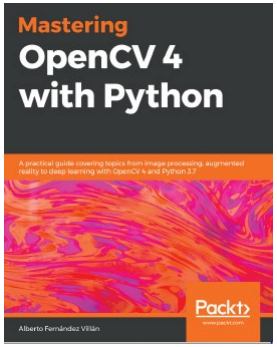
OpenCV Python

디지털영상처리/컴퓨터비전

k nearest neighbor



Mastering OpenCV 4 with Python, Alberto, Packt, Pub. 2019

교재 10장의 일부..

2021년 2학기

서경대학교 김진헌

기본 개념

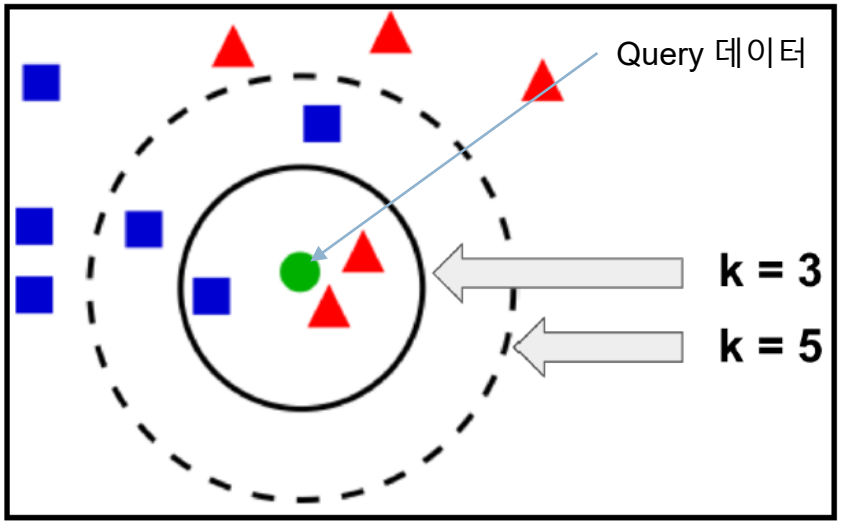
1

- KNN은 지도 학습의 단순한 형태
  - instance-based learning, or lazy learning
    - 학습데이터의 generalization을 보류. ⇔ eager learning

학습데이터를 쌓아두었다가 문의가 들어오면 판단
음반, 영화 처럼 과거 학습데이터가 별로 필요없을 때 유효.

학습을 완료하여 일반화가
이루어지면 문의를 받는다. 예:
DNN 문자인식

- 분류와 회귀모델에서 모두 사용가능하다.
  - 회귀모델에서는 근접한 여러 레이블의 출력값을 평균하여 출력한다



그림에서 초록 원으로 표기된 데이터는 k=3을 정하면
빨간 삼각형 그룹으로 간주하지만 k=5로 정하면
숫자가 더 많은 푸른 사각형 그룹으로 간주한다.

k-nearest neighbor의 기본 아이디어
1. 먼저 학습 데이터를 통해 clustering을 실행한다. 이 과정은 반드시 k-means 알고리즘으로 수행할 필요는 없다.
2. testing 과정에서는 미지의 테스트 데이터가 들어오면 주변 k개의 데이터가 어떤 클러스터에 속하는지
살펴서 가장 많은 개수의 데이터가 속하는 클러스터에 속하는 것으로 판정한다.

그림에서 빨간 삼각형과 파란 사각형은
미리 학습시켜 놓은 모델이 보유하고
있는 데이터이다.

knn clustering의 단점(1)

- Dimensionality 증가에 불리
  - 유클리디안 거리로 최근접 분류를 한다면 차원이 높아질 수록 거리 차이는 별로 없는 상황이 발생할 가능성이 높다.
    - 몇개의 차원이 심각하게 분류 판단에 작용할 경우가 있음.
  - 대책: 아날로그 값에 대해서는 correlation coefficient를 사용할 수도 있다.

- σ_X is the standard deviation of X
- μ_X is the mean of X
- σ_Y is the standard deviation of Y
- μ_Y is the mean of Y

$$\rho_{X,Y} = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

같으면 1, 반대면 -1, 관련이 없으면 0

knn clustering의 단점(2)

- 다수결 투표의 한계
  - 분포가 편향되어 있을 때– 많은 분포를 가진 레이블이 결정에 영향을 줄 것임.
    - 거리를 가중치에 반영하여 영향을 축소
    - 데이터 표현을 추상화하는 방법도 있다
      - self-organizing map (SOM)에서는 각 포인트가 분포도에 상관없이 어떤 레이블을 대표한다. knn을 SOM에 적용한다.
- 개선 방안: Feature Extraction
  - 예: PCA, LDA, CNN
  - 학습할 데이터가 방대할 때 학습 데이터를 특징을 추출해 feature vector로 줄여서 knn에 입력한다.
  - Dimension reduction에도 기여한다.

OpenCV가 다루고 있는 머신 러닝 자료 소개 - in python

opencv machine learning tutorials

- 위치: http://docs.opencv.org ⇒ main page ⇒ opencv-python tutorials ⇒ machine learning

•**K-Nearest Neighbour**

•Learn to use kNN for classification Plus learn about handwritten digit recognition using kNN

•**Support Vector Machines (SVM)**

•Understand concepts of SVM

•**K-Means Clustering**

•Learn to use K-Means Clustering to group data to a number of clusters. Plus learn to do color quantization using K-Means Clustering

cv::ml::KNearest Class Reference abstract

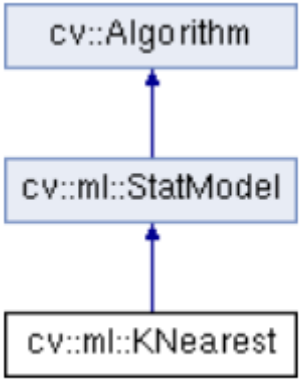
Machine Learning

The class implements K-Nearest Neighbors model.

```
#include <opencv2/ml.hpp>
```

