머신러닝 프로젝트 결과보고서

[MNIST dataset을 이용한 머신러닝 모델 최적화 및 분석]

기계배움 : 전민재, 유승종, 문준원, 박태성

1. 프로젝트 개요 및 목표

MNIST 데이터베이스 (Modified National Institute of Standards and Technology database)는 손으로 쓴 숫자들로 이루어진 대형 데이터베이스이며, 다양한 화상 처리 시스템을 트레이닝하기 위해 일반적으로 사용된다. 이 데이터베이스 또한 기계 학습 분야의 트레이닝 및 테스트에 널리 사용된다. NIST의 오리지널 데이터셋의 샘플을 재 혼합하여 만들어졌다 MNIST의 흑백 그림들은 28x28 픽셀의 바운딩 박스와 앤티엘리어싱 처리되어 그레이스케일 레벨이 들어가 있도록 평준화되었다.

MNIST 데이터베이스로만 학습된 모델에 임의로 제작된 데이터를 넣는다면 28x28 픽셀의 바운딩 박스와 앤티엘리어싱, 그레이스케일 레벨이 들어가 있도록 평준화가 되어있지 않기 때문에 좋지못한 성능을 보인다. 따라서 입력되는 데이터에 대한 별도의 처리 과정이 필요하다. 임의로 제작된 않은 데이터들에 대해서 전처리과정을 통한 데이터 가공으로 최대한 MNIST 데이터와 비슷하게 만들어 성능을 증가시키는 방법을 찾고자한다.

2. 수행계획 및 역할 분담

No	내용		팀원	2021年 11月															
				1주차				2주차				3주차				4주차			
계획	계획서 작성		전원																
	역할 분담		전원																
설계 및 개발	Data 수집		전원																
	수집한 data format 분석 및 image제단 과 labeling MNIST data 행렬에 instance추가 방식 및 데이터 분할		전민재																
	수집한 데이터 이진화 및 잡음제거 데이터 중앙 정렬		유승종																
	학습에 사용할 모델 선택 및 학습평가	KNN	문준원																
		Randomforest SGD classifier	전민재																
		softmax classifier	박태성																
		voting	전민재																
	최종 모델 선정 및 모듈화		전민재																
테스트 및 디버깅	경연용 코드 제작 및 개선방안 고찰		전민재																
결과 보고서 작성 ppt제작	결과 보고서 작성 ppt제작		전민재 유승종																

3. 수행과정

가. MNIST dataset 개요 및 분석: 새로운 data instance 를 추가하기 위한 data format 분석 (Get the data/Discover and visualize the data)

```
In [5]: 1 type(X_train[0])
Out[5]: numpy.ndarray

In [6]: 1 type(y_train[0])
Out[6]: numpy.uint8
```

MNIST 데이터베이스의 data 를 X_train 에 저장하고 target 을 y_train 에 넣고 type 을 알아보았다.

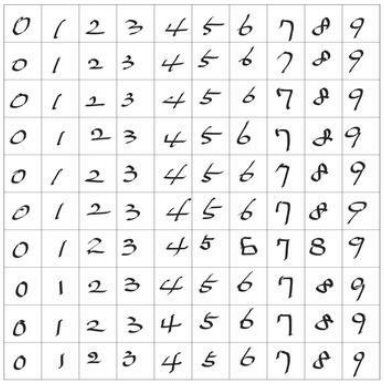
```
In [7]: 1 len(X_train[0])
Out[7]: 784

In [9]: 1 y_train[0]
Out[9]: 7
```

데이터의 길이는 784로 조사했던 28*28픽셀이라는 것을 알 수 있었다.

