K-Nearest Neighbors

Olena Smotrova

02/02/2018

Contents

Classification. K-Nearest neighbors.

Predicting a qualitative responses for an observation can be referred to as *classifying* that observation, since it involves assigning the observation to a category.

Many approaches attempt to estimate the conditional distribution of Y given X, and then classify a given observation to the class with highest estimated probability. One such method is the *K-nearest neighbors* (KNN) classifier. Given a positive integer K and a test observation x_0, represented by N_0. It then estimates the conditional probability for class j as the fraction of points N_0 whose response values equal j:

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

Finally, KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability. There is not a strong relationship between the *training error rate* and *test error rate.* With $K = 1$, the KNN training error rate equals 0, but the test error rate may be quite high. In general, as we use more flexible classification methods, the training error will decline but the test error rate may not. As K decreases, the method becomes more flexible.

Improving Performance.

Cross-Validation.

To estimate the error of the KNN for a particular K we could use *n-fold cross-validation* (n-fold CV)

- Divide the training set into n equal pieces: $S_1, S_2, ...S_n$
- Classify each point in S_i using KNN with training set $S - S_i$, $i = 1, ..., n$
- Define ϵ_i as a fraction of S_i that is incorrectly classified
- Take the average estimated error KNN $= \frac{1}{n} \sum_{i=1}^{n} \epsilon_i$

An extreme type of cross-validation is n-fold CV on a training set of size n. If we want to estimate the error of KNN, this amounts to classifying each training point by running KNN on the remaining (*n-1*) points, and then looking at the fraction of mistakes made. It is commonly called *leave-one-out-cross-validation* (LOOCV).

Choose a metric

Let X be the space in winch data lie. A distance function $d : X \times X \to \mathbb{R}$ is a *metric* if it is satisfies following properties:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0 \Leftrightarrow x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Measuring distance in $\mathbb{R}^m$ is l_p distance, $p \geq 1$:

$$\parallel x - y \parallel_p = \sqrt[p]{\sum_{i=1}^{m} |x_i - y_i|^p}$$

- l_2 distance is Euclidean distance (usual choice)
- l_∞ distance: $\parallel x - y \parallel_\infty = max_i |x_i - y_i|$

Data

Description

MNIST ("Modified National Institute of Standards and Technology") is the de facto "hello world" data set of computer vision. Since its release in 1999, this classic data set of handwritten images has served as the basis for bench marking classification algorithms.

The data file contains gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. Each row in data set represents an image stretched into a vector with 784 coordinates.

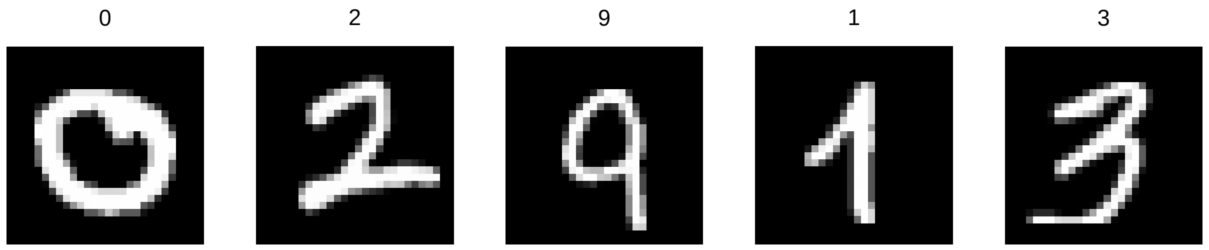
The data set, (train.csv), has 785 columns. The first column, called *label*, is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the data set has a name like $pixelx$, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as $x = i * 28 + j$, where i and j are integers between 0 and 27, inclusive. Then $pixelx$ is located on row i and column j of a 28 x 28 matrix, (indexing by zero). For example, $pixel31$ indicates the pixel that is in the fourth column from the left, and the second row from the top. Data source (*Kaggle, 2012*)

```
## 'data.frame':    42000 obs. of  785 variables:
##  $ label   : int  1 0 1 4 0 0 7 3 5 3 ...
##  $ pixel0  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel1  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel2  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel3  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel4  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel5  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel6  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel7  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel8  : int  0 0 0 0 0 0 0 0 0 0 ...
##   [list output truncated]
```

Visualization

There are some randomly selected examples of handwritten digits and their labels from the data set.



KNN in R

We will perform KNN using the *knn*() function, winch is part of *class* library. The function requires

- A matrix or a data frame of training set cases
- A matrix or a data frame of test set cases
- A vector containing the class labels for the training observations
- A value for K, the number of nearest neighbors to be used by classifier

```r
library(caTools)
library(class)
```

We use *sample.split*() function from *caTools* to split data set randomly into training and test sets.

```r
set.seed(1000)
spl = sample.split(digits$label, 0.75)

digitsTrain = subset(digits, spl == TRUE)
digitsTest = subset(digits, spl == FALSE)

labelTrain = digitsTrain$label
```

Note. A seed must be set in order to ensure reproducibility of results. In *knn*() if several observations are tied as nearest neighbors, R randomly break the tie.

```r
knn.pred = knn(digitsTrain[ ,2:785], digitsTest[ , 2:785], labelTrain, k = 3 )
```

Error test rate defines as a fraction of incorrectly classified observations.

```r
table(knn.pred, digitsTest$label)
```

```
##
## knn.pred    0    1    2    3    4    5    6    7    8    9
##        0 1021    0    2    0    1    0    7    1    4    3
##        1    1 1165    8    0    4    0    3   17   13    3
##        2    0    1 1001    5    0    0    1    1    3    1
##        3    1    0    4 1058    0   14    0    0   15    6
##          4     0     0     0     0   986     0     2     2     4    15
##          5     3     0     3    14     0   918     3     0    18     2
##          6     5     0     2     1     3    12  1018     0     7     1
##          7     1     2    19     1     1     0     0  1072     5    21
##          8     0     2     3     7     0     0     0     0   935     2
##          9     1     1     2     2    23     5     0     7    12   993
```

```r
mean(knn.pred == digitsTest$label)
```

```
## [1] 0.9682857
```

The results of KNN with K=3 are very good for these data. Visualizations below show five randomly selected examples of digits with labels defined by KNN classifier.