STAT 3690 Lecture 31

zhiyanggeezhou.github.io

Zhiyang Zhou (zhiyang.zhou@umanitoba.ca)

Apr 13, 2022

Classification

- Predictive task in which the response takes values across K discrete categories (i.e., not continuous)
  - For one subject, to predict its class label Y when its features $\mathbf{X}$ is observed
  - Binary classification: $K = 2$
  - Having training data with known class labels
  - E.g.
    * Given scanned handwritten digits: 28×28 grid of pixels each reflecting the value of grey scale; see Lec 23. From vectorized pictures determine what digit was written.
    * Predicting the region of Italy in which a brand of olive oil was made, based on its chemical composition; see Lec 29.

- Bayes classifier
  - Classify according to posterior $\Pr(Y = k \mid \mathbf{X} = \boldsymbol{x}) = f_k(\boldsymbol{x})\pi_k / \sum_{\ell=1}^{K} f_\ell(\boldsymbol{x})\pi_\ell$, $k = 1, \ldots, K$
    * $f_k(\boldsymbol{x})$: the probability density/mass function of $\mathbf{X}$ conditioning on Class k
    * $\pi_k = \Pr(Y = k)$: prior probability of Class k
  - Bayes classifier

$$h(\boldsymbol{x}) = \arg\max_{k=1,\ldots,K} \Pr(Y = k \mid \mathbf{X} = \boldsymbol{x}) = \arg\max_{k=1,\ldots,K} f_k(\boldsymbol{x})\pi_k$$

Linear discriminant analysis (LDA)

- Assuming $f_k(\boldsymbol{x})$ = density of $MVN_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$
- LDA classifier

$$h(\boldsymbol{x}) = \arg\max_{k=1,\ldots,K} \delta_k(\boldsymbol{x})$$

  - Discriminant functions $\delta_k(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln \pi_k$
    * Linear functions with respect to $\boldsymbol{x}$
- Empirical version
  - Training data: $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{1, \ldots, K\}$, $i = 1, \ldots, n$
    * n_k: the number of training observations in class k, $k = 1, \ldots, K$
  - Estimation for $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}$ and π_k
    * $\hat{\pi}_k = n_k/n$
    * $\hat{\boldsymbol{\mu}}_k = n_k^{-1} \sum_{i=1}^{n} \boldsymbol{x}_i \cdot \mathbf{1}(y_i = k)$
    * $\widehat{\boldsymbol{\Sigma}} = (n-1)^{-1} \sum_{k=1}^{K} \sum_{i=1}^{n} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top \cdot \mathbf{1}(y_i = k)$
  - Empirical LDA classifier

$$\hat{h}(\boldsymbol{x}) = \arg\max_{k=1,\ldots,K} \hat{\delta}_k(\boldsymbol{x})$$

* $\hat{\delta}_k(\boldsymbol{x}) = \boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k - \frac{1}{2} \hat{\boldsymbol{\mu}}_k^\top \widehat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k + \ln \hat{\pi}_k$

- Example (Fisher's or Anderson's `iris` data)
  - 50 flowers from each of 3 species of iris: setosa, versicolor, and virginica.
  - Measurements in centimeters of the variables sepal length and width and petal length and width.

```
options(digits = 4)
set.seed(3690)
picked = sample.int(nrow(iris), size = floor(nrow(iris)/3))
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")]

# follow formulas
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-K)
SigmaPoolInv = solve(SigmaPool)

deltaksLda = matrix(0, nrow = nrow(Xtest), ncol = K)
for (k in 1:K){
  deltaksLda[,k] = as.matrix(Xtest) %*% SigmaPoolInv %*% Muks[k,] -
    .5* as.vector(t(Muks[k,]) %*% SigmaPoolInv %*% Muks[k,]) +
    log(piks[k])
}
(resLda = apply(deltaksLda, 1, FUN = function(x){labels[which.max(x)]}))
# use MASS::lda
objLda = MASS::lda(Xtrain, Ytrain, method   = "moment")
# comparison
mean(resLda == predict(objLda, Xtest)$class)
mean(Ytest != resLda)
```

Quadratic discriminant analysis (QDA)

- Assuming $f_k(\boldsymbol{x})$ = density of $MVN_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
- QDA classifier
$$h(\boldsymbol{x}) = \arg\max_{k=1,\dots,K} \delta_k(\boldsymbol{x})$$
  - Discriminant functions $\delta_k(\boldsymbol{x}) = -\boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{x} + 2\boldsymbol{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k + 2\ln\pi_k - \ln\det\boldsymbol{\Sigma}_k$
    - Quadratic functions with respect to $\boldsymbol{x}$
- Empirical version
  - Training data: $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{1,\dots,K\}$, $i = 1,\dots,n$
    - n_k: the number of training observations in class k, $k = 1,\dots,K$
  - Estimation for $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}$ and π_k
    - $\hat{\pi}_k = n_k/n$
    - $\hat{\boldsymbol{\mu}}_k = n_k^{-1}\sum_{i=1}^n \boldsymbol{x}_i \cdot \mathbf{1}(y_i = k)$
    - $\widehat{\boldsymbol{\Sigma}}_k = (n_k - 1)^{-1}\sum_{i=1}^n (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top \cdot \mathbf{1}(y_i = k)$
  - Empirical classifier
$$\hat{h}(\boldsymbol{x}) = \arg\max_{k=1,\dots,K} \hat{\delta}_k(\boldsymbol{x})$$
    - $\hat{\delta}_k(\boldsymbol{x}) = -\boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}_k^{-1}\boldsymbol{x} + 2\boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}_k^{-1}\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_k^\top \widehat{\boldsymbol{\Sigma}}_k^{-1}\hat{\boldsymbol{\mu}}_k + 2\ln\hat{\pi}_k - \ln\det\widehat{\boldsymbol{\Sigma}}_k$

- Example (`iris` data, con'd)

```
deltaksQda = matrix(0, nrow = nrow(Xtest), ncol = K)
for (k in 1:K){
  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]]))
}
(resQda = apply(deltaksQda, 1, FUN = function(x){labels[which.max(x)]}))
# use MASS::qda
objQda = MASS::qda(Xtrain, Ytrain, method   = "moment")
# comparison
mean(resQda == predict(objQda, Xtest)$class)
mean(Ytest != resQda)
```