« Bài 5: K-means Clustering: Simple Applications (/2017/01/04/kmeans2/)
Bài 7: Gradient Descent (phần 1/2) » (/2017/01/12/gradientdescent/)

# Bài 6: K-nearest neighbors

KNN (/tags#KNN) Regression (/tags#Regression) Classification (/tags#Classification)
Supervised-learning (/tags#Supervised-learning) MNIST (/tags#MNIST) Iris (/tags#Iris)

*Jan 8, 2017*

Nếu như con người có kiểu học "nước đến chân mới nhảy", thì trong Machine Learning cũng có một thuật toán như vậy.

**Trong trang này:**



# 1. Giới thiệu

## Một câu chuyện vui

Có một anh bạn chuẩn bị đến ngày thi cuối kỳ. Vì môn này được mở tài liệu khi thi nên anh ta không chịu ôn tập để hiểu ý nghĩa của từng bài học và mối liên hệ giữa các bài. Thay vào đó, anh thu thập tất cả các tài liệu trên lớp, bao gồm ghi chép bài giảng (lecture notes), các slides và bài tập về nhà + lời giải. Để cho chắc, anh ta ra thư viện và các quán Photocopy quanh trường mua hết tất cả các loại tài liệu liên quan (*khá khen cho cậu này chịu khó tìm kiếm tài liệu*). Cuối cùng, anh bạn của chúng ta thu thập được một chồng cao tài liệu để mang vào phòng thi.

Vào ngày thi, anh tự tin mang chồng tài liệu vào phòng thi. Aha, đề này ít nhất mình phải được 8 điểm. Câu 1 giống hệt bài giảng trên lớp. Câu 2 giống hệt đề thi năm ngoái mà lời giải có trong tập tài liệu mua ở quán Photocopy. Câu 3 gần giống với bài tập về nhà. Câu 4 trắc nghiệm thậm chí cậu nhớ chính xác ba tài liệu có ghi đáp án. Câu cuối cùng, 1 câu khó nhưng anh đã từng nhìn thấy, chỉ là không nhớ ở đâu thôi.

Kết quả cuối cùng, cậu ta được 4 điểm, vừa đủ điểm qua môn. Cậu làm chính xác câu 1 vì tìm được ngay trong tập ghi chú bài giảng. Câu 2 cũng tìm được đáp án nhưng lời giải của quán Photocopy sai! Câu ba thấy gần giống bài về nhà, chỉ khác mỗi một số thôi, cậu cho kết quả giống như thế luôn, vậy mà không được điểm nào. Câu 4 thì tìm được cả 3 tài liệu nhưng có hai trong đó cho đáp án A, cái còn lại cho B. Cậu chọn A và được điểm. Câu 5 thì không làm được dù còn tới 20 phút, vì tìm mãi chẳng thấy đáp án đâu - nhiều tài liệu quá cũng mệt!!

Không phải ngẫu nhiên mà tôi dành ra ba đoạn văn để kể về chuyện học hành của anh chàng kia. Hôm nay tôi xin trình bày về một phương pháp trong Machine Learning, được gọi là K-nearest neighbor (hay KNN), một thuật toán được xếp vào loại lazy (machine) learning (máy lười học). Thuật toán này khá giống với cách học/thi của anh bạn kém may mắn kia.

# 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 (/2016/12/27/categories/#classification-phan-loai) và Regression (/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:

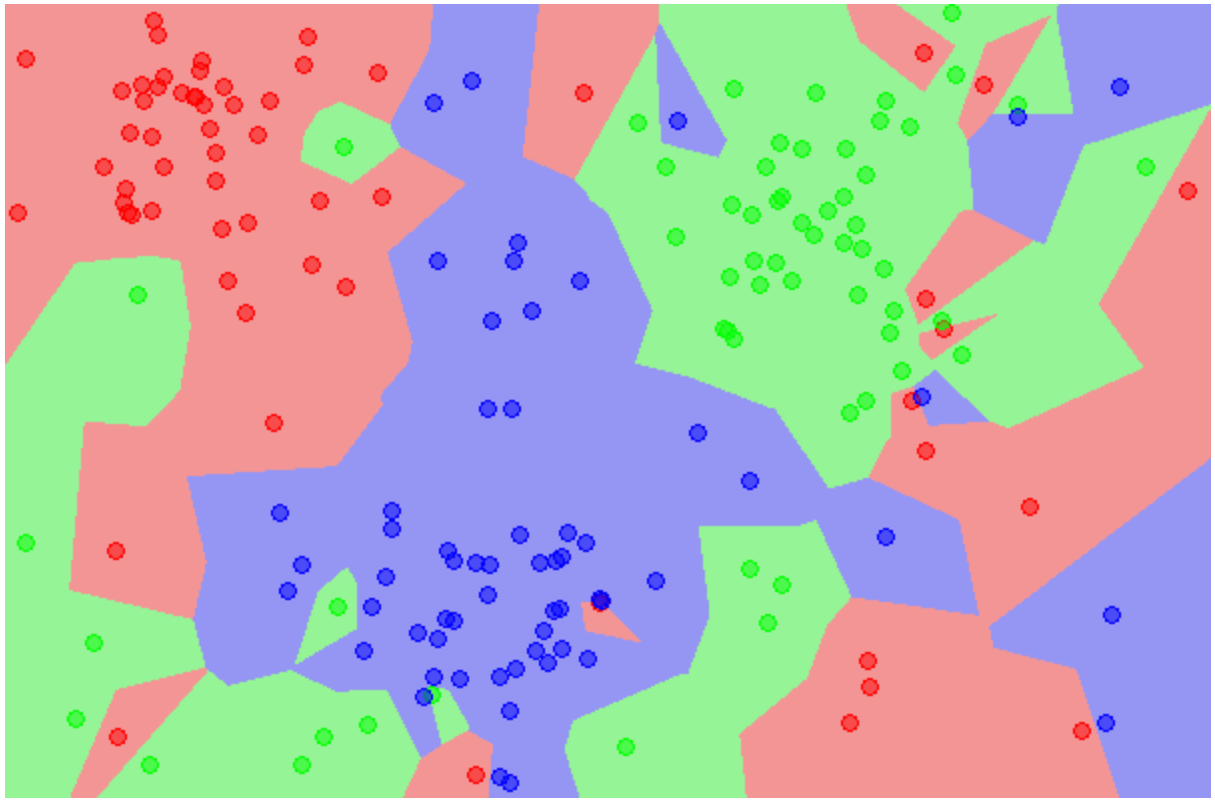
| **Ngôn ngữ người** | **Ngôn ngữ Máy Học** | **in Machine Learning** |
|---|---|---|
| Câu hỏi | Điểm dữ liệu | Data point |
| Đáp án | Đầu ra, nhãn | Output, Label |

| Ngôn ngữ người | Ngôn ngữ Máy Học | in Machine Learning |
|---|---|---|
| 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.



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 thẳng nối 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 (/math/#-norms-chuan) trong tab Math (/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 (/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).

Iris setosa Iris versicolor Iris virginica

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 (/math/#norm2). Các bạn cũng có thể thử bằng cách thay `p = 1` cho norm 1 (/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_p
```

```
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:

$$w_i = \exp\left(\frac{-||\mathbf{x} - \mathbf{x}_i||_2^2}{\sigma^2}\right)$$

trong đó $\mathbf{x}$ là test data, $\mathbf{x}_i$ là một điểm trong K-lân cận của $\mathbf{x}$, $w_i$ là trọng số của điểm đó (ứng với điểm dữ liệu đang xét $\mathbf{x}$), $\sigma$ là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần $\mathbf{x}$ 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_p
```

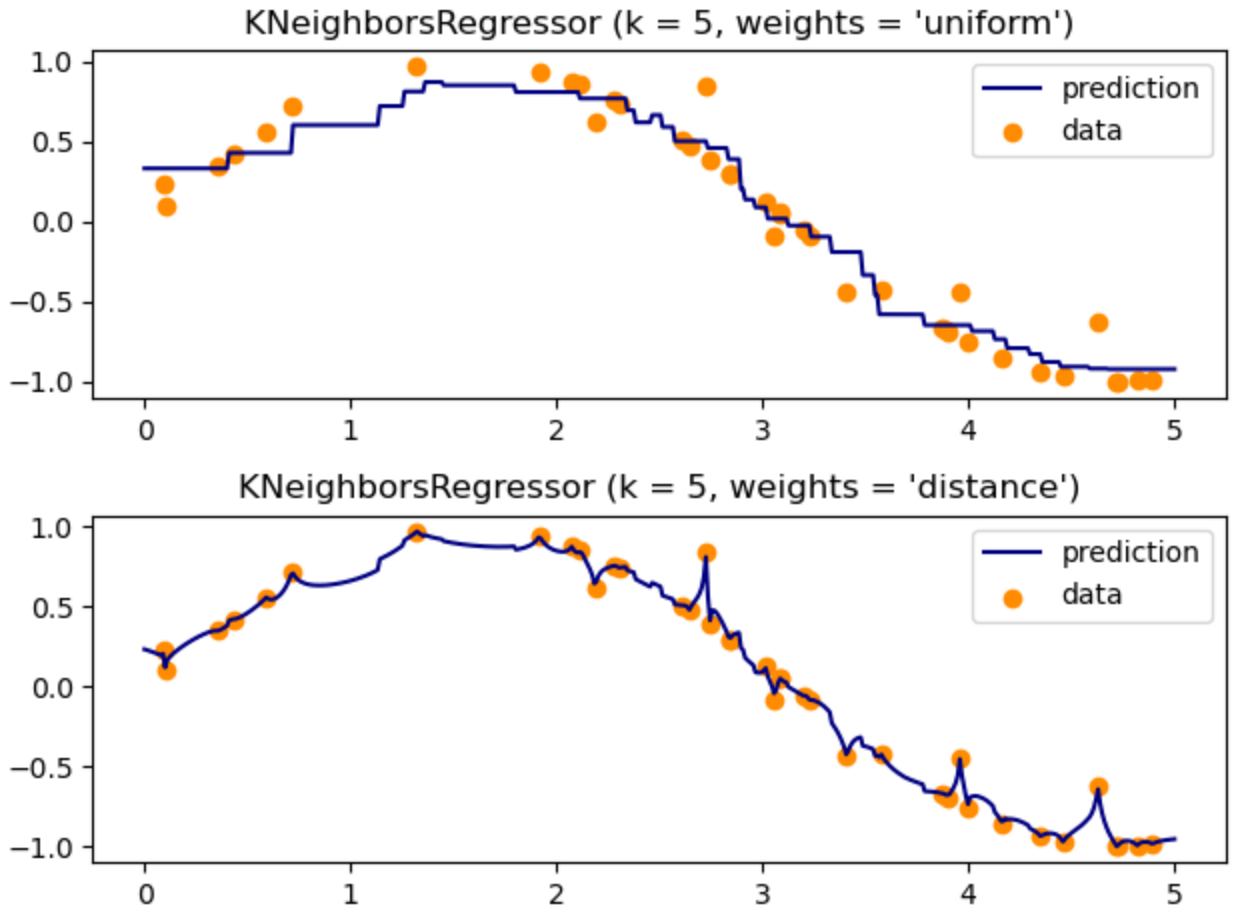
```
Accuracy of 10NN (customized weights): 98.00 %
```

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.

# 4. 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.




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 (/math/#norm0) mà tôi đã giới thiệu trong Tab Math (/math/).

## Ưu điểm của KNN

1. Độ phức tạp tính toán của quá trình training là bằng 0.
2. Việc dự đoán kết quả của dữ liệu mới rất đơn giản.
3. Không cần giả sử gì về phân phối của các class.

## Nhược điểm của KNN

1. KNN rất nhạy cảm với nhiễu khi K nhỏ.
2. 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 thể 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 (/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/

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/
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).

# 5. Tài liệu tham khảo


1. sklearn.neighbors.NearestNeighbors (http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors)
2. sklearn.model_selection.train_test_split (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
3. Tutorial To Implement k-Nearest Neighbors in Python From Scratch (http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/)

*Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (https://www.facebook.com/groups/257768141347267/) để nhận được câu trả lời sớm hơn.*
*Bạn đọc có thể ủng hộ blog qua 'Buy me a cofee' (/buymeacoffee/) ở góc trên bên trái của blog.*
*Tôi vừa hoàn thành cuốn ebook 'Machine Learning cơ bản', bạn có thể đặt sách tại đây (/ebook/).*
*Cảm ơn bạn.*

---

« Bài 5: K-means Clustering: Simple Applications (/2017/01/04/kmeans2/)
Bài 7: Gradient Descent (phần 1/2) » (/2017/01/12/gradientdescent/)

---

Total visits: