Optimization and Analysis of ML Model Using MNIST Dataset

3조
백현우 이요원 강성욱 박서영

목차

설계 목표 및 프로젝트 설명

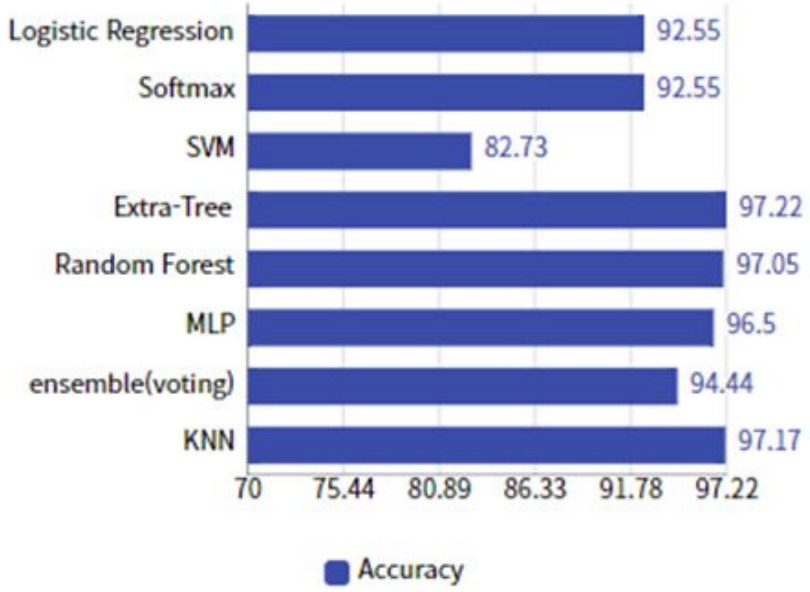
1) 0~9까지의 숫자와 +, -, x, /, = 기호의 **handmade dataset을 제작** 후 Hand-made MNIST dataset과 original MNIST dataset을 이용해 **inference 결과 비교** 및 분석
2) Dataset의 특성의 차이를 다양한 attribute 계산을 통해 분석 후 **Handmade dataset의 성능 저하 원인** 정리
3) Original MNIST dataset과 machine learning 알고리즘을 이용하여 **성능이 좋은 모델 후보군을 찾고** 인식 성능을 개선시킨 machine learning model에 학습
4) 도출해낸 최종 모델을 선택, 별도의 hand-made MNIST test dataset을 이용하여 **평가**

수행 일지 및 역할 분담

모임일자/팀원	백현우	강성욱	이요원	박서영
10월 24일	-수행계획서 작성 및 프로젝트 전반적 이해			
11월 08일		-모델 별 accuracy 비교 및 후보군 도출		
				-MNIST pixel 이진화 코드
11월 15일	-Handmade validation set 제작	-Gaussian noise 추가 코드	-Pixel 중앙 정렬 코드 -Handmade MNIST, original MNIST 성능 비교 및 분석	-Grid search 코드 구현
11월 17일	-중간 발표 준비			
11월 26일	-operation dataset cleaning 작업			
	-Dataset 통합		-Digit dataset cleaning 작업	
11월 28일	-Pipelining 구현		-Preprocessing accuracy 검증 -Label cleaning 코드	-Cleaning accuracy 검증 -Pixel 중앙 정렬 코드 develop
	-전체 코드 통합	-Label cleaning 코드 develop	-Operation dataset accuracy 검증 -Operation dataset rotating	-Label cleaning 코드 develop
11월 30일	-error analysis - operation rotating develop		-최종 training, test dataset 제작	-Learning curve 코드
		- Learning curve 코드 develop		
	-최적 hyper parameter 도출 및 최종 학습			
12월 2일	-최종 발표 준비			