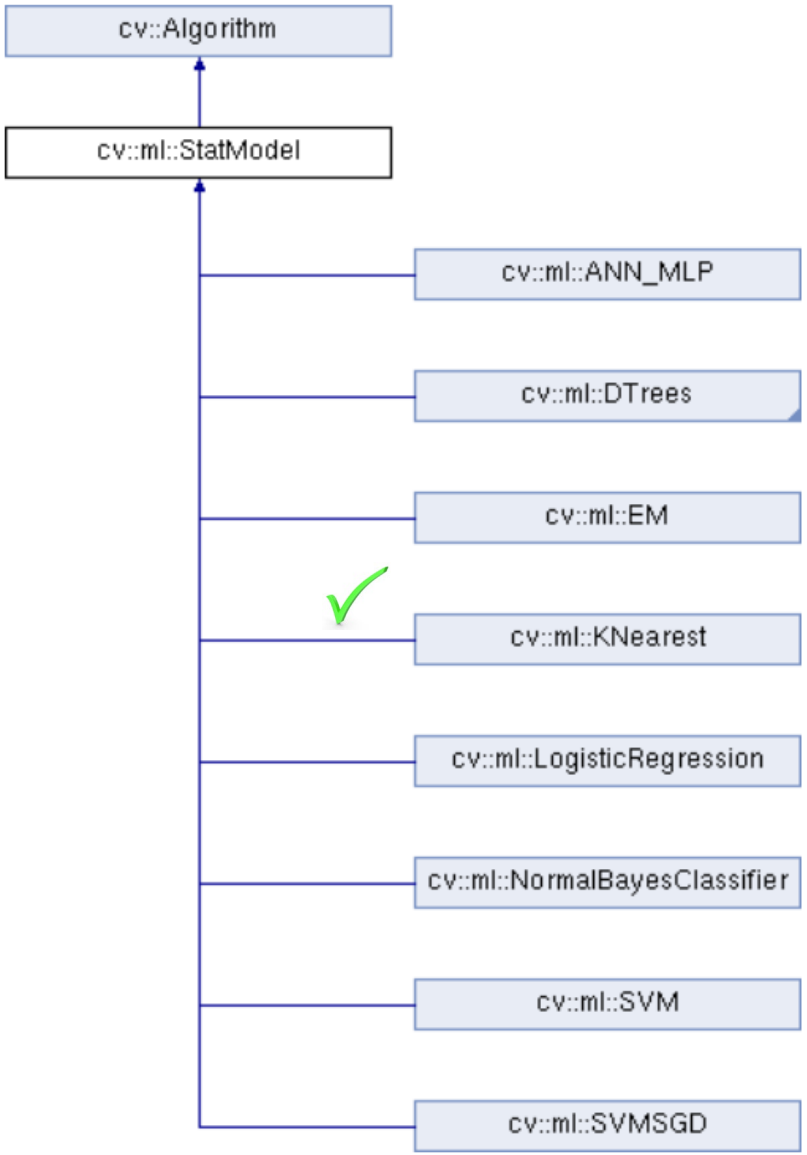
Inheritance diagram for cv::ml::KNearest:

knn 함수의 상속 다이어그램



공통함수
train(), predict()

전용함수: findNearest



knn 함수

Static Public Member Functions

static **Ptr**< **KNearest** >	**create** ()
	Creates the empty model. More...
static **Ptr**< **KNearest** >	**load** (const **String** &filepath)
	Loads and creates a serialized knearest from a file.

A static member function is a special member function, which is used to access only static data members, any other normal data member cannot be accessed through static member function. Just like static data member, static member function is also a class function;
파이썬에서는 정적변수는 클래스 내부에는 선언되었지만, 메서드에서는 선언되지 않은 변수를 말한다. 클래스 변수와 차이가 없는 듯 하다. 참조

Public Member Functions

virtual float	**findNearest** (**InputArray** samples, int k, **OutputArray** results
	const =0
	Finds the neighbors and predicts responses for input vectors.

The public member function can access any private, protected and public member of its class. Not only public member function, any member function of a class can access each and every other member declared inside the class.

KNN을 사례로 본…

ml 모듈의 학습과 testing 절차

- creation: 모델객체 생성
  - knn = cv2.ml.KNearest_create()
  - ANN의 사례: retval=cv.ml.ANN_MLP_create()
  - SVM의 사례: retval=cv.ml.SVM_create()

- training: 학습 데이터를 이용한 모델 학습
  - knn.train(data, cv2.ml.ROW_SAMPLE, labels)
  - k-NN training: 학습용 데이터, data와 이를 지도학습 시킬 수 있는 label이 필요하다.

- testing: 미지의 데이터 세트에 대한 응답 반환
  - ret, results, neighbours, dist = knn.findNearest(sample, k)
  - retval, results=cv.ml_StatModel.predict(samples[, results[, flags]])

knn의 학습 함수

knn = cv2.ml.KNearest_create()
여기서 생성한 객체가 이곳으로 들어간다.

knn

- *retval = cv.ml_StatModel.train(samples, layout, responses)*
  - cv.ml.Knearest_create가 생성한 객체의 메소드로 구현
- *samples*: training samples
- layout: 학습 데이터가 row 배열인지, column 배열인지 지정

ROW_SAMPLE Python: cv.ml.ROW_SAMPLE	each training sample is a row of samples
COL_SAMPLE Python: cv.ml.COL_SAMPLE	each training sample occupies a column of samples

- *responses*: vector of responses associated with the training samples. *label=어떤 그룹에 속하는지를 숫자로 표현*

- retval, 학습 기법에 대한 문서를 찾을 수 없음. 정상 수행이면 True인듯.
- 학습된 데이터를 모두 맞히지 못하는 상황이 있는 것을 보면 자체 generalization 알고리즘이 있는 것으로 추정됨.

예제 1_0: knn 분류 기법의 간단한 활용 사례

1_0_knn_introduction.py

- 개요:
  - 본 프로그램은 k-Nearest Neighbour의 활용 기법을 연마하기기 위한 것이다.
  - 임의의 테스트 데이터 1개가 어느 그룹에 속하는지를 knn 알고리즘(k=3)을 이용해 판단한 결과를 보인다.
- 절차:
  - 1) 임의로 생성한 16개의 (x,y) 좌표 *data*를 임의로 2개의 그룹으로 나누어 학습 데이터를 생성한다.
  - 랜덤하게 배정된 *labels*를 이용하여 학습 데이터를 생성하였다.
  - 2) test data, *sample*을 1개 랜덤하게 생성한다. 이 데이터가 어느 그룹에 속하는지를 k-nn 방법으로 분류할 예정이다.
  - 3) knn 학습 모델을 구축한다.
    - **knn** = cv2.ml.KNearest_create()
    - knn.train(*data*, cv2.ml.ROW_SAMPLE, *labels*)

예제 1_0: knn 분류 기법의 간단한 활용 사례

- 4) test 데이터가 어느 그룹에 속하는지 k-nn으로 분류한다.
  - ret, results, neighbours, dist = knn.findNearest(*sample*, k)
    - test 데이터 수를 N이라고 하면;
    - result: 테스트 데이터가 속한 그룹의 레이블, [N, 1]
    - neighbours: 가까운 거리의 레이블을 k개를 순서대로 나열, [N, k]
    - dist: 테스트 데이터와 가까운 학습데이터의 거리를 순서대로 반환, [N, k]
- 5) 위 결과를 그림으로 표현한다.

```
num of train data=16, num of test data=1, k=3
result: [[0.]], shape=(1, 1)
neighbours: [[0. 1. 0.]], shape=(1, 3)
distance: [[ 565. 1753. 2341.]], shape=(1, 3)
```

```
# Take points with label 0 (will be the red triangles) and plot them:
red_triangles = data[labels.ravel() == 0]
plt.scatter(red_triangles[:, 0], red_triangles[:, 1], 200, 'r', '^')

# Take points with label 1 (will be the blue squares) and plot them:
blue_squares = data[labels.ravel() == 1]
plt.scatter(blue_squares[:, 0], blue_squares[:, 1], 200, 'b', 's')

# Plot the sample point:
plt.scatter(sample[:, 0], sample[:, 1], 200, 'g', 'o')
```

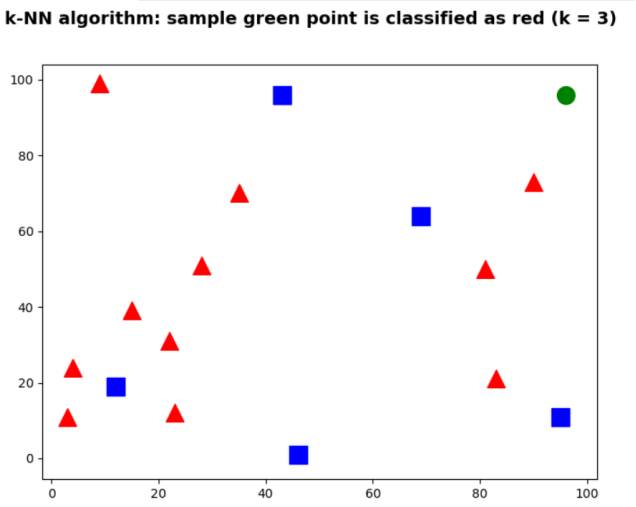


그림 해석: 녹색원은 타이틀에 적힌 대로 k=3일 때 red triangle 그룹에 속하는 것으로 판단되었다.

