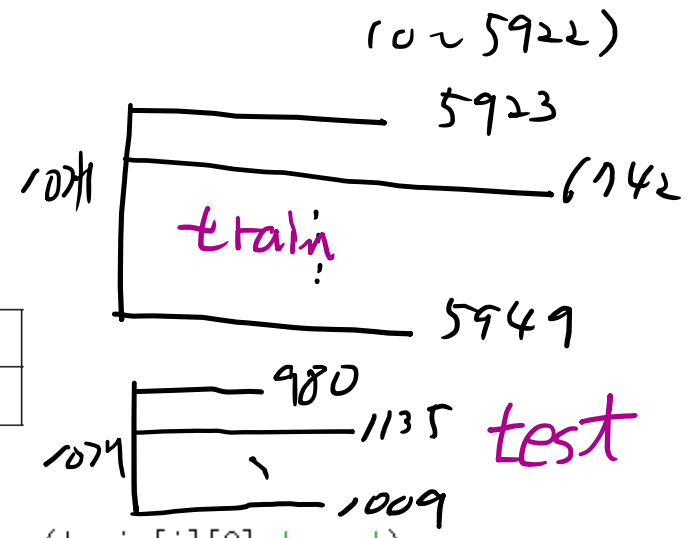
k Nearest Neighbor Algorithm

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)
```

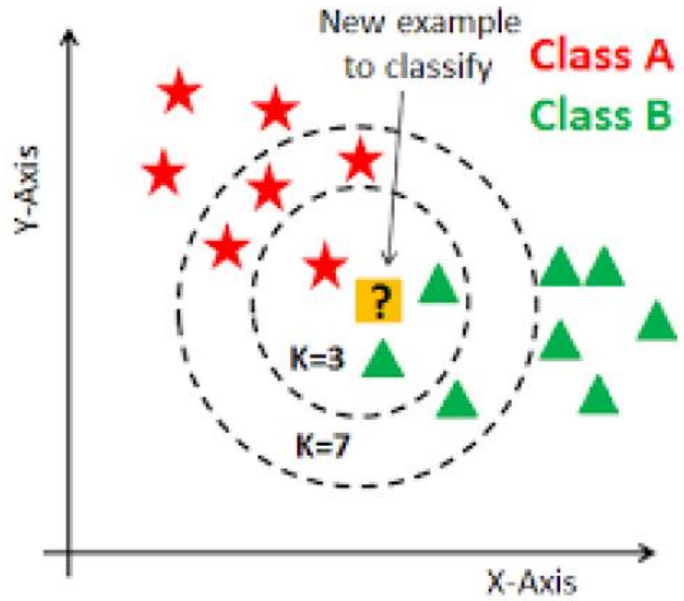
```
10
5923
(28, 28)
10
980
(28, 28)
```

