

# On Choosing Thresholds for Duplicate Detection

(Research-in-Progress)

**Uwe Draisbach**

Hasso Plattner Institute, Potsdam, Germany  
uwe.draisbach@hpi.uni-potsdam.de

**Felix Naumann**

Hasso-Plattner-Institute, Potsdam, Germany  
felix.naumann@hpi.uni-potsdam.de

**Abstract:** Duplicate detection, i.e., the discovery of records that refer to the same real-world entity, is a task that usually depends on multiple input parameters by an expert. Most notably, an expert must specify some similarity measure and some threshold that declares duplicity for record pairs if their similarity surpasses it. Both are typically developed in a trial-and-error based manner with a given (sample) dataset.

We posit that the similarity measure largely depends on the nature of the data and its contained errors that cause the duplicates, but that the threshold largely depends on the *size of the dataset* it was tested on. In consequence, configurations of duplicate detection runs work well on the test dataset, but perform worse if the size of the dataset changes. This weakness is due to the transitive nature of duplicity: In larger datasets transitivity can cause more records to enter a duplicate cluster than intended. We analyze this interesting effect extensively on four popular test datasets using different duplicate detection algorithms and report on our observations.

**Key Words:** Data Quality, Entity Resolution, Duplicate Detection

## 1 Introduction

Duplicate detection is the process of finding multiple records in a dataset that represent the same real-world entity [16]. It has been a research topic for several decades and remains relevant due to the ever increasing size and complexity of datasets, especially in application areas, such as master data management, customer relationship management, or data warehousing.

Duplicate detection faces two challenges: First, an exhaustive comparison of all record pairs has a quadratic complexity and is not feasible, even with modern hardware. For example, Facebook has about 900m users, which results in more than  $400 \cdot 10^{15}$  comparisons. Second, there is no key that can be used to identify those records that represent the same real-world entity. Due to typos and erroneous or incomplete data, we need some similarity measure to be used in combination with a threshold to decide whether a record pair represents the same real-world entity or not.

For the first problem, several algorithms have been proposed that select only a subset of candidate pairs with a high probability of being duplicates, whereas pairs with a low probability are omitted. In this way, the computational effort can be reduced significantly, without a highly negative effect on the duplicate detection result. However, this paper focuses on the second challenge, deciding whether a record pair is a duplicate or not. In the past decades, many similarity measures have been proposed to determine the similarity of strings and numbers. These measures can be used to calculate the similarity of individual attribute values and

these attribute similarities can then be aggregated to an overall record similarity. The selection of relevant attributes and their best similarity measure is a domain specific task that requires a domain expert.

A threshold can then be used to classify whether a record pair is a duplicate or not. If the similarity is above the threshold, the pair is a duplicate and both records belong to the same cluster. Additional records are added to the cluster, if the similarity to at least one record in the cluster is above or equal to the threshold. In this way, we also classify all other records in that cluster as a duplicate of the new record. However, it might be possible that some of the records in a cluster have a similarity below the threshold and are classified as duplicate only due to the transitive relationship. Thus, the selection of a good threshold is very important. On the one hand, the threshold should not be too high so that no true duplicates are missed. On the other hand, it should not be so low that many non-duplicates are classified as duplicate, either because the calculated similarity is above the threshold or because the calculated similarity of another pair in that cluster is above the threshold. Choosing the optimal threshold is one of the main difficulties of configuring a duplicate detection program for a given dataset.

Interestingly, this problem becomes harder, if the size of the dataset increases over time. The optimal threshold should not be evaluated only once, but there is a necessity to evaluate it again if new records are added. This observation is the main contribution of this paper. Figure 1 illustrates the problem. In Fig. 1(a) we have three records A, B, and C. The edges show the similarity between these records. Records A and B are a duplicate, so they belong to the same cluster. If we want to select a good threshold for this sample, we can use any value  $> 0.8$  and  $\leq 0.9$ . Then  $\langle A, B \rangle$  would be classified correctly as duplicate and the record pairs  $\langle A, C \rangle$  and  $\langle B, C \rangle$  would be correctly classified as non-duplicate.

