Practicum 2 (part A)

Choice model specification and estimation (LCM)

1. Estimation and assessment of the Latent Class Model (LCM)

Last week, you gained first-hand experience building and estimating the Multinomial Logit (MNL) model. You applied the MNL model to the Colorectal Cancer (CRC) screening dataset, which investigated the influence of screening characteristics on non-participation in CRC screening in the Netherlands. Using the MNL model, you obtained insights into preferences by assuming all respondents are identical and have the same preferences (i.e., preference homogeneity). Assuming preference homogeneity is often an unrealistic assumption.

In this practicum, you will learn to build the Latent Class Model (LCM). The LCM assumes that there are (latent) classes of respondents with different preferences but that all respondents are identical within each class. Preference heterogeneity is identified from unobserved (random) variables.

Before moving on, please follow the steps below to set up your environment correctly:

1. Go to Canvas > Modules > Week 5 > Tutorial 2
2. Download the syntax file "***Practicum2_lcm.R***"
3. Go to download and find "***Practicum2_lcm.R***"
4. Move the syntax file to the folder created for the last week's practical
5. Open the syntax file "***Practicum2_lcm.R***", and let's build an Latent Class model

When estimating an LCM, there is a trade-off between a good model fit (resembling the data as closely as possible) and the number of parameters (looking for the most parsimonious model). The more (latent) classes you have in your data, the better the model fit of the LCM will be (do you know why?), but the more parameters are in the model (do you know why?).

Before you start, let us remember Apollo's basic structure that we introduced last week.

How does the Apollo package work?

In the Apollo package, all types of choice models use the same code structure. Each choice model is created by using the same building blocks. As the model complexity increases, components are added to the standard building blocks.

The standard building blocks to build a Latent Class model are:

- *Building block: Inputs*

In the "Inputs" block (see steps 1-9 in the syntax file), the set of information must be explicitly defined to set up the R-environment correctly for the Apollo package. As we will see, there is one additional component (step 9) in the "Inputs" block compared to the MNL syntax.

We start by setting up the R-environment.

- *Step 1*: clear the memory of the R-environment

```
36  ### Step 1: Clear memory
37  rm(list = ls())
```

This step is optional but highly recommended (!). Clearing the memory removes all saved variables, data frames, and objects in your R-environment. When you do not clear the environment and you are (re)running a (adjusted) syntax file, it could result in error messages.

- *Step 2*: Set a working directory for the R-environment

```
39  ### Step 2: Set working directory for R initialization
40  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

This step is optional but highly recommended (!). In this line, we instruct the R-environment to use the working directory (i.e., folder) in which the syntax file is stored. By doing so, we do not need to specify the full path to specific files, such as the dataset file.

- *Step 3 + 4*: Load and initialize the Apollo code

```
### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()
```

Here, we load and initialize the Apollo package into the R-environment.

- *Step 5*: Set the core controls

```
### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = ,
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = ,
  # USER ACTION: Specify the column with the respondent id
  indivID         = ,
  # Define number of cores used during estimation (used to speed up estimation time)
  nCores          = 5,
  # Define seed used for any random number generation
  seed            = 100,
  # USER ACTION: Set path to the folder on your PC where the model results will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-processing syntax
  outputDirectory = paste(getwd(),'outputs',"practicum_lcmodel_3class_covar",sep=.Platform$file.sep)
)
```

As we have loaded and initialized the Apollo package in steps 3 and 4, we need to instruct Apollo how to call our model (= *modelName*), what our model is (= *modelDescr*), which column of our dataset contains the respondent identifiers (= *indivID*), and where to save the model results (*outputDirectory*).

Note that, compared to the MNL, one control ("*seed*") was added to the Latent Class model. It controls a random number generation process. Since the log-likelihood function of the LCM is not concave, there may exist multiple optima in the function that "trick" the optimization of the coefficients to an early stop. To avoid this issue, Apollo runs the Latent Class model several times using different random starting coefficient values to identify the best model. To ensure the results are reproducible, this control fixes the seed of the random number generation process.

- *Step 6*: Load the dataset into the R-environment

```
### Step 6: Load data
# Set path to the folder on your PC where the dataset is stored
path_data = paste(getwd(),'data',"dataset.csv",sep=.Platform$file.sep)
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)
```

Because we have set the working directory (see step 2) in which the syntax file and dataset are stored, we do not need to specify the path to the dataset. We only need to check the name of the dataset whether it is correct or not (see purple names).

- *Step 7*: Initialize all parameters that need to be estimated in the choice model

Here, taste (e.g., attribute coefficients and alternative specific constants) and class membership parameters (e.g., deltas and gammas) must be initialized with starting values prior to the model estimation procedure.

