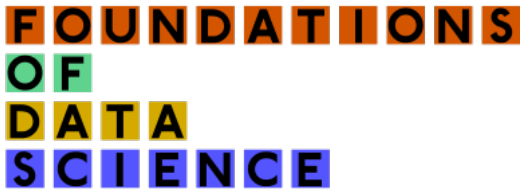

Inf2 – Foundations of Data Science 2021
Topic: Intro to supervised learning: training, testing and validation: Nearest neighbours

Hiroshi Shimodaira*, Iain Murray*, David C. Sterratt

14th November 2021

Further reading (not examinable):
Hastie et al. (2009) pp 463–471

1 Video: Classification

Supervised learning of classifications Suppose a bank has data on previous customers it has given loans to, including variables such as their income, housing status and employment status. Each of these sets of variables – also referred to as **feature vectors** – has a **label**, indicating whether the customer did or didn't pay back their loan. The bank might want to predict whether a new customer will be able pay back a bank loan from their features, i.e. to predict whether they belong to the class of customers who paid or the class of customers who didn't pay. This is an example of the **classification problem**, which we define as the problem of predicting the correct label for an unlabelled input feature vector.

Supervised and unsupervised learning Classification is an example of a supervised learning process. In a **supervised learning** process, there is a **training set** of data in which each data item has a number of **features** and a known **label**. The goal of supervised learning is to predict the label of an item that has not been previously seen from its features. In contrast, in **unsupervised learning** processes, the training set does not contain any labels, and the goal is to learn something about the structure of the data. We will return to unsupervised learning in a later topic.

Visualising the classification problem To visualise the classification problem, we'll use a toy example: the fruit data set, collected by Iain Murray (Murray, 2006). He bought pieces of fruit and measured their height and width (features) and noted the type of fruit (the label). Figure 1 visualises

*These lecture notes are based on the ones written by Iain Murray and Hiroshi Shimodaira for the Inf2B course, which ran until the academic year 2019/20.

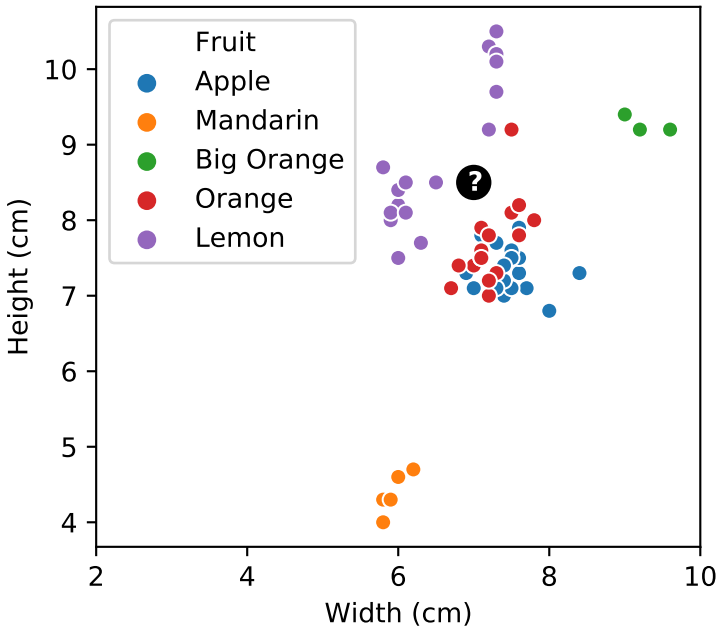

Figure 1: The supervised learning problem, as applied to fruit. We are given the labels of the fruit with various widths and heights. We are then presented with an unknown piece of fruit with given width and height (the test point, represented by the question mark). The task is then to predict what type of fruit it is.

the data. In the context of the fruit, the classification problem is using this dataset to build a machine to predict the class of a piece of unidentified fruit automatically just by measuring its width and height. We will refer to the feature vector of this unidentified fruit as the **test point**.

