# **CREATING MASTER-DATA FROM DATASET USING SIMILARITY-SCORES OF TEXT-FIELDS**

|  |  |  |
| --- | --- | --- |
| Rohan Gurasale  Deloitte-USI, A&C  Mumbai, India  @gmail.com | Vikrant Deshpande  Deloitte-USI, A&C  Mumbai, India  vikrant.deshpande09876@gmail.com | Roopal Gupta  Deloitte-USI, A&C  Mumbai, India  @gmail.com |
|  |  |  |

## **INTRODUCTION**

Most conglomerates today, have many separate applications and systems (ie. ERP, CRM) where data that crosses organizational departments or divisions, can easily become fragmented, or duplicated. Reporting critical KPI (Key-Performance-Indicators) for a business accurately becomes difficult. Questions like “How many entities do we govern?”, “Which of them are most profitable?”, require a cleaned, accurate, master dataset. The data-capturing phase itself might lack a standardized approach, resulting in fundamental discrepancies rendering the data unusable for reporting. An incorrect address in the customer-master might mean orders, bills and marketing literature are all sent to the wrong address; an incorrect account number in an account master could lead to huge fines!

**MOTIVATION FOR THE USE-CASE**

This paper focuses on *masterizing* clinical-data in terms of hospitals/sites, that the pharmaceutical-client manages. For example- the same site “Kadlec Regional Medical Center”, might be reported differently as “Kadlec Clinic Hematology and Oncology” across the client’s source systems. Our goal is to identify a *golden entity* (Master Record) to which other duplicate records can be matched, and maintain their *source-to-master* *linkage* (Cross-Reference). Although industry-standard tools are available (Informatica, Oracle, SAP, etc.) that can be used with third party collaborators like *Address-Doctor-Service*, or *Dun&Bradstreet* to retrieve the standardized asset-data, this use-case was intended to prove that open-source code and libraries could be leveraged to produce near-standardized results.

## **IMPLEMENTATION CHOICES**

The intuition behind identifying unique entities within a dataset is as follows:

* Within a dataset of *n* records, we’d have to compare the 1st record with the remaining (*n - 1*) records, the 2nd record with the remaining (*n - 2*) records, and so on. Thus, there would be unique combinations to be considered.
* Between 2 different datasets of *m* and *n* records each, there would similarly be unique combinations to be considered. These *n* records can be thought of as the previously identified master records.

At each individual-combination level ie. for the participating records, a string-comparison algorithm [a] will be used to compute a match-score of the relevant feature-strings. Let *str1*=“Kadlec Regional Medical Center” and *str2*="Kadlec Clinic Hematology and Oncology".

1. **Edit distance based** **algorithms** (ex- Levenshtein) compute the number of character-level operations needed to transform one string to another. More the number of operations, less is the similarity between the two strings. For example- the Levenshtein distance will be 25, and the normalized-similarity will be:

*0.325*

Jaro-Winkler is a similar directional-algorithm which checks for characters of str1 occurring in a window of some size within str2.

1. **Token-based** **algorithms** (ex- Jaccard-index) will find the similar tokens in both string-sets. More the number of common tokens (words or n-gram characters), more is the similarity between the sets.

=

Using words as tokens, the score *0.125*, while individual character-tokens gives *0.558*.

1. **Sequence-based algorithms** (ex- Ratcliff-Obershelp similarity) try to find the longest sequences present in both strings. First, remove the longest common substring from both strings, and split the originals into the left and right parts of the common substring. Repeat this recursively for both the left and right parts, until the size of any broken part is less than a default value. The score is twice the number of characters found in common divided by the total number of characters in the two strings.

*0.45*

1. **Cosine-similarity** can be summarized as a widely used NLP technique that uses a matrix of word-embeddings: where each cell in a column, represents the weight by which the word associates to that row/attribute. Two words *x* and *y*, are first converted to their word-vector from this word-embedding matrix, and the cosine formula is applied to identify sematic similarity:

Subadhra Parthasarathy et al. [b] identified that Levenshtein produces results on-par with cosine-similarity.

## The RecordLinkage library [c] in R, provides two main functions to generate candidates for deduplication within a single dataset, or candidates for identifying duplicates between two different datasets. The Python equivalent library [d] is limited by the array-size that Pandas can hold when the number of candidate-pairs is ginormous.

## Python for easy data-wrangling.

## R for only computations since in-memory statistical computations are much faster.

## **ARCHITECTURE**

## Sort the dataset by relevant features so that minibatches contain most of the duplicates already.

## Describe the recursive approach diagram end-to-end from sourcing-data to creating list of csv-files.

## **FUTURE SCOPE**

## Lemmatize each word before match-scores using NLTK (Natural Language Processing ToolKit) in Python.

## Implement an incremental approach to match: incoming dataset of delta-records vs the already identified master-records.

## Implement a better way to identify master-records, than just choosing the very first occurring record amongst the subset of potential duplicates. Possible to choose the golden master-record based on number of common occurrences within that subset of potential duplicates.

## **REFERENCES**

## \*\* the Medium blog that summarizes these algorithms (<https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227>)

## \*\* the other research paper stating advantages of Levenshtein (<https://amedeloitte.sharepoint.com/:f:/r/sites/AIIEEmergentCapabilities/Shared%20Documents/04.%20NLP%20-%20NUG/Text%20Search?csf=1&web=1&e=DFImLT>)

## \*\* the RecordLinkage R library (<https://CRAN.R-project.org/package=RecordLinkage>)

1. \*\* the RecordLinkage Python library (<https://recordlinkage.readthedocs.io/en/latest/ref-index.html>)