A Data Type-Driven Property Alignment Framework for Product Duplicate Detection on the Web

Gijs van Rooij, Ravi Sewnarain, Martin Skogholt, Tim van der Zaan, Flavius Frasincar, and Kim Schouten

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands
{g.vanrooij1,r.sewnarain1,m.skogholt,tvanderzaan1}@gmail.com
{frasincar,schouten}@ese.eur.nl

Abstract. During the last decade daily life has morphed into a world of broadband ubiquity, where devices facilitate constant engagement. As a consequence of this, the area of e-commerce has seen an immense growth. Despite the market opportunities for retailers and the ease for customers to acquire products through webshops, the shift to digital retail has its drawbacks. For example, it leads to cluttered and incomparable information among different webshops, which calls for an automated method to regain homogeneity in product representations. This paper presents a product duplicate detection solution, which exploits a data type-driven property alignment framework. Based on the performed experiment, we show a statistically significant improvement of the F_1-score from 47.91% to 78.13% compared to an existing state-of-the-art approach.

Keywords: Web products, data types, property matching, duplicate detection

1 Introduction

The evidence for e-commerce popularity surrounds us, whether it is commuters immersed in their tablet to order groceries or a shopper buying a book on her phone to avoid the hassle of carrying it home and save time spent on shopping. Online shopping has become widely used and a common way of acquiring products. Digital retail sales has hit an all time high and keeps growing according to recent projections by eMarketer [5]. As stated by Jeff Jordan of Andreessen Horowitz: 'We're in the midst of a profound structural shift from physical to digital retail'. The prosperity in e-commerce leads to more laborious product comparisons among different webshops, since characteristics of products are represented by different lexical representations and webshops provide different characteristics on products. This calls for an automated framework on product duplicate detection to regain homogeneity in product representations. The basic requirement of this framework involves enabling a fair comparison among products of different webshops. Although a lot of methods have been proposed on duplicate detection [3,15], the idea of pre-processing data first, apart from data cleaning [4], is not a popular one. One of the few methods related to property alignment is [9], which takes the characteristics of products (which will be referred to as properties from now on) into

account, as well as the use of different measurement units. Those ideas are both adopted and further extended upon in our research. Many methods on duplicate detection use either TF-IDF [11] or use some lexical similarity measure [13], but lack usage of a semantic similarity between words. In addition, besides some of the existing approaches on duplicate detection, such as [13], most approaches do not take typographical errors into account. To solve these drawbacks of existing methods our method exploits the semantic similarities between words and tackles the problem of typographical errors. In addition, we propose to use an elaborate pre-processing part to determine matching properties to be used in the remaining steps of our solution.

A typical product duplicate detection framework consists of three steps. The first step involves reducing the number of comparisons between products by so-called blocking. This ensures that only the products within the same block or partition of the dataset are compared. Next, the similarity scores between product-pairs within the same block are calculated. In the last step product duplicates are determined based on these scores. Our solution for product duplicate detection includes an additional step which precedes the above-mentioned steps. This step consists of an extensive data type-driven property alignment (sub)framework. Products are characterized by several properties, each property consists of a property Key and a property Value (from now on referred to as simply Key and Value). For example, for the 'aspect ratio' of a television the property Key is 'aspect ratio' and the property Value could be '4:3'. These together form a property or Key-Value pair (KVP). First, a data type is determined for a property Value. Next, the similarity of a pair of properties is evaluated based on the similarity of the two Keys and the similarity of the corresponding Values. After the extensive data type-driven property alignment (sub)framework is used to find matching properties, the product duplicate detection framework is applied, which makes use of these matched properties. The product duplicate detection framework is built upon the state-of-the-art Multi-component Similarity Method (MSM) [2]. This framework determines the product duplicates among different webshops, using a novel similarity function and an adapted single linkage clustering algorithm.

The evaluation of the proposed framework is done on a dataset that describes 1446 televisions accompanied by their characteristics or so-called properties. The data is a collection of products from two webshops, i.e., *bestbuy.com*, and *newegg.com*. The dataset contains 774 and 672 products for these webshops, respectively. Each unique television has a 'modelID', 'title', 'url', and 'webshop' along with its list of properties. In total 200 unique Keys are present in the dataset. The product with the most comprehensive characteristics contains 61 properties, whereas the product with least exhaustive characteristics only contain one property. The characteristics of *bestbuy.com* consists of 37 properties on average, whereas the characteristics of *newegg.com* only contains 22 properties on average.

