STAT 3690 Lecture 32

zhiyanggeezhou.github.io

Zhiyang Zhou (zhiyang.zhou@umanitoba.ca)

Apr 18, 2022

Misclassification/error rate

- Population: $\Pr(Y \neq h(\mathbf{X}))$
  - $h(\cdot)$: the classifier to be evaluated
- Apparent estimation
  - Implementation
    1. Fit a classifier according to training data
    2. Apply the fitted classifier to training data as well
    3. Estimate the error rate by the misclassification proportion
  - Comments
    * Training and testing with identical data points
    * Severe underestimation likely
- Parametric estimation
  - Implementation
    1. Express $\Pr(Y \neq h(\mathbf{X}))$ in terms of unknown parameters
    2. Plug in estimates of unknown parameters
  - Comment
    * Able to derive the analytical form of $\Pr(Y \neq h(\mathbf{X}))$ in rare cases
    * Underestimation likely

For $K=2$ and $X \mid Y=k \sim MVN(\mu_k, \Sigma)$,

the error rate of LDA classifier $h(X)$

$$\begin{aligned}
&= \Pr(Y \neq h(X)) \\
&= \Pr(Y=1, h(X)=2) + \Pr(Y=2, h(X)=1) \\
&= \Pr(Y=1, \delta_1(X) < \delta_2(X)) + \Pr(Y=2, \delta_1(X) > \delta_2(X)) \\
&= \pi_1 \Pr(\delta_1(X) < \delta_2(X) \mid Y=1) + \pi_2 \Pr(\delta_1(X) > \delta_2(X) \mid Y=2)
\end{aligned}$$

Let $U = \delta_1(X) - \delta_2(X) = X^T \Sigma^{-1}(\mu_1 - \mu_2) - \frac{1}{2}\mu_1^T \Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T \Sigma^{-1}\mu_2 + \ln(\pi_1/\pi_2)$, then

$$\begin{aligned}
E(U \mid Y=1) &= \mu_1^T \Sigma^{-1}(\mu_1 - \mu_2) - \tfrac{1}{2}\mu_1^T \Sigma^{-1}\mu_1 + \tfrac{1}{2}\mu_2^T \Sigma^{-1}\mu_2 + \ln(\pi_1/\pi_2) \\
&= \tfrac{1}{2}\mu_1^T \Sigma^{-1}\mu_1 - \mu_1^T \Sigma^{-1}\mu_2 + \tfrac{1}{2}\mu_2^T \Sigma^{-1}\mu_2 + \ln(\pi_1/\pi_2) \\
&= \tfrac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) + \ln(\pi_1/\pi_2)
\end{aligned}$$

$$E(U \mid Y=2) = -\tfrac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) + \ln(\pi_1/\pi_2)$$

$$\operatorname{var}(U \mid Y=1) = (\mu_1 - \mu_2)^T \Sigma^{-1} \cancel{\Sigma \Sigma^{-1}} (\mu_1 - \mu_2)$$

$$\operatorname{var}(U \mid Y=2) = (\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)$$

Further,

$$\frac{U - \frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) - \ln(\pi_1/\pi_2)}{\sqrt{(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)}} \,\Big|\, Y=1 \sim N(0,1)$$

$$\frac{U + \frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) - \ln(\pi_1/\pi_2)}{\sqrt{(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)}} \,\Big|\, Y=2 \sim N(0,1).$$

So, $\Pr(Y \neq h(X))$

$$\begin{aligned}
&= \pi_1 \Pr(U < 0 \mid Y=1) + \pi_2 \Pr(U > 0 \mid Y=2) \\
&= \pi_1 \Phi\left(\frac{-\frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) - \ln(\pi_1/\pi_2)}{\sqrt{(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)}}\right) \\
&\quad + \pi_2 \Phi\left(\frac{-\frac{1}{2}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) + \ln(\pi_1/\pi_2)}{\sqrt{(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2)}}\right),
\end{aligned}$$

where $\Phi(\cdot)$ is the standard normal cdf.

- Estimation via M-fold cross validation (CV)
  - Implementation
    1. The dataset is randomly partitioned into M chunks.
    2. Train one classifier upon each combination of $M-1$ chunks.
    3. Apply each classifier to the corresponding remaining chunk and compute the empirical error rate.
    4. Estimate the population error rate by averaging these M empirical error rates.
  - Comment
    * Leave-one-out CV $\Leftrightarrow$ n-fold CV
- Estimation via $M \times L$-fold CV
  - Implementation
    1. Repeat the four steps of M-fold CV L times.
    2. Average all the ML resulting empirical error rates.
  - Comment
    * $M \times 1$-fold CV $\Leftrightarrow$ M-fold CV

```
options(digits = 4)
set.seed(3690)
L = 1; M = nrow(iris) # Leave-one-out CV
# L = 1; M = 10 # 10 fold CV
# L = 10; M = 10 # 10by10 fold CV

# initiation
errLda1 = matrix(0, nrow = L, ncol = M)
errLda2 = matrix(0, nrow = L, ncol = M)
errQda1 = matrix(0, nrow = L, ncol = M)
errQda2 = matrix(0, nrow = L, ncol = M)

for (l in 1:L){
  idx_new = sample(1:nrow(iris), size = nrow(iris))
  folds = cut(1:nrow(iris), breaks = M, labels=FALSE)
  # follow formulas
  for (m in 1:M){
    # Segement your data by fold using the which() function
    picked = idx_new[which(folds == m, arr.ind=TRUE)]
    train = iris[-picked,]
    Xtrain = train[, !(names(train) %in% c("Species"))]
    Ytrain = train$Species
    test = iris[picked,]
    Xtest = test[, !(names(test) %in% c("Species"))]
    Ytest = test[, names(test) %in% c("Species")]

    labels = unique(iris$Species)
    K = length(labels)
    p = ncol(Xtrain)
    n = nrow(Xtrain)
    nks = numeric(K)
    piks = numeric(K)
    Muks = matrix(0, nrow = K, ncol = p)
    Sigmaks = list()
    for (k in 1:K){
```

```
    Xtrain_k = Xtrain[Ytrain == labels[k],]
    nks[k] = nrow(Xtrain_k)
    piks[k] = nks[k]/n
    Muks[k,] = colMeans(Xtrain_k)
    Sigmaks[[k]] = cov(Xtrain_k)
    if (k==1){
      SigmaPool = Sigmaks[[k]] * (nks[k]-1)
    }else{
      SigmaPool = SigmaPool + Sigmaks[[k]] * (nks[k]-1)
    }
  }
  SigmaPool = SigmaPool/(n-1)
  SigmaPoolInv = solve(SigmaPool)

  deltaksLda = matrix(0, nrow = nrow(Xtest), ncol = K)
  deltaksQda = matrix(0, nrow = nrow(Xtest), ncol = K)
  for (k in 1:K){
    # LDA
    deltaksLda[,k] = as.matrix(Xtest) %*% SigmaPoolInv %*% Muks[k,] -
      .5* as.vector(t(Muks[k,]) %*% SigmaPoolInv %*% Muks[k,]) +
      log(piks[k])

    # QDA
    SigmakInv = solve(Sigmaks[[k]])
    deltaksQda[,k] = -diag(as.matrix(Xtest) %*% SigmakInv %*% t(as.matrix(Xtest))) +
      2* as.matrix(Xtest) %*% SigmakInv %*% Muks[k,] -
      as.vector(t(Muks[k,]) %*% SigmakInv %*% Muks[k,]) +
      2* log(piks[k]) -
      log(det(Sigmaks[[k]]))
  }
  resLda = apply(deltaksLda, 1, FUN = function(x){labels[which.max(x)]})
  resQda = apply(deltaksQda, 1, FUN = function(x){labels[which.max(x)]})
  errLda1[l, m] = mean(Ytest != resLda)
  errQda1[l, m] = mean(Ytest != resQda)
}

# use MASS
for (m in 1:M){
  # Segment your data using the which() function
  picked = idx_new[which(folds == m, arr.ind=TRUE)]
  train = iris[-picked,]
  Xtrain = train[, !(names(train) %in% c("Species"))]
  Ytrain = train$Species
  test = iris[picked,]
  Xtest = test[, !(names(test) %in% c("Species"))]
  Ytest = test[, names(test) %in% c("Species")]

  for (k in 1:K){
    # LDA
    objLda = MASS::lda(Xtrain, Ytrain, method = "moment")
    # QDA
    objQda = MASS::qda(Xtrain, Ytrain, method = "moment")
  }
```

```
    errLda2[l, m] = mean(Ytest != predict(objLda, Xtest)$class)
    errQda2[l, m] = mean(Ytest != predict(objQda, Xtest)$class)
  }
}

mean(errLda1)
mean(errLda2)
mean(errQda1)
mean(errQda2)
```