Data Science Course project: ACM Sigmod 2020*

ACM Sigmod 2020 competition
Primary Topic: DM, Secondary Topic: NLP
Course: 2019-2A – Group: 102 – Submission Date: 2020-04-19

Boris Tseitlin
University of Twente
b.tseitlin@student.utwente.nl

Anton Broilovskiy
Higher School of Economics
Anton.Broilovskiy@yandex.ru

ABSTRACT

In this paper, we describe our approach to the Entity Resolution task of the ACM Sigmod 2020 programming competition. The challenge was to develop a system for accurately matching the descriptions of the same digital cameras. We describe our model in detail, the achieved results and finish with a discussion on approaches that didn't work.

1 INTRODUCTION

ACM SIGMOD is an annual programming contest. The subject of the contest of 2020 was to construct an Entity Resolution system. Entity Resolution is the problem of identifying and matching different manifestations of the same real-world object in a dataset. In literature it is also called Record Linkage, Deduplication, Approximate Match, Entity Clustering. [1].

In the task, the participants were given specifications of digital cameras (e.g. Canon EOS 5D Mark II) from multiple e-commerce websites. The challenge was to develop a system for matching the specifications of the same camera models with high precision and recall.

We approached the task a binary classification problem, concluded extensive preprocessing of source text data, created a cross-validation pipeline that imitates the evaluation process of the contest, used a majority voting ensemble of classifiers. In the end, we achieved an F1-score of 0.9, which put is in the top 20 teams on the leaderboard.

2 BACKGROUND AND RELATED WORK

2.1 Related competitions

Quora question pairs [4] was a Kaggle competition, where the task was to find near-duplicates among questions in natural language. It had a couple of differences from the ACM SIGMOD 2020 competition. First, the task was a regression one, of predicting probabilities of duplicates. Second, it contained questions in natural language, whilst in ACM SIGMOD 2020 the camera descriptions were not. Methods from this competition were not directly applicable to our task but were useful for inspiration. Common solutions included extracting pairwise features (e.g. number of common neighbors) and using siamese BiLSTM neural networks. Ensembles of gradient boosted trees were also common among solutions.

Avito Duplicate Ads Detection [4] was another similar Kaggle competition. The task was to find duplicates among classified postings using the texts and pictures. The main focus of most solutions to this competition was on using image data, so the approaches were not applicable to our task.

3 APPROACH

We aimed to develop a model that, given the features obtained from a pair of specifications, would output 1 if those specifications described the same camera model, and 0 otherwise.

The biggest obstacle with this approach was finding a way to achieve feasible computation times. As the number of specifications was rather large (see 4.1), comparing each two of them using a binary classifier was very time-consuming. In our approach, we used brands extracted from specification texts to decrease the number of pairs that needed to be checked.

The solution consisted of the following major blocks:

(1) Extracting text data from JSON's,
(2) Preprocessing specification text data,
(3) Establishing an evaluation scheme using brands,
(4) Obtaining features for each pair of specifications,
(5) Training a classifier model.

3.1 Data extraction

We converted source data to a table with the following columns:

- *spec_id* — unique specification id, contains website where it was posted,
- *page_title* — text of web page title, the only field that's guaranteed to be present in any specification,
- *brand* — text extracted from field "brand" of JSON specification, if it's present, and empty otherwise,
- *model* — same as brand, but for field "model",
- *type* — same as brand, but for field "type",
- *megapixels* — same as brand, but for field "megapixels",
- *all_text* — values of all attributes of the JSON specification, extracted by recursively traversing the JSON.

All fields were converted to lowercase and stripped of duplicate and trailing spaces. Only alphanumeric characters and dashes were kept.

One of the challenges of the dataset was that fields "brand", "model", "type" and "megapixels" could contain not only single-word strings but also sentences, lists, and objects. To counter that we used a special extraction scheme for these fields. Every time a single-word string brand was encountered, it was added to a set of known brands. If a list or object brand was encountered, it was scanned for a known brand using the set of known brands. If no known brand was found, the first word was used as the brand. Same for field "model".

*

Other preprocessing applied to fields ”brand”, ”model”, ”type” and ”megapixels” included:

- *brand*: replace strings of length less than 3 with NULL.
- *model*: if the string does not contain both numbers and letters, or has a length less than 3, replace it with NULL,
- *type*: convert to a few major categories using rules, e.g. replace ”digital camera”, ”digital”, ”digital slr” with ”dslr”,
- *megapixels*: keep only numbers and dashes while replacing all separators (spaces, dots, commas) with dashes so that all ways to describe megapixels of a camera were converted to the same format.

