Nearest Neighbour Classification

Prof. Dr. H. H. Takada

Quantitative Research – Itaú Asset Management
Institute of Mathematics and Statistics – University of São Paulo

Do As Your Neighbour Does

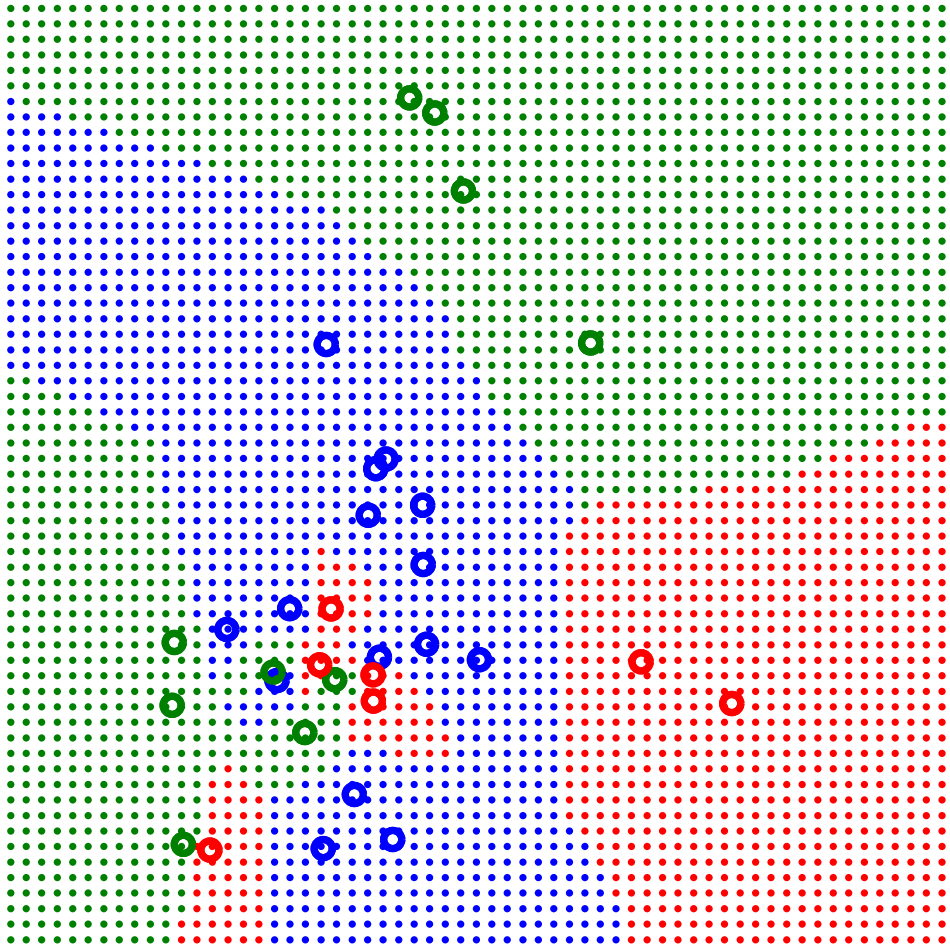
- Each input vector $\mathbf{x}$ has a corresponding class label, $c^n \in \{1, \ldots, C\}$. Given a dataset of N train examples, $\mathcal{D} = \{\mathbf{x}^n, c^n\}, n = 1, \ldots, N$, and a novel $\mathbf{x}$, we aim to return the correct class $c(\mathbf{x})$.
- Successful prediction typically relies on smoothness in the data.
- Nearest neighbour methods are a useful starting point since they readily encode basic smoothness intuitions and are easy to program.
- For novel $\mathbf{x}$, find the nearest input in the training set and use the class of this nearest input.

