**7조 머신러닝 프로젝트 최종보고서**

**[Hand-made MNIST dataset을 이용한 머신러닝 모델 최적화 및 분석]**

**20236079 김희균, 20201590 송준규, 20192447 정승연, 20201610 정하연**

**//수정 필요 //코드에서 다시계산 필요**

**1. 프로젝트 목표 수립**

// precision or recall?

// 높은 점수 보다 오버피팅을 막는 최대한 내성이 강한 모델을 만들자 (노이즈, 테두리, Shifted)

**~~2. 데이터 구하기~~**

// 작성 및 이어 붙이기

**3. 데이터 분석 (사실에 의한 통찰)**

**(1) 15개(0~9, +, -, /, x, =) 클래스 이외의 데이터**

Handmade dataset를 모두 plot하여 15개 클래스 이외에도 12, 10 등 다른 숫자나 기호로 라벨링된 데이터가 존재함을 알게 되었습니다.

**(2) 잘못된 라벨링**

클래스 별 데이터의 특징을 알아보기 위해 15개의 클래스 순서대로 데이터를 나열했습니다. 0으로 라벨링된 데이터, 1로 라벨링된 데이터, 2로 라벨링된 데이터, … 와 같은 순서대로 나열한 것입니다.



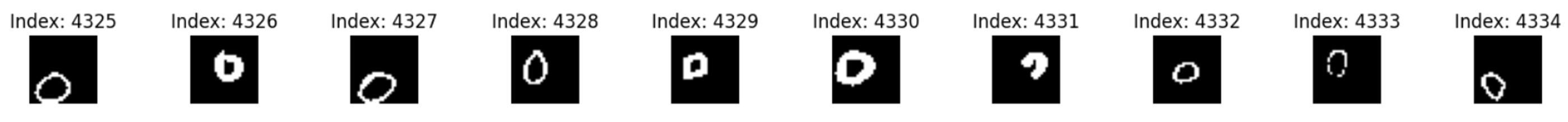


위는 0으로 라벨링된 데이터 10개를 임의로 plot한 결과입니다. 3216, 3220, 3222 데이터는 육안으로 보면 각각 1, 5, 3에 해당하는 데이터인데, 0으로 잘못 라벨링 되었습니다.

**(3) 학습에 애매한 데이터**

****

위는 0으로 라벨링된 또 다른 10개의 데이터를 plot한 결과입니다. 4055 데이터는 0의 위쪽 부분이 잘려 숫자 0의 형태라고 보기 어렵습니다.



위 이미지에서 4331 데이터는, 4330 데이터와 같은 굵기와 크기를 가져서 0의 왼쪽 아래부분이 잘렸다고 볼 수 있지만, 클래스 7로도 분류될 수 있는 이미지입니다.

이와 같이 라벨링은 제대로 되었지만 심하게 잘린 데이터, 혹은 다른 클래스와 유사하여 혼동을 유발할 수 있는 데이터가 존재함을 알게 되었습니다.

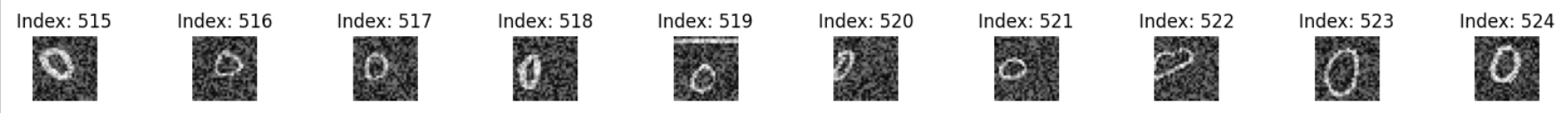
**(4) Handmade dataset 데이터 수**

약 60000개의 Original MNIST dataset과 달리, Handmade MNIST dataset은 약 30000개만 존재함을 알게되었습니다. 추후 handmade dataset 전처리 과정을 거친 후, original MNIST dataset를 일부 추가하여 데이터 개수에 보완이 필요함을 알게 되었습니다.

**(5) 픽셀값(feature 값)의 차이**

Original MNIST dataset의 픽셀값은 0에서 255 사이의 값이고, Handmade MNIST dataset의 픽셀값은 0과 1 사이의 값이었습니다. Handmade 와 Original MNIST dataset을 통합한 dataset을 학습하려면, 픽셀값을 통일하는 (스케일링) 과정이 필요함을 알게 되었습니다.

