Data Integration

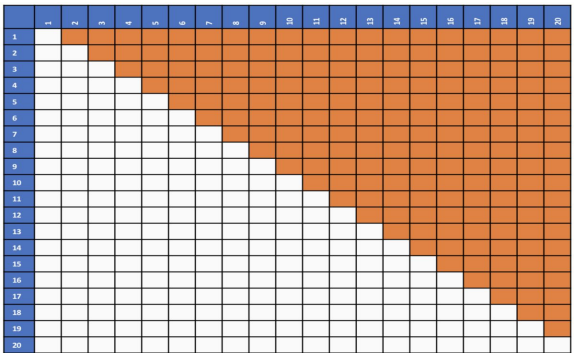
Assignment 3: Data Deduplication

Group BB

Student Name	Student ID	Course of Study
Shady Alkhouri	389561	Computer Science, Master
Karel Smejkal	388638	Computer Science, Master
Ayham Kanhoush	391136	Computer Science, Master

Task 1: Brute-Force Duplicate Detection:

1. Simple brute force algorithm - take one record and compare to all records that are after it. The better visualization of algorithm shows the next picture:



In my case I first compared SSN(only if it wasn't 0), if Levenshtein distance for more than 80% I compared LastNames of these tuples and if this similarity was more than 70% I compared FirstNames of these tuples and if this distance was more than 70% I added this tuple to result. As I did 3 comparison in order to check if the tuple is really duplicate it increased the run time(because of complexity of Levenshtein distance) but I can be quite sure that given tuple is duplicate.

In the other part of task 1 I took the records I skipped before(the ones that has SSN 0) and compared them every other record(LastName and FIrstName) in whole DB to find the missed record, but this time with higher precision for Levenshtein distance of 85%.

Input: whole InputDB.csv database - cleaned DB from last assignment
Output: OutputDB-Task1.csv - found tuples that are marked as duplicate by my algorithm
Similarity function: Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. [1], Time complexity: O(n*m) where n and m are the lengths of the two strings [2]
Time complexity: In worst case:

$$\mathbf{O(n^2)}*O(m*o)*O(p*q)*O(r*s)$$

where
- n is number of records in dataset
- m is length of the first SSN
- o length of the second SSN
- p length of the first LastName

q length of the second LastName
r length of the first FirstName
s length of the second FirstName

note: exactly **O($(n^2-n)/2$)** in the first part
and **O(n^2)** in the secondpart(without the complexity of comparing 2 SSNs)

- I also didn't count complexity of cleaning DB from last assignment

2.

Part1:
Duplicate Detection Results:
Precision = 0.999920279818
Recall = 0.405877237334
F1 = 0.577387391535
Total time: *719.9233908851942 mins*

Part1+Part2:
Duplicate Detection Results:
Precision = 0.893884155733
Recall = 0.185458590353
F1 = 0.307184156399
Total time of both parts together: *1620.698122549057 mins*

3. Upsides: relatively high precision(but depends if you compare right elements)
Downsides: Very slow even with good choice with elements, Also in the second part I checked the same values with themself. This had huge impact on precision and recall.

Task 2: Partition-Based Duplicate Detection:

1. For this task I used module from python: recordlinkage
https://media.readthedocs.org/pdf/recordlinkage/latest/recordlinkage.pdf
which takes pandas dataframe as input and based on Sorted Neighborhood algorithm and sorts tuples so that similar tuples are close to each others and compares tuples within a small window. I sorted tuples by LastName and as window size I put 3 tuples. Than I compared LastName, FirstName and SSN elements between this tuples and when all of them had Jaro Distance higher than 75% I marked as a match
Input: InputDB.csv - cleaned DB from last assigment
Output: Tuples marked as duplicate
Similarity function: Jaro distance - Jaro distance between two words is the minimum number of single-character transpositions required to change one word into the other. [3]
Time complexity:
O(n*log(n))*O(m*o)*O(p*q)*O(r*s) *if w < log(n)*
else **O(w*n)***O(m*o)*O(p*q)*O(r*s)

where n is number of records in dataset

w is window size
m is length of the first SSN
o length of the second SSN
p length of the first LastName
q length of the second LastName
r length of the first FirstName
s length of the second FirstName

note: exactly $O(n)+O(n*\log(n))+O(w*n)$ [4]

- I also didn't count complexity of cleaning DB from last assignment

2. Duplicate Detection Results:
   Precision = 0.95679003824
   Recall = 0.280339771463
   F1 = 0.433626768274
   Total time: 24.81 seconds

3. **Upside**: Very fast and relatively high precision
   Downsides: very dependent on the window size and also on what column do I sort on. If I choose wrong, precision and recall can be much worse and also the time complexity can be much higher(for example SSN where are a lot of elements with same value) and also on similarity measure for time complexity

References:

[1] https://en.wikipedia.org/wiki/Levenshtein_distance
[2] http://www.levenshtein.net
[3] https://en.wikipedia.org/wiki/Jaro–Winkler_distance
[4] https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/folien/SS13/DPDC/DPDC_14_SNM.pdf