Generating Realistic Test Datasets for Duplicate Detection at Scale Using Historical Voter Data

Fabian Panse
panse@informatik.uni-hamburg.de
Universität Hamburg
Hamburg, Germany

André Düjon
1duejon@informatik.uni-hamburg.de
Universität Hamburg
Hamburg, Germany

Wolfram Wingerath
wolfram.wingerath@baqend.com
Baqend
Hamburg, Germany

Benjamin Wollmer
wollmer@informatik.uni-hamburg.de
Universität Hamburg
Hamburg, Germany

ABSTRACT

The detection of duplicates is an essential task in data cleaning and integration and has steadily gained importance especially for researchers and practitioners that need to process and integrate large volumes of potentially unclean data on a daily basis. To evaluate the quality and performance of duplicate detection algorithms, labeled test data are required that provide information on the contained duplicates. Current approaches for generating test data, however, are either not scalable (and therefore limited to small datasets) or not able to generate realistic data values and errors, especially outdated values. In this paper, we propose a scheme for generating test datasets that addresses both these issues and present a test dataset generated with it. Our approach relies on using historical data from the North Carolina voter register which (1) is realistic as it contains actual voter data and (2) facilitates generating realistic duplicates through the fact that current data values were collected at every election through manually filled out applications. The generated test dataset comprises more than 120 million records with up to 90 attribute values each. To the best of our knowledge, we are the first who provide realistic test data for duplicate detection at this scale.

1 INTRODUCTION

Duplicates are data records (e.g., tuples in the relational case) that refer to the same real-world object. They can result from errors in data management, but also occur because separately developed data sources overlap in their universes of discourse (e.g., many actors and movies are stored in both IMDB[1] and TMDb[2]). The detection of duplicates is an important task in data cleaning [12, 16] and integration [8, 9]. Detecting duplicates is quite simple when they are exact, i.e. they agree in all of their values. However, it can be extremely difficult if some of their values disagree due to typos, phonetic or transformation errors, heterogeneous forms of presentation as well as missing or outdated values [14].

The challenge of detecting such so-called *fuzzy* duplicates has opened up its own field of research and has since been studied intensively [4, 7, 23, 31]. However, the best approach to find them strongly depends on (i) the considered domain (e.g., movies, persons, or proteins), (ii) the characteristics of the given data (e.g., volume, data model and heterogeneity), and (iii) the quality and cost requirements of the user (e.g., good results vs. short runtimes and recall vs. precision). Due to the resulting diversity of use cases, none of the existing algorithms has turned out to be a generally applicable and superior solution. Instead, in every use case, it remains a difficult (and expensive) task to choose and configure them so that they provide adequate results.

Such a configuration process requires the evaluation and comparison of different algorithms and parameter settings. This in turn requires test datasets that do not only provide a *gold standard* (a.k.a. *ground truth*) labeling the dataset's duplicates [21], but resemble the required real-life properties as well as possible. Current approaches to test data generation either (i) struggle with the generation of realistic data values and errors (especially outdated values), (ii) cannot guarantee the soundness of the gold standard[3], or (iii) scale badly and thus can only be used to generate small datasets. However, realistic values and errors as well as correctly labeled duplicates are an important prerequisite for a test dataset. Moreover, in times of big data many duplicate detection algorithms focus on scalability (e.g., [13, 17, 30]) so that an evaluation of their key functionalities requires large test datasets with millions of records.

Using a historical dataset to generate test data seems to be a straightforward solution to some of the aforementioned problems, because the mapping between records and real-world objects is part of the data so that the duplicates are already labeled. Thus, it scales much better than, for example, labeling the duplicates in an unclean dataset manually. In addition, historical datasets are perfectly suited to generate outdated values, because these values are an inherent part of the data. One of these historical datasets is provided by the State of North Carolina (short NC) [26]. This dataset contains information on voters registered to the individual elections and – at the time of our study – consisted of 45 snapshots covering a time period of 16 years with a total number of over 500 million records and a large schema with 90 attributes. These numbers make it a perfect candidate to evaluate the suitability of the aforementioned idea, because the large number of records allows us to generate a test dataset of large size and the large time span provides us many outdated values (even more than one for the same object property). Furthermore, since voters often have to re-register at regular intervals by manually filled out forms[4], the registration data contain typos, values confused between attributes, heterogeneous forms of presentation and missing values which makes this dataset particularly useful for

[1] Internet Movie Database: https://www.imdb.com
[2] The Movie Database: https://www.themoviedb.org

[3] To clearly distinguish between errors in the duplicate labels and the operational data, we use the term *soundness* w.r.t. the correctness of the gold standard.
[4] https://dl.ncsbe.gov/?prefix=Voter_Registration/

the generation of fuzzy duplicates. Finally, its large size gives us the opportunity to customize the test data to different user requirements by selecting a suitable subset of all records (the more data available, the more flexible the selection process). However, the big amount of redundant data as well as the ongoing publication of new snapshots also pose some challenges to the generation process making it a non-trivial task.

The contributions of this paper can be summarized as:

(i) A comprehensive list of desiderata for test datasets for duplicate detection.
(ii) An approach for generating and storing test data based on the historical voter register from North Carolina.
(iii) A realistic test dataset generated with our approach.
(iv) An extensive experimental evaluation to analyze the quality and prove the usability of the generated test dataset.

We provide the generated dataset to other researchers[5]. It will help them to evaluate their algorithms (such as runtime behavior or robustness against a varying number of data errors) and to compare them with those of other research projects. It is particularly valuable to the research community through a combination of properties that is unique to the best of our knowledge:

- It contains more than 120 million records and 640 million duplicate pairs making it suitable to evaluate duplicate detection algorithms at scale,
- it contains real-life errors of various types including typos, abbreviations, phonetic errors and outdated values,
- its large size qualifies it to customize the test data to different user requirements without losing necessary volume,
- it provides precalculated plausibility and heterogeneity scores, which support the user to remove (or repair) potentially unsound duplicate clusters and adapt the datas' heterogeneity to her own requirements, and
- it provides meta information that allows the user to reproduce experiments using previous versions of this dataset.

The rest of this paper is structured as follows: In Section 2, we describe the input to our study, i.e. the voter register from North Carolina. Thereafter, in Section 3, we discuss several aspects affecting the test datas' quality, usability and reproducibility. In Sections 4 and 5, we describe our approach for using the historical voter data for test data generation. In Section 6 we present an experimental study that evaluates the quality and usability of the generated test dataset. Finally, we discuss related work in Section 7 before we conclude the paper and give an outlook on future research in Section 8.

2 NORTH CAROLINA VOTER REGISTER

The voter register from North Carolina was created and is still maintained by the *North Carolina State Board of Elections*[6] according to the *Help America Vote Act* (HAVA) of 2002. The provided voter records are considered public information per NC General Statutes (§132-1, §163-82.10) [1, 25], but do not include dates of birth, driver's license numbers and social security numbers because they are confidential under state law [1, 20, 26].

In addition to current data, the register provides a voter history in the form of a series of snapshots [24]. The first publicly available snapshot is from 2005-11-25. New snapshots were (and still are) created at every New Year's Day and the date of every election (general, primary and municipal) [26]. At the time of our study, the register contained 45 snapshots. The schema of these snapshots evolved over time, but was consistent for the last 41 snapshots. Since the first four snapshots are missing necessary information to clearly identify a voter, we excluded them from our study. The characteristics of the remaining snapshots are presented (in an aggregated form) in Table 1. The whole voter history contains 522,463,029 records representing a total of 13,569,512 distinct persons.

Table 1: Overview of the snapshots included in this study

	#snap-	#total	#new		rate of new	
year	shots	records	records	objects	records	objects
2008	1	9.7 M	9.7 M	9.4 M	100%	96.8%
2009	1	9.7 M	0.7 M	37 K	6.8%	5.6%
2010	2	20.2 M	13.1 M	189 K	64.9%	1.4%
2011	1	10.3 M	2.3 M	225 K	22.2%	9.9%
2012	4	41.8 M	19.9 M	820 K	47.6%	4.1%
2013	1	11.4 M	11.1 M	41 K	97.1%	0.4%
2014	4	47.3 M	7.5 M	432 K	15.8%	5.8%
2015	4	49.0 M	6.6 M	223 K	13.5%	3.4%
2016	4	50.9 M	7.7 M	587 K	15.1%	7.6%
2017	4	54.1 M	3.6 M	245 K	6.7%	6.7%
2018	3	41.7 M	23.7 M	374 K	56.9%	1.6%
2019	7	99.8 M	5.5 M	354 K	5.5%	6.5%
2020	4	60.8 M	8.0 M	596 K	13.1%	7.4%
2021	1	15.9 M	0.8 M	62 K	5.1%	7.6%
total	41	522.5 M	120.8 M	13.57 M	23.1%	11.2%

M = million, K = thousand

Each snapshot corresponds to a large tab-separated values (TSV) file. As it turned out during data profiling, these files are formatted differently. While the older files (if not updated later) are in UTF-8, the newer files are in UTF-16. Since none of the provided attributes is expected to contain characters that are not part of the UTF-8 character set, we converted all files to UTF-8 before importing them into our dataset. Here it is important to note that occasional conversion errors do not spoil our test dataset, since they also happen in real-life, as long as they do not corrupt the correct mapping between records and objects required for the gold standard (i.e., they do not concern the NCIDs).

Every record in the snapshot files specifies an entry to the voter register and consists of 90 attributes. We grouped these attributes into four semantic categories:

- personal information (38 attributes) such as names, age, address data, phone number, race code and sex,
- information on the districts the voter is registered in (38 attributes), such as school, water and fire district,
- information on the voter that is closely related to the election she is registered to (11 attributes), such as voter status and registration date, and
- meta data for administrating the snapshots and identifying voters/records within them (3 attributes), which are the NCID as well as the snapshot and load date.

The NCID is a unique number for each voter currently or previously registered in North Carolina. A voter's NCID will follow him from one county to another when she migrates within the state of NC. Thus, the NCID can be used to uniquely identify the individual voters and therefore can serve as an object-id. To our surprise, we discovered that in every snapshot many voters are represented by more than one record. A closer look revealed that at most only one of them has not the voter status *removed* (and hence is not outdated). This means that every snapshot already corresponds to a historical dataset.

[5]Please write an email to dbis-research@informatik.uni-hamburg.de.
[6]https://www.ncsbe.gov/

3 TEST DATA DESIDERATA

Before we describe in which way we used the history of the NC voter register to generate test data in Section 4, we will take a closer look on the desired properties of such a test dataset. A suitable test dataset has to ensure a high

- *quality*, i.e., the test data should enable meaningful evaluation results,
- *usability*, i.e., the user should be able to customize the test data according to her requirements, and
- *reproducibility*, i.e., the user should be able to reproduce the results of past evaluations that used previous versions of this dataset in order to achieve adequate comparability.

In the rest of this section, we will discuss these requirements and the problems that are related with them in more detail. The way we handled them in our generation process will be described in the remaining course of this paper.

3.1 Quality

A test dataset is of good quality if its gold standard is sound, its data contain real(istic) errors of different types and it contains only few exact duplicates.

3.1.1 Soundness of the Gold Standard. A test dataset for duplicate detection consists of a set of data records and the corresponding gold standard that specifies the duplicate status between the individual records. While errors in the actual data are quite desirable (see Section 3.1.2), it is extremely important that the gold standard is sound, because even a small number of incorrectly labeled record pairs (i.e., *false positives* and/or *false negatives*) can render evaluation results completely useless.

In a perfect world, the mappings between the voter records and actual voters are sound. However, almost no dataset is free of errors. Thus, it make sense to perform a soundness check on the test data because, as we illustrate in Figure 3, there may be clusters whose records do not seem to represent the same voter although they share the same NCID. Marking those clusters allows the user to remove or repair them before using the test dataset. Since we often cannot distinguish between sound and unsound clusters with absolute confidence, it does not seem wise to use a Boolean flag as a marker, but to compute similarities which reflect a kind of likelihood that these clusters are sound (i.e., all their records represent the same voter). The user can then use these similarities to decide which risk she wants to take to include unsound clusters into her test data. We refer to this similarity as *plausibility* in the rest of this paper and discuss a calculation of plausibility scores for the NC test dataset in Section 6.2.

3.1.2 Error Diversity. The results of an evaluation with a test dataset are only representative if this dataset contains real-life (or at least realistic) data values and errors. In our case, both are real because they originate from a real-life dataset. Moreover, users want to evaluate algorithms that should later be applied to error-prone data. This requires that the test dataset contains errors of various kinds and not only outdated values. This includes typos, abbreviations, invalid values, inconsistencies and different forms of representation. In other words a test dataset of high quality has to contain several problems of data quality.

3.1.3 Amount of Exact Duplicates. Another aspect that affects the usefulness of evaluation results is the number of exact duplicates contained in the test data. The detection of such duplicates is rather simple and every duplicate detection algorithm – no matter how primitive – should be able to detect them. Thus if this number dominates the number of fuzzy duplicates by far, an accurate detection of the latter becomes less relevant in order to achieve a good evaluation result. For example, if 90% of all duplicate pairs are exact, even the most primitive algorithm would achieve a recall of 0.9 or higher if it is able to compare values on equivalence. Moreover, an algorithm that classifies only the exact duplicates as such and all other pairs as non-duplicates (precision is 1.0) would even achieve a F_1-score of 0.9 which is a pretty good result. However those algorithms are completely useless when it comes to real-life use cases where fuzzy duplicates need to be detected. While this aspect is of little relevance in many approaches to test data generation (the number of exact duplicates is usually very small there), it is of great importance when using historical data, since many of the given snapshots overlap to a large extent, so that their combination leads to many exact duplicates. As we will see in Section 4, by simply combining the individual snapshots of the NC voter register we produced a relative amount of exact duplicates of over 90%.

The actual definition of an exact duplicate pair is that both records share the exact same value in every attribute. However, in the case of the NC voter data, solely removing those duplicates which are completely identical does not solve this problem, because often many of the remaining duplicate records only differ in some minor aspects, such as:

- *Meta Data Attributes*: Many duplicate records only differ in some date values, such as snapshot or registration date, that are less relevant for the duplicate detection process.
- *Time-related Attributes*: The voters' age values increase by one every year and thus cause that some duplicates are no longer exact, although none of the other characteristics of the corresponding person changed.
- *Whitespaces*: Many values contain leading and trailing whitespaces that are simply to detect and remove by trimming all data values in an initial preparation step.

We describe how we addressed this problem in Section 4.

3.2 Usability

Since we aim to provide test data for duplicate detection at scale, the resulting dataset should contain several million records. Besides size, the requirements of the individual users can vary from one evaluation to another. Therefore it is advantageous if the test data can be adapted to the needs of the respective use case in terms of several data characteristics, such as the number of clusters, the cluster sizes, or the degree of heterogeneity (a.k.a. *dirtiness*). This can be accomplished by applying a postprocessing step, which selects a subset of all data carefully. Further options for customization are the removal and merge of attributes, changing the character of the attributes' values. A flexible and unconstrained customization requires that the test dataset contains (i) many duplicate clusters of various sizes and (ii) duplicate records of different degrees of heterogeneity, so that the user has a large set to choose from. In addition, it requires that the user can adjust the characteristics of the test data with relative ease. This can be supported by storing all records of one cluster together and providing precalculated heterogeneity scores.

The heterogeneity of the individual duplicate clusters (or duplicate pairs) represents the degree to which the duplicate records differ from each other. At the same time, it can be considered as a measure on the difficulty of detecting the fuzzy duplicates within this dataset because duplicates are usually the more difficult to detect, the more their values differ. Thus, this information does

Table 2: Statistical results of the generation process

duplicate removal	*#records*	*#dupl. pairs*	*cluster size*		*#removed*	
			avg.	*max.*	*records*	*pairs*
no	522.5 M	12,108.2 M	38.50	399	0%	0%
exact	166.3 M	1,225.0 M	12.26	104	68.2%	89.9%
trimming	120.8 M	648.2 M	8.90	77	76.9%	94.6%
person data	58.7 M	136.7 M	4.33	51	88.8%	98.9%

*The number of objects (i.e., clusters) was always 13.57 M.

not only provide interesting insights into the nature of the test data, but also allows the user to customize the level of difficulty of her test dataset individually by filtering out clusters/records whose heterogeneity is not within a requested range. This can be useful when the user wants to test her algorithms with datasets of different degrees of dirtiness. It is important to note that such filtering can theoretically be performed on any test dataset. However, only a large number of clusters and records allows the user to compose arbitrary subsets without running into the problem of producing a too small output.

3.3 Reproducibility

The NC voter register is subject to constant change and new snapshots are published regularly. This gives the opportunity to extend the generated test dataset on a regular basis, too, which does not only provide data on new voters (i.e., more duplicate clusters), but also new data on already existing voters (i.e., larger duplicate clusters and higher degrees of duplicate heterogeneity). In general, the longer the time span covered by a test dataset, the more outdated values it contains. Moreover, a longer time span increases the chance of obtaining outdated values even for attributes that do not change very frequently (e.g., the last name).

In performance and quality evaluation, *reproducibility* [27] (a.k.a. *repeatability* [28]) is an important aspect because it is necessary to enable a fair comparison between (the evaluation results of) different algorithms especially if they are evaluated at different times by different parties. In this context, reproducibility means that another evaluation process (or at least its experimental setting) can be reproduced exactly. This includes the use of the same test data. Thus, test datasets that change over time pose a problem for reproducibility, especially if their size does not allow a separate storage of every intermediate version. To solve this problem, the datasets must be enriched by information that allows the user to reconstruct any of the old versions on request. It is important to note that such a reconstruction does not only apply to the actual records, but also all meta data (e.g., statistics and similarities) that are stored to describe them. We discuss how we ensure reproducibility for our test dataset in Section 5.1.

4 TEST DATA GENERATION

Since every snapshot of the voter register already contains outdated records and we wanted to reduce execution time as well as the number of exact duplicates as much as possible, we started to experiment with a single snapshot. Because we planned to use as much data as possible and the size of the snapshots grew monotonically over time, we used the most recent one. At the time of our analysis, this was the one created at 2021-01-01 and indeed, among all available snapshots, it contains the most records having the voter status *removed* (and thus are potentially outdated). The total number of records in this snapshot is 15,863,484 (8,308,925 *removed*) which corresponds to 3.04% of all voter records. However, the experiments showed that the amount of historical information stored in the individual snapshots is rather low compared to the whole history and thus only provides small clusters (see Figure 1a). Therefore, we also evaluated the entire voter history containing all snapshots available. By doing so, we examined a total of 522,463,029 records.

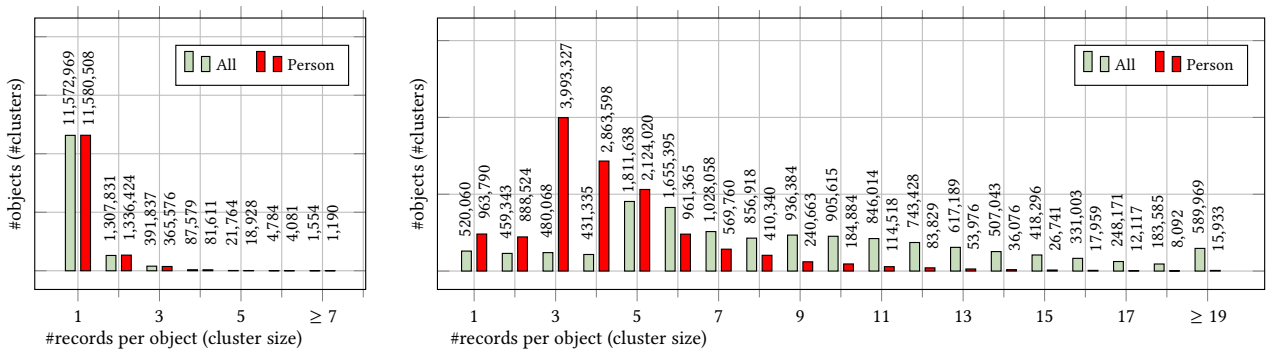
One of our goals was to analyze the amount of (near) exact duplicates within the resulting test data. Moreover, we think that most potential users are only interested in the personal data of the voters, since the election and district attributes are very case-specific. Therefore, we executed our approach four times: (i) one time without removing any duplicates, (ii) one time with removing all exact duplicates, (iii) one time with removing all duplicates that were exact after their values have been trimmed (i.e., leading and trailing whitespaces were deleted), and (iv) one time with removing all duplicates whose personal data were equivalent (after trimming attribute values). To check the equivalence of duplicate records efficiently, we used the *Message-Digest Algorithm 5* (short MD5) to calculate a hash value for each record. A record was then not imported into the test dataset when it already contained a record with the same hash value. Of course, collisions between non-exact records cannot be excluded for sure, but such a collision only means that the test dataset loses a duplicate record and thus does not have severe consequences if it happens a few times[7]. The input to the hash function is the concatenation of the values of all relevant attributes to a single large string. As mentioned in Section 3.1.3, some meta data and time-related attributes can reduce the number of near exact duplicates drastically and therefore were not included into the concatenation. These attributes are the different dates (snapshot, load, registration and cancellation) and the age[8].

The number of resulting objects (i.e., duplicate clusters), records, duplicate pairs, the average and maximal number of records per object (i.e., duplicate cluster size), as well as the number of removed records and duplicate pairs are listed in Table 2. The number of distinct duplicate clusters (i.e., objects) per cluster size (i.e., number of records per object) is presented in Figure 1b (one time for all attributes and one time for the personal data only). Using a single snapshot did not produce any exact duplicate which was even less than expected. In contrast, using all snapshots produced hundred of millions of exact duplicates. Obviously, the average number of records per voter decreased when these exact duplicates were removed (e.g., 8.90 without whitespaces) and decreased further on when we restricted the data to the personal attributes (4.33), but was still large compared to the single-snapshot approach (1.18). In total, the number of records that were removed because of being exact duplicates was up to 76.9% and the number of removed duplicate pairs was up to 94.6% when all attributes where taken into account and up to 88.8% and 98.9% if only person data was considered. These large amounts of exact duplicates illustrate the importance of their removal, because otherwise every evaluation of duplicate detection algorithms using these data would suffer from the effects described in Section 3.1.3.

To estimate the value of future snapshots, we counted the number of new clusters (i.e., the snapshot contains an NCID that

[7] MD5 produces 128 bit hashes, which means that it is relatively unlikely that the hash values of two different inputs collide.

[8] In the case of age values, the most obvious solution is to transform them into years of birth because the latter do not change. However, such a calculation would enclose a part of the dates of birth, which were originally removed from the data due to privacy reasons. We have therefore decided to use them only for internal calculations (e.g., plausibility) and not to store them in the resulting test data.



(a) Single Snapshot **(b) All Snapshots**

Figure 1: Distribution of the number of records per object (i.e., cluster size) after removing exact duplicates (trimming)

was never used before) and new records (i.e., the snapshot contains a record that was not part of any of the previous snapshots) per snapshot-year. These numbers are presented in Table 1 (note that the second includes the first). The last two columns show the percentage of rows that result in a new record (*new record rate*) and the percentage of new records that lead to a new cluster (*new object rate*). As expected, the number of new clusters and new records is the highest for the first snapshot. Surprisingly, the numbers of the following snapshots vary enormously (we expected an almost constant number). Investigations revealed that in some snapshots the formats of one or two attributes changed (e.g., from ‚64TH HOUSE' to ‚NC HOUSE DISTRICT 64') so that each of their records were considered to be ‚new' even if – apart from that – they were identical with one of the already existing ones. This again shows us the importance of providing the user with an instrument that allows her to filter out records based on their similarities (see Section 6.5). However, as one can see, even the last five snapshots contributed a significant amount of new clusters and records to the test data. Thus, we can expect the same for future snapshots.

5 TEST DATA STORAGE

The voter history is originally given as a set of TSV files. However, we want to store our test dataset by using a data model that is more suitable with respect to its later usage, i.e. to evaluate duplicate detection algorithms as well as potential extensions with data from new snapshots. As a consequence, the data model has to satisfy three essential requirements:

- To customize the test data, we need to select and reduce duplicate clusters based on user-defined specifications. Moreover, when we integrate additional snapshot data, we need to calculate statistics by comparing duplicate records (e.g., plausibility and heterogeneity). Both require fast and collective access to all records of the same cluster.
- Every record of the voter data has 90 attributes, but only a few records have values for district-related attributes. This means that millions of records have missing values in at least 38 attributes. Thus, we require the underlying data model to provide an efficient handling of sparse data.
- Working on hundreds of gigabytes requires scalable software solutions.

Schemaless NoSQL data models are much better suited to store sparse data than the relational model which requires the definition of a rigid schema. Moreover, many NoSQL data stores are designed to handle *aggregates* each of which is a collection of related data that we wish to treat as a unit [29]. Thus, they allow an easy and efficient way to access all the records of a certain person as we need it for customization and future extensions. Among all the available NoSQL data stores, we decided to use the document store MongoDB [22]. In contrast to the relational data model which is *aggregate-ignorant*, document stores are strongly *aggregate-oriented* [29] because they allow to (i) group records by storing them within the same document and (ii) nest different documents hierarchically. Furthermore, MongoDB is highly scalable. Besides its schemaless structure, MongoDB has three features that are especially helpful for this work [22]:

- *Indexes:* Since our test dataset contains millions of nested documents, indexes are very important to efficiently select those documents from the dataset.
- *Aggregation Pipeline:* Multi-stage pipelines can be used to transform documents into an aggregated result. Available pipeline stages provide tools for filtering, transformation, grouping and sorting. These pipelines enable users to extract relevant subsets of the data and thus to customize their own test datasets.
- *Compass:* MongoDB has a powerful GUI called *Compass*. It enables the user to easily interact with the stored data with full CRUD functionality. It is very helpful for exploring, generating, adjusting and using the test data. Moreover, it allows to monitor load jobs of new snapshots and helps to identify mistakes at an early stage.

In our case, we created one document for every person (i.e., duplicate cluster) that in turn contains a document for every record of this person (which are grouped into an array) and in addition a document containing some relevant meta data including the hash values of the stored records. Since most users will be interested in the personal data only, we split every record into four parts (person, district, election and meta) and stored them into different subdocuments.

5.1 Future Updates & Reproducibility

The NC State Board of Elections publishes a new snapshot at every election and every New Year's Day. Moreover, we have observed that they published some old snapshots belatedly (e.g., the snapshot from 2010-11-02 was first published in May 2019). To improve our test dataset both in size and heterogeneity, we will update it regularly.

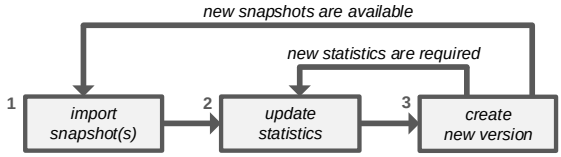


Figure 2: Update process for new snapshots and statistics

5.1.1 Update Process. The update process is depicted in Figure 2 and consists of three steps. The first step corresponds to a (parallel or sequential) import of one or more new snapshots. In the second step, current statistics are updated and (if required) new statistics are calculated. In the last step, a new version number is associated to the test data, versioning-related meta data are updated and the new version is published. As illustrated by this figure, the update process (and thus the creation of a new version) can be triggered by two reasons:

- new snapshots are available, or
- new statistics are required.

Logically, in the second case, the first step is skipped and the update process starts immediately with the second one.

5.1.2 Reproducibility. As described in Section 3.3, the import of new data poses some challenges to the repeatability of evaluation processes that were using previous versions of our test dataset. Since no record is ever removed from the test dataset, it grows monotonically, which means that the set of all records of the current version will always be a subset of all records of every future version. Thus, theoretically, it is sufficient to add a field to every record which is monotonically increasing with every new update. This field can be the date of import or the number of the first version containing this record (the snapshot date is not suitable because it is not monotonically increasing due to belatedly published snapshots). To reconstruct an earlier version, the user can use this field to filter out all records whose field value is greater than the value of the desired version.

However, we also want to allow users to limit their evaluation to an arbitrary subset of snapshots (e.g., a certain time interval). For doing this, we additionally store an array with the dates of all snapshots containing the corresponding record.

To reconstruct statistics, such as the number of records or snapshots per cluster, we enriched the meta data of every duplicate cluster by a map that counts how many new records were inserted per snapshot. The reconstruction of similarity scores is discussed in Section 5.2.

5.2 Storing Similarity Scores

To support users in customizing their data by (i) removing further near exact duplicates, (ii) repairing potentially unsound clusters, and (iii) restricting the data to a certain range of heterogeneity, we associate every record with three version-similarity maps (one for plausibility and two for heterogeneity). Every value of each of these maps corresponds to another map that assigns a similarity score to each of the previously existing records of the same cluster. Since the order of these records never change, this approach does not only avoid expensive recalculations, it also enables reproducibility because existing similarity scores are never updated or deleted. While the first heterogeneity map takes all attributes into account, the second is limited to the personal attributes in order to facilitate a customization of personal data[9]. In addition every cluster is associated with three version-similarity maps storing the aggregated values of their records.

Our understanding of plausibility and heterogeneity may change over time and/or we just may want to use other similarity measures to compute them. The versioning of the similarity scores protects reproducibility against such future changes, because we create a new version every time we use a new measure.

[9]Note, the plausibility is already limited to personal attributes (see Section 6.2) and thus does not need to be stored twice.

	NCID†	*last_name*	*first_name*	*midl_name*	*sex*	*age*	*year*◇
r_1	XX001	LARRELL	LEWIS	ANTWAN	MALE	17	2014
r_2	XX001	LEWIS	LARRELL	ANTAWN	MALE	18	2015
r_3	XX001	LEWIS	LARRELL	A	MALE	22	2018
r_4	ZZ002	FIELDS	MARY	ELIZABETH	FEMALE	62	2012
r_5	ZZ002	BETHEA	JOSHUA	ELIZABETH	MALE	92	2014

†The NCIDs are pseudonymized for data privacy reasons.
◇The snapshot year in which this record was collected.

Figure 3: Examples of erroneous and unsound clusters

6 EXPERIMENTAL EVALUATION

When we explored the snapshots, we noticed several errors within the data. Some records contain typos, abbreviations, or have values confused between attributes. Moreover, some notations have changed over time (e.g., ‚1ST CONGRESSIONAL‘ vs. ‚CO. DISTRICT 1‘ or ‚66 AND ABOVE‘ vs. ‚Age Over 66‘). One example is presented in Figure 3. The first and last names of at least one record of voter XX001 got mixed up. In addition, either the middle name of r_1 or r_2 contains a typo (‚ANTWAN‘ vs. ‚ANTAWN‘) and the middle name of r_3 is abbreviated. Remember that a proper evaluation of duplicate detection algorithms requires errors of many different types and not just outdated values. Thus, such real-life data errors are very welcome in our test dataset, since they challenge the detection of duplicates, but do not corrupt the gold standard. However, we also detected some duplicate clusters that contain records that hardly refer to the same person. An excerpt of one of those examples is depicted in Figure 3 where the two records r_4 and r_5 share the same NCID, but obviously describe different persons. Such unsound clusters are a real threat to the quality of our test data because they spoil the gold standard and thus will negatively affect every future evaluation if they remain in the dataset.

In order to evaluate the quality and usability of our test dataset beyond these first impressions, we conducted a series of experiments, which are described in the rest of this section.

6.1 Evaluated Datasets

To better understand the results of the evaluation of our test dataset, we compare them with those of three manually labeled test datasets that are commonly used in the literature. We acquired all three datasets as TSV files[10] from the dataset repository of the Hasso Plattner Institute[11].

- *Cora*: This dataset contains bibliographical information on scientific papers including title, authors, publisher and year. The schema of the TSV file consists of 17 attributes including an artificial id. The file contains 1,878 records which form 182 clusters.
- *Census*: This dataset contains personal information including name values (first, middle and last), an address and a zip code per person (6 attributes in total). It contains 841 records which form 483 clusters.

[10]We used the non-prepared versions where special characters are not removed.
[11]http://hpi.de/naumann/projects/repeatability/datasets

Table 3: Characteristics of evaluated datasets

	Cora	Census	CDDB	NC1	NC2	NC3
#records	1,879	841	9,763	24,761	22,739	25,530
#attributes	17	6	7	38	38	38
#duplicate pairs	64,578	376	300	19,916	15,993	22,735
#clusters	182	483	9,508	10,000	10,000	10,000
#non-singletons	118	345	221	10,000	10,000	10,000
max. clustersize	238	4	6	7	7	8
avg. clustersize	10.32	1.74	1.03	2.45	2.27	2.55
max. heterog.†	0.63	0.46	0.65	0.25	0.43	0.72
avg. heterog.†	0.171	0.15	0.217	0.106	0.305	0.433

†The presented heterogeneity scores are pair-based.

- *CDDB*: This dataset includes information on 9,763 music CDs randomly extracted from freeDB[12]. In the TSV version of this dataset, all tracks of a CD are concatenated to a single string by using the pipe symbol as a delimiter. After doing so, the schema of this file consists of 7 attributes. The 9,763 records form 9,508 clusters.

For all three datasets the duplicate information is provided by a list of pairs. Several characteristics of these datasets are presented in Table 3. Interestingly, the duplicate distributions of these sets are quite different. Whereas the Cora dataset contains very large clusters (up to 238 records) and its average cluster size is 10.32, the maximal and average cluster sizes of the Census and CDDB datasets are quite small (1.74 or 1.03 respectively). The three datasets NC1 to NC3 are described in Section 6.5.

6.2 Plausibility Check

As we have explained in Section 3.1.1, it is very important that the gold standard of the test data is sound. However, as shown in Figure 3, we have also seen that this does not always seem to be the case. To keep the threat of an unsound gold standard to a minimum, we performed a quality check by calculating a plausibility score for every pair of duplicate records.

In this plausibility check, we have the basic assumption that all records of the same cluster are duplicates and the similarity scores should only reflect (significant) contradictions to this assumption. Consequently, the similarity measure should compensate simple errors and differences in data representation as we know it from duplicate detection algorithms. Due to our basic assumption, this compensation can be even stricter. Therefore, word confusions within an single attribute value or between different values as well as missing or abbreviated values should not reduce similarity at all, because they are more an indication of unknown or erroneous values than a clear sign of a non-duplicate. Moreover, to compensate outdated values we should only use attributes whose values rarely change and that are either very identifying (i.e., two records with the same value are likely duplicates) or discriminating (i.e., two records with different values are likely no duplicates). In our use case, we decided to use:

- the three name values (first, middle and last),
- the sex code,
- the year of birth (which we derived from the snapshot-date and the age value), and
- the place of birth.

It is not uncommon that values are confused between the three name attributes. Thus, we computed a single *name similarity* before combining it with the similarity scores of the other attributes. To compensate errors in the name order, but also within the individual name values (e.g., typos), we computed the name similarity by using the hybrid Generalized Jaccard Measure [8] with an extended version of the Damerau-Levenshtein Similarity [4] as the internal token similarity measure, i.e.:

$$sim_{name}(o_i, o_j) = GenJacc_{DamLev}(names(o_i), names(o_j)) \quad (1)$$

where $names(o_i) = \{fname(o_i), mname(o_i), lname(o_i)\}$.

The Damerau-Levenshtein Similarity was extended to a proper handling of missing and abbreviated values. The comparison to a missing value results in a similarity of 1. The same holds true if one token is a prefix of the other because in both cases we do not have any evidence to mistrust the given data. In the case of the sex code, typos and different representations can almost be excluded for sure. Thus there are actually only four possibilities: The compared values agree (i.e., $sim_{sex} = 1$), disagree (i.e., $sim_{sex} = 0$), one of them is undesignated (i.e., it has the value ‚U') or missing. Since we do not have any contradiction in the later two cases, we set the similarity to $sim_{sex} = 1$, too.

We computed the year of birth (short YoB) as *snapshot-date* − *age*. Since the actual YoB can be one year earlier if the person has not yet had birthday when the snapshot was made, we introduced a tolerance of 1. Moreover, we assumed a similarity of 0 if the age difference was 10 or greater. This led to the following formula:

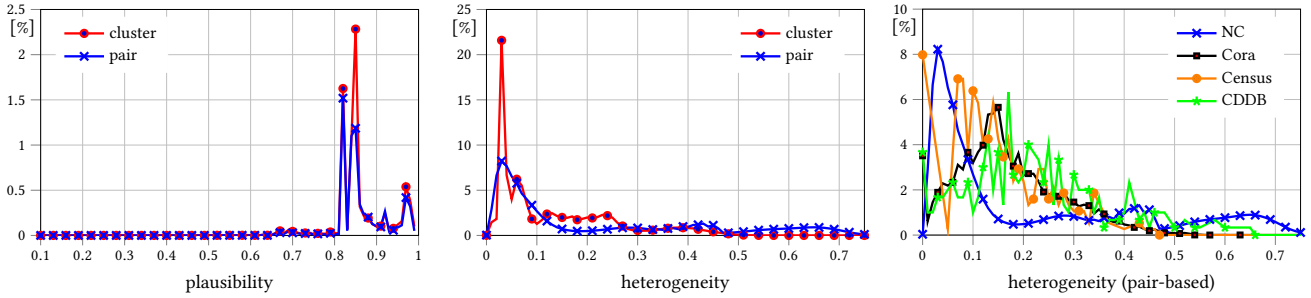
$$sim_{YoB}(o_i, o_j) = 1 - min\left(1, \frac{max(0, |YoB(o_i) - YoB(o_j)| - 1)}{10}\right) \quad (2)$$

In the case of the place of birth we simply computed the extended Damerau-Levenshtein Similarity between the two values. The final similarity score was then calculated as the weighted average of the previously presented scores where we considered the name similarity to be more important (weight 0.55) than the others (each 0.15). A cluster is already unsound, if only one of its records refers to another voter regardless of how plausible the other records are actually duplicates. Thus, we computed the plausibility of a cluster as the minimal plausibility of its records.

We performed our plausibility check on the dataset with 120 million records (exact duplicates were removed after trimming). The results show that only a few clusters of this dataset are highly suspicious to be unsound. The average cluster plausibility is 0.988. 91.7% of all clusters (and 93.3% of all duplicate pairs) have the maximum possible value 1.0. The distribution of the remaining 8.3% (or 6.7% resp.) are presented in Figure 4a. The minimal plausibility of all clusters (and pairs) is 0.06. 6.4% of all clusters have a plausibility lower than 0.9, 0.47% (=61,548 clusters) lower than 0.8 and only 0.0049% (=641 clusters) lower than 0.5. The pair-based values are similar. As a comparison, the two clusters from Figure 3 have a plausibility of 0.82 (XX001) and 0.33 (ZZ002) respectively which matches our intuition that the differences in the first cluster are probably the result of data errors in the name values while the second cluster contains obvious non-duplicates.

An appropriate scoring of plausibility heavily depends on the domain of the data, since we should only use attributes that are less volatile and are either very identifying or discriminating. Moreover, it also depends on the quality of the data, since typical error patterns (e.g., an incorrect encoding of special symbols) are no significant evidence for an unsound cluster and should therefore be compensated in the scoring process. It is therefore difficult to make comparisons between the plausibility of datasets defined on different schemas without creating any noticeable bias. For this reason, we decided not to include such a plausibility calculation for the Cora, Census and CDDB datasets.

[12]http://www.freedb.org/



(a) Plausibility of NC test dataset **(b) Heterogeneity of NC test dataset** **(c) Heterogeneity of NC, Cora, Census and CDDB**

Figure 4: Plausibility and heterogeneity distributions of clusters and duplicate pairs for different datasets

6.3 Duplicate Heterogeneity

To score the heterogeneity between two duplicate records we want to take all their differences into account and thus do not actually want to apply measures that compensate them. At the same time, there are differences that should cause a larger heterogeneity than others. For example, difference in upper and lower case or confusions of tokens are less significant than replacing the original strings with completely different letters or tokens.

To address the problem resulting from uppercase letters, we decided to compare every two values one time with and one time without lowercasing them. To address the problem of token confusions, we decided to compare every two values one time with a sequential and one time with a hybrid similarity measure. Together this results in four comparisons for every two values. We used the average of the four resulting scores as final similarity. As the sequential similarity measure, we chose Damerau-Levenshtein. Since the Generalized Jaccard Measure is computationally too expensive when working on 90 attributes, we used the Monge-Elkan Similarity[13] [23] as the hybrid measure instead (using Damerau-Levenshtein as the internal token similarity measure)[14]. To calculate the heterogeneity between two records, we used the weighted average of their inverse value similarities (i.e., the records are the more heterogeneous, the less similar they are).

If the resulting heterogeneity scores should reflect any kind of difficulty to detect the corresponding fuzzy duplicates, identifying attributes, such as names, should be weighted higher than others. At the same time and in contrast to plausibility, we want to compare the heterogeneity (i.e., dirtiness) of different datasets in order to enable an evaluation of algorithms with respect to a varying dirtiness of the data. To ensure a fair comparison, no context knowledge and external expertise should be included into the scoring process and all necessary information should be taken from the dataset itself. To accomplish this, we used the same similarity measure for all attributes and weighted every attribute by its uniqueness[15], where we quantified this uniqueness by the attribute's entropy [19]. Since duplicate records distort these uniqueness scores (e.g., an otherwise unique id can occur multiple times), we initially created a canonical record[16] per duplicate cluster and used them to compute the weights. The heterogeneity of a cluster was then computed as the average heterogeneity of its records. Since a consideration of clusters of size one is pointless, we restricted our analyses to clusters with at least two records.

The results of our analysis are depicted in Figure 4b. They show that – despite of outdated values and other data errors – most of the duplicate records are very similar and the dataset as a whole is quite clean and homogeneous. However, since we removed exact duplicates, almost none of the clusters is completely homogeneous and most of them have a heterogeneity of around 0.03 (21.6%). In the case of the duplicate pairs, we do not have such a large peak, but also here you can see, that most values are in the range between 0.02 and 0.06. Considering this fact, the average heterogeneity (0.13 for clusters and 0.218 for pairs) is surprisingly large. The maximal heterogeneity (0.79 for clusters and 0.88 for pairs) is also very high. As a comparison, the two clusters from Figure 3 have a heterogeneity of 0.395 (XX001) and 0.366 (ZZ002) respectively. Interestingly, the less plausible cluster is more homogenous. That is because although cluster ZZ002 contains records referring to different persons, these records form two very homogenous groups (one with six records similar to r_4 and one with four records similar to r_5).

To score the heterogeneity of the Cora, Census and CDDB datasets, we used the same settings (i.e., the same similarity measures and all attributes are weighted based on their entropy). The pair-based distributions of all three datasets are depicted in Figure 4c. The heterogeneity scores of the Cora dataset are almost normally distributed. Most of them have a value of 0.15, the maximal value is 0.63 and the average is 0.171. The Census dataset has three peaks at 0, 0.07 and 0.1, a maximal heterogeneity of 0.46 and an average of around 0.15. In general, except of the large number of very homogenous pairs, its distribution has some similarity to this of the Cora dataset, but is less regular. The CDDB dataset is the dirtiest of them. Its maximal heterogeneity is 0.65 and its average is 0.217. The high heterogeneity results primarily from the fact that many language-specific symbols, such as accents, were incorrectly converted when the dataset was created. Thus, many duplicate records are very dissimilar if we do not compensate those irregularities in the matching process or repair it during preparation.

If we compare these distributions with the one from the NC dataset, we see that there is only little resemblance to our voter data although the average of the NC dataset is very close to that

[13] Since this measure is non-symmetric, we computed it in both directions and used the average as final score.

[14] There are millions of possible similarity measures (including various settings) and we had to choose one even though this could possibly generate a bias in the evaluation processes using this test data. However, as illustrated in Section 6.5, this bias was almost not existing in our experiments.

[15] To ensure reproducibility, these weights must not change although the dataset will grow with future updates. We therefore limited their computation to the first 5 million clusters and then hard-coded them into the source code.

[16] These records are built by using the most frequent value per attribute.

Table 4: Statistics of different irregularities within the NC, the Cora and the Census datasets

			NC (total 137 M pairs)			*Cora* (total 65 K pairs)			*Census* (total 376 pairs)		
	error type	*example*[†]	*most frequent attribute*	*frequency total*	*frequency in %*	*most frequent attribute*	*frequency total*	*frequency in %*	*most frequent attribute*	*frequency total*	*frequency in %*
singleton[◇]	outlier	age = 5091	age	280 K	0.48	year	20	1.06	last_name	5	0.59
	abbreviation	midl_name = A.	midl_name	7.4 M	12.6	booktitle	1	0.05	middle_name	671	79.8
	missing	mail_addr3 = *null*	mail_addr3	58.7 M	100	institution	1.7 K	86.8	middle_name	170	20.2
pair-based[★]	typo	ADELL ↔ ADELE	midl_name	1.2 M	0.87	title	15 K	22.6	last_name	243	64.6
	OCR-error	DICOL3 ↔ DICOLE	last_name	1.3 K	0.00	-	-	-	-	-	-
	phonetic	WHITE ↔ WYATT	midl_name	1 M	0.75	title	29 K	44.4	last_name	200	53.2
	prefix	KIM ↔ KIMBERLY	midl_name	5.4 M	3.94	volume	19 K	29.1	first_name	65	17.3
	postfix	BRAGG ↔ FORT BRAGG	last_name	230 K	0.17	pages	5 K	8.03	street_address	5	1.33
	formatting	JRS RIDGE ↔ J.R.S RIDGE	midl_name	208 K	0.15	title	27 K	42.4	street_address	20	5.32
	transp. tokens	KIM DUC ↔ DUC KIM	race_desc	748 K	0.55	authors	202	0.31	-	-	-
	confused values	(LUKE, HAL) ↔ (HAL, LUKE)	first/midl_name	21 K	0.02	title/booktitle	20	0.03	-	-	-
	integrated value	(SUE ANN, *null*) ↔ (SUE, ANN)	midl/last_name	244 K	0.18	title/year	4	0.01	-	-	-
	scattered values	(NGAN HA, THI) ↔ (NGAN, HA THI)	midl/last_name	24 K	0.02	-	-	-	-	-	-

[†]Selected from any attribute of the NC dataset.
[◇]Singletons are normalized using the total number of records (NC = 58.7 M, Cora = 1,879, Census = 841).
[★]Pair-based irregularities are normalized using the total number of duplicate pairs (NC = 136.7 M, Cora = 65 K, Census = 376).

of the CDDB dataset. Interestingly, the majority of pairs of the NC dataset is much cleaner than those of the other three datasets, but it has also a higher percentage in the range between 0.4 and 0.7. Thus, our dataset contains many homogenous records, which may need some additional pollution (see Section 8). However, as we will show in Section 6.5, its large dispersion allows us to easily achieve an average heterogeneity of 0.433 (which is significantly higher than this of the CDDB dataset) by adjusting the voter data based on the precalculated heterogeneity scores.

6.4 Diversity of Error Types

To allow an extensive evaluation of the capabilities of duplicate detection algorithms, we chose source data whose collection process promises many different types of errors. To test this assumption, we conducted a statistical analysis on the personal attributes of our test dataset by searching for several kinds of irregularities within these data (see Table 4). Here, we distinguished between irregularities that can be identified by analyzing single records (so-called *singletons*) and ones that can only be detected by comparing two duplicate records (so-called *pair-based* irregularities). In the first case, we evaluated every record individually leading to a frequency that can be normalized using the total number of records. In the second case, we compared every two duplicate records, counted the number of times the individual irregularities occur and normalized them using the total number of duplicate pairs. Moreover, we distinguished between irregularities that concern a single attribute and those that concern multiple attributes (record-level). We evaluated the following singletons:

- *outlier*: A value that is outside a predefined range (e.g., age > 110) or contains a character that is unusual for its associated domain (e.g., the first name ‚X ÆA-12‘)[17].
- *abbreviation*: A value that consists of a single letter, possibly followed by a punctuation mark.
- *missing*: A value that is null, an empty string or any other value indicating missing information (e.g., ‚-‘ or ‚unknown‘).

As pair-based irregularities, we analyzed:

- *typo*: Two values whose lowercase versions differ only in one character or contain a character transposition. These are exactly those values having a Damerau-Levenshtein distance of 1. In order not to interpret a complete replacement of one value by the other as a typing error, we only considered values longer than two characters.
- *OCR-error*: Two distinct string values which only differ at those positions where one of them has a digit. If both characters are digits, they need to be identical.
- *phonetic error*: Two values that are not identical after removing non-letter characters, are both longer than two and have the same Soundex code.
- *prefix/postfix*: Two values where one of them is a prefix/-postfix of the other after removing a potential punctuation mark from the end of the shorter value. Such pre- and postfix situations indicate abbreviations or forgotten token/characters.
- *different formatting*: Two values that only differ in non-alphanumerical characters (e.g., a hyphen, space or punctuation mark between tokens).
- *transposed tokens*: Two values whose token sets are identical, but their token order is different.
- *confused values*: Two records whose values are confused between two different attributes (e.g., the first and last name of one record are transposed).
- *integrated value*: Two records where in one of them the value of one attribute is integrated into another (e.g., a middle name stored as a second token in the first name).
- *scattered values*: Two records having the same set of tokens assigned differently to two attributes. To avoid possible overlaps with the previous two types, we only counted scattered values that are not integrated or confused.

Obviously some of these irregularities overlap (or sometimes even include each other) so that we counted some errors for more than one type. For example, some OCR-errors are also typos. Moreover, it is important to note that we consider these irregularities as indications of particular error types, but cannot always classify them with absolute confidence. For instance, not every two distinct values that have the same Soundex code are an actual phonetic error. This also applies to irregularities on record-level. Not every assignment of the same value to different

[17]Note that not every outlier corresponds to an actual data error.

attributes corresponds to a mistake. For example, in the U.S., it is not atypical to take the old last name as the middle name when getting married. Such a constellation can therefore also indicate an outdated record instead of a data entry error. Nevertheless, the errors are real even if their assignment to the individual types may be disputed. It should also be mentioned that our definitions of the individual error types do not cover them completely so that our analysis was not able to find every actual error. For example, OCR errors that do not contain digits (e.g., ‚Tim' vs. ‚Tirn') were not recognized as such. However, despite these minor inaccuracies, we think that our experiment provides a good overview of the wide variety of different error types contained in the voter data.

The results of this analysis are presented in Table 4 (grouped by singletons and pair-based irregularities), were we list the absolute values in combination with their percentages. To achieve greater comparability, we also evaluated the Census and the Cora datasets. We selected the Census dataset because it has a similar domain as our voter data and selected the Cora dataset because it has similarly large clusters. As we can see, the percentages of the Census and the Cora datasets are much higher than those of the NC dataset. For example, the percentage of typos in the attribute *last_name* of the Census dataset is 65%. This means that out of 376 duplicate pairs, 243 differed in this attribute by only one character or had two consecutive characters transposed. Although the percentages of the NC dataset are much smaller than those of the Cora and the Census datasets, the absolute numbers are many times larger. This shows the potential for customizing smaller datasets with higher error percentages, but still containing million of records. Moreover, the NC dataset contains irregularities that are (almost) not contained in the Cora and Census datasets. Examples are OCR-errors or errors that affect more than one attribute.

6.5 Usability

There are various ways to customize a test dataset by using the precalculated heterogeneity scores. In our experiments, we sketched a very simple approach and let the development of more sophisticated approaches to the user (or future research resp.). This approach consists of three steps. In the first step, we defined a lower and an upper bound $h_\perp$ and $h_\top$ for the heterogeneity scores. In the second step, we randomly selected 100,000 clusters from our whole dataset, scanned over all records of every cluster in their sorting order and removed every record whose heterogeneity to its preceding (not removed) records was not in the requested range $[h_\perp, h_\top]$. In the third step, we sorted the reduced clusters by their size and selected the 10,000 largest clusters as test input. We applied this approach for the three settings $(h_\perp, h_\top) \in \{(0.06, 0.2), (0.2, 0.4), (0.3, 0.7)\}$ while restricting the schema to the personal attributes. The result are the three test datasets NC1, NC2 and NC3. The characteristics of these datasets are depicted in Table 3. As these numbers show, even though we only used 0.07% of all clusters as input, these datasets are larger than the Cora, Census and CDDB datasets.

To evaluate the difficulty of detecting fuzzy duplicates within the individual datasets, we applied three duplicate detection algorithms each using another similarity measure[18] (the same for all attributes). The first measure (short *ME/DL*) was the same combination of the Monge-Elkan and the Damerau-Levenshtein Similarity as we used it to calculate the heterogeneity scores (see Section 6.3). The other two measures were the Jaro-Winkler Similarity (sequential) and the Jaccard Similarity using trigrams (token-based) [8]. The similarity of two records was always computed as the weighted average similarity of their values. Since we observed that the name values are sometimes confused between the individual attributes, we matched every combination of them and used the 1:1 matching with the highest similarity for aggregation. To weight the individual attributes we used again their entropy. In this case, however, we calculate it using all records (i.e., including the duplicates), since the user does not know these duplicates in advance. In addition, the entropy was calculated solely based on the records of the customized datasets. Thus, the resulting weights differed from the ones we used to calculate the heterogeneity scores (e.g., 0.66 vs. 0.48 for the first name). In the case of the larger datasets (CDDB and NC1-NC3), we reduced the initial search space by applying a multi pass of the Sorted Neighborhood Method [23] where we conducted one pass for each of the five most unique attributes and used a window of size $w = 20$. Since a few true duplicate pairs were lost through this reduction (always less than 1%), we added them back to the search space before starting the actual matching process.

The results of these duplicate detection algorithms applied to the different test datasets are depicted in Figure 5. As we can see in Figure 5a to 5c, the quality of the duplicate detection algorithms decreased when we increased the heterogeneity of the test data, since the more difficult it was to separate the duplicate from the non-duplicate pairs. In the first case, the test dataset was very clean and we could achieve almost a perfect F_1-score for all three measures. Moreover, for two out of three measures, this score was high for every threshold between 0.65 and 0.85, which made it easier to select an appropriate value for this threshold without knowing these numbers. In the case of the second dataset, the maximal F_1-score was still pretty solid (i.e., close to 0.8), but the threshold had to be set much more carefully and the quality of a setting already depended on the individual measures. For example, for Jaccard the best threshold was 0.57, but for Jaro-Winkler it was 0.75 and there was no threshold that worked well for all measures. Finally, in the case of the last dataset, the maximal F_1-score decreased significantly and even a score of 0.4 was hard to achieve. All this shows that the precalculated heterogeneity scores can be perfectly used to adjust the test data to increase the difficulty of detecting fuzzy duplicates. Moreover, as we can see, the ME/DL Similarity did not perform better than the Jaccard Similarity which shows that using this measure to calculate the test datas' heterogeneity scores has not produced any noticeable bias. Finally, when we compare the results of the three customized test datasets NC1 to NC3 with the results of the Cora, Census and CDDB datasets, we see that they show similar patterns as NC2 in terms of the maximally achieved F_1-score and the shapes and positions of the individual graphs. Only the Census dataset differs a little bit, because here Jaro-Winkler scored much better than for the other datasets and the single graphs correspond less to a bell shape. In summary, this shows that the sheer amount of data of our original test dataset enables us to create test data that are cleaner (NC1), equally clean (NC2) and dirtier (NC3) than these real-life use cases giving us the opportunity to design our test data in the way our evaluation goals require. We repeated this experiment with different parts of our original test dataset as input. Since the compositions of the generated datasets differ slightly, there were also slight differences in the resulting graphs, but the findings were always the same.

[18] Here, we tried to cover a wide range of measures by using a hybrid, a sequential, and a token-based measure.

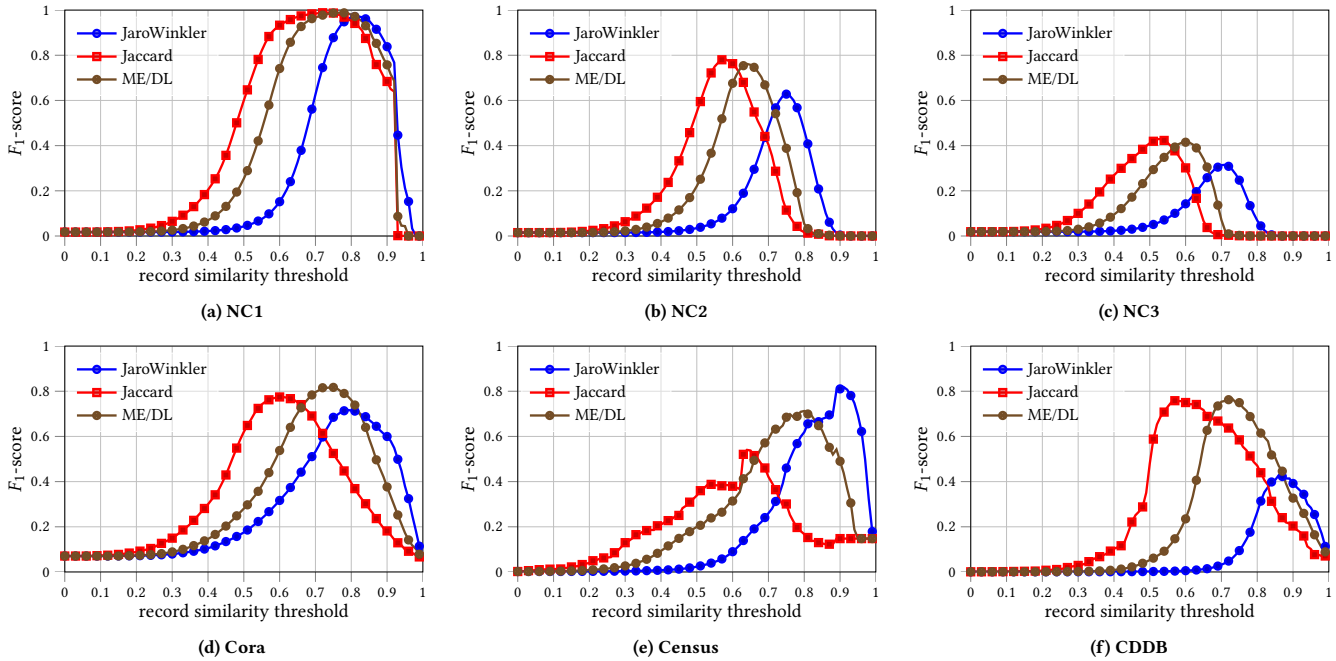


Figure 5: F_1-scores in relation to similarity thresholds for several similarity measures and test datasets

7 RELATED WORK

Almost all existing test datasets have been either artificially generated (e.g., by using an automatic generation tool) or been manually labeled. While artificial datasets have the obvious disadvantage of not containing actual real-life duplicates and errors (including outdated values), labeling duplicates manually is extremely expensive so that this approach is only an option for very small datasets. For example, the commonly used Cora, Census and CDDB datasets contain less than 10,000 records each (see Section 6.1). The essential shortcoming of an automatic labeling approach [32] is that the resulting gold standard is often not sound and biased towards the used algorithms, which can significantly distort evaluation results. One example of artificially created test data is the ERIQ dataset [31] which contains 100,000 records of customer data. Although this number is much larger than those of the manually labeled datasets, it is still small compared to typical big data applications. Further test datasets (manually labeled or artificially generated) can be found on the websites of the Hasso Plattner Institute[11], the University of Leipzig[19], and the Magellan project[20]. To the best of our knowledge, none of the existing test datasets provides precalculated similarity scores that help users to customize datasets on their basis.

Automatic test data generation tools can be divided into two classes: *Data synthetization* tools that generate all data values – including duplicates and errors – from scratch and *data pollution* tools that get a clean dataset as input and pollute it with duplicates, errors and inhomogeneities. Data synthetization tools, such as DBGen [14] or the Febrl Data Set Generator [3], are very efficient so that large datasets can be generated in short time [15]. However, since every data value is fictional, it is almost impossible to guarantee that the resulting values patterns are realistic. Data pollution tools, such as GeCo [6], TDGen [2], or DaPo [15], are the best option to generate test data with realistic value patterns because real-life data can be used as input. Moreover, if a broad spectrum of error types is supported they are nearly domain-independent. Except DaPo, however, existing pollution tools are strongly limited with respect to their scalability so that generating large datasets is either impossible or extremely time consuming [15]. A major problem that all these tools have in common is an appropriate simulation of outdated values and the complex error patterns that result from them.

Table 5: Previous usage of North Carolina voter data

	number of				cluster sizes†	
paper	records	attr.	clusters	dupl. pairs	avg	max
[18]	14,183	25	?	?	?	?
[5, 10]	8,261,838	19	8,110,137	155,469	2.02	6
[11]	200,000	6	100,000	200,000	2	2
[30]	5,000,000	4	3,500,840	3,331,384	4	5
[30]	10,000,000	4	6,625,848	14,995,973	7.7	10

†of non-singletons

Data of the NC voter register have been already used as test data in several works (see Table 5). Alas most of these uses are not fully documented and/or the provided links are outdated[21]. Thus, we cannot say exactly which data were used as input. The small size of the first dataset indicates that only a small portion of the voter register was used. The second dataset was created by Christen [5] in an earlier attempt to use the temporal changes of the voter data for generating test data with realistic outdated values. He regularly downloaded the current voter registration file on a bi-monthly basis over a time period of three years and combined these self-made snapshots after removing exact duplicates. However, instead of using the inherent gold standard

[19]https://dbs.uni-leipzig.de/en/research/projects/object_matching/benchmark_datasets_for_entity_resolution
[20]https://sites.google.com/site/anhaidgroup/useful-stuff/data
[21]ftp://www.app.sboe.state.nc.us [10, 11] and ftp://alt.ncsbe.gov/data/ [5]

provided by the NCID, he created it artificially by applying a rule-based duplicate detection approach. Thus, this gold standard is not guaranteed to be sound and biased towards algorithms using a similar detection approach.

Durham et al. [11] randomly selected 100,000 records from the voter register and generated polluted versions of these records by artificially introducing typos, semantic and phonetic errors. The last two datasets were created similarly by artificially polluting a randomly selected set of voter records with duplicates and errors using the GeCo tool [6]. Thus, these three datasets do not contain real-life errors and especially lack in realistic outdated values.

Except the last one, the sizes and duplicate distributions of these five datasets are nowhere near the numbers of the test dataset we generated in our study. Moreover, none of these authors enhanced their data with useful statistics, such as plausibility and heterogeneity scores, as we did in this study. Finally, to the best of our knowledge, we are the first who discuss the aspects of quality, usability and reproducibility in the context of test data for duplicate detection in such a depth.

8 CONCLUSION & FUTURE WORK

In this paper, we presented a large-scale dataset for evaluating duplicate detection algorithms and the approach behind its generation. Extracted from an historical voter register from North Carolina, our test dataset contains more than 120 million records and 640 million duplicate pairs making it uniquely suitable for evaluating duplicate detection at scale. Besides the dataset's size, our study focused on its quality, usability and reproducibility. The records' historical nature means that they contain many outdated values and – since data was often entered manually – also a large variety of other error types, such as typos, phonetic errors or confusions between attributes. While the data values themselves contain many errors, the underlying gold standard is largely error-free which is a mandatory requirement to ensure meaningful evaluation results. In general, its large size as well as its large number of different data errors makes the dataset perfectly suitable for users who want to customize their own datasets based on the requirements of their respective evaluation goals. To support such customizations, we equipped the individual records with similarity scores modeling their plausibility and heterogeneity. Finally, we integrated several mechanisms to ensure reproducibility when the dataset is growing with future updates. In summary, our approach enables generating large-scale test datasets with realistic errors including outdated values (which are hard to synthesize) and without the need for labeling duplicates manually (which is extremely labor-intensive).

Our plans for future work targets two different ways to extend our approach. First, we intend to generalize the procedure described here and apply it to historical corpora from other domains. This will provide the research community with large-scale test datasets beyond use cases that revolve around personal data. Second, we plan to combine our approach with a scalable data pollution tool, such as DaPo, to unite the strengths of having real outdated values and being able to inject additional errors at will. Our goal here is to increase the flexibility for customization and thereby facilitate generating test datasets geared for specific user demands. We think our presented work is useful to other researchers and we hope that our current line of research will pave the way for novel solutions that combine approaches using historical data with methods of data pollution in creative ways.

ACKNOWLEDGMENTS

We thank the staff of the North Carolina State Board of Elections for their extensive and valuable discussions on the structure and origins of the data.

REFERENCES

[1] North Carolina General Assembly. 2020. NC General Statutes. https://www.ncleg.gov/Laws/GeneralStatutesTOC. [Online; accessed 14-02-2021].
[2] Tobias Bachteler and Jörg Reiher. 2012. *Tdgen: A Test Data Generator for Evaluating Record Linkage Methods.* Technical Report wp-grlc-2012-01. German Record Linkage Center.
[3] Peter Christen. 2009. Development and User Experiences of an Open Source Data Cleaning, Deduplication and Record Linkage System. *SIGKDD Explorations* 11, 1 (2009), 39–48.
[4] Peter Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer.
[5] Peter Christen. 2014. *Preparation of a Real Temporal Voter Data Set for Record Linkage and Duplicate Detection Research.* Technical Report. The Australian National University.
[6] Peter Christen and Dinusha Vatsalan. 2013. Flexible and Extensible Generation and Corruption of Personal Data. In *CIKM, USA*. 1165–1168.
[7] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. *Entity Resolution in the Web of Data.* Morgan & Claypool Publishers.
[8] AnHai Doan, Alon Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann.
[9] Xin Luna Dong and Divesh Srivastava. 2011. *Big Data Integration* . Morgan & Claypool Publishers.
[10] Uwe Draisbach, Peter Christen, and Felix Naumann. 2020. Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection. *ACM J. Data Inf. Qual.* 12, 1 (2020), 3:1–3:30.
[11] Elizabeth Ashley Durham, Murat Kantarcioglu, Yuan Xue, Csaba Tóth, Mehmet Kuzu, and Bradley A. Malin. 2014. Composite Bloom Filters for Secure Record Linkage. *IEEE Trans. Knowl. Data Eng.* 26, 12 (2014), 2956–2968.
[12] Venkatesh Ganti and Anish Das Sarma. 2013. *Data Cleaning: A Practical Perspective.* Morgan & Claypool Publishers.
[13] Lise Getoor and Ashwin Machanavajjhala. 2013. Entity Resolution for Big Data. In *KDD*. 1527.
[14] Mauricio Hernández and Salvatore Stolfo. 1998. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Min. Knowl. Discov.* 2, 1 (1998), 9–37.
[15] Kai Hildebrandt, Fabian Panse, Niklas Wilcke, and Norbert Ritter. 2020. Large-Scale Data Pollution with Apache Spark. *IEEE Trans. Big Data* 6, 2 (2020), 396–411.
[16] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning.* ACM.
[17] Lars Kolb, Andreas Thor, and Erhard Rahm. 2012. Dedoop: Efficient Deduplication with Hadoop. *Proc. VLDB Endow.* 5, 12 (2012), 1878–1881.
[18] Ioannis K. Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate Detection with Matching Dependencies. *Proc. VLDB Endow.* 13, 5 (2020), 712–725.
[19] Luís Leitão and Pável Calado. 2011. Duplicate Detection through Structure Optimization. In *CIKM*. 443–452.
[20] Michael McDonald. 2019. United States Elections Project. http://voterlist.electproject.org/states/north-carolina. [Online; accessed 14-02-2021].
[21] David Menestrina, Steven Whang, and Hector Garcia-Molina. 2010. Evaluating Entity Resolution Results. *PVLDB* 3, 1 (2010), 208–219.
[22] MongoDB. 2019. MongoDB Documentation. https://docs.mongodb.com/.
[23] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection.* Morgan & Claypool Publishers.
[24] North Carolina State Board of Elections. 2021. Historical Voter Registration Snapshots. https://dl.ncsbe.gov/?prefix=data/Snapshots/. [Online; accessed 14-02-2021].
[25] North Carolina State Board of Elections. 2021. Public Data. https://s3.amazonaws.com/dl.ncsbe.gov/data/ReadMe_PUBLIC_DATA.txt. [Online; accessed 14-02-2021].
[26] North Carolina State Board of Elections. 2021. Voter Registration Data. https://www.ncsbe.gov/results-data/voter-registration-data. [Online; accessed 14-02-2021].
[27] Mateusz Pawlik, Thomas Hütter, Daniel Kocher, Willi Mann, and Nikolaus Augsten. 2019. A Link is not Enough - Reproducibility of Data. *Datenbank-Spektrum* 19, 2 (2019), 107–115.
[28] Julian Risch and Ralf Krestel. 2019. Measuring and Facilitating Data Repeatability in Web Science. *Datenbank-Spektrum* 19, 2 (2019), 117–126.
[29] Pramod J. Sadalage and Martin Fowler. 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley Professional.
[30] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. 2018. Scalable Matching and Clustering of Entities with FAMER. *CSIMQ* 16 (2018), 61–83.
[31] John R. Talburt. 2011. *Entity Resolution and Information Quality.* Morgan Kaufman Publ. Inc.
[32] Tobias Vogel, Arvid Heise, Uwe Draisbach, Dustin Lange, and Felix Naumann. 2014. Reach for Gold: An Annealing Standard to Evaluate Duplicate Detection Results. *J. Data and Information Quality* 5, 1-2 (2014), 5:1–5:25.