NATIONAL AND KAPODESTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSC THESIS

WinnER: A Winner-Take-All Hashing-Based Unsupervised Model for Entity Resolution Problems

Konstantinos A. Nikoletos

Supervisors: **Alexios Delis,** Professor NKUA
Vassilios Verykios, Professor Hellenic Open University

ATHENS

MARCH 2022

ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ

WinnER: A Winner-Take-All Hashing-Based Unsupervised Model for Entity Resolution Problems

Κωνσταντίνος Α. Νικολέτος

Επιβλέποντες: **Αλέξιος Δελής**, Καθηγητής ΕΚΠΑ
Βασίλιος Βερύκιος, Καθηγητής Ελληνικό Ανοιχτό Πανεπιστήμιο

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2022

BSC THESIS

WinnER: A Winner-Take-All Hashing-Based Unsupervised Model for Entity Resolution Problems

Konstantinos A. Nikoletos
S.N.: 1115201700104

SUPERVISORS: **Alexios Delis,** Professor NKUA
Vassilios Verykios, Professor Hellenic Open University

ΠΤΥΧΙΑΚΗ

WinnER: A Winner-Take-All Hashing-Based Unsupervised Model for Entity Resolution Problems

Κωνσταντίνος Α. Νικολέτος
Α.Μ.: 1115201700104

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αλέξιος Δελής**, Καθηγητής ΕΚΠΑ
Βασίλιος Βερύκιος, Καθηγητής Ελληνικό Ανοιχτό Πανεπιστήμιο

ABSTRACT

In this study, we propose an end-to-end unsupervised learning model that can be used for Entity Resolution problems on string data sets. An innovative prototype selection algorithm is utilized in order to create a rich euclidean, and at the same time, dissimilarity space. Part of this work, is a fine presentation of the theoretical benefits of a euclidean and dissimilarity space. Following we present an embedding scheme based on rank-ordered vectors, that circumvents the Curse of Dimensionality problem. The core of our framework is a locality hashing algorithm named Winner-Take-All, which accelerates our models run time while also maintaining great scores in the similarity checking phase. For the similarity checking phase, we adopt Kendall Tau rank correlation coefficient, a metric for comparing rankings. Finally, we use two state-of-the-art frameworks in order to make a consistent evaluation of our methodology among a famous Entity Resolution data set.

SUBJECT AREA: Pattern Recognition and Machine Learning

KEY WORDS: unsupervised-learning, clustering, entity-resolution, wta-hashing, prototype-selection

ΠΕΡΙΛΗΨΗ

Σε αυτή τη μελέτη, προτείνουμε μια ολοκληρωμένη ιδέα για ένα μοντέλο μη επιβλεπόμενης μηχανικής μάθησης, το οποίο μπορεί να χρησιμοποιηθεί σε προβλήματα ανεύρεσης όμοιων οντοτήτων σε ένα σύνολο συμβολοσειρών, οι οποίες περιγράφουν το ίδιο φυσικό αντικείμενο, ενώ διαφέρουν σαν συμβολοσειρές. Στην μεθοδολογία αυτή, χρησιμοποιείται ένας καινοτόμος αλγόριθμος επιλογής πρωτοτύπων προκειμένου να δημιουργηθεί ένας ευκλείδειος και ταυτόχρονα ανομοιόμορφος χώρος. Μέρος αυτής της μελέτης, είναι μια πλήρης παρουσίαση των θεωρητικών πλεονεκτημάτων ενός ευκλείδειου και ταυτόχρονα ανομοιογενούς χώρου. Στη συνέχεια, παρουσιάζουμε μια μέθοδο διανυσματοποίησης του αρχικού συνόλου δεδομένων, η οποία βασίζεται στη μετατροπή των διανυσμάτων σε βαθμωτά διανύσματα, μια τεχνική η οποία αντιμετωπίζει το γνωστό πρόβλημα της Μηχανικής Μάθησης, το πρόβλημα των μεγάλων διαστάσεων. Το κεντρικό και πιο καθοριστικό κομμάτι αυτής της μεθοδολογίας, είναι η χρήση ενός αλγορίθμου κατακερματισμού, ο οποίος ονομάζεται Winner-Take-All. Με αυτόν τον αλγόριθμο μειώνεται καθοριστικά ο χρόνος εκτέλεσης της μεθοδολογίας μας ενώ ταυτόχρονα παρέχει εξαιρετικά αποτελέσματα κατά την φάση ελέγχου ομοιότητας μεταξύ των οντοτήτων. Για τη φάση ελέγχου ομοιότητας, υιοθετούμε τον συντελεστή συσχέτισης κατάταξης Kendall Tau, μια ευρέως αποδεκτή μέτρηση για τη σύγκριση των βαθμωτών διανυσμάτων. Τέλος χρησιμοποιούμε δύο σύγχρονα μοντέλα προκειμένου να κάνουμε μια ολοκληρωμένη αξιολόγηση της μεθοδολογίας μας, σε ένα διάσημο σύνολο δεδομένων, στοχευμένο για ανεύρεση όμοιων οντοτήτων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αναγνώριση προτύπων και Μηχανική Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: μη-επιβλεπόμενη-μάθηση, κατηγοριοποίηση, εύρεση-όμοιων-οντοτήτων, κατακερματισμός, επιλογή-προτύπων

ACKNOWLEDGEMENTS

I would like to express my heartfelt appreciation to my supervisors, Pr. Alex Delis for his excellent mentoring and Pr. Vassilios Verykios for providing me with the unique chance to work on his proposal while also engaging in several fruitful discussions. Both have reinforced my desire to become a member of the research community. I also want to express my gratitude to Dr. Dimitris Karapiperis and Dr. George Papadakis for their assistance when needed.

CONTENTS

FIGURES LIST

TABLES LIST

ALGORITHMS LIST

1. INTRODUCTION

Entity resolution, often known as de-duplication, is a popular study topic nowadays. Entity resolution (ER), is the process of identifying when two references to the same real-world entities are equivalent. This procedure may appear to be simple, but when it comes to Big Data problems, it becomes really challenging. Many recent research efforts in generating models for ER problems, are based on Deep Learning, which employs Neural Networks with multiple layers. In order to train and finally be ready to function, these approaches need a significant amount of time and computational resources. Consider as an example DeepER [23], which is one of the deep learning frameworks designed for ER.

In this study, we are attempting to develop an unsupervised Machine Learning (ML) model that is both very efficient, much simple, and it relies on principal ideas of Pattern Recognition. Most research is done for Supervised Learning, as it is a lot safer technique to get solid findings. However, most data sets are unlabeled. Meta's AI Chief Yann LeCun famously said at NIPS 2016 that "if intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake". We took up the challenge and created an unsupervised framework since we believe unsupervised learning has a great promise. Our goal is to develop a robust model that can be used to a variety of ER problems.

This work builds on the paper *Entity Resolution in Dissimilarity Spaces* [29] by Vassilios Verykios and Dimitris Karapiperis, architects of this model corpus. They created an efficient and really robust model, that makes unsupervised learning into string ER data sets, with great space and time complexity. Our work, focuses mainly on experiment and finally improve certain parts of the initial idea. Within that approach, we used the theoretical proofs from the paper [16] to convert the initial Dissimilarity space into a Euclidean-Dissimilarity space. In addition, we performed some optimizations to the prototype selection algorithm (Algorithm 1), where we discovered and resolved a vulnerability. In comparison to similar ER clustering frameworks, the key distinctive feature of this model, is the use of a hashing schema in combination with the rank-ordered transformation of the initial data. Moreover, in this study, we offer a detailed presentation of the framework described in the initial paper, as well as our modifications and the results we managed to achieve. This framework is an open-source project, as we also developed an end-to-end model, using Python.

WinnER, the name of this unsupervised model, consists of four logical parts. Figure 1 depicts the components and workflow of this model. The first part of WinnER (Section 2) introduces a Euclidean-Dissimilarity space and a String Clustering and Prototype Selection algorithm. The inspiration for this algorithm was the "Vantage Objects Approach" from the paper *Efficient image retrieval through vantage objects* [15]. The next part (Section 3), is consequently a phase of embeddings construction, based on the "Vantage Objects" selected in the previous phase. The third part (Section 4), is one of the most significant ideas of this model, and it's about a hashing algorithm named Winner-Takes-All. It was firstly, developed and published by Jay Yagnik, in the paper *The Power of Comparative Reasoning* [17]. This addition, made a huge impact on the time complexity of this methodology. The fourth part of the model (Section 5), is a parallelized algorithm for similarity checking between the objects that fell in the buckets, formed from the hashing step. We will provide a detailed analysis (Section 6) on this model's behavior over the CORA data set. Finally, we compare our work to other ER models, generated with a framework called JedAI [13] and show that it challenges and outperforms other cutting-edge models.

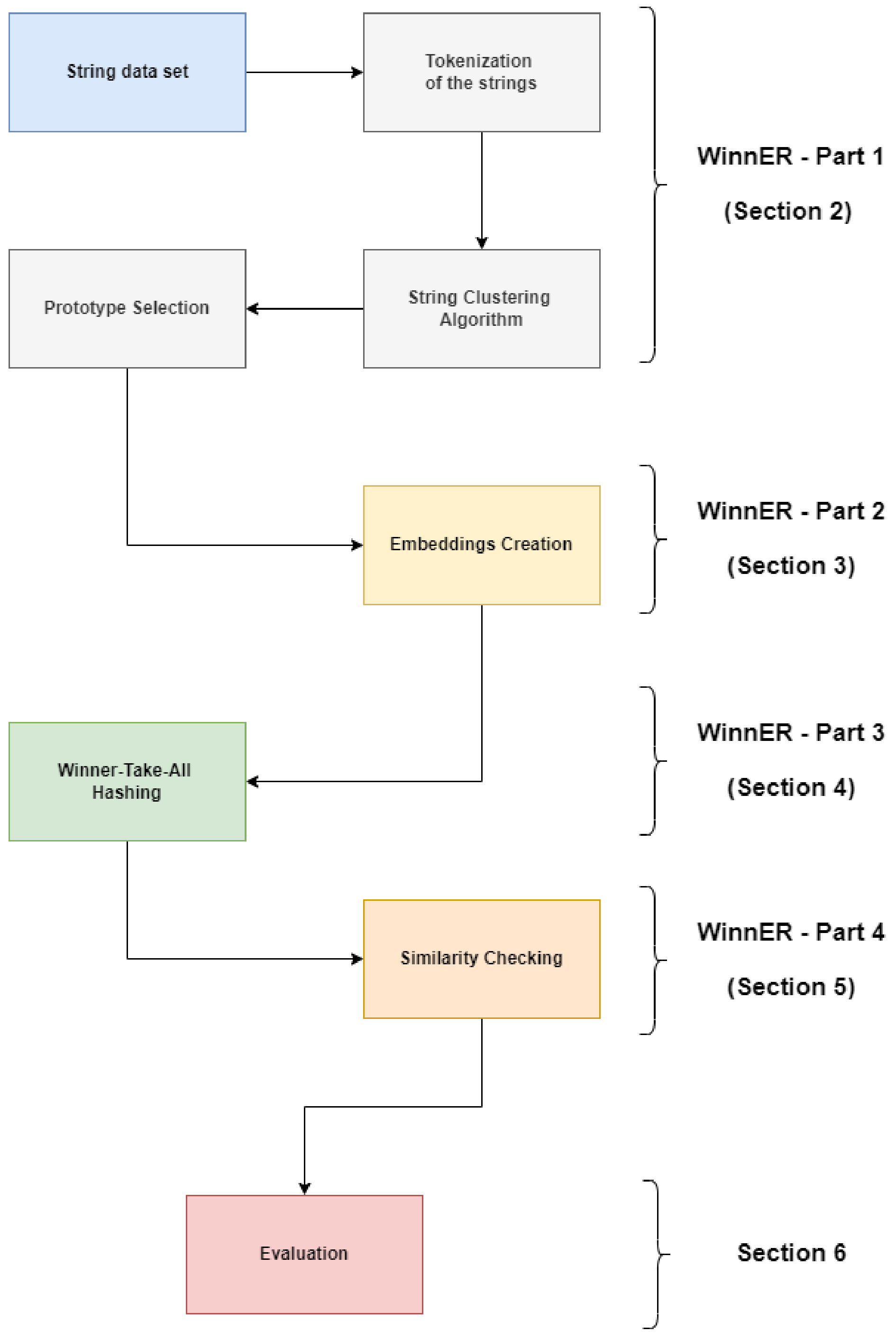

String data set
Tokenization of the strings
String Clustering Algorithm
Prototype Selection
Embeddings Creation
Winner-Take-All Hashing
Similarity Checking
Evaluation
WinnER - Part 1 (Section 2)
WinnER - Part 2 (Section 3)
WinnER - Part 3 (Section 4)
WinnER - Part 4 (Section 5)
Section 6

Figure 1: WinnER Workflow

2. SPACE CONSTRUCTION

This is the first part of this model. Input string data, must now be converted into a space of objects. There are plenty of algorithms for prototype selection, however, many of them are sensitive to data variability and so they cannot perform equally well in diverse data sets. In order to circumvent this situation, a Dissimilarity Space Of Objects [25] will be build.

We examine and compare prominent string metrics, like Edit and Jaccard distance and from this research, we present a novel distance stemming from the study in papers [16, 30]. These studies demonstrate that a metric can be both euclidean and dissimilarity at the same time. Nevertheless, from a more theoretical and detailed perspective, a euclidean distance that yields a dissimilarity score has a lot of potential. We should emphasize at this point that we will conduct a deep study of the string distance selection because it is at the heart of our model's prototype selection phase. We call this hybrid string distance Euclidean-Jaccard, and we will present how it works both theoretically and experimentally.

Additionally, we give a brief description of the ”tokenization” approach that this metric requires in order to work properly, as a metric in the Jaccard family. As a result, the Euclidean-Jaccard distance requires sets, thus the strings must first be converted into sets of characters or words. This transformation may be accomplished in a variety of methods, but we will concentrate on the n-grams method.

We next present the prototype selection part of this framework, where we use the algorithm described in the initial paper [29], which is derived from the Vantage Objects schema in the paper [15]. We will go through the clustering and prototype selection algorithms we utilize, as well as the theoretical properties that support this schema. We will also describe our tweaks and optimizations to this algorithm in order to better integrate it into our framework. Finally, we evaluate the performance of the prototype selection technique using the Maximum Mean Discrepancy (MMD) [2], a statistic measure between two data distributions. Measuring the MMD between the two distributions formed, one of the prototypes and the other of the initial data set, is one method to accomplish the evaluation.

2.1 Dissimilarity Space

In order to construct a Dissimilarity Space, it is needed to define a metric that will calculate how dissimilar two objects are (strings in our case). For calculating dissimilarity between two strings, there are two well-known metrics, Edit Distance [8] and Jaccard Distance [28], both of which are examined in this study. However, in this study, we will use a hybrid Euclidean-Jaccard dissimilarity metric [16] as it has the most potential and outperforms standard distance measures such as Jaccard and Edit distance. We will begin by describing some of the most important aspects of the Edit and Jaccard distances, and following we will make an extended report of the dissimilarity we adopt. At this time, it is also worth noting that one of the most significant parts of this project is the space construction, described in this chapter. Our study starts by defining the most famous dissimilarity distance, which is Edit or Levenshtein distance.

2.1.1 Edit or Levenshtein Distance

Edit Distance [8] is a metric of quantifying how dissimilar two strings are by measuring the number of operations required to turn one string into the other. Edit Distance is a distance metric, as it satisfies all the requirements of a distance metric, which are the following for three random elements e_1, e_2, e_3:

$$
\begin{aligned}
&d(e_1, e_2) \geq 0 && \textit{(non-negativity)}\\
&d(e_1, e_2) = 0 \textit{ iff } e_1 = e_2 && \textit{(identity rule)}\\
&d(e_1, e_2) = d(e_2, e_1) && \textit{(symmetry)}\\
&d(e_1, e_3) \leq d(e_1, e_2) + d(e_2, e_3) && \textit{(triangle inequality)}
\end{aligned}
$$

At this point, we should note that Edit Distance is not a Euclidean metric. Neither the space created with Edit Distance is. We can prove that Edit Distance is not a Euclidean distance in a the following proof. Take as an example Table 1.

Table 1: Edit Distance between four strings

strings	ab	abc	abd	abcd
ab	0	1	1	2
abc	1	0	1	1
abd	1	1	0	1
abcd	2	1	1	0

In the above table, we see four simple strings and the Edit Distance between them. Now, observe the following equalities.

$$
\begin{aligned}
d_{edit}(\text{"ab"}, \text{"abcd"}) &= 1 + 1 = 2 && \textit{(Distance of "ab" and "abcd")}\\
&= d_{edit}(\text{"ab"}, \text{"abc"}) + d_{edit}(\text{"abc"}, \text{"abd"}) = 2 && (1)\\
&= d_{edit}(\text{"ab"}, \text{"abd"}) + d_{edit}(\text{"abc"}, \text{"abd"}) = 2 && (2)
\end{aligned}
$$

Assume that the space that will be created is a Euclidean Geometric space. Hence, from Equation 1 strings ab, abc and abcd must be points of the same straight line. Moreover, from the Equation 2 strings ab, abd, and abcd must also be points of the same straight line. We also know from the Euclidean Geometry, that from two points only one line passes. As a result, all four strings must be points of the same straight line. This implies, however that strings abc and abd should coincide. This means that $d_{edit}(\text{"abc"}, \text{"abd"}) = 0$, which is false from the previous Table1. The bellow figure makes clear this example in a 2-dimension space.

