LSH-based Pre-Selection for Duplicate Detection in Online Products through Logistic Regression

Jonathan Rietveld

Erasmus University Rotterdam
Rotterdam, The Netherlands
666788jr@student.eur.nl

Abstract. As technology enables retailers to ever more easily sell their products through webshops, consumers are faced with an ever increasing number of choices online. As such, aggregating websites which list the same product for comparison is an ever more relevant pursuit. However, as the number of products listed grows, so does the computational cost of such duplicate detection. This paper proposes the combined use of Locality Sensitive Hashing (LSH) and logistic regression to efficiently and effectively identify duplicate products based on information found on online product listings. It also discusses how such information may be cleaned for use in the procedure. It was found that LSH can decrease the computational burden of the duplicate detection process by 85% while still achieving a final F1 score of 27%.

Keywords: Duplicate Detection · Locality Sensitive Hashing · Minhashing · Logistic Regression · Online Retailing

1 Introduction

As online retailers grow ever larger, especially after physical stores were inaccessible during the recent COVID-19 pandemic, it is ever more common to find the same product available on the websites of different webshops. This leads to various difficulties; customers might struggle to differentiate between duplicate products as opposed to very similar products, webshop comparators like `https://tweakers.net` [3] might struggle to aggregate lists of retailers for a single product, and so on. This work describes how Locality Sensitive Hashing (LSH) can be used to decrease the computational cost of detecting duplicates among such products by efficiently filtering out pairs of different products while minimising the loss of true duplicates. Subsequently, a logistic regression is performed using various similarity metrics as predictors in order to definitively classify a pair of products as duplicate or unique. Specifically, a set of televisions is analysed for which the model ID's are known, whereby the cost of the procedure can be evaluated in terms of product pairs which are falsely flagged as different as a function of the decrease in computational cost. As the scale of products available online grows, such a procedure might prove ever more compelling as a way to make processing such large sets of data tractable.

The data used in this analysis is described in section 2. Section 3 describes the methodology behind the product representation, the application of LSH and the subsequent duplicate detection. Lastly, various performance metrics are reported in section 4, followed by concluding remarks and suggestions for future work in section 5.

The methodology as described in the section 3 was implemented in Python [6] and can be found at [5].

2 Data

The dataset consists of 1624 listings of televisions across four webshops. For each of these listings, the model ID, shop name and webpage title are known, as well as a set of various product features that were listed as key-value pairs. Examples of such features are screen size, brightness, colour, number of video inputs, et cetera. As a model ID is in general not easily available on a website, it is not used in the duplicate detection throughout this work. Rather, it is only considered as the source of truth to verify the accuracy of the reported duplicate pairs. Furthermore, the listed product features vary greatly across shops, but even across products in the same shop. Section 3.1 discusses how relevant information may still be distilled from such inconsistent data.

With 1624 listings, there are $\binom{1624}{2} = 1317876$ total pairs of products. Out of these, 399 pairs are indeed the same product, or about 0.3‰. This is highly unbalanced data, which means the procedure must be very careful to discard a pair as a duplicate. However, such an imbalance is representative of typical product availability in webshops, as for instance Tweakers [3] lists on the order of 1600 unique models of television, such that if each model is available on ten webshops, about $1600 \cdot \binom{10}{2} / \binom{10 \cdot 1600}{2} = 0.6‰$ of all pairs of listings refer to the same product [1].

3 Methodology

3.1 Data Cleaning

LSH requires a representation of each product as a set of *features*. Such features may be of any data type, but as most information on webshops is presented in text form, duplicate detection in webshops naturally lends itself to features of the String type. Building on the work done in [2] and [4], sequences are extracted from the title and used as features. This makes sense, as webshops are incentivised to succinctly summarise a product in the page title. To accommodate for differences in formulation across shops, some common units found in titles like *inch* and *hertz* are first standardised, and all characters are replaced by their lowercase equivalents. Subsequently, strings are extracted from the titles by the

[1] Although the number ten is chosen rather arbitrarily, the result only varies between 0.3‰ and 0.625‰ for true values anywhere from 2 to ∞.

following regular expression, which matches any sequence of characters which contain two out of alphabetical tokens, numerical tokens and special characters:

```
([a-zA-Z0-9]*(([0-9]+[^0-9, ]+)|([^0-9, ]+[0-9]+))[a-zA-Z0-9]*)
```

Additionally, numerical strings with potential measurement units were extracted from the product features by the following regular expression:

```
(\d+(\.\d+)?[a-zA-Z]+|^\d+(\.\d+)?)
```