Definition of a classifier To solve a specific classification problem, we construct a **classifier**. A classifier is a function that takes a feature vector $\mathbf{x}$ and returns a class c where c is a member of a set C. In principle, we can construct a classifier however we want to, as long as it matches this definition. Regardless of how the classifier is constructed, it will have two important properties: **decision boundaries** and **classification errors**.

Decision boundaries 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 2 we plot this two-dimensional training data for the two classes. We can see that it is not possible to draw a straight line to separate the two classes.

We now have three new, unlabelled examples which we would like to classify (represented as stars in Figure 3):

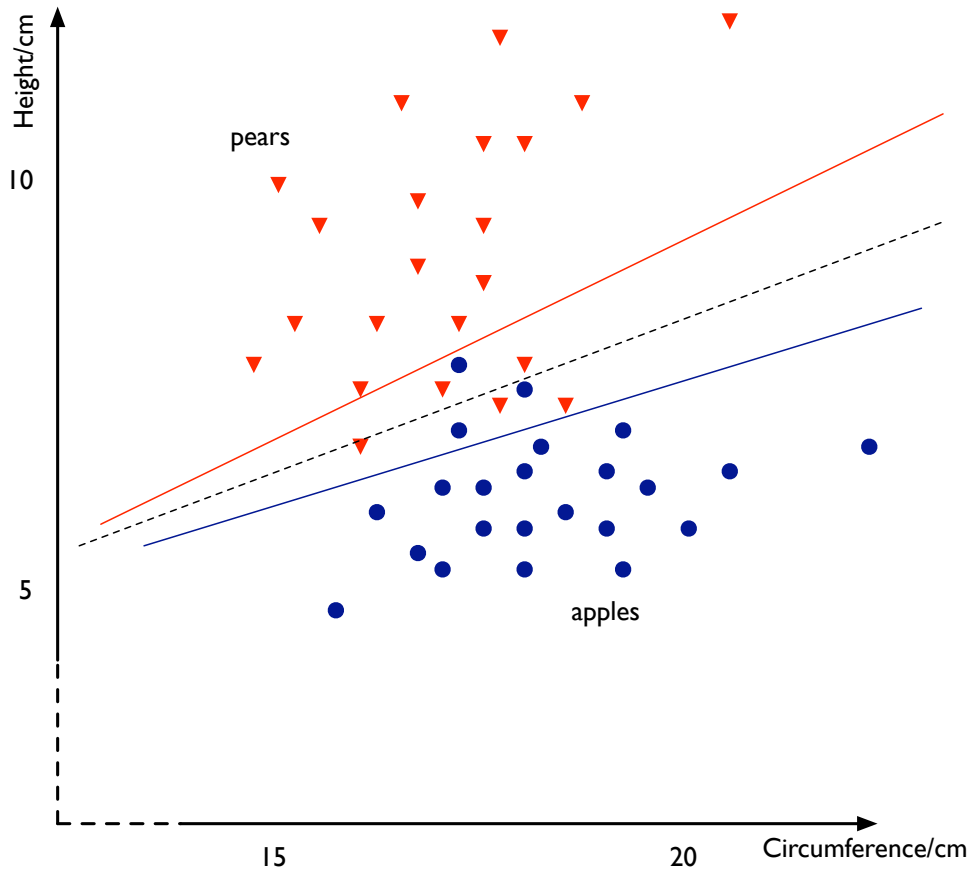

Figure 2: Training data for apples and pears. It is not possible to draw a straight line to perfectly separate the classes in this feature space. Three possible lines are drawn, but each results in a number of misclassifications.

- $(16, 10)$: 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.
- $(18, 7)$: it's 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 2. 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.