**(6) 노이즈가 있는 데이터, shift된 데이터, 테두리가 남은 데이터**

****

현재 가지고 있는 테스트 데이터셋 이외의 어떠한 데이터가 들어와도 강력한 모델을 만들기 위해서는, 노이즈가 있는 학습 데이터셋을 구축하는 작업이 필요함을 인지하게 되었습니다.

이를 위해서는 노이즈로 분류되는 threshold를 정의해야하고, 모델이나 함수 중 어떤 방식으로 노이즈를 처리할 지 결정하는 과정이 필요했습니다.

또한, Index 512과 같이 handmade dataset을 작성할 때 썼던 템플릿의 테두리가 포함된 데이터가 존재함을 알게 되었고, 학습에 방해가 될 수 있다고 판단했습니다.

index 520, 522와 같이 숫자가 중앙에서 벗어나 상하좌우로 치우친 데이터도 존재했습니다. 데이터의 일관성이 떨어지므로 중앙화를 하거나 상하좌우로 치우친 데이터를 추가로 제작하는 과정이 필요하겠다는 생각을 하게 되었습니다.

그리고 최종적으로 test dataset에도 관련 데이터를 구축하여 검증할 필요가 있다고 판단하였습니다.

**(7) 비슷한 숫자 및 기호의 존재**



Index 3888과 같이 0이지만 자칫 6으로 판별될 수도 있는 애매한 데이터가 존재하기에, confusion matrix 분석을 통해 가장 많이 혼동되는 라벨을 파악할 필요가 있다고 생각했습니다.

Index 3866과 같이 기울어진 이미지도 변수로 두어 기울어진 정도를 달리한 데이터셋을 구축하는 아이디어도 생각해보았습니다.

(**8) 데이터 크기 (28\*28)**

현재 Original, Handmade MNIST dataset 모두 28\*28 크기인데, 다른 크기를 입력으로 받아도 처리가능한 모델을 만들 수 있겠다는 아이디어 또한 도출되었습니다.

**4. 학습 데이터 전처리, 파이프라인 구축**

앞서 언급한 데이터의 문제점을 해결하기 위해 다양한 방법의 전처리를 진행했습니다. 30448개의 학습 데이터 중 4199개 (13.8%)가 삭제되었고, 테스트 데이터 4350개 중 620개 (14.3%)가 삭제되었습니다.

Handmade MNIST dataset에서 test dataset의 비율은 20%로 설정하였습니다.

**(1) 15개(0~9, +, -, /, x, =) 클래스 이외의 데이터 제거**

15개 클래스를 잘 분류하는 모델을 만드는 것이 프로젝트의 목적이기 때문에, Handmade MNIST dataset에서 다른 숫자나 기호로 라벨링된 데이터는 모두 삭제하였습니다.

valid\_labels = set(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '-', '/', 'x', '='])

# 숫자 및 연산 기호가 아닌 데이터의 인덱스 찾기

invalid\_trval\_indices = [i for i, label in enumerate(merged\_trval\_labels) if label not in valid\_labels]

# 숫자 및 연산 기호가 아닌 데이터 제거

d\_X\_train = np.delete(merged\_trval\_images\_flat, invalid\_trval\_indices, axis=0)

d\_y\_train = np.delete(merged\_trval\_labels, invalid\_trval\_indices)

**(2) 잘못된 라벨링**

Handmade MNIST dataset을 각 클래스마다 4부분으로 나누어 팀원 한 명이 한 부분씩 맡아 잘못된 라벨을 육안으로 찾아내는 작업을 진행했습니다. 각자 육안으로 판별하는 기준이 다르기 때문에 한 사람이 한 클래스를 모두 담당하지 않고 클래스 별로 모든 사람이 동일하게 작업할 수 있도록 데이터를 나누었습니다. 학습 데이터, 테스트 데이터 모두 동일한 방식으로 진행하였습니다.

아래 index 2998과 같이 다른 숫자인데 0으로 잘못 라벨링된 데이터를 찾아 indices\_wrong 리스트에 저장하여 나중에 모두 제거하였습니다.

****

indices\_wrong = [ 2998, 3018, 3053, 3057, ... ]

**(3) 학습에 애매한 데이터**

학습에 애매한 데이터 또한 육안으로 판별하여 리스트에 저장하였고, 4명이 각각 찾은 인덱스를 병합하여 학습 데이터에서 모두 제거하였습니다.

아래 index 2995와 같이 0인데 6으로 판별될 수도 있어 애매한 데이터들, 또는 너무 작아 점처럼 보이는0을 삭제 대상으로 선정하였습니다.

