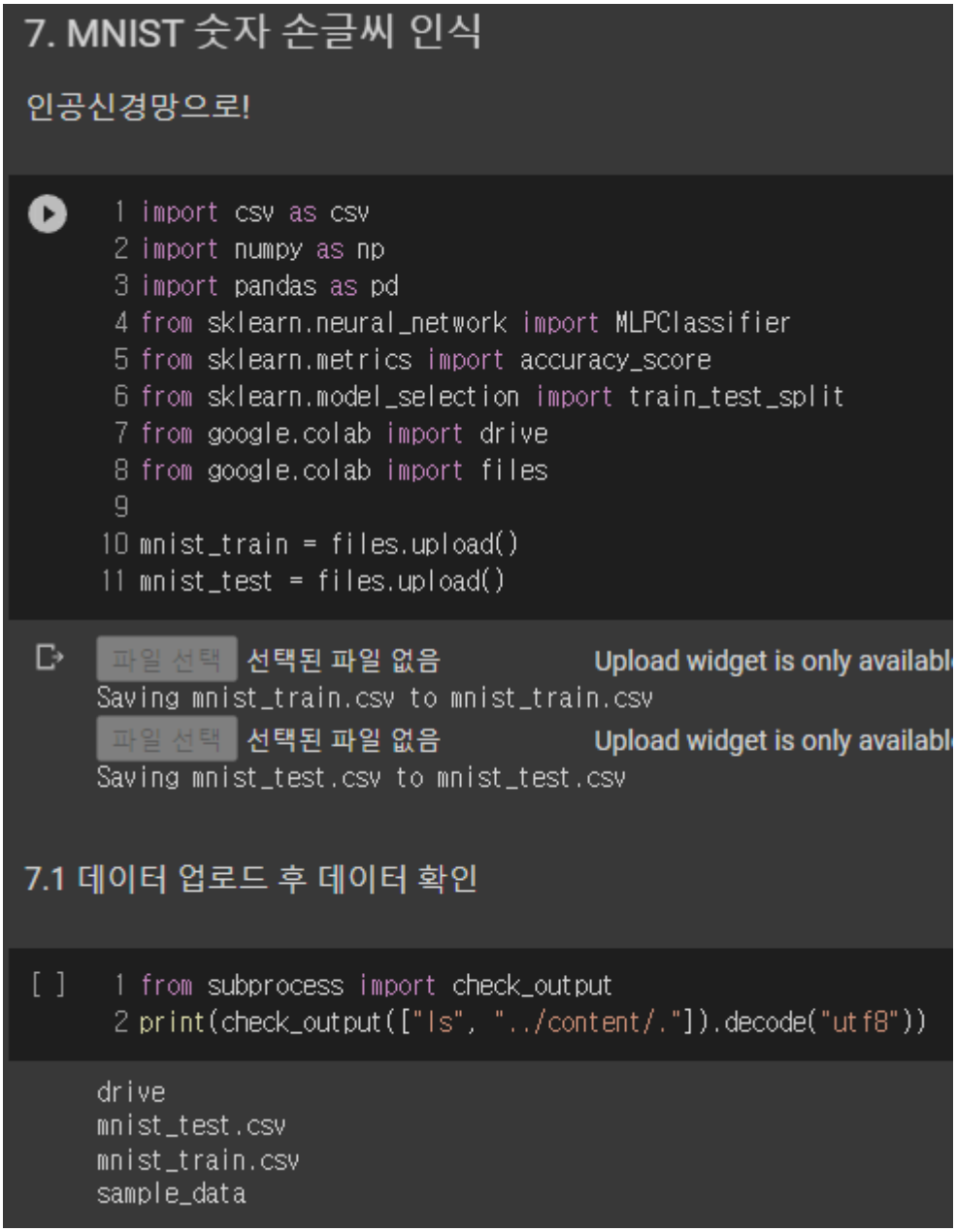
```
X_te = np.array([[1, 0], [0, 1], [0, 0], [1, 1]])
y_te = np.array([[1], [1], [0], [0]])

plot_confusion_matrix(mlp, X_te, y_te)
```



그 결과 XOR 데이터에 대해 정확한 결과값을 도출하는 것을 확인하였다.

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

해당 과제는 주어진 ipynb 파일의 내용을 그대로 따라가며 실행하였다. 모델의 학습에 사용된 MNIST 데이터는 강의 자료에 나와있는 캐글 페이지가 접속되지 않아 구글링을 통하여 얻었다. 작성된 코드는 다음과 같다.



모든 코드는 구글의 colaboratory 환경에서 실행되었다.

7.2 pandas 활용 데이터 구조 관찰하기

head() 메서드 활용!

```
[ ]  1 import csv as csv
     2 import pandas as pd
     3
     4 train_df = pd.read_csv("mnist_train.csv",header=0)    # find the directory where the data exists.
     5 train_df.head()
```

	5	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.608	0.609	0.610	0.611	0.612	0.613	0.614	0.615	0.616	0.617
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

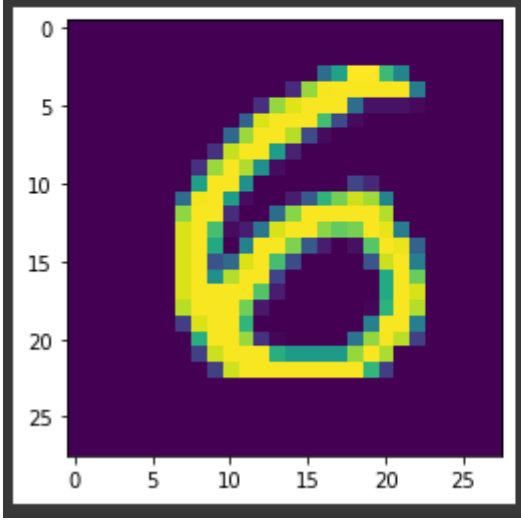
5 rows × 785 columns

주어진 데이터는 5*785의 형태로 저장된 것을 확인할 수 있었다.

```python
import csv as csv
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pickle

#########################################################################
# 1. Data load and organization
train_df = pd.read_csv("mnist_train.csv",header=0)     # find the directory where the data exists.
print(train_df.shape)

train_data = train_df.values
# data size 확인하기
print(train_data[0::,0].shape)
print(train_data[0::,1::].shape)

X_train, X_test, y_train, y_test = train_test_split(train_data[0::,1::], train_data[0::,0], test_size=0.2, random_state=0)

# show an image
print("Image of a MNIST handwrite digit")
plt.imshow(np.reshape(X_train[8,::],(28,28)))


#########################################################################
# 2. MLP learning
clf = MLPClassifier(solver='sgd')  # clf  : (설명보기: 스페이스바) MLPClassifier 클래스의 객체
clf.fit(X_train, y_train)   # train!
neural_output = clf.predict(X_test) # 테스트 데이터의 input data를 입력해주면 output label을 예측한다
print("sgd")
print(accuracy_score(y_test, neural_output)) # 테스트 데이터의 output label과 비교해서 정확도를 산출해준다


#########################################################################
# 3. save the model to disk
filename = 'finalized_model.sav'
pickle.dump(clf, open(filename, 'wb'))  # pickle 패키지를 사용


```

```
41 ##############################################################################
42 # 4. save the predicted output label
43 output = neural_output
44 predictions_file = open("neural_output.csv", "w")
45 open_file_object = csv.writer(predictions_file)
46 ids = range(neural_output.__len__())
47 ids = [x+1 for x in ids]
48 open_file_object.writerow(["ImageId", "Label"])
49 open_file_object.writerows(zip(ids, output))
50 predictions_file.close()
51 print('Saved "neural_output" to file.')
52
53
54
55
```

```
(59999, 785)
(59999,)
(59999, 784)
Image of a MNIST handwrite digit
sgd
0.9204166666666667
Saved "neural_output" to file.
```



모델의 정확도는 0.92 수준으로 나타났다.

7.4 저장한 모델 다시 불러와서 적용하기!

```
import csv as csv
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pickle

train_df = pd.read_csv("mnist_train.csv",header=0)   # find the directory where the data exists.
train_data = train_df.values

X_train, X_test, y_train, y_test = train_test_split(train_data[0::,1::], train_data[0::,0], test_size=0.2, random_state=0)


#####################################################################
# load the model from disk
filename = 'finalized_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
# predict by saved model
neural_output = loaded_model.predict(X_test)
print("sgd_from_saved_model")
print(accuracy_score(y_test, neural_output))

```

```
sgd_from_saved_model
0.9204166666666667
```

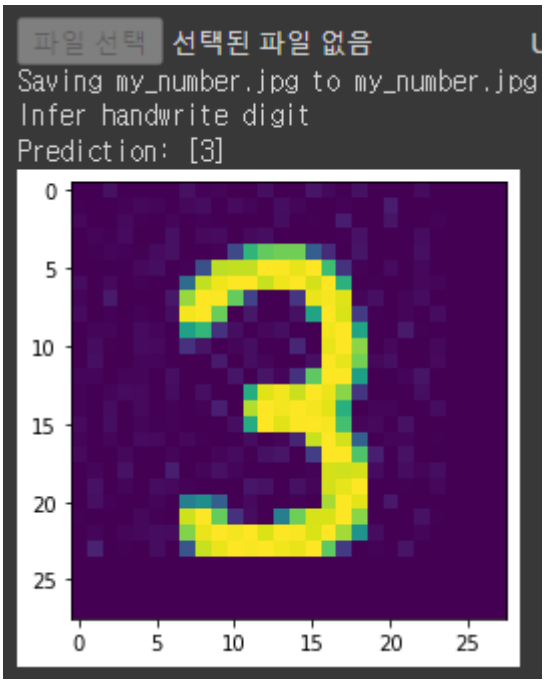
저장된 모델의 정확도가 0.92인 것을 통해 동일한 모델임을 확인할 수 있었다.

7.5 MNIST로 학습한 모델 불러와서 자신의 숫자 손글씨 쓰기에 적용하기!

```
import csv as csv
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pickle
from PIL import Image
import matplotlib.pyplot as plt

my_number = files.upload()

#########################################################################################
# 1. my handwriting image loading
im = plt.imread('my_number.jpg')    # find the directory where the data exists.
im = im[:,:,1]
#im = im.astype('float32') / 255.
im = im.astype('float32')


im = np.array(im)
im = - im + 1
plt.imshow(im)  # show the image

```

```
25 ##########################################################################
26 # 2. load the model from disk
27 filename = 'finalized_model.sav'
28 loaded_model = pickle.load(open(filename, 'rb'))
29 # print(np.reshape(im, (-1, 28*28)).size)
30 neural_output = loaded_model.predict(np.reshape(im, (-1,28*28)))
31 print("Infer handwrite digit")
32 print("Prediction:", neural_output)
33
```

숫자 3을 28*28 크기의 캔버스에 손으로 적어 모델에 입력한 결과 다음과 같이 출력되었다.



손으로 적은 숫자 3이 정확하게 3으로 인식된 것을 확인할 수 있었다.