FORTE 2021-2027

Documentation of the algorithm matching Swedish persons to EPO patents

Laurent Bergé

January 26, 2023

Abstract

The objective of this document is twofold. First, explain the requirements to apply the code to generate the data for the entire Swedish population. Second, it details the contents of the various data bases that are created to reach the final objective of the matching data set.

Contents

1 General presentation of the objective of the code

The overarching objective of the code is to create a data set with Swedish person identifiers associated to patent numbers. In our case, the final data set will contain the five main variables that follow:

- id_se: the Swedish individual identifier (unique for each person living in Sweden)
- id_inv_seq: the inventor-patent identifier. One `id_se` can be matched to several `id_inv_seq`. For instance if Bengt Akesson has 5 patents at the EPO, his `id_se` should be matched to 5 `id_inv_seq`.
- appln_id: application identifier of the patent
- type_match: how the `id_se`×`id_inv_seq` was matched. There are three ways: i) `EM` (via the first EM algorithm), ii) `address` (the inventor shared the same address with the Swedish person), iii) `EM-patent` (last step of the algorithm when patent information is added for the matching)
- prob: the probability to be a true match. Only present for Bayesian matching (EM algorithm), hence it is missing for `type_match="address"`.

This final data set is named ALGO-4_base_match_patents.fst.

Since these variables are central, henceforth I will use the variables names `id_se` and `id_inv_seq` directly without referring to their literal meaning.

2 Software requirements

To be able to run the code on the full Swedish data set, there are software requirements that are detailed here.

2.1 Software

The following software requirements:

- *R* version 3.6.0 or greater (see installation for windows, for mac or for linux).
- RStudio to be able to run the code interactively and debug if needed.
- compilation tools: the software contains C++ code that requires compilation. For Windows, only the software Rtools needs to be installed (beware the version, since it should match R's). For Mac, developer tools are required (see help here for R≥4.0.0 or here for lower versions). For linux, a compiler is required and possibly some development tools (see step 1 here).

2.2 R packages

Within R, the following packages need to be installed with `install.packages("pkg_name")` (note that these installations are included in the code in the file `.Rprofile`, detailed later):

- data.table: for data manipulation
- fst: to import/export data
- Rcpp: to run C++ code from R
- dreamerr: for error handling
- fixest: for some data manipulation functions

```
|-- Working directory
|   |-- main.R
|   |-- src_algo_steps.R
|   |-- src_EM.R
|   |-- src_utilities.R
|   |-- src
|   |   |-- EM_functions.cpp
|   |   |-- integers.cpp
|   |   |-- string_dist.cpp
|   |-- _DATA
|   |   |-- _RAW
|   |   |   |-- STAT-SE
|   |   |   |   |-- OE_lev_RTB_1978.txt
|   |   |   |   |-- ...
|   |   |   |   |-- OE_lev_RTB_2020.txt
|   |   |   |   |-- OE_lev_Jobb_1985.txt
|   |   |   |   |-- ...
|   |   |   |   |-- OE_lev_Jobb_2020.txt
|   |   |   |-- OECD
|   |   |   |   |-- 202202_EPO_Inv_reg_small.txt
|   |   |   |   |-- 202202_EPO_IPC_small.txt
|   |   |   |   |-- 202202_EPO_App_reg_small.txt
```

Table 1: Summary of the data structure.

3 Running the code

To be able to fully run the code, the (RStudio) project must have a specific structure. All files must be located, and named, in a very precise way: any deviation will lead to failure. The file structure required is given in Table 1.

There are three main sets of files that are now described:

- `main.R`: high level functions to run the full algorithm. The user should interact (mostly) only with that file.
- source code: all the functions needed to run the algorithm (this is where the heavy lifting is)
- input data sets: the source data

3.1 The main function: `main.R`

This file has two parts. The first part is the code to run the algorithm. The second illustrates two functions which have general scope and can be readily applied to other projects.

3.1.1 Running the algorithm

The file `main.R` contains the commands to fully run all the algorithm, up to the final data set containing the matches between inventors and Swedish persons.

It is composed of 7 functions, each linked to a single step of the algorithm. These functions are special since they do not return anything. Instead they save directly the data sets on the disk. This feature allows the functions to be run asynchronously (i.e. across sessions): the information created is never lost and the user does not need to take care of data management.

For any of these functions, by default if the output file is already created and up to date (i.e. posterior to the input data), the function is skipped. If for some reason the user wants to re-run the code anyway, he can use the argument `hard = TRUE`.

The 7 functions are:

- step0_cleaning_se
- step0_cleaning_patents
- step1_name_matching
- step2_bilateral_vars
- step3_EM_algorithm
- step4_patent_match
- step5_descriptive

For complex steps, like `step0_cleaning_patents`, the function will call sub functions.[1] At any time, these sub functions can be called directly in lieu of the main function (there are simply more of them). The source code for these functions is in the file `src_algo_steps.R`.

3.1.2 General functions for application to other projects

In a second part, `main.R` illustrates two functions of general scope created for this project. These functions are critical in the algorithm and can be readily applied to many other projects:

- match_by_name (in `src_utilities.R`): matches identities from two different data sets using names. Always takes into account the maximum, non-contradictory, set of information. Also allows the fuzzy matching of names (while retaining this information), name swapping (which is especially useful for last names in the case of mariage), and using only initials.
- em_matching (in `src_EM.R`): algorithm to create groups of observations based on distance variables. This is typically applied to "potential" matches with the objective to create two groups of such matches. The outcome is a probability to belong to each group, hence it allows to separate true matches from wrong matches and gives an estimation of the confidence of the classification. The classification is totally data driven. One can easily see if the algorithm makes sense by making use of the `summary` and `plot` methods to figure out the influence of each variable and the distribution of the variables' parameters per group.

There is one example for each of these functions in the last section of `main.R`. The examples are directly taken from their application in the project. Both these functions are documented in the source code.

3.2 Source code

There is two types of source code: R and C++ files.

[1] For example `step0_cleaning_patents` relies on the following sub functions: 1) `step0_PATENT_inv`, 2) `step0_PATENT_app`, 3) `step0_PATENT_add`, 4) `step0_PATENT_add_missing`, 5) `step0_PATENT_names` and 6) `step0_PATENT_employer`.