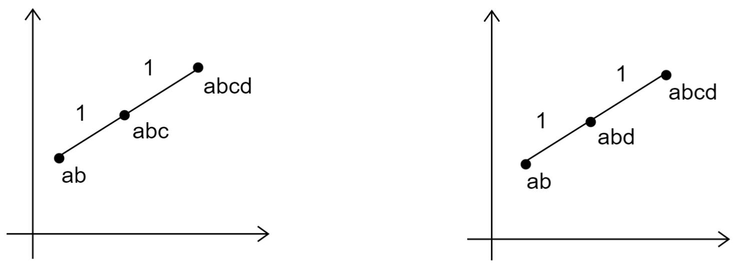


Figure 2: Visualizing strings of Table 1 in a Euclidean Space

It is highly important to make a brief reference here, to Euclidean Spaces. Euclidean Space is a property-rich space that has shown to be one of the most effective in the creation of embeddings. Embeddings are vectors that ”describe” each string. It will be much explained in the next section, section 3. One significant distinction is that one Euclidean metric provides linearity, whereas Edit Distance does not.

Edit Distance is a popular string metric, used in many applications. However it is not suitable for all problems. As a distance, it has a number of flaws, and it frequently fails to generate distances that show the difference between two strings. As an example, consider the two strings ”paper” and ”paapeer”. These strings are semantically same but have Edit Distance equal to four. We will see that some metrics, such as Jaccard, can measure the distance between them as zero, implying that they are same.

Another disadvantage of edit distance is its high complexity of $\Theta(mn)$, where m,n are the lengths of the strings. Therefore large strings, will make the whole process computationally expensive. Edit Distance, on the other hand, is a more simpler metric that takes into consideration the string’s ordering and was one of the metrics that we thoroughly tested for suitability in our model. Although, in some cases, a Jaccard-like distance appears to perform better than the Edit distance, this is why the Edit distance was not used in the space construction.

2.1.2 Jaccard Distance

Jaccard Distance [28] is a dissimilarity metric between sample sets. It is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union. Let A, B be two sets (set of words or chars in our case) and d jaccard distance:

$$d_{jaccard}(A,B) = \frac{\mid A \cup B \mid - \mid A \cap B \mid}{\mid A \cup B \mid}$$

where $d_{jaccard} \in [0,1]$.

Same as Edit Distance, Jaccard distance is a distance metric as it satisfies the requirements of a distance metric. Let’s now take the Table 1 and apply to the four strings, the Jaccard Distance. For this example we will tokenize the four strings into ”bigrams”. The definitions of tokenization and ”bigrams” will be explored in detail in a next subsection. So, Table 1 becomes:

Table 2: Jaccard Distance using Bigramms

strings	ab	abc	abd	abcd
ab	0.0	0.5	0.5	0.66
abc	0.5	0.0	0.66	0.33
abd	0.5	0.66	0.0	0.75
abcd	0.66	0.33	0.75	0.0

We can see how Jaccard distance works in the table above. Consider the difference between ”abd” and ”abcd”. The difference between these two strings is 0.75, or 75%. This dissimilarity can be easily transformed to similarity distance. In the same example, ”abd” and ”abcd” are similar by 1-0.75=0.25 or 25% (Jaccard Index).

2.1.3 Euclidean-Jaccard Distance

We have shown, so far that both Jaccard and Edit distance do not match Euclidean criteria. However, the scenario where a distance is at the same time a dissimilarity distance and fulfills the Euclidean criteria, appears to be rather promising. This is conceivable, and it stems from Gower and Legendre's paper titled *Metric and Euclidean properties of dissimilarity coefficients* [16], in which they established a number of conditions for a family of coefficients. Various metrics and characteristics of dissimilarity coefficients have been defined. They devised a formula from which these characteristics are derived:

$$T_\theta = \frac{\alpha}{\alpha + \theta(b + c)} \tag{3}$$

where α, b, c represent the number of binary asymmetric variables in a pair of binary vectors that happen to have two one's, an 1 and a 0 or a 0 and an 1 value correspondingly. If we replace θ with non-negative values then some fairly well-known coefficients will occur from the above equation 3. Furthermore, in [16] it is shown that the distance measure $\sqrt{1 - T_\theta}$ for $\theta \geq 1$ is a Euclidean Distance. As a consequence:

$$\sqrt{1 - T_\theta} = \sqrt{1 - d_{jaccard\_similarity}} = \sqrt{d_{jaccard\_distance}}$$

So the squared root of Jaccard Distance is a Euclidean metric. The table of the four strings comparison will be:

Table 3: Euclidean-Jaccard Distance using Bigrams

strings	ab	abc	abd	abcd
ab	0.0	0.7	0.7	0.81
abc	0.7	0.0	0.81	0.57
abd	0.7	0.81	0.0	0.86
abcd	0.81	0.57	0.86	0.0

The dissimilarity used in this framework is the Euclidean-Jaccard distance. There are a variety of advantages to this distance. To begin with, it is a Euclidean distance, as we mentioned before, which means it satisfies all of the criteria of a Euclidean space. Furthermore, it is a dissimilarity distance that fits our dissimilarity measure criterion. Euclidean-Jaccard, also, does not require a great time or memory complexity, as it only requires the same as Jaccard. For all of these reasons, Euclidean-Jaccard was chosen as the dissimilarity distance for our model's space construction.

2.1.4 Text transformation for Jaccard Distance

When using Jaccard Distance (same as Euclidean-Jaccard), the format of the two sets (A, B) is essential. To utilize this measure, every string must firstly converted into a set of words or characters. This can be accomplished in a variety of ways. In this model, a **tokenizer** is used to break down the initial text. The first step is to split the initial string into a list of words. Afterwards it's needed to create the sets (tokens are unique). This can be made by transforming the list created, into a set.

Nevertheless, the information about the order of words will be lost in this method. For this reason, it is necessary to generate sets made up of n-grams. **N-grams** are N consecutive tokens. If N=1 are called ”unigrams”, if N=2, then are called ”bigrams” and if N=3 are called ”trigrams”. Consider the string transformation below as an example.

Table 4: Word Tokenization example

Initial String	”entity resolution model”
Tokenization into words	”entity”, ”resolution”, ”model”
Unigrams	”entity”, ”resolution”, ”model”
Bigrams	”entity resolution”, ”resolution model”
Trigrams	”entity resolution model”

This process can also be applied in character level. Instead of tokenization in words, will perform character tokenization.

Table 5: Char Tokenization example

Initial String	”entity resolution model”
Tokenization into chars	”e”, ”n”, ”t”, ”i”, ”t”, ”y”, ” ”, ”r”, ”e”, ”s”, ...
Set of tokens	”e”, ”n”, ”t”, ”i”, ”y”, ” ”, ”r”, ”s”, ...
Unigrams	”e”, ”n”, ”t”, ”i”, ”y”, ” ”, ”r”, ”s”, ...
Bigrams	”en”, ”nt”, ”ti”, ”it”, ”ty”, ”y ”, ” r”, ”re”, ”es”, ”so”, ...
Trigrams	”ent”, ”nti”, ”tit”, ”ity”, ”ty ”, ”y r”, ” re”, ”res”, ”eso”, ”sol”, ...

Both of these transformations have benefits and drawbacks. The first example’s sets (word-tokenization) include fewer elements, which may result in faster transformation but lower accuracy when comparing two strings. The second strategy, char-tokenization, will, on the other hand, be more accurate. Both of these approaches will be tested on an actual data set, and the results will reveal which is the superior of the two.

2.2 String Clustering & Prototype Selection

Aim of this part of the model is to choose a set of strings, that will be our Prototypes. This process is consisted of two logical parts. The first one is the String Clustering phase, in which clusters of strings will be produced. The second one is the Prototype Selection, that will take the string clusters and will generate a set of Prototypes. These Prototypes will be the points of the embedding process. This part of the framework we propose is a combination of the works *Efficient image retrieval through vantage objects* by Jules Vleugels and Remco C. Veltkamp [15] and *Representation and Recognition in Vision* by S. Edelman [26]. In the first paper, it is firstly defined an approach called *Vantage Objects* for embedding the dissimilarity space of the initial objects (not necessarily strings) into a reduced dimensionality metric space. This approach in combination with a prototype selection schema, described in the second paper and very accurately named *The Chorus of Prototypes*, is the schema adopted in the paper [26] and also in this study. It will become obvious the superiority among other prototype selection algorithms, both in theoretical and experimental aspects.

2.2.1 String Clustering Phase

The String Clustering technique is not complicated at all, which is why it is both fast and efficient. When this procedure is finished, it will generate a finite number of clusters (which can be fine-tuned) and two strings for each cluster. With these two representative strings, we can generate an inner product space. More specifically, the first string will serve as the origin and the second as the endpoint of a vector, from which all of the other strings will be projected. If the strings projected by this vector meet certain criteria, they will become members of this cluster.

This algorithm begins by traversing the string set. For every string, iterates the number of clusters generated up to this point. And here's the gist of it. Let's begin with the first string in the initial data set. If the first representative has not yet been set, this string will become the first (left) representative, and the algorithm will proceed to the next string. Because the first representative of this cluster has been set in the first loop, a dissimilarity distance will be calculated between the current string and the first representative. If this distance is less than *d*, where *d* is the maximum distance threshold, this string becomes the second (right) representative. Now that both representatives have been initialized, the condition that must be met when a new string *s* is examined for membership in this cluster is:

$$\mathit{distance(s, left\_representative)} + \mathit{distance}(s, \mathit{right\_representative}) \leq d$$

Where *d* is a dissimilarity distance, like Edit, Jaccard or Euclidean-Jaccard distance. If this condition is satisfied, then this string will become member of this cluster. This is briefly the method adopted for the String Clustering phase.

Let A, B, C, D, E and F are strings in the original set. Also assume that, A and B are the first strings that this algorithm process, and have a distance less that *d*. As a result A and B will be this cluster representatives. Now we need to check C for membership to the cluster, A and B, represent. This experiment is depicted in the figure below.

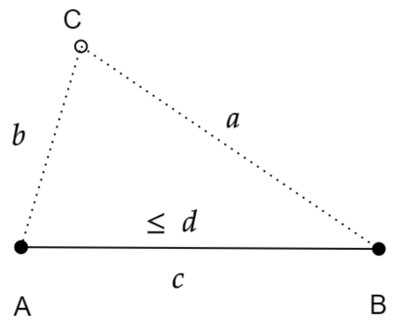


Figure 3: Triangle formed from A, B, C

C, will be tested for membership. Consider that $a + b$ is less than *d* and thus C becomes a member of this cluster. Since A and B are the representatives, we get:

$$d_{AC} = b \leq d \qquad d_{BC} = a \leq d$$
$$a + b \leq d \qquad c \leq d$$

Now we need to check the rest of the strings D, E and F. So we have the situation shown in the bellow figure:

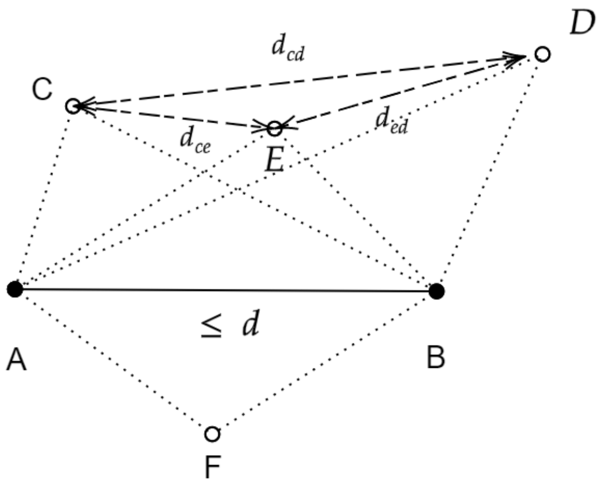


Figure 4: Visualization example of String Clustering Properties

D and E now must be checked for membership to this cluster. D and E will only join if the sum of their distances from A and B (representatives) is less that *d*. Assume that this is satisfied and D and E join the cluster. We want to see, if that would be enough, to compare every string in the data set to only two strings, meaning to the representatives, in order to enter to that cluster. So we must now examine what occurs between C, D, and E. Based on our assumptions, we get:

$$d_{AC} + d_{BC} \leq d \qquad d_{AE} + d_{BE} \leq d \qquad d_{AD} + d_{BD} \leq d \tag{4}$$

Please remember that all of the distances, used in this model are metrics, as they satisfy the three requirements. We refer you to Section 2.1 where we mention them. One of the requirements is Triangle Inequality. Thus we can apply the Triangle Inequality to the triangles $\overset{\triangle}{ACE}$ and $\overset{\triangle}{CEB}$ and we get:

$$d_{CE} \leq d_{AE} + d_{AC} \qquad d_{CE} \leq d_{BE} + d_{BC} \tag{5}$$

If we sum the above inequalities (5) we get:

$$\begin{aligned} 2d_{CE} &\leq d_{AE} + d_{AC} + d_{BE} + d_{BC} \Rightarrow \\ &\leq (d_{AE} + d_{BE}) + (d_{AC} + d_{BC}) \overset{(4)}{\Longrightarrow} \\ &\leq d + d \\ &\leq 2d \end{aligned}$$

Hence,

$$d_{CE} \leq d$$

E and F strings can be demonstrated in the same way that the distance, from each of the items within the cluster, is less than or equal to d . This proof revealed that all of the elements in this cluster have a distance less than d, between them, which is possible by simply comparing a string to the cluster representatives. This makes the entire process incredibly efficient. The complexity of this approach is $O(nk)$, where n is the number of strings in the initial data set and k is the maximum number of clusters. Keep in mind that k and d are hyper-parameters that can be modified.

This approach has a minor flaw in that it is somewhat dependent on the relative ordering of the input strings. Some clusters may contain elements with similar characteristics. This is the consequence of the representatives' initialization procedure. It will be detailed in further depth in the following part, where we will apply an optimization to get around this issue.

2.2.2 Prototype Selection

The next part of the prototype selection algorithm is to find the most distinct strings from each cluster created in the previous step. We will use the same technique described in Algorithm 1 of the first publication [29], but with an optimization as we spotted a vulnerability that will have a separate reference. Our aim in this part is to produce a set of objects, that will serve as the prototypes for the embedding phase. The string clustering step yielded a number of clusters, each with two representative strings. As the Prototype, we might choose one of the two representatives from each cluster. However, this would result in a low-quality set of prototypes. Instead, we will use a different approach to locate an indicative string in each cluster, which will become one of the prototypes.

The basic idea, is to find one distinguishing string from every cluster. One that will represent the strings of this cluster while also being distinct from the others chosen from the other clusters. This will assist us in creating a rich Dissimilarity space and, as a result, embeddings that will represent each string in great detail. Shortly, embeddings are vectors, one for every string, and are consisted by the distances of each string from all the prototypes. We provide an extensive analysis for the embeddings in Section 3. If the prototype selection method is successful, the final prediction and the scores will be high.

This part of the algorithm, starts as follows. Firstly, for each cluster, calculates the projections of every string, with the leftmost representative of this cluster. This projection distance helps us create a string ranking. Keep in mind that we need a distinct string. These projections can be produced quickly and without great complexity, as we will take advantage of all the distances calculated and saved from the string clustering phase. After these projections are calculated, they will be sorted and the median value of these distances, will be selected. This distance, we remind you, is the distance between one cluster string and one representative. As a consequence, the string whose distance has been chosen will become the Prototype string. It is extremely helpful to visualize a two dimension example in order to better grasp this strategy. This approach, nevertheless, can be extended to N dimensions. Figure 5 depicts the basic concept.

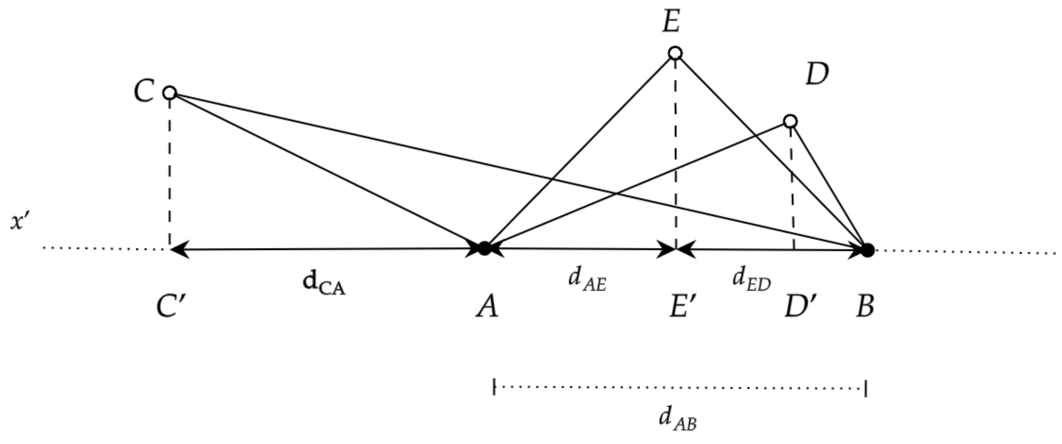


Figure 5: 2-Dimension example of Projections Calculation

Let A, B, C, D and E nodes represent strings, same as the previous graphics. A and B are the representatives and C, D and E are strings that belong to this cluster. Also assume that the left representative is A. Based on the process described previously, we need to calculate all projections of C, D and E from A. The (orthogonal) projection of a point to a line, for example C from line $x'x$, is the point of $x'x$ at which the normal line of $x'x$ passing through C, intersects with $x'x$. Lets now calculate the projection of C to A, d_{CA}, projected in the line formed by A and B. Please note that we assume that A and B are already projected, which means that we do not have to do anything about these two strings.

$$\begin{aligned}
CA^2 &= CC'^2 + C'A^2 \Leftrightarrow && \textit{(Pythagorean Theorem on } \overset{\triangle}{CC'A}\textit{)} \\
CA^2 &= CC'^2 + d_{CA}^2 \Leftrightarrow && (C'A = d_{CA}) \\
CA^2 &= CB^2 - (d_{CA} + AB)^2 + d_{CA}^2 \Leftrightarrow && (\overset{\triangle}{C'CB}\text{: } CC'^2 + C'B^2 = CB^2) \\
CA^2 &= CB^2 - d_{CA}^2 - AB^2 - 2 \cdot d_{CA} \cdot AB + d_{CA}^2 \Leftrightarrow && (\textit{Algebraic identity}) \\
CA^2 &= CB^2 - AB^2 - 2 \cdot d_{CA} \cdot AB \Leftrightarrow \\
2 \cdot d_{CA} \cdot AB &= CB^2 - AB^2 - CA^2
\end{aligned}$$

therefore,

$$d_{CA} = \frac{CB^2 - AB^2 - CA^2}{2 \cdot AB}$$

With the same method we can prove this approximated projection distance for all the strings into the cluster. After this process has finished we store all these distances in an array and then we sort them. Finally we peak the median value of this array and the string whose projection distance is the median value, becomes one of the Prototypes.

All of the above information may now be summed up in the Algorithm 1. It's the same approach as in the paper [29], but with more detail and a final optimization where we included a check between the final set of prototypes. This is the optimization is described in the following section.

Algorithm 1 String Clustering and Prototype Selection Algorithm

```
1:  $S$ : Array of strings (dataset)
2:  $d$ : Max distance threshold
3:  $k$ : Max number of clusters
4:
5:  procedure PrototypeSelection($S, k, d$)
6:      $i \leftarrow 0$                                    ▷ Variable initialization
7:      $y \leftarrow 0$
8:      $C \leftarrow array(0 : size(\mathrm{S}))$                  ▷ C: Cluster array
9:      $r \leftarrow array([2, k])$                ▷ r: 2-dimension Representatives array
10:
11:     for $i < size(\mathrm{S}), i \leftarrow i + 1$ do                ▷ String clustering phase
12:         while $j < k$ do                          ▷ iteration through clusters
13:             if $r_0(j) = \emptyset$ then          ▷ case empty first representative for cluster j
14:                 $r_0(j) \leftarrow S[i]$
15:                 $C[i] \leftarrow j$                ▷ store in C that i-string belongs to cluster j
16:                 break
17:             else if $r_1(j) \neq \emptyset$ and distance$(S[i], r_0(j)) \leq d$ then
18:                 $r_1(j) \leftarrow S[i]$
19:                 $C[i] \leftarrow j$
20:                 break
21:             else if $r_0(j) \neq \emptyset$ and $r_1(j) \neq \emptyset$                ▷ triangle inequality check
22:                 and (distance$(S[i], r_0(j))$ + distance$(S[i], r_0(j))) \leq d$ then
23:                 $C[i] \leftarrow j$
24:                 break
25:             else
26:                 $j \leftarrow j + 1$
27:             end if
28:         end while
29:     end for
30:
31:     Projections $\leftarrow$ emptyList()                       ▷ Prototype selection phase
32:     Prototypes $\leftarrow$ emptyList()
33:     SortedProjections $\leftarrow$ emptyList()
34:     finalNumOfClusters $\leftarrow k$
35:     $j \leftarrow 0$
36:     for $j < k, j \leftarrow j + j$ do
37:         aprxDistances $\leftarrow$ AprxProjectionDistancesOfCluster$(r_1[j], r_0[j], Cluster(j))$
38:         if aprxDistances $= \emptyset$ then                ▷ no prototype from this cluster
39:             finalNumOfClusters $\leftarrow$ finalNumOfClusters$-1$
40:             continue
41:         end if
42:         Projections$(j) \leftarrow$ aprxDistances
43:         SortedProjections$(j) \leftarrow$ Sort(Projections$(j)$)
44:         Prototypes$(j) \leftarrow$ median(SortedProjections$(j)$)
45:     end for
46:     OptimizeClusterSelection(Prototypes, finalNumOfClusters)
47:
48:     return Prototypes, finalNumOfClusters
49: end procedure
```

2.3 Optimization

During the testing of this prototype selection procedure, an issue arose. The problem was that many of the prototypes chosen were very similar. This occurred as a result of the vulnerability discussed during the Clustering phase. The relative order of strings in the data set makes this approach susceptible. This created some complications during the embedding phase, when the space was being built. Many strings with distinct semantics had remarkable similar embeddings and hence were predicted as the same. As a result, at the end of the process, we included a sanity check function that performs a basic task. It calculates the distances between all of the chosen prototypes and checks if they are less than a certain threshold of similarity. If a pair of prototypes fails to meet this criterion, one of the prototypes is removed from the set.

2.4 Evaluation of Prototype Selection

After the development the prototype selection part, it is essential to evaluate it in a variety of ways. The first one is a metric we used, called **Maximum Mean Discrepancy** (MMD) and was firstly referenced in the paper [2]. MMD, measures the discrepancy between two distributions. The selection of prototypes creates a density distribution of prototypes. We want to see if the distribution of prototypes differs from the data distribution. The maximum mean discrepancy estimates the difference between two distributions and is the supremum over a function space of expectancies discrepancies between the two distributions. The squared MMD (MMD^2) measure can be calculated using the formula below.

$$MMD^2 = \frac{1}{m^2}\sum_{i,j=1}^{m} k(z_i, z_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(z_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(x_i, x_j)$$

where,

- **k** is the distance metric
- **m** is the number of prototypes
- **n** is the number of data points x in our original dataset
- **z** are the prototypes

The goal of MMD-critic is to minimize MMD^2. The closer MMD^2 is to zero, the better the distribution of the prototypes fits the data. The key to bringing MMD^2 down to zero is the term in the middle, which calculates the average proximity between the prototypes and all other data points (multiplied by 2). If this term adds up to the first term (the average proximity of the prototypes to each other) plus the last term (the average proximity of the data points to each other), then the prototypes explain the data perfectly.

Another method of evaluating the prototype selection is to perform dimensionality reduction into the embeddings, that will be created. We can get a quick glimpse of the data distribution using the Principal Component Analysis (PCA) algorithm or the Multidimensional Scaling (MDS) algorithm. Keep in mind that these methods are not ideal, as a multidimensional problem is reduced to a two- or three-dimensional problem in order to be visualized. Some information may be lost as a result of the reduction in dimensions. We shall, however, use these methods because they can provide us a view of the prototype selection process. This will be explained more in the next sections.

3. VANTAGE EMBEDDINGS

This is the second part of our model. At this point we have created a schema that generates a set of prototypes from a given data set. It is now needed to transform the initial strings into numerical arrays. In Pattern Recognition, this method is called vectorization and the vectors are also called embeddings. **Embeddings** are a number of vectors, one for each string, that describes it, in a unique way. In our case we want to produce a vector (imagine it like a list of numbers) for every string. In this model we will adopt an embedding technique based on the Vantage Objects. The basic idea of the Vantage Objects comes from the paper, *Efficient image retrieval through vantage objects* [15], by *J. Vleugels and R. C. Veltkamp*. They first mentioned and used a method called Vantage Objects in order to create an efficient object indexing. The Prototypes generated in the previous step of the model will be the Vantage Objects in our study.

3.1 Prototypes as the Vantage Objects

In the previous section, we demonstrated one algorithm that processes a big set of strings and finally collects the most distinctive strings into a set. This is the set of Prototypes, as we have mentioned before. Let this set called P and $P = \{p_1, p_2, ..., p_k\}$, are some of the strings s of the initial data set S, where k is the number of prototypes. We shall now construct a Vantage-Dissimilarity Space based on them. Each string will be ”transformed” into a list of numbers. These numbers, are going to be the distances from each prototype. The process of Vectorization, starts by iterating all strings. For every string, it iterates the set of Prototypes P and calculates the distance between string s and prototype p. This distance can be the same distance we used in the previous part, meaning a Jaccard distance. If we use the same distance metric as before, we will avoid many calculations, as these distances have already been calculated and saved. Returning to the process of embeddings, after we’ve calculated the distance we save it into a vector. So in the end of this process we will produce a vector of distances from all the prototypes, for all strings. And this will be the embedding of each string.

Bellow we provide a visualization of this technique in order to be better explained. Assume we have strings $s_1, s_2, s_3, ..., s_n \in S$ where n is the number of strings in the initial data set. Also, assume that $p_1, p_2, ..., p_k \in P$ are the prototype strings, selected at the clustering phase and d the dissimilarity distance used in the Clustering phase. The graphic below displays the embeddings in a concise manner.

$$
\begin{array}{lcl}
s_1 & \longrightarrow & [d(s_1, p_1),\ d(s_1, p_2),\ \dots\ ,\ d(s_1, p_k)\ ] \\
s_2 & \longrightarrow & [d(s_2, p_1),\ d(s_2, p_2),\ \dots\ ,\ d(s_2, p_k)\ ] \\
s_3 & \longrightarrow & [d(s_3, p_1),\ d(s_3, p_2),\ \dots\ ,\ d(s_3, p_k)\ ] \\
\vdots & & \qquad\qquad\qquad \vdots \\
s_n & \longrightarrow & [d(s_n, p_1),\ d(s_n, p_2),\ \dots\ ,\ d(s_n, p_k)\ ]
\end{array}
$$

Figure 6: Embedding process

3.2 Selection of embedding distance

Maybe the most important algorithmic decision, in the embedding phase, is the selection of the distance metric that will be used. Euclidean-Jaccard is the distance, that will serve in this framework as the embedding metric. Another option, is to use a norm of the Euclidean-Jaccard distance. We shall give a short definition of the statistical term norm and we will explain why we did not finally adopt a norm in the embedding procedure.

3.2.1 Predefined distances

Aim of the embedding schema is to create embeddings that best ”describe” each string. Euclidean-Jaccard is the distance used as the embedding metric, for a number of reasons. Firstly, we take advantage of all the distances calculated in the prototype selection phase, by utilizing the Euclidean-Jaccard distance. This is both simple and time efficient as the majority of the distances needed for the embeddings, have been already calculated and saved. Furthermore, using the distance that the space was built is also a reliable and widely used strategy.

Though, using a distance metric like Euclidean-Jaccard, may be problematic some times. In Pattern Recognition, there is a well known phenomenon called Curse of Dimensionality [4]. This is caused when we face high-dimension problems. It will be thoroughly explained in the following section. Keep in mind that by smoothing and normalizing the data, we can tackle the issues that arise from this situation. This is accomplished by employing distance norms rather than just distances.

3.2.2 Distances Norms

Vector norms are widely used not only in Pattern Recognition and Machine Learning, but also in a wide range of Mathematical disciplines. The size of a vector is frequently calculated using norms. To begin, we will define what is a norm. In linear algebra, the size of a vector v is called the norm of $\vec{v}$. For example lets consider vector $\vec{v}$ of ordered numbers.

$$\vec{v} = (v_1, v_2, ..., v_n) \qquad\qquad (v_i \in C \; for \; i \; = 1, 2, ..., n)$$

$L_1 - norm, L_2 - norm$ and $L_\infty - norm$ are some of the widely used norms. In this work we examine the $L_\infty - norm$. $L_\infty - norm$ is the infinity norm (also known as the ∞-norm, max norm, or uniform norm) of a vector $\vec{v}$ is defined as the maximum of the absolute values of its components:

$$||\vec{v_i}||_\infty = \max\{|v_i| \colon i = 1, 2, ..., n\}$$

Norms are not metrics as they do not calculate distances. However we can use norms to vectors consisted of distances. This way, we perform a small normalization when creating the embeddings. This is not always the best technique. For example, in the next step of our model, we apply a hashing algorithm, and this smoothness may have the opposite impact than expected. In this study we examined the $L_\infty - norm$ as this norm seem to fit the most with the hashing approach selected in the next section. Although, using the Euclidean-Jaccard distance without a norm came out to be the best solution. This algorithmic decision occurred after an experimental study on this model, in which we tested both $L_\infty - norm$ and the Euclidean-Jaccard distance without a norm. The experiments showed us that $L_\infty - norm$ had no impact on increasing the performance of the model. On the other hand, the infinity norm, needed a noticeable time of execution in comparison of just using the distance itself.

3.3 Curse of Dimensionality

The curse of dimensionality [4], is a well-known problem in Machine Learning and occurs when we have high-dimension data. Although the curse of dimensionality poses significant challenges for pattern recognition algorithms, it does not preclude us from developing successful algorithms for high-dimensional environments. There are two explanations for this. First, real data is frequently limited to a region of the space with lower effective dimensionality, particularly in the directions where significant fluctuations in the target variables occur. Second, real data usually have some smoothness properties, which means that small changes in the input variables will produce small changes in the target variables for the most part, and we can use local interpolation-like techniques to make target variable predictions for new input variable values.

In our study, the issue that arises from the high-dimension data will be tackled in simple but efficient way. The embeddings created in this phase will be rank-ordered. We shall give a small definition of this technique. Figure 7 depicts how one embedding is rank-ordered (i.e using Jaccard distance as embedding distance). Rank-ordering a vector, means that we create a new vector where the values are the rankings of the initial sorted vector. For example, 0.3 is the fourth element in the ranking that would emerged if we sorted the initial vector from the minimum 0.01, to the maximum value which is the number 0.89.

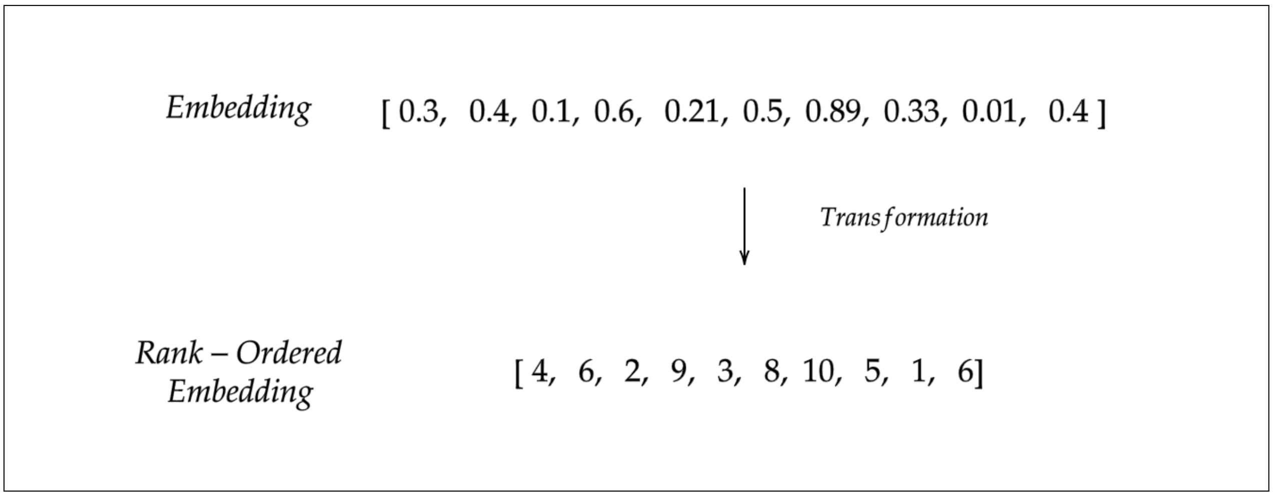

Embedding [0.3, 0.4, 0.1, 0.6, 0.21, 0.5, 0.89, 0.33, 0.01, 0.4]
Transformation
Rank – Ordered Embedding [4, 6, 2, 9, 3, 8, 10, 5, 1, 6]

Figure 7: Rank-Ordering an Embedding example

Rank-ordering the embeddings gives a number of advantages. As previously stated, is an effective technique to address the issues that arise as a result of the high dimensionality of ER problems. At the same time, these rank-ordered embeddings will be compared for similarity, with metrics mainly being used to compare full or partial rankings. Finally, rank-ordering the embeddings enables us to utilize the WTA hashing as it is developed for rank-ordered data. WTA hashing is a significant part of our framework and thus rank-ordering is essential.

4. WINNER-TAKE-ALL HASHING ALGORITHM

Hashing is the third and possibly the most important part of the model. In terms of time and memory complexity, entity resolution problems are quite demanding. This is due to the fact that numerous comparisons must be perfromed in order to determine which objects are similar and which are not. Consider a problem using a data set of 1,000 strings, which is considered extremely small nowadays. If we want to find which strings are the same as real-world entities, in a brute force way, we would compare each string with all the others. This means we would have to do 1,000,000 comparisons only to complete this task. This method would take a long time to complete and would require a lot of computing resources. As a result, a hashing method is utilized in this work, which can minimize the number of comparisons to 10% in many cases. Following the previous example, this model can produce predictions with 100,000 comparisons and fairly good scores.

