|  |  |
| --- | --- |
| **ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ**  TRƯỜNG ĐẠI HỌC BÁCH KHO KHOA ĐIỆN – ĐIỆN TỬ **BỘ MÔN VIỄN THÔNG**  o0o  F:\Hoi sinh vien\Logo - Baner\logo\Logo-Truong.gif  **BÀI TẬP LỚN XỬ LÝ ẢNH**  **KNN CLASSIFIER**  **GVHD:** Nguyễn Khánh Lợi  **Nhóm 9**  SVTH | **MINH**  A  MSSV |
| Nguyễn Nhựt Linh | 1811861 |
| Lê Hữu Khánh | 1812590 |
| Lương Trương Hùng | 1711598 |
| Nguyễn Bảo Kỳ  Trần Cảnh Hưng | 1811020  1812503 |
|  |  |

**LỜI CẢM ƠN**

Đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Khánh Lợi đã nhiệt tình hướng dẫn chúng em trong suốt quá trình học môn Xử lý ảnh, cung cấp cho chúng em nhiều kiến thức hay, bổ ích và rất cần thiết cho chúng em.

Trong quá trình học tập, cũng như là trong quá trình làm bài tiểu luận này khó tránh khỏi sai sót, rất mong Thầy bỏ qua. Đồng thời do trình độ lý luận cũng như kinh nghiệm thực tiễn còn hạn chế nên bài tiểu luận không thể tránh khỏi những thiếu sót, chúng em rất mong nhận được ý kiến đóng góp của Thầy để chúng em học thêm được nhiều kinh nghiệm và hoàn thành tốt hơn bài báo cáo tốt nghiệp sắp tới.

Chúng em xin chân thành cảm ơn!

**MỤC LỤC**

Contents

[**I.** **GIỚI THIỆU** 2](#_Toc121295623)

[**II.** **LÝ THUYẾT** 2](#_Toc121295624)

[1. Khái niệm: 2](#_Toc121295625)

[2. Các bược thực hiện KNN 2](#_Toc121295626)

[**3.** **Ví dụ về bài toán KNN:** 2](#_Toc121295627)

[4. Ví dụ về KNN nhiễu 4](#_Toc121295628)

[**5.** **Cách chọn K** 4](#_Toc121295629)

[6. Khoảng cách trong không gian vector 4](#_Toc121295630)

[**7.** **Ưu điểm, nhược điểm KNN** 7](#_Toc121295631)

[**III.** **ỨNG DỤNG CỦA THUẬT TOÁN KNN** 8](#_Toc121295632)

[**1.** **Dùng thuật toán KNN cho mặt phẳng 2D:** 8](#_Toc121295633)

[**2.** **Dùng thuật toán KNN nhận diện chữ số viết tay (bài toán KNN với data là những vecto n chiều)** 15](#_Toc121295634)

[**IV.** **KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN** 26](#_Toc121295635)

1. **GIỚI THIỆU**

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression. KNN còn được gọi là một thuật toán Instance-based hay Memory-based learning

1. **LÝ THUYẾT**

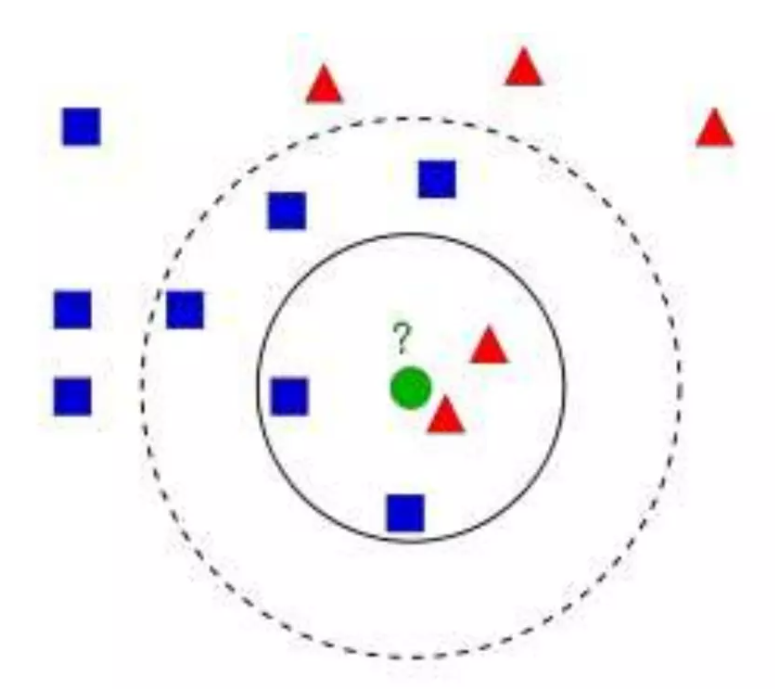
## Khái niệm:

Thuật toán K láng giềng gần nhất trong tiếng Anh là K-Nearest Neighbor, viết tắt là KNN.

Thuật toán K láng giềng gần nhất là một kĩ thuật học có giám sát (supervised learning) dùng để phân loại quan sát mới bằng cách tìm điểm tương đồng giữa quan sát mới này với dữ liệu sẵn có.

## Các bược thực hiện KNN

* Bước 1: Ta có D là tập các điểm dữ liệu đã được gắn nhãn và A là dữ liệu chưa được Phân loại
* Bước 2: Đo khoảng cách (Euclidian, Manhattan, Minkowski, Minkowski hoặc Trọng số) từ dữ liệu mới A đến tất cả các dữ liệu khác đã được phân loại trong D.
* Bước 3: Chọn K (K là tham số mà bạn định nghĩa) khoảng cách nhỏ nhất.
* Bước 4: Kiểm tra danh sách các lớp có khoảng cách ngắn nhất và đếm số lượng của mỗi lớp xuất hiện.
* Bước 5: Lấy đúng lớp (lớp xuất hiện nhiều lần nhất).
* Bước 6: Lớp của dữ liệu mới là lớp mà bạn đã nhận được ở bước 5.
  1. **Ví dụ về bài toán KNN:**



Giả sử bài toán được đặt ra: mình mới quen một ngời bạn, tuy nhiên mình là fan của Us-Uk vậy nên mình cần biết người bạn này có phải là fan của K-pop hay không. Qua thời gian tìm hiểu mình đã thu thập được một số dự liệu và đã biểu hiện dưới dạng hình vễ trên

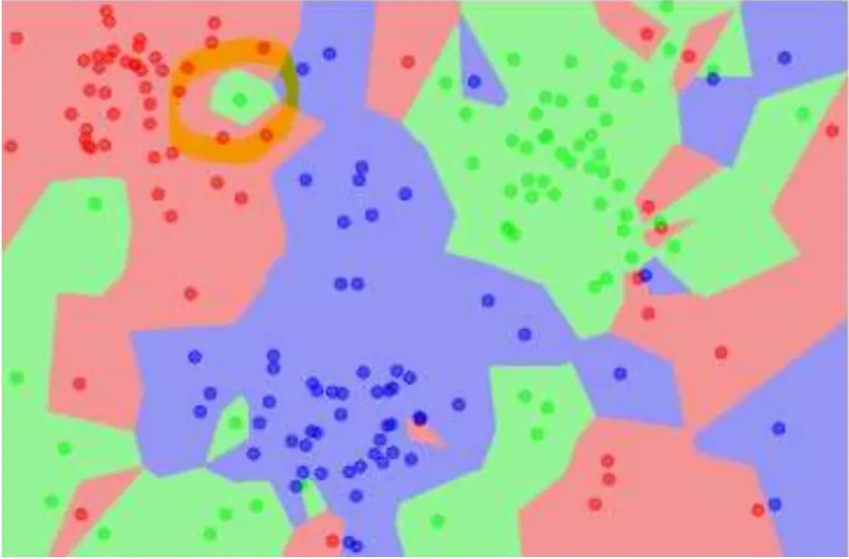
Ta dễ dàng nhìn thấy có hai loại: hình vuông màu xanh biểu diễn cho nhưng người là fan của K-pop, tam giủ màu đỏ biểu diễn cho những người không là fan của K-pop, hình tròn màu xanh là người bạn mình muốn biết còn lại biểu diễn độ thân thiết của bạn đó với những người bạn

