International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 14(2022) pp. 374-385

Submitted: 21 Feb, 2022; Accepted: 25 Mar, 2022; Publish: 4 July, 2022

Detecting near duplicate dataset with machine learning

Marc Chevallier[1], Nicoleta Rogovschi[1], Faouzi Boufarès[1], Nistor Grozavu[1] and Charly Clairmont[2]

[1]LIPN Laboratory, Sorbonne Paris Nord University
99 Av. Jean Baptiste Clément, 93430 Villetaneuse, France
mchevallier@lipn.univ-paris13.fr nicoleta.rogovschi@lipn.univ-paris13.fr
faouzi.boufares@lipn.univ-paris13.fr
nistor.Grozavu@lipn.univ-paris13.fr

[2]Synaltic
24 Rue de l'Église, 94300 Vincennes
cclairmont@synaltic.fr

Abstract: **This paper introduces the concept of near duplicate dataset, a quasi-duplicate version of a dataset. This version has undergone an unknown number of row and column insertions and deletions (modifications on schema and instance). This concepts is interesting for data exploration, data integration and data quality. To formalise these insertions and deletions, two parameters are introduced. Our technique for detecting these quasi-duplicate datasets is based on features extraction and machine learning. This method is original because it does not rely on classical techniques of comparisons between columns but on the comparison of metadata vectors summarising the datasets. In order to train these algorithms, we introduce a method to artificially generate training data. We perform several experiments to evaluate the best parameters to use when creating training data and the performance of several classifiers. In the studied cases, these experiments lead us to an accuracy rate higher than 95%.**

Keywords: Machine Learning ,Entity Resolution, Record Linkage, Data Quality, Data Integration, Data Profiling

I. Introduction

In our post-digital transition world, data has become a valuable asset for any company, regardless of its scale, and great efforts are made to choose how to store it optimally [40]. Data is widely used to make decisions, but most of the data stores in the world lack quality and this lack of quality has a huge cost. In order to improve the quality of data, we believe it is essential to learn about data and the minimum we need to know is whether we already have the data. There is a lot of literature to identify near duplicate documents [1]. In the database world, nearly duplicated datasets (also known as fuzzy duplicated [4]) detection is a sub-domain of record linkage domain [5] also called entity resolution [6, 8]. Figure 1 describe the classical process of detection of near duplicated data. The data is first prepared (enriched, cleaned, standardised) to make it more easily comparable [15]. The second step is to reduce the search space in order to reduce the number of comparisons to be performed [14]. The elements of each candidate pair are then compared at the attribute level using classical or learned similarity measures [13]. The results of these comparisons are then used to determine whether the pair of candidates is duplicated or not. This decision can be made using simple models using distances or more complex ones based on machine learning[7, 12]. Finally, in a last step, the results are clustered in order to obtain a consistency in the results[11].
In this paper, we present an original method using machine learning that does not presuppose any knowledge of the data, nor does it attempts to identify its schema, and summarises each dataset as a metadata vector. This research focuses on semi-structured data and therefore to "instance level modifications"[22]. This study is part of a larger project aiming at extracting metadata in data lake type architectures. This metadata is then used to facilitate data integration, improve data quality and make data exploration more accessible [16].

II. Problem description

We consider a dataset *DS* as an instance *I* of an unknown relational schema *R*, composed of *K* columns.
Each record is called a tuple *t* (a row). Each column *C* in the dataset can be seen as a multiset [2]. $C = (e_1, e_2, .., e_L)$ contains *L* elements. We call *L* the length of the column. Each *e* is an element of a multiset *D* called domain. A near duplicate dataset (NDDS) is defined as a modified version of an existing dataset. In this new version θ columns have been

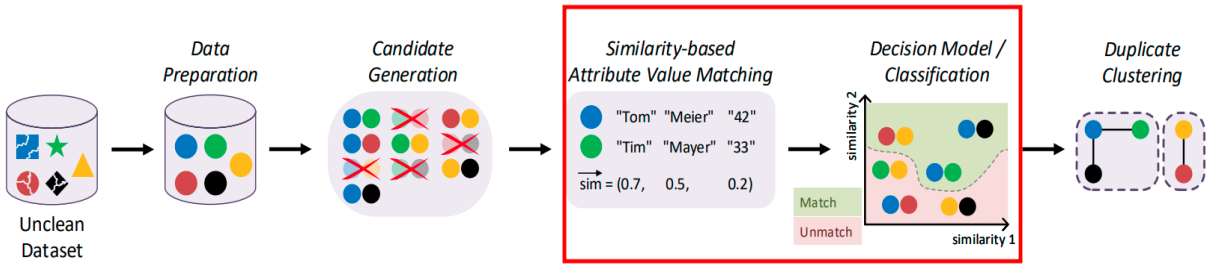

Figure. 1: The five steps of a typical duplicate detection pipeline based on pairwise record comparisons as described by Panse and Naumann [22]. The steps we study in this paper are circled in red

added and ι deleted.
We denote $\zeta = 10 * \frac{\theta+\iota}{K}$
Moreover beta lines have been deleted and gamma lines inserted.
We denote $\alpha = \frac{\beta+\gamma}{L}$.
The problem is to identify if a DS is a near duplicate version of another dataset with unknown α and ζ parameters.

III. Features

In order to identify near duplicates datasets we will use an approach based on machine learning. To do so we have to extract from each dataset a fixed-length vectors of features. A way to perform this extraction is introduced in Sherlock [35] (for Semantic Data Type Detection) but only for columns. We successfully applied this method to near duplicate columns detection [31]. In order to use this method for datasets we concatenate each dataset in a single column. We have introduced several changes to the original features.
First all datasets are preprocessed to be only in lowercase. This transformation allows us to reduce the number of features to extract. Indeed a part of the features depends on the count of the number of each character in each of the cells, each additional character increases the number of features by 6. This choice allows us to reduce the time of extraction of the features as well as to facilitate the learning process without reducing accuracy.
Secondly, we do not use all the features present in Sherlock. First of all, out of the four main types of features extracted in Sherlock, two are from word embedding [28, 27]. These features are not useful in this case and will not be used. Finally in the original characteristics the length of the column to be studied is a particularly important piece of information. Indeed this length is used as a characteristic directly and also intervenes in the calculation of several characteristics. Thus in the use of these features for semantic type recognition using a random forest, the column length is the feature with the most weight in the decision making [35]. However, we do not want the size of the column to intervene in the decision making process. We do not consider that the length of the dataset should be involved because it is very easy to change the length of a dataset using duplicate elements of the dataset to make it appear unduplicated. We therefore chose to keep only the features that do not depend on the column size to form our final feature vector. In order to show the difference between the two approaches we have performed an experiment. We will randomly (uniformly) draw 300 subsets of size 5, 10, 20, 30, 50, 100, 200, 500, 1000, 2000, 5000 in a column of the semantic type Date in YYYY format. We will then extract from each of these subsets the feature vectors using our method as well as Sherlock's method (without word embeding). Then we will compute the norm of these vectors for each of the subset sizes and represent them in figures 3 and 4. Thus, it can be seen that the norms of the feature vectors from the original method grow proportionally to the increase in the size of the columns studied. On the contrary, the norm of the feature vectors resulting from our method shows little variation, these variations being mainly present in the small dimension subsets.

