Data matching
with
Levenshtein automata

The French Ministry of the Interior has to treat large databases. One of those is the 50M+ French drivers database.

For Privacy reasons, this file is never crossed with other identity database, except for verbalisation.

— OUR ISSUE —

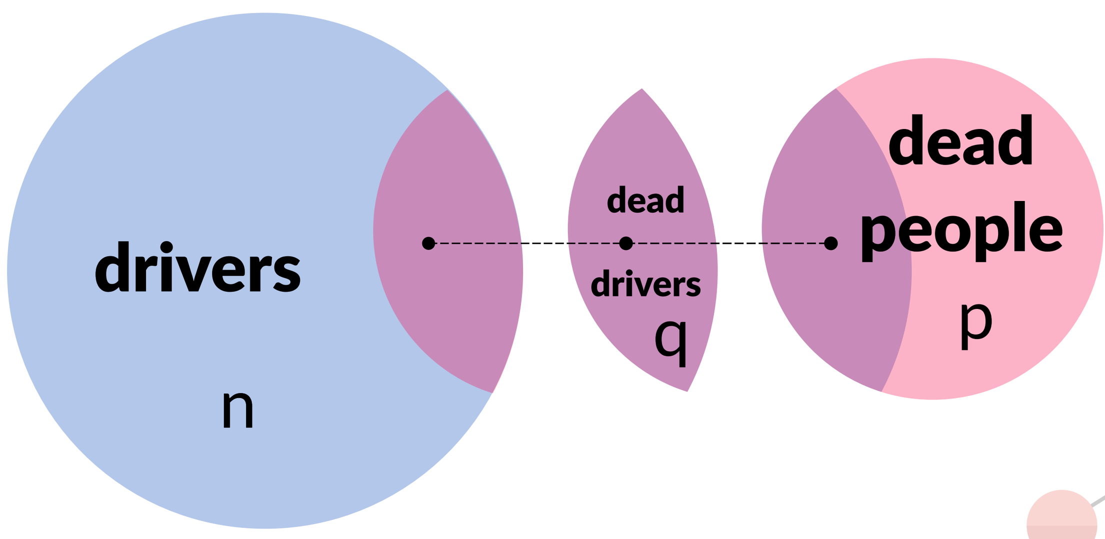
INSEE takes a census of dead people in a database (22M since 1970).

The Ministry of the Interior stores the French drivers in another database.

However, these 2 databases are **not synchronized** :
some people continue to loose points on their driving license whereas they are deceased when someone uses their vehicle and did not change its owners

Consequently, the main stake is to remove the dead drivers from the database containing all the French drivers, without any identity number as the law forbids to store unique identity number as the INSEE's one.

To achieve this goal with **high accuracy and good recall**, data matching with **fuzziness** must be used. **Levenshtein Automata** is one of the algorithm chosen by the **matchID** team.



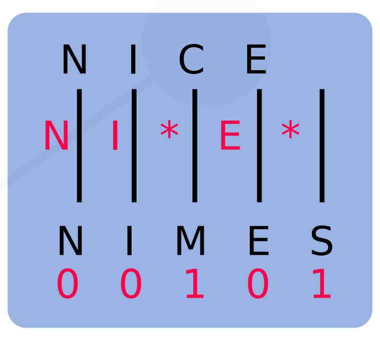
— DATA MATCHING THROUGH LEVENSHTEIN AUTOMATA —

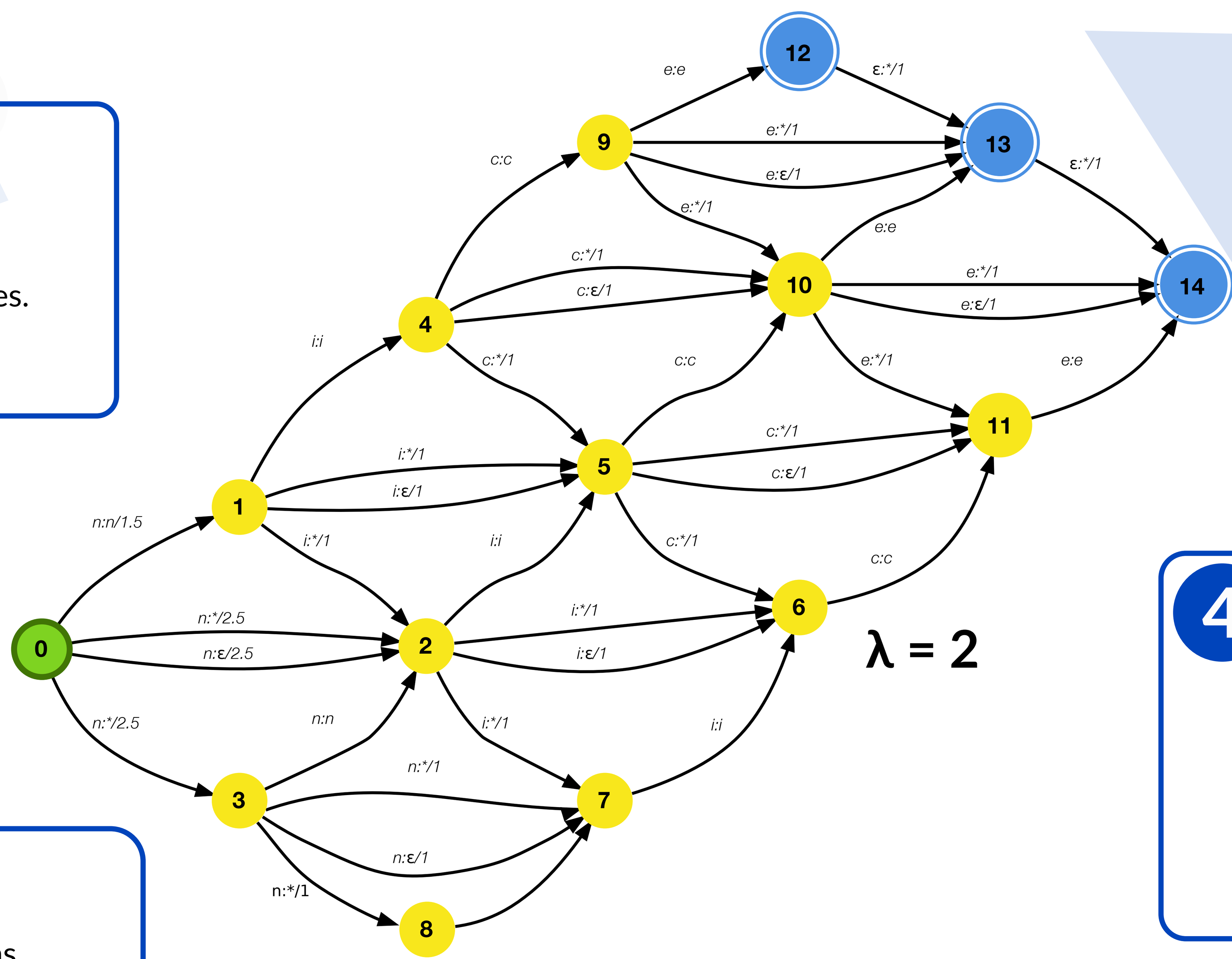
1 DATA PRE-PROCESSING

Standardization of the items contained both

in the Reference and in the Hypothesis databases.

Complexity : **O(n+p)**





5 ASSESSMENT

Machine Learning on the score in order

to maximize the recall and the precision.

Complexity : **O(q)**

4 SCORING

Attribute a score to each match to

determine if the classification between

matches and non-matches was efficient.

Complexity : **O(q)**

2 INDEXING

Sorting and efficient storing of the items

contained in the Reference database

to make the search of the Hypothesis items faster.

(methods : Levenshtein Automata,

BK-Tree, SymSpell, Q-Gram...)

Complexity : **O(n log(n) + p log(p))**

3 DATA MATCHING OR SEARCH STEP

Determines if the items of the Hypothesis database

belong or not to the Reference database.

Complexity : **O(p log(n))**

— LEVENSHTEIN ADVANTAGES —

Maximize the Recall-Precision

Accelerate the Search Process

Can be composed with diverse edit distance

— RESULTS —

Strict Matching : **90%** Recall-Precision vs Fuzzy Matching : **95%** Recall-Precision.

Indexing Process on a database of **10 000 items** :
40s with Levenshtein Automata in Python vs **10s** with in Lucene

Matching Performance :
50 queries per second with Lucene, huge cpu consumption
From **2 to 3 times faster** with Levenshtein Automata in Python

LEVENSHTEIN INDEXING

Limit the typing errors
Ex : Tristam <=> Tristan
Adapted for transliterated varieties
λ = 1

— OTHER OPTIONS —

In this study, we focused on Levenshtein indexing and distance computation,

other options could be proposed such as :

SYMSPELL

Adapted for mispellings correction
Ex : coukd <=> could
Based on deleted varieties

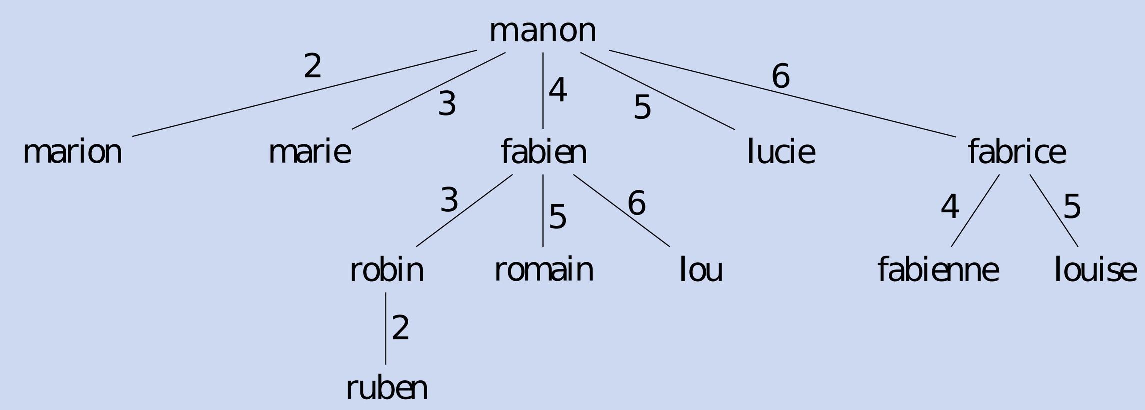
Q-GRAMS INDEXING

Adapted for long strings
such as adresses
No matter the order of the items
Ex : 3 Bd Garibaldi

Drawbacks : noise on short volume

Q-Grams Distance Garibaldi - Vivaldi : 8
Potential Bi-Grams : ga-ar-ri-ib-ba-al-ld-di-vi-iv-va

BK-TREE INDEXING



PHONETICS INDEXING

Adapted for transliterated varieties
Ex : Sherazat <=> Shéhérazade

Drawbacks : noise on short volume

Soundex Distance :
SHERAZAT S683
SHEHERAZADE S683

Tuteur : Fabien ANTOINE / **Stagiaire :** Manon RIVOIRE