The remainder of this paper is organized as follows. Sect. 2 elaborates upon existing techniques on product duplicate detection and property alignment. We describe the proposed methods in Sect. 3 and in Sect. 4 we evaluate the results of our proposed methods and in Sect. 5 we discuss our results and give some suggestions for further research.

2 Related Work

The aim of our research is to better detect product duplicates by using an extensive property alignment framework. In order to do so, we use existing methods and introduce new ones. As will be discussed in both Subsect. 2.1 and Subsect. 2.2, there are existing methods that already give a reasonably good performance, however, they have shortcomings that we will address by means of extensions and new methods. In Subsect. 2.1 we present an overview of current methods that address property alignment. Subsect. 2.2 shows some of the current methods that deal with duplicate detection.

2.1 Property Alignment

The idea of intelligently pre-processing data, before pairwise product comparisons are conducted, is not popular. Most of the research concerns data cleaning, see for example [4], which provides an overview of methods on data preparation. Many are concerned with the Values corresponding to a Key to determine the data type of the Key, whereas [9] formalizes product information in an ontology in order to compare products. Data can be represented in different ways, for example as a string or an integer. One can imagine that comparing strings with integers makes no sense. Using such information should make the property alignment more accurate. [9] introduced the use of measurement units. '2m' is not the same as '2lbs.' and therefore they should not be considered as a match. In our method we introduce Block Classification where the insight gained from [9] is used. Furthermore, we add Value Classification and Property Classification in order to align properties and detect property matches, making use of the Block Classification.

2.2 Duplicate Detection

The first method we discuss for duplicate detection is the one proposed by [11], which is based on the popular Term Frequency–Inverse Document Frequency method (TF-IDF). First, it is determined how many times a unique word occurs in a document, which is the Term Frequency. Then, the Inverse Document Frequency is calculated by taking the logarithm of the number of documents minus the logarithm of the number of documents containing that word. The final TF-IDF score for each word is calculated by dividing the term frequency by the inverse document frequency. A cosine similarity matrix is then constructed for all combinations of products. A shortcoming of this method is that it makes no use of the semantic similarities between words.

[14] proposes the Title Model Words Method (TMWM). This method determines if two products are duplicates by considering a similarity score. At first, an initial product title similarity score is calculated. This is done by calculating the cosine similarity of two product titles (interpreted as a bag of words). Then, the product title similarity score is compared with a threshold α to determine if the two products are duplicates. If the products are not duplicates based on the product title similarity score, *model words* are extracted for the two products. Model words are words consisting of both numeric and non-numeric tokens. The two products are considered to be different if the normalized Levenshtein distance of the non-numeric part is smaller than the threshold 0.5, while

the normalized Levenshtein distance of the numeric part is larger than the threshold of 0.5. If the model words match, i.e., the numeric part is equal and non-numeric part is approximately the same, then it could be an indication of duplicate products. Therefore, the authors update the initial similarity score by taking a weighted average of the initial product similarity title score and the average Levenshtein distance of the model words. Based on the final similarity score and a threshold δ, it is determined if the products are duplicates. This method does not consider different representations of data. Therefore, products that are in fact duplicates, can be labelled as different.

[1] introduces the Hybrid Similarity Method (HSM), which extends the TMWM method. Instead of only comparing the title, [1] uses information from the product properties in order to construct a similarity measure. This measure is based on two methods. The first method checks for each combination of properties from both products if the corresponding keys are matching. Then, the similarity between the corresponding Values is calculated. The authors use the cosine similarity as the first part of the similarity measure. The second part concerns properties with no matching Key. Model words are extracted from the Values of these properties and combined in two sets, one for each product. Then, the percentage of matching model words between the two sets is calculated, which is the second part of the similarity measure. At last, the authors of [1] introduce a weight θ which is based on the number of Key matches and take a weighted average of both parts to construct the similarity measure. A shortcoming of this method is that it is only applicable for comparing two shops.Differently, [2] proposes a Multi-Component Similarity Method(MSM), which is applicable to more than two shops.

MSM builds on HSM and makes use of TMWM. It introduces a hierarchical clustering approach, which allows for comparing multiple shops. Its similarity measure consists of three parts. At first, a list of brands is introduced to detect if products have different brands. If they are not different, the Keys are given a score by making use of q-grams [12]. The authors use q-grams, in order to deal with typographical errors and abbreviations at both the Key and Value level. The second part is the same as in HSM, where the percentage of matching model words is calculated. The third part consists of calculating a similarity measure based on TMWM, where TMWM is adjusted to return a similarity score instead of a boolean result.

We extended the model of [2] by introducing a data-type property alignment framework as a pre-processing. Therefore, our approach in finding the similarity measure between properties differs. Furthermore, instead of using a hierarchical clustering approach, we use a trained threshold, which we compare with the similarity score, in order to determine if two products are duplicates. In order to overcome typographical errors and abbreviations, we will make use of q-grams similarity as well, which is actually the same as the Jaccard Similarity with q-shingles [10], later on called k-shingles.

3 Method

To determine property matches and detect duplicates of products over different webshops several methods have been incorporated in two frameworks. This section describes the two frameworks and the various methods that have been implemented. The first framework is the property alignment framework, which is described in Subsect. 3.1

and the second framework is the product duplicate detection framework, which is described in Subsect. 3.2.

3.1 Property Alignment Framework

This subsection explains the property alignment framework and all incorporated methods in detail. Subsubsections 3.1.1, 3.1.2, and 3.1.3 elaborate upon the *Block-*, *Value-*, and *Property Classification* methods, respectively. In Subsubsect. 3.1.4 the property matching is explained. Subsubsections 3.1.1, 3.1.2, and 3.1.3 build upon each other's results in order to classify each property to a certain type. These property types are last used in Subsubsect. 3.1.4 to match properties.

3.1.1 Block Classification

Our goal is to align or match properties of TVs from different webshops. The data from different webshops is not easily comparable. In order to tackle this problem, we introduce *Block Classification*. Recall that a property of a product consists of a Key and a corresponding Value. Keys are directly comparable, since each Key is a string. However, Values are not always a string and therefore comparing Values is not as straightforward. The Value representation can differ quite significantly between webshops and even within a webshop. For example, the Key 'Brightness' is described with the Value '450 cd/m^2'. This Value consists of two separated parts (split by a space), namely an integer and a string. The same key is also described with the Values '350 cd/m2' and '450Nit' for other products. The latter consists of two parts as well, viz. an integer and a string. However, the representation of the Value is not separated by a space. Clearly, this shows that Values are not easily comparable.

In order to deal with the previously mentioned issues, we propose *Block Classification*. The idea is to partition a Value based on white spaces, where the resulting partitions of the Value are called blocks. Now that the Values are split into blocks, one can compare the blocks of the Values. This makes sense if one compares integers with integers, strings with strings, and so on. Therefore, we introduce *Type Classification* of the blocks. First it is checked whether a block is strictly Numerical. In this case, it is sequentially checked if the block is of the type: Integer, Float, Fraction, Ratio, Percentage, or Dash.

If a block strictly consists of alphabetical characters, it is checked whether the block belongs to the block type Boolean, Measure, or Compound. If Boolean, Measure, and Compound are not suitable as types, the block will be classified as a Word block. A Boolean block can have either 'Yes' or 'No' as Value. The block type Measure represents a measurement unit listed in Table 1. The expressions of the measurement units vary over webshops, for example, the weight of a product might be given in Kilogram(s), Kilo(s), Kg(s), Pound(s), or Lb(s). The block type Compound represents the symbol 'x' or the word 'times' to connect two or more blocks, for example: '52 x 6.8'. If a block is neither Numerical nor Alphabetical, the block type Universal will be assigned.

Table 1: Measuring units expressed by different unit symbols.

Measuring unit	Unit Symbol
Weight	Kilogram(s), Kilo(s), Kg(s), Pound(s), Lb(s)
Frequency	Hertz, Hz
Energy & Electricity	Watt(s), W, kWh, Joule(s), kiloJoule(s), kJ, J, Volt(s), V
Sound intensity	Bel(s), Decibel(s), B, dB
Length	Inch(es), cm, mm, ”
Brightness	cd/m2, cd/m^2, cd/m, Nit

Even though *Type Classification* is introduced, one can still find cases where a comparison is not optimal. Consider for example the Values ‘180Hz’ and ‘180 Hz’, which describe the same information. The first block is classified as Universal, while for the latter Value the blocks are classified as Integer and Measure. Clearly, both Values are equal. In order to overcome this problem, an Universal block, which contains an integer and a measurement unit (in this case Hz), will be split into 2 blocks, we use the measurement units form Table 1. Namely, an Integer block and a Measure block. In this way, one is able to compare 180Hz and 180 Hz. Table 2 shows how several Values are split into block and how these blocks are subsequently classified. Also it is indicated how Values containing a measurement unit are classified, such as ‘180Hz’.

Table 2: Examples of *Block Classification*.

