***Báo cáo tuần 2+3***

***Bài 5: K-means Clustering: Simple Applications***

Note : Không dám nói code của tác giả bài viết sai nhưng code đó đã cũ , 1 số cái không còn phù hợp với các bản cập nhật hiện giờ , ngoài ra cũng có 1 số lỗi nhỏ nhưng lại khiến chương trình không chạy được , em đã sửa lại , trích dẫn và lưu ý lại hết , mong rằng bài báo cáo của em có thể giúp 1 phần nào cho thầy và các học sinh khóa sau

1. ***Phân nhóm chữ số viết tay***

*Bộ cơ sở dữ liệu MNIST :*

[Bộ cơ sở dữ liệu MNIST](http://yann.lecun.com/exdb/mnist/) là bộ cơ sở dữ liệu lớn nhất về chữ số viết tay và được sử dụng trong hầu hết các thuật toán nhận dạng hình ảnh (Image Classification).

MNIST bao gồm hai tập con: tập dữ liệu huấn luyện (training set) có tổng cộng 60k ví dụ khác nhau về chữ số viết tay từ 0 đên 9, tập dữ liệu kiểm tra (test set) có 10k ví dụ khác nhau. Tất cả đều đã được gán nhãn. Hình dưới đây là ví dụ về một số hình ảnh được trích ra từ MNIST.



Mỗi bức ảnh là một ảnh đen trắng (có 1 channel), có kích thước 28x28 pixel (tổng cộng 784 pixels). Mỗi pixel mang một giá trị là một số tự nhiên từ 0 đến 255. Các pixel màu đen có giá trị bằng 0, các pixel càng trắng thì có giá trị càng cao (nhưng không quá 255). Dưới đây là một ví dụ về chữ số 7 và giá trị các pixel của nó. (*Vì mục đích hiển thị ma trận pixel ở bên phải, tôi đã resize bức ảnh về 14x14*)

Ảnh có chứa bàn

Mô tả được tạo tự động

***Bài toán phân nhóm giả định***

***Bài toán****:* Giả sử rằng chúng ta không biết nhãn của các chữ số này, chúng ta muốn phân nhóm các bức ảnh gần giống nhau về một nhóm.

Trước khi áp dụng thuật toán [K-means clustering](https://machinelearningcoban.com/2017/01/01/kmeans/), chúng ta cần coi mỗi bức ảnh là một điểm dữ liệu. Và vì mỗi điểm dữ liệu là 1 vector (hàng hoặc cột) chứ không phải ma trận như số 7 ở trên, chúng ta phải làm thêm một bước đơn giản trung gian gọi là vectorization (vector hóa). Nghĩa là, để có được 1 vector, ta có thể tách các hàng của ma trận pixel ra, sau đó đặt chúng cạnh nhau, và chúng ta được một vector hàng rất dài biểu diễn 1 bức ảnh chữ số.

***Chú ý****: Cách làm này chỉ là cách đơn giản nhất để mô tả dữ liệu ảnh bằng 1 vector. Trên thực tế, người ta áp dụng rất nhiều kỹ thuật khác nhau để có thể tạo ra các vector đặc trưng (feature vector) giúp các thuật toán có được kết quả tốt hơn.*

### **Làm việc trên Python**

Download cơ sở dữ liệu MNIST từ web : https://web.archive.org/web/20220331130319/https://yann.lecun.com/exdb/mnist/ (hai file t10k-images-idx3-ubyte.gz và t10k-labels-idx1-ubyte.gz vì thư viện python-mnist cần cả hai file này để load dữ liệu từ tập test).

(nhiều khi bị lỗi , phải đổi dấu . ở 2 file trên thành dấu – mới chạy được)

**Trước tiên chúng ta cần khai báo một số thư viện:**

# *%reset*

*import* numpy *as* np

*from* mnist.loader *import* MNIST

*import* matplotlib

matplotlib.use('TkAgg')

*import* matplotlib.pyplot *as* plt

*from* sklearn.cluster *import* KMeans

*from* sklearn.neighbors *import* NearestNeighbors

*from* sklearn.preprocessing *import* normalize

*from* display\_network *import* \*

**Để hiện thị nhiều bức ảnh các chữ số cùng một lúc, tôi có dùng thêm hàm số**[**display\_network.py**](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/display_network.py). ( để cùng 1 thư mục với code )

**Thực hiện thuật toán K-means clustering trên toàn bộ 10k chữ số.**

**from** display\_network **import** **\***

mndata = MNIST('/home/anhvietnx1/project/Project1/MNIST') # *path to your MNIST folder*

mndata.load\_testing()  #*loads the testing images and labels from the MNIST dataset.*

X = mndata.test\_images

X0 = np.asarray(X)[:1000,:]/256.0 #*lấy 1000 mẫu đầu tiên từ X , chia 256 để chuẩn hóa về 0 or 1*