002 데이터 분석

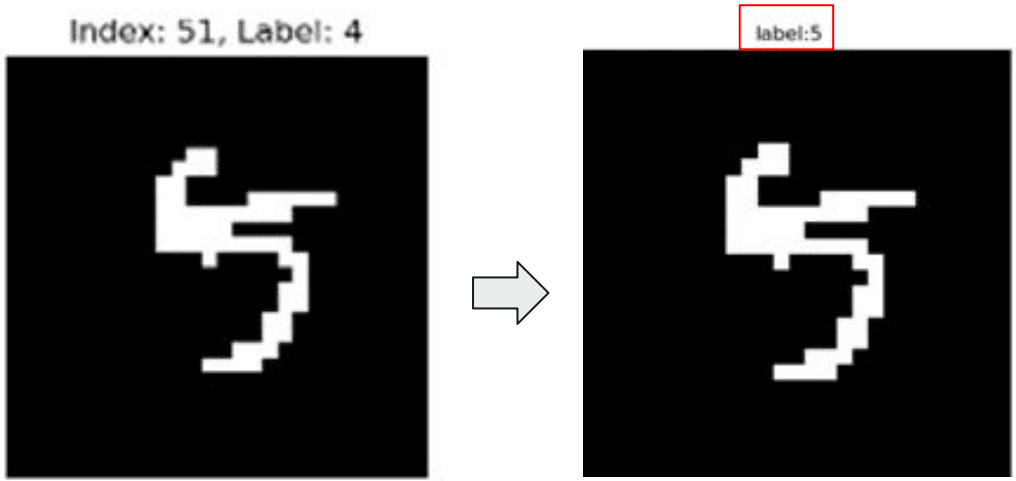
최적의 모델 후보군 도출



Extra tree
KNN
ensemble

Data Cleaning

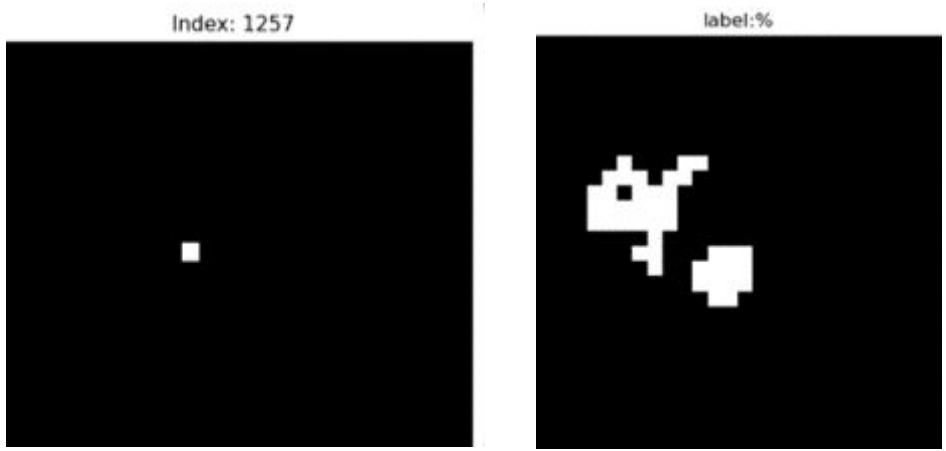
- 오류 레이블 유형



① Data는 정확하나 다른 레이블이 설정된 경우 -> **올바른 레이블로 변경**

Data Cleaning

- 오류 레이블 유형



② 식별 불가능 하거나 레이블에 없는 기호를 작성한 경우 -> **삭제**

Data Cleaning

- **오류 레이블 유형**

③ 숫자 데이터셋에 포함된 기호 레이블/데이터

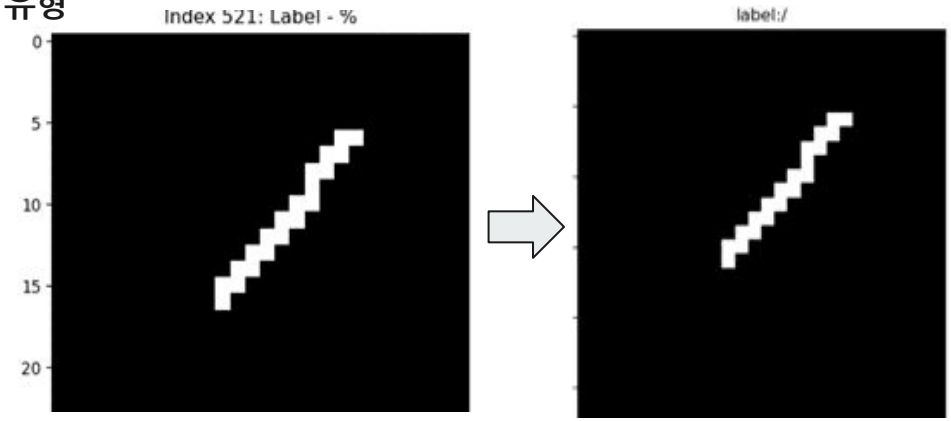
```
correct_op_label = ['+','-','x','/','=']

op_mask=[]
for wrong_label in op_TrVal_label:
    if wrong_label not in correct_op_label:
        op_mask.append(False)
    else:
        op_mask.append(True)
```

→ 조건에 맞는 데이터만 골라내는 **마스크** 활용

(5가지 기호 중 하나인 경우에만 True값 할당,
False값 제외하고 배열 재생성)

Data Cleaning

- 오류 레이블 유형



④ %, * 와 X 는 각각 /와 x로 통합

```
Extra Trees Accuracy: 0.25692857142857145
Extra Trees Training Time: 53.9805223941803 seconds
```

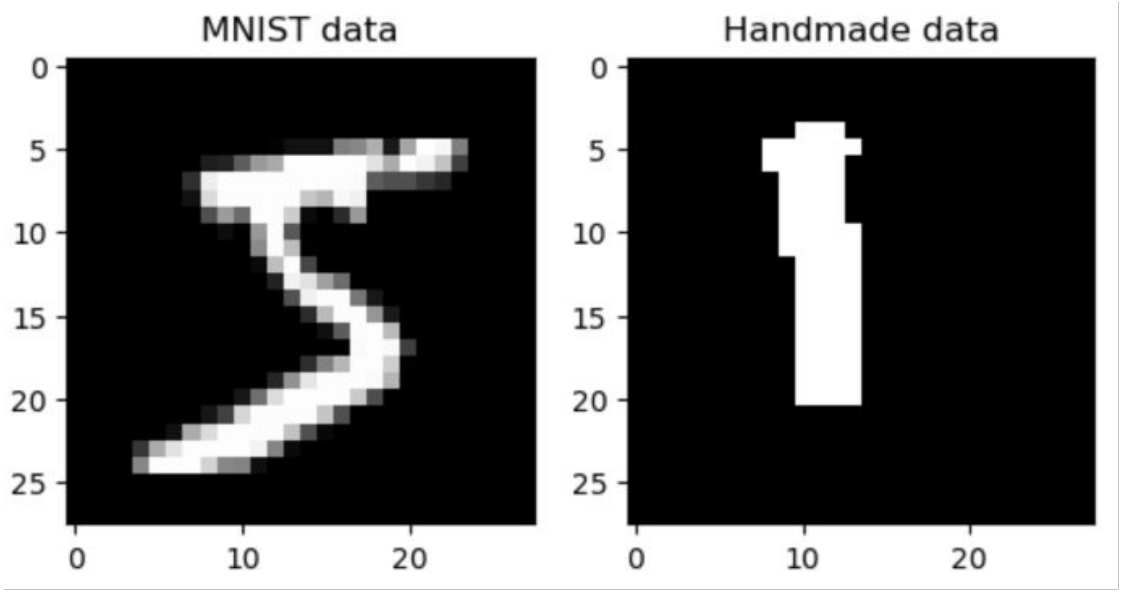
〈 데이터 클리닝 전 〉

```
Extra Trees Accuracy: 0.38442857142857145
Extra Trees Training Time: 53.72251558303833 seconds
```