R files. There are 4 R source files:

- src_utilities.R: various utilities, mostly for data management and cleaning
- src_EM.R: the R side of the EM algorithm
- src_algo_steps.R: all the steps of the matching algorithm
- .Rprofile: this is a special file that is launched at startup, loads all the required R packages and functions, also compiles and loads the C++ code

All these files **must be located in the working directory**.

C++ files. There are 3 C++ source files:

- EM_functions.cpp: the `C++` side of the EM algorithm
- integers.cpp: internal code of a widely used data management tool to create indexes (very fast)
- string_dist.cpp: fast implementation of various string distances used in the algorithm

All these files **must be located in the folder `src`, itself located in the working directory**.

3.3 Input data sets

There is two types of input data sets: 1) the Swedish data sets on individuals and employment, and 2) patent related data.

Statistics Sweden data. Starting from the working directory, the folder `_DATA/_RAW/STAT-SE` **must** contain the files `OE_lev_Jobb_xxxx.txt` (with `xxxx` the year) and `OE_lev_RTB_xxxx.txt` (with `xxxx` the year). Beware that there must be one file per year, the start and end year do not matter. These files must contain the variables described in Section 4.1.1, that is, they must be similar to the data sets used for development.

Again, all these input files must be in the folder `_DATA/_RAW/STAT-SE`. However, if for some reason it is not possible, you can modify the option `LOCATION_STAT_SE` to provide an alternative location of these input files. The code to modify `LOCATION_STAT_SE` is at the beginning of the file `.Rprofile`.

Patent data. The input patent data comes from the REGPAT data base from the OECD (Maraut et al., 2008). We use three data sets:

- 202202_EPO_Inv_reg_small.txt: information on the identity of the inventors, including the addresses
- 202202_EPO_IPC_small.txt: contains IPC (technological) codes for the patents, as well as the priority years
- 202202_EPO_App_reg_small.txt: information on the applicants, including the addresses

These files are an excerpt of the main REGPAT data sets, the only difference is that they include only patents produced by inventors whose address is reported to be in Sweden.[2]

These files must be placed in the folder `_DATA/_RAW/OECD`. These files are provided and their names must not be modified.

[2]More precisely, it contains the information on all the patents for which at least one inventor reports a personal address located in Sweden.

As for the Statistics Sweden data, it is possible to change the location of those input files by modifying the option `LOCATION_OECD` to provide an alternative location of these input files. The code to modify `LOCATION_OECD` is at the beginning of the files `.Rprofile`.

3.4 How to

At startup, thanks to the `.Rprofile` file (see explanations in the next paragraph) all the required functions will be automatically loaded. Then the user simply has to run the commands in `main.R`. This file has to be run line by line (there are 7 lines). To summarize the full process:

1. at startup, the `.Rprofile` file automatically:
   (a) installs the necessary R packages and loads them
   (b) compiles and loads the C++ functions
   (c) loads the necessary R functions
   (d) ⇒ the user has nothing to do in this step
2. the user has to run `main.R` line by line:
   (a) there are 7 functions to run, so 7 lines

Note that the steps of the algorithm are sequential (one step cannot be run before the previous step) but since their outcome data sets are saved on disk (in the `_DATA` folder), they can be run independently across different R sessions. This is especially helpful for debugging: since if there is a problem in the process there is no need to re-run all the previous code.

The `.Rprofile` file and automatic running: Important. Note that there is a file named `.Rprofile` which sources the code necessary to run the different steps of the algorithm. This is a special file that is sourced at the beginning of each R session. So when you open the R session (or open the RStudio project), this code will be run. Always. This is handy so you don't have to think about running that file.

However, when run *for the very first time*, several R packages will be installed and the C++ code will be compiled. These are critical steps which are usually error-prone. At this point, debugging may be needed to ensure everything works as expected. Since the code is run automatically, it can be a bit unsettling, but keep in mind that it is only the code contained in the `.Rprofile` file. If there are errors at that time, simply go to the `.Rprofile` file to run the code line by line and try to fix the issues (usually problems with package installation, like internet connections, or problems with the compilation of C++ linked to compilation tools that may not be present).

In particular, the code lines related to C++ of the form `Rcpp::sourceCpp("file_name.cpp", cacheDir = "_CPP_CACHE")`) may lead to **an error the very first time they are automatically run** and require to be run line by line.

3.5 Reading data produced by the code

The code produces two kind of files: `.fst` and `.RData` files.

`.fst` files. `.fst` files are simply tabular data sets saved in a compressed yet fast to open format. They are written/read by the R package fst. They can be read from R with the function `readfst` (internal to the algorithm, located in `src_utilities.R`) or `read_fst` (from the `fst` package).

Example from the project: `base_bilat = readfst("_DATA/ALGO-2_all.fst")`.

.RData files. `.RData` files are special: they contain R objects saved "as such". This format is only used to save the results of the EM algorithms which contain multiple data of various types. To load the object, you need to use the function `load` within R.

Example from the project: `load("_DATA/ALGO-4_em-result-patent_match.RData")`.

The previous line will make the object `res_em_pat` loaded within the R session. It can then readily be used (for instance `plot(res_em_pat)`).

4 Description of the Tables produced by the algorithm

The algorithm contains 5 steps. At each step, several data sets are produced and saved on the disk. This Section gives an overview of all the tables produced by displaying their variables. The detailed description of the content of each variable is written in a dedicated section accessed by clicking on the hyperlink in the table title.