#*kích thước 1000,784 do mỗi mẫu có kích thước 1,784*

X = X0

K = 10

kmeans = KMeans(n\_clusters=K, n\_init=10).fit(X)

pred\_label = kmeans.predict(X)

print(type(kmeans.cluster\_centers\_.T))

print(kmeans.cluster\_centers\_.T.shape)

A = display\_network(kmeans.cluster\_centers\_.T, K, 1)

f1 = plt.imshow(A, interpolation='nearest', cmap = "jet")

f1.axes.get\_xaxis().set\_visible(False)

f1.axes.get\_yaxis().set\_visible(False)

plt.show()

# *plt.savefig('a1.png', bbox\_inches='tight')*

# *a colormap and a normalization instance*

cmap = plt.cm.jet

norm = plt.Normalize(vmin=A.min(), vmax=A.max())

# *map the normalized data to colors*

# *image is now RGBA (512x512x4)*

image = cmap(norm(A))

*import* imageio

image = cmap(norm(A))

image = (image \* 255).astype(np.uint8) # *convert to uint8*

imageio.imwrite('aa.png', image)

(source code có thể được tìm thấy [tại đây](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb))

Đây là kết quả của đoạn code trên

<class 'numpy.ndarray'>

(784, 10)

Đến đây, sau khi đã tìm được các center và phân nhóm dữ liệu vào từng cluster, tôi muốn hiển thị xem center trông như thế nào và các bức ảnh được phân vào mỗi cluster có giống nhau hay không. Dưới đây là kết quả khi tôi chọn ngẫu nhiên 20 bức ảnh từ mỗi cluster.

#*chon vai anh tu cluster*

print(type(pred\_label))

print(pred\_label.shape)

print(type(X0))

<class 'numpy.ndarray'>

(784, 10)

<class 'numpy.ndarray'>

(1000,)

<class 'numpy.ndarray'>

Ảnh có chứa văn bản, bàn phím, lát gạch

Mô tả được tạo tự động

Mỗi hàng tương ứng với một cluster, cột đầu tiên có nền xanh bên trái là centers tìm được của các clusters (màu đỏ hơn là các pixel có giá trị cao hơn). Chúng ta thấy rằng các center đều hoặc là giống với một chữ số nào đó, hoặc là kết hợp của hai/ba chữ số nào đó. Ví dụ: center của nhóm thứ 4 là sự kết hợp của các số 4, 7, 9; của hàng thứ 7 là kết hợp của chữ số 7, 8 và 9.

Tuy nhiên, các bức ảnh lấy ra ngẫu nhiên từ mỗi nhóm trông không thực sự giống nhau. Lý do có thể là những bức ảnh này ở xa các center của mỗi nhóm (mặc dù center đó đã là gần nhất).

Tổng kết : Code trên thực hiện việc áp dụng thuật toán KMeans để phân loại dữ liệu hình ảnh số viết tay trong tập MNIST. Sau đó, nó hiển thị các trung tâm cụm (cluster centers) trên một hình ảnh. Tiếp theo, code chọn ra 20 ảnh từ mỗi cụm và hiển thị chúng trên một hình ảnh.

Cụ thể, đoạn code thực hiện các công việc sau:

* Tải tập dữ liệu MNIST và giới hạn số lượng mẫu ở đầu tiên thành 1000 mẫu.
* Sử dụng KMeans để phân loại dữ liệu thành K cụm, ở đây K=10. Sau đó, lưu trữ nhãn dự đoán của mỗi mẫu trong biến pred\_label.
* Hiển thị các trung tâm cụm trên một hình ảnh sử dụng hàm display\_network.
* Với mỗi cụm, chọn ra 20 mẫu gần nhất đến trung tâm cụm và lưu chúng vào biến X1. Tiếp theo, chọn ra 20 mẫu đầu tiên trong cụm và lưu chúng vào biến X2.
* Hiển thị các mẫu được chọn từ X2 trên một hình ảnh.

1. ***Object Segmentation (tách vật thể trong ảnh)***

***Đặt vấn đề***

Chúng ta cùng thử áp dụng thuật toán K-means clustering vào một bài toán xử lý ảnh khác: tách vật thể.

Giả sử chúng ta có bức ảnh dưới đây và muốn một thuật toán tự động nhận ra vùng khuôn mặt và tách nó ra.

Ảnh có chứa người, trang phục, ăn mặc, trẻ

Mô tả được tạo tự động

Credit ảnh: [Trọng Vũ](https://www.facebook.com/photo.php?fbid=1219980151402370&set=a.113129725420757.13101.100001711890571&type=3&theater)

### **Lên ý tưởng**

Có 3 màu chủ đạo trong bức ảnh => 3 cluster

Bức ảnh có ba màu chủ đạo: hồng ở khăn và môi; đen ở mắt, tóc, và hậu cảnh; màu da ở vùng còn lại của khuôn mặt. Vậy chúng ta có thể áp dụng thuật toán K-means clustering để phân các pixel ảnh thành 3 clusters, sau đó chọn cluster chứa phần khuôn mặt (phần này do con người làm).