Consider a new record D that is inserted into the dataset, e.g., a new customer is added to a customer database, with D as a duplicate of C. This case is shown in Fig. 1(b). With the previous threshold,  $\langle C, D \rangle$  is classified correctly as duplicate, as well as  $\langle A, D \rangle$  as non-duplicate. An issue might arise with pair  $\langle B, D \rangle$ . If the chosen threshold is  $\leq 0.84$ , this pair is classified as duplicate, even though it is a non-duplicate. This problem is even aggravated, because the records belong to a cluster with multiple records: Due to transitivity record pairs  $\langle A, D \rangle$ ,  $\langle A, C \rangle$ , and  $\langle B, C \rangle$  are also classified as duplicates.

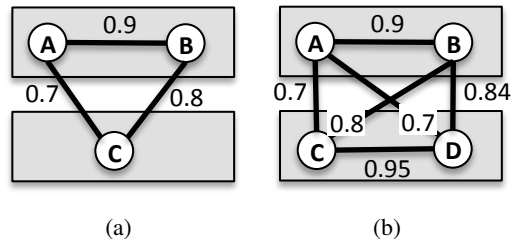


Figure 1: Illustration of threshold selection problem

Such observations shall serve as a warning that a once configured threshold might not be appropriate for larger datasets. In our experience, thresholds are often manually configured based on a sample of data. We show that the setting is in general no longer optimal for larger datasets, even if they have the same properties. Section 2 describes and analyzes in detail various experiments showing that indeed thresholds are quite sensitive to dataset size. Section 3 discusses related work. Finally, Section 4 concludes the paper.

## 2 Threshold Experiments

To evaluate our assumptions from the introduction, we conducted some experiments. This section first describes the used datasets and the experiment setup, and then shows our results.

## 2.1 Datasets

For our experiments, we used four datasets. Some of them are real-world datasets often used in other papers, and some of them were artificially generated. The used datasets are:

**Febrl** We used the Febrl dataset generator [3] to create two artificial datasets. Both contain 10,000 clusters, i.e, information about 10,000 entities. The smaller dataset (Febrl sm.) contains an additional 1,000 duplicate records with up to three duplicates per cluster. The larger Febrl dataset (Febrl la.) contains 10,000 additional duplicates with up to nine duplicates per cluster. Furthermore, the duplicates in the larger dataset can have more modifications than those in the smaller one.

**CD** The CD dataset<sup>1</sup> is a randomly selected extract from freeDB.org. It contains information for 9,760 CDs, including artist, title, and songs. The dataset has been used in several papers [4, 11].

**Cora** The Cora Citation Matching dataset<sup>2</sup> comprises 1,879 references of research papers and is often used in the duplicate detection community [2, 5]. We have described the definition of a gold standard for the Cora dataset in [7].

Table 1 shows an overview of the datasets used for the experiments, including the number of records and clusters and the maximum cluster size.

Dataset	# Rec.	# Clust.	Max. Cl. S.	Type
Febrl (sm.)	11,000	10,000	4	artificial
Febrl (la.)	20,000	10,000	10	artificial
CD	9,760	9,505	6	real-world
Cora	1,879	118	238	real-world

Table 1: Overview of datasets.

## 2.2 Experiment Setup

For all datasets, we implemented one or more similarity functions for a pair-wise comparison. Each similarity function calculates the similarity of single attributes and then aggregates those attribute similarities to a record pair similarity. The similarity is a value from 0.0 to 1.0, with 1.0 for identical records and 0.0 for records with no similarity.

For all datasets, we used two pair-selection algorithms. The first one is the naive approach, which creates the Cartesian product of all records. Because we use only symmetric similarity functions, we do not have to consider ordered pairs, which means that if we create a pair  $\langle A, B \rangle$  we do not create  $\langle B, A \rangle$ .

The second pair-section algorithm is the well-known Sorted-Neighborhood-Method (SNM) [10], which sorts the records based on one or more sorting keys and then slides a fixed-size window over the sorted records. Only those records within the same window are compared. As in most real-life scenarios, we use multiple keys to avoid that due to erroneous values in the attributes used as sorting key, real duplicates are far away in the sorting order. We use a window size of 20 for all SNM experiments. For both algorithms we calculate the transitive closure of duplicates after each run: if  $\langle A, B \rangle$  and  $\langle B, C \rangle$  are classified as duplicate, also  $\langle A, C \rangle$  is a duplicate, regardless of the similarity of  $\langle A, C \rangle$ .

