Dependent Mixture Models and Joint Conditional Density Models

```
## [1] "C:/Users/Admin/anaconda3/envs/hmm-paper/Lib/R/library"
## [2] "C:/Users/Admin/Documents/R/win-library/3.6"
```

Overview and Notions

We use these notions:

- $\boldsymbol{a}$ – a vector,
- $\boldsymbol{A}$ – a matrix,
- A – a tensor

Preparation steps

Data

We need to prepare 0th-, 1st-, and 2nd-order datasets, conditioned on the kind of preceding commit. There are extra functions to get those datasets.

Packages

We will need a version of `mmb` for this, and we take an early but stable version by referring to a specific commit ID.

Model Preparation

In this section, we will prepare initial state probability vectors/matrices and transition matrices/tensors. These are needed for the dependent mixture models. In the joint conditional density models, we only use initial state probabilities conditionally (we test with and without).

Initial State Probabilities

When we define (1st-order) dependent mixture models, then there is a finite set of states $\mathcal{S} = \{a, c, p\}$ (corresponding to the maintenance activities of commits and summing to one), a vector $\boldsymbol{\pi} \in \mathcal{R}^{\|S\|}$ of initial state probabilities.

The intial probabilities are, regardless of the order of the model, always build from the $t - 0$ (zeroth-order) commits, so we can always use the dataset `commits_t0`. In other words, each model, regardless of its order, uses the same $\phi_1(j)$ with the same $\boldsymbol{\pi}$, so that we only need to generate this once.

```
initProbs <- c(
  a = nrow(commits_t0[commits_t0$label == "a", ]),
  c = nrow(commits_t0[commits_t0$label == "c", ]),
  p = nrow(commits_t0[commits_t0$label == "p", ])
) / nrow(commits_t0)

print(initProbs)
```

```
##         a         c         p
## 0.2160279 0.2369338 0.5470383
```

Transition Matrices and Tensors

With increasing order, models require tensors with higher order for their transitions. For a 1st-order model, this order is 2 (a quadratic matrix actually). If more than one previous state is considered, then the matrix becomes a tensor of order 1 + number of previous states considered ($\mathcal{R}^3$ for a 2nd-order model).

Every model's $\phi_2(j)$ depends on the likelihood of the current state and its previous state, hence we need transition probabilities. Hence, ϕ's order corresponds to the order of this tensor. So, for capturing transitions between two consecutive states, that order is two, and results in a matrix. For transitions between three states (a 2nd-order model), this becomes a 3-dimensional tensor. Any of these matrices or tensors always sum to the amount of possible start states, as the sum of probabilities of possible transitions from any state j is 1. On each axis of these matrices or tensors we find each possible state (here: $\mathcal{S} = \{a, c, p\}$), so that the size of this matrix/tensor is $\mathcal{R}^{\|S\|^{T+1}}$ (where T is the order of the model).

Similar to the initial state probabilities, we define a matrix for all $\phi_2(j)$, and a tensor for all $\phi_3(j)$ (as we are only evaluating 1st- and 2nd-order models). $\phi_3(j)$ is then used as $\phi_t(j)$ in 2nd-order models (and similarly, $\phi_2(j)$ is used as $\phi_t(j)$ in 1st-order models).

As a convention, the dimensions in these matrices and tensors are ordered from most recent to oldest, i.e., $\boldsymbol{A}_{t_0,t_{-1},\ldots,t_{-T}}$. This means that we can query similar to ".. what is the probability of t_0, given that we were in t_{-1} before and t_{-2} befor that?" using that notion.

```
transprobs_1stOrder <- matrix(data = 0, nrow = 3, ncol = 3)
colnames(transprobs_1stOrder) <- levels(commits_t1$label)
rownames(transprobs_1stOrder) <- levels(commits_t1$label)

for (t_1 in levels(commits_t1$label)) { # column-wise
  for (t_0 in levels(commits_t1$label)) {
    # Sum how often we went from t_1 to t_0
    transprobs_1stOrder[t_0, t_1] <- transprobs_1stOrder[t_0, t_1] +
      sum(commits_t1$label_t1 == t_1 & commits_t1$label == t_0)
  }

  # Normalize all options for ending up in t_0 coming from t_1:
  transprobs_1stOrder[, t_1] <- transprobs_1stOrder[, t_1] /
    sum(transprobs_1stOrder[, t_1])
}

print(transprobs_1stOrder)
```

```
##           a         c         p
## a 0.3965517 0.1250000 0.1311475
## c 0.2758621 0.4107143 0.1475410
## p 0.3275862 0.4642857 0.7213115
```

As an example, to go over `p` to `a` (or to end up in `a` having gone over `p`), we select the transition probability as 0.1311475. For any higher-dimension tensors, we prepend dimensions, so that we can follow this scheme (going over .. to ..).

Transition Tensor for 2nd-order Models

We do this in an extra section as we will work with actual tensors and the initialization is a bit different. We stick to the same indexing convention as for 2D-matrices.

```
install.packagesCond("tensorr")
library("tensorr")
```

```
## Warning: package 'tensorr' was built under R version 3.6.3
```

```
##
## Attaching package: 'tensorr'
```

```
## The following object is masked from 'package:base':
##
##     norm
```

```
# Create a dense 3x3x3 tensor
transprobs_2ndOrder <- dtensor(array(data = 0, dim = c(3,3,3)))
dimnames(transprobs_2ndOrder) <-
  list(levels(commits_t0$label), levels(commits_t0$label), levels(commits_t0$label))

# Now let's fill the tensor using a numeric mapping a=1, c=2, p=3:
m <- c("a" = 1, "c" = 2, "p" = 3)
for (t_2 in levels(commits_t1$label)) {
  i2 <- m[t_2]
  for (t_1 in levels(commits_t1$label)) {
    i1 <- m[t_1]
    for (t_0 in levels(commits_t1$label)) {
      i0 <- m[t_0]

      # Sum how often we went from t_2, over t_1, to t_0
      transprobs_2ndOrder[i0, i1, i2] <- transprobs_2ndOrder[i0, i1, i2] +
        sum(commits_t2$label_t2 == t_2 &
            commits_t2$label_t1 == t_1 &
            commits_t2$label == t_0)
    }
  }

  # Normalize each 3x3x1 tensor:
  n <- transprobs_2ndOrder[,, i2] / sum(transprobs_2ndOrder[,, i2])
  transprobs_2ndOrder[,, i2] <- array(n, dim = dim(n))
}

#transprobs_2ndOrder <- transprobs_2ndOrder / sum(transprobs_2ndOrder)
print(transprobs_2ndOrder)
```

```
## <A 3x3x3 dense tensor>
## , , a
##
##      a    c    p
## a 0.12 0.00 0.08
## c 0.14 0.06 0.02
## p 0.18 0.20 0.20
##
## , , c
##
##            a          c          p
## a 0.04651163 0.04651163 0.02325581
## c 0.06976744 0.20930233 0.04651163
## p 0.04651163 0.16279070 0.34883721
##
## , , p
##
##            a          c          p
```

```
## a 0.06521739 0.03260870 0.06521739
## c 0.05434783 0.03260870 0.13043478
## p 0.02173913 0.08695652 0.51086957
```

As an example, to go from `p` to `c` and then `a`, the probability is 0.0326087. We have to use the `m[label]`-notation, as indexing of dimensions does not work on other dimensions other than the last for some reason.

$$\phi_t^1(j) = \tag{1}$$
$$\phi_2^2(j) = \tag{2}$$
$$\vdots \tag{3}$$
$$= \frac{\sum_{i=i}^{N} \left[\phi_{t-1}(i) \; \boldsymbol{A}_{ij} \; \boldsymbol{b}_j(O_t) \right]}{\sum_{i=1}^{N} \phi_1(i)}. \tag{4}$$