Lecture 6: Nearest Neighbours Classification

Iain Styles

25 October 2019

Classification

In regression, we are interested in predicting the values of a *continuous* variable. In classification problems, we are interested in predicting *categorical* variables. Common examples of classification problems include:

- Determining what type of object is present in an image.
- Sorting documents into different types.
- Determining whether a set of diagnostic tests implies that a patient has a disease.

Classification is, like regression, a *supervised* learning technique. A classifier is trained on some set of training data, and then used to classify unseen examples.

The MNIST Dataset

A classic example of a classification problem, is the MNIST set of handwritten digits. This dataset consists of a training set of 70,000 images of handwritten numerical digits (0–9), divided into a training set of 60,000 images and a test set of 10,000 images. Each image has a class label that indicates which digit it contains. There is also a test set of 10,000 examples. The images in both sets have been rescaled so that they are the same size: 28×28 pixels

MNIST is one of the most frequently used datasets for classification tasks because the data is well curated and requires little or no preprocessing before it can be used. It is based on an original dataset originally developed by the National Institute of Standards and Technology (NIST) in the USA and has been modified to make it more suitable for machine learning applications: the original dataset had substantial bias, with the training set taken from employees of the American Census Bureau employees, and the testing set from American high school students. Modified NIST – MNIST – rebalances this dataset. Full details of how this was done can be found at the database's website, `http://yann.lecun.com/exdb/mnist/`. A few examples from the MNIST training set are shown in Figure 1.

We will use the MNIST dataset as a vehicle through which we will study a range of algorithms for classification. All of the methods that we will study will require data that is "vectorial" in nature; that is, we need a way in which our image data can be represented in vector form. The simplest way to do this is to "unroll" the images by rastering across them as shown in Figure 2. One hundred

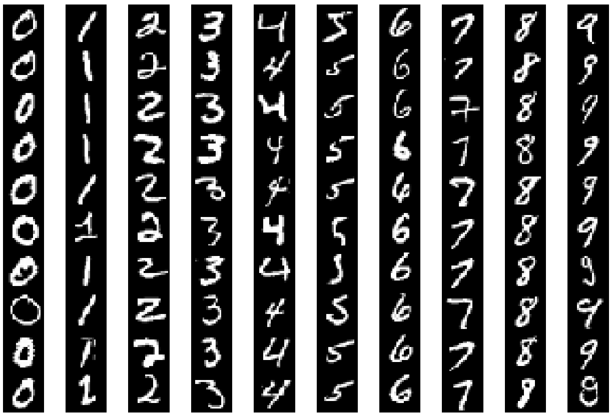

Figure 1: Ten samples from each category of the MNIST training dataset.

real examples of vectorised MNIST images from each category are shown in Figure 3.

k-nearest-neighbours Classification

Let us begin to explore the classification problem by considering one of the conceptually simplest ideas of all: we will try to classify MNIST digits according to how similar they are to examples for which we already know the category. This is a very crude form of learning in that all we do is memorise the training set and see how similar new samples are to those that we already know the label for. The simplest possible example of this is nearest-neighbour classification, where for each sample with an unknown category, we find it's nearest-neighbour in the training data. The k-nearest-neighbours algorithm is given in Algorithm 1.

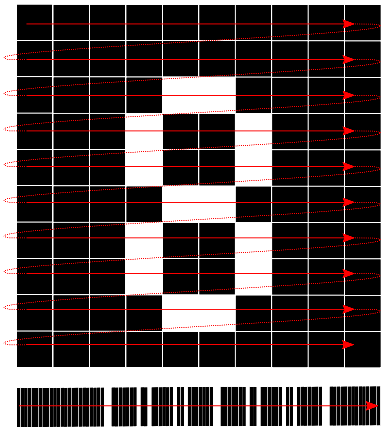

Figure 2: (Top) raster scanning an image to form (Bottom) a vectorised representation

```
Data: A set of labelled training data
Data: A set of unlabelled test data
Data: Integer $k$
Result: For each item in the test set, returns the most common
        label of that items $k$ nearest neighbours in the training
        set.
for each item $x$ in test set do
    for each item $y$ in training set do
        Compute similarity $d(x, y)$
    end
    Find the $k$ most similar items to $x$ in the training set.
    Compute the most common label
end
```

Algorithm 1: k-nearest neighbours classification.

