### Thuật toán 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).

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.

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.

## **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).



### 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.

**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.

#### **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 %

Với khoảng cách ở được tính là khoảng cách theo [norm 2](https://machinelearningcoban.com/math/#norm2).

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 %

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

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.

Scikit-learn giúp chúng ta đơn giản hóa việc này bằng cách gán gía trị weights = 'distance'

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 %

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:

**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 %