Phương phán đơn giản nhất để kiểm tra xem bạn đó chơi thân với người bạn nào nhấtm tức là tìm xem điểm gần chấm xanh thuộc class nào (hình vuôn hay tâm giác). Từ hình trên ta dễ dàng nhận thấy điểm gần chấm xanh là hình tam giác màu đỏ, do đó nó sẽ được phân vào lớp tam giác màu đỏ

Có một vấn đề trong phương pháp trên, xung quanh chấm xanh xuất hiện rất nhiều hình vuông màu xanh nên việc xét điểm gần nhất là chưa khả thị. Vì vậy, ta sẽ xét k điểm gần nhất. Giả sử, ta lấy K=3, dựa theo hình trên ta dễ dàng nhận ra có hai hình tam giác đỏ và một hình vuông xanh có khoảng cách gần chấm xanh nhất, do đó chấm xanh được phân vào lớp tam giác đỏ. Lấy K=7, ta có năm hình vuông xanh và hai hình tam giác đỏ, lúc này chấm xanh được xếp vào lớp hình vuông xanh. Trường hợp lấy K=4, ta nhận thấy ssẽ có hai hình vuông xanh và hai hình tam giác đỏ, đây là trường hợp có điểm bằng nhau, với trường hợp này KNN sẽ xử lý bằng cách so sánh tổng khoảng cách của các hình gần nhất với điểm ta đang xét.

Do xuất hiện trường hợp có điểm bằng nhau, vì vậy người ta thường chọn k là số lẻ. Đó cũng là ý tưởng của KNN

## Ví dụ về KNN nhiễu



Hình trên là bài toán phân lớp với ba lớp: đỏ, lam, lục. Mỗi điểm dữ liệu mới sẽ được gắn nhãn theo màu của điểm đó mà nó thuộc về. Trong hình này, chú ý vùng khoanh tròn màu vàng, ta nhận thấy rằng điểm điểm màu lục nằm giữa hai vùng lớn với nhiều dữ liệu đỏ và lam, điểm này rất có thể là nhiễu đến việc dữ liệu test nếu rơi vào vùng sẽ có nhiều khả năng cho kết quả sai lệch

* 1. **Cách chọn K**
* Nếu số lượng mẫu có sẵn vô hạn, thì càng lớn càng tốt
* K nhỏ hơn: phương sai cao hơn (kém ổn định hơn)
* Quy tắc ngón tay cái: k = sqrt(n), n: số mẫu
* k = 1: cho hiệu quả, nhưng có thể ảnh hưởng với “noise”
* K lớn hơn có thể cải thiện hiệu suất, nhưng k quá lớn sẽ phá hủy tính cục bộ
* K nhỏ hơn: phương sai cao hơn (kém ổn định hơn)
* K lớn hơn: độ lệch cao hơn (ít chính xác hơn)
* Nếu chúng ta có nhiều mẫu, kNN hoạt động tốt

## Khoảng cách trong không gian vector

Trong không gian một chiều, việc đo khoảng cách giữa hai điểm đã rất quen thuộc: lấy trị tuyệt đối của hiệu giữa hai giá trị đó. Trong không gian hai chiều, tức mặt phẳng, chúng ta thường dùng khoảng cách Euclid để đo khoảng cách giữa hai điểm

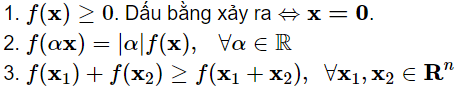
Việc đo khoảng cách giữa hai điểm dữ liệu nhiều chiều, tức hai vector, là rất cần thiết trong Machine Learning. Chúng ta cần đánh giá xem điểm nào là điểm gần nhất của một điểm khác; chúng ta cũng cần đánh giá xem độ chính xác của việc ước lượng; và trong rất nhiều ví dụ khác nữa.

Và đó chính là lý do mà khái niệm norm (chuẩn) ra đời. Có nhiều loại norm khác nhau mà các bạn sẽ thấy ở dưới đây:

Để xác định khoảng cách giữa hai vector y và z, người ta thường áp dụng một hàm số lên vector hiệu x = y − z. Một hàm số được dùng để đo các vector cần có một vài tính chất đặc biệt.

**Định nghĩa:**

Một hàm số f() ánh xạ một điểm x từ không gian n chiều sang tập số thực một chiều được gọi là norm nếu nó thỏa mãn ba điều kiện sau đây:



**Một số chuẩn thường dùng:**

Giả sử các vectors x= [x1;x2;…;xn ], y=[y1;y2;…;yn ]

Nhận thấy rằng khoảng cách Euclid chính là một norm, norm này thường được gọi là norm 2:



Với p là một số không nhỏ hơn 1 bất kỳ, hàm số sau đây:



được chứng minh thỏa mãn ba điều kiện bên trên, và được gọi là norm p.

Nhận thấy rằng khi p → 0 thì biểu thức bên trên trở thành số các phần tử khác 0 của x. Hàm số (2) khi p = 0 được gọi là giả chuẩn (pseudo-norm) 0. Nó không phải là norm vì nó không thỏa mãn điều kiện 2 và 3 của norm. Giả-chuẩn này, thường được ký hiệu là ||x||0 , khá quan trọng trong Machine Learning vì trong nhiều bài toán, chúng ta cần có ràng buộc “sparse”, tức số lượng thành phần “active” của x là nhỏ.

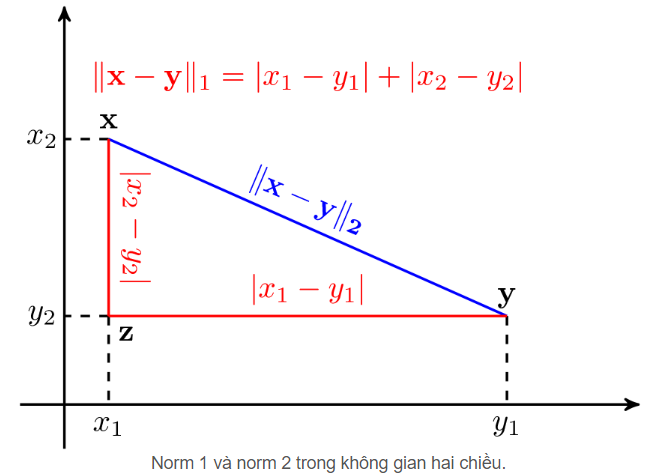
Có một vài giá trị của p thường được dùng:

+ Khi p = 2 chúng ta có norm 2 như ở trên.

+ Khi p=1 chúng ta có:



là tổng các trị tuyệt đối của từng phần tử của x. Norm 1 thường được dùng như xấp xỉ của norm 0 trong các bài toán có ràng buộc “sparse”. Dưới đây là một ví dụ so sánh norm 1 và norm 2 trong không gian hai chiều:



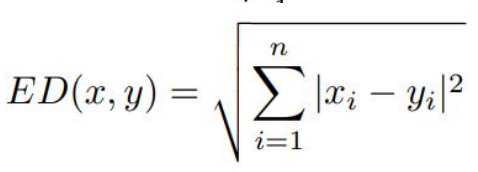
Norm 2 (màu xanh) chính là đường thằng “chim bay” nối giữa hai vector x và y. Khoảng cách norm 1 giữa hai điểm này (màu đỏ) có thể diễn giải như là đường đi từ x tới y trong một thành phố mà đường phố tạo thành hình bàn cờ. Chúng ta chỉ có cách đi dọc theo cạnh của bàn cờ mà không được đi thẳng.