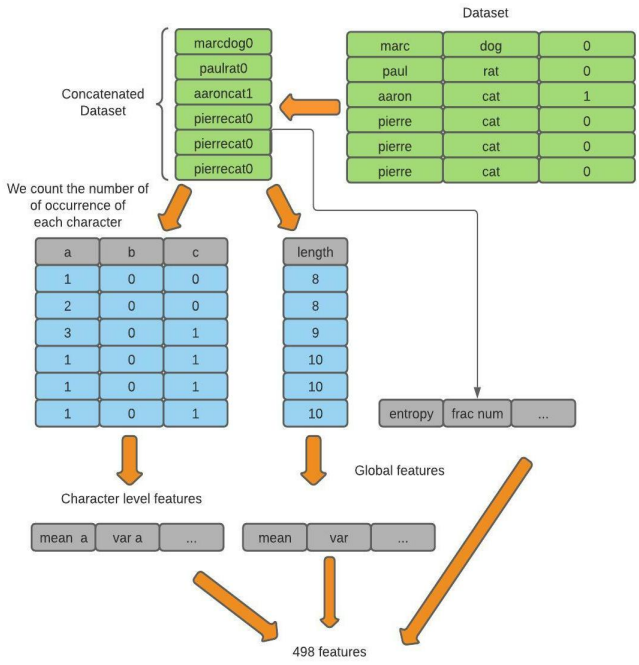

Figure. 2: Features extraction process

We extract two kinds of features from the concatenated dataset. The first one is built with global statistics on the concatenated dataset : entropy, the average number of each type of cells (numerical, special, none), descriptive statistics (min, max ,var, mean, median) on a vector containing the length of

the string in each element (for a total of 22 features).
For the second type of features, we start by counting the number of occurrences of each character in the following list, {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'é', 'è','! ', '¨', '#', '$', '%', '&', '"', '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '¡', '=', '¿', '?', '@', '[', ']', '_', '`', '{', '|', '}', '~', ´, '\x0c'} in each element, so we can form vectors from this count.
Then we extract descriptive statistics (min, max ,var, mean, median, presence or absence of the character, presence or absence of the character in all rows) from these vectors to form our feature vectors (for a total of 476 features). Figure 2 summarises this step. The final result is a vector of 498 features.

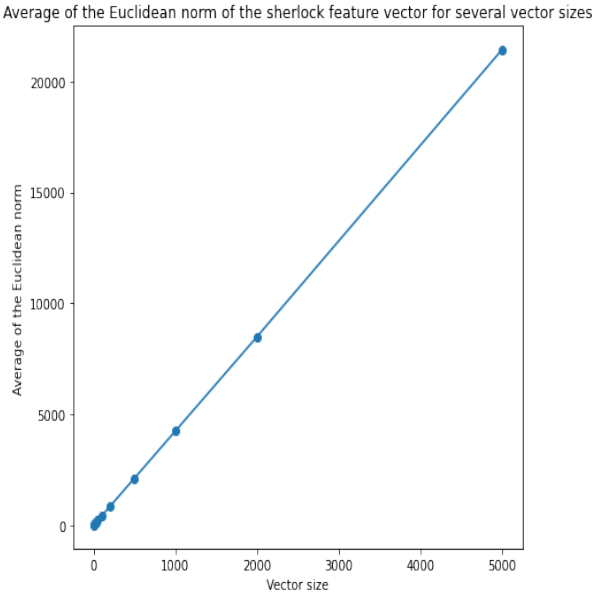

Figure. 3: Average of the Euclidean norm of the sherlock feature vector for several vector sizes

Moreover, in algorithm 3, we will need to merge two feature vectors into one, this can be done in several ways, we will test two. For two vectors v_1, v_2 of the same dimension. The first possibility is :

$$\text{if} |v_1[k]| + |v_2[k]| \neq 0, V[k] = \frac{|v_1[k] - v_2[k]|}{|v_1[k]| + |v_2[k]|} \text{ else} V[k] = 0 \tag{1}$$

The second:

$$V[k] = |v_1[k] - v_2[k]| \tag{2}$$

Formula 1 was originally chosen for the detection of near duplicated columns[31]. Indeed after testing against formula 2 it increased the accuracy obtained by 3% (for a random forest [24]) so we kept it for the detection of near duplicated datasets[32].

An important thing to note about these characteristics is the speed of extraction. Indeed it is a limiting factor in the use

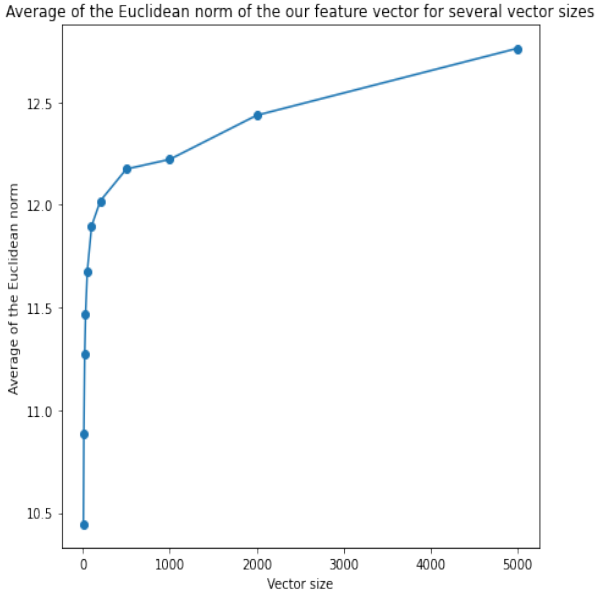

Figure. 4: Average of the Euclidean norm of the our feature vector for several vector sizes

of our method. Beyond 100 000 lines it becomes too long to extract the features. The extraction time as a function of the number of lines is represented in figure 5 and table 1(we use an optimised version of the implementation made in Sherlock which is about twice as fast). The execution times presented are calculated from an average of 50 repetitions of the same calculation.

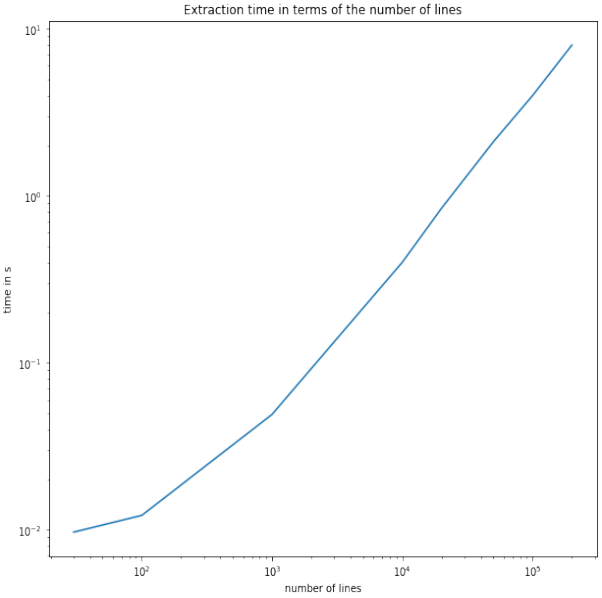

Figure. 5: Extraction time in terms of the number of lines

IV. Algorithms

The algorithm 1 generates a dataset from a set of columns named universe. The dataset built this way counts nbr_col columns and nbr_line lines. The function pick_in_universe(universe) randomly picks a column in the

Table 1: Execution times in terms of the number of lines

nbr of lines	1000	10000	20000	50000	100000	200000
time in s	0.0491	0.40	0.84	2.10	3.98	7.98

universe (and returns the index of this column in the universe). The function generate_col_from_ori randomly picks nbr_line elements in a list of elements. We use this function to generate a column of the wanted size from a big list of elements.
The algorithm 2 creates a near duplicate version of an existing dataset. The function concat_elem concatenates for each line in the dataset all the elements of the line. So when we call this function the dataset is reduced to a single column. The function random_line adds and deletes to a dataset concatenated this way. We will use this algorithm in order to generate examples of NDDS for our learning and test sets.

Algorithm 1: RandomDataset generates a dataset from a list of columns named universe. The algorithm also returns the indexes of the columns used in the universe

input : *universe, nbr_col, nbr_line*
initialisation : $DataSet \leftarrow []$, $index\_universe \leftarrow []$
for $i \leftarrow 1$ **to** nbr_col **do**
  $original, index \leftarrow pick\_in\_universe($
   $universe)$
  $index\_universe[i] \leftarrow index$
  $DataSet[i] \leftarrow generate\_col\_from\_ori($
   $original, nbr\_line)$
end
return DataSet,index_universe

The algorithm 3 generates an example of each class. The function make_features extracts temporary feature vector from each dataset and builds the final feature vector using the formula 1 or 2.
In order to validate the pertinence of our method we use the algorithm 3 in a loop. We generate this way 1000 examples of each class (with formula 1) and then we project them in a bi-dimensional space using t-SNE [3] (perplexity : 30). The result is represented in fig 7.

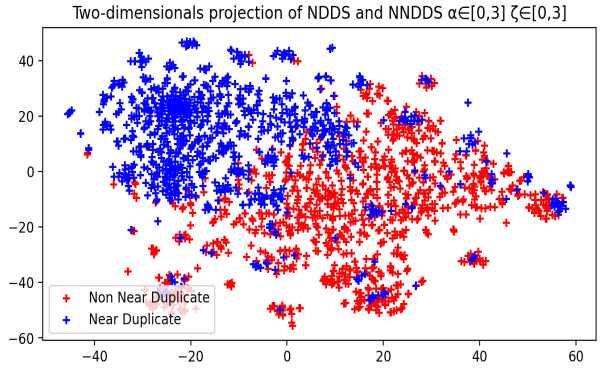

Figure. 6: NNDDS and NDDS for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$, 10 columns with formula 1

Algorithm 2: AlterateDataset generates a near duplicate version of a dataset by adding and deleting columns and then lines to the dataset. The function returns the near duplicate version (with columns concatenated) and the number of columns used to build it

input : *universe, Dataset ,list_index, α , nbr_modif*
initialisation : $cDataset \leftarrow copy(Dataset)$,
 $index\_universe \leftarrow []$,
 $clist\_index \leftarrow copy(list\_index)$,
 $nbr\_alteration \leftarrow round(\alpha$*length(cDataset))
for $i \leftarrow 0$ **to** nbr_modif **do**
  if $Random(True, False)$ **then**
    $original, index \leftarrow pick\_in\_universe($
     $universe)$ add
     $generate\_col\_from\_ori(original,$
    $length(cDataset[0])$ to cDataset
    add index to $index\_universe$
  else
    $r = random(0, length(cDataset) - 1)$
    delete $cDataset[r]$
    delete $index\_universe[r]$
  end
end
$cDataset = concat\_elem(cDataset)$
for $i \leftarrow 0$ **to** $nbr\_alteration$ **do**
  if $Random(True, False)$ **then**
  else
    add
     $random\_line(universe, index\_universe)$
    to cDataset
  end
  randomly delete an element from cDataset
  return cDataset, length($index\_universe$)
end

Algorithm 3: GenerateExemple returns two features vectors, one for a couple (DS,NNDDS) and the other for the couple (DS,NDDS)

input : *universe, nbr_col, α, ζ, minline, maxline*
$Dataset, indice\_universe \leftarrow RandomDataset($
 $universe, nbr\_col, RandBetween(minline, maxline)$
$alterateDS, len\_indice \leftarrow AlterateDataset($
 $universe, Dataset, indice\_universe, \alpha round((\zeta/10)*$
 $nbr\_col)$
$negDs, useless \leftarrow RandomDataset($
 $universe, len\_indice, RandBetween($
 $minline, maxline))$
$Dataset \leftarrow concat\_elem(Dataset)$
$negDs \leftarrow concat\_elem(negDs)$
$x\_pos \leftarrow make\_features(Dataset, alterateDS)$
$x\_neg \leftarrow make\_features(Dataset, negDs)$
return x_pos, x_neg

V. Experiments

Our experimental setup is a Colab notebook with a Xeon 2.30GHz 4 cores CPU and 25go of Ram and a tesla P100 (16go).
During all our experiments we will use the algorithm 3 in a loop in order to generate our learning and test sets. For the learning set the universe is formed of 100 columns containing each a minimum of 8000 elements. For the test set the universe is formed of 50 columns containing each a minimum of 8000 elements. Those two universes come from a manual collect of data on kaggle dataset [1].
In most experiments, if not specified, the number of lines for each dataset is randomly chosen between 50 and 300.

A. Influence of ζ parameter

Our first experiment uses the algorithm 3 in order to generate 7500 examples of each class with the following parameters : α equal to 2, number of columns (nbr_col) equal to 10, ζ equal to 0, 1, 2, 3 or randomly selected between 0 and 4 (for each couple of examples). In each scenario a random forest (RF) classifier [24] is trained (200 estimators, max depth 18) to distinguish between NNDS and NDDS. Tests are done with the datasets generated the same way (without the version with the random selection of ζ) but only with 500 examples of each class. The aim of this is to determine the best value of ζ parameter to use.

B. Influence of the number of columns

In our second experiment we will explore the influence of the columns number used to build the examples. We will generate multiple learning and test sets with the following parameters : α randomly picked between 0 and 3 (for each generated example), ζ randomly picked between 0 and 4 (for each generated example). The number of columns will vary : 5, 10, 20 or random choice between 5 and 22 (for each example) for the learning set; 5, 8, 11, 14, 17 and 20 for the test set. The number of examples and the classifier are identical to those in experiment V-A.

C. Evaluating multiple classifier

In our third experiment we generate our learning set using following parameters : α randomly picked between 0 and 3 (for each generated example), ζ randomly picked between 0 and 4 (for each generated example), the number of columns randomly picked between 5 and 22 (for each generated example). For the test sets we use following parameters : α and ζ varying between 0 and 3, the number of columns randomly picked between 5 and 22 (for each generated example). We decided to test 6 classifiers : a boosting algorithm, two gradient boosting algorithms, a Random forest, a deep neural network and a stacking algorithm. The six classifiers we use are adaboost[29] (500 estimators), a Light Gradient Boosting Machine (LGBM)[26], (200 estimators, max depth 14, nbr of leaf 12000), catboost [25] classifier (3000 iterations, depth 4, learning rate 0,1), a Random forest classifier (200 estimators, max depth 18), TabNet[34](n_d 8, n_a 8, n_steps 3) and a Stack of the 5 previous classifier (cross_validation 3,Meta-Model : logistic regression)[30].