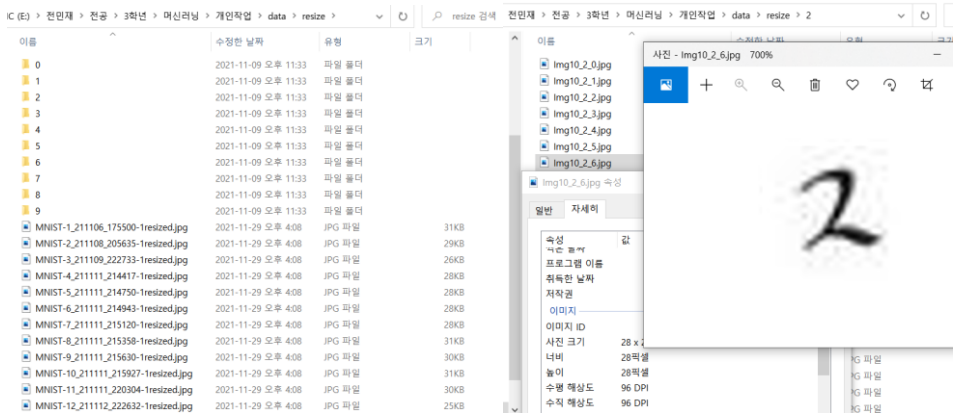
```
1 X_train[0]
array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   1.,  57., 146., 101., 146., 146.,
       153., 254., 254., 254., 254., 254., 254., 255., 254., 180.,  12.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        52., 253., 253., 253., 253., 253., 253., 253., 253., 253., 253.,
       253., 253., 253., 253., 240.,  43.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,  99., 253., 253., 253., 252.,
       248., 248., 248., 248., 239., 139., 181., 251., 253., 253., 130.,
        10.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0., 117., 253., 253., 228.,  83.,   0.,   0.,   0.,   0.,   0.,
         0.,  42., 204., 253., 209.,  47.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,  33., 208., 253., 253.,  89.,
         0.,   0.,   0.,   0.,   0.,   0.,   0., 154., 251., 253., 176.,
         2.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,  33., 235., 253., 219.,  73.,   0.,   0.,   0.,   0.,   0.,
         1.,  83., 228., 253., 149.,  16.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 180., 253., 182.,
        10.,   0.,   0.,   0.,   0.,   0.,  89., 253., 253., 216.,  72.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
```

0~255의 픽셀 강도, Grey scale image을 가지고 있다는 것도 확인할 수 있었다.

나. 새로운 data instance 수집 및 MNIST 에 추가 (구체적인 방법 제시)



배포한 양식에 따라 손 글씨 작성 후 이미지로 변환한다. 이미지는 380 x 380 해상도로 변환 후 이미지를 읽어 들여 이미지당 100 개의 28 x 28 크기의 jpg 로 변환한다. 제작된 이미지는 0~9 까지 10 열이 있으므로 하나씩 자를 때마다 labeling 을 진행할 수 있다. 즉, 첫째 열은 0 파일에 저장하게 만든다. 같은 방식으로 반복해 같은 숫자끼리 모아 이름이 같은 파일에 존재한다면 같은 label 을 가지고 있다는 의미한다.



```
#28*28resize
crop_image = crop_image.resize((28, 28),Image.LANCZOS)
```

28*28 데이터로 인코딩하는 과정에서 파이썬 Pillow(PIL)를 사용해 LANCZOS 기법으로 선명하게 필터링 한다.

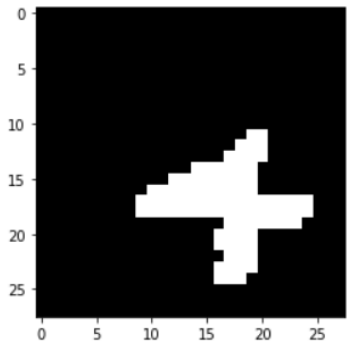
만들어진 데이터는 단순히 28*28 size 로 MNIST 데이터형식과 같아졌지만 글자의 기울어짐, 흐릿함, 쏠림과 같은 현상은 여전히 존재한다. 이러한 문제점을 해결하기위해 추가적인 전처리를 진행한다.

```
# 모폴로지(dilate->팽창연산 적용: 글자 잡음 제거 , erode -> 침식연산 적용 :글자 주변 잡음 제거)
k = cv2.getStructuringElement(cv2.MORPH_RECT, (2,2)) #구조화 요소 커널, 사각형 (3x3) 생성
ero_img = cv2.erode(img, k)

imgray = cv2.cvtColor(ero_img, cv2.COLOR_BGR2GRAY) #이거 안하면 이진화 불가
```

cv2 모듈을 사용해 erode 함수를 부른다. 모폴로지 연산을 사용해 글자의 주변 잡음을 제거하여 숫자를 의미하는 비트만을 남긴다.