+ Khi p → ∞, ta có norm p chính là trị tuyệt đối của phần tử lớn nhất của vector đó:



Ngoài khoảng cách Euclid ngoài ra còn có các khoảng cách Minkowski, Mahattan, Chebyshev:

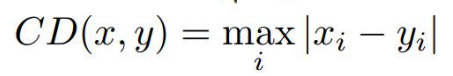
* Euclidean Distance;



+Ưu điểm: Phổ biến, dễ hiểu, dễ implement, kết quả tốt trong nhiều usecase, đặc biệt hiệu quả với dữ liệu ít chiều.

+Nhược điểm: Euclide distance có thể bị ảnh hưởng bởi đơn vị của feature. Chính vì vậy cần phải normalize trước khi tính toán. Vấn đề thứ 2, khi số chiều vector space tăng lên, Euclide Distance trở nên kém hiệu quả. Một phần nguyên nhân do dữ liệu thực tế thường không chỉ nằm trong Euclide Metric Space

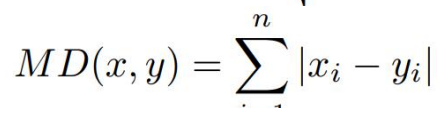
* Chebyshev Distance:



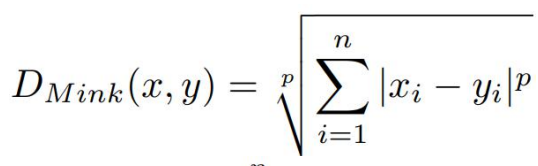
+ Nhược điểm: Chỉ tốt với vài trường hợp cụ thể do không có tính generalize tốt như các phương pháp khác như Euclidean hay Cosine similarity.

+ Nhược điểm: Đặc điểm của measrue này cho phép tính được số bước đi tối thiểu từ ô này sang ô khác. Nên có thể có lợi trong các trò chơi, game cho phép di chuyển 8 hướng. Trong thực tế Chebyshev distance thường được sử dụng trong warehouse logistics (điều khiển cần trục di chuyển)

* Mahattan Distance:



* Minkowski distance:



+Có thể thấy Minkowski distance là trường hợp tổng quát của các distance bên:

p = 1 => Manhattan distance

p = 2 => Euclidean distance

p = ∞ => Chebyshev distance

+ Ưu điểm: Minkowski có các nhược điểm tương tự các phương pháp kể trên. Tham số pp có thể cần phải điều chỉnh rất nhiều để tìm được giá trị phù hợp

+ Nhược điểm: Việc có thể chọn lựa tham số pp thích hợp giúp việc điều chỉnh mô hình trở nên linh họat hơn

Dưa trên các ưu, nhược điểm đã phân tích, trong bài này chúng em chọn khoảng cách Euclipdean Distance trong bài cáo này.

* 1. **Ưu điểm, nhược điểm KNN**
* Ưu điểm:

+ Dễ sử dụng và cài đặt

+ Việc dự đoán kết quả của dữ liệu mới dễ dàng.

+ Độ phức taho tính toán nhỏ

* Nhược điểm:

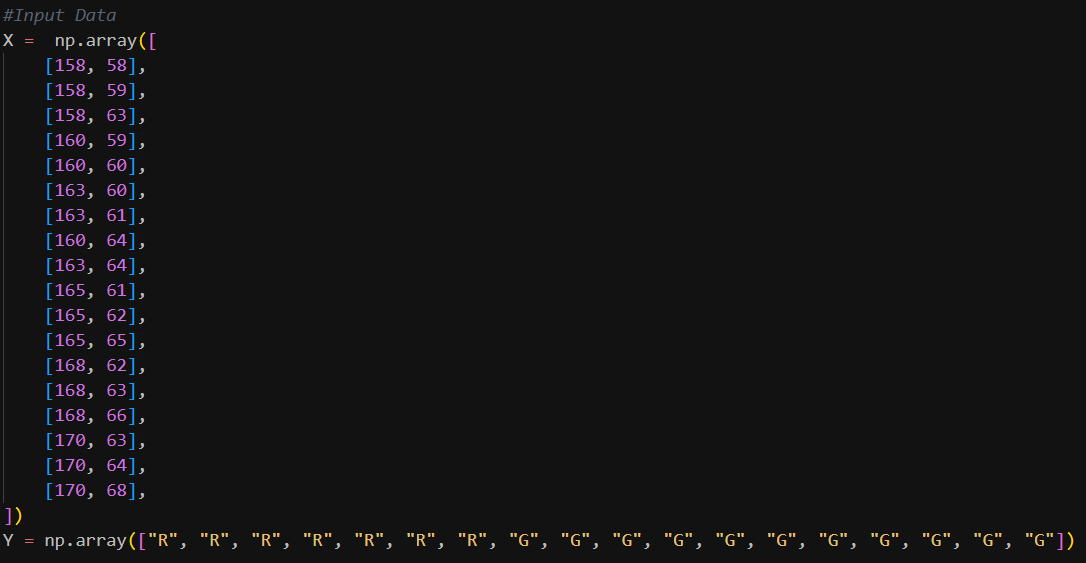
+ KNN rất nhạy cảm với nhiễu khi K nhỏ.

+ Như đã nói, KNN là một thuật toán mà mọi tính toán đều nằm ở khâu test. Trong đó việc tính khoảng cách tới từng điểm dữ liệu trong training set sẽ tốn rất nhiều thời gian, đặc biệt là với các cơ sở dữ liệu có số chiều lớn và có nhiều điểm dữ liệu. Với K càng lớn thì độ phức tạp cũng sẽ tăng lên. Ngoài ra, việc lưu toàn bộ dữ liệu trong bộ nhớ cũng ảnh hưởng tới hiệu năng của KNN.

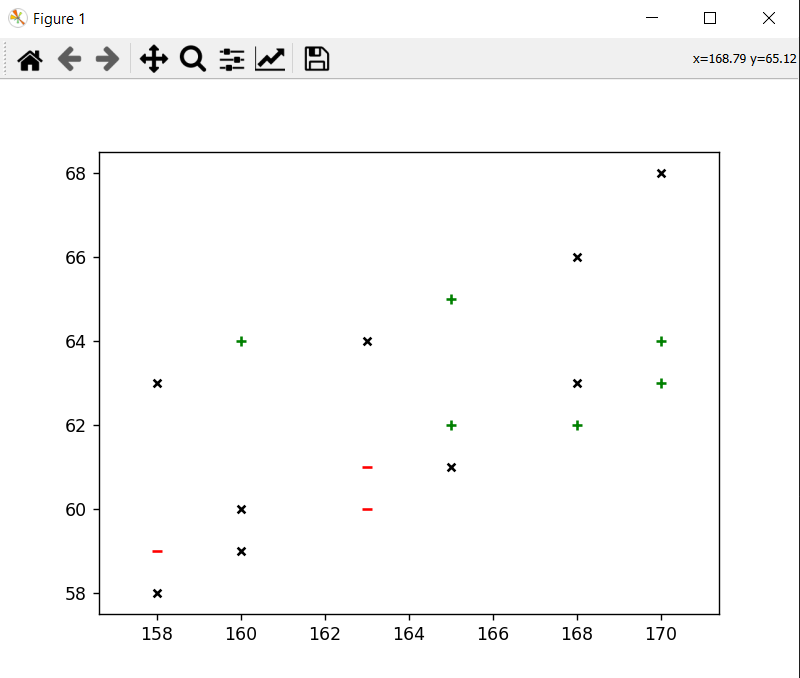
1. **ỨNG DỤNG CỦA THUẬT TOÁN KNN**
2. **Dùng thuật toán KNN cho mặt phẳng 2D:**

Cho 1 tập data gồm n phần tử và mỗi phần tử có dạng x = [x1, x2] và sử dụng 50% data để train và còn lại để test.