예제 1_2: 랜덤 데이터와 랜덤 레이블에 기반한 knn모델의 분류 시연 프로그램

1_2_knn_introduction_varying_k.py

- 개요:
  - 본 프로그램은 k-Nearest Neighbour의 활용 기법을 연마하기기 위해 학습 모델이 있다고 가정하고, k-nn 알고리즘이 k에 따라 어떻게 다른 결과를 나타낼 수 있는지 보이고자 한다.
  - 본 예제의 상황은 데이터의 레이블이 랜덤으로 설정되었기 때문에 근접한 데이터라고 해도 사실 전혀 상관이 없다.
  - 이 때문에 k를 크게 하면 할 수록 받아들이기 어려운 결과를 낼 수 밖에 없다.
- 절차:
  - 단계 1: 학습 데이터를 만든다. 랜덤한 값(x, y)으로 데이터를 L개 생성한다.
    - 이들을 2개의 그룹으로 랜덤하게 나눈다.
  - 단계 2: knn 학습 모델을 구축한다.
  - 단계 3: test data를 N개 생성한다.
  - 단계 4: test 데이터가 어느 그룹에 속하는지 k를 바꾸어 가며 knn으로 분류한다.
    - 이때 학습 데이터와 테스트 데이터 및 분류 결과를 화면에 한 화면에 보인다.
    - 이후 k를 바꾸어 가면 같이 실행한 결과를 보인다.

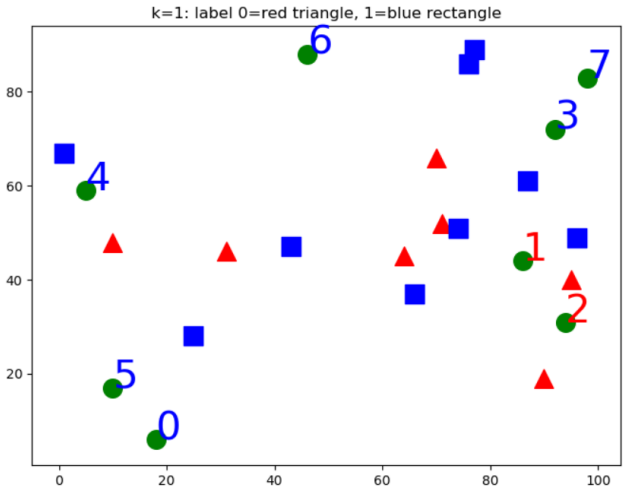
예제 1_2: 랜덤 데이터와 랜덤 레이블에 기반한 knn모델의 분류 시연 프로그램

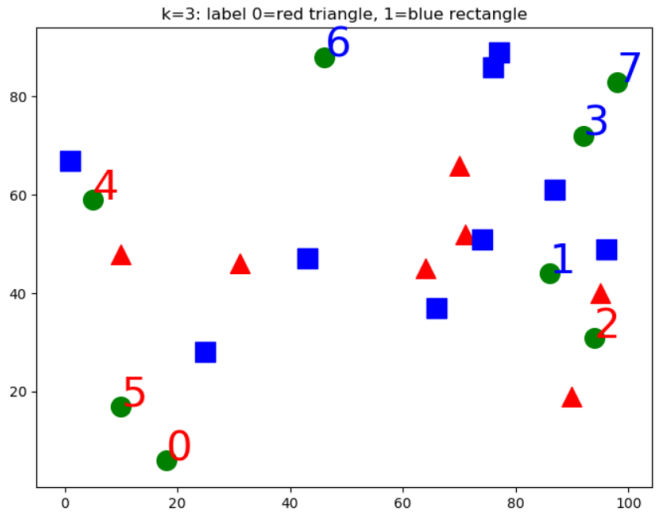
1_2_knn_introduction_varying_k.py

```
Step 1: L=16, Making random training data & labels..
data for training: <class 'numpy.ndarray'> (16, 2)
Labels for training: <class 'numpy.ndarray'> (16, 1)

Step 2: Creating & training a knn model..

Step 4: N=8, Making random test data ..
sample: <class 'numpy.ndarray'> (8, 2)

Step 5: L=16, N=8, Testing the test data  ..
```

```
test=sample: L=16, N=8, k=1-------
results: <class 'numpy.ndarray'> (8, 1)
neighbours: <class 'numpy.ndarray'> (8, 1)
dist: <class 'numpy.ndarray'> (8, 1)

test=sample: L=16, N=8, k=3-------
results: <class 'numpy.ndarray'> (8, 1)
neighbours: <class 'numpy.ndarray'> (8, 3)
dist: <class 'numpy.ndarray'> (8, 3)
```

같은 데이터(학습, 테스트)에 대해 k=1, k=3에 대해 분류 결과가 다르다.





학습데이터는 라벨이 0이면 red 삼각형, 1이면 blue 사각형으로 표기되어 있다.
녹색원은 테스팅하는 데이터(8개)로 분류 결과에 따라 번호(0~7)로 표기되어 있다,
KNNN 판단 결과가 red 혹은 blue 색상의 번호로 표기하였다.

