STAT 3690 Lecture Note

Part IX: Classification

Zhiyang Zhou (zhiyang.zhou@umanitoba.ca, zhiyanggeezhou.github.io)

2023/Mar/27 16:04:04:36

Classification

- Predictive task in which the response takes values across K discrete categories (i.e., not continuous)
  - Having training data with known class labels
  - Predict one subject's label Y according to p-vector $\mathbf{X}$
  - Binary classification: $K = 2$
  - E.g.
    * Given a scanned handwritten digit, determine what digit was written.
    * Predict the region of Italy in which a sample of olive oil was made, according to its chemical composition.

Bayes classifier

- Classification according to posteriors

$$\Pr(Y = k \mid \mathbf{X} = \boldsymbol{x}) = \frac{f_k(\boldsymbol{x})\pi_k}{\sum_{\ell=1}^{K} f_\ell(\boldsymbol{x})\pi_\ell}, \quad k = 1, \ldots, K$$

  - $f_k(\boldsymbol{x})$: the probability density/mass function of $\mathbf{X}$ conditioning on Class k
  - $\pi_k = \Pr(Y = k)$: prior 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, from the perspective of Bayes classifier)

- Assuming $f_k(\boldsymbol{x})$ = density of $\mathrm{MVN}_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$
- LDA classifier (population version)

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

  - 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 $\boldsymbol{x}$

$$\max_{k=1,\cdots,K} f_k(x)\, \pi_k$$

$$= \max_{k=1,\cdots,K} (2\pi)^{-\frac{p}{2}} \{\det(\Sigma)\}^{-\frac{1}{2}} \exp\{-\tfrac{1}{2}(x-\mu_k)^T \Sigma^{-1} (x-\mu_k)\}\, \pi_k$$

$$= \max_{k=1,\cdots,K} \exp\{-\tfrac{1}{2}(x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}\mu_k)\}\pi_k$$

$$= \max_{k=1,\cdots,K} \exp(x^T\Sigma^{-1}\mu_k - \tfrac{1}{2}\mu_k^T\Sigma^{-1}\mu_k)\pi_k$$

$$= \max_{k=1,\cdots,K} x^T\Sigma^{-1}\mu_k - \tfrac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \ln \pi_k$$

- LDA classifier (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
    * $\hat{\pi}_k = n_k/n$
    * $\hat{\boldsymbol{\mu}}_k = n_k^{-1} \sum_{i=1}^n \boldsymbol{x}_i \cdot \mathbf{1}(y_i = k)$
    * $\widehat{\boldsymbol{\Sigma}} = (n-1)^{-1} \sum_{k=1}^K \sum_{i=1}^n (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top \cdot \mathbf{1}(y_i = k)$
  - Empirical LDA classifier
$$\hat{h}(\boldsymbol{x}) = \arg \max_{k=1,\dots,K} \hat{\delta}_k(\boldsymbol{x})$$
    * $\hat{\delta}_k(\boldsymbol{x}) = \boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k - \frac{1}{2} \hat{\boldsymbol{\mu}}_k^\top \widehat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k + \ln \hat{\pi}_k$

