The Curse of Dimensionality

Ruoqing Zhu

Last Updated: October 03, 2024

Contents

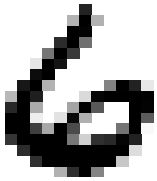
Example: Handwritten Digit Data

Let's consider another example using the handwritten digit data. Each observation in this data is a 16×16 pixel image. Hence, the total number of variables is 256. At each pixel, we have the gray scale as the numerical value.

```
  # Handwritten Digit Recognition Data
  library(ElemStatLearn)

  # the first column is the true digit
  dim(zip.train)
## [1] 7291  257
  dim(zip.test)
## [1] 2007  257

  # look at one sample
  image(zip2image(zip.train, 1), col=gray(256:0/256), zlim=c(0,1),
        xlab="", ylab="", axes = FALSE)
## [1] "digit  6  taken"
```



We use 3NN to predict all samples in the testing data. The model is fairly accurate.

```r
  library(class)
  # fit 3nn model and calculate the error
  knn.fit <- knn(zip.train[, 2:257], zip.test[, 2:257], zip.train[, 1], k=3)

  # overall prediction error
  mean(knn.fit != zip.test[,1])
## [1] 0.05430992

  # the confusion matrix
  table(knn.fit, zip.test[,1])
##
## knn.fit   0   1   2   3   4   5   6   7   8   9
##       0 355   0   7   1   0   3   3   0   4   1
##       1   0 257   0   0   2   0   0   1   0   0
##       2   2   0 183   2   0   2   1   1   0   0
##       3   0   0   1 154   0   4   0   1   5   0
##       4   0   4   1   0 182   0   2   4   0   3
##       5   0   0   0   7   2 146   1   0   1   1
##       6   0   2   1   0   2   0 163   0   1   0
##       7   1   1   2   1   2   0   0 138   1   4
##       8   0   0   3   0   1   1   0   1 152   0
##       9   1   0   0   1   9   4   0   1   2 168
```

The Curse of Dimensionality

Many of the practical problems we encounter today are high-dimensional. The resolution of the handwritten digit example is $16 \times 16 = 256$. Genetic studies often involves more than 25K gene expressions, etc. For a given sample size n, as the number of variables p increases, the data becomes very sparse. Nearest neighbor methods usually do not perform very well on high-dimensional data. This is because for any given target point, there will not be enough training data that lies close enough. To see this, let's consider a p-dimensional hyper-cube. Suppose we have $n = 1000$ observations uniformly spread out in this cube, and we are interested in predicting a target point with $k = 10$ neighbors. If these neighbors are really close to the target point, then this would be a good estimation with small bias. Suppose $p = 2$. We know that if we take a cube (square) with both height and width $l = 0.1$, then there will be $1000 \times 0.1^2 = 10$ observations within the square. In general, we have the relationship

$$l^p = \frac{k}{n}$$

Try different p, we have

- If $p = 2$, $l = 0.1$
- If $p = 10$, $l = 0.63$
- If $p = 100$, $l = 0.955$

This implies that if we have 100 dimensions, then the nearest 10 observations would be 0.955 away from the target point at each dimension, this is almost at the other corner of the cube. Hence there will be a very large bias. Decreasing k does not help much in this situation since even the closest point can be really far away, and the model would have large variance.

add highd.png here

An Experiment

Let's try a small experiment on this high-dimensional issue. Suppose we have a 10 dimensional setting with covariates uniformly distributed on $(0,1)^{10}$, and Suppose the outcome is a regression model that depends only on the first variable. The following code setup this model. And we want to know what is the expected performance. Recall the idea of simulation in the Ridge regression, we could repeatedly perform this many times to estimate the **approximation error**:

$$E\left[\left(\widehat{f}(x_0) - f(x_0)\right)^2\right]$$

where $x_0 = c(0, 0, 0, \ldots, 0)$.

```
  # Let's try a new package FNN
  library(FNN)
## Warning: package 'FNN' was built under R version 4.3.3
  p = 10
  n = 100

  # the target prediction point
  # be careful that this needs to be a 1xp matrix
  x0 = matrix(rep(0, p), nrow = 1, ncol = p)

  # number of simulations
  nsim = 300

  # vector to store predicted values
  allerror = rep(NA, nsim)

  for (l in 1:nsim)
  {
    # generate data
    X = matrix(runif(n*p), nrow = n, ncol = p)
    y = X[, 1] + X[, 2] + X[, 3] + rnorm(n, sd = 0.5)

    # "5-nearest neighbor" regression using the FNN package
    # for this question, use the "brute force" algorithm to search for the NNs
    knn.fit = knn.reg(train = X, test = x0, y = y,
                      k = 5, algorithm = "brute")

    # record the prediction error of this run
    # the truth f(x_0) is 0
    allerror[l] = (knn.fit$pred - 0)^2
  }

  # the prediction error
  mean(allerror)
## [1] 1.012854
```

Practice question

Use the same code, calculate the prediction error at $x_0 = c(1, 1, 1, 0, \ldots, 0)$. What are the prediction errors if $p = 5, 10$ and 20?

```r
# Let's try a new package FNN
library(FNN)
p = 20
n = 100

# the target prediction point
# be careful that this needs to be a 1xp matrix
x0 = matrix(c(rep(1, 3), rep(0, p-3)), nrow = 1, ncol = p)

# number of simulations
nsim = 300

# vector to store predicted values
allerror = rep(NA, nsim)

for (l in 1:nsim)
{
  # generate data
  X = matrix(runif(n*p), nrow = n, ncol = p)
  y = X[, 1] + X[, 2] + X[, 3] + rnorm(n, sd = 0.5)

  # "5-nearest neighbor" regression using the FNN package
  # for this question, use the "brute force" algorithm to search for the NNs
  knn.fit = knn.reg(train = X, test = x0, y = y,
                    k = 5, algorithm = "brute")

  # record the prediction error of this run
  # the truth f(x0) is 3
  allerror[l] = (knn.fit$pred - 3)^2
}

# the prediction error
mean(allerror)
```

Discussion

However, why our model performs well in the handwritten digit example? There is possibly (approximately) a lower dimensional representation of the data so that when you evaluate the distance on the high-dimensional space, it is just as effective as working on the low dimensional space. Moreover, the digit classes can be effectively separated if we only focus this low dimensional space. Hence, the performance can be satisfactory.

Dimension reduction is an important topic in statistical learning and machine learning. Many methods, such as sliced inverse regression (Li, 1991) and UMAP (McInnes, et al. 2018) have been developed based on this concept.

add manifold.png here

Think about the previous example of the dimensionality issue. What if all covariates are highly linearly dependent? Would that lead to an approximately low dimensional representation and potentially improve the prediction accuracy? Can you use a simulation study to confirm that?