3.2 Preprocessing

After raw data had been obtained, extensive preprocessing was required to make it useful. Our pipeline consisted of the following steps.

Preprocessing text fields. This included filtering all words with document frequency less than 5, filtering stopwords using NLTK [7], fixing known common typos (e.g. replace ”cannon” with ”canon”), keeping only the first 500 words of each text field value to avoid rare extra-long sequences, standardizing megapixels description (e.g. replace both ”megapixel” and ”megapixels” with ”mp”).

Stemming. We applied stemming to *page_title* and *all_text*.

Populating brands using all_text. Only a small fraction of specifications had a filled ”brand” field, however, it was common for page titles to contain the brand of the camera. We used the set of already-known brands and a list of known digital camera manufacturers to search for brands in the *all_text* strings. After this manipulation, the number of specifications without a brand was greatly reduced. Brands with a total frequency of less than 15 were replaced with NULL to avoid errors.

Populating models and types using all_text. We applied the same approach to *model* amd *all_text* fields.

Populating brands using models. Models are unique to their brand. E.g. an ”EOS 5D” camera is definitely a Canon camera. Using this knowledge it was possible to create a mapping from each camera model to the most frequent brand for this camera model and apply this mapping to fill in the brands of specifications, where a model was present, but a brand was not.

Dropping bad specifications. The dataset included not only digital cameras but also digital accessories, CCTV cameras, and other items. However, finding duplicates among them was not part of the task, so they could only hinder the performance of the model. The last step of preprocessing was dropping such specifications. We dropped specifications if they matched any of the following:

- *type* is not a digital camera and contains a known CCTV brand (e.g. ”hikvision”) in *all_text*, or any of the other words found in CCTV camera descriptions (e.g. ”security”, ”dome”, ”ip camera”).
- Contains words commonly found in camera bag or case descriptions and no *brand*, *type* or *megapixels* were identified.
- Known non-camera brand identified (e.g. ”neopine”, which is a camera bag producer).

Dropping bad specifications not only improved the performance of the model but also made computation times considerably smaller.

3.3 Establishing an evaluation scheme using brands

Brands of specifications allowed us to greatly reduce the number of pairs that needed to be evaluated. A Canon camera can not be the same entity as a Nikon camera. Following this intuition, we established the following evaluation scheme: for each brand, we would run our model on pairs of specifications of this brand. This allowed for feasible computation times: under an hour to generate submission. Of course, the challenge was extracting brands really well. Under that scheme, if a specification was mislabeled with a brand, it would never be compared to any specifications that describe the same entity.

3.4 Obtaining features for each pair of specifications

It was important to make the features general because the labeled dataset did not include all possible cameras of the whole dataset. The model must perform well when given pairs of camera specifications for camera models that it had never encountered in training.

We used 21 features. Each feature was computed from a pair of specifications.

- n_common_tokens — number of common tokens (words) between specification page titles.
- n_common_tokens_normed — same as before, but normed by the total amount of tokens in page titles.
- sum_len_common_tokens — sum of lengths of common tokens,
- special_n_common_tokens — number of common tokens, such that these tokens include both numbers and letters. Under the hypotheses, that such tokens are more important than others: this is how camera models are usually described.
- special_sum_len_common_tokens,
- number_n_common_tokens — number of common numbers-only tokens,
- number_sum_len_common_tokens,
- n_common_symbols_models — length of common subsequence between model fields of specifications, starting from the beginning of strings,
- same_model — binary feature, true if model fields of specifications are the same,
- same_n_common_symbols_types — same as n_common_symbols_models, but for the type field.
- same_type,
- same_n_common_symbols_megapixels — same as n_common_symbols_models, but for the megapixels field.
- same_megapixels,
- lev_ratio — normed Levenstein distance between specification page titles,
- cosine_sim_tfidf — cosine similarity between TF-IDF vectors of page titles,
- jaccard_sim — Jaccard similarity between sets of tokens of page titles,
- sum_len_common_tokens_all_text — same as sum_len_common_tokens, but computed from the all_text field,
- n_common_tokens_all_text,

- special_n_common_tokens_all_text,
- special_n_common_tokens_all_text_normed,
- same_site — true if both specifications belong to the same e-commerce site.

To minimize computation times we precomputed as much as we could before the actual feature extraction. For example, TF-IDF vectors could be computed before-hand once, as well as vector norms used in the computation of cosine similarities.