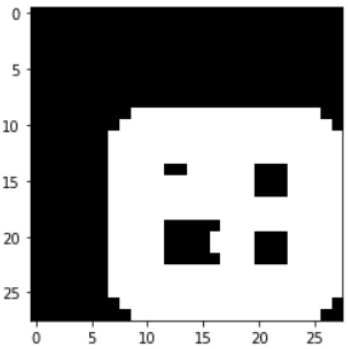
```
ret, img_binary = cv2.threshold(imgray, 127, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)  # 이진화
```

그 후 threshold 값을 127 로하여 이진화 작업한다.



이진화가 끝났으면 중앙 정렬을 위해서 숫자를 감싸는 contour 작업으로 숫자의 중심좌표를 구한다.

```
contours, hierarchy = cv2.findContours(vers_img, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)  # 컨투어
#반전된 이미지로 컨투어를 진행

if len(contours) == 0:
    print("컨투어 0") #컨투어를 하지 못한 -> 이미지 파일 손상 등등의 문제

else:
    contr = contours[0]
    x, y, w, h = cv2.boundingRect(contr)  # 최소한의 사각형 그려주는 함수
    cv2.rectangle(img_binary, (x, y), (x + w, y + h), (0, 255, 0), 3) #(x,y)->왼쪽 하단 꼭지점, (x+w,y+h)-> 오른쪽 상단 꼭지점
    # w: 사각형의 width, h: 사각형의 high  x: 왼쪽 아래 꼭지점의 x좌표  y: 오른쪽 위 꼭지점의 y좌표
    # 사각형을 img_binary에 그려 주었다 -> 사각형 추후 처리해야 한다 아니면 숫자가 바뀜
    print(" 가로 중간값 = %f" % (x_mid))
    print(" 세로 중간값 = %f" % (y_mid))
```



여기서 나온 x_mid 와 y_mid 즉, 숫자의 중심 좌표가 이미지의 중심인 (17,17)에서 얼만큼 떨어져 있는지를 구한 다음 원본을 이동시킨다.

주의점으로 현재의 이미지는 contour 사각형이 그려져 있으므로 이 이미지는 평행이동을 얼만큼 해야하는지 구하는 용도로만 사용한다.

```
final_img = [[255 for col in range(28)] for row in range(28)]
```

위 코드를 통해 새로운 데이터공간을 생성한다.

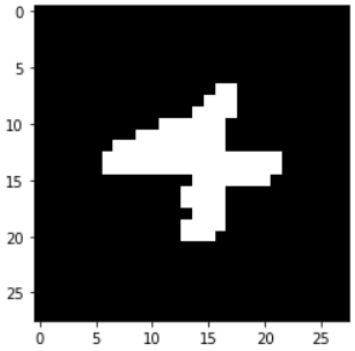
```
x_er = math.floor(x_mid -14)
y_er = math.floor(y_mid -14)
```

중앙으로부터 얼만큼 떨어졌는지 저장한다.

```
if y_mid !=  14 : # 숫자가 좌우 쏠림
    for i in range (28):
        for j in range (28):
            if cpy_img2[j][i] == 0: #데이터가 있으면
                final_img[j-round(y_er)][i] = 0
else: #이미 상하정렬되어 있으면
    for i in range (28):
         for j in range (28):
            final_img[i][j] = cpy_img2[i][j] #deep copy
```

```
import math
x_er = math.floor(x_mid -14)
y_er = math.floor(y_mid -14)
print(x_er)
if x_mid !=  14 : # 숫자가 좌우 쏠림
    for i in range (27):
        for j in range (27):
            if img_final_binary[i][j] == 0: #데이터가 있으면
                cpy_img2[i][j-round(x_er)] = 0
```

이진화된 이미지를 훑어가면서 숫자가 있는 부분(픽셀값이 0 인 부분)이 발견되면 도화지에 옮겨 그리는데 위에 구한 값만큼 평행이동한 지점에 0 의 값을 할당한다.



