mml87_nb

Kelvin

27 June 2017

Log likelihood

For a Naive Bayes model with binary variable, its parameters are $\{P(Y_i = T), P(X_{ij}|Y_i = T), P(X_{ij}|Y_i = F)\}$. To simply the notations, we use $\{p_{i0}, p_{ij1}, p_{ij2}\}$ to denote the above probabilities respectively. The likelihood of Naive Bayes given a data set is then

$$l = \prod_i^n P(Y_i = T|\vec{X_i})^{Y_i}(1 - P(Y_i = T|\vec{X_i}))^{1-Y_i}$$

where the posterior probability of Y_i given a vector $\vec{X_i} =< X_{i1}, \ldots, X_{im} >$ is

$$\begin{aligned} P(Y_i = T|\vec{X_i}) &= \frac{P(Y_i = T)\prod_{j=1}^m P(X_{ij}|Y_i = T)}{P(\vec{X_i})} \\ &= \frac{P(Y_i = T)\prod_{j=1}^m P(X_{ij}|Y_i = T)}{P(Y_i = T)\prod_{j=1}^m P(X_{ij}|Y_i = T) + (1 - P(Y_i = T))\prod_{j=1}^m P(X_{ij}|Y_i = F)} \\ &= \frac{p_{i0}\prod_{j=1}^m p_{ij1}}{p_{i0}\prod_{j=1}^m p_{ij1} + (1 - p_{i0})\prod_{j=1}^m p_{ij2}} \end{aligned}$$

The negative loglikelihood

$$\begin{aligned} L &= -\sum_{i=1}^n \left[Y_i \ln p(Y_i|\vec{X_i}) + (1 - Y_i)\ln(1 - p(Y_i|\vec{X_i}))\right] \\ &= -\sum_{i=1}^n \left[Y_i \ln p_{i0} + (1 - Y_i)\ln(1 - p_{i0}) + Y_i\sum_{j=1}^m \ln p_{ij1} + (1 - Y_i)\sum_{j=1}^m \ln p_{ij2} - \ln\left(p_{i0}\prod_{j=1}^m p_{ij1} + (1 - p_{i0})\prod_{j=1}^m p_{ij2}\right)\right] \end{aligned}$$

Notice that the log likelihood function that stated above is different from what mostly shown by others, which do not consider the case where $Y_i = T$ and $Y_i = F$. Hence, it is not sure whether we can use their MLE of NB parameters as shown below.

$$\begin{aligned} P(Y = y) &= \frac{count(y)}{n} \\ P(X_j = x|Y = y) &= \frac{count(x) \cup count(y)}{count(y)} \end{aligned}$$

The following are implementations of MLE of NB parameters.

MLE implementations

A function to calculate maximum likelihood estimation of nb parameters.

```
# y is the output node
# x is a set of input nodes
# estimated parameters are stored in a list
# smoothing is additive smoothing constant to avoid 0 probabilities but it can take any other values
```

```r
# y parameters is always store at the end
mle_est_nb = function(y, x, data, smoothing = 1) {
  xIndices = which(colnames(data) %in% x)
  yIndex = which(colnames(data) == y)
  lst = list()
  for (i in 1:length(x)) {
    lst[[i]] = t((table(data[, c(yIndex, xIndices[i])]) + smoothing) /
      (rowSums(table(data[, c(yIndex, xIndices[i])])) + smoothing * nlevels(data[, xIndices[i]])))
  } # end for i
  # always store y parameters at the last
  lst[[length(x) + 1]] = (table(data[, yIndex]) + smoothing) / (nrow(data) + smoothing * nlevels(data[,
  names(lst) = c(x, y)
  return(lst)
}
```

Fisher information matrix

Define $\theta_0 = P(Y = y)$ and $\theta_j = P(X_j = x|Y = y), \forall j \in [1, m]$. The second derivative of the above negative log likelihood w.r.t each parameter is

$$\frac{\partial^2 L}{\partial \theta_k^2} = \sum_{i=1}^{n} \left[\frac{1}{\theta_k^2} - \left(\frac{\prod_{j=0}^{\neq k} \theta_j}{\sum_y \prod_{j=0}^{m} \theta_j} \right)^2 \right]$$

$$\frac{\partial^2 L}{\partial \theta_k \partial \theta_l} = \sum_{i=1}^{n} \left[\frac{(\sum_y \prod_{j=0}^{m} \theta_j)(\sum_{j=0}^{\neq k,l} \theta_j) - (\prod_{j=0}^{\neq k} \theta_j)(\prod_{j=0}^{\neq l} \theta_j)}{\left(\sum_y \prod_{j=0}^{m} \theta_j\right)^2} \right]$$

to be a vector of parameters for Naive Bayes, where $\theta_0 = P(Y = y_1)$ and $\theta j1 = P(X_j = x_j|Y = y_1)$ and $\theta j2 = P(X_j = x_j|Y = y_2)$ respectively. Then we have the following

$$P(X) = \sum_{Y=y} P(Y = y) \prod_{j=1}^{m} P(X_j|Y = y)$$

$$\frac{\partial^2 L}{\partial \theta_0^2} = \sum_{i=1}^{n} \left[\frac{1}{\theta_0^2} - \left(\frac{\prod_{j=1}^{m} \theta_{j1} - \prod_{j=1}^{m} \theta_{j2}}{P(X)} \right)^2 \right]$$

$$\frac{\partial^2 L}{\partial \theta_{k1}^2} = \sum_{i=1}^{n} \left[\frac{1}{\theta_{k1}^2} - \left(\frac{\theta_0 \prod_{j=1}^{m,j\neq k} \theta_{j1}}{P(X)} \right)^2 \right]$$

$$\frac{\partial^2 L}{\partial \theta_{k2}^2} = \sum_{i=1}^{n} \left[- \left(\frac{(1 - \theta_0) \prod_{j=1}^{m,j\neq k} \theta_{j2}}{P(X)} \right)^2 \right]$$

for $k \in [1, m]$.

FIM implementations

```r
# p(x_ij|y=k)
# xIndices is a vector of x indices in data
# xIndex is the index of the particular x that we want
```

```
p_ijk = function(dataPoint, pars, xIndices, xIndex, yValue) {
  xValue = dataPoint[[xIndex]]
  xParsIndex = which(xIndices == xIndex)
  return(pars[[xParsIndex]][yValue, xValue])
}

# \prod_j p(x_ij|y=k)
# i is the dataPoint index
# j is the X index in data
# k is the value of Y, i.e. k \in [1, arity(y)]
prod_pijk = function(dataPoint, pars, xIndices, yValue) {
  mm = 1
  for (i in 1:length(xIndices)) {
    xValue = dataPoint[[xIndices[i]]]
    mm = mm * pars[[i]][xValue, yValue]
  }
  return(mm)
}
```

```
dag = empty.graph(c("X1", "X2", "Y", "X3"))
dag = set.arc(dag, "Y", "X1")
dag = set.arc(dag, "Y", "X2")
dag = set.arc(dag,  "Y", "X3")
set.seed(10086)
cpts = randCPTs(dag, 2, 1)
data = rbn(cpts, 100)
nVars = ncol(data)

y = "Y"
x = c("X1", "X3")
arities = sapply(data, nlevels)
yIndex = which(colnames(data) == y)
xIndices = which(colnames(data) %in% x)

# mle of parameters with smoothing
pars = mle_est_nb(y, x, data, 1)

# p(y=T)
py1 = pars[[length(pars)]][[1]]
# p(y=F)
py2 = pars[[length(pars)]][[2]]

# a vector of p_yi
#p_y = sapply(data[, yIndex], p_yi, pars = pars, yIndex = yIndex)

# a vector of \prod_j p(x_ij|y=1)
prodPij1 = apply(data, 1, prod_pijk, pars = pars, xIndices = xIndices, yValue = 1)
# a vector of \prod_j p(x_ij|y=2)
prodPij2 = apply(data, 1, prod_pijk, pars = pars, xIndices = xIndices, yValue = 2)

# a vector of p_xi
px = py1 * prodPij1 + py2 * prodPij2

# a matrix of p(x_j|y=T) and p(y_j|y=F)
```

```
probsMatrix = c()
for (j in xIndices) {
  probsMatrix = cbind(probsMatrix, apply(data, 1, p_ijk, pars = pars, xIndices = xIndices, xIndex = j, y
}
```

Off diagonal of FIM

```
# empty FIM
fimDim = (arities[yIndex] - 1) + length(x) * arities[yIndex]
fim = matrix(0, nrow = fimDim, ncol = fimDim)

mm = prodPij1 * prodPij2/(px^2) # a common contant
fim[1, -1] = colSums(mm/probsMatrix) # fill in 1st row of FIM
fim[1, seq(3, ncol(fim), 2)] = -1 * fim[1, seq(3, ncol(fim), 2)]
for (rowIndex in 2:(nrow(fim) - 1)) {
  if (ncol(fim) - rowIndex == 1) {
    fim[rowIndex, -(1:rowIndex)] = sum(-mm * py1 * py2 / (probsMatrix[, rowIndex - 1] * probsMatrix[, -
  } else {
    fim[rowIndex, -(1:rowIndex)] = colSums(-mm * py1 * py2 / (probsMatrix[, rowIndex - 1] * probsMatrix
  } # end else
}

fim = fim + t(fim)
fim
```

```
##           [,1]      [,2]       [,3]      [,4]       [,5]
## [1,]    0.0000 259.68210 -143.10823 241.17853 -217.41871
## [2,]  259.6821   0.00000  -32.40441 -54.91157  -49.57339
## [3,] -143.1082 -32.40441    0.00000 -30.31565  -27.31008
## [4,]  241.1785 -54.91157  -30.31565   0.00000  -36.89407
## [5,] -217.4187 -49.57339  -27.31008 -36.89407    0.00000
```

Diagonal of FIM

```
# assume all variables are binary, hence the diag[1] is always the 2nd derivative w.r.t. p_i0
diag(fim)[1] = sum(prodPij1/(py1 * px) + prodPij2/(py2 * px) - ((prodPij1 - prodPij2)/px)^2)

# two vectors to store 2nd derivative w.r.t. p_ik1 and p_ik2 respectively, where the 1 and 2 correspond
z = c()
for (i in 1:length(xIndices)) {
  v = sum(py1 * prodPij1/(probsMatrix[,i * 2 - 1]^2 * px) - (py1 * prodPij1/(px * probsMatrix[,i * 2 -
  w = sum(py2 * prodPij2/(probsMatrix[,i * 2]^2 * px) - (py2 * prodPij2/(px * probsMatrix[,i * 2]))^2)
  z = c(z, v, w)
}

diag(fim)[-1] = z
fim
```

```
##           [,1]      [,2]       [,3]      [,4]       [,5]
## [1,] 1139.1745 259.68210 -143.10823 241.17853 -217.41871
## [2,]  259.6821  60.01580  -32.40441 -54.91157  -49.57339
## [3,] -143.1082 -32.40441   18.03590 -30.31565  -27.31008
```

```
## [4,]  241.1785 -54.91157  -30.31565 104.43649  -36.89407
## [5,] -217.4187 -49.57339  -27.31008 -36.89407   43.05963
```

```
det(fim)
```

```
## [1] 40867729578
```

```
log(det(fim))
```

```
## [1] 24.43361
```