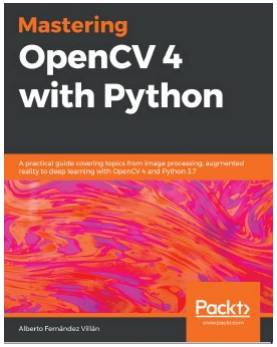
OpenCV Python

디지털영상처리/컴퓨터비전

k nearest neighbor



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

교재 10장의 일부..

Rev.4 2024년 1학기

서경대학교 김진헌

차례

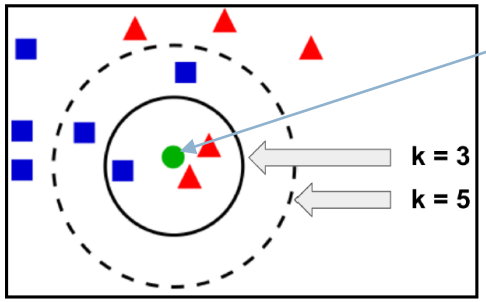
1

1. 기본 개념

- KNN은 지도 학습의 단순한 형태이다.
  - instance-based learning, or lazy learning
    - 학습데이터의 generalization을 보류. ⇔ eager learning
    - 학습과정은 사례(학습 데이터+레이블)를 table에 모아 놓는 것이라 할 수 있다.
  - 간단하면서도 강력한 비모수적(nonparametric) 기법이다
    - 데이터의 형태나 분포에 상대적으로 자유롭게 적용될 수 있으며, 모집단에 대한 가정이 필요하지 않아 매우 유연하다.
- 데이터가 모아지면(학습 완료), testing 과정에서는 입력 데이터와 가장 가까운 거리(Euclidean distance)를 갖는 학습 데이터의 레이블을 K개 추려 그 수가 많은 그룹의 레이블을 반환한다.
- 분류와 회귀모델에서 모두 사용가능하다.
  - 회귀모델에서는 근접한 여러 레이블의 출력값을 평균하여 출력한다

학습데이터를 쌓아두었다가 문의가 들어오면 판단
음반, 영화처럼 과거 학습데이터가 별로 없을 때 유효.

학습을 완료하여 일반화가 이루어지면 문의를 받는다. 예: Deep Learning



Query 데이터(녹색 점)

본 사례의 유클리디언 거리는 (x, y) 각 축 간의 차이를 제곱하여 더한 것에 root를 취한 것이다.

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

그림에서 빨간 삼각형과 파란 사각형은 미리 학습시켜 놓은 KNN 모델이 보유하고 있는 학습 데이터(위치와 레이블)이다.

1.1 KNN의 학습의 메모리 문제

- 단순히 데이터를 모아 놓는 것이라면 많은 메모리가 필요하다 → 메모리를 줄이기 위한 대책
  - 아래의 대다수 기법이 검색 고속화에도 도움을 준다.
- **차원 축소(Dimensionality Reduction)** ← feature extraction.
  - kNN은 차원이 매우 높은 데이터에 대해 공간 밀도가 늘어날 수 있습니다. 따라서 차원 축소 기법을 사용하여 데이터를 저차원 공간으로 변환하면 메모리 사용량을 줄일 수 있습니다.
- **희소 표현(Sparse Representation)**
  - 대규모 데이터셋의 경우 대부분의 데이터 포인트 사이의 거리가 멀 수 있습니다. 이 경우, 희소 표현을 사용하여 거리 행렬을 저장하여 메모리를 절약할 수 있습니다. 이는 희소 행렬 구조나 해싱 기법을 사용하여 구현될 수 있습니다.
- **분할된 kNN(Incremental kNN)**
  - 데이터를 여러 그룹으로 분할하여 kNN을 적용하는 방법입니다. 각 그룹은 메모리에 따로 저장되고, 새로운 데이터가 주어질 때마다 적합한 그룹을 선택하여 예측을 수행합니다.
- **근사 kNN(Approximate kNN)**
  - 근사 kNN 기법은 메모리 사용량을 줄이기 위해 데이터를 압축하거나 요약하여 사용합니다. 이를 통해 거리 계산을 위해 저장되는 데이터의 양을 줄일 수 있습니다.

1.2 KNN 테스팅의 탐색의 문제

- 일일이 많은 학습 데이터와 query input과의 유사도를 모두 점검하는데 시간이 많이 소요된다.
  → 탐색 법에 대한 대책:
- 거리 행렬의 계산 최적화
  - 거리 행렬은 학습 데이터 포인트 간의 모든 거리를 포함하고 있습니다. 이를 효율적으로 계산하기 위해 병렬 처리 및 행렬 연산을 사용하여 거리 행렬의 계산을 최적화하는 연구가 있었습니다.
- 근사적인 이웃 탐색
  - 모든 학습 데이터 포인트와의 거리를 계산하는 것이 아니라, 근사적인 이웃 탐색 기법을 사용하여 거리 계산을 줄입니다. 이러한 기법에는 KD 트리(K-dimensional tree), Ball tree, Cover tree 등이 있습니다.
- 차원 축소 기법
  - 차원 축소 기법을 사용하여 데이터의 차원을 줄이고, 이를 통해 거리 계산의 복잡성을 낮출 수 있습니다. PCA(Principal Component Analysis), LSH(Locality-Sensitive Hashing) 등의 기법이 사용될 수 있습니다.
- 분할 및 병렬화
  - 대규모 데이터셋을 여러 부분으로 분할하고, 병렬 처리를 통해 각 부분을 동시에 처리하여 계산 속도를 향상시키는 방법이 있습니다.

1.3 KNN의 원리적 한계

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

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

- 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

같으면 1, 반대면 -1, 관련이 없으면 0

  - → 절댓값을 취한 단순한 거리 보다는 - 값도 존재하는 유사도를 사용한다는 뜻.
    → 유클리디언 거리(Euclidean distance) 이상의 유사도 측정 방안이 필요. ← 이 자체가 KNN의 "손쉬움"의 장점을 버리게 만드는 아이디어.
- 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.

1.4 OPenCV KNN의 최적화 기법 옵션

- 주로 cv2.ml.KNearest_create() 함수를 호출할 때 매개변수로 전달된다.
- Algorithm Type(알고리즘 유형)
  - kNN 구현에서 사용할 알고리즘을 지정하는 옵션입니다. OpenCV의 KNearest_create() 함수에는 algorithmType 매개변수가 있으며, 이를 통해 cv2.ml.KNEAREST_BRUTE_FORCE (기본값) 또는 cv2.ml.KNEAREST_KDTREE와 같은 다른 알고리즘을 선택할 수 있습니다.
- Parallelization(병렬화)
  - 거리 계산을 병렬화하여 계산 속도를 향상시킬 수 있는 옵션입니다. OpenCV의 kNN 구현에서는 이러한 옵션을 지원합니다.
- Data Storage Format(데이터 저장 형식)
  - 학습 데이터와 테스트 데이터의 저장 형식을 지정하는 옵션입니다. OpenCV의 kNN 구현에서는 데이터를 효율적으로 저장하기 위해 다양한 저장 형식을 지원합니다.
- Distance Metric(거리 측정 방법)
  - 거리 측정에 사용할 메트릭을 선택하는 옵션입니다. 일반적으로 유클리드 거리가 사용되지만, 다른 거리 측정 방법을 선택할 수도 있습니다.

2. OpenCV KNN 함수

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

opencv machine learning tutorials

- **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

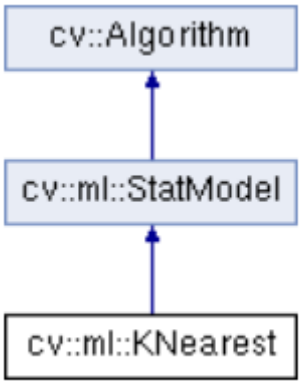
2.1 KNN Class

The class implements K-Nearest Neighbors model.

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

Inheritance diagram for cv::ml::KNearest:

knn 함수의 상속 다이어그램

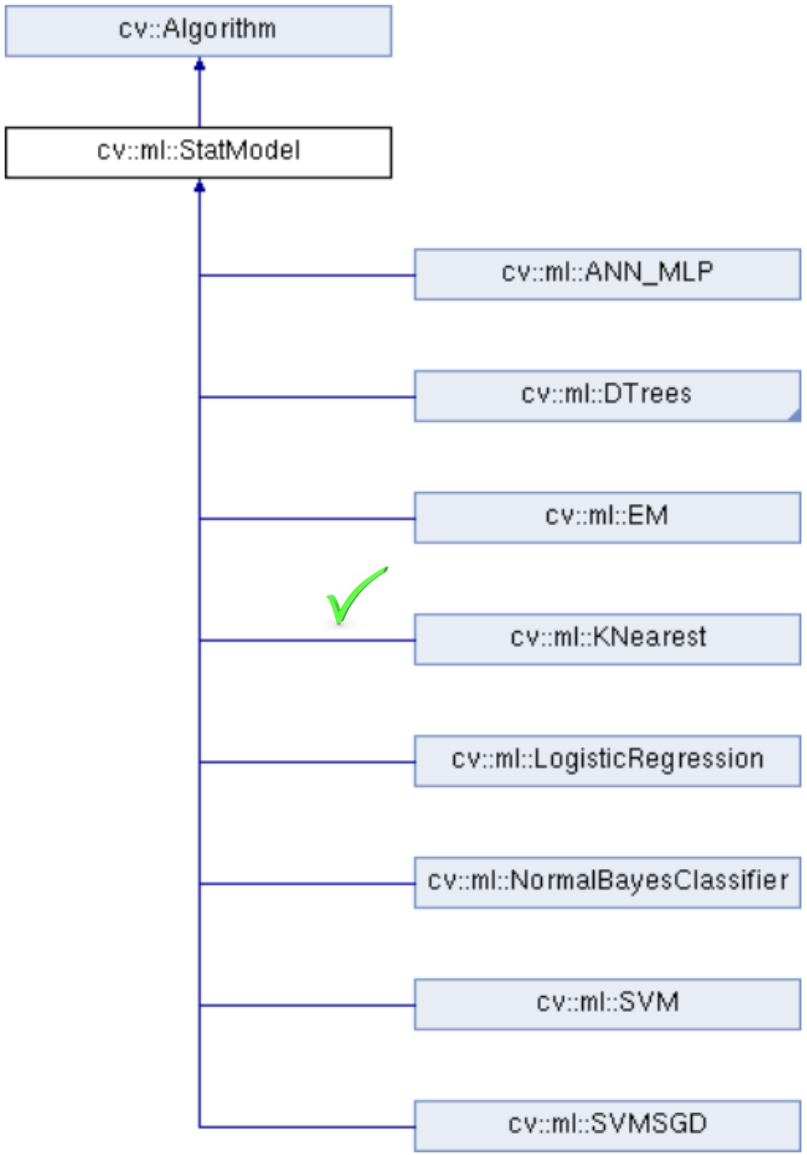


공통함수
train(), predict()

전용함수: findNearest

cv::ml::KNearest Class Reference abstract

Machine Learning



2.2 knn 멤버 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.

