Multi-component Similarity Method for Web Product Duplicate Detection

Ronald van Bezu
ronaldvanbezu@gmail.com

Sjoerd Borst
s.v.borst@gmail.com

Rick Rijkse
rickrijkse780@gmail.com

Jim Verhagen
j.m.verhagen@gmail.com

Damir Vandic
vandic@ese.eur.nl

Flavius Frasincar
frasincar@ese.eur.nl

Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, the Netherlands

ABSTRACT

Due to the growing number of Web shops, aggregating product data from the Web is growing in importance. One of the problems encountered in product aggregation is duplicate detection. In this paper, we extend and significantly improve an existing state-of-the-art product duplicate detection method. Our approach employs a novel method for combining the titles' and the attributes' similarities into a final product similarity. We use q-grams to handle partial matching of words, such as abbreviations. Where existing methods cluster products of only two Web shops, we propose a hierarchical clustering method to handle multiple Web shops. Applying our new method to a dataset of TV's from four Web shops reveals that it significantly outperforms the Hybrid Similarity Method, the Title Model Words Method, and the well-known TF-IDF method, with an F_1 score of 0.475 compared to 0.287, 0.298, and 0.335, respectively.

1. INTRODUCTION

With a substantial growth in the amount of Web shops [13], the field of product duplicate detection is becoming increasingly important. When aggregating or comparing data from different Web shops, one should take into account that a single product might be represented in different ways. Therefore, duplicate detection methods are used to identify which product descriptions represent the same product.

Such duplicate detection methods on the Web are harder to devise than in relational databases. On the Web we have to deal with unstructured data, where the keys and their values are not predefined. API's and techniques such as text mining could be used to extract useful information from the Web, to be employed in the duplicate detection process. Currently, the field of duplicate detection on the Web is mostly focused on pair-wise duplicate detection [3, 15]. However, since the Web is crowded with Web shops selling identical items, future duplicate detection algorithms must be able to handle more than two products.

SAC'15 April 13-17, 2015, Salamanca, Spain.
Copyright 2015 ACM 978-1-4503-3196-8/15/04
http://dx.doi.org/10.1145/2695664.2695818 ...$15.00.

We propose hierarchical clustering as a solution to the detection of product duplicates for more than two Web shops. Hierarchical clustering (using a bottom-up, single-linkage approach) fits the concept of initially treating each product as separate, and incrementally clustering products that are similar. We also argue that certain requirements, such as not clustering products within a Web shop, which is assumed to only contain different products, are well- supported by the hierarchical clustering through implementation of heuristics of thresholds and rules in the iterations of the clustering procedure.

In this paper, we also aim to both extend and improve a state-of-the-art duplicate detection algorithm for products on the Web, called the Hybrid Similarity Method [3]. This method builds on a procedure that uses model words in the product titles to detect duplicates, called the Title Model Words Method [15], and is designed to cluster products from only two Web shops. We adapt these methods to calculate dissimilarity values between all products, needed for hierarchical clustering. Additionally, we make improvements through the use of q-grams, to deal with typographical errors and abbreviations, as a different similarity measure between strings, checking products titles for different brands that are manually provided, and using the Title Model Words Method in a more dynamic and less strict way.

The structure of the paper is as follows. First, we discuss existing literature that is related to the topic of product duplicate detection on the Web in Sect. 2. In Sect. 3, we describe a solution to cluster duplicate products dealing with multiple Web shops and based on this solution we present our Multi-component Similarity Method. Next, we evaluate its performance relative to the original Hybrid Similarity Method and two other benchmark models in Sect. 4. In Sect. 5, we conclude and identify possible further research.

2. RELATED WORK

Although the field of duplicate detection on the Web is relatively young, many algorithms for duplicate detection can be found in the literature for databases and information networks. Because the problem of duplicate detection can be reduced to a problem of obtaining a similarity for every pair of objects, the literature goes back to 1983's paper of Salton and Mcgill [11], where a similarity-based information retrieval system is introduced. More recently, the method from [5] was introduced, where trainable measures of textual similarities were used to improve the process of finding duplicate records.

In this paper we extend existing methods for product duplicate detection on the Web to overcome some of their shortfalls. One method we build on is the Title Model Words Method (TMWM) [15], which we also use to benchmark our approach. This method uses the so-called model words extracted from the title of product descriptions on the Web to perform duplicate detection. First, the cosine similarity of two product names is calculated. A threshold α is used to determine if the considered products are duplicates. If the products are not duplicates according to the cosine similarity, the method continues by extracting model words for the products under consideration. Model words are defined as words consisting of both numeric and non-numeric tokens. When the non-numeric part is approximately the same while the numeric part is not, the two products are different. Here, approximately the same means that the Levenshtein distance is smaller than a threshold of 0.5. When the model words match, we might have duplicate products. An initial product name similarity is calculated as a weighted average of the cosine similarity and the average Levenshtein similarity. If the two products' model words contain at least one pair where the non-numeric part is approximately the same and the numeric part is equal, the product name similarity is updated to a weighted average of the initial product name similarity and the average Levenshtein similarity of the involved model words. Based on this final similarity and a threshold ε, we conclude whether the two products are duplicates.

Another baseline method we use is based on the Term Frequency – Inverse Document Frequency method (TF-IDF) [11]. This is a well-known method in the offline duplicate detection domain [8]. The term frequency is defined as the number of times a word occurs in a document. The inverse document frequency measures whether this word is common or rare across all documents. The Inverse Document Frequency is defined as the logarithm of the number of documents minus the logarithm of the number of documents containing the word. The TF-IDF value is defined as the term frequency divided by the inverse document frequency. We compute the TF-IDF value for each unique word that occurs in the product attribute values. These values are stored in a vector and we use the cosine similarity of these vectors to obtain a similarity matrix for all combinations of products. This similarity matrix is transformed into a dissimilarity matrix on which agglomerative hierarchical clustering is applied. We use an adjusted single-linkage approach, as explained in the next section, and combine clusters until the dissimilarity value is higher than a predefined threshold value ε of 0.5.

The Hybrid Similarity Method (HSM) in [3] builds on TMWM of [15] by including additional information given by the product attributes. As a first step, TMWM is used to determine if the two products under consideration are duplicates based on the product title. If this is not the case, a new similarity measure is calculated consisting of two parts. First the attribute keys of both products are compared to check for similar keys, based on a chosen measurement. For the Key-Value Pairs (KVP) that are found, if any, an average similarity is calculated between the corresponding attribute values. Second, for all KVP where no matching keys were found, the model words from the attributes values are extracted. Next, the percentage of matching model words between the sets of model words of the two products is computed. The similarity is calculated as the weighted average of the average similarity of the values of the matched keys and the percentage of matching model words from the values of the keys that did not have a match.

Although HSM takes away most of the shortcomings of TMWM, HSM has its own difficulties. One shortcoming of HSM is that its solution lacks invariance in the order of product comparison. When some product A from the second Web shop is considered to be a duplicate of some product C of the first Web shop, they are immediately clustered. If instead a different product B from the second Web shop would have a better match to product C, but is considered later on in the algorithm, it will therefore not be clustered to product C because its cluster is full.

Furthermore, the generality of the model comes at a cost of precision. We argue that a more specific approach (for our domain of TV's) can result in a boost in performance. At the same time HSM is not so general when it comes to the number of Web shops it is able to compare, namely two. We instead propose an algorithm that is not limited to a given amount of Web shops.

Last, we argue that the use of TMWM in HSM is too restrictive. It is applied as a definite first step of duplicate detection in the algorithm. When duplicates are indeed detected, the algorithm directly proceeds to the next product combination. However, in the evaluation of the algorithms [3], it is shown that TMWM has only a precision of 0.556, whereas HSM has a precision of 0.741. TMWM thus declares a false duplicate in 44% of the cases, and HSM further improves this to 26% for the set of products declared non-duplicates by TMWM. Although its parameters are optimized with stricter thresholds when applied in HSM, we argue that using TMWM as a definitive positive-duplicate detection step could in fact be detrimental to HSM's precision. Instead, we propose to use TMWM as an additional similarity measure in the algorithm.

3. METHODS

In this section, we present several methods that address the issues described in the previous section. First, we discuss the handling of multiple Web shops in the Multi-component Similarity Method (MSM) through hierarchical clustering in Sect. 3.1. Second, we propose an extended version of HSM [3] in Sect. 3.2.

3.1 Hierarchical Clustering

HSM could be extended to cluster products of multiple Web shops by subsequently clustering all products of a shop to existing clusters, starting with each product of the first Web shop in its own cluster. However, we argue that the solution would not be invariant in the chosen order of Web shops, for the same reason as given in the previous section. Therefore, we apply a clustering method. As we want to compare HSM to MSM and other baseline methods in a fair way, we apply the clustering method to all models.

Two popular methods are K-means and hierarchical clustering [12]. K-means aims at partitioning all observations into k sets. Since we do not know how many duplicate products there are, k is unknown. We apply hierarchical clustering, which does not impose such prior information. The idea of hierarchical clustering is to construct a whole range of cluster solutions, where going from a solution of $K + 1$ to K clusters, two of the $K + 1$ clusters must be merged.

In this way, a hierarchy of clusters is obtained. There are two types of hierarchical clustering: agglomerative (bottom-up approach) and divisive (top-down approach). A natural way to cluster the products is the so-called agglomerative approach, which starts with each product in a separate cluster. We merge the two closest clusters C_i and C_j based on the dissimilarities between the products. We use a criterion based on minimum or single linkage clustering. We define the dissimilarities between the new cluster C_n and the other clusters C_k as

$$d_{C_n,C_k} = \min(d_{C_i,C_k}, d_{C_j,C_k}) \qquad (1)$$

with $d_{C_n,C_k} = \infty$ if $\max(d_{C_i,C_k}, d_{C_j,C_k}) = \infty$.