D. Optimisation

In this experiment we will try to improve the execution time of the algorithm. First the feature extraction process requires counting the characters present in each element, the computation time increases with the number of lines in the dataset. This increase can be seen in figure 5. Thus, when the dataset has more than 100000 lines the extraction time becomes a problem.
To overcome this problem we propose two alternative methods of feature extraction to reduce the computation time. In the first one we will not extract directly the features on the whole concatenated data-set but on samples of it. Then for each sample we will extract the characteristics as before. Our final feature vector is then constructed by taking the average of the results obtained on the samples dimension by dimension. This gives us a vector of the same size as before, we will call this technique "method 1". The second method is simply to extract features on a sample of the concatenated data-set and not on the whole, we call this "method 2". The experimental parameters are the same as in experiment V-C. However, we modify the size of the generated datasets, which is randomly chosen between 2000 and 3000 in order to simulate datasets of larger size.

VI. Results

This part presents the results of all the experiments described in the experimental section.

A. Influence of ζ parameter

Table 2: Accuracy for various ζ values

Learning ζ \ Test ζ	0	1	2	3	mean
0	**1**	0.764	0.649	0.578	0.748
1	0.999	0.994	0.941	0.876	0.952
2	0.999	**0.998**	0.984	0.951	0.983
3	0.997	0.997	**0.989**	**0.971**	**0.988**
random(0,4)	**0.998**	0,998	0,985	0.954	0.984

Table 2 contains the results of the first experiment. Firstly, we can observe that selecting a low value of ζ in order to build the learning set leads to low accuracy. This is normal because this algorithm faces totally unseen situations. On the contrary, when ζ is high, results are good in all situations, the system had already seen examples equivalent to a lower ζ (because of the randomness when we add and delete columns in 2). In another of our work (which only focuses on columns) we have discovered that using a random α (between 0 and 3) for every generated near duplicate example in the training set leads to the best accuracy [31]. So it is interesting to observe that it is not the case for the ζ parameter, setting ζ to 3 leads to similar or better results than using a random ζ.

[1] https://www.kaggle.com/datasets

B. *Influence of the number of columns*

Results of the second experiment in the tab 3 show us that the number of columns used to build the examples for the learning set has no influence on the results (in our scenario). This is interesting because it means that we can have good results without having to create examples for each number of columns.

C. *Evaluating multiple classifier*

The tabs 4 to 7 and the fig 7 to 18, represent all the results of the third experiment. We can observe the same pattern on the heatmaps, a progressive degradation of the results with the increase of the α and ζ parameters. The effects of increasing α are quite small compared to those of ζ. Indeed, a dataset usually contains many more rows than columns, so changes at the column level have a greater impact on the features we extract than changes at the row level. This is especially true when the number of columns is low. We can see that using formula 1 or formula 2 has little influence on the results of most of the classifiers. Only RF and TabNet show a noticeable loss of accuracy with the use of formula 2. Despite the use of a powerful implementation specialised for tabular data, the neural network shows performances below those of other classifiers. The random forest presents slightly inferior results than the boosting algorithms, indeed although its results are the best when zeta is low this classifier suffers from a greater loss of accuracy than the others for higher zeta values. The boosting algorithms have similar results, but Catboost is the algorithm with the best overall results. Finally we can see that stacking in this problem does not bring any benefit.

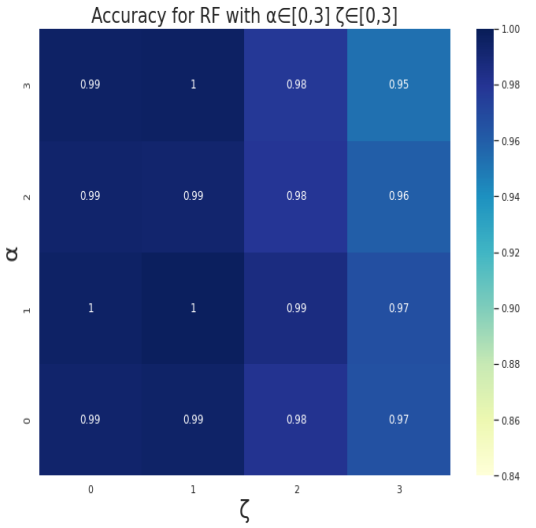

Figure. 7: Accuracy of RF classifier for $\alpha \in [0,3]$ and $\zeta \in [0,3]$ with for formula 1

Tables 8 and 9 describe the most important characteristics when making decisions for catboost and RF. First of all we can see that the most important characteristic in all cases is the entropy. Indeed, it will describe globally the dataset and changes in the dataset will lead to changes in the entropy. Then we can notice the presence of a large number of statis-

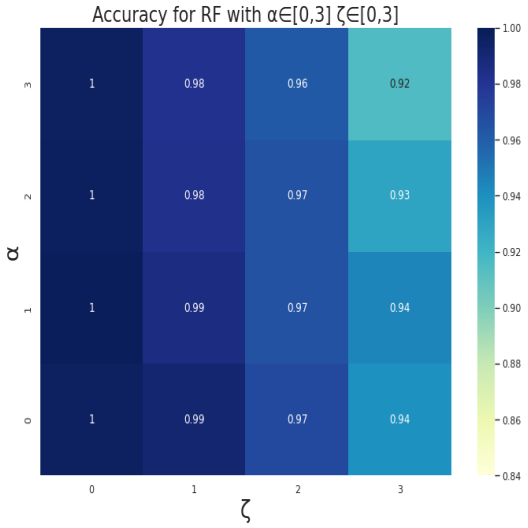

Figure. 8: Accuracy of RF classifier for $\alpha \in [0,3]$ and $\zeta \in [0,3]$ with for formula 2

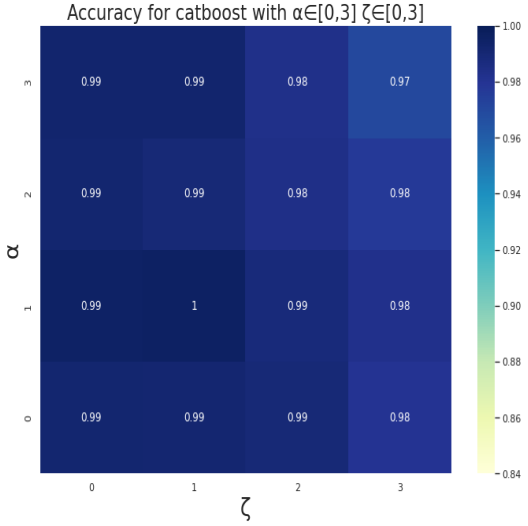

Figure. 9: Accuracy of catboost classifier for $\alpha \in [0,3]$ and $\zeta \in [0,3]$ with for formula 1

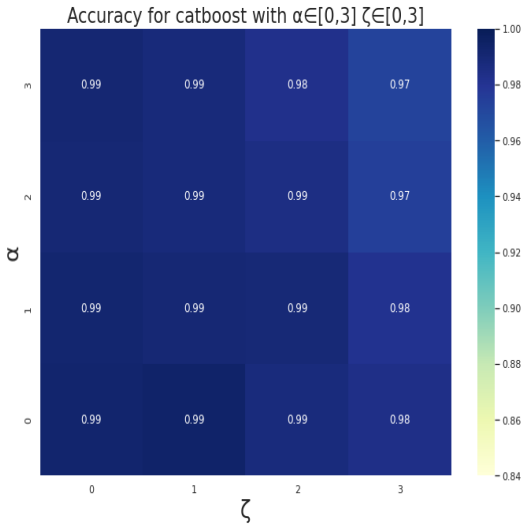

Figure. 10: Accuracy of catboost classifier for $\alpha \in [0,3]$ and $\zeta \in [0,3]$ with for formula 2

Table 3: Accuracy in function of the number of columns used to build training set and test set

train nbr col \ Test nbr col	5	8	11	14	17	20	mean
5	0.967	0.955	0.968	0.967	0.971	**0.978**	0.968
10	0.967	**0.968**	**0.971**	**0.976**	0.976	0.976	**0.972**
20	0.961	0.962	**0.971**	0.974	**0.979**	**0.978**	0.971
random(5,22)	**0.969**	0,965	0,965	0.973	0.974	**0.978**	0.970

Table 4: Accuracy of six classifiers for different couples of (α,ζ) values with formula 1 part 1.

models \ (α,ζ)	(0,0)	(1,0)	(2,0)	(3,0)	(0,1)	(0,2)	(0,3)	(1,1)	(1,2)
Catboost	0.992	0.995	0.992	0.993	0.992	**0.99**	0.981	0.996	0.989
RF	**0.994**	0.996	**0.994**	**0.995**	**0.995**	0.981	0.966	**0.998**	0.987
LGBM	0.991	0.995	0.991	0.992	0.992	0.988	0.984	0.995	0.987
TabNet	0.965	0.968	0.975	0.969	0.966	0.948	0.926	0.957	0.939
adaboost	0.99	0.994	**0.994**	0.991	0.989	0.988	**0.986**	0.993	0.983
Stacking	0.992	**0.997**	0.993	0.993	0.993	0.989	0.982	0.996	**0.99**

Table 5: Accuracy of six classifiers for different couples of (α,ζ) values with formula 1 part 2.

models \ (α,ζ)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	mean
Catboost	**0.983**	0.99	**0.984**	**0.98**	0.993	0.984	0.971	0.9874
RF	0.967	**0.992**	0.979	0.96	**0.996**	0.978	0.953	0.9830
LGBM	0.982	0.989	**0.984**	0.971	0.99	0.979	0.968	0.9859
TabNet	0.924	0.95	0.930	0.908	0.96	0.925	0.91	0.9446
adaboost	0.981	0.991	0.982	0.973	0.992	0.98	**0.977**	0.9861
Stacking	0.982	**0.992**	**0.984**	0.974	0.994	**0.985**	0.97	**0.9876**

Table 6: Accuracy of six classifiers for different couples of (α,ζ) values with formula 2 part 1.

models \ (α,ζ)	(0,0)	(1,0)	(2,0)	(3,0)	(0,1)	(0,2)	(0,3)	(1,1)	(1,2)
Catboost	0.993	0.992	0.99	0.991	0.994	0.988	0.984	0.991	0.99
RF	**0.997**	**0.999**	**0.997**	**0.997**	0.992	0.97	0.94	0.986	0.966
LGBM	0.996	0.995	0.994	0.995	0.996	**0.99**	**0.979**	0.993	**0.989**
TabNet	0.995	0.992	0.99	0.979	0.973	0.938	0.897	0.957	0.923
adaboost	0.996	0.998	0.996	0.995	0.996	0.988	0.971	0.993	0.985
Stacking	**0.997**	0.998	**0.997**	**0.997**	**0.998**	0.986	0.968	**0.994**	0.985

Table 7: Accuracy of six classifiers for different couples of (α,ζ) values with formula 2 part 2.

models \ (α,ζ)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	mean
Catboost	**0.982**	**0.989**	**0.986**	**0.975**	0.99	**0.983**	**0.973**	**0.9866**
RF	0.945	0.983	0.966	0.93	0.982	0.962	0.915	0.9701
LGBM	0.979	0.989	0.981	0.968	0.99	0.98	0.96	0.9856
TabNet	0.9	0.945	0.913	0.876	0.948	0.903	0.856	0.9365
adaboost	0.973	0.986	0.978	0.961	0.991	0.978	0.96	0.9839
Stacking	0.964	**0.993**	0.98	0.957	**0.992**	0.975	0.946	0.9827

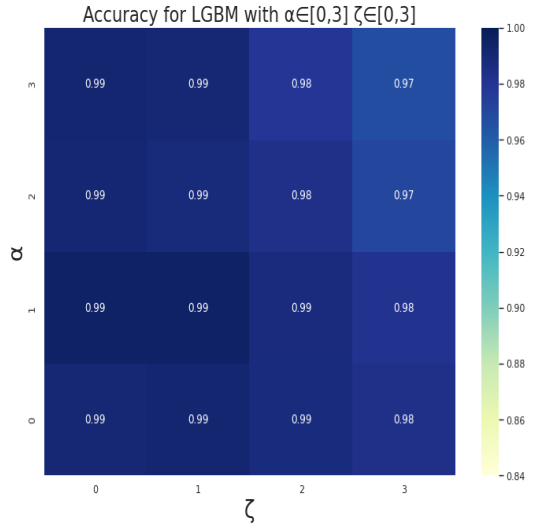

Figure. 11: Accuracy of LGBM classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 1

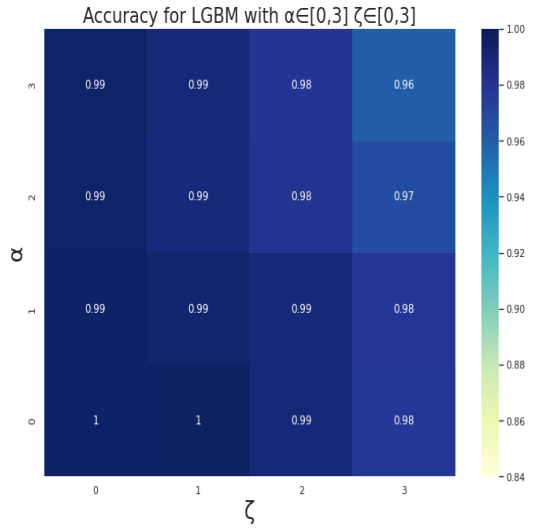

Figure. 12: Accuracy of LGBM classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 2

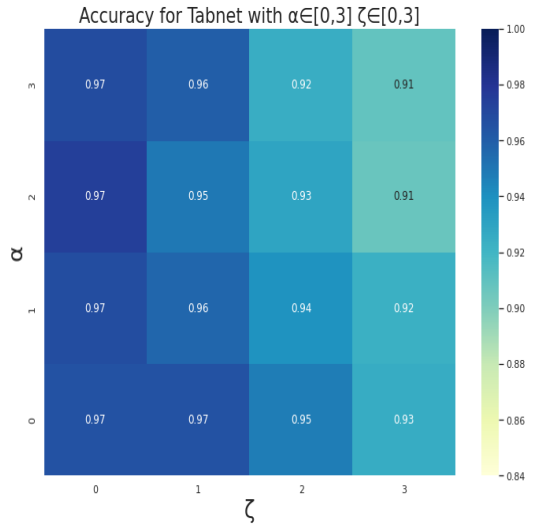

Figure. 13: Accuracy of TabNet classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 1

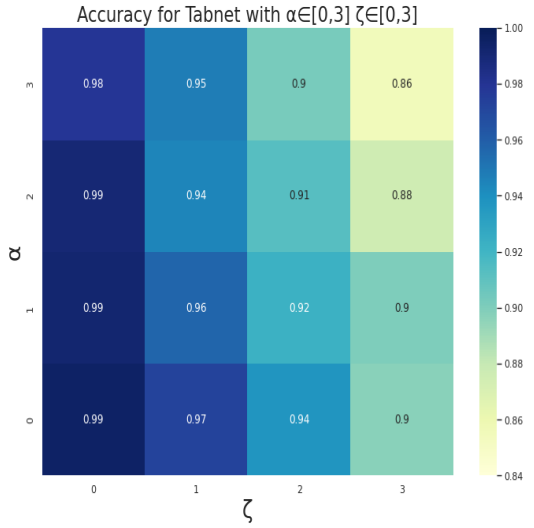

Figure. 14: Accuracy of TabNet classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 2

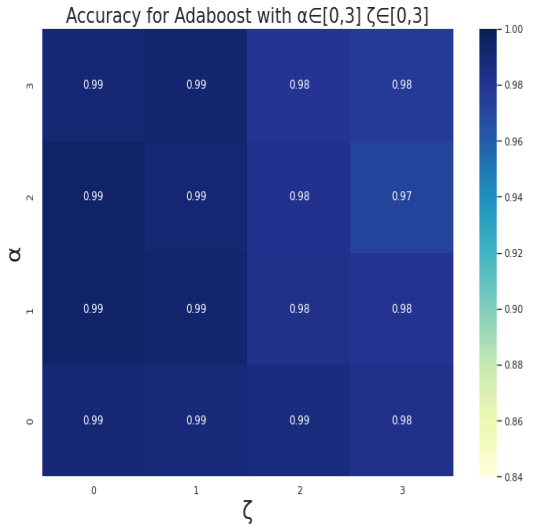

Figure. 15: Accuracy of Adaboost classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 1

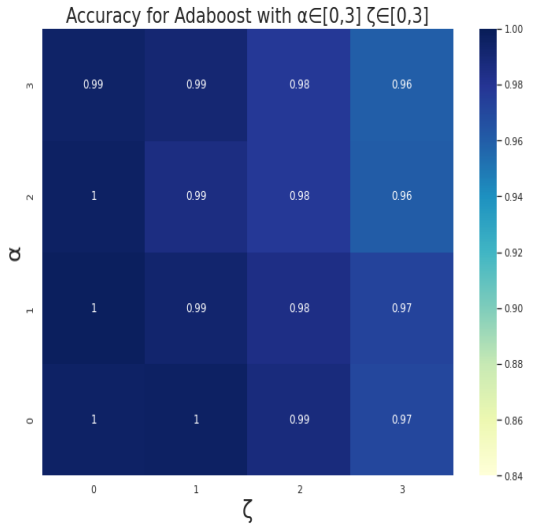

Figure. 16: Accuracy of Adaboost classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 2

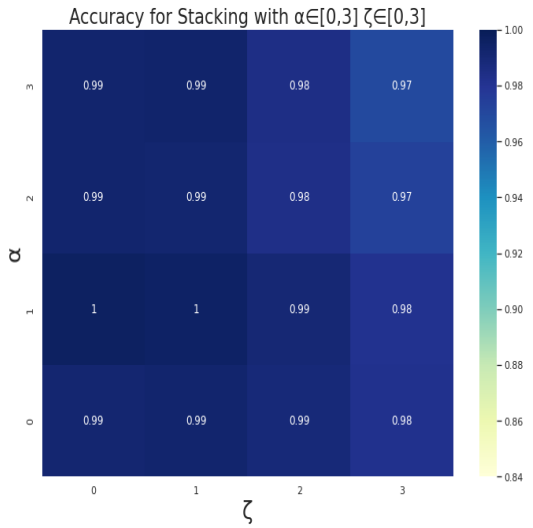

Figure. 17: Accuracy of Stack classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 1

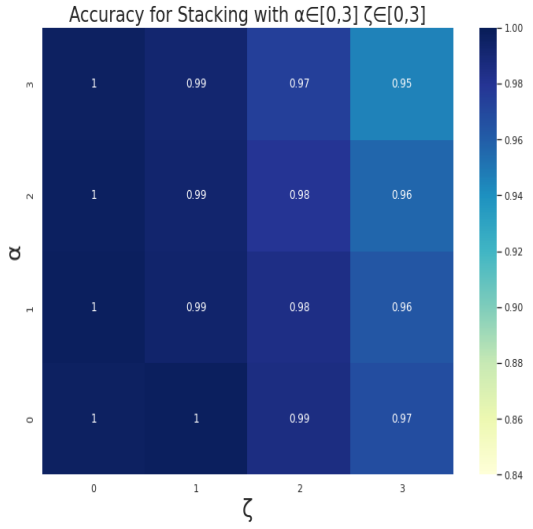

Figure. 18: Accuracy of Stack classifier for $\alpha \in [0, 3]$ and $\zeta \in [0, 3]$ with for formula 2

tics on the separating characters such as ".', '/' or "-" these characters are very present in a small number of types (semantic) of columns and very few in the others which makes them very distinctive. Finally, we notice in this top 10 the presence of many statistics which concern letters not very frequent in English (the language of our examples) for example 'z'[20], but also of the number 9 which is a number rarely present in the real data [21]. This makes a change in the number of these characters very distinctive.

Table 8: Top 10 features for catboost and Rf models with formula 1, ranked in descending order by their gini impurity score.

catboost formula 1	RF formula 1
Entropy	Entropy
Mean number of 'f'	Mean number of ','
Median number of '.'	Mean number of 'z'
Mean number of '-'	Mean number of 'q'
Var number of '.'	Var number of ','
Median number of '0'	Var number of 'q'
Mean number of '/'	Mean number of '/'
Mean number of '+'	Mean number of '-'
Var number of '9'	Var number of '0'
Mean number of '-'	Mean number of 'f'

Table 9: Top 10 features for catboost and Rf models with formula 2, ranked in descending order by their gini impurity score.

catboost formula 2	RF formula 2
entropy	entropy
Std of numerical chars cells	Std of number of numerical chars cells
Mean number of '-'	Mean number of ','
Mean number of '.'	Mean number of '-'
Mean number of 'f'	var number of '.'
Mean number of 'q'	var number of 'f'
Std number of words cells	Mean number of words cells
'.' char in all cells	Std of number alphabetical chars in cells
Mean number of '/'	max number of ','
Median of string length	max number of '0'

After all these experiments we can conclude that formula 2 is the best because it gives very similar results to formula 1 while being easier to calculate. Moreover, in other experiments we have performed in which more similar negative examples (e.g. an altered version with alpha equal to 5) are used, formula 2 continues to have the same results while those of formula 1 decrease. Moreover, it should be noted that these results are dependent on the method of generation and testing and that not all cases can be covered and that the algorithms must be adapted to each situation. For example, by modifying the function that RandomDataset (algorithm 1) we can generate two datasets with the same number of rows and whose columns have the same semantic type. Furthermore as these columns are sampled in the same way from the columns present in the universe they are extremely similar while being different. By using this method to create 1000 additional negative examples during training we can modify the results on the test data. The accuracy of catboost on our previous test method falls to 93.8% although it is 97.1% on the examples generated in the way we just described (accuracy calculated on examples generated from test data). Since there is no ground truth, it is necessary to adapt the method of generating learning examples according

to the expected real data.

