Inf Retrieval J (2018) 21:253–272
https://doi.org/10.1007/s10791-017-9320-z

# Website replica detection with distant supervision

**Cristiano Carvalho[1] · Edleno Silva de Moura[2] · Adriano Veloso[1] · Nivio Ziviani[1,3]**

Received: 31 October 2016 / Accepted: 9 October 2017 / Published online: 29 November 2017


**Abstract** Duplicate content on the Web occurs within the same website or across multiple websites. The latter is mainly associated with the existence of website replicas—sites that are perceptibly similar. Replication may be accidental, intentional or malicious, but no matter the reason, search engines suffer greatly either from unnecessarily storing and moving duplicate data, or from providing search results that do not offer real value to the users. In this paper, we model the detection of website replicas as a pairwise classification problem with distant supervision. That is, (heuristically) finding obvious replica and non-replica cases is trivial, but learning effective classifiers requires a representative set of non-obvious labeled examples, which are hard to obtain. We employ efficient Expectation-Maximization (EM) algorithms in order to find non-obvious examples from obvious ones, enlarging the training-set and improving the classifiers iteratively. Our classifiers employ association rules, being thus incrementally updated as the EM process iterates, making our algorithms time-efficient. Experiments show that: (1) replicas are fully eliminated at a false-positive rate lower than 0.005, incurring in + 19% reduction in the number of duplicate URLs, (2) reduction increases to + 21% by using our site-level algorithms in conjunction with existing URL-level algorithms, and (3) our classifiers are more than two orders of magnitude faster than semi-supervised alternative solutions.

✉ Adriano Veloso
adrianov@dcc.ufmg.br

Cristiano Carvalho
cristiano@dcc.ufmg.br

Edleno Silva de Moura
edleno@dcc.ufam.br

Nivio Ziviani
nivio@dcc.ufmg.br

[1] Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

[2] Department of Computer Science, Universidade Federal do Amazonas, Manaus, Brazil

[3] Kunumi, Belo Horizonte, Brazil

**Keywords** Replica detection · Distant Supervision · Expectation-Maximization

## 1 Introduction

Detecting and removing duplicate URLs is of paramount importance to search engines. According to Fetterly et al. (2003), around 29% of the Web is estimated to be composed of near-duplicate URLs[1]—documents having similar content but referenced by different strings. By other estimates,[2] as many as 30% of the pages on the Web are duplicates of other pages. This stresses critical components of search engines, including crawling, indexing, ranking, and presentation.

Duplicate (or near-duplicate) URLs can be divided into two categories, depending on whether they occur within the same website and have the same hostname (intra-site duplicate URLs), or across multiple websites (inter-site duplicate URLs). The inter-site duplicate URLs requires site-level algorithms that process URLs across different websites, while the intra-site duplicate URLs requires URL-level algorithms that process URLs within the same website. While most of the existing algorithms consider the intra-site URL de-duplication scenario (Bar-Yossef et al. 2009; Dasgupta et al. 2008; Koppula et al. 2010), it is clear that a complete solution must also comprise algorithms for dealing with inter-site duplicate URLs.

In this paper, we deal with inter-site duplicate URLs by detecting and removing website replicas. The existence of inter-site duplicate URLs is mainly associated with the emergence of *website replicas*—different websites that are (perceptibly) similar in terms of content and structure. They exist mainly because (1) websites are pre-built and sold to multiple people, (2) websites move to another hosting company, (3) websites have both WWW and non-WWW versions indexed, (4) websites are mirrored for the sake of load balancing, or (5) multiple or similar copies of the website may increase chances of getting listed on search engines. Once detected, replicas may get penalized or even removed from the search results.

With more than one billion active websites in the Internet,[3] typical replica detection algorithms are impracticable, since they perform a full comparison involving all pairs of websites in the (possibly large) collection. Fortunately, clusters of URLs with similar content (or simply dup-clusters Agarwal et al. 2009; Yang and Callan 2006) can be found efficiently, revealing pairs of websites sharing some content and being therefore replica candidates. The number of replica candidates narrows down to a scale for which it is possible to apply more sophisticate detection approaches.

In here, we model the website replica detection task as a classification problem. In this case, a training-set is used to produce a classifier that relates features associated with pairs of websites to their likelihood of being replicas. The classifier is used to indicate which pairs are replicas in a test-set. The main advantage of the classification approach is that it avoids having to precisely define what is replica and what is not. Definitions of replica are based on the perceptual meaning of the websites, being fuzzy by nature, in the sense that it is necessary to consider not only the amount of duplicate content in order to characterize replication. In contrast, following the classification approach, we need only to provide sufficient training examples and the classifier automatically learns to differentiate pairs that are replicas from those that are not.

The main drawback of the classification approach is that it is based on training-sets with labeled pairs of websites. The annotation process necessary to create labeled data may be unfeasible due to the disproportionate number of negative examples. While it is hard to

[1] The terms “duplicate” and “near-duplicate” are used interchangeably.

[2] https://blog.seoprofiler.com/matt-cutts/.

[3] According to an October 2014 survey from netcraft.com.

separate a diversified and representative set of replica examples, finding obvious cases might be trivial, such as cases including WWW and non-WWW versions, as well as small segment variations in the hostname. This alternative to the manual labeling process is called Distant Supervision, and results in a limited set of positive replica examples plus a large amount of unlabeled data. We refer to this set of examples as PU data (i.e., positive + unlabeled data), and binary classifiers can be learned from PU data by simply considering unlabeled data as negative examples. However, this strategy leads to classifiers with poor performance due to a potentially large number of false-negative examples in PU data.

To overcome this problem we employ Expectation-Maximization (EM) algorithms[4] that refine the training-set iteratively (Dempster et al. 1977), by enlarging the set of positive examples from an initial set of obvious cases. For this, we use a partial classifier to evaluate the likelihood of an unlabeled example being positive or negative, and the process continues iteratively by changing the label of the ones that are likely to be positive—an operation called label transition—so that after some iterations the combination of labels is expected to converge to the one for which the observed data is most likely, and the final training-set is obtained.

The counterpart strategy of learning classifiers from negative examples plus unlabeled data (or simply, NU data) is also effective. That is, we can easily find obvious non-replica cases, and use EM algorithms in order to improve the training-set with non-obvious positive examples. We show that classifiers learned from PU and NU data lead to complementary results in the sense that the errors associated with the classifiers are not correlated. Thus, we propose effective approaches to combine the predictions performed by the two classifiers in order to further improve detection performance.

*Contributions and findings* The major contributions of this paper are:

- We propose algorithms to detect website replicas, which cause a significant number of duplicate URLs in the collection crawled by web search engines. We combine classifiers built from PU and NU data by exploiting a central concept of Economics—Pareto Efficiency (Palda 2011)—which greatly reduces the number of duplicate URLs. Our experiments reveal a 19% reduction in the number of duplicate URLs, after removing all website replicas, with a false-positive rate of 0.005.[5]
- Given the large number of unlabeled examples in our learning scenarios, the EM process may encompass millions of label-transition operations. Each transition operation modifies the training-set and a new classifier must be built. In order to keep execution times practical, we employ rule-based classifiers coupled with strategies to maintain it up-to-date incrementally until the EM process converges to the final training-set.
- A crucial issue of EM algorithms concerns the decision of whether or not performing the label-transition operation. We make use of an entropy-minimization approach (-Davis et al. 2012) to find the optimal threshold for each unlabeled example instead of using a single threshold which is applied to all unlabeled examples indistinctly. We extended this approach so that it is able to deal with both PU and NU data simultaneously.

[4] An Expectation-Maximization algorithm is a general approach to iterative computation of maximum-likelihood estimates when the observations can be viewed as incomplete data.

[5] These numbers were obtained by considering only replica candidates, and not all possible pairs of websites in the collection.

- Our classifiers employ low-cost features that do not fetch page content, and are learned on a demand-driven basis (i.e., a specific classifier is produced for each replica candidate). As a result, learning a classifier for a given candidate takes no more than 0.1 s, enabling the classifiers to be used at crawling time.
- We collected a large dataset comprising + 30 millions URLs associated with + 170,000 replica candidates. We expended significant time, effort and resources to obtain such dataset, which shall be made available at publication time.
- We evaluate the interplay between our algorithms which eliminate inter-site duplicate URLs by removing website replicas, and existing algorithms that eliminate intra-site duplicate URLs (Dasgupta et al. 2008). Specifically, applying these algorithms in conjunction leads to a more complete solution, enabling a reduction of + 21% in the size of the collection.

## 2 Related work

Many studies focused on characterizing and alleviating the large amount of duplicate data in the Web. Initial efforts have addressed the problem of detecting near duplicated documents, where the content of documents are compared, for example, by using shingles, that is, sets of contiguous terms of the documents (Ye et al. 2008).

Although very efficient, the use of textual content to detect duplicates at crawling time is very expensive. As a consequence, previous works have attempted to minimize the use of page content during the duplicate detection. In particular, Bharat and Broder (1999) presented a two-stage solution, where textual content is used only in the second phase. Specifically, replica candidates are first selected by syntactically comparing the URLs found in each server in order to determine if they are similar to some extent. In a second phase, each candidate is tested to determine if the corresponding websites are indeed replicas. This test consists on collecting samples of pages from each server to verify the occurrence of pages in common.

Another alternative for eliminating duplicate content is to detect DUST (Different URLs with Similar Text) pages. In this case, the problem is formulated as detecting duplicate content by examining only URLs. The first algorithm to adopt this strategy was DustBuster (Bar-Yossef et al. 2009), which removes DUST by finding rewrite rules able to transform a given URL into another URL likely to have similar content. Rules consist of sub-string substitutions learned from crawl logs or web logs. The work in Dasgupta et al. (2008) present another formalization of URL rewrite rules which is able to capture all previous rules found by DustBuster, and also more general patterns, such as the presence of irrelevant sub-strings, complex URL token transpositions and session-id parameters. Experiments showed a 60% reduction in the number of duplicate URLs.

Agarwal et al. (2009) extended the work in Dasgupta et al. (2008) to derive rules from samples of URLs, thus reducing the cost to infer such rules. Further, they used decision trees to learn a small number of higher precision rules to minimize the number of rules deployed to the crawler. They evaluated their algorithm in a set of 8 million URLs, achieving a 42% reduction using the top 9% most precise rules. Koppula et al. (2010) implemented their algorithm using a distributed framework and extended the URL and rule representations to include two additional patterns. They evaluated the method with 3 billion URLs.

Lei et al. (2010) proposed an algorithm in which a URL pattern tree is built from clusters of duplicated URLs. The algorithm was evaluated in a collection with 70 million URLs, achieving twice the reduction of DUST reported in Dasgupta et al. (2008), using 46% of the rules and consuming half of the learning time. Rodrigues et al. (2013) presented a multi-sequence alignment algorithm to derive rules to eliminate DUST. As with the previous algorithm, sample URLs are clustered by comparing their content (i.e., dup-clusters). They first align all the URLs in the dup-clusters obtaining a consensus pattern for each dup-cluster. Rules are then derived from these patterns, leading to improvements up to 54% in terms of reduction of duplicate URLs.

Bharat et al. (2000) introduced a new problem, the detection of near-duplicated websites without using page content. The detection is performed by computing only features extracted at crawl time, normally derived from the URL strings and linkage structure. In particular, the authors present algorithms for replica detection based on evidence derived from URL strings, IP addresses and link information.

Carvalho et al. (2007) extended Bharat et al. (2000) by taking advantage of the content of Web pages in a very efficient way. In particular, they propose an algorithm for replica detection called NormPaths, in which the norm of a page is used as a signature for its content. Using the norm implies no additional computation cost since that value was already calculated as a byproduct of the page indexing. As a consequence, NormPaths provides gains up to 47% in terms of precision when compared with Bharat et al. (2000). This algorithm is used as a baseline in this work and is described in Sect. 4.1.3.

Liu et al. (2003) study the problem of building text classifiers using positive and unlabeled examples. They propose an approach to solving the problem based on a biased formulation of SVM, and show experimentally that it is more accurate than the existing techniques. This algorithm is also used as a baseline in this work and is described in Sect. 4.1.3.

Cho et al. (2000) are concerned with a slightly different problem, which is to find duplicated web collections instead of site replicas. Note that finding duplicated collections includes finding replicated sites or fragments of replicated sites. Their approach first clusters pages with similar content and then expands these clusters such that they represent whole collections. The main disadvantage of this strategy is that it requires calculating the similarity between pages based on their content, an expensive task.

In this work, we address the problem of detecting website replicas at crawl time without using page content, as in Bharat et al. (2000), Carvalho et al. (2007), Dasgupta et al. (2008). The main difference between this work and the aforementioned ones is the use of classifiers to combine a diverse set of replication features. Expectation-Maximization algorithms are used in order to learn classifiers avoiding any manual labeling effort.

## 3 Algorithms

In this section, we present classification algorithms for detecting replica cases from a set of replica candidates. In this case, we are given as input a training-set (denoted as $\mathcal{D}$) which consists of examples of the form $\langle p, \ell \rangle$, where $p$ is a set of features associated with a pair of websites and $\ell \in \{\ominus, \oplus\}$ is a binary variable that specifies whether or not the corresponding websites should be considered as replicas. The training-set is used to produce a classifier that relates patterns in $p$ to the value of $\ell$. The classifier is used to predict which pairs of websites in the test-set (denoted as $\mathcal{T}$) are replicas.

### 3.1 Features

In this section, we present the features needed to capture the similarity between websites, which are used as evidence of possible replication:

- Edit Distance (ndist): Given a pair of websites $A$ and $B$ and the corresponding hostnames $u_A$ and $u_B$, the edit distance is given as the number of removal, insertion and modification operations that are necessary to transform $u_A$ into $u_B$.
- Hostname Matching (nmatch): Segments of the hostname are treated as terms, so that each hostname is represented as a term vector. The weight of term $t$ is given by:

$$w(t) = \frac{\log(len(t))}{1 + \log(df(t))}. \tag{1}$$

  where $len(t)$ is the number of characters in $t$, and $df(t)$ is the number of hostnames containing $t$.
  The similarity between a pair of websites $A$ and $B$ is given by $\cos(a, b)$, where $a$ and $b$ are the term vectors associated with the corresponding hostnames.
- 4 Octets (ip4): Identical or highly similar IP addresses are indicative of two hosts on the same server or subnet, which may imply replication. IP addresses are grouped according to their octets, so that similar addresses are placed together in the same group. Given a pair of websites $A$ and $B$, the value of this feature is given by:

$$ip(A, B) = \begin{cases} \dfrac{1}{|\mathcal{G}| - 1} & \text{if } A \text{ and } B \text{ are in the same group } \mathcal{G} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

- 3 Octets (ip3): Same as the above feature, but using only the first three octets of the IP address.
- Full Path Matching (fullpath): URLs within a website are treated as terms, so that a website is represented as a term vector. The weight of term $t$ is given by:

$$w(t) = 1 + \log\left(\frac{maxdf}{df(t)}\right) \tag{3}$$

  where $maxdf$ is the maximum value of $df(t)$ over all $t$.

Most of the above features were proposed in Bharat et al. (2000), but there may exist many other features that we could exploit to build our classifiers. However, we restrict our attention to the above features because: (1) the amount of computation needed to compute them does not compromise the practicality of the search engine, mainly because the necessary information is already stored in the index, and (2) they are not based on the content of the pages, and thus can be used at crawling time. Feature values are first discretized (Fayyad and Irani 1993) and then assigned to intervals.

### 3.2 Demand-driven classifiers

Our classifier is denoted as $\mathcal{R}$ and is composed of a set of classification rules that are extracted from the training-set $\mathcal{D}$. Rule-based classifiers were chosen because they can be built efficiently (Agrawal et al. 1993; Veloso et al. 2006), and as will be discussed in Sect. 3.3, classification rules can be updated incrementally in an efficient way. The

definitions presented in this section were first presented in Davis et al. (2012) in another context and repeated here for completeness and reading understanding.

**Definition 3.1** A classification rule has the form $\{\mathcal{X} \rightarrow \ell\}$, where the antecedent $\mathcal{X}$ is a set of features, and the consequent $\ell \in \{\oplus, \ominus\}$ indicates if the prediction is positive or negative. The cardinality of rule $\{\mathcal{X} \rightarrow \ell\}$ is given by the number of features in the antecedent, that is $|\mathcal{X}|$. The support of $\mathcal{X}$ is denoted as $\sigma(\mathcal{X})$, and is the number of examples in $\mathcal{D}$ having $\mathcal{X}$ as a subset. The confidence of rule $\{\mathcal{X} \rightarrow \ell\}$ is denoted as $\theta(\mathcal{X} \rightarrow \ell)$, and is the conditional probability of $c$ given the features in $\mathcal{X}$, that is, $\theta(\mathcal{X} \rightarrow \ell) = \dfrac{\sigma(\mathcal{X} \cup \ell)}{\sigma(\mathcal{X})}$.

Specifically, our classifier $\mathcal{R}$ is represented as a pool of entries $\langle key, properties \rangle$, where $key = \{\mathcal{X}, \ell\}$ and $properties = \{\sigma(\mathcal{X}), \sigma(\mathcal{X} \cup \ell), \theta(X \rightarrow \ell)\}$. Each entry in the pool corresponds to a rule, and the key is used to facilitate fast access to rule properties. Once the classifier $\mathcal{R}$ is extracted from $\mathcal{D}$, rules are collectively used to approximate the likelihood of an arbitrary example being positive ($\oplus$) or negative ($\ominus$). Basically, $\mathcal{R}$ is interpreted as a poll, in which each rule $\{\mathcal{X} \rightarrow \ell\} \in \mathcal{R}$ is a vote given by $\mathcal{X}$ for $\oplus$ or $\ominus$. Given an example $x$, a rule $\{\mathcal{X} \rightarrow \ell\}$ is only considered a valid vote if it is applicable to $x$.

**Definition 3.2** A rule $\{\mathcal{X} \rightarrow \ell\} \in \mathcal{R}$ is said to be applicable to example $x$ if $\mathcal{X} \subseteq x$, that is, if all features in $\mathcal{X}$ are in example $x$.

We denote as $\mathcal{R}_x$ the set of rules in $\mathcal{R}$ that are applicable to example $x$. Thus, only and all the rules in $\mathcal{R}_x$ are considered as valid votes when classifying $x$. Further, we denote as $\mathcal{R}_x^\ell$ the subset of $\mathcal{R}_x$ containing only rules predicting $\ell$. Votes in $\mathcal{R}_x^\ell$ have different weights, depending on the confidence of the corresponding rules. Weighted votes for $\ell$ are averaged, giving the score for $\ell$ with regard to $x$:

$$s(x, \ell) = \frac{1}{|\mathcal{R}_x^\ell|} \sum_{i=1}^{|\mathcal{R}_x^\ell|} \theta(\mathcal{X} \rightarrow \ell)^{(i)}, \quad \text{with } \ell \in \{\ominus, \oplus\} \tag{4}$$

where $\theta(\mathcal{X} \rightarrow \ell)^{(i)}$ is the $i^{th}$ rule in $\mathcal{R}_x^\ell$. Finally, the likelihood of $x$ being a negative example is given by the normalized score:

$$\alpha(x, \ominus) = \frac{s(x, \ominus)}{s(x, \ominus) + s(x, \oplus)} \tag{5}$$

In order to avoid the huge search space for rules, our rule extraction approach projects the training-set according to the example being processed (i.e., a pairs of websites). More specifically, rule extraction is delayed until a candidate $x$ is given for classification. Then, features in $x$ are used as a filter that configures the training-set $\mathcal{D}$ so that just rules that are applicable to $x$ can be extracted.

Table 1 illustrates this filter process, which produces a projected training-set, denoted as $\mathcal{D}_x$, containing only features that are present in $x$.

We now present Algorithm 1, which summarizes the main steps discussed in this section. The algorithm receives as input the training-set $\mathcal{D}$, and an arbitrary pair of websites $x$ (i.e., a replica candidate). The algorithm learns classifier $\mathcal{R}_x$ and returns $\alpha(x, \ominus)$.

**Table 1** Illustrative example. Training-set $\mathcal{D} = \{y_1, y_2, \ldots, y_8\}$, and instance $x$. Also, projected training-set $\mathcal{D}_x$

| | $p$ | | | | | $\ell$ |
|---|---|---|---|---|---|---|
| | ip4 | ip3 | ndist | nmatch | fullpath | |
| $y_1$ | [0.1–0.3] | [0.3–0.5] | [8–10] | [0.2–0.5] | [0.3–0.5] | $\oplus$ |
| $y_2$ | [0.1–0.3] | [0.1–0.3] | [10–14] | [0.1–0.2] | [0.1–0.3] | $\ominus$ |
| $y_3$ | [0.5–0.8] | [0.1–0.3] | [8–10] | [0.1–0.2] | [0.1–0.3] | $\oplus$ |
| $y_4$ | [0.1–0.3] | [0.5–0.8] | [8–10] | [0.2–0.5] | [0.3–0.5] | $\oplus$ |
| $y_5$ | [0.3–0.5] | [0.5–0.8] | [5–8] | [0.5–0.7] | [0.3–0.5] | $\oplus$ |
| $y_6$ | [0.1–0.3] | [0.3–0.5] | [8–10] | [0.5–0.7] | [0.3–0.5] | $\ominus$ |
| $y_7$ | [0.1–0.3] | [0.1–0.3] | [10–14] | [0.1–0.2] | [0.3–0.5] | $\ominus$ |
| $y_8$ | [0.3–0.5] | [0.5–0.8] | [5–8] | [0.5–0.7] | [0.3–0.5] | $\oplus$ |
| $x$ | [0.1–0.3] | [0.1–0.3] | [8–10] | [0.2–0.5] | [0.1–0.3] | ? |
| | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | |
| $y_1$ | [0.1–0.3] | – | [8–10] | [0.2–0.5] | – | $\oplus$ |
| $y_2$ | [0.1–0.3] | [0.1–0.3] | – | – | [0.1–0.3] | $\ominus$ |
| $y_3$ | – | [0.1–0.3] | [1–2] | – | [0.1–0.3] | $\oplus$ |
| $y_4$ | [0.1–0.3] | – | – | [0.2–0.5] | – | $\oplus$ |
| $y_5$ | – | – | – | – | – | $\oplus$ |
| $y_6$ | [0.1–0.3] | – | – | – | – | $\ominus$ |
| $y_7$ | [0.1–0.3] | [0.1–0.3] | – | – | – | $\ominus$ |
| $y_8$ | – | – | – | – | – | $\oplus$ |

**Algorithm 1** Learning Classifiers.

**Given:**

$\mathcal{D}$: the training set

$x$: an arbitrary pair of websites

1. project $\mathcal{D}$ according to $x$, resulting in $\mathcal{D}_x$
2. extract rules $\{\mathcal{X} \rightarrow \ell\}$ from $\mathcal{D}_x$, resulting in $\mathcal{R}_x$
3. calculate $\alpha(x, \ominus)$ according to Equation (5)

### 3.3 Expectation-Maximization (EM)

Learning classifiers is subject to a data acquisition bottleneck, since the creation of a training-set requires human annotators to manually inspect pairs of websites. The cost associated with this annotation process may render vast amounts of examples unfeasible. In many cases, however, the acquisition of some labeled examples may be effortless. Specifically, we may consider WWW and non-WWW versions, as well as .com and .net variations as positive examples. Similarly, negative examples come from pairs of websites that do not share any content or path. However, learning classifiers directly from such examples would lead to poor replica detection performance, since there may be many false-positives and false-negatives in the training-set. In here, we employ EM algorithms (Davis et al. 2012) in order to enhance the classifiers. However, in contrast to Davis et al. (2012) where only PU data are used, we assume two scenarios:

1. PU data: the training-set $\mathcal{D}$ is composed of a small set of positive examples plus a large amount of unlabeled examples.

2. NU data: the training-set $\mathcal{D}$ is composed of a small set of negative examples plus a large amount of unlabeled examples.

In both scenarios, unlabeled examples are initially treated either as negative or positive ones, so that classifiers can be built from $\mathcal{D}$. Therefore, $\mathcal{D}$ may contain false-negatives or false-positives. EM algorithms employ a classifier $\mathcal{R}$ which assigns to each example $x \in \mathcal{D}$ a probability $\alpha(x, \ominus)$ of being negative, as shown in Eq. (5). Then, label-transition operations $x^{\ominus \to \oplus}$ (or $x^{\oplus \to \ominus}$) are performed, so that the training-set $\mathcal{D}$ becomes $\{(\mathcal{D} - x^{\ominus}) \cup x^{\oplus}\}$ (or $\{(\mathcal{D} - x^{\oplus}) \cup x^{\ominus}\}$). In the end of the EM process, it is expected that the assigned labels converge to the combination for which the data is most likely.

Another difference from Davis et al. (2012) is that we allow only uni-directional transitions, in order to ensure faster convergence. That is, in the PU scenario only operations $x^{\ominus \to \oplus}$ are allowed, while in the NU scenario only operations $x^{\oplus \to \ominus}$ are allowed. In both cases, a crucial issue that affects the effectiveness of the EM algorithms concerns the decision of whether or not performing the label-transition operation.

*Best entropy cut* The algorithm proposed in Davis et al. (2012) finds the threshold $\alpha^x_{min}$ that provides the best entropy cut in the probability space induced by $\mathcal{D}_x$. Specifically, given examples $\{y_1, y_2, \ldots, y_k\}$ in $\mathcal{D}_x$ and considering $\ell^{y_i} \in \{\ominus, \oplus\}$ to be the label associated with example $y_i$, the algorithm first calculates $\alpha(y_i, \ominus)$ for each $y_i \in \mathcal{D}_x$, and then the values for $\alpha(y_i, \ominus)$ are sorted in ascending order. Ideally, there is a cut $\alpha^x_{min}$ such that:

$$\ell^{y_i} = \begin{cases} \oplus & \text{if } \alpha(y_i, \ominus) \leq \alpha^x_{min} \\ \ominus & \text{otherwise} \end{cases}$$

However, there are more difficult cases for which it is not possible to obtain a perfect separation in the probability space. Thus, the algorithm finds the best cut (or separation). The basic idea is that any value for $\alpha^x_{min}$ induces two partitions over the space of values for $\alpha(y_i, \ominus)$ (i.e., one partition with values that are lower than $\alpha^x_{min}$, and another partition with values that are higher than $\alpha^x_{min}$). The algorithm sets $\alpha^x_{min}$ to the value that minimizes the entropy of these two partitions. Once $\alpha^x_{min}$ is calculated, it can be used to activate a label-transition operation. Next we present the basic definitions in order to detail the algorithm.

**Definition 3.3** Consider $\mathrm{N}_{\ominus}(\mathcal{D}_x)$ the number of examples in $\mathcal{D}_x$ for which $\ell^y = \ominus$. Similarly, consider $\mathrm{N}_{\oplus}(\mathcal{D}_x)$ the number of examples in $\mathcal{D}_x$ for which $\ell^y = \oplus$. Consider $\mathcal{O} = \{\ldots, \langle \ell^{y_i}, \alpha(y_i, \ominus)\rangle, \ldots, \langle \ell^{y_j}, \alpha(y_j, \ominus)\rangle, \ldots\}$ a list such that $\alpha(y_i, \ominus) \leq \alpha(y_j, \ominus)$. Also, let $f$ be a candidate value for $\alpha^x_{min}$. In this case, $\mathcal{O}_f(\leq)$ is a sub-list of $\mathcal{O}$, that is, $\mathcal{O}_f(\leq) = \{\ldots, \langle \ell^y, \alpha(y, \ominus)\rangle, \ldots\}$, such that for all elements in $\mathcal{O}_f(\leq)$, $\alpha(y, \ominus) \leq f$. Similarly, $\mathcal{O}_f(>) = \{\ldots, <\ell^y, \ \alpha(y, \ominus) >, \ldots\}$, such that for all elements in $\mathcal{O}_f(>)$, $\alpha(y, \ominus) > f$. That is, $\mathcal{O}_f(\leq)$ and $\mathcal{O}_f(>)$ are partitions of $\mathcal{O}$ induced by $f$.

The algorithm works as follows. First, it calculates the entropy in $\mathcal{O}$, as shown in Eq. (6). Second, it calculates the sum of the entropies in each partition induced by $f$, according to Eq. (7). Third, it sets $\alpha^x_{min}$ to the value of $f$ that minimizes $E(\mathcal{O}) - E(\mathcal{O}_f)$.

$$E(\mathcal{O}) = -\left(\frac{\mathrm{N}_{\ominus}(\mathcal{O})}{|\mathcal{O}|} \times \log \frac{\mathrm{N}_{\ominus}(\mathcal{O})}{|\mathcal{O}|}\right) - \left(\frac{\mathrm{N}_{\oplus}(\mathcal{O})}{|\mathcal{O}|} \times \log \frac{\mathrm{N}_{\oplus}(\mathcal{O})}{|\mathcal{O}|}\right) \tag{6}$$

$$E(\mathcal{O}_f) = \frac{|\mathcal{O}_f(\leq)|}{|\mathcal{O}|} \times E(\mathcal{O}_f(\leq)) + \frac{|\mathcal{O}_f(>)|}{|\mathcal{O}|} \times E(\mathcal{O}_f(>)) \tag{7}$$

Thus, instead of using a single threshold value, which would be applied to all examples indistinctly, we use the *best entropy cut*, which is a specific $\alpha^x_{min}$ threshold for each example $x \in \mathcal{D}$.

*Incremental updates* A particular challenge is the algorithm performs several label transition operations during the EM process, and each transition operation modifies $\mathcal{D}$ and invalidates parts of the current classifier $\mathcal{R}$, which must be properly updated. Fortunately, $\mathcal{R}$ can be maintained up-to-date incrementally, so that the updated classifier is exactly the same one that would be obtained by re-constructing it from scratch. According to Davis et al. (2012), all rules $\{\mathcal{X} \rightarrow c\} \in \mathcal{R}$ that have to be updated due to operation $x^{\ominus\rightarrow\oplus}$ (or $x^{\oplus\rightarrow\ominus}$) are those for which $\mathcal{X} \subseteq x$. By definition, $\mathcal{R}_x$ contains only and all such rules.

Updating $\theta$ values is straightforward. For operation $x^{\ominus\rightarrow\oplus}$, it suffices to iterate on $\mathcal{R}_x$, incrementing $\sigma(\mathcal{X} \cup \oplus)$ and decrementing $\sigma(\mathcal{X} \cup \ominus)$. A similar procedure is necessary for operation $x^{\oplus\rightarrow\ominus}$. The corresponding values for $\theta(\mathcal{X} \rightarrow \ominus)$ and $\theta(\mathcal{X} \rightarrow \oplus)$ are simply obtained by computing $\frac{\sigma(\mathcal{X} \cup \ominus)}{\sigma(\mathcal{X})}$ and $\frac{\sigma(\mathcal{X} \cup \oplus)}{\sigma(\mathcal{X})}$, respectively.

Algorithm 2 summarizes the main steps discussed in this section. The algorithm receives as input an initial training-set which is updated iteratively. At the end of the EM process, the algorithm returns the final training-set $\mathcal{D}$. Non-obvious replica cases are found from obvious replica or non-replica cases.