Constructing classifiers using supervised learning To construct the classifier automatically we need:

1. a set of training data containing feature vectors and their labels
2. an algorithm that we train using the training data

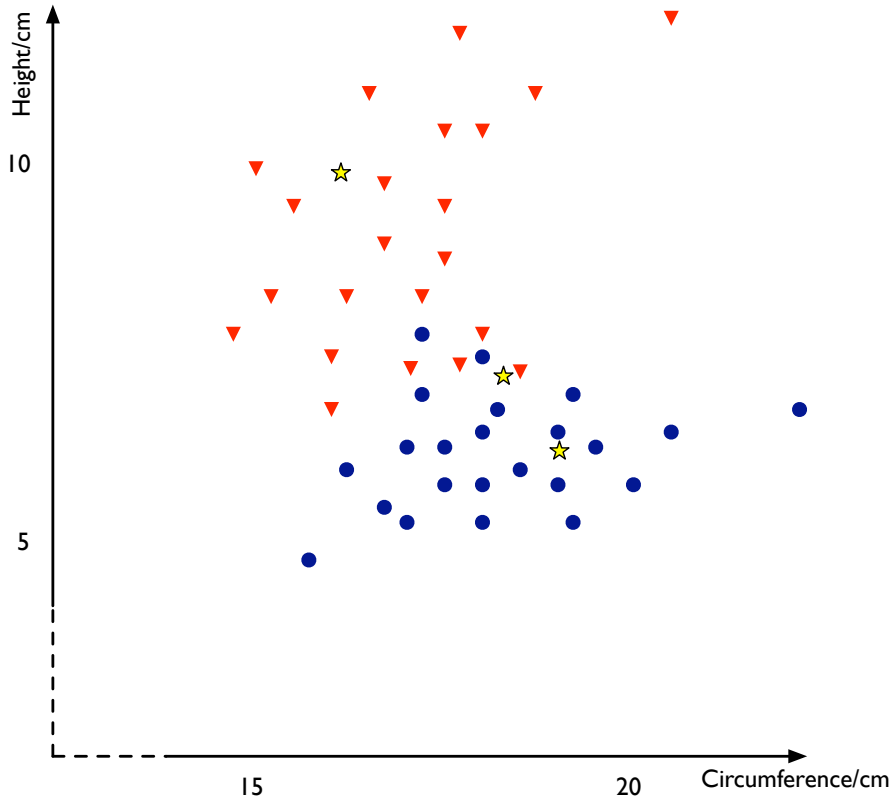

Figure 3: The training data for apples and pears, together with three test points.

3. **hyperparameters**, numbers that control how the algorithm learns and predicts

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.

Regression as supervised learning We have already encountered another form of supervised learning: linear regression. Here the independent variables are synonymous with feature vectors. Rather than being associated with a label, a categorical variable, the independent variables are associated with the dependent variable, which is a numeric variable. However, the task is still to predict a value of this "label" for an unseen value of the independent variable, and we "train" linear regression by computing the best estimates for the coefficients from existing data.

2 Video: Nearest neighbour classification

Principle of nearest neighbour classification Nearest neighbour classification (or one-nearest neighbour classification to be precise) 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.

Decision boundaries for nearest neighbour classification What do the decision boundaries look like for nearest neighbour classification? 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, where the boundaries of regions are shown as dotted lines. 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**.

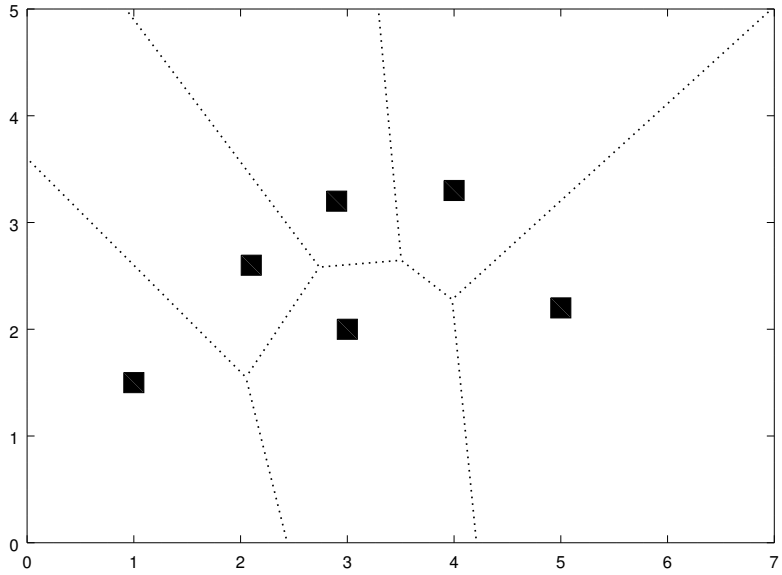

Figure 4: Decision boundaries by 1-NN for data points of distinct classes from each other