To expand on the work this work, additional attention was paid to the available product features. It was found that shops are reasonably consistent about using the same key for the same feature, although such keys do vary significantly across shops. It is therefore feasible to manually construct a list of the keys that each shop uses for a specific feature like screen diagonal or weight. By constructing such lists, the numeric value for weight, screen diagonal and refresh rate were extracted, as these are properties that are reported quite frequently. It was found that by searching for one or two keys per product feature per shop, these numbers could be found for about three out of every four products. Because some shops might report the numbers rounded to integers whereas others might report decimal numbers, rather than using these numbers directly, the sample deciles were calculated for the weight and diagonal. This was not found to be necessary for the refresh rate. The resulting numbers are cast into Strings of the form '*feature value*', e.g. 'Weight 2' for a product which falls into a the second weight decile.

Additionally, the product brand was extracted from the maps through testing for a manually constructed set of keys for each shop. Although this only yields brands for about one out of every four products, this does yield a near-exhaustive set of brands. By then searching every product's title for the brands in this set, the brand can be found for roughly three out of four products.

Combining the matches of the two regular expressions with the information on brand, weight, diagonal size and refresh rate yields the product representations as used for LSH. From these representations, any feature that occurs in more than 500 of the products was removed, as such features contribute little information in terms of differentiating between products.

3.2 Locality Sensitive Hashing

LSH is a fuzzy hashing procedure, which assigns inputs to 'buckets' such that similar inputs are assigned to similar buckets. However, as the product representations are quite long sets of Strings, and Strings do not naturally lend themselves well to similarity comparison, the products are first summarised into signatures through minhashing.

First, the universal set of product features is created as the union of the set representation of each product. Each product is then represented as a column in the binary matrix B. Because B is a very sparse matrix it pays to use sparse matrix methods to save on memory and compute cost, although it was found the universal set typically has around 5000 features so the size of B in memory may not be practically large. This binary representation is then converted to numeric signatures using standard minhashing procedures with 432 hashes. 432 is chosen as it has many prime factors, allowing many fractions of comparisons to be tested for performance evaluation.

It should be noted that because LSH is a fuzzy hashing procedure, it will yield a sizeable number of false positives, i.e. products that get hashed to the same bucket although they are in fact distinct products. However, fuzzy matching will have high recall. As the goal is to filter non-duplicates from the total set of potential duplicates, that is more important in this application.

The number of bands b and rows r determine the chance that LSH hashes two products to the same bucket. Therefore these parameters determine the recall and precision of LSH, where higher values of b and lower values of r decrease the chance that LSH suggests any two products to be duplicate. The lower the precision, the more products have to be checked with the duplicate detection method, which increases computational burden. As the goal of LSH is to filter non-duplicates, the optimal values should yield large recall for a desired reduction of the number of pairs that have to be checked, i.e. for a desired precision.

3.3 Duplicate Detection

There are various ways to check if two products are duplicates. [2] and [4] use the Multi-component Similarity Method, as described in [1]. In this work a simpler method is used. Because the model ID of each television is available in the dataset, a logistic regression can be fit on the data and used to predict the probability that any two products are duplicates. As predictors, two similarity metrics are used. First, the *SequenceMatcher* from the Python standard library [6] is used to calculate the similarity of the set presentations. Secondly, the Jaro-Winkler similarity of the cleaned titles is calculated. The regression can be fit on the available dataset and applied to new products.

Due to the very low percentage of duplicates among the total set of pairs, the training data is highly unbalanced and the logistic regression will generally yield very low estimates. This can be compensated for by adjusting the probability threshold above which two products are labelled as duplicates. Other techniques exist to adjust for unbalanced data, but they were not explored in this work.

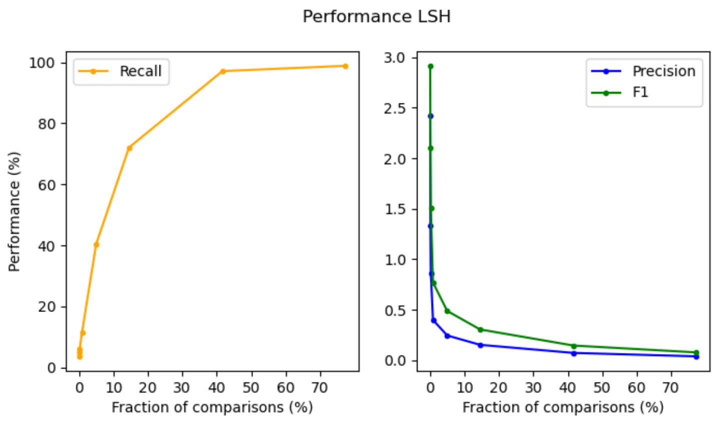
4 Results

The performance of LSH and the duplicate detection were evaluated through bootstrap with replacement over five bootstraps. For the duplicate detection, the logistic regression model was trained on the training split and applied to the test split to evaluate out-of-sample accuracy.