For the **Febrl** dataset, we have two similarity functions, so we are able to evaluate if the effect of an increasing best threshold depends on the similarity measure. Both of them calculate the average similarity

<sup>1</sup>[http://www.hpi.uni-potsdam.de/naumann/projekte/dude\\_duplicate\\_detection.html#c14715](http://www.hpi.uni-potsdam.de/naumann/projekte/dude_duplicate_detection.html#c14715)

<sup>2</sup><http://people.cs.umass.edu/~mccallum/data.html>

of the attributes *first name*, *last name*, *address*, *suburb*, and *state*. The first similarity function uses the Jaro-Winkler similarity [19], whereas the second one uses the Levenshtein distance [14]. For SNM, we use three different sorting keys:  $\langle \textit{first name}, \textit{last name} \rangle$ ,  $\langle \textit{last name}, \textit{first name} \rangle$ , and  $\langle \textit{postcode}, \textit{address} \rangle$ .

The similarity function for the **Cora** dataset calculates the average Jaccard coefficient [16] of attributes *title* and *author* using bigrams. Additionally, we use rules that set the similarity of a record pair to 0.0. These rules are (1) the year attribute has different values, (2) one reference is a technical report and the other is not, (3) the Levenshtein distance of attribute pages is greater than two, and (4) one reference is a journal, but the other one a book. For SNM, the used Cora sorting keys are  $\langle \textit{ReferenceID}, \textit>Title}, \textit{Author} \rangle$ ,  $\langle \textit>Title}, \textit{Author}, \textit{Refer.ID} \rangle$ , and  $\langle \textit{Author}, \textit>Title}, \textit{Refer.ID} \rangle$ .

For the **CD** dataset, we use a similarity function that calculates the average Levenshtein similarity of the three attributes *artist*, *title*, and *track01*, but also considers NULL values and string-containment. The similarity function is the same as described in [6]. For SNM, the three used sorting keys are  $\langle \textit{Artist}, \textit>Title}, \textit{Track01} \rangle$ ,  $\langle \textit>Title}, \textit{Artist}, \textit{Track01} \rangle$ , and  $\langle \textit{Track01}, \textit{Artist}, \textit>Title} \rangle$ .

Our evaluation is based on the F-Measure, i.e., the harmonic mean of precision (fraction of correctly detected duplicates and all detected duplicates) and recall (fraction of detected duplicate pairs and the overall number of existing duplicate pairs). We evaluate all datasets with different threshold values. The threshold is increased by 0.01 in every iteration up to 1.0. Additionally, we increase the number of records to measure the effect of an increased number of records on the selection of the threshold. To summarize, we have five parameters that are evaluated in our experiments:

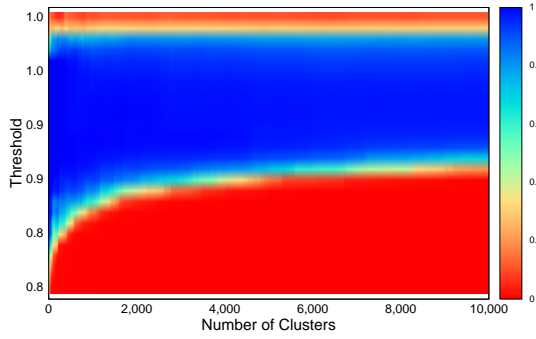
- **Dataset:** Febrl (sm.), Febrl (la.), Cora, and CD.
- **Pair-selection algorithm:** All pairs (Naive) and Sorted-Neighborhood-Method (SNM).
- **Similarity measure:** JaroWinkler and Levenshtein for Febrl datasets, Jaccard for the Cora and Levenshtein for the CD dataset.
- **Number of clusters / records:** For both Febrl datasets we start with 10 clusters and increase by 10 clusters until we use the entire dataset. For Cora and CD, we start with 10 records and increase by 10 records in each iteration step.
- **Threshold values:** Threshold values increased by increments of 0.01 up to 1.0.

We have exhaustively evaluated all parameter combinations and report on a large subset of them. The described effects were observable for all combinations.

