sentinel partners

White Paper

Empowering MDM With Efficient Data Matching in the Cloud

White Paper

Internal Research Project

For: General Publication

Document Reference: INT-AI-2022-01

Document Date: 20/05/2022

Prepared by:	*Rafi Trad*	Date:	20/05/2022

Document status:

Commercially Insensitive
Public distribution permissible

Table of Contents

Executive Summary

Identifying real-world entities and tracing them across multiple datasets is an essential goal of any Master Data Management (MDM) project.

Following any preliminary data validation and rationalisation of the source datasets, Data Matching is the first major step to achieve that goal, which can be exact or approximate/fuzzy. Approximate matching is flexible and forgiving when data changes slightly across Big Data sources; a very desirable feature but a costly one in terms of performance.

In this paper, we will explain how we achieved a radical speedup factor estimated at 600x in matching compared to a simple approach, and what technologies we used. Such an improvement can help in achieving faster and better data governance, and we highlight where to employ this type of approach as a culmination to the paper.

Problem Statement

Linking datapoints to connect data entities is a fundamental task in any Master Data Management (MDM) pipeline (JOSHI, 2019). At Sentinel Partners, we call this step "matching" as seen in Figure 1, which allows us to perform entity resolution. Entity resolution is "determining when references to real-world entities are equivalent (refer to the same entity) or not equivalent (refer to different entities)" (Talburt, 2011). In other words, it allows us to identify the same real-world entity (person, property, etc.) across different datasets and data feeds.

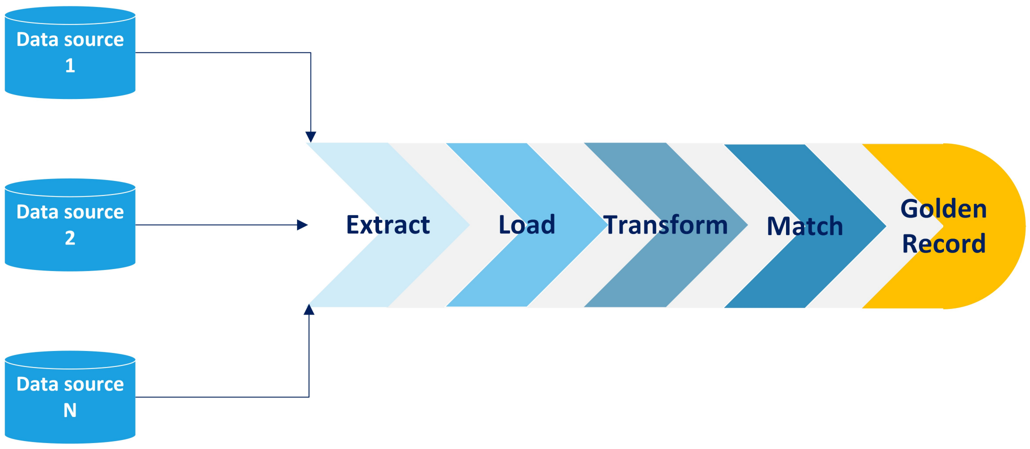


Figure 1 Basic Sentinel Partners MDM pipeline; matching is a fundamental step.

To this end, datapoints must be matched to each other as the first step. In fact, exact and approximate (fuzzy) matching plays a key role in identifying real-world entities in data to obtain a single view of each entity (Talburt, 2011). However, it is neither straightforward nor efficient to match datapoints. There are two approaches to achieve this: handcrafted matching rules and brute force matching.

Crafting customised matching rules requires expert knowledge of the source data content that is not always readily available, and it might fail to find all possible matches. The resultant matching rules are tailored to each domain, and thus they need considerable modification and maintenance across domains or datasets.

On the other hand, brute force matching is useful as it gives us the full picture, but it is very inefficient and needs a generic way of matching any two datapoints.

Let’s concentrate on brute force matching. Assume that we have a small dataset of 1000 entities (properties, people, etc.) with their addresses (address lines 1 through 6, in addition to the postcode) and we need to match duplicate entities in the dataset. Assume that an entity’s address might change slightly throughout the dataset. This forces us to use approximate matching as exact matching cannot detect slight changes in data. Many database management systems and programming languages offer fuzzy matching functions, but the problem with these solutions is they are not optimised for large datasets out-of-the-box.