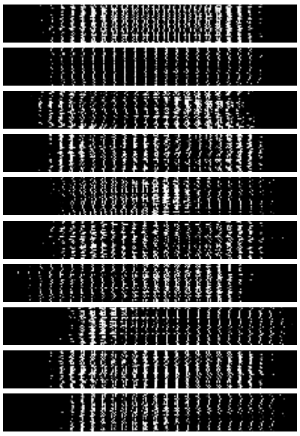

Figure 3: One hundred vectorised samples from each MNIST category. Each row of each sub-image corresponds to a single vectorised sample. Top–bottom: 0–9.

The is shown graphically in Figure 4. The k-nn method has a major advantage over other methods: its training time is zero – there is no training phase, there is just the training dataset. The disadvantage, though, is that the time taken to make predictions is proportional to the size of both the training and testing datasets: every element of the test set has to be compared to every element of the training set. This could be a major inconvenience if either is very large.

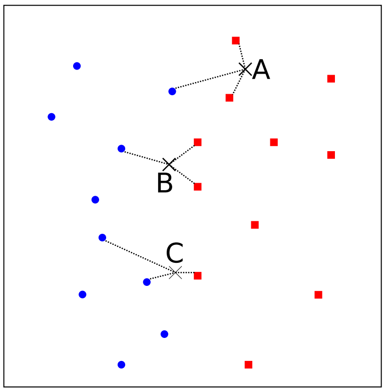

Figure 4: A illustration of the k-nearest-neighbours algorithms for $k = 3$. Two classes of labelled training data are shown: blue circles and red squares. The unknown test points, marked as black crosses (A, B, C) are marked along with their 3 nearest neighbours. The majority class of the three neighbours is assigned to the test point: A: red; B: red; C: blue. Notice that the value of k affects this assignments: the nearest neighbour of point C is red.

One might regard it as entirely unreasonable to think that such a simple method could succeed at the task of classifying MNIST digits. There are several objections:

- Vectorising the images loses much of their spatial information. The visual cues that humans rely on are lost (for example closed vs open loops in 5 vs 6), and cannot be recovered from the vector representation alone since since the original image dimensions are not known to the classifier.

- There is substantial variability between characters (eg the 1's in Figure 1), suggesting that simple pixel-by-pixel comparison will be highly unreliable.

Nevertheless, it will be instructive for us to see how we this works in practice. For our first experiments, we will choose 10000 random examples from the training dataset, and 1000 random examples from the testing dataset. We will use the Euclidean distance as our measure of similarity, and where smaller values indicate higher similarity. For images vectors $\mathbf{x}$ and $\mathbf{y}$, this is given by

$$d(\mathbf{x},\mathbf{y}) = \sqrt{((\mathbf{x}-\mathbf{y})^{\mathrm{T}}(\mathbf{x}-\mathbf{y}))} = \sqrt{\sum_i (x_i - y_i)^2}. \qquad (1)$$

where i indexes the components of the vector (pixels of the image). We first try a very simple $k = 1$ nearest-neighbour classification. The results of this are shown in the *confusion matrix* shown in Figure 5.

