Simulation Results

Ali Turfah

4/29/2020

Part 1: Model Set-Up

This simulation will consider three jointly distributed random variables (call them X, Y, Z). The assumption is that $X \neq Y$, $Y \neq Z$, and $X \neq Z$ ever, and that $P(X = a, Y = b, Z = c) = P(X = b, Y = a, Z = c) = ...$ for all such combinations of a, b, c.

```r
set.seed(10)
n.classes = 15

# Generate Data with 3-D Joint Structure
true.joint = array(numeric(1), dim=rep(n.classes, 3))
for (i in 1:n.classes) {
  for (j in i:n.classes) {
    if (i == j) {
      next
    }
    for (k in j:n.classes) {
      if (k == i || k == j) {
        next
      }
      true.joint[i, j, k] = runif(1)
      true.joint[i, k, j] = true.joint[i, j, k]
      true.joint[j, i, k] = true.joint[i, j, k]
      true.joint[j, k, i] = true.joint[i, j, k]
      true.joint[k, j, i] = true.joint[i, j, k]
      true.joint[k, i, j] = true.joint[i, j, k]
    }
  }
}
true.joint = true.joint / sum(true.joint)
```

From this true marginal distribution we can calculate a variety of roll-ups/slices; in this case I will consider a 2-D joint distribution (the distribution of X, Y with Z integrated out), the marginal distributions for the random variables, and both the 2-D and 3-D conditional distributions ($X|Y$ and all of $X, Y|Z$, $X, Z|Y$, and $Y, Z|X$ respectively).

```r
# 2-D Joint Distribution
true.joint.2D = apply(true.joint, 1, colSums)

# Marginal Distribution
true.marginal = apply(true.joint, 1, sum)

# 3-D Conditional Distribution
# Conditional Matrix: mat[i, j, k] = p(i, j|k) = p(i, j, k) / p(k)
```

```r
true.conditional.1 = array(numeric(1), dim=rep(n.classes, 3))
true.conditional.2 = array(numeric(1), dim=rep(n.classes, 3))
true.conditional.3 = array(numeric(1), dim=rep(n.classes, 3))
for (i in 1:n.classes) {
  for (j in i:n.classes) {
    if (i == j) {
      next
    }
    for (k in j:n.classes) {
      if (k == i || k == j) {
        next
      }
      true.conditional.3[i, j, k] = true.joint[i, j, k] / true.marginal[k]
      true.conditional.3[i, k, j] = true.joint[i, k, j] / true.marginal[j]
      true.conditional.3[j, i, k] = true.joint[j, i, k] / true.marginal[k]
      true.conditional.3[j, k, i] = true.joint[j, k, i] / true.marginal[i]
      true.conditional.3[k, j, i] = true.joint[k, j, i] / true.marginal[i]
      true.conditional.3[k, i, j] = true.joint[k, i, j] / true.marginal[j]

      true.conditional.2[i, j, k] = true.joint[i, j, k] / true.marginal[j]
      true.conditional.2[i, k, j] = true.joint[i, k, j] / true.marginal[k]
      true.conditional.2[j, i, k] = true.joint[j, i, k] / true.marginal[i]
      true.conditional.2[j, k, i] = true.joint[j, k, i] / true.marginal[k]
      true.conditional.2[k, j, i] = true.joint[k, j, i] / true.marginal[j]
      true.conditional.2[k, i, j] = true.joint[k, i, j] / true.marginal[i]

      true.conditional.1[i, j, k] = true.joint[i, j, k] / true.marginal[i]
      true.conditional.1[i, k, j] = true.joint[i, k, j] / true.marginal[i]
      true.conditional.1[j, i, k] = true.joint[j, i, k] / true.marginal[j]
      true.conditional.1[j, k, i] = true.joint[j, k, i] / true.marginal[j]
      true.conditional.1[k, j, i] = true.joint[k, j, i] / true.marginal[k]
      true.conditional.1[k, i, j] = true.joint[k, i, j] / true.marginal[k]
    }
  }
}
rm(list=c('i', 'j', 'k'))


# 2-D Conditional Distribution
# Conditional Matrix: mat[i, j] = p(i|j) = p(i, j) / p(j)
true.conditional.2D = apply(true.conditional.1, 1, colSums)
```

Some sanity checks are included below, to ensure that the matrices/calculations are valid

```r
# Integrating the 3D joints over the other dimensions give the same results
stopifnot(all(true.joint.2D == apply(true.joint, 2, colSums)))
stopifnot(all(true.joint.2D == apply(true.joint, 3, colSums)))

# Each slice of the 3-D conditionals are valid distributions
stopifnot(sum(apply(true.conditional.1, 1, sum)) == n.classes)
stopifnot(sum(apply(true.conditional.2, 2, sum)) == n.classes)
stopifnot(sum(apply(true.conditional.3, 3, sum)) == n.classes)

# Integrating any of the conditionals over the corresponding axis gives the same results
stopifnot(all(true.conditional.2D == apply(true.conditional.2, 2, colSums)))
stopifnot(all(true.conditional.2D == apply(true.conditional.3, 3, colSums)))
```

Part II: Sampling Scheme

The purpose here is to see, in the event that the full joint distribution is not available, how closely we can estimate it from the derived distributions. In specific we will focus our attention on the 2-D conditionals and the marginal distribution. The joint density will be estimated using the quantity

$$P(X = a, Y = b, Z = c) = P(X = a, Y = b|Z = c)P(Z = c) \approx P(X = a|Z = c)P(Y = b|Z = c)P(Z = c)$$