Đây là một bức ảnh màu, mỗi điểm ảnh sẽ được biểu diễn bới 3 giá trị tương ứng với màu Red, Green, và Blue (mỗi giá trị này cũng là một số tự nhiên không vượt quá 255). Nếu ta coi mỗi điểm dữ liệu là một vector 3 chiều chứa các giá trị này, sau đó áp dụng thuật toán K-means clustering, chúng ta có thể có kết quả mong muốn. Hãy thử xem

***Làm việc trên Python***

**Khai báo thư viện và load bức ảnh:**

**Link download ảnh :** <https://drive.google.com/file/d/1aSLjmDCjnASsnCysy8riIUR4eCKcSckC/view?usp=share_link>

*import* matplotlib.image *as* mpimg

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*from* sklearn.cluster *import* KMeans

img = mpimg.imread('girl3.jpg')

plt.imshow(img)

imgplot = plt.imshow(img)

plt.axis('off')

plt.show()

#*chuyển đổi kích thước của ảnh img từ (m, n, c) thành một ma trận 2 chiều X có kích thước (m \* n, c).*

# *Trong đó, m và n là chiều rộng và chiều cao của ảnh và c là số kênh màu*

X = img.reshape((img.shape[0]\*img.shape[1], img.shape[2]))

*for* K *in* [2, 5, 10, 15, 20,100]:

    kmeans = KMeans(n\_clusters=K, n\_init='auto').fit(X)

    label = kmeans.predict(X)

    # *array img4 which has the same shape as the input array X but all*

    # *its elements are initialized to zero*

    img4 = np.zeros\_like(X)

    # *replace each pixel by its center*

*for* k *in* range(K):

        img4[label == k] = kmeans.cluster\_centers\_[k]

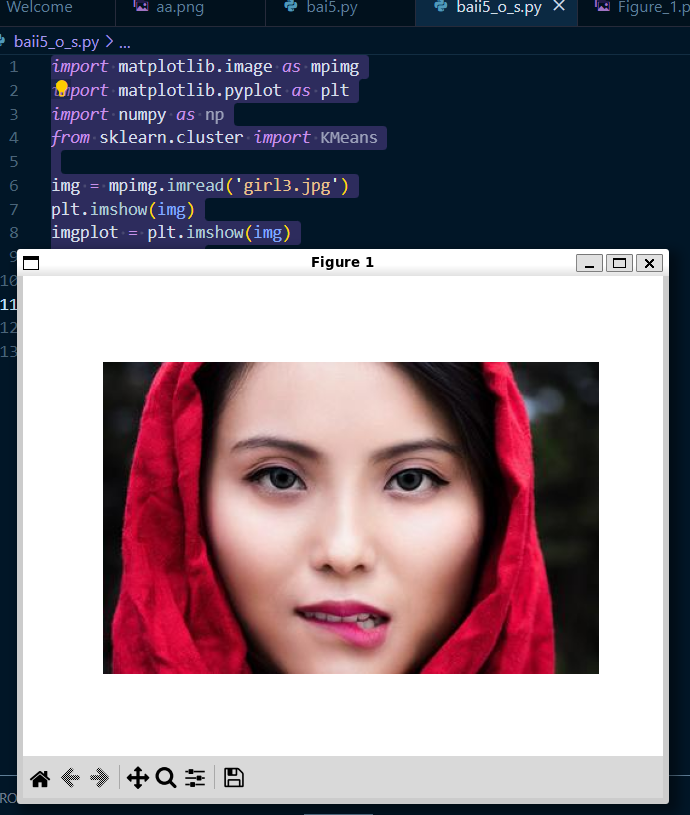
    # *reshape and display output image*

    img5 = img4.reshape((img.shape[0], img.shape[1], img.shape[2]))

    plt.imshow(img5, interpolation='nearest')

    plt.axis('off')

    plt.show()



***Tóm tắt :*** Sử dụng thư viện matplotlib để đọc ảnh và hiển thị ảnh gốc. Tiếp theo, nó chuyển đổi ảnh thành một ma trận 2 chiều X với mỗi hàng là một vector biểu diễn một điểm ảnh trong không gian RGB. Sau đó, code áp dụng thuật toán KMeans để phân cụm các điểm ảnh thành K cụm với K lần lượt là 2, 5, 10, 15, 20. Đối với mỗi giá trị K, code thay thế mỗi điểm ảnh bằng tâm của cụm tương ứng và hiển thị ảnh kết quả.

Ảnh có chứa văn bản

Mô tả được tạo tự động

***Kết quả :***

Ảnh có chứa Website

Mô tả được tạo tự độngThậm chí với k =100 , có thể thấy gần giống ảnh thật

Ảnh có chứa người, tóc, tư thế, nhìn chằm chằm

Mô tả được tạo tự động

***Kết luận :*** khi số lượng clusters tăng lên, chất lượng bức ảnh đã được cải thiện. Đồng thời, chúng ta chỉ cần lưu các centers và label của mỗi điểm ảnh là đã có được một bức ảnh nén (có mất dữ liệu).