Squared Euclidean Distance

- For vectors $\mathbf{x}$ and $\mathbf{x}'$ representing two different datapoints, we measure 'nearness' using a dissimilarity function $d(\mathbf{x}, \mathbf{x}')$. A common dissimilarity is the squared Euclidean distance

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^{\mathsf{T}}(\mathbf{x} - \mathbf{x}')$$

  which can be more conveniently written $(\mathbf{x} - \mathbf{x}')^2$.
- Based on the squared Euclidean distance, the decision boundary is determined by the perpendicular bisectors of the closest training points with different training labels. This partitions the input space into regions classified equally and is called a Voronoi tessellation.



Here there are three classes, with training points given by the circles, along with their class. The dots indicate the class of the nearest training vector. The decision boundary is piecewise linear with each segment corresponding to the perpendicular bisector between two datapoints belonging to different classes, giving rise to a Voronoi tessellation of the input space.

Comments

The nearest neighbour algorithm is simple and intuitive. There are, however, some issues:

- A limitation of the Euclidean distance is that it does not take into account how the data is distributed. For example if the length scales of the components of the vector $\mathbf{x}$ vary greatly, the largest length scale will dominate the squared distance, with potentially useful class-specific information in other components of $\mathbf{x}$ lost.
- The Mahalanobis distance

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{x}')$$

  where $\boldsymbol{\Sigma}$ is the covariance matrix of the inputs (from all classes) can overcome some of these problems since it effectively rescales the input vector components.

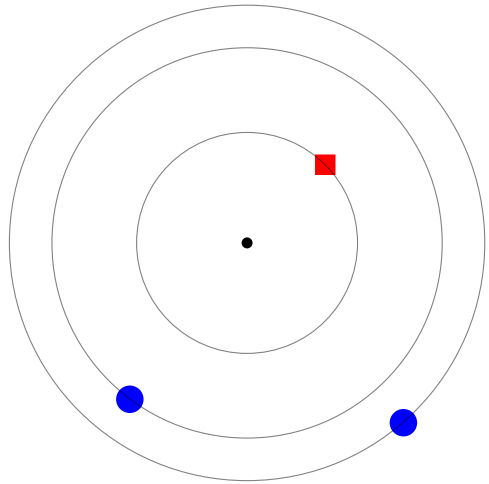
Comments

- The whole dataset needs to be stored to make a classification since the novel point must be compared to all of the train points. This can be partially addressed by removing datapoints which have little or no effect on the decision boundary.
- Particularly for low dimensional data, finding the nearest neighbour can be speeded up by various techniques that exploit the geometry of the space (see later).
- Each distance calculation can be expensive if the datapoints are high dimensional. Principal Components Analysis, is one way to address this and replaces $\mathbf{x}$ with a low dimensional projection $\mathbf{p}$. The Euclidean distance of two datapoints $\left(\mathbf{x}^a - \mathbf{x}^b\right)^2$ is then approximately given by $\left(\mathbf{p}^a - \mathbf{p}^b\right)^2$. This is both faster to compute and can also improve classification accuracy since only the large scale characteristics of the data are retained in the PCA projections.
- It is not clear how to deal with missing data or incorporate prior beliefs and domain knowledge.

K-Nearest Neighbours

- If your neighbour is simply mistaken (has an incorrect training class label), or is not a particularly representative example of his class, then these situations will typically result in an incorrect classification.
- By including more than the single nearest neighbour, we hope to make a more robust classifier with a smoother decision boundary (less swayed by single neighbour opinions).
- If we assume the Euclidean distance as the dissimilarity measure, the algorithm considers a hypersphere centred on the test point $\mathbf{x}$. The radius of the hypersphere is increased until it contains exactly K train inputs. The class label $c(\mathbf{x})$ is then given by the most numerous class within the hypersphere.