〈데이터 클리닝 후〉

Hand-MNIST vs Original-MNIST

- data 시각적 분석



- Pixel 값의 차이
- MNIST는 Handmade Dataset 보다 상대적으로 중앙에 배치됨

Hand-MNIST vs Original-MNIST

- data 값 분석

```
MNIST : [  0.   0.   0.   0.   0.   0.   0.   0.   0.  14.   1. 154. 253.  90.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
```

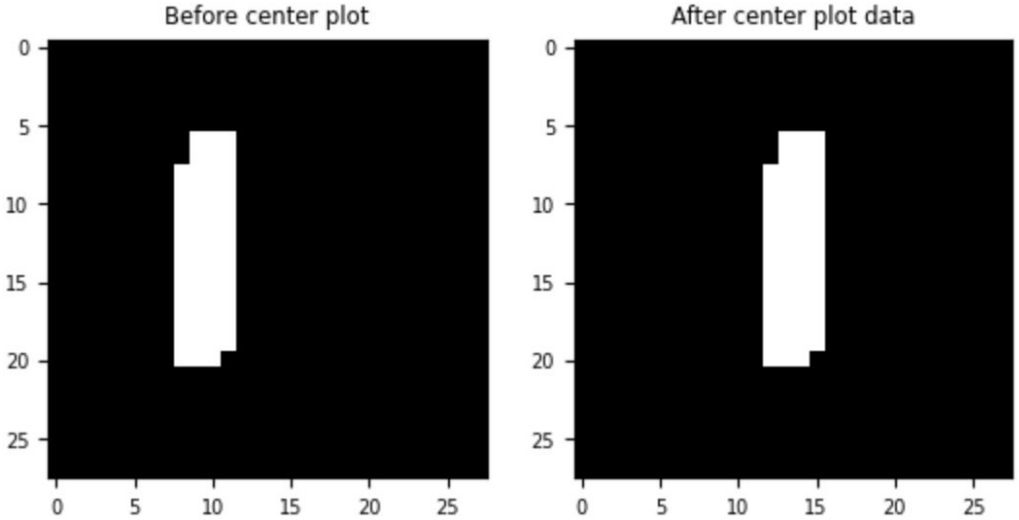
MNIST data 값 => 0 - 255 사이 정수 값

```
HM : [0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
```