Now we assume that each data point belongs to either of two classes, say, red or blue. To obtain the decision boundary we combine those boundaries which are between regions of different classes, as illustrated in Figure 5. The resultant boundary is referred to as being **piecewise linear**. Figure 6 shows the decision boundary and decision regions in the case of three classes.

Application of 1-nearest neighbour to a real dataset Figure 7 shows 1 nearest-neighbour classification applied to the fruit dataset. Note that we've standardised the variables, so that the data spreads out roughly equally in both directions. In common with other distance-based methods, we would like the results of clustering to be independent of the units we measure the variables in. It can be seen that the decision boundary is quite complex, with islands of apple amongst the oranges. We'll explore in the next video if this might be a problem.

3 Video: Evaluation

Classification error function When we discussed linear regression, we evaluated its performance by an **error function**, such as the root mean squared error, which was a function of the actual and predicted values y_i and $\hat{y}_i$.

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 $\hat{c}_i$ different from the true class label c_i. The classification

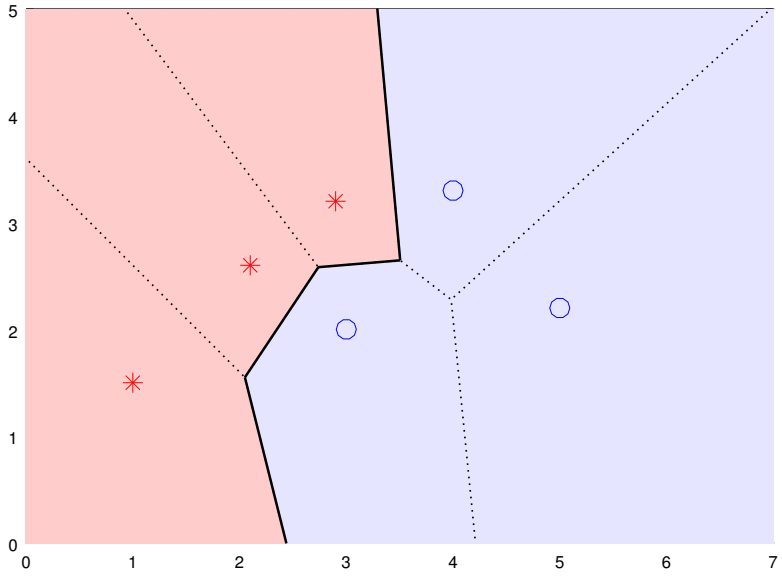

Figure 5: Decision boundary and decision regions for a 1-nearest neighbour classifier for a training data set of two classes, where training samples of one class are shown with '*' in red, those of the other class are shown with 'o' in blue. The Euclidean distance is used as the distance measure in this example.