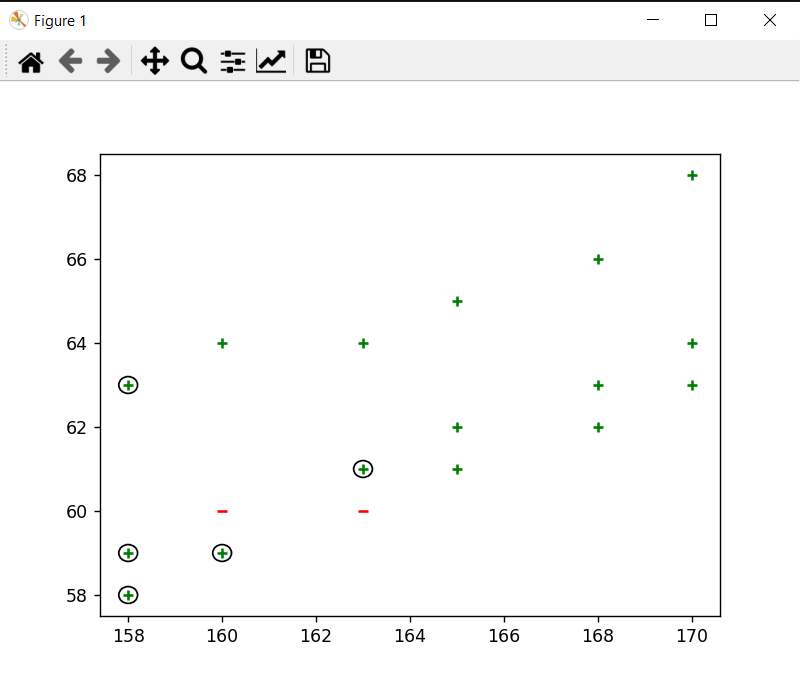
Cụ thể tại bài toán này, ta có tập data X gồm có 18 phần tử và label của mỗi data là Y:



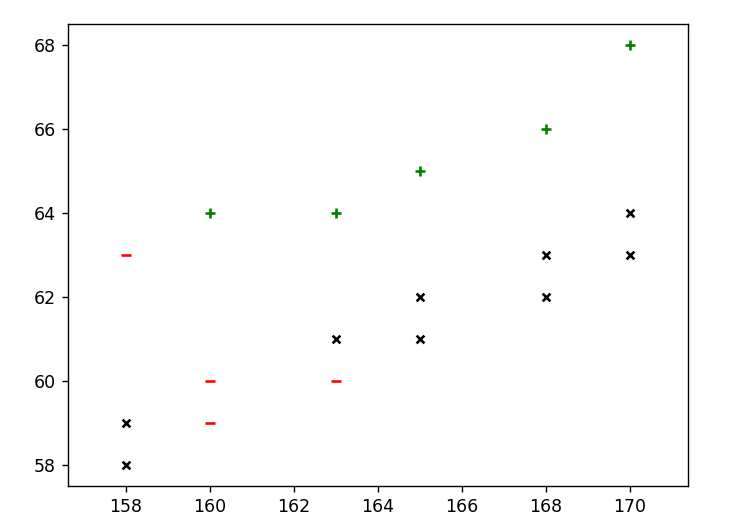
* Tập trainning gồm có 9 phần tử bất kì với những phần tử có label là “R” được biểu diễn bằng “-“ màu xanh còn những phần tử có label là “G” được biểu diễn bằng “+” màu đỏ. Còn lại là tập test được biểu diễn bằng “x” màu đen:

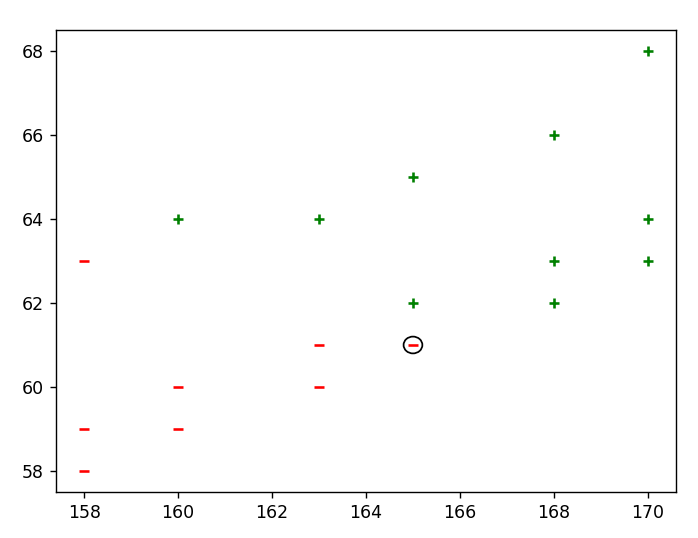


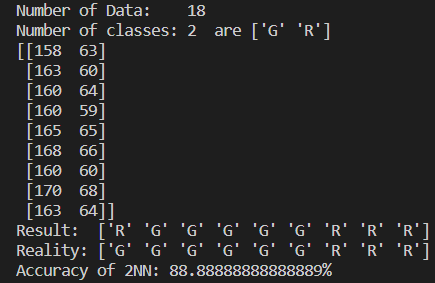
* Và những điểm khoanh tròn là những điểm vi phạm so với label đã được đặt trước đó:



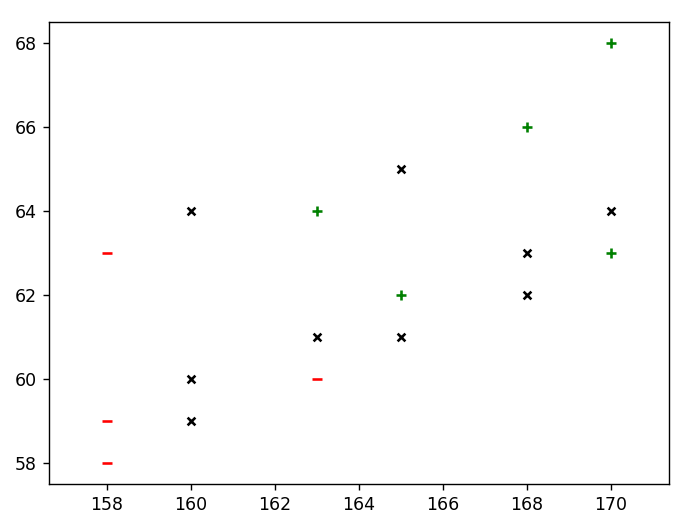
* Với k = 1 và sử dụng khoảng cách Euclidian:

