Fast Entity Resolution With Mock Labels and Sorted Integer Sets

SIGMOD 2020 Contest Finalist Paper

Mark Blacher
Friedrich Schiller University Jena
Germany
mark.blacher@uni-jena.de

Julien Klaus
Friedrich Schiller University Jena
Germany
julien.klaus@uni-jena.de

Matthias Mitterreiter
Friedrich Schiller University Jena
Germany
matthias.mitterreiter@uni-jena.de

Joachim Giesen
Friedrich Schiller University Jena
Germany
joachim.giesen@uni-jena.de

Sören Laue
Friedrich Schiller University Jena
Germany
soeren.laue@uni-jena.de

ABSTRACT

Entity resolution is the task of identifying equivalent real-world entities across different sources. We found that standard methods based on unsupervised learning are slow during the resolution step. Thus, we propose an alternative approach with mock labels. Mock labels are sets of strings that ideally represent real-world entities. Using a running example, we show how to create mock labels from the training data and use them in the resolution step to match real-world entities efficiently. We applied this approach in the SIGMOD 2020 Programming Contest and achieved significantly faster execution times in the resolution step compared to other top ranked teams. Since all the top ranked teams had the same F-measure, the execution speed of the resolution step was decisive.

CCS CONCEPTS

• **Information systems → Entity resolution**.

KEYWORDS

entity resolution, data cleansing, programming contest

1 INTRODUCTION

Entity resolution is a problem that occurs in data integration, data cleansing, web search, and e-commerce search scenarios [4]. Given two or more sources, we need to identify real-world entities that do not have unique identifiers that tell us which records from one source match those in the other sources. In addition, records representing the same entity may contain different information. For example, if we use digital cameras as real-world entities, one record may have the brand or the model misspelled, and another may be missing some fields such as pixel count or zoom factor. Furthermore, it is possible that information within individual records is contradictory, such as an incorrect pixel count for a particular camera model. An entity resolution algorithm has to deal with these inconsistencies in the data and identify records corresponding to the same real-world entity as best as possible [2].

1.1 Background

There are basically two types of entity resolution, namely Pair-wise Entity Resolution, and Group-Wise Entity Resolution.

Definition 1.1 (Pair-wise Entity Resolution). Given a set of objects S, the output is a set of pairs $\{(o_i, o_j) \mid o_i, o_j \in S, o_i$ and o_j refer to the same real-world entity$\}$.

Definition 1.2 (Group-wise Entity Resolution). Given a set of objects S, the output is a family of sets $\{S_i \mid$ all objects in S_i refer to the same real-world entity$\}$ where $S_i \cap S_j = \emptyset$ if $i \neq j$.

Pairwise Entity Resolution approaches often use similarity or distance functions for determining whether two objects refer to the same real-world entity. Group-wise Entity Resolution approaches additionally use clustering techniques for assigning an object to a group or cluster. The results of these two types of entity resolution are not always consistent [4].

1.2 SIGMOD 2020 Programming Contest

The challenge of the SIGMOD 2020 Programming Contest was to design an entity resolution system for cameras. The input data consisted of about 30,000 e-commerce websites. Each website was provided as a JSON formatted file that specified a real-world product offer. Not every website contained a single valid camera. Camera accessories such as bags or zoom lenses, TVs, and entire camera bundles were also part of the dataset. Also, the product descriptions in the JSON files were inconsistent (varying attribute names, different semantics and syntax). Only the page title attribute was always present. See Figure 1 for an example of a JSON formatted website that represents a specific camera.

```
{
  "<page title>": "Canon PowerShot SX 500IS | eBay",
  "brand": "Canon",
  "manufacturer warranty": "No",
  "megapixels": "16.0 MP",
  "model": "SX500IS",
  "optical zoom": "30x"
}
```

Figure 1: Example specification for a camera in JSON format. The page title exists on all websites, whereas the brand and model attributes are only specified on eBay websites.

The task was to find as many identical camera pairs as possible among all e-commerce websites (Pair-wise Entity Resolution). A valid submission consisted of a CSV file containing only the matching camera pairs found during the resolution step. The Submissions were ranked based on the F-measure [3], which was computed on a secret evaluation dataset. The five finalist teams all achieved an F-measure of 0.99. Because of this tie, the running time of the resolution step was decisive [1].

1.3 Outline

The rest of this paper is organized as follows. Throughout the paper we use a running example. In Section 2 we describe the training step of our approach. We show that by preprocessing the training data and using additional information about synonymous camera names from the web, it is possible to create mock camera labels that roughly represent the real-world camera entities. In Section 3 we describe our resolution step, that is, how we match the contents of the JSON files with the generated mock labels. To speed up the execution time, we use sorted integer sets for the mock labels and the real-world camera descriptions. Furthermore, we give recommendations on how to speed up the cleansing of input data. In Section 4 we evaluate our solution. Finally, in Section 5 we conclude this paper with a discussion of the features and limitations of our approach.

2 TRAINING

Our approach is based on a concept we call *mock labels*. We call them mock labels because they are not real identifiers, but only sets of semi-automatically generated keywords. They may or may not represent a real-world entity. For the sake of simplicity, we also refer to mock labels in the text as *labels*. For creating mock camera labels, we use clean websites from the training data which give rise to about 950 labels. These labels are simply extracted from the corresponding page titles by using stop words. We also use information about synonymous camera names from Wikipedia which creates another 50 labels. Additionally, we combine the attributes *brand* and *model* from provided eBay websites to create roughly another 1800 labels. In Figure 2 we summarize our approach by constructing mock camera labels for the running example.

3 RESOLUTION

In the training step described in the previous section we generated mock camera labels. In this section we describe the resolution step of our implementation, that is, how we match these labels with textual descriptions of real-world cameras. Our resolution step is subdivided into four parts:

- First, we convert the mock camera labels to an efficient internal representation (Section 3.1).
- Second, we create simplified mock labels that allow us to match even incomplete camera descriptions (Section 3.2).
- Third, we extract camera descriptions from the JSON files and preprocess them efficiently (Section 3.3).
- Finally, we match the camera mock labels with the extracted camera descriptions (Section 3.4).

3.1 Internal Representation of Mock Labels

Internally, we represent a mock label as a sorted integer set. Representing mock labels as sorted sequences of integers enables us to compute matchings in Section 3.4 with real-world camera descriptions faster than it would be possible using only strings. In Figure 3 we convert the mock labels from Figure 2 to sorted integer sets.

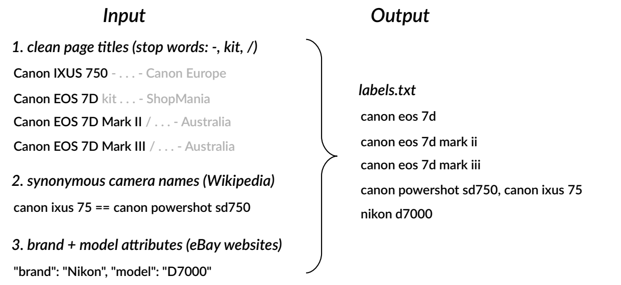
Input
Output
1. clean page titles (stop words: -, kit, /)
Canon IXUS 750 - . . . - Canon Europe
Canon EOS 7D kit . . . - ShopMania
Canon EOS 7D Mark II / . . . - Australia
Canon EOS 7D Mark III / . . . - Australia
2. synonymous camera names (Wikipedia)
canon ixus 75 == canon powershot sd750
3. brand + model attributes (eBay websites)
"brand": "Nikon", "model": "D7000"
labels.txt
canon eos 7d
canon eos 7d mark ii
canon eos 7d mark iii
canon powershot sd750, canon ixus 75
nikon d7000

Figure 2: Semi-automatic generation of mock camera labels. We use three sources to generate the labels. 1. The page titles of websites like *canon-europe*, *shopmania* or *shopbot* are very clean. We extract labels from these websites by using stop words. 2. Camera names may differ between countries. We download synonymous camera names from Wikipedia. 3. We combine the *brand* and *model* attributes of eBay websites from the training data to create additional labels. The merged results of the three sources are saved to a text file (labels.txt). Each line in labels.txt contains one or more comma separated sets of strings. Ideally, each line represents a real-world camera entity.

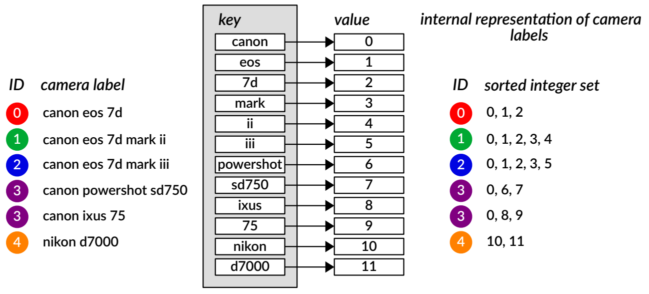

Figure 3: Conversion of mock camera labels to sorted integer sets. We map each unique token (key) in camera labels to a unique value. Based on these key-value-mappings, we convert camera labels to sorted integer sets. A camera can have different names in different countries. Therefore, repeating IDs reference the same cameras (see, for example, ID=3).

3.2 Simplified Labels to Handle Missing Tokens

It is not always possible to create a match with a camera label because some tokens may be missing in real-world camera descriptions. We consider tokens that do not represent the camera brand and alphanumeric model information as non-critical for the resolution step. Therefore, we create a second set of camera labels without these specific tokens (see Figure 4). We use the simplified camera labels in cases where the original camera labels fail to create a match.

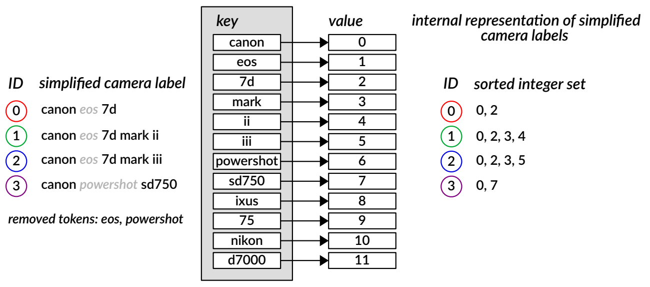

Figure 4: Simplification of camera labels by removing certain tokens. The (greyed out) tokens *eos* and *powershot* are removed from the camera labels. We include only camera labels that are affected by the removal of tokens in the simplified camera labels set. Note that the cameras *canon ixus 75* and *nikon d7000* are not included in the simplified camera labels set as they contain no tokens that can be removed. Note also that we use the key-value-mappings of the original camera labels to convert the simplified camera labels into an internal representation.

3.3 Efficient Preprocessing of Input Data

Reading the input data from SSD and cleansing takes up a considerable part of the overall running time of the resolution step. The following findings are important to speed up preprocessing of the input data:

- Reading many small files concurrently, with multiple threads (compared to a single thread), takes advantage of the internal parallelism of SSDs and thus leads to higher throughput [5].
- C-string manipulation functions are often significantly faster than their C++ pendants. For example, locating substrings with `strstr` is around five times faster than using the C++ `std::string` function `find`.
- Hardcoding regular expressions with *while, for, switch* or *if-else* statements results in faster execution times than using standard RegEx libraries, where regular expressions are compiled at runtime into state machines.
- Changing strings in place, instead of treating them as immutable objects, eliminates allocation and copying overhead.

To achieve fast preprocessing of the input data, we use 16 threads to read the JSON files from SSD and clean the page titles. We perform string operations in-place with C library functions, and hardcode our regex expressions. The result of our preprocessing step are sorted integer sets that represent the page titles (see Figure 5).

3.4 Matching of Mock Labels

After converting the page titles into sorted integer sets, we try to assign unique camera IDs to them. First, we try to match labels from the original camera labels with the page title. If we do not find a match, we try to match the simplified labels with the page title. We consider labels as matches if all the corresponding keywords are also part of the title. If we can match more than one label with the title, then we remove all labels that are subsets of other labels. This allows us, for example, to distinguish between the cameras *canon eos 7d* and *canon eos 7d mark ii*, because the first camera description

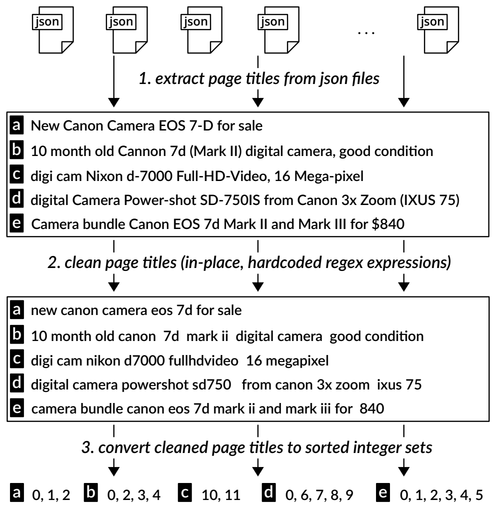
json
...
1. extract page titles from json files
a New Canon Camera EOS 7-D for sale
b 10 month old Cannon 7d (Mark II) digital camera, good condition
c digi cam Nixon d-7000 Full-HD-Video, 16 Mega-pixel
d digital Camera Power-shot SD-750IS from Canon 3x Zoom (IXUS 75)
e Camera bundle Canon EOS 7d Mark II and Mark III for $840
2. clean page titles (in-place, hardcoded regex expressions)
a new canon camera eos 7d for sale
b 10 month old canon 7d mark ii digital camera good condition
c digi cam nikon d7000 fullhdvideo 16 megapixel
d digital camera powershot sd750 from canon 3x zoom ixus 75
e camera bundle canon eos 7d mark ii and mark iii for 840
3. convert cleaned page titles to sorted integer sets
a 0, 1, 2
b 0, 2, 3, 4
c 10, 11
d 0, 6, 7, 8, 9
e 0, 1, 2, 3, 4, 5

Figure 5: Conversion of page titles to sorted integer sets. We carry out all processing steps in parallel with 16 threads. First, we extract the page titles from the JSON websites. Next, we clean the page titles in-place with C-string library functions, and hardcoded regex expressions. Finally, we convert the cleaned page titles into sorted integer sets based on the key-value-mappings of the original camera labels.

is a subset of the second. If the remaining labels have the same camera ID, then we can clearly assign a camera ID to a page title. Figure 6 shows our approach for assigning unique camera IDs to page titles for the running example.

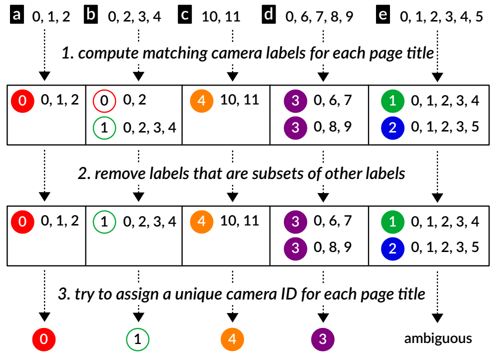

Figure 6: Assignment of camera IDs to page titles. First, we compute the matching camera labels for each page title. If all integer tokens of a label are within the title, we consider the label a match. Note that we use the simplified labels for the page title *b* because we cannot match a label from the original camera labels. Next, we remove labels that are subsets of other labels. If all remaining labels have the same camera ID, we assign the ID to the page title.

To avoid iterating over all camera labels when searching for matches with the page title, we store all camera labels in a sorted vector. We iterate only over relevant ranges in the vector. To look up the relevant ranges, we index the initial keywords of the labels. For even faster matching of camera labels with the page title, we recommend storing the camera labels in a sorted trie data structure, as shown in Figure 7. Finding all labels that match a page title, is then a matter of traversing the trie and comparing integer tokens. In real-world camera descriptions it can happen that the tokens of a label are scattered in the page title. The following traversing strategy finds these scattered labels in the page title: If there is no successor in a trie node for the current token in the page title, then ignore the token and continue with the next token in the page title.

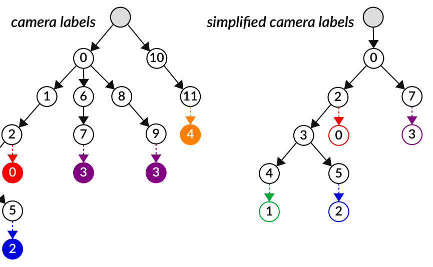

Figure 7: Storing the mock camera labels in a sorted trie.

4 EVALUATION

Our approach is related to Group-wise Entity Resolution. In the training step we semi-automatically construct mock labels that can be understood as centroids of clusters. Ideally, each label represents one real-world entity. In the contest solution we have also labels that represent several real-world entities, or no real-world entity at all. In the resolution step we try to assign each real-world camera description to one cluster with set operations on sorted integer sets. After assigning all camera descriptions to clusters, we compute the Cartesian product of the cameras in each cluster and store the unique pairs of identical cameras in a CSV file.

The quality of the results of our resolution step depends mainly on the labels generated in the training step. With the 2800 semi-automatically generated mock labels we achieved on the hidden evaluation dataset an F-measure of 0.97. To achieve an F-measure of 0.99, we manually added about 200 contest specific labels to the semi-automatically generated ones. We extracted these additional labels by inspecting page titles that did not match with any semi-automatically generated labels. Since all five best placed teams reached an F-measure of 0.99, the running time of the resolution step was decisive. We were chosen as the overall winner. Table 1 shows also the running times of the resolution step of the five best placed teams.

5 CONCLUSIONS

In this paper we presented our entity resolution approach, that we submitted in the SIGMOD 2020 Programming Contest. Our approach is based on mock labels that we generate in a training step and a fast matching algorithm that computes set intersections on sorted integer sets in the resolution step. Our matching algorithm knows nothing about the actual labels. The creation of labels in the training step and the matching of camera descriptions with the labels in the resolution step are decoupled. In contrast, rule-based approaches, in which the rules for clustering are hardcoded into the source code, are rigid and must be written from scratch for new entity resolution tasks. But, there are also limitations of our approach. For the matching we use set operations on integer tokens. One integer token represents one word. If entities can only be distinguished by word order, then our approach needs to be adapted. By allowing n-grams as tokens in the labels and not just individual words, it is possible to integrate word order semantics into our approach. Syntactic variations in the data can also be challenging in our resolution step. In the contest we solved this problem by preprocessing the input data by hardcoded regex expressions and in-place string operations. A more general approach would be to use distance metrics for deciding which tokens are present in real-world descriptions. Still, the biggest challenge remains, namely to generate representative labels from the data. If the training data is unstructured or incomplete, it may be impossible to extract representative labels for the resolution step. In the Programming Contest, the training data was fairly well-structured and complete, that is, the generation of mock labels was mainly a semi-automatic extraction task. This again shows that it is important to look at the data first before deciding on an approach.

Table 1: Comparison of the F-measure and the running times of the resolution step of the five best placed teams. The input data for the resolution step consisted of 29,787 in JSON formatted e-commerce websites. Measurements were taken on a laptop running Ubuntu 19.04 with 16 GB of RAM and two Intel Core i5-4310U CPUs. The underlying SSD was a 500 GB 860 EVO mSATA. We cleared the page cache, dentries, and inodes before each run to avoid reading the input data from RAM instead of the SSD.

Team	Language	F-measure	Running time (s)
PictureMe (**this paper**)	C++	0.99	**0.61**
DBGroup@UniMoRe	Python	0.99	10.65
DBGroup@SUSTech	C++	0.99	22.13
eats_shoots_and_leaves	Python	0.99	28.66
DBTHU	Python	0.99	63.21

ACKNOWLEDGMENTS

This work has been supported by the German Research Foundation (DFG) grant GI-711/5-1 within the Priority Program 1736 Algorithms for Big Data.

REFERENCES

[1] Database Research Group at Roma Tre University. 2020. *ACM SIGMOD Programming Contest 2020.* Retrieved June 24, 2020 from http://www.inf.uniroma3.it/db/sigmod2020contest/index.html
[2] Hector Garcia-Molina. 2004. Entity Resolution: Overview and Challenges. In *Lecture Notes in Computer Science.* Springer, 1–2. https://doi.org/10.1007/978-3-540-30464-7_1
[3] David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. 2010. Evaluating Entity Resolution Results. *Proceedings of the VLDB Endowment* 3, 1-2 (Sept. 2010), 208–219. https://doi.org/10.14778/1920841.1920871
[4] Hongzhi Wang. 2014. *Innovative Techniques and Applications of Entity Resolution.* IGI Global. https://doi.org/10.4018/978-1-4666-5198-2
[5] Zhenyun Zhuang, Sergiy Zhuk, Haricharan Ramachandra, and Badri Sridharan. 2016. Designing SSD-Friendly Applications for Better Application Performance and Higher IO Efficiency. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).* IEEE. https://doi.org/10.1109/compsac.2016.94