P T	0	1	2	3	4	5	6	7	8	9
0	83	1	1	0	0	0	5	0	10	0
1	0	100	0	0	0	0	0	0	0	0
2	1	11	53	2	1	0	3	4	25	0
3	0	11	2	48	0	1	4	3	28	3
4	2	9	0	0	42	0	2	3	16	26
5	2	7	0	4	0	36	2	0	43	6
6	3	6	0	0	0	1	80	0	10	0
7	0	11	0	1	0	0	1	75	4	8
8	2	13	0	6	1	3	3	4	65	3
9	0	5	1	1	4	0	0	4	2	83

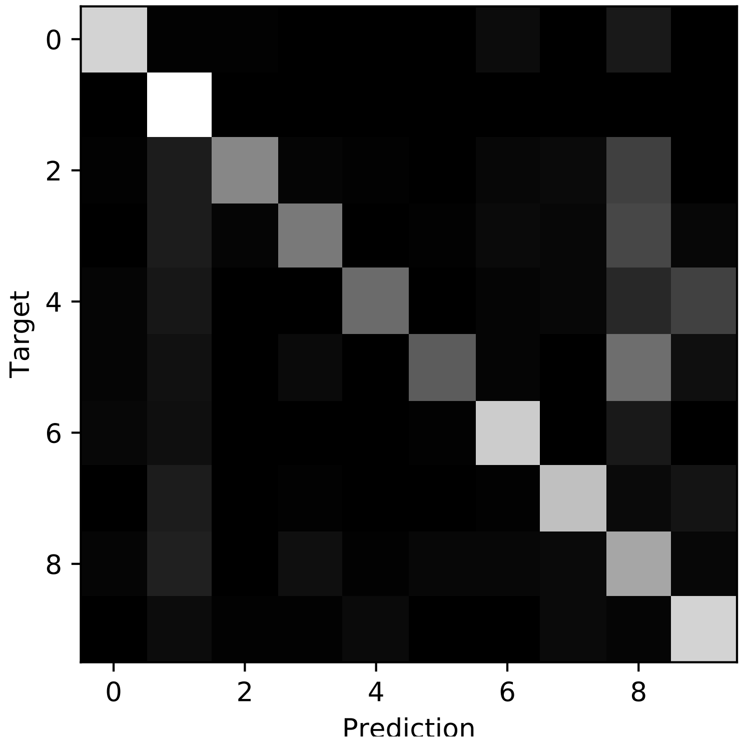

Figure 5: MNIST classification results (Target T vs Prediction P) with kNN for $k = 1$.

We may be justifiably surprised at how well this has done! Some character classes (1) are predicted with 100% accuracy. The overall accuracy (average of the diagonal elements) is 67%. Some characters appear harder than others: 5 is hard to distibnguish from 8 (but not the other way, curiously); 4 is hard to distinguish from 9. Both of these confusions make a lot of sense: they are things we may have difficulty with in cases of poor handwriting.

Can we do better? Let's try some more neighbours. In Figures 6, 7 and 8 we show the confusion matrices for $k = 3$, $k = 5$, and $k = 7$.

P T	0	1	2	3	4	5	6	7	8	9
0	95	1	0	0	0	0	1	0	3	0
1	0	100	0	0	0	0	0	0	0	0
2	4	14	68	0	0	0	1	2	11	0
3	2	13	4	64	0	1	3	2	8	3
4	2	13	1	0	51	0	4	2	3	24
5	5	13	0	10	1	39	2	0	24	6
6	2	7	0	0	1	1	88	0	1	0
7	0	18	2	1	1	1	0	68	3	6
8	3	18	0	3	1	3	3	4	65	0
9	1	7	0	1	1	0	0	2	2	86