This strategy is also called *blocking* and is a technique for splitting sets of records into smaller subsets using a criteria function (i.e. a hashing method), with only records belonging to the same block being checked for matches. All records with the same blocking key are placed into the same block in standard blocking, where they are compared pairwise. In Entity Resolution problems, blocking is a common strategy. It may be used to solve ER issues in a single set or to link records from different sets. Furthermore, blocking is a strategy used in cutting-edge frameworks with a proven performance boost, as evidenced by the papers [5, 9, 10, 11, 12].

Jay Yagnik's Locality Sensitive Hashing (LSH) Approach, called Winner-Take-All (WTA) Hashing, is used in this study in order to embed a blocking schema. Firstly mentioned in the work *The Power of Comparative Reasoning* [17], Yagnik presented a very efficient Hashing algorithm, which can create groups of objects from the initial data set that share common properties with a very low computational cost. WTA schema has a fairly basic function, but it is also one of the fastest hashing techniques available. This algorithm takes rank-ordered vectors as input and generates a hash for each one. This hash is utilized as an indicator in our model to generate groups or buckets of seemingly similar embeddings. This phase is extremely important in our model because it drastically minimizes the amount of time required to perform flawlessly.

4.1 Functionality

Hashing approaches, are used in various ML applications. MinHash [1] is one of the most famous algorithms for hashing and it is used in many state-of-the-art projects. In the original paper of WTA, it is proven that it is a generalization of MinHash. It is essential to provide a sample of WTA functioning at this time. Assume X is a collection of embeddings. The initial stage in this schema is to permute all the input vectors (embeddings). Permutation is a mathematical technique for rearranging a list of elements. WTA permutations are performed in a random sequence. The permutations that will be conducted in our case must be consistent for all samples. As a result, we will need a predefined random sequence, let us call it θ, to reorder all of the embedding vectors. For each of these embeddings, WTA algorithm will keep, the first K elements. From these elements, the maximum value will be selected. The index of this maximum value will become the hash code, that will indicate the bucket that this string belongs.

Permutation vector θ	$[\ \theta_0\ \ \theta_1\ \ \theta_2\ \ \theta_3\ \ \theta_4\ \ \theta_5\ ]$	$[\ 1\ 4\ 2\ 5\ 0\ 3\ ]$
Example vector X	$[\ x_0\ \ x_1\ \ x_2\ \ x_3\ \ x_4\ \ x_5\ ]$	$[\ 10\ 5\ 2\ 6\ 12\ 3\ ]$
Permuted X′	$[\ \theta_{x_0}\ \ \theta_{x_1}\ \ \theta_{x_2}\ \ \theta_{x_3}\ \ \theta_{x_4}\ \ \theta_{x_5}\ ]$	$[\ 5\ 12\ 2\ 3\ 10\ 6\ ]$

Figure 8: Permutation of vector *X* with *θ*

To better grasp this method, lets illustrate it. We used an example from Yagnik's original paper. Vectors a, b, c and d are the input (*X*) of the algorithm. Consider them as the vectors that will be produced after the embedding phase. WTA starts by generating the random sequence *θ*, as shown in Figure 9.

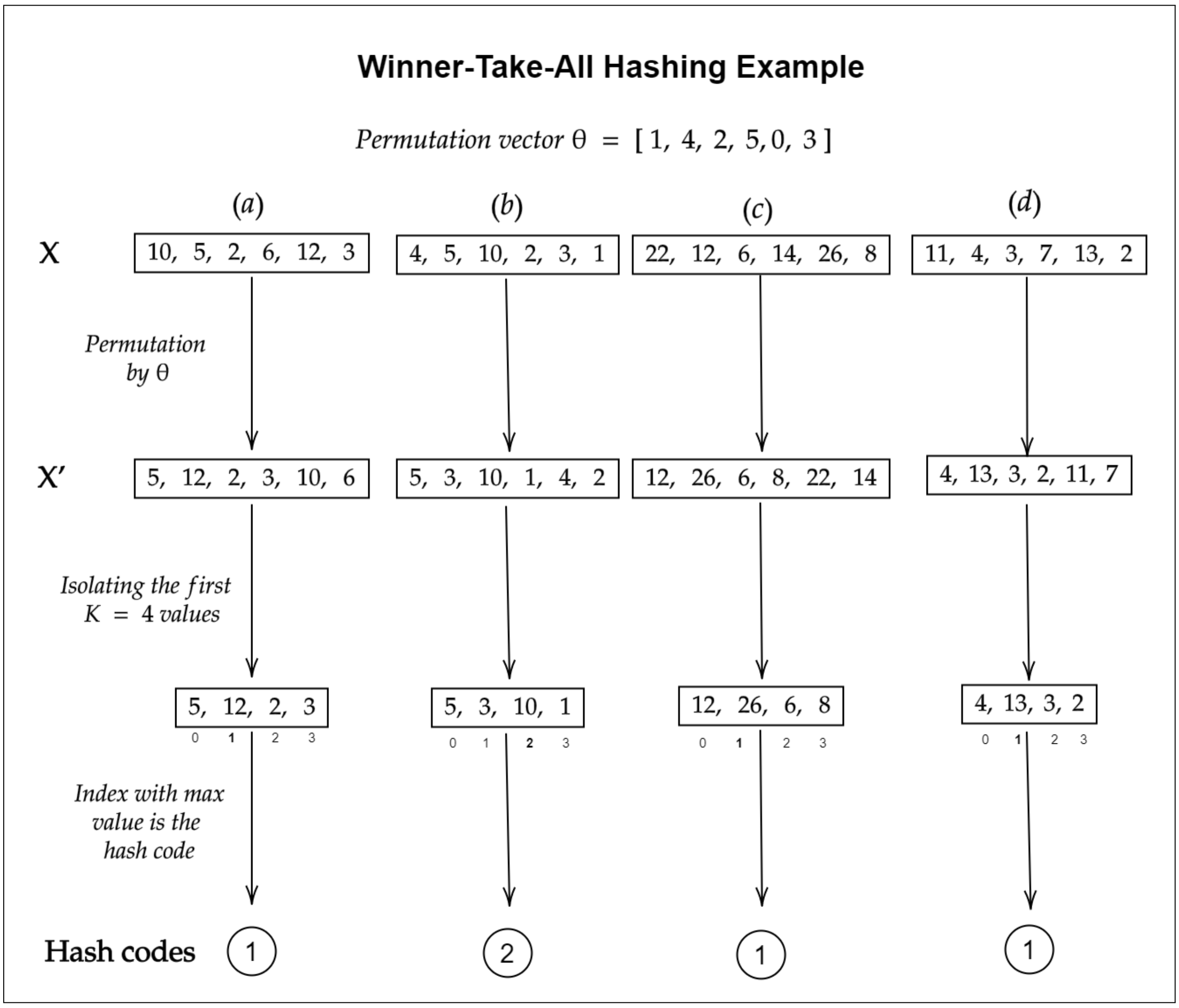


Figure 9: WTA Example

Figure 9 presents the flow of WTA hashing. Starts by taking as input the rank-ordered vectors. After that, permutes each vector, based on theta and then from the first K elements of each vector finds the maximum value and returns its index. To better grasp this hashing schema, consider Figure 9:(a) as an example. Firstly, (a) is permuted with θ, then on the first K elements, the maximum value is peeked. In case of (a), the maximum value is 12. The index of this value is 1. So 1 is the hash code of (a). Keep in mind that this procedure, iterates first for (a) after for (b), etc. WTA hashing provides a hash code for each embedding, in O(n) time, where n is the number of embeddings. This is one of the most essential aspects of WTA hashing.

4.2 Hashing more than once

This hashing process can be repeated multiple times in order to improve the model's scores. A WinnER hyper-parameter is the number of WTA hashings that will be executed. A hyper-parameter is a model variable that is provided as input and can be fine-tuned. This is not the only hyper-parameter of the model, as has been mentioned before. Hyper-parameters enable ML algorithms to fit a wide range of data sets and data types. At the same time, a large number of these factors can make the model difficult to employ because it will take a long time to fine-tune and produce sufficient results. As a result, all hyper-parameters in this work are rigorously tested for usability, and their number is kept to a minimum.

We perform the hashing multiple times in order to increase the possibility of comparing same objects and thus the number of successful comparisons. This method has both pluses and minuses. To begin, several buckets will end up containing same objects. This means that some pairs of object will be compared several times. However, some items that are similar but have not hashed to the same hash code, in the first hashing, may hash in the next hashing as WTA contains a randomization. The more hashing is performed, the more pairs will be compared, and hence the more similar pairs will be successfully predicted. On the other hand, this procedure will increase the model's overall execution time.

Following our research, we discovered that the number of hashings should be between 2 and 5, and that anything above these limitations produces a marginally better result while increasing significantly the execution time, causing the hashing to lose its purpose. These limits were discovered in many data sets as a result of our experiments. When utilizing hashing algorithms, multiple hashings is a frequent strategy. However, there is a trade-off between execution time and increased similarity checking results. For this reason, the number of hashings is hyper-parameter. The number of hashings required to achieve the best results in the shortest amount of time will be determined by the experimental findings and the hyper-parameter tuning.

5. SIMILARITY CHECKING

This is the final part of our model. WTA hashing has produced a number of buckets. Now it is time to check all of the string embeddings inside each bucket for thorough similarity. The technique of determining if two objects are similar is known as similarity checking. A similarity metric can be used to accomplish this. This metric, in general, takes two vectors (in our instance, embeddings) as input and generates a prediction. This prediction is a number between a fraction of numbers and most commonly, between zero and one. The closer the prediction is to one, the more similar they are. To perform, we must first establish a similarity threshold. We will assume that these two objects are similar if the similarity measure gives a value greater than this threshold. If the prediction falls below that threshold, however, these items should be considered as different.

5.1 Similarity Metrics

In this schema, we have rank-ordered embeddings. Only a few metrics for comparing rank-ordered vectors have been developed. Some of them are built for full rankings exclusively, while others are only used for partial rankings. In statistics, these metrics are found as rank correlation coefficients. A rank correlation coefficient measures the degree of similarity between two rankings. Some of the more popular rank correlation statistics include Spearman's ρ [31], Kendall's τ [20], Goodman and Kruskal's γ [19], Somers' D [24] among others. In this work we will mainly focus on the Spearman's ρ and Kendall's τ. Both these metrics are rank correlation coefficients and produce a value inside the interval $[-1, 1]$ where is:

- 1 if the agreement between the two rankings is perfect; the two rankings are the same.
- 0 if the rankings are completely independent.
- −1 if the disagreement between the two rankings is perfect; one ranking is the reverse of the other

Spearman's ρ and Kendall's τ were the two similarity metrics, that stood out as we have rank ordered embeddings. For this reason we shall present both metrics and explain why Kendall's τ was the one selected to be the similarity metric in our model.

5.1.1 Spearman's rank correlation coefficient

Spearman's rank-order correlation [31] is the non-parametric version of the Pearson correlation [18]. Spearman's correlation coefficient, (ρ, also signified by r_s) measures the strength and direction of association between two ranked variables.

The Spearman correlation between two variables is equal to the Pearson correlation between the rank values of those two variables. Spearman's correlation determines the strength and direction of the monotonic relationship between your two variables rather than the strength and direction of the linear relationship between your two variables, which is what Pearson's correlation determines. A monotonic relationship is a relationship that does one of the following:

- as the value of one variable increases, so does the value of the other variable
- as the value of one variable increases, the other variable value decreases.

Spearman's correlation measures the strength and direction of monotonic association between two variables. Monotonicity is ”less restrictive” than that of a linear relationship.

There are two methods to calculate Spearman's correlation depending on whether variables have not tied ranks or the opposite. Let X and Y be two variables and n the size of pairs. The formula for when there are no tied ranks is:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where $d_i = |x_i - y_i|$ the difference between the pair ranks i and n number of pairs. The formula of Spearmans rank correlation in case that we have ties is:

$$\rho = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2\sum_{i=1}^{n}(y_i-\overline{y})^2}}$$

where:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

5.1.2 Kendall Tau rank correlation coefficient

Kendall Tau is a measure of rank correlation [20] that calculates the similarity of the orderings of the data when ranked by each of the quantities. Kendall Tau metric, is the difference of the number of concordant pairs to the number of discordant, divided by the number of the possible pairs. Concordant and discordant pairings are calculated by comparing the classifications of two variables (i.e X and Y) on the same two items. The pairs are concordant if the classifications are in the same direction. The pair is discordant if the classification direction is not the same. More specifically for a pair of observations (X_i, Y_i) and (X_j, Y_j):

- **Concordant pair** if ($X_i > X_j$ and $Y_i > Y_j$) or ($X_i < X_j$ and $Y_i < Y_j$)
- **Discordant pair** if ($X_i > X_j$ and $Y_i < Y_j$) or ($X_i < X_j$ and $Y_i > Y_j$)

Let **C** be the number of concordant pairs, **D** the number of discordant pairs and **n** the number of pairs. The Kendall τ coefficient is defined as:

$$\tau = \frac{\mathbf{C} - \mathbf{D}}{\binom{\mathbf{n}}{2}}$$

where $\binom{n}{2} = \frac{n(n-1)}{2}$ is the binomial coefficient for the number of ways to choose two items from n items. The denominator is the total number of pair combinations, so $\tau \in [-1, 1]$. We can remark that if X and Y are totally the same then $\tau = 1$ and on the opposite if X and Y are totally different, $\tau = -1$. If $\tau = 0$, then X and Y are independent. Kendall's rank coefficient can also be calculated as:

$$\tau = \frac{2}{n(n-1)} \sum_{i<j} sgn(x_i - x_j) sgn(y_i - y_j)$$

where $sgn(x)$ is the sign of x. Also, it is really important to notice that we have not mentio-ned anything about the case where $x_i = x_j$ or $y_i = y_j$. This is the situation of a tie. In this model, the Kendall's Tau coefficient used, does not take into consideration the ties.

In this model Kendall's rank coefficient is the similarity checking metric, for a number of reasons. The Spearman's coefficient is a metric that is mostly used and has impact on data sets that show linearity [22]. Also Andrew Gilpin notes in the paper ”Table for conversion of Kendall's Tau to Spearman's Rho within the context measures of magnitude of effect for meta-analysis” [7], that Kendall's ρ approaches a normal distribution more rapidly than Spearman's, in cases where N, the data set size tends to be large. Our model is designed to be used in big data problems, so this is a crucial factor. Finally, we conducted many experiments, checking both of these metrics and Kendall's rank coefficient outperformed in most of the cases Spearman's coefficient.

5.2 Optimization

Similarity checking is a time-consuming process, which is why a hashing schema is re-quired. In order to speed up this component of the model, we added a a Bloom Filter for avoiding redundant comparisons and a thread pool for parallel similarity checking between the buckets.

5.2.1 Bloom Filter

Bloom filter, is a probabilistic data structure and generally it is used to test whether an element is a member of a set. It is employed in our model to prevent performing redundant comparisons. Remember that we hash with the WTA algorithm more than once, thus we may compare the same pair of items multiple times. To avoid this, we included a bloom filter that saves all the pairs that have been compared for similarity. Briefly, before every similarity check between a pair of objects, we search to the bloom filter if these objects have already been compared. If this is true, this check is avoided and it proceeds to the next pairs comparison. On the other hand, if this query returns false, the pair is compared, and the bloom filter for this pair is updated.

Keep in mind though, that bloom filter is a probabilistic data structure. False positive matches are possible, but false negatives are not. In other words, a query returns either ”possibly in set” or ”definitely not in set”. However, we use this structure because it is exceptionally quick, with a search time of O(1), and it does not require a lot of memory.

5.2.2 Concurrent Similarity Checking with Threads

Every bucket undergoes its own set of checks for similarity. As a result, we may do this check for each bucket in parallel. We decided to use a number of threads for parallelization, with a maximum of 16. Every thread has some shared data structures, such as the table where we save if a pair is similar. Locks were employed to prevent race conditions and errors when updating these data structures. The time spent on the similarity checking phase was greatly reduced, by almost 30%, as a result of this addition. Remember that this is the most time-consuming stage in our unsupervised learning model.

6. EVALUATION

In this section, we will make a detailed presentation of the model scores to the CORA data set and present its superiority among other applications in the same data set. Our criteria are the four scores Recall, Precision, F1 and Accuracy as well as the time needed in order to make these predictions.

Starting, by evaluating this model upon the CORA data set. We will not only present the results from this data set, but we will also give a quick evaluation of each component of the model separately. We will examine the prototype selection, display the embeddings, observe WTA acceleration, and finally remark on the results and see how the similarity metrics we suggested worked. Keep in mind that in order for our model to work, it must be fine-tuned. The fine-tuning will be done with the help of Optuna [27], a cutting-edge framework. We will go over how we used it and why it is so crucial in our work in a few words. Finally, we will examine the performance of various different ER models and compare them to ours. These models will be created using JedAI [13], another state-of-the-art framework.

6.1 Evaluation based on CORA data set

In all ML applications, at the stage of the development, some data sets are selected for evaluation. All aspects of the model development are rigorously tested using these data sets. CORA data set was one of the data sets that met the requirements of the aims we set in this project. More specifically, this model should evaluate a single data collection (dirty or clean) containing strings and determine which of these strings are identical quickly and accurately. As a result, we will conduct a more thorough analysis of this data set.

CORA is a well known Dirty ER data set, consisted of real-world entity profiles with bibliographical information for scientific papers. In a nutshell, the CORA data collection comprises records with citation information on published articles, including titles, authors, year of publication, and other details. The data set has a respective ”gold” data set that provides information on which records are a match based on the id.

6.1.1 Attribute Analysis

One of the most important steps in unsupervised learning problems is to understand and point out, data sets attributes. For this reason, we gathered all the needed information in the bellow table 6.

Table 6: CORA data set attributes

Attributes	#Count
Data set size	1295 papers
Papers without any duplicates	19 papers
Papers with at least one duplicate	1276 papers
Mean size of a cluster	13 papers
Average string size	164 chars
Minimum string size	38 chars
Maximum string size	366 chars
Median string size	164 chars

Shortly, CORA data set consists of 1295 papers/strings, of which 1276 have at least one other that refer to the same physical paper and 19 of them have no other one same and are unique.

Table 7: CORA data set attribute columns

CORA data set attribute columns	
Attribute	**Details**
id	Unique identifier for every paper
address	Place of paper creation
author	Author/s in full name or in abbreviation
editor	Papers editor
institution	University or research Center
month	Month published
note	Small part of abstract
pages	Number of pages
publisher	Symposium, book, etc
title	Paper tiltle
venue	Scientific area
volume	Volume identifier if it’s part of a book
year	Year published

All of the lengths of the strings are visualized in the diagram 10 below. We should point out that CORA is a dirty ER data set, and as a result, we had to clean the text. This cleaning procedure includes removing punctuation and converting all strings to lowercase.

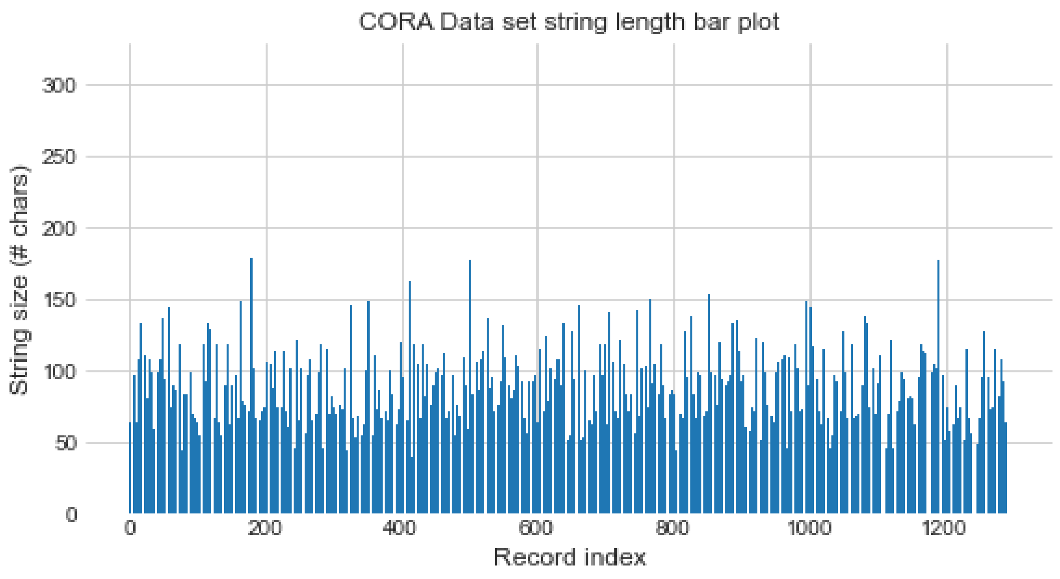


Figure 10: Strings length plot - CORA data set

Another important factor to consider is the features of the data set that will be used to attain the best results. We visualized (Figure 11) the missing ratio per attribute to determine which attributes will be utilized. We found that not all characteristics have all of the information needed for all of the papers, therefore these columns might be misleading. We can now effortlessly notice that more than six attributes have missing ratio greater than 50%. As a result, only the attributes title and author will be used in our experiments because they are the only ones that have no missing data.

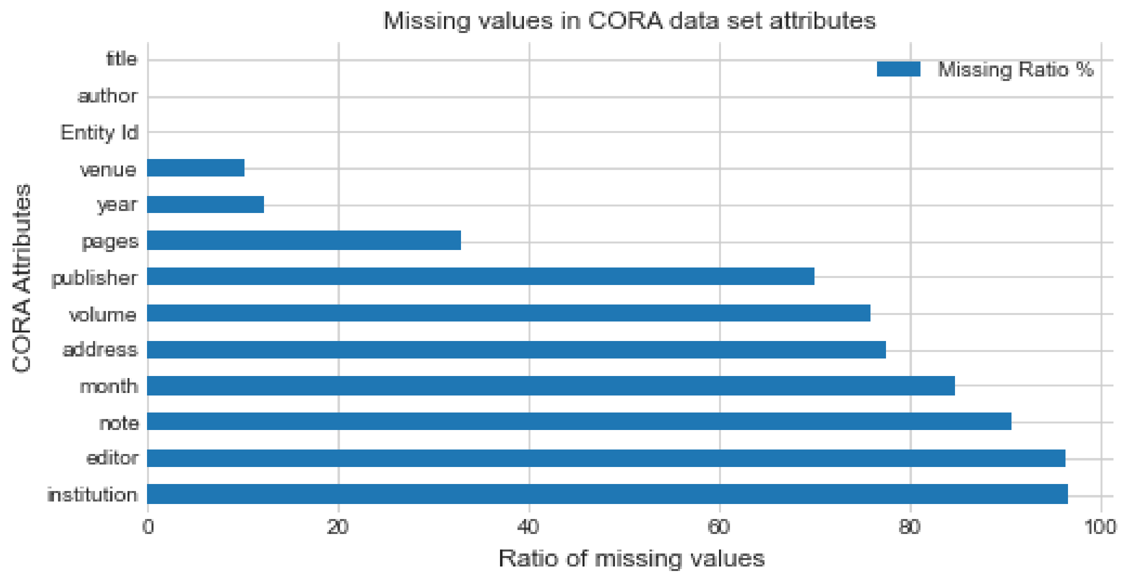


Figure 11: Missing ratio - CORA data set

6.1.2 Selection of string metric

The first step in our process is to choose one of the metrics discussed in Section 2.1. To accomplish so, we examined the first 30 strings of the CORA data set to evaluate how Edit distance, Jaccard, and Euclidean-Jaccard performed. We know that identical papers exist in this selection because of the ”gold” data set and form the groups:

- Group 1: {0, 1, 2, 3, 4, 5, 6, 7}, (red)
- Group 2: {11, 12, 13}, (green)
- Group 3: {16, 17, 15}, (blue)
- Group 4: {18, 19, 20, 21, 22}, (yellow)
- Group 5: {26, 27, 28} (orange)

We created heatmaps based on the distances generated by these metrics. More specifically, we illustrated every distance between all the papers-strings (30x30). A matrix is formed by these distances. This matrix was visualized a heatmap, as shown in Figure 12.

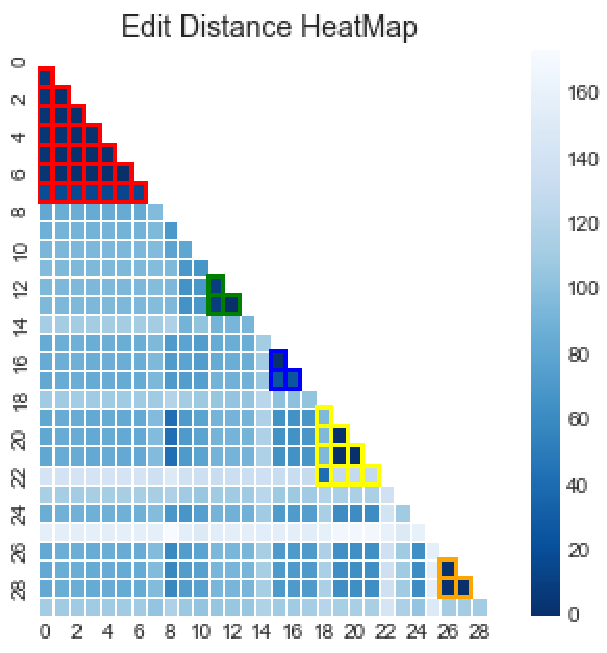


Figure 12: Edit Distance on CORA subset

We made the heatmap lower triangular for visualization purposes as it is symmetric. The goal of this visualization technique is to determine which distance measure and tokenization method are the most effective. The color diversity and intensity are the criterion. We want the same objects to have intense coloring, which implies a significant similarity. The previously specified groups are represented by the heatmap boxes with a colorized contour. Each group corresponds to a distinct color.

At this point, we should evaluate the Edit distance heatmap shown in Figure 12. The first criterion is to study the groups and the distances that have developed between them. We observe a dark coloring between the strings in the groups in almost all the cases. Remember that the distances we are looking at, are all dissimilarity distances, thus the darker colors are mapped to values near to zero. Similar strings are indicated by an edit distance of zero or near to zero. The comparison of pairs 19, 20, 21, and 22 is also one of our criterion, as this is a tough comparison between true identical strings that Edit distance can not correctly distinguish, as shown by the heatmap.

Another requirement is the color distinction, between similar and different objects. This requirement is not totally met. For example, we can see a minor color variation between the green group's (dark blue color) strings and the strings 10 to 12, which are similarly relatively dark colored, indicating a high degree of similarity. This observation is a flaw of the edit distance, in this experiment.

The color variation between the strings is the last requirement. The string distance must offer as precise distances as possible, as well as distances that cover the entire spectrum. We want a distance that not only discovers the most similar strings, but also one that is sensitive to little variations in the strings. Edit distance appears to fulfill this criteria. We can see that the strings have several scores, demonstrating that edit distance takes into account even minor modifications.

We next examine the Jaccard and Euclidean-Jaccard distances. In order to test these distances, we must first convert the strings into sets of tokens. As a result, we experimented with the variations between trigrams and bigrams, as well as the tokenization level, which refers to the character or word level. We remind you that these terms were discussed extensively in Section 2.

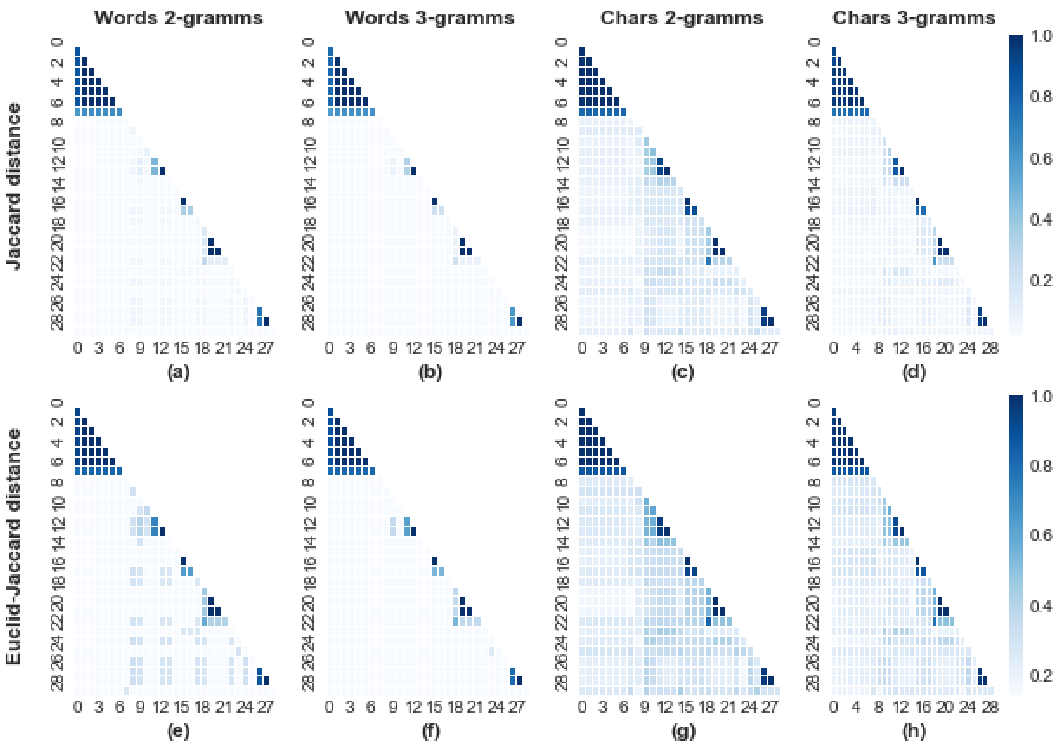


Figure 13: Jaccard with Euclidean-Jaccard comparison on CORA subset

Figure 13 can be used to generate some quite useful comments. At this point we examine, if the first requirement is met, namely, whether the distances between identical strings can be distinguished. We can see that this condition holds true in almost all cases of n-grams, tokenization, and the two metrics. This is due to the dark blue color in all comparisons between strings that belong to the same group. Following, the distances between the strings 19, 20, 21, and 22 are also wick as it can be seen from the color intensity in all the cases except from the plots Figure 13:(e), (f), (g) and (h). These plots have been made using the Euclidean-Jaccard distance, an indication that proves our selection on using this metric as the dissimilarity metric of this framework.

In terms of color difference between similar and dissimilar objects, it is fulfilled in all cases and with higher precision than Edit distance. Figures 13:(a), (b), (e) and (f), depict a significant color difference between the group distances and the strings that do not belong to them, which is a good sign that this criteria is being met.. This big difference in distances, however, is a flaw for the next requirement.

The third requirement, namely the color variance between the strings, is also met, in the plots Figure 13:(c), (d), (g) and (h) made with char-tokenization and the same time fails in the rest plots. We can also observe that the contrast between word-tokenization and char-tokenization is even greater, as the colors between the different strings are nearly white, whereas the colors between similar objects are very dark.

This analysis and visualizations (Figure 12 and 13), sheds a lot more clarity on the string metric selection procedure. We see that Edit Distance does not show the distances between the 30 first strings as evident as Jaccard or Euclidean-Jaccard. Furthermore, char-tokenization seems to perform better, as we can see that the color difference between identical objects is dark while the distances between all the strings have a wide spectrum. To summarize, char-tokenization appears to be more accurate, particularly in Euclidean-Jaccard with 2-grams, which has also resulted in a great separation between strings 19, 20, 21, and 22, which have a difficult comparison, although they refer to the same paper.

For these reasons, we concentrated our CORA data set experiments on 2- and 3-grams and char-tokenization. At the same time, this analysis confirmed our previous assumptions and claims about the Euclidean-Jaccard metric. In this simple example, it became clear that this metric outperforms the other metrics for this task. Remember that the metric used to create the space in the initial idea [29] is Edit distance. This is a major difference between our framework and the one presented in the initial paper.

6.1.3 Hyper-parameter tuning using Optuna

Our model (WinnER) has the bellow five hyper-parameters:

- **Max number of clusters**: Number of maximum clusters that will be created in Prototype Selection phase. We remind you that it is an upper bound and the real number of clusters that will be formed will be less than this parameter.
- **Max dissimilarity distance**: The maximum distance between two strings that can join a cluster. The conditions are met if the distances between the strings being compared are smaller than this threshold.
- **Window size or K**: This is the WTA hashing hyper-parameter. After the permutation, K is the number of elements that will be selected.
- **Number of hashings**: Number of WTA-hashing executions.
- **Similarity threshold**: Threshold of similarity checking phase. If the similarity metric between two rank-ordered vectors is greater than this threshold then these elements are considered same.

One of the most critical steps for a machine learning model to offer high performance is hyper-parameter optimization. Optuna [27] is a well-known Python library for hyper-parameter optimization that is simple to use and well-designed. It supports a wide range of optimization techniques. The internal implementations of Optuna are described in this article, with a focus on the software components.

Optuna operates in a straightforward yet effective manner. It takes an interval for each hyper-parameter and generates a random number within that interval. It also requires one or more scores, as well as whether they should be maximized or minimized. It starts the pruning after a number of trials, according to the official documentation, in order to acquire the best result for the target score.

6.1.4 CORA best scores

There are four scores to determine if the predictions are correct or incorrect. But first, we shall define certain terms related to predictions. The list below displays every possible combination of a successful or unsuccessful prediction.

- **True Negative (TN)**: the case was negative and predicted negative
- **True Positive (TP)**: the case was positive and predicted positive
- **False Negative (FN)**: the case was positive but predicted negative
- **False Positive (FP)**: the case was negative but predicted positive

By utilizing these four measurements, we can use the scores Accuracy, Precision, Recall and F1-score. **Accuracy** is the fraction of predictions our model got right.

$$\textbf{Accuracy} = \frac{\textbf{TP} + \textbf{TN}}{\textbf{TP} + \textbf{FP} + \textbf{TN} + \textbf{FN}}$$

Precision (accuracy of positive predictions) refers to a classifier's ability to avoid labeling a negative occurrence as positive. It is calculated as the ratio of true positives to the sum of true positives and false positives for each class.

$$\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FP}}$$

The capacity of a classifier to detect all positive cases is known as **Recall** (fraction of positives properly identified). The ratio of true positives to the sum of true positives and false negatives is the definition for each class.

$$\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FN}}$$

The **F1-score** is a weighted harmonic mean of precision and recall, with 1.0 being the highest and 0.0 being the lowest. F1 scores are lower than accuracy measurements because they factor on precision and recall.

$$\textbf{F}1 - \textbf{score} = \frac{\textbf{2*(Recall * Precision)}}{\textbf{Recall + Precision}}$$

In order to find the best scores, we did an ”optuna search”. Apart from running our model, Optuna also provides us with a number of plots illustrating the impact of hyper-parameter tuning. We execute each combination of tokenization, ngrams, and metric a hundred times (Jaccard distance and Euclidean-Jaccard distance). Using Euclidean-Jaccard distance with 3-grams (or 2-grams) and character tokenization has the best results so far. This optuna study's goal is to maximize the F1-score. Remember F1-score or F-Measure, is the score that evaluates both the positives and negatives that predicted true or false, in contrast with Recall or Precision. Having a high-score on F1-score in CORA data set is difficult. This happens due to the fact that the number of comparisons between similar strings is significantly lower than the number of comparisons between different strings. Hence, Recall is also a score that is really important in our experiments. Remember that recall refers to a classifier's ability to discover positive cases, or similar strings in our situation. F1-score is referenced as the Objective Value in the plots below. We can examine how the Optuna framework evaluated our model in each trial on F1-score in the Figure 14.