2.3 training과 testing 요약

* KNN을 기본으로 살펴본 …
ml 모듈의 training과 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)
    - K값 지정 가능. 반환 값에 추가정보 많음
  - retval, results=cv.ml_StatModel.predict(samples[, results[, flags]])

2.4 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 경우도 있는 것을 보면~~ 자체 generalization 알고리즘이 있는 것으로 추정됨.
  - 학습된 데이터는 모두 맞추어야 정상임… 대용량 학습데이터의 실험을 통해 입증가능.

3. 간단한 활용 실험(1_0)

- 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 L=16개의 (x,y) 좌표 *data*를 임의로 2개의 그룹으로 나누어 학습 데이터를 생성한다.
  - 랜덤하게 배정된 *labels*를 이용하여 학습 데이터를 생성하였다.

```
number of Learning data=16
Training data set: data.shape=(16, 2), labels.shape=(16, 1)
```

  - 2) test data, *sample*을 1개 랜덤하게 생성한다. 이 데이터가 어느 그룹에 속하는지를 k-nn 방법으로 분류할 예정이다.

```
Testing data set: sample.shape=(1, 2)
```

  - 3) knn 학습 모델을 구축한다.
    - **knn** = cv2.ml.KNearest_create()
    - knn.train(*data*, cv2.ml.ROW_SAMPLE, *labels*)

- 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]

```
For K=3: return values of findNearest(sample, k)
results.shape=(1, 1), neighbours.shape=(1, 3), dist.shape=(1, 3)
```

- 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')
```

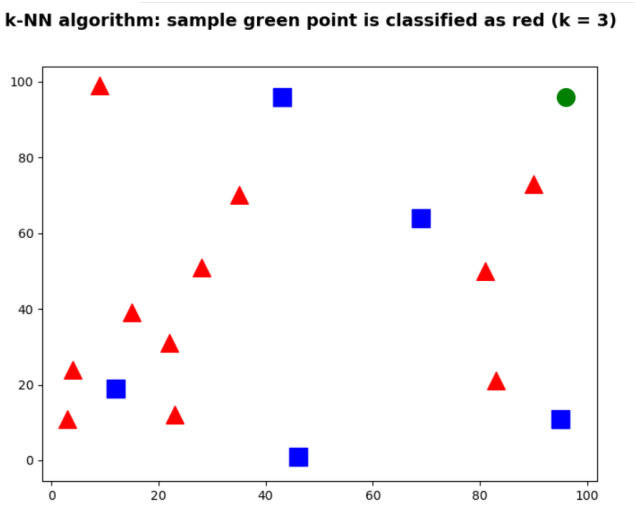
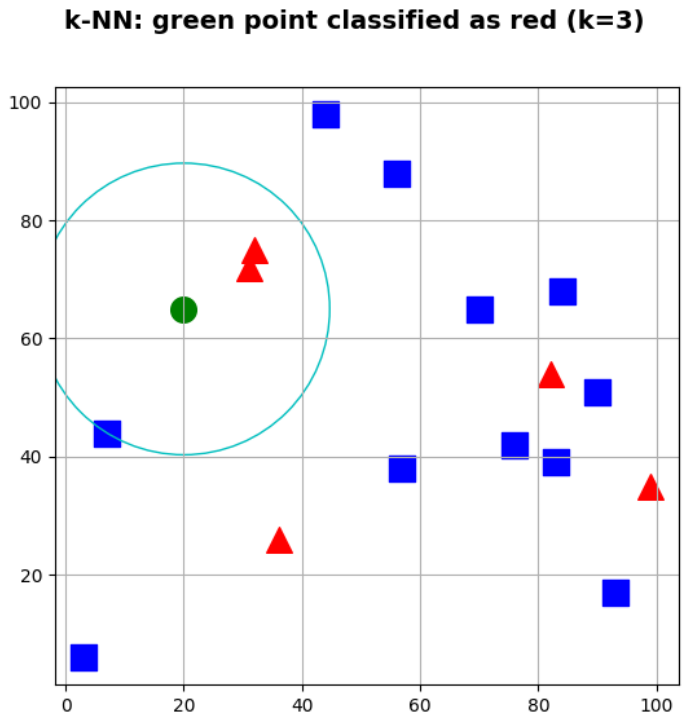


그림 해석: 녹색원은 타이틀에 적힌 대로 k=3일 때 red triangle 그룹에 속하는 것으로 판단되었다.

4. 샘플 이웃의 K개를 둘러싸는 원 그리기 실험(1_0)

1_0_knn_introduction2_circle.py



num of train data=16, num of test data=1, k=3
result: [[0.]], shape=(1, 1)
neighbours: [[0. 0. 1.]], shape=(1, 3)
distance: [[170. 244. 610.]], shape=(1, 3)
sample location=[[20 65]]
dist_max=24.7 ← sqrt(610)

- sample 데이터 주변의 k개를 둘러싸는 원을 그려보자.

```
# img가 없으므로 cv2.circle() 함수는 쓸 수가 없다.
import matplotlib.patches as patches
# 반지름=제일 먼 데이터의 거리. k-1번째.거리가 L2-norm이라 root처리가 필요.
dist_max = np.sqrt(dist[0, k-1])
sample = np.int32(sample)   # 샘플 데이터의 좌표를 정수형으로 바꾼다.
print(f"sample location={sample}")
print(f"dist_max={dist_max:#.1f} ")
shp=patches.Circle( xy: (sample[0, 0], sample[0, 1]), radius=dist_max,
                    color='c', fill=False)     # True이면 채움
plt.axis('scaled')      # x, y 비율을 맞추면 타원으로 보이는 것이 해결됨.
plt.gca().add_patch(shp)
plt.grid("on")
```

5. 랜덤 데이터/레이블의 분류(1_2)

1_2_knn_introduction_varying_k.py

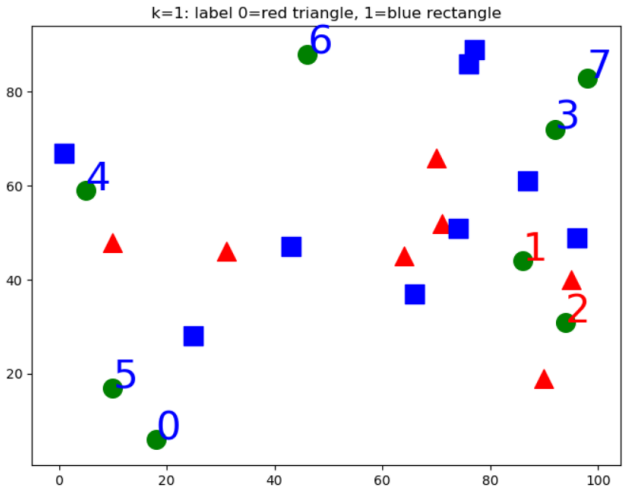
- 개요:
  - 본 프로그램은 k-Nearest Neighbour의 활용 기법을 연마하기기 위해 학습 모델이 있다고 가정하고, k-nn 알고리즘이 k에 따라 어떻게 다른 결과를 나타낼 수 있는지 보이고자 한다.
  - 본 예제의 상황은 데이터의 레이블이 랜덤으로 설정되었기 때문에 근접한 데이터라고 해도 사실 전혀 상관이 없다.
  - 이 때문에 k를 크게 하면 할 수록 받아들이기 어려운 결과를 낼 수 밖에 없다.
- 절차:
  - 단계 1: 학습 데이터를 만든다. 랜덤한 값(x, y)으로 데이터를 L개 생성한다.
    - 이들을 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.

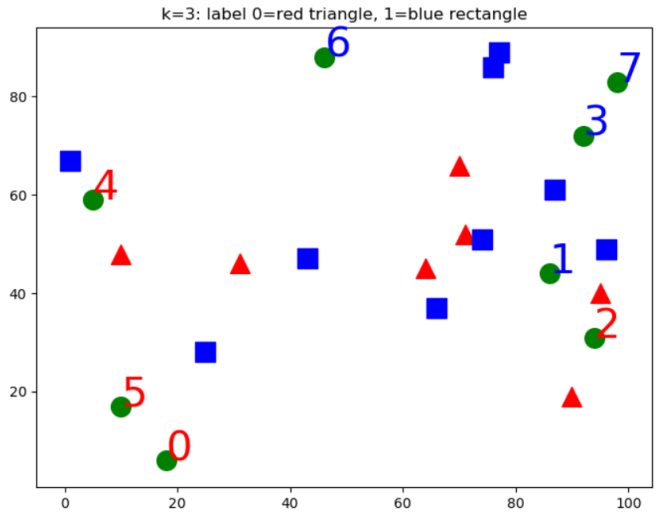
1_2_knn_introduction_varying_k.py

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

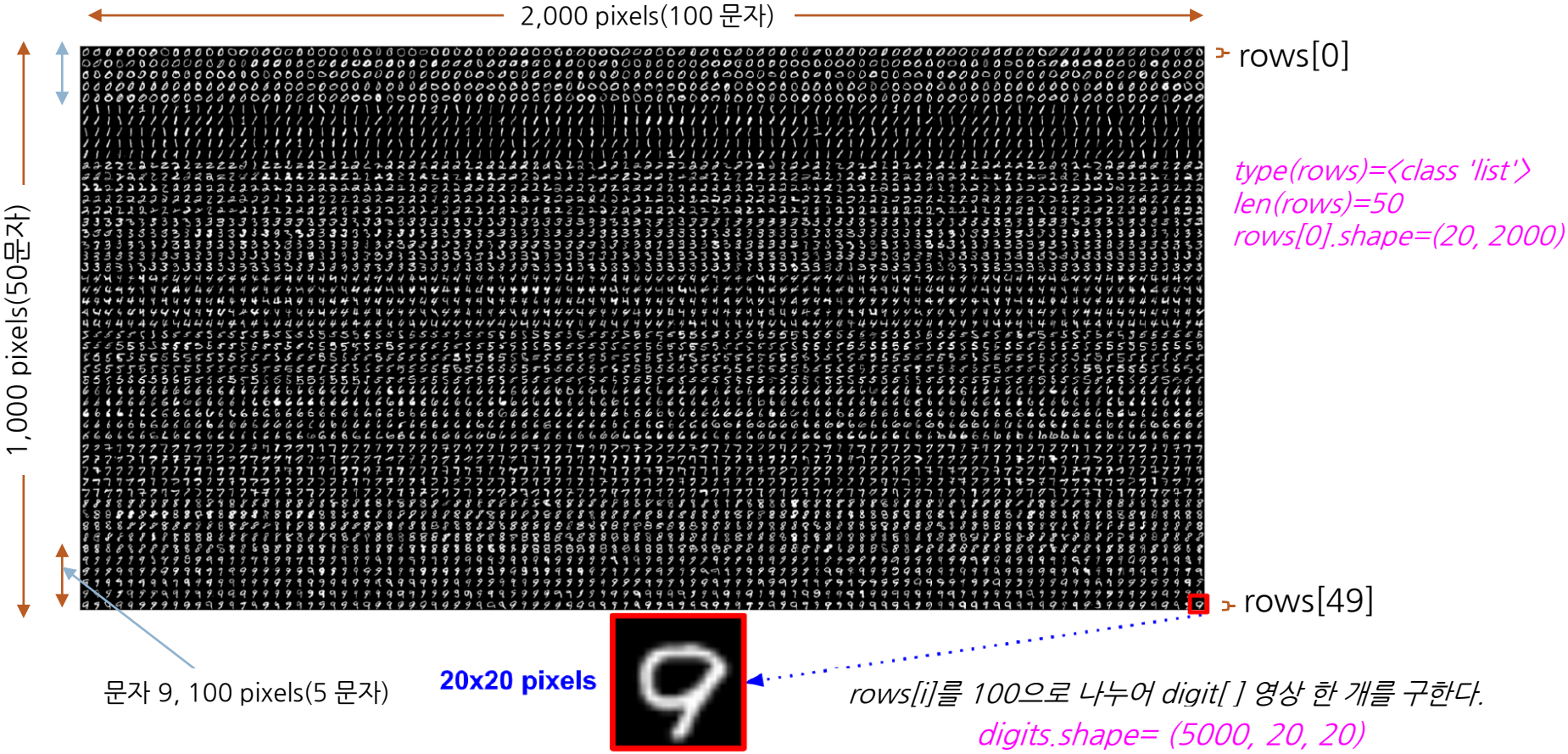
총 16개의 학습데이터를 랜덤하게 0, 1의 레이블을 붙여 빨간 삼각형(0), 파란 사각형(1)으로 표기하였다.

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.

6. 필기체 문자인식(2_0)

17 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.

프로그램의 흐름

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)
- 7. k=[1, 3, 5] 등과 같이 k의 변화에 따른 정확도를 관찰한다. 학습 데이터와 테스트 데이터에 적용다.
  - 학습된 데이터를 테스팅하니 K=1이면 100% 정확도를 보였다.

데이터 타입의 관찰


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

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

(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,)

@@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)
```

영상 해상도: 가로x세로=2,000x1,000

글자의 수: 가로x세로=100x50

20x20의 폰트를 가진 글자가 총 5,000개

5,000개의 라벨을 생성한다.

5,000개의 데이터를 생성한다. 데이터의 차원이 400인 셈이다.

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 model with k=3.
1) type(ret)= <class 'float'> ret= 4.0
2) result.shape= (2500, 1) | result[0, 0]= 4.0
3) neighbours.shape= (2500, 3) | neighbours[0]= [4. 9. 4.]
4) dist.shape= (2500, 3) | dist[0]= [609174. 828973. 927925.]
5) Sampling test_result vs labels_test: 0 to 20
result: 4 4 8 8 7 2 9 1 0 7 5 5 0 6 6 3 2 6 9 8
 label: 4 4 8 8 7 2 8 1 0 7 5 5 0 0 6 3 2 6 9 8
6) Accuracy(k=3) for test data:  93.36

@@7. Accuracy for test and train data with varying k
k=1: Accuracy for test data:  93.56  k=1: Accuracy for train data: 100.00
k=3: Accuracy for test data:  93.36  k=3: Accuracy for train data:  96.76
k=5: Accuracy for test data:  93.12  k=5: Accuracy for train data:  95.44
```

2,500개의 테스트에 대한 분류 결과
그 중 첫번째 결과는 문자 4이다.

가까운 3개의 판별 결과가 각각 4, 9, 4임을 알려주고 있다. 4가 2개이므로 최종 결과는 4임을 result를 통해 반환하였다.

3개의 결과 4, 9, 4에 대한 거리를 나타낸다.

일부 결과를 출력하였다. 예측 값이 틀린 부분이 2개 발견된다.

현 실험 결과로는 k=1일 때가 미지의 데이터에 대해 가장 정확도가 높다.
신뢰성 있는 결론을 도출하기 위해서는 cross validation 절차가 더 필요하다.
cross validation : 데이터 군집을 다양하게 자르고 k를 다양하게 바꾸어 가면서 정확도를 비교하는 절차

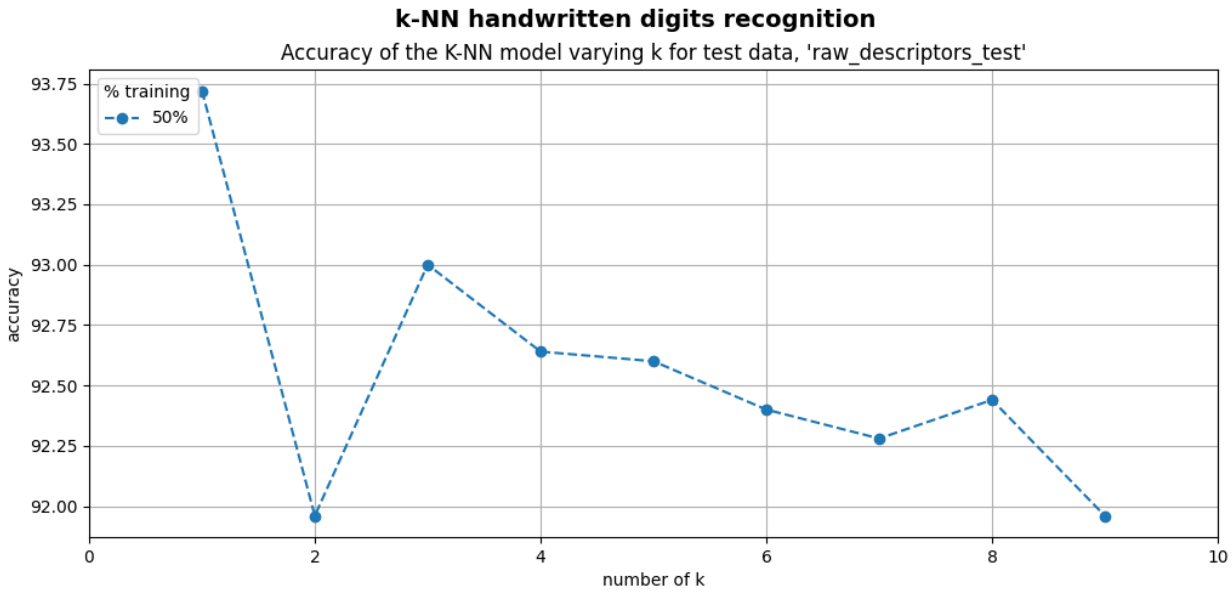
6.1 K의 증가에 따른 정확도의 변화(2_1)

21 2_1_knn_handwritten_digits_recognition_varying_k.py

```
1) Preparing data to train and test.....
raw_descriptors_train.shape=(2500, 400)
raw_descriptors_test.shape=(2500, 400)
labels_train.shape=(2500,)
labels_test.shape=(2500,)
2) Training KNN model - raw pixels as features
```

```
4) Repeating 'knn.findNearest' for varying k....
5) Show all results using matplotlib capabilities...
len(results)=1: type(key)=<class 'str'>, key=50
results[key]=[93.72, 91.96, 93.0, 92.64, 92.60000000000001,
results[key]=
[ 93.72 91.96 93.00 92.64 92.60 92.40 92.28 92.44 91.96 ]
```

```
3) Creating a dictionary, 'results' to store the multiple accuracies when testing
type(results)=<class 'collections.defaultdict'>
type(results['50'])=<class 'list'>, results['50']=[]
```



- 바로 이전 프로그램의 7단계에서 소개한 k의 변화에 따른 test 데이터에 대한 accuracy의 변화를 그림으로 출력하였다.

- 단계 3에서 accuracy(변수명: result)의 값들을 단순히 list에 저장한 것이 아니라, key='50'의 사전형 자료를 collection.defaultdic 모듈의 함수에서 빌려와 10개의 result를 '*results = defaultdict(list)*' 로 선언한 점이 챙겨볼 만한 대목이다.

- defaultdic()으로 선언한 사전은 사전에 없는 key로 액세스해도 3단계 '*results['50']*'은 empty list([])를 반환하고 오류를 발생하지 않는다.
- 일반 사전형 자료는 없는 key를 인덱스로 액세스하면 오류가 발생한다.

6.2 K와 학습 데이터의 비율의 증가에 따른 정확도(2_2)

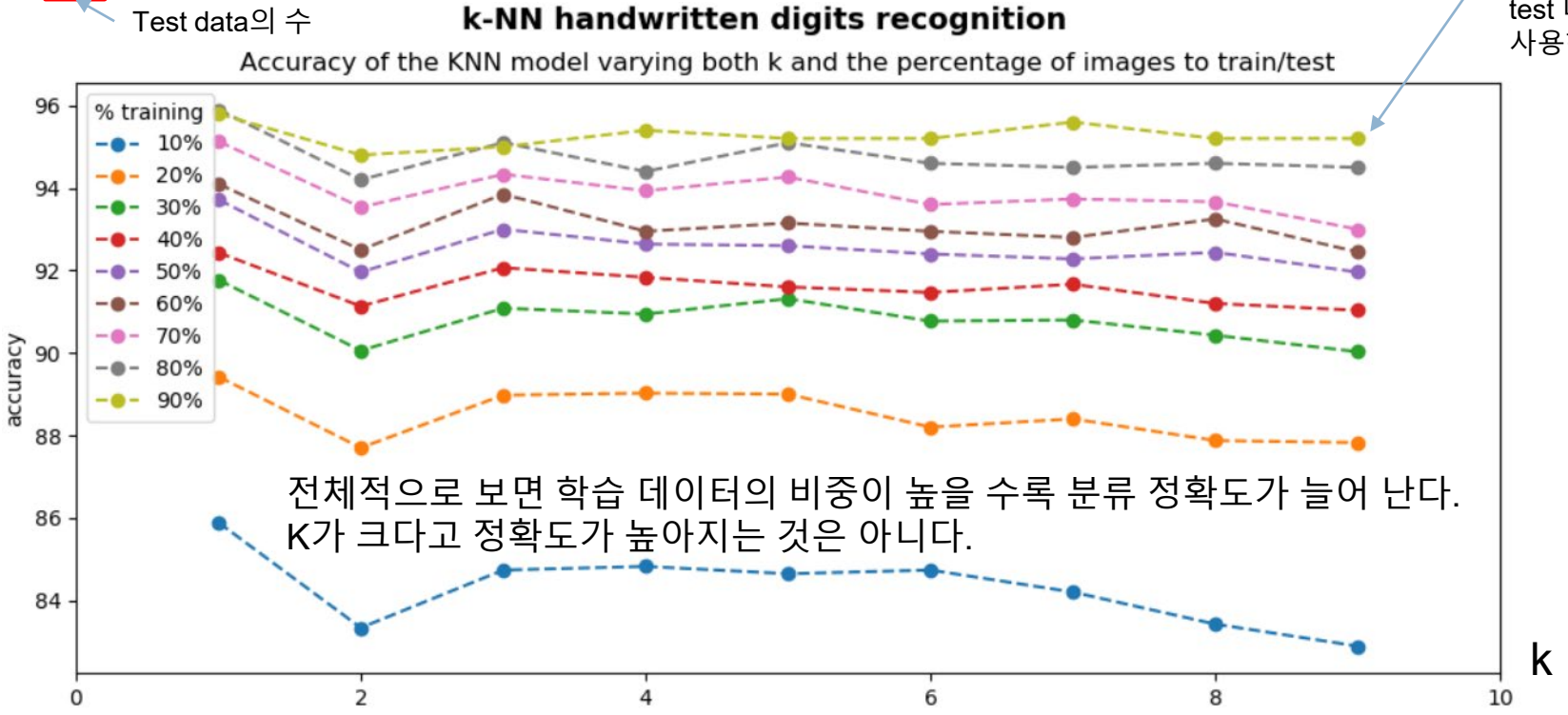
22 2_2: knn_handwritten_digits_recognition_k_training_testing.py

```
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

5,000의 데이터 중 90%인 4,500를 학습데이터로 사용. 10%인 500개를 test 데이터로 사용하였음.

Test data의 수

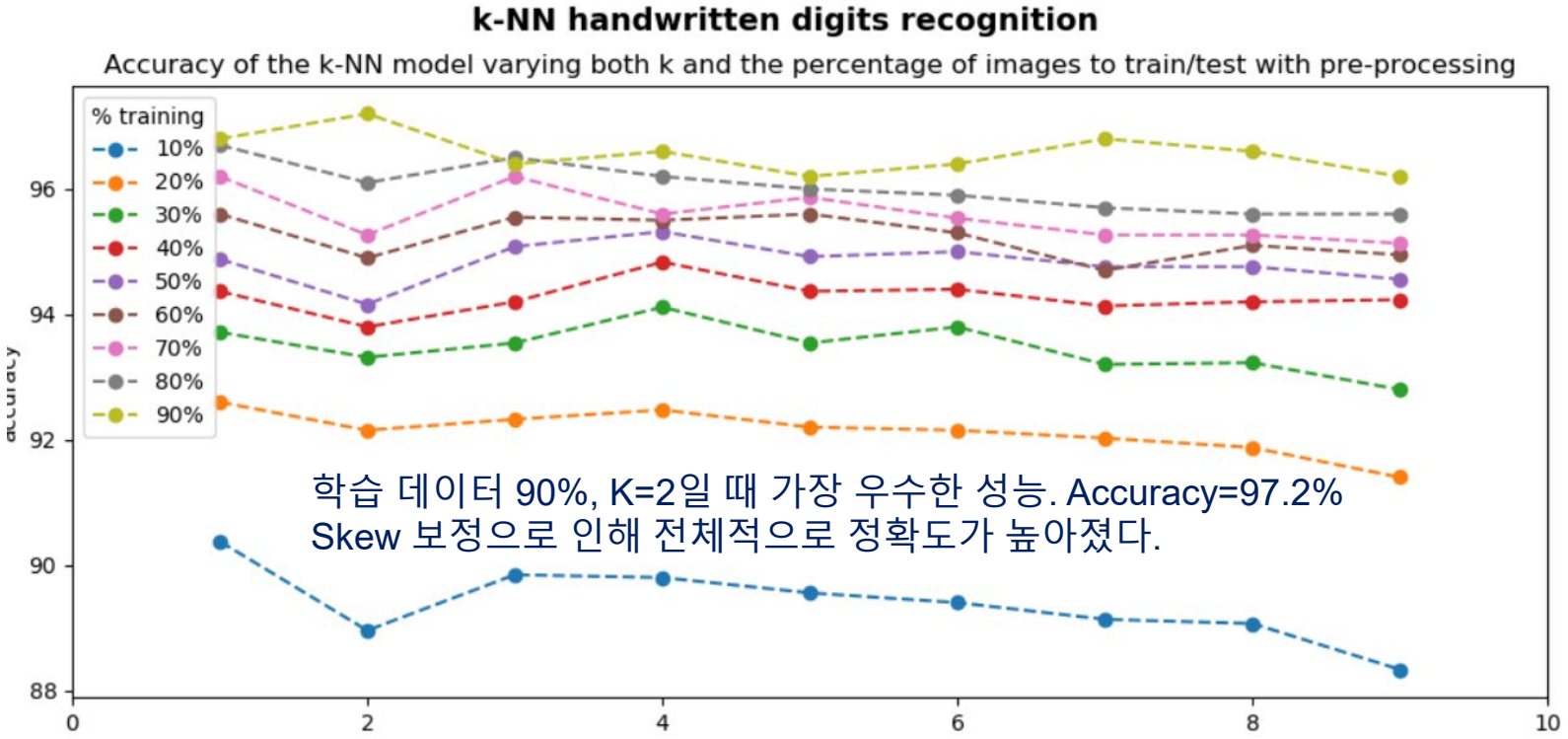


전체적으로 보면 학습 데이터의 비중이 높을 수록 분류 정확도가 늘어 난다.
K가 크다고 정확도가 높아지는 것은 아니다.

6.3 전처리(skew 보정)를 통해 정확도를 높혀 본 사례(2_3)

23 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
```



Skew 보정

원본 영상

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

```
def deskew(img):
    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, dsize: (SIZE_IMAGE, SIZE_IMAGE),
                         flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
    return img
```

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

```
SIZE_IMAGE = 20
```

Shear-X

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \lambda_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

M은 2x3 매트릭스로서 x축을 따라 shear-X 변형을 주면서 x축으로 translation을 반영한 것이다.

The central moments **Moments::mu**$_{ji}$ are computed as:

Translation에 견실

$$\mathrm{mu}_{ji} = \sum_{x,y} \left(\mathrm{array}(x,y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i \right)$$

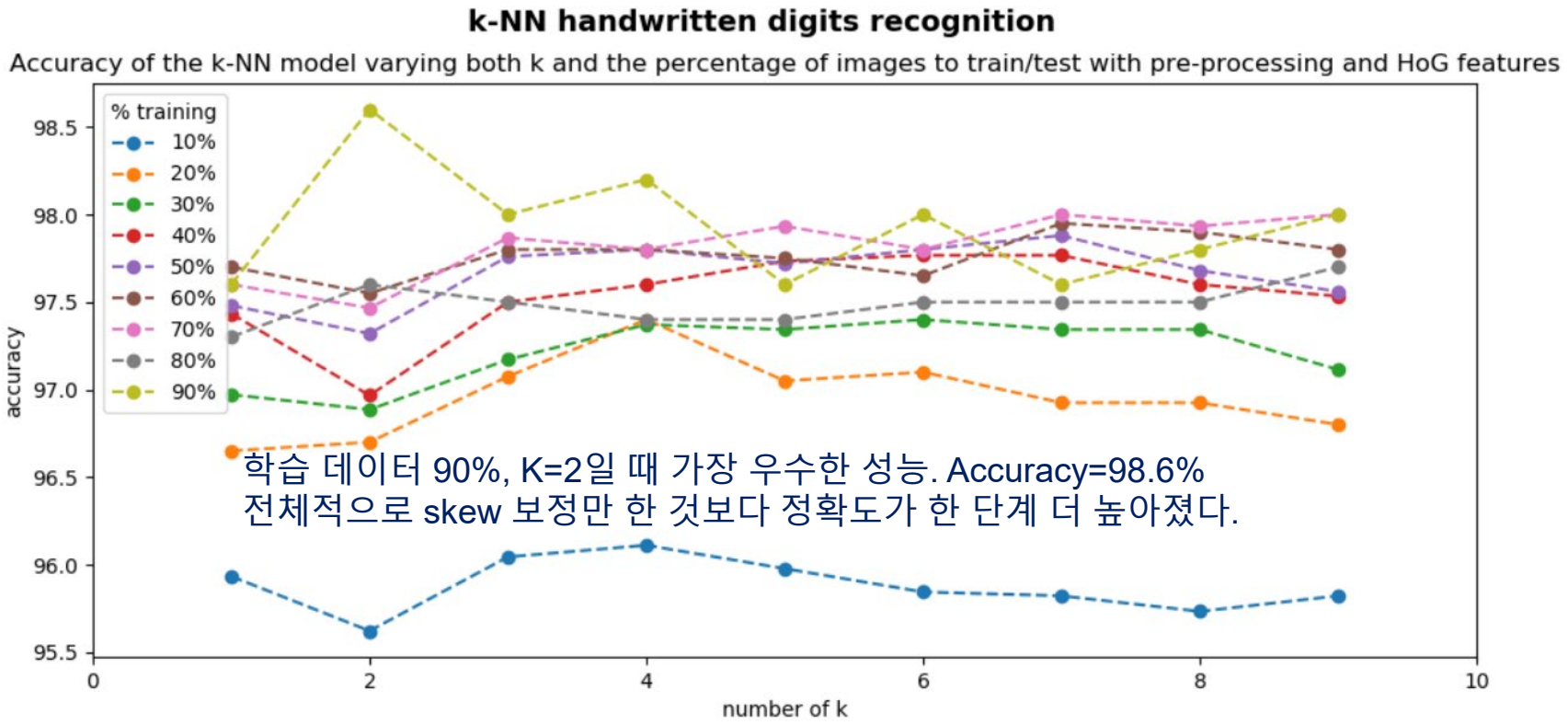
where $(\bar{x}, \bar{y})$ is the mass center: $\bar{x} = \frac{\mathrm{m}_{10}}{\mathrm{m}_{00}}, \; \bar{y} = \frac{\mathrm{m}_{01}}{\mathrm{m}_{00}}$

미션: skew 보정을 실행하는 별도의 프로그램을 제작하여 보자. 위의 그림처럼 출력하면서 매트릭스 파라미터를 바꾸어 보면서 skew 보정의 원리를 이해해 보자.

6.4 전처리(skew보정/HOG descriptor)를 통해 정확도를 높혀 본 사례(2_4)

25 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
```



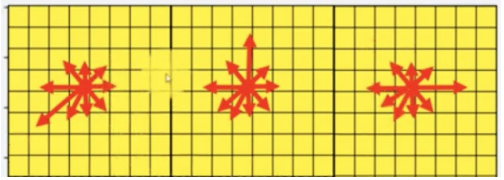
HOG descriptor

A Valuable Introduction to the HOG Feature Descriptor

2_4_knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

```
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: Histogram of Oriented Gradients
영상을 타일(8x8)로 나누어 각 타일별로 (x, y) 방향의 기울기와 방향 정보를 구하고 이를 히스토그램(분포 어레이)으로 만들어 특징 기술자를 만든다. 객체의 모양과 구조에 관한 특징을 원본 영상보다 차원을 축소시키면서 수치 벡터화할 수 있다.



20x20 차원 → 144 차원의 feature vector로 줄어듦
교훈: 분류기의 성능 향상을 위해서는 영상의 특징을 보존할 수 있으면서 차원을 줄여 분류기에 입력하는 것이 중요

HOG 디스크립터 크기 확인 법 2개

```
print("hog descriptor size: '{}'".format(hog.getDescriptorSize()))
hog descriptor size: '144'
```

```
hog_ret = hog.compute(deskew(img))
print(f"type(hog_ret)={type(hog_ret)}, hog_ret.shape={hog_ret.shape}"); exit()
type(hog_ret)=<class 'numpy.ndarray'>, hog_ret.shape=(144,)
```

7. 문제

- 문제(1)
  - cs231n CNN 강좌 중 knn부분: opencv가 학습 데이터에 대해 100% 인식하는지의 여부 판단
- 문제(2)
  - 전처리 활용의 유용성 타진
- 문제(3, 4)
  - 모노 영상 사용 및 최선의 결과를 산출하는 분류기 설계

문제(1) CS231n Convolutional Neural Networks for Visual Recognition 강좌 중 knn부분 우리말 번역 강좌

- 배경설명
  - 세트 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
    - 영상의 화소값들의 차이의 절댓값 혹은 제곱승을 모두 더한 L1거리, L2 거리로 근사로 판단하는 단순한 판단 알고리즘을 사용하였다. 실험에 의하면 L1 거리는 38.6%, L2거리 35.4%의 성능을 보였다; 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 제시한 것으로 이해하도록 하자.
    - 데이터를 학습용과 test용으로 구분, L1vs L2 거리,
    - Hyperparameter를 선정하기 위한 Cross validation 기법 소개
    - Nearest neighbo의 장점 및 단점
- **문제 (1)**
  - 여기서 모델 학습(generalization)은 시도하지 않았으며 모든 데이터를 다 저장하는 방식으로 시도하였다. => 그렇다면 학습 데이터는 100% 정확도를 가지는지 확인해 보자. 반면 OpenCV 함수는 generalization을 행하는 것으로 보인다. OpenCV 함수를 쓰면 어떻게 달라질지 직접 해보기 전에 예상으로 답해 보시오.

문제(2) CS231n Convolutional Neural Networks for Visual Recognition 강좌 중 knn부분 우리말 번역 강좌

- KNN의 성능 개선을 위한 노력의 영향 검토
  - Skew correction + HOG를 이용해 feature extraction한 데이터 세트를 사용하여 KNN 이 비록 단순한 generalization(예: 유클리디안 거리)를 사용하더라도 성능이 개선될 수 있다.
  - digit.png에 주어진 5,000개의 비교적 소용량 데이터 세트에 대해서는 위의 전처리와 단순한 분류기 만으로도 상당한 수준의 인식률을 보였다.
- **문제 (2) - 전처리 활용의 유용성 타진**
  - 1. 마지막 예제에 대해 2,500개의 학습 데이터에 대해서는 과연 OpenCV의 knn이 100% 의 정확도를 달성했는지 그래프를 통해 확인하시오.
  - 2. 학습과정에 전혀 generalization을 행하지 않는 cs231n 강좌의 예제에서 제시한 train, predict 함수를 HOG 전처리 프로그램를 시행하였을 경우에 어떻게 달라지는지 비교해 보고자 한다. - 강좌에서는 3채널 컬러 영상을 사용하였다. 비교하려면 역시 3채널 영상으로 해야 할 것이다.
    - 1) 전처리를 전혀 시행하지 않고 cs231n의 함수로 아래 예제의 실험을 수행하시오 . knn_handwritten_digits_recognition_k_training_testing.py(픽셀 단위의 비교)
    - 2) Skew correction + HOG 전처리를 수행하고 cs231n의 함수로 아래 예제의 실험을 수행하시오 . knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

문제(3,4) CS231n Convolutional Neural Networks for Visual Recognition 강좌 중 knn부분 우리말 번역 강좌

- **문제 (3) - 모노로 데이터를 처리하였을 경우 성능에 미치는 영향 검토**
  - 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.
  - 전 페이지의 아래 문제를 모노로 처리하여 성능의 개선 효과를 검토하시오.
    - 2) CIFAR-10 데이터를 모노변환하여 Skew correction + HOG 전처리를 수행한다. 이후 cs231n의 함수와 OpenCV 함수로 처리한 결과를 비교하시오 . Knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py
- **문제 (4) - 최선의 결과를 산출하는 분류기 설계**
  - cs231n 강좌의 예시에 나온 것 처럼 50,000 개의 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.

8. 학습데이터의 분류 성능

knn은 학습한 데이터는 모두 인식하는가?

- 현 단계에서는 추정
  - 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.
    - 이 같은 특성은 Deep Learning의 특징과는 다른 점이다. DL는 네트워크 규모를 키우면 학습 데이터는 100%에 가까운 분류 성능을 보인다.
    - knn은 사실 일반화만 하지 않으면 원리적으로 테이블 방식이기 때문에 100%이다. Knn 함수에 일반화를 약하게 하는 기법이 없는지 확인이 필요해 보인다.
  - 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.

8.1 추정 성능을 실험으로 알아보자

knn은 학습한 데이터는 모두 인식하는가?

32 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.

8.2 실험한 내용에 대한 고찰

knn은 학습한 데이터는 모두 인식하는가?

1_1_checking_knn_model_accuracy.py

- 8.1에서 오류가 있을 수 있다고 전제를 하였지만 아무튼 실험 결과로 보면 실험 데이터가 커지면 학습 결과도 제대로 못 맞추는 것으로 보인다.
- 그런데 digit.png은 학습 데이터가 제법 큰데도 100% 잘 맞춘다.
- 모순이 있어 보이는데 여러분의 생각은 어떤지?
  - 실험 방법이 잘 못되지 않았나?
  - 단순히 코딩이 잘 못되지 않았나?
  - 아니면 적당한 선에서 generalization이 일어나기 때문에 그 이전 규모의 학습 데이터를 쓰는 것이 맞는 것일까?
- 문서에 의해 해답을 찾을 것인가 vs 실험을 통해 그 해답을 찾을 것인가? 여러분들이 그 답을 찾으려 조원들간에 토론을 해보고 필요하다면 실험도 해서 그 해답을 제시해 보기 바랍니다.

8.3 get_accuracy() 함수의 활용

1_1_checking_knn_model_accuracy.py

```
def get_accuracy(predictions, labels):
    """Returns the accuracy based on the coincidences between predictions and labels"""

    accuracy = (np.squeeze(predicti
    return accuracy * 100

# 보류: 다른 방법으로 정확도를 계산해보려 했는데 안됨. 현재 어디가 오류인지 모르겠음..
# 다른 예제에서 사용하였던 함수로 accuracy를 구해본다.
acc = get_accuracy(results, labels)
print(f"k={k}: Accuracy2={acc:#6.2f}")
```

2_1_knn_handwritten_digits_recognition_varying_k.py 에서 정의하여 사용되었던 것임.

- 다른 예제에서 사용했던 get_accuracy() 함수가 여기서는 계속 50% 대의 정확도로 고정되어 출력된다.
- 어떤 오류가 숨겨져 있는지 원인을 파악하여 해결 해 보자.

8.4 knn 모델의 학습 데이터 저장

35 1_1_checking_knn_model_accuracy.py

knn 모델은 학습한 데이터를 모두 저장하는가?
모델의 크기가 학습 데이터의 양에 따라 달라지는가?

```
검토 1 - knn 모델의 크기는? 학습 모델을 저장할 수는 없나?
    현재 상황: scikit-learn에는 iris의 모든 데이터 세트가 저장된 세트를 load 할 수 있다.
        http://blog.naver.com/PostView.nhn?blogId=owl6615&logNo=221457206097&parentCategoryNo=1
        그러나 이는 아직 모델을 저장하는 기능이 있다는 것을 뜻하는 것은 아니다.
    본 코딩에는 sys.getsizeof() 함수로 knn 모델의 크기를 학습 전과 학습 후로 나누어 판단해 보려 하였으나
    둘 다 같은 크기로 결과가 나와 모델 객체가 지정하는 학습 데이터는 포함하지 않는 것으로 판명되었다.

    knn 객체를 pickle() 함수로 통째로 파일로 저장하여 간접적으로 크기 변화를 관찰하고자 하였으나,
    pickle() 함수가 knn 객체의 저장은 실행히 못하고 오류로 중단되었다.
    현재 이 부분은 주석문으로 처리하였다.
```

- 의도: KNN 모델을 저장해 보려는 시도는 학습 데이터가 적을 때와 많을 때 그 파일 크기가 달라지는지 확인해보고자 하는 것이다. 즉, 학습 데이터는 어떤 경우든지 100% 맞춘다고, lazy learning을 채택한 KNN이라면 일단은 모델안에 학습 데이터를 저장하고 있지 않은가 하는 궁금증을 갖지 않을 수 없다.
- 단순히 2차원 평면의 수백만개의 학습 데이터 뿐만 아니라 차원이 수천 차원으로 늘어나면 메모리 용량이 상당할 터인데 OpenCV는 이러한 데이터를 어떻게 관리하는지 파악하는데 도움이 되지 않을까 싶다.
- 현재는 저장하는 것 조차 실패하였는데 위 의도에 맞는 시도라면 pickle 이외의 다른 시도도 가능하다.

9. 검토: k=1vs k=2에 관한 chatGPT 질의

- 질문 요지: k-nearest neighbour 알고리즘에서 다른 모든 조건이 같음에도 k=1로 선택한 것과 k=2로 선택한 것이 분류 성능이 다른 것이 발견되는데 이것을 설명해 줄 수 있어?

- 여러 차례 질의/응답을 교환 끝에 …

- 재 질문: 두 이웃의 거리가 거의 같은 수준으로 멀리 떨어져 있다면 k=2가 k=1로 선택한 것과 다른 레이블을 선택할 수 있다고 했는데 두 경우에 거리 말고 다른 조건이 선택에 영향을 주나?

9.1 chatGPT의 응답

- 예, 거리 외에도 다른 조건이 k=1과 k=2의 선택에 영향을 줄 수 있습니다. 주요한 다른 요소 중 하나는 노이즈와 이상치입니다.
- **노이즈 및 이상치**: 데이터에 노이즈나 이상치가 있을 경우, k=1에서는 이러한 노이즈나 이상치에 민감하게 반응할 수 있습니다. 반면에 k=2에서는 이러한 노이즈나 이상치가 더 넓은 영역에 분포할 가능성이 있으며, 두 이웃의 거리가 거의 같은 수준으로 멀리 떨어져 있다면, 이러한 노이즈나 이상치를 무시하고 더 안정적인 예측을 할 수 있습니다.
- **데이터의 밀도와 분포**: 데이터의 밀도와 분포도 k=1과 k=2의 선택에 영향을 줄 수 있습니다. 데이터가 밀집한 지역에서는 k=1이 더 정확한 예측을 할 수 있을 것이고, 데이터가 희소한 지역에서는 k=2가 더 안정적인 예측을 할 수 있을 것입니다.
- **클래스의 균형**: 데이터셋에 클래스 간의 불균형이 있을 경우, k=1과 k=2의 선택에 영향을 줄 수 있습니다. 한 클래스가 다른 클래스에 비해 상대적으로 적은 경우, k=1은 해당 클래스를 더 많이 선택할 가능성이 있습니다. 반면에 k=2는 더 안정적으로 다수 클래스를 선택할 수 있습니다.
- 이러한 요소들은 k=1과 k=2의 선택에 영향을 주며, 데이터의 특성에 따라 어떤 k 값을 선택해야 하는지를 결정하는 데 중요한 역할을 합니다.
- **추가 대답 중의 내용 정리:** k가 작을수록 모델은 더 복잡해지고 작은 노이즈에 민감해질 수 있습니다. 하지만 k가 클수록 모델은 더 부드러워지고 일반화 능력이 향상될 수 있습니다. 하지만 k를 너무 크게 선택하면 이웃의 다양성이 부족해 과적합(overfitting)될 수 있으므로 조심해야 합니다.

여러분 생각은 어떠세요?.. ⇨

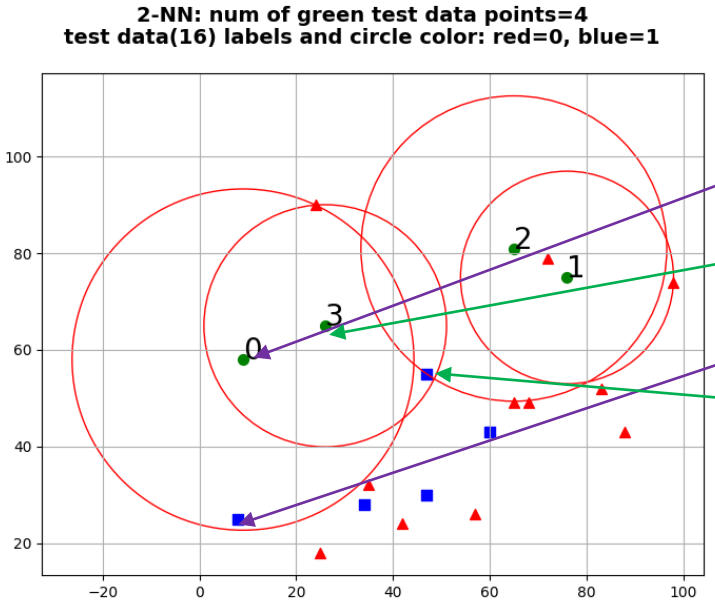
대답의 모든 세부 사항은 공감이 되지 않지만,
아무튼 **k가 짝수인 것이 가능**하고, **이런 정책이 도움**을 줄 수 있는데,
학습 데이터의 분포가 영향을 주는 것 같다는 생각은 듭니다.
9.2의 실험을 통해 추정을 해 보니 이해가 갈 것 같습니다.

9.2 검토 방안 및 실험

- 1. OpenCV가 지원하는 KNN 알고리즘에 대한 심도 싶은 분석 - 이론적 접근, 혹은 파라미터 설정
- 2. 코딩 실험으로 대치하기
  - ~~학습된 모델에서 K=1, k=2로 findNearest()를 수행한 결과가 달라질 때 distance를 검토하기. 이때 예비용으로 k=5, 10일 때 findNearest()를 수행한 결과의 distance와도 함께 비교해 본다.~~
  - ~~모두 같은 distance를 유지하는지 확인한다.~~
  - 불일치할 때의 두 레이블간의 비율 %를 구하고, 그때의 영상도 함께 출력하는 프로그램을 설계한다.

실험 코딩을 완료하였습니다.
나머지는 팀 과제 중의 하나로 계획하고 있습니다. …

실험 1.1: K=2, 7:3 비율, 반전




```
num of train_data=16, num of test_data=4, k=2
result: [[0.]
 [0.]
 [0.]
 [0.]], shape=(4, 1)
neighbours: [[1. 0.]
 [0. 0.]
 [0. 1.]
 [1. 0.]], shape=(4, 2)
distance: [[1090. 1249.]
 [  32.  485.]
 [  53. 1000.]
 [ 541.  629.]], shape=(4, 2)
```

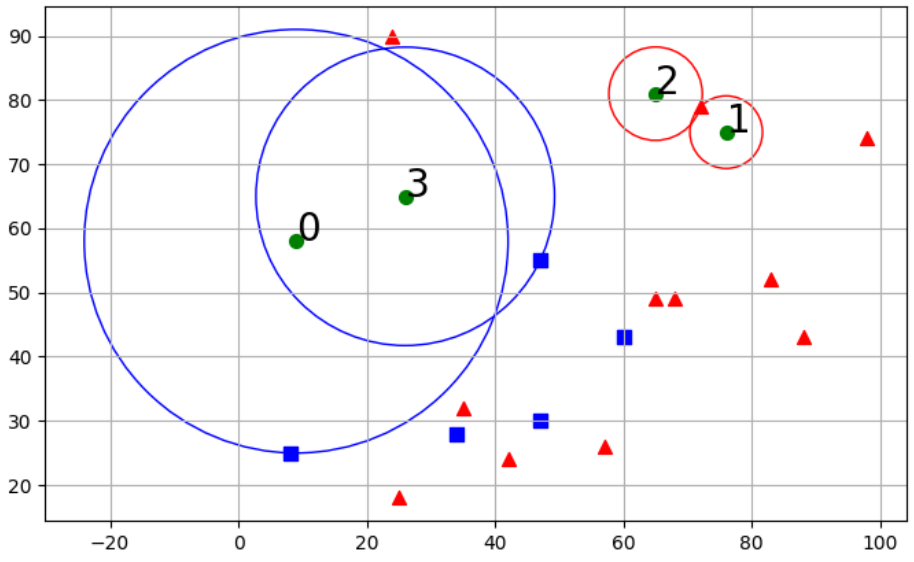
반전이라 함은 neighbour의 1위가 result의 최종 결과로 채택되지 못한 경우를 말합니다.

빨간 삼각형 11개 + 파란 사각형 5개로 1개의 데이터가 7:3으로 편중되었다.
Test data 0에 대해 0(red)로 판별하였는데, 0번째 neighbour은 첫번째로 1(blue)를 꼽고 있으며 0번째 distance는 레이블1의 거리=1,090이 레이블 0의 거리=1249보다 가깝다. Test data 3에 대해서도 result와 neighbour의 결과가 일치하지 않는다.
→ 분포도가 낮은 쪽으로 판정해 주었다고 가정해 본다? → 아니다. 그렇게 단순한 기법이 아닐 듯 싶다.

seed = 370001; np.random.seed(seed)

실험 1.2: K=1, 7:3 비율, 정상

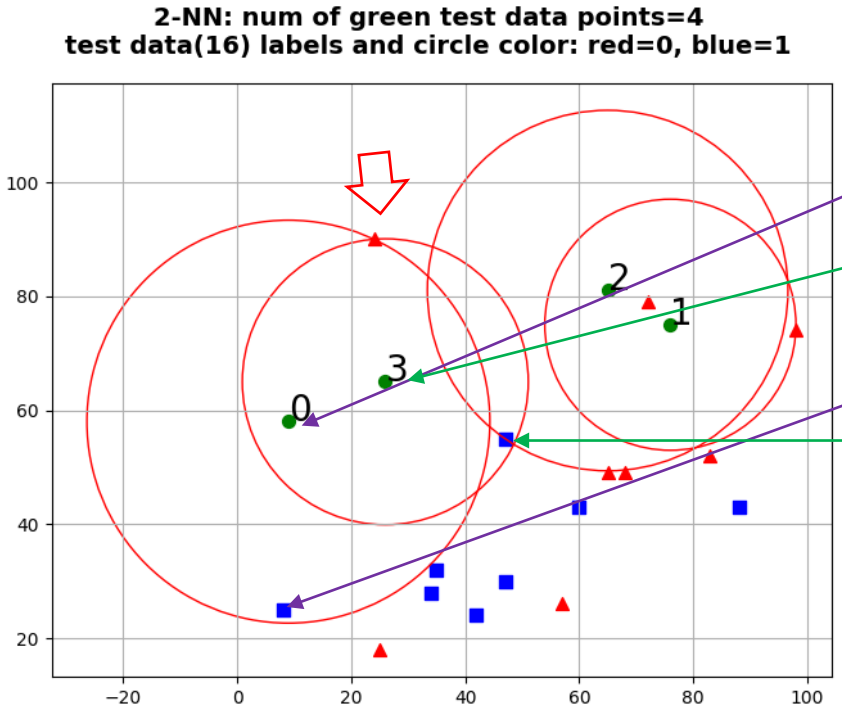
정상이라 함은 neighbour의 1위가 result의 최종 결과로 채택된 경우를 말합니다.




