Classification: k Nearest Neighbours

1 Introduction

Recall that in classification problems, the task is to decide to which of a predefined, finite set of categories an object belongs.

In this lecture, we look at another classification method. The method we look at is known as the k-*nearest-neighbours* or kNN method.

Just as with the probabilistic methods that we looked at in the previous two lectures, we need a dataset of examples. Each example describes an instance and gives the class to which it belongs. As before, we'll assume instances are described by a set of attribute-value pairs, and there is a finite set of class labels L. So the dataset comprises examples of the form $\langle\{A_1 = a_1, A_2 = a_2, \ldots, A_n = a_n\}, \mathit{class} = cl\rangle$. For a particular instance x, we will refer to x's value for attribute A_i as $x.A_i$.

In the probabilistic methods that we looked at, the learning step involved computing probabilities from the dataset. Once this was done, in principle the dataset could be thrown away; classification was done using just the probabilities.

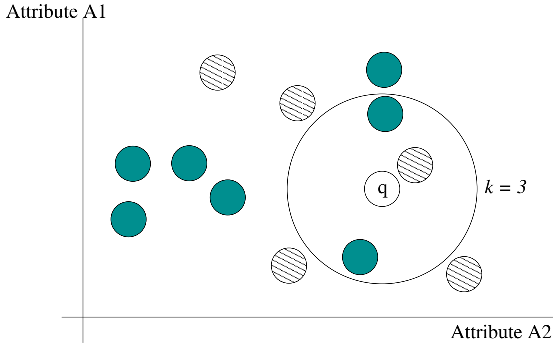
In k-nearest-neighbours, the learning step is trivial: we simply store the dataset in the system's memory. Yes, that's it!

In the classification step, we are given an instance q (the query), whose attributes we will refer to as $q.A_i$ and we wish to know its class. In kNN, the class of q is found as follows:

1. Find the k instances in the dataset that are closest to q.
2. These k instances then vote to determine the class of q.

Ties (whether they arise when finding the closest instances to q or when voting for the class of q) are broken arbitrarily.

In the following visualisation of this method, we assume there are only two attributes A_1 and A_2, and two different classes (where circles with solid fill represent instances in the dataset of one class and circles with hashed fill represent instances in the dataset of the other class). Query q is here being classified by its 3 nearest neighbours:



All that remains to do is discuss how distance is measured, and how the voting works.

2 Distance

2.1 Local distance functions, global distance functions and weights

A *global distance function*, dist, can be defined by combining in some way a number of *local distance functions*, dist_A, one per attribute.

The easiest way of combining them is to *sum* them:

$$\mathrm{dist}(x, q) =_{\text{def}} \Sigma_{i=1}^{n} \mathrm{dist}_{A_i}(x.A_i, q.A_i)$$

More generally, the global distance can be defined as a *weighted sum* of the local distances. The weights w_i allow different attributes to have different importances in the computation of the overall distance. Weights sometimes lie between zero and one; a weight of zero would indicate a totally irrelevant attribute.

$$\mathrm{dist}(x, q) =_{\text{def}} \Sigma_{i=1}^{n} w_i \times \mathrm{dist}_{A_i}(x.A_i, q.A_i)$$

A *weighted average* is also common:

$$\mathrm{dist}(x, q) =_{\text{def}} \frac{\Sigma_{i=1}^{n} w_i \times \mathrm{dist}_{A_i}(x.A_i, q.A_i)}{\Sigma_{i=1}^{n} w_i}$$

The weights can be supplied by the system designer. There are also ways of learning weights from a dataset.

What we haven't discussed is how to define the local distance functions. We will exemplify the different definitions by computing the distance between query q and x_1 and x_2 below:

x_1

sex	*female*
weight	60
amount	4
meal	*full*
duration	90
class	*over*

x_2

sex	*male*
weight	75
amount	2
meal	*full*
duration	60
class	under

q

sex	*male*
weight	70
amount	1
meal	*snack*
duration	30
class	?

The attributes and their values are: sex (male/female), weight (between 50 and 150 inclusive), amount of alcohol consumed in units (1-16 inc.), last meal consumed today (none, snack or full meal), and duration of drinking session (20-320 minutes inc.). The classes are: over or under the drink driving limit.

2.2 Hamming distance

The easiest local distance function, known as the *overlap function*, returns 0 if the two values are equal and 1 otherwise:

$$\mathrm{dist}_A(x.A, q.A) =_{\text{def}} \begin{cases} 0 & \text{if } x.A = q.A \\ 1 & \text{otherwise.} \end{cases}$$

If the global distance function is defined as the sum of the local distances, then we're simply counting the number of attributes on which the two instances disagree. This is called *Hamming distance*. Weighted sums and weighted averages are also possible.

Class Exercise. What is the Hamming distance between x_1 and q and between x_2 and q?