Additionally, we need to perform 1 million comparisons to cover the 1000 records in our example, and this number increases rapidly with more data. Vertical scaling is therefore not a workable solution.

To conclude, more efficient and flexible matching approaches are needed to handle large datasets in general. Brute force matching with a generic matching mechanism, that has to be fault tolerant, is a feasible domain-agnostic solution. Such a solution enhances entity resolution, a crucial task in master data management projects and pipelines.

Our Solution

The main concept of our solution is to look at the datapoints differently. Each datapoint has many data attributes, some of which can be irrelevant for matching. We splice the relevant data attributes to form what we call data “sentences”, and we generate one sentence per datapoint. Then we transform all data sentences to numbers and process this interim representation of data instead, because computers are extremely fast processing numbers compared to any other form of data.

In the parlance of artificial intelligence, this process is called vectorisation, and we borrow many methods from an AI (Artificial Intelligence) field called Natural Language Processing (NLP) (What is Natural Language Processing?, 2020) to vectorise our data sentences and make them data vectors.

The semantics of these data sentences do not matter, and we want to focus on the literal characteristics only; that’s why Postgres DB text search, AI deep learning (embeddings) and similar methods were not viable options. At the end, we compare the data vectors, which are purely numerical now, to find the best matches.

On the technical side, our solution exploits the limitless power of cloud computing and integrates smoothly into our software (Sentinel Data Platform). That means clients do not have to deal with cloud configuration as the cloud architecture is entirely transparent to them. The solution is written in Python and called from Java, and it can run on both Azure Machine Learning Services and Azure Synapse Analytics (big data processing), but we will use the latter. Next, we will detail the methodology and our technical implementation.

Fuzzy Matching Methodology

Figure 2 shows the workflow of the implemented fuzzy matching solution. A datapoint comprises several attributes, like age and gender for people, tenure types for properties, and so on.

Our solution needs a set of relevant data attributes to consider when matching datapoints. Each attribute will be converted to text to form a **data "word"**. The concatenation of all data words for a datapoint forms its **data "sentence"**, so each datapoint will be converted to a data sentence consequently. This is a simple procedure, but care must be taken to handle specific edge cases (missing data) and ensure consistent sentence generation.

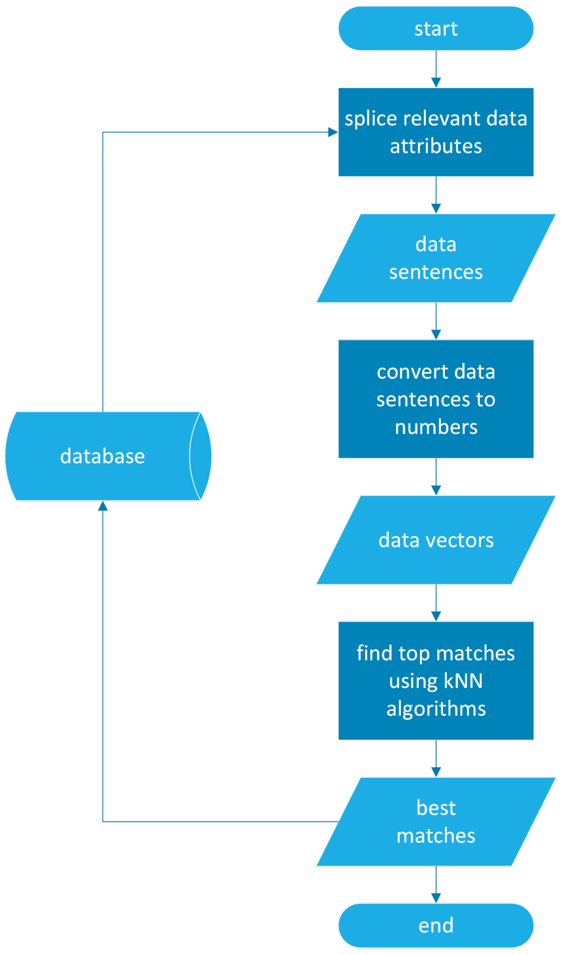

start
splice relevant data attributes
data sentences
convert data sentences to numbers
data vectors
find top matches using kNN algorithms
best matches
end
database

Figure 2 Our fuzzing matching solution flowchart

The data sentences are processed further using established methods from NLP to convert them to numbers. There are many approaches to represent texts as numbers in NLP, but since we are interested in the literal characteristics of the data sentences, we opt for the n-gram approach (n-gram, 2022). After this stage, which is called vectorisation, each data sentence will be a 1-dimensional array of simple numbers, or a **data vector**. This does not change the original datapoints but represents them in a more convenient way for computers.