****

indices\_vague = [ 2995, 3040, 3181, 3440, ... ]

**(4) Handmade dataset 데이터 수**

(1)~(3) 과정을 거쳐 약 2000개의 데이터가 제거되었고, 총 nnnn개의 데이터가 남았습니다. 클래스 별로 개수를 출력해본 결과 nnn(이미지 첨부)개가 남아있었다. 총 ***60,000개(Original MNIST dataset 70,000ㄱㅐ 아닌가)***의 데이터를 구축하기 위해, 각 라벨 별로 train\_set은 4200개, test\_set에는 700개의 데이터가 존재하도록 Original Dataset을 추가하였습니다.

X\_train shape: (26249, 784)

y\_train shape: (26249,)

X\_test shape: (3730, 784)

y\_test shape: (3730,)

지워진 데이터 개수: 1738

perfect\_X\_train shape: (24520, 784)

perfect\_y\_train shape: (24520,) # -1729

perfect\_X\_test shape: (3535, 784)

perfect\_y\_test shape: (3535,) # -195

Train Set Label Counts:

Label 7: 1359 instances

Label 9: 1313 instances

Label 0: 1386 instances

Label 5: 1316 instances

Label 3: 1372 instances

Label 2: 1326 instances

Label 8: 1377 instances

Label 6: 1438 instances

Label +: 2978 instances

Label 4: 1377 instances

Label [/](https://file+.vscode-resource.vscode-cdn.net/): 2823 instances

Label x: 2243 instances

Label 1: 1292 instances

Label -: 2920 instances

Test Set Label Counts:

Label 0: 188 instances

Label [/](https://file+.vscode-resource.vscode-cdn.net/): 420 instances

Label -: 430 instances

Label +: 400 instances

Label x: 311 instances

Label 4: 217 instances

Label 7: 206 instances

Label 5: 183 instances

Label 3: 187 instances

Label 8: 172 instances

Label 6: 189 instances

Label 1: 200 instances

Label 2: 229 instances

Label 9: 203 instances

**(5) 픽셀값(feature 값)의 차이**

**A screenshot of a computer

Description automatically generated**

여러가지 정규화 방법을KNN 파이프라인에 추가하여 실험해본 결과, Normalizer가 가장 성능이 높아 Normalizer로 진행하게 되었습니다.

Min-Max Scaling (최대-최소 스케일링)으로 픽셀값의 최소를 0, 최대를 1로 설정하여 Original MNIST dataset의 값을 그 비율에 맞게 조정하는 정규화(normalization)를 진행했습니다.

// chatGPT한테 물어보니까 Normalizer(norm=’max’)로 하면 픽셀값을 0과 1사이의 값으로 정규화하는 ‘최대-최소 스케일링’ 할 수 있음

from sklearn.preprocessing import Normalizer

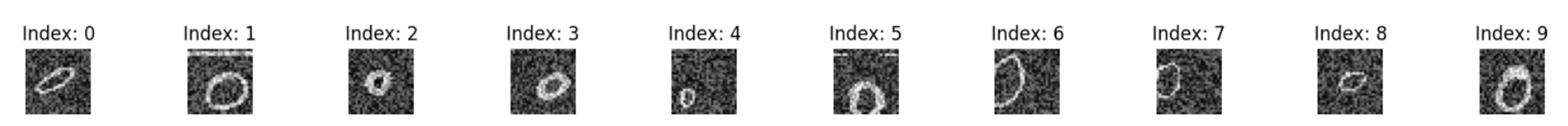
knn\_pipeline = Pipeline([

('normalizer', Normalizer()), # 데이터 정규화

('knn', KNeighborsClassifier()) # KNN 모델

])

**(6) 노이즈가 있는 데이터, shift된 데이터, 테두리가 남은 데이터**



모델 파이프라인에 노이즈를 제거하는 과정을 포함하기 위해 함수를 정의했습니다. 0과 0.5사이의 랜덤 값을 노이즈로 정의하고, 임계값을 0.4로 설정하여 0.4~1.0 이상의 값만 의미있는 값으로 판별되도록 denoise\_with\_max를 정의했습니다. 초기에는 이론적으로 가장 strict한 0.66으로 설정했지만, 다양한 임계값을 시도해본 결과 0.4일때 가장 노이즈를 효과적으로 제거함을 확인하였습니다.