Bài học rút ra : Có lẽ bài toán K-Clustering về phần ứng dụng này là 1 bài toán cực hay và quan trọng trong trong đời sống

***Bài 6: K-nearest neighbors***

***Giới thiệu***

*K-nearest neighbor*

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](https://en.wikipedia.org/wiki/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](https://machinelearningcoban.com/2016/12/27/categories/#classification-phan-loai) và [Regression](https://machinelearningcoban.com/2016/12/27/categories/#regression-hoi-quy). KNN còn được gọi là một thuật toán [Instance-based hay Memory-based learning](https://en.wikipedia.org/wiki/Instance-based_learning).

Có một vài khái niệm tương ứng người-máy như sau:

| **Ngôn ngữ người** | **Ngôn ngữ Máy Học** | **in Machine Learning** |
| --- | --- | --- |
| Câu hỏi | Điểm dữ liệu | Data point |
| Đáp án | Đầu ra, nhãn | Output, Label |
| Ôn thi | Huấn luyện | Training |
| Tập tài liệu mang vào phòng thi | Tập dữ liệu tập huấn | Training set |
| Đề thi | Tập dữ liểu kiểm thử | Test set |
| Câu hỏi trong dề thi | Dữ liệu kiểm thử | Test data point |
| Câu hỏi có đáp án sai | Nhiễu | Noise, Outlier |
| Câu hỏi gần giống | Điểm dữ liệu gần nhất | Nearest Neighbor |

Với KNN, trong bài toán Classification, label của một điểm dữ liệu mới (hay kết quả của câu hỏi trong bài thi) được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data có thể được quyết định bằng major voting (bầu chọn theo số phiếu) giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label. Chi tiết sẽ được nêu trong phần tiếp theo.

Trong bài toán Regresssion, đầu ra của một điểm dữ liệu sẽ bằng chính đầu ra của điểm dữ liệu đã biết gần nhất (trong trường hợp K=1), hoặc là trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.

***Một cách ngắn gọn, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách*chỉ*dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận),*không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu*. Hình dưới đây là một ví dụ về KNN trong classification với K = 1.***

Ảnh có chứa bản đồ

Mô tả được tạo tự động

Bản đồ của 1NN (Nguồn: [Wikipedia](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm))

Ví dụ trên đây là bài toán Classification với 3 classes: Đỏ, Lam, Lục. Mỗi điểm dữ liệu mới (test data point) sẽ được gán label theo màu của điểm mà nó thuộc về. Trong hình này, có một vài vùng nhỏ xem lẫn vào các vùng lớn hơn khác màu. Ví dụ có một điểm màu Lục ở gần góc 11 giờ nằm giữa hai vùng lớn với nhiều dữ liệu màu Đỏ và Lam. Điểm này rất có thể là nhiễu. Dẫn đến nếu dữ liệu test rơi vào vùng này sẽ có nhiều khả năng cho kết quả không chính xác.

Khoảng cách trong không gian vector

Trong không gian một chiều, khoảng cách giữa hai điểm là trị tuyệt đối giữa hiệu giá trị của hai điểm đó. Trong không gian nhiều chiều, khoảng cách giữa hai điểm có thể được định nghĩa bằng nhiều hàm số khác nhau, trong đó độ dài đường thằng nổi hai điểm chỉ là một trường hợp đặc biệt trong đó. Nhiều thông tin bổ ích (cho Machine Learning) có thể được tìm thấy tại [Norms (chuẩn) của vector](https://machinelearningcoban.com/math/#-norms-chuan) trong tab [Math](https://machinelearningcoban.com/math/).

2. Phân tích toán học

Thuật toán KNN rất dễ hiểu nên sẽ phần “Phân tích toán học” này sẽ chỉ có 3 câu. Tôi trực tiếp đi vào các ví dụ. Có một điều đáng lưu ý là KNN phải nhớ tất cả các điểm dữ liệu training, việc này không được lợi về cả bộ nhớ và thời gian tính toán - giống như khi cậu bạn của chúng ta không tìm được câu trả lời cho câu hỏi cuối cùng.

3. Ví dụ trên Python

Bộ cơ sở dữ liệu Iris (Iris flower dataset).

[Iris flower dataset](https://en.wikipedia.org/wiki/Iris_flower_data_set) là một bộ dữ liệu nhỏ (nhỏ hơn rất nhiều so với [MNIST](https://machinelearningcoban.com/2017/01/04/kmeans2/#bo-co-so-du-lieu-mnist). Bộ dữ liệu này bao gồm thông tin của ba loại hoa Iris (một loài hoa lan) khác nhau: Iris setosa, Iris virginica và Iris versicolor. Mỗi loại có 50 bông hoa được đo với dữ liệu là 4 thông tin: chiều dài, chiều rộng đài hoa (sepal), và chiều dài, chiều rộng cánh hoa (petal). Dưới đây là ví dụ về hình ảnh của ba loại hoa. (Chú ý, đây không phải là bộ cơ sở dữ liệu ảnh như MNIST, mỗi điểm dữ liệu trong tập này chỉ là một vector 4 chiều).



