Part 1: KNN Notes

Authors
Stephanie Dinh
Ying-Yu Huang
Patricia Sieng

K-Nearest Neighbors (KNN) is a non-parametric technique which uses k, the number of nearest neighbors, to classify data into groups. The goal of KNN classification is to automatically assign a "class" of features to each case. It primarily works by implementing the following steps.

First, a positive integer K is chosen along with a test observation x_0. The classifier then calculates the distances of the surrounding points and finds the k closest points to that observation. Finally, the class is chosen based on the majority class of the k points.

The choice of *k* is very important in KNN. In the simplest case, if you have *k* = 1, then it means that your model will be classified to the class of the single nearest neighbor. If the K is too small, the classification of a case can become very sensitive and dependent on the classification of its closest neighbor. Reversely, if the k is too large, it can introduce bias into the classification decision, even though it reduces the variability. In the most extreme case where k is equal to the entire data set, each case would be classified into its own class, which is not very helpful in analysis.

Each case is described by a vector of features $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_p \end{bmatrix}$, with p features

The # of classes = r
Classes = CL1,CL2….,CLr
Cj = group of Nj cases, where j=1….r

Given case x which is described by p features, we assume that x belongs to class Cj

according to an “expert”. The expert here is not necessarily someone who has extensive knowledge, it can just be a normal person with the capacity to classify and categorize various “cases” or observations. We collect these cases in a “training set” where all the true classes have been designated. Ideally the training set would be very large in order to gather a lot of information, but the draw back in reality is the cost of time and money.

Example1:

Given the training set is a list of cases, case 1 … case N, where N is sufficiently large, say 100,000. Each case in our training set has the true classification y1,…,yN, where y is the true class, ‘good’, ‘ok’, or ‘bad’.

3 classes

	good	ok	bad
Name	C1	C2	C3
Size	n1	n2	n3

The sizes correspond to the number of cases in the training set. As the classes are defined by descriptive names, we may need to assign numerical numbers to clean the data.

Once the training set is obtained, the goal is to build an intelligent program to understand how the decision of the classification was reached.

Using machine learning, a program, software, or algorithm can be built which scans the training set information, and then generates a smart decision-making software, for example a Blackbox. The Blackbox will take the input and compute the number of the class it believes the case belongs to.

Evaluate performance

The Blackbox may be right or wrong in its classification, so to test the quality of the algorithm, we need to use new cases, case N+1, N+2,…,N+1000, for which we know the true answers y(N+1)……y(N+1000). Of our N cases in the training set, we reserve a portion to test the quality of our algorithm. These M cases will be our test set, where M<N, and generally M is 20% of N.

Applying the Blackbox on test set cases x(1), x(2),…,X(M), gives us the Blackbox decisions BB1, BB2, …, BBM, which will be the numbers 1,2, or 3, corresponding to the classes.

As we have the true answers (y1,y2…,yM), we need to see how often the algorithm

correctly classified the cases. For example, the Blackbox will give the correct answer for case 117 if $BB117 = y117$, and false answer if $BB117 \neq y117$. This can be more easily seen by creating a confusion matrix.

Confusion matrix

True class / BB class	1	2	3
1	Z_{11}	Z_{12}	Z_{13}
2	Z_{21}	Z_{22}	Z_{23}
3	Z_{31}	Z_{32}	Z_{33}

- Z_{11} - number of cases in the test set:{true class=1, BB class=1} → correct classification
- Z_{12} - number of cases in the test set:{true class=1, BB class=2} → wrong classification

The sum the first row $Z11 + Z12 + Z13 = q1$ is the number of cases of class1 in test set. q2 and q3 can be similarly calculated for the amount of class 2 and class 3 cases in the test set respectively.

A strong diagonal Z_{11}, Z_{22}, Z_{33} indicates a good confusion matrix as it tells us to what extent the algorithm was able to correctly classify the cases.

Calculating Z_{11}/q_1 gives us z_{11}, the percentage of class 1 cases the Blackbox correctly classified. Similar calculations can be done for the correct classification of the other two classes, as well as percentages for incorrect classifications.

These percentages are not only important in evaluating the performance of our algorithm, but also gives us a clearer view as to the probability of a mistake being made. We must carefully weigh that probability against the real-world consequences, as mistakes (especially in a medical context) can be damaging and costly.

Example2: Automatic Classification Algorithm

Given the training set, case(1)….case(N), we want to classify each case as good(1) or bad(0), so we have only two classes. The true classes are denoted y1,…,yN, and so are equal to 0 or 1. We are assuming each case is described by vector of features

$X=\begin{bmatrix} x_{(1)} \\ \vdots \\ \vdots \\ x_{(p)} \end{bmatrix}$. For each new case $z=\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ \vdots \\ z_{10} \end{bmatrix}$, we want to use our algorithm to automatically classify the case.

Step 1:

To start KNN, we first must fix the parameter k to some positive integer. The algorithm will change depending on what k we choose. In this example, we fix k, the number of neighbors, to 15.