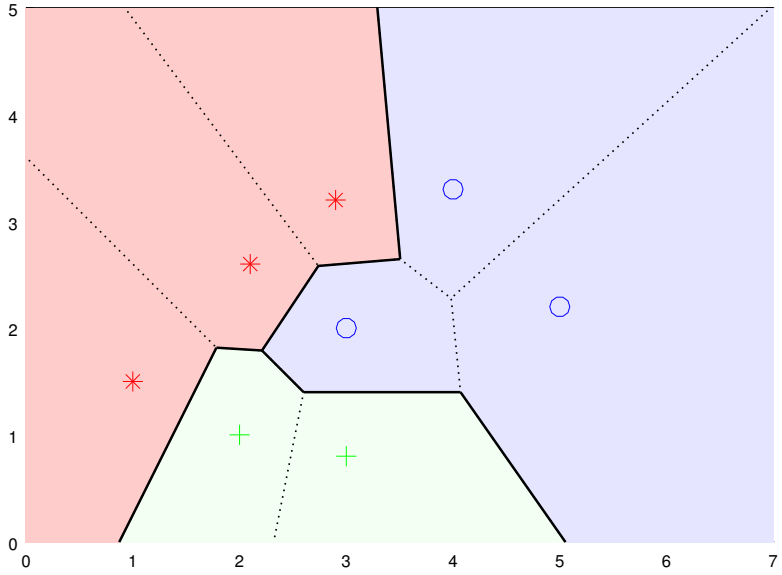

Figure 6: Decision boundary and decision regions for a 1-nearest neighbour classifier for three classes.

error is often expressed as the percentage of the total number of items that is misclassified, the **error rate**.

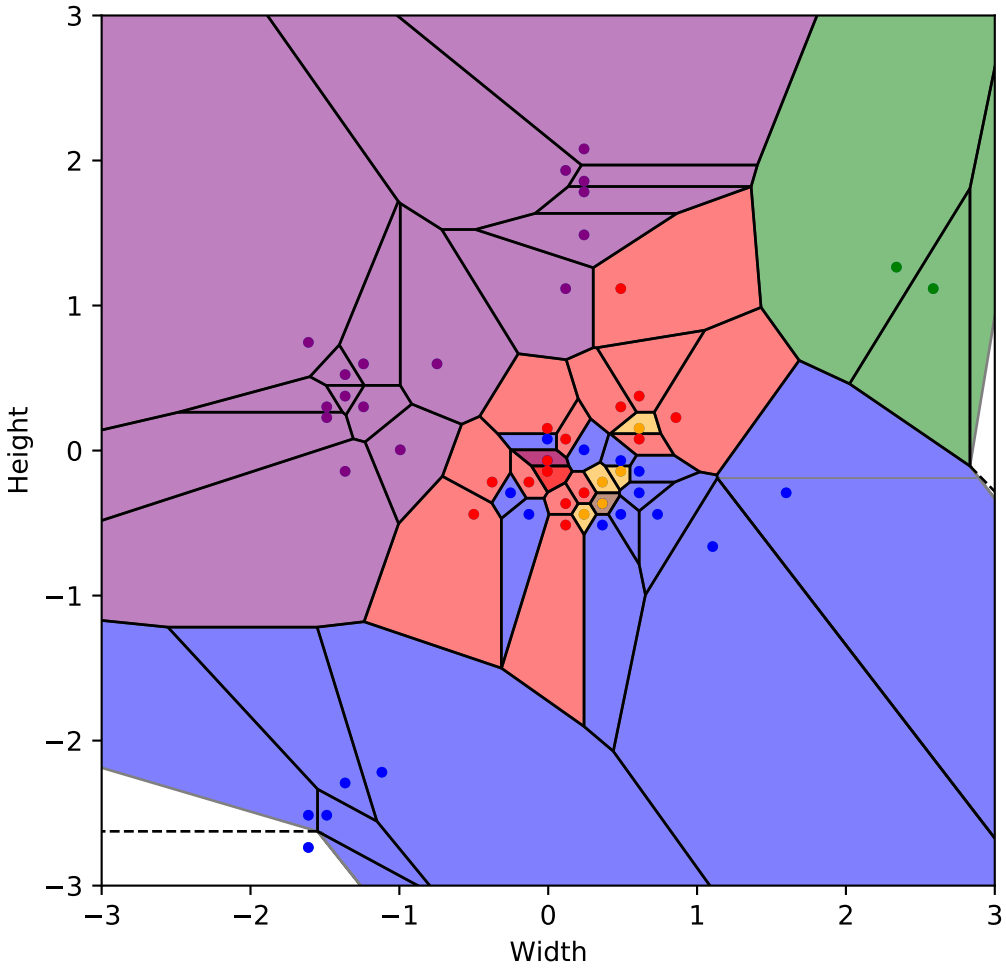

Figure 7: Decision regions for one nearest neighbour classification applied to the fruit data set. The variables have been standardised to make the scales on both axes similar. Some regions are darker shade of blue or red. This indicates that there are 2 points labelled with “apple” or “orange” in the dataset with the same features. There is one region that is purple, amongst the blue and red region. There are two data points corresponding to this region with identical coordinates, one labelled with orange and one with apple.