Property value	Set of blocks	Types of blocks
180Hz	‘180’ + ‘Hz’	Integer + Measure
180 Hz	‘180’ + ‘Hz’	Integer + Measure
4:3 and 16:9	‘4:3’ + ‘and’ + ‘16:9’	Ratio + Word + Ratio
B007B9PMCO	‘B007B9PMCO’	Universal
122 W	‘122’ + ‘W’	Integer + Measure
Yes	‘Yes’	Boolean
52 x 6.8	‘52’ + ‘x’ + ‘6.8’	Integer + Compound + Float
52% humidity	‘52%’ + ‘humidity’	Percentage + Word
55-2/3 mm	‘55-2/3’ + ‘mm’	Dash + Measure

3.1.2 Value Classification

The goal is to compare properties. Recall that a property consists of a Key-Value pair. As mentioned before, comparing Values is not straightforward. In the previous section

we introduced *Block Classification* and we have shown how to classify parts (blocks) of Values. However, we do not want to compare parts of the Values, but the Values themselves. Therefore, we introduce *Value Classification*, where the results of the *Block Classification* is used to determine the type of the corresponding Value.

The classification of a Value depends on the number of blocks. Three options are considered: one block, two blocks, and three or more blocks. For each of these options the *Value Classification* differs. In case a Value consists of one block, three types are considered: Boolean, Quantitative, and Qualitative. A Value is of type Boolean if the block is a Boolean block. A Value is classified as Quantitative if the block is strictly Numerical. And lastly, a Value is of type Qualitative if the block is either a Word block or a Universal block.

In case a Value consists of two blocks, the following three types are considered: Measure, Quantitative, and Qualitative. A Value is of type Measure if the first block is Numerical followed by a Measure block. A Value is of type Quantitative if the first block is of type Numerical followed by a Word block. If the two blocks are of any other combination of types, the Value is classified as Qualitative.

In case a Value consists of three or more blocks, two types are considered: Compound, and Qualitative. At first it is checked whether one or more blocks are of type Compound. A Value is classified as a Compound Value, if the following three restrictions are satisfied. First, the one or two blocks before a Compound block must be either Numerical followed by a Word block or Numerical followed by a Measure block or simply a Numerical block. Secondly, between two Compound blocks, the one or two blocks must be Numerical followed by a Word or a Measure block or simply a single Numerical block. At last, the first block after the last compound block must be Numerical. For example, '52 mm x 45 mm x 20' would be classified as a compound Value. This is visualized in Eq. 1, where a Compound Value is considered. A Value is Qualitative, if there is no Compound block present.

$$Value: \overbrace{\underbrace{Block_i + Block_j}_{\text{Numerical + Word or Measure}} \; x \; \underbrace{Block_k + Block_l}_{\text{Numerical + Word or Measure}} \; x \; \underbrace{Block_m}_{\text{Numerical Block}}}^{\text{Compound Type Value}} \tag{1}$$

3.1.3 Property Classification

In order to compare two properties we also classify the type of the properties, in other words *Property Classification* is introduced. To determine the type of a property the previously determined Value types are used. First, for each unique Key the Values belonging to this Key are aggregated in a list. For example, the key 'Brightness' will have multiple observations in the dataset all with different Values. For example, '450 Nit' could be found or '350 cd/m' or other Values with different representations. Obviously, all these Values together indicate the type of the property. It can also happen that certain Values have been misspelled or otherwise incorrectly entered. For example, it could be that for the Key 'Brightness' there is a Value which is simply '250'. This Value would be classified as a Quantitative Value, while the property 'Brightness' should still be classified as a Measure type. This is why the type of the property is equal to the type

that has been assigned most frequently to the Values belonging to the unique Key. This way certain errors in the Values are circumvented.

The whole process of *Block Classification*, *Value Classification*, and *Property Classification* is shown below in Alg. 1.

Algorithm 1 Property Classification

```
Initialize list of products with properties, X
Initialize list of aggregated properties, A
for Unique key k ∈ X do
    Initialize list of values, V
    for prop ∈ X with key k do
        Split value v ∈ prop by space
        Initialize list of blocks, B
        for substring s ∈ v do
            Create block b(s)
            Determine type of b(s)
            Add block b(s) to B
        end for
        Add list B to value v
        Determine type of value v based on B
        Add value v ∈ prop to list of values V
    end for
    Create aggregated property AP_k with key k and list of values, V
    Determine type of aggregated property AP_k
    Add aggregated property AP_k to A
end for
```

As can be seen in this Algorithm 1, the different values are first split into a list of Blocks after which the type of the Blocks is determined. When this is done the type of the value can be determined based on the Blocks. Finally, a list of aggregated properties is filled and each aggregate property is classified as being of a certain type based on the Values.

