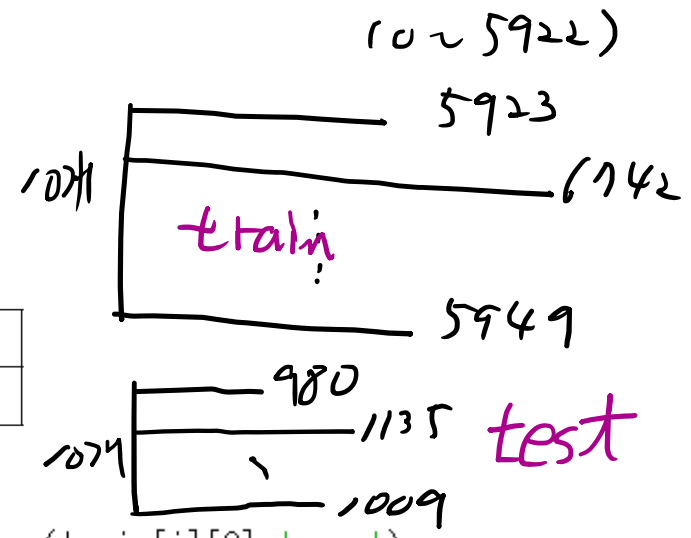
Template Matching

MNIST Dataset

- LeCun 교수가 만들어 공개한 dataset
- 60,000 training data and 10,000 test data
- Handwritten digit images (size 28×28) with 0~1 values
- Each image can be flattened to 784 dimensional vector (1-D numpy array)

MNIST dataset

Train	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
test	980	1135	1032	1010	982	892	958	1028	974	1009




```python
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pdb

def init_data():
    with open('train.bin','rb') as f1:
        train=pickle.load(f1)

    with open('test.bin','rb') as f2:
        test=pickle.load(f2)

    return train,test


train,test=init_data() # read data file:train.bin, test.bin

print(len(train))
print(len(train[0]))
print(train[0][0].shape)
print(len(test))
print(len(test[0]))
print(test[0][0].shape)
```

```
10
5923
(28, 28)
10
980
(28, 28)
```

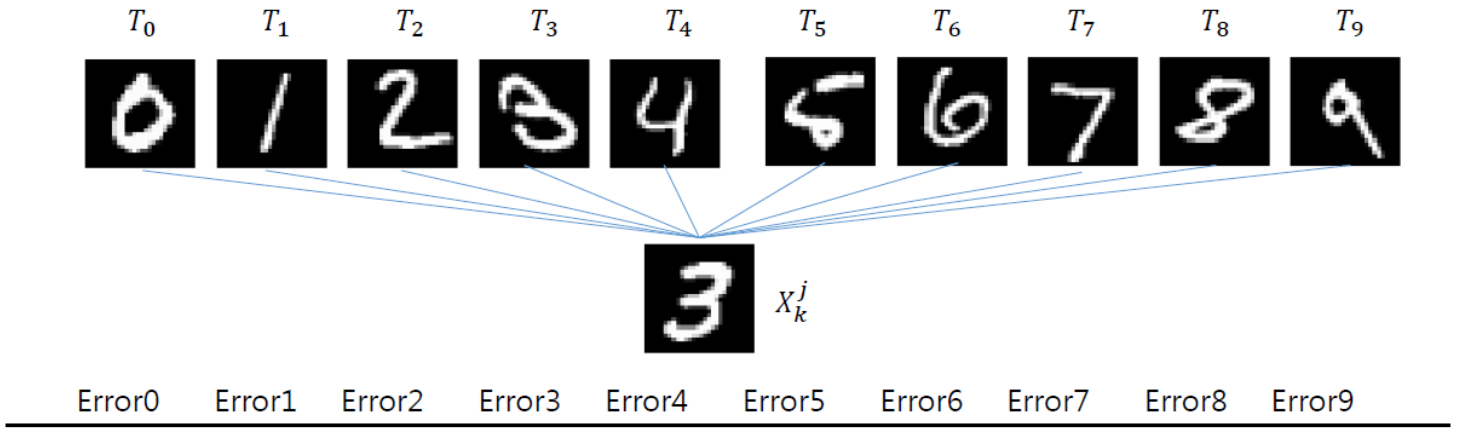
```python
for i in range(10):
    plt.subplot(2,5,i+1),plt.imshow(train[i][0],'gray')
    plt.axis('off')
    print(len(train[i]))

plt.show()
```

```
5923
6742
5958
6131
5842
5421
5918
6265
5851
5949
```