4.1 Tables of Step 0: Data cleaning

4.1.1 Swedish data

input:

OE_lev_RTB_xxxx.txt
LopNr
Kon
FodelseAr
TillTal
FNamn
MNamn
ENamn
Namn
CoAdr
Adress
PostNr
PostOrt

⟹ **output:**

STAT-SE_indiv.fst		
Renamed variables + treatment		
	old name	*treatment*
id_se	LopNr	
gender	Kon	
birth_year	FodelseAr	
firstname_code	TillTal	
firstname_all	FNamn	lower case
maiden_name	MNamn	lower case
family_name	ENamn	lower case
street_location	Adress	cleaning
postcode	PostNr	
city	PostOrt	cleaning
New variables		
year_range		
year_start		
year_end		
id_se_seq		
Deleted variables		
Namn		
CoAdr		

OE_lev_Jobb_xxxx.txt
LopNr
LopNr_PeOrgNr
FtgNamn
AstKommun
FtgKommun
FtgAdr
FtgPostNr
FtgPostAnst

input: ⟹ **output:**

STAT-SE_job.fst		
Renamed variables + treatment		
	old name	*treatment*
id_se	LopNr	
id_firm	LopNr_PeOrgNr	
se_emp	FtgNamn	cleaning
se_emp_postcode	FtgPostNr	
se_emp_city	FtgPostAnst	cleaning
se_emp_street_name	FtgAdr	cleaning
se_emp_street_nb	FtgAdr	cleaning
se_emp_building	FtgAdr	cleaning
New variables		
year_range		
year_start		
year_end		
Deleted variables		
LopNr_PeOrgNr		
AstKommun		
FtgKommun		

STAT-SE_indiv-addresses.fst
id_se
postcode
city
street_name
street_nb
building

STAT-SE_indiv-names.fst
id_se_seq
id_se
first_name_1
first_name_2
first_name_3
fam_name_1
fam_name_2
name_raw
birth_year
year_start
year_end

swedish_first_names.fst
name
freq
name_ascii

4.1.2 Patent data

OECD_inventors.fst
id_inv_seq
appln_id
epo_app_nbr
epo_pub_nbr
person_id
inv_name
city
postal_code
inv_country
inv_address
year_prio

OECD_applicants.fst
appln_id
epo_app_nbr
epo_pub_nbr
person_id
app_name
city
postal_code
app_country
app_address

PATENT_inv-address.fst
raw
city
postcode
street_name
street_nb
building

PATENT_address-missing.fst
id_inv_seq
address_missing

PATENT_inv-names.fst
id_inv_seq
name_raw
first_name_1
first_name_2
first_name_3
fam_name_1
fam_name_2
year_prio

PATENT_inv-employer.fst
appln_id
id_inv_seq
inv_emp
inv_emp_postcode
inv_emp_city
inv_emp_street_name
inv_emp_street_nb
inv_emp_building
year_prio

4.2 Tables of Step 1: Matching names

ALGO-1_matched-names.fst
id_inv_seq
id_se
match_type
match_qual

4.3 Tables of Step 2: Creating bilateral variables

ALGO-2_address.fst
id_inv_seq
id_se
address_cat
address_match

ALGO-2_employer.fst
id_inv_seq
id_se
same_emp

ALGO-2_age.fst
id_inv_seq
id_se
age

ALGO-2_name_proba.fst
id_inv_seq
id_se
proba_name
proba_name_rank

ALGO-2_all.fst
id_inv_seq
id_se
match_qual
address_cat
age
same_emp
proba_name
proba_name_rank
n_match

4.4 Tables of Step 3: EM algorithm

ALGO-3_em-result-n_match-LE-5.RData
n
tau
prms
ll_all
method
influence
info_vars

ALGO-3_base-all-matches.fst
id_inv_seq
id_se
type_match
prob

4.5 Tables of Step 4: Adding patent information

ALGO-4_bilateral_patents.fst
id_inv_seq
id_se
match_qual
address_cat
address_match
age
same_emp
proba_name
proba_name_rank
n_match
same_coauth
max_tech_sim
same_app

ALGO-4_em-result-patent_match.RData
n
tau
prms
ll_all
method
influence
info_vars

ALGO-4_base_match_patents.fst
id_inv_seq
id_se
appln_id
type_match
prob

4.6 Tables of Step 5: Descriptive data sets to understand the matching

ALGO-5_EM-sample-info.fst	
id_inv_seq	max_tech_sim
id_se	same_app
match_qual	type_match
address_cat	prob
address_match	max_prob
age	inv_name
same_emp	year_prio
proba_name	inv_address
proba_name_rank	inv_emp
n_match	se_name
prob_qual	se_address
same_coauth	se_emp

5 Details of the contents of the data sets

5.1 Step 0: Data cleaning

5.1.1 STAT-SE_indiv.fst

Data set containing the names and addresses of Swedish persons. This is a concatenation of the raw files named `OE_lev_RTB_xxxx.txt`.

This data set originally contains one line per person per year. Persons can: i) change name (e.g. marriage), ii) change address. However both these changes are rare. Thus, to avoid unnecessary duplicate lines, identical names and addresses are aggregated and the starting and ending years of the $name \times address$ are recorded so that no information is lost (while reducing considerably the size of the data set).

Sources. The raw data sets are `OE_lev_RTB_xxxx.txt`. There is one such data set per year. (Link to Swedish data.)

Variables.

- id_se: the unique identifier of the Swedish person
- id_se_seq: sequential $id\_se \times name \times address$ identifier
- gender: gender, 1 (male) or 2 (female)
- birth_year: year of birth
- firstname_code: when available, a two digits code identifying which first name is the one used by the person. For instance, “20” means that the person uses her second first name. Or “32” means that the person uses her third and second first name. The vast majority of the cases is “10”. This information will be used when we match on names (Section 4.2).
- firstname_all: space separated list of all first names. Examples: “caroline linda marie” or “ann-louise”.

- maiden_name: maiden name, this information is rarely present (only in 3% of the case in the test sample). This information will be used when matching by name (Section 4.2).
- family_name: the family name. May contain several. Examples: “jäghammar”, “larsson-jones”, “hådén del pino”.
- street_location: the street name of the person’s personal address
- postcode: postcode of the person’s personal address
- city: the city of the person’s personal address
- year_range: character variable compactly representing all the years
- year_start: year of the first occurrence of this person’s name and address
- year_end: year of the last consecutive occurrence of this person’s name and address

5.1.2 STAT-SE_job.fst

Data set containing the raw data on job information for Swedish persons. This is a concatenation of the raw files named `OE_lev_Jobb_xxxx.txt` (with `xxxx` a year). Importantly the information is aggregated for each individual so that the number of observations in this data set is substantially lower than the sum of observations in all the `OE_lev_Jobb_xxxx.txt` data sets.

Sources. The raw data sets are `OE_lev_Jobb_xxxx.txt`. There is one such data set per year. (Link to Swedish data.)

Treatment. We extract different address components from the raw address field. The street number is removed from the street name and any extra information is cleaned except the building number which is also extracted. Some abbreviations in the names are also corrected (e.g. “sankt” can be written in various forms).

Details can be found in the function `extract_se_address_statse`.

Variables.

- id_se: the unique identifier of the Swedish person
- se_emp: name of the employer
- se_emp_postcode: postcode of the employer’s address
- se_emp_city: city of the employer’s address
- se_emp_street_name: street name without number of the employer’s address
- se_emp_street_nb: street number of the employer’s address
- se_emp_building: building of the employer’s address
- year_range: character variable compactly representing all the years
- year_start: start year of the person having this employer
- year_end: end year of the person having this employer

5.1.3 STAT-SE_indiv-addresses.fst

Starting from the raw address, a parsing is performed to cleanly extract the various information contained in the addresses.

Sources. The source of this data is STAT-SE_indiv.fst. (Link to Swedish data.)

Treatment. The street number is removed from the street name and any extra information is cleaned except the building number which is also extracted. Some abbreviations in the names are also corrected (e.g. "sankt" can be written in various forms).

Details can be found in the function `extract_se_address_statse`.

Variables.

- id_se: the unique identifier of the Swedish person
- postcode: postcode
- city: city name. Example: "stockholm 16 " becomes "stockholm".
- street_name: full street name without number. Example: "sandgardsg 34 b" becomes "sandgardsg".
- street_nb: the number of the address. Example: sandgardsg 34 b" becomes "34".
- building: the building number, when available (mostly missing). Example: "sandgardsg 34 b" becomes "b".

5.1.4 STAT-SE_indiv-names.fst

The names of the persons is a critical information. This data set contains the name information of the Swedish persons.

Sources. The source of this data is STAT-SE_indiv.fst. (Link to Swedish data.)

Treatment. When appropriate the names are cleaned, completed and modified to account for spelling variations or usage names. The main modifications performed are:

- cleaning the suffixes and merging the particles in the family names
- duplicating the person with respect to his usage name when the usage name is different from "10" (for example if the usage name is "20" we duplicate the person in which her first first name is only her second first name)
- completing initials with the full names when possible using the information in previous/later years
- applying spelling variations (some letters, like "ö" can be written "oe" in patents, so we duplicate the names accounting for these variations)

Variables.

- id_se_seq: sequential $id\_se \times name \times address$ identifier
- id_se: the unique identifier of the Swedish person
- first_name_1: first first name
- first_name_2: second first name
- first_name_3: third first name
- fam_name_1: first family name
- fam_name_2: second family name
- name_raw: all first names concatenated to all family names
- birth_year: year of birth
- year_start: start year of the person having this specific name
- year_end: end year of the person having this specific name

Comment. The number of lines in this data set is larger from the input data set due to the name variations for some persons.

5.1.5 swedish_first_names.fst

This data set listing the Swedish first names and their frequencies is used to identify whether last names should be substituted with first names in the patent data set.

Source. The source of information on the names is 5.1.1. (Link to Swedish data.)

Variables.

- name: the first name
- freq: the frequency of this first name in the Swedish population
- name_ascii: the first name in ASCII format (i.e. without accentuated characters)

5.1.6 OECD_inventors.fst

Data set containing the information on inventors and patents.

Sources. The information comes directly from the REGPAT data set of the OECD (Maraut et al., 2008).
The raw data sets used are:

- 202202_EPO_Inv_reg_small.txt for all information but the priority date
- 202202_EPO_IPC_small.txt for the priority date

(Link to Patent data.)

Restriction. The following restriction is applied:

- only inventors whose country is Sweden are selected (all the other inventors are dropped)

Treatment. The raw address (`inv_address`) is cleaned: all characters are turned into ASCII and set to lower case.

Variables.

- id_inv_seq: sequential inventor-patent identifier
- appln_id: the patent application id
- epo_app_nbr: the EPO application number
- epo_pub_nbr the EPO publication number
- person_id: sequential identifier of $name \times addresses$
- inv_name: raw name of the inventor (character string that contains both the first and the last name). Example: "vallestad, anne e., sivil-ing.", "hjelmvik, torbernt".
- city: city name as extracted by the OECD. Example: "JaRFaLLA".
- postal_code: postcode as extracted by the OECD
- inv_country: country of the inventor, based on the inventor's address
- inv_address: raw address. Example: "orionvagen 20,s-175 60 jarfalla".
- year_prio: priority year of the patent

5.1.7 OECD_applicants.fst

Information on the applicants of the patents produced by Swedish inventors.

Sources. The information comes directly from the REGPAT data set of the OECD (Maraut et al., 2008). The raw data sets used is:

- 202202_EPO_App_reg_small.txt

(Link to Patent data.)

Restriction. The following restriction is applied:

- only patents produced by Swedish inventors are selected (the country is identified with the inventor's address)

Treatment. The variables `app_name` and `app_address` are set to ASCII and put to lowercase.

Variables.

- appln_id: the patent application id
- epo_app_nbr: EPO application number
- epo_pub_nbr: the EPO publication number
- person_id: sequential identifier of $name \times addresses$
- app_name: applicant name. Example: “modul-system sweden ab”, “telefonaktiebolaget lm ericsson (publ)”.
- city: city of the applicant as extracted by the OECD
- postal_code: postcode of the applicant as extracted by the OECD
- app_country: country of the applicant, based on the applicant’s address
- app_address: raw applicant’s address. Example: “sjobacka fallhagen 1,s-59076 vreta kloster”, “dragarbrunnsgatan 35,753 20 uppsala”.

5.1.8 PATENT_inv-address.fst

The address of the inventors as deduced from the raw address character string. The objective is to extract a formatted address set in a similar format to the data set STAT-SE_indiv-addresses.fst.

Sources. The raw data comes from OECD_inventors.fst. (Link to Patent data.)

Treatment. The form of the addresses vary wildly, the algorithm extracting the information tries to drop the non informative parts, like company names, and deal with the various formats. Details can be found in the function `extract_se_address_patents`.

Variables.

- raw: raw address in free text form. Example: “hogabergsvaden 17a,436 40 askim”.
- city: extracted city name. Example: “hogabergsvaden 17a,436 40 askim” becomes “askim”.
- postcode: extracted postcode. Example: “hogabergsvaden 17a,436 40 askim” becomes “43640”.
- street_name: extracted street name without number. Example: “hogabergsvaden 17a,436 40 askim” becomes “hogabergsvaden”.
- street_nb: extracted street number. Example: “hogabergsvaden 17a,436 40 askim” becomes “17”.
- building: extracted building number. Example: “hogabergsvaden 17a,436 40 askim” becomes “a”.

Note on the key. This data set contains exactly the same number of observations as OECD_inventors.fst and in the exact same order. So the observation number is identical to the variable `id_inv_seq` of OECD_inventors.fst.

5.1.9 PATENT_address-missing.fst

In patent data it is common that the inventor address is not disclosed, the patent reporting instead the applicant address. In that case we should consider that the address of the inventor is missing. This data set compiles the inventor addresses that should not be considered as their real address.

Sources. The data comes from OECD_inventors.fst and OECD_applicants.fst. (Link to Patent data.)

Treatment. The address of the applicant is compared to the address of the inventor and if there is a match, the address of the inventor is flagged as missing. Note that extra care is taken to dismiss applicants that are individual inventors, in which case the applicant's address is truly the address of the inventor.

Variables.

- id_inv_seq: sequential inventor-patent identifier
- address_missing: binary variable equal to 1 if the address is missing

Note on the key. This data set contains exactly the same number of observations as OECD_inventors.fst and in the exact same order.

5.1.10 PATENT_inv-names.fst

The name is a very important information. Unfortunately, the full name (first and family names) in the patent data set comes in a not always consistent character string. Hence a specific algorithm is to be employed to extract the information on names as precisely as possible. The final data is structured in a similar way as STAT-SE_indiv-names.fst.

Sources. The sources of the data are: i) OECD_inventors.fst for the names, ii) STAT-SE_indiv-names.fst to identify valid Swedish first names. (Link to Patent data.)

Treatment. Some of the main treatment are: i) merge particles, ii) clean extraneous information, iii) sort out company names from real first names. Deduce which name is a first name

Details can be found in the function *extract_name_patents*.

Variables.

- id_inv_seq: sequential inventor-patent identifier
- name_raw: the full raw character string identifying the inventor name
- first_name_1: first first name
- first_name_2: second first name
- first_name_3: third first name
- fam_name_1: first family name
- fam_name_2: second family name
- year_prio: priority year of the patent

5.1.11 PATENT_inv-employer.fst

This data set contains the information on the applicants of the inventor-patent.

Sources. The sources of the data is: i) OECD_applicants.fst for the information on the applicants, ii) OECD_inventors.fst for the information on the priority year of the patent. (Link to Patent data.)

Treatment. The name of the applicant (`inv_emp`) is slightly cleaned (acronyms are harmonized as well as company diminutives). The address is extracted from the raw address character string. The process is similar to PATENT_inv-address.fst.

Variables.

- appln_id: the patent application id
- id_inv_seq: sequential inventor-patent identifier
- inv_emp: applicant name of the patent
- inv_emp_postcode: postcode of the applicant
- inv_emp_city: postcode of the applicant's address
- inv_emp_street_name: street name without number of the applicant's address
- inv_emp_street_nb: street number of the applicant's address
- inv_emp_building: building number of the applicant's address
- year_prio: priority year of the patent

5.2 Step 1: Matching names

5.2.1 ALGO-1_matched-names.fst

This data set is the outcome of the algorithm matching the inventors identities and the Swedish persons by name. This is a critical step and special care is taken to ensure the matching makes sense and is as exhaustive as possible. This data set gives all the possible $id\_se \times id\_inv\_seq$ pairs.

Sources. The information on the names come from STAT-SE_indiv-names.fst for the Swedish persons and PATENT_inv-names.fst for the inventors. (Link to Tables of Step 1: Matching names.)

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- match_type: character vector reporting how the match of the names was performed. Example: "FSTLM" means that the match has been performed on the first, second, and third first names, and on the first and second last names (in this case, this is an extremely strong match). The meaning of the letters are: F: first first name, S: second first name, T: third first name, L: first last name, M: second last name

("M" is for *maiden name*). An extra code can be put following the letters: "?": means that the name was fuzzy matched, ".": means that only the initial was available for the matching. Example: "F?S.L" means that the first name was fuzzy matched, only the initial of the second name was available for matching, and the full last name was matched.

- match_qual: integer categorical variable measuring match quality, 1: low, 2: mid, 3: high. Typically names that were fuzzy matched tend to be low quality, the more the number of first/last names over which the matching is done, the higher the quality.

Examples. It is easier to see how the `match_type` codes relates to the names with an example:

	match_type	inventor name	Swedish name
	Without fuzzy matching		
1	FL	svensson, per	per-åke, svensson
2	FSL	jansson, lars-erik	lars-erik, jansson
3	FS.L	akesson, bengt a.	bengt-åke, åkesson
	With fuzzy matching		
4	F?L	svensson, bernt	berit marianne, svensson
5	F?S.L	karlsson, peter, c.	petter christer julius, söderstrand karlsson
6	FL?	johnsson, anders	thor anders rickard, johansson
7	FS?L	svensson, britt-mari	britt-marie maj, svensson

The inventor and person names in the previous table are the raw ones (they were only set to lower case). We can see that the strength of the match differ across cases and this will be taken into account when we decide whether a match is true or not in the EM step.

5.3 Step 2: Creating bilateral variables

5.3.1 ALGO-2_address.fst

This is an intermediary data set containing the comparison between inventor and Swedish persons addresses.

Sources. The source of the potential matches is ALGO-1_matched-names.fst, the sources for the addresses are PATENT_inv-address.fst, PATENT_address-missing.fst and STAT-SE_indiv-addresses.fst. (Link to Tables of Step 2: Creating bilateral variables.)

Treatment. For each potential $id\_se \times id\_inv\_seq$ match, a comparison is made between their respective addresses. All the fields (city, postcode, street name, street number and building number) are used to make the address comparison and detect whether the address is the same (i.e. a match) or not. Fuzzy matching is applied when appropriate to account for potential misspells.

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- address_cat: categorical variable from 0 to 2. They mean, 0: unmatched, 1: missing address and 2: matched.
- address_match: character categorical variable taking the following three values: unmatched, missing and matched.

Examples. Here are some examples of addresses that are considered identical by the algorithm:

inventor address	Swedish address
Without misspells	
ruddamsvagen 11, 5 tr,s-114 21 stockholm	ruddammsvagen 11 5 tr, 11421, stockholm
With misspells	
egnahemsv.5,s-61234 finspong	egnahemsvagen 5, 61234, finspang
klockargardsvagen 13,s-194 53 upplauds vasby	klockargardsv 13, 19400, upp vasby
altfioigatan 22,42146, v. frolunda	altfiolg 22, 42146, vastra frolunda
ruddamsvagen 11, 5 tr,s-114 21 stockholm	ruddammsvagen 11 5 tr, 11421, stockholm
fredrikslundsvagen 43,s-168 34 bromma	fredrikslundsv 43, 1 tr, 16834, bromma
erik dahlbergs alle 9b,s-115 20 stockholm	erik dahlbergsallen 9 b, 11520, stockholm
st. eriksgatan 43 a,112 34 stockholm	sankt eriksgatan 43 a 3 tr, 11234, stockholm
tavelsj v gen 18,se-120 59 sweden	tavelsjovagen 18 lgh 1201, 12059, arsta

5.3.2 ALGO-2_employer.fst

This is an intermediary data set containing the comparison between inventor and Swedish employers.

Sources. The source of the potential matches is ALGO-1_matched-names.fst, the sources for the addresses and names of the employers are PATENT_inv-employer.fst for the inventors and STAT-SE_job.fst for the Swedish persons. (Link to Tables of Step 2: Creating bilateral variables.)

Treatment. To assess whether two employers are the same, the algorithm uses the names and addresses of the employers. Name matching is fuzzy (while remaining conservative) to account for the elusive nature of company names. When the names do not match, an identical address can compensate given a minimal string proximity between the names.

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- same_emp: binary indicator, 1 meaning the employers are identical

Examples. Here are examples of employers that the algorithm considers identical:

inventor employer	Swedish person employer
telefonab lm ericsson	telefonabet lm ericsson
thermia varme ab	thermia ab
johansson christer	johansson lars goran christer
berggren berit	berggren bernt ove lennart
volvo technology corporation	abet volvo
pharmacia upjohn company llc	pharmacia upjohn ab
rexroth mecman ab	abet mecman
envac centralsug ab	abet centralsug

5.3.3 ALGO-2_age.fst

This intermediary data set contains the age at invention.

Sources. The source of the potential matches is ALGO-1_matched-names.fst, the source for the priority year of the patent (i.e. year of invention) is OECD_inventors.fst, the source of the date of birth is STAT-SE_indiv.fst. (Link to Tables of Step 2: Creating bilateral variables.)

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- age: the age at invention, i.e. the difference between the year of invention (priority year) and the birth date

5.3.4 ALGO-2_name_proba.fst

This intermediary data set contains the probability to have a given first name and last name. This frequency is used in the EM algorithm.

Sources. The source of the potential matches is ALGO-1_matched-names.fst, the source for the Swedish names is STAT-SE_indiv.fst and for the inventor names is PATENT_inv-names.fst. (Link to Tables of Step 2: Creating bilateral variables.)

Treatment. The full name probability is based on the first name and family name distributions in the Swedish population. For each Swedish person, the full name probability is the average of the probability of the (first) first-name and the family name. The same probability is computed for the inventors. The final probability is the product of the two (since the names can differ in case of fuzzy matching).

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person

- proba_name: continuous variable between 0 and 1, normalized by the largest name probability. The probability to have a given first first-name and family name combination. The first- and family-name probabilities are based on the distribution of names in the Swedish population. This is an average between the Swedish person probability and the inventor probability since the names of the two can be different due to fuzzy matching.
- proba_name_rank: the rank of the variable `proba_name` normalized into 0 and 1.

5.3.5 ALGO-2_all.fst

This is the final data set of the bilateral variables. It combines all the bilateral variables between the inventor and the Swedish person.

Sources. The source of the potential matches is ALGO-1_matched-names.fst. The source for: i) the address matches is ALGO-2_address.fst, ii) the employer matches is ALGO-2_employer.fst, iii) the age at invention is ALGO-2_age.fst, iv) the name probabilities is ALGO-2_name_proba.fst. (Link to Tables of Step 2: Creating bilateral variables.)

Treatment. All variables of this data set were created in the previous steps, except the variable `n_match`. This variable corresponds to the number of `id_se` (i.e. distinct Swedish persons) that were considered as a potential match for an `id_inv_seq` (from the name matching of ALGO-1_matched-names.fst).

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- match_qual: categorical variable measuring match quality, 1: low, 2: mid, 3: high.
- address_cat: categorical variable from 0 to 2. They mean, 0: unmatched, 1: missing address and 2: matched.
- address_match: character categorical variable taking the following three values: unmatched, missing and matched.
- age: the age at invention, i.e. the difference between the year of invention (priority year) and the birth date
- same_emp: binary indicator, 1 meaning the employers are identical
- proba_name: continuous variable between 0 and 1, normalized by the largest name probability. The probability to have a given first first-name and family name combination. The -name and family name probabilities are based on the distribution of names in the Swedish population. This is an average between the Swedish person probability and the inventor probability since the names of the two can be different due to fuzzy matching.
- proba_name_rank: the rank of the variable `proba_name` normalized into 0 and 1.
- n_match: number of `id_se` that are considered as potential matches for a single `id_inv_seq`

5.4 Step 3: EM algorithm

5.4.1 ALGO-3_em-result-n_match-LE-5.RData

This R object is a list that contains all the output of the EM algorithm classifying the $id\_se \times id\_inv\_seq$ matches as true matches. The end product of the algorithm is, for each $id\_se \times id\_inv\_seq$ pair, the probability to belong to the two classes (the two classes being the one of the true matches and the one of the false matches).

This object can be loaded with `load("_DATA/ALGO-4_em-result-patent_match.RData")`. The name of the object containing the EM output is `res_em`.

Sources. The EM algorithm is performed using the $id\_se \times id\_inv\_seq$ pairs and bilateral variables from the data set ALGO-2_all.fst. (Link to Tables of Step 3: EM algorithm.)

Restrictions. Note that only `id_inv_seq` matched to 5 or less `id_se` are used to run the algorithm. This is to avoid an over population of false matches (which would be over 95% by construction, many single patents are matched to hundreds of different Swedish persons) leading the algorithm to run poorly.

For `id_inv_seq` matched to 6 or more `id_se`, only the ones sharing the same address are matched (contrary to the EM algorithm which uses other information to infer the match).

Variables.

- n: number of observations
- tau: matrix of size $n \times 2$ of the probabilities to belong to each of the two classes. In this case the first column represents the probability of being a false match while the second column represents the probability of being a true match (the sum of the two columns equals to 1).
- prms: the generative parameters of each variable used in the EM algorithm, for each class. It is a list of length K with K the number of variables.
- ll_all: the likelihood evolution at each step of the algorithm
- method: the law used to model each variable. It is a vector of length K with K the number of variables.
- influence: it is a vector of length K giving approximately the influence of each variable on the classification. It provides the approximate number of observations which would change classes if the variable was dropped.
- info_vars: internal, generic information on each variable

Visualizing the results of the classification. You can use `summary(res_em)` or `plot(res_em)` to see the result of the classification.

5.4.2 ALGO-3_base-all-matches.fst

Once the EM algorithm is run, we gather the pairs having over 80% probability to be a true match. We also gather, for `id_inv_seq` matched to 6 or more `id_se`, the pairs which have the same address.

This data set contains all the $id\_se \times id\_inv\_seq$ pairs that are considered as matches.

Sources. The source of the matches of the EM algorithm is ALGO-3_em-result-n_match-LE-5.RData, the source for the address matching is ALGO-2_all.fst. (Link to Tables of Step 3: EM algorithm.)

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- type_match: categorical variable identifying how the $id\_se \times id\_inv\_seq$ pair was matched. Either: i) EM (via the first EM algorithm), or ii) address (the inventor shared the same address with the Swedish person).
- prob: the probability to be a true match. Only present for Bayesian matching (EM algorithm), hence it is missing for `type_match="address"`.

5.5 Step 4: Adding patent information

5.5.1 ALGO-4_bilateral_patents.fst

One the main data set identifying the matches has been created, we have a set of patents for the Swedish persons (`id_se`) who were matched. Using the $id\_se \times id\_inv\_seq$ pairs that could not be matched, we leverage the patent information to create new bilateral variables to help identify true matches.

This data set contains the new bilateral variables based on patent information.

Sources. Constructing this data set requires many sources. First, the `id_se` that were matched are from ALGO-3_base-all-matches.fst, and the set of all potential matches as well as existing bilateral variables are from ALGO-2_all.fst.

The information on patent inventors from the Swedish sample are from OECD_inventors.fst, while the information on the inventors in general are from `202202_EPO_Inv_reg_small.txt` of the REGPAT data base.

The information on patent applicants is from PATENT_inv-employer.fst.

The information on IPC codes (patent technological classes) are from `202202_EPO_IPC_small.txt` of the REGPAT data base.

(Link to Tables of Step 4: Adding patent information.)

Treatment. We create 3 additional bilateral variables based on patents:

1. whether the patent of `id_inv_seq` shares a co author in common with the pool of patents of the Swedish person (`id_se`)
2. whether, and at what level of precision, the patent of `id_inv_seq` shares IPC codes with the pool of patents of the Swedish person (`id_se`)
3. whether the patent of `id_inv_seq` shares an applicant in common with the pool of patents of the Swedish person (`id_se`)

Variables. The variables created in this step are in bold.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- match_qual: categorical variable measuring match quality, 1: low, 2: mid, 3: high.
- address_cat: categorical variable from 0 to 2. They mean, 0: unmatched, 1: missing address and 2: matched.
- address_match: character categorical variable taking the following three values: unmatched, missing and matched.
- age: the age at invention, i.e. the difference between the year of invention (priority year) and the birth date
- same_emp: binary indicator, 1 meaning the employers are identical
- proba_name: continuous variable between 0 and 1, normalized by the largest name probability. The probability to have a given first first-name and family name combination. The -name and family name probabilities are based on the distribution of names in the Swedish population. This is an average between the Swedish person probability and the inventor probability since the names of the two can be different due to fuzzy matching.
- proba_name_rank: the rank of the variable `proba_name` normalized into 0 and 1.
- n_match: number of `id_se` that are considered as potential matches for a single `id_inv_seq`
- **same_coauth**: binary variable taking value 1 if the patent of `id_inv_seq` shares at least one co-author in common with the patent pool of `id_se`
- **max_tech_sim**: integer categorical variable taking values 0, 1, 3, 4, or 7. It reprensents the largest IPC digit for which there was a matching between the IPCs of the patent of `id_inv_seq` and the patent pool of id_se. Illustration: if equal to 1, this means that the IPC codes match at most on IPC1, for example "**C**12M 1/00 Apparatus for enzymology or microbiology" and "**C**01D 3/00 Halides of sodium, potassium, or alkali metals in general" match only at most "C". If there was an IPC of `id_inv_seq` equal to "**C01D** 3/00 Halides of sodium, potassium, or alkali metals in general" and one of the patent pool of `id_se` equal to "**C01D** 5/00 Sulfates or sulfites of sodium, potassium, or alkali metals in general", then we would have `max_tech_sim=4`. A value of 0 means that there was no match, even at the largest level
- **same_app**: binary variable taking value 1 if at least one applicant of the patent of `id_inv_seq` matches at least one applicant of the patent pool of `id_se`.

5.5.2 ALGO-4_em-result-patent_match.RData

The results of the EM algorithm applied to the bilateral data set of potential matches enhanced with patent information. This is an R object consisting of a list of several elements.

This object can be loaded with `load("_DATA/ALGO-4_em-result-patent_match.RData")`. The name of the object containing the EM output is `res_em_pat`.

Source. The source of the potential matches as well as the bilateral variables is ALGO-4_bilateral_patents.fst. (Link to Tables of Step 4: Adding patent information.)

Restriction. This algorithm is run over only on $id\_se \times id\_inv\_seq$ pairs for which the id_se has been found to have at least one patent (in other words, has been matched to at least another `id_inv_seq` in the previous steps).

Variables.

- n: number of observations
- tau: matrix of size $n \times 2$ of the probabilities to belong to each of the two classes. In this case the first column represents the probability of being a false matche while the second column represents the probability of being a true match (the sum of the two columns equals to 1).
- prms: the generative parameters of each variable used in the EM algorithm, for each class. It is a list of length K with K the number of variables.
- ll_all: the likelihood evolution at each step of the algorithm
- method: the law used to model each variable. It is a vector of length K with K the number of variables.
- influence: it is a vector of length K giving approximately the influence of each variable on the classification. It provides the approximate number of observations which would change classes if the variable was dropped.
- info_vars: internal, generic information on each variable.

Visualizing the results of the classification. You can use `summary(res_em_pat)` or `plot(res_em_pat)` to see the result of the classification.

5.5.3 ALGO-4_base_match_patents.fst

This is the main output data set containing all $id\_se \times id\_inv\_seq$ pairs considered to be true matches. In other words, it contains the list of all patents (`appln_id`) produced by Swedish persons (`id_se`).

Sources. The sources are ALGO-3_base-all-matches.fst for the matching without the patent information, and ALGO-4_em-result-patent_match.RData and ALGO-4_bilateral_patents.fst for the matching with the patent information. The patent identifier (`appln_id`) comes from OECD_inventors.fst. (Link to Tables of Step 4: Adding patent information.)

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- appln_id: application identifier of the patent

- type_match: how the $id\_se \times id\_inv\_seq$ was matched. There are three ways: i) `EM` (via the first EM algorithm), ii) `address` (the inventor shared the same address with the Swedish person), iii) `EM-patent` (last step of the algorithm when patent information is added for the matching).

- prob: the probability to be a true match. Only present for Bayesian matching (EM algorithms), hence it is missing for `type_match="address"`.

5.6 Step 5: Descriptive data sets to understand the matching

5.6.1 ALGO-5_EM-sample-info.fst

This is a data set for exploration only. It aggregates raw information on all $id\_se \times id\_inv\_seq$ pairs to which an EM algorithm was applied. It contains the probability of being a true match as well as all relevant information on the inventor identity and the Swedish identity. Note that it also contains information on the pairs that were not identified as true matches.

Sources. The information on the EM classification comes from ALGO-3_em-result-n_match-LE-5.RData and ALGO-4_em-result-patent_match.RData. The bilateral variables used in the EM algorithms come from ALGO-2_all.fst and ALGO-4_bilateral_patents.fst. The information on inventors and patents come from PATENT_inv-names.fst, PATENT_inv-address.fst andd PATENT_inv-employer.fst. The information on the Swedish persons come from STAT-SE_indiv.fst and STAT-SE_job.fst.

(Link to Tables of Step 5: Descriptive data sets to understand the matching.)

Treatment. When there is multiple lines for a single Swedish person or inventor, the information is concatenated in a single character string. For instance if a person works at Ericsson and at Volvo, the information will be reported as "`ericsson; volvo`".

Variables.

- id_inv_seq: sequential inventor-patent identifier
- id_se: the unique identifier of the Swedish person
- match_qual: categorical variable measuring match quality, 1: low, 2: mid, 3: high.
- address_cat: categorical variable from 0 to 2. They mean, 0: unmatched, 1: missing address and 2: matched.
- address_match: character categorical variable taking the following three values: unmatched, missing and matched.
- age: the age at invention, i.e. the difference between the year of invention (priority year) and the birth date
- same_emp: binary indicator, 1 meaning the employers are identical
- proba_name: continuous variable between 0 and 1, normalized by the largest name probability. The probability to have a given first first-name and family name combination. The -name and family name probabilities are based on the distribution of names in the Swedish population. This is an average between the Swedish person probability and the inventor probability since the names of the two can be different due to fuzzy matching.

- proba_name_rank: the rank of the variable `proba_name` normalized into 0 and 1.
- n_match: n_match: number of `id_se` that are considered as potential matches for a single `id_inv_seq`
- prob_qual: categorical variable combining the information on the name probability and the name matching quality. It is the combination of name probabilities equal to "`low`", "`mid`" or "`high`", to name match qualities equal to "`low`", "`mid`" or "`high`".
- same_coauth: binary variable taking value 1 if the patent of `id_inv_seq` shares at least one co-author in common with the patent pool of `id_se`
- max_tech_sim: integer categorical variable taking values 0, 1, 3, 4, or 7. It represents the largest IPC digit for which there was a matching between the IPCs of the patent of `id_inv_seq` and the patent pool of `id_se`. Illustration: if equal to 1, this means that the IPC codes match at most on IPC1, for example "**C**12M 1/00 Apparatus for enzymology or microbiology" and "**C**01D 3/00 Halides of sodium, potassium, or alkali metals in general" match only at most "C". If there was an IPC of `id_inv_seq` equal to "**C01D** 3/00 Halides of sodium, potassium, or alkali metals in general" and one of the patent pool of `id_se` equal to "**C01D** 5/00 Sulfates or sulfites of sodium, potassium, or alkali metals in general", then we would have `max_tech_sim=4`. A value of 0 means that there was no match, even at the largest level.
- same_app: binary variable taking value 1 if at least one applicant of the patent of `id_inv_seq` matches at least one applicant of the patent pool of `id_se`.
- type_match: how the $id\_se \times id\_inv\_seq$ was matched. Either: i) `EM` (via the first EM algorithm), ii) `EM-patent` (last step of the algorithm when patent information is added for the matching).
- prob: the probability to be a true match as obtained from the EM algorithm
- max_prob: the max matching probability across all `id_se` for a given `id_inv_seq`
- inv_name: raw name of the inventor as displayed in the patent document
- year_prio: priority year of the patent
- inv_address: raw address of the inventor as displayed in the patent document
- inv_emp: aggregation of the applicant names present in the patent document
- se_name: aggregation of the different full names of the Swedish person (unique names across all years)
- se_address: aggregation of the different addresses in which lived the Swedish person (unique addresses across all years)
- se_emp: aggregation of the employer names of the Swedish person (unique employers across all years)

References

MARAUT, S., H. DERNIS, C. WEBB, V. SPIEZIA, AND D. GUELLEC (2008): "The OECD REGPAT Database," *OECD Science, Technology and Industry Working Papers.*