```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

k Nearest Neighbor Algorithm

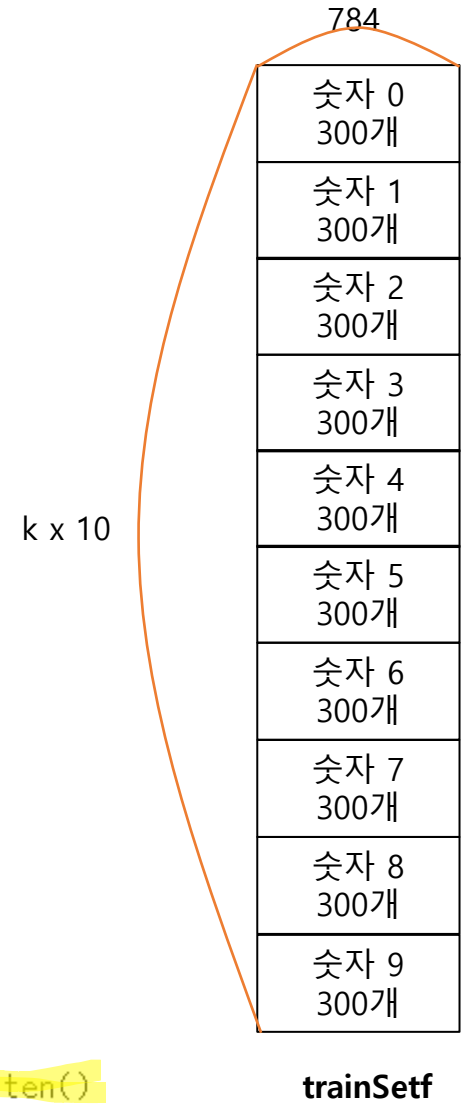


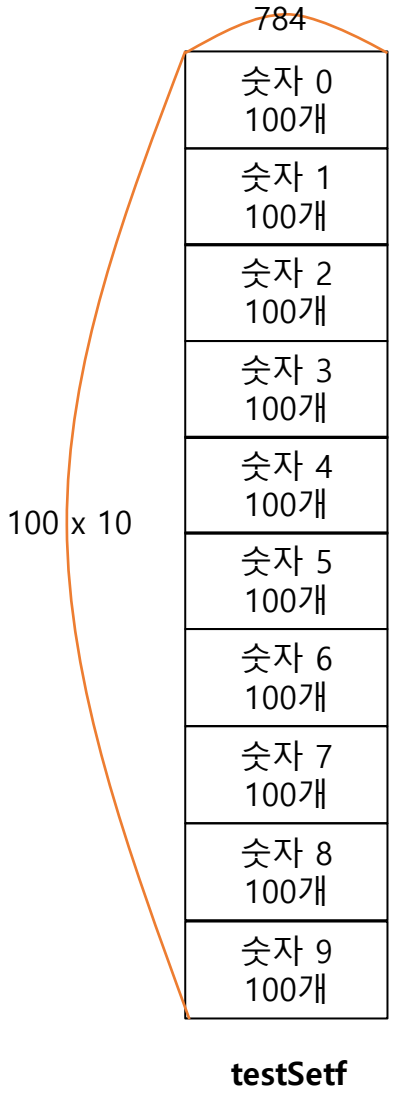
k-Nearest Neighbor
Classify (**X**, **Y**, *x*) // **X**: training data, **Y**: class labels of **X**, *x*: unknown sample
for $i = 1$ **to** m **do**
  Compute distance $d(\mathbf{X}_i, x)$
end for
Compute set I containing indices for the k smallest distances $d(\mathbf{X}_i, x)$.
return majority label for $\{\mathbf{Y}_i$ where $i \in I\}$

- 특징
  - ✓ 하나의 파라미터 k
  - ✓ 단순함에 비해 압도적인 인식률
- 단점
  - ✓ 데이터의 크기에 비례하는 계산량 증가(속도가 느리다)
  - ✓ 메모리 사용량이 크다

functionsRev.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 data_ready2(train,test,k=300):
    trainSetf=np.zeros((k*10,28*28))
    testSetf=np.zeros((100*10,28*28))

    for i in range(len(train)):
        for j in range(k):
            trainSetf[i*k+j,:]=train[i][j].flatten()
    for i in range(len(test)):
        for j in range(100):
            testSetf[i*100+j,:]=test[i][j].flatten()
    return trainSetf,testSetf
```



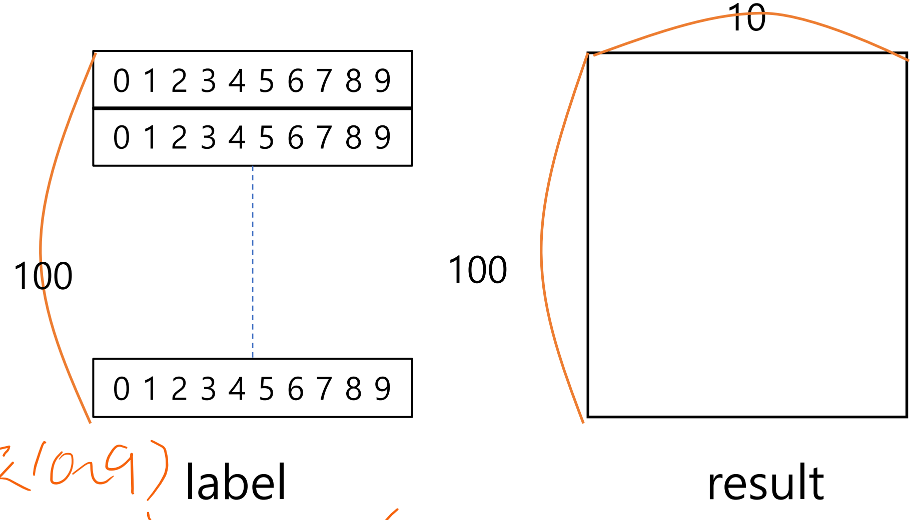


⑤ histogram에서 max에 해당하는 argument가 인식결과!