0MNIST.py

Template matching

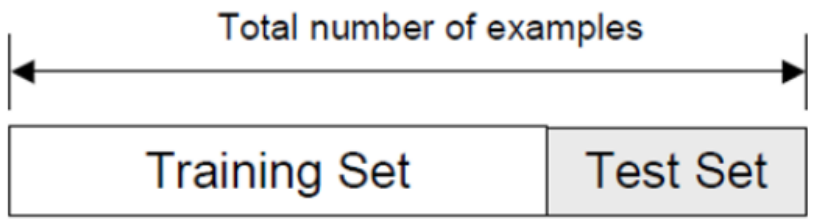
$$\underset{i}{\operatorname{argmin}} \sum_{x,y} |T_i - X_k^j|$$



- 가장 기초적인 인식기
  - ✓ Feature Engineering이 불필요
  - ✓ 모든 픽셀 값과 위치 정보를 특징점(인식기의 입력)으로 사용
  - ✓ Parameter-Free 인식기
  - ✓ Euclidean Distance 기준으로 인식

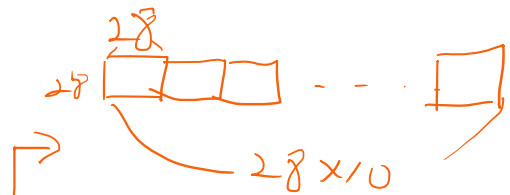
Template matching – hold out method

- training에서 읽어들인 각 클래스별 300개의 데이터를 training set으로, test에서 읽어들인 각 클래스별 100개의 데이터를 test set으로 구분하라.



- Training set의 데이터로 각 클래스별 템플릿 T_i를 구성하라.

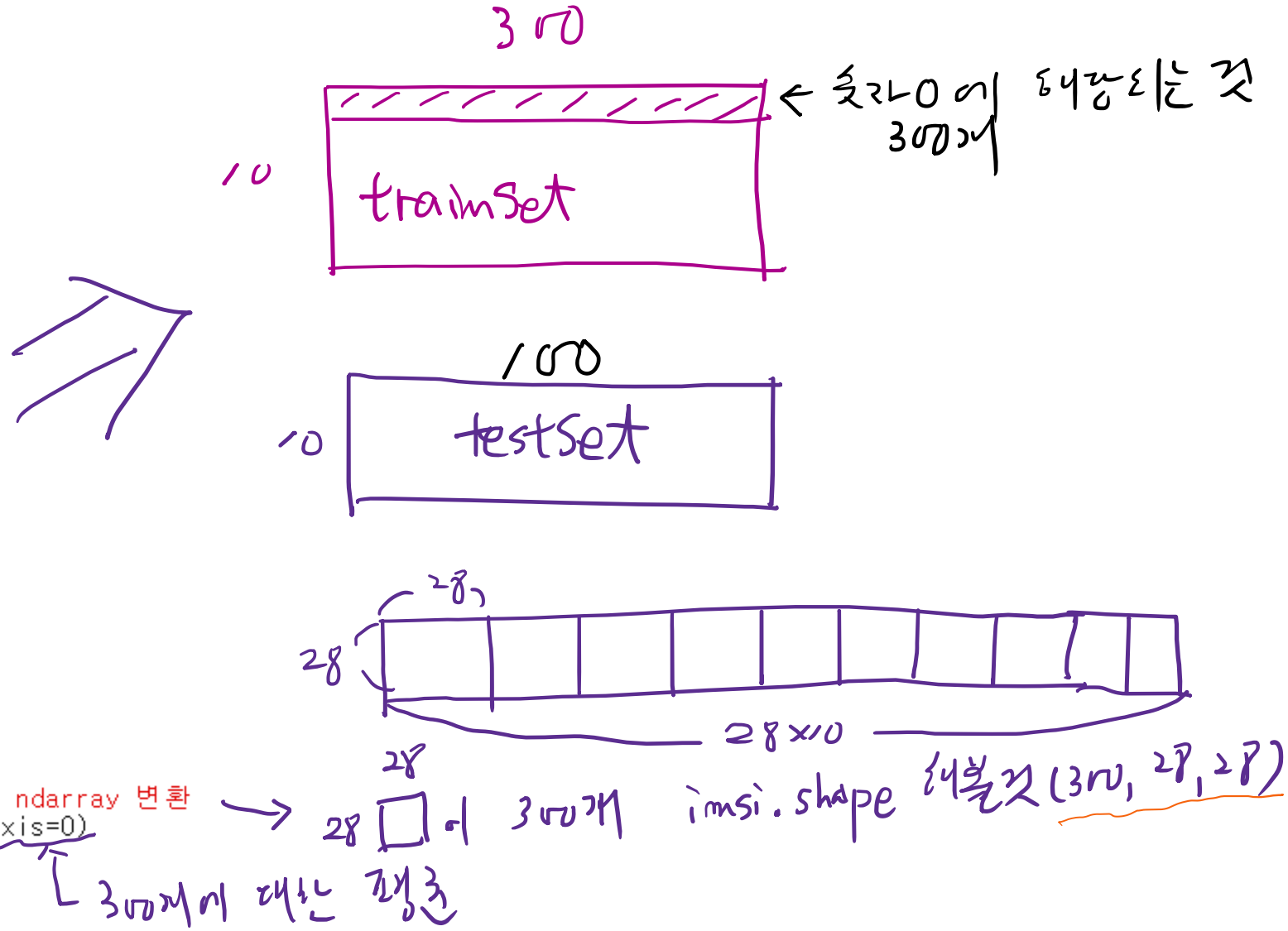
$$T_i = \frac{1}{300}\sum_{k=1}^{300} X_k^i$$




```
def createTmpl(trainSet):
    tmpl=np.zeros((28,28*10))
    print(tmpl.shape)
    for i in range(10):
#         pdb.set_trace()
        imsi=np.array(trainSet[i]) # list -> ndarray 변환
        tmpl[:,i*28:(i+1)*28]=np.mean(imsi,axis=0)
        print(np.mean(imsi,axis=0).shape)
    return tmpl
```

1MatchingMNISTstep1.py

```python
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pdb

def init_data():
    with open('train.bin','rb') as f1:
        train=pickle.load(f1)

    with open('test.bin','rb') as f2:
        test=pickle.load(f2)

    return train,test

def data_ready1(train,test,k=300):
    trainSet=[]
    testSet=[]
#    pdb.set_trace() #중단점 표시
    for i in range(10):
        trainSet.append(train[i][0:k])
        testSet.append(test[i][0:100])
    return trainSet,testSet

def createTmpl(trainSet):
    tmpl=np.zeros((28,28*10))
    print(tmpl.shape)
    for i in range(10):
#        pdb.set_trace()
        imsi=np.array(trainSet[i]) # list -> ndarray 변환
        tmpl[:,i*28:(i+1)*28]=np.mean(imsi,axis=0)
        print(np.mean(imsi,axis=0).shape)
    return tmpl
```



```
train,test=init_data() # read data file:train.bin, test.bin

print(len(train))
print(len(train[0]))
print(train[0][0].shape)

for i in range(10):
    plt.subplot(2,5,i+1),plt.imshow(train[i][0],'gray')
    plt.axis('off')
    print(len(train[i]))

plt.show()

trainSet,testSet=data_ready1(train,test,300) # 일부분의  data 가져오기
print(len(trainSet[0]))
print(len(testSet[0]))

tmpl=createTmpl(trainSet) # template 가져오기
plt.imshow(tmpl)
plt.show()
```

1MatchingMNISTstep1.py

Template matching – hold out method

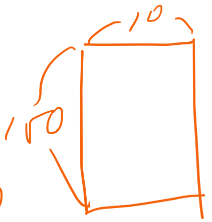
- Template matching을 통해 test set을 인식해보자. 그리고 그 결과를 result, 100x10 행렬에 저장해보자. (tmplMatch)