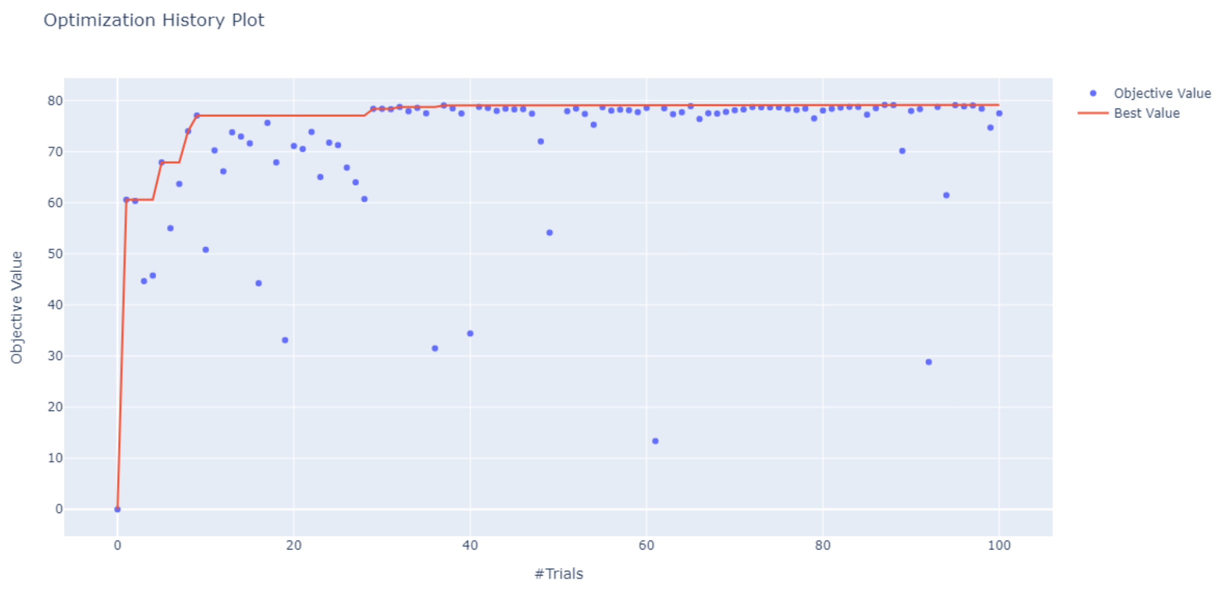
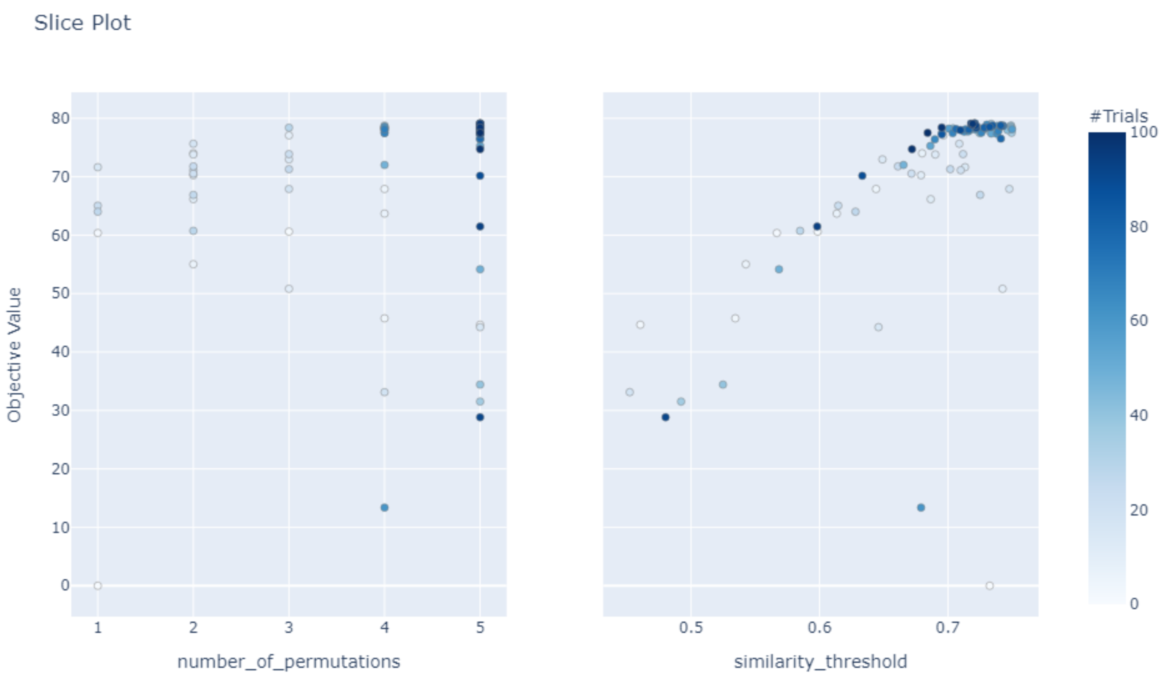


Figure 14: Optuna Optimization History Plot

It is obvious that the maximum is close to 80% and that most of the trials finished with a score greater than 70%. Aim of the ”optuna search”, is to achieve the highest F-Measure by fine-tuning our models five hyper-parameters, as shown in the following plots of Figure 15 and 16, respectively. The darker color denotes higher F1-score.

Figure 15 can be used to drown a number of remarks. These visualizations were really helpful in the fine-tuning phase of the model. To begin, we can see that Optuna’s search is incredibly effective, as seen by the color in the scatter plots, which demonstrate that as the trials progressed, the Objective value (F-Measure) increased. At this stage, we will examine at each hyper-parameter separately, using Figure 15 as a guide. Starting with the number of permutations (i.e. number of hashings), we see that with just one permutation, we can acquire an F-Measure of more than 70%. Though, as predicted, the best results were obtained with three or more hashings.

Following, the similarity threshold, seems to perform the best around 0.7 and 0.75. This is evident from the fact that most trials used values from this range to achieve the best result. The maximum dissimilarity distance and the maximum number of clusters are two measures that are completely dependent on the nature of the data set. Both of these parameters require fine-tuning through experimentation in order to achieve the optimum results. Finally, for values between 5 and 10, the window size (i.e. K in WTA hashing) yields the best results. This is also something that should be expected, as it is stated in the WTA original publication [17] that this hashing algorithm works best for window sizes smaller than 10.



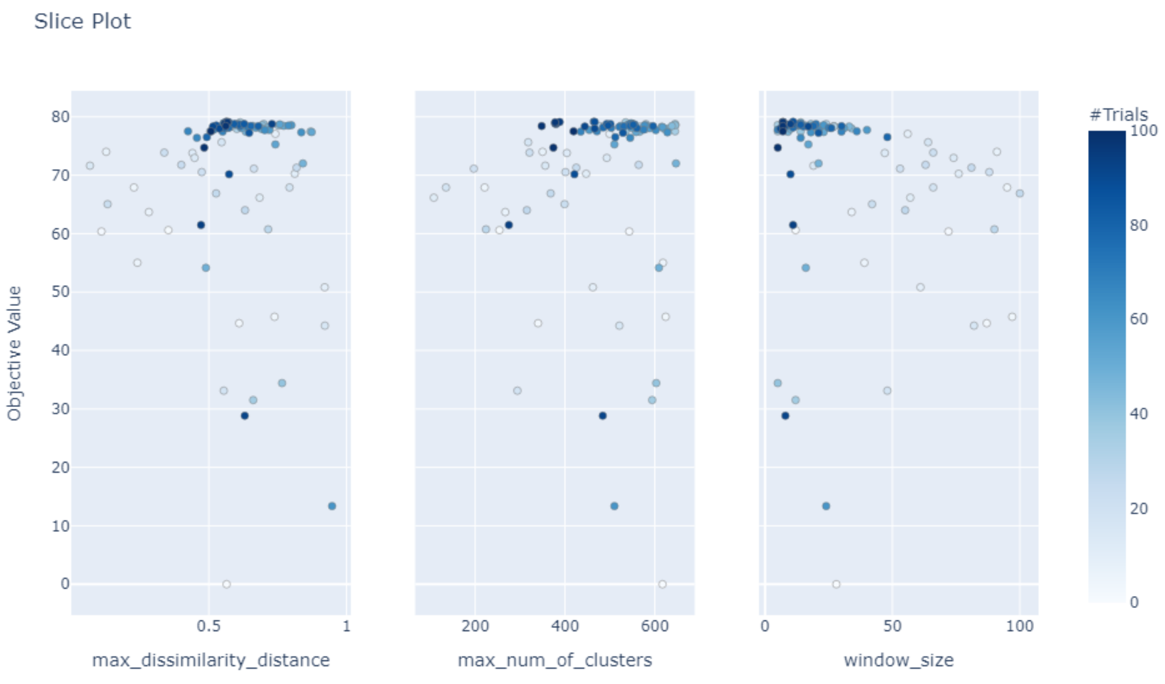


Figure 15: Hyper-parameters values on CORA subset

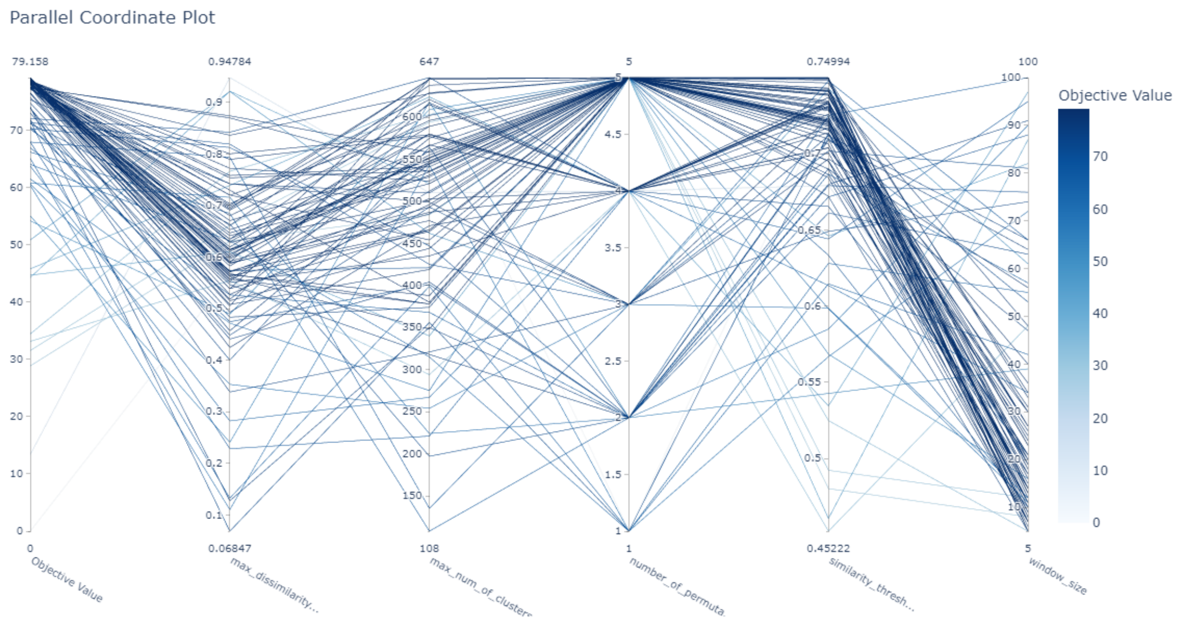


Figure 16: Hyper-parameters in comparison with F-Measure on CORA subset

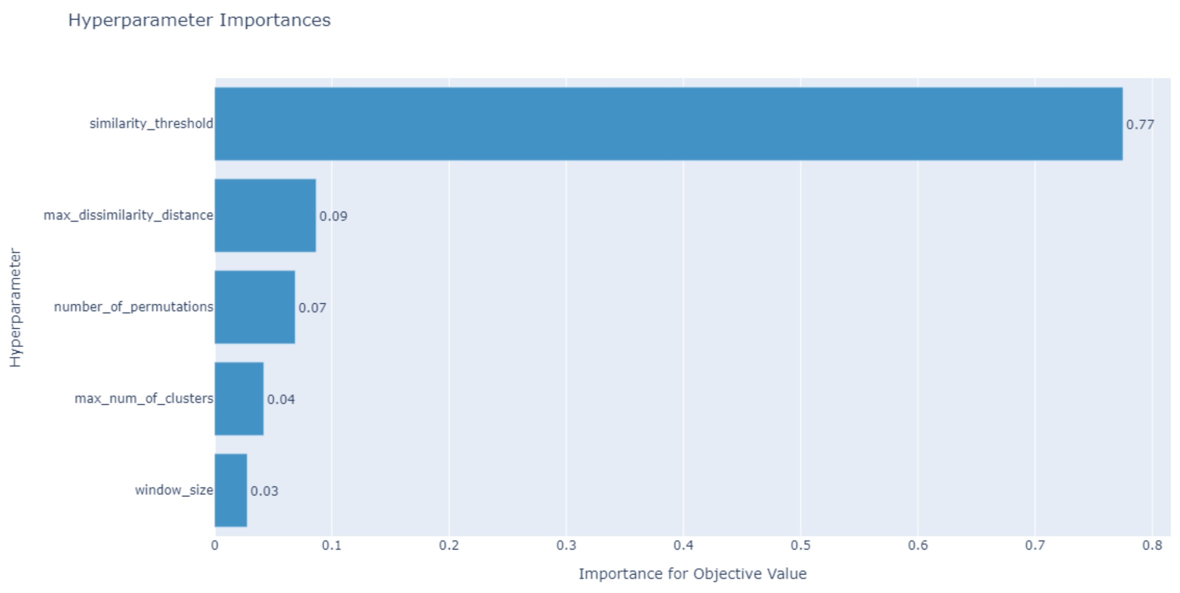
In order for Optuna to perform this fine-tuning, we set specific boundaries for each hyper-parameter, as shown in Table 8. Optuna tested our model in multiple random values between the minimum and the maximum value that we defined. In the figures 15 and 16, we can see which of these values found to increase the objective score. Figure 16, is a great visualization since it shows the overall fine-tuning as well as how each hyper-parameter affects the others and the final score. From this figure, we can note that by increasing the number of permutations (i.e number of hashings) we get better F1-scores and that is expected as increasing the number of hashing increases also the number of comparisons and hence the probability of finding more similar strings. Furthermore, we can see that with a similarity threshold near to 0.7, we receive greater F1-scores. It would also be beneficial to utilize the recall score instead of the F1 as the Objective value, as identifying similar pairs in this data set is difficult. However, boosting the recall score lowers the accuracy and precision, making our model less accurate in these aspects. For this reason, we decided to use F1 score as the target score, because F1-score is the score that takes into consideration both precision and recall.

The upper bound of the number of clusters has no effect on F-Measure (i.e F1-score), as we can get good results with multiple values. We can also see that the max number of clusters gives the best results between the interval of 400 and 650. Note here that this number is something less than the data set size divided by two.

Table 8: Optuna hyper-parameter intervals

Hyper-Parameter	Min	Max
Max number of clusters	200	700
Max dissimilarity distance	0.1	0.9
Window size or K	5	100
Number of hashings	1	5
Similarity threshold	0.5	0.75

Optuna further provides us with two additional visuals that highlight the influence of each hyper-parameter on both F1 and execution time. This is depicted in Figure 17. We can observe that the similarity threshold has the most impact on increasing F1-score, while the window size and maximum number of clusters have little impact. The prior parameter, on the other hand, has no effect on the model's execution time. The second plot of the same figure shows how choosing the ”wrong” window size can lead to a significant increase in execution time. This is also expected as selecting the window size, affects the size of the buckets that will be formed from the WTA hashing. A window size greater that 30, causes less in number and greater in size buckets. As a result, having bigger buckets leads to greater number of total comparisons and hence greater time of execution. The same remark holds for number of permutations hyper-parameter, as an increase on the number of permutations, increases also the number of total comparisons and results in a considerable impact on the execution time.



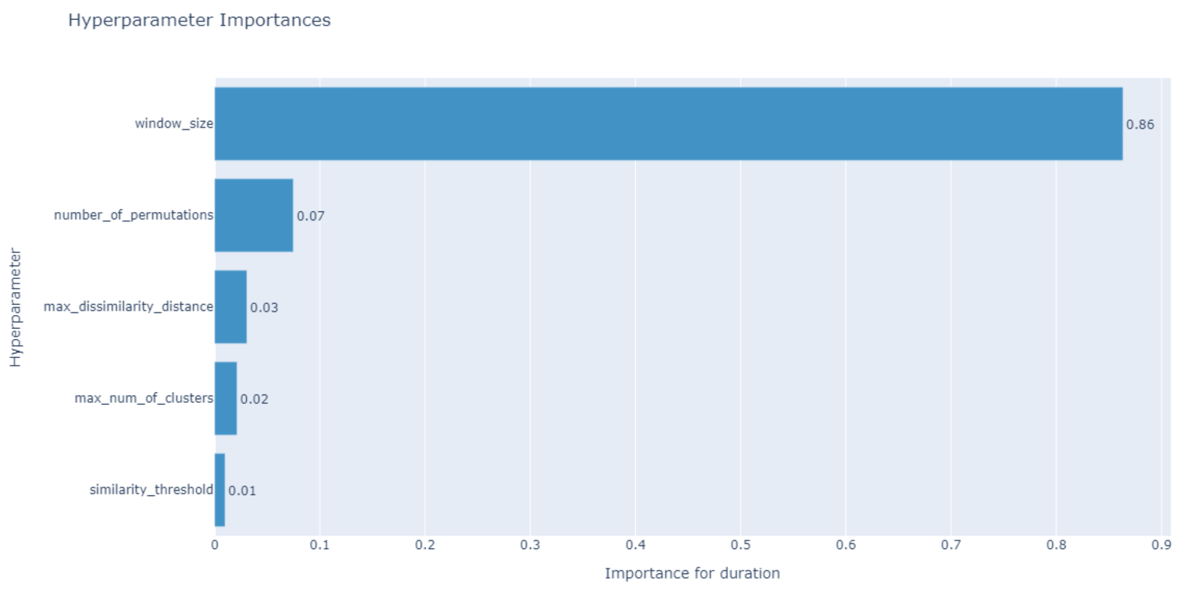


Figure 17: Importance of hyper-parameters for Recall and Execution time

From all these trials we also saved a number of measurements, in order to make a data analysis and have a better view on the models performance. For example for each execution made by Optuna we gathered data about the number of prototypes selected, the average bucket size, the execution time of each part and also the total number of comparisons, successful or not. These measurements among others helped us verify our theoretical assumptions, as well as made us re-design certain parts of this model. To better grasp these measurements, we created a number of plots. Figure 18 shows the impact of the two hyper-parameters of the prototype selection phase, in the resulting number of prototypes selected.

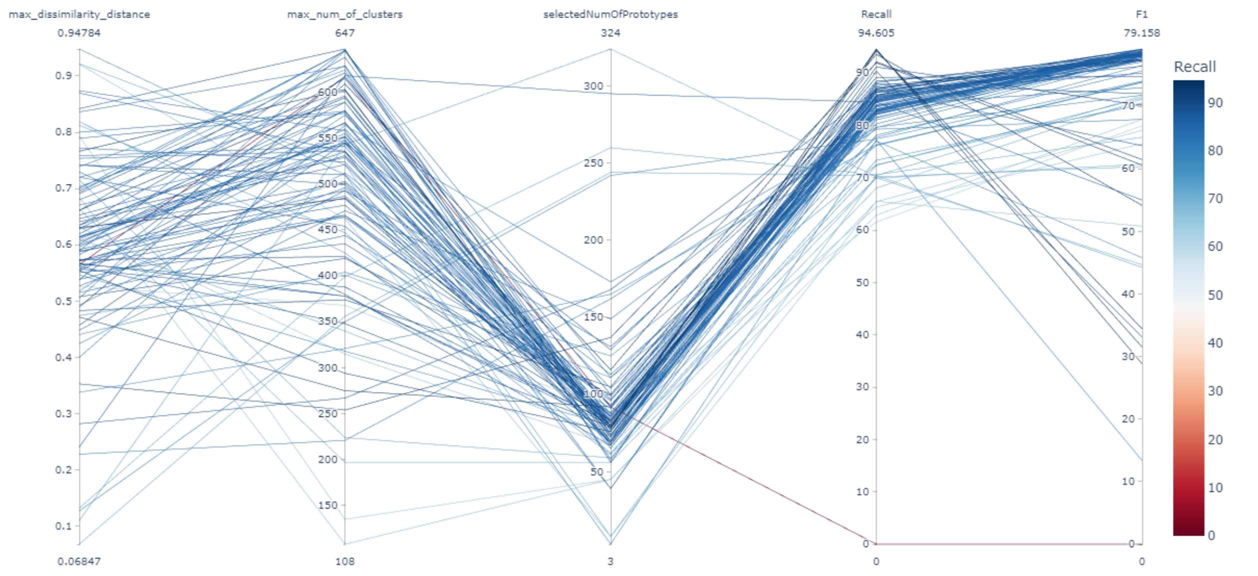


Figure 18: Importance of prototype selection hyper-parameters

It is obvious that a number of prototypes close to 100 results in higher scores. Keep in mind that 100 prototypes for a data set of approximately 1200 strings means that we set a prototype for every 12 strings. In a similar vein, we can evaluate the hashing schema.

In the Figure 19, we can observe the two hyper-parameters of the WTA hashing algorithm in accordance with the average size of the buckets formed. All of these metrics are represented graphically, together with the F1, recall, accuracy score, and time spent on the similarity checking phase. We need to see if the hashing boosted the similarity checking step at all, and if so, at what values of hyper-parameters.

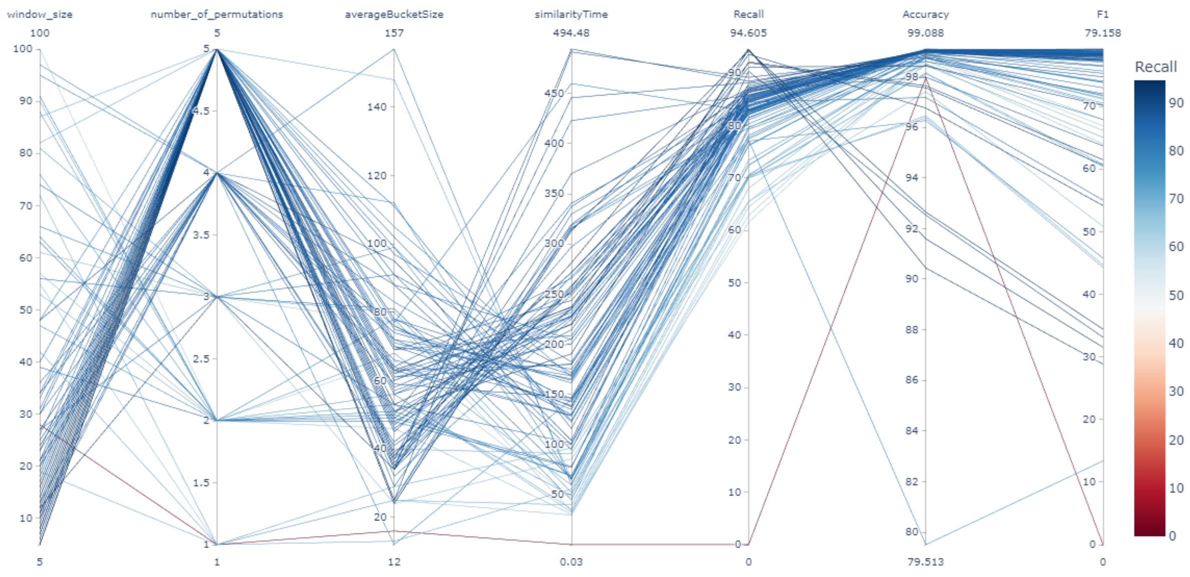


Figure 19: WTA Hashing hyper-parameters evaluation

A number of observations can be drawn from the Figure 19. To begin with, the window size (the length of the vector from which WTA obtains the hash) does not have to be high. The best results appear to be achieved with a window size of 5 to 40. Also, increasing the number of permutations (number of hashings) improves the scores, keeps the average bucket size above 100, but can result in a long similarity checking time, as expected. Another obvious point is that by having as the Objective Value the F1-score, we increase the recall score, while also having accuracy above 99%. Another handy plot we have created is one that shows the number of successful comparisons (Figure 20). This is one of the model's most important features. The most difficult aspect of ER problems is keeping the number of comparisons limited while maintaining high scores.

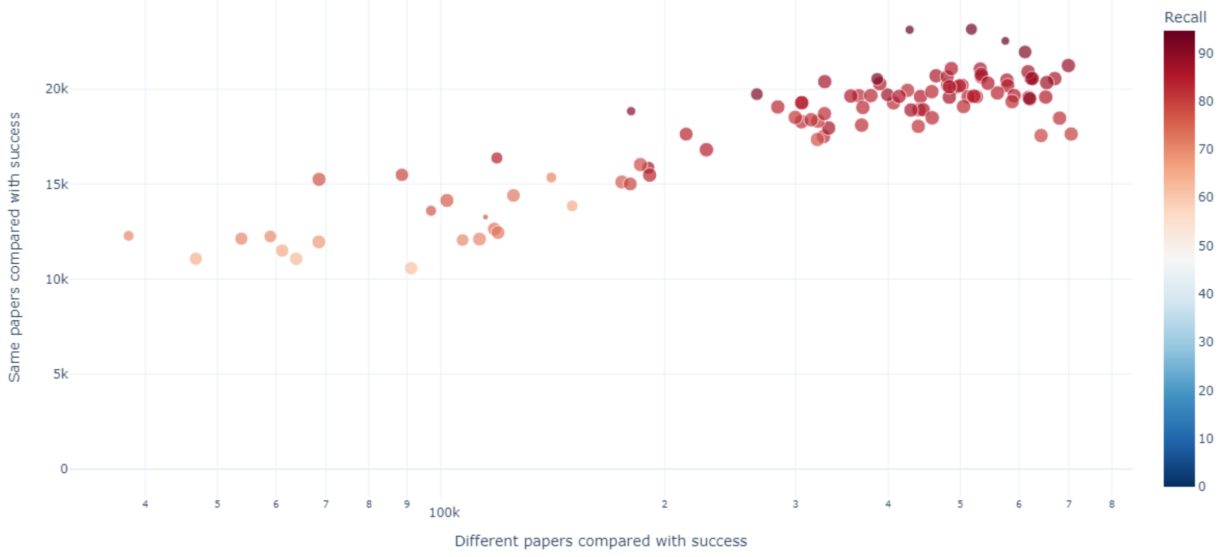


Figure 20: Successful comparisons evaluation

The F1-score improves as the number of comparisons increase. This is clear from the prior plot. Even with less than 100.000 comparisons, we can achieve good results. This is a great outcome, as we minimize the number of comparisons to around 70% while maintaining satisfactory results. At this point we should present the best scores achieved in CORA data set. Table 9 shows the best trials, not only on F-Measure but in all ways.

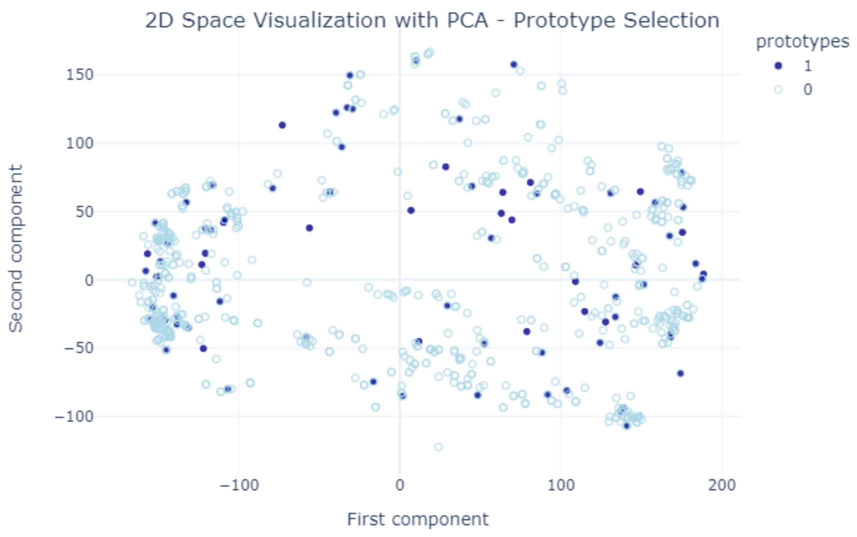
Table 9: Best scores on CORA

Trial id	Recall	F1	Precision	Accuracy	Total Time (m)
97	87.02	79.05	72.43	99.06	06:15
87	86.91	79.16	72.68	99.06	04:35
95	86.91	79.11	72.60	99.06	05:22
88	86.30	79.12	73.05	99.07	04:18
96	86.28	78.88	72.64	99.05	05:40
37	84.71	79.05	74.11	99.08	03:52
65	84.56	78.92	73.98	99.07	03:57
84	84.41	78.77	73.83	99.07	04:32
83	83.95	78.78	74.21	99.07	05:48
41	82.65	78.77	75.24	99.09	02:50

Lets focus on the most interesting trial, trial 37, as in this trial we get great results in about 4 minutes. We achieved a score of F-Measure near 80% and recall around 85%. For these experiments we did not utilize a GPU or numerous cores of high performance CPUs. It will be much interesting if we take a deeper look to the best trial.

6.1.5 Prototype selection

In this part, we should examine our models performance in the prototype selection phase separately. Firstly, the time needed of the algorithm we used to select 87 strings from a data set of approximately 1200 strings is around 2 seconds, which is amazingly good. However we do not only need to have a quick selection but also a selection of prototypes with a distribution similar to the data set. In order to examine this, we created some visualizations, by performing some really famous dimension reduction algorithms. PCA and MDS are the ones used as mentioned also in Section 2. We created two- and three-dimension plots in order get a view of the selection. Figure 21 depicts the prototypes selected among all the strings in a two-dimension space.



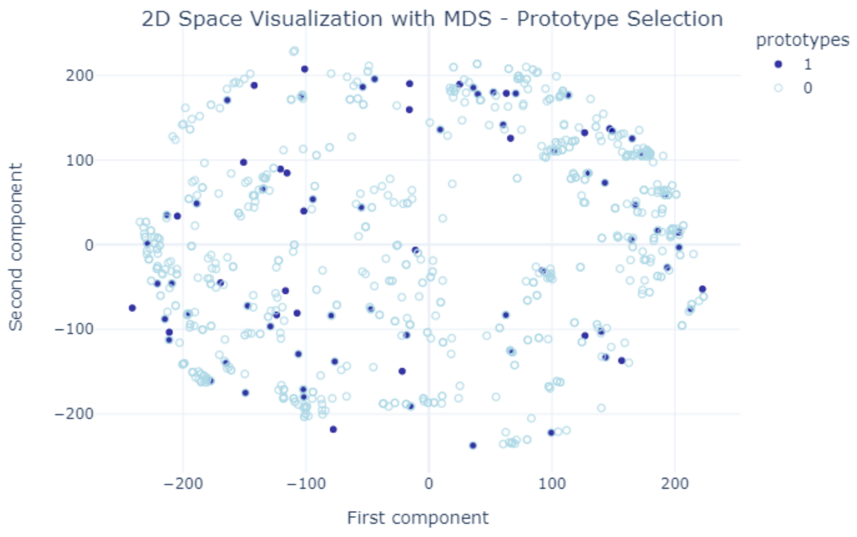


Figure 21: 2-Dimension Prototype Selection with PCA and MDS

We can observe that in this trial, we have a really good prototype selection, as the prototypes selected span all the string space. The above findings denote a prototype distribution, close to the one of the initial data set. Keep in mind that PCA and MDS are methods for reducing the number of dimensions in a data set while keeping the most critical information. This can be done, by projecting high-dimensional data linearly onto the principal components of variation (PC). However, because to the dimension reduction, a lot of information may be lost. Doing a dimension reduction is very beneficial as it is the only way to visualize a high-dimension space and gain a sense of the data set distribution. Furthermore, increasing the number of principal components used, results in less lost information. As a result, we made 3-dimension plots in Figure 22, using both PCA and MDS. In all of the figures, prototypes chosen distribution is seems very accurate, when compared to the data set's initial distribution.

