Lab 10: k-NN and a Comparison of Classification Methods

Johnny Hong

Stat 154, Fall 2017

Introduction

In this lab we explore study the k-nearest neighbors (k-NN) algorithm. Then we will compare the classification algorithms we have studied so far (logistic regression, LDA, QDA, k-NN) based on six different synthetic datasets.

k-Nearest Neighbors

The idea of k-NN is to classify a new observation based on the majority vote of the k nearest observations in the training set. Let $S_k(x)$ be the set of k training observations that are closest to x. If all the predictors are real-valued, the typical choice for the distance measure is the Euclidean distance: given $x, w \in \mathbb{R}^p$,

$$d(x, w) = \sqrt{\sum_{j=1}^{p} (x_j - w_j)^2}.$$

Once $S_k(x)$ has been identified, we can compute the estimated conditional probability that an observation with predictor x belongs to class c as

$$\hat{P}(Y = c|x) = \frac{1}{k} \sum_{i \in S_k(x)} I(y_i = c),$$

for each $c \in \{1, ..., K\}$, where y_i is the class of the ith observation.

k-NN classifies an observation with predictor vector x to class c^* if $c^* \in \arg\max_c \hat{P}(Y = c|x)$. Whenever a tie arises, c^* is randomly chosen from the set of maximizers.

Your turn

- Implement a function called `my_knn()` that generates k-NN predictions.
- `my_knn()` should accept the following inputs:
  - `X_train`: an $n_{\text{train}} \times p$ predictor matrix
  - `X_test`: an $n_{\text{test}} \times p$ predictor matrix
  - `Y_train`: the response vector for the training set
  - `k`: the number of neighbors

- `my_knn()` should return a factor vector containing the k-NN predictions for the test set.
- Check your answer with `knn()` in the R package `class`.

```
train_idx <- sample(nrow(iris), 90)
train_set <- iris[train_idx, ]
test_set <- iris[-train_idx, ]

my_knn_pred <- my_knn(train_set[, -5], test_set[, -5], train_set$Species, k=1)
knn_pred <- knn(train_set[, -5], test_set[, -5], train_set$Species, k=1)
table(my_knn_pred == knn_pred)
```

```
##
## TRUE
##   60
```

k-NN CV

The number of neighbors k is a hyperparameter that has to be tuned. Similar to other machine learning algorithm, cross-validation can be used.

Your turn

- Implement a function called `find_k_CV()` that finds the optimal k based on CV-misclassification rate.
- `find_k_CV` should accept the following inputs:
  - `X_train`: an $n_{\text{train}} \times p$ predictor matrix
  - `Y_train`: the response vector for the training set
  - `k`: an integer vector, each of which is a candidate for the number of neighbors, defaulted to `1:10`
  - `nfold`: a scalar indicating the number of folds
- `find_k_CV` should return a scalar, which is the optimal number of neighbors (in terms of minimizing the CV-misclassification rate)

The output of your function should be like the following:

```
find_k_CV(train_set[, -5], train_set[, 5])
```

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

Comparisons

In this section, we will follow closely Chapter 4.5 in ISL. The objective is to compare the predictive power for various classification algorithms we have studied so far under various hypothetical scenarios. See ISL for a more detailed discussion.

