Classification and K-Nearest Neighbours

Hiroshi Shimodaira*

January-March 2020

In the previous chapter we looked at how we can use a distance measure to organise a data set into groupings. In this chapter—and for most of the rest of the course—we look at the case where each data point has a *label*, identifying it as a member of a class. In the recognition of handwritten digits, for example, there are ten possible labels corresponding to the digits '0'–'9'. The problem of classification is to predict the correct label for an input feature vector. The classifier is obtained using a set of training data containing feature vectors and their labels. This is referred to as *supervised* learning, since the label for a training vector acts as supervision for the classifier when it is learning from training data.

Let's introduce some notation: $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$ is a D-dimensional (input) feature vector, which has class label c. The *training set* is a set of N feature vectors and their class labels; the n'th training item consists of a feature vector $\mathbf{x}_n$ and its class label c_n. The j'th element of the n'th feature vector is written x_{nj}. A learning algorithm is used to train a classifier using the training set. *Testing* involves assigning a class to an unlabelled feature vector $\mathbf{x}$.

We can measure how accurate a classifier is using an *error function*. A suitable error function for classification is the number of items that are misclassified—i.e., the number of times the classifier assigns a class label different from the true class label. The classification error is often expressed as the percentage of the total number of items that is misclassified, the 'error rate'. We also need to specify on which set of data the error function was measured. The 'training set error rate' is the percentage of misclassifications that the classifier makes on the training set after the learning algorithm has been applied. The 'test set error rate' refers to errors made by the classifier on a set of data not used by the learning algorithm—the test set.[1]

4.1 A simple example

Consider the problem of determining apples from pears. In this example case we have two features for each piece of fruit: its circumference (at the widest point) and its height, each measured in cm. Apples are 'more spherical' than pears which tend to be longer than they are wide. But some pears are relatively short and some apples are taller than expected. In this case we have an input vector of the form $\mathbf{x} = (x_1, x_2)^T$, where x_1 is the circumference and x_2 the height. The class C can take two values A or P (standing for apples and pears).

We have a set of training data: height and circumference measurements, along with class labels. In Figure 4.1 we plot this two dimensional training data for the two classes. We can see that pears tend to have a greater height and smaller circumference compared with apples; however it is not possible to draw a straight line to separate the two classes (Figure 4.1).

[1] The technique of using separate data sets for training and testing is referred to as *cross validation*.

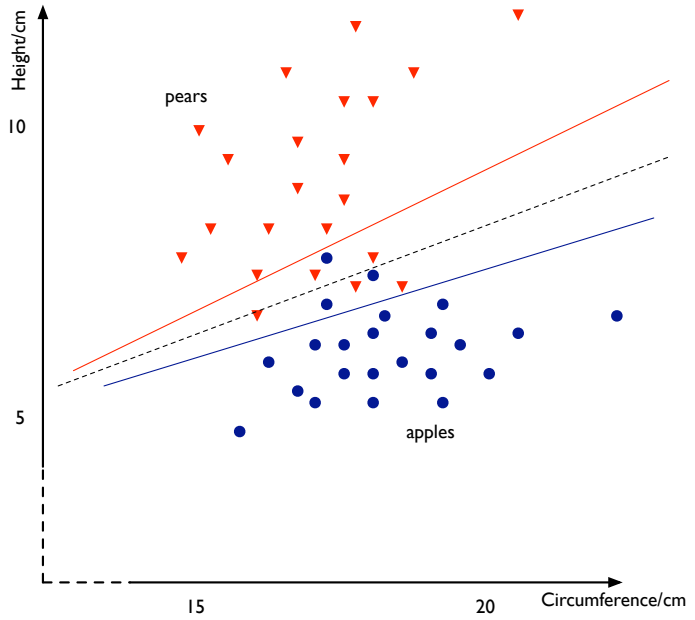

Figure 4.1: Training data for apples and pears. It is not possible to draw a straight line to perfectly separate the classes in this feature space. Three possible lines are drawn, but each results in a number of misclassifications.

We now have three new, unlabelled examples which we would like to classify (represented as stars in Figure 4.2):

- $(16, 10)$: all the training data in the region of this point is classified as P, so we classify this point as P.
- $(19, 6)$: looking at the training data it seems obvious to class this as A.
- $(17, 7)$: its not obvious in which class this example should be classified; the feature vector gives us evidence whether we have an apple or a pear, but does not enable us to make an unambiguous classification.

We can draw a straight line such that one side of it corresponds to one class and the other side to the other—as in the three possible lines shown in Figure 4.1. Such a line is called a *decision boundary*; if the data was three-dimensional, then the decision boundary would be defined by a plane. For one-dimensional data, a decision boundary can simply be a point on the real line. Intuitively it seems possible to find an optimal decision boundary, based on minimising the number of misclassifications in the training set. Although we use the training set to determine the decision boundary, we are most interested in making optimal decisions on a test set.

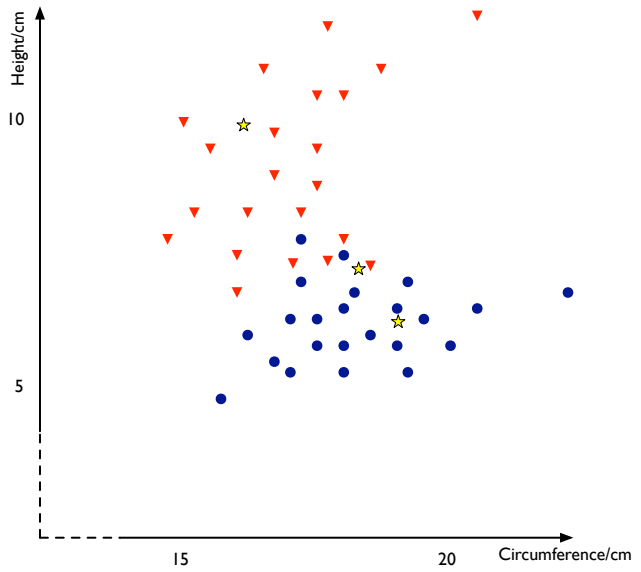

Figure 4.2: The training data for apples and pears, together with three test points.

4.2 K-Nearest Neighbour

An intuitive way to decide how to classify an unlabelled test item is to look at the training data points nearby, and make the classification according to the classes of those nearby labelled data points. This intuition is formalised in a classification approach called K-nearest neighbour (k-NN) classification.

Nearest neighbour classification has a very simple basis: to classify a test item, find the item in the training set which is closest and assign the test item to the same class as the selected training item. If there happens to be an identical item in the training set then it makes sense to assign the test item to the same class. Otherwise, the class of the member in the training set which is most similar to the test item is our best guess. We use a distance measure (e.g., Euclidean distance) to determine similarity. If we have a representation for which the distance measure is a reasonable measure of similarity, then the nearest neighbour method will work well.

4.2.1 K-Nearest Neighbour classification algorithm

Rather than just using the single closest point, the K-nearest neighbour approach looks at the K points in the training set that are closest to the test point; the test point is classified according to the class to which the majority of the K-nearest neighbours belong. For the first two items above, the value of K is not really important: $(16, 10)$ is classified as P and $(19, 6)$ is classified as A, no matter how many nearest neighbours are considered. However the third example above, $(17, 7)$, is ambiguous, and this is reflected in the sensitivity of the classifier to the value of K:

- 1-nearest: classified as pear

- 2-nearest: tie (one apple and one pair are nearest neighbours). In this case, just choose randomly between the two classes
- 3-nearest: classified as apple
- 5-nearest: classified as pear
- 9-nearest: classified as apple

K-nearest neighbour is very efficient at training time, since training simply consists of storing the training set.[2] Testing is much slower, since it can potentially involve measuring the distance between the test point and every training point, which can make K-nearest neighbour impractical for large data sets.

We can write the K-nearest neighbour algorithm precisely as follows, where X is the training data set with class labels so that $X = \{(\mathbf{x}, c)\}$, Z is the test set, there are C possible classes, and r is the distance metric (typically the Euclidean distance):

- For each test example $\mathbf{z} \in Z$:
  - Compute the distance $r(\mathbf{z}, \mathbf{x})$ between $\mathbf{z}$ and each training example $(\mathbf{x}, c) \in X$
  - Select $U_k(\mathbf{z}) \subseteq X$, the set of the k nearest training examples to $\mathbf{z}$
  - Decide the class of $\mathbf{z}$ by the majority voting:

$$c(\mathbf{z}) = \arg\max_{j \in \{1,\ldots,C\}} \sum_{(\mathbf{x},c) \in U_k(\mathbf{z})} \delta_{jc} \tag{4.1}$$

where the Kronecker delta $\delta_{jc} = 1$ when $j = c$ and zero otherwise. It is sometimes possible to store the training data set in a data structure that enables the nearest neighbours to be found efficiently, without needing to compare with every training data point (in the average case). One such data structure is the k-d tree. However, these approaches are usually only fast in practice with fairly low-dimensional feature vectors, or if approximations are made.

4.2.2 Decision boundaries by K-NN classification

K-nearest neighbour classifiers make their decisions on the basis of local information. Rather than trying to draw decision boundaries across the whole space (as in Figure 4.1), K-nearest neighbour techniques make decisions based on a few local points. As such they can be quite susceptible to noise, especially if K is small: a small shift in the location of a test point can result in a different classification since a new point from the training data set becomes the nearest neighbour. As K becomes larger, the classification decision boundary becomes smoother since several training points contribute to the classification decision.

What do the decision boundaries look like for nearest neighbour classification? Consider a special case of 1-nearest neighbour classification in which each training data point belongs to a distinct class from the others. In that case each training data point defines a region around it; test points within this region will be classified to the same class as the training data point. These regions are illustrated for a simple case in Figure 4.3, where the boundaries of regions are shown as dotted lines. As you may

[2] In practice, responsible machine learning practitioners will try out different choices of K, and different distance measures, possibly optimising free parameters of a distance measure. Then training requires testing different choices, and becomes expensive.

understand now, each boundary is given as the perpendicular bisector of the line segment between the two corresponding data points. This partitioning formed by a set of data points is sometimes called *Voronoi diagram* or *Voronoi tessellation*.

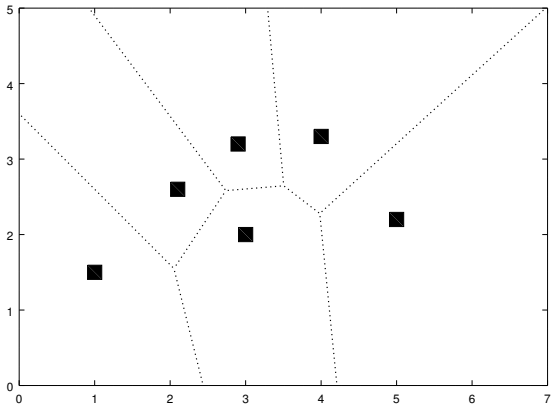

Figure 4.3: Decision boundaries by 1-NN for data points of distinct classes from each other

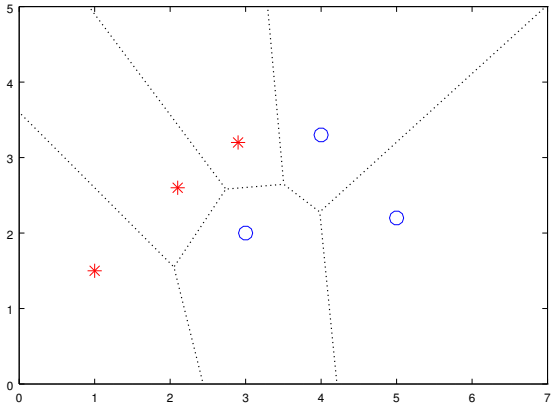

Figure 4.4: Nearest neighbour regions for a training data set of two classes, where training samples of one class are shown with '*' in red, those of the other class are shown with 'o' in blue. The Euclidean distance is used as the distance measure in this example.