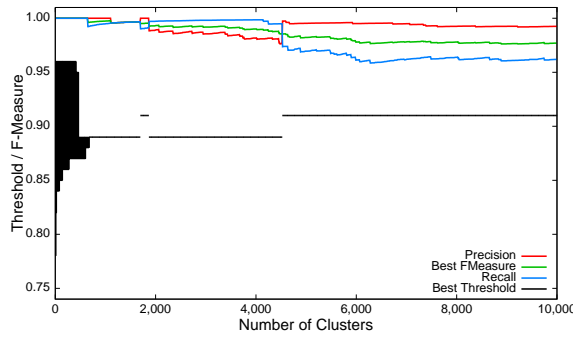
## 2.3 Experimental Results

In this section we describe the results of our experiments. As mentioned before, we evaluated different dataset sizes and different threshold values for the classification as a duplicate or non-duplicate. Our results are shown in Figures 2 to 5. Each figure shows on the left a heatmap for one combination of dataset, algorithm, and similarity measure. On the x-axis, we have an increasing number of records and the y-axis shows the different threshold values. The color determines the observed F-Measure values. For every heatmap, we also created an additional chart with the same axes, showing for the same experiment the threshold that achieved the best F-Measure. Additionally, this chart shows the precision, recall, and F-Measure values for this threshold.

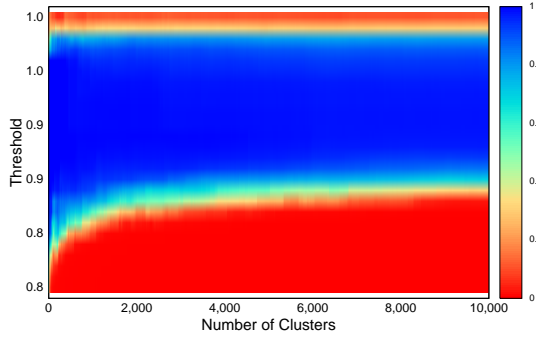
Figure 2 shows the result for the small and the large Febrl datasets for the Jaro-Winkler similarity measure and the two pair-selection algorithms, naive and SNM. For each dataset we show the number of clusters on the x-axis. As expected, the best threshold value increases with an increasing number of clusters. For a small number of clusters, we have a window for the best threshold. This makes it easy for a user to find



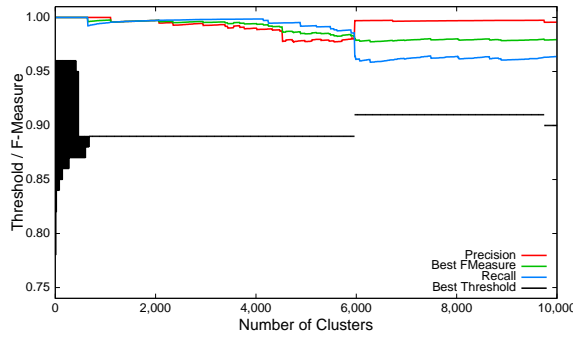
(a) F-Measure for Febrl (sm.) with naive algorithm



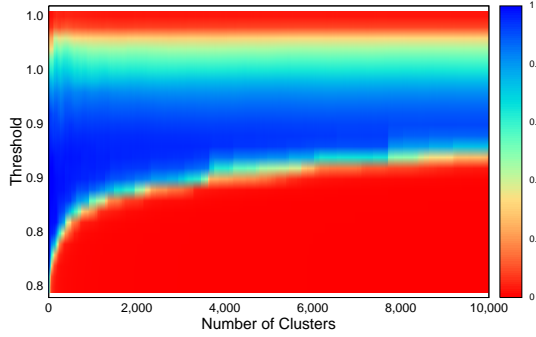
(b) Best threshold/F-Measure (Febrl (sm.), naive algorithm)



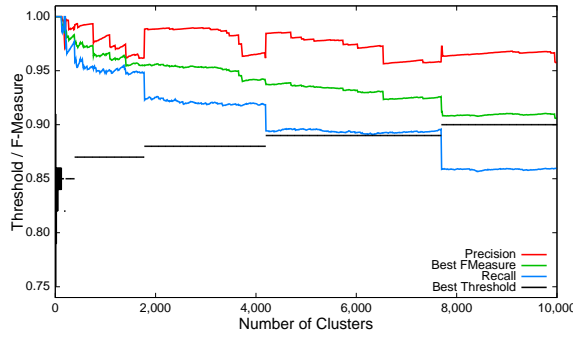
(c) F-Measure for Febrl (sm.) with SNM algorithm



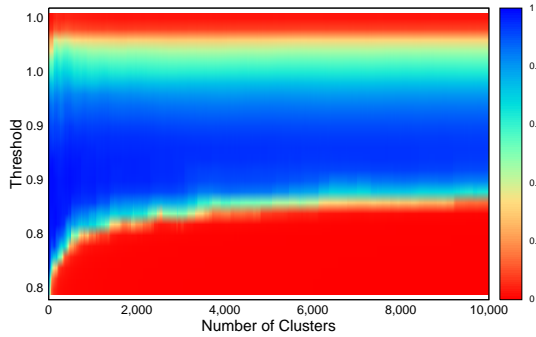
(d) Best threshold/F-Measure (Febrl (sm.), SNM algorithm)