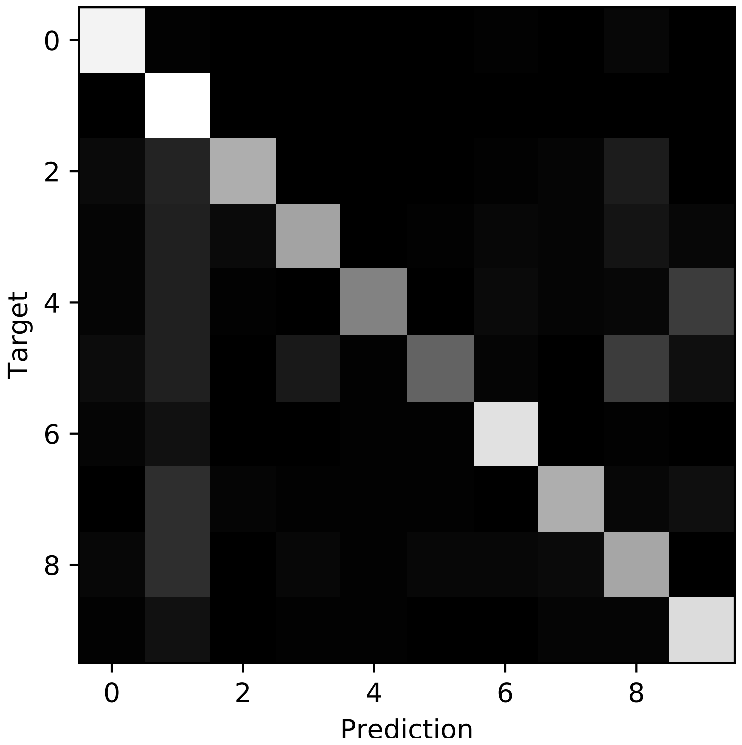

Figure 6: MNIST classification results (Target T vs Prediction P) with kNN for $k = 3$.

The overall accuracy in these cases is 72%, 74%, and 75%. The consensus voting approach used in $k$nn appears to bring significant gains when compare to the simple nearest neighbour approach. We do seem to be reaching a point of diminishing returns though,

P T	0	1	2	3	4	5	6	7	8	9
0	97	1	0	0	0	0	1	0	1	0
1	0	100	0	0	0	0	0	0	0	0
2	3	17	69	1	0	0	2	3	5	0
3	1	19	1	60	0	0	6	3	7	3
4	2	12	1	0	50	0	5	1	4	25
5	5	9	0	5	2	51	2	0	19	7
6	2	7	0	0	1	1	89	0	0	0
7	0	18	0	0	1	1	0	73	2	5
8	3	18	1	3	0	1	4	5	65	0
9	1	9	0	0	0	0	0	1	4	85

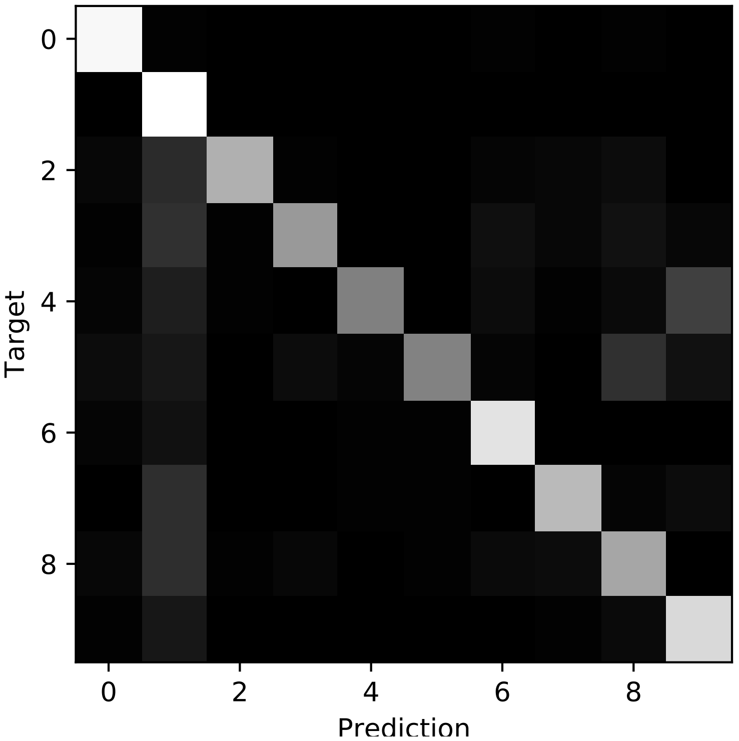

Figure 7: MNIST classification results (Target T vs Prediction P) with *k*NN for $k = 5$.