Step 2:

Compute the distance between new case z and the old case X1

$$\|z - X1\| = d(z,1) = \sqrt{\left(z_1 - x(1)\right)^2 + \cdots \left(z_{10} - x(10)\right)^2},$$

then repeat the computation for d(z,2)…d(z,N).

The general formula for the distance between the vectors:

$$d(z,j) = \|z - Xj\| = \sqrt{\left(z_1 - x1(j)\right)^2 + \cdots \left(z_p - xp(j)\right)^2}.$$

Step 3:

The distances are then sorted d1,…,dN in increasing order. Since we want the closest points and have set k to 15, we take the first 15 distances and their associated cases.

Step 4:

Of these 15 closest neighbors of z, we can score the classes by counting the number of "good" cases $sc1$, and the number of "bad" cases $(sc2 = 15 - sc1)$. If $sc1 > sc2$, we decide that z is in the "good" class, and vice-versa if $sc1 < sc2$, z will be classified into the "bad" class.

We can also take the score and divide it by the number of neighbors to find the probability that the associated class is the true class. For example, if sc1 is 10, then the probability that the true class is "good" is $\frac{10}{15}$. We also call this the reliability.

Step 5:

To gauge how well our algorithm does, we can compute the performance percentage on the test set, or the percentage that was correctly classified. Different numbers of k will change the performance of our classification. As the relation of the performance to the number k is not monotonic, we can try various k's and keep the highest performing k.

Weights

*One of the weaknesses of the KNN method is that it gives equal importance to all

features. To solve this issue, we can introduce coefficients or weights to the features

$$0 < W1 < W2 < \cdots < Wp,\ W1 + W2 + \cdots + Wp = 1.$$

Weighted distance:

$$Dj(z) = W1\ \left(z_1 - x1(j)\right)^2 + W2\left(z_2 - x2(j)\right)^2 + \cdots . + Wp\left(z_p - xp(j)\right)^2$$

If all features are equally important, we can use ordinary weights, in which

$$W1 = 1 \ldots Wp\ \ = \ \ 1.$$

There are various bootstrap methods to discover the weights of the features:
The first of which is focusing on one feature f, and temporarily discarding the rest. Keeping a good number k (as found in part 5 of the previous example) we apply KNN. If we want to describe case j, $Y(j) = X_f(j)$. Applying KNN on the training set will give us a performance percentage with feature f, since we had removed all other features. $W_f/(W_1 + \cdots + W_p)$
The second method is done by discarding one feature at a time and comparing the performance of all features to the performance of one eliminated feature. For example, if a feature is removed and the performance had little change, we can assume the feature was not important. This is one method to reduce a large amount features which are not helpful.

Pre-Treatment of Data

The first task is to change variables from qualitative to quantitative, as well as "discrete" features/characteristics to "continuous".

Example 3:

We may have features, for instance, which explain sociological data, medical data, or in this example various professions.

$$profession\{pro1, \ldots, pro23\},$$

Here pro# is non-numeric and names the 23 different profession. We cannot just arbitrarily assign numbers to these professions, as there is no way to determine how far away one profession should be from another. In order to be impartial, we need to increase the dimension of the data. We need to transform each of the 23 features into numerical features, which will be binary vectors of unit length.
Profession 2 and 4 for example, will become

$$pro2\ \ = \ \ [0100 \ldots \ldots \ldots 0], pro4\ \ [000100 \ldots \ldots \ldots 0], and\ \ so\ \ on.$$

Now we are able to accurately compute the distance between two binary vectors,

$$\|\mathrm{pro2} - \mathrm{pro4}\|^2 = 2,\ \ \|\mathrm{pro2} - \mathrm{pro4}\| = \sqrt{2}.$$

The distance between any two profession will be 2. This is the method to replace the

qualitative data to quantitative. Once all the features have been transformed into numerical features, now we have features $\mathrm{x} = \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \vdots \\ \vdots \\ \mathrm{x}_N \end{bmatrix}$ for each case. With these numerical features, we need to normalize data to account for the difference in units. We compute the average value of x,

$$\bar{x} = \frac{x(1)+x(2)+\cdots+x(N)}{N},$$

then we take a look at the centered coordinates

$(x(j) - \bar{\mathrm{x}})$, which we will call $(y(1) \ldots . y(N))$.

Now we can compute the standard deviation:

$$std1 = \sqrt{var\big(y(1)\big)}, \ldots, stdN = \sqrt{var\big(y(N)\big)}.$$

Lastly, we can use this to rescale the data.

X(1) will be replaced by $\frac{x(1)}{std(1)}$, X(2) by $\frac{x(2)}{std(2)}, \ldots, and \;\; X(N) \;\; by \frac{x(N)}{std(N)}$.

It's always good practice to carefully examine the data and decide whether it would be beneficial to center the information, as not all situations will return better results with centering the data.