```
num of train_data=16, num of test_data=4, k=1
result: [[1.]
 [0.]
 [0.]
 [1.]], shape=(4, 1)
neighbours: [[1.]
 [0.]
 [0.]
 [1.]], shape=(4, 1)
distance: [[1090.]
 [  32.]
 [  53.]
 [ 541.]], shape=(4, 1)
```

실험 1.1과 같은 조건
result와 neighbor 모두 일치.
seed = 370001; np.random.seed(seed)

실험 2.1 K=2, 5:5 비율, 반전




```
num of train_data=16, num of test_data=4, k=2
result: [[0.]
 [0.]
 [0.]
 [0.]], shape=(4, 1)
neighbours: [[1. 0.]
 [0. 0.]
 [0. 1.]
 [1. 0.]], shape=(4, 2)
distance: [[1090. 1249.]
 [  32.  485.]
 [  53. 1000.]
 [ 541.  629.]], shape=(4, 2)
```

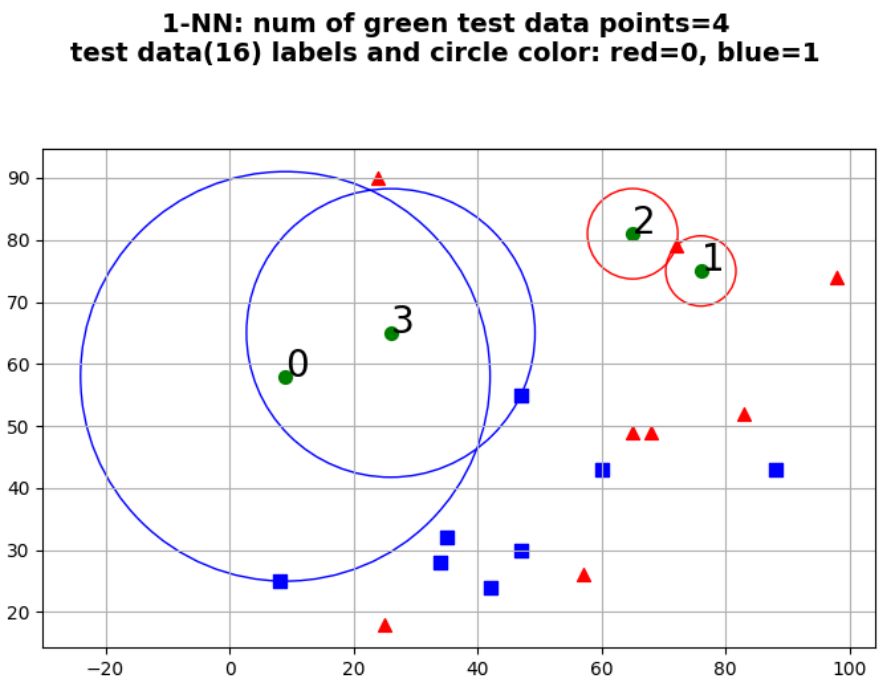
반전이라 함은 neighbour의 1위가 result의 최종 결과로 채택되지 못한 경우를 말합니다.

빨간 삼각형: 파란 사각형 비율 = 5:5 → 같은 분포인데도 반전이 일어난다.
Test data 0, 3번에 대해 result와 neighbour가 반전 결과를 보이고 있다.
추정: 전체적으로 보면 0, 3번 지역은 2번에 의해 영향 받은 overlap 지역과 화살표(⇩)로 마킹한 지점의 영향을 받는다고 볼 수 있다. 이때문에 1위가 아주 아깝지 않은 이상 2위가 result로 반환되는 역전 현상을 보이는 것으로 추정된다.

seed = 370001; np.random.seed(seed)

실험 2.2 K=1, 5:5 비율, 정상

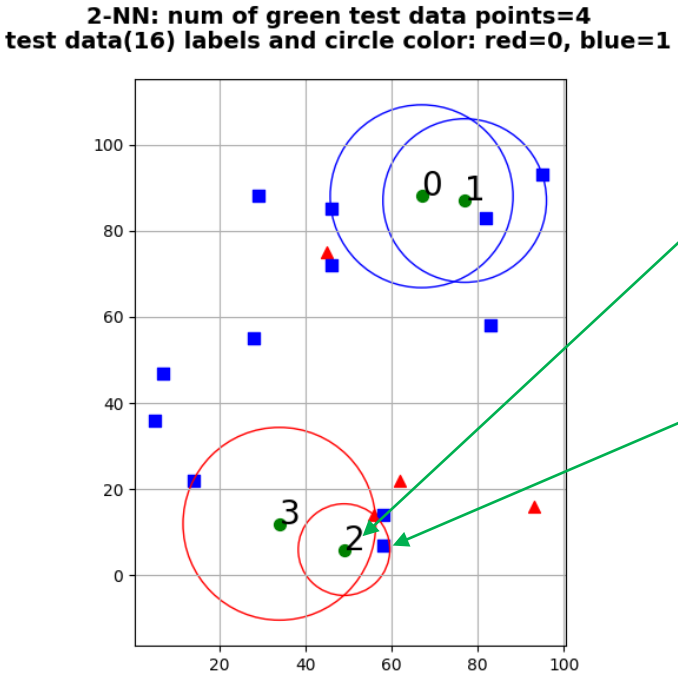
정상이라 함은 neighbour의 1위가 result의 최종 결과로 채택된 경우를 말합니다.




```
num of train_data=16, num of test_data=4, k=1
result: [[1.]
 [0.]
 [0.]
 [1.]], shape=(4, 1)
neighbours: [[1.]
 [0.]
 [0.]
 [1.]], shape=(4, 1)
distance: [[1090.]
 [  32.]
 [  53.]
 [ 541.]], shape=(4, 1)
```

seed = 370001; np.random.seed(seed)

실험 3.1 k=2, 3:7 비율, 반전

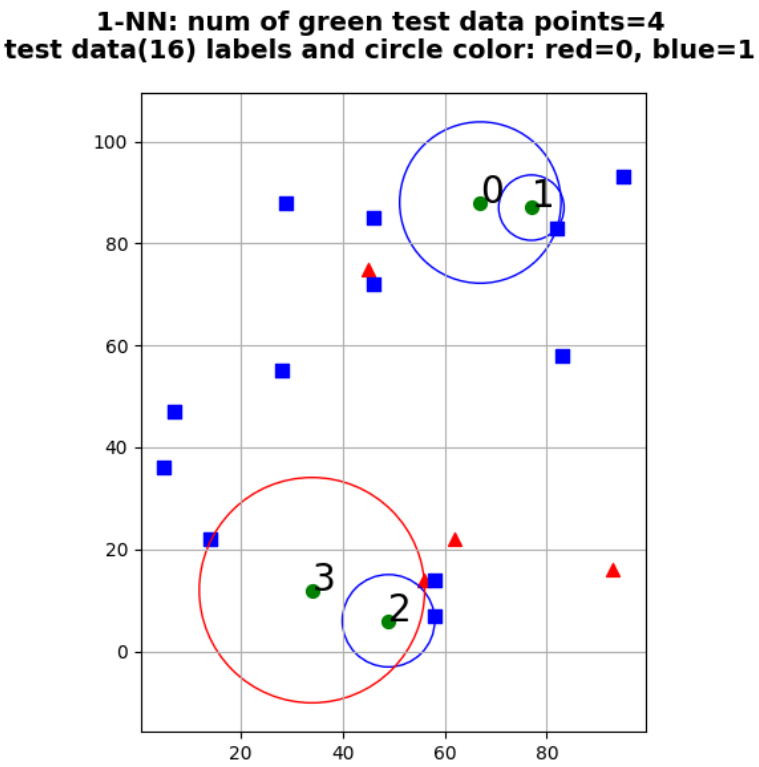



```
num of train_data=16, num of test_data=4, k=2
result: [[1.]
 [1.]
 [0.]
 [0.]], shape=(4, 1)
neighbours: [[1. 1.]
 [1. 1.]
 [1. 0.]
 [0. 1.]], shape=(4, 2)
distance: [[250. 450.]
 [ 41. 360.]
 [ 82. 113.]
 [488. 500.]], shape=(4, 2)
```

반전이라 함은 neighbour의 1위가 result의 최종 결과로 채택되지 못한 경우를 말합니다.

빨간 삼각형: 파란 사각형 비율 = 3:7 → 분포도가 낮은 쪽으로 판별해 주었다.
Test data 2번에 대해 result와 neighbour가 반전 결과를 보이고 있다.
이하 2페이지 모두 16개의 난수 위치 지정에 대해서는 같은 조건의 seed를 사용하였다.
seed = 370601
np.random.seed(seed)

실험 3.2 k=1, 3:7 비율, 정상



정상이라 함은 neighbour의 1위가 result의 최종 결과로 채택된 경우를 말합니다.

```
num of train_data=16, num of test_data=4, k=1
result: [[1.]
 [1.]
 [1.]
 [0.]], shape=(4, 1)
neighbours: [[1.]
 [1.]
 [1.]
 [0.]], shape=(4, 1)
distance: [[250.]
 [ 41.]
 [ 82.]
 [488.]], shape=(4, 1)
```

9.3 현재까지 알아낸 사항

- K=2의 판단에 분포도가 영향을 미치는 것인가?
  - 학습 데이터 중에 좌표 부분은 고유하고, 레이블의 분포를 바꾸어(7:3, 5:5, 4:6) 실험해 보았는데 분포가 많거나, 적다고 해서 어느 한 쪽에 유리한 판단(단순한)을 하는 것 같지 않다.
- 그보다는 대략 다음과 같이 전체적인 판세를 고려하여 판단하는 것으로 보인다.
  - 첫 번째 순위의 query data와의 거리 vs (두 번째 순위의 거리 + 주변 판세의 가중치)
- 사실 이것이 더 합리적인 기법으로 여겨지는데 k의 차수가 높아질 때도 이런 복잡 메커니즘을 사용할 것인지의 여부는 설계자의 재량에 달렸다고 보는 것이 맞을 듯 싶다. 복잡도의 문제일 듯…
  - k가 커져도 이런 메커니즘을 사용하는지..
  - 짝수일 때만 이런 메커니즘을 사용하는지..,
  - K=3, 5,… 즉, 홀수일 때도 이런 메커니즘을 사용하는지 …

9.4 새로운 실험 테마

- 새로운 실험 테마　　팀 과제 중의 하나로 계획하고 있습니다. …
  - 실험적으로 이를 확인하는 간접적인 방법은 query data point를 k=2, 4..일 때 전 영역으로 확장하여 색칠을 해보면 알게 될 것으로 보인다.
    - 색칠한 영상에 학습 데이터를 마킹해 보고, 그 위에 circle를 그려보면 분명해 질 것이다.
  - K=1, 3, 5일 때도 마찬가지 이다.
- 선결 주제 - 아래 기법의 추가 개발이 필요함.
  - 모든 실험에 같은 학습 데이터를 사용하게 만들어야 한다. → seed() 함수
  - 학습 데이터의 레이블의 분포(비율)를 통제할 수 있어야 한다. → choice() 함수
  - 같은 학습 데이터의 위치에 다른 레이블 배정되는 일이 없어야 한다. → 미 실현

9.5 참고: L1-norm vs L2 norm

출처: Norms and machine learning

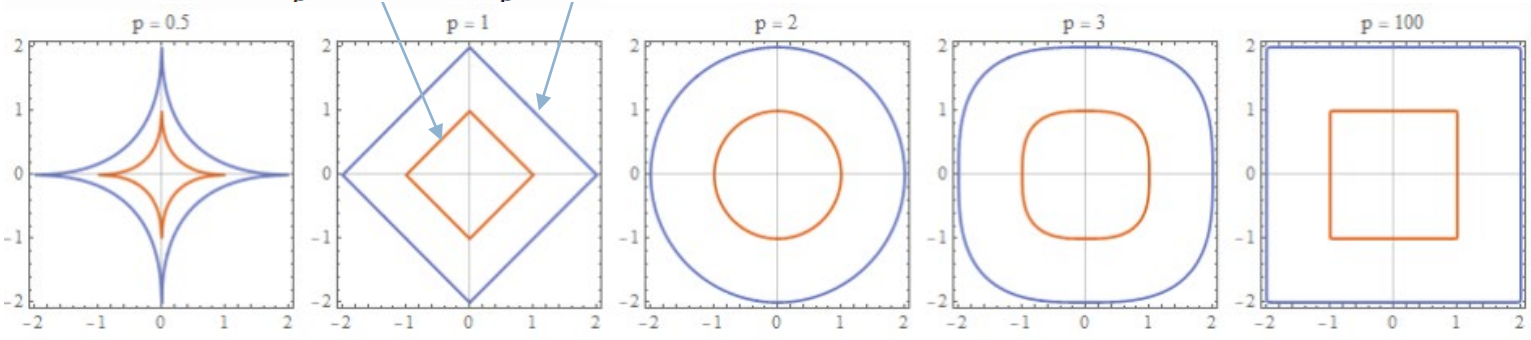
$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$

For $p = 1$, we get $\ell_1 = |x_1| + |x_2| + \ldots + |x_n|$

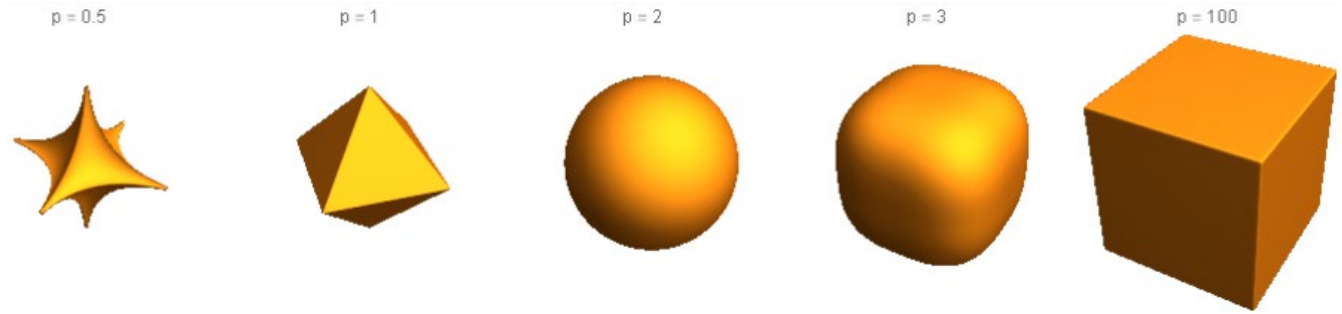
For $p = 2$, $\ell_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$

For $p = 3$, $\ell_3 = \sqrt[3]{|x_1|^3 + |x_2|^3 + \ldots + |x_n|^3}$

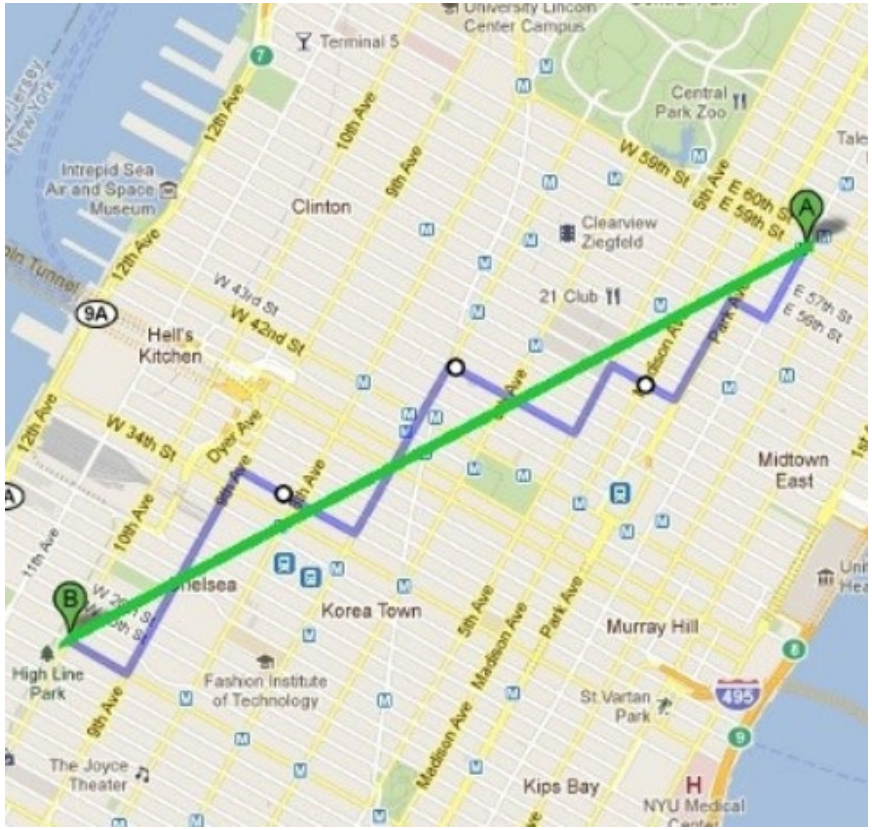
boundaries for $\ell_p = 1$ and $\ell_p = 2$ *in* $\mathbb{R}^2$



boundary for $\ell_p = 1$ in $\mathbb{R}^3$



shape of the ℓ_p norm for various values of p



- p 의 값에 따라 ℓ_p - norm의 값이 달라지기 때문에 어떤 문제를 해결하는 가에 따라 어떤 ℓ_p 를 사용할 것인지 정해야 한다.
- 좌측의 지도에서 최적의 경로를 찾아 A-B간의 거리를 판단할 때는
  - 택시의 경우에는 보라색의 ℓ_1 norm을 사용하고,
  - 헬리콥터의 경우에는 녹색의 ℓ_2 norm을 사용해야 할 것이다.

참고문헌-메모리 최적화

- **희소 표현(Sparse Representation):**
  - A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in ICML, 2013.
  - D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in ICLR, 2015.
- **분할된 kNN(Incremental kNN):**
  - D. D. Lewis, "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval," in ECML, 1998.
  - Y. Li, W. Dai, and Q. Yang, "Towards Scalable and Reliable Collaborative Filtering for Large-scale Recommender Systems," in KDD, 2014.
- **근사 kNN(Approximate kNN):**
  - A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," in VLDB, 1999.
  - J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 888-905, 2000.
- **차원 축소 기법:**
  - C. M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006.
  - L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," in Journal of Machine Learning Research, vol. 9, pp. 2579-2605, 2008.

참고문헌-탐색법 최적화

- **거리 행렬의 계산 최적화:**
  - J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in OSDI, 2004.
  - Y. Zhang, D. Hu, J. Wang, and J. Hu, "Large-Scale k-Nearest Neighbor Search on Heterogeneous Platforms," in IEEE Transactions on Computers, vol. 64, no. 1, pp. 145-159, 2015.
- **근사적인 이웃 탐색:**
  - P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993.
  - S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," in Journal of the ACM, vol. 45, no. 6, pp. 891-923, 1998.
- **차원 축소 기법:**
  - C. M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006.
  - A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," in VLDB, 1999.
- **분할 및 병렬화:**
  - D. Comer, "The Ubiquitous B-Tree," in ACM Computing Surveys, vol. 11, no. 2, pp. 121-137, 1979.
  - M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," in Communications of the ACM, vol. 24, no. 6, pp. 381-395, 1981.