Antonopoulos Ilias
p3352004
iantonopoulos@aueb.gr

Contents

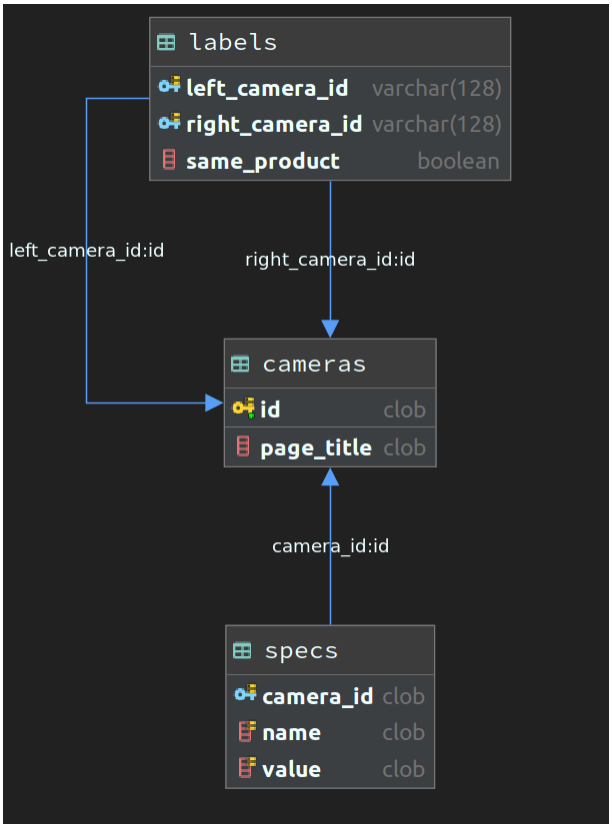
1 Data Loading & Database Schema

Let's begin by exploring the data.

By extracting the **camera_specs.tar.gz**, we can see the following structure:

```
buy.net/

      4233.json
      4236.json
      ...
      ...
      ...
      6785.json

cammarkt.com/

...
...
...

www.wexphotographic.com/
```

Given that each camera entity, apart from the page title information, can have any number of features (specs) available and that feature-overlap and consistency between any two products is far from the norm, we will utilize the following schema:



We will write 2 pythonic loaders (UDFs) that will create and populate the tables **cameras** (holding the primary identifier and the page title) and **specs** (holding a reference to the camera and the different camera specs with their corresponding values.
A point-of-interest here is that some of the JSON files contain an array of legitimate values for a particular key (e.g. *./www.camerafarm.com.au/705.json*). In that case, our specs loader function will deduplicate the array of values before attempting to generate all the specs-values mappings.

For these two tables, we will utilize the MonetDB command CREATE TABLE FROM LOADER.

Now, on with the **sigmod_medium_labelled_dataset.csv** data. This is our annotated data, in a CSV format. We will utilize the COPY INTO FROM MonetDB command to efficiently bulk-insert the CSV data into a new table called **labels**.

Having transferred our dataset to a MonetDB instance, we will additionally perform some ALTERations to our tables in order to add ensure referential integrity and thus come up with a working chema:

a) enforce a primary key on **cameras.id** column.

b) enforce a foreign key constraint to the **specs.camera_id** (referencing **cameras.id**)

c) enforce a unique constraint on **specs** table, for the columns (**camera_id**, **name**, **value**)

d) enforce a primary key on labels (**left_camera_id**, **right_camera_id**) combination.

e) enforce a foreign key constraint to the **labels.left_camera_id** (referencing **cameras.id**)

f) enforce a foreign key constraint to the **labels.right_camera_id** (referencing **cameras.id**)

Now that we have a working database schema, we can proceed with some data exploration to build an understanging towards the entity resolution task.

2 Exploratory Data Analysis

We have **29787** products (i.e. cameras), and an astonishing **4661**(4660 + the page title) different specs across these products. This unusually high number of specs stems from the fact that the JSON files were essentially unstructured information, with little overlap, even between products of the same retailer.

We are also equipped with **46665** product pairs (out of the pool of all possible product pairs that can be formulated from our dataset), for which we know the ground truth i.e. if they are the same or not.

Table 1: select same_product, count(*) as count from labels group by same_product;

same_product?	count
1	3582
0	43083

As we've expected - given the fact that the label corresponds to a product pair - the problem is imbalanced. We have, comparatively, very few products that are "matching".

It is illuminating to calculate the support for each spec (i.e. the ratio of non-null values for this spec). We will include <page title> in the analysis, as the most prominent camera spec, even though we chose to include it in the cameras table and not in the specs table:

Table 2: top 10 specs, based on support (rounded to 2 decimal digits)

spec	support
<page title>	1
brand	0.53
model	0.5
megapixels	0.46
type	0.46
screen size	0.41
optical zoom	0.39
mpn	0.35
condition	0.33
upc	0.26

By taking a closer look at these 10 specs we can suspect that they contain significant discriminative ability and will most probably be of vital importance to our machine learning component.
It is also highly informative to get the number of specs with support below a given threshold:

Table 3: no. of specs with support below x%

# specs	support threshold
4638	10%
4600	5%
4502	2%
4383	1%
3527	0.1%
62	0.01%

This is in line with our initial impression of the data being very much unstructured: for example, 4383 out of the total 4662 specs have less than 1% support!

3 Blocking

We would like to block together cameras of the same brand. This will help us restrict the ”potential matching” space, as cameras that belong to different blocks should not, in principle, refer to the same camera!

To implement the blocking step, we will introduce a **brand_id** to our **cameras** table, that is going to be a reference to a new **brands** table. We will seed the **brands** table with a curated set of common camera brands (Canon, Kodak etc.) that are indirectly available to our dataset. All we have to do then is develop a MonetDB/Python UDF that, given a text, extracts the camera brand based on our curated selection. We can now populate the **brand_id** attribute according to the return values of this UDF, when applied in the <page title> camera attribute.

brand extracted ?	count
yes	25032
no	4755

So, we have managed to accurately retrieve the brand from <page title> on 25032 out of our total 29787 cameras. We will deal with the rest, 4755, later.

Table 4: top 10 blocks, based on size

brand	size
canon	5474
nikon	4705
sony	3214
hikvision	2002
fujifilm	1505
olympus	1435
panasonic	1304
samsung	1064
kodak	750
ricoh	709

Table 5: bottom 5 blocks, based on size

brand	size
garmin	7
epson	6
wespro	4
contour	4
motorola	3

In total, we have formed **47** blocks.

Before continuing, we should note that a number of pre-processing actions were taken in order to normalize the <page title> attribute, pre-blocking, such as:

- converted to lowercase
- sanitization: punctuation removal, dashes, etc.
- fixing common brand aliases (e.g. cannon → canon)

4 Filtering

After block formation, we would also like to filter out ”easy” matches, by extracting-via-heuristics and comparing the models (D3200, 600D, 2032i etc.) of the cameras. We would thus end up - for each block - with a subset of cameras that match both on brand and model (and basically refer to the same camera).

To implement the filtering step, we will have to develop a MonetDB/Python UDF that, given a text, extracts the camera model based on heuristics. We will use regular expressions to capture the **key observation** that almost all camera models are numbers that are followed or preceded by a selection of letters.

We will, in the same fashion as before, base the model extraction on the <page title> attribute alone. For now, we basically want to keep things simple and reach a baseline filtering performance that allows a fairly limited amount of false positives. Equipped with this baseline, we can later return to the filtering process and possibly enhance it by utilizing the available information outside <page title>.

From this point on, we will make use of the term **signature** and say that a camera has a signature if and only if it has a brand extracted (i.e. has been classified to a block) and also a model extracted.