3.5 Training a classifier model

A majority voting classifier ensemble was used in our final approach, which struck a balance between speed and good precision. Even though the number of features was small, using such a basic ensemble provided a boost in precision. The core of the ensemble constituted of LightGBM [6] estimators with different parameters. LightGBM is an extremely fast and powerful implementation of Gradient Boosted Trees. Our final ensemble included:

- LightGBM with default parameters,
- LightGBM with 500 trees,
- LightGBM with default parameters and a learning rate of 0.01
- LightGBM with 500 trees and a learning rate of 0.01,
- LogisticRegression,
- Gaussian NB classifier,
- MLP classifier.

3.6 Cross-validation

Our cross-validation scheme deserves a special mention, as it was quite specific to the task at hand. Due to how the dataset is structured (see 4.1), one could not simply use the standard methods like K-fold cross-validation on the labeled dataset. The labeled dataset did not include all the possible duplicate and non-duplicate pairs. The labeled dataset did not include all possible camera models, so such a scheme would not indicate how well the model generalizes to unseen camera models. Also, this approach would not properly test the choice of the evaluation scheme (see 3.3).

The cross-validation scheme we came up with imitated the evaluation scheme of the contest (see 4.2) as close as possible.

The algorithm:

(1) Select a subset of **camera models** (not to be confused with pairs) from the labeled dataset and use pairs describing these models as the hold-out set. Use the rest of the labeled pairs as the training set. We picked 5 random camera models from those present in the labeled dataset.
(2) Train the model using the training set.
(3) Evaluate the model on **all specifications** and produce labeling of all specifications.
(4) Compute precision, recall, and F1-score by evaluating only the pairs included in the hold-out set.

The biggest distinction from the usual K-Fold approach is that on each fold the model produces labeling for **all specifications**, not only those included in the hold-out set. This approach mimics the way submission is done: the task is to produce labeling for all specifications, and which of those are included in the hidden evaluation dataset is not known upfront.

We found scores produced by such a cross-validation scheme positively correlated with the leaderboard scores. The downside of this approach was that it took as much time as submitting: at least an hour per fold. Due to this reason, we used only 3 folds, which produced a high variance in scores.

4 EXPERIMENTS

4.1 Dataset

The dataset [2] included 29 787 specifications in JSON format, collected across 24 different e-commerce websites. Each specification was stored in a file, and each file was in a directory corresponding to a particular e-commerce website (e.g. www.ebay.com).

Example specification:

```
{
  "<page title>": "Samsung Smart WB50F Digital Camera White
                   Price in India with Offers &
                   Full Specifications | PriceDekho.com",
  "brand": "Samsung",
  "dimension": "101 x 68 x 27.1 mm",
  "display": "LCD 3 Inches",
  "pixels": "Optical Sensor Resolution (in MegaPixel)\n16.2 MP"
  "battery": "Li-Ion"
}
```

A dataset of labels was provided.

It contained three columns: "left_spec_id", "right_spec_id", "label". It included a total of 297 651 pairs, out of which 44 039 were labeled as the same entity.

The goal was to find all pairs of specifications that described the same products, among the specifications provided.

4.2 Contest evaluation

Submitted solutions were ranked on the basis of F-measure, also called F1-score.

A secret evaluation dataset was used to compute the score. It contained a subset of camera models and a subset of specifications of these camera models. The score was evaluated only on those pairs that were included in the evaluation dataset. This dataset was disjoint from the provided labeled dataset so that no camera specification included in the labeled dataset was present in the evaluation dataset.

4.3 Challanges of the dataset

Large amount of possible pairs. While almost 30 thousand specifications is not a large dataset by itself, if one would try to compare each two of them, it would result in 4 432 753 pairs to compare. To put it in perspective, at 100 comparisons per second it would take approximately 12 hours. It's obvious that not all pairs have to be compared. It was possible (and practically necessary) to select a subset of pairs for comparison, which is why we came up with a scheme described in 3.3.

Long computation times. Because of the large number of possible pairs and the dataset evaluation scheme, each evaluation took a lot of time. Each run of the cross-validation pipeline would take a lot of time. We have not found a way to optimize the hyperparameters of our models: a grid search would take too much time. The organizers also prohibited the use of GPUs.