Your turn

- Simulate six datasets (each has $p = 2$ predictors):
  - Scenario 1: Simulate 100 observations, half are from $N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$, half from $N\left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$. Treat the first half as class 1 and the rest as class 2.
  - Scenario 2: Simulate 100 observations, half are from $N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & -0.5\\-0.5 & 1\end{bmatrix}\right)$, half from $N\left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1 & -0.5\\-0.5 & 1\end{bmatrix}\right)$. Treat the first half as class 1 and the rest as class 2.
  - Scenario 3: Simulate 100 observations, half are from $t_4\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}\right)$, half from $t_4\left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}\right)$. Treat the first half as class 1 and the rest as class 2.
  - Scenario 4: Simulate 100 observations, half are from $N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}\right)$, half from $N\left(\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1 & -0.5\\-0.5 & 1\end{bmatrix}\right)$. Treat the first half as class 1 and the rest as class 2.
  - Scenario 5: Simulate two independent sequences of $N(0,1)$ random variables $(X_{1,1}, X_{2,1}, ...X_{100,1})$ and $(X_{1,2}, X_{2,2}, ...X_{100,2})$. Let $Y_i \sim \text{Ber}(p_i)$ for $i = 1, ..., 100$, where
  $$\text{logit}(p_i) = \beta_0 + \beta_1 X_{i,1}^2 + \beta_2 X_{i,2}^2 + \beta_3 X_{i,1} X_{i,2},$$
  with $(\beta_0, \beta_1, \beta_2, \beta_3) = (0, 2, -1, 2)$.
  - Scenario 6: Simulate two independent sequences of $N(0,1)$ random variables $(X_{1,1}, X_{2,1}, ...X_{n,1})$ and $(X_{1,2}, X_{2,2}, ...X_{n,2})$. Let
  $$Y_i = \begin{cases} 1 & \text{if } X_{i,1}^2 + X_{i,2}^2 > \chi_2^2(0.5) \approx 1.386 \\ 0 & \text{otherwise.} \end{cases},$$
  where $\chi_2^2(0.5)$ is the median of a χ^2 distribution with 2 degrees of freedom.
- Remark: You can use the following function `gen_datasets` to simulate the six datasets. The code for drawing from a non-central multivariate t-distribution is from https://stats.stackexchange.com/questions/68476/drawing-from-the-multivariate-students-t-distribution.

```
set.seed(100)

expit <- function(x) {
  exp(x) / (1 + exp(x))
}

gen_datasets <- function() {
  id <- diag(c(1, 1))
```

```
  df1 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                    rbind(rmvnorm(50, mean=c(0, 0), sigma = id),
                          rmvnorm(50, mean=c(1, 1), sigma = id)))

  covmat <- matrix(c(1, -0.5, -0.5, 1), nrow=2)
  df2 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                    rbind(rmvnorm(50, mean=c(0, 0), sigma = covmat),
                          rmvnorm(50, mean=c(1, 1), sigma = covmat)))

  mu <- c(0, 0); sigma <- matrix(c(1, 1/2, 1/2, 1), 2); nu <- 4
  n <- 50 # Number of draws
  x_first <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
                 * sqrt(nu / rchisq(n, nu))) + mu)
  mu <- c(1, 1); sigma <- matrix(c(1, 1/2, 1/2, 1), 2); nu <- 4
  n <- 50 # Number of draws
  x_second <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
                  * sqrt(nu / rchisq(n, nu))) + mu)
  df3 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                    rbind(x_first, x_second))


  covmat2 <- matrix(c(1, 0.5, 0.5, 1), nrow=2)
  df4 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                    rbind(rmvnorm(50, mean=c(0, 0), sigma = covmat2),
                          rmvnorm(50, mean=c(1, 1), sigma = covmat)))

  x <- matrix(rnorm(200), ncol=2)
  df5_temp <- data.frame(x ^ 2, x[, 1] * x[, 2])

  beta <- c(0, 2, -1, -2)
  y <- apply(df5_temp, 1, function(row) {
    p <- expit(sum(c(1, row) * beta))
    sample(x=c(0, 1), size=1, prob=c(1-p, p))
  })
  df5 <- data.frame(y=factor(y), x)

  x <- matrix(rnorm(200), ncol=2)
  y <- 1 * (x[, 1]^2 + x[, 2]^2 > qchisq(p=0.5, df=2))
  df6 <- data.frame(y=factor(y), x)

  list(df1, df2, df3, df4, df5, df6)
}
```

- Repeat the following 100 times:

  - Simulate 6 datasets via `gen_datasets()`.
  - For each dataset, use 80% of the data as the training set and the remaining 20% as the test set.
    * Fit logistic regression, LDA, QDA, k-NN with 1 neighbor, and k-NN-CV using the training set.
    * For each model, compute the test error rate and generate predictions on the test set.
  - Store the 5×6 matrix of error rates.
- You should now have a $5 \times 6 \times 100$ array of test error rates.
- For each of the six scenario, make a boxplot of test error rates. See Figure 4.10 and Figure 4.11 in ISL.