So, by the end of the filtering step, we expect that a subset of our cameras will each be equipped with a signature (and thus constitute ”solved” cases, as we are aware of information sufficient for matching).

found signature ?	count
yes	21181
no	8606

Table 6: top 10 signatures, based on size (i.e. how many cameras share the same signature)

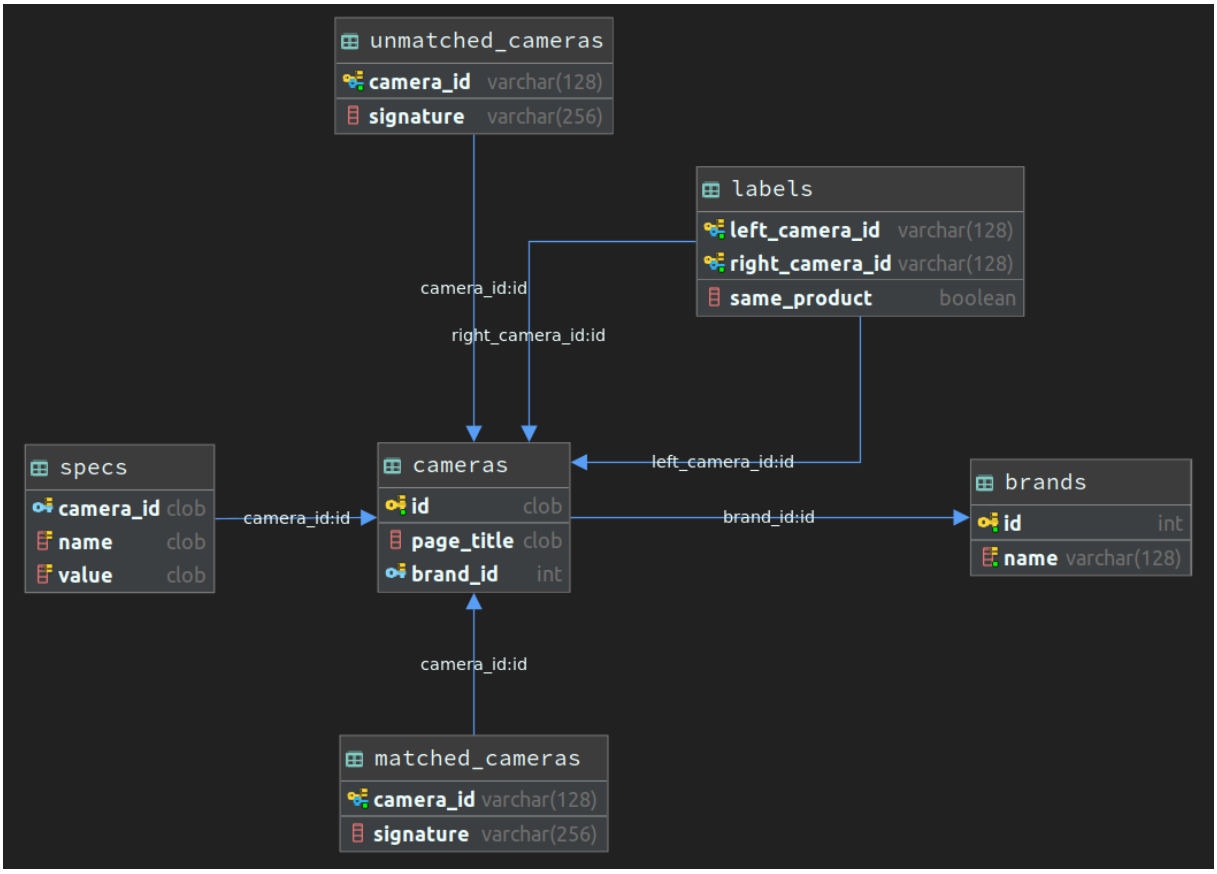
signature	size
canon_5d	247
canon_7d	209
canon_3i	191
hikvision_i5	189
nikon_d3200	182
canon_60d	169
hikvision_ip66	167
canon_t3	166
nikon_d3100	155
hikvision_3332i	152

In total, we have managed to extract **3204** distinct signatures, across all blocks. It is informative to also assess how many distinct signatures were extracted per block.

Table 7: top 10 blocks, based on number of distinct signatures

brand	size
sony	527
canon	382
panasonic	295
fujifilm	279
nikon	273
samsung	227
olympus	199
kodak	179
ge	154
casio	114

And this is the database schema, at the end of filtering.



We have introduced 2 new proxy tables that host matched and unmatched cameras after the blocking & filtering process. We could, however, simply introduce a column **cameras.signature** and we would still be able to differentiate ”unmatched” and ”matched” cameras by checking for the presence of signature (null or not).

We will wait and see how the project will evolve at the later phases and reconsider accordingly.