// 노이즈의 정의: 표기 MAX값*1.5 -> 모든 픽셀 MAX*1.5로 나눠서 0.66이하는 노이즈로 판단 but test 과정중 0.5가  노이즈 없애는 선에서 결과 가장 좋음

// 노이즈는 위의 공식을 함수로 해서 처리 // 모델로 하려 했으나 표기값을 1.0으로 맞추지 못하고 진행해 번거로움 존재+데이터 너무 많이 필요

전체 데이터셋의 20%(Original MNIST dataset의 20%, Handmade MNIST dataset의 20%)에 노이즈를 추가하였습니다. 최종적으로 구축된 데이터셋의 비율은 다음과 같습니다. 학습 데이터와 테스트 테이터의 비율이 동일합니다. 여기서 left bottom, left top, right bottom, right top 데이터를 따로 분리하여 구축한 이유는, 중앙화에 대한 모델이 아닌 함수를 구축했기 때문입니다. 모델 학습에 영향을 끼치지 않고, shifted 데이터가 존재해도 함수로 잘 처리된다는 것을 정확히 보여주고 싶어 위치 별 데이터 수를 정하게 되었습니다.

A close up of text

Description automatically generated

Figure 이걸 고민한 내용을 넣어야하는데...

**A diagram of a diagram

Description automatically generated with medium confidence**

Figure 이거 ppt로 다시 만들어서 깔끔하게 첨부하기

def denoise\_with\_max(X):

# Assuming X is a 3000x784 MNIST-like dataset

threshold = 0.4 # Threshold setting #0.66

# Create an empty array to store processed images

X\_processed = np.zeros\_like(X)

# Process each image in the dataset

for i in range(X.shape[0]):

max\_val = X[i].max() # Calculate the maximum value within each image

# Values below the threshold are set to 0, values above are set to 1

X\_processed[i] = (X[i] > threshold \* max\_val).astype(int)

return X\_processed

shift된 데이터를 다루기 위해 상하좌우로 치우쳐진 데이터를 일정 비율 생성하고 이에 적응한 ‘모델’로 학습시키려고 했으나, 교수님께서 상하좌우로 치우져진 데이터는 큰 의미가 없다는 조언을 해 주셨고, 360도로 100개 정도 다른 위치에 존재하는 데이터를 구축하지 않는 이상 파이프라인에 중앙화 과정을 넣는 것이 효과적임을 알게 되었습니다.

저희는 중앙화 하는 함수를 구현하고, 추가로 숫자와 기호를 최대화하는 함수도 구현하였습니다. 중앙화, 최대화 과정을 거친 이미지 예시는 아래와 같습니다.

A black and white image of a logo

Description automatically generated

def center\_image(image):

img\_reshaped = image.reshape(28, 28)

non\_empty\_rows = np.where(img\_reshaped.any(axis=1))[0]

non\_empty\_cols = np.where(img\_reshaped.any(axis=0))[0]

# 예외 처리: 이미지에 아무것도 없는 경우 pass

if len(non\_empty\_rows) == 0 or len(non\_empty\_cols) == 0:

return img\_reshaped.reshape(-1)

top, bottom = non\_empty\_rows[0], non\_empty\_rows[-1]

left, right = non\_empty\_cols[0], non\_empty\_cols[-1]

center\_y, center\_x = (top + bottom) // 2, (left + right) // 2

shift\_y, shift\_x = 14 - center\_y, 14 - center\_x

shifted\_img = np.roll(img\_reshaped, shift\_x, axis=1)

shifted\_img = np.roll(shifted\_img, shift\_y, axis=0)

return shifted\_img.reshape(-1)

def enlarge\_image\_to\_square(image):

img\_reshaped = image.reshape(28, 28)

non\_empty\_rows = np.where(img\_reshaped.any(axis=1))[0]

non\_empty\_cols = np.where(img\_reshaped.any(axis=0))[0]

# 이미지에 아무것도 없는 경우 28x28의 빈 이미지로 확장

if len(non\_empty\_rows) == 0 or len(non\_empty\_cols) == 0:

enlarged\_image = np.zeros((28, 28), dtype=np.uint8)

return enlarged\_image.flatten()

# 가장 작은 정사각형의 좌표 찾기

top, bottom = non\_empty\_rows[0], non\_empty\_rows[-1]

left, right = non\_empty\_cols[0], non\_empty\_cols[-1]

width = right - left + 1

height = bottom - top + 1

# 가장 작은 정사각형 생성

max\_side = max(width, height)

square\_image = np.zeros((max\_side, max\_side), dtype=np.uint8)

start\_row = (max\_side - height) // 2

start\_col = (max\_side - width) // 2

square\_image[start\_row:start\_row + height, start\_col:start\_col + width] = img\_reshaped[top:bottom + 1, left:right + 1]

# 정사각형 이미지를 28x28로 resize

enlarged\_image = Image.fromarray(square\_image.astype('uint8'))

enlarged\_image = enlarged\_image.resize((28, 28), Image.BICUBIC)

# 확대된 이미지를 배열로 변환

enlarged\_data = np.array(enlarged\_image).flatten()

return enlarged\_data

하지만 아래 6번째 이미지와 같이 노이즈가 있을 경우, 7번째 이미지와 같이 테두리가 있는 경우 중앙화가 되지 않는 한계가 있었습니다.

A black and white logo

Description automatically generated

// 테두리 및 노이즈는 라벨을 보고 기호를 인식할 수 있냐 없냐로 판단: 아래쪽 테두리랑 '-'  있으면 =처럼 보임,

테두리 또한 변수에서 제외

**(7) 비슷한 숫자 및 기호의 존재**

// Confusion Matrix는 학습 과정 후 분석

기울어진 데이터에 대해서는 수정하지 않았습니다. 1과 /를 구별하는 가장 중요한 요소가 기울기라고 판단했기 때문입니다.

**(8) 데이터 크기(28\*28)**

어떤 크기의 이미지가 들어오든 28\*28로 변환하는 모델은 논의한 결과 픽셀이 다른 데이터를 새로 구축하지 않는 이상 구현하기 어렵다고 판단하였습니다. 현재 가진 데이터를 확대에서 픽셀을 달리하는 방법을 생각해보았지만, 이 과정에서 픽셀이 깨지기 때문에 새로운 데이터를 구축하는 방안이 더 효과적일 것이라 생각했습니다.

// 위에서 해결 안된 데이터 학습시켜 빼려 했으나 구현상 어려움 및 해당 모델 정확도 너무 낮음

// 0~1 표기, 255 표기는 다른 데이터는 0~1000일수 있으니

일반화 하기보다 냅둠 -> denoise할때만 쓰는데 최대값 기준으로 작성해서 무관

**5. 모델 선택하고 훈련 시키기**

Knn, svm, extra trees, softmax, decision tree, random forest, mlp 모델을 활용하기로 결정했습니다.

**(1) 15개의 이외의 라벨만 제외된 Handmade MNIST dataset**

15개 이외의 라벨만 제외한 데이터로 모든 모델을 Gridsearch로 학습시킨 결과는 다음과 같았습니다.

A screenshot of a graph

Description automatically generated

A graph with purple lines

Description automatically generated with medium confidenceA graph with blue bars

Description automatically generated

SVM은 학습 시간이 매우 오래 걸려 VotingClassifier에 추가하지 않았고, 6개의 모델을 합친 VotingClassifier(all)은 soft일 때 0.7190, hard일 때 0.7357의 정확도를 보였습니다. Softmax와 Decision Tree는 정확도가 0.5보다 낮아서 해당 모델을 제외하고 VotingClassifier를 학습시켜본 결과, soft일 때는 0.7499, hard일 때는 0.7458로 조금 더 높은 성능을 보였습니다.

여기서 정확도를 측정하기 위해 사용한 테스트 데이터셋도 Handmade MNIST dataset에서 15개 이외의 라벨만 제외된 데이터로 구성되었습니다.

**(2) 잘못 라벨링된 데이터와 학습에 애매한 데이터 제거, 학습 데이터를 중앙화 및 최대화**

그 다음으로, 잘못 라벨링된 데이터와 애매한 데이터를 제거했습니다. 이 과정에는 아직 데이터 자체를 중앙화하고 최대화하여 정제된 데이터셋을 구축하는 것이 성능을 높일 수 있는 방법이라고 생각하여 학습 데이터 자체를 중앙화 및 최대화 하였습니다.

아직 Original MNIST dataset을 추가하지 않아 클래스 별 데이터 개수가 상이하고, 총 24520개의 학습 데이터와 3535개의 테스트 데이터가 존재하는 상황입니다. 학습 파이프라인에 Normalizer()를 추가하였습니다.

A screenshot of a table

Description automatically generated

여기서 사용된 Test dataset도 train dataset과 같은 전처리 작업이 수행되었습니다.