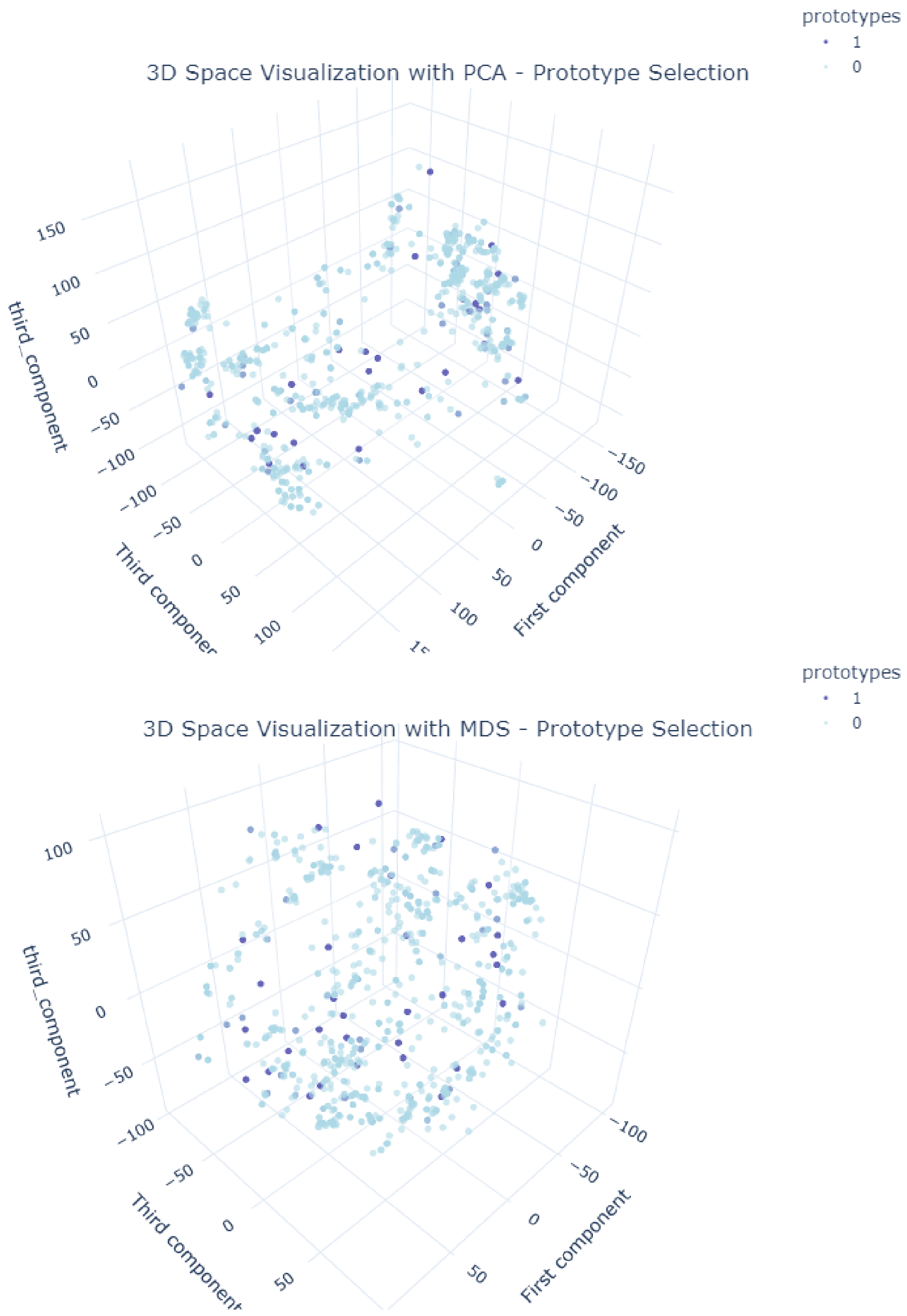


Figure 22: 3-Dimension Prototype Selection with PCA and MDS

It is also interesting to create a heatmap out of the dissimilarity distances between the prototypes chosen. In our case, the dissimilarity distance is the Euclidean-Jaccard distance. Figure 23 illustrates this heatmap. As can be seen from the color bar, we have made a pretty solid prototype selection. The prototypes are nearly all 90 percent different from one another. This is critical because we require prototypes that are as unique and different as possible. Prototype selection algorithm aims in selecting the most distinguishing strings. If these assumptions are true, the resulting space formed, is rich and the embeddings created will ”describe” each string in a unique way. And if we have ”detailed” and precise embeddings then it is most likely to get high scores.

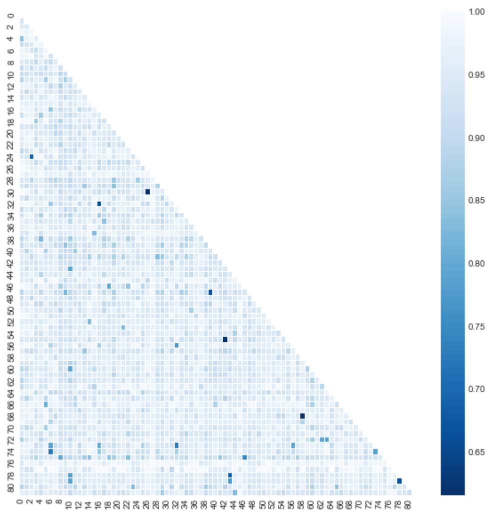


Figure 23: Dissimilarity between Prototypes

It is indeed critical to note at this point that the prototype selection optimization we performed appears to be functioning and producing quite satisfying prototypes. We obviously do not have prototypes that are similar. As we have mentioned in Section 2, there is a metric called Maximum Mean Discrepancy (MMD), that measures the discrepancy between two distributions. This metric provides us with an indication of the difference between the distribution of the prototypes and the initial distribution of the data set. Remember that the closer this metric is to zero, the closer are the distributions. This statistic yielded roughly 0.0114 in this trial. This is a great score as it is very close to zero and as we mentioned before this is the goal.

Overall, there are also some disadvantages, the most significant of which is that this algorithm must be fine-tuned in order to achieve these results. In a nutshell, the prototype selection provided a group of unique and different prototypes in a very short time for the best trial. K-Means, DBSCAN, and other clustering algorithms with the same design are slower and require fine-tuning as well.

6.2 Comparison with other ER models using JedAI

In order to evaluate the model we have developed, it is much needed to compare it with other existing models, developed for Entity Resolution. In this study, we utilized a state-of-the-art toolkit named JedAI. JedAI [13], is a toolkit for Entity Resolution that implements numerous state-of-the-art, domain-independent methods, and provides an intuitive Graphical User Interface that can be used for ER experiments and evaluation on data sets.

JedAI ER workflow consists of eight steps.

1. **Data reading**
2. **Schema Clustering** groups together syntactically (not semantically) similar attributes. This can improve the performance of all workflow steps.
3. **Block Building** clusters entities into overlapping blocks in a lazy manner that relies on unsupervised blocking keys: every token in an attribute value forms a key. Blocks are then extracted, based on its equality or on its similarity with other keys.
4. **Block Cleaning** aims to clean a set of overlapping blocks from unnecessary comparisons, which can be either redundant (i.e., repeated) or superfluous (i.e., between non-matching entities). Its methods operate on the coarse level of individual blocks or entities.
5. Similar to Block Cleaning, **Comparison Cleaning** aims to clean a set of blocks from both redundant and superfluous comparisons. Unlike Block Cleaning, its methods operate on the finer granularity of individual comparisons.
6. **Entity Matching** compares pairs of entity profiles, associating every pair with a similarity in [0,1]. Its output comprises the similarity graph, i.e., an undirected, weighted graph where the nodes correspond to entities and the edges connect pairs of compared entities.
7. **Entity Clustering** takes as input the similarity graph produced by Entity Matching and partitions it into a set of equivalence clusters, with every cluster corresponding to a distinct real-world object.
8. **Evaluation**

In this part, we compare our models performance to other well-known ER algorithms. All of these models were created quickly and efficiently with JedAI, which is simple to use and gives consistent results in order to conduct our experiments. Following table summarizes some of the workflows.

Table 10: JedAI results to CORA

Workflow	Block Building	Block Cleaning	Comparison Cleaning	Entity Matching	Entity Clustering	Recall (%)	F-Measure (%)	Precision (%)	Total Time (s)
1	Standard/Token Blocking, LSH SuperBit Blocking	Block-Filtering, Comparison-based Block Purging	-	Profile Matcher	Merge-Center Clustering	79.14	77.54	76.00	09:64
3	Standard/Token Blocking, LSH MinHash	Size-based Block Purging	-	Profile Matcher	Markov Clustering	74.58	65.37	58.19	00:47
3	Q-Grams Blocking	Comparison-based Block Purging	-	Profile Matcher	Correlation Clustering	64.18	69.32	75.35	01:09
4	Standard/Token Blocking	Block-Filtering	Cardinality Node Pruning (CNP-JS)	Profile Matcher	Connected Components Clustering	82.89	80.33	77.93	00:11
5	Standard/Token Blocking	Block-Filtering	Cardinality Node Pruning (CNP-JS)	Profile Matcher*	Connected Components Clustering	81.18	85.23	91.20	00:07

* Profile Matching done in all cases with CHARACTER_BIGRAM_GRAPHS, GRAPH_VALUE_SIMILARITY except from workflow 5 that was conducted with TOKEN_UNIGRAMS_TF_IDF, GENERALIZED_JACCARD_SIMILARITY

Starting from the first three workflows, we can see some popular schemes and algorithms used in ER. F-Measure is between 65% and 75%, which is a an average score for a trivial workflow like a Merge-Center Clustering or the Markov Clustering. These first workflows are presented, in order to compare our model with some basic algorithms. WinnER, with the appropriate tuning, outperforms these jedai-workflows. Workflows 1, 2, and 3 have lower F-Measure and Recall scores than WinnERs, but they have higher Precision score. Remember Precision is the classifiers ability to avoid labeling a negative occurrence as positive.

Another major criterion is the time of execution of each workflow. JedAI is a framework with great time of execution. JedAI runs as a Spring Boot application and has been optimized in all of its parts. So comparing in detail the time of execution with our model is not accurate. We can only compare these times roughly and note that in about 3 to 6 minutes, WinnER gives great results as a Python program that runs in a local machine. More parallelization can be added in our model and its part of the future work.

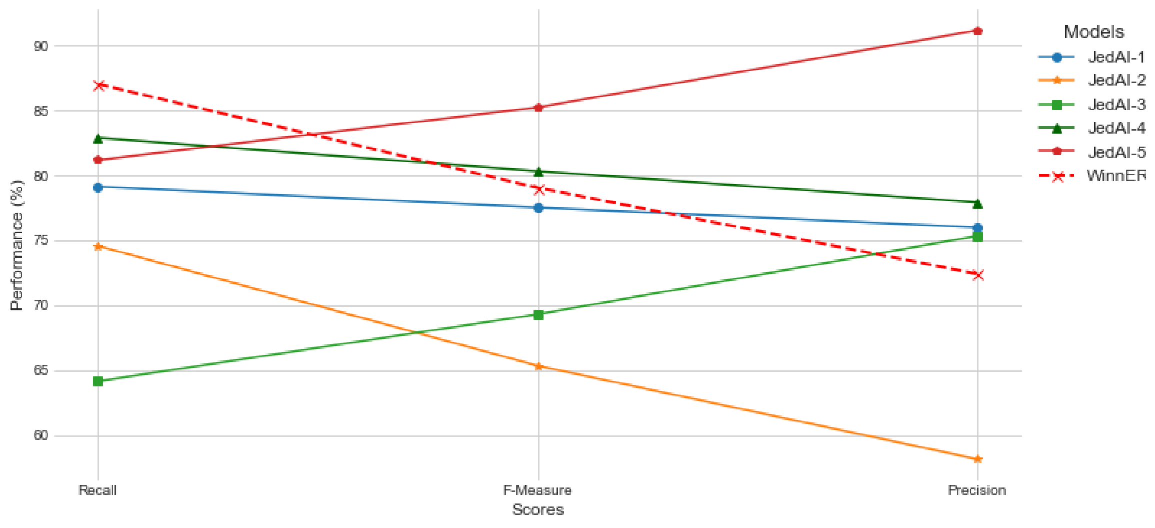


Figure 24: Performance of JedAI workflows compared with WinnER

JedAI-workflow 5 and 6 are among the best we managed to get. These scores can also be be found in the paper *Three-Dimensional Entity Resolution with JedAI* [14]. Figure 24 depicts all the scores shown in the previous table in comparison with our model. WinnER has the greatest Recall score when compared to the other models. However WinnER receives a low Precision score at the same time as it is obvious from the above figure. This comparison demonstrates us the ability of our model to predict with high accuracy the similar pairs, while keeping Precision above 70%.

These results demonstrate that our methodology gives results close to or even better from other well known methods. Without a doubt, this methodology looks promising and results in fairly good results.

7. CONCLUSION

In this study, we present a novel model for tackling ER problems on string data sets. To begin, we provided a Prototype Selection scheme that, as the results show, can result in a set of rigorously selected prototypes as well as in a rich Euclidean and Dissimilarity, Embedding Space. Afterwards, a rank-ordered technique was introduced to avoid the Curse of Dimensionality and to enjoy the advantages that rank-ordered embeddings offer. Following that, we introduced the work's most important component, the Winner-Take-All hashing algorithm, which boosts in time our model while maintaining high Recall scores. By employing this blocking strategy, we were able to reduce the number of comparisons by around 80% and limit the similarity checking phase to only the most similar pairs. In the last section, we presented in much detail, our models performance while also utilized and presented, two state-of-the-art frameworks, Optuna and JedAI. With these two frameworks we presented how our model achieves great recall score and an F-Measure that challenges other successful ER frameworks. To conclude, we developed an end-to-end model that can be used in string ER problems and produce high-performance and robust scores with the less execution time.

TERMINOLOGY TABLE

Ανάλυση οντοτήτων	Entity Resolution
Μηχανική Μάθηση	Machine Learning

ABBREVIATIONS, ACRONYMS

ER	Entity Resolution
ML	Machine Learning
VO	Vantage Objects
WTA	Winner Take All
PCA	Principal Component Analysis
MDS	Multi Dimensional Scaling
LSH	Locality Sensitive Hashing
MMD	Maximum Mean Discrepancy
AI	Artificial Intelligence

REFERENCES

[1] A. Z. Broder. 1997. On the resemblance and containment of documents, Compression and Complexity of Sequences: Proceedings. 1997

[2] A. Mülle, (1997). Integral Probability Metrics and Their Generating Classes of Functions. Advances in Applied Probability, 29(2), 429–443. https://doi.org/10.2307/1428011

[3] B. Chen, A. Shrivastava: Revisiting Winner Take All (WTA) Hashing for Sparse Datasets.

[4] Bellman R.E. Adaptive Control Processes. in: Princeton University Press, Princeton, NJ, 1961.

[5] D. Karapiperis, A. Gkoulalas-Divanis and V. Verykios. 2021. MultiBlock: A Scalable Iterative Approach for Progressive Entity Resolution. In 2021 IEEE International Conference on Big Data (Big Data). 10.1109.

[6] D. Critchlow. 1985. Metric Methods for Analyzing Partially Ranked Data. Springer-Verlag.

[7] Gilpin, A. R. (1993). Table for conversion of Kendall's Tau to Spearman's Rho within the context measures of magnitude of effect for meta-analysis. Educational and Psychological Measurement, 53(1), 87-92.

[8] G. Navarro. 2001. A guided tour to approximate string matching. in: ACM Computing Surveys. 33 (1): 31–88. 2001.

[9] G. Papadakis, D. Skoutas, E. Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. ACM Comput. Surv. 53, 2.

[10] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederee, W. Nejdl, A blocking framework for entity resolution in highly heterogeneous information spaces, IEEE TKDE 25 (12) (2013) 2665–2682.

[11] G. Papadakis, G. Papastefanatos, T. Palpanas, M. Koubarakis, Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking, in: EDBT, 2016, pp. 221–232.

[12] G. Papadakis, E. Ioannou, C. Niederee, T. Palpanas, W. Nejdl, Eliminating the redundancy in blocking-based entity resolution methods, in: JCDL, 2011, pp. 85–94.

[13] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas and M. Koubarakis, The return of JedAI: End-to-End Entity Resolution for Structured and Semi-Structured Data, in: VLDB, 2018

[14] G. Papadakis, G. Mandilaras, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, M. Koubarakis, Three-Dimensional Entity Resolution with JedAI, 2020

[15] J. Vleugels and R. C. Veltkamp. 2002. Efficient image retrieval through vantage objects. Pattern Recognition, 35(1):69 – 80.

[16] J. C. Gower P. Legendre: Metric and Euclidean properties of dissimilarity coefficients, Journal of Classification volume 3, pages5–48.

[17] J. Yagnik, D. Strelow, D. A. Ross, and R. Lin. 2011. The power of comparative reasoning. In 2011 International Conference on Computer Vision. 2431–2438.

[18] K. Pearson. 1895. Notes on regression and inheritance in the case of two parents. in: Proceedings of the Royal Society of London. 1895.

[19] L. A. Goodman, Kruskal, William H. (1954). Measures of Association for Cross Classifications. Journal of the American Statistical Association. 1954

[20] M. Kendall, 1938. A New Measure of Rank Correlation. in: Biometrika. 1938.

[21] M. Kendall. 1970. Rank Correlation Methods (4th ed.). Griffin, London.

[22] M. Kendall and J.D. Gibbons, (1990) Rank Correlation Methods. 5th Edition. 1990.

[23] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, DeepER - Deep Entity Resolution in CoRR, vol. abs/1710.00597, 2017.

[24] R. H. Somers, (1962). A new asymmetric measure of association for ordinal variables. American Sociological Review. 1962

[25] Robert P.W. Duin and Elżbieta Pękalska. 2012. The dissimilarity space: Bridging structural and statistical pattern recognition. in: Pattern Recognition Letters. 33(7):826-832.

[26] S. Edelman. 1999. Representation and Recognition in Vision. MIT Press.

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2623–2631.

[28] Tanimoto TT. 1958. An Elementary Mathematical theory of Classification and Prediction. in: Internal IBM Technical Report. 1957.

[29] V. Verykios and D. Karapiperis. 2021. Entity Resolution in Dissimilarity Spaces. in: 25th Pan-Hellenic Conference on Informatics.

[30] V. Verykios, D. Karapiperis and P. Tsompanopoulou, 2021. Embedding Strings in Euclidean Spaces for Highly Efficient Entity Resolution, *Unpublished paper*

[31] Wayne W. Daniel, (1990). Spearman rank correlation coefficient. Applied Nonparametric Statistics (2nd ed.). 1990