You can think of a data vector as an arrow in space, each arrow will have a specific magnitude (length) and direction, and the space in which these data vectors live is called a Vector Space. By modelling our datapoints as vectors in space, we open the door to powerful algebraic computations that run extremely well on computers. This modelling approach is called **Vector Space Model (VSM)** (Tam, 2021).

The next step is to assess how similar any two data vectors are, which tells us how similar two datapoints are from a geometric perspective. Again, there are a lot of algebraic calculations that can find such similarity scores almost instantly, like dot product and cosine similarity. We adapt a mechanism to produce a similarity score that satisfies the following:

1- It is scaled to fall into [0, 100] and convey a confidence level. A score of 100 means a 100% match, or an exact match essentially
2- It takes into consideration the length of data sentences, which corresponds to the magnitude of the associated data vectors
3- It considers the orientation of data vectors

Finally, we are not interested in all matching scores for all pairs of datapoints. We are specifically interested in the best K matches, say the top 1 or 2 matches for each datapoint. We navigate the VSM and pick the top K matches in a K-Nearest Neighbours (KNN) fashion (Nelson, 2020), which is a machine learning algorithm. KNN can help identify the closest K neighbours to a data vector (which represents a datapoint) and these neighbours are the matches we are after. We can specify the number of top matches as a parameter too.

As an output, we end up with the best match(es) for each datapoint, with a confidence score to help identify underlying issues in data and solve them. A confidence score of 100 means there is a definite match, so our solution can be seen as both exact and approximate matching solution.

Technical Details

We use Python and Java to implement the solution and integrate it into our Sentinel Data Platform. The solution runs in Microsoft Azure and utilises Azure Synapse Analytics (previously known as Azure Data Factory). In addition to that, we utilise an Azure Data Lake Gen 2 to read and write data files where needed (see Figure 3), but we can talk directly to databases.

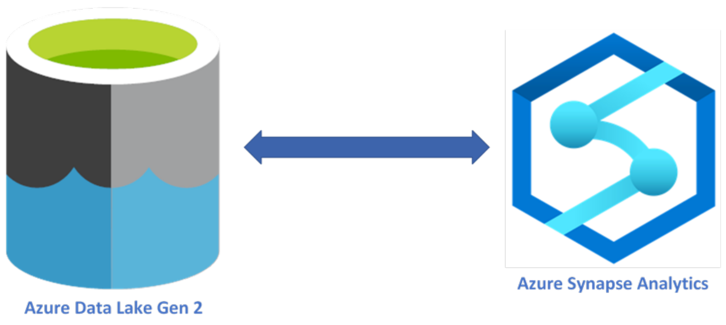
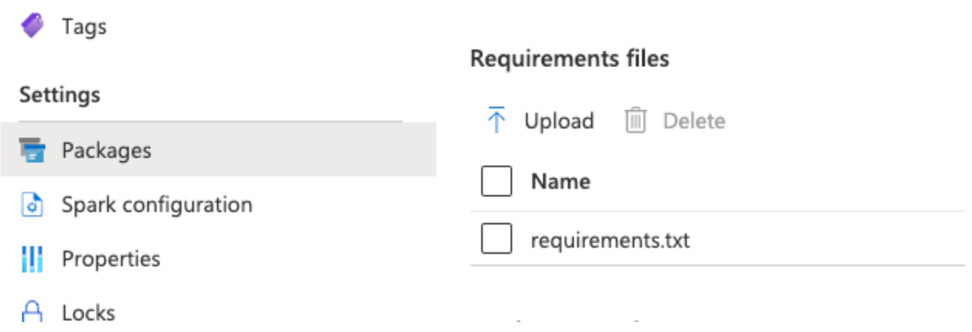


Figure 3 Azure components used in our matching solution

To run the code, we execute an Azure pipeline from within our Hub, and consume the results that are stored in the data lake. We need to create an Azure Synapse workspace, configure its big data Apache Spark pools, and link the workspace to a Gen2 data lake.

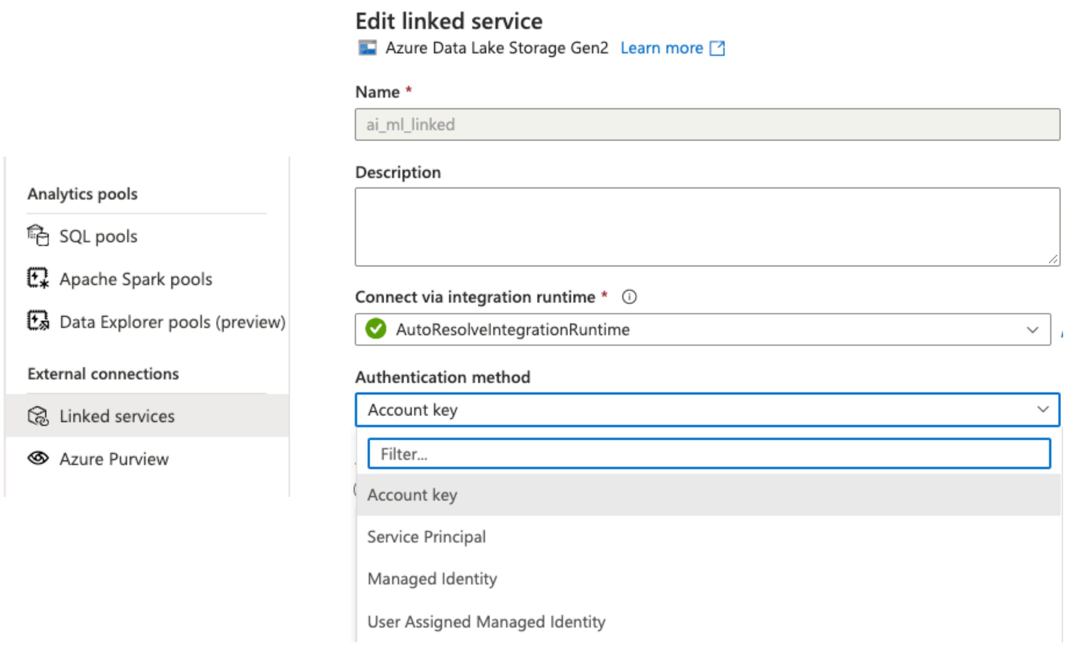
To configure the Apache spark pools, we expose all necessary software packages with their versions to the spark pool of executors. It is simple to do so using the Packages option in Azure Synapse, where we upload a requirements text file that specifies all required packages and their versions:



Based on the uploaded requirements.txt file, Spark configures all its distributed executors, after which we can run the solution without the need to re-configure any Apache Spark Pools.

Another setting we configure is the scaling of Apache Spark executors. Our code can run on any executor size, aka. node size, but we opt for the large option (16 Cores, 128GB RAM). The smallest size provides us with 4 cores and 32GB RAM and can run the solution, but it takes more time of course.

The last step is linking a data lake to Azure Synapse workspace, so that we can read from it and write to it in Synapse. It is possible to set up as many linked services as needed in Synapse using various authentication methods, as shown in the following screenshots (account keys, managed identities, etc.):



