**CREATING MASTER-DATA**

**FROM DATASET USING**

**SIMILARITY-SCORES OF TEXT-FIELDS**

|  |  |  |
| --- | --- | --- |
| Rohan Gursale  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  roopalgupta123@gmail.com |

**Table of Contents**

**Introduction1**

Motivation for the Case Study2

Applications of Master 3

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

## **INTRODUCTION**

Most organizations are shifting towards data-driven decision making, and their strategic business initiatives and strategic decisions are derived from analyzing data captured in the data hub. However, lack of accurate, uniform and semantically consistent master data assets in an organization leads to a significant impact on their business and thereby precluding them from achieving their goals and outcomes they desire.

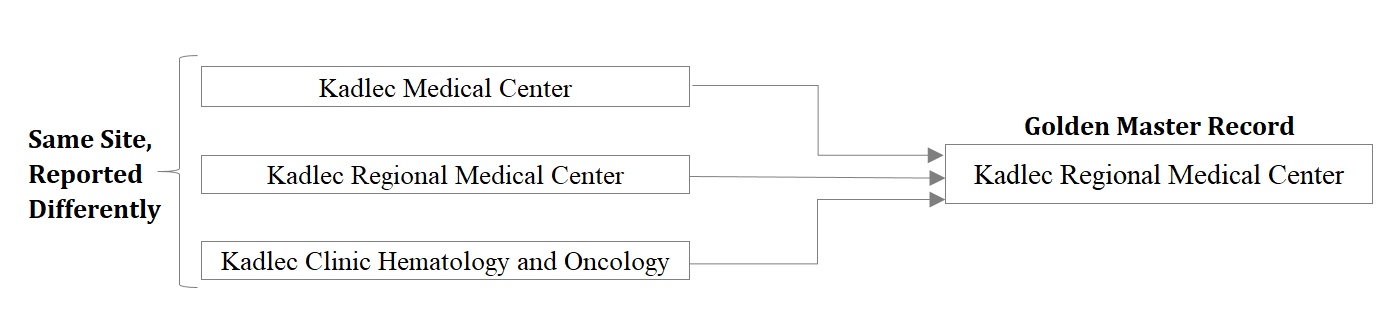
Nowadays, conglomerates have many separate applications and systems (viz. ERP, CRM) where data crosses organizational departments or divisions can become fragmented or duplicated. Reporting critical KPIs (Key-Performance-Indicators) accurately for a business becomes difficult. Questions like “How many entities do we govern?”, “Which of them are most profitable?” require a cleaned and accurate master dataset. The data-capturing phase itself might lack a standardized approach, resulting in fundamental discrepancies rendering the data unusable for reporting.

Master data assets enable organizations to seamlessly leverage the vast amount of dynamic data owned by them. It improves data quality, enhances data governance process, and provides a holistic and comprehensive vision while deriving insights from a master data set.

## **MOTIVATION FOR THE CASE-STUDY**

This paper focuses on generating ‘master’ records from the enterprise-wide collection of data and de-duplicating it. We implemented a solution framework as a proof of concept for a pharmaceutical client to *masterize* their clinical sites/hospital’s address related data.

In our use-case, the same site “Kadlec Regional Medical Center”, might be reported differently as “Kadlec Clinic Hematology and Oncology” but with the same address, across the client’s different 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). In other words, the intention is to be able to identify all records that point to the same golden master entity within and across all data sources.



Although industry-standard tools are available (Informatica MDM, 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 case study was intended to prove that open-source code and libraries could produce near-standardized results.

## **BUSINESS IMPACT** (Section needed or not? Better header for this?)

Generating and maintaining a Master dataset from data collated via various third-party vendors, internal source systems, and data integrated in-cases of acquisitions and mergers at one central data-hub repository is a challenging endeavor.