D. Optimisations

Table 10: Accuracy of six classifiers for the two formulas for the methods 1 and 2

models \ formula	1	2
Catboost m1	0.9835	**0.9852**
Catboost m2	0.9615	0.9711
RF m1	**0.9673**	0.9573
RF m2	0.9560	0.9528
LGBM m1	**0.9774**	**0.9774**
LGMB m2	0.9566	0.9660
TabNet m1	**0.8883**	0.8412
TabNet m2	0.8343	0.8700
adaboost m1	0.9741	**0.9750**
adaboost m2	0.9442	0.9638

As in our previous tests, catboost shows the best results in all scenarios. Method 2, although more easily parallelizable, gives inferior results to method 1. It is particularly interesting to note that the loss of accuracy is negligible with catboost. Thus by using method 1 and catboost we can divide by two the extraction time while losing only 0.3% of accuracy. These results must however be treated with caution because our example generation algorithm does not generate all possible cases.

In conclusion, we can add that in a realistic environment, two optimisations are possible. First we can reduce the search space of the candidate peers. To do this we can use a local sensitive hashing algorithm. We have tested the version using random projections to generate the hash code. On the examples we tested the nearly duplicated elements were always in the same bucket [18, 9](choose carefully the number of bits and hashtables). Thus for a new dataset, we will only test datasets in the same bucket as the new value with our algorithm. The results can be further improved by using a density sensitive hashing function [10].

Secondly, to more finely select the datasets identified as almost duplicated we can calibrate [19] the output probabilities of the classifier and use a threshold.

VII. Conclusion

In this study we introduced the new concept of nearly duplicated datasets. In order to identify these nearly duplicated datasets we developed a method using a classifier to distinguish nearly duplicated datasets. To test this method, we introduced a technique to generate relevant examples to train our classifier. In order to optimise results we performed several experiments to determine the best parameters to use in order to generate the training set. All these investigations led us to a good recognition rate of more than 95% in useful cases. More research needs to be done to generate more different cases to make learning more resilient. In future work, we will attempt to reduce the number of features using feature selection algorithms [17, 38, 39]. We will also test other metrics as well as evaluate the effects of increasing the size of the universe.

In addition, further studies need to be done with classical data generation [36] and data pollution [37] tools in order to evaluate the results of our algorithm under other conditions as well as to establish a benchmark for comparison with other algorithms.

Acknowledgements

I gratefully acknowledge Astrid Balick for her generous support. Supported by organization Synaltic

References

[1] Broder, A. Identifying and filtering near-duplicate documents. *Annual Symposium On Combinatorial Pattern Matching*. pp. 1-10 (2000)

[2] Haas, P., Naughton, J., Seshadri, S. & Stokes, L. Sampling-Based Estimation of the Number of Distinct Values of an Attribute. *Proceedings Of The 21th International Conference On Very Large Data Bases*. pp. 311-322 (1995)

[3] Van der Maaten, L. & Hinton, G. Visualizing High-Dimensional Data Using t-SNE. *Journal Of Machine Learning Research*. **9** pp. 2579-2605 (2008)

[4] Panse, F. & Naumann, F. Evaluation of Duplicate Detection Algorithms: From Quality Measures to Test Data Generation. *2021 IEEE 37th International Conference On Data Engineering (ICDE)*. pp. 2373-2376 (2021,4)

[5] Herzog, T. Data quality and record linkage techniques. (Springer,2007)

[6] Papadakis, G., Ioannou, E. & Palpanas, T. Entity resolution: Past, present and yet-to-come: From structured to heterogeneous, to crowd-sourced, to deep learned. (2020), EDBT/ICDT 2020 Joint Conference ; Conference date: 20-03-2020 Through 02-04-2020

[7] Jahns, V. Principles of Data Integration by Anhai Doan, Alon Halevy, Zachary Ives. *SIGSOFT Softw. Eng. Notes*. **37**, 43 (2012,9), https://doi.org/10.1145/2347696.2347721

[8] Ngueilbaye, A., Wang, H., Mahamat, D. & Elgendy, I. SDLER: stacked dedupe learning for entity resolution in big data era. *The Journal Of Supercomputing*. **77**, 10959-10983 (2021,3), https://doi.org/10.1007/s11227-021-03710-x

[9] Wang, J., Shen, H., Song, J. & Ji, J. Hashing for Similarity Search: A Survey. *ArXiv*. **abs/1408.2927** (2014)

[10] Lin, Y., Cai, D. & Li, C. Density Sensitive Hashing. *IEEE Transactions On Cybernetics*. **44** pp. 1362-1371 (2014)

[11] Draisbach, U., Christen, P. & Naumann, F. Transforming Pairwise Duplicates to Entity Clusters for High-Quality Duplicate Detection. *J. Data And Information Quality*. **12** (2019,12), https://doi.org/10.1145/3352591

[12] Christen, P. Automatic Record Linkage Using Seeded Nearest Neighbour and Support Vector Machine Classification. *Proceedings Of The 14th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 151-159 (2008), https://doi.org/10.1145/1401890.1401913

[13] Naumann, F. & Herschel, M. An Introduction to Duplicate Detection. *Synthesis Lectures On Data Management*. **2**, 1-87 (2010,1), https://doi.org/10.2200/s00262ed1v01y201003dtm003

[14] Papadakis, G., Skoutas, D., Thanos, E. & Palpanas, T. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.*. **53** (2020,3), https://doi.org/10.1145/3377455

[15] Koumarelas, I., Jiang, L. & Naumann, F. Data Preparation for Duplicate Detection. *Journal Of Data And Information Quality (JDIQ)*. **12** pp. 1 - 24 (2020)

[16] Abedjan, Z., Golab, L., Naumann, F. & Papenbrock, T. Data Profiling. *Synthesis Lectures On Data Management*. **10**, 87 (2018,11), https://doi.org/10.2200/s00878ed1v01y201810dtm052

[17] Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N. & Clairmont, C. Seeding Initial Population, in Genetic Algorithm for Features Selection. *Advances In Intelligent Systems And Computing*. pp. 572-582 (2021), https://doi.org/10.1007/978-3-030-73689-7_55

[18] Andoni, A. & Indyk, P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*. **51**, 117-122 (2008,1), https://doi.org/10.1145/1327452.1327494

[19] Niculescu-Mizil, A. & Caruana, R. Predicting Good Probabilities with Supervised Learning. *Proceedings Of The 22nd International Conference On Machine Learning*. pp. 625-632 (2005), https://doi.org/10.1145/1102351.1102430