We keep on merging until the minimum dissimilarity between clusters is higher than a predefined threshold ε. Other clustering criteria are maximum or complete linkage, and average linkage clustering. A disadvantage of single linkage is the risk of cluster "chaining", however, since we have multiple components of similarities in the models, it is possible that in a cluster of products A, B and C, the pair A and B matches highly only on the title, while B and C only match highly on the product features. It is then possible that A and C have few matching similarities, while they are still duplicates. B is then the linking product between A and C, making single linkage ideal for this situation.

Note that this value of ε is analogous to the earlier threshold δ of HSM as in [3], where two products are only eligible as best clustering option if their similarity is larger than δ, in our method smaller than dissimilarity value $\varepsilon = 1 - \delta$. In TMWM, this ε is analogous to $1 - \beta$ because only title-similarities of either 1 or greater than β are considered for clustering. These analogies show why hierarchical clustering is suitable for extending the existing methods to handle multiple Web shops, while at the same time keeping their structure intact.

3.2 Multi-component Similarity Method

As mentioned before, our method builds on HSM [3]. A serious disadvantage of the model as previously defined, is that it can only handle comparisons between two Web shops. We have used hierarchical clustering to overcome this drawback. To be able to apply a hierarchical clustering method a dissimilarity measure between every pair of products is needed. Therefore we adjust and extend the algorithm in the following manner.

HSM starts by assigning all products of the first Web shop to their own clusters, and loops for each product of the second Web shop over all these products to cluster the products with the highest similarity. Our method, however, does not cluster products during these iterations. Instead, we loop over all product combinations and calculate a similarity value, that is later transformed into a dissimilarity value for hierarchical clustering. Since similarity values are calculated for textual content we decided to pre-process the product descriptions by removing all common characters like comma's, slashes, and white space. In this way, we remove any noise from the data. Furthermore, each capital letter is replaced by its lowercase counterpart.

We assume that there are no duplicate products within a Web shop. This is an assumption that is made for all other models as well. We therefore start our algorithm by checking for the pair of products under consideration if their shop names are equal. If this is the case, we assign a dissimilarity of positive infinity.

The pseudo-code of our algorithm is given in Algorithm 1. As mentioned in Sect. 2 we aim at an improved precision by giving up some of the HSM's generality. This is translated in our algorithm in the form of the *diffBrand()* function, which uses a manually created list of the most common brand names. By including this list we are able to check if two products are from different brands. A dissimilarity value of positive infinity is assigned to prevent the clustering of these products. The reasoning behind this approach is that products produced by different companies will not be duplicates. Although creating this list is done manually, including it as an input variable is an elegant way of providing the algorithm with specific task information. Also, small lists of brand names for different product types can be easily found on the Web [16].

Our proposed similarity function consists of three parts. The first part uses the KVP of the products that are not assigned a dissimilarity value by using the brand list. The keys of these products are compared. Where HSM uses a cosine- and a Jaro-Winkler measure to compare two keys, we decided to use overlapping q-grams as a similarity measure. The cosine similarity is sensitive to misspellings, while the Jaro-Winkler is token-based. The overlapping q-gram measure uses tokens of q characters, in our case q = 3, taken from a sliding window from left to right of the string, inserting additional dummy characters at the start and end of the string. As a result of the overlapping tokens, partial matches of words still contribute to the similarity, which accounts for words with misspellings or abbreviations. In this way, it forms a tradeoff between the cosine- and Jaro-Winkler similarity. If the keys match, the q-gram similarity between the corresponding values is calculated, and the similarity is added with a weight to the similarity of the two products. We calculate the q-gram similarity value between two strings $s1$ and $s2$, with their amount of tokens $n1$ and $n2$, respectively, by dividing the amount of (one-on-one) matching tokens by the total amount of tokens. This is equivalent to:

$$\frac{n1 + n2 - qGramDistance(s1, s2)}{n1 + n2},$$

where $qGramDistance(s1, s2)$ is the number of different q-gram occurrences in $s1$ and $s2$ [14].

The KVP that had no key-match contribute to the second part of our similarity measure. We extract model words from the values of these KVP and calculate the percentage of matching words. This procedure is the same as in HSM.

The final part of the similarity is obtained by calculating a similarity based on TMWM. One of the weak features of HSM was the restrictiveness of the implementation of TMWM, as mentioned in Sect. 2. We have implemented a weighted *titleSim* as the third part of our similarity. TMWM is adjusted to return similarities instead of a clustering of products. When the original TMWM clustered two products based on the cosine similarity measure of the product names, it now returns a similarity of 1, and when products were eligible for clustering based on the model words, it now returns the calculated similarity. If, however, products were not clustered, the adjusted TMWM function returns a similarity of -1 which is not used in the final calculation of our method. Therefore we calculate the similarity as:

$$hSim = \theta_1 * avgSim + \theta_2 * mwPerc + \mu * titleSim * \mathbf{1}_{\{titleSim \neq -1\}}, \quad (2)$$

where $\mathbf{1}_{\{\bullet\}}$ is an indicator function. The parameters θ_1 and θ_2 together form the remaining weight of $1 - \mu$. Of this remaining weight, θ_1 is defined to be the proportion of matching keys relative to the lowest amount of KVP of both products.

After transforming the similarities from (2) to dissimilarities ($1 - hSim$) we run our clustering algorithm (i.e., on line 43 the function hClustering($disSim$, ε)) where the matrix of dissimilarities is an input alongside the dissimilarity threshold ε. The algorithm stops when the smallest dissimilarity between the built clusters becomes larger than the dissimilarity threshold ε.

4. EVALUATION

In this section, we compare the performance of MSM with the performance of the benchmark methods TF-IDF, TMWM, and HSM. We apply the methods to a dataset consisting of 1629 TV's, of which 1262 unique, obtained from four different Web shops: Amazon.com [1], Newegg.com [10], Best-Buy.com [4] and TheNerds.net [7]. The data of these shops consists of 163, 672, 744, and 20 TV's, respectively. The products have on average 29 key-value pairs as stated features in their product descriptions.

The interpretation of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) in this work is slightly different from usual. The dataset contains 409 duplicate pairs of TV's. If two duplicate TV's (one pair) from different Web shops are clustered correctly, then this is counted as one true positive. If two TV's that are not duplicates are clustered, this is counted as one false positive. TV's that are not duplicates and that are not clustered are counted as true negative. If two duplicate TV's are not clustered, this is counted as one false negative. When a cluster contains more than two TV's, all pairs are checked and counted. So a cluster of three TV's containing two duplicate TV's and one TV that is not a duplicate of the other two, is counted as one true positive and two false positives.

Since TF-IDF, TMWM, and HSM are built on handling just two Web shops, we can not make a fair comparison. For this reason we have rebuilt these baseline methods. Instead of a clustering solution, the adjusted methods provide a matrix of dissimilarities which in turn is processed by our hierarchical clustering algorithm. These dissimilarities fall in one of three categories: zero dissimilarity when the algorithm detects a duplicate, positive infinity when the algorithm indicates two products are not duplicates, and a dissimilarity $\in (0, 1)$, otherwise.

As we do not want the performance results to be specific for one given dataset, we apply the bagging bootstrapping procedure check [6] to provide robust performance comparisons between the methods. In each bootstrapping iteration, we randomly sample approximately 63% of the data with replacement as the training set, and use the rest as the evaluation set, where the ratio of duplicates versus non-duplicates of the original dataset is maintained.

For each sample, the training set is first used to identify which particular parameter set has the highest performance in terms of F_1-measure by performing a grid search. Next, the obtained best parameter set is used on the evaluation set,

Algorithm 1 Multi-component Similarity Method

The input: Set $S = \{S_1, ..., S_K\}$, where $S_k \in S$ contains all products of Web shop k. Parameters α and β are threshold values for TMWM, γ is the threshold similarity for two keys to be considered equal, ε is the dissimilarity threshold for hierarchical clustering, and μ is the fixed weight of the TMWM similarity, if it returns a duplicate. Furthermore, the following functions are used:

- calcSim(q, r) the q-gram similarity for strings q and r;
- sameShop(p_i, p_j) true if shop is same for products i and j;
- diffBrand(p_i, p_j) true if brands of products i and j are different;
- key(q) the key from key-value pair (KVP) q; value(q) returns the value from KVP q;
- exMW(p) all model words from the values of the attributes from product p;
- mw(C, D) percentage of matching model words from two sets of model words;
- TMWMSim(p_i, p_j, α, β) the TMWM similarity between the products i and j using the parameters α and β;
- minFeatures(p_i, p_j) the minimum of the number of product features that product i and j contain;
- hClustering($dist$, ε) returns the clusters.

```
1:  $n$ = total number of products
2:  $dist = \mathbb{R}^{n \times n}$
3:  for all $k = 1, \ldots, K$ do
4:    for all $p_i \in S_k$ do
5:      for all $p_j \in S \setminus S_k$ do
6:        if sameShop($p_i, p_j$) or diffBrand($p_i, p_j$) then
7:          $dist_{p_i,p_j} = \infty$
8:        else
9:          $sim = 0$
10:         $avgSim = 0$
11:         $m = 0$ {number of matches}
12:         $w = 0$ {weight of matches}
13:         $nmk_i = KVP_i$ {non-matching keys of $p_i$}
14:         $nmk_j = KVP_j$ {non-matching keys of $p_j$}
15:         for all KVP $q$ in $KVP_i$ do
16:           for all KVP $r$ in $KVP_j$ do
17:             $keySim$ = calcSim($key(q), key(r)$)
18:             if $keySim > \gamma$ then
19:               $valueSim$ = calcSim($value(q), value(r)$)
20:               $weight = keySim$
21:               $sim = sim + weight * valueSim$
22:               $m = m + 1$
23:               $w = w + weight$
24:               $nmk_i = nmk_i - q$
25:               $nmk_j = nmk_j - r$
26:             end if
27:           end for
28:         end for
29:         if $w > 0$ then
30:           avgSim = $\frac{sim}{w}$
31:         end if
32:         $mwPerc$ = mw(exMW($nmk_i$),exMW($nmk_j$))
33:         $titleSim$ = TMWMSim($p_i$,$p_j$,$\alpha$, $\beta$)
34:         if $titleSim = -1$ then
35:           $\theta_1 = m/minFeatures(p_i, p_j)$
36:           $\theta_2 = 1 - \theta_1$
37:           $hSim = \theta_1 * avgSim + \theta_2 * mwPerc$
38:         else
39:           $\theta_1 = (1 - \mu) \cdot \frac{m}{minFeatures(p_i, p_j)}$
40:           $\theta_2 = 1 - \mu - \theta_1$
41:           $hSim = \theta_1 \cdot avgSim + \theta_2 \cdot mwPerc + \mu * titleSim$
42:         end if
43:         $dist_{p_i,p_j} = 1 - hSim$ {transform to dissimilarity}
44:       end if
45:     end for
46:   end for
47: end for
48: return hClustering($dist$, $\varepsilon$) {return the found clusters}
```

generating a value of the F_1-measure, precision, and recall. We use 50 bootstrap-samples to provide accurate estimates of these performance measures. These 50 bootstrap-samples are the same for all methods. Also, we apply a Wilcoxon signed rank test [17] to check if there are significant differences between the methods in terms of F_1-measure.

Again, for a fair comparison between the methods, we apply hierarchical clustering to all methods to allow for clustering of multiple products, instead of extending the immediate sequential clustering as done in HSM. We do stress, however, that hierarchical clustering requires dissimilarities between all products among different Web shops, causing extremely long running times when a lot of different parameter sets need to be checked, in combination with 50 bootstraps. Therefore, optimization by remembering similarity values between strings and, where possible, between products is necessary.

Our framework to run bootstraps with the algorithm is set up in such a way that, with a multicore system, each core runs the algorithm for all the bootstrap samples of a single parameter set. Figure 1 shows an illustration of our framework. Especially with a lot of bootstraps, calculating dissimilarities between products in each new sample for hierarchical clustering would be a waste of CPU time. Instead, we calculate the dissimilarities between all possible combinations of products of the different Web shops, 788,000 pairs for the dataset we use. This means that for each bootstrap we perform, all needed dissimilarities are known. Note, however, that our total set of products is still small enough to gain an advantage out of this method. When the amount and size of the bootstraps are relatively small compared to a much larger set of products, this method could use more running time because of an explosion in the amount of combinations of the total product set.

Since the choice of ε for hierarchical clustering has no influence on the product dissimilarities, we can achieve additional gains in running time by storing the dissimilarities of the parameter set (without considering ε) in a local cache for the CPU core that is assigned to that parameter set. We can then let that core run the algorithm on all the parameter sets while only varying ε, without having to recalculate all dissimilarities. For example, when using 10 test values for ε, this already reduces running time by a factor of close to 10.

Additional reductions in running time can be achieved in the calculation of product dissimilarities itself. Because products in the same category use the same terminology, we can pre-calculate the similarity values between all strings that are used throughout the algorithm. We store these in a global cache that is used by all cores to calculate their initial product dissimilarities for the bootstraps. Again, the reduction in running time can vary, in this case depending on the algorithm: when a lot of calculations of similarities between strings in the product description are occluded, for example through conditions that are met early on in the algorithm, many similarities would never be used. In our case, however, we have to calculate 23 million similarity values, with a running time of only 10 minutes on a single core of a standard PC. We used a PC with a 2.4GHz quad core CPU and 4GB of RAM.

With the framework optimizations above, we are able to reduce the running time on a standard PC, theoretically, from a few hundred years to a few hours. Note that storing the local- and global caches to disk can also be convenient when it would be necessary to re-run the program in case of a system-crash or for testing purposes. An alternative, less favorable, way to reduce running time would be focused on reducing the amount of data that is considered, as done in [18].

In this section, we first discuss the results of the three benchmark methods TF-IDF, TMWM, and HSM in Sect. 4.1, 4.2 and 4.3, respectively. Next, we discuss the results of our MSM in Sect. 4.4, followed by a comparison with the results of the benchmark methods in Sect. 4.5.

4.1 TF-IDF Method

The TF-IDF method uses the parameter δ to decide whether two products are duplicates. In the new framework that is capable of handling multiple Web shops a dissimilarity measure is used instead of a similarity measure. For this reason δ is converted to ε where $\varepsilon = 1 - \delta$. The TF-IDF algorithm was trained using a grid search for the values of ε ranging from 0.1 to 0.9 with steps of 0.1. On all 50 training sets the value of 0.9 performed the best. Applying this value on the 50 test sets resulted in a F_1-measure of 0.335 on average. The average precision was 0.337, and the average recall was 0.334.

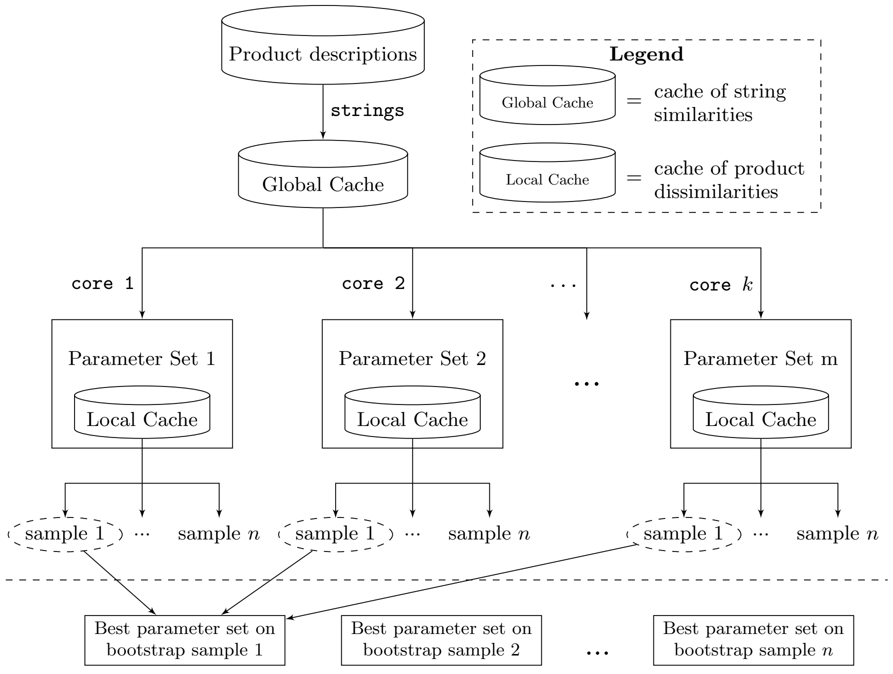
4.2 Title Model Words Method

TMWM uses two parameters α and ε. The parameter α is used as a threshold for the cosine similarity of the product titles. The higher α is, the more similar the titles have to be in order that the associated products are considered duplicates. If the cosine similarity of two titles is not higher than α, the algorithm calculates a weighted similarity and model words as explained in Sect. 2. In the new framework, all similarities are converted to dissimilarities to be used in the hierarchical clustering. As with the TF- IDF model, the parameter ε is used as threshold to compare the dissimilarity between two products.

For both parameters the training was done using a grid search ranging from 0.1 to 0.9 with steps of 0.1. Table 1 shows the mean and standard deviation from the best values found using the 50 training sets. These values are lower than the values found in [3], where TMWM is used for two Web shops. We suspect this is due to the method of clustering. In the two Web shops case, two products are clustered as soon as the similarity is higher than one of the parameters. Low values for the parameters are undesirable because this can result in many false positives. By using hierarchical clustering this problem is overcome, all similarities (or dissimilarities in our case) are calculated before products are clustered. Then the most similar products are clustered and since we do not allow the clustering of products from the same Web shop, the results show that lower values for parameters are optimal. Using the best values of each of the 50 training sets on its corresponding test set resulted in an average F_1-measure of 0.298. The average precision was 0.349

Table 1: Means and standard deviations of the best values over 50 training sets for TMWM

	Mean	Standard deviation
α	0.622	0.084
ε	0.356	0.081



Product descriptions
strings
Global Cache
Legend
Global Cache
= cache of string similarities
Local Cache
= cache of product dissimilarities
core 1
core 2
core k
Parameter Set 1
Parameter Set 2
Parameter Set m
Local Cache
sample 1
sample n
Best parameter set on bootstrap sample 1
Best parameter set on bootstrap sample 2
Best parameter set on bootstrap sample n

Figure 1: Overview of the caching framework, with m parameter sets, k processing cores, and n bootstrap samples.

Table 2: Means and standard deviations of the best values over 50 training sets for HSM

	Mean	Standard deviation
α	0.612	0.033
β	0.240	0.049
γ	0.776	0.056
ε	0.824	0.102

and the average recall was 0.309. Compared to the results in [3], we find more false positives (our precision is lower) and we find less false negatives (our recall is higher). This is expected when using lower values of α and ε.

4.3 The Hybrid Similarity Method

The Hybrid Similarity Method uses 4 parameters. The first two parameters are the parameters of TMWM: α and β, where β is equal to the ε of TMWM as defined in Sect. 2. For all parameters we have used the same grid search ranging from 0.1 to 0.9 with steps of 0.1. For most training sets, the best value for α was 0.6, for a few sets it was 0.7. This value is comparable with the value found for the standalone TMWM. The best value for β was either 0.2 or 0.3. This parameter is slightly stricter than the value in the standalone TMWM model. The third parameter γ is the threshold for deciding whether two keys are equal. The best value for γ was not very stable for the 50 training sets, it was 0.7 or 0.8 most of the time, but was 0.6 or 0.9 for a few training sets. The fourth parameter is ε. This parameter has the same function as in TF-IDF and TMWM and is the threshold for the clustering algorithm. When ε is higher, more products are clustered, since a higher dissimilarity is allowed. While training the best value for ε, it was either 0.8 or 0.9. This means a small similarity in keys and model words is enough to cluster two products. The mean and standard deviations of the 4 parameters are found in Table 2. Using the best values of each of the 50 training sets on its corresponding test set resulted in an average F_1-measure of 0.287. The average precision was 0.237 and the average recall was 0.381. These results are surprising to us, because HSM is outperformed by both TF-IDF and TMWM. We compare the different models in more detail in Sect. 4.5.

4.4 Multi-component Similarity Method

MSM uses 5 parameters. For all parameters in this model we used a grid search ranging from 0 to 1 with steps of 0.1 to find the best value for each of the 50 training sets. Table 3 shows the means and standard deviations of the parameters of MSM.

The first two parameters are the parameters of TMWM: α and β. α has the same interpretation as in TMWM and HSM. The best value for α was 0.6 for all training sets but one, where the best value was 0.7. This value is comparable to the value found by TMWM and HSM. β is the threshold to decide (if no match based on the cosine-similarity occurred) whether the model continues using the similarity

Table 3: Means and standard deviations of the best values over 50 training sets for MSM

	Mean	Standard deviation
α	0.602	0.014
β	0.000	0.000
γ	0.756	0.101
ε	0.522	0.082
μ	0.650	0.168

Table 4: Average performance measures of all methods

Method	F_1-measure	precision	recall
TF-IDF	0.335	0.337	0.334
TMWM	0.298	0.349	0.309
HSM	0.287	0.237	0.381
MSM	0.475	0.445	0.512

(*titleSim*) of the TMWM model or sets this similarity to -1. When *titleSim* is -1, the model does not use the similarity found by TMWM in the computation of the final similarity (*hSim*). The best value for β for all 50 training sets was 0. This has an important implication: *titleSim* is never set to -1 which implies that the value found by TMWM is always used in computing *hSim*, and titles with low similarity (i.e., close to 0) are very informative by lowering the final product similarity.

The third parameter γ has the same function as in HSM, it is the threshold for deciding whether two keys are equal. The best values found for the 50 training sets show the same behavior as in HSM. For most training sets the best value was either 0.7 or 0.8, but for a few sets the best value was either 0.6 or 0.9.

The fourth parameter ε is again used for the clustering algorithm. The best value of ε was also not very stable for the 50 training sets. The best values found were mostly 0.4, 0.5, or 0.6, for a few sets it was 0.7. These values for ε imply that the clustering algorithm requires a lower dissimilarity to cluster two TV's than in HSM.

The fifth parameter is μ, which is the weight given to the similarity found by TMWM *titleSim* in the computation of *hSim*. Since the optimal value of β was always equal to 0 for every training set, μ is an important parameter. The value of μ was unstable for the 50 training sets. The best values found were mostly in the range from 0.5 to 0.7, but for some sets the value was 0.3, 0.4, or 0.9. With an average optimal value of μ of 0.650, we can conclude that the TMWM's similarity value played the largest role in the final similarity value, compared to the similarities of the product attributes.

Using the best values of each of the 50 training sets on their corresponding test set, resulted in an average F_1-measure of 0.475. The average precision was 0.445, and the average recall was 0.512. All values of the performance measures are clearly higher than the performance values of the measures found for the three baseline methods: TF-IDF, TMWM, and HSM. In the next section we analyze these performance measures to determine whether they are statistically significant higher for MSM.

4.5 Comparison of All Methods

The main performance measure we use to assess whether one method performs better than other methods is the F_1-measure. We also looked at the precision and recall of all methods. As can be seen in Table 4, MSM scores better on these average measures than all three baseline methods: TF-IDF, TMWM, and HSM. We use Wilcoxon signed rank tests to assess whether the differences in the F_1-measure are statistically significant. We use a significance level of 0.05 for these tests. In Table 5, the one-sided p-values of the Wilcoxon signed rank test are shown. This table clearly shows that the value of the F_1-measure for MSM is statistically significantly higher than the F_1-measure of all three baseline methods, even at a 0.001 significance level. To assess how big the impact is of using the function *diffBrand(i,j)*, we ran HSM where we only added this function. The average F_1-measure of HSM with *diffBrand(i,j)* was now 0.391. This indicates that this function contributed to more than half of the increase in performance of MSM. It is remarkable that HSM, contrary to [3], is now the worst performing method in terms of F_1 measure. The application to more than two Web shops could play a role in this. The method also has an even lower F_1 score than TMWM it was built on. A possible explanation could be over-fitting on the training datasets. Although our proposed MSM had the highest amount of parameters of all methods and still has the highest performance in terms of F_1 measure, the good performance of the TF-IDF method, which has only one parameter in our application, is in line with this explanation.

Where in [3] the TF-IDF method had the worst performance, it performed quite well as second-best in our setting of multiple Web shops. This could be due to the fact that the TF-IDF method takes into account the whole set of words in the dataset of all Web shops, where the other methods do not.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a method for the challenging problem of product duplicate detection on the Web. Our proposed method builds on the state-of-the-art Hybrid Similarity Method [3], which extends the Title Model Words Method [15] by making use of a similarity on key features of the products. We further extended this framework on several aspects.

First, we introduced brand-comparison of two product titles to exclude duplicates of different brands, using a manually created list of brands. In every industry the number of brands is limited, making it an excellent tool for duplicate detection. Although this forms an additional manual learning step for the algorithm, we think that maintaining this list can be easily done, and perhaps automated, through lists found on the Web. We showed this is worth considering, as over half of the improvement in the F_1 score is due to this extension. Second, we introduced the use of q-grams with token length $q = 3$. Contrary to the word-based cosine similarity in the Hybrid Similarity Method, the q-gram similarity supports partial word-matching, making it robust in a context of short, informative descriptions with abbreviations on Web shops. Third, where the Hybrid Similarity Method used the Title Model Words Method as a definitive first clustering step, we instead extracted a similarity value from it and used it less strictly in the clustering procedure. Its optimal weight in the final similarity value showed that

Table 5: One-sided p-values of the Wilcoxon signed rank test ($H_0 : \mu_{row} = \mu_{column}$ versus $H_A : \mu_{row} < \mu_{column}$).

p-value	TF-IDF	TMWM	HSM	MSM
TF-IDF	x	1.000	1.000	0.000
TMWM	0.000	x	0.999	0.000
HSM	0.000	0.001	x	0.000
MSM	1.000	1.000	1.000	x

it played the largest role, compared to the similarities of the product attributes.

To deal with multiple Web shops, we proposed a single-linkage hierarchical clustering approach. Contrary to the Hybrid Similarity Method, our proposed algorithm does not make assumptions on the order of products to cluster. Although we solved resulting scalability issues by preventing redundant calculations through caching, the algorithm now suffers from a more severe exponential growth in the amount of product dissimilarities. Further research into scalability and efficient parameter-selection is necessary, for example through locality-sensitive hashing [2, 9].

Applying our new method to a dataset of TV's of four Web shops revealed that it significantly outperforms three baseline methods: the Hybrid Similarity Method, Title Model Words Method, and the well-known TF-IDF method, with an F_1 score of 0.475, compared to 0.287, 0.298, and 0.335, respectively. Based on the surprisingly good performance of the TF-IDF method, we believe it is worthwhile to do further research into the use of information from the whole dataset of products when comparing two individual products. One option would be to introduce this TF-IDF similarity between two products as an additional component in the between-product similarity of MSM.

Acknowledgment

The authors of this paper are partially supported by the Netherlands Organisation for Scientific Research (NWO), under the Mozaiek project *'Semantic Web Enhanced Product Search (SWEPS)'* (project 017.007.142), and the Dutch national program COMMIT.

6. REFERENCES

[1] Amazon.com, Inc. http://www.amazon.com.

[2] A. Andoni and P. Indyk. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[3] M. Bakker, F. Frasincar, and D. Vandic. A Hybrid Model Words-Driven Approach for Web Product Duplicate Detection. In *Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE 2003)*, volume 7908 of *Lecture Notes in Computer Science*, pages 149–161, 2013.

[4] Best Buy Co., Inc. http://www.bestbuy.com.

[5] M. Bilenko and R. Mooney. 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)*, pages 39–48, 2003.

[6] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[7] Computer Nerds International, Inc. http://www.thenerds.net.

[8] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[9] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB 1999)*, pages 518–529. Morgan Kaufmann Publishers Inc., 1999.

[10] Newegg Inc. http://www.newegg.com.

[11] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.

[12] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson International Edition, 1st edition, 2006.

[13] I. Thomas, W. Davie, and D. Weidenhamer. Quarterly Retail E-Commerce Sales 3d Quarter 2013. *U.S. Census Bureau News*, 2013.

[14] E. Ukkonen. Approximate String-Matching with Q-grams and Maximal Matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[15] D. Vandic, J.-W. van Dam, and F. Frasincar. Faceted Product Search Powered by the Semantic Web. *Decision Support Systems*, 53(3):425–437, 2012.

[16] Wikipedia: The free encyclopedia. http://wikipedia.org/wiki/List_of_television_manufacturers.

[17] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[18] C. Xiao, W. Wang, X. Lin, J. Yu, and G. Wang. Efficient Similarity Joins for Near Duplicate Detection. *ACM Transactions on Database Systems (TODS)*, 36(3):A:1– A:40, 2011.