```
### Step 7: Initialise all parameters that needs to be estimated in your MNL model
# USER ACTION: Define the (1) class-specific and (2) class membership parameters
#              followed by assigning a starting value. The class-specific
#              alternative specific constant for the opt-out option and the
#              constants for the class membership models are already defined.
#              Please, complete the list with the parameters that are missing.
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(# Class 1
              asc_out_1      = 0,

              # Class 2
              asc_out_2      = 0,

              # Class 3
              asc_out_3      = 0,

              # Class membership - class 1
              delta_1        = 0,

              # Class membership - class 2
              delta_2        = 0,

              # Class membership - class 3
              delta_3        = 0)
```

Note the differences between this code and the MNL.

- Taste coefficients: in the Latent Class model you specify one vector of parameters for each desired latent class.
- Class membership coefficients: these are new parameters specific to the Latent Class model. For each latent class there should be:

- A constant "delta" that supports the identification of class membership (and estimated class sizes) from unobserved heterogeneity. Alone, this constant assigns the same class membership probabilities to every respondent.
- You can add additional coefficients (let us say *gamma*) to be estimated with support of observed variables (e.g., sociodemographic, attitudes). Combined with the constant, these coefficients allow each respondent to have different membership probabilities conditional on their observed characteristics.

Action required! Go to your **Practicum5_lcmodel.R** file, read the user actions, and complete the code in step 7. But do not run the syntax file yet (!).

- *Step 8*: Define which parameters should kept fixed during the estimation

```
### Step 8
##  USER ACTION: Complete the list with parameters (as initialised above) that
##               should be kept fixed during estimation (in quotes)
apollo_fixed = c("delta_3")
```

Remember that "*apollo_fixed*" is a list of parameters that should be kept fixed at their initial values. Just like with one level of each categorical attribute, the membership coefficients of one latent class also can't be estimated.

Action required! Go to your **Practicum5_lcmodel.R** file, read the user actions, and complete the code in **step 8**. But do not run the syntax file yet (!).

- *Step 9*: Detail the class membership model

This is the new component of the "*Inputs*" block. Apollo does not recognize the structure of the coefficients initialized in **step 7** ("*apollo_beta*"). This component is where the coefficients are mapped into the Latent Class model structure.

```
### Step 9: Define class membership model
apollo_lcPars=function(apollo_beta, apollo_inputs){
  lcpars = list()
  ## USER ACTION: Complete the empty lists by specifying the missing class-specific parameters
  ##              which are needed for the class-specific utility functions
  lcpars[["asc_out"]] = list(asc_out_1, asc_out_2, asc_out_3)
  lcpars[["b_eff"]] = list()
  lcpars[["b_fneg"]] = list()
  lcpars[["b_freq_2yr"]] = list()
  lcpars[["b_freq_3yr"]] = list()
  lcpars[["b_wdiag_2wks"]] = list()
  lcpars[["b_wdiag_3wks"]] = list()
  lcpars[["b_wfup_4wks"]] = list()
  lcpars[["b_wfup_8wks"]] = list()

  ## List of class-membership functions:
  ##  These must use the same names as in classAlloc_settings (see below), order is irrelevant
  V=list()
  # USER ACTION: Define class-membership function for class 1
  V[["class_1"]] = delta_1

  # USER ACTION: Define class-membership functions for class 2
  V[["class_2"]] = delta_2

  # USER ACTION: Define class-membership functions for class 3
  V[["class_3"]] = delta_3

  ## Define settings for class-membership model
  classAlloc_settings = list(
    # USER ACTION: Attach class-membership functions to the respective classes
    classes      = c(class_1=, class_2=, class_3=),
    # USER ACTION: Define which classes are "available" in our study, all classes are "available"
    avail        = ,
    # USER ACTION: Attach list of class-membership functions
    utilities    =
  )

  lcpars[["pi_values"]] = apollo_classAlloc(classAlloc_settings)
  return(lcpars)
}
```

The "*lcPars*" list maps the taste coefficients into the attributes. You can recognize the attributes from last week's practical. In the Latent Class model, for every attribute/attribute level there will be a sub-list of the coefficients defined in **step 7** associated with the attribute/attribute level. Each sub list will contain as many coefficients as the desired latent classes.

The "*V*" list maps the coefficients of the membership function into each latent class. When the membership class has only constants the mapping is as you see in the figure above. To add observed variables, the mapping should be completed multiplying one coefficient by one observed variable as in the utility function.

Action required! Go to your **Practicum5_lcmodel.R** file, read the user actions, and complete the code in **step 10**. But do not run the syntax file yet (!).

- *Building block: Validate inputs*

The final step in preparing the code and data for model estimation is to make a call to:

- *Step 10*: Validation of the inputs

```
146  ### Step 10: Checkpoint for model inputs
147  apollo_inputs = apollo_validateInputs()
```

The function runs several checks and produces a consolidated list of model inputs. This function takes no arguments but looks at the R-environment for the various inputs required to build and estimate the model.

- *Step 11*: Define the choice model and likelihood function

```
### Step 11: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

  ### Attach inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### Define settings for MNL model component that are generic across classes
  mnl_settings = list(
    # USER ACTION: Attach utility functions to the alternatives in your dataset
    alternatives  = c(ALT1=, ALT2=, ALT3=),
    # USER ACTION: Define which alternatives are "available" in each choice task; in our study, all alternatives are "available"
    avail         = list(ALT1=, ALT2=, ALT3=),
    # USER ACTION: Specify the column containing the chosen alternative; beware, no dummies are used (!)
    choiceVar     =
  )