This estimation makes the assumption that X, Y are independent given Z, which is not true. However, note that as `n.classes` increases the two get closer together in magnitude.

```
paste(true.conditional.3[1, 2, 3] != true.conditional.2D[1, 3] * true.conditional.2D[2, 3],
      true.conditional.3[1, 2, 3],
      true.conditional.2D[1, 3] * true.conditional.2D[2, 3])
```

```
## [1] "TRUE 0.00538141331505904 0.004271776946928"
```

For this estimation, I am looking at the combinations not the permutations. This is kind of subtle but I can 'create' the permutations from a given combination since there are only 6 (3!) total permutations for each combination. Due to the symmetry of the conditional probabilities ($2, 1|3$ and $1, 2|3$ both correspond to the first line in the `joint_123` summaton), I can consider 3 instead of 6 and just double the resulting probability. Also note that I'm not truly "sampling" from the distribution, but instead recording probabilities of combinations occurring. This is because as `n.classes` grows the individual probabilities become incredibly small which leaves out a large chunk of the data when actually generating the samples if the total sample size is not sufficiently large.

```
possibilities = combn(1:n.classes, 3)
sampled.results = matrix(numeric(1), nrow=ncol(possibilities), ncol=n.classes)
total_pct = 0

for (i in 1:ncol(possibilities)) {
  cmb_1 = possibilities[1, i]
  cmb_2 = possibilities[2, i]
  cmb_3 = possibilities[3, i]

  joint_123 = 0 +
    true.conditional.2D[cmb_1, cmb_3] * true.conditional.2D[cmb_2, cmb_3] * true.marginal[cmb_3] +
    true.conditional.2D[cmb_1, cmb_2] * true.conditional.2D[cmb_3, cmb_2] * true.marginal[cmb_2] +
    true.conditional.2D[cmb_3, cmb_1] * true.conditional.2D[cmb_2, cmb_1] * true.marginal[cmb_1]
  joint_123 = joint_123 * 2

  # Instead of actually generating a sample, why not work with the probabilities directly
  # Use this value as a weight, using a one-hot encoding
  sampled.results[i, c(cmb_1, cmb_2, cmb_3)] = joint_123

  total_pct = total_pct + joint_123
}

stopifnot(total_pct < 1)
print(paste("Total probability captured in sample:", round(total_pct, 4)))
```

```
## [1] "Total probability captured in sample: 0.9269"
```

This total probability captured increases as `n.classes`$\rightarrow \infty$. From the `sample.results` matrix which gives some the weight for each combination, it is possible to get the empirical/estimated marginal, conditional, and joint 2-D distributions

```r
# Empirical Marginal
empirical.marginal = colSums(sampled.results) / sum(sampled.results)

# Empirical 2-D Joint
empirical.joint.2D = matrix(numeric(1), nrow=n.classes, ncol=n.classes)
for (i in 1:n.classes) {
  for (j in 1:n.classes) {
    if (i == j) {
      next
    }
    empirical.joint.2D[i, j] = sum(
        sampled.results[which(sampled.results[, i] > 0 &sampled.results[, j] > 0), ]
      )
  }
}
empirical.joint.2D = empirical.joint.2D / sum(empirical.joint.2D)

# Empirical 2-D Conditional
empirical.conditional.2D = matrix(numeric(1), nrow=n.classes, ncol=n.classes)
for (i in 1:n.classes) {
  for (j in 1:n.classes) {
    if (i == j) {
      next
    }
    empirical.conditional.2D[i, j] = empirical.joint.2D[i, j] / empirical.marginal[j]
  }
}
```

Part III: Evaluation

```r
set.seed(10)
KL.DIVERGENCE <- function(p, q, fix.val=1e-30) {
  # fix.val is used to correct for zeroes, which KL-Divergence does not like
  p[which(p == 0)] = fix.val
  q[which(q == 0)] = fix.val
  sum(p * log(p / q))
}

# Random Vectors
rand.vec <- function() {
  rand.v = runif(n.classes)
  rand.v / sum(rand.v)
}

# Random Matrices with same overall structure
# - 0's along diagonals
# - Symmetric
rand.mat <- function() {
  rand.m = matrix(0, nrow=n.classes, ncol=n.classes)
  for (i in 1:n.classes) {
    for (j in i:n.classes) {
      if (i == j) {
        next
      }
      temp = runif(1)
      rand.m[i, j] = temp
      rand.m[j, i] = temp
    }
  }
  rand.m / sum(rand.m)
}

# Evaluation
rbind(
  random=c(
    marginal=KL.DIVERGENCE(true.marginal, rand.vec()),
    joint2D=KL.DIVERGENCE(true.joint.2D, rand.mat()),
    conditional2D=KL.DIVERGENCE(true.joint.2D, rand.mat())
  ),
  empirical=c(
    marginal=KL.DIVERGENCE(true.marginal, empirical.marginal),
    joint=KL.DIVERGENCE(true.joint.2D, empirical.joint.2D),
    conditional=KL.DIVERGENCE(true.conditional.2D, empirical.conditional.2D)
  )
)
```

```
##               marginal    joint2D conditional2D
## random    1.486356e-01 0.31070364    0.24676554
## empirical 5.932717e-06 0.00179376    0.02679496
```

To compare the estimated distributions with the true ones, I will use the KL Divergence. The scale is a bit difficult to interpret but at the very least it stands that the values for the random vectors and matrices are larger than those for the empirically estimated ones by around an order of magnitude.