k nearest neighbours

The dirty secret of machine learning

Christos Dimitrakakis

December 1, 2015

1 Introduction

One of the simplest methods in machine learning is k-nearest neighbours. It relies on the intuition that *similar data have similar labels*. Usually this assumption must hold for any machine learning algorithm to work. However, we have to define what we mean by similarity. The difficulty is that frequently it is very hard to formulate the right notion of similarity for a given data-set. Nearest neighbour algorithms take a notion of similarity as given, and do not attempt to discover it automatically. Nevertheless, in the right hands, they can be quite effective. An external method for discovering a good similarity can be easily coupled with them.

In this setting, similarity will be translated to closeness. Consequently, all that a nearest neighbour algorithm needs to be able to classify is a set of labelled data, and a specific notion of distance.

Intuition behind nearest neighbour

- We are given a set of training data $\mathcal{D}_T = (\boldsymbol{x}_t, y_t)_{t=1}^T$.
- We are asked to classify a new data point $\boldsymbol{x}$.
- We assume that nearby points have similar labels.
- So the label of the new point is probably the same as that of the closest training points.

Questions

- How do we identify "closeness"?
- What if the neighbours of a point have different labels?

The nearest neighbour algorithm

The nearest neighbour algorithm is probably the simplest classification method. We are given a set of training data $\mathcal{D}_T = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T)\}$ and are asked to classify a new point $\boldsymbol{x}$. The idea is to find the closest point (x^*, y^*) in the training data and use its label. First, however, we need to define a way to measure the distance between x and any point x_t in the training set. This is done by specifying a function $d(\boldsymbol{x}^*, \boldsymbol{x})$. The choice of function is mainly application-dependent.

Nearest neighbour

Input Data $\mathcal{D}_T$, distance d, new point $\boldsymbol{x}$.
Find $(\boldsymbol{x}^*, y^*) \in \mathcal{D}_T$ such that

$$d(\boldsymbol{x}^*, \boldsymbol{x}) \leq d(\boldsymbol{x}', \boldsymbol{x}) \quad \forall (\boldsymbol{x}', y') \in \mathcal{D}_T.$$

Return y^*.

Mathematically, a distance has the following properties.

Definition 1.1 (Distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$)**.** A distance, or metric $d(\boldsymbol{x}, \boldsymbol{x}')$ satisfies

- $d(\boldsymbol{x}, \boldsymbol{x}') = 0$ if and only if $\boldsymbol{x} = \boldsymbol{x}'$ (identity)
- $d(\boldsymbol{x}, \boldsymbol{x}') = d(\boldsymbol{x}', \boldsymbol{x})$ (symmetry)
- $d(\boldsymbol{x}_1, \boldsymbol{x}_3) \leq d(\boldsymbol{x}_1, \boldsymbol{x}_2) + d(\boldsymbol{x}_2, \boldsymbol{x}_3)$ (triangle inequality)
- It follows that $d(\boldsymbol{x}, \boldsymbol{x}') \geq 0$ (positivity)

1.1 Distances

While the choice of distance is generally application-dependent, two common distances are the following.

Example 1.1 (The Euclidean metric)**.** When $\mathcal{X} = \mathbb{R}^n$ then we can define the Euclidean metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, with

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{i=1}^{n} |x(i) - x'(i)|^2}$$

The Euclidean distance is the familiar notion of distance we use in everyday life. It usually makes sense when the data consists of vectors of real numbers, since then we can define operations such as subtraction, absolute value and square.

Example 1.2 (The Manhattan metric)**.** For any $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ we can define the Hamming metric $d : \mathcal{X}^n \times \mathcal{X}^n \to \mathbb{R}$, to be

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sum_{i=1}^{n} |x(i) - x'(i)|.$$

This can be also seen as the grid, or taxi-cab distance between the two points.

Example 1.3 (The Hamming metric)**.** For any $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ we can define the Hamming metric $d : \mathcal{X}^n \times \mathcal{X}^n \to \mathbb{R}$, to be

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{n} \mathbb{I}\{x(i) \neq x'(i)\},$$

that is the number of features where the two points $\boldsymbol{x}, \boldsymbol{x}'$ differ. Note that $\mathbb{I}\{\}$ is an indicator function taking the value 1 when its argument is true and 0 otherwise. So, this formalises the notion of counting the number of features that are different.

Note that in this case, we only need to be able to compare two features to see if they are equal. Hence, this distance is applicable to any setting.

