Tuning Large Scale Deduplication with Reduced Effort

Guilherme Dal Bianco, Renata Galante, Carlos A. Heuser
Universidade Federal do Rio Grande do Sul
Porto Alegre, RS, Brazil
{gbianco,galante,heuser}@inf.ufrgs.br

Marcos André Gonçalves
Universidade Federal de Minas Gerais
Belo Horizonte, MG, Brazil
mgoncalv@dcc.ufmg.br

ABSTRACT

Deduplication is the task of identifying which objects are potentially the same in a data repository. It usually demands user intervention in several steps of the process, mainly to identify some pairs representing matchings and non-matchings. This information is then used to help in identifying other potentially duplicated records. When deduplication is applied to very large datasets, the performance and matching quality depends on expert users to configure the most important steps of the process (e.g., blocking and classification). In this paper, we propose a new framework called FS-Dedup able to help tuning the deduplication process on large datasets with a reduced effort from the user, who is only required to label a small, automatically selected, subset of pairs. FS-Dedup exploits Signature-Based Deduplication (Sig-Dedup) algorithms in its deduplication core. Sig-Dedup is characterized by high efficiency and scalability in large datasets but requires an expert user to tune several parameters. FS-Dedup helps in solving this drawback by providing a framework that does not demand specialized user knowledge about the dataset or thresholds to produce high effectiveness. Our evaluation over large real and synthetic datasets (containing millions of records) shows that FS-Dedup is able to reach or even surpass the maximal matching quality obtained by Sig-Dedup techniques with a reduced manual effort from the user.

Categories and Subject Descriptors

H.2 [**Database Management**]: System

General Terms

Algorithms, Experimentation

Keywords

Deduplication, Signature-based deduplication

SSDBM '13, July 29 - 31 2013, Baltimore, MD, USA

1. INTRODUCTION

Record deduplication is the task of identifying which objects are potentially the same in data repositories. Although an old problem, it still continues to receive significant attention from the database community due to its inherent difficulty, especially in the context of large datasets. Deduplication has an important role in many applications such as data integration [21, 19], social networks [28], Web crawling and indexing [35], etc. For instance, a service aimed at collecting scientific publications on the Web to create a central repository (e.g. Citeseer [18]) may suffer a lot in the quality of its provided services (e.g., search, recommendation) as there is a large number of replicated publications dispersed on the Web. Deduplication is applied to improve the data quality, identifying if a new collected object already exists in the data repository (or a close version of it).

Standard deduplication usually relies in the users to configure or tune the whole process. The user intervention, which may be required in several steps of the process, can be made *directly*, *indirectly* or combining both strategies. Approaches that require *indirect intervention* do not demand an expert user. A non-expert user just labels a set of pairs used for training a classifier to be applied to an unseen dataset [9]. The first challenge of such approach is how to select a "representative" set of pairs to create the training set which maximizes matching quality. For this task, active learning approaches have been proposed to automatically select a representative set of pairs [26, 1, 7]. A second challenge is how to efficiently generate pairs of candidate records. Since the all-against-all comparison is unfeasible, blocking strategies are usually applied to group together records that share common features, and only the pairs inside each group are considered for next steps [24, 5].

Blocking is essential to speed up the deduplication on large datasets. The problem is how to configure it. Usually a *direct intervention* is used to tune the blocking method (e.g., by setting proper similarity thresholds), implying that in most cases a combination of both *direct* (for blocking) and *indirect* (for labeling) intervention has to be performed. For instance, in both [1] and [7] active learning approaches are proposed for deduplication aimed at reducing the training set size. A blocking method, manually tuned by expert users, is used in both works along with the active learning approach.

The *direct intervention* requires that the user manually tune parameters or thresholds to configure the deduplication task. Most studies based on *direct intervention* focus on performance issues. For instance, [12, 6, 35, 32]) recently pro-

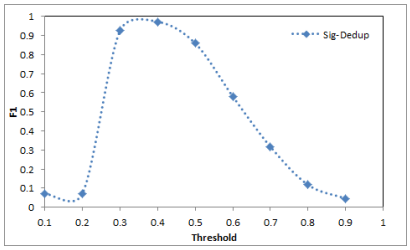

Figure 1: Matching quality when threshold is ranged.

posed efficient and scalable deduplication approaches that rely on inverted indexes. These approaches, collectively called Signature-based Deduplication (Sig-Dedup), supply the inverted indexes with heuristics and a set of filters, manually configured, which efficiently prune out a large number of non-matching pairs. However, the quality of the blocking and matching phases of Sig-Dedup depends on thresholds defined by expert users. The ideal threshold identification is a hard task that demands user knowledge about the noise level of the data and the specific deduplication approach to be applied. To illustrate this, Figure 1 shows the match quality[1] of a synthetic dataset when the threshold varies. Real world datasets rarely have gold standards and the users must try several configurations and assess somehow the output quality to identify the ideal threshold.

In this context, we propose a new framework, named FS-Dedup (Framework for Signature-based Deduplication), aimed at selecting the "optimal configuration" in the context of large scale deduplication tasks with a reduced manual effort. To identify the specific configuration of each dataset we only require a non-expert user to label a reduced set of automatically selected pairs. FS-Dedup incorporates state-of-the-art Sig-Dedup algorithms in its deduplication core to address high performance issues. Then, a set of strategies are proposed to help setting its parameters, removing most of the configuration details concerns from the user. In more details, the main contributions of this paper include: (*i*) a strategy to identify a blocking configuration that maximizes the number of true pairs; (*ii*) a greedy heuristics that separates candidate pairs into levels or subsets to be able to best identify where "ambiguous pairs" lie; and (*iii*) a final strategy to identify the best configuration of the classification method based on the reduced set of pair previously labeled.

We should stress that the user is only required to label a set of pairs, while FS-Dedup is responsible to tune the entire deduplication process (i.e. blocking and classification step). To demonstrate the effectiveness of our framework, we compare it with both the best manual configuration of the Sig-Dedup as well as with a state-of-the-art active learning for deduplication [7]. Experimental results on synthetic and two real datasets (one with about three million of records) show that FS-Dedup successfully achieves optimal effectiveness (in some cases, even superior to Sig-Dedup) with seven times less labeled pairs in the initial training set compared to [7].

The rest of this paper is organized as follows. We first introduce the background and then review related work. Our proposed framework is described in Section 3. Experimental results are presented in Section 4. We conclude the paper in Section 5 with glimpses at future work.

[1]The effectiveness is assessed by the F-1 metric and the dataset used is described in Section 4.

2. BACKGROUND AND RELATED WORK

In this section, we present the background and related work to FS-Dedup. First, we specify the main concepts of the Sig-Dedup algorithms adopted as deduplication core by FS-Dedup. Next, we explain the notion of *fuzzy region* that represents a subset composed of ambiguous pairs. Finally, we discuss related work closer to our approach.

2.1 Signature-Based Deduplication (Sig-Dedup)

The Sig-Dedup method has been proposed to efficiently handle large deduplication problems. Sig-Dedup maps the dataset strings into a set of signatures ensuring that pairs with similar substrings must contain overlapped signatures. The signatures are computed using the well-known inverted index method [4].

A straightforward deduplication task based on inverted indexes can be described as composed of four phases:(*i*) *preprocessing*: the dataset is fragmented and normalized into a set of tokens; (*ii*) *indexing*: the tokens are mapped into inverted lists, and each list blocks all records that share at least one token in common; (*iii*) *blocking*: each inverted list defines a block and within each block an all-against-all comparison is performed; (*iv*) *verification phase*: similarity functions measure the similarity degree among pairs. A matching pair is defined when it meets a predefined threshold.

The first drawback to produce inverted indexes for the deduplication tasks is that all tokens or sub-strings of each record must be evaluated. This evaluation produces candidate pairs with quadratic growth [35]. The second drawback is that the entire record must be analyzed before we can insert it into the index. This procedure can be expensive when the record is large.