정확도가 너무 낮은 모델은 향후 Voting Classifier에 추가하지 않으려 했으나, 전반적으로 좋은 성능을 보였기에 최종 데이터를 구축한 후 다시 어떤 모델들을 사용할 지 설정하기로 결정했습니다.

하지만 앞서 언급했듯 이 방법은 큰 단점이 있었습니다. 중앙화가 되어있지 않은 데이터로 테스트할 경우 overfitting된 결과를 마주하게 됩니다. 프로젝트의 목적은 어떠한 데이터가 들어와도 좋은 성능을 보이는 모델을 만드는 것이라는 점이기 때문에, 이 과정을 함수로 구현하여 테스트 과정이 진행될 때마다 수행될 수 있도록 변경하였습니다.

**(3) 최종 데이터 구축**

4-(6)에서 언급된 구조로 학습 데이터를 구축했습니다. 우선 각 클래스 별로 몇 개의 데이터가 있는지 count한 후, Original Dataset을 추가하여 모든 클래스가 동일한 데이터 개수를 가지도록 보완하였습니다. 데이터 개수가 2배 정도 차이나는 경우도 있었기에 이 과정은 필수적이었습니다.

그리고 전체 데이터의 20%를 shifted된 데이터로 만들었습니다. 파이프라인에 중앙화 및 최대화 과정을 (전처리 과정에서 진행하는 것이 아닌) 함수로 구현하여 파이프라인에 추가하였기에, 이 기능이 잘 수행되는지 검증을 위해 모든 클래스에 같은 비율로 shift를 진행하였습니다.

그리고 전체 데이터의25%에 노이즈를 추가하였습니다. 또한 파이프라인에서 노이즈를 제거하는 과정을 앞서 서술한 함수 형태로 추가하였습니다.

여기서 사용한 테스트 데이터도 학습 데이터와 동일한 비율로 노이즈가 추가되었습니다.

**A white rectangular object with red lines

Description automatically generated**

Figure 학습 후 채워넣어야

거의 모든 모델이 예상보다 좋은 결과를 보여, 7개의 모델을 모두 VotingClassifier에 넣어 학습시켰고, 결과는 (soft/hard)일 때 정확도가 nn로 가장 좋았습니다.

**6. 모델을 상세히 조절하기**

최종 데이터를 가지고 GridSearch를 진행했습니다. Validation의 Fold 수를 5로 설정하였습니다.

앞서 언급했지만, Handmade MNIST Dataset 중 20%로 설정한 test dataset은 학습 데이터와 동일하게 다양한 상황에 대한 데이터가 포함되어 있습니다. Overfitting이나 Underfitting을 방지하기 위해 해당 작업을 진행했습니다.

**7. 피드백**

모델을 학습 시켜서 전처리 하는 과정이 흡족 스럽지는 않았음(노이즈, 테두리, 위치);

위치의 경우 데이터가 이미지당 100개씩은 필요하다는 피드백 -> 일반화 테두리도 그럴 것이다

게시판: Case A, B 문제: B는 상상이상 데이터 많이 필요, A는 연산 오버헤드 생각보다 안큼 ; A win!

**9. 더 추가해야 할 내용**

- 진행된 일정

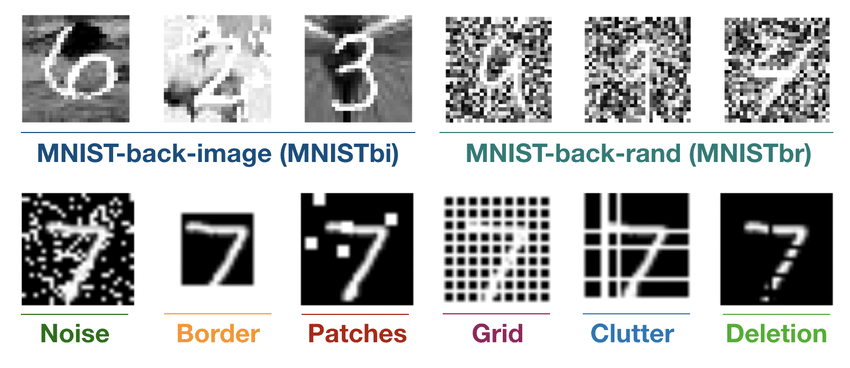
- 역할 분담

- 중간보고서 모음

- 누가 어떤 아이디어 냈는가

- 기여도

- ipynb 최종 파일 만들기

와 이런 노이즈도 있네 허허

그냥 게시판 질문 억지로 찾은건데 오히려 좋아~

**A close-up of a paper

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**