```
def knn(trainSet, testSet, k):

    trS1, trS2=trainSet.shape
    teS1, teS2=testSet.shape
    trS3=int(trS1/10)
    teS3=int(teS1/10)

    label=np.tile(np.arange(0,10),(teS3,1))
    result=np.zeros((teS3,10))

    for i in range(teS1):
        imsi=np.sum((trainSet-testSet[i,:])**2,axis=1)
        #pdb.set_trace()
        no=np.argsort(imsi)[0:k]
        hist,bins=np.histogram(no//trS3,np.arange(-0.5,10.5,1))
        result[i%teS3,i//teS3]=np.argmax(hist)
    return result
```

trS1=3000, trS2=784, teS1=1000, teS2=784

trS3=300, teS3=100



→ no를 trS3로 나눈 몫(0~9) → 어느 숫자에 해당되는지의 정보!

→ 0~99 → 0~9

① 3000×784 와 1×784의 broadcasting 계산 → 3000×784

② axis=1로 더하기 → imsi는 3000개의 거리계산결과!

③ no: k개의 nearest neighbor sort되어 나옴. (몇번째 trainSet에 해당되는지에 대한 argument)

④ [0 1 2 --- 9]에 histogram 만들기.

```python
def createTmpl(trainSet):
    tmpl=np.zeros((28,28*10))
    print(tmpl.shape)
    for i in range(10):
        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
 error[:,140:168].sum(), error[:,168:196].sum(), error[
            result[j,i] = np.argmin(errorSum)
    return result
```

```python
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))

    TP = []; TN = []; FN = []; FP = []
    for i in range(10):
##          TP.append(((result == label) & (label == i)).sum())
##          TN.append(((result == label) & (label != i)).sum())
##          FP.append(((result != label) & (result == i)).sum())
##          FN.append(((result != label) & (label == i)).sum())
        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
```

2kNNMNISTRev.py

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

#import functions as fs
import functionsRev as fs


t1=time.time()
train,test=fs.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=fs.data_ready2(train,test) # 일부분의  data 가져오기

result=fs.knn(trainSet,testSet,k=10)

acc,pre,rec,f1=fs.calcMeasure(result) # 성능지표 계산
t2=time.time()
print(acc,pre,rec,f1) # class 별 성능
print(t2-t1)
print('Acc=',acc.mean())
print('F1=',f1.mean())
```

Flatten 수행하여 784차원의 데이터를 가져옴

K Nearest Neighbor – hold out method

- k를 1, 5, 15, 50, 100, 200, ~~500~~으로 늘려가며, 아래 표를 완성시키라.

F1	Total	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
K=1											
K=5											
K=15											
K=50											
K=100											
K=200											

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, *, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*, ***kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int*, *default=5*

Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'} or callable, default='uniform'*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*

the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of `DistanceMetric` for a list of available metrics. If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

Methods

fit(X, y)	Fit the k-nearest neighbors classifier from the training dataset.
get_params([deep])	Get parameters for this estimator.
kneighbors ([X, n_neighbors, return_distance])	Finds the K-neighbors of a point.
kneighbors_graph ([X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict(X)	Predict the class labels for the provided data.
predict_proba(X)	Return probability estimates for the test data X.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

fit(*X*, *y*) [source]

2kNNMNISTscikitRev.py

```
from sklearn.neighbors import KNeighborsClassifier
#import functions as fs
import functionsRev as fs
import numpy as np
import time

t1=time.time()
train,test=fs.init_data() # read data file:train.bin, test.bin


trainSet,testSet=fs.data_ready2(train,test) # 일부분의  data 가져오기
label=np.tile(np.arange(0,10),(300,1))

knn=KNeighborsClassifier(n_neighbors=10,weights="distance",metric="euclidean")
#knn=KNeighborsClassifier(n_neighbors=10)
knn.fit(trainSet,label.T.flatten()) # T : transpose
result=knn.predict(testSet)
result=result.reshape(10,100).T


acc,pre,rec,f1=fs.calcMeasure(result) # 성능지표 계산, acc.sum()해볼것
t2=time.time()
print(acc,pre,rec,f1) # class 별 성능
print(t2-t1)
print('Acc=',acc.mean())
print('F1=',f1.mean())
```