Error function for one-nearest neighbour classification For one-nearest neighbour classification, the error rate when we consider members of the training set is 0, since the closest point in the training set to a member of the training set is itself[1].

Evaluating generalisation to unseen data This sounds very promising, until we remember that the job of the classifier is to classify data points that we haven't seen before. It may be that the classifier will not **generalise** to data that haven't seen. In order to estimate how well the classifier generalises, we can split our original data set into a training set and a **testing set**. The training and testing sets are mutually exclusive, and a typical split might be 70% for training and 30% for testing. We train the classifier using the training set, and then evaluate the performance using the testing set. **We are not allowed to use the test set to train the classifier** – otherwise our estimate of its performance on the test set will be too optimistic.

In summary, 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 the testing set.[2]

Training and testing set notation We'll use the notation: $\mathbf{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(D)})^T$ to denote 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 i'th training item consists of a feature vector $\mathbf{x}_i$ and its class label c_i. The j'th element of the i'th feature vector is written x_{ij}.

4 Video: k-Nearest neighbour classification

Principle of k-nearest neighbour classification 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, $(18, 7)$, 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

[1] Unless we have two data points with exactly the same features and different labels.

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

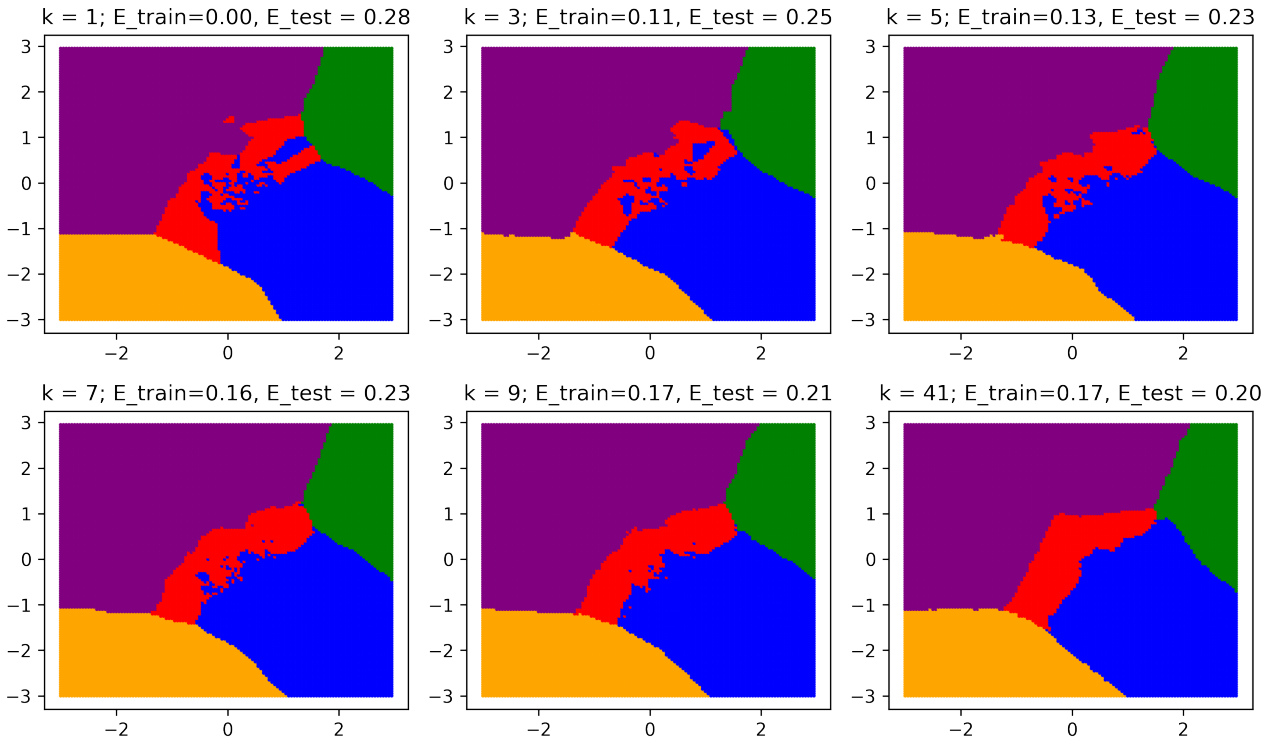

Figure 8: Decision regions, training error and testing error for various values of k.