K-Nearest Neighbours

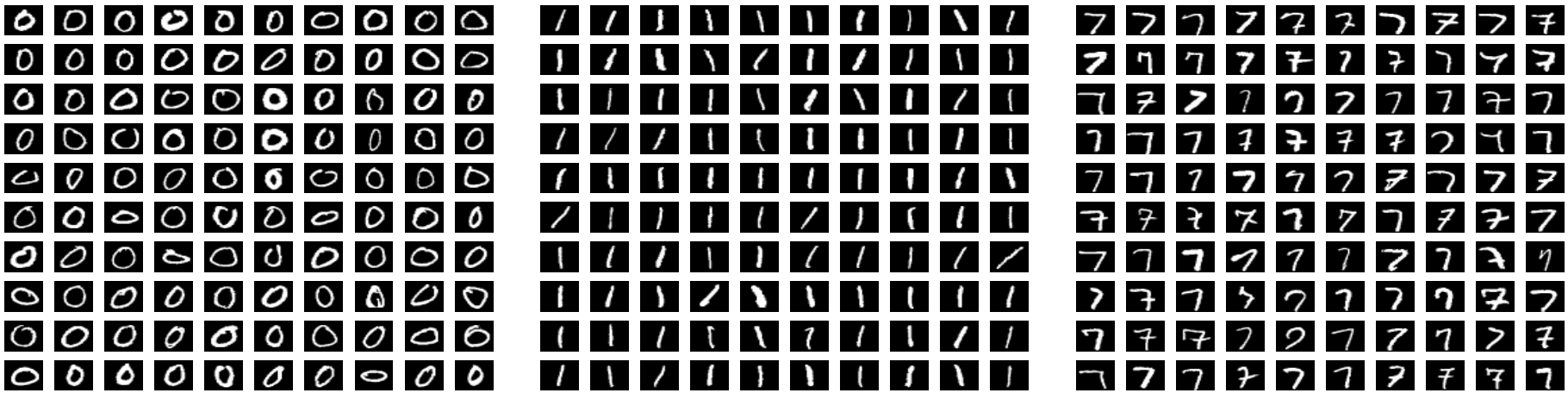


In K-nearest neighbours, we centre a hypersphere around the point we wish to classify (here the central dot). The inner circle corresponds to the nearest neighbour, a square. However, using the 3 nearest neighbours, we find that there are two round-class neighbours and one square-class neighbour – and we would therefore classify the central point as round-class. In the case of a tie, one may increase K until the tie is broken.

Choosing K

- Whilst there is some sense in making $K > 1$, there is certainly little sense in making $K = N$ (N being the number of training points).
- For K very large, all classifications will become the same – simply assign each novel $\mathbf{x}$ to the most numerous class in the train data.
- This suggests that there is an optimal intermediate setting of K which gives the best generalization performance.
- This can be determined using cross-validation.

Handwritten Digit Example



Some of the train examples of the digit zero, one and seven. There are 300 train examples of each of these three digit classes.

Ones versus Zeros

- Consider two classes of handwritten digits, zeros and ones.
- Each digit contains $28 \times 28 = 784$ pixels. The training data consists of 300 zeros, and 300 ones.
- To test the performance of the nearest neighbour method (based on Euclidean distance) we use an independent test set containing a further 600 digits
- The nearest neighbour method, applied to this data, correctly predicts the class label of all 600 test points.
- The reason for the high success rate is that examples of zeros and ones are sufficiently different that they can be easily distinguished.

Ones versus Sevens

- We repeat the above experiment, now using 300 training examples of ones, and 300 training examples of sevens. Again, 600 new test examples (containing 300 ones and 300 sevens) were used to assess the performance.
- This time, 18 errors are found using nearest neighbour classification – a 3% error rate for this two class problem.
- As an aside, the best Machine Learning methods classify real world digits (over all 10 classes) to an error of less than 1 percent – better than the performance of an 'average' human.

Probabilistic Interpretation of Nearest Neighbours

Consider the situation where we have data from two classes – class 0 and class 1. We make the following mixture model for data from class 0, placing a Gaussian on each datapoint:

$$p(\mathbf{x}|c=0) = \frac{1}{N_0} \sum_{n \in \text{ class } 0} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2 \mathbf{I}\right)$$