Handmade data => 0 또는 1의 이진화된 값

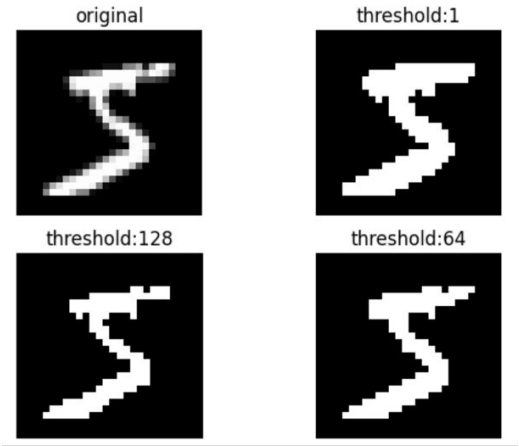
Preprocessing

- Img center plot



Preprocessing

- **MNIST binarize == MNIST data -> Handmade data**

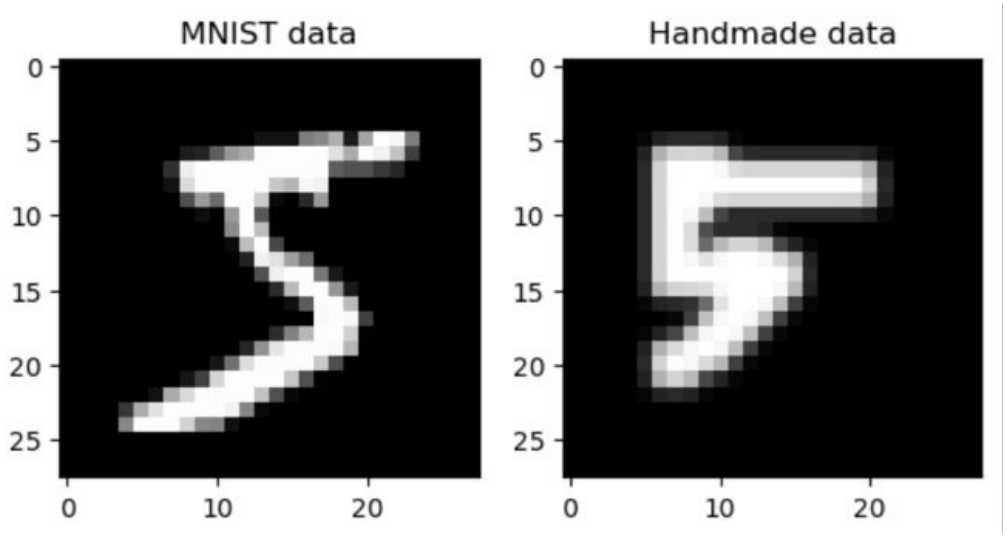


Threshold 값을 1, 64, 128로 설정하여 이미지 비교

원본 이미지와 분포가 가장 비슷한 임계값으로 결정

Preprocessing

- **Add Gausian noise == Handmade data -> MNIST data**



Preprocessing

- Verify preprocessing

Case. 1 Center plot + MNIST Binarize

Accuracy

```
Extra Trees Accuracy: 0.53
Extra Trees Training Time: 29.703039169311523 seconds
```

Case. 2 Center plot + HM Gaussian noise

Accuracy

```
Extra Trees Accuracy: 0.51
Extra Trees Training Time: 31.199922800064087 seconds
```

Data 전처리는 MNIST를 이진화하고
Handmade data를 center plot 하기로 결정

Pipeline

- scaler
  - CenterScaler()

```
def transform(self, X, y=None):
    if X.ndim == 2:
        X = X.reshape(-1, 28, 28)

    CenteredImage = []
    for image in X:
        if image.max() > 1:
            CenteredImage.append(image)
        else:
            CenteredImage.append(self.center_image(image))
    return np.array(CenteredImage).reshape(-1, 784)
```

  - PixelScaler()

```
def transform(self,X,y=None):
    if X.ndim == 3:
        X = X.reshape(-1, 784)
    PixelImage = []
    for image in X:
        if image.max() > 1:
            PixelImage.append((image>64).astype(int))
        else:
            PixelImage.append(image.astype(int))

    return np.array(PixelImage)
```

Pipeline

- PixelScaler 검증

```python
# PixelScaler transform
p_scr = PixelScaler()
X_pixel = p_scr.transform(X)
```

non binary MNIST data → binary MNIST data

```
13  print(X[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   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   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   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   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   0   0   0   0   0
    0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
```