Decision boundaries produced 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 2), 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 (Figure 8).

k-nearest neighbour algorithm We can write the k-nearest neighbour algorithm precisely as follows, where and d is the distance metric (typically the Euclidean distance):

- For an unseen example $\mathbf{x}$:
  - Compute the n distances $d_i = d(\mathbf{x}, \mathbf{x_i})$ between $\mathbf{x}$ and the features of each training example $\mathbf{x}_i$, $i \in 1, \ldots n$.
  - Sort the distances from lowest to highest and find the indices $i_1, \ldots i_k$ of the k lowest values of d_i
  - Find the classes that belong to the closest points, i.e. $c_{i_1}, \ldots, c_{i_k}$
  - Each of these represents a vote for a class. Count the votes for each class and return the one with the largest number.
  - If there is a tie, choose randomly, or look at the $k+1$th neighbour to resolve the tie.

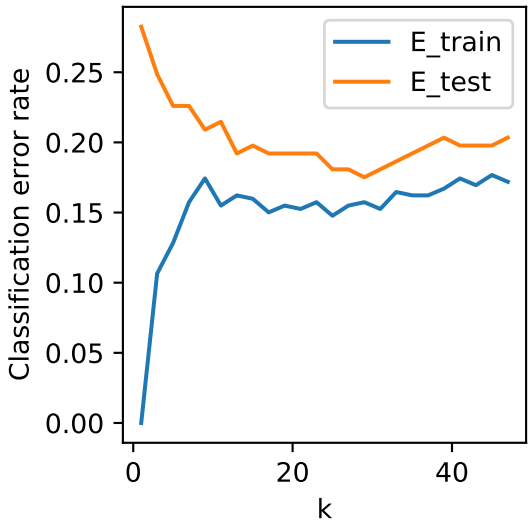

Figure 9: Classification error rate for various values of k.

Choosing k The value k is **hyperparameter**: a number that we can choose to get the best performance from the algorithm. Figure 8 shows the classification error for various values of k on the training set and the testing set. As k increases the error on the training set initially increases rapidly, as explained in the last video. The testing error decreases a little and then starts rising around $k = 9$, indicating that a somewhat larger k helps generalisation. Both testing and training error then increase.

This graph suggests that we can look at the error on the testing set to set k. **But this would break the rule of using the test data to train the classifier, since our choosing the best hyperparameter k is part of the training process.** We have really been using the test data **validation data**, that is, data used to help us validate our choice of hyperparameter.

Thus, we need to divide our dataset into 3 parts:

- Training data (about 50%): used to train the classifier for any particular value of k.
- Validation data (about 25%): used to compare performance of the trained classifier for different values of k.
- Testing data (about 25%): used to assess the performance of the trained classifier with the one value of k that we have chosen.

The precise fractions of data are not crucial. There are more sophisticated ways of undertaking validation that we won't go into here.

Computational efficiency of k-nearest neighbour classification k-nearest neighbour is very efficient at training time, since training simply consists of storing the training set.[3] Testing is much slower, since, in the simplest implementation, it would involve measuring the distance between the test point and every training point, which can make k-nearest neighbour impractical for large data sets.

[3] 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.

Improving the efficiency of k-NN 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.

References

Hastie, T., Tibshirani, R. et al. (2009). *The elements of statistical learning*. Springer, second ed. URL http://dx.doi.org/10.1007/b94608

Murray, I. (2006). 'Oranges, lemons and apples dataset'. URL http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/