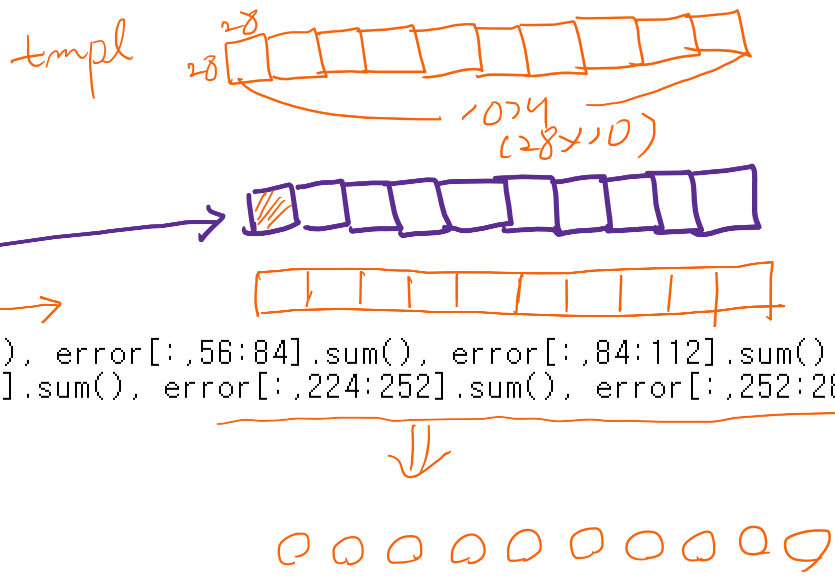
$$\underset{i}{\operatorname{argmin}} \sum_{x,y} |T_i - X_k^j|$$

- 위 성능을 accuracy, precision, recall, f1 score로 표현하라. 클래스별 성능 그리고 전체 성능을 계산하라. (calcMeasure)

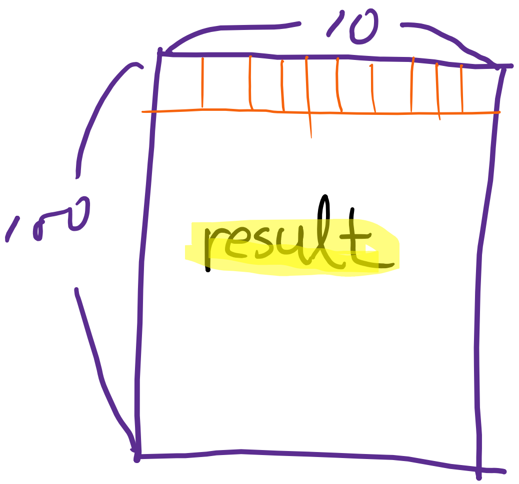
```
def tmplMatch(tmpl, testSet):
    result = np.zeros((100,10))

    for i in range(len(testSet)):
        for j in range(len(testSet[0])):
            imsiTest = np.tile(testSet[i][j], (1,10))
            error = np.abs(tmpl-imsiTest)
            errorSum = [error[:,0:28].sum(), error[:,28:56].sum(), error[:,56:84].sum(), error[:,84:112].sum(), error[:,112:140].sum(),
 error[:,140:168].sum(), error[:,168:196].sum(), error[:,196:224].sum(), error[:,224:252].sum(), error[:,252:280].sum()]
            result[j,i] = np.argmin(errorSum)
    return result
```

```python
def tmplMatch(tmpl, testSet):
    result = np.zeros((100,10))

    for i in range(len(testSet)):
        for j in range(len(testSet[0])):
            imsiTest = np.tile(testSet[i][j], (1,10))
            error = np.abs(tmpl-imsiTest)
            errorSum = [error[:,0:28].sum(), error[:,28:56].sum(), error[:,56:84].sum(), error[:,84:112].sum(), error[:,112:140].sum(),
error[:,140:168].sum(), error[:,168:196].sum(), error[:,196:224].sum(), error[:,224:252].sum(), error[:,252:280].sum()]
            result[j,i] = np.argmin(errorSum)
    return result
```

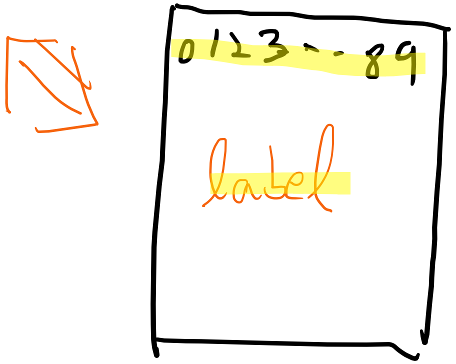




28의 absolute error를 더한 값이 1024.



testset[i][j]에 대한 인식결과
(absolute error sum이 최소인 index)



← 이런 형태가 나오면 제대로 인식한 것임
0 ~ 9 순서에서 어긋나면 인식이 틀린 것!