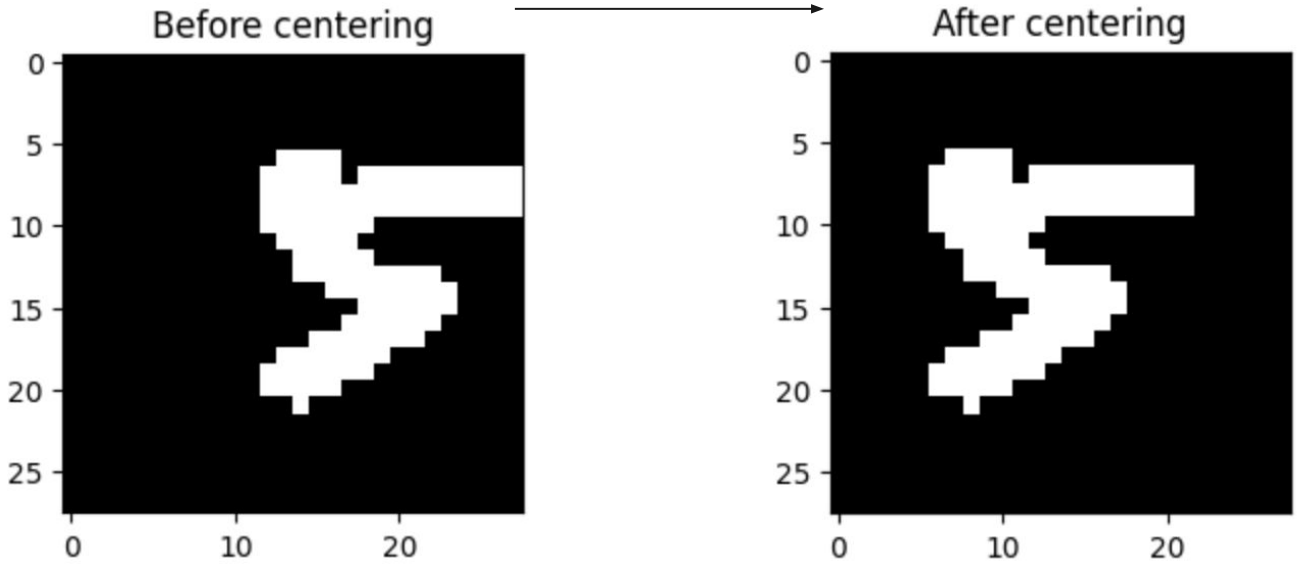
```
10  print(X_binary_1[0])
```

```
(70000, 784)
[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 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 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 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 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 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 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 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 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 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 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 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 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 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 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 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 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 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
 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 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 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 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 1 1 1 1 1 1 1
```

Pipeline

- **CenterScaler 검증**

```
# CenterScaler transform
c_scr = CenterScaler()
X_scaled = c_scr.transform(hm_digit_test_data)
```



003 Development 과정 및 성능 검증

MNIST학습 → handmade test
(ExtraTreeClassifier)

```
Extra Trees Accuracy: 0.1
Extra Trees Training Time: 35.386985063552856 seconds
```

```
et_clf = ExtraTreesClassifier(n_estimators=100, random_state=42)
```

MNIST학습 → handmade test
(Pipelining model)

```
Extra Trees Accuracy: 0.47
Extra Trees Training Time: 27.963784217834473 seconds
```

```
verif_model = Pipeline([('cnt_scr', CenterScaler()), ('pix_scr', PixelScaler()),
                        ('ex_clf', ExtraTreesClassifier(n_estimators=100, random_state=42))])
```

Dataset development

- operator rotate

```
Label +: 2901 samples
Label -: 2940 samples
Label /: 2857 samples
Label 0: 8380 samples
Label 1: 9344 samples
Label 2: 8400 samples
Label 3: 8588 samples
Label 4: 8255 samples
Label 5: 7754 samples
Label 6: 8351 samples
Label 7: 8750 samples
Label 8: 8289 samples
Label 9: 8390 samples
Label =: 2859 samples
Label x: 2901 samples
```