[20] OHLMAN, H. Subject-Word Letter Frequencies with Applications to Superimposed Coding. *Proceedings Of The International Conference On Scientific Information*. (1959,12), https://doi.org/10.17226/10866

[21] Benford., F. The law of anomalous numbers.. (Proc. of the American Philosophical Society,1938)

[22] Panse, F. & Naumann, F. Evaluation of Duplicate Detection Algorithms: From Quality Measures to Test Data Generation. *2021 IEEE 37th International Conference On Data Engineering (ICDE)*. pp. 2373-2376 (2021)

[23] Karl Pearson F.R.S. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, And Dublin Philosophical Magazine And Journal Of Science*. **2**, 559-572 (1901)

[24] Breiman, L. Random Forests. *Machine Learning*. **45**, 5-32 (2001,10)

[25] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. & Gulin, A. CatBoost: unbiased boosting with categorical features. (2019)

[26] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. & Liu, T. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances In Neural Information Processing Systems*. **30** (2017)

[27] Pennington, J., Socher, R. & Manning, C. GloVe: Global Vectors for Word Representation. *Proceedings Of The 2014 Conference On Empirical Methods In Natural Language Processing (EMNLP)*. pp. 1532-1543 (2014,10)

[28] Le, Q. & Mikolov, T. Distributed Representations of Sentences and Documents. *Proceedings Of The 31st International Conference On Machine Learning*. **32**, 1188-1196 (2014,6,22)

[29] Freund, Y. & Schapire, R. A Short Introduction to Boosting. *In Proceedings Of The Sixteenth International Joint Conference On Artificial Intelligence*. pp. 1401-1406 (1999)

[30] Wolpert, D. Stacked generalization. *Neural Networks*. **5**, 241-259 (1992)

[31] Chevallier, M., Boufares, F., Grozavu, N., Rogovschi, N. & Clairmont, C. Near duplicate column identification: a machine learning approach. *2021 IEEE Symposium Series On Computational Intelligence (SSCI)*. (2021,12), https://doi.org/10.1109/ssci50451.2021.9659897

[32]

[33] Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N. & Clairmont, C. Detecting Near Duplicate Dataset. *Proceedings Of The 13th International Conference On Soft Computing And Pattern Recognition (SoCPaR 2021)*. pp. 394-403 (2022), https://doi.org/10.1007/978-3-030-96302-6_36

[34] Arik, S. & Pfister, T. TabNet: Attentive Interpretable Tabular Learning. (2020)

[35] Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç. & Hidalgo, C. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. *Proceedings Of The 25th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 1500-1508 (2019), https://doi.org/10.1145/3292500.3330993

[36] Christen, P. & Pudjijono, A. Accurate Synthetic Generation of Realistic Personal Information. *Proceedings Of The 13th Pacific-Asia Conference On Advances In Knowledge Discovery And Data Mining*. pp. 507-514 (2009), https://doi.org/10.1007/978-3-642-01307-2_47

[37] Christen, P. & Vatsalan, D. Flexible and Extensible Generation and Corruption of Personal Data. *Proceedings Of The 22nd ACM International Conference On Information & Knowledge Management*. pp. 1165-1168 (2013), https://doi.org/10.1145/2505515.2507815

[38] Eid, H. & Abraham, A. Adaptive feature selection and classification using modified whale optimization algorithm. *International Journal Of Computer Information Systems And Industrial Management Applications*. **10** pp. 174-182 (2018)

[39] Chotchantarakun, K. & Sornil, O. An Adaptive Multilevels Sequential Feature Selection. *International Journal Of Computer Information Systems And Industrial Management Applications*. **13** pp. 10-19 (2021)

[40] Trung, H. Database Performance Evaluation and Applications of Data Science for IoT Platform Analysis *International Journal Of Computer Information Systems And Industrial Management Applications*. **13** pp. 124-135 (2021)

Author Biographies

Marc Chevallier born in 1991 in France has completed a master of Science in Engineering at Sorbonne University, France in 2015 and a master in innovation management at Sorbonne University, France in 2018. He is currently doing his Ph.D. in computer science within the LIPN laboratory at the Sorbonne Paris Nord University, France. He is also a member of the company Synaltic which supports his work since 2018. His main research topics focus on data profiling using machine learning as well as feature selection using genetic algorithms.

Nicoleta Rogovschi born in 1983 in Moldova received her Master of Computer Science degree from Paris 13 University in 2006 in Machine Learning. She is currently an Associate Professor in Computer Science at the Paris Descartes University. She completed her Ph.D. in Computer Science (Probabilistic Unsupervised Learning) in 2009 in the Computer Science Laboratory of Paris 13 University and the HdR (Hability to direct researches) in 2021 at Sorbonne Paris Nord University. She's research is with the Data Mining (GFD) Team. Her research interests include Probabilistic Learning, Unsupervised Learning, Clustering and Co-Clustering methods for different types of data in different contextes : anonymization, recommender system, opinion detection,... She is also a member of EGC, AFIA, IEEE, INNS, INNS AML group. Nicoleta Rogovschi codirected 5 PhD students and tens of Master Research students.

Faouzi Boufares is assistant professor in computer science at Paris XIII University since 1990. He is a member of the computer science laboratory of the University of Paris Nord. His research work, since his PhD thesis in 1986, has focused on databases. The main topics recently have been data profiling and semantics to automatically detect and correct anomalies in data when integrating heterogeneous data. Data quality was the main objective to better support decision making. His research interests are in data engineering and data science. He has directed several theses in the field, which have led to several publications, namely on data quality and deduplication. Recently, in his research team, the principles of machine learning are used to serve data quality. His teachings focused on the analysis and design of information systems, databases and data warehouses, and decision support systems.

Nistor Grozavu born in 1984 in Moldova received his HdR degree from Sorbonne Paris Nord University in 2020 and PhD degree in Unsupervised Machine Learning in 2009 from Paris 13 University. He is currently Full Professor in Computer Science at CY Cergy Paris University. His research is with the MIDI team from ETIS Laboratory. His research interests include Unsupervised Learning, Transfer Learning, Dimensionality reduction, Collaborative Learning, Machine Learning by Matrix Factorization and content-based information retrieval, Quantum Machine Learning. These researches are applied in different applications for text mining, visual information retrieval, recommendation systems, fraud detection, etc. via ANR, FUI, PEPS CNRS, AUF projects. He is also a member of IEEE, INNS, and the co-founder of the INNS Autonomous Machine Learning group. Nistor is the co-author of a patent on visual information retrieval, published two book chapters, 12 peer-reviewed journal papers, and more than 40 international conference papers. Nistor Grozavu co-supervised 2 post-doctorants, 8 PhD students and supervise each year 1-2 Master2 internship.

Charly Clairmont born in 1978 in France has completed a Master in Computer Science at University of Reims Champagne Ardenne in 2000 and a Master in Business Management at ISG Paris in 2002. Co-Founder of Altic, now Synaltic. Since 2004 it has participated actively in the promotion of Business Intelligence in France, mainly in the open source area with large meetup groups like Hadoop User Group and Paris Spark Meetup. Via Synaltic, he participated in various research projects in paneuropean collaborations or with French Universities like Sorbonne Paris Nord.