"Messy" data. Of all fields that JSON specifications could have, only "<page title>" was guaranteed to be present. Every field could

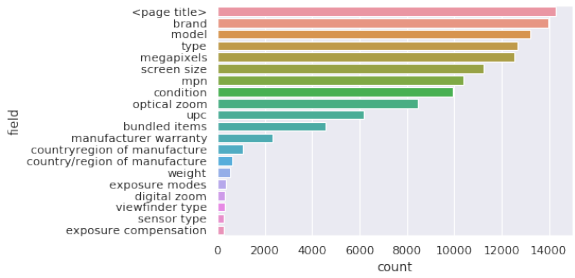

Figure 1: Non-empty field count in specifications.

be empty, have any kind of data type and any kind of format. JSON schemas of specifications of different websites were radically different. Figure 1 shows the total counts of non-empty field values for each field in the dataset.

Few labels. Only around 300 thousand pairs were labeled of the 4.5 million total possible pairs.

4.4 Results

With our best approach, described in this article we obtained an F1-score of **0.9**, which put us in top-20 on the contest leaderboard. This result was in-line with our cross-validation results.

5 DISCUSSION

5.1 Experiments with triplet loss and LSH

Our initial attempt at reducing the number of pairs to compare was the following. First, produce a latent space embedding of specification page titles, such that specifications describing the same real-world product would appear close in that latent space, and others would be far. Then, build an index on these embeddings, so that retrieving the closest neighbors of a specification is fast. Finally, for each embedding, retrieve the closest neighbors, use the classifier model on the pairs $(embedding, neighbor)$. This would ensure that only similar specifications are evaluated.

We used a triplet loss neural network [5] to produce the embeddings and Annoy locally sensitive hashing [3] library to build the index.

However, recall achieved by this model was close to zero. Embeddings produced for the labeled instances were decently separated in latent space, but the model performed poorly on previously unseen data, as seen in Figure 2.

Our hypothesis is that the labeled dataset did not contain enough information to produce a decent embedding for the whole dataset.

5.2 Experiments with graph-based features

At its core "describes the same camera model" is a transitive relationship on camera specifications. We thought hard about a way to provide information about transitivity to our classifier. One hypothesis was to use graph-based features. First, make a graph based on some distance metric between specification page titles. Let each specification be a vertex of the graph. If the distance between two specifications is below some threshold, produce an edge between

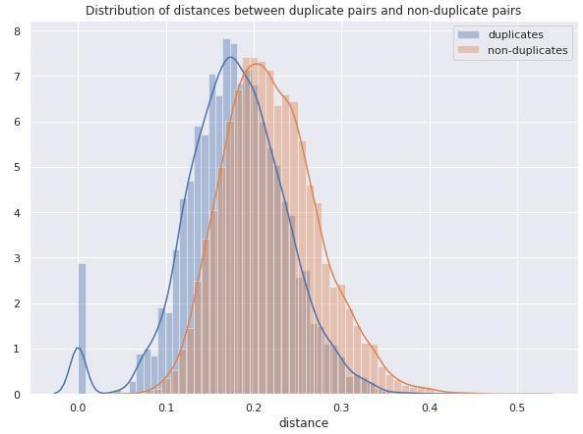

Figure 2: Distributions of distances between pairs describing the same camera (duplicates) and pairs describing different cameras (non-duplicates)

their corresponding vertices. Then pairwise features like the number of common neighbors could be computed for each pair of specifications.

We attempted to create such a graph based on Levenstein ratio with a threshold of 0.9. For reasons unknown to us, the features obtained from this graph did not provide any improvement to our model.

6 CONCLUSIONS

This project was a challenging task. It required us to build a model that is both fast and performant. Using preprocessing techniques, a way to reduce the subset of compared pairs to specifications of the same brand and an ensemble of classifiers we managed to achieve a decent result. Our best solution was ranked among the top 20 in the contest.

REFERENCES

[1] 2020. *ACM SIGMOD 2020 Programming Contest.* http://www.inf.uniroma3.it/db/sigmod2020contest/index.html
[2] 2020. *ACM SIGMOD 2020 Programming Contest task description.* http://www.inf.uniroma3.it/db/sigmod2020contest/task.html
[3] 2020. ANNOY library. https://github.com/spotify/annoy. Accessed: 2020-04-19.
[4] 2020. *Quora Question Pairs.* https://www.kaggle.com/c/quora-question-pairs
[5] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*. Springer, 84–92.
[6] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.
[7] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1 (ETMTNLP '02)*. Association for Computational Linguistics, USA, 63–70. https://doi.org/10.3115/1118108.1118117