**Algorithm 2** Expectation-Maximization.

**Given:**
$\mathcal{D}$: initial training-set with obvious cases
1. for each $x \in \mathcal{D}$
2. learn classifier $\mathcal{R}_x$ from $\mathcal{D}$ using Algorithm 1
3. calculate $\alpha^x_{min}$

**Expectation step:**
4. if (PU data)
   perform operation $x^{\ominus\rightarrow\oplus}$ if $\alpha(x, \ominus) \leq \alpha^x_{min}$
5. else if (NU data)
   perform operation $x^{\oplus\rightarrow\ominus}$ if $\alpha(x, \ominus) > \alpha^x_{min}$
6. update $\mathcal{D}$ accordingly

**Maximization step:**
7. update $\mathcal{R}_x \subseteq \mathcal{R}$ and $\alpha(x, \ominus)$

*Convergence* Since we assume that label-transition operations are uni-directional, that is, a specific training example $x$ may fall into only one of the two label-transition operations (either, $x^{\ominus\rightarrow\oplus}$ or $x^{\oplus\rightarrow\ominus}$), then at some point of the EM process the labels associated with the training examples will not change anymore. Since the training examples did not have their labels changed, the classifier remains unchanged, and therefore it will not change the labels of other training examples in the next iteration. At this point, convergence is clearly achieved.

### 3.4 Replica detection

Algorithm 3 obtains the replica candidates in descending order. Once the final training-set $\mathcal{D}$ is obtained using our EM algorithm, it is used to build classifier $\mathcal{R}$ which is used to detect website replicas in the test set. Let $\alpha(x, \oplus) = 1 - \alpha(x, \ominus)$ be the likelihood of $x \in \mathcal{T}$ being a replica. Replica candidates are sorted in descending order according to $\alpha(x, \oplus)$, so

that the final ranking is an ordered list $\{x_1, x_2, \ldots, x_k\}$ such that there is no pair $(x_i, x_j)$ for which $\alpha(x_i, \oplus) > \alpha(x_j, \oplus)$, given that $i > j$. Finally, the first $n$ candidates are predicted as being replicas.

**Algorithm 3** Replica Detection.

**Given:**
$\mathcal{D}$: the training-set
$\mathcal{T}$: the test-set
$n$: the number of replicas
1. enhance $\mathcal{D}$ using Algorithm 2
2. for each $x \in \mathcal{T}$
3. learn classifier $\mathcal{R}_x$ from $\mathcal{D}$ using Algorithm 1
4. calculate $\alpha(x, \oplus)$
5. sort candidates in $\mathcal{T}$ in descending order according to $\alpha(x, \oplus)$, and pick the first $n$ candidates

### 3.5 Pareto-efficient aggregation

We may combine classifiers built from PU and NU data into a stronger one. Our aggregation approach is based on a basic concept from Economics called Pareto Efficiency (Palda 2011). Specifically, aggregation is said to be efficient if privileging the predictions of a specific classifier would harm the predictions of the other. Pareto Efficiency is related to the notion of dominance in the space induced by the predictions of both classifiers.

**Definition 3.4** Let $\alpha^P(x, \oplus)$ be the likelihood of $x$ being replica according to the classifier built from PU data. Also, let $\alpha^N(x, \oplus)$ be the likelihood of $x$ being replica according to the classifier built from NU data.

Each replica candidate $x \in \mathcal{T}$ is placed in a bi-dimensional space, with coordinates $\alpha^P(x, \oplus)$ and $\alpha^N(x, \oplus)$. Candidate $a$ is said to dominate candidate $b$ iff both of the following conditions are hold:

- $\alpha^P(a, \oplus) \geq \alpha^P(b, \oplus)$ and $\alpha^N(a, \oplus) \geq \alpha^N(b, \oplus)$
- $\alpha^P(a, \oplus) > \alpha^P(b, \oplus)$ or $\alpha^N(a, \oplus) > \alpha^N(b, \oplus)$

Thus, the dominance operator relates two candidates so that the result of the operation has two possibilities: (1) one candidate dominates the other or (2) both candidates do not dominate each other.

**Definition 3.5** The Pareto frontier $\mathcal{P}$ is composed of all replica candidates that are not dominated by any other candidate. Formally, the Pareto frontier is a list of $k$ replica candidates $\mathcal{P} = \{x_1, x_2, \ldots, x_k\}$ such that there is no pair $(x_i, x_j)$ for which $x_i$ dominates $x_j$.

Figure 1 illustrates the case candidate $a$ dominates candidate $b$. The figure also illustrates the Pareto frontier for a set of replica candidates $x \in \mathcal{T}$ with coordinates $\alpha^P(x, \oplus)$ and $\alpha^N(x, \oplus)$. The Pareto frontier contains either replica candidates that excel in one classifier, or candidates with a proper balance between both classifiers. The final step involves sorting these candidates according to the likelihood of being replica.

**Definition 3.6** Let $dom(x)$ return the number of candidates that are dominated by candidate $x$. The final ranking is an ordered list $\{x_1, x_2, \ldots, x_k\}$ such that there is no pair $(x_i, x_j)$ for which $dom(x_i) > dom(x_j)$, given that $i > j$. That is, most dominant candidates appear first in the ranking.

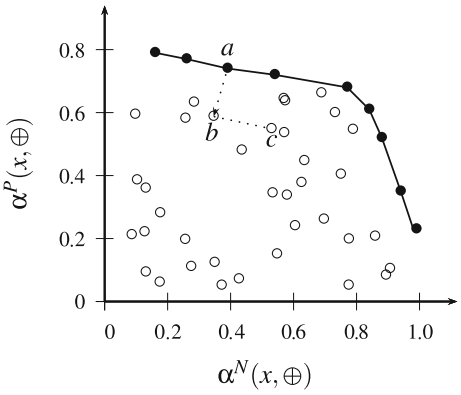


**Fig. 1** Neither $b$ or $c$ dominates each other, but $a$ dominates $b$. Not dominated points form the Pareto frontier

Algorithm 4 presents the main steps discussed in this section It receives as input the values of $\alpha^P(x,\oplus)$ and $\alpha^N(x,\oplus)$ for each replica candidate $x \in \mathcal{T}$, and it returns the $n$ most dominant candidates, which are predicted as being replicas.

**Algorithm 4** Pareto-Efficient Aggregation.

**Given:**
The $\alpha^P(x,\oplus) \times \alpha^N(x,\oplus)$ space obtained using Algorithm 3
$n$: the number of replicas
1. find the Pareto frontier $\mathcal{P} = \{x_1, x_2, \ldots, x_k\}$
2. for each $x_i \in \mathcal{P}$
3. calculate $dom(x_i)$
4. sort candidates in $\mathcal{P}$ in descending order according to $dom(x)$, and pick the first $n$ candidates (assuming $k \leq n$)

## 4 Experimental evaluation

In this section, we assess the effectiveness of our proposed approach. In Sect. 4.1, we describe the evaluation methodology, dataset, metrics, baselines and upper bound used in our investigations. In Sect. 4.2, we present experimental results for the evaluation of the proposed algorithms in terms of reduction ratio, true-positive and false-positive rates, precision/recall, detection performance, and execution time.

### 4.1 Experimental setup

#### *4.1.1 Evaluation methodology and dataset characterization*

We crawled the Web from September to October, 2010, resulting in a large collection of +250 million URLs. Then, we randomly selected +30 million URLs with no restrictions regarding content duplication or quality. Thus, our collection is a reliable representation of the actual Web. The resulting dataset has 583,411 websites, and thus the number of possible pairs is huge (about $2 \times 10^{11}$), precluding any manual or semi-automatic labeling. Once we are interested in identifying replicas, we separate replica candidates and discarded obvious cases of non-replicas, as discussed next.

*Dup-clusters* We produced fingerprints of the content of URLs, and then hashed URLs with identical fingerprints into the same (dup-)cluster (Broder et al. 1997), as shown in

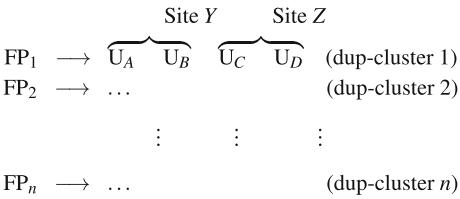


**Fig. 2** Intra-site duplicate URLs (e.g, $U_A$ and $U_B$) and inter-site duplicate URLs (e.g., $U_A$ and $U_C$)

Fig. 2. In the figure, each $FP_i$ is a content fingerprint, and each $U_j$ is a URL. Near-duplicate URLs are those within the same dup-cluster (i.e., identical fingerprints). URLs in a dup-cluster and within the same website are referred to as intra-site duplicate URLs (e.g, $U_A$ and $U_B$), while those within different websites are referred to as inter-site duplicate URLs (e.g., $U_A$ and $U_C$). Different websites within the same dup-cluster are treated as replica candidates. Different websites not having any dup-cluster in common are discarded as obvious non-replicas.

Specifically, there are 172,004 websites sharing at least one fingerprint with other website. By summing the number of pairs of websites within each cluster, we obtained 111 million replica candidates. From this set of candidates we separate approximately 50,000 obvious replica cases to serve as positive examples (WWW and non-WWW versions, as well as small segment variations in the hostname, such as .com, .org, .gov, .edu), and 800,000 pairs of websites not sharing any dup-clusters to serve as negative examples. The size of a dup-cluster is given by the number of URLs in it. Figure 3 (left) shows the dup-cluster size distribution. Clearly, many dup-clusters contain few URLs, and few dup-clusters contain many thousands of URLs. Figure 3 (middle) shows the number of dup-clusters in which the websites appear. Again, few websites appear in many clusters, and many websites appear in few clusters. Figure 3 (right) shows the correlation between the number of intra-site and inter-site duplicate URLs per website. The correlation curve shows that a three-fold increase in the number of inter-site duplicate URLs gives a ten-fold increase in the number of intra-site duplicate URLs.

It is hard to define replicas and non-replicas. Complicated cases include: (1) many replicated websites have dynamic content, such that each time the crawler retrieves a dynamic page, its content may be different, and (2) many websites may have pages with the same or similar content even though they are not replicas (e.g., websites that are related to each other, such as a central site and its affiliates). In order to produce labels that are needed to evaluate our classifiers,[6] we randomly selected a suitable sample of 1,600,000 website pairs from the 111 million possible replica candidates, and after manual inspection, 6853 pairs were labeled as replicas, and the remaining pairs as non-replicas.[7] From the 6853 replicas, 354 are WWW and non-WWW versions or simple hostname variations (e.g., .com, .org, .gov, .edu), but the remaining 6499 replicas are non-obvious cases, involving websites with completely different hostnames. There are 49,636 websites within the 1,600,000 pairs, and 3765 of them appear in at least one of the 6853 replica cases.

Figure 4 (left) shows the size distribution for replicated and non-replicated websites. Replicated websites have an average of 224 URLs, while non-replicated websites have an average of 135 URLs.

[6] These labels are only used to evaluate the classifiers, and are not used to learn them.

[7] Due to the large number of candidates, we cannot assure the non-existence of false-negatives, but we assure the non-existence of false-positives.

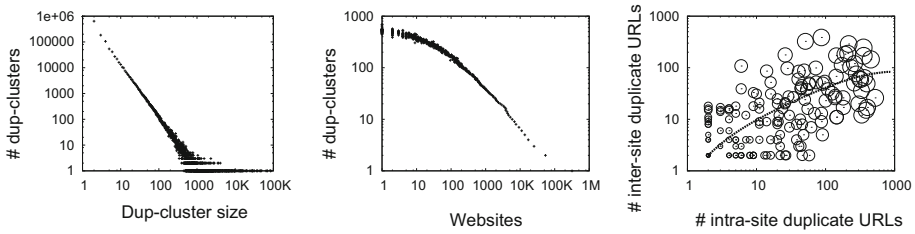


**Fig. 3** Left—dup-cluster size distribution. Middle—distribution of websites per dup-cluster. Right—correlation between the number of intra-site and inter-site duplicate URLs per website

Figure 4 (middle) shows the fraction of websites appearing in at most $x$ replica cases. As can be seen, most of the 3765 replicated websites appear in few replica cases, but some of them appear in more than 100 replica cases. Eliminating all 3765 replicated websites would remove 12.1% of near-duplicate content (843,526 URLs would be removed, from a total of 6,948,501 near-duplicate URLs). It is worth noting that we have considered all URLs found within a same cluster as a near-duplicate URL, and thus the reported reduction ratio is likely to be an overestimation.

Figure 4 (right) shows the similarity distribution (fraction of pairs having similarity above $x$) and replica concentration (fraction of pairs having similarity above $x$ that are replicas). The similarity between websites $A$ and $B$ is given in terms of the number of dup-clusters containing both websites. Specifically, let $C_A$ be the set of dup-clusters containing website $A$. In this case, the similarity between $A$ and $B$ is given as the Jaccard Coefficient $\frac{C_A \cap C_B}{C_A \cup C_B}$. While highly similar websites are likely to be replicas, similarity solely is not enough to completely separate replicas from non-replicas. Some replicas involve websites that are not highly similar.

### *4.1.2 Metrics*

We conducted a five-fold cross validation in our experiments. The dataset with 1,600,000 candidates was arranged into five folds, including training and test In addition to the four folds, the training-set also contains 50,000 positive examples (PU data), or 800,000 negative examples (NU data). It is worth noting that although we have the correct labels for all examples in the training-set, the only labels we used for learning our classifiers are those obtained from obvious replicas and non-replicas. The results reported are the average of the five runs, and we used the Wilcoxon signed-rank test (Wilcoxon 1945) for determining if the difference in performance was statistically meaningful. In all cases, we only drawn conclusions from results that were significant in at least 5% level. All experiments were run on a Linux-based dedicated PC with Core I7 and 4 GB RAM.

Performance is given in terms of the following metrics:

- Precision: given as the fraction of replicas that are correctly detected.
- Recall or True Positive Rate (TPR): given as the fraction of replicas that are removed from the collection.
- False Positive Rate (FPR): given as the fraction of non-replicas that are removed from the collection.

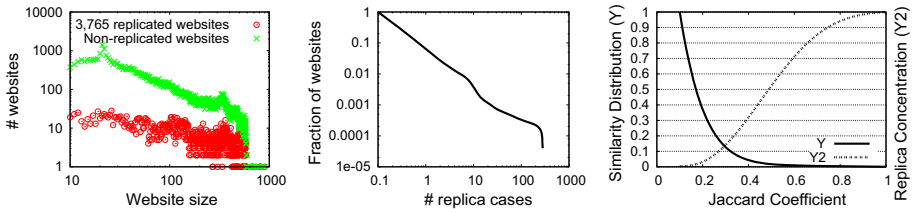


**Fig. 4** (Color online) Left—website size distribution for replicated and non-replicated websites. Middle—fraction of websites appearing in at most $x$ replica cases. Right—similarity distribution and replica concentration

- Reduction Ratio (RR): let $U$ be the set of duplicate URLs in the original collection. Also, let $U^*$ be the subset of $U$ which is obtained after removing website replicas. The reduction ratio is given as $\frac{|U| - |U^*|}{|U|}$.
- Replica Detection Rate (RDR); let $I$ be a list of $k + 1$ replica candidates, where exactly one pair is known to be a replica and the remaining $k$ pairs are randomly selected. Let $i$ be the position of the replica after sorting all $k + 1$ pairs in descending order of $\alpha(x, \ominus)$. This process is repeated $n$ times, and the replica detection rate is given as $\frac{1}{n}\sum^{n}\frac{1}{i}$.

### *4.1.3 Baselines and upper bound*

We considered two algorithms in order to provide baseline comparison:

- NormPaths algorithm (Carvalho et al. 2007): this algorithm is devised to detect and remove website replicas. NormPaths computes website similarities by representing pages using their paths (i.e., URL without access method and host name) and norms. Specifically, it creates a tuple $\langle \text{path}, \text{signature} \rangle$ for each URL in the website, and for each tuple it creates an inverted list $L$ composed of all websites containing the tuple. The similarity between a pair of websites $A$ and $B$ is given by Eq. (8), where $|L|$ is the number of websites in $L$, and $S$ is the set of all lists $L$. Replica candidates are sorted in descending order according to $sim(A, B)$, and the first $n$ candidates are considered as being replicas.

$$sim(A, B) = \sum \frac{1}{|L|}, \quad \forall L \in \mathcal{S} | A \in L \text{ and } B \in L \tag{8}$$

- Biased SVM (Liu et al. 2003) (or simply B-SVM): this algorithm is representative of the state-of-the-art semi-supervised learning algorithms under PU data. B-SVM uses a soft-margin SVM as the underlying classifier, which is re-constructed from scratch after each EM iteration. It employs a single transition threshold for the entire unlabeled dataset.