Confusion Matrix

- **"Accuracy" or "Recognition Ratio"**

$$Accuracy = \frac{No.(Correctly\ Classified\ Samples)}{No.(Presented\ Samples)} \qquad (8.4.1)$$

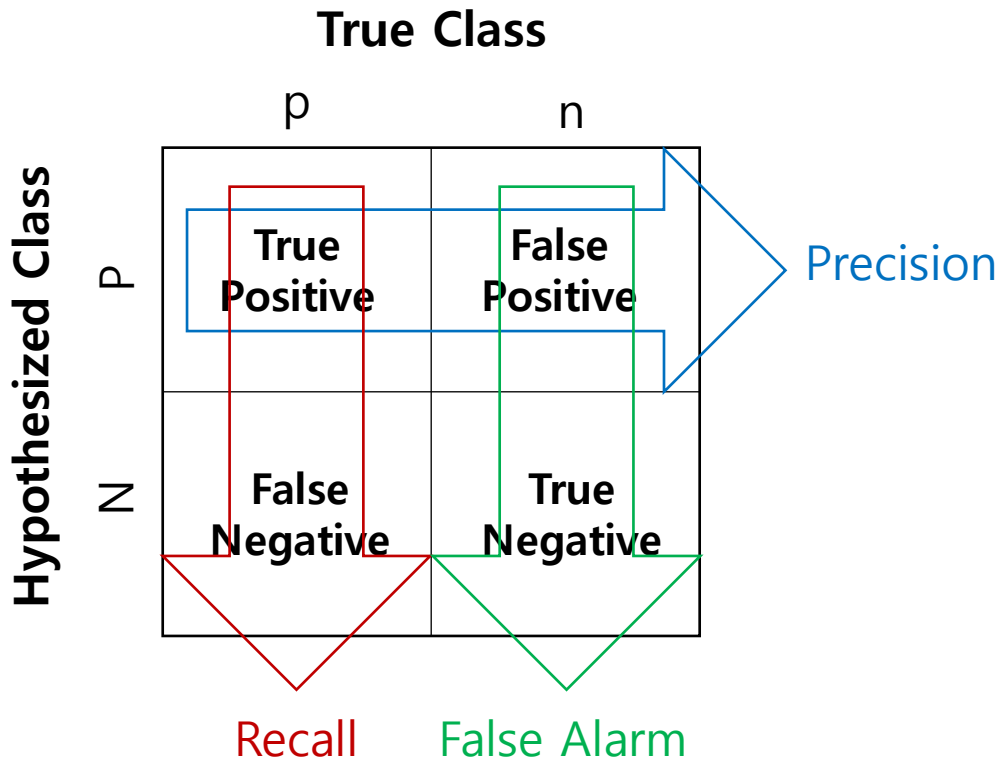
- **"Precision and Recall"**

  TP : True Positive
  FP : False Positive
  TN : True Negative
  FN : False Negative Positive

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

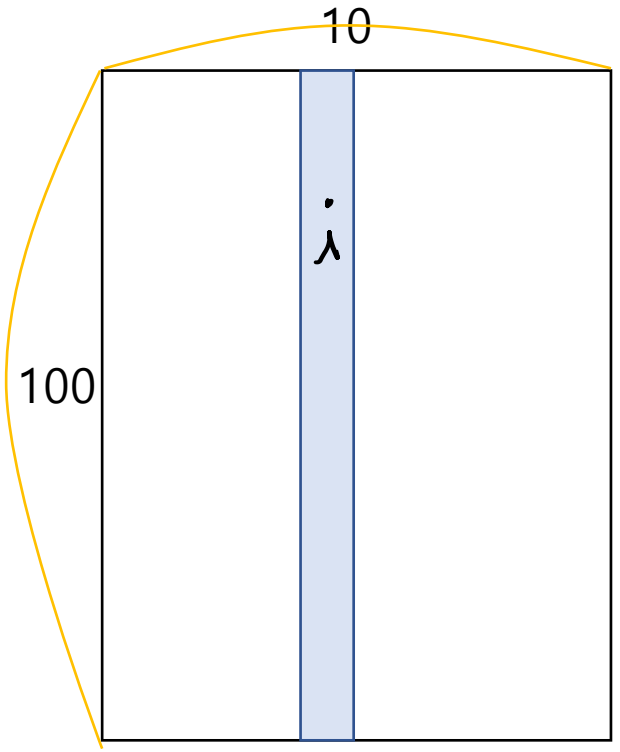
$$Recall = \frac{TP}{TP + FN}$$



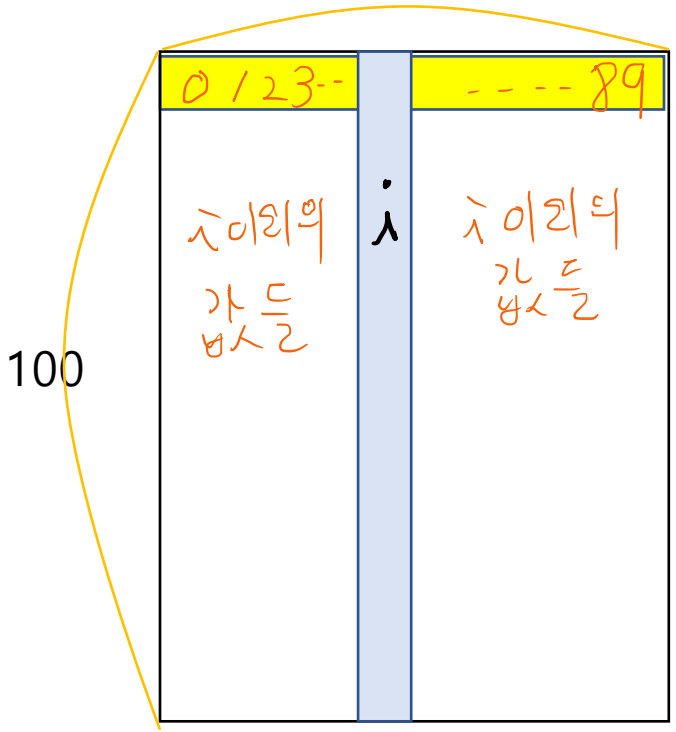
$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

the harmonic mean of precision and recall:

Confusion Matrix for Handwritten Digit Recognition



result



label

TP : label이 i 이고 result도 i 인 것의 개수
(입력이 숫자 i 라고 맞춘 것)

FN : label이 i인데 result는 i가 아닌 것의 개수
(입력이 숫자 i 라고 맞추지 못한 것)

TN : label이 i이외의 것인데 result도 i 이외의 것
(입력이 숫자 i 이외의 것이라고 맞춘 것)

FP : label이 i이외의 것인데 result는 i 인 것의 개수
(입력이 숫자 i 이외의 것이라고 맞추지 못한 것)

		True Class	
		P(숫자 i)	N(others)
Hypothesized Class	P(i)	**True Positive**	**False Positive**
	N(others)	**False Negative**	**True Negative**

[Confusion Matrix for digit i]

1MatchingMNISTrevOH.py

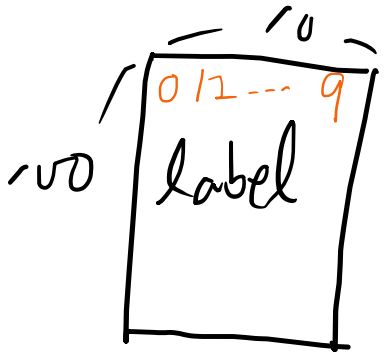
아래의 내용을 1MatchingMNISTstep1.py 에 추가 함

```
def calcMeasure(result):

    # acc = (tp+tn)/ (tp+fn+fp+tn)
    # pre = tp/ (tp+fp)
    # rec = tp/ (tp+fn)
    # f1 = 2*pre*rec/(pre+rec)
    s1, s2 = result.shape
    label = np.tile(np.arange(0,s2), (s1,1))
    pdb.set_trace()

    TP = []; TN = []; FN = []; FP = []
    for i in range(10):
        TP.append(((result == label) & (label == i)).sum())
        TN.append(((result != i) & (label != i)).sum())
#        FP.append(((result != label) & (label == i)).sum())
#        FN.append(((result == i) & (label != i)).sum())
# Below two lines are corrected by S.-H. Oh
        FN.append(((result != label) & (label == i)).sum())
        FP.append(((result == i) & (label != i)).sum())


    TP = np.array(TP); TN = np.array(TN); FN = np.array(FN); FP = np.array(FP)
    acc = (TP+TN)/(TP+TN+FP+FN)
    pre = TP/(TP+FP)
    rec = TP/(TP+FN)
    f1 = 2*pre*rec/(pre+rec)

    return acc, pre, rec, f1


result=tmplMatch(tmpl,testSet) # matching 결과 가져오기
print(result.shape)

acc,pre,rec,f1=calcMeasure(result) # 성능지표 계산
print(acc,pre,rec,f1) # class 별 성능
#print(acc.mean(),pre.mean(),rec.mean(),f1.mean())
```

