



Supervised Meta-blocking

George Papadakis, George Papastefanatos, Georgia Koutrika

HP Laboratories
HPL-2015-16

Keyword(s):

blocking; entity resolution

Abstract:

Entity Resolution matches mentions of the same entity. Being an expensive task for large data, its performance can be improved by blocking, i.e., grouping similar entities and comparing only entities in the same group. Blocking improves the run-time of Entity Resolution, but it still involves unnecessary comparisons that limit its performance. Meta-blocking is the process of restructuring a block collection in order to prune such comparisons. Existing unsupervised meta-blocking methods use simple pruning rules, which offer a rather coarse-grained filtering technique that can be conservative (i.e., keeping too many unnecessary comparisons) or aggressive (i.e., pruning good comparisons). In this work, we introduce supervised meta-blocking techniques that learn classification models for distinguishing promising comparisons. For this task, we propose a small set of generic features that combine a low extraction cost with high discriminatory power. We show that supervised meta-blocking can achieve high performance with small training sets that can be manually created. We analytically compare our supervised approaches with baseline and competitor methods over 10 large-scale datasets, both real and synthetic.

Supervised Meta-blocking

George Papadakis[§], George Papastefanatos[§], Georgia Koutrika[◇]

[◇] HP Labs, USA koutrika@hp.com

[§] Institute for the Management of Information Systems, Research Center “Athena”, Greece
(gpapadis, gpapas)@imis.athena-innovation.gr

ABSTRACT

Entity Resolution matches mentions of the same entity. Being an expensive task for large data, its performance can be improved by blocking, i.e., grouping similar entities and comparing only entities in the same group. Blocking improves the run-time of Entity Resolution, but it still involves unnecessary comparisons that limit its performance. Meta-blocking is the process of restructuring a block collection in order to prune such comparisons. Existing unsupervised meta-blocking methods use simple pruning rules, which offer a rather coarse-grained filtering technique that can be conservative (i.e., keeping too many unnecessary comparisons) or aggressive (i.e., pruning good comparisons). In this work, we introduce supervised meta-blocking techniques that learn classification models for distinguishing promising comparisons. For this task, we propose a small set of generic features that combine a low extraction cost with high discriminatory power. We show that supervised meta-blocking can achieve high performance with small training sets that can be manually created. We analytically compare our supervised approaches with baseline and competitor methods over 10 large-scale datasets, both real and synthetic.

1. INTRODUCTION

Entity Resolution (ER) is the process of finding and linking different instances (profiles) of the same real-world entity [9]. It is an inherently quadratic task, since, in principle, each entity profile has to be compared with all others. For Entity Resolution to scale to large datasets, *blocking* is used to group similar entities into blocks so that profile comparisons are limited within each block. Blocking methods may place each entity profile into only one block, forming disjoint blocks, or into multiple blocks, creating redundancy [4].

Redundancy is typically used for reducing the likelihood of missed matches – especially for noisy, highly heterogeneous data [9, 21]. In particular, redundancy-positive blocking is based on the intuition that the more blocks two entities share, the more likely they match [22]. To illustrate, consider the profiles in Figure 1(a): profiles p_1 and p_3 correspond to the same person and so do p_2 and p_4 . As an example of a redundancy-based blocking method, let us consider Token Blocking [21], which creates one block for every distinct token

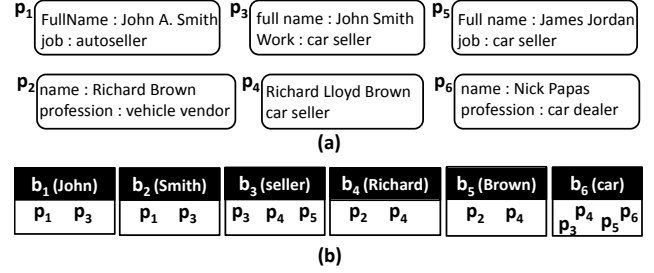


Figure 1: (a) A set of entity profiles, and (b) the redundancy-positive block collection produced by Token Blocking.

that appears in at least two profiles. The resulting block collection is shown in Figure 1(b). We observe that both pairs of matching profiles can be detected, as they co-occur in at least one block.

However, redundancy brings about repeated comparisons between the same entity profiles in different blocks. In the example of Figure 1(b), block b_2 repeats the comparison contained in block b_1 , while b_5 repeats the comparison in b_4 . Hence, b_2 and b_5 contain one *redundant* comparison each. Furthermore, there are several comparisons between non-matching entities, which we call *superfluous* comparisons. Block b_3 entails 3 superfluous comparisons between the non-matching profiles p_3, p_4 and p_5 . In b_6 , all 3 comparisons involving p_6 are superfluous, while the rest are redundant, repeated in b_3 . Overall, while blocking improves entity resolution times, it still involves unnecessary comparisons that limit its performance: superfluous ones between non-matching entities, and redundant ones, which repeatedly compare the same entity profiles. In our example, the total number of comparisons in the blocks of Figure 1(b) is 13 compared to 15 of the brute-force method. This number could be further reduced – without affecting the recall of blocking-based ER – by avoiding the redundant and the superfluous comparisons.

Meta-blocking is a method that takes as input a redundancy-positive block collection and transforms it into a new block collection that generates fewer comparisons, but keeps most of the detected duplicates [22]. To achieve this, existing meta-blocking techniques operate in two phases. First, they map the input block collection to a graph, called *blocking graph*; its nodes are the entity profiles, while its edges connect two nodes if the corresponding profiles co-occur in at least one block. By definition, the graph eliminates all redundant comparisons: each pair of co-occurring profiles is connected with a single edge, which means that they will be compared only once. In the second phase, meta-blocking techniques use the graph to prune superfluous comparisons. For this task, each edge is assigned a weight leveraging the fundamental property of redundancy-positive block collections that the similarity of two entity profiles is proportional to their co-occurrences in blocks. High

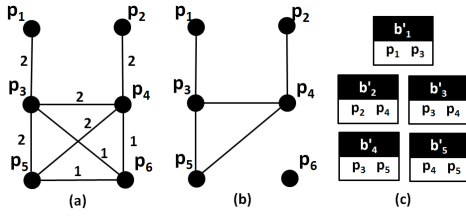


Figure 2: (a) A blocking graph mapping the blocks in Figure 1, (b) possible pruned blocking graph, (c) the derived blocks.

weights are given to the *matching edges* (i.e., edges likely connecting duplicates) and lower weights to the *non-matching* ones.

As an example, the blocks in Figure 1(b) can be mapped to the blocking graph depicted in Figure 2(a). The edge weights are typically defined in the interval $[0,1]$ through normalization, but for simplicity, we consider that each edge weight in this example is equal to the absolute number of blocks shared by its adjacent entities. Different pruning algorithms can be used to remove edges with low weights and hence discard part of the superfluous comparisons. For example, one such strategy, called Weight Edge Pruning, discards all edges having a weight lower than the average edge weight across the entire graph [22]. For the blocking graph of Figure 2(a), the average edge weight is 1.625. The resulting pruned blocking graph is shown in Figure 2(b). The output block collection is generated from the pruned blocking graph by placing the adjacent entities of every edge into a separate block as shown in Figure 2(c). As the result of meta-blocking, the new block collection contains just 5 comparisons and does not miss any matches.

Existing meta-blocking methods use simple pruning rules such as “if $weight < threshold$ then discard edge” for removing comparisons. Consequently, they face two challenges: assigning representative weights to edges and choosing a good threshold for removing edges. We argue that determining if an edge is a good candidate for removal is in fact a multi-criteria decision problem. Combining these criteria into a single scalar value inevitably misses valuable information. Furthermore, pruning based on a single threshold on the weights is a rather coarse-grained filtering technique that can be conservative (i.e., keeping many superfluous comparisons) or aggressive (i.e., pruning good comparisons). In our example in Figure 2(c), the final block collection retains 3 superfluous comparisons in b'_3 , b'_4 and b'_5 ; increasing the threshold so as to further reduce these comparisons would prune the matching comparisons, as well, because they have the same weight as the superfluous ones.

In this paper, we argue that accurate identification of non-matching edges requires learning composite pruning models from the data. We formalize meta-blocking as a binary classification task, where the goal is to identify matching and non-matching edges. We propose supervised meta-blocking techniques that compose generic, schema-agnostic information about the co-occurring entities into comprehensive feature vectors, instead of summarizing it into unilateral weights, as unsupervised methods do.

For example, the blocks of Figure 1(b) can be mapped to the blocking graph of Figure 3(a), where each edge is associated with a feature vector $[a_1, a_2]$. The feature a_1 is the number of common blocks shared by the adjacent entities, and a_2 is the total number of comparisons contained in these blocks. The resulting feature vectors are fed into a classification algorithm that learns composite rules (or models) to effectively distinguish matching and non-matching edges. In our example, a composite rule could look like “if $a_1 \leq 2$ & $a_2 > 5$ then discard edge”, capturing the intuition that the more blocks two profiles share and the smaller these blocks are, the more likely the profiles match. Figure 3(b) shows the graph

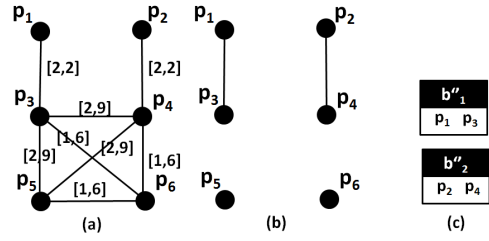


Figure 3: (a) A blocking graph mapping the blocks in Figure 1, (b) a possible pruned blocking graph, (c) the derived blocks.

generated by this rule, and Figure 3(c) depicts the resulting blocks; compared to the blocks in Figure 2(c), they have no superfluous comparisons, thus achieving higher efficiency for the same recall.

We identify and examine three aspects that determine the performance of supervised meta-blocking techniques: (a) the set of features annotating the edges of the blocking graph, (b) the training data, and (c) the classification algorithm and its configuration.

Using more features may help make the pruning of the non-matching edges more accurate. However, the computational cost for meta-blocking gets higher. Moreover, the classification features should be generic enough to apply to any redundancy-positive block collection. With these issues in mind, we propose a small set of generic features that combine a low extraction cost with high discriminatory power and we evaluate their performance using real data. Furthermore, to facilitate the understanding of the space of possible features, we divide it according to five dimensions.

Selecting training data, we face two issues. The first one is a class imbalance problem: the vast majority of the edges in a blocking graph are non-matching. In order to build representative training sets, we select the most suitable technique for our task among established solutions. The second issue regards the training set size. In general, large training sets increase the accuracy and robustness of the learned model. However, they yield complex, inefficient classifiers that require time-consuming training. In addition, the manual creation of large training sets in the absence of ground-truth is a painful and challenging process. We show that we can achieve high performance with small training sets that can be manually created making supervised meta-blocking a practical solution.

We consider a representative sample of state-of-the-art classifiers: Naïve Bayes, Bayesian Networks, Decision Trees and Support Vector Machines. We show that our supervised techniques are robust with respect to different classifiers and their configurations by examining their performance over several large-scale datasets.

Finally, we evaluate the performance of supervised meta-blocking by comparing to (a) the brute-force Entity Resolution, which executes all comparisons included in the input set of blocks, (b) the top-performing unsupervised meta-blocking methods [22], and (c) the iterative blocking [25]. Note that the iterative blocking constitutes the only other method in the literature that, similarly to meta-blocking, receives an existing block collection and aims at processing it in a way that improves its original performance: it propagates every detected pair of duplicates to all associated blocks in order to identify additional matches and to save unnecessary comparisons. We perform a scalability analysis, which involves 7 large-scale synthetic datasets of various sizes, ranging from 10 thousand to 2 million entities. Our experiments demonstrate that our supervised techniques exhibit significantly better time efficiency than the best alternatives, while achieving equivalent recall.

In summary, this paper makes the following contributions:

- We formalize supervised meta-blocking as a classification problem and we demonstrate how it can be used to significantly enhance the quality of a redundancy-positive block collection.

- We map the space of possible classification features along five dimensions and we select a small set of generic features that combine a low extraction cost with high discriminatory power. We evaluate their performance using real data.
- We show that small training sets, which can be manually created, can achieve high performance, making supervised meta-blocking a practical solution for Entity Resolution.
- We show that our supervised techniques are robust with respect to different classifiers and their configurations by examining their performance over several large-scale datasets.
- We perform a thorough scalability analysis, comparing supervised meta-blocking against the best competitor approaches.

The rest of the paper is structured as follows. Section 2 presents related work, Section 3 provides a brief overview of unsupervised meta-blocking, Section 4 introduces supervised meta-blocking, and Section 5 describes the real-world datasets and the metrics used in the evaluation. Sections 6 and 7 cover feature and training set selection, while in Section 8, we fine-tune the classification algorithms. Section 9 experimentally compares supervised meta-blocking with competitor techniques and finally, Section 10 concludes the paper.

2. RELATED WORK

There is a large body of work on Entity Resolution [9, 19]. Blocking techniques group similar entities into blocks so that profile comparisons are limited within each block. These methods can be distinguished into *schema-based* and *schema-agnostic* ones.

Schema-based methods (e.g., Sorted Neighborhood [12], Suffix Array [7], HARRA [14], Canopy Clustering [17], and q-grams blocking [10]) group entities based on knowledge about the semantics of their attributes. These approaches are only suitable for homogeneous information spaces, like databases, where the quality of the schema is known a-priori. In contrast, schema-agnostic blocking techniques cluster entities into blocks without requiring any knowledge about the underlying schema(ta). For instance, in Token Blocking [21], every token that is shared by at least two entities creates an individual block. Total Description [20] improves on Token Blocking by considering the most discriminative parts of entity URIs instead of all their tokens. In the same category fall Attribute Clustering [21] and TYPiMatch [16]. These techniques are preferred in the context of heterogeneous information spaces, which involve large volumes of noisy, semi-structured data that are loosely bound to various schemata [11].

Both schema-based and schema-agnostic blocking methods usually produce redundancy-positive block collections [22]. Meta-blocking operates on top of them, improving the balance between precision and recall by restructuring the block collection [22].

All the aforementioned approaches rely on an unsupervised functionality. Supervised learning has been applied to blocking-based ER with the purpose of fine-tuning the configuration of schema-based blocking methods: in [1, 18], the authors propose methods for learning combinations of attribute names and similarity metrics that are suitable for extracting and comparing blocking keys. Supervised learning has also been applied to generic ER in order to classify pairs of entities into matching and non-matching, by adapting similarity metrics and the corresponding thresholds to a particular domain [2, 6, 8, 24]. Other works introduce methods for facilitating the construction of the training set [23], while in [3], the authors propose supervised techniques for combining the decisions of multiple ER systems into an ensemble of higher performance. No prior work has applied supervised learning techniques to the task of meta-blocking.

3. PRELIMINARIES

In this section, we introduce the main concepts and notation used in the paper and we provide a brief overview of existing unsupervised meta-blocking techniques. Table 1 summarizes notation.

An *entity profile* p is a uniquely identified collection of information described in terms of name-value pairs. An *entity collection* \mathcal{E} is a set of entity profiles. Two profiles $p_i, p_j \in \mathcal{E}$ are *duplicates* or *matches* ($p_i \equiv p_j$) if they represent the same real-world object.

Entity Resolution comes in two forms. *Clean-Clean ER* receives as input two duplicate-free but overlapping entity collections and returns as output all pairs of duplicate profiles they contain. *Dirty ER* receives as input a single entity collection that contains duplicates in itself and returns the set of matching entity profiles. Blocking can be used to scale both forms of ER to large entity collections by clustering similar profiles into *blocks* so that comparisons are restricted among the entity profiles within each block b_i .

The quality of a *block collection* B can be measured in terms of two competing criteria, namely precision and recall, which are estimated through the following established measures [1, 7, 18, 21]:

(i) *Pairs Quality (PQ)* assesses precision, i.e., the portion of non-redundant comparisons between matching entities. It is defined as: $PQ(B) = |\mathcal{D}(B)|/||B||$, where $\mathcal{D}(B)$ represents the set of detectable matches, i.e. the pairs of duplicate profiles that co-occur in at least one block, and $|\mathcal{D}(B)|$ stands for its size. $||B||$ is called *aggregate cardinality* and denotes the total number of comparisons contained in B : $||B|| = \sum_{b_i \in B} ||b_i||$, where $||b_i||$ is the cardinality of b_i (i.e., the number of pair-wise comparisons it entails). PQ takes values in $[0, 1]$, with higher values indicating higher precision for B , i.e., fewer superfluous and redundant comparisons.

(ii) *Pair Completeness (PC)* assesses recall, i.e., the portion of duplicates that share at least one block and, thus, can be detected. It is formally defined as: $PC(B) = |\mathcal{D}(B)|/|\mathcal{D}(\mathcal{E})|$, where $\mathcal{D}(\mathcal{E})$ represents the set of duplicates contained in the input entity collection \mathcal{E} , and $|\mathcal{D}(\mathcal{E})|$ stands for its size. PC values are in the interval $[0, 1]$, with higher values indicating higher recall for B .

Note that we follow a known best practice [1, 4, 18, 25], examining the quality of a block collection independently of profile matching techniques. In particular, we assume an oracle exists that correctly decides whether two entity profiles match or not. Thus, $\mathcal{D}(B)$ is equivalent to the set of matching comparisons in B . The rationale of this approach is that a block collection with high precision and high recall guarantees that the quality of a complete ER solution is as good as that of the selected entity matching algorithm.

There is a clear trade-off between the precision and the recall of B : as more comparisons are executed (i.e., higher $||B||$), its recall increases (i.e., higher $|\mathcal{D}(B)|$), but its precision decreases, and vice versa. The redundancy-positive block collections achieve high PC at the cost of lower PQ , as they place every entity profile into multiple blocks. Meta-blocking aims at improving this balance by restructuring a redundancy-positive block collection B into a new one B' of higher precision but equivalent recall. More formally:

PROBLEM 1 (META-BLOCKING). *Given a redundancy-positive block collection B , the goal of meta-blocking is to restructure B into a new collection B' that achieves significantly higher precision, while maintaining the original recall ($PQ(B') \gg PQ(B)$, $PC(B') \approx PC(B)$).*

Existing meta-blocking techniques rely their functionality on the *weighted blocking graph* (\mathcal{G}_B), a data structure that models the block assignments in the block collection B . As illustrated in Figure 2(a), \mathcal{G}_B is formed by creating a node for every entity profile in B and an undirected edge for every non-redundant pair of co-occurring profiles. Formally, this structure is defined as follows:

p_i	an entity profile
$B, B , B $	a block collection, its size (# of blocks), its cardinality (# of comparisons)
$b_i, b_i , b_i $	a block, its size (# of entities), its cardinality (# of comparisons)
G_B, V_B, E_B	the generalized blocking graph of B , its nodes and its edges
$G_{v_i}, V_{v_i}, E_{v_i}$	the neighborhood of node v_i , its nodes and its edges
$B_i \subseteq B, B_i $	the set of blocks containing p_i and its size (# of blocks)
$B_{i,j} \subseteq B$	the set of blocks shared by the p_i and p_j ($B_{i,j} = B_i \cap B_j$)
$ B_{i,j} $	the size of $B_{i,j}$, i.e., # of comparisons between p_i and p_j
$\mathcal{D}(B), \mathcal{D}(B) $	the set of detected duplicates in B and its size
$p_i.comp()$	the set of comparisons entailing p_i (including the redundant ones)

Table 1: Summary of main notation.

DEFINITION 1 (WEIGHTED BLOCKING GRAPH). *Given a block collection B , its weighted blocking graph is a graph $\mathcal{G}_B = \{V_B, \mathcal{E}_B, W_B\}$, where V_B is the set of its nodes such that $\forall p_i \in B \exists n_i \in V_B$, $\mathcal{E}_B \subseteq V_B \times V_B$ is the set of undirected edges between all pairs of co-occurring entity profiles in B , and W_B is the set of edge weights that take values in the interval $[0, 1]$ such that $\forall e_{i,j} \in \mathcal{E}_B \exists w_{i,j} \in W_B$.*

As explained in Section 1, the blocking graph enhances precision by eliminating all redundant comparisons without any impact on recall, since it contains no parallel edges. Then, meta-blocking applies a pruning algorithm in order to discard part of the superfluous comparisons at a small cost in recall. These algorithms are distinguished into four categories, based on their functionality and the type of threshold they incorporate [22]:

- **Cardinality Edge Pruning (CEP)** sorts all edges in descending order of their weight and retains those in the top K ranking positions. Therefore, K constitutes a global cardinality threshold that is applied to the entire graph.
- **Cardinality Node Pruning (CNP)** does the same, but retains the top k edges for each node. k is also a global cardinality threshold, but is applied to the neighborhood of each node.
- **Weight Edge Pruning (WEP)** discards all edges of the blocking graph that have a weight lower than a global weight threshold (the average edge weight in our case).
- **Weight Node Pruning (WNP)** applies a local weight threshold to the neighborhood of each node, discarding those adjacent edges with a weight lower than it.

4. SUPERVISED META-BLOCKING

We consider that a comparison between profiles p_i and p_j can be captured by a feature vector $f_{i,j} = [a_1(p_i, p_j), a_2(p_i, p_j), \dots, a_n(p_i, p_j)]$, where $\{a_1, a_2, \dots, a_n\}$ is a set of features, and $a_k(p_i, p_j)$ ($k = 1 \dots n$) is the value of feature a_k for this pair. For instance, the number of common blocks the adjacent profiles share could be such a feature. By replacing edge weights with feature vectors, we extend the weighted blocking graph \mathcal{G}_B into the *generalized blocking graph* G_B , formally defined as follows:

DEFINITION 2 (GENERALIZED BLOCKING GRAPH). *Given a block collection B , its generalized blocking graph is a graph $G_B = \{V_B, E_B, F_B\}$, where V_B is the set of nodes such that $\forall p_i \in B \exists n_i \in V_B$, $E_B \subseteq V_B \times V_B$ is the set of undirected edges between all pairs of co-occurring entity profiles in B , and F_B is the set of feature vectors that are assigned to every edge such that $\forall e_{i,j} \in E_B \exists f_{i,j} \in F_B$.*

The elements of F_B are fed to a classifier that labels all edges of the blocking graph as `likely_match` or `unlikely_match`, if they are highly likely to connect two matching or non-matching entity profiles, respectively. We measure the performance of this process using the following notation:

- $TP(E_B)$ denotes the true positive edges of E_B , which connect matching profiles and are correctly classified as `likely_match`.
- $FP(E_B)$ are the false positive edges of E_B , which are adjacent to non-matching profiles, but are classified as `likely_match`.

- $TN(E_B)$ are the true negative edges of E_B , which connect non-matching profiles and are correctly categorized as `unlikely_match`.
- $FN(E_B)$ are the false negative edges of E_B , which connect matching profiles, but are categorized as `unlikely_match`.

After classifying all edges, supervised meta-blocking derives the pruned blocking graph G_B^{cl} by discarding those edges labeled as `unlikely_match` (i.e., $TN(E_B)$ and $FN(E_B)$). The edges retained in G_B^{cl} belong to the sets $TP(E_B)$ and $FP(E_B)$: $E_B^{cl} = TP(E_B) \cup FP(E_B) = E_B - (TN(E_B) \cup FN(E_B))$. The output of supervised meta-blocking is the block collection B_{cl} that is derived from G_B^{cl} by creating a block of minimum size for every retained edge $e_{i,j} \in E_B^{cl}$. Thus, its PC and PQ can be expressed in terms of the edges in E_B^{cl} as follows:

$$PC(B_{cl}) = \frac{|\mathcal{D}(B_{cl})|}{|\mathcal{D}(E)|} = \frac{|TP(E_B)|}{|\mathcal{D}(E)|} = \frac{|\mathcal{D}(B)| - |FN(E_B)|}{|\mathcal{D}(E)|},$$

$$PQ(B_{cl}) = \frac{|\mathcal{D}(B_{cl})|}{||B_{cl}||} = \frac{|TP(E_B)|}{|TP(E_B)| + |FP(E_B)|}.$$

We now formally define the task of supervised meta-blocking as:

PROBLEM 2 (SUPERVISED META-BLOCKING). *Given a redundancy-positive block collection B , its generalized blocking graph $G_B = \{V_B, E_B, F_B\}$, the classes $C = \{\text{likely_match}, \text{unlikely_match}\}$, and a training set $E_{tr} = \{ \langle e_{i,j}, c_k \rangle : e_{i,j} \in E_B \wedge c_k \in C \}$, the goal of supervised meta-blocking is to learn a classification model that minimizes the cardinality of the sets $FN(E_B)$ and $FP(E_B)$ so that the block collection B_{cl} resulting from the pruned graph G_B^{cl} achieves higher precision than B (i.e., $PQ(B_{cl}) \gg PQ(B)$), while maintaining the original recall (i.e., $PC(B_{cl}) \approx PC(B)$).*

4.1 Classification Algorithms

In principle, any algorithm for supervised learning can be used for edge classification in supervised meta-blocking. However, it should have a limited overhead for correctly categorizing most edges of the blocking graph. Further, it should be compatible with the pruning algorithm at hand. Supervised meta-blocking learns global pruning models that apply to the entire blocking graph and not to a specific neighborhood. Thus, it can be applied to *CEP*, *CNP* and *WEP*, substituting their thresholds with a classification model. In the first two cases, though, the output of the classification model should sort the edges of the blocking graph in ascending order of the likelihood that they belong to the class `likely_match`. Given that most classifiers simply produce a category label for every instance, this is only possible with probabilistic classifiers: they associate every instance with the probability that it belongs to every class, thus enabling their sorting. Note, though, that supervised meta-blocking is not compatible with *WNP*: applying a global threshold or classification model to *WNP* renders it equivalent to *WEP*.

Based on the above, we have selected four state-of-the-art approaches that are commonly used in classification tasks [26]: (i) Naïve Bayes (NB), (ii) Bayesian Networks (BN), (iii) C4.5 decision trees, and (iv) Support Vector Machines (SVM). For their implementation, we used the open-source library WEKA, version 3.6. Unless stated otherwise, we employ their default configuration, as provided by WEKA.

These approaches encompass two probabilistic classification algorithms that are compatible with *CEP* and *CNP*, namely Naïve Bayes and Bayesian Networks. In addition, they involve functionalities of diverse sophistication. On the one extreme, SVM involves complex statistical learning, while on the other extreme, Naïve Bayes relies on simple probabilistic learning. The latter actually operates as a benchmark for deciding whether the additional computational cost of the advanced classifiers pays off: comparable performance across all algorithms provides strong indication for the robustness of our classification features.

To solve the supervised meta-blocking problem, we need to determine the features to annotate the edges of the blocking graph (Section 6) and the appropriate training set, both in terms of size and composition (Section 7). In Section 5, we introduce the datasets and metrics to be used for the evaluation of the proposed solution.

5. DATASETS & METRICS

Datasets. We consider both Clean-Clean and Dirty ER and we employ the real-world datasets used in the earlier meta-blocking work [22]. Table 2 summarizes the characteristics of the entity collections and their blocks for each dataset.

D_{movies} is a collection of 50,000 entities shared among the individually clean sets of IMDB and DBPedia movies. The ground truth for this dataset stems from the “imdbid” attribute in the profiles of the DBPedia movies. Its blocks were created using Token Blocking (cf. Section 2) in conjunction with Block Purging, which discards blocks containing more comparisons than a dynamically determined threshold [21]. The resulting block collection exhibits nearly perfect recall at the cost of 27 million comparisons. Out of them, 22 million comparisons are non-redundant, forming the edges of the blocking graph.

Our second Clean-Clean ER dataset, $D_{infoboxes}$, consists of two different versions of the DBPedia Infobox dataset¹. They contain all name-value pairs of the infoboxes in the articles of Wikipedia’s English version, extracted at October 2007 for $DBPedia_1$ and October 2009 for $DBPedia_2$. The large time period that intervenes between the two collections renders their resolution challenging, since they share only 25% of all name-value pairs among the common entities [21]. As matching entities, we consider those with the same URL. For the creation of blocks, we applied Token Blocking and Block Purging. The resulting block collection entails 40 billion comparisons; 34 billion of them are non-redundant.

Finally, our Dirty ER dataset D_{BTC09} comprises more than 250,000 entities from the Billion Triple Challenge 2009 (BTC09) data collection². Its ground-truth consists of 10,653 pairs that were identified through their identical value for at least one inverse functional property. Its blocks correspond to a subset of those derived by applying the *Total Description* approach [20] (cf. Section 2) to the entire BTC09 data collection (see [22] for more details on how we selected this subset). They achieve very high PC at the cost of 130 million comparisons, out of which 78 million are non-redundant.

Metrics. To assess the impact of supervised meta-blocking on *blocking effectiveness*, we consider the relative reduction in PC , formally defined as: $\Delta PC = \frac{PC(B_{cl}) - PC(B)}{PC(B)} \cdot 100\%$, where $PC(B)$ and $PC(B_{cl})$ denote the recall of the original and the restructured block collection, respectively. Negative values indicate that meta-blocking reduces PC , while positive ones indicate higher recall.

To assess the impact of supervised meta-blocking on *blocking efficiency*, we use the following metrics:

- *Classification time CT* is the average time (in milliseconds) required by the learned model to categorize an individual edge of the blocking graph – excluding the time to build its feature vector.
- *Overhead time OT* is the total time required by meta-blocking to process the input blocks, i.e., to train the model, build the feature vectors of all edges, classify them and produce the new blocks.
- *Resolution time RT* is the sum of OT and the time required to execute all comparisons that are classified as `likely_match` using an entity matching technique. As such, we employ the Jaccard similarity of all tokens in the values of two entity profiles – regardless of the associated attribute names.

¹<http://wiki.dbpedia.org/Datasets>

²<http://km.aifb.kit.edu/projects/btc-2009>

	D_{movies}		$D_{infoboxes}$		D_{BTC09}
	DBP	IMDB	DBP ₁	DBP ₂	
Entities	27,615	23,182	1,19·10 ⁶	2,16·10 ⁶	253,353
Name-Value Pairs	186,013	816,012	1,75·10 ⁷	3,67·10 ⁷	1,60·10 ⁶
Existing Matches		22,405		892,586	10,653
Blocks		40,430		1,21·10 ⁶	106,462
PC		99.39%		99.89%	96.94%
Comparisons in Blocks		2,67·10 ⁷		3,98·10 ¹⁰	1,31·10 ⁸
Brute-force RT		26 min		~320 hours	64 min
Edges		2,26·10 ⁷		3,41·10 ¹⁰	7,77·10 ⁷
Nodes		5,06·10 ⁴		3,33·10 ⁶	2,53·10 ⁵

Table 2: Overview of the real-world datasets.

- CMP denotes the absolute number of comparisons contained in the restructured block collection (i.e., $\|B_{cl}\|$).

For these metrics, the lower their value is, the higher is the efficiency of meta-blocking. Note that OT and RT do not consider the time to randomly select the training set, due to its negligible computational cost (see Section 7). We also estimate efficiency using PQ , with higher values indicating more efficient meta-blocking.

6. FEATURES FOR META-BLOCKING

Features for supervised meta-blocking describe the edges of the generalized blocking graph and should pertain to the corresponding comparisons or to the adjacent entities. These features should adhere to the following principles: (i) they should be *generic*, so that they are not tailored to a specific application; (ii) they should be *effective*, so that they yield high classification accuracy distinguishing between likely and unlikely matches; (iii) they should be *efficient*, involving low extraction cost and overhead, so that the classification time is significantly lower than the comparison time of its adjacent entities, and (iv) they should be *minimal*, in the sense that incorporating additional features has marginal benefit on the performance of meta-blocking. Similar principles were defined in [3] for classification tasks related to Entity Resolution (see Section 2).

Feature Categorization. To help understand candidate features for supervised meta-blocking and their appropriateness, we divide their space along five dimensions: schema-awareness, source of evidence, target, complexity and scope (see Figure 4).

Schema awareness. Classification features can be divided into schema-agnostic and schema-based ones.

Schema-agnostic features rely on the structural information of the blocking graph and the characteristics of the blocks.

Schema-based features rely on the quality and the semantics of the attribute names that describe the input entity profiles. Thus, they exclusively consider blocks and parts of the blocking graph that are associated with specific attributes.

Source of evidence. Given a block collection B , there are two main sources for extracting classification features: the blocks contained in B and the blocking graph G_B . *Block-based* features exclusively consider evidence of the former type, while *graph-based* ones rely on topological information about the blocking graph. *Iterative* features are graph-based features associated with a node that depend on the scores assigned to its neighboring nodes. Similarly to link analysis techniques, such as PageRank, these features may be computed by assigning a prior value to every node (or edge) and iteratively adjusting it, processing the entire blocking graph according to a mathematical formula until convergence.

Target. Depending on the part of the graph they annotate, classification features are divided into *edge-specific*, which pertain to individual edges and *node-specific*, which pertain to individual nodes.

Complexity. Classification features can be categorized into *raw* and *derived* ones. The raw attributes encompass atomic information that is explicitly available in the input block collection or its

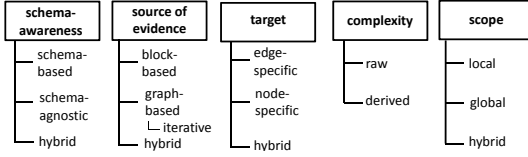


Figure 4: Categorization of classification features.

blocking graph; on the other hand, the *derived* features combine multiple features into a single, composite feature.

Scope. Classification features are *local* when they consider information that is directly related to the annotated item (i.e., the edge or its adjacent nodes). *Global* features consider information from the entire blocking graph.

Most criteria (with the exception of complexity) define two complementary categories. Thus, features from both categories can be combined into *hybrid* ones, which may exhibit higher performance.

6.1 Candidate Features

We introduce our candidate features and explain their appropriateness for our meta-blocking problem (notation is in Table 1).

Common.Blocks. A schema-agnostic, block-based feature is **Common.Blocks**, i.e., the number of blocks shared by two profiles:

$$\text{Common.Blocks}(e_{i,j}) = |B_{i,j}|.$$

This feature captures the inherent trait of redundancy-positive block collections that the more blocks two entity profiles share, the more likely they are to match.

Based on **Common.Blocks**, we could define schema-based features that take into account the subset of common blocks stemming from the values of specific attributes (e.g., $\text{Common.Blocks}_{title}$ for the attribute “Title”). However, such schema-based features are application-specific and have limited generality. In addition, they are crafted for homogeneous information spaces, like databases, and cannot handle heterogeneous ones, characterized by very diverse schemata (e.g., Web of Data) and constituting the most common source of redundancy-positive blocking [22]. Therefore, we focus hereafter on schema-agnostic classification features, which are completely decoupled from the nature and the semantics of the attributes describing the input entity collections.

Entity.Blocks. Another block-based feature is **Entity.Blocks**, which is inversely proportional to the number of blocks containing a specific entity/node:

$$\text{Entity.Blocks}(v_i) = \log \frac{|B|}{|B_i|}.$$

This feature is inspired from the inverse document frequency (IDF) that is commonly used in Information Retrieval (IR). The rationale behind it is that the higher its value is (i.e., lower $|B_i|$), the more likely p_i is to match with one of its co-occurring entity profiles. In contrast, a profile that is contained in an excessively high number of blocks is highly likely to contain noise. For instance, it could be the result of falsely merging several profiles that correspond to different real-world objects.