Edit linked service
Azure Data Lake Storage Gen2

Name *
ai_ml_linked

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

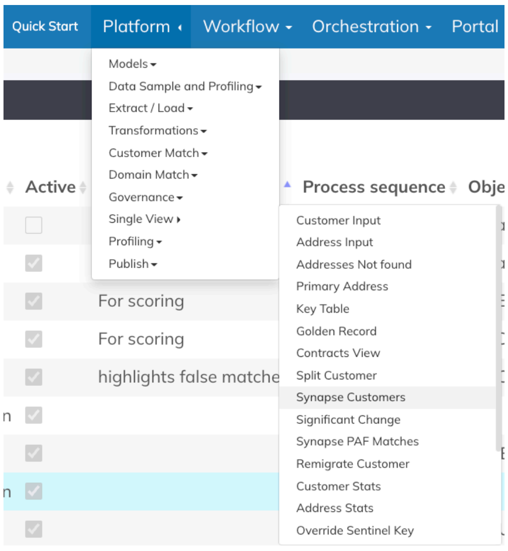
Authentication method
Account key

Account key
Service Principal
Managed Identity
User Assigned Managed Identity

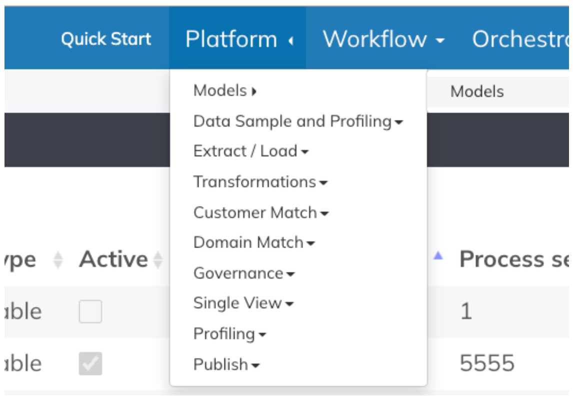
We link our Synapse workspace to a data lake using an encrypted storage *account key*, which can be rotated every while to provide maximum security. This option does not rely on timed authentication tokens, and it can thus accommodate any workload. Using other options might lead to a failed execution, as an authentication token might expire before the code finishes execution, and that wastes our result set instead of saving it back to the data lake. That being said, our solution can integrate directly into databases and read/write from/to them without data lakes. Moreover, the result set can be imported and consumed by the Sentinel Data Platform with minimum effort.

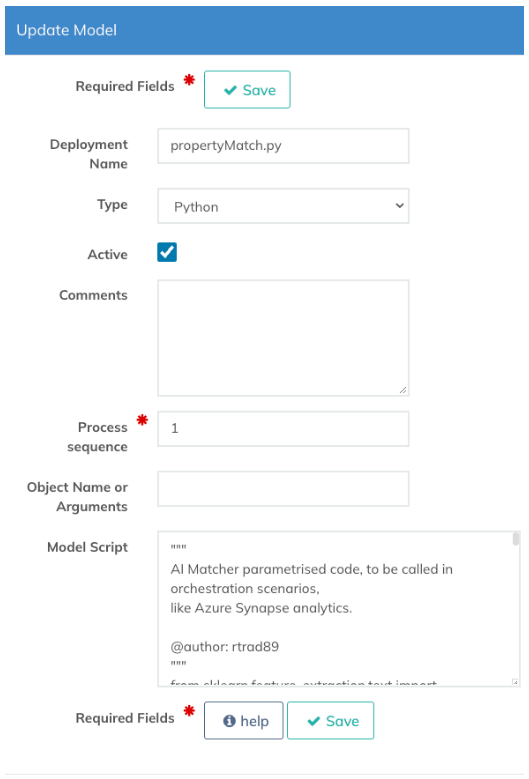
How Sentinel's Data Platform Integrates Everything

Sentinel's Data Platform can be configured to seamlessly integrate with Azure resources. It can access Azure Data Lake Gen2 storage accounts from its user interface as we see in the next image:



Besides, pure Python scripts and many other kinds of scripts can be maintained and run locally as well from the Data Platform, so the solution can run on-premises if desired:




Update Model

Required Fields * Save

Deployment Name: propertyMatch.py

Type: Python

Active: ✓

Comments:

Process sequence *: 1

Object Name or Arguments:

Model Script:

```
"""
AI Matcher parametrised code, to be called in orchestration scenarios,
like Azure Synapse analytics.

@author: rtrad89
"""
```

Required Fields * help Save

Results and Conclusion

We put our solution to test against a very simple database approach as a naïve baseline. Here are the details of our experiment:

Workload	Dataset 1: 31,390,938 datapoints Dataset 2: 38,787 datapoints
Task	Find the exact match or the best approximate match from dataset 1 for each point in dataset 2
Configuration	Baseline: Azure Database for Postgres with 2 vCores Our solution: Azure Synapse Analytics with one large node, 16 vCores

The baseline approach simply runs SIMILARITY Postgres function on each pair of data sentences to produce a similarity score. SIMILARITY function relies on n-grams under the bonnet. We run the function once for each pair of datapoints, so it has to run 1.2T times to finish the job. Afterwards, we need to select the best matches, which is an additional overhead, but we will neglect this aspect.