Ví dụ về Iris flower dataset (Nguồn: [Wikipedia](https://en.wikipedia.org/wiki/Iris_flower_data_set))

Bộ dữ liệu nhỏ này thường được sử dụng trong nhiều thuật toán Machine Learning trong các lớp học. Tôi sẽ giải thích lý do không chọn MNIST vào phần sau.

Thí nghiệm

Trong phần này, chúng ta sẽ tách 150 dữ liệu trong Iris flower dataset ra thành 2 phần, gọi là training set và test set. Thuật toán KNN sẽ dựa vào trông tin ở training set để dự đoán xem mỗi dữ liệu trong test set tương ứng với loại hoa nào. Dữ liệu được dự đoán này sẽ được đối chiếu với loại hoa thật của mỗi dữ liệu trong test set để đánh giá hiệu quả của KNN.

Trước tiên, chúng ta cần khai báo vài thư viện.

Iris flower dataset có sẵn trong thư viện [scikit-learn](http://scikit-learn.org/).

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** sklearn **import** neighbors, datasets

**Tiếp theo, chúng ta load dữ liệu và hiện thị vài dữ liệu mẫu**. Các class được gán nhãn là 0, 1, và 2.

iris **=** datasets.load\_iris()

iris\_X **=** iris.data

iris\_y **=** iris.target

**print** 'Number of classes: %d' **%**len(np.unique(iris\_y))

**print** 'Number of data points: %d' **%**len(iris\_y)

X0 **=** iris\_X[iris\_y **==** 0,:]

**print** '\nSamples from class 0:\n', X0[:5,:]

X1 **=** iris\_X[iris\_y **==** 1,:]

**print** '\nSamples from class 1:\n', X1[:5,:]

X2 **=** iris\_X[iris\_y **==** 2,:]

**print** '\nSamples from class 2:\n', X2[:5,:]

Number of classes: 3

Number of data points: 150

Samples from class 0:

[[ 5.1 3.5 1.4 0.2]

[ 4.9 3. 1.4 0.2]

[ 4.7 3.2 1.3 0.2]

[ 4.6 3.1 1.5 0.2]

[ 5. 3.6 1.4 0.2]]

Samples from class 1:

[[ 7. 3.2 4.7 1.4]

[ 6.4 3.2 4.5 1.5]

[ 6.9 3.1 4.9 1.5]

[ 5.5 2.3 4. 1.3]

[ 6.5 2.8 4.6 1.5]]

Samples from class 2:

[[ 6.3 3.3 6. 2.5]

[ 5.8 2.7 5.1 1.9]

[ 7.1 3. 5.9 2.1]

[ 6.3 2.9 5.6 1.8]

[ 6.5 3. 5.8 2.2]]

Nếu nhìn vào vài dữ liệu mẫu, chúng ta thấy rằng hai cột cuối mang khá nhiều thông tin giúp chúng ta có thể phân biệt được chúng. Chúng ta dự đoán rằng kết quả classification cho cơ sở dữ liệu này sẽ tương đối cao.

***Tách training và test sets***

Giả sử chúng ta muốn dùng 50 điểm dữ liệu cho test set, 100 điểm còn lại cho training set. Scikit-learn có một hàm số cho phép chúng ta ngẫu nhiên lựa chọn các điểm này, như sau:

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(

iris\_X, iris\_y, test\_size**=**50)

**print** "Training size: %d" **%**len(y\_train)

**print** "Test size : %d" **%**len(y\_test)

Training size: 100

Test size : 50

Sau đây, tôi trước hết xét trường hợp đơn giản K = 1, tức là với mỗi điểm test data, ta chỉ xét 1 điểm training data gần nhất và lấy label của điểm đó để dự đoán cho điểm test này.

clf **=** neighbors.KNeighborsClassifier(n\_neighbors **=** 1, p **=** 2)

clf.fit(X\_train, y\_train)

y\_pred **=** clf.predict(X\_test)

**print** "Print results for 20 test data points:"

**print** "Predicted labels: ", y\_pred[20:40]

**print** "Ground truth : ", y\_test[20:40]

Print results for first 20 test data points:

Predicted labels: [2 1 2 2 1 2 2 0 2 0 2 0 1 0 0 2 2 0 2 0]

Ground truth : [2 1 2 2 1 2 2 0 2 0 1 0 1 0 0 2 1 0 2 0]

Kết quả cho thấy label dự đoán gần giống với label thật của test data, chỉ có 2 điểm trong số 20 điểm được hiển thị có kết quả sai lệch. Ở đây chúng ta làm quen với khái niệm mới: ground truth. Một cách đơn giản, ground truth chính là nhãn/label/đầu ra thực sự của các điểm trong test data. Khái niệm này được dùng nhiều trong Machine Learning, hy vọng lần tới các bạn gặp thì sẽ nhớ ngay nó là gì.