(s1=100, s2=10)

(0~9)

tile의 세로방향 쌓기



ex)

```
import numpy as np
b=np.array([[1,2,],[3,4]])
np.tile(b,2)
np.tile(b,(2,1))
```

(가로방향)

(세로방향)

1MatchingMNISTrevOH.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pdb

def init_data():
    with open('train.bin','rb') as f1:
        train=pickle.load(f1)

    with open('test.bin','rb') as f2:
        test=pickle.load(f2)

    return train,test

def data_ready1(train,test,k=300):
    trainSet=[]
    testSet=[]
#    pdb.set_trace() #중단점 표시
    for i in range(10):
        trainSet.append(train[i][0:k])
        testSet.append(test[i][0:100])
    return trainSet,testSet

def createTmpl(trainSet):
    tmpl=np.zeros((28,28*10))
    print(tmpl.shape)
    for i in range(10):
#        pdb.set_trace()
        imsi=np.array(trainSet[i]) # list -> ndarray 변환
        tmpl[:,i*28:(i+1)*28]=np.mean(imsi,axis=0)
        print(np.mean(imsi,axis=0).shape)
    return tmpl
```

```
def tmplMatch(tmpl, testSet):
    result = np.zeros((100,10))

    for i in range(len(testSet)):
        for j in range(len(testSet[0])):
            imsiTest = np.tile(testSet[i][j], (1,10))
            error = np.abs(tmpl-imsiTest)
            errorSum = [error[:,0:28].sum(), error[:,28:56].sum(
 error[:,140:168].sum(), error[:,168:196].sum(), error[:,196:224
            result[j,i] = np.argmin(errorSum)
    return result

def calcMeasure(result):

    # acc = (tp+tn)/ (tp+fn+fp+tn)
    # pre = tp/ (tp+fp)
    # rec = tp/ (tp+fn)
    # f1 = 2*pre*rec/(pre+rec)
    s1, s2 = result.shape
    label = np.tile(np.arange(0,s2), (s1,1))
    pdb.set_trace()

    TP = []; TN = []; FN = []; FP = []
    for i in range(10):
        TP.append(((result == label) & (label == i)).sum())
        TN.append(((result != i) & (label != i)).sum())
#        FP.append(((result != label) & (label == i)).sum())
#        FN.append(((result == i) & (label != i)).sum())
# Below two lines are corrected by S.-H. Oh
        FN.append(((result != label) & (label == i)).sum())
        FP.append(((result == i) & (label != i)).sum())
```

```
    TP = np.array(TP); TN = np.array(TN); FN = np.array(FN); FP = np.array(FP)
    acc = (TP+TN)/(TP+TN+FP+FN)
    pre = TP/(TP+FP)
    rec = TP/(TP+FN)
    f1 = 2*pre*rec/(pre+rec)

    return acc, pre, rec, f1

train,test=init_data() # read data file:train.bin, test.bin

print(len(train))
print(len(train[0]))
print(train[0][0].shape)

for i in range(10):
    plt.subplot(2,5,i+1),plt.imshow(train[i][0],'gray')
    plt.axis('off')
    print(len(train[i]))

plt.show()

trainSet,testSet=data_ready1(train,test,300) # 일부분의  data 가져오기
print(len(trainSet[0]))
print(len(testSet[0]))

tmpl=createTmpl(trainSet) # template 가져오기
plt.imshow(tmpl)
plt.show()

result=tmplMatch(tmpl,testSet) # matching 결과 가져오기
print(result.shape)

acc,pre,rec,f1=calcMeasure(result) # 성능지표 계산
print(acc,pre,rec,f1) # class 별 성능
#print(acc.mean(),pre.mean(),rec.mean(),f1.mean())
```

Step1.

Template matching – hold out method

- Template 생성에 사용하는 TrainSet의 개수를 10, 50, 100, 200, 300, 500, 1000으로 늘려가며, 아래 표를 완성시키라.

F1	Total	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
#10											
#50											
#100											
#200											
#300											
#500											
#1000											

Template matching – random subsampling