Mixture of Gaussians with cross-validation

Ayush Agarwal

Date Submitted : Monday, 04/11/2019

Premise

We have (X, Y) from a bivariate mixture of Gaussians with some unknown number of classes C.

That is :-

$f(x, y|\mu_1, ..., \mu_C, \Sigma_1, ..., \Sigma_C, \pi_1, ..., \pi_C) = \sum_{c=1}^{C} \pi_c f(x, y|\mu_c, \Sigma_c)$

where each $\mu_c \in \mathbb{R}^2$, Σ_c is a 2×2 covariance matrix, and π_c is the mixture probability between 0 and 1.

Objective

Our goal is to estimate $\mu = (mu_1, ..., \mu_C)$, $\Sigma_1, ..., \Sigma_C$ and $\pi = (\pi_1, ..., \pi_C)$

Assuming we know that the number of classes/groups here can only be C = 2, 3, 4, 5, or 6. We will use a **5-fold Cross Validation** with negative log-likelihood as a loss function, and to verify the same, we will also use the Akaike Information Criterion (AIC).

Theoritical Background

EM Algorithm for Mixture of Gaussians

Consider a J-group normal mixture, where $x_1, ..., x_n \sim \sum_{j=1}^{J} p_j \phi(x_i|\mu_j, \sigma_j)$

where $\phi(.|\mu, \sigma)$ is the normal density.

This is the clustering/finite mixture problem in which EM is typically used for.

Define indicator variable for observation i : $(y_{i1}, y_{i2}, ..., y_{iJ})$ follows a multinomial distribution (with trail number=1) and cell probabilities $p = (p_1, p_2, ..., p_J)$

Clearly, $\sum_j y_{ij} = 1$. Given $y_{ij*} = 1$ and $y_{ij} = 0$ for $j, j*$, we assume

$x_i \sim N(\mu_{j*}, \sigma_{j*})$

One can check, marginally, $x_i \sim \sum_{j=1}^{J} p_j \phi(x_i|\mu_j, \sigma_j)$

Here, $\{x_i\}_i$ is the observed data; $\{x_i, y_{i1}, ..., y_{iJ}\}_i$ is the complete data.

Observed-data log likelihood

$l(\mu, \sigma, p|x) = \sum_{ij} log\{\sum_{j=1}^{J} p_j \phi(x_i|\mu_j, \sigma_j)\}$

Complete-data log likelihood

$l_C(\mu, \sigma, p|x, y) = \sum_{ij} y_{ij}\{\log p_j + \log \phi(x_i|\mu_j, \sigma_j)\}$

$= \sum_{ij} y_{ij}\{\log p_j - (x_i - \mu_j)^2/2\sigma_j^2 - \log \sigma_j\}$

E-Step

Evaluate for $i = 1, ..., n$ and $j = 1, ..., J$, $\omega_{ij}^{(k)} = E(y_{ij}|x_i, \mu^{(k)}, \sigma^{(k)}, p^{(k)}) = Pr(y_{ij} = 1|x_i, \mu^{(k)}, \sigma^{(k)}, p^{(k)})$
$= \frac{p_j^{(k)} f(x_i|\mu_j^{(k)}, \sigma_j^{(k)})}{\sum_j p_j^{(k)} f(x_i|\mu_j^{(k)}, \sigma_j^{(k)})}$

M-step

Maximize complete-data log likelihood with y_{ij} replaced by ω_{ij}

$p_j^{(k+1)} = n^{-1} \sum_i \omega_{ij}^{(k)}$

$\mu_j^{(k+1)} = \sum_i \omega_{ij}^{(k)} x_i / \sum_i \omega_{ij}^{(k)}$

$\sigma_j^{(k+1)} = \sqrt{\sum_i \omega_{ij}^{(k)} (x_i - \mu_j^{(k)})^2 / \sum_i \omega_{ij}^{(k)}}$

K-Fold Cross Validation

Let $K : \{1, 2, ..., n\} \mapsto \{1, 2, ..., k\}$ be a map

And $K(i)$: indicates the partition segment(fold) to with the i^{th} observation belongs.

Let $f^{-\kappa}(x)$ be the fitted function for the κ partition removed.

Then, prediction error $CV(\hat{f}) = n^{-1} \sum_{i=1}^{n} L(y_i, f^{-\kappa(i)}(x))$

Here $L(.)$ is the loss function.

Algorithm for this problem

- Choose a Loss function(here negative log-likelihood) for the CV, and a CV method. (Here 5-fold)
- For the 5-fold Cross Validation, Fit the GMM on the training set using an appropriate starting value.
- Calculate Loss for the test data using the estimates from above fit.
- Run for each number of classes, choose the model with *min* Loss value.
- Output clusters using the Expectation Matrix.

R-code

```
################# Missing Packages ##################
if(!require(ggplot2)){
  install.packages("ggplot2")
  library(ggplot2)
}
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```r
if(!require(mvtnorm)){
  install.packages("mvtnorm")
  library(mvtnorm)
}
```

```
## Loading required package: mvtnorm
```

```r
#########################################################
set.seed(42)

# Finds AIC values
aic <- function(data, class, theta)
{
  loss <- 0

  for(c in 1:class)
  {
    mu.temp  <- theta[(cl+1+2*(c-1)):(cl+2+2*(c-1))]
    sig.temp <- matrix(theta[(3*cl+1+4*(c-1)):(3*cl+4+4*(c-1))],byrow=TRUE,ncol=2,nrow=2)
    loss <- loss +  theta[c]*dmvnorm(data, mu.temp, sig.temp)
  }

    fin <- - 2*sum(log(loss)) + 2*(6*class - 1) # No. of params  = 6*C -  1
    return(fin)
}

loglike <- function(data, class, theta)
{
  loss <- 0

    for(c in 1:class)
  {
    mu.temp  <- theta[(cl+1+2*(c-1)):(cl+2+2*(c-1))]
    sig.temp <- matrix(theta[(3*cl+1+4*(c-1)):(3*cl+4+4*(c-1))],byrow=TRUE,ncol=2,nrow=2)
    loss <- loss +  theta[c]*dmvnorm(data, mu.temp, sig.temp)
  }

    fin <-  - sum(log(loss))
    return(fin)
}


GMMforcl <- function(dat1,cl,flag=0)
{
  n = nrow(dat1)
  dat1.kmeans <- kmeans(dat1,cl,nstart=10)
  dat1.kmeans.cluster <- dat1.kmeans$cluster
  dat1.df <- data.frame(x = dat1, cluster = dat1.kmeans.cluster)


  mu.init = as.vector(t(dat1.kmeans$centers))
  sig2.init <- rep(c(var(dat1[,1]), cov(dat1[,1],dat1[,2]), cov(dat1[,2], dat1[,1]), var(dat1[,2])),cl)
```

```
  dat1.kmeans$pi <- dat1.kmeans$size/sum(dat1.kmeans$size)
  pi.init <- dat1.kmeans$pi


  tol <- 1e-6
  itr <- 0
  diff <- 1000
  theta = c(pi.init,mu.init,sig2.init)
  current <- theta

  Exp <- matrix(0, nrow = n, ncol = cl)

  while(diff > tol)
  {
    itr <- itr + 1

    #E step
    for(c in 1:cl)
    {
      mu.temp  <- current[(cl+1+2*(c-1)):(cl+2+2*(c-1))]
      sig.temp <- matrix(current[(3*cl+1+4*(c-1)):(3*cl+4+4*(c-1))],byrow=TRUE,ncol=2,nrow=2)
      Exp[,c] <- current[c]*dmvnorm(dat1, mean=mu.temp,sigma=sig.temp)
    }

    Exp <- Exp/(rowSums(Exp))

    # M-step

    for(c in 1 :cl)
    {
      theta[c] = mean(Exp[,c])
      theta[(cl+1+2*(c-1)):(cl+2+2*(c-1))] = colSums(Exp[,c]*dat1)/sum(Exp[,c])
      theta[(3*cl+(4*(c-1)+1)):(3*cl+ 4*(c-1) +4)] = cov.wt(dat1, Exp[,c])$cov
    }

    diff <- max(abs(theta-current))
    current <- theta
  }


  if(flag==1)
  ExpGlobal <<- Exp

  return(theta)
}


# 5-fold Cross-validation
dat <- read.table("http://home.iitk.ac.in/~dootika/assets/course/MixG_data/170187.txt",
                  header = F, col.names = c("X", "Y"))
n = nrow(dat)

permutation <- sample(1:n, replace = FALSE)
```

```
K <- 5

test.index <- split(permutation, rep(1:K, length = n, each = n/K))

classes <- 2:6
CV.Loglike <- numeric(length = length(classes)+1)
CV.AIC <- numeric(length = length(classes)+1)

for(cl in 2:6)
{
  temp3 <- 0
  temp4 <- 0

  for(k in 1:K)
{
  dat.train <- dat[-test.index[[k]], ]
  dat.test <- dat[test.index[[k]], ]
  dat.fit <- GMMforcl(dat.train,cl)
  temp3 <- temp3 + loglike(data = dat.test, class = cl, theta = dat.fit)
}

dat.fit <- GMMforcl(dat,cl)
temp4 <- aic(data = dat, class = cl, theta=dat.fit)

CV.Loglike[cl] <- temp3/n
CV.AIC[cl] <- temp4/n
}
```

Computation

Calculating Starting Value

We use the *K-means* clustering algorithm. The output of the K-means algorithm gives us centres to be used as mu, proportion of data split to be used as pi, and the variation in data to be used as sigma matrix.

K-means

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups(clusters), where each data point belongs to only one group.