2.3 Manhattan distance for numeric attributes

If an attribute is numeric, then the local distance function can be defined as the *absolute difference* of the values:

$$\text{dist}_A(x.A, q.A) =_{\text{def}} |x.A - q.A|$$

If the global distance is computed as the sum of these local distances, then we refer to it as the *Manhattan distance*. Weighted sums and weighted averages are also possible.

One weakness of this kind of scheme is that if one of the attributes has a relatively large range of possible values, then it can overpower the other attributes. In the example, this might be the case with *duration* relative to *weight* and *amount*. Therefore, local distances are often *normalised* so that they lie in the range $0 \ldots 1$. There are many ways to normalise, some being better-motivated than others. For simplicity, we will look at only one. We will divide by the range of permissible values:

$$\text{dist}_A(x.A, q.A) =_{\text{def}} \frac{|x.A - q.A|}{A_{max} - A_{min}}$$

where A_{max} is the largest possible value for attribute A, and A_{min} is its smallest possible value. We'll call this the *range-normalised absolute difference*.

The other weakness of this scheme, of course, is that it can be used only on numeric attributes.

2.4 Heterogeneous local distance functions

We can combine absolute distances and the overlaps in order to handle both numeric and symbolic attributes:

$$\text{dist}_A(x.A, q.A) =_{\text{def}} \begin{cases} \frac{|x.A - q.A|}{A_{max} - A_{min}} & \text{if } A \text{ is numeric} \\ 0 & \text{if } A \text{ is symbolic and } x.A = q.A \\ 1 & \text{otherwise.} \end{cases}$$

As usual, the global distance can be computed as a sum, weighted sum or weighted average of the local distances.

Class Exercise. Taking a sum of local distances defined heterogeneously, what is the distance between x_1 and q?

2.5 Knowledge-intensive distance functions

Human experts can sometimes define domain-specific local distance functions, especially for symbolic-valued attributes. In this way, they can bring their prior knowledge to bear.

A simple but common example is when there is already some total ordering defined over the values of the symbolic attribute. For example, the last meal a person ate has values *none*, *snack* and *full*. These can be thought of as totally ordered by the amount of food consumed:

$$\textit{none} < \textit{snack} < \textit{full}$$

We can assign integers to the values in a way that respects the ordering: *none* $= 0$, *snack* $= 1$ and *full* $= 2$. Now, we can use a range-normalised absolute difference function on these integers. So, for example,

$$\begin{aligned} \text{dist}_{\textit{meal}}(\textit{none}, \textit{snack}) &= \frac{|0 - 1|}{2 - 0} = 0.5 \\ \text{dist}_{\textit{meal}}(\textit{none}, \textit{full}) &= \frac{|0 - 2|}{2 - 0} = 1 \end{aligned}$$

Class Exercise. Let the weights for *sex*, *weight*, *amount*, *meal* and *duration* be 4, 3, 5, 1 and 2 respectively. Taking a weighted sum of local distances defined heterogeneously, where $\text{dist}_{\textit{meal}}$ is defined as above, what is the distance between x_2 and q?

To explore this idea a bit further, we'll invent another attribute, one that wasn't present in the original dataset, *type*, with values $\{\textit{lager}, \textit{stout}, \textit{whiteWine}, \textit{redWine}, \textit{whisky}, \textit{vodka}\}$, to denote the main kind of beverage the person has been drinking. We'll look at a variety of ways of defining some knowledge-intensive local distance functions for this attribute.

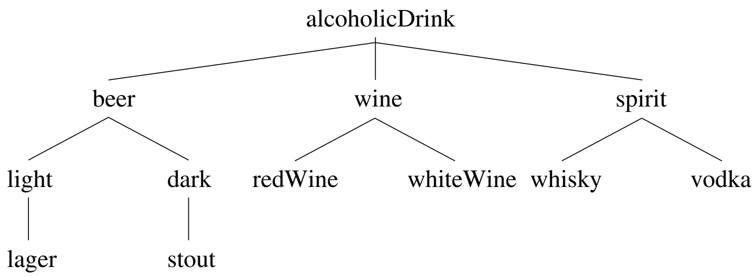
One possibility, of course, is to define a total ordering, based perhaps on alcohol content, e.g.:

$$\textit{lager} < \textit{stout} < \textit{whiteWine} < \textit{redWine} < \textit{whisky} < \textit{vodka}$$

and then to assign consecutive integers and compute a range-normalised absolute difference. Hence, e.g.,

$$\text{dist}_{\textit{type}}(\textit{stout}, \textit{whisky}) = \frac{|1 - 4|}{5 - 0} = 0.6$$

Another possibility is to define a *taxonomy* (class hierarchy) of beverage types, such as the following:



Based on distances in this tree, we can again compute range-normalised distances. For example,

$$\text{dist}_{\textit{type}}(\textit{stout}, \textit{whisky}) = \frac{5}{5 - 0} = 1.0$$

The final approach that we will consider is simply to enumerate all the distances in a matrix. This allows the designers to use whatever distances they feel make sense in their domain. Note that, since the diagonal of the matrix will represent the distance between a value and itself, it should be filled in with zeros. And, assuming that distance is symmetric, the lower triangle will be a reflection along this diagonal of the upper triangle.