3.1.4 Property Matching

In the previous subsections we introduced *Block Classification*, *Value Classification*, and *Property Classification*. These are the building blocks for the property matching method. In order to match properties, a quantification of the similarity between properties is needed. To compare properties a *Property Score* is calculated to represent the similarity between two properties. Property matching is based on the similarity between two Keys (Key Score) and the similarity between two Values (Value Score). The Property Score is a weighted average between these two scores. If the Property Score exceeds a certain trained threshold, then the two compared properties are considered as a match.

The Key Score is a combination of two similarity measures; the lexical similarity and the semantic similarity. To find the lexical similarity, the Jaccard similarity with k-shingles [10], with k representing the number of considered characters, is used. For this metric we vary k between 2 and 8. For the semantic similarity between two Keys the meaning of the words are considered, where WordNet [8] is used as a semantic lexicon. One can understand that Keys which have the same meaning, but are represented with different words, should have sets of synonyms (Synsets) in common and thus a high (semantic) similarity. First, the lemmas (the root of a word) of all the words in both Keys are retrieved. These lemmas can then be disambiguated and assigned a certain sense from WordNet using Lesk [6], where the context is defined as the words in the other keys of the current type of product, i.e., televisions in our case. A word sense is simply the meaning of a word with a set of synonyms and a certain gloss. When comparing two Keys with certain synsets assigned to each individual word or compound words, if this is the case, the Jaccard similarity is used on the glosses of the synsets. Basically, two sets are created with the combined glosses and the Jaccard similarity is taken over these two sets of glosses after stopwords[1] have been removed. The Key Score is a weighted average of the lexical similarity score and the semantic similarity score. This is summarized in Eq. 2.

$$Key\,score = (1-\delta) \times Lexical\,Sim.\,score + \delta \times Semantic\,Sim.\,score, \quad (2)$$

where δ is a trained weight.

The Value Score depends on the type of the property. First, we iterate through all the Values belonging to the unique Keys of the compared properties. Only the Values that have the same data type as the property are used for the comparison. In case both properties are of type Qualitative, a lexical similarity metric is used. Here, the same lexical similarity is used as in the Key Score, i.e., k-shingles Jaccard Similarity. In case both Values are Boolean, we first calculate the fraction of the number of times 'Yes' occurs in the Values. Then, one minus the absolute value of the difference between the fractions is calculated and this is the Value Score. In case both of the Values are either of type Quantitative, Measure, or Compound, the Mann-Whitney U test is used [7]. This test is used to check whether the null hypothesis of a equal distribution of Values is statistically significant. Whenever both Values are Quantitative, the Numerical blocks are used in the Mann-Whitney U test. Whenever both Values have been classified as Measure Values and the Measure blocks are of the same unit, the Mann-Whitney U test is performed on the Numerical blocks of both Values. When both Values are Compound, the Numerical blocks are used in the Mann-Whitney U test. The Value score is then equal to the p-value of the Mann-Whitney U test. Now that the Key and Value Score have been obtained, the Property Score is a weighted average of the Key and Value score. This is shown in Eq. 3.

$$Property\,score = (1-\theta) \times Key\,score + \theta \times Value\,score, \quad (3)$$

where θ is a trained weight. If this Property Score is higher than a certain trained threshold, the properties are classified as match.

[1] The stopwords used can be found at http://www.ranks.nl/stopwords

3.2 Product Duplicate Detection Framework

The product duplicate detection framework consists of 3 heuristics to decrease the number of product comparisons before doing the actual duplicate detection computations. After heuristics-based preprocessing, the framework contains 2 steps to match products that are considered duplicates. First, the so-called shop heuristic assumes that there are no product duplicates present within the same shop. With this heuristic a lot of unnecessary comparisons are avoided. The second heuristic is the brand heuristic. This assumption presumes there are no product duplicates with the same brand. The third heuristic is the screen size heuristic. The assumption is that there are no product duplicates with different screen sizes. The second and third heuristics will be explained in more detail in Subsubsections 3.2.1 and 3.2.2, respectively. To test whether products need to be indicated as being duplicates or not, the similarity between product titles and a list of properties are measured by means of a title-based score (i.e., title score) and properties-based score (i.e., property score). The first step consists of a score calculated based on the two product titles of the products that are being compared, as can be found in Subsubsect. 3.2.3. The second step consists of a score that is calculated based on the properties of the two products, as can be found in 3.2.4. Last, these steps will be combined into a so-called 'product score' as will be explained in Subsubsect. 3.2.5.

3.2.1 Finding the Brand for All Products

To find the brand of all the products in our dataset we make use of a list of well-known TV brands and manufacturers from Wikipedia[2]. This can easily be extended for other types of products as there are many such lists of manufacturers. First we scan all the products with their properties and count how often a property Value contains one of the TV brands from the list. The property Key for which a Value is most often found in the TV list, is then denoted as the brand key. All brands which are not present in the list, but belong to the brand key are added to the brand list. When this is done all the products that do not have the same brand are no longer compared by the algorithm.

3.2.2 Finding the Screen Size for All Products

The third heuristic assumes product duplicates can not have different screen sizes. Almost all products comprise the screen size in their product title, where the screen sizes that are in the titles are represented by some digits followed by the ”(inch) sign and can easily be extracted from the title. The products that do not contain their screen size in the product title are included in the comparison. All of the products with different screen sizes are no longer compared, since it is impossible that two products with different screen sizes are duplicates. This heuristics is only used to decrease the number of unnecessary comparisons of products and henceforth decrease the time it takes to run the algorithm.

[2] https://en.wikipedia.org/wiki/List_of_television_manufacturers

3.2.3 Scoring of the Products on Their Titles

The similarity score of compared products based on their titles is found using the lexical similarity between these titles. First both titles are cleaned by removing the brand and two common words: 'refurbished' and 'open box:'. We consider these words to be meaningless in comparing products and we found these by analyzing the available data. Next, the similarity score is calculated using the same lexical similarity measure as for the property alignment framework, i.e., k-shingles Jaccard Similarity. This final score represents the title score for the two products that are being compared.

3.2.4 Scoring of the Products on the Values of Their Matching Keys

For the two products that are being compared a score is calculated for all their properties that either have the same Key or have been matched by our property matching algorithm. Each pair of matched properties adds value to the total property score. Note, each score of the separate pairs of matched properties is weighted with a learned weight dependent on the type of the properties.

If two properties are both of type Qualitative, Boolean, or are non-matching types, the similarity is calculated using the lexical similarity measure. For two properties that are both Measure properties, the score is simply 1, if the absolute value of the difference of the values is smaller than 1. The idea is that small differences in numerical values do not indicate any dissimilarity, however, we need to take integers into account. For example, the number of HDMI ports can be 0,1,2,... If one product has 1 port and the compared product has 2, then this indicates dissimilarity between the two products. This is why we chose for an absolute difference smaller than 1. If either the type of measurement differs or the absolute value of the difference between the Values is larger than 1, the score is -1. For two properties that are both of type Quantitative the score is again 1 if the absolute value of the difference of the two Values is smaller than 1 and the score is -1, otherwise. However, if the Values do not solely consist of one numerical Value, the remainder is also taken into account by using the lexical similarity measure. The score of the two properties of type Quantitative is then the average of the score of the numerical part and the score of the non-numerical part. For Compound properties the score is simply the average of the score of each individual part. Each individual part is treated as a Quantitative type comparison and is simply scored 1 or -1 dependent on the absolute value of the difference of the two Values.

At last, the weighted scores are aggregated and the total score is normalized by dividing by the number of scores added up, such that the total property score ranges from 0 to 1. This can be seen in Equation 4:

$$Property\,score = \frac{\sum_{p_i=p_j} score(p_i, p_j) \times w_i}{Number\,of\,times\,p_i\,matches\,p_j}, \tag{4}$$

where p_i is property i and w_i is the weight dependent on the type of the two properties. This score on the similarity of the matching properties is used later on to indicate duplicate products.

3.2.5 Final Score Based on Matching Keys and Titles

The final score between two products is a weighted average of the score calculated between the two titles and the score calculated for all the matched properties. This results in the following equation:

$$Product\,score = (1 - \tau) \times title\,score + \tau \times property\,score, \tag{5}$$

where τ is a trained parameter. If this product score exceeds a certain trained threshold the products being compared are indicated as product duplicates.

4 Evaluation

In Subsect. 4.1 we describe the process of evaluating our algorithms on property matching and product duplicate detection. Subsect. 4.2 provides the results on the framework for property matching and Subsect. 4.3 evaluates the obtained results from the product duplicate detection framework.