***Phương pháp đánh giá (evaluation method)***

Để đánh giá độ chính xác của thuật toán KNN classifier này, chúng ta xem xem có bao nhiêu điểm trong test data được dự đoán đúng. Lấy số lượng này chia cho tổng số lượng trong tập test data sẽ ra độ chính xác. Scikit-learn cung cấp hàm số [accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) để thực hiện công việc này.

**from** sklearn.metrics **import** accuracy\_score

**print** "Accuracy of 1NN: %.2f %%" **%**(100**\***accuracy\_score(y\_test, y\_pred))

Accuracy of 1NN: 94.00 %

1NN đã cho chúng ta kết quả là 94%, không tệ! Chú ý rằng đây là một cơ sở dữ liệu dễ vì chỉ với dữ liệu ở hai cột cuối cùng, chúng ta đã có thể suy ra quy luật. Trong ví dụ này, tôi sử dụng p = 2 nghĩa là khoảng cách ở đây được tính là khoảng cách theo [norm 2](https://machinelearningcoban.com/math/#norm2). Các bạn cũng có thể thử bằng cách thay p = 1 cho [norm 1](https://machinelearningcoban.com/math/#norm0), hoặc các gía trị p khác cho norm khác. (Xem thêm [sklearn.neighbors.KNeighborsClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html))

Nhận thấy rằng chỉ xét 1 điểm gần nhất có thể dẫn đến kết quả sai nếu điểm đó là nhiễu. Một cách có thể làm tăng độ chính xác là tăng số lượng điểm lân cận lên, ví dụ 10 điểm, và xem xem trong 10 điểm gần nhất, class nào chiếm đa số thì dự đoán kết quả là class đó. Kỹ thuật dựa vào đa số này được gọi là major voting.

clf **=** neighbors.KNeighborsClassifier(n\_neighbors **=** 10, p **=** 2)

clf.fit(X\_train, y\_train)

y\_pred **=** clf.predict(X\_test)

**print** "Accuracy of 10NN with major voting: %.2f %%" **%**(100**\***accuracy\_score(y\_test, y\_pred))

Accuracy of 10NN with major voting: 98.00 %

Kết quả đã tăng lên 98%, rất tốt!

***Đánh trọng số cho các điểm lân cận***

Là một kẻ tham lam, tôi chưa muốn dừng kết quả ở đây vì thấy rằng mình vẫn có thể cải thiện được. Trong kỹ thuật major voting bên trên, mỗi trong 10 điểm gần nhất được coi là có vai trò như nhau và giá trị lá phiếu của mỗi điểm này là như nhau. Tôi cho rằng như thế là không công bằng, vì rõ ràng rằng những điểm gần hơn nên có trọng số cao hơn (càng thân cận thì càng tin tưởng). Vậy nên tôi sẽ đánh trọng số khác nhau cho mỗi trong 10 điểm gần nhất này. Cách đánh trọng số phải thoải mãn điều kiện là một điểm càng gần điểm test data thì phải được đánh trọng số càng cao (tin tưởng hơn). Cách đơn giản nhất là lấy nghịch đảo của khoảng cách này. (Trong trường hợp test data trùng với 1 điểm dữ liệu trong training data, tức khoảng cách bằng 0, ta lấy luôn label của điểm training data).

Scikit-learn giúp chúng ta đơn giản hóa việc này bằng cách gán gía trị weights = 'distance'. (Giá trị mặc định của weights là 'uniform', tương ứng với việc coi tất cả các điểm lân cận có giá trị như nhau như ở trên).

clf **=** neighbors.KNeighborsClassifier(n\_neighbors **=** 10, p **=** 2, weights **=** 'distance')

clf.fit(X\_train, y\_train)

y\_pred **=** clf.predict(X\_test)

**print** "Accuracy of 10NN (1/distance weights): %.2f %%" **%**(100**\***accuracy\_score(y\_test, y\_pred))

Accuracy of 10NN (1/distance weights): 100.00 %

Aha, 100%.