완료가 되면 이미지를 확인을 하여서 중앙 정렬 및 전처리 작업이 완료되었는지를 확인한다.

```
#한픽셀 반전해서 담기
    cnt=0
    bit=[]
    for i in range(28):
        for j in range(28):
            a= 255-final_img[i][j]
            #비트담기
            if a==0:
                cnt+=1
            bit.append(a)
```

지금까지 이미지를 반전시켜서 작업하였으므로 저장을 할 때 반전시켜 저장한다.

```
    #이미지가 모두 침식됐다면 원본으로 대체
    if cnt>780:
        bit=[]
        for i in range(28):
            for j in range(28):
                a= 255-img[i][j][0]
                #비트담기
                bit.append(a)
    new_X.append(bit)
    new_y.append(t)
```

Contour 와 중앙정렬 처리 후 숫자의 인식 과정에서 주변 테두리 같은 것이 이미지에 남아있어 컨투어의 크기가 전체로 잡혀서 이미지가 모두 침식된 이미지를 발견할 수 있다. 그 경우는 원본을 넣도록 예외처리한다.

```
new_X=np.array(new_X)
new_y=np.array(new_y)
#시작파일로 맞추기위해 상위파일로가기(필요 시)
os.chdir("../")
```

Mnist 가 np-array 이므로 새로 추가한 이미지들을 numpy array 로 바꿔주고 종료한다.

전처리 완료된 데이터 확인

```
#확인
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
for i in [0,2600,5032]:
    some_digit = new_X[i]
    some_digit_image = some_digit.reshape(28, 28)
    plt.imshow(some_digit_image, cmap=mpl.cm.binary)
    plt.axis("off")

    #save_fig("some_digit_plot")
    plt.show()

```

Plot 으로 전처리가 잘되었는지 확인한다.

다. Dataset 분류: 기존 data 와 새로운 data 를 training/validation/test dataset 으로 어떻게 분배할지를 결정. Test dataset 은 2500 개 이상의 data instance 를 포함하고 새로운 data instance 를 10% 이상 포함 (Prepare the data)

기본 MNIST data 70000개에 새로운 data 16400개를 추가할 것이다. (총 instance 86400)
Train/Validation/Test = 6/2/2 으로 정하여 진행한다. New instance 또한 6/2/2 으로 정해 나눈다.

	Train	Validation	Test
MNIST	42000	14000	14000
NEW instance	9840	3280	3280

원본MNIST 데이터와 새로운 데이터 병합

```
from sklearn.model_selection import train_test_split
mnist = fetch_openml('mnist_784', version=1)
mnist.target = mnist.target.astype(np.uint8)
X_train_, X_test, y_train_, y_test = train_test_split(mnist["data"], mnist["target"],test_size=0.2, random_state=42)
X_train,X_valid,y_train,y_valid=train_test_split(X_train_, y_train_,test_size=0.25, random_state=42)
print(len(X_train),len(y_valid),len(y_test))
```

```
42000 14000 14000
```

```
len(new_X)
```

```
16400
```

```
x,x_,y,y_=train_test_split(new_X,new_y,test_size=0.2,random_state=42)
x,xvalid,y,yvalid=train_test_split(x,y,test_size=0.25,random_state=42)
print(len(x),len(xvalid))
X_train=np.concatenate((X_train, x), axis=0)
y_train=np.concatenate((y_train, y), axis=0)
X_valid=np.concatenate((X_valid, xvalid), axis=0)
y_valid=np.concatenate((y_valid, yvalid), axis=0)
X_test=np.concatenate((X_test, x_), axis=0)
y_test=np.concatenate((y_test, y_), axis=0)
```

```
9840 3280
```

```
print(len(X_train),len(y_train),len(X_valid),len(y_valid),len(X_test),len(y_test))
```

```
51840 51840 17280 17280 17280 17280
```

라. 학습에 사용할 모델 선택: 다수의 후보를 대상으로 기본 학습 결과를 이용하여 최종 모델 선택. 근거와 학습 계획 제시 (Select and train a model)

여러 개의 클래스를 직접 처리하는 SGD 분류기, 랜덤포레스트 분류기, KNN 분류기와 softmax 모델을 통해 학습을 진행하였다.