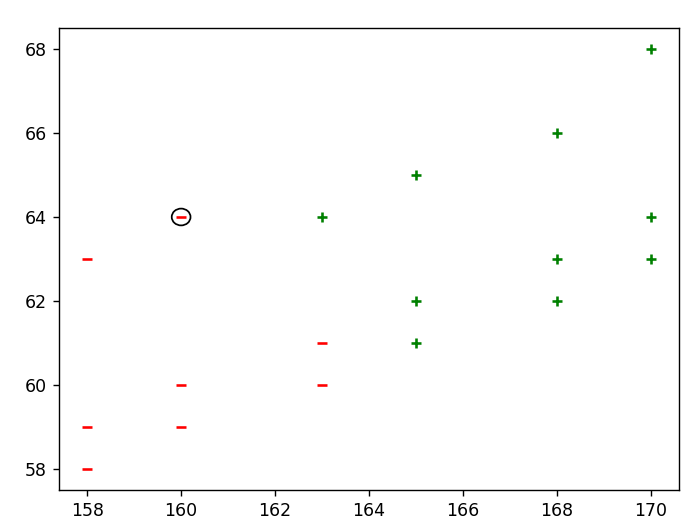


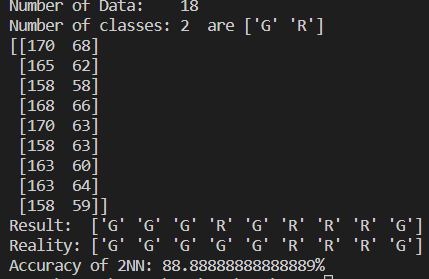




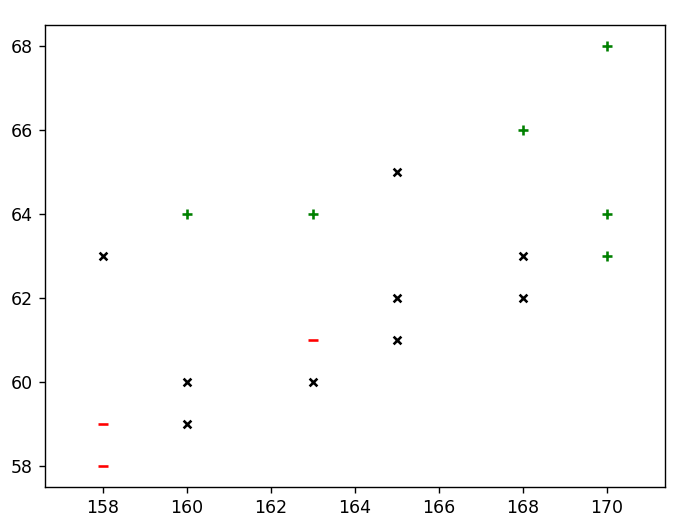
* Với k = 5 và sử dụng khoảng cách Euclidian:

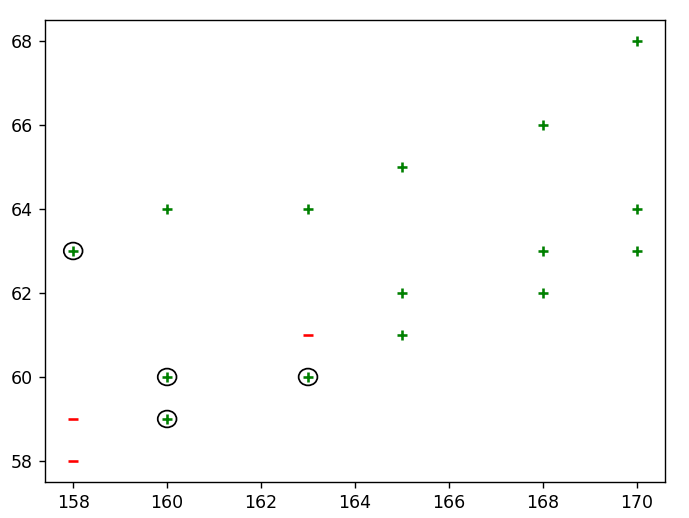


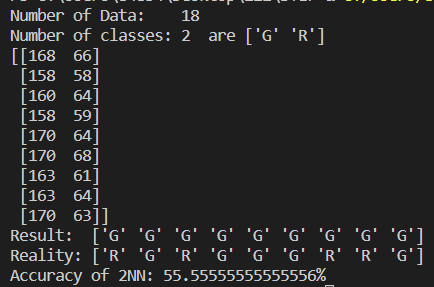




* Với k = 7 và sử dụng khoảng cách Euclidian:







* Nhận xét:

+ K nhỏ hơn: phương sai cao hơn (kém ổn định hơn)

+ K lớn hơn: độ lệch cao hơn (ít chính xác hơn)

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn import neighbors

from sklearn.metrics import accuracy\_score

*#Input Data*

X =  np.array([

    [158, 58],

    [158, 59],

    [158, 63],

    [160, 59],

    [160, 60],

    [163, 60],

    [163, 61],

    [160, 64],

    [163, 64],

    [165, 61],

    [165, 62],

    [165, 65],

    [168, 62],

    [168, 63],

    [168, 66],

    [170, 63],

    [170, 64],

    [170, 68],

])

Y = np.array(["R", "R", "R", "R", "R", "R", "R", "G", "G", "G", "G", "G", "G", "G", "G", "G", "G", "G"])

*#kNN*

def kNN(*X*, *Y*):

*#Choose Test and Train*

    print("Number of Data:    {}".format(len(X)))

    print("Number of classes: {}  are {}".format(len(np.unique(Y)), np.unique(Y))) *#np.unique(Y): name of class*

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, *test\_size*=9)

    print(X\_train)

*#Plot*

    for i, sample in enumerate(X\_train):

        if y\_train[i] == 'R':

            plt.scatter(X\_train[i, 0], X\_train[i, 1], *s*=40, *marker*='\_', *color*='red')

        else:

            plt.scatter(X\_train[i, 0], X\_train[i, 1], *s*=40, *marker*='+', *color*='green')

    for i, sample in enumerate(X\_test):

        plt.scatter(X\_test[i, 0], X\_test[i, 1], *s*=20, *marker*='x', *color*='black')

    plt.axis('equal')

    plt.show()

*#Calc distance*

    clf = neighbors.KNeighborsClassifier(*n\_neighbors* = 1, *p* = 2)*#p = 2 the standard Euclidean distance*

    clf.fit(X\_train, y\_train)

    y\_pred = clf.predict(X\_test)

    print("Result:  {}".format(y\_pred))

    print("Reality: {}".format(y\_test))

    print("Accuracy of {}NN: {}%".format(2, (100\*accuracy\_score(y\_test, y\_pred))))

*#Plot*

    for i, sample in enumerate(X\_train):

        if y\_train[i] == 'R':

            plt.scatter(X\_train[i, 0], X\_train[i, 1], *s*=40, *marker*='\_', *color*='red')

        else:

            plt.scatter(X\_train[i, 0], X\_train[i, 1], *s*=40, *marker*='+', *color*='green')

    for i, sample in enumerate(X\_test):

        if y\_pred[i] == 'R':

            plt.scatter(X\_test[i, 0], X\_test[i, 1], *s*=40, *marker*='\_', *color*='red')

        else:

            plt.scatter(X\_test[i, 0], X\_test[i, 1], *s*=40, *marker*='+', *color*='green')

    for i, sample in enumerate(y\_pred):

        if sample != y\_test[i]:

            circle1 = plt.Circle((X\_test[i, 0], X\_test[i, 1]), .2, *color*='black', *fill* = False)

            plt.gcf().gca().add\_artist(circle1)

    plt.show()