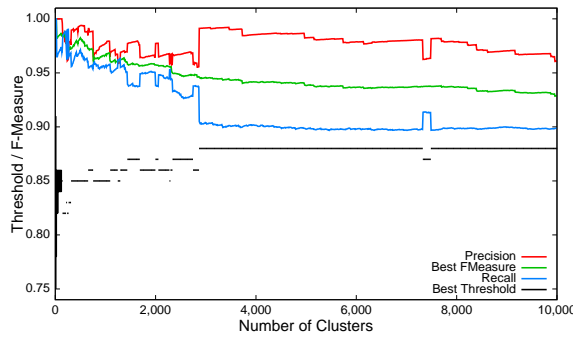
(e) F-Measure for Febrl (la.) with naive algorithm



(f) Best threshold/F-Measure (Febrl (la.), naive algorithm)

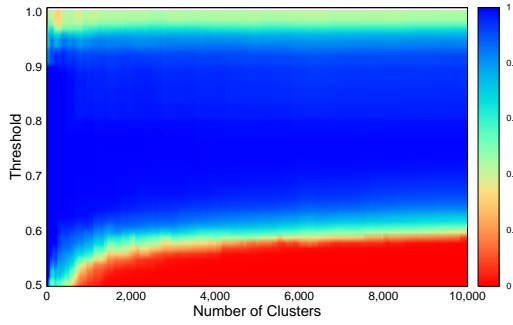


(g) F-Measure for Febrl (la.) with SNM algorithm

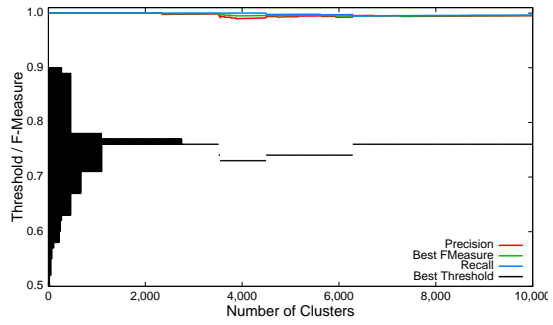


(h) Best threshold/F-Measure (Febrl (la.), SNM algorithm)

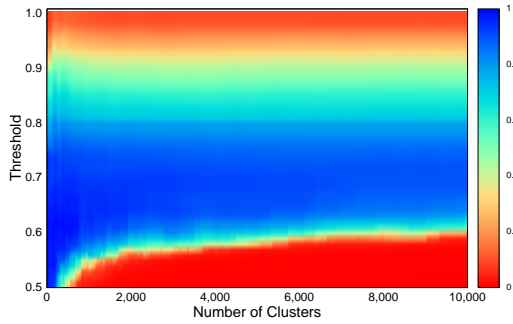
Figure 2: Results of experiments with the Febrl datasets and JaroWinkler measure. The figures show on the one side in a heatmap the F-Measure values for different numbers of clusters and different threshold values. On the other side, the figures show the best F-Measure values with the respective threshold.



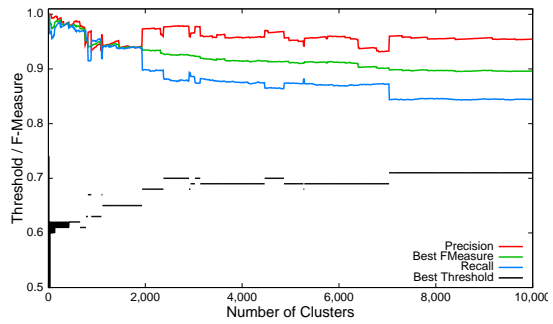
(a) F-Measure for Febrl (sm.) with SNM algorithm



(b) Best threshold/F-Measure (Febrl (sm.), SNM alg.)

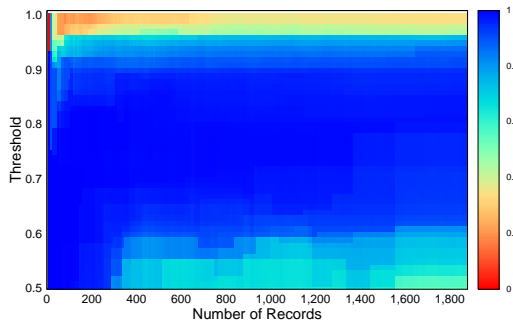


(c) F-Measure for Febrl (la.) with SNM algorithm

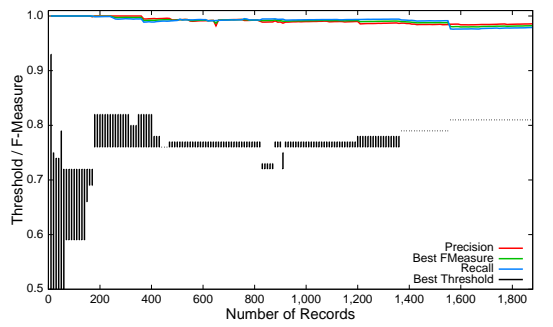


(d) Best threshold/F-Measure (Febrl (la.), SNM alg.)

Figure 3: Results of experiments with the Febrl datasets and Levenshtein measure.

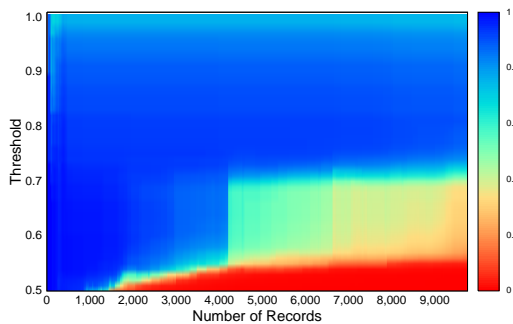


(a) F-Measure for Cora with SNM algorithm

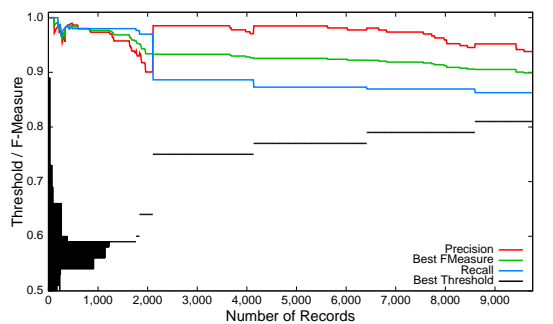


(b) Best threshold/F-Measure (Cora, SNM alg.)

Figure 4: Results of experiments with the Cora dataset.



(a) F-Measure for CD with SNM algorithm



(b) Best threshold/F-Measure (CD, SNM alg.)

Figure 5: Results of experiments with the CD dataset.

a good threshold for this dataset. But with an increasing number of clusters the window becomes smaller until it is only a single optimal threshold value. For even large datasets, the best threshold value increases slowly, although there are a few outliers as we can see in Fig. 2(b) and especially in Fig. 2(h). We can also see that the best F-Measure value decreases with an increasing number of clusters.

Figure 3 also shows results for the two Febrl datasets, but this time with the Levenshtein similarity and only for the Sorted-Neighborhood-Method as pair selection algorithm. We can see again, that for small cluster sizes we have a window for the selection of the best threshold. This window of optimal thresholds shrinks until it is only a single value. With an increasing number of clusters, this threshold value then increases. We also calculated for this similarity measure the results with the naive pair-selection algorithm. The results (not shown) are similar to those with SNM, so the results do not depend on the pair selection algorithm.

Figure 4 shows the results for the Cora dataset. Please note, that here we show the number of records on the x-axis. We also evaluated both pair selection algorithms, but again the results are very similar and we show only the results for the Sorted-Neighborhood-Method. Due to the used rules in the similarity measure, we achieve very high F-Measure values, as both precision and recall are high. Again, we can observe a window for the best threshold in the beginning that becomes smaller with an increasing number of records.

The results of the CD experiments also confirm our hypothesis, that with an increasing number of records the selection of good thresholds becomes more difficult. As for the Febrl and the Cora dataset, we first have a window for the best threshold, which becomes smaller and finally the best threshold value increases with the number of records. Thus, the best threshold changes with dataset size. A good threshold for a small (possibly sampled) dataset is not necessarily a good threshold for a larger (possibly complete) dataset. As data grows overtime, earlier selected thresholds are no longer a good choice.