KNN

```
param_grid =[{'n_neighbors' : [3, 5, 10] ,'leaf_size' : [10, 30, 50]}]


knn_clf = KNeighborsClassifier(weights='distance')
grid_search = GridSearchCV(knn_clf, param_grid,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(estimator=KNeighborsClassifier(weights='distance'),
             param_grid=[{'leaf_size': [10, 30, 50],
                          'n_neighbors': [3, 5, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

```
grid_search.best_estimator_
```

```
KNeighborsClassifier(leaf_size=10, n_neighbors=3, weights='distance')
```

grid search 를 통해 적당한 parameter 를 찾고 학습을 진행한다.

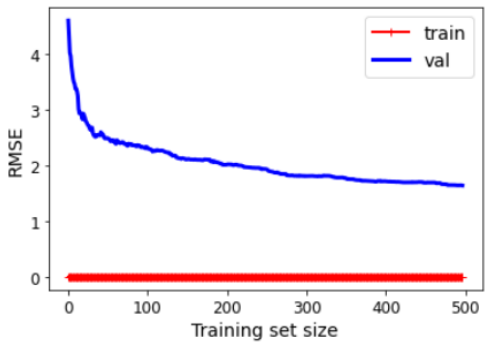
```
knn_clf1 = KNeighborsClassifier(weights='distance', n_neighbors = 3, leaf_size = 10 )
time1=time.time()
knn_clf1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
```

```
학습시간 : 0.049193382263183594
```

```
knn_clf1.score(X_valid,y_valid)
```

```
0.9729285714285715
```

학습시간은 짧았지만 score 하는 단계에서 너무 많은 시간이 소요된다. 따라서 빠른 시간 내에 결과를 출력해야하는 프로젝트에 적합하지 않는다고 판단한다.



Learning curve 또한 시간이 많이 소요되어 500 개의 epoch 로 진행하였다. 결과적으로 train set 에 대해서는 오차가 없지만 valid set 에 대해 오차가 크게 존재함으로 overfitting 됨을 알 수 있다.

SGD

```
from sklearn.model_selection import GridSearchCV
sgd1_clf = SGDClassifier(random_state=42)
param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'max_iter':[500,1000,1500,2000]}
  ]
grid_search = GridSearchCV(sgd1_clf, param_grid,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(estimator=SGDClassifier(random_state=42),
             param_grid=[{'max_iter': [500, 1000, 1500, 2000]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

```
grid_search.best_estimator_
```

```
SGDClassifier(max_iter=500, random_state=42)
```

grid search 를 통해 적당한 parameter 를 찾고 학습을 진행한다.

```
sgd_clf = SGDClassifier(max_iter=300, tol=1e-3, random_state=42)
time1=time.time()
sgd_clf.fit(X_train, y_train)
time2=time.time()
print("학습시간 :", time2-time1)
```

```
학습시간 : 84.74410772323608
```

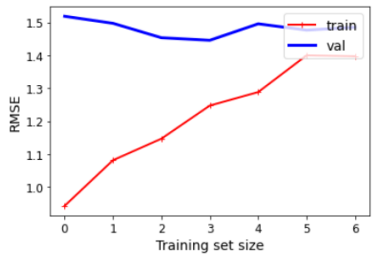
```
sgd_clf.score(X_valid,y_valid)
```

```
0.8640714285714286
```

학습 시간이 오래 걸리고 score 또한 좋지 않아 프로젝트에 적합하지 않다고 판단한다.

```
def plot_learning_curves(model, X_,X_valid,y_,y_valid):
    X_train, X_val, y_train, y_val = X_,X_valid,y_,y_valid
    train_errors, val_errors = [], []
    for m in range(6000, 42001,6000):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))

    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
    plt.legend(loc="upper right", fontsize=14)   # not shown in the book
    plt.xlabel("Training set size", fontsize=14) # not shown
    plt.ylabel("RMSE", fontsize=14)              # not shown

#knn_clf1 = KNeighborsClassifier(weights='distance', n_neighbors = 3, leaf_size = 10 )
plot_learning_curves(sgd_clf,X_train,X_valid,y_train,y_valid)
            # not shown in the book
plt.show()                                       # not shown
```

학습시간이 오래걸리기 때문에 learning curve 를 그릴 때도 시간이 많이 소요된다. 따라서 데이터 6000 개단위로 learning curve 를 그려 시간을 단축하였다.



