Association Rules

John Williams

Association Rules measures how frequently items are associated with each other across a set of transactions. **Support** and **Confidence** are two measures reflecting the usefulness and certainly of an association. Support is the percentage of transactions including the items of interest. Confidence is the percentage of transaction incluing one item that also include other item(s). Association rules are interesting if they satisfy minumum thresholds of support and confidence. Below are two two implementations of association rules. The first utilizes an *imperative* programming style, while the second uses a functional programming style. The imperative function has several levels of loops and numerous conditional statements, while the functional implementation has no explicit loops and fewer conditional statements. The imperative implementation is space efficient at the expense of complexity.

Imperative Programming Implementation of Association Rules

```
items <- factor(sort(c("Beer", "Nuts", "Diaper", "Coffee", "Eggs", "Milk")))
transactions <- list(
  list(Tid = 10, itemset = factor(c("Beer", "Nuts",   "Diaper"),                levels(items))),
  list(Tid = 20, itemset = factor(c("Beer", "Coffee", "Diaper"),                levels(items))),
  list(Tid = 30, itemset = factor(c("Beer", "Diaper", "Eggs"),                  levels(items))),
  list(Tid = 40, itemset = factor(c("Nuts", "Eggs",   "Milk"),                  levels(items))),
  list(Tid = 50, itemset = factor(c("Nuts", "Coffee", "Diaper", "Eggs", "Milk"), levels(items)))   )
association_rules1 <- function(trans.list, minsup=0.5, minconf=0.5) {
  for (i in 1:length(items)) {
    supsum  <- 0
    for (k in 1:length(trans.list)) if (items[i] %in% trans.list[[k]]$itemset) supsum <- supsum + 1
    for (j in 1:length(items)) {
      if (i == j) next
      confsum <- 0
      for (k in 1:length(trans.list))
        if ( items[i] %in% transactions[[k]]$itemset && items[j] %in% transactions[[k]]$itemset) confsu
      if (supsum > 0) {
        sup  <- supsum/length(trans.list)
        conf <- confsum/supsum
        if (sup >= minsup)
          if (conf >= minconf)
            cat(levels(items)[i],"->",levels(items)[j],"(",round(sup,2)*100,"%,",round(conf,2)*100,"%)\
association_rules1(transactions, minsup = 0.5, minconf = 0.5)
```

```
## Beer -> Diaper ( 60 %, 100 %)
## Diaper -> Beer ( 80 %, 75 %)
## Diaper -> Coffee ( 80 %, 50 %)
## Diaper -> Eggs ( 80 %, 50 %)
## Diaper -> Nuts ( 80 %, 50 %)
## Eggs -> Diaper ( 60 %, 67 %)
## Eggs -> Milk ( 60 %, 67 %)
## Eggs -> Nuts ( 60 %, 67 %)
## Nuts -> Diaper ( 60 %, 67 %)
## Nuts -> Eggs ( 60 %, 67 %)
## Nuts -> Milk ( 60 %, 67 %)
```

Functional Programming Implementation of Association Rules

```
items <- sort(c("Beer", "Nuts", "Diaper", "Coffee", "Eggs", "Milk"))
transactions <- list(
  list(Tid = 10, itemset = c("Beer", "Nuts",   "Diaper")),
  list(Tid = 20, itemset = c("Beer", "Coffee", "Diaper")),
  list(Tid = 30, itemset = c("Beer", "Diaper", "Eggs")),
  list(Tid = 40, itemset = c("Nuts", "Eggs",   "Milk")),
  list(Tid = 50, itemset = c("Nuts", "Coffee", "Diaper", "Eggs", "Milk"))  )
rules <-  data.frame(
  itemA = rep(as.character(""),            (length(items)* (length(items) - 1))),
  itemB = rep(as.character(""),            (length(items)* (length(items) - 1))),
  sup   = rep(as.integer(0), (length(items)* (length(items) - 1 ))),
  conf  = rep(as.integer(0), (length(items)* (length(items) - 1 ))),
  stringsAsFactors = F )
rule_index <- 0
name_rule <- function(item_a, item_b) {
  if (item_a != item_b) {
    rules[rule_index, 1] <- item_a
    rules[rule_index, 2] <- item_b
    assign("rules", rules, envir = .GlobalEnv)
    assign("rule_index", rule_index + 1, envir = .GlobalEnv)  } }
set_rule_names <- function(items) {
  assign("rule_index", 1, envir = .GlobalEnv)
  xxx <- sapply (items, function(y) sapply(items, function(x)  name_rule(y,x)))  }
print_rules <- function(items, trans, minsup, minconf) {
  ntrans <- length(trans)
  xxx <- sapply(1:nrow(rules), function(i) {
    if ((rules[i,3]/ntrans >= minsup) && (rules[i,4]/rules[i,3] >= minconf))
      cat(rules[i,1],"->",rules[i,2],
          "(",round(rules[i,3]/ntrans,2)*100,"%,",
          round(rules[i,4]/rules[i,3],2)*100,"%)\n")    } )  }
get_support <- function(items, trans.list) {
  xxx <- sapply(items, function(item) sapply(trans.list, function(trans) {
    if (item %in% unlist(trans)) {
      row_vals <- which(rules$itemA == item)
      xxx <- sapply(seq_along(row_vals), function(i) {
        rules[row_vals[i],3] <- rules[row_vals[i],3] + 1
        assign("rules", rules, envir = .GlobalEnv)  } ) }})) }
get_confidence <- function(trans)  {
  xxx <- sapply(1:nrow(rules), function(i) {
    xxx <- sapply(1:length(trans), function(j) {
      if ((rules[i,1] %in% unlist(trans[j])) && (rules[i,2] %in% unlist(trans[j])) ) {
        rules[i,4] <-  rules[i,4] + 1
        assign("rules", rules, envir = .GlobalEnv)  } })}) }

association_rules2 <- function(transactions, items, minsup=0.5, minconf=0.5) {
  set_rule_names(items)
  get_support(items, transactions)
  get_confidence(transactions)
  print_rules(items, transactions, minsup, minconf) }

association_rules2(transactions, items, minsup = 0.5, minconf = 0.5)
```

```
## Beer -> Diaper ( 60 %, 100 %)
## Diaper -> Beer ( 80 %, 75 %)
## Diaper -> Coffee ( 80 %, 50 %)
## Diaper -> Eggs ( 80 %, 50 %)
## Diaper -> Nuts ( 80 %, 50 %)
## Eggs -> Diaper ( 60 %, 67 %)
## Eggs -> Milk ( 60 %, 67 %)
## Eggs -> Nuts ( 60 %, 67 %)
## Nuts -> Diaper ( 60 %, 67 %)
## Nuts -> Eggs ( 60 %, 67 %)
## Nuts -> Milk ( 60 %, 67 %)
```