4.1 Evaluation Method

The so-called bootstrap method, i.e., random sampling with replacement, is used to assign measures of accuracy to sample estimates. The number of random samples is set to 50. Each random sample consisting of numerous TVs represents the training set, whereas the remainder of the TVs not allocated to the random sample is used as a test set for validation. Additionally, all the parameters used throughout are trained using a genetic algorithm. This includes: the two thresholds, the weights in the Key score, Property Score, and Property Score, and the weights in the Product Score.

When evaluating the algorithm, a 'match' for the property matching framework means two different properties are considered the same and a 'match' for the product duplicate detection framework means two products are considered as duplicates. The F_1-measure, which is the harmonic mean of the precision and recall, is used as the performance measure. Additionally, the precision and recall themselves are used for measuring the performance. The precision is how accurate the algorithm is, i.e., the correct matches divided by the correct matches plus incorrect matches. The recall is an indication of how much is missed by the algorithm, i.e., the correct matches divided by the correct matches plus the missed matches, that our algorithm should have indicated.

The evaluation of the property matching framework and of the product duplicate detection framework are conducted separately. In order to be able to find performance measures for the algorithms the results of the algorithms should be related to a golden standard. For the property matching framework this golden standard is based on the so-called inter annotator agreement (IAA)[3]. Matches between Keys are added to the golden standard in case a sufficient percentage of selections by the annotators overlap. In our case we have used four annotators. If three out of four annotators indicate the same

[3] https://corpuslinguisticmethods.wordpress.com/2014/01/15/what-is-inter-annotator-agreement/

match then it is considered a sufficient match. In our case the four annotators agreed unanimously in 82.2% of the cases. When two products share the same ModelID, they are considered as duplicates, which acts as our golden standard. Note that the ModelIDs are often missing on the Web, which makes our framework interesting for duplicate detection, but we have used datasets, where the ModelID is present, to evaluate our algorithm and to be able to train the parameters.

4.2 Property Alignment Framework

In Table 3, the reported F_1-measures of the 50 bootstraps for the property matching framework can be seen. These results are with using the semantic similarity and with the Jaccard similarity using different k-shingles as the lexical similarity measure. When using the semantic similarity measure, the results were higher than without, which is why we only reported these. The algorithm was executed for *k* varying between 2 and 8. The highest results were achieved for $k = 3$. For both $k = 7$ and $k = 8$, there was a significant drop of around 5 percentage points in performance for the algorithm. This is quite an intuitive results, since a lot of the Keys are quite short. This means that for a high *k*, the Keys are completely in a single shingle and would have to be identical otherwise the similarity is 0. This is, obviously, not always the case and this is probably why the results start dropping around $k = 7$.

Table 3: The average F_1-measure for the property alignment framework. The results are for $k = 2$ to $k = 8$ and with the semantic similarity measure

Metric	Mean	Std. Dev.
Jaccard 2-shingle	79.69	2.60
Jaccard 3-shingle	81.55	1.96
Jaccard 4-shingle	81.14	1.86
Jaccard 5-shingle	80.26	2.06
Jaccard 6-shingle	78.32	2.48
Jaccard 7-shingle	73.19	2.66
Jaccard 8-shingle	71.13	2.30

4.3 Product Duplicate Detection Framework

In Table 4, the reported F_1-measures of the 50 bootstraps of the product duplicate detection framework can be seen. These results have been gained using the semantic similarity and with k ranging from 2 to 8. For the product duplicate detection the highest performance was achieved for $k = 4$. For the product duplicate detection framework, the drop in performance is much less severe, when k is increased. This is probably due to the fact that the titles are often quite long for products. As this is the application of the lexical similarity measure with the most impact, it seems that a high k has less negative influence, since the titles are still compared with a lot of shingles instead of a single shingle.

Table 4: The average F_1-measure for the product duplicate detection framework. The results are for $k = 2$ to $k = 8$ and with the semantic similarity measure

Metric	Mean	Std. Dev.
Jaccard 2-shingle	76.41	2.84
Jaccard 3-shingle	77.88	2.68
Jaccard 4-shingle	78.13	2.65
Jaccard 5-shingle	77.37	2.75
Jaccard 6-shingle	76.65	2.69
Jaccard 7-shingle	76.32	2.52
Jaccard 8-shingle	76.10	2.63

To compare our method to that of MSM, we have executed our algorithm on the same bootstraps as MSM, with 50 bootstraps in total. The results can be seen in Table 5. On average our methods scores 30.21% better than the MSM method. The standard deviations can also be found in Table 5. Since these are paired observations we can use a paired t-test. Our method significantly outperforms MSM, even with a 99.9% significance level.