Note that both of these distances are invariant and symmetric: shifting points, or changing the order of attributes does not make any difference in the distances calculated. This means that if we think that certain regions contain outliers, or that some attributes are more important than others, we should use another type of distance.

Exercise 1 (Implement a function returning the Euclidean distance)**.**

```
EuclideanDistance <- function(x, y)
{
  distance <- ..?? ## calculate distance
  return (distance)
}
```

One way to implement the Euclidean distance is through use of the `dist()` and `rbind()` functions. *The* ***`dist()`*** *function* returns the matrix of distances between the rows of a matrix. For a matrix $\boldsymbol{X}$, we obtain the matrix $\boldsymbol{A} =$ `dist`$(\boldsymbol{X})$ with

$$a_{i,j} = d(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

where $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ are the i and j-th rows of $\boldsymbol{X}$ respectively.

Use *the* ***`rbind()`*** function to place $\boldsymbol{x}, \boldsymbol{x}'$ in two rows of the same matrix.

Exercise 2 (Implement a function returning the Hamming distance)**.**

```
Hamming <- function(x, y)
```

```
  {
3   distance <- ..?? ## calculate distance
    return (distance)
5 }
```

Solution. The `dist()` function only works for numeric arguments. Hence, you must implement a `for` loop

```
1 HammingDistance <- function(x, y) {
    distance <- 0
3   for (i in 1:n) {
      if (x[i] != y[i]) {
5       distance <- distance + 1
      }
7   }
    return (distance)
9 }
```

□

2 The k-nearest neighbour algorithm

The idea of nearest neighbour is to classify a point by simply counting the number of times each label is predicted by its k closest neighbours.

k-nearest neighbour

1: **Input** Data $\mathcal{D}_T$, distance d, number of neighbours k, new point x
2: Find the k closest neighbours of x

$$S(x) = \{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_k^*\},$$

with corresponding labels $y_1^*, \ldots, y_k^*$.
3: Count the number of neighbours labelled y:

$$n_y(x) = \sum_{t=1}^{k} \mathbb{I}\{y_t^* = i\},$$

where $\mathbb{I}\{A\} = 1$ whenever A is true and 0 otherwise.
4: **Return** $y^* = \arg\max_y n_y(x)$, the label agreeing with most neighbours.

Classification with nearest neighbour with the `class` package.

The following exercise illustrates classification with the nearest neighbour algorithm. There are many packages in R which implement nearest neighbours in one form or another. We are going to use the `class` package. The invocation of the method is given below.

```
test.prediction <- class::knn(train.features,
                              test.features,
                              train.labels,
                              k = num.neighbours)
```

We would like to measure the number of errors in the prediction. For this reason, we define a simple function to measure the average number of errors given a set of labels and predictions.

```
## A function to calculate the classification error
ClassificationError <- function(labels, prediction)
{
    return (mean(labels != prediction))
}
```

We can put all of that together in a simple exercise, where we change the number of neighbours of our algorithm to see the effect in the training and testing set.

Exercise 3. The purpose of this exercise is to investigate the performance of nearest neighbour algorithms when the number of neighbours and features change.

- Get the script `knnExample.R`
- Run it. How does the classification error change?
- Increase the amount of training data by changing the `nTraining` variable. What happens?
- When is the training set error close to the testing error and why?
- Repeat the experiment for 10 and 100 features. Are the results the same? Why?

Speeding up nearest neighbour

One problem concerning nearest neighbour is that the larger the training set, the longer we have to spend searching for neighbours.

Example 2.1. In particular, consider what is the complexity of finding the k nearest neighbours in the following setup.

- A training dataset with T examples
- A testing set with M examples.
- Naively, it takes T comparisons to find the closest neighbour of any one example.
- How can we find the k nearest neighbours?
- Method A: Calculate the distance to all T points, then sort. This takes $O(T \log T)$ time and $O(T)$ space.
- Method B: Use a tree structure to store points and distances. Can take $O(k \log T)$.

3 Variants and extensions of nearest neighbours.

Specifying uncertainty

One problem with k nearest neighbour is that overfitting is hard to detect. In particular, we are usually more uncertain the more neighbours we use; but our training error also becomes (usually) higher the more neighbours we use. Perhaps there are some ways to get around that problem.

- Should the output depend only on the labels of the nearest neighbours?
- Could we make it so the output depends on how close the neighbours are?
- What if it depended on all the training examples?
- Could we use boot-strapping to get a better idea of the uncertainty of our predictions?

The following is a variant of nearest neighbour, whereby the importance of different examples is weighted according to their distance from the new point.

Distance sensitive k-nearest neighbour

1: **Input** Data $\mathcal{D}_T$, distance d, number of neighbours k, new point x, decreasing function f.
2: Find the k closest neighbours of x

$$S(x) = \{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_k^*\},$$

with corresponding labels $y_1^*, \ldots, y_k^*$
3: Calculate the weight of labelled y:

$$w_y(x) = \sum_{t=1}^{k} \mathbb{I}\{y_t^* = i\} f\left[d(x_t^*, x)\right].$$

4: **Return** $y^* = \arg\max_y w_y(x)$, the label with the most weight.

The idea is to reduce the influence of far-away neighbours. Common choices for $f(d)$ include $f(d) = 1/d$, and $f(d) = e^{-\alpha d}$, where $\alpha \geq 0$ is a constant.

However, there is no reason to actually limit ourselves to the k closest points, if we are going to weigh points by distance. We could simply look at all points instead! For simplicity, in the following we omit the distance and refer to the kernel, or similarity, function f directly.

Kernel classifier

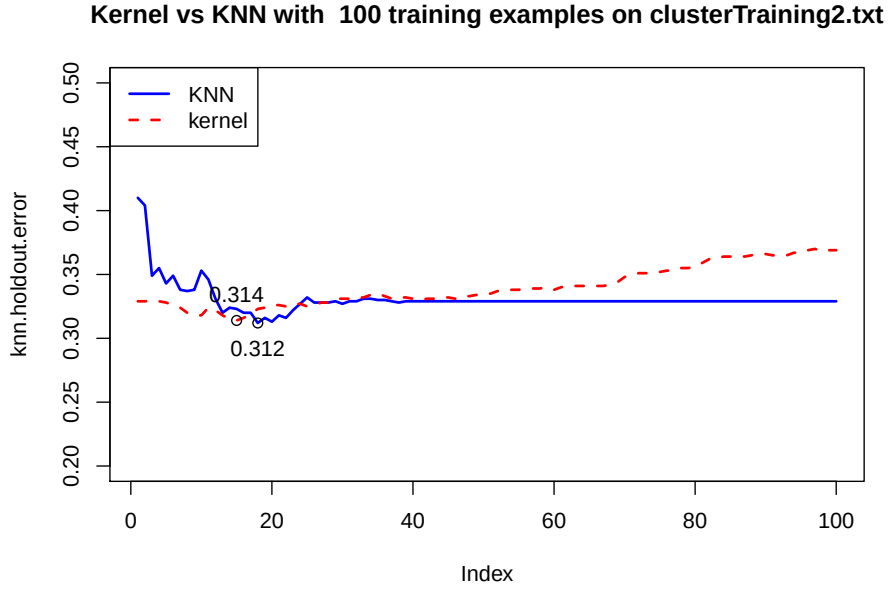
1: **Input** Data $\mathcal{D}_T$, distance d, number of neighbours k, new point x, kernel f

2: Calculate the weight of labelled y:

$$w_y(x) = \sum_{t=1}^{T} \mathbb{I}\{y_t^* = i\} f(x_t^*, x).$$

3: **Return** $y^* = \arg\max_y w_y(x)$, the label with the most weight.

Here is a comparison between KNN a Gaussian Kernel on the validation set, using 100 training examples.



4 Self-paced exercises.

In this exercise, we shall investigate how to select hyper-parameters using decision trees and nearest neighbours.

Respond privately to this exercise on `piazza.com`, class FDD 2 using the tag `hw3`

Exercise 4. The purpose of this exercise is to investigate the performance of nearest neighbour and decision trees using holdouts or cross validation.

- Using holdouts or cross-validation [but the same method for both classifiers] on the 2d-clustering training set, select the best parameters for the nearest neighbour and the decision tree. For the nearest neighbour these include:

  1. The number of neighbours k.
  2. The distance function ρ.

  For the decision trees, these include:

  1. Minsplit (or alternatively, the complexity parameter).
  2. The method used to add new features (information, classification, etc)

  `http://www.cse.chalmers.se/~chrdimi/downloads/fouille/clusterTraining2.txt`

- Then validate those parameters, by training them in the complete training set and measuring their performance in a separate test set. `http://www.cse.chalmers.se/~chrdimi/downloads/fouille/clusterTesting2.txt`

- How certain are you that their differences in testing are real? Quantify your uncertainty using bootstrapping.

- What are, in your opinion, the advantages and disadvantages of nearest neighbour over decision trees?

Make sure you train, validate and test both methods on the exact same data.