P T	0	1	2	3	4	5	6	7	8	9
0	95	1	0	0	0	0	3	0	1	0
1	0	100	0	0	0	0	0	0	0	0
2	1	17	70	0	0	0	2	4	6	0
3	1	20	0	61	0	1	6	2	5	4
4	3	9	0	0	55	0	4	1	2	26
5	5	9	1	5	1	51	3	0	17	8
6	2	7	0	0	0	1	90	0	0	0
7	1	17	1	0	1	0	0	75	1	4
8	3	16	1	2	0	1	4	5	66	2
9	1	7	0	0	0	0	0	2	2	88

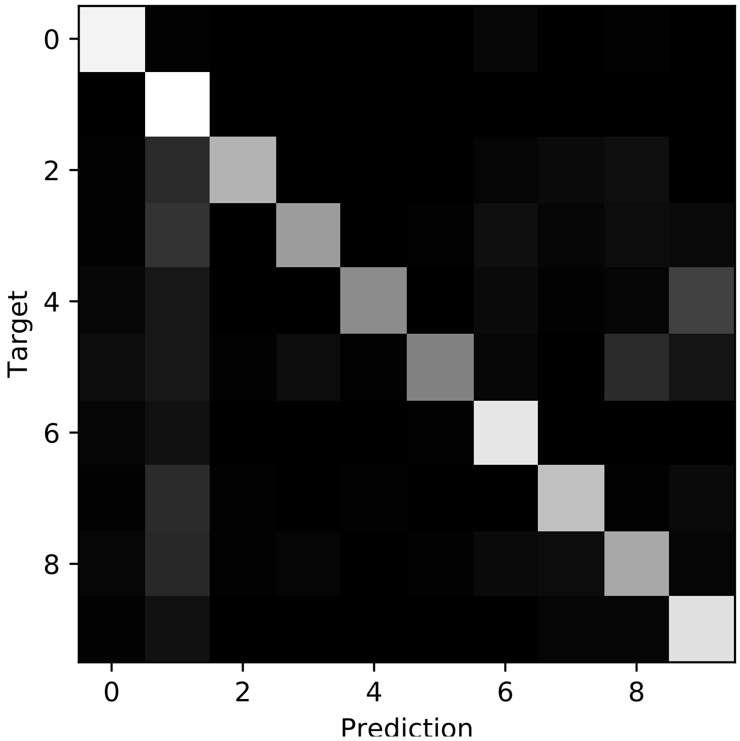

Figure 8: MNIST classification results (Target T vs Prediction P) with kNN for $k = 7$.

and the confusion matrices show us that whilst 0, 1, 6, and 9 can be identified very accurately indeed with this approach, 3, 4, and 5 are proving much more difficult. Is there anything we can do to improve this?

There are many courses of action one could take to improve this but two of the most interesting are to:

1. Change the similarity metric that we use. We have not justified our choice of Euclidean distance as the "correct" measure of similarity and there are many other possible choices of metric that we could use here. In fact, a very popular approach is to *learn* the metric from the data: to find the metric that maximises the classification accuracy. This is an advanced topic that we will cover at the end of the module, if there is time to do so.

2. Reduce the dimensionality of the data. Each image is a point in a 784-dimensional space, and it is known that our intuitions on the behaviour and properties of vectors spaces do not readily translate from the low-dimensional spaces with which we are familiar to such high-dimensional spaces.

We will next consider what problems high dimensional spaces can cause, how reducing their dimensionality can be acheived in practice, and what performance benefits can be realised from this approach.

Reading

Section 2.5.2 of Bishop, Pattern Recognition and Machine Learning, contains a brief treatment of nearest-neighbour methods.