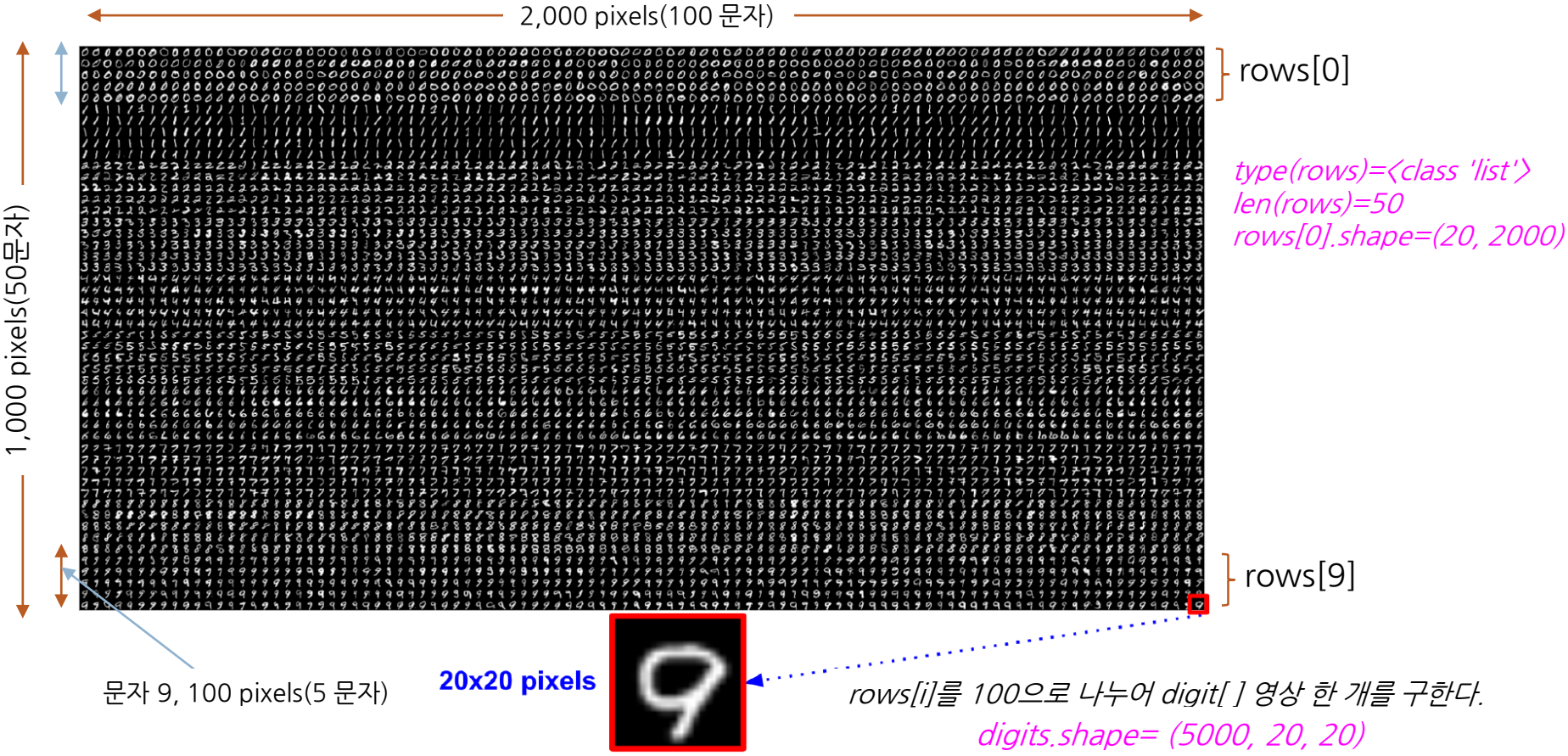
총 16개의 학습데이터를 랜덤하게 0, 1의 라벨을 붙여 빨간 삼각형(0), 파란 사각형(1)으로 표기하였다.

추가로 8개의 랜덤 위치에 존재하는 녹색점을 표기하였다.

점의 우측 상단에 숫자가 테스트 데이터의 일련 번호이다.
같은 데이터에 대해 K=1, 3일 때의 결과를 비교하여 보인 것이다.

예제 2_0: knn을 이용한 필기체 문자인식

2_0_knn_handwritten_digits_recognition_introduction_b.py



digit.png은 20x20으로 이루어진 손글씨 폰트가 100x50(가로x세로)개로 만들어진 것이다.
각 다섯줄 마다 같은 문자가 가로로 100개씩 나열되어 있다. 문자 0이 5줄, 문자 1이 5줄, …, 문자 9가 5줄.
전체적으로 문자 0이 500개, 문자 1이 500개, …, 문자 9가 500개 총합 5,000개의 문자 폰트가 배열되어 있다.

예제 2_0: knn을 이용한 필기체 문자인식

2_0_knn_handwritten_digits_recognition_introduction_b.py

- 1. 영상 파일을 읽어들이고 라벨링을 행한다.
  - digits.png에는 20x20으로 이루어진 손글씨 폰트가 100x50(가로x세로)개 나열되어 있다. 레이블(0~9)은 프로그램으로 자체 생성한다.
  - 영상파일을 가로로 자른 후 다시 세로로 잘라 (5000, 20, 20)의 어레이 변수에 저장한다.
  - digits.png에는 20x20으로 이루어진 손글씨 폰트가 100x50(가로x세로)개 나열되어 있다. 레이블(0~9)은 프로그램으로 자체 생성한다.
  - 영상파일을 가로로 자른 후 다시 세로로 잘라 (5000, 20, 20)의 어레이 변수에 저장한다.
- 2. 학습을 위해 영상 데이터를 랜덤 변수를 이용하여 뒤 섞는다.
- 3. 파일로 존재하였던 영상데이터를 학습 데이터의 개수(5000개) 길이의 리스트로 재편성한다.
- 4. png 파일에서 제공한 당초의 학습용 데이터를 knn 모델링하기 위한 학습 데이터와 테스트 데이터로 반씩 나눈다.

2~4. 학습에 적합한 데이터로 정렬한다.

- 5. knn 모델을 생성하고, 학습데이터로 학습시킨다.
  - OpenCV에서는 두 함수가 필요하다. - create(), train()
- 6. 나머지 반의 데이터로 테스트를 행하고 분류 성능을 분석한다.
  - 사용되는 함수: 결과=findNearest(테스트_데이터, k)
  - k변화에 따른 정확도 변화를 관찰한다. 학습된 데이터를 테스팅하니 K=1이면 100% 정확도를 보였다.

예제 2_0: knn을 이용한 필기체 문자인식

2_0_knn_handwritten_digits_recognition_introduction_b.py

```
@@1. trainin data(20x20 image) and labels(0~9) will be produced;
(1) digits_img.shape= (1000, 2000)
```

영상 해상도: 가로x세로=2,000x1,000

```
(2a) number_cols= 100.0
(2b) number_rows= 50.0
```

글자의 수: 가로x세로=100x50

```
(3) type(rows)=<class 'list'> | len(rows)=50 | rows[0].shape=(20, 2000)

(4a) len(digits)= 5000
(4b) digits.shape= (5000, 20, 20)
(4c) np.arange(NUMBER_CLASSES)= [0 1 2 3 4 5 6 7 8 9]
(4d) type(labels)= <class 'numpy.ndarray'> | labels.shape= (5000,)
```

20x20의 폰트를 가진 글자가 총 5,000개

5,000개의 라벨을 생성한다.

```
@@2. Training data being shuffled.
1) type(shuffle)= <class 'numpy.ndarray'> shuffle.shape= (5000,)

@@3. Arrange the training data as sequence list for knn-modeling
Elements of the list are flattened images.
1) raw_descriptors: type= <class 'list'> | length= 5000
2) raw_descriptors[0].shape= (400,)
3) raw_descriptors: type= <class 'numpy.ndarray'> | shape= (5000, 400)
```

5,000개의 데이터를 생성한다. 데이터의 차원이 400인 셈이다.

