Cleaning and Visualizing a dirty set of restaurant data

Florian Loher
Technical University of Applied Science Regensburg
florian.loher@st.oth-regensburg.de

***Abstract*—This document shows a possible approach to cleaning and visualizing the dirty dataset provided at https://hpi.de/naumann/projects/repeatability/datasets/restaurants-dataset.html. It describes how the data is first audited, then cleaned in MongoDB, removing duplicates, using a common search engine to find correct restaurant names and standardizing road and city names. Lastly the data is visualized by generating a website that contains an OSM map and markers indicating the location of each restaurant.**

***Index Terms*—MongoDB, Data cleaning**

I. Introduction

Big data is a rapidly growing field of research that already gained overwhelming interest in the general public. The amount of data is increasing at an exponential rate an is likely to grow further at this rate. To be able to leverage the power of data, the need for ways to clean is as high as ever.

Data cleaning, also referred to as data scrubbing or data cleansing, is a research field concerned with improving the quality of faulty data. Typical aspects that are sought to be improved are the amount of duplicates, type errors or inconsistencies[1].

In this article I am going to outline a possible approach to detecting duplicates in a dataset of 864 restaurants[1]. I first audit the data. Then I generate a training set, standardize fields and choose *SoftTF-IDF* string matching measure with *Jaro-Distance* as sub measure as the algorithm for duplicate detection. After creating a gold standard for the training set I train the thresholds for restaurant name, phone number and address similarity that determine if two records will be declared duplicates. Lastly the detection algorithm is run on the test data and compared to the gold standard[2]. Using and comparing different sizes of training data I show that even with a training set 10% the size of the test data (86 records, 22 of which are part of duplicates) a recall of $\approx 86\%$ is with a precision of $\approx 95\%$ typically achieved.

[1]Restaurant data provided at https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/projekte/repeatability/Restaurants/restaurants.tsv

[2]Gold standard provided at https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/projekte/repeatability/Restaurants/restaurants_DPL.tsv

A. Related work

Vogel, Heise, Draisbach, *et al.* [2] describe the limitations of the restaurant data set and its gold standard for comparisons of certain duplicate detection algorithms. Bilenko and Mooney [3] use the data of restaurants to evaluate the performance of their proposed two-level learning algorithm for duplicate detection, *MARLIN* (Multiply Adaptive Record Linkage with INduction).

B. Outline

In section II I am going to introduce basic terminology concerning data cleaning and describe the string-matching algorithms I use. Section

II. Duplicate detection

A major field of research is duplicate detection i.e. trying to find methods that recognize if two distinct records in a given data set actually represent the same real world entity. Such pairs of records are called duplicates. The degree to which an algorithm, method or process is able to detect duplicates can be measured if a gold standard addressing the relevant data set is available. A gold standard is a complete list of every duplicate that exists in the data and therefore also a complete list of all non duplicates. The way of comparing the algorithm to the gold standard is by calculating its precision and recall. Precision is defined as $\frac{|TP|}{|TP|+|FP|}$ where $|TP|$ is the number of records both the algorithm and the gold standard recognize as duplicates (true positives) and $|FP|$ is the number of records the algorithm recognizes as duplicates but the gold standard does not (false positives). Recall is defined as $\frac{|TP|}{|TP|+|FN|}$ where $|TP|$ is the same as before and $|FN|$ is the number of true duplicates the algorithm does not find.

A. String matching

Duplicate detection relies on the matching of strings i.e. comparing fields of distinct records in a given dataset and calculating how similar they are, based on a chosen measure.

Token-based measures are typically optimized to handle differences where substrings are swapped or rotated. Frequent differences between fields that represent the same entity are for example swapping first and last name, or having a title prepended or put at the end. By splitting

the strings into tokens and comparing the resulting sets of strings these rotations cease to impact those measures.

WHIRL, a token-based similarity measure proposed by [4] additionally utilizes the *TF-IDF* (term frequency, indirect document frequency) weighing scheme. This also prioritizes terms that appear rarely in the document giving credit to the thought that strings are probably semantically equivalent if they share a lot of defining terms whereas frequent terms, such as *street* in a field that represents an address, are less likely to point to duplicates.

A further subtlety is introduced by [1] with the measure *SoftTF-IDF*. Instead of needing exact matches for the tokens like WHIRL, SoftTF-IDF utilizes a character-based similarity measure to calculate similiarity values for tokens that are not exact matches.

Character-based similarity measures often handle typing errors very well and are typically optimized for specific types of strings such as names.

B. Matching algorithms for names and addresses

III. Auditing and data preparation

Before trying to find duplicate data in any given dataset most data cleaning approaches start with a phase of preparing the data. Data preparation usually contains the steps parsing, transformation and standardization. This includes but is by far not limited to discovering which types of fields are present in the data and removing unnecessary characters from fields.

In the case of the restaurants data I am looking at a quick audit reveals that entries in the "phone" fields do not conform to a common format. The phone field of record 97 is *212/627-8273* while record 98 contains the value *212-627-8273*. The separation characters between numbers are not uniform. This can be observed throughout the entirety of the data set. A non extensive record of further inconsistencies can be found in table I.

In the parsing and transformation phase I copy the data to an instance of *MongoDB*[3] in order to make the data easier accessible in the following stages. Each row of the original data is transformed into a record in the database. In MongoDB the equivalent to tables of relational databases are collections. Each step of my data cleaning process creates a new collection with the updated data.

The process of standardization is split up, targeting each field of the records separately. Because of that the order in which the fields are standardized is irrelevant to the resulting data.

Each field is standardized by the use of different regular expressions that are suited to find the irregularities in the field. Furthermore replacements are done using dictionaries of standardized values. For example the address field contains street types which are sometimes abbreviated. One regular expression is used to find all abbreviations in a field and a dictionary containing street types and their common abbreviations enables the replacement of each instance by its non abbreviated form.

[3]MongoDB is a document database. For more information visit https://www.mongodb.com/

field	type of inconsistency	example
city	– irregular use of abbreviations – use of city district name as city name	sometimes *la* instead of *los angeles* *hollywood* instead of *los angeles*
type	– irregular use of abbreviations – use of different separators for types	sometimes *bbq* instead of *barbecue* *greek and middle eastern* and *russian/german*)
address	– irregular use of abbreviations – presence of directions – irregular choice of abbreviations	sometimes *blvd* instead of *boulevard*, sometimes *ninth* instead of *9th* *60 w. 55th st. between 5th and 6th ave.* *blvd* or *blv*
phone	– irregular use of separation characters	sometimes *212/627-8273* instead of *212-627-8273*

Table I: Types of inconsistencies present in restaurant data

IV. Generation of training data

Since similarity measures typically return a value between 0 and 1 thresholds must be set for which two records are considered semantically equivalent. One way of finding such thresholds is to set them according to prior experience. A similar approach is the use of training data. In this case I am using a subsets of the original data of different sizes. In section VIII I will compare the effect the size of the training data has in regards to precision and recall.

The subsets are chosen randomly with the restriction of including duplicates at the same ratio as in the complete data set. This is necessary as random pairs of records are very unlikely to be duplicates. Considering for example a set of 100 records containing 50 pairs of duplicates. Regardless which records are chosen both of them are going to be part of a duplicate pair. The probability that both records are part of the same duplicate pair is $P(x \in D) \cdot P(y \equiv x) = 1 \cdot \frac{1}{99} \approx 1,01\%$, where $P(x \in D)$ it the probability that a randomly chosen record x is part of a duplicate and $P(y \equiv x)$ is the probability that the two randomly chosen records x, y are part of the same duplicate. This calculation shows that a completely random subset is likely to contain a duplicate ration that is considerably lower than the original data. However it is exactly those duplicates that greatly contribute to the value of the ratio. Therefore it is necessary to include at least some duplicates in the training set deliberately.

Although finding all duplicates in a data set is very difficult and most often requires a lot of manual work, finding a small subset of duplicates is comparatively easy since not all pairs of records need to be considered. One way of finding duplicates may be sorting the records and comparing adjacent records. This process is deliberately

left out in this work. Instead the provided gold standard is utilized to ensure the same ratio of duplicates in the training set as in the original data.

V. Choice of string-matching algorithm SoftTF-IDF

As described in section II different measures exist for calculating the similarity between fields of records. In order to mitigate the effect of swapped elements in the string a token-based measure is advantageous. The TF-IDF measure also prioritizes terms that appear rarely in the document giving credit to the thought that strings are probably semantically equivalent if they share a lot of defining terms whereas frequent terms, such as *street* in a field that represents an address, are less likely to point to duplicates[4].

Furthermore to decrease the risk of terms not matching because of typing errors or irregular elements that were not targeted by the standardization process the comparison between terms is itself done by a similarity measure, the JaroDistance in this case. The resulting SoftTF-IDF measure is implemented by the python library *py_stringmatching*[4]

VI. Training with training data

VII. Detection of duplicates in restaurant data

VIII. Comparison of different sizes of training data

IX. Conclusions

References

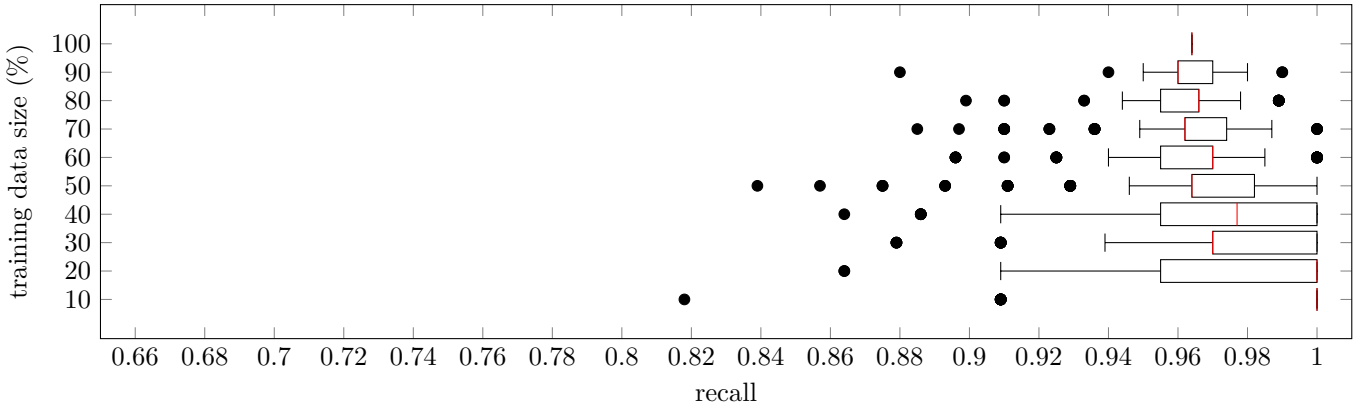
[1] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg, "Adaptive name matching in information integration", *IEEE Intelligent Systems*, vol. 18, no. 5, pp. 16–23, 2003, ISSN: 1541-1672. DOI: 10.1109/MIS.2003.1234765.

[2] T. Vogel, A. Heise, U. Draisbach, D. Lange, and F. Naumann, "Reach for gold, An annealing standard to evaluate duplicatedetection results", *Journal of Data and Information Quality*, vol. 5, no. 1-2, pp. 1–25, 2014, ISSN: 19361955. DOI: 10.1145/2629687.

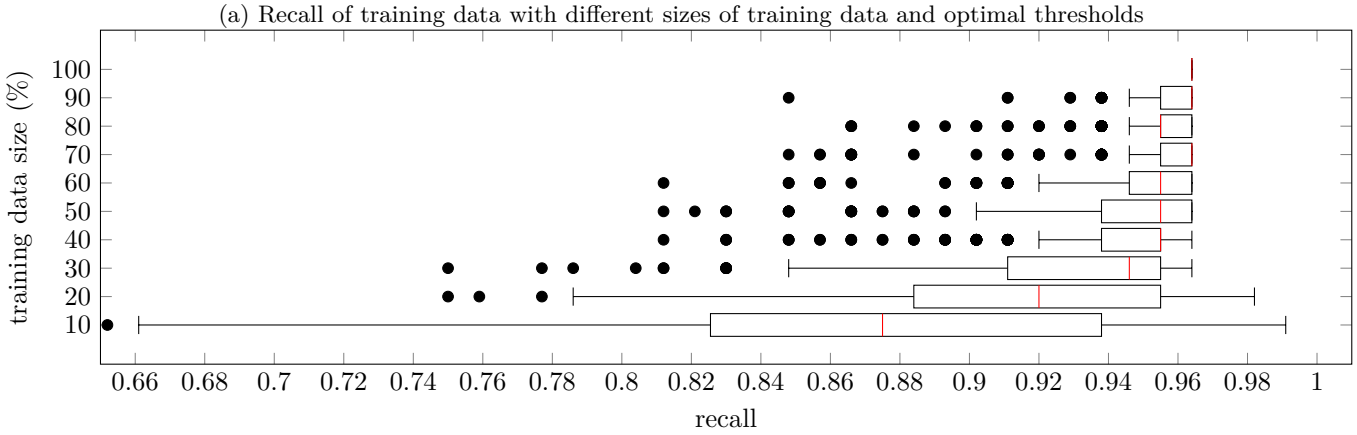
[3] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures", in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (Washington, D.C,), T. Senator, Ed., ACM Special Interest Group on Knowledge Discovery in Data and ACM Special Interest Group on Management of Data, New York, NY: ACM, 2003, p. 39, ISBN: 1581137370. DOI: 10.1145/956750.956759.

[4] W. W. Cohen, "Integration of heterogeneous databases without common domains using queries based on textual similarity", *ACM SIGMOD Record*, vol. 27, no. 2, pp. 201–212, 1998, ISSN: 01635808. DOI: 10.1145/276305.276323.

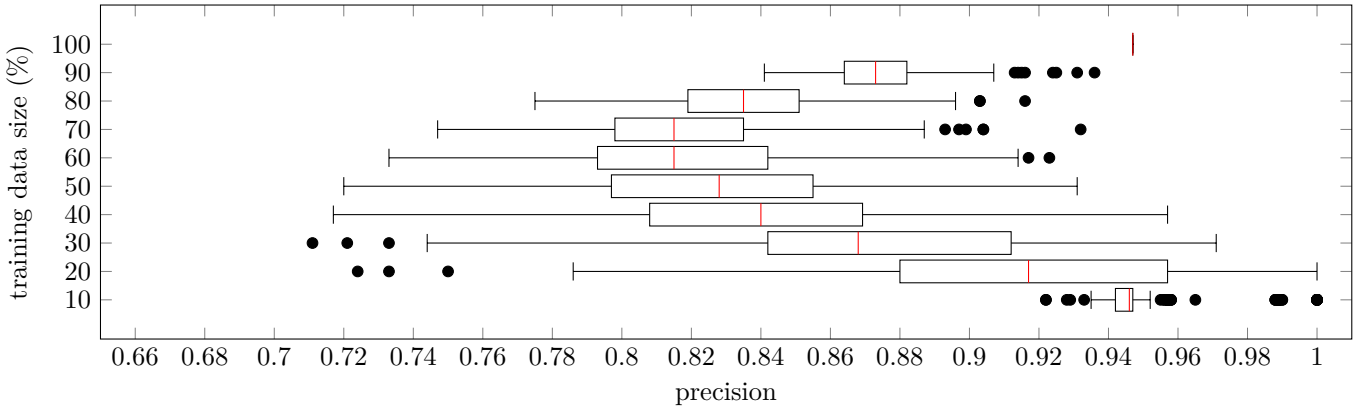
[4]More information on py_stringmatching can be found at https://sites.google.com/site/anhaidgroup/projects/magellan/py_stringmatching



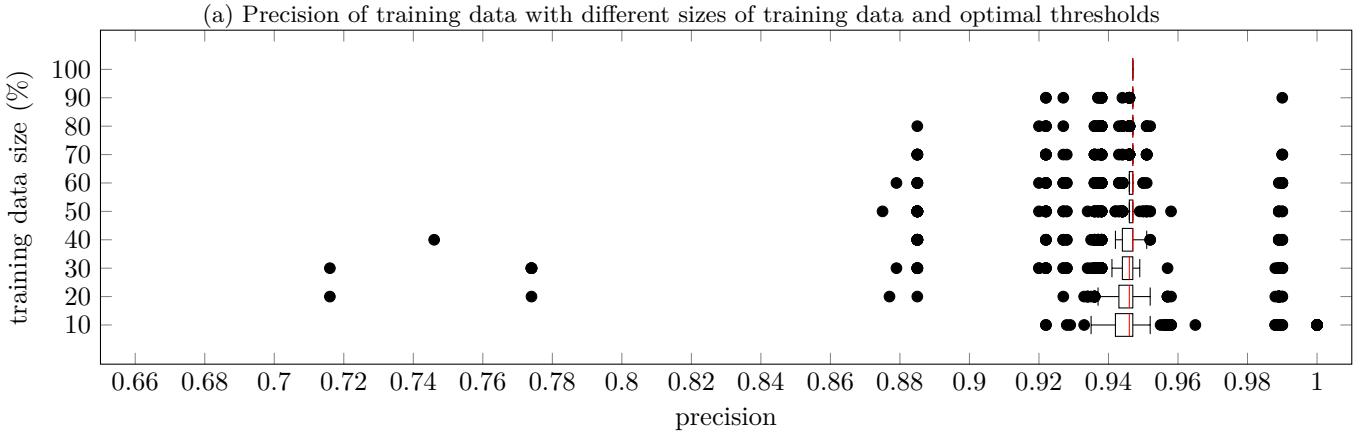
(a) Recall of training data with different sizes of training data and optimal thresholds



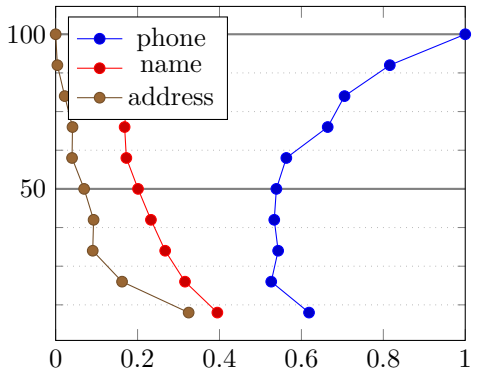
(b) Recall of test data with different sizes of training data and optimal thresholds



(a) Precision of training data with different sizes of training data and optimal thresholds



(b) Precision of test data with different sizes of training data and optimal thresholds


100
50
0
0.2
0.4
0.6
0.8
1
phone
name
address