**8.2 – knn은 학습한 데이터는 모두 인식하는가?**

=> 학습할 데이터가 지나치게 많아질 경우에 정확도가 떨어진다. 랜덤데이터로 학습을 하게 된다면 데이터의 수가 많아질수록 모델의 성능은 향상되지만 중복되는 데이터가 많아지고 질적인 문제로 정확도가 떨어지는 것 같다.

#1. 학습데이터(랜덤)를 10으로 하고 랜덤범위를 2로 적게하여 정확도를 출력

----------------------------------------------------------------------------------------------------

L = 10  
data = np.random.randint(0, 2, (L, 2)).astype(np.float32)  
labels = np.random.randint(0, 2, (L, 1)).astype(np.float32)  
  
knn = cv2.ml.KNearest\_create()  
  
import time  
s\_time = time.time()  
knn.train(data, cv2.ml.ROW\_SAMPLE, labels)  
  
from collections import Counter  
K = [1]  
def print\_ret\_values(ret\_val, rslt\_v='no', nfg\_v='no', dst\_v='no'):  
 retval, results, neighbours, dist = ret\_val  
 print('results:', type(results), results.shape)  
 if rslt\_v != 'no': print(results)  
 print('neighbours:', type(neighbours), neighbours.shape)  
 if nfg\_v != 'no': print(neighbours)  
 print('dist:', type(dist), dist.shape)  
 if dst\_v != 'no': print(dist)  
  
def get\_accuracy(predictions, labels):  
 accuracy = (np.squeeze(predictions) == labels).mean()  
  
for k in K:  
 print(f'\ntest=train: k={k}, num of test data={len(data)} -------')  
 s\_time = time.time()  
 ret\_val = knn.findNearest(data, k)  
 e\_time = time.time()  
 print(f'testing time: whole={e\_time - s\_time:#.2f}, unit={(e\_time - s\_time)/len(data):#.2f}')  
 print\_ret\_values(ret\_val, rslt\_v='yes', nfg\_v='yes', dst\_v='yes')  
 print(labels)  
 ret, results, neighbours, dist = ret\_val  
  
 cmp = labels == results  
 cmp\_f = cmp.flatten()  
 dict = Counter(cmp\_f)  
 print(f'test=train: L={L}, k={k}: Accuracy={dict[True] \* 100 / len(cmp):#.2f}%')

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **data** | **results** | **neighbours** | **dist** | **labels** |
| [[0. 1.]  [1. 1.]  [1. 0.]  [0. 1.]  [1. 0.]  [1. 0.]  [0. 1.]  [0. 0.]  [0. 0.]  [1. 0.]] | [[1.]  [0.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]] | [[1.]  [0.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]  [1.]] | [[0.]  [0.]  [0.]  [0.]  [0.]  [0.]  [0.]  [0.]  [0.]  [0.]] | [[1.]  [0.]  [1.]  [1.]  [0.]  [0.]  [1.]  [1.]  [1.]  [0.]] |



데이터가 [1.0.]일 때 results와 neighbours가 [1.]인데 labels가 [0.]으로 출력되어 불일치하는 것을 확인했다. 앞서 3번째와 같은 데이터의 값인 [1.0]에서는 labels은 [1.]으로 할당으로 했지만 다음 똑같은 데이터의 값인 [1.0]에서는 labels을 [0.]으로 할당했습니다. 기존 labels에 할당된 값이 같은 것에 새롭게 다시 labels을 부여하여 정확도가 떨어진다.

테스트 할려는 데이터가 많을수록 중복되는 데이터가 많아질 수 밖에 없다. 하지만 이 중복된 데이터가 다 똑같은 값을 할당하지 않기 때문에 정확도가 떨어진다. 또한 랜덤 데이터는 특정 패턴이나 구조가 부족할 수 있기 때문에, 데이터의 특성을 제대로 반영하지 못하고 성능 저하를 초래할 수 있다. 랜덤데이터는 실험적 또는 시뮬레이션된 데이터로 어떠한 구조적 패턴이나 연관성을 반드시 반영하지 않을 수 있다. 이는 종종 데이터의 특성이 불명확하거나, 특정 문제에 대해 잘못된 정보를 제공할 수 있다. 랜덤 데이터는 실제 문제와 관련성이 낮고 노이즈도 많다. 모델에서 유의미한 정보를 제공하지 못하고, 모델이 불필요하거나 잘못된 패턴으로 학습할 경우도 있다.