**Chú ý:** Ngoài 2 phương pháp đánh trọng số weights = 'uniform' và weights = 'distance' ở trên, scikit-learn còn cung cấp cho chúng ta một cách để đánh trọng số một cách tùy chọn. Ví dụ, một cách đánh trọng số phổ biến khác trong Machine Learning là:

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

trong đó x là test data, xi là một điểm trong K-lân cận của x, wi là trọng số của điểm đó (ứng với điểm dữ liệu đang xét x), σ là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần x thì trọng số càng cao (cao nhất bằng 1). Với hàm số này, chúng ta có thể lập trình như sau:

**def** **myweight**(distances):

sigma2 **=** .5 *# we can change this number*

**return** np.exp(**-**distances**\*\***2**/**sigma2)

clf **=** neighbors.KNeighborsClassifier(n\_neighbors **=** 10, p **=** 2, weights **=** myweight)

clf.fit(X\_train, y\_train)

y\_pred **=** clf.predict(X\_test)

**print** "Accuracy of 10NN (customized weights): %.2f %%" **%**(100**\***accuracy\_score(y\_test, y\_pred))

Accuracy of 10NN (customized weights): 98.00 %

Trong trường hợp này, kết quả tương đương với kỹ thuật major voting. Để đánh giá chính xác hơn kết quả của KNN với K khác nhau, cách định nghĩa khoảng cách khác nhau và cách đánh trọng số khác nhau, chúng ta cần thực hiện quá trình trên với nhiều cách chia dữ liệu training và test khác nhau rồi lấy kết quả trung bình, vì rất có thể dữ liệu phân chia trong 1 trường hợp cụ thể là rất tốt hoặc rất xấu (bias). Đây cũng là cách thường được dùng khi đánh giá hiệu năng của một thuật toán cụ thể nào đó.

1 vài testcase:   
20 test data points: (20 case đầu để so , ở dưới là 60 , 20 này là 20 đầu tiên)

|  |  |
| --- | --- |
| [[4.9 3.1 1.5 0.1]  [4.5 2.3 1.3 0.3]  [5.1 3.8 1.6 0.2]  [7.2 3.6 6.1 2.5]  [5.1 3.8 1.5 0.3]  [5. 3.6 1.4 0.2]  [4.6 3.6 1. 0.2]  [5.1 3.4 1.5 0.2]  [5.1 2.5 3. 1.1]  [5.7 3.8 1.7 0.3] | [5.7 2.5 5. 2. ]  [5.9 3. 4.2 1.5]  [6.3 3.4 5.6 2.4]  [5.1 3.5 1.4 0.2]  [5.4 3.4 1.7 0.2]  [5.7 2.6 3.5 1. ]  [7. 3.2 4.7 1.4]  [6.2 2.8 4.8 1.8]  [6.7 3.1 4.4 1.4]  [5.6 2.9 3.6 1.3]] |

Training size: 100

Test size : 50

Print results for 60 test data points:

Predicted labels: [0 0 0 2 0 0 0 0 1 0 2 1 2 0 0 1 1 2 1 1 0 2 2 1 1 2 2 1 2 1]

Ground truth : [0 0 0 2 0 0 0 0 1 0 2 1 2 0 0 1 1 2 1 1 0 2 2 1 1 1 2 1 2 1]

Accuracy of 1NN: 98.00 %

Accuracy of 10NN (1/distance weights): 100.00 %

Accuracy of 10NN (customized weights): 98.00 %

20 test data points:

|  |  |
| --- | --- |
| [[5.2 2.7 3.9 1.4]  [6.2 2.9 4.3 1.3]  [4.8 3.4 1.9 0.2]  [6.5 3.2 5.1 2. ]  [4.7 3.2 1.6 0.2]  [5.5 4.2 1.4 0.2]  [7.7 2.6 6.9 2.3]  [6.4 2.8 5.6 2.2]  [6.8 2.8 4.8 1.4]  [6.7 3. 5.2 2.3] | [6.2 3.4 5.4 2.3]  [5.1 3.3 1.7 0.5]  [5. 2. 3.5 1. ]  [5. 3. 1.6 0.2]  [6.7 3.3 5.7 2.1]  [5.5 2.4 3.8 1.1]  [6.3 2.7 4.9 1.8]  [6.9 3.2 5.7 2.3]  [6.4 3.1 5.5 1.8]  [5.8 2.7 4.1 1. ]] |

Training size: 100

Test size : 50

Print results for 60 test data points:

Predicted labels: [1 1 0 2 0 0 2 2 1 2 2 0 1 0 2 1 2 2 2 1 1 1 1 1 0 2 0 2 2 0]

Ground truth : [1 1 0 2 0 0 2 2 1 2 2 0 1 0 2 1 2 2 2 1 1 1 2 1 0 2 0 2 2 0]

Accuracy of 1NN: 96.00 %

Accuracy of 10NN (1/distance weights): 88.00 %

Accuracy of 10NN (customized weights): 94.00 %

***Kết luận :*** Đôi khi có thể thấy chỉ dùng 1 điểm lân cận (neighbor) thì còn được kết quả cao hơn cả dùng nhiều neighbor nữa .

***Thảo luận***

***KNN cho Regression***

Với bài toán Regression, chúng ta cũng hoàn toàn có thể sử dụng phương pháp tương tự: ước lượng đầu ra dựa trên đầu ra và khoảng cách của các điểm trong K-lân cận. Việc ước lượng như thế nào các bạn có thể tự định nghĩa tùy vào từng bài toán.