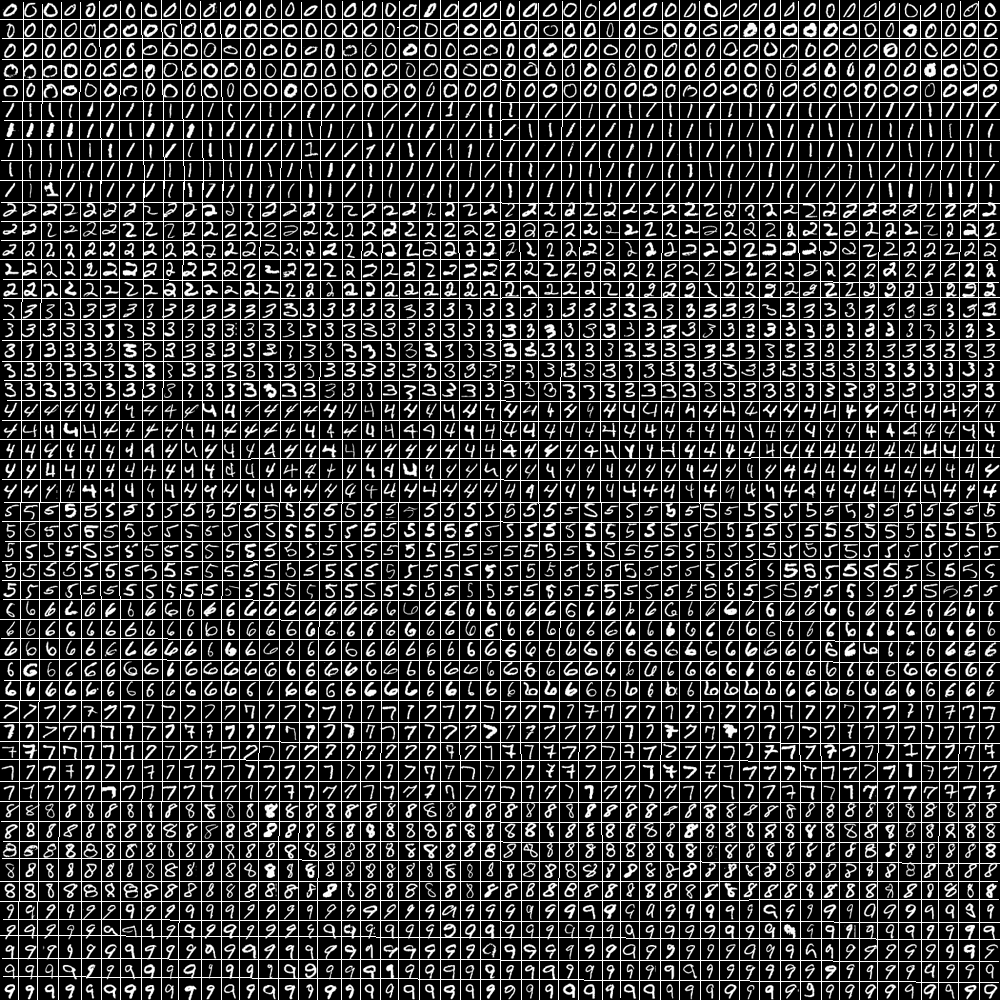
*#Main*

if \_\_name\_\_ == "\_\_main\_\_":

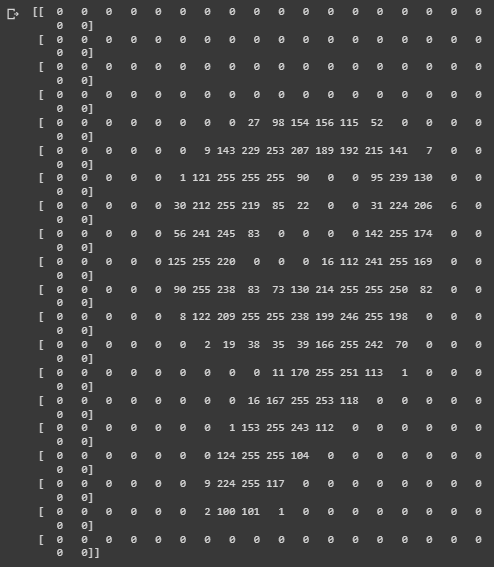
    kNN(X, Y)

1. **Dùng thuật toán KNN nhận diện chữ số viết tay (bài toán KNN với data là những vecto n chiều)**

Lấy data từ tập ảnh chữ viết tay có kích thước ảnh là 1000x1000 pixels.Trong đó có 50x50 chữ viết tay bao gồm từ 0-9. Nên ta cắt ảnh đó thành ma trận gồm 50 hàng và 50 cột

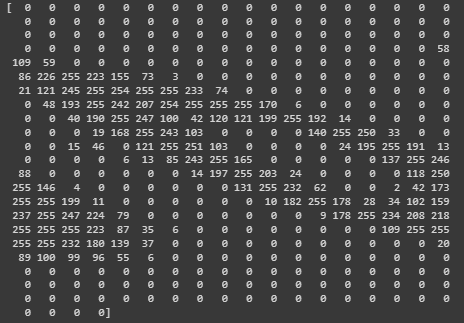


**Từ đó tập data gồm có 2500 phần tử và mỗi phần tử là 1 bức ảnhc ó kích thước là 20x20 pixels)**

****

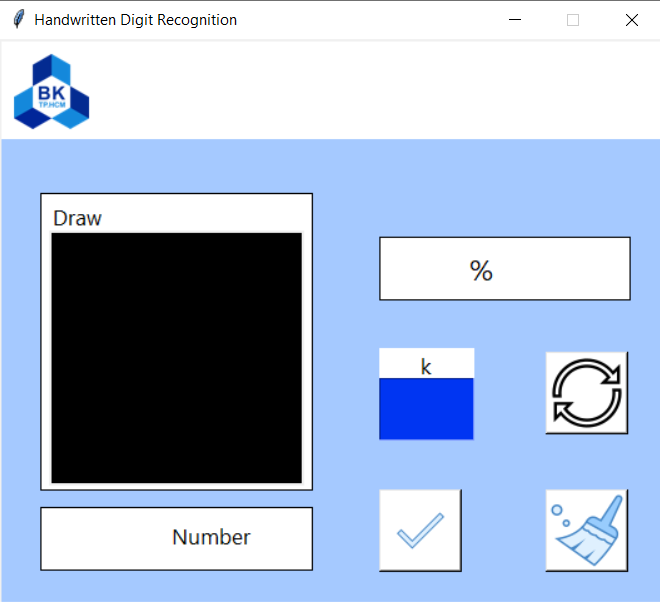
**(ví dụ về hình ảnh số 9)**

**(Chính vị vậy ta phải chuyển bức ảnh đó từ dạng ma trận 20x20 sang ma trận 400x1 để có thể so sánh và tính toán)**

****

**(sau khi biến trở thành 1 hàng)**

* Giao diện:



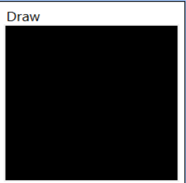
+  nhập k

+  update lại giá trị của k sau khi nhập

+  tỷ lệ chính xác của model trên tập test)