Train set 에 대한 오차와 valid set 에 대한 오차의 차이가 줄어들지만 오차 자체가 크게 존재해 underfitting 이 발생하는 것을 확인했다.

Softmax

```
from sklearn.model_selection import GridSearchCV
import math
import time

softmax_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs', C = 10, random_state=42)
param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'C':[0.001,0.01,0.1,0,1,10,100,1000]},
    {'max_iter':[100,200,300,400,500,600]}
  ]
grid_search = GridSearchCV(softmax_reg, param_grid,
                           n_jobs = 4,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
```

grid search 를 통해 적당한 parameter 를 찾고 학습을 진행한다.

```
grid_search.best_params_
```

```
{'C': 0.001}
```

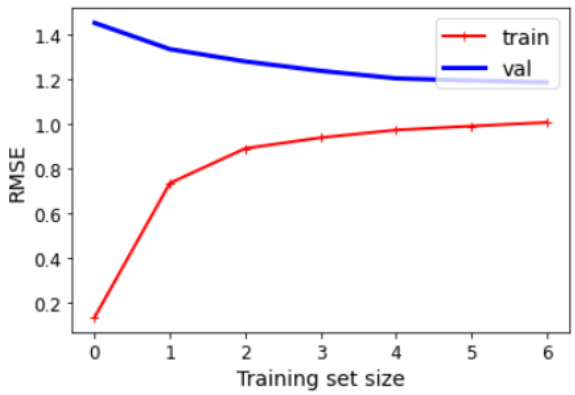
```
softmax_reg_1 = LogisticRegression(multi_class="multinomial", C=0.001, random_state=42)
time1=time.time()
softmax_reg_1.fit(X_train, y_train)
time2=time.time()
print("학습시간 :", time2-time1)
```

```
학습시간 : 17.86153244972229
```

```
softmax_reg_1.score(X_valid, y_valid)
```

```
0.9165714285714286
```

적당한 학습시간과 90 이 넘는 score 을 확인했다.



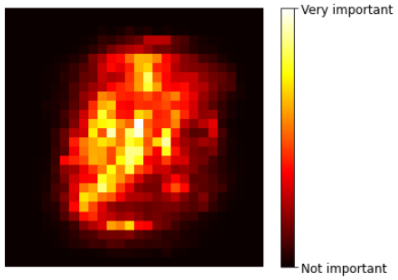
Learning curve 를 통해 학습이 진행될수록 train set 에 대한 오차와 valid set 에 대한 오차가 줄어드는 것을 관측할 수 있었고 오차가 크지 않아 모델 선정에 있어 후보가 될 수 있다고 판단한다.

Randomforest

```
#feature의 중요도
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = mpl.cm.hot,
               interpolation="nearest")
    plt.axis("off")
```

```
plot_digit(rnd_clf.feature_importances_)

cbar = plt.colorbar(ticks=[rnd_clf.feature_importances_.min(), rnd_clf.feature_importances_.max()])
cbar.ax.set_yticklabels(['Not important', 'Very important'])

save_fig("mnist_feature_importance_plot")
plt.show()
```

```
Saving figure mnist_feature_importance_plot
```



Parameter 선정과 학습 전 randomforest 모델을 적용할 때 MNIST 데이터에 대해 각 feature 에 대한 중요도를 구해보았다. 데이터의 feature 중 중앙부를 제외한 테두리 부분이 중요도가 떨어지는 것을 확인할 수 있다. 이 결과를 통해 기존 이미지 전처리 단계에서 추가적인 작업을 수행하고자 한다.

```
#테두리 최식
for x in [0,1,2,31,32,33]:
    for y in range(34):
        try:
            load_image[x,y]=(255,255,255)
        except:
            load_image[x,y]=255
for y in [0,1,2,31,32,33]:
     for x in range(34):
        try:
            load_image[x,y]=(255,255,255)
        except:
            load_image[x,y]=255
```

원본 데이터에서 자른 한 개의 데이터를 28*28 로 resize 하기전 테두리를 제거하는 코드로 이는 importance 따라 진행되는 학습과정에 있어 불필요한 테두리 잡음을 제거하는데 도움이 된다. 또한 contour 과정에서 발생할 수 있는 테두리 검출과 같은 문제를 줄여 성능을 향상시킨다.