- Example 9.1 (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(Ytrain)
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)
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])
}
(resLda1 = apply(deltaksLda, 1, FUN = function(x){labels[which.max(x)]}))
# use MASS::lda
objLda = MASS::lda(Xtrain, Ytrain, method   = "moment")
(resLda2 = predict(objLda, Xtest)$class)
# comparison
mean(resLda1 == resLda2)
mean(Ytest != resLda1)
```

Quadratic discriminant analysis (QDA, from the perspective of Bayes classifier)

- Assuming $f_k(\boldsymbol{x})$ = density of $\mathrm{MVN}_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
- QDA classifier (population version)

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

  - 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 $\boldsymbol{x}$

$$\max_{k=1,\cdots,K} f_k(x)\pi_k$$
$$= \max_{k=1,\cdots,K} (2\pi)^{-\frac{p}{2}} \{\det(\Sigma_k)\}^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu_k)^\top\Sigma_k^{-1}(x-\mu_k)\right\}\pi_k$$
$$= \max_{k=1,\cdots,K} \{\det(\Sigma_k)\}^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(x^\top\Sigma_k^{-1}x - 2x^\top\Sigma_k^{-1}\mu_k + \mu_k^\top\Sigma_k^{-1}\mu_k\right)\right\}\pi_k$$
$$= \max_{k=1,\cdots,K} -x^\top\Sigma_k^{-1}x + 2x^\top\Sigma_k^{-1}\mu_k - \mu_k^\top\Sigma_k^{-1}\mu_k + 2\ln\pi_k - \ln\det(\Sigma_k)$$

- QDA classifier (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
    - $\hat{\pi}_k = n_k/n$
    - $\hat{\boldsymbol{\mu}}_k = n_k^{-1}\sum_{i=1}^n \boldsymbol{x}_i \cdot \mathbf{1}(y_i = k)$
    - $\widehat{\boldsymbol{\Sigma}}_k = (n_k-1)^{-1}\sum_{i=1}^n(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top \cdot \mathbf{1}(y_i = k)$
  - Empirical classifier

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

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

- Example 9.2 (`iris` data, con'd)

```
# follow formulas
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]]))
}
(resQda1 = apply(deltaksQda, 1, FUN = function(x){labels[which.max(x)]}))
# use MASS::qda
objQda = MASS::qda(Xtrain, Ytrain, method   = "moment")
(resQda2 = predict(objQda, Xtest)$class)
# comparison
mean(resQda1 == resQda2)
mean(Ytest != resQda1)
```

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 $X|Y=k \sim MVN(\mu_k, \Sigma)$,

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

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

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

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

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

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

```r
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))
```

```r
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)
```

A joint application of LDA/QDA & PCA

- Revisit the dataset of handwritten digits Part 7: `mnist` is a list with two components: `train` and `test`. Each of these is a list with two components: images and labels.
  - The `images` component is a matrix with each row for one image consisting of 28*28 = 784 entries (pixels). Their value are integers between 0 and 255 representing grey scale.
  - The `labels` components is a vector representing the digit shown in the image.
  - Uninvertible $\mathbf{S}_k$ because of the shared blank on canvas

```
options(digits = 4)
mnist = dslabs::read_mnist()
Xtrain = mnist$train$images
Ytrain = mnist$train$labels
Xtest = mnist$test$images
Ytest = mnist$test$labels

# The 3690th image in the training set
i0 = 3690
Ytrain[i0]
image(
  matrix(Xtrain[i0,], ncol = 28),
  col = gray.colors(12, rev = TRUE), axes = FALSE, main = "3690th image in the training set")
```

```r
# Build classifiers according to PC scores
decompXtrain = prcomp(Xtrain)
s = which(cumsum((decompXtrain$sdev)^2)/sum((decompXtrain$sdev)^2)>=.9)[1]
PCscoresXtrain = decompXtrain$x[,1:s]
objLda = MASS::lda(PCscoresXtrain, Ytrain, method = "moment")
objQda = MASS::qda(PCscoresXtrain, Ytrain, method   = "moment")

# Label prediction according to PC scores
xbarXtrain = colMeans(Xtrain)
PCscoresXtest = sweep(Xtest, 2, xbarXtrain) %*% decompXtrain$rotation[,1:s]
resLda = predict(objLda, PCscoresXtest)$class
resQda = predict(objQda, PCscoresXtest)$class
mean(resLda != Ytest)
mean(resQda != Ytest)
```

Alternative methods for classification in the view of regression

- (Multinomial) logistic regression
- k-nearest neighbors (k-NN)
- Tree-based
  - Decision tree/classification and regression tree (CART)
  - Random forest