We considered as upper bound performance the result obtained using classifiers built from the gold-standard training-set, that is, the training-set with the correct labels. The corresponding replica detection rate will be contrasted against the detection rate obtained using classifiers built from the training-set with labels produced by our EM algorithms.

## 4.2 Experimental results

In this section, we assess the effectiveness of our algorithms proposed in Sect. 3. In particular, we aim to answer the following research questions:

- Q1. How do existing website replica detection algorithms perform at crawl time?
- Q2. Can we improve the performance of existing website replica detection algorithms using our EM approach?
- Q3. What is the impact of using classifiers to combine a diverse set of replication features?
- Q4. What is the efficiency of our proposed EM strategy?

In the remainder of this section, Sect. 4.2.1 addresses questions Q1 and Q2. Questions Q3 and Q4 are addressed in Sects. 4.2.2 and 4.2.3, respectively.

### *4.2.1 Website replica detection*

In order to answer questions Q1 and Q2, the first set of experiments concerns evaluating the features proposed in Sect. 3.1. Since it would be prohibitive to conduct a study involving all possible combinations of features, we analyze only some combinations of them. First, we applied each feature in isolation in order to determine its individual performance. Second, because some features may be more useful when not taken in isolation, we studied the impact of each of them by removing it from the feature-set to be used.

Table 2 shows the performance numbers, in terms of AUC (Area Under the Curve), associated with each feature. We consider two scenarios: (1) the feature is used in isolation to represent replica candidates, and (2) all features are used to represent replica candidates, except the one we want to investigate. When taken in isolation, the best features are *ndist* and *fullpath*. The worst features considering the PU data are *ip3* and *ip4*, while *nmatch* performs poorly with NU data.

In most of the cases, discarding only one feature results in virtually the same performance as when using all available features. This suggests that some of the features are redundant. In fact, such redundancy is clear for some sets of features, such as *ip3* and *ip4* (which refer to the IP address), and *nmatch* and *ndist* (which refer to the URL string). Discarding the *fullpath* feature results in the highest performance decrease.

Figure 5 (left) shows the ROC analysis for the evaluation of our classifiers. The NormPaths algorithm offers the worst trade-off between TPR and FPR, achieving a

**Table 2** Performance of different feature combinations

| Features | AUC (area under the TPR/FPR curve) | | | |
|---|---|---|---|---|
| | In isolation | | All except | |
| | PU | NU | PU | NU |
| – | 0.9908 | 0.9688 | 0.9908 | 0.9688 |
| *ip3* | 0.5132 | 0.5116 | 0.9706 | 0.9411 |
| *ip4* | 0.5288 | 0.5283 | 0.9701 | 0.9431 |
| *nmatch* | 0.6565 | 0.2312 | 0.9646 | 0.9391 |
| *ndist* | 0.7716 | 0.6528 | 0.9631 | 0.9272 |
| *fullpath* | 0.7187 | 0.6274 | 0.9550 | 0.9206 |

performance number of 0.9415 in terms of AUC. Classifiers built using PU and NU data achieve performance numbers of 0.9908 and 0.9688, respectively. Combining both classifiers using the proposed Pareto-Efficient aggregation algorithm improves the performance to 0.9950. The B-SVM algorithm shows competitive performance, providing numbers as high as 0.9824 in terms of AUC. In summary, ROC analysis reveals that the area under the curve is higher than 0.98, indicating that our algorithms quickly detect all website replicas in the collection.

Figure 5 (middle) shows the precision/recall analysis for the evaluation of our classifiers. Again, the NormPaths algorithm offers the worst numbers. Also, classifiers built using PU data greatly outperform those built from NU data, as well as B-SVM. Combining classifiers built from PU and NU data, using the proposed Pareto-Efficient aggregation algorithm, shows a slightly superior performance.

Figure 5 (right) shows the trade-off between reduction ratio and false-positive rate. For this analysis, it is important to consider the reduction ratio obtained at the cost of very low FPR values. Thus, the figure shows the trade-off considering FPR values of at most 0.005. At this scale, the performance of the competing algorithms differ greatly. Specifically, NormPaths is the worst performer, providing a 14% reduction with a FPR of 0.005. At the same precision level, classifiers learned from PU and NU data achieved a reduction of 17 and 18%, respectively. Combining the predictions of both classifiers leads to a reduction of 19%, which is extremely close to the upper bound performance. The gains over the baseline are more impressive if we consider lower FPR scales. For instance, the best performer (PU + NU) achieved a reduction of 12% with FPR of 0.001, while NormPaths achieved a reduction of only 7% at this precision scale.

Table 3 shows the best reduction ratio obtained by our site-level classifiers (PU + NU), with different FPR values. The table also shows the reduction ratio obtained by an URL-level algorithm (Dasgupta et al. 2008). There are 6,948,501 duplicate URLs in the collection: 5,416,003 intra-site duplicates and 843,526 inter-site duplicates. Further, 409,771 URLs occur as intra and inter-site duplicates simultaneously. We considered two configurations of the URL-level algorithm: (1) without rule-error ($\epsilon = 0.0$) which eliminated about 4% of duplicates, and (2) with a high rule-error ($\epsilon = 0.1$) which eliminated + 23% of the duplicates. The reduction provided by our classifiers depends on the FPR allowed, and varies from 7.9% (with no false-positives) to + 19% (with FPR of 0.005). Finally, combining both URL elimination strategies provides a reduction ratio greater than 21% with $\epsilon = 0.0$. The reduction ratio increases to 37% for $\epsilon = 0.1$.

In summary, removing replicas reduces the size of the collection by 19% with a false-positive rate of only 0.005. Our algorithms can be used in conjunction with URL-level algorithms (Dasgupta et al. 2008), and in this case the reduction ratio increases to more

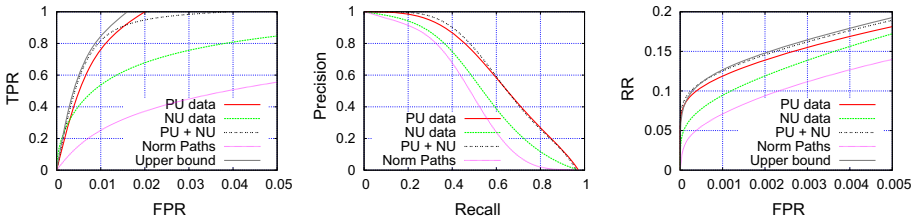


**Fig. 5** (Color online) Left—ROC analysis. Middle—precision/recall analysis. Right—reduction ratio and false-positive rate

**Table 3** Reduction ratio after removing duplicate URLs

| | | FPR | | |
|---|---|---|---|---|
| | | 0.000 | 0.001 | 0.005 |
| # duplicate URLs | 6,948,501 | – | – | – |
| # intra-site URLs | 6,514,746 | – | – | – |
| $\rightarrow \epsilon = 0.0$ | 293,374 | – | – | – |
| $\rightarrow \epsilon = 0.1$ | 1,628,685 | – | – | – |
| # inter-site URLs | 843,526 | 555,880 | 865,384 | 1,331,215 |
| # inter $\cap$ intra | 409,771 | 286,485 | 446,182 | 758,927 |
| $\rightarrow \epsilon = 0.0$ | 64,947 | 38,835 | 70,232 | 110,284 |
| $\rightarrow \epsilon = 0.1$ | 151,683 | 98,377 | 170,827 | 388,022 |
| RR intra | 0.9376 | – | – | – |
| $\rightarrow \epsilon = 0.0$ | 0.0422 | – | – | – |
| $\rightarrow \epsilon = 0.1$ | 0.2344 | – | – | – |
| RR inter | 0.1213 | 0.0791 | 0.1245 | 0.1916 |
| RR inter + intra | 1.0000 | – | – | – |
| $\rightarrow \epsilon = 0.0$ | 0.1541 | 0.1166 | 0.1567 | 0.2179 |
| $\rightarrow \epsilon = 0.1$ | 0.3340 | 0.3002 | 0.3343 | 0.3701 |

than 21%, by eliminating both intra-site and inter-site duplicate URLs. If errors as high as 10% are allowed during intra-site de-duplication, then the reduction ratio goes to + 37%.

### *4.2.2 Replica detection rate*

In order to answer question Q3, Table 4 shows detection rate numbers for the different algorithms. The classifier built from PU data is very effective if we consider less than 1000 replica candidates, achieving detection rate numbers that are close to the upper bound. On the other hand, the classifier built from NU data performed poorly, even if we consider less than 10 replica candidates. However, its performance seems to remain almost constant as the number of candidates increases, indicating that the detected replicas received very high scores. NormPaths showed the same trend, and achieved similar numbers.

In summary, our algorithms are able to detect a replica case infiltrated in 1000 arbitrary replica candidates, with probability greater than 99%.

### *4.2.3 Execution time and incremental updates*

In order to answer question Q4, the EM process runs entirely offline, resulting in the final training-set, which is then used to produce classifiers. Table 5 shows the time spent during the EM process. Creating the training-set from PU data takes more than 2.5 h if partial classifiers are built from scratch after each label transition operation. This time decreases to 112 s, if partial classifiers are incrementally updated. Creating the training-set from NU data requires more iterations, and thus more time is spent. Finally, it takes less than 0.1 s to build a classifier for an arbitrary replica candidate.

In summary, execution time is decreased by three orders of magnitude, by incrementally updating the classifiers during the EM process. During crawling, the time spent to evaluate a replica candidate is no greater than 0.1 s.

**Table 4** Replica detection rate

| Algorithms | Number of candidates | | | |
|---|---|---|---|---|
| | 10 + 1 | 100 + 1 | 1000 + 1 | 10,000 + 1 |
| PU data | 0.9900 | 0.9891 | 0.9615 | 0.7656 |
| NU data | 0.6590 | 0.4429 | 0.4300 | 0.4287 |
| PU + NU | 1.0000 | 1.0000 | 0.9905 | 0.8272 |
| NormPaths | 0.6619 | 0.4192 | 0.4105 | 0.4088 |
| B-SVM | 0.9805 | 0.9622 | 0.9349 | 0.7034 |
| Upper bound | 1.0000 | 0.9901 | 0.9586 | 0.7555 |

**Table 5** Execution time in seconds

| Offline | | Online | |
|---|---|---|---|
| EM process | Time | Learning classifiers | Time |
| PU (from scratch) | 8838.98 | Rule extraction | 0.0575 |
| PU (incremental) | 112.26 | Prediction | 0.0086 |
| NU (from scratch) | 9172.37 | Aggregation | 0.0232 |
| NU (incremental) | 131.11 | | |

## 5 Conclusions

The presence of duplicate URLs adversely impacts both the efficiency and the effectiveness of information retrieval systems. Duplicate URLs may be divided into intra-site duplicates and inter-site duplicates, depending on whether they occur within the same website or across multiple websites. While most of duplicate content falls into the intra-site category, a complete solution must also deal with inter-site duplicates. The existence of inter-site duplicate URLs is mainly associated with website replicas—pairs of websites that either completely match or are appreciably similar in terms of content and structure.

In this paper we proposed algorithms for detecting website replicas, and evaluate the impact of removing website replicas from the collection. Our algorithms employ binary classifiers which are built either from obvious replica cases or from obvious non-replica cases. Expectation-Maximization is used to enhance the classifiers iteratively, by finding non-obvious examples from obvious ones, resulting in effective classifiers with no manual labeling effort. Finally, classifiers built from replica cases are combined with classifiers built from non-replica cases in a way that the errors associated with a classifier are compensated by the other, improving replica detection performance.

Our experiments show a reduction of almost 8% in the number of duplicate URLs. If false-positive rates as low as 0.005 are allowed, the reduction increases to 19%. Furthermore, our classifiers can be used in conjunction with URL-level algorithms, which eliminate intra-site duplicate URLs, and in this case the reduction in the number of duplicate URLs goes to 21%.

**Acknowledgements** We thank the partial support given by the Brazilian National Institute of Science and Technology for the Web (Grant MCT-CNPq 573871/2008-6), Project Models, Algorithms and Systems for the Web (Grant FAPEMIG/PRONEX/MASWeb APQ-01400-14), and authors' individual grants and scholarships from CNPq.

## References


Agarwal, A., Koppula, H. S., Leela, K. P., Chitrapura, K. P., Garg, S., Pavan Kumar, et al. (2009). URL normalization for de-duplication of web pages. In *Proceedings of the 18th ACM conference on information and knowledge management* (pp. 1987–1990).

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Record*, *22*(2), 207–216.

Bar-Yossef, Z., Keidar, I., & Schonfeld, U. (2009). Do not crawl in the dust: Different URLs with similar text. *ACM Transactions on the Web*, *3*(1), 3:1–3:31.

Bharat, K., & Broder, A. (1999). Mirror, mirror on the web: A study of host pairs with replicated content. *Computer Networks*, *31*(11–16), 1579–1590.

Bharat, K., Broder, A., Dean, J., & Henzinger, M. R. (2000). A comparison of techniques to find mirrored hosts on the www. *Journal of the American Society for Information Science*, *51*(12), 1114–1122.

Broder, A. Z., Glassman, S. C., Manasse, M. S., & Zweig, G. (1997). Syntactic clustering of the web. *Computer Network ISDN System*, *29*(8–13), 1157–1166.

Carvalho, A Ld C., Moura, E. S. d, Silva, A. S. d, Berlt, K., & Bezerra, A. (2007). A cost-effective method for detecting web site replicas on search engine databases. *Data Knowledge Engineering*, *62*(3), 421–437.

Cho, J., Shivakumar, N., & Garcia-Molina, H. (2000). Finding replicated web collections. *SIGMOD Record*, *29*, 355–366.

Dasgupta, A., Kumar, R., & Sasturkar, A. (2008). De-duping URLs via rewrite rules. In *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 186–194).

Davis, A., Veloso, A., Silva, A., Laender, A. H. F., & Meira-Jr., W. (2012). Named entity disambiguation in streaming data. In *Proceedings of he 50th annual meeting of the Association for Computational Linguistics* (pp. 815–824).

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, *39*(1), 1–38.

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th international joint conference on artificial intelligence* (pp. 1022–1029).

Fetterly, D., Manasse, M., & Najork, M. (2003). On the evolution of clusters of near-duplicate web pages. In *Proceedings of the 1st conference on Latin American Web Congress* (pp. 37–45).

Koppula, H. S., Leela, K. P., Agarwal, A., Chitrapura, K. P., Garg, S., & Sasturkar, A. (2010). Learning URL patterns for webpage de-duplication. In *Proceedings of the 3rd ACM international conference on web search and data mining* (pp. 381–390).

Lei, T., Cai, R., Yang, J.-M., Ke, Y., Fan, X., & Zhang, L. (2010). A pattern tree-based approach to learning URL normalization rules. In *Proceedings of the 19th international conference on world wide web* (pp. 611–620).

Liu, B., Dai, Y., Li, X., Lee, W. S., & Yu, P. S. (2003). Building text classifiers using positive and unlabeled examples. In *Proceedings of the 3rd IEEE international conference on data mining* (pp. 179–188).

Palda, F. (2011). *Pareto's republic and the new science of peace*. Ottawa: Cooper-Wolfling.

Rodrigues, K. W. L., Cristo, M., de Moura, E. S., & da Silva, A. S. (2013). Learning URL normalization rules using multiple alignment of sequences. In *Proceedings of the 20th international symposium on string processing and information retrieval* (pp. 197–205).

Veloso, A., Meira, W., Jr., & Zaki, M. (2006). Lazy associative classification. In *IEEE international conference on data mining* (pp. 645–654).

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 80–93.

Yang, H., & Callan, J. (2006). Near-duplicate detection by instance-level constrained clustering. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 421–428).

Ye, S., Wen, J.-R., & Ma, W.-Y. (2008). A systematic study on parameter correlations in large-scale duplicate document detection. *Knowledge and Information Systems*, *14*, 217–232.