하지만 랜덤데이터의 범위를 높이고 학습데이터의 수를 적게한다면 중복되는 데이터가 적어지기 때문에 정확도는 높아질 것이다. 중복된 값들이 얼마나 정확한 값을 할당하는 지에 따라서에 달라진다.

Knn모델에서는 k=1일때는 본인자신에 대한 거리만 측정하기 때문에 정확도가 당연히 높을 것이라고 생각한다. 하지만 앞서 학습데이터의 수가 10개이고 랜덤범위가 적을 때 주변을 생각하지 못해 이미 존재하는 데이터가 치명적일 수 있다. 그래서 중복되는 데이터를 다 똑같은 할당을 하는지에 따라 다르기 때문에 코딩을 실행했을 때마다 정확도가 계속해서 달라진다.







# 2. 20 x 20 으로 5000개의 문자데이터의 학습정확도 출력

---------------------------------------------------------------------

SIZE\_IMAGE\_ROW = 20  
SIZE\_IMAGE\_COL = 20  
NUMBER\_CLASSES = 10

def load\_digits\_and\_labels(big\_image):  
 digits\_img = cv2.imread(big\_image, 0)  
 number\_cols = digits\_img.shape[1] / SIZE\_IMAGE\_COL  
 number\_rows = digits\_img.shape[0] / SIZE\_IMAGE\_ROW  
 rows = np.vsplit(digits\_img, number\_rows)

digits = []  
 for row in rows:  
 row\_cells = np.hsplit(row, number\_cols)  
 for digit in row\_cells:  
 digits.append(digit)  
 digits = np.array(digits)

labels = np.repeat(np.arange(NUMBER\_CLASSES), len(digits) / NUMBER\_CLASSES)  
 return digits, labels

def get\_accuracy(predictions, labels):  
 accuracy = (np.squeeze(predictions) == labels).mean()  
 return accuracy \* 100

digits, labels = load\_digits\_and\_labels('../data/digits.png')

for i in digits[0]:  
 print(i)  
shuffle = np.random.permutation(len(digits))  
digits, labels = digits[shuffle], labels[shuffle]  
  
raw\_descriptors = []  
for img in digits:  
 raw\_descriptors.append(np.float32(img.flatten()))  
raw\_descriptors = np.array(raw\_descriptors)  
partition = int(0.5 \* len(raw\_descriptors))  
raw\_descriptors\_train, raw\_descriptors\_test = np.split(raw\_descriptors, [partition])  
labels\_train, labels\_test = np.split(labels, [partition])  
  
knn = cv2.ml.KNearest\_create()  
knn.train(raw\_descriptors\_train, cv2.ml.ROW\_SAMPLE, labels\_train)  
  
ret, result, neighbours, dist = knn.findNearest(raw\_descriptors\_test, 1)  
acc = get\_accuracy(result, labels\_test) # test 데이터를 분류했을 때의 정확도를 확인  
print(f"k={1}: Accuracy for test data: {acc:#6.2f}")  
  
ret, result, neighbours, dist = knn.findNearest(raw\_descriptors\_train, 1)  
acc = get\_accuracy(result, labels\_train) # 학습데이터를 분류했을 때의 정확도를 확인  
print(f"k={1}: Accuracy for train data: {acc:#6.2f}")

스크린샷, 블랙, 디자인이(가) 표시된 사진

자동 생성된 설명



Digit.png에서는 총5000개의 데이터에서 중복된 값이 거의 없다는 것을 확인할 수 있다. 그래서 K=1일 때에는 주변을 생각하지 않고 자기자신을 검증하기 때문에 당연히 학습데이터의 정확도가 100%이다.

또한 이미지 데이터는 특정 구조와 패턴을 가지고 있고, 시각적인 정보도 포함되어 있다. 보통 높은 차원의 데이터이며, 다양한 특성을 가지고 있다. 랜덤 데이터보다 매우 구체적인 정보를 포함하고 있어서 많은 정보의 데이터를 통해 대량의 데이터에서도 높은 정확도를 출력할 수 있다.

Knn은 학습한 데이터를 모두 인식하긴 하다. 아무리 K=1이라도 학습할 데이터가 얼마나 질적인 문제를 갖고 있느냐에 따라서 정확도가 달라지는 것 같다. k-NN 알고리즘 자체에서 자동적으로 일반화가 이루어진다고 보기는 어렵다. k-NN 모델의 성능을 최적화하기 위해서는 데이터 전처리, 적절한 k 값 선택, 차원 축소 등 다양한 기법을 적용해야 할 뿐만 아니라 학습할 데이터에 대한 이해도 필요하다고 생각한다.