To summarize, we have observed that the effect described in Figure 1 can be seen in all of our experiment configurations: We continuously add records to our datasets, which increases the probability that we add a false duplicate to an existing cluster. Thus, the F-Measure value decreases and an adaption of the threshold is necessary to improve it. The problem becomes worse if we have larger clusters, as we can see from the comparison of the Febrl datasets. The threshold window for good F-Measure values is smaller for the large Febrl dataset and additionally the F-Measure value decreases much faster. It is worth noting that the effect does not depend on the used similarity measure, as we can see from our results with the Febrl datasets and the JaroWinkler and Levenshtein similarity measures.

### 3 Related Work

The past decade has seen a renaissance of duplicate detection research, spawning many new algorithms, variations of old algorithms, use cases, and surveys, such as [4, 8]. Our contribution lies not in a new duplicate detection algorithm or framework, but rather in observing a general property of the duplicate detection process, namely the sensitivity of the selected threshold to an increase of the dataset size. Thus, in this section we discuss other work that attempts to evaluate duplicate detection algorithms, showing that most of them do not address this particular issue that is the focus of this paper.

Hassanzadeh et al. analyze several clustering algorithms for duplicate detection with the focus on finding an algorithm that is robust to the threshold with regard to the used approximate join [9]. They used 29 datasets with different sizes and evaluated different thresholds. However, they do not evaluate different pair selection algorithms and do not consider changes of the datasets sizes and their influence on threshold choice.

Köpcke et al. have also addressed the issue of comparing different duplicate detection approaches [13]. They state that the used configuration of the algorithms, including the threshold, is one of the decisive factors for the resulting match quality. So for non-learning based approaches, they had to optimize the used

thresholds. This optimization was done for a fixed dataset size, so their evaluation does not include the effect of adding records to a dataset. The same holds true for the comparison of learning-based approaches, for which they use different training set sizes, but do not vary the dataset size.

## 4 Conclusions

The goal of this paper was to show that the selection of a good threshold for the classification of record pairs as duplicate depends on the size of the dataset. This issue is relevant in practice, when the threshold is selected based on a small subset or when the dataset size increases over time.

Our experiments on four artificial and real-world datasets have shown that this problem is independent of the used datasets, similarity measures, or the pair-selection algorithm. Our next step will be to confirm our observations with further datasets and similarity measures. In a second step, we shall try to predict when and how the threshold should be adapted, for instance through interpolation of the best-threshold graphs. Such a step could be one building block to automate the significant manual overhead of configuring a duplicate detection program. Others have regarded automatically selecting blocking keys [1, 12, 15, 18] or similarity measures [2, 17].

So far, we have considered only non-learning based approaches. Köpcke et al. have shown for learning-based approaches that the training set size has a great impact on the F-Measure value [13]. Another research area is to evaluate whether the learning-based approaches are robust to an increasing test set size compared to non-learning based approaches. The goal could be to determine a suitable ratio of the training set and the data set size, that suggest when to repeat the training step with a larger training set size.

## References

- [1] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 87–96, 2006.
- [2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ACM SIGKDD International Conference of Knowledge Discovery and Data Mining*, pages 39–48, 2003.
- [3] P. Christen. Probabilistic data generation for deduplication and data linkage. In *International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, pages 109–116. Springer LNCS, 2005.
- [4] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2011.
- [5] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. ACM, 2005.
- [6] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, Lyon, France, 2009.
- [7] U. Draisbach and F. Naumann. DuDe: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.
- [8] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.



- [9] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293, 2009.
- [10] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
- [11] M. Herschel, F. Naumann, S. Szott, and M. Taubert. Scalable iterative graph duplicate detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(11), 2012.
- [12] B. Kenig and A. Gal. MFIBlocks: An effective blocking algorithm for entity resolution. *Information Systems*, 38(6):908–926, 2013.
- [13] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1):484–493, 2010.
- [14] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
- [15] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 440–445, 2006.
- [16] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection (Synthesis Lectures on Data Management)*. Morgan and Claypool Publishers, 2010.
- [17] T. Vogel and F. Naumann. Instance-based “one-to-some” assignment of similarity measures to attributes. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 412–420, 2011.
- [18] T. Vogel and F. Naumann. Automatic blocking key selection for duplicate detection based on unigram combinations. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2012.
- [19] W. E. Winkler and Y. Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. In *U.S. Decennial Census. Technical report, US Bureau of the Census*, pages 11–13, 1991.