It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

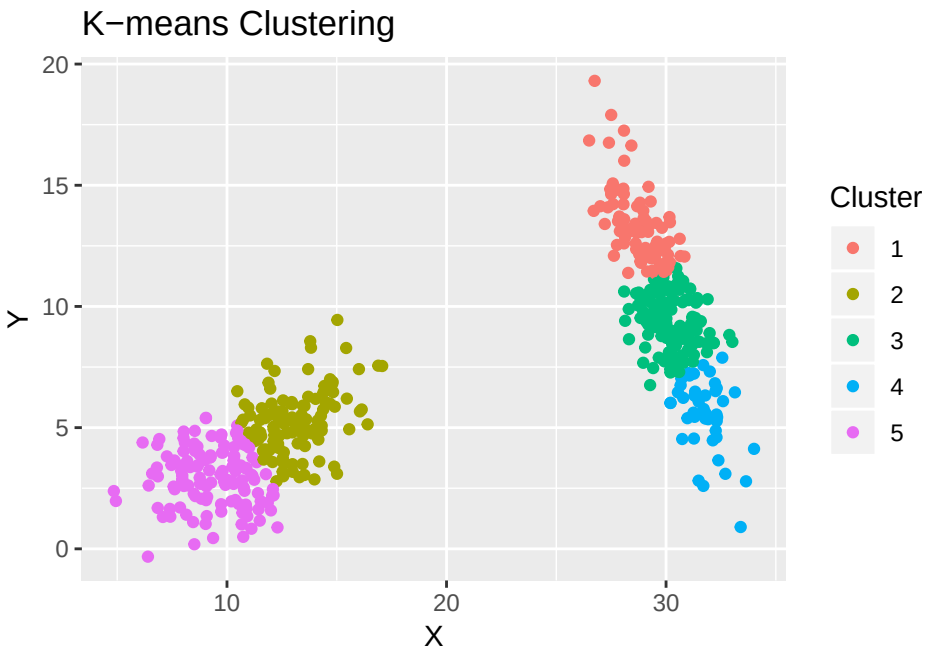
The way k-means algorithm works is as follows:

- Specify number of clusters K.
- Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
- Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

```
# K means clustering for random number of classes
dat1.kmeans <- kmeans(dat,5,nstart=10)
  dat1.kmeans.cluster <- dat1.kmeans$cluster
  dat1.df <- data.frame(x = dat, cluster = dat1.kmeans.cluster)

  print(ggplot(data = dat1.df,aes(y = x.Y, x = x.X, color = factor(cluster))) +
          geom_point() + xlab("X") + ylab("Y") + scale_color_discrete(name = "Cluster") +
          ggtitle("K-means Clustering"))
```



Choosing the final model

Negative Log-Likelihood vs AIC

```
CV.AIC
```

```
## [1] 0.000000 9.256733 9.265500 9.252755 9.241464 9.256585
```

```
CV.Loglike
```

```
## [1] 0.000000 4.627651 4.650794 4.651208 4.672597 4.696346
```

We can clearly see that the model selected via AIC yields 5 classes, while Log-likelihood outputs 2 classes.

Since the AIC values are calculated for the whole dataset, the error outputed is Training Error which is NOT a good estimate for test error in general due to the fact that we can make the training error arbitrarily small.

On the other hand, Log-likelihood values come from a 5-Fold cross validastion model, where we are calculating Test Error, hence, we choose the model with the lowest log-likelihood value.

```
CV.Loglike <- CV.Loglike[2:6]
CV.AIC <- CV.AIC[2:6]
Class.Final <- which.min(CV.Loglike) + 1 #choosing classes

####### Final fit ##########

ExpGlobal <- matrix(0,nrow=n,ncol=Class.Final)
dat.fit <- GMMforcl(dat,Class.Final,flag=1)

mu.est <- matrix(0,nrow=Class.Final,ncol=2)
sig.est <- list()
cl <- Class.Final
for(i in 1:Class.Final)
{
  sig.est[[i]] = matrix(dat.fit[(3*cl+1+4*(i-1)):(3*cl+4+4*(i-1))],byrow=TRUE,ncol=2,nrow=2)
  mu.est[i,] = dat.fit[(cl+1+2*(i-1)):(cl+2+2*(i-1))]
}
mix.est <- dat.fit[1:Class.Final]
```

Calculating Clusters

For each row in the Expectation matrix, the highest value represents the cluster that particular point belongs to.

```
cluster <- numeric(length=n)
for(i in 1:n)
{
  cluster[i] <- which.max(ExpGlobal[i,])
}
dat$cluster <- cluster
```

Plots and Estimates

Final Estimates

```
mix.est
```

```
## [1] 0.514 0.486
```

```
mu.est
```

```
##          [,1]     [,2]
## [1,] 30.03779 9.950537
## [2,] 11.13843 4.035726
```

```
sig.est
```

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

```
##           [,1]      [,2]
## [1,]  1.935249 -3.233177
## [2,] -3.233177  8.972374
##
## [[2]]
##          [,1]     [,2]
## [1,] 5.729344 2.345866
## [2,] 2.345866 2.882506
```

Final Clusters

```
print(ggplot(data = dat,aes(y = Y, x = X, color = factor(cluster))) +
        geom_point() + xlab("X") + ylab("Y") + scale_color_discrete(name = "Cluster") +
        ggtitle("Final Clusters"))
```