The 800 Pound Gorilla in the Corner: Data Integration
Assignment 3: Duplicate Detection

June 2018

GROUP L

Henrik Dichmann	358361
Fiona Wille	376585
Ariane Ziehn	358027

1 General information

Dataset S(RecID, FirstName, MiddleName, LastName, Address,City, State, ZIP, POBox, POCityStateZip, SSN, DOB)

2 Task

The main task is to detect all the duplicate tuples that refer to the same real world entities. Table 1 shows two possible duplicate tuples. As you can see, while most of the values in the tuples are the same, some values are different because of representation problems (such as "ARKANSAS" and "Ar"), typos (such as "AGAUS" and "AUGAS"), or missing values (such as "1979" and "").

2.1 Task 1: Brute-Force Duplicate Detection

Here, we want to compare all the possible pairs of tuples with each other.

1. Provide an algorithm. Specify the input, output, similarity function, and time complexity.
2. Implement the algorithm and report the precision, recall, F1, and runtime.
3. What is the upsides and downsides of this method?

2.2 Task 2: Partition-Based Duplicate Detection

Here, we want to reduce the time complexity of brute-force approach by dividing dataset into partitions. Thus, instead of comparing all the possible pairs of tuples in the whole dataset, we need to compare tuples just inside of their partition.

1. Provide an algorithm. Specify the input, output, similarity function, and time complexity.
2. Implement the algorithm and report the precision, recall, F1, and runtime.
3. What is the upsides and downsides of this method?

3 Solution

In general, the following input, output and similarity function have been used for this assignment.

- Input: cleaned data file from Assignment 2
- Output: List of identified duplicate tuple ID's
- Similarity function: Monge Elkan

3.1 Brute-Force

3.1.1 Algorithm

For the brute-force approach, all rows have been collected in a HashMap at first. Afterwards, for each tuple a complete iteration above the entire HashMap is processed and the tuple columns *firstname, lastname and Zip* are tested for similarity with a minimal threshold of 80%.

3.1.2 Metrics

- Time complexity: O(n^2-n)
- Precision: 0.9512
- Recall: 0.0980
- F1: 0.1777
- Runtime: 16.89 hours
  Our implemented brute-force approach takes way too long compared to the partition-based duplicate detection. This long computation time can be explained by the intense computation of comparing each entry with each other entry. Although we filtered the diagonal out, meaning we do not compare tuples with themselves, we still compare the tuples in both directions (S1 with S2 and S2 with S1). This could

be optimised, but for the blind brute-force approach we tried to keep it as simple as possible. Further, we only consider 3 of the 12 given columns to find duplicates, which explains the low recall. The high precision result leeds to the conclusion that Monge Elkan as similarity function is a good choice for the selected columns.

Another approach could focus the similarity for surname and first name and social security number.

3.1.3 Upsides & Downsides

Upsides	Downsides
Easy to implement	Computation intense
Ok as test	No use in real life, too inefficient
not much previous knowledge about data required	

Table 1: Brute-Force Ups & Downs

3.2 Partition-Based

3.2.1 Algorithm

For the this approach, all rows have been collected in a HashMap at first. Additional, each row got a key, a combination of the first letter of the last name and the first number of the ZIP. All tuples with the same key are afterwards compared among the columns *firstname, lastname and Zip* for similarity with a minimal threshold of 80%.

3.2.2 Metrics

- Time complexity: $O((k_i^2 - k_i) * i)$
  with k = number of tuples per key i and i = number of partitions/different keys
- Precision: 0.9712
- Recall: 0.0928
- F1: 0.0177
- Runtime: 4.24 hours
  In general, the usage of keys to partition the data could speed up the overall run time of the application with 75%. As no further optimisation have been considered, recall and F1 are equivalent to the brute-force approach.

3.2.3 Upsides & Downsides

Upsides	Downsides
Easy to implement	Still computation intense
much faster than the brute-force approach	properties of columns need to be known

Table 2: Partion-Based Ups & Downs