As fig. 1 shows, the recall of LSH grows with the fraction of comparisons as the precision drops. The recall grows until 100%, although the precision and thereby the F1 score fail to reach above 1%. Because the recall almost reaches unity near 45% fraction of comparisons, there seems to be little extra value in increasing the fraction beyond that point. This is also reflected in fig. 2, where the final F1 score does not increase above this fraction of 45%. Notably, although the recall after LSH does increase as the fraction grows from 15% to 45%, the final recall does not. This indicates that the logistic regression fails to identify the extra candidate pairs that are duplicates as such. However, the final precision does in conjunction with the precision of LSH in fig. 1, indicating that the logistic regression reliably identifies non-duplicates among the extra candidate pairs. Consequently, there is only a marginal change in F1 score between 15% and 75% fraction of comparisons, increasing only from 27% to 30%. It is reasonable to assume that this trend would continue as the fraction of comparisons approaches unity. Therefore, if we take the cost of applying LSH to be negligible, LSH enables a decrease in computational cost of 85% at only a loss of 3 percentage points in F1 score.

In summary, the procedure outlined in this work achieves good F1 at a low fraction of comparisons, indicating it can be effectively used to significantly decrease computational costs. However, as it only achieves a maximum precision of 23%, it may not suffice in applications where such a large number of false negatives is unacceptable, and alternatives to logistic regression may provide better precision, perhaps at the cost of recall.

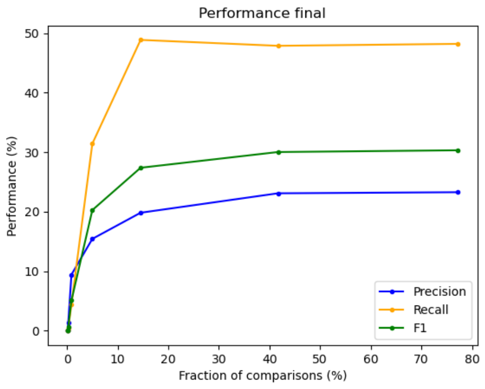
Fig. 1: Recall, precision and F1-score (F1*) of the candidate pairs found through LSH.



5 Conclusions

This paper suggests a way to efficiently identify duplicates among pairs of products listed on the web. Through pre-selection with LSH, the computational cost

Fig. 2: Recall, precision and F1-score of the final pairs found through logistic regression on the candidate pairs.



of doing so is reduced by 85% while maintaining an F1 score of 27% compared to approximately 30% when isn't used. The logistic regression used for duplicate detection proved reliable, as it could achieve consistent precision even when the number of false candidates found through LSH increased. However, it was not able to achieve high precision. Further research might investigate how the precision could be improved by for instance adjusting the logistic regression for the unbalanced data or through exploring other duplicate detectiong algorithms. Additionally, although this work lays out some ways to distill meaningful information from product feature lists, this may be expanded on in manifold creative ways to improve the product representations and thereby significantly increase the precision that LSH achieves.

References

1. van Bezu, R., Borst, S., Rijkse, R., Verhagen, J., Vandic, D., Frasincar, F.: Multi-component similarity method for web product duplicate detection. Proceedings of the 30th Annual ACM Symposium on Applied Computing (2015). https://doi.org/10.1145/2695664.2695818
2. van Dam, I., van Ginkel, G., Kuipers, W., Nijenhuis, N., Vandic, D., Frasincar, F.: Duplicate detection in web shops using lsh to reduce the number of computations. Proceedings of the 31st Annual ACM Symposium on Applied Computing (2016). https://doi.org/10.1145/2851613.2851861
3. Funnekotter, W.: (1998), `https://tweakers.net/`
4. Hartveld, A., van Keulen, M., Mathol, D., van Noort, T., Plaatsman, T., Frasincar, F., Schouten, K.: An lsh-based model-words-driven product duplicate detection method. Advanced Information Systems Engineering p. 409–423 (2018). https://doi.org/10.1007/978-3-319-91563-0_25

5. Rietveld, J.: LSH-based Pre-Selection for Duplicate Detection in Online Products through Logistic Regression (Dec 2023), `https://github.com/Encephala/CSBA-individual-assignment`
6. Van Rossum, G., Drake Jr, F.L.: Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam (1995)