Now, we assume that each data points belongs to either of two classes, say, red or blue shown classes as is shown in in Figure 4.4,

To obtain the decision boundary we combine those boundaries which are between regions of different

classes, as illustrated in Figure 4.5. The resultant boundary is referred to as being *piecewise linear*. Figures 4.6 and 4.7 show the case of three classes.

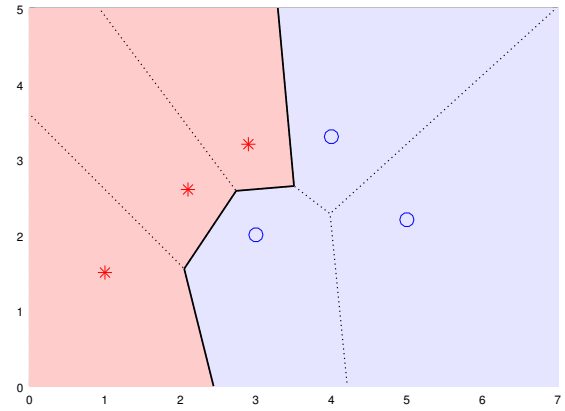

Figure 4.5: Decision boundary and decision regions for a 1-nearest neighbour classifier for two classes.

4.3 Reading

For further reading on K-nearest neighbour see

- Mitchell, sections 8.1 and 8.2
- Duda, Hart and Stork, sections 4.4, 4.5 and 4.6

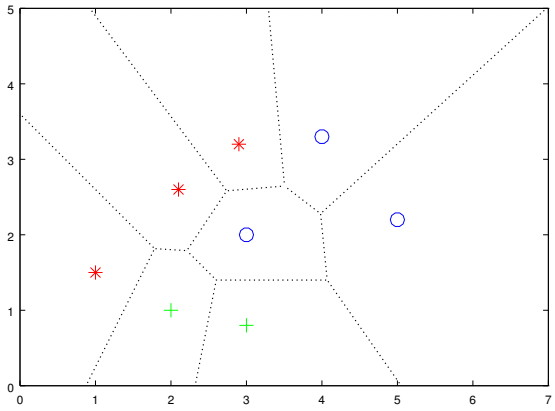

Figure 4.6: Nearest neighbour regions for a training data set of three classes.

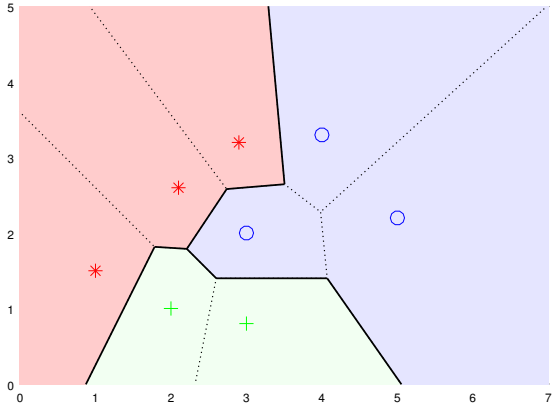

Figure 4.7: Decision boundary and decision regions for a 1-nearest neighbour classifier for three classes.

Exercises

1. Using Figure 4.3 as an example, explain why each line segment of Voronoi diagram is given as the perpendicular bisector of the line segment between the two corresponding data points.

2. Randomly plot several points on a paper, draw a corresponding Voronoi diagram by hand.