예제 2_0: knn을 이용한 필기체 문자인식

2_0_knn_handwritten_digits_recognition_introduction_b.py

```
@@4. Spliting the data into training and testing (50% for each one)
1) labels_train: type= <class 'numpy.ndarray'> shape= (2500,)
2) labels_test: type= <class 'numpy.ndarray'> shape= (2500,)

@@5. Training KNN model - raw pixels as features

@@6. Test the created model.
1) type(ret)= <class 'float'> ret= 1.0
2) result.shape= (2500, 1) | result[0]= [1.]
3) neighbours.shape= (2500, 3) | neighbours[0]= [1. 1. 1.]
4) dist.shape= (2500, 3) | dist[0]= [ 88687. 116511. 144120.]
k=1: Accuracy for test data:  93.72
k=3: Accuracy for test data:  93.00
k=5: Accuracy for test data:  92.60
k=7: Accuracy for test data:  92.28

k=1: Accuracy for train data: 100.00
k=3: Accuracy for train data:  96.36
k=5: Accuracy for train data:  95.12
k=7: Accuracy for train data:  94.16
```

2,500개의 테스트에 대한 분류 결과
그 중 첫번째 결과는 문자 1이다.

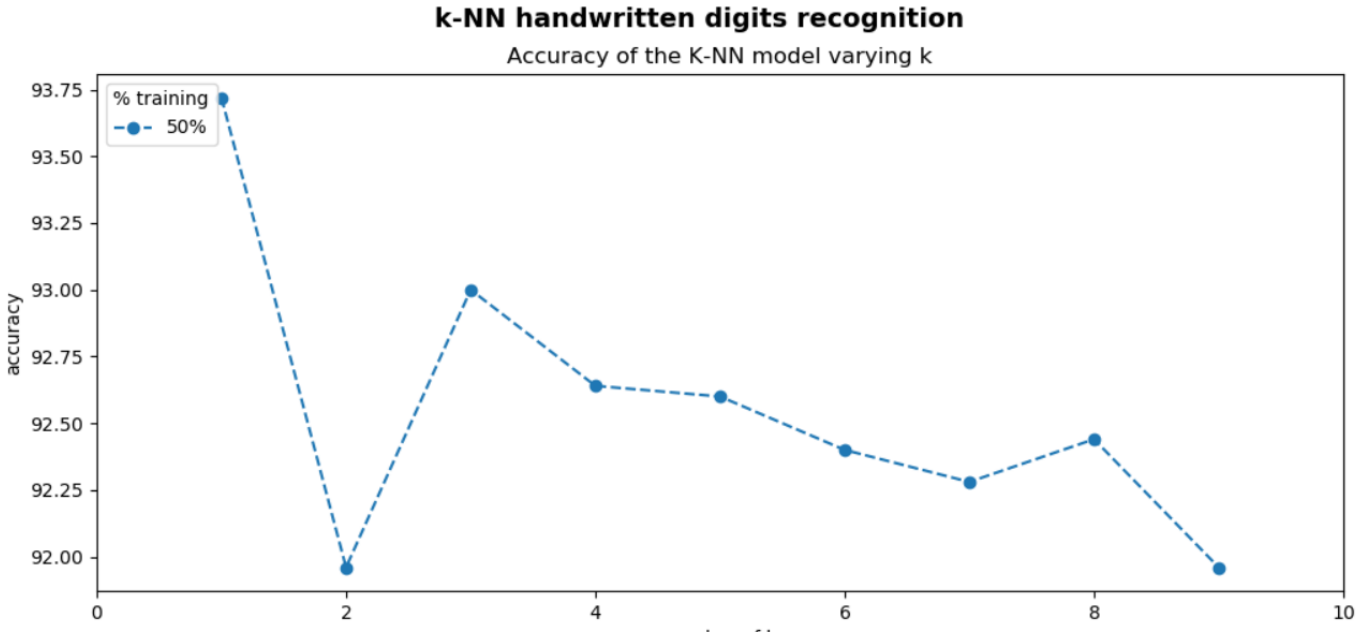
가까운 3개의 판별결과도 모두
문자 1이다.

현 실험 결과로는 k=1일 때가 미지의 데이터에 대해 가장 정확도가 높다.
최종 확인을 위해서는 cross validation 절차가 더 필요하다.
데이터 군집을 다양하게 자르고 k를 다양하게 바꾸어 가면서 정확도가 가장 높을 때를 실험적으로 검증

예제 2_1: K를 바꾸어가며 정확도를 변화를 관찰

2_1_knn_handwritten_digits_recognition_varying_k.py

```
type(labels_test)= <class 'numpy.ndarray'>  | labels_test.shape= (2500,)
Training KNN model - raw pixels as features
[]
k=1 : 93.72
k=2 : 91.96
k=3 : 93.00
k=4 : 92.64
k=5 : 92.60
k=6 : 92.40
k=7 : 92.28
k=8 : 92.44
k=9 : 91.96
[93.72, 91.96, 93.0, 92.64, 92.60000000000001, 92.4, 92.28, 92.44, 91.96]
```

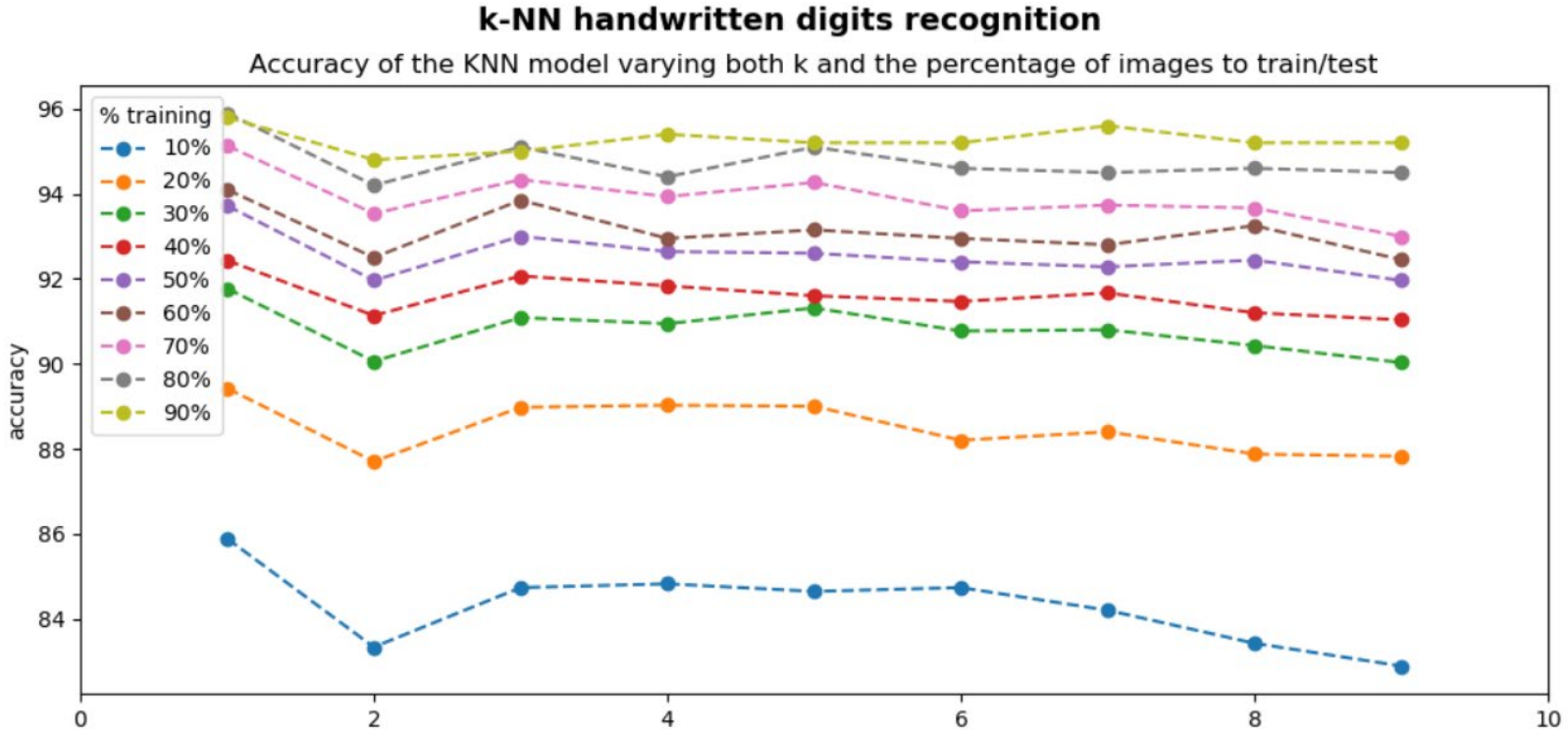


예제 2_2: 학습에 사용되는 데이터의 비율을 높혀가며 정확도를 관찰(k증가)

18 2_2: knn_handwritten_digits_recognition_k_training_testing.py

```
Training KNN model using 3500 training data, which is 70.00% ...
1500: k:Acc => 1:95.1 2:93.5 3:94.3 4:93.9 5:94.3 6:93.6 7:93.7 8:93.7 9:93.0

Training KNN model using 4000 training data, which is 80.00% ...
1000: k:Acc => 1:95.9 2:94.2 3:95.1 4:94.4 5:95.1 6:94.6 7:94.5 8:94.6 9:94.5

Training KNN model using 4500 training data, which is 90.00% ...
500: k:Acc => 1:95.8 2:94.8 3:95.0 4:95.4 5:95.2 6:95.2 7:95.6 8:95.2 9:95.2
```

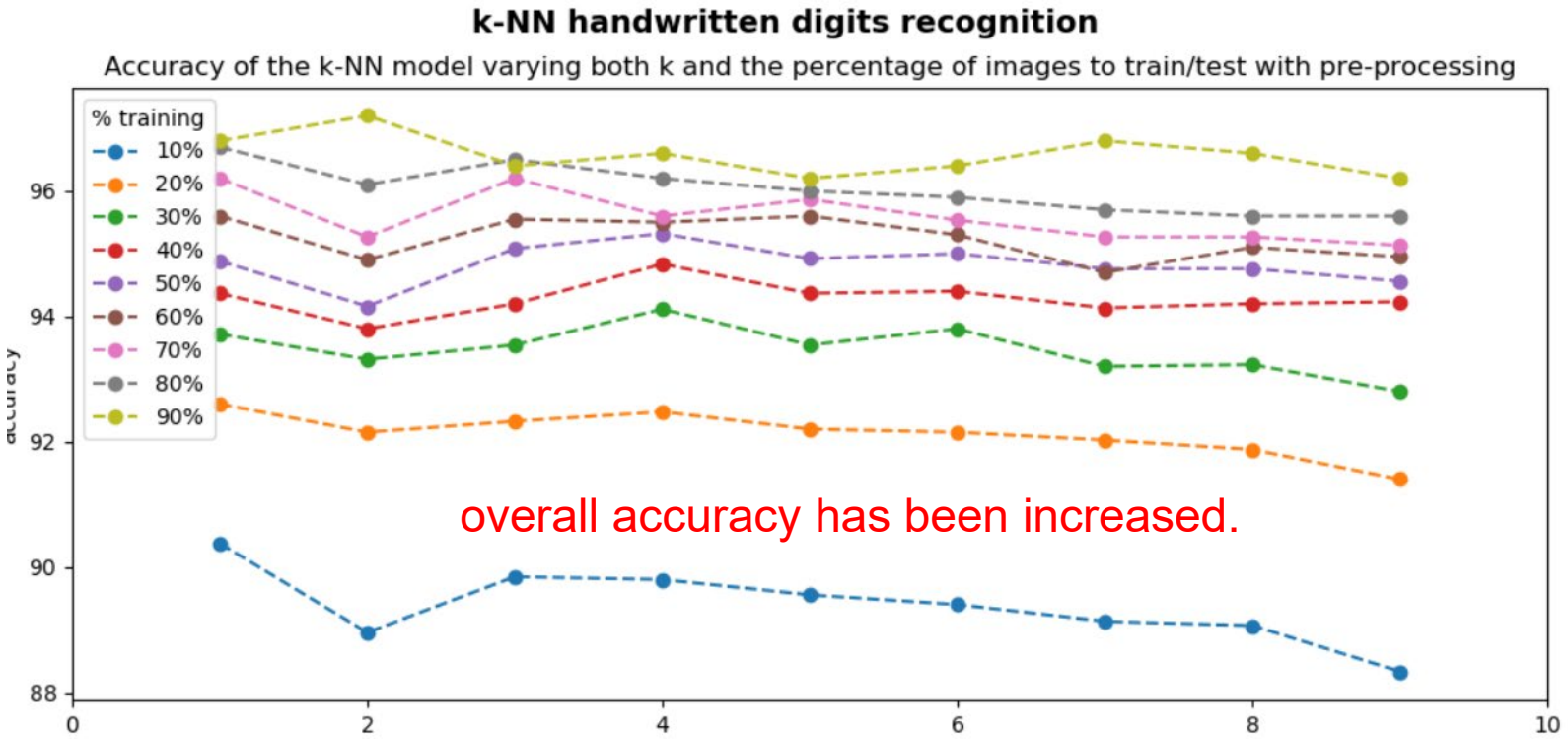
K=9일 때 정확도는 93.0



예제 2_3: 전처리(skew 보정)를 통해 정확도를 높혀 본 사례

19 2_3_knn_handwritten_digits_recognition_k_training_testing_preprocessing.py

```
Training KNN model using 3500 training data, which is 70.00% ...
1500: k:Acc => 1:96.2 2:95.3 3:96.2 4:95.6 5:95.9 6:95.5 7:95.3 8:95.3 9:95.1

Training KNN model using 4000 training data, which is 80.00% ...
1000: k:Acc => 1:96.7 2:96.1 3:96.5 4:96.2 5:96.0 6:95.9 7:95.7 8:95.6 9:95.6

Training KNN model using 4500 training data, which is 90.00% ...
500: k:Acc => 1:96.8 2:97.2 3:96.4 4:96.6 5:96.2 6:96.4 7:96.8 8:96.6 9:96.2
```



예제 2_3: 전처리(skew 보정)를 통해 정확도를 높혀 본 사례

2_3_knn_handwritten_digits_recognition_k_training_testing_preprocessing.py

원본 영상

Skew를 바로 잡은 영상
이것을 knn 학습 및 테스트 데이터로 사용한다.

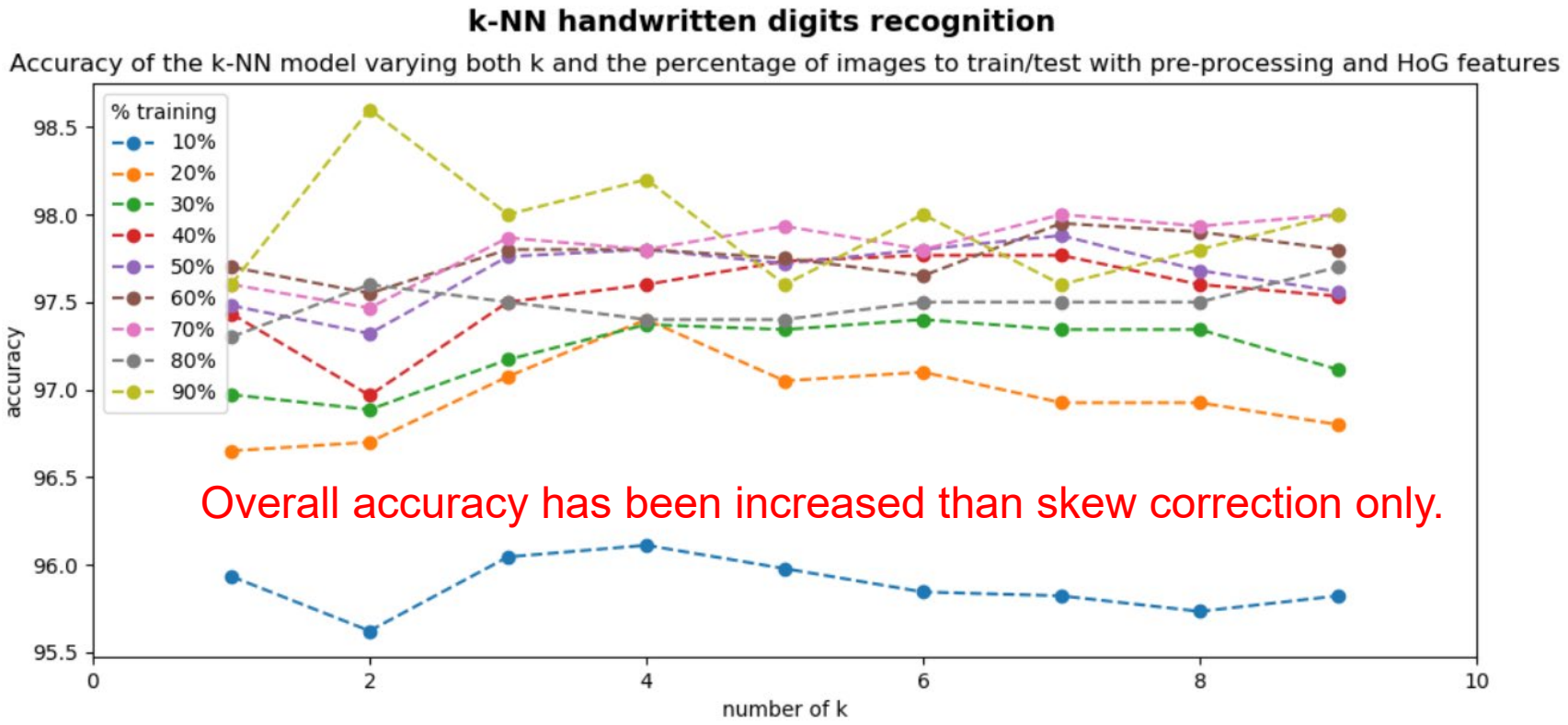
a measure of the skew can be calculated by the ratio of the two central moments (mu11/mu02).

```
m = cv2.moments(img)
if abs(m['mu02']) < 1e-2:
    return img.copy()
skew = m['mu11'] / m['mu02']
M = np.float32([[1, skew, -0.5 * SIZE_IMAGE * skew], [0, 1, 0]])
img = cv2.warpAffine(img, M, (SIZE_IMAGE, SIZE_IMAGE),
                     flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
```

예제 2_4: 전처리(skew보정/hog descriptor)를 통해 정확도를 높혀 본 사례

21 2_4_knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

```
Training KNN(skew correction & Hog) model using 3500 training data, which is 70.00% ...
1500: k:Acc => 1:97.6 2:97.5 3:97.9 4:97.8 5:97.9 6:97.8 7:98.0 8:97.9 9:98.0

Training KNN(skew correction & Hog) model using 4000 training data, which is 80.00% ...
1000: k:Acc => 1:97.3 2:97.6 3:97.5 4:97.4 5:97.4 6:97.5 7:97.5 8:97.5 9:97.7

Training KNN(skew correction & Hog) model using 4500 training data, which is 90.00% ...
500: k:Acc => 1:97.6 2:98.6 3:98.0 4:98.2 5:97.6 6:98.0 7:97.6 8:97.8 9:98.0
```

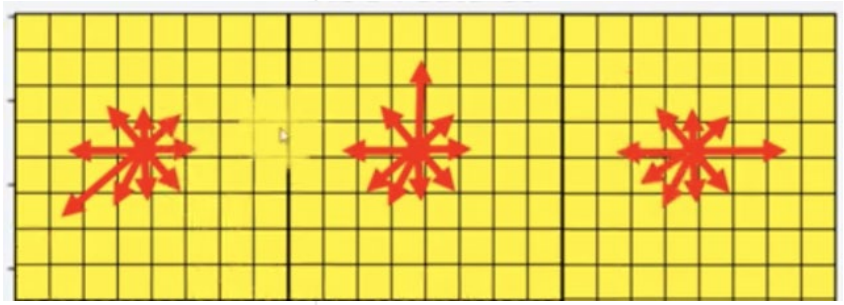


예제 2_4: 전처리(skew보정/hog descriptor)를 통해 정확도를 높혀 본 사례

A Valuable Introduction to the HOG Feature Descriptor

22 2_4_knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

HOG: Histogram of Oriented Gradients
영상을 타일(8x8)로 나누어 각 타일별로 (x, y) 방향의 기울기와 방향 정보를 구하고 이를히스토그램(영역별로 그래디언트가 나타난 수를 센다)으로 만들어 특징 기술자를 만든다. 객체의 모양과 구조에 관한 특징을 영상보다 차원이 축소하면서 수치 벡터화할 수 있다.



```
hog = cv2.HOGDescriptor((SIZE_IMAGE, SIZE_IMAGE),    # winSize,
                        (8, 8),      # blockSize,
                        (4, 4),      # blockStride,
                        (8, 8),      # cellSize,
                        9,           # nbins,
                        1,           # derivAperture,
                        -1,          # winSigma,
                        0,           # histogramNormType,
                        0.2,         # L2HysThreshold,
                        1,           # gammaCorrection,
                        64,          # nlevels,
                        True)        # signedGradient

hog descriptor size: '144'
```

검토(1)

CS231n Convolutional Neural Networks for Visual Recognition 강좌 중 knn부분 우리말 번역 강좌

- CIFAR-10 dataset 데이터 세트에 대해 k=1로 설정하여 시도한 코드가 있다.
  - 60,000개의 작은 이미지로 구성되어 있고, 각 이미지는 32x32 픽셀 크기이다. 각 이미지는 10개의 클래스중 하나로 라벨링되어 있다.
  - 이중 50,000개는 학습 데이터 세트로, 10,000개는 테스트 데이터 세트로 사용한 결과: k=1일 때 정확도는
    - L1거리38.6%, L2거리 35.4%: L2 거리는 L1 거리에 비해 두 벡터간의 차가 커지는 것에 대해 훨씬 더 크게 반응한다. 즉, L2 거리는 하나의 큰 차이가 있는 것보다 여러 개의 적당한 차이가 생기는 것을 선호한다.
- knn을 영상 화소에 적용하는 것은 non-sense이지만, CNN 기술의 소개에 앞서 data driven approach의 일반적인 문제를 소개하기 위한 사례로 제시
  - 데이터를 학습용과 test용으로 구분, L1vs L2거리,
  - Hyperparameter를 선정하기 위한 Cross validation 기법 소개
  - Nearest neighbor의 장점 및 단점
- 검토
  - 여기서 모델 학습(generalization)은 시도하지 않았으며 모든 데이터를 다 저장하는 방식으로 시도하였다. => 그렇다면 학습 데이터는 100% 정확도를 가질 것이다. 반면 반면 OpenCV 함수는 generalization을 행하는 것으로 보인다. OpenCV 함수를 쓰면 어떻게 달라질까?

검토(2)

- Skew correction + HOG를 이용해 feature extraction한 데이터 세트를 사용하여 KNN이 비록 단순한 generalization (예: 유클리디안 거리)를 사용하더라도 성능이 개선될 수 있다.
- 주어진 5,000개의 비교적 소용량 데이터 세트에 대해서는 위의 전처리와 단순한 분류기 만으로도 상당한 수준의 인식률을 보였다.
- 미션
  - 1. 마지막 예제에 대해 2500개의 학습 데이터에 대해서는 과연 OpenCV의 knn이 100%의 정확도를 보이는지 확인하시오.
  - 2. 학습과정에 전혀 generalization을 행하지 않는 cs231n 강좌의 예제에서 제시한 train, predict 함수를 HOG 전처리 프로그램의 성능을 비교 검토하고자 한다.
    - 1) 전처리를 전혀 시행하지 않고 cs231n의 함수로 아래 예제의 실험을 수행하시오 .
      knn_handwritten_digits_recognition_k_training_testing.py
    - 2) Skew correction + HOG 전처리를 수행하고 cs231n의 함수로 아래 예제의 실험을 수행하시오 .
      knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

검토(3)

- CIFAR-10 dataset 데이터 세트를 모노로 변환 한 후 Skew correction + HOG를 이용해 feature extraction한 데이터 세트를 사용하여 OpenCV KNN 함수로 분류를 시행하여 성능을 검능하고자 한다.
- 미션:
  - cs231n 강좌의 예시에 나온 것 처럼 50,000개의 학습데이터, 10,000개의 테스트 데이터에 대해 최선의 인식률을 제시하시오.
  - 이때 강좌의 사례에 제시된 것과 같은 5 fold cross validation을 수행하여 가장 최적의 hyper parameter 선정하면 더 좋다.
  - 주어진 기술 이외의 추가 기술을 사용해서 개선하는 것은 공정한 평가를 위해 허용하지 않는다.
  - 단, Skew correction과 HOG의 적용 여부 및 파라미터 설정은 자유로 선택할 수 있다.

검토(4)

- 학습방법
  - 교과서에 “학습과정에서 벡터와 레이블을 모두 저장해 둔다. “는 어구가 있지만 OpenCV 내 자체 알고리즘은 generalization을 시행하는 것으로 추정된다.
  - 단순 저장은 아닌 것에는 틀림이 없다. (근거: 학습데이터의 수를 40만개 정도로 늘리니 학습 데이터에 대한 정확도가 50여 %로 떨어졌다.)
  - 그 많은 학습 데이터를 개별로 저장한다는 것도 무리해 보이고, 설사 저장했다해도 근처 값을 찾아내는데 걸리는 시간 소요가 많을 것이기 때문에 합리적이지도 않아 보인다.
  - K=1일때의 학습데이터의 정확도가 100%가 되지 못하는 것을 보면 어느 정도의 generalization을 하는 것은 틀림없다고 말해도 되지 않을까?
    - 그러나 digit.png의 5,000개 문자 데이터에 대한 분류 작업(예제 2_0)은 K=1이면 100%를 달성하였다.
- Modified knn
  - K개의 다수결로 결정하는 것이 아니라 거리를 1/distance의 가중치로 두어 분류하는 수정된 알고리즘도 있다.
- Testing 알고리즘
  - 오류가 발생하였을 때 관찰된 사실인데 brutal matching을 사용하는 것으로 추정된다.

검토(4)
예제 1_1: knn은 학습한 데이터는 모두 인식하는가?

1_1_checking_knn_model_accuracy.py

```
test=train: k=1, num of test data=40 -------
testing time: whole=0.00, unit=0.00
results: <class 'numpy.ndarray'> (40, 1)
neighbours: <class 'numpy.ndarray'> (40, 1)
dist: <class 'numpy.ndarray'> (40, 1)

test=train: L=40, k=1: Accuracy=100.00%


test=train: k=1, num of test data=400 ------
testing time: whole=0.00, unit=0.00
results: <class 'numpy.ndarray'> (400, 1)
neighbours: <class 'numpy.ndarray'> (400, 1)
dist: <class 'numpy.ndarray'> (400, 1)

test=train: L=400, k=1: Accuracy=98.50%


test=train: k=1, num of test data=4000 -------
testing time: whole=0.01, unit=0.00
results: <class 'numpy.ndarray'> (4000, 1)
neighbours: <class 'numpy.ndarray'> (4000, 1)
dist: <class 'numpy.ndarray'> (4000, 1)

test=train: L=4000, k=1: Accuracy=91.10%
```

* 본 페이지의 실험은 아직 충분히 검토한 것이 아니라 오류가 있을 수 있습니다...

```
test=train: k=1, num of test data=40000 -------
testing time: whole=1.28, unit=0.00
test=train: L=40000, k=1: Accuracy=62.24%


test=train: k=1, num of test data=400000 -------
testing time: whole=192.51, unit=0.00
test=train: L=400000, k=1: Accuracy=51.21%
```

100%인식이 이루어지지 않는 것으로 미루어 내부에서 generalization이 이루어지고 있음을 추정할 수 있다. -> digit.png 파일은 4,000 데이터를 모두 학습시키면 100%를 보였다.

2차 과제 공고(예비)

- 페이지 23에 나온 CIFAR-10 데이터 세트에 대하여 OpenCV 함수를 사용하여 정확도 성능 결과를 제시하시오.
  - CS231n 강좌 처럼 학습:50,000개, 테스트:10,000개를 적용하여 산출한 결과와 비교할 것.
  - 초점: KNN 함수 예측이 일반화를 배부에서 거치는가? 거친다면, 안거치게 할 수도 있는가? 이들간의 trade-off가 존재하는가?
- 페이지 25의 검토(3)
  - KNN 성능 향상을 적용하여 비교 검토하시오.
  - 시간이 많이 걸려 실험 시간이 부족하다면 데이터 세트를 ½, 혹은 ¼, 1/8로 줄여 사용할 수 있음.
    - 처음 시작할 때는 1/10 이하로 줄여서 검토할 것을 추천함.
- 예정 기한: 12월 22일(기말 고사 종료 시점)까지
- 세부 사항 변경이 있을 수 있으며, 정식 공지는 구글 클래스 룸에 올리고 SMS 문자로 다시 공지할 예정입니다..