Several works have addressed the reduction of the quadratic candidate generation [12, 6, 35, 31]. *Chaudhuri et. al.* [12], for instance, proposed the *prefix filtering*, aimed at pruning out large number of unrelated pairs, indexing only the less frequents tokens of each record. More specifically, the *preprocessing phase* tokenizes each record using words or NGrams (sub-strings) to create token sets. The NGram tokenization can improve the effectiveness in datasets with short character mistakes compared with the word tokenization. However, it produces both, inverted index structures with vast number of tokens and large blocks, increasing the computational demands. In the *indexing phase*, each token set is sorted following the global frequencies (global ordering ϑ). The sorted set, called signature vector, aims at identifying which tokens are rarer and which ones are more frequent in the dataset (e.g. stop words). In the *indexing phase* the less frequent tokens are indexed, selecting only the prefix of each signature vector. It produces blocks with fewer records and, consequently, reduces the number of candidate pairs. More formally, the *prefix filtering* is defined as below:

DEFINITION 1. *Assume that all the tokens in each record are ordered by a global ordering ϑ. Let p-prefix of a record be the first p tokens of the record. If Jaccard(x,y)$\geq$ t then*

the (n)-prefix of x and (n)-prefix of y must share at least one token. [12]

Besides *prefix filtering*, *length filtering* may also be applied to remove records whose length variation is higher than a specified threshold [6]. Xiao et al. [35] observed that after the insertion of the prefix filtering the number of candidate pairs continues with quadratic growth. Thus, two additional filters are proposed to reduce the number of candidate pairs. The first, called *positional filtering*, verifies the token position variations in the prefix of signature vector. The second, called *suffix filtering*, verifies if the candidate pairs have position variations using a recursive binary search in the suffix of the signature vector.

Overall, Sig-Dedup approaches represent the state-of-the-art in large scale deduplication task. Bayardo et al. efficiently performed a deduplication with millions of records [35]. However, the effectiveness of Sig-Dedup based approaches depends on the definition of the threshold values, defined by expert users. FS-Dedup aims exactly at identifying such thresholds, extracting the maximum effectiveness with reduced effort from the user, as we shall see in our experimental evaluation.

2.2 Fuzzy Region

Fellegi and Sunter (1969) developed the first statistical foundations of deduplication. The *Fellegi and Sunter* definition is used as theoretical grounds for several approaches (e.g. [13, 14]). Researches based on Fellegi and Sunter relies on the definition of two boundaries, named α and β. These boundaries, defined by the user, produce three sets: A_1 contains the pairs above β; A_3 composed of pairs below α; A_2 is composed of the pairs between α and β. The set A_1 is sent to the output as matching pairs, the set A_3 is discarded, and the set A_2 must be sent to the human judgment due to the high degree of ambiguity. We called set A_2 the *fuzzy region*.

In large scale deduplication, the *fuzzy region* may contain a large number of pairs, turning infeasible the human judgment. The manual definition of both boundaries α and β may also be imprecise. Our framework proposes a strategy to identify the *fuzzy region* by using a reduced training set. FS-Dedup configures the classifier with such training set, providing a more practicable identification of the matching pairs inside the *fuzzy region*.

2.3 Related Work

Deduplication methods use a variety of solutions, ranging from those focus on supervised strategies to those that use unsupervised approaches. In this section, we present a brief review of some related methods. For more thorough reviews, see [16, 23, 34, 20, 15].

Supervised approaches require as input a training set composed of pairs labeled as matching or non-matching. The training set is used by the classifier to tune a model that is applied to the unseen dataset to predict the matching pairs. For instance, Bilenko et al. [9] proposed a framework (Marlin) that combines multiple similarity functions with the Support Vector Machine (SVM) learning algorithm, focusing on the learning process. They also show that a technique based on SVM outperforms others machine learning approaches (i.e. decision trees). Chaudhuri et al. [11] combined rules and similarity functions to produce a decision tree classifier that extracts the information on a given training set. Wang et al. [33] observed that [11] does not scale well due to redundancy when a large number of similarity functions is used. Then, they proposed a new efficient rule based method to select the best threshold and similarity function able to remove equivalent similarity functions and thresholds. These approaches assume that training set ideally contains the diversity and most important patterns present in practice. However, such training set is hard to obtain, even for expert users, especially in large real world datasets. FS-Dedup tries exactly to tackle such issues.

In recent years, several studies have been developed aimed at creating training sets for deduplication with reduced manual effort, by exploiting *active learning*. Such approaches are used to actively select the most informative pairs to learn a classifier. Most works on active learning address the 0-1 loss metric[2]. These researches can not be directly applied to the deduplication because of the number of non-matching far exceeds the number of matching pairs [1, 7]. In [27] and [29] active learning approaches for deduplication based on a committee of learners are proposed. The uncertainty between the committee members defines if the pair will be queried. Arasu et al. [1] examine two previous solutions [27, 29] and point out their limitation with respect to the ability of creating a representative training set when large datasets are matched. Then, they propose to create a N-dimensional feature space and actively select pairs using binary search over this space.

Recently, Bellare et al. [7] proposed a strategy to map any active learning approach based on 0-1 loss to an appropriate deduplication metric under precision constraints. Such strategy, here referred as ALD, projects the quality estimation of the classifiers as points in a two-dimensional space. A binary search is applied over the space to select the optimal classifier under precision constraints. The dimensions of the space correspond to the classifiers effectiveness, estimated using an "oracle" of pairs randomly selected and manually labeled. ALD invokes the IWAL active learning [8] to select the informative pairs and produce the classifiers. IWAL identifies the pairs to be manually labeled by assigning a probability based on the divergence between the current best hypothesis and an alternative hypothesis (i.e. the hypothesis that predict the pairs as a non-matching). To define the training set size, IWAL adopts a stop criterion which is estimated by a threshold manually defined by the user. Bellare et al. performed experiments showing that their approach outperforms their baseline [1], delivering state-of-the-art performance in active learning for deduplication.

The aforementioned methods address deduplication in large datasets by incorporating a blocking step, manually tuned. This step may lead to poor performance or bad matching quality if not properly done. In this paper we propose to deal with the blocking and classification steps as a single task, avoiding that a non-optimal user configuration degrades the matching quality. FS-Dedup also identifies the region where the ambiguous pairs are concentrated in order to automatically select pairs to compose a reduced training set. Although not completely fair to our approach, since the active learning baselines use manually tuned blocking thresholds and provides precision guarantees, we also include the state-of-the-art ALD method as one of our baselines as they also intend to minimize user effort.

[2]The 0-1 loss computes matching quality based on fractions of the pairs that are incorrectly classified.

Unsupervised approaches for deduplication do not use a training set and typically employ similarity functions to quantify the similarity between pairs. As discussed in Section 2.1, methods based on inverted indexes have been used to improve the efficiency in large scale deduplication, called signature-based deduplication (Sig-Dedup). The first important technique to speed up the Sig-Dedup algorithms is the prefix filtering principle, proposed by [12]. Sig-Dedup has been further enhanced by several recent proposals [12, 6, 35, 31, 32]. Overall, these approaches aim at reducing the number of candidate pairs combining sophisticated filters and heuristics. Additionally, in [30] the Sig-Dedup algorithm proposed by [35] is extended to the MapReduce platform. Awekar et al. [3] proposed a technique, using the Sig-Dedup algorithm, to reduce the runtime when the user submits several executions, varying the threshold value. It stores a log to discard pairs previously evaluated.

Nevertheless, studies that focus on Sig-Dedup approaches neglect the problem of selecting the best configuration. Even for an expert user the number of possible threshold alternatives may be very large. In this work, we extend Sig-Dedup focusing on the identification of the optimal configuration. Our framework combines a set of strategies to configure the Sig-Dedup algorithms (including the blocking step), using a reduced set of pairs automatically selected to be manually labeled. The framework proposed in this paper is orthogonal to the specific optimization used in the Sig-Dedup algorithms and can be benefit from new Sig-Dedup improvements.

3. FS-DEDUP - A FRAMEWORK FOR SIGNATURE-BASED DEDUPLICATION

In this section, we present our proposed framework for signature-based deduplication, named FS-Dedup, which is able to tune most of the deduplication process in large datasets with a reduced user effort. From the point of view of the user, FS-Dedup can be seen as a single task, avoiding an expert user intervention in specific steps (i.e. blocking and classification phases). The non-expert user intervention is requested only to label a set pairs automatically selected by our framework.

In the following, we provide an overview of FS-Dedup steps, illustrated in Figure 2:

1. *Sorting step:* in this step, the dataset is blocked to create a sorted set of candidate pairs, without user intervention. The challenge of such step is to avoid an excessive generation of candidate pairs. We propose a strategy to identify the threshold to configure this step focused on recall maximization.

2. *Selection step:* identifies the *fuzzy region* boundaries. A greedy strategy to define the *fuzzy region* boundaries is proposed to automatically select candidate pairs to be labeled by a non-expert user with the goal of reducing effort. After defining the *fuzzy region* boundaries, the pairs inside the *fuzzy region* are sent to the *Classification step*. The set below the *fuzzy region* is discarded while the set above is automatically sent to the output as matching pairs.

3. *Classification step:* classifies the candidate pairs that belong to the *fuzzy region* as a matching or not. Two classification methods are used: *FS-Dedup-NGram* and *FS-Dedup-SVM*. *FS-Dedup-NGram* applies substring tokenization to detect matching pairs with short character mistakes. *FS-Dedup-SVM* uses Support Vector Machines (SVM) to identify the matching pairs.

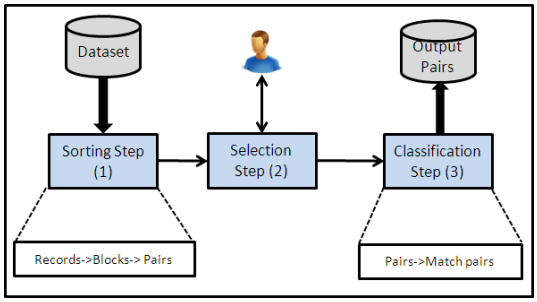

Figure 2: FS-Dedup steps overview

Next, we detail each step of FS-Dedup.

3.1 Sorting step

The *Sorting step* identifies the blocking threshold using the Sig-Dedup filters (e.g., regarding the number of tokens to be used) that maximize recall. We call such blocking threshold the *initial threshold*. Ideally, the set of candidate pairs produced using the *initial threshold* should be composed of only matching pairs. As the *Sorting step* is performed without user intervention, we use generalization to be closer to the ideal scenario. Notice that, also to avoid user intervention, the *initial threshold* represents a single global threshold for all blocks.

It is noteworthy that the set of candidate pairs are produced using the Sig-Dedup filters (i.e. prefix, length, position, and suffix filtering) configured with the *initial threshold*. Such threshold mainly defines how many tokens are indexed by the sorted record. Notice that in this step, the similarity value of each pair is not used to prune out pairs since we do not know the exact threshold value able to discard non-matching pairs. Additionally, we assume that in large datasets (composed of several millions of records) the number of matching pairs represents an small subset of dataset. For instance, two identical datasets that are matched have less matching pairs (e.g. two million of pairs) than the total number of records in such datasets (e.g. two millions of record from each dataset). This generalization does not hold when a record has several thousands of duplicates, producing true pairs almost quadratically. It means that the dataset is mainly composed of redundant information, which is very unexpected in real data repositories.

In large datasets, performing the Sig-Dedup filters with different thresholds can be infeasible due to high computational costs. Therefore, we propose a stop criterion to estimate the global *initial threshold* (th). A random subset (with size empirically estimated) is selected from the dataset. This subset is matched using a variable threshold which varies in fixed ranges. The stop criterion specifies that the number of pairs satisfying the Sig-Dedup filters must be lower than the subset size. The random subset naturally decreases the number of true matching pairs compared to the entire dataset. Thus, if the Sig-Dedup filters create more candidate pairs than the input set, we have a clear indica-

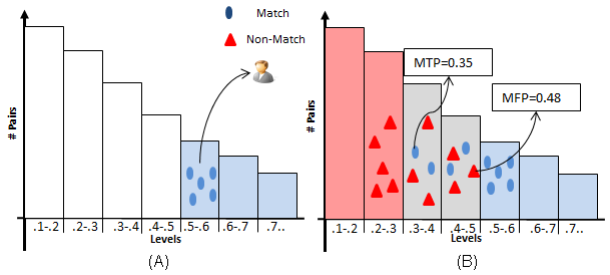

Figure 3: Illustration of the Sample Selection and Manual Labeled strategies.

tion that frequent tokens have indeed been indexed. When the threshold value is incrementally increased, less tokens in the sorted record are indexed, reducing the number of candidate pairs. On the other hand, a high threshold value selects few tokens in the sorted record and a lot of matching pairs can be pruned out. The stop criterion produces a threshold to avoid both a large generation of candidate pairs and recall degradation. We formalize this notion below:

DEFINITION 2. *Consider a subset s, created from a dataset D random sampling and a range of thresholds with fixed step th_j=0.2, 0.3, 0.4, ..., and 0.9. For each threshold th_j, the respective subset s is matched using th_j. The initial threshold will be the first th_j that results in a number of candidate pairs smaller than the number of records in s.*

After defining the global *initial threshold* value for the blocking process, the entire dataset is matched to create the set of candidate pairs. At the end, the candidate pairs are sorted using the similarity value to create a ranking.

3.2 Selection step

The *Selection step* identifies the boundaries of the *fuzzy region* which, to be effectively defined, depends on two main factors: (i) the quality of the sample selection of candidate pairs to be manually labeled (ideally, the sample should be able to describe the factors to identify the *fuzzy region*) which should be representative of the whole dataset; and (ii) the expected manual labeling effort which should be minimized without an inaccurate boundary definition. Our selection step addresses both factors at the same time, as we detail next.

3.2.1 Sample selection strategy

The sample selection strategy creates a balanced set of candidate pairs. We propose to discretize the ranking of candidate pairs generated in the Sorting step into fixed levels, in order to avoid that non-matching pairs dominate the sample selection. The fixed *levels* contain a subset of candidate pairs, making easier to determine the boundaries of the *fuzzy region*. More specifically, the ranking, created in the *Sorting step* is fragmented into 9 *levels* (0.1-0.2, 0.2-0.3,..., and 0.9-1.0), using the similarity value of each candidate pair. Inside each *level*, we randomly select candidate pairs to create the sample set to be manually labeled.

3.2.2 Manual labelling strategy

The proposed manual labeling strategy effectively identifies the *fuzzy region* boundaries, as we shall see. The strategy takes advantage of the sample selection strategy (described before) avoiding requesting the user to label unnecessary levels as a whole. One sample at each time is manually labeled to define the *fuzzy region* boundaries. In the following we describe in details the proposed process to identify the *fuzzy region*:

DEFINITION 3. *Let Minimum True Pair-(MTP) represent the matching pair with lowest similarity value among the set of candidate pairs.*

DEFINITION 4. *Similarly, let Maximum False Pair-(MFP) represent the non-matching pair with highest similarity value among the set of non-matching pairs.*

The *fuzzy region* is identified using pairs manually labeled. The user is requested to manually label a sample randomly selected from each level, called *labeling of levels*. However, the sample, labeled by the user, may result in a MTP and MFP far from ideal. To minimize this problem, we assume that the levels that the pairs MTP or MFP belong to define the *fuzzy region* boundaries. For instance, if the MTP and MFP values are 0.35 and 0.75, all the pairs with similarity value between 0.3 and 0.8 belong to the *fuzzy region*. We call the *fuzzy region* boundaries of α and β.

A straightforward method to find the MTP and MFP is label one sample from each level. However, some levels above and below the *fuzzy region* may not be informative to identify the α and β values, wasting manual effort. Thus, we propose to start the *labeling of levels* from an *Intermediate Level* (IL) with similarity values between [0.5-0.6]. After labeling *IL*, three scenarios may happen:

1. *IL* contains only non-matching pairs meaning that the *fuzzy region* belongs to levels above *IL*. Thus, only the levels above *IL* are manually labeled until we achieve a level that contains only matching pairs, thus identifying the MFP and MTP values.

2. *IL* contains only matching pairs. Thus, the *fuzzy region* belongs to the levels below the *IL*. Only levels below *IL* are labeled until we reach a level that contains only non-matching pairs.

3. If *IL* contains both matching and non-matching pairs, the *fuzzy region* belongs to the levels above and below *IL*, as described in the Scenarios 1 and 2. Thus, the levels above the *IL* are manually labeled until we reach a level that contains only matching pairs, identifying the MFP values. And, the levels below the *IL* are manually labeled until we achieve a level that contains only non-matching pairs, identifying the MTP value.

Algorithm 1 details the strategy to identify the *fuzzy region*, described above. Initially, the function *LabelRandomSetofPairs(L_i)* (Lines 2,6,17,28, and 34) selects a random set of candidate pairs inside Level i and these pairs are then presented to the user for labeling as a true (matching pair) or false (non-matching pair). The function *SelectHighestFalsePair* and *SelectLowestTruePair* selects the MTP and MFP within the input level, following definitions 3 and 4. The algorithm starts by identifying whether the *IL* belongs to *Scenario 1, 2, or 3*. If the *IL* is just composed of true pairs (Line 3) then the *fuzzy region* is defined as being below the *IL* (Scenario 1). Next, the levels below i are labeled until it achieves a *level* that has only false pairs (Lines

Algorithm 1 Finding the Fuzzy Region boundaries

Require: Set of levels $L = l_1, l_2, l_3..l_9$
1: $i \leftarrow 5$; $MFP \leftarrow Null$; $MTP \leftarrow Null$;
2: $L_{Pi} \leftarrow$ LabelRandomSetofPairs(L_i)
3: **if** L_{Pi} contains only *True* **then** { **SCENARIO 1**}
4: **while** L_{Pi} does not contains only *False* pairs **do**
5: $i \leftarrow i$-1
6: $L_{Pi} \leftarrow$ LabelRandomSetofPairs(L_i)
7: **if** L_{Pi} not contains only *False* and $MFP = Null$ **then**
8: $MFP \leftarrow SelectHighestFalsePair(L_{Pi})$;
9: **end if**
10: **end while**
11: $MTP \leftarrow SelectLowestTruePair(L_{Pi+1})$;
12: **return** MTP, MFP and L_P
13: **end if**
14: **if** L_{Pi} contains only *False* **then** { **SCENARIO 2**}
15: **while** L_{Pi} does not contains only *True* **do**
16: $i \leftarrow i$+1
17: $L_{Pi} \leftarrow$ LabelRandomSetofPairs(L_i)
18: **if** L_{Pi} does not contains only *True* and $MTP = Null$ **then**
19: $MTP \leftarrow SelectLowestFalsePair(L_{Pi})$;
20: **end if**
21: **end while**
22: $MFP \leftarrow SelectHighestFalsePair(L_{Pi-1})$;
23: **return** MTP, MFP and L_P
24: **end if**
25: **if** L_{Pi} contains *False* and *True* **then** { **SCENARIO 3**}
26: **while** L_{Pi} does not contains only *True* **do**
27: $i \leftarrow i$+1
28: $L_{Pi} \leftarrow$ LabelRandomSetofPairs(L_i)
29: **end while**
30: $MFP \leftarrow SelectHighestFalsePair(L_{Pi-1})$;
31: $i \leftarrow 5$
32: **while** L_{Pi} does not contains only *False* **do**
33: $i \leftarrow i$-1
34: $L_{Pi} \leftarrow$ LabelRandomSetofPairs(L_i)
35: **end while**
36: $MTP \leftarrow SelectLowestTruePair(L_{Pi+1})$;
37: **return** MTP, MFP and L_P
38: **end if**

4-9). The first level composed of both true and false pairs defines the MFP pair (Line 8). The lowest level composed of true and false pairs defines the MTP pair (Line 11). If the *IL* has only false pairs (Line 14), the *fuzzy region* is above the *IL* (Scenario 2). Thus, levels above the *IL* contains the MTP and MFP pairs. In *Scenario 3* (Lines 25-38), the *fuzzy region* is on the *IL*. Thus, the levels above the *IL* are labeled (Lines 26-29) until it reaches a level composed only of true pairs. The level before the *IL* contains the MFP pairs (Line 30). To identify the MTP, levels below the *IL* are labeled until it reaches a level composed only of false pairs. The level before the *IL* defines the MTP pair (Line 36). The algorithm output is the MTP, MFP and the labeled pairs.

The similarity value of the pairs MFP and MPF identifies α and β values. For instance, Figure 3 illustrates *Scenario 1*. First, the *IL* is manually labeled (Figure 3-A). Such level contains only matching pairs (represented by the blue circles). Next, the *level* [.4-.5] is labeled and it is discovered that it is composed of both matching and non-matching pairs. The MFP is defined by the pair with similarity value of 0.48 (highest non-matching pairs). This level defines β as 0.5, representing the end of the *fuzzy region* (Figure 3(B)). The *labeling of levels* continues until it identifies that level [.2-.3] contains only non-matching pairs. The MTP is defined by pairs with similarity value of 0.35 and α receives the value 0.3. Finally, the *fuzzy region* is created by all candidate pairs with similarity value between α and β.

After the boundaries are defined, the pairs belonging to the *fuzzy region* are sent to the *Classification Step*. This process substantially reduces the number of candidate pairs due to the pruning of the non-matching pairs with similarity below α. The set above β is sent to the output as matching pairs as the labeling strategy indicates that only matching pairs are present in such set.

3.3 Classification step

The *Classification step* aims at categorizing the candidate pairs belonging to the *fuzzy region* as a matching or non-matching. Two classifiers are used in this step: FS-Dedup-NGram and FS-Dedup-SVM. FS-Dedup-NGram maps each record to a global sorted token set and then applies both, the Sig-Dedup filtering and a defined similarity function, over the sets. The token set does not consider the attribute positions, allowing exchange of attributes values. The FS-Dedup-NGram drawback is that different attributes receive the same importance (i,e., have the same weights). In other words, an unimportant attribute value with large length may dominate the token set, producing distortions in the matching. On the other hand, FS-Dedup-SVM assigns different weights to different attributes of the feature vector, by using the SVM algorithm, based on their relative discriminative power. However, there is not a unique and globally suitable similarity function able to be adapted to different application [33], making difficult to configure the method for different situations. Moreover, long text attributes may be mapped to non-appropriated feature values causing loss of information in the classification process.

As both methods have advantages and drawbacks, we exploit both of them in our FS-Dedup framework. FS-Dedup-SVM and FS-Dedup-NGram are both trained using the labeled set created in the *selection step*. The training set is composed of only pairs within the *fuzzy region*. This is because the pairs below and above *fuzzy region* represent highly similar or dissimilar pairs and they do not promote training set diversity, important for difficult ambiguous cases. The candidate pairs predict as matchings are sent the output of this step.

FS-Dedup-NGram aims at identifying from the labeled sample where the matching and non-matching are concentrated to select the threshold that removes the non-matching pairs. Thus, to identify the matching pairs inside the *fuzzy region* using the NGram tokenization, yet another specific threshold is required: the *NGram Threshold*. In the following, we describe the proposed strategy to automatically identify the *NGram Threshold*:

1. The similarity of each labeled pair is recomputed using a similarity function along with the NGram tokenization.
2. The labeled pairs are sorted incrementally by the similarity value.
3. A sliding window with fixed-size N is applied over the sorted pairs. The sliding window is relocated in one position until it achieves the last windows with only non-matching pairs. The similarity value of the first matching pair after the last windows with only non-matching pairs defines the *NGram threshold* value.

Notice that we can only perform such procedure because we have the labels of the pairs given by the user in the previous Selection step. Sig-Dedup is configured with the *Ngram*

Table 1: Datasets statistics.

Dataset	n	# match	avg_len	|U|
Clean	105,000	5,000 (5%)	82	235,362
Dirty-Clean	120,000	20,000 (20%)	81	257,083
Dirty	150,000	50,000 (50%)	82	291,639
DBLP	1,995,539	-	169	2,013,244
Citeseer	811,408	-	129	644,479
IMDB	1,080,000	-	55	551,583
NetFlix	160,000	-	76	70,654

threshold value to be applied to the all *fuzzy region* pairs. Finally, the candidate pair that survives the filtering phase and meet the *Ngram threshold* value is output as matching pairs.

To execute FS-Dedup-Ngram classifier we set the Sig-Dedup process to disable the blocking phase since the *Classification step* already receives candidate pairs as input. Although Ngram tokenization increases the computational demands compared to word-level tokenization used in the Sorting Step, it is only applied in the *fuzzy region* pairs, thus minimizing the computational costs.

4. EXPERIMENTAL EVALUATION

In this section we present our experimental results which demonstrate the effectiveness and reduction in manual efforts promoted by our framework. We first describe the used datasets, the evaluation metrics, the configuration of our framework and the baselines. Then, we compare the effectiveness and manual label efforts of our proposed strategy in comparison with the baselines.

4.1 Datasets

To evaluate our framework, we use synthetic and real datasets. The real datasets are created merging two other different real datasets in the same domain to produce a deduplication scenario. Since real datasets do not have gold standards, we also use synthetic datasets to create controlled scenarios in order to better evaluate the methods.

We create synthetic datasets by using the Febrl Dsgen tool [13]. This generator works by first creating the original records based on real world vocabularies (e.g. names, surnames, street name, among others). Next the original records are randomly changed (e.g. merging words, inserting character mistakes, deleting attributes) to create the matching pairs. Errors are based on noise patterns found on real data. We simulate three scenarios: the *clean* dataset contains 5% of duplicates; the *dirty-clean* dataset contains 20% of duplicates; and the *dirty* the dataset contains 50% of duplicates. The records are synthetically built with 10 attributes: "first name", "surname", "age", "sex", "address", "state", "street number", "phone number", "date of birth", and "social security number".

Since large scale real datasets are not easily available with gold standards, due to privacy and confidentiality constraints, we manually create two real dataset. The first dataset, named *IMDBxNetflix*, was created accessing the public APIs of both IMDB[3] and NetFlix[4]. Such datasets store information about movies. The IMDB dataset has more than a million records and the public available Netflix dataset contains about 160,000 records. Netflix focus on specific market information, therefore it can not be considered as a "pure" subset of IMDB movies [17]. Only a small fraction of both datasets represents matching pairs. We integrate the datasets using the common attributes: *title, director, and release year*.

The second real dataset, named as *DBLPxCiteseer*, is made by merging the DBLP [5] and the Citeseer[6] datasets. Both, DBLP and Citeseer are digital libraries storing information about scientific (Computer Science) publications. The Citeseer dataset indexes new entries by automatically crawling web pages [18]. In such scenario, the Citeseer entries are affected by different levels of noise, for example, incomplete attributes, different name conventions, and other impurities. Contrarily, DBLP entries are manually inserted, resulting in higher quality [25]. We produced the DBLPx-Citeseer dataset using the attributes *title, author, and publication year*. Table 1 summarizes the synthetic and real dataset configurations.

4.2 Evaluation Metrics

The matching quality of deduplication algorithms is usually assessed by the following standard measures [22]: *precision, recall* and *F1. Precision* measures the rate of pairs correctly identified in the entire dataset. Recall measures the rate of correct pairs compared with all true pairs. The *F1* is the harmonic mean between *recall* and *precision*.

For the synthetic datasets, we run each experiment 10 times, reporting the means and the standard deviation. The results are compared using statistical significance tests (t-test pared) with a 95% confidence interval. As the real datasets do not have gold standards we asked 5 users (Computer Science graduate students) to label the training set containing 100 pairs sampled from each level, resulting in 900 pairs. For both datasets we create the test set manually evaluating all candidate pairs from the years 1988, 1989, and 1990 [2], resulting in 3,009 and 3,137 pairs in the DBLPxIMDB and IMDBxNetFlix datasets, respectively. The pairs with incomplete or ambiguous information were labeled using additional information e.g. the respective websites, metadata, among other types of information.

4.3 Experimental setup

To run FS-Dedup we used the Sig-Dedup implementation PPJoin++ [35] as deduplication core. We configured PPJoin++ with NGram tokenization and used Jaccard as the similarity function. We experimentally identifies the *sliding window* size of two for synthetic datasets and five for the real datasets as the best ones. All matching pairs are known in the synthetic datasets resulting in a small *sliding window* value. On the other hand, the gold standard created by the users for the real datasets may be susceptible to label mistakes. Therefore, to avoid distortions in the classifier configuration we set a *sliding window* value of five in these datasets.

For FS-Dedup-SVM, we used the libSVM package [10] as an implementation of SVM algorithm. We used a RBF kernel and the best parameters γ and cost were obtained with cross-validation in the training set. Features were created computing NGram-based Jaccard similarity which is known to be efficiently computable [16].

[3]http://www.imdb.com
[4]http://www.netflix.com
[5]http://www.informatik.uni-trier.de/ ley/db/
[6]http://citeseer.ist.psu.edu/

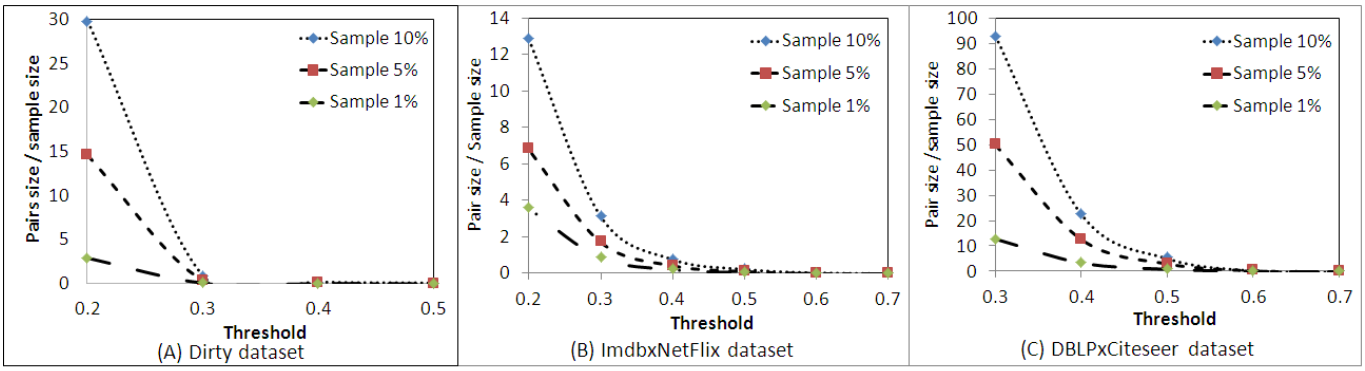

Figure 4: Number of candidate pairs created by various *initial thresholds*.

Table 2: Recall of Sig-Dedup filters in the dirty dataset.

Dataset	Th	Recall	#pairs
Dirty	0.2	0.990	43,629,641
	0.3	**0.979**	**2,776,461**
	0.4	0.935	263,872
	0.5	0.833	90,292

Table 3: Recall of Sig-Dedup filters in the IMDBxNetFlix dataset.

Dataset	Th	Recall	# pairs
IMDBxNetFlix	0.3	1.00	33,730,220
	0.4	**1.00**	**8,381,906**
	0.5	0.99	2,340,065
	0.6	0.94	243,109

Table 4: Recall of Sig-Dedup filters in the DBLPxCiteseer dataset.

Dataset	Th	Recall	# pairs
DBLPxCiteseer	0.5	1.00	69,313,296
	0.6	**0.99**	**13,180,090**
	0.7	0.9	2,122,995
	0.8	0.7	317,818

As a baseline, we also considered the active learning algorithm proposed in [7]. An implementation of ALD is available by request. We defined the precision threshold value as 0.85 and the oracle was defined with size of 100 pairs, as suggested in the original work. The active learning thresholds tested, as used in the author implementation, include the following values: 1.10^{-3}, 1.10^{-4}, ...,1.10^{-6}, that define the final training set size. Additionally, we test the thresholds 1.10^{-7}, 1.10^{-8}, and 1.10^{-9} to minimize the final training set size. The set of candidate pairs was the same produced by the Sorting step and the features were generated in the same way as for FS-Dedup-SVM.

4.4 Identifying the Initial Threshold

In these experiments, we evaluated the strategy for creating the set of candidate pairs. Since in this step we aim at maximizing recall while avoiding an excessive generation of candidate pairs, we focused on the identification of the *initial threshold* value following *Definition 2* (Section 3.1). We run experiments using random sample with 1%, 5%, and 10% of entire dataset.

Figure 4 reports the results for the *Dirty*, DBLPxCiteseer and IMDBxNetFlix datasets. In the Y axis, the *Pair size/Sample size*(P/S) refers to the rate of candidate pairs created over the number of records in the sample. We adopt such normalization to summarize and better illustrate the number of candidate pairs created by each threshold value. If P/S value is lower than one then the number of candidate pairs is lower than the sample size, as considered in Definition 2. Since all synthetic datasets present similar behavior (i.e. use the same *initial threshold* value), we detail here just the results for the *Dirty* dataset.

We can notice that, in these first experiments, the synthetic and real datasets produced different curves regarding the size of the set of candidate pairs. This is somewhat expected due to the data patterns and noise levels in each dataset. The *Dirty* dataset (Figure 4-A) produces a large set of candidate pairs when using a 0.2 threshold value. This happens because a substantial number of frequent tokens (e.g. stop words) are indexed with a threshold value of 0.2. For instance, the 0.2 threshold with sample size of 1% produces a P/S value of 30 (about 450,000 pairs from 15,000 records). On the other hand, considering a threshold value of 0.3 reduces substantially the set of candidate pairs with P/S value lower than one. This qualifies the 0.3 threshold to be used as the *initial threshold* in the synthetic datasets.

To evaluate the recall reached by the *initial threshold*, we explored the threshold values of [0.2, 0.3, 0.4, and 0.5] in the entire dataset, shown in detail in Table 2. Note that the threshold value of 0.2 retrieves almost all true pairs with drawback of 16 times more candidate pairs than the *initial threshold* (0.3 threshold). The *initial threshold* prunes out only 2% of true pairs, while threshold values of 0.4 and 0.5 remove about 6.9% and 20% of the true pairs. Our strategy successfully finds the *initial threshold* value that maximizes recall with a feasible set of candidate pairs to be labeled.

The IMDBxDBLP dataset (Figure 4-B) produces relatively less candidate pairs than the other datasets. This is because the NetFlix dataset contains almost nine times less records than the IMDB one, resulting in a relative low number of candidate pairs. Note that the threshold value of 0.3 with sample size of 1% produces P/S value lower than one (about 12,400 pairs), qualifying 0.3 threshold value to be used as *initial threshold*. However, the same threshold (0.3) with a sample size of 5% and 10% produces a P/S value of 2.0 and 3.8, respectively. This happens because the

1% sample is not big enough to select the frequent tokens. Then, we adopted a default sample size of 5%. In such scenario, the threshold valueof 0.4 becomes qualified to be used as the *initial threshold* as its P/S value is lower than one. Table 3 shows the recall and the number of candidate pairs produced by each threshold in the entire dataset. The *initial threshold* recovers all true pairs. The threshold value of 0.3 recovers all true pairs with drawback of producing more than 24 millions of candidate pairs that the set created by the *initial threshold*.

The DBLPxCiteseer dataset results (Figure 4-C) show a higher number of candidate pairs compared with other datasets. For instance, a 0.3 threshold value with a sample of 10% produces a P/S value of 90. The large number of candidate pairs in DBLPxCiteseer dataset is a combination of both, large dataset size and record length (as illustrated in the Table 1). Such characteristics force the Sig-Dedup filters to be more aggressive, avoiding the indexing of more frequent tokens. In this context, 0.6 value is defined as *initial threshold* since it is the first one to produce a P/S value lower than one, following *Definition 2*.

Table 4 illustrates the recall behaviour as we vary the threshold value in the entire DBLPxCiteseer dataset. Recall close to the maximum is achieved by the *initial threshold* with value of 0.6. When the threshold value is changed to 0.7, recall drops by almost 10%. The 0.5 threshold value recovers all true pairs with drawback of almost eight times more candidate pairs than the selected initial threshold value.

Overall we claim that the proposed strategy properly identifies the *initial threshold* value that achieves the maximum (or close to the maximum) recall and a feasible set of candidate pairs, without user intervention. For instance, the cartesian product between DBLP and Citeseer results in about $1.6 * 10^{12}$ candidate pairs. Our strategy produces about $1.3 * 10^{6}$ candidate pairs, missing only 1% of true matching pairs.

4.5 Effectiveness of FS-Dedup

In this set of experiments, we evaluate the effectiveness of FS-Dedup-NGram and FS-Dedup-SVM. We use as baseline Sig-Dedup configured with the optimal threshold value. Remind that the main objective of FS-Dedup is to identify the optimal configuration to maximize the effectiveness of the Sig-Dedup algorithm with minimum user effort. We should first discuss a set of extensive experiments in the synthetic dataset. After, we analyse the real datasets.

To simplify, we denote as *level size* the number of pairs randomly selected within each level to be manually labeled. Tables 7 and 8 illustrate details about the manual effort and the effectiveness of the deduplication process for both FS-Dedup-(NGram and SVM). The first column specifies level sizes of 10, 50, 100, 500, or 1000 pairs. Columns α and β show the mean of 10 runs with respective standard deviation values (σ) of the *fuzzy region* boundary. The average number of labeled pairs (without deviations) appears in the "# Sel" column. The last two set of columns, named "Fs-Dedup-NGram" and "FS-Dedup-SVM" present the "precision", "recall", and "F1" with their standard deviations (σ). Finally, in the last column we compare both "FS-Dedup-NGram" and "FS-Dedup-SVM" with statistical significance tests in order to identify improvements. The symbol $\uparrow$, $\downarrow$, and o represent a significant positive variation of 'FS-Dedup-NGram" over "FS-Dedup-SVM", significant negative variation and a non significant variation, respectively.

Table 5: F1 for Sig-Dedup using four thresholds in the synthetic datasets.

Dataset \ Th	0.3	0.4	0.5	0.6
Clean	0.92	**0.98**	0.86	0.69
Dirty-clean	0.89	**0.976**	0.87	0.66
Dirty	0.92	**0.975**	0.86	0.694

Table 6: F1 for Sig-Dedup using five thresholds in the real datasets.

Dataset \ Th	0.4	0.5	0.6	0.7	0.8
IMDBxNetflix	0.694	0.83	0.914	**0.923**	0.896
DBLPxCiteseer	0.795	0.902	**0.921**	0.876	0.790

Experiments on the synthetic datasets. In the first experiment, our objective is to manually identify the optimal configuration of Sig-Dedup based on the F1 values. We range the threshold values and evaluate the effectiveness of Sig-Dedup for each value. This experiment is reported in Table 5. The maximum F1 value is reached with threshold value of 0.4 in all synthetic datasets.

Figure 5 shows the comparison of the effectiveness of FS-Dedup-SVM and FS-Dedup-NGram against Sig-Dedup (tuned with optimal threshold) with different level sizes. Sig-Dedup-SVM produces a competitive result when compared to Sig-Dedup-NGram when a small level size is used (10 pairs). When the level size increases to 500 pairs, Sig-Dedup-NGram successfully achieves the maximum F1 value obtained by Sig-Dedup while Sig-Dedup-SVM has a slight improvement, although clearly insufficient to reach the F1 value of Sig-Dedup. Note that in the *dirty dataset*, a level size of 100 pairs is enough to FS-Dedup-NGram to produce maximum F1 value. This happens because this dataset has a large number of matching pairs (compared with other synthetic datasets), which makes easier for the sample selection strategy to select the matching pairs and accurately identify the *fuzzy region*.

To summarise the results, Table 7 shows that FS-Dedup-NGram is more effective than FS-Dedup-SVM on synthetic datasets in basically all level sizes but the one with size 10. Considering the 15 experiments performed (three datasets and five level size variations) and the values of F1, FS-Dedup-NGram statistically achieves better results in 11 experiments, ties in three and looses only in a single case. The results also show that FS-Dedup-NGram was able to reach a F1 value equal or greater 94% in all synthetic datasets using a level size of 100 pairs. When we increase the level size, FS-Dedup-NGram improves recall, keeping the precision close to the maximum. The improvement in recall is explained by decreasing values for the α boundary. The new α boundary enables to recover more matching pairs. On the other hand, when more candidate pairs are added into the *fuzzy region* FS-Dedup-SVM precision tends to drop. This is caused by a substantial number of candidate pairs that are misclassified as false positive pairs by the SVM. For instance, the only case in which FS-Dedup-SVM outperforms FS-Dedup-NGram is achieved with a *fuzzy region* with a small set of candidate pairs (α value of 0.34). When the α value is changed to 0.27 (level size = 100) the precision of Fs-Dedup-SVM drops about 10%.

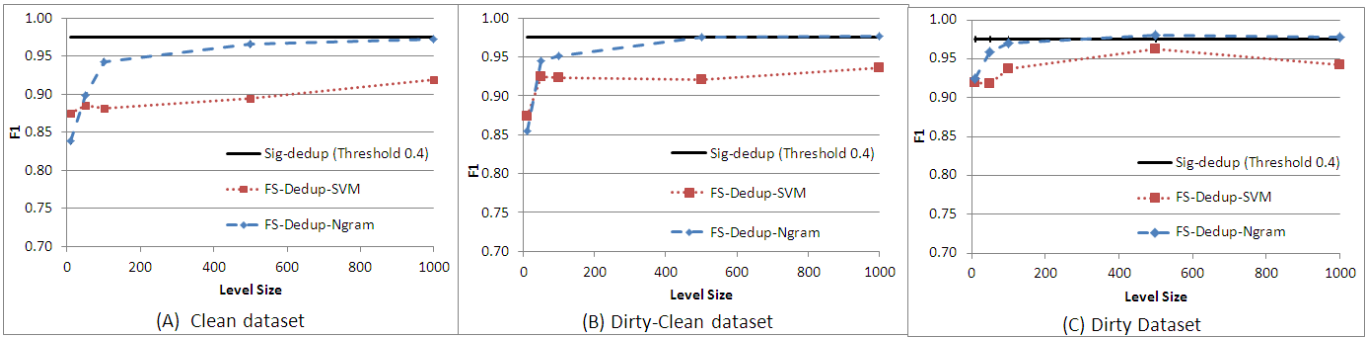

Figure 5: Comparison of F1 under different level sizes in the synthetic datasets.

We also compare α and β boundaries and the manual labeling efforts in the synthetic datasets. We can observe that when more pairs are labeled the value of α decreases while β value is more stable. This is explained by the fact that the lowest levels are composed of a large number of non-matching pairs meaning that matching pairs become rarer. When the level size increases (100, 500, or 1000 pairs) the *fuzzy region* can be identified more accurately. Overall, samples from level values 0.1 until 0.6 are manually labeled by the user and the levels above 0.5 are defined as true matching pairs. In other words, only five of nine levels are labeled, avoiding waste of manual effort. Notice that FS-Dedup-NGram required about 200 labeled pairs to become stable regarding precision, more pairs being necessary to identify precisely the *fuzzy region* boundaries (i.e. improving recall). As FS-Dedup-NGram is able to correctly approximate the *fuzzy region* with a level size of 50, a more accurate identification of *fuzzy region* requires more pairs to be labeled. Further studies on the reduction of the training set are left for future work.

Experiments on real datasets. We now discuss results obtained when comparing the effectiveness and the manual efforts of our proposed framework with Sig-Dedup in the two real datasets (IMDBxNetflix and DBLPxCiteseer). First, we manually identified the threshold values corresponding to the optimal configuration of Sig-Dedup. Table 6 shows that ideal threshold of DBLPxCiteseer and IMDBxNetflix have value of 0.6 and 0.7, respectively.

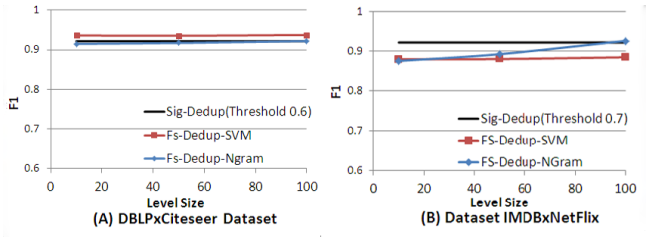

Figure 6: Comparison of F1 under different level sizes in real datasets.

Figure 6 compares the effectiveness of Sig-Dedup (tuned with the optimal threshold) with FS-Dedup-SVM and FS-Dedup-NGram, ranging the level size to 10, 50, and 100 pairs. DBLPxCiteseer dataset shows interesting results: both FS-Dedup-NGram and FS-Dedup-SVM achieve the maximum F1 value or close to maximum with a minimum level size of 10 pairs (Figure 6-A). Differently from what happened in the synthetic datasets, FS-Dedup-SVM statistically outperformed Sig-Dedup and FS-Dedup-NGram in all levels by about 2%. This is explained due to specific noise patterns in such datasets. As mentioned earlier, the Citeseer dataset contains data automatically crawled from the Web, resulting in some records with incomplete information. The noise patterns are better identified by the SVM algorithm with a smaller training set than FS-Dedup-NGram. These results are detailed in the Table 8 which also highlights that the increasing in the training set does not result in any significant gains in effectiveness. The *fuzzy region* is identified between the boundaries values of 0.4 and 0.8, and less than 50 labeled pairs were enough to configure FS-Dedup.

The dataset IMDBxNetFlix reaches a F1 value of 0.88 with level size of 10 pairs in both FS-Dedup-SVM and FS-Dedup-NGram (Figure 6-B). When the level size is increased to 100 pairs FS-Dedup-NGram successfully reaches the maximum F1 value of Sig-Dedup, similar to what happened in the synthetic datasets. FS-Dedup-SVM, contrarily, has no gains in terms of F1 with the growth of the training size. Table 8 details the results regarding effectiveness and manual efforts. We believe that the better results of FS-Dedup-NGram in this dataset is due to the lower level of noise, (e.g., title variations "Brazil vs Brazil, the movie"). Note that six levels or 330 pairs manually labeled pairs were enough to identify the optimal configuration of FS-Dedup-NGram.

Overall, we conclude that FS-Dedup is able to identify the optimal configuration regarding the matching quality offered by high performance Signature-based algorithm with little manual effort. The experiments show that FS-Dedup-NGram achieves optimal configuration in all datasets but one. The combination of Sig-Dedup and SVM (FS-Dedup-SVM) achieves competitive results only when the dataset possesses hard matching patterns due to noise data.

4.6 Label Effort Comparison

We now report on our final experiments comparing the label efforts and effectiveness of FS-Dedup with that of the method proposed by Bellare et al, referred as ALD, a state-of-the-art active learning deduplication method. To enable experimental repetition and fair comparison, we only report experiments in the synthetic datasets in this section. As the labeled sets for the real datasets were already selected used our proposed method, this would give an unfair advantage to ALD if compared to its use in a real-world scenario in

Table 7: Synthetic datasets results when comparing the FS-Dedup-SVM and FS-Dedup-NGram in terms of effectiveness and number of labeled pairs

Data set	level size	$\alpha(\sigma)$	$\beta(\sigma)$	# Pairs	FS-Dedup-SVM Precision (σ)	FS-Dedup-SVM Recall(σ)	FS-Dedup-SVM F1(σ)	FS-Dedup-NGram Precision(σ)	FS-Dedup-NGram Recall(σ)	FS-Dedup-NGram F1(σ)
Clean	10	.34±(.05)	.50±(.00)	36	.99±(.01)	.79±(.10)	**.87±(.07)**	.96±(.11)	.77±(.12)	.84±(.09)↓
	50	.31±(.05)	.50±(.00)	194	.98±(.03)	.82±(.11)	.89±(.07)	1.00±(.00)	.82±(.11)	.90±(.07) *o*
	100	.27±(.04)	.50±(.00)	430	.90±(.15)	.89±(.04)	.88±(.09)	.99±(.01)	.90±(.04)	**.94±(.03)**↑
	500	.20±(.06)	.50±(.00)	2500	.87±(.09)	.92±(.04)	.89±(.04)	.99±(.00)	.94±(.04)	**.97±(.02)**↑
	1000	.18±(.06)	.50±(.00)	5222	.90±(.07)	.95±(.03)	.92±(.04)	.99±(.00)	.95±(.03)	**.97±(.02)**↑
Dirty-Clean	10	.32±(.06)	.43±(.05)	32	.98±(.01)	.79±(.10)	.87±(.06)	.95±(.05)	.79±(.14)	.86±(.08) *o*
	50	.26±(.05)	.50±(.00)	220	.97±(.05)	.88±(.05)	.92±(.03)	1.00±(.00)	.90±(.05)	**.95±(.03)**↑
	100	.24±(.06)	.50±(.00)	460	.94±(.07)	.91±(.05)	.92±(.03)	.99±(.02)	.92±(.05)	**.95±(.03)**↑
	500	.16±(.05)	.51±(.03)	2750	.90±(.06)	.95±(.01)	.92±(.03)	.99±(.01)	.96±(.00)	**.98±(.00)**↑
	1000	.20±(.00)	.50±(.00)	5000	.92±(.04)	.95±(.00)	.94±(.02)	1.00±(.00)	.96±(.00)	**.98±(.00)**↑
Dirty	10	.29±(.04)	.42±(.05)	33	.99±(.11)	.86±(.04)	.92±(.06)	.98±(.02)	.87±(.04)	.92±(.02)*o*
	50	.22±(.06)	.49±(.03)	235	.93±(.12)	.92±(.05)	.92±(.07)	.99±(.02)	.93±(.05)	**.96±(.03)**↑
	100	.20±(.05)	.50±(.00)	500	.94±(.07)	.94±(.03)	.94±(.04)	.99±(.01)	.95±(.03)	**.97±(.02)**↑
	500	.15±(.05)	.50±(.00)	2750	.97±(.03)	.96±(.01)	.96±(.02)	.99±(.01)	.97±(.00)	**.98±(.00)**↑
	1000	.11±(.03)	.50±(.00)	5900	.92±(.04)	.96±(.00)	.94±(.02)	.99±(.01)	.97±(.00)	**.98±(.01)**↑

Table 8: Real datasets results when comparing FS-Dedup-SVM and FS-Dedup-NGram in terms of effectiveness and number of labeled pairs

Data set	level size	$\alpha(\sigma)$	$\beta(\sigma)$	# Sel	FS-Dedup-SVM Precision (σ)	FS-Dedup-SVM Recall(σ)	FS-Dedup-SVM F1(σ)	FS-Dedup-NGram Precision(σ)	FS-Dedup-NGram Recall(σ)	FS-Dedup-NGram F1(σ)
IMDBx NetFlix	10	.42±(.08)	.65±(.00)	47	.81±(.07)	.97±(.01)	.88±(.04)	.79±(.08)	.99±(.02)	.88±(.04) *o*
	50	.32±(.08)	.76 ±(.00)	330	.80±(.03)	.98±(.01)	.88±(.01)	.81±(.06)	.99±(.00)	**.89±(.04)**↑
	100	.30±(.00)	.90±(.00)	796	.80±(.00)	.99±(.00)	.88±(.00)	.98±(.00)	.93±(.00)	**.92±(.00)**↑
DBLPx CiteSeer	10	.47±(.06)	.72±(.00)	45	.91±(.01)	.96 ±(.01)	**.94±(.01)**	.86±(.02)	.97±(.01)	.91±(.01)↓
	50	.43±(.00)	.80±(.00)	285	.91±(.01)	.96±(.01)	**.94±(.01)**	.88±(.03)	.95±(.02)	.92±(.01)↓
	100	.40±(.00)	.80±(.00)	581	.92±(.00)	.95±(.00)	**.94±(.00)**	.90±(.00)	.94±(.00)	.92±(.00)↓

which it would have to randomly choose a sample from the entire dataset.

Figure 7 shows the results comparing ALD with FS-Dedup, using F1 values and the number of pairs labeled. As we can see, in all datasets, the initial (smallest) sample sets (size of 36, 32, 33 for the Clean, Dirty-Clean, and Dirty datasets, respectively) selected by both, FS-Dedup-SVM and FS-Dedup-NGram, already produce very good effectiveness. For FS-Dedup-NGram, there is also some improvement as more pairs are selected until a stabilization point around 500 samples, while FS-Dedup-SVM effectiveness remains stabler, but somewhat less effective, with the increase in the number of pairs. In comparison, ALD which starts always with a sample size of around 100 pairs (in fact for the Dirty dataset the size of this initial set is almost 200 due to the high number of candidate pairs) to configure the oracle and train the classifiers, initiates with a very poor effectiveness in the *Clean* and *Dirty-Clean* datasets, being only able to get close to FS-Dedup-NGram after almost 2000 pairs and 700 pairs have been selected and labeled in both datasets respectively. In the Dirty dataset, all methods are basically tied since the initial choices, due to the abundance of match pairs in this dataset. In fact, when comparing the smallest training set size selected by both FS-Dedup-NGram and FS-Dedup-SVM, they are *3.8, 4.7, and 7.5 times* smaller than the initial training set selected by ALD on Clean, Dirty-Clean, and Dirty datasets, respectively. In sum, these experiments show that FS-Dedup minimize substantially the manual label efforts and produce competitive results in terms of effectiveness when compared to state-of-the-art active learning methods for large scale deduplication.

5. CONCLUSIONS

In this paper, we presented a new framework to identify the optimal configuration on large scale deduplication. The framework, named FS-Dedup, allows the user to tune the entire deduplication process (i.e. blocking and classification phases), invoking the user to only label a reduced set of pairs automatically selected by FS-Dedup. We demonstrate how to take advantage of the high performance Signature based algorithms (Sig-Dedup) to achieve the maximum matching quality offered by Sig-Dedup. We evaluated FS-Dedup using synthetic and real datasets (one with about three million of records), and empirically show that FS-Dedup is able to identify the best configuration with a small training set. As future work, we intend to incorporate active learning strategies to our FS-Dedup to improve the selection within each level, which is currently performed randomly.

6. ACKNOWLEDGMENTS

This research is partially funded by InWeb - The Brazilian National Institute of Science and Technology for the Web (MCT/CNPq/FAPEMIG grant number 573871/2008-6), and by the authors's individual research grants from FAPEMIG, CNPq, CAPES and PFRH-17 PETROBRAS.

7. REFERENCES

[1] A. Arasu, M. Gotz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD, 2010*.

[2] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE, 2009*.

[3] A. Awekar, N. F. Samatova, and P. Breimyer. Incremental all pairs similarity search for varying

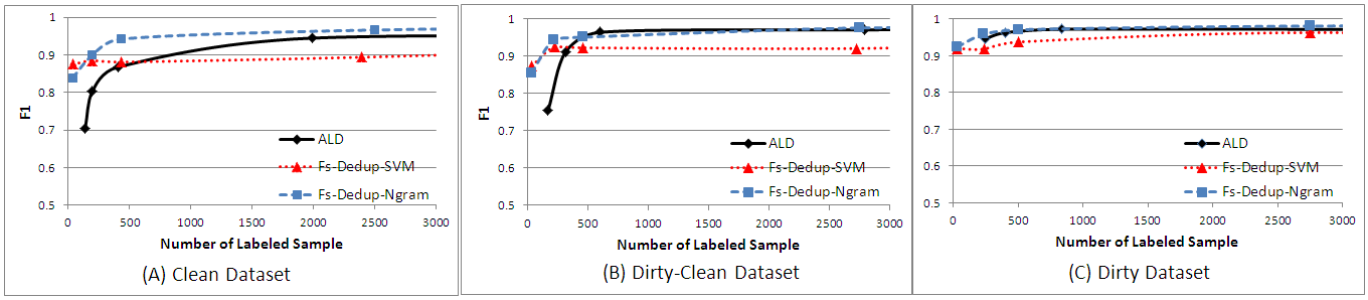

Figure 7: Effectiveness and manual efforts comparison of FS-Dedup against the baseline ALD.

similarity thresholds. In *SNA-KDD, 2009*.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *KDD WORKSHOPS*, 2003.

[6] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW, 2007*.

[7] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *KDD, 2012*.

[8] A. Beygelzimer, S. Dasgupta, and J. Langford. Importance weighted active learning. In *ICML, 2009*.

[9] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD, 2003*.

[10] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[11] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *VLDB, 2007*.

[12] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE, 2006*.

[13] P. Christen and T. Churches. Febrl - freely extensible biomedical record linkage. Technical report, 2002.

[14] M. de Carvalho, M. Gonçalves, A. Laender, and A. da Silva. Learning to deduplicate. In *ACM/IEEE-CS 2006*.

[15] C. F. Dorneles, R. Gonçalves, and R. dos Santos Mello. Approximate data instance matching: a survey. *KAIS, 2011*.

[16] A. K. Elmagarmid, P. G. Ipeirotis, Vassilios, and S. Verykios. Duplicate record detection: A survey. *TKDE, 2007*.

[17] J. Gemmell, B. I. P. Rubinstein, and A. K. Chandra. Improving entity resolution with global constraints. *CoRR*, abs/1108.6016, 2011.

[18] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *DL,2008*.

[19] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD, 95*.

[20] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD, 2006*.

[21] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[22] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[23] H. Müller and j. Problems, Methods and Challenges in Comprehensive Data Cleansing. Technical Report HUB-IB-164, Berlin, 2003.

[24] C. Peter. Performance and scalability of fast blocking techniques for deduplication and data linkage. *Proc. VLDB Endow.*, 1(2):1253–1264, 2007.

[25] V. Petricek, I. Cox, H. Han, I. Councill, and C. Giles. A comparison of on-line computer science citation databases. In *Research and Advanced Technology for Digital Libraries, 2005*.

[26] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD, 2002*.

[27] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD 2004*.

[28] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *KDD*, pages 678–684. ACM, 2005.

[29] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *SIGKDD, 2002*.

[30] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD,2010*.

[31] J. Wang, G. Li, and J. Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *ICDE*, pages 458 –469, april 2011.

[32] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering: an adaptive framework for similarity join and search. In *SIGMOD 2012*.

[33] J. Wang, G. Li, J. X. Yu, and J. Feng. Entity matching: how similar is similar. *Proc. VLDB Endow.*, 4(10):622–633, July 2011.

[34] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[35] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *TODS, 2011*.