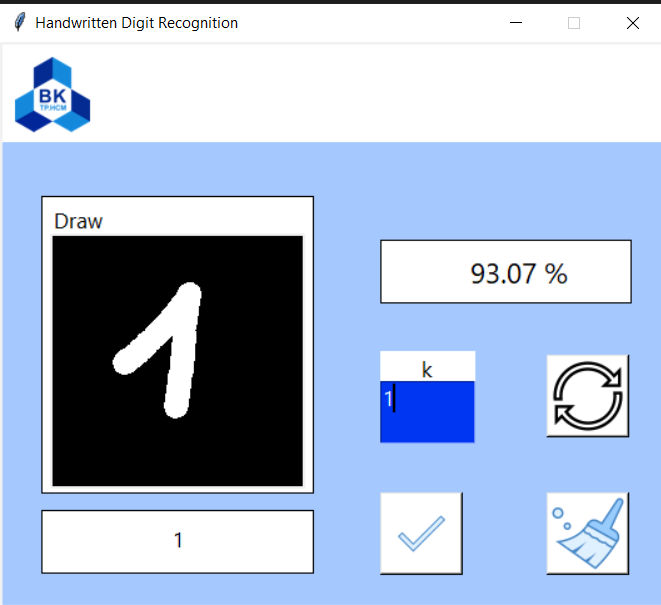
+  kiểm tra số viết tay

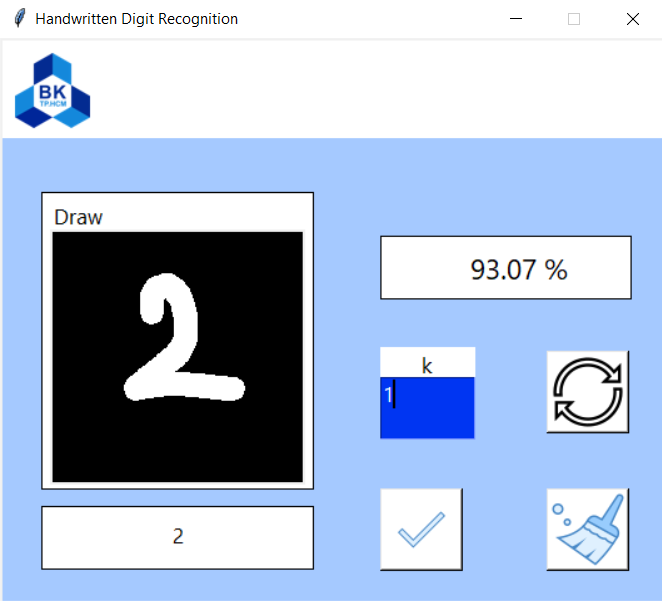
+ ( clear khung viết

+ viết chữ số cần kiểm tra tại đây)

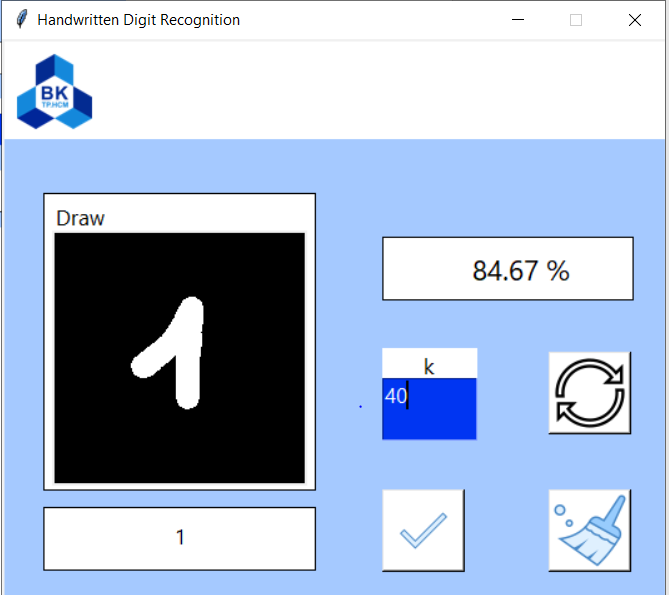
+ kết quả số nhận diện được)

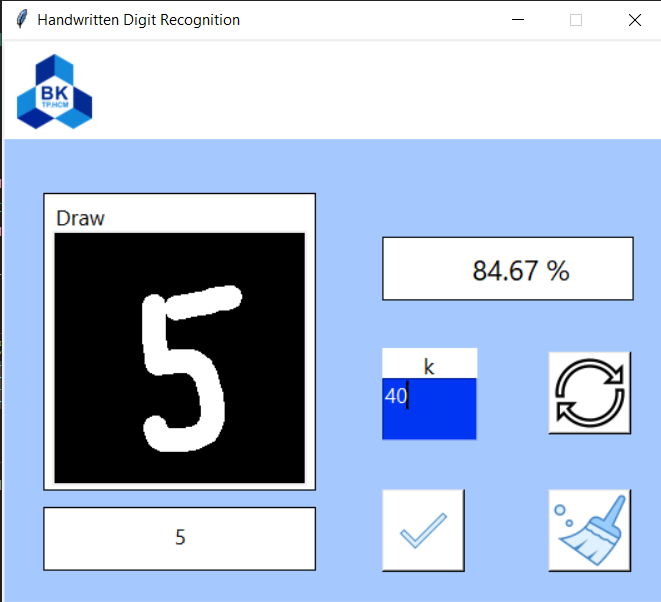
* Kiểm tra với một số trường hợp của k khác nhau:
  + Với k = 1



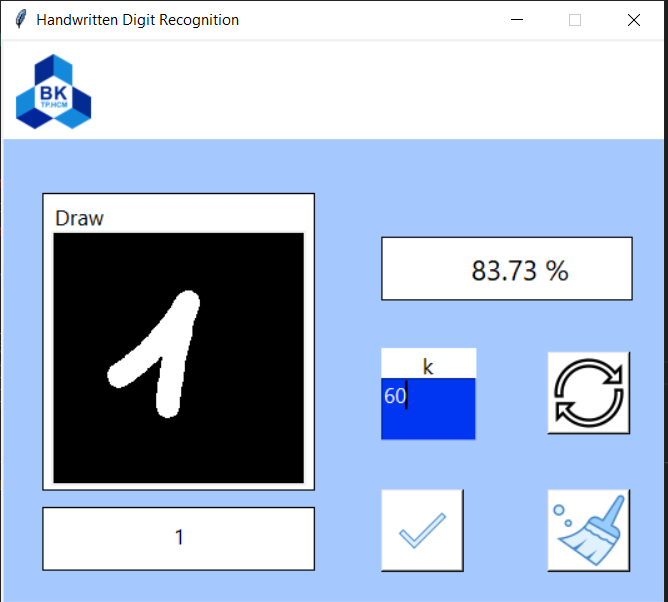


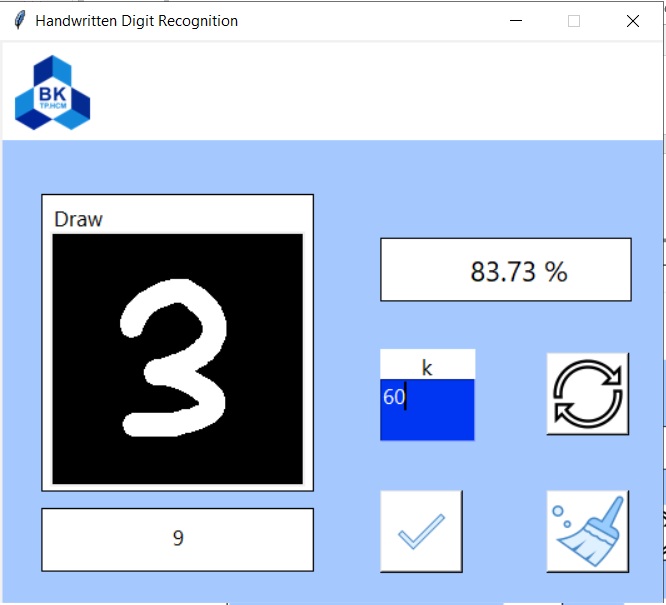
* + Với k = 40





* + Với k = 60





**Nhận xét:**

* Kết quả nhận diện chưa thật sự tối ưu, chưa có độ chính xác cao
* K càng lớn thì độ chính xác càng thấp

**CODE:**

from tkinter import \*

import tkinter

from PIL import ImageTk, Image, ImageDraw

import PIL

import numpy as np

import cv2

from matplotlib import pyplot as plt

from sklearn.cluster import KMeans

from sklearn import neighbors

from sklearn.metrics import accuracy\_score

img = cv2.imread('digits.png', 0)

cells = [np.hsplit(row, 50) for row in np.vsplit(img, 50)]

*#70% data for train 30% for test*

trainVar = np.array(cells)[:,:35].reshape(-1, 400)

testVar = np.array(cells)[:,35:50].reshape(-1, 400)