We have looked at only three ways of defining a distance function for *type*. For other attributes there could be numerous other domain-specific approaches.

3 Voting

Once we have obtained q's k-nearest-neighbours using the distance function, it is time for the neighbours to vote in order to predict q's class. Two approaches are common.

Majority voting: In this approach, all votes are equal.

For each class $cl \in L$, we count how many of the k neighbours have that class. We return the class with the most votes.

Inverse distance-weighted voting: In this approach, closer neighbours get higher votes.

While there are better-motivated methods, the simplest version is to take a neighbour's vote to be the inverse of its distance to q:

$$vote(x_i) =_{\text{def}} \begin{cases} \infty & \text{if } \mathrm{dist}(x_i, q) = 0 \\ \frac{1}{\mathrm{dist}(x_i,q)} & \text{otherwise} \end{cases}$$

Then we sum the votes and return the class with the highest vote.

Class Exercise. Suppose $k = 3$ and q's 3-nearest-neighbours from the dataset are instances x_7, x_{35} and x_{38}. (For conciseness, I won't show their attribute-values — they aren't needed at this step anyway.) Here are their classes and the distances we computed:

neighbour	*class*	dist
x_7	*under*	0.2
x_{35}	*over*	0.5
x_{38}	*over*	0.4

What is q's predicted class using (a) majority voting, and (b) inverse distance-weighted voting?

There are numerous other voting schemes. Human experts might even define a domain-specific scheme. For example, in spam-filtering, the cost of misclassifying ham as spam (when legitimate email ends up in your spam folder) is higher than the cost of misclassifying spam as ham (when spam ends up in your in-box). A domain-specific voting scheme might be defined to skew the classifier away from the former kind of error. For example, we could predict *class* = *spam* if all k neighbours vote *spam*; otherwise, we would predict *class* = *ham*.

4 Using k-Nearest-Neighbours for Other Tasks

We've been looking at classification. But k-nearest-neighbours is useful for other tasks too.

4.1 Regression

In classification, there is a finite set of class labels L from which we must choose. In regression, the task is, given the description of an object, to predict a real value.

Examples abound: from meteorological data, predict tomorrow's rainfall; from information about a software project, predict how long it will take to complete; from information about stock market movements, predict the value that my shares will have by close of business tomorrow.

In regression, the dataset comprises examples of the form $\langle\{A_1 = a_1, A_2 = a_2, \ldots, A_n = a_n\}, value = r\rangle$, where r is a real number. The only part of k-nearest-neighbours that needs changing is the voting.

In other words, again we find the k instances in the dataset that are closest to q. Now we must use their *value*s to predict q's *value*. While there may be better-motivated approaches, the simplest approach is to take the mean of the neighbours' *value*s.

Class Exercise. Suppose our drinkers dataset no longer contains a *class* for each instance (*over* or *under*). Instead, it specifies the blood alcohol content (BAC) of the person, as measured by a breathalyser. This will be a real in the range $0 \ldots 100$ (a percentage).

Suppose $k = 3$ and q's 3-nearest-neighbours from the dataset are instances x_7, x_{35} and x_{38}. Here are their BACs and the distances we computed:

neighbour	*BAC*	dist
x_7	20	0.2
x_{35}	60	0.5
x_{38}	40	0.4

What is q's predicted BAC?

Don't go away with the idea that regression is something that kNN methods can do that naïve Bayesian methods cannot: it is just as possible to apply naïve Bayesian methods to regression problems. In fact, neither is likely to be the best method to use for regression!

4.2 Product recommendation

If there is time in the lecture, we will see how kNN can be used within product recommender systems.

Exercise

The *Pants Pizza Parlour* sells pizzas with optional toppings: pepperoni, pineapple and pickled onion. Every day this week you have tried a pizza (A to E) and kept a record of which you liked:

	pepperoni	*pineapple*	*pickledOnion*	*liked*
A	true	true	true	false
B	true	false	false	true
C	false	true	true	false
D	false	true	false	true
E	true	false	false	true

1. Show how the naïve Bayes classifier would classify $\{$*pepperoni* = true, *pineapple* = true, *pickledOnion* = false$\}$.
2. Are *pineapple* = true and *pickledOnion* = true conditionally independent given *liked* = false? Show your working.
3. Using Hamming distance throughout,
   (a) show how the $1NN$ classifier would classify $\{$*pepperoni* = false, *pineapple* = false, *pickledOnion* = true$\}$;
   (b) show how the $3NN$ classifier with majority voting would classify $\{$*pepperoni* = false, *pineapple* = true, *pickledOnion* = true$\}$; and
   (c) show how the $3NN$ classifier with inverse distance-weighted voting would classify $\{$*pepperoni* = false, *pineapple* = true, *pickledOnion* = true$\}$.