Nevertheless, once this goal is achieved, it can lead to significant benefits:

* A ‘single version of truth’ is maintained for each associated entity across the organization
* Orchestrates synchronization and seamless collaboration between multiple cross-functional channels of the business
* Provides holistic view of the data-assets owned by the organization
* Increases reliability and trust on the data insights and analytics derived from Masterized data entities

(Some more pointers can be added..accuracy/accountability/reduced time & effort/saves from potential incorrect decisions and loses/know your data better)

# **LITERATURE SURVEY & IMPLEMENTATION CHOICES**

For this use-case of de-duplicating records and generating a master set, we compute the similarity between two textual strings and determine if they are a probabilistic data match. If two or more records are believed to belong to the same entity i.e. the golden master record, they are ‘linked’ together.

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

* Within a dataset of *n* records, we must 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:

n2 unique combinations to be considered.

* Between 2 different datasets of *m* and *n* records each, there would similarly be unique combinations to be considered.

At an individual combination level i.e. for the participating records, a string-comparison algorithm [1] will be used to compute a match-score of the relevant feature-strings. There are several text similarity algorithms available based on different use-cases, which are further discussed below:

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 these character addition/subtraction/replacement operations less is the similarity between the two strings. For example- the Levenshtein distance between *str1* and *str2* will be 25, and the normalized-similarity will be:

*0.325*

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

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

=

For *str1* and *str2,* using words as tokens the score *0.125*, while using 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 [2]: 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-vectors from this word-embedding matrix, and the cosine formula is applied to identify semantic similarity:

Kaitlin Coltin et al. observed that Levenshtein produces results on par with Cosine-similarity, when matching potential duplicate organization names against a master list [4]. Bearing in mind that a high volume of our dataset contained junk characters and spelling errors, and considering the anagram-possibility scenario of Jaccard-measure, coupled with their comparison against cosine-similarity, it was concluded that Levenshtein algorithm fits best in our use-case. In contrast to their approach, our aim is to identify the unique entities in our dataset with no standard set available, deduplicating the input by comparing it against itself was vital for our process. Similarly, applying machine-learning techniques like a clustering or classification algorithm wasn’t possible since there isn’t a target variable/list to train or test on.

## **TECHNICAL CONSIDERATIONS**

The **RecordLinkage** library in R provides two main functions to generate candidates for deduplication within a single dataset (hereafter called the *dedup* function), or candidates for identifying duplicates between two different datasets (hereafter called the *linkage* function) [5]. The Python equivalent library is limited by the array-size that Pandas can hold when the number of candidate-pairs is ginormous [6]. However, Python’s easy-to-use data-wrangling features, ability to invoke a child-subprocess like R-scripts, topped with some deployment-server versioning limitations, led to developing the end-to-end structural pipeline in Python. R is used only for generating match-scores (indirectly by using a pre-compiled C function), since the in-memory statistical computations are much faster, and Pandas cannot generate such massive sized-DataFrames in-memory [7].

Since the deployment server supported only Python 2.5x and R 3.4x, we had to refactor the Python code, and reverse-engineer the RecordLinkage library in R (since it requires R >= 3.5.0). The original core capabilities of the *dedup* and *linkage* functions were maintained, but the cursory code supporting phonetic algorithms, blocking datasets, etc. was removed to speed up the algorithm. The Levenshtein function can be implemented in multiple ways [3], but we picked the source code written in C by Joe Conway, Murat Sariyar, and Andreas Borg [5] since it is already used as part of the RecordLinkage package. The source code in C was pre-compiled into its binaries, and reloaded into R using the following commands:

* ***R CMD SHLIB levenshtein.c***
* ***dyn.load("levenshtein.so")***

Binaries generated in Windows have the .*dll* (Dynamically Linked Libraries) extension and in Unix the .*so* (Shared Object) extension.

Once the *dyn.load*() function loads the binaries and the symbols within, the Levenshtein function can be invoked from within R, to return the number of characters replaced/added/removed to make the strings match.