*#Labels*

k = np.arange(10)

trainLabels = np.repeat(k, 175)

testLabels = np.repeat(k, 75)

countImages = 1

kVar = 40

clf = neighbors.KNeighborsClassifier(*n\_neighbors* = kVar, *p* = 2, *weights* = 'distance')*#p = 2 the standard Euclidean distance*

def kNN():

    global trainVar, trainLabels, testVar, clf, kVar

    clf = neighbors.KNeighborsClassifier(*n\_neighbors* = kVar, *p* = 2, *weights* = 'distance')*#p = 2 the standard Euclidean distance*

    clf.fit(trainVar, trainLabels)

    y\_pred = clf.predict(testVar)

    lbl2['text'] = "%.2f %%" %(100\*accuracy\_score(testLabels, y\_pred))

def PaintNums(*event*):

    color='white'

    x1, y1 = (event.x-7), (event.y-7)

    x2, y2 = (event.x+7), (event.y+7)

    cv.create\_oval(x1, y1, x2, y2, *fill*=color, *outline*=color, *width*=5)

    draw.line([x1, y1, x2, y2], *fill*=color, *width*=5)

def CheckImage():

    global countImages, clf, image1, testVar

    filename = "Images/image" + str(countImages) + ".png"

    image1.save(filename)

    countImages += 1

    image = cv2.imread(filename, 0)

    image = cv2.resize(image, (20, 20))

    print(image)

    kNN()

    lbl1['text'] = str(clf.predict(image.reshape(-1, 400))[0])

def ClearImage():

    global image1, draw

    cv.delete("all")

    image1 = PIL.Image.new("RGB", (200, 200), "black")

    draw = ImageDraw.Draw(image1)

def SetPara():

    global kVar

    kVar = int(inputtxt1.get(1.0, END))

    print(kVar)

window = Tk()

window.title("Handwritten Digit Recognition")

window.geometry("530x450")

window.resizable(False, False)

*#add background*

canvas1 = Canvas(window, *width*=700, *height*=600, *bg*='white')

canvas1.place(*x* = 0, *y* = 0)

canvas1.create\_rectangle(0, 80, 700, 600, *fill*="#A5C9FF", *outline*="#A5C9FF")

canvas1.create\_rectangle(33, 123, 250, 360, *fill*="white", *outline*="black")

canvas1.create\_rectangle(304, 158, 504, 208, *fill*="white", *outline*="black")

canvas1.create\_rectangle(33, 374, 250, 424, *fill*="white", *outline*="black")

canvas1.create\_rectangle(325, 247, 379, 319, *fill*="white", *outline*="white")

*#add logo*

img1 = PIL.Image.open("logoBK.png")

img1 = img1.resize(((60, 60)))

photoImg = PIL.ImageTk.PhotoImage(img1)

lbl1 = tkinter.Label(window, *image*=photoImg, *bg*='white', *width*=80, *anchor*=W)

lbl1.place(*x* = 10, *y* = 10)

*#add text1: Draw*

lbl = tkinter.Label(window, *text*="Draw",*bg*='white', *fg*="black", *font*=("Segoe UI", 13), *anchor*=E)

lbl.place(*x* = 40, *y* = 128)

*#add text: k*

lblk = tkinter.Label(window, *text*="      k  ",*bg*='white', *fg*="black", *font*=("Segoe UI", 13), *anchor*=E)

lblk.place(*x* = 304, *y* = 247)

*#add draw box*

cv = Canvas(window, *width*=200, *height*=200, *bg*='black')

cv.place(*x* = 40, *y* = 153)

image1 = PIL.Image.new("RGB", (200, 200), "black")

draw = ImageDraw.Draw(image1)

cv.bind('<B1-Motion>', PaintNums)

*#add button Chech*

iB1 = PIL.Image.open("iconCheck.png")

photoImg1 = PIL.ImageTk.PhotoImage(iB1)

button1 = tkinter.Button(window, *image*=photoImg1, *command*=CheckImage, *bg* = 'white', *width*=60,*height*=60)

button1.place(*x* = 304, *y* = 360)

*#add button Clear*

iB2 = PIL.Image.open("iconClear.png")

photoImg2 = PIL.ImageTk.PhotoImage(iB2)

button2 = tkinter.Button(window, *image*=photoImg2, *command*=ClearImage, *bg* = 'white', *width*=60,*height*=60)

button2.place(*x* = 437, *y* = 360)

*#add button Set*

iB3 = PIL.Image.open("iconSet.png")

photoImg3 = PIL.ImageTk.PhotoImage(iB3)

button3 = tkinter.Button(window, *image*=photoImg3, *command*=SetPara, *bg* = 'white', *width*=60,*height*=60)

button3.place(*x* = 437, *y* = 250)

*#add text2: Number*

lbl1 = tkinter.Label(window, *text*="Number",*bg*='white', *fg*="black", *font*=("Segoe UI", 13), *anchor*=E)

lbl1.place(*x* = 135, *y* = 383)

*#add text3: %*

lbl2 = tkinter.Label(window, *text*="%",*bg*='white', *fg*="black", *font*=("Segoe UI", 17), *anchor*=E)

lbl2.place(*x* = 373, *y* = 165)

*#input Text 1*

inputtxt1 = Text(window, *height* = 2, *width* = 8,*fg*='white', *bg* = "#0035F2", *font*=("Segoe UI", 13))

inputtxt1.place(*x* = 304, *y* = 271)

window.mainloop()

1. **KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

Phương pháp dễ nhận thấy nhất trong thuật toán KNN là quan sát giữa dữ liệu mới và dữ liệu có sẵn . Vậy nên trong quá trình cải tiến KNN nhiều học giả đã chọn phương pháp này. Tuy nhiên việc làm thế nào để đạt được hiệu quả tối ưu của quá trình phân hoạch đầu tiên là vô cùng phức tạp.

Bên cạnh việc phân hoạch đầu tiên, một vấn đề khác được đặt ra: đó là việc cải thiện tốc độ thuật toán bằng việc tinh lọc quá trình tính toán để xác định vector trọng tâm và khoảng cách euclid từ các điểm đến điểm trung tâm. Các phương pháp tính hiện nay cần thay thế bằng phương pháp tính mới đơn giản hơn nhưng vẫn giữ được độ chính xác.

**Kết luận**: K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Dù vẫn còn nhiều hạn chế về khả năng tìm ra kết quả cuối cùng chính xác nhất nhưng KNN vẫn là thuật toán cơ bản nhất. Có rất nhiều thuật toán mới đã được tìm hiểu và thực sự đã khắc phục được rất nhiều nhược điểm của KNN nhưng vẫn không thể thay thế được vì độ phức tạp và cách tiếp cận đơn giản mà KNN đã sở hữu. Vậy nên có thể khẳng định, KNN là một trong nhưng cơ sở của các thuật toán đang có hiện nay. Vì thế trong xu hướng phát triển mới của các thuật toán, thay vì tìm một thuật toán mỏi hiệu quả, tập trung cải thiện những yếu điểm mà KNN gặp phải để đạt được những cải thiện về tốc độ và khả năng tìm ra lời giải tốt đang là ưu tiên. Và trên thực tế đã có rất nhiều thuật toán đã ra đời với mục đích cải thiện về tốc độ và tim ra lời giải tối ưu nhất.

**Nguồn tham khảo:**

1. <https://machinelearningcoban.com/math/#-norms-chuan>
2. <https://machinelearningcoban.com/2017/01/08/knn/#-phan-tich-toan-hoc>
3. <https://viblo.asia/p/distance-measure-trong-machine-learning-ByEZkopYZQ0>
4. Artificial Intelligencek- NearestNeighbors, Pham Viet Cuong Dept. Control Engineering &Automation, FEEEHo Chi Minh City University of Technology