7˚ , -7˚ 추가

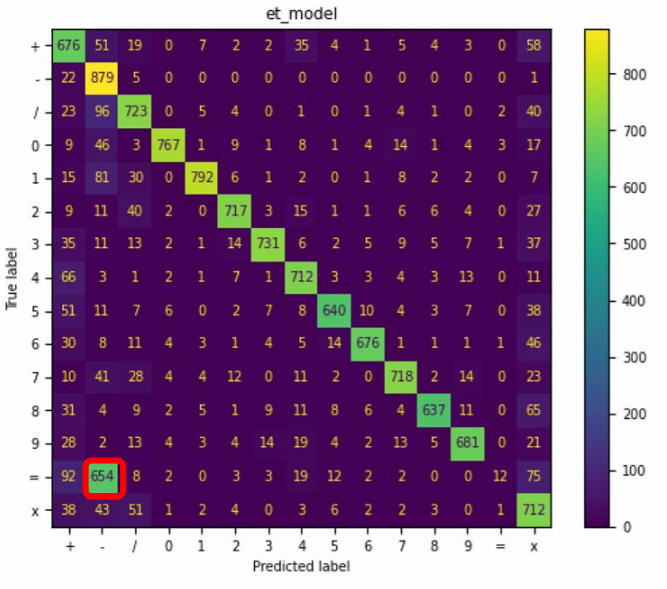
```
Label +: 8685 samples
Label -: 8774 samples
Label /: 8553 samples
Label 0: 8380 samples
Label 1: 9344 samples
Label 2: 8400 samples
Label 3: 8588 samples
Label 4: 8255 samples
Label 5: 7754 samples
Label 6: 8351 samples
Label 7: 8750 samples
Label 8: 8289 samples
Label 9: 8390 samples
Label =: 8555 samples
Label x: 8686 samples
```

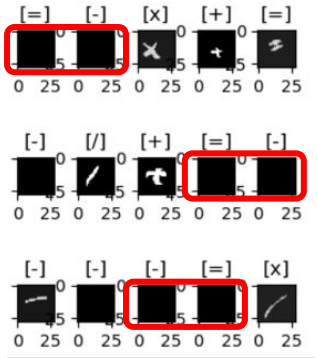
003 Development 과정 및 성능 검증

```
Extra Trees Accuracy: 0.9108730800323362
Extra Trees Training Time: 214.72025561332703 seconds
```

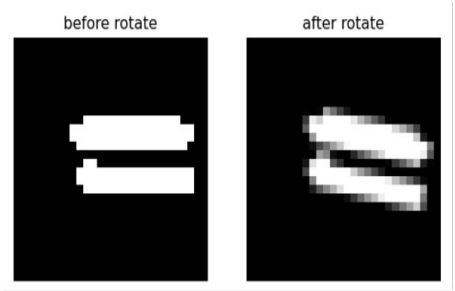
⇨

```
Extra Trees Accuracy: 0.781034349073428
Extra Trees Training Time: 135.0235869884491 seconds
```





003 Development 과정 및 성능 검증




```
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.62890625, 0.53125   , 0.05859375, 0.        , 0.        ,
 0.15625   , 0.34375   , 0.5       , 0.6875    , 0.84375   ,
 1.        , 1.        , 1.        , 1.        , 1.        ,
 0.93847656, 0.34375   , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.09375   , 0.8828125 ,
 1.        , 0.98535156, 0.34375   , 0.1875    , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.03125   , 0.21875   , 0.375     , 0.5625    , 0.75      ,
 0.14160156, 0.01757812, 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.27246094, 1.        ,
 1.        , 1.        , 1.        , 1.        , 1.        ,
 0.8125    , 0.625     , 0.46875   , 0.28125   , 0.125     ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
```

```
rotated = (rotated > 0.6).astype(int)
rotated = (rotated != 0).astype(int)
```

(임계값 설정 후 binary화)