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

KNN cho bài toán Regression (Nguồn: [Nearest Neighbors regression](http://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html#sphx-glr-auto-examples-neighbors-plot-regression-py))

***Chuẩn hóa dữ liệu***

Khi có một thuộc tính trong dữ liệu (hay phần tử trong vector) lớn hơn các thuộc tính khác rất nhiều (ví dụ thay vì đo bằng cm thì một kết quả lại tính bằng mm), khoảng cách giữa các điểm sẽ phụ thuộc vào thuộc tính này rất nhiều. Để có được kết quả chính xác hơn, một kỹ thuật thường được dùng là Data Normalization (chuẩn hóa dữ liệu) để đưa các thuộc tính có đơn vị đo khác nhau về cùng một khoảng giá trị, thường là từ 0 đến 1, trước khi thực hiện KNN. Có nhiều kỹ thuật chuẩn hóa khác nhau, các bạn sẽ được thấy khi tiếp tục theo dõi Blog này. Các kỹ thuật chuẩn hóa được áp dụng với không chỉ KNN mà còn với hầu hết các thuật toán khác.

Sử dụng các phép đo khoảng cách khác nhau

Ngoài norm 1 và norm 2 tôi giới thiệu trong bài này, còn rất nhiều các khoảng cách khác nhau có thể được dùng. Một ví dụ đơn giản là đếm số lượng thuộc tính khác nhau giữa hai điểm dữ liệu. Số này càng nhỏ thì hai điểm càng gần nhau. Đây chính là [giả chuẩn 0](https://machinelearningcoban.com/math/#norm0) mà tôi đã giới thiệu trong Tab [Math](https://machinelearningcoban.com/math/).

***Ưu điểm của KNN***

Độ phức tạp tính toán của quá trình training là bằng 0.

Việc dự đoán kết quả của dữ liệu mới rất đơn giản.

Không cần giả sử gì về phân phối của các class.

Nhược điểm của KNN

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.

***Tăng tốc cho KNN***

Ngoài việc tính toán khoảng cách từ một điểm test data đến tất cả các điểm trong traing set (Brute Force), có một số thuật toán khác giúp tăng tốc việc tìm kiếm này. Bạn đọc có thẻ tìm kiếm thêm với hai từ khóa: [K-D Tree](http://pointclouds.org/documentation/tutorials/kdtree_search.php) và [Ball Tree](https://en.wikipedia.org/wiki/Ball_tree). Tôi xin dành phần này cho độc giả tự tìm hiểu, và sẽ quay lại nếu có dịp. Chúng ta vẫn còn những thuật toán quan trọng hơn khác cần nhiều sự quan tâm hơn.

Try this yourself

Tôi có viết một đoạn code ngắn để thực hiện việc Classification cho cơ sở dữ liệu [MNIST](https://machinelearningcoban.com/2017/01/04/kmeans2/#bo-co-so-du-lieu-mnist). Các bạn hãy download toàn bộ bộ dữ liệu này về vì sau này chúng ta còn dùng nhiều, chạy thử, comment kết quả và nhận xét của các bạn vào phần comment bên dưới. Để trả lời cho câu hỏi vì sao tôi không chọn cơ sở dữ liệu này làm ví dụ, bạn đọc có thể tự tìm ra đáp án khi chạy xong đoạn code này.

Enjoy!

*# %reset*

**import** numpy **as** np

**from** mnist **import** MNIST *# require `pip install python-mnist`*

*#* [*https://pypi.python.org/pypi/python-mnist/*](https://pypi.python.org/pypi/python-mnist/)

**import** matplotlib.pyplot **as** plt

**from** sklearn **import** neighbors

**from** sklearn.metrics **import** accuracy\_score

**import** time

*# you need to download the MNIST dataset first*

*# at:* [*http://yann.lecun.com/exdb/mnist/*](http://yann.lecun.com/exdb/mnist/)

mndata **=** MNIST('../MNIST/') *# path to your MNIST folder*

mndata.load\_testing()

mndata.load\_training()

X\_test **=** mndata.test\_images

X\_train **=** mndata.train\_images

y\_test **=** np.asarray(mndata.test\_labels)

y\_train **=** np.asarray(mndata.train\_labels)

start\_time **=** time.time()

clf **=** neighbors.KNeighborsClassifier(n\_neighbors **=** 1, p **=** 2)

clf.fit(X\_train, y\_train)

y\_pred **=** clf.predict(X\_test)

end\_time **=** time.time()

**print** "Accuracy of 1NN for MNIST: %.2f %%" **%**(100**\***accuracy\_score(y\_test, y\_pred))

**print** "Running time: %.2f (s)" **%** (end\_time **-** start\_time)

### **Source code**

iPython Notebook cho bài này có thể [download tại đây](https://github.com/tiepvupsu/tiepvupsu.github.io/tree/master/assets/knn/KNN.ipynb).