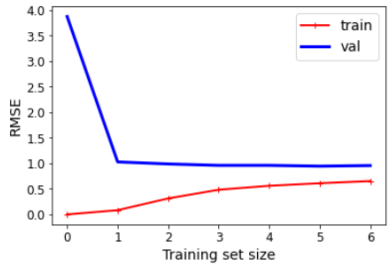
n_estimators=500,max_depth=17 일 때

```
rnd_clf = RandomForestClassifier(max_depth=17, n_estimators=500,random_state=42)
time1=time.time()
rnd_clf.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
rnd_clf.score(X_valid,y_valid)
```

```
학습시간 : 154.16461062431335
```

```
0.9660714285714286
```

n_estimators=500, max_depth=17 일 때 parameter 에서 적당한 학습시간과 score 를 얻을 수 있었다.



Learning curve 는 train set 과 valid set 에 대해 적은 오차를 보이고 있음을 관측할 수 있다.

따라서 프로젝트에 가장 적합한 모델으로 판단한다.

마. 모델 최적화 및 분석 (Fine tune the model)

- Model 과 training hyperparameter 의 최적화를 통해 최대 성능을 획득.

- 최적화과정 제시 및 결과 분석. 학습시간, 예측시간(inference time), 정확도 측면에서 분석. Epoch 에 따른 learning curve 제시.

- 예측시간은 정확도 94% 이상을 만족하는 범위에서 가장 작은 모델 선택.

새로운 데이터들을 MNIST 데이터에 추가하여 결과를 살펴본다.

KNN(socoring 시간 오래걸림)

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf1 = KNeighborsClassifier(weights='distance', n_neighbors = 3, leaf_size = 10 )
time1=time.time()
knn_clf1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
```

학습시간 : 39.2401328086853

SGD

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter=300, tol=1e-3, loss='log',random_state=42)
time1=time.time()
sgd_clf.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
sgd_clf.score(X_valid,y_valid)
```

학습시간 : 216.3063189983368 0.7309606481481481

SoftMax

```
from sklearn.linear_model import LogisticRegression
softmax_reg_1 = LogisticRegression(multi_class="multinomial", C=0.001, random_state=42)
softmax_reg_1.fit(X_train, y_train)
time1=time.time()
softmax_reg_1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
softmax_reg_1.score(X_valid,y_valid)
```

학습시간 : 23.729557037353516 0.8205439814814814

Random Forest

```
rnd_clf = RandomForestClassifier(max_depth=25, n_estimators=100,random_state=42)
time1=time.time()
rnd_clf.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
rnd_clf.score(X_valid,y_valid)
```

학습시간 : 33.976980447769165

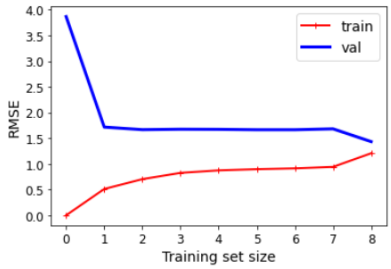
0.9428240740740741

```
rnd_clf.score(X_test,y_test)
```

0.9412615740740741

새로운 데이터들을 추가하니 모델 전체적으로 학습시간이 늘고 score 가 줄어듬을 확인하였다.

그 중 RandomForest 가 유일하게 score 가 94 를 넘는 결과를 보였다.



새로운 데이터가 들어오고 train set 과 valid set 에 대한 오차가 커지긴 했지만 적절히 fitting 된 것을 확인할 수 있다.