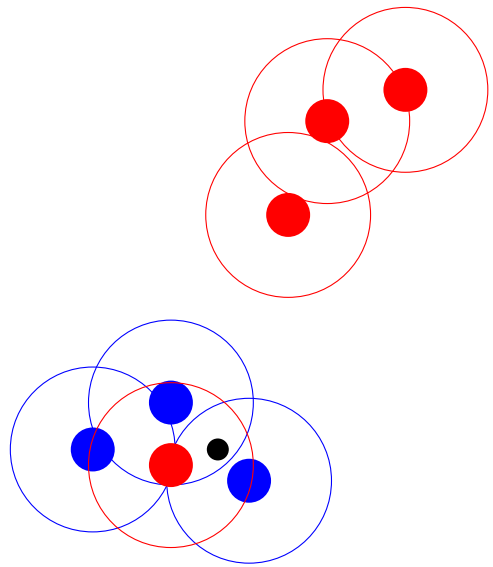
where D is the dimension of a datapoint $\mathbf{x}$ and N_0 are the number of train points of class 0, and σ^2 is the variance.
Similarly, for data from class 1:

$$p(\mathbf{x}|c=1) = \frac{1}{N_1} \sum_{n \in \text{ class } 1} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2 \mathbf{I}\right)$$

To classify a new datapoint $\mathbf{x}^*$, we use Bayes' rule

$$p(c=0|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=0)p(c=0) + p(\mathbf{x}^*|c=1)p(c=1)}$$

Probabilitic Interpretation



A probabilistic interpretation of nearest neighbours. For each class we use a mixture of Gaussians to model the data from that class $p(\mathbf{x}|c)$, placing at each training point an isotropic Gaussian of width σ^2. The width of each Gaussian is represented by the circle. In the limit $\sigma^2 \to 0$ a novel point (small black dot) is assigned the class of its nearest neighbour. For finite $\sigma^2 > 0$ the influence of non-nearest neighbours has an effect, resulting in a soft version of nearest neighbours.

Maximum Likelihood

- The maximum likelihood setting of $p(c = 0)$ is $N_0/(N_0 + N_1)$, and $p(c = 1) = N_1/(N_0 + N_1)$.
- An analogous expression holds for $p(c = 1|\mathbf{x}^*)$.
- To see which class is most likely we may use the ratio

$$\frac{p(c = 0|\mathbf{x}^*)}{p(c = 1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c = 0)p(c = 0)}{p(\mathbf{x}^*|c = 1)p(c = 1)} \tag{1}$$

  If this ratio is greater than one, we classify $\mathbf{x}^*$ as 0, otherwise 1.

Limiting case

$$\frac{p(c=0|\mathbf{x}^*)}{p(c=1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=1)p(c=1)}$$

- If σ^2 is very small, the numerator is dominated by that term for which datapoint $\mathbf{x}^{n_0}$ in class 0 is closest to the point $\mathbf{x}^*$. Similarly, the denominator will be dominated by that datapoint $\mathbf{x}^{n_1}$ in class 1 which is closest to $\mathbf{x}^*$.
- Taking the limit $\sigma^2 \to 0$, with certainty we classify $\mathbf{x}^*$ as class 0 if $\mathbf{x}^*$ is closer to $\mathbf{x}^{n_0}$ than to $\mathbf{x}^{n_1}$.
- The nearest (single) neighbour method is therefore recovered as the limiting case of a probabilistic generative model.

When your nearest neighbour is far away

- For a novel input $\mathbf{x}^*$ that is far from all training points, Nearest Neighbours, and its soft probabilistic variant will confidently classify $\mathbf{x}^*$ as belonging to the class of the nearest training point.
- This is arguably opposite to what we would like, namely that the classification should tend to the prior probabilities of the class based on the number of training data per class.
- A way to avoid this problem is, for each class, to include a fictitious large-variance mixture component at the mean of all the data, one for each class.
- For novel inputs close to the training data, this extra fictitious component will have no appreciable effect. However, as we move away from the high density regions of the training data, this additional fictitious component will dominate since it has larger variance than any of the other components.
- As the distance from $\mathbf{x}^*$ to each fictitious class point is the same, in the limit that $\mathbf{x}^*$ is far from the training data, the effect is that no class information from the position of $\mathbf{x}^*$ occurs.

Matlab

```
setup; demoNearNeigh;
```