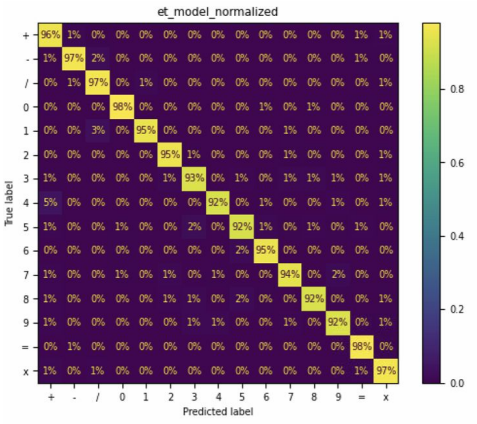
```
Extra Trees Accuracy: 0.9108730800323362
Extra Trees Training Time: 214.72025561332703 seconds
```

⇨

```
Extra Trees Accuracy: 0.9490579204466155
Extra Trees Training Time : 103.21750450134277 seconds
```

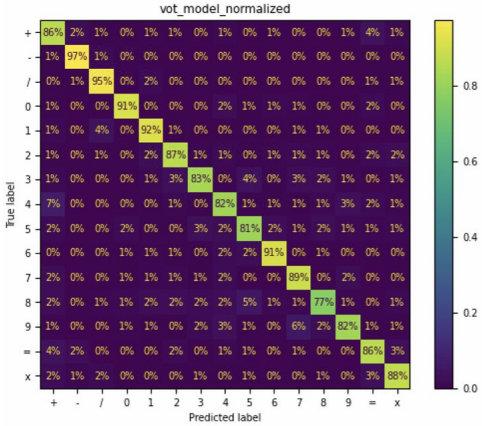
모델별 성능비교

1. Extra Tree



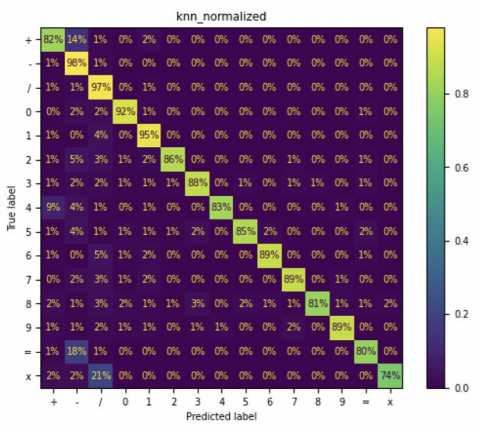

```
Extra Trees Accuracy: 0.9490579204466155
Extra Trees Training Time : 103.21750450134277 seconds
```

2. Voting_clf




```
Voting clf Accuracy: 0.8735364813522525
Voting clf Training Time: 673.5969154834747 seconds
```

3. KNN

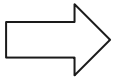



```
knn clf Accuracy: 0.8715205086454214
knn clf Training Time: 21.7827787399292 seconds
```

accuracy 기준 : Extra Tree > Voting_clf > KNN
학습시간 기준 : KNN >> Extra Tree >> Voting_clf

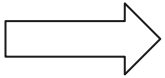
hyperparameter 조정

```
MemoryError: could not allocate 3932160 bytes
```



1. 130000개의 data
2. 784개의 input feature
3. Random patch

1. max_depth
2. min_samples_split
3. min_samples_leaf
4. min_weight_fraction_leaf
5. max_leaf_nodes
6. max_features
7. bootstrap



max_depth , n_estimators, bootstrap 사용

003 Development 과정 및 성능 검증

max_depth / n_estimaotrs	20	25	30	35	40
500	60.54	59.76	88.22	59.62	64.64
1000	74.51	59.68	92.83	59.64	69.61
1500	61.94	59.71	81.83	59.65	59.99

bootstrap = True 고정

004 **결론**

최종모델 성능 확인

최적화 모델 → MNIST

```
Extra Trees MNIST Accuracy: 0.9759
Extra Trees MNIST Training time: 232.67128014564514
```

최적화 모델 →combined

```
Extra Trees test accuracy: 0.8724735322425409
Extra Trees test Training time: 431.06891989707947
```

accuracy 감소의 원인 ?

1. 10 classes → 15 classes
2. handmade data의 low quality

최종모델 Learning curve