Node.Degree. This is a graph-based feature and it is equal to the degree of node $v_i \in V_B$:

$$\text{Node.Degree}(v_i) = |V_{v_i}|.$$

In essence, **Node.Degree** is equivalent to the number of non-redundant comparisons involving the entity p_i that corresponds to the node v_i . The intuition here is that the lower its degree is, the more likely p_i is to match with one of its co-occurring entity profiles.

Iterative.Degree. This is an iterative, graph-based feature that is based on the following premise: for every node v_i , the lower the degrees of its neighboring nodes are, the higher is the likelihood

that p_i is matching with one of them and, thus, the higher the score of v_i should be. It is similar to **Node.Degree**, as it initially associates every node with a prior score equal to the portion of non-redundant comparisons involving it (i.e., $|V_{v_i}|/|E_B|$). They differ though in that the **Iterative.Degree** gradually modifies the score of a node v_i , $ID(v_i)$, according to the following formula:

$$ID(v_i) = ID_0(v_i) + \sum_{v_k \in V_{v_i}} \frac{ID(v_k)}{|V_{v_k}|},$$

where $ID_0(v_i)$ is the prior score assigned to v_i and $v_k \in V_{v_i}$ are the nodes connected with v_i on the blocking graph. This formula is similar to the one defining PageRank with priors, where the damping factor d , which determines the behavior of the random surfer, is set equal to 0. It can be calculated using a simple iterative algorithm; after converging, nodes connected with many nodes of high degree receive the lowest scores, while the highest scores are assigned to nodes connected with few nodes of low degree.

The extraction of such iterative, graph-based features is computationally expensive and it does not scale well to blocking graphs with millions of nodes and billions of edges. Nevertheless, we include **Iterative.Degree** in our approach and we investigate whether its low efficiency is counterbalanced by high discriminatory power.

Transitive.Degree. A possible surrogate of higher efficiency is the **Transitive.Degree** feature. It lies in the middle of **Node.Degree** and **Iterative.Degree**, considering the aggregate degrees of the nodes that lie within the neighborhood of v_i as follows:

$$\text{Transitive.Degree}(v_i) = \sum_{v_k \in V_{v_i}} |V_{v_k}|.$$

Common.Neighbors. This graph-based feature amounts to the portion of adjacent entity profiles shared by a pair of co-occurring profiles. More formally, it is defined as follows:

$$\text{Common.Neighbors}(e_{i,j}) = \frac{|V_{v_i} \cap V_{v_j}|}{|V_{v_i} \cup V_{v_j}|}.$$

High values indicate that p_i and p_j co-occur with the same entities, either in their common blocks or in blocks they do not share. In the latter case, the common neighbors actually help deal with patterns missed due to noise in entity profiles. For example, consider the entity profiles $p_1 = \{<name, John>, <surname, Smith>\}$, $p_2 = \{<name, Jon>, <surname, Smith>\}$ and $p_3 = \{<name, John>, <surname, Smith>\}$, where $(p_1 \equiv p_2) \neq p_3$; Token Blocking [21] (cf. Section 2) yields two blocks $b_{John} = \{p_1, p_3\}$ and $b_{Smith} = \{p_2, p_3\}$, with p_1 and p_2 co-occurring in none of them. Nevertheless, **Common.Neighbors** provides strong evidence for their match.

Jaccard.Similarity. This feature captures the portion of all comparisons (including the redundant ones) that involve a specific pair of entity profiles:

$$\begin{aligned} \text{Jaccard.Sim}(e_{i,j}) &= \frac{|p_i.comp() \cap p_j.comp()|}{|p_i.comp() \cup p_j.comp()|} \\ &= \frac{|B_{i,j}|}{|p_i.comp()| + |p_j.comp()| - |B_{i,j}|}. \end{aligned}$$

Higher values of this ratio indicate a stronger pattern of co-occurrence for p_i and p_j and, hence, the more likely p_i and p_j are to match.

Note that on the target dimension, **Node.Blocks**, **Node.Degree**, **Iterative.Degree** and **Transitive.Degree** are node-specific, while **Common.Blocks**, **Common.Neighbors** and **Jaccard.Sim** are edge-specific features. Although the distinction between edge- and node-specific features seems trivial, there are two major qualitative differences between them. First, a feature vector has to include two values for every node-specific feature – one for each of the adjacent entities – thus broadening the search space by two dimensions. In contrast, edge-specific attributes are computed only once per feature vector, adding a single dimension to the search space. Second,

	source of evidence			target			complexity		scope		
	block-based	graph-based	iterative	edge-specific	node-specific	hybrid	raw	derived	local	global	hybrid
CF_IBF	✓					✓		✓			✓
Jaccard_Sim	✓			✓				✓	✓		
RACCB	✓							✓		✓	
Node_Degree		✓			✓		✓		✓		
Iterative_Degree			✓		✓			✓		✓	
Transitive_Degree		✓			✓			✓		✓	

Figure 5: Categorization of the features of our approach.

edge-specific features are expected to exhibit higher discriminatory power than the node-specific ones, because every value of the latter participates in as many feature vectors as the degree of the corresponding node; in contrast, every value of the edge-specific features pertains to a single feature vector.

(Reciprocal) Aggregate Cardinality of Common Blocks. From the aforementioned features, only *Common.Blocks* and *Node.Degree* are raw. In general, there is no rule-of-thumb for a-priori determining which form of features achieves the best performance in practice. Even different forms of derived features may lead to significant differences in classification accuracy. As an example, consider two edge-specific features that use the same information, but in different forms: the *Aggregate Cardinality of Common Blocks* (ACCB) and the *Reciprocal Aggregate Cardinality of Common Blocks* (RACCB) attributes. The former sums the cardinalities of the blocks shared by the adjacent entities (raw feature): $ACCB(e_{i,j}) = \sum_{b_k \in B_{i,j}} \|b_k\|$. The latter sums the inverse of the cardinalities of common blocks (derived feature):

$$RACCB(e_{i,j}) = \sum_{b_k \in B_{i,j}} \frac{1}{\|b_k\|}.$$

Both features rely on the premise that the smaller the blocks two entities co-occur in, the more likely they are to be matching. Hence, the lower the value of ACCB is, the more likely the co-occurring entities match, and vice versa for RACCB. Preliminary experiments, though, demonstrated that ACCB achieves significantly lower classification accuracy than RACCB, due to its low discriminativeness: it assigns identical or similar values to pairs of co-occurring entities that share blocks of totally different cardinalities. For instance, consider two pairs of entities: the first co-occurs in 3 blocks containing 1, 2 and 4 comparisons, while the second shares 2 blocks with 2 and 5 comparisons; ACCB takes the same value for both edges (7), while RACCB amounts to 1.75 and 0.70 for the first and the second pair, respectively, favoring the entities that are more likely to match.

Co-occurrence Frequency-Inverse Block Frequency. Derived features can come in more advanced forms than RACCB, combining multiple features through linear or non-linear functions. However, they should be used with caution for several reasons: (i) they involve a higher extraction cost than the comprising raw features; (ii) their form might be too complex to be interpretable; (iii) they are usually correlated with the raw features they comprise and, thus, are incompatible with them, when applied to classifiers with strong independence assumptions (e.g., Naïve Bayes); (iv) some classification algorithms may operate better with raw features, learning themselves the linear or non-linear associations between the input features. For these reasons, the derived features should involve a low extraction cost and transform as few raw features as possible.

Here, we combine *Common.Blocks* and *Entity.Blocks* into a feature inspired from the TF-IDF measure employed in IR. We call it Co-occurrence Frequency-Inverse Block Frequency (CF_IBF) and formally define it as:

$$CF_IBF(e_{i,j}) = |B_{i,j}| \cdot \log \frac{|B|}{|B_i|} \cdot \log \frac{|B|}{|B_j|}.$$

FS 1:	CF_IBF,RACCB,Node_Degree
FS 2:	CF_IBF,RACCB,Transitive_Degree
FS 3:	CF_IBF,RACCB,Node_Degree,Transitive_Degree
FS 4:	CF_IBF,RACCB,Jaccard_Sim,Transitive_Degree
FS 5:	CF_IBF,RACCB,Jaccard_Sim,Node_Degree
FS 6:	CF_IBF,RACCB,Transitive_Degree,Iterative_Degree
FS 7:	CF_IBF,RACCB,Node_Degree,Iterative_Degree
FS 8:	CF_IBF,RACCB,Jaccard_Sim,Node_Degree,Transitive_Degree
FS 9:	CF_IBF,RACCB,Jaccard_Sim,Node_Degree,Iterative_Degree
FS 10:	CF_IBF,RACCB,Jaccard_Sim,Transitive_Degree,Iterative_Degree

Table 3: Top 10 feature sets selected.

Experiments showed that this form outperforms the individual features, because *Entity.Blocks* is of limited usefulness when used independently, but it is valuable for extending *Common.Blocks*, which otherwise suffers from low discriminativeness (it amounts to 1 or 2 for the vast majority of edges). In this way, we also restrict the dimensionality of our approach by two degrees.

Approach Overview. Our approach considers all the candidate features, except for ACCB due to low discriminativeness, and *Common.Neighbors*, which violates the requirement for generality, as it does not apply to *Clean-Clean ER*. In this case, the resulting blocking graph is bipartite, since every entity of the one collection is exclusively connected with entities from the other collection (only comparisons between different collections are allowed) [22]. As a result, every pair of co-occurring entities shares no neighbors.

On the whole, our approach annotates every edge of the blocking graph with the following nine-feature vector:

[CF_IBF($e_{i,j}$), Jaccard_Sim($e_{i,j}$), RACCB($e_{i,j}$), Node_Degree(v_i), Node_Degree(v_j), Iterative_Degree(v_i), Iterative_Degree(v_j), Transitive_Degree(v_i), Transitive_Degree(v_j)].

We selected these features because they cover all feature categories (as illustrated in Figure 5), they are schema-agnostic, applying to any block collection (generality principle), and they form a limited search space that allows for rapidly training classification models of low complexity and overhead (efficiency principle). Most of them also involve a low extraction cost.

Note also that most of the aforementioned features are local with respect to their scope; the exceptions are *Entity.Blocks*, which involves a hybrid functionality that combines local with global information, and *Iterative_Degree* and *Transitive_Degree*, which consider global information about the neighbors of a specific node. In the general case, local features are expected to exhibit higher effectiveness and efficiency than the global ones, because the latter consider more general information and convey a higher extraction cost. As an example, consider a boolean global feature that sorts all edges of E_B in descending order of *Common.Blocks* and returns true for those ranked in the top 1% positions and false otherwise.

6.2 Feature Selection

To satisfy the minimality principle, we examine the relative performance of each combination of features, called *feature set* (FS), in order to identify the one achieving the best balance between effectiveness and efficiency. There is a clear trade-off here: fewer features mean less complex and less time-consuming learned model (higher efficiency), but lower effectiveness.

The selected features yield 63 combinations. Due to their high number, our feature selection process has two phases. *First*, we extracted the top 10 performing feature sets automatically. *Then*, we selected the best set by examining their relative performance analytically. We use all four classification algorithms over D_{movies} . The models were trained using 1,000 labeled edges, equally partitioned between matching and non-matching edges, that were randomly se-

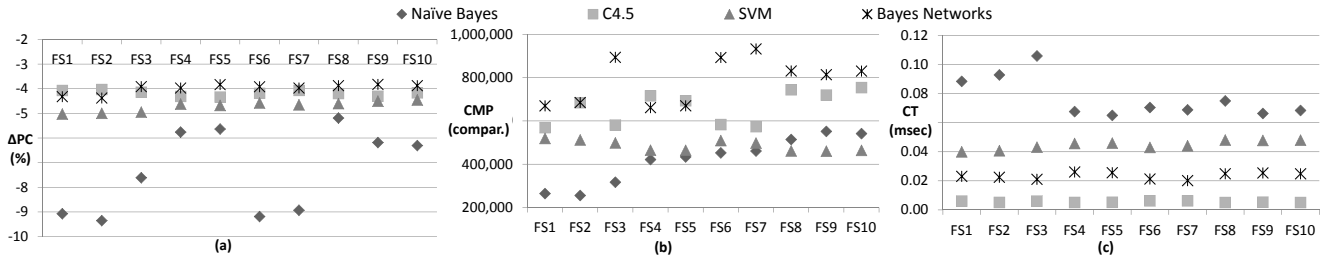


Figure 6: Performance of the feature sets in Table 3 over D_{movies} with respect to (a) the relative reduction in recall (ΔPC), (b) the absolute number of retained comparisons (CMP) and (c) the classification time of an individual edge (CT).

lected from the entire blocking graph. The remaining edges formed the disjoint testing set. To derive accurate performance estimations, we repeated this process 10 times and considered the average value of each metric.

To identify the top 10 performing feature sets, we sort the feature sets in descending order of their total F-measure and select those placed in the top 10 positions. The *Total F-measure* ($TF1$) of a feature set FS_i is the sum of the F-measures corresponding to each classification algorithm: $TF1(FS_i) = \sum_{j \in \{NB, C4.5, SVM, BN\}} F1_j(FS_i)$, where the F-measure for a feature set FS_i and an algorithm j is computed as³: $F1_j(FS_i) = 2 \cdot PC \cdot nPQ / (PC + nPQ)$, with nPQ denoting the normalized Pairs Quality across all feature sets for the algorithm j (i.e., $nPQ_j(FS_i) = \frac{PQ_j(FS_i)}{\max(PQ_j(FS_1), PQ_j(FS_2), \dots, PQ_j(FS_{63}))}$).

The resulting top 10 feature sets are shown in Table 3. Collectively, they involve all features of our approach, a strong indication for the high utility of each feature. Moreover, each feature set comprises at least 3 features, revealing that our features are compatible and complementary, working best when used in conjunction.

To select the best feature set out of the top 10 performing ones, we evaluate their effectiveness through ΔPC and their efficiency through CMP and CT (see Section 5). Figures 6(a), (b) and (c) present ΔPC , CMP and CT , respectively. The horizontal axes correspond to the feature sets. We can identify the optimal feature set by examining their relative performance across the three figures. The closer a feature set is placed to the horizontal axis of each figure across all classification algorithms, the better is its performance.

Figure 6(a) shows that most feature sets exhibit limited variation in ΔPC (between -3.5% and -5% for most algorithms). Only Naïve Bayes is rather unstable, yielding five outliers: $FS1$, $FS2$, $FS3$, $FS6$ and $FS7$ have an unacceptable impact on recall (over -7%) and hence they are not considered any further.

For the remaining sets, Figures 6(a) and 6(b) reveal a trade-off between ΔPC and CMP : the higher ΔPC for a particular feature set and classification algorithm, the lower CMP gets, and vice versa. Hence, none of the feature sets excels in both metrics. To identify the set with the best balance between ΔPC and CMP , we consider their average values across all classifiers. Only $FS4$ and $FS5$ achieve the most stable performance: $FS4$ requires $5.66 \pm 1.25 \times 10^5$ comparisons, and $FS5$ gives $5.65 \pm 1.17 \times 10^5$. They are also the most efficient compared to $FS8$, $FS9$ and $FS10$, which require 12.5% more comparisons, on average.

Finally, to decide between $FS4$ and $FS5$, we compare them in terms of CT . Figure 6(c) shows negligible differences (in absolute

numbers) between them – in the order of 1/100 of a millisecond. Given, though, that CT concerns a single edge, these differences become significant when meta-blocking is applied to large blocking graphs with millions of edges. We choose $FS4$ by 3%, on average.

In the following, we exclusively consider the feature set $FS5$, comprising CF_IBF , $RACCB$, $Jaccard_Sim$ and $Node_Degree$, which combines a low extraction cost with high discriminatory power.

7. TRAINING SET

The quality of the learned classification model also depends on the training set and in particular on its composition and size.

The definition of supervised meta-blocking (Problem 2) makes no assumptions about the training set. However, the vast majority of the edges in the blocking graph connect non-matching entities and thus correspond to superfluous comparisons. If the training set involves the same class distributions as the set of edges, E_B , it will be heavily imbalanced in favor of the *unlikely_match* class. As a result, the classifier would be biased towards assigning every instance to the majority class, pruning most of the edges.

This situation constitutes a *class imbalance problem*, which is typical in supervised learning (as an example, consider the task of spam filtering, where the vast majority of e-mails is not spam) with several solutions [15]: *oversampling* randomly replicates instances of the minority class until the class distribution is balanced, *cost-sensitive learning* incorporates high misclassification cost for the minority class when training the classifier, and *ensemble learning* trains a set of classifiers that collectively take classification decisions through a form of weighted voting. Unfortunately, cost-sensitive and ensemble learning increase the complexity of the classification model, while oversampling yields excessively large training sets prone to overfitting (too many repetitions of the same instances). Instead, we use *undersampling*, which randomly selects a subset of the same cardinality from both classes. The training set is equally partitioned between *likely_match* and *unlikely_match* edges. This approach is best for small training sets, which can be manually labeled in the absence of ground truth.

The size of the training set, called *sample size*, affects both the effectiveness and the efficiency of supervised meta-blocking: the smaller the sample size is, the lower is the complexity of the learned model and the more efficient is its training and its use. However, this comes at the cost of lower classification accuracy, as the simpler the learned model is, the more likely it is to miss patterns. To identify the break-even point in this trade-off, we experimentally perform a sensitivity analysis for the sample size with respect to effectiveness (ΔPC) and efficiency (CT and CMP).

Training Set Selection. We apply the selected feature set to the four classifiers over D_{movies} using training sets of various sizes. Due to undersampling, these training sets were specified in terms of the minority class cardinality (i.e., the number of matching entities in

³Note that the F-measure for blocking-based ER is defined as $F1 = \frac{2 \cdot PC \cdot PQ}{PC + PQ}$ [4]. We employ nPQ instead of PQ , because the latter takes very low values for redundancy-positive block collections. In fact, PQ is lower than PC by one or two orders of magnitude, thus dominating $F1$, which ends up assigning high scores to feature sets with a few comparisons and a few detected duplicates.

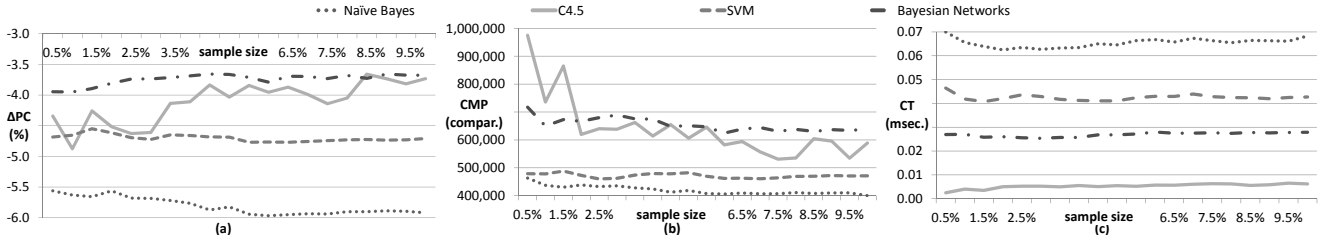


Figure 7: Learning curves of our approach over D_{movies} with respect to (a) ΔPC , (b) CMP and (c) CT . The horizontal axes correspond to the sample size, i.e., the number of training instances expressed as a portion of the minority class cardinality.

the ground truth). Each one was equally partitioned between (randomly selected) matching and non-matching edges, whose number ranged from 0.5% to 10% of the minority class size, with a step of 0.5%. For every sample size, we repeated the process 10 times and considered the average of the aforementioned metrics. Figures 7(a), (b) and (c) depict the learning curves with respect to ΔPC , CMP and CT , respectively.

Figure 7(a) shows that most classifiers exhibit rather stable recall, with a variance at most 1.2%. In all cases, the variance in ΔPC gets lower as the sample size increases. Especially for sample sizes that exceed 5% of the minority class cardinality (i.e., around 1,100 labeled instances per class), there is no variance in practice. These patterns demonstrate that our proposed feature set is comprehensive, robust enough and effective even when trained over small training sets – regardless of the classification algorithm.

Figure 7(b) shows the evolution of CMP with the increase in sample size. Most classifiers start from high values, but gradually converge to lower values as the sample size increases. Similar to ΔPC , the variance in CMP decreases in proportion to the sample size and becomes negligible for sample sizes larger than 5% of the minority class. Regarding CT , Figure 7(c) shows that all classifiers exhibit a relatively stable, good performance regardless of the sample size. The average CT is close to the time observed for the sample size equal to 5%.

Consequently, a sample size equal to 5% of the minority class achieves a performance equivalent to that of much larger ones. In the following, we exclusively consider training sets comprising 5% of the edges labeled as `likely_match` and an equal number of edges labeled as `unlikely_match`.

8. CLASSIFIERS CONFIGURATION

The performance of the selected classification algorithms depends on their internal parameter configuration. Fine-tuning may significantly enhance classification accuracy, but it may also lead to over-fitting, which increases the complexity of the learned model and inflates the overhead of meta-blocking. To assess how their configuration affects the performance of our approach, we perform analytical fine-tuning of their parameters. For each algorithm, we examine two parameters that are fine-tuned in parallel:

- **C4.5:** we study the maximum number of instances per leaf node, ranging from 2 to 5, and the confidence interval, ranging from 0.1 and 0.5 with a step of 0.05 (36 configurations in total). By default, Weka sets the former parameter to 2 and the latter to 0.25.
- **SVM:** we consider two kernels, the linear and the RBF, and we vary the complexity parameter C from 1 to 10 with a step of 1 (20 configurations in total). Weka’s default configuration incorporates a linear kernel function with the complexity constant C set to 1.
- **Bayesian Networks:** we use three search algorithms: simulated annealing, hill climbing and genetic search. We use each one with global and with local scope (6 configurations in total). The default

configuration of Weka uses the hill climbing search of local scope.

- **Naïve Bayes:** two boolean parameters that determine the processing of numeric attributes, i.e., use of supervised discretization and of kernel density estimator (4 configurations in total). By default, Weka sets both parameters to `false`.

For each classifier, we apply every possible configuration to 10 randomly selected training sets from the blocking graph of D_{movies} using the sample size and the features determined above. Due to the large number of configurations, we consider only the default, the optimal and the average performance for each classification algorithm and metric. As optimal, we define the configuration with the largest F-measure (again, $F1$ employs nPQ instead of PQ). The following configurations were selected in this way: the use of both supervised discretization and kernel estimator for Naïve Bayes; 5 instances per leaf and confidence interval equal to 0.1 for C4.5; linear kernel with $C=9$ for SVM; simulated annealing with global scope for Bayesian Networks. Figures 8(a) to (c) depict the experimental outcomes with respect to ΔPC , CMP and CT , respectively.

We first investigate the relative *performance* of the default and the optimal configuration. For Naïve Bayes, the optimal one puts more emphasis on effectiveness, increasing ΔPC by executing more comparisons. However, its overall efficiency is significantly increased, as its overhead (CT) is reduced to 1/5. For C4.5 and SVM, the optimal configurations decrease CMP by 5%, while exhibiting practically identical ΔPC and CT with the default ones. On the other hand, the optimal configuration for the Bayesian Networks reduces CMP by 20% for almost the same PC as the default one, but puts a toll on efficiency: CT increases by 25%. Hence, we choose the default configuration for Bayesian Networks, while for the other algorithms we choose the optimal ones, due to their slightly better balance between effectiveness and efficiency.

We now examine the *robustness* of the classification algorithms with respect to their configuration based on the distance between the default, the optimal and the average performance for all configurations. We observe that C4.5 is practically insensitive to fine-tuning, despite the large numbers of configurations considered. The same applies to SVM with respect to ΔPC and CMP ; its average CT (2.66 msec), though, is two orders of magnitude higher than the default and the optimal one. This is because the RBF kernels are 10 times slower when classifying an individual edge than the linear ones, which exhibit a rather stable CT . A similar situation appears in the case of Naïve Bayes, where the average CT amounts to 3.80 msec, due to the inefficiency of a single configuration: supervised discretization for numeric attributes without the kernel estimator. The other two metrics, though, advocate that Naïve Bayes is rather sensitive to its configuration. Finally, the Bayesian Networks exhibit significant variance with respect to CMP and CT , but the overall efficiency is relatively stable across all configurations. We can conclude that for the selected feature set and sample size, most classifiers are rather robust with respect to their configuration.

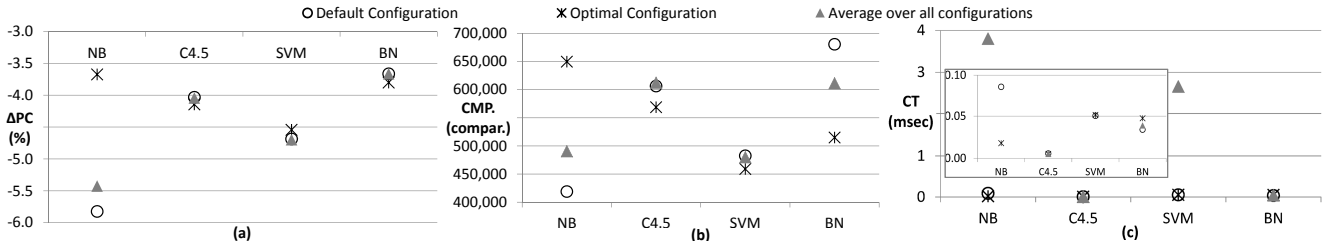


Figure 8: Effect of parameter configuration for each classification algorithm over D_{movies} with respect to (a) ΔPC , (b) CMP and (c) CT . The sub-figure in (c) zooms into the interval $[0, 0.1]$ to highlight differences in CT that are concealed in the interval $[0, 4]$.

9. EXPERIMENTAL EVALUATION

We now compare the performance of our supervised meta-blocking techniques with the best performing unsupervised ones over three pruning algorithms: *WEP*, *CEP* and *CNP*. Remember that *WEP* is compatible with all classification algorithms, while *CEP* and *CNP* are only compatible with Naïve Bayes and Bayesian Networks. To compare the supervised and the unsupervised techniques on an equal basis, we adapted the latter so that they exclude the edges used for training by the former. Hence, we applied them 10 times to each dataset and derived their performance from the average of the relevant metrics (this explains why in some cases their performance is slightly different from that reported in [22]).