Voting

위에서 학습했던 모델들을 Voting classifier 을 사용해 학습시켜 성능 향상을 시도한다.(hard, soft)

```
from sklearn.ensemble import VotingClassifier

voting_clf_1 = VotingClassifier(estimators=[('rnd', rnd_clf),('log', softmax_reg_1),('sgd',sgd_clf)], voting='hard')
time1=time.time()
voting_clf_1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
voting_clf_1.score(X_valid,y_valid)
```

학습시간 : 301.9377176761627

0.8619212962962963

```
voting_clf_1 = VotingClassifier(estimators=[('rnd', rnd_clf1),('log', softmax_reg_1)], voting='soft')
time1=time.time()
voting_clf_1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
voting_clf_1.score(X_valid,y_valid)
```

학습시간 : 313.66132616996765 0.7341435185185186

학습시간이 프로젝트의 목표와 다르게 지나치게 소요되고 score 또한 잘 나오지 않았다.

Score 가 좋지 않았던 sgd 모델을 제거하고 다시 학습해 보았다.

```
voting_clf_1 = VotingClassifier(estimators=[('rnd', rnd_clf),('log', softmax_reg_1)], voting='hard')
time1=time.time()
voting_clf_1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
voting_clf_1.score(X_valid,y_valid)
```

학습시간 : 180.9268114566803

0.8876157407407408

```
voting_clf_1 = VotingClassifier(estimators=[('rnd', rnd_clf),('log', softmax_reg_1)], voting='soft')
time1=time.time()
voting_clf_1.fit(X_train,y_train)
time2=time.time()
print("학습시간 :", time2-time1)
voting_clf_1.score(X_valid,y_valid)
```

학습시간 : 188.89919805526733

0.8891203703703704

학습 결과 시간은 줄어들고 score 가 증가했지만 목표한 score 에 도달하지 못하였다.

4. 결론

랜덤 포레스트는 앙상블 머신러닝 모델이다. 다수의 의사결정 트리를 만들고, 그 나무들의 분류를 집계해서 최종적으로 분류한다. 따라서 결측치 비율이 높아져도 높은 정확도를 나타낸다.

feature 의 중요성을 파악할 수 있어 데이터 전처리를 통한 성능향상 방법을 제시해 줄 수 있다. 이를 통해 적절한 전처리과정을 설계하였고 높은 score 얻을 수 있다.

수행일지

팀원	전민재	유승종	문준원	박태성
역할	프로젝트 총괄	데이터 전처리	모델 학습	모델 학습
모임일자	발표 내용			
1 주차	데이터수집 및 역할분담, 계획서 작성, github 을 이용한 자료공유			
2 주차	수집한 data format 분석 및 image제단 과 labeling 코드 설명 MNIST data 행렬에 instance 추가 방식 및 데이터 분할 코드 설명	Contour 를 사용한 전처리 제안	수집한 data format 분석 및 image제단 과 labeling 코드 제작 실패 knn 학습 진행계획	MNIST data 행렬에 instance 추가 방식 및 데이터 분할 실패 sgd 학습 진행계획
3 주차	Randomforest 모델 학습 및 learning curve	이진화 및 가운데 정렬 제안	knn 모델 학습 및 learning curve	sgd 학습 진행계획
4 주차	sgd 모델 학습 및 learning curve 모든 모듈 결합 후 설명	이진화 및 가운데 정렬 코드 설명	Knn 자료조사 계획 대외 활동으로 추가적인 프로젝트 활동 불가	컴퓨터가 망가져서 프로젝트 진행 및 소통 불가
5 주차	Softmax 모델 학습 및 learning curve Voting 모델 학습 변경된 프로젝트 보고서 및 ppt 제작	보고서 및 ppt 제작	대외 활동으로 추가적인 프로젝트 활동 불가	컴퓨터가 망가져서 프로젝트 진행 및 소통 불가 Softmax, sgd 학습 진행

Reference

위키백과 [랜덤포레스트]

https://ko.wikipedia.org/wiki/%EB%9E%9C%EB%8D%A4_%ED%8F%AC%EB%A0%88%EC%8A%A4%ED%8A%B8