Table 5: This table shows the F_1-measures for MSM and our proposed algorithm, as well as the difference in performance. Our algorithm was executed using the semantic similarity measure and $k = 4$-shingles

Method	F_1-measure	Std. Dev.	95% Interval
MSM	47.91	3.05	[41.93 - 53.89]
Our method	78.13	2.65	[72.94 - 83.32]
Difference	30.21	3.65	[23.01 - 37.36]

Regarding the complexity analysis of the developed algorithms, we can state that the Property Alignment framework has complexity $O(p^2)$ where p is the number of (unique) properties in the dataset, and the Product Duplicate Detection Framework has complexity $O(n^2\ \bar{p}^2)$ where n is the number of products and $\bar{p}$ is the average number of properties per product in the dataset.

5 Conclusion

In this paper we have proposed and implemented an extensive property alignment framework, which makes use of inferred data types. It has an F_1 of 81.55% with the best performing similarity measure on 50 bootstraps. Additionally, we have significantly improved upon the MSM algorithm with regard to the product duplicate detection. Our results are on average 30.21% higher, when using the F_1-measure. The significance of the improvement has been tested with a t-test and our results are significantly better than those of MSM, even with a 99.9% significance level. Moreover, the framework can easily be adapted to other classes of products.

The first suggestion for future work is to introduce a framework that can handle different measurement units, such that, for example, properties measured in feet can be compared to properties measured in meters. Furthermore, our method searches for the similarity of property pairs based on the similarity of property Keys *and* property Values. Additionally, it could prove useful to exploit the similarity of Keys with Values for finding additional property matches. For example, for the property 'Parental Control' in our dataset the Value was most often 'V-Chip', while there is also a property with Key 'V-Chip', where the Values are 'Yes' or 'No'.

References

1. de Bakker, M., Frasincar, F., Vandic, D.: A Hybrid Model Words-Driven Approach for Web Product Duplicate Detection. In: 25th International Conference of Advanced Information Systems Engineering (CAiSE 2013). pp. 149–161. Springer (2013)
2. van Bezu, R., Borst, S., Rijkse, R., Verhagen, J., Vandic, D., Frasincar, F.: Multi-Component Similarity Method for Web Product Duplicate Detection. In: 30th Symposium On Applied Computing (SAC 2015). pp. 761–768. ACM (2015)
3. Bilenko, M., Mooney, R.J.: Adaptive Duplicate Detection using Learnable String Similarity Measures. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003). pp. 39–48. ACM (2003)
4. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Transactions on Knowledge and Data Engineering 19(1), 1–16 (2007)
5. eMarketer: Retail Sales Worldwide Will Top $22 Trillion This Year (December 2014), `http://www.emarketer.com`
6. Lesk, M.: Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In: 5th Annual International Conference on Systems Documentation (SIGDOC 1986). pp. 24–26. ACM (1986)
7. Mann, H.B., Whitney, D.R.: On a Test of whether one of two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics pp. 50–60 (1947)
8. Miller, G., Beckwith, R., Felbaum, C., Gross, D., Miller, K.: Introduction to WordNet: An On-Line Lexical Database. In: International Journal of Lexicography (special issue), pp. 3(4):235–312 (1990)
9. Nederstigt, L.J., Aanen, S.S., Vandic, D., Frasincar, F.: FLOPPIES: A Framework for Large-Scale Ontology Population of Product Information from Tabular Data in E-commerce Stores. Decision Support Systems 59, 296–311 (2014)
10. Rajaraman, A., Ullman, J.D.: Finding Similar Items. In: Mining of Massive Datasets. vol. 77, pp. 73–80. Cambridge University Press Cambridge (2012)
11. Salton, G., Fox, E.A., Wu, H.: Extended Boolean Information Retrieval. Communications of the ACM 26(11), 1022–1036 (1983)
12. Ukkonen, E.: Approximate String-Matching with Q-grams and Maximal Matches. Theoretical Computer Science 92(1), 191–211 (1992)
13. Vandic, D., van Dam, J.W., Frasincar, F.: A Semantic-Based Approach for Searching and Browsing Tag Spaces. Decision Support Systems 54(1), 644–654 (2012)
14. Vandic, D., Van Dam, J.W., Frasincar, F.: Faceted Product Search Powered by the Semantic Web. Decision Support Systems 53(3), 425–437 (2012)
15. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient Similarity Joins for Near-Duplicate Detection. ACM Transactions on Database Systems 36(3), 15 (2011)