Our solution finished the task in 1 hour and 39 minutes (**5940** seconds) – including Azure Synapse resource allocation/dispatch overhead. The baseline was too inefficient to run the task with full data, so we reduced dataset 1 to **1000** datapoints only, and they took 15 minutes and 20 seconds, or **920** seconds. If time increases linearly with data (which is a generous assumption in favour of the baseline method), we estimate that the baseline solution would take:

$$2\ vCores\ baseline\ time = 920 \times \frac{31390938}{1000}\ seconds = 28{,}879{,}663\ seconds$$

We must adjust for having 1/8 the vCores available to the baseline solution compared to our solution. Assuming that adding cores increases performance linearly (which is again a generous assumption in favour of the baseline method, because having two cores does not double the performance but yields less than that improvement due to coordination overheads), our final estimation would be:

$$16\ vCores\ baseline\ time = \frac{28{,}879{,}663}{8} = 3{,}609{,}957\ seconds = 42\ days$$

In comparison, our solution is estimated to be ~**600 times faste**r than the naïve baseline. This is a drastic improvement over an out-of-the-box method. In addition to that, using KNN algorithms and VSM, we can locate top K approximate matches from dataset 1 with no further manual processing, i.e., top 1, 2 or more matches at will. Furthermore, we observed that the confidence scores produced by our solution were more intuitive than the similarity scores produced by Postgres. This shows that our solution which relies on cloud computing with advanced data processing can be much better than conventional solutions, both in terms of efficiency and effectiveness.

Possible Use Cases

There are many ways to employ the presented solution in real-world scenarios. Knowing the best possible match for a datapoint locally or from another dataset, in an efficient and definitive manner, can empower both data transformation and tailored data matching procedures.

Moreover, the computed confidence scores can guide targeted exploratory data analysis, which aims to discover new validation and matching rules for adoption in a final configuration. Cleaning data becomes easier as well by focussing on low-quality matches and investigating why a better match couldn't be found. The presented solution generates valuable insights from datasets with little to no data cleaning beforehand, and it enriches a wide spectrum of data management processes.

Any match score can be explained, and while it is harder to understand than plain conditions or even regular expressions, it does pay off as we have hoped to show in this paper.

Bibliography

JOSHI, D. (2019, August 19). Retrieved from https://www.informatica.com/blogs/data-processing-pipeline-patterns.html

Nelson, D. (2020, August 23). *What is a KNN (K-Nearest Neighbors)?* Retrieved from https://www.unite.ai/what-is-k-nearest-neighbors/

n-gram. (2022, February 20). (Wikipedia) Retrieved from https://en.wikipedia.org/wiki/N-gram

Talburt, J. R. (2011). *Entity Resolution and Information Quality* . Elsevier.

Tam, A. (2021, October 23). *A Gentle Introduction to Vector Space Models* . (Machine Learning Mastery) Retrieved from https://machinelearningmastery.com/a-gentle-introduction-to-vector-space-models/

What is Natural Language Processing? (2020, July 2). (IBM) Retrieved from https://www.ibm.com/cloud/learn/natural-language-processing

sentinel partners

About Sentinel

Sentinel is a Data Solutions & Consultancy company providing businesses with the technology needed to gain insights and take control of their data. Specialising in Master Data Management, with 10+ years' experience in the industry we pride ourselves on our ability to empower business with high value systems and our commitment to Data Protection.

Offices

Head Office

19 Highfield Road
Edgbaston
Birmingham
B15 3BH

Consultancy Centre

Kingsway House
40-41 Foregate Street
Worcester
WR1 1EE

Call us: +44(0)800 612 2116

Email us: info@sentinelpartners.co.uk