We also employ the state-of-the-art approach of Iterative Blocking [25] as an additional baseline method. Given that its performance depends heavily on the processing order of blocks, we apply it to 10 random shuffles of each dataset’s blocks and present the average value of each metric. Note that for Clean-Clean ER, we consider the best possible performance of iterative blocking, assuming that all pairs of detected matches are propagated to the subsequently processed blocks so that their entities do not participate in any other comparison.

We implemented our approaches in Java 1.7⁴ and tested them on a server with Intel i7-4930K 3.40GHz and 32GB RAM, running Debian 7. Graphs were implemented using the JUNG framework⁵.

9.1 In-depth Analysis over Real Datasets

Table 4 presents the performance of the baseline⁶ and our supervised techniques with respect to the three pruning algorithms over the datasets of Table 2. For each dataset, we considered the unsupervised meta-blocking in conjunction with the weighting scheme that yields the best trade-off between PC and PQ ; for D_{movies} and $D_{infoboxes}$, we used the weighting schemes *CBS*, *EJS* and *ECBS* for *WEP*, *CEP* and *CNP*, respectively, while for D_{BTC09} , we employed the *ARCS* scheme in all cases (see [22] for more details). We now examine each pruning algorithm separately.

WEP. We observe that supervised meta-blocking consistently achieves a better balance between effectiveness and efficiency over D_{movies} and $D_{infoboxes}$. It executes almost an order of magnitude fewer comparisons than the unsupervised method with a minor increase of PC . As a result, precision consistently increases by at least 4 times. The higher overhead (OT) is counterbalanced by

the considerably lower number of comparisons, resulting overall in significantly improved resolution times (RT).

For D_{BTC09} , supervised meta-blocking improves efficiency to a similar extent at the cost of slightly lower recall (the only exception is SVM, whose PC is significantly lower than the unsupervised method by 7.5%). Both the number of comparisons and the overhead time are almost half, leading to significantly better RT .

CEP. For D_{movies} and $D_{infoboxes}$, supervised meta-blocking achieves significantly higher recall, increased by more than 10%. Its overhead time, though, is more than twice that of unsupervised meta-blocking. Given that both approaches execute the same number of comparisons, the classification models exhibit notably increased resolution time. In the case of D_{BTC09} , supervised meta-blocking decreases PC and PQ to a minor extent, while increasing the resolution time by 25%. For each dataset, these patterns are consistent across both probabilistic models.

CNP. For D_{movies} and D_{BTC09} , supervised meta-blocking reduces the number of executed comparisons to a significant extent, at the cost of a lower PC . PQ almost doubles, but the higher overhead than unsupervised meta-blocking leads to an increased resolution time. The same applies to $D_{infoboxes}$, as both OT and RT are significantly higher than unsupervised meta-blocking. In this case, though, the number of comparisons is practically the same, while PQ gets slightly higher, because PC slightly increases.

Iterative Blocking. We observe that iterative blocking achieves the lowest overhead time and the highest recall across all datasets: for the Clean-Clean ER datasets D_{movies} and D_{BTC09} , PC is equal to that of the input block collection, while for the Dirty ER dataset (D_{BTC09}), it increases by 1%. However, this comes at the cost of rather low efficiency; iterative blocking actually executes so many comparisons that its resolution time is practically identical with the brute-force approach of performing all comparisons in the input block collection. For Clean-Clean ER, its run-time lies in the middle between supervised and unsupervised meta-blocking, due to the ideal settings we consider (i.e., none of the matched entities participates in any comparison after their detection). In a more realistic scenario, though, its efficiency is expected to be lower than that of unsupervised meta-blocking. We can conclude, therefore, that Iterative Blocking is only appropriate for applications that place recall in priority and are satisfied with rather conservative savings in efficiency. For the rest of them, supervised meta-blocking offers a better balance between effectiveness and efficiency.

Conclusions. For *WEP*, our techniques leverage small training samples and feature vectors to significantly increase efficiency at a negligible cost in effectiveness (if any). This consistent behavior is important, since *WEP* is compatible with practically any blocking-based ER application. It stems from the low computational cost and the comprehensiveness of our features. The latter aspect can be inferred from the performance of Naïve Bayes, which is directly comparable with the more complicated algorithms in all cases. The best performance, though, is achieved when combining supervised

⁴We have publicly released the code of our implementations at <http://sourceforge.net/projects/erframework>.

⁵<http://jung.sourceforge.net>

⁶For unsupervised meta-blocking, OT measures the time required for the creation of the weighted blocking graph and the pruning of its edges. For iterative blocking, OT estimates the time required for the propagation of the detected duplicates to the subsequently processed blocks and the re-processing of the related blocks (in case of Dirty ER).

	D_{movies}						D_{infoboxes}						D_{BTC09}					
	<i>CMP</i> ($\times 10^5$)	<i>PQ</i> (%)	<i>PC</i> (%)	ΔPC (%)	<i>OT</i> (sec)	<i>RT</i> (sec)	<i>CMP</i> ($\times 10^8$)	<i>PQ</i> (%)	<i>PC</i> (%)	ΔPC (%)	<i>OT</i> (hours)	<i>RT</i> (hours)	<i>CMP</i> ($\times 10^6$)	<i>PQ</i> (%)	<i>PC</i> (%)	ΔPC (%)	<i>OT</i> (min)	<i>RT</i> (min)
Brute-force	20.27	1.09	99.39	0	1	89	40.46	0.02	99.89	0	1	34	123.62	0.01	98.22	1.32	1	62
Iterative Blocking	20.27	1.09	99.39	0	1	89	40.46	0.02	99.89	0	1	34	123.62	0.01	98.22	1.32	1	62
Unsupervised	27.04	0.75	94.64	-4.78	13	104	33.97	0.02	95.47	-4.43	12	41	4.14	0.23	94.63	-2.40	6	8
Naive Bayes	6.50	3.14	95.74	-3.67	39	56	3.76	0.22	99.09	-0.80	27	31	2.10	0.45	93.46	-3.58	4	5
C4.5	5.69	3.57	95.27	-4.15	20	40	2.98	0.28	99.00	-0.89	19	21	1.81	0.53	94.01	-3.02	3	4
SVM	4.59	4.40	94.87	-4.54	35	50	4.54	0.18	97.30	-2.60	27	31	2.55	0.35	87.49	-9.75	4	5
Bayesian Networks	6.51	3.13	95.75	-3.66	33	51	3.76	0.22	99.09	-0.80	25	29	2.12	0.45	93.50	-3.54	4	5
(a) WEP																		
Unsupervised	5.70	3.17	84.89	-14.59	9	23	0.26	2.72	82.09	-17.82	11	12	0.94	0.99	92.03	-5.06	3	4
Naive Bayes	5.69	3.56	95.34	-4.08	39	55	0.26	3.06	92.58	-7.32	26	27	0.94	0.96	89.47	-7.70	4	5
Bayesian Networks	5.69	3.56	95.35	-4.07	34	49	0.26	3.08	92.70	-7.20	23	24	0.94	0.96	89.49	7.68	4	5
(b) CEP																		
Unsupervised	11.00	1.87	96.67	-2.74	8	45	0.49	1.64	96.68	-3.21	12	13	1.75	0.53	90.86	-6.27	3	4
Naive Bayes	7.22	2.82	95.46	-3.95	39	59	0.47	1.79	98.38	-1.51	26	27	1.00	0.88	87.40	-9.84	4	5
Bayesian Networks	7.23	2.81	95.47	-3.94	34	54	0.47	1.78	98.38	-1.51	23	24	1.01	0.87	87.41	-9.83	4	5
(c) CNP																		

Table 4: Performance of supervised meta-blocking and the baselines over all datasets with respect to (a) WEP, (b) CEP, (c) CNP.

	Entity Collections		Block Collections			
	Entities	Duplicates	Blocks	Compar.	PC	Brute-force RT
D_{10K}	10,000	8,615	11,088	3.35×10^5	98.97%	4 sec
D_{50K}	50,000	42,668	40,569	7.42×10^6	98.77%	75 sec
D_{100K}	100,000	84,663	72,733	2.91×10^7	98.73%	5 min
D_{200K}	200,000	170,709	123,648	1.19×10^8	99.02%	23 min
D_{300K}	300,000	254,686	166,099	2.70×10^8	99.09%	45 min
D_{1M}	1,000,000	849,276	441,999	2.94×10^9	99.04%	8 hrs
D_{2M}	2,000,000	1,699,430	863,528	1.17×10^{10}	99.03%	33 hrs

Table 5: Overview of the synthetic datasets.

meta-blocking with C4.5, which reduces the resolution time by 50% across all datasets for practically the same effectiveness.

With respect to *CEP*, which is only suitable for incremental ER, unsupervised meta-blocking exhibits significantly higher efficiency, due to its lower overhead. However, the high *OT* time of the classification models is rendered insignificant, when advanced, time-consuming entity matching methods are used. Then, supervised meta-blocking should be preferred due to its consistently higher recall. For the same reason, it should