  ### List of utility functions for each latent class: these must use the same names as in mnl_settings (see above), order is irrelevant
  # USER ACTION: Set number of latent classes you are estimating in the model
  ##       Note: You can call class-specific parameters by NAME_PARAM[[s]]; see example in ALT3 for the class-specific
  ##             alternative specific constant
  for(s in 1:'REPLACE THIS WITH THE NUMBER OF CLASSES YOU WANT TO ESTIMATE'){
    V=list()
    # USER ACTION: Define utility function for alternative 1 for class "s"
    V[["ALT1"]] =

    # USER ACTION: Define utility function for alternative 2 for class "s"
    V[["ALT2"]] =

    # USER ACTION: Define utility function for alternative 3 for class "s"
    V[["ALT3"]] = asc_out[[s]]

    mnl_settings$utilities = V
    mnl_settings$componentName = paste0("Class_",s)

    ### Compute within-class choice probabilities using MNL model
    P[[paste0("Class_",s)]] = apollo_mnl(mnl_settings, functionality)

    ### Take product across observations for same individual (i.e., considering the panel structure of the data)
    P[[paste0("Class_",s)]] = apollo_panelProd(P[[paste0("Class_",s)]], apollo_inputs ,functionality)
  }

  ### Compute latent class model probabilities
  lc_settings   = list(inClassProb = P, classProb=pi_values)
  P[["model"]] = apollo_lc(lc_settings, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}
```

In the figure above, the code structure *for(s in 1:S)* {} is a loop. It means that the command inside the loop will be repeated *S* times and *[[s]]* will be replace by the number controlling the loop lap (1, 2, or 3 if *S*=3).

Therefore, there will be 9 utility functions or 3 (as in the MNL) for each class. The loop structure helps to automate the process of writing the utility functions.

Question 1.1 Consider an LCM with 2 and 3 classes. Write down the class-specific utility functions for all three choice alternatives, assuming that the attributes of "Effectiveness" and "Risk false negative" are coded as linear.

Answer:

Question 1.2 What is the difference in the number of parameters between the model with 2 and 3 classes, considering that no explanatory variables are included?

Answer:

Before moving on to the following questions, please carefully read the following points:

- The LCM probabilistically allocate respondents to classes during the estimation procedure. Because we add and/or remove variables in the model and because of the probabilistic nature of the class-membership model, the LCM can use different classes as a reference class each time you (re-)estimate the model. Although the classes are the same in terms of class shares and class-specific preference parameters, the order of the classes can differ from already estimated LCMs.
- When (re)estimating a choice model in Apollo, please do not forget to change the model's name and output directory in step 5 of the R-code.

- *Building block: Estimation*

When you have completed step 11, you have initialized all essentials and built your MNL model. Now, it is time to start the estimation procedure.

- *Step 12*: Search for starting value

```
### Step 12: Searching for starting value (recommended to ensure model convergence!)
apollo_beta = apollo_searchStart(apollo_beta,
                                 apollo_fixed,
                                 apollo_probabilities,
                                 apollo_inputs,
                                 searchStart_settings=list(nCandidates=2))
```

This is another new step compared to the MNL model. In contrast with the MNL, the likelihood function of the MNL is not concave. It means that it might not have a global optimum. To avoid the risk of being trapped in a local optimum, Apollo initiates the estimation from several starting values and selects the most promising one.

- *Step 13*: Model estimation

```
### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs)
```

By running step 13, the estimation procedure will be started by Apollo. There are no actions required. It will be all handled automatically for you.

- *Building block: Output*

The last building block is to instruct Apollo to print the model output and save the results in the folder that we have specified in step 5 (see "*outputDirectory*").

- *Step 14*: Print model output with two-sided p-values

```
### Step 14: Print model output with two-sided p-values
### Note: if one-sided p-values are needed, set "printPVal=1" (p-values are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)
```

After completing the estimation procedure, we instruct Apollo on how to process the model results. For now, you do not need to adjust anything. However, in some situations, you might want additional information. The Apollo manual explains what information you can extract and how you can do this (see here, page 37).

Now let us interpret the results of the Latent Class model that you estimated.

Question 1.3 Complete the R-code by following the instructions indicated as **# USER ACTION** and estimate the LCM with a different number of classes to see which model fits best. You may limit your estimations to 2-3 classes to save time. Based on the model results, how many classes would you select for your analysis and why?

Answer:

In the following questions, we will use the LCM with 3 classes.

Question 1.4 Which class is larger, and how many respondents are assigned each class?

Answer:

Question 1.5 What do you conclude when you compare class 2 with the MNL model you estimated in Question 2.7 in last week's practicum?

Answer:

Question 1.6 What do you conclude about class 3, compared to class 2?

Answer:

Question 1.7 The CRC dataset also includes socio-demographic variables. Include gender, age, and educational background in the LCM with 3 classes as explanatory variables for the class membership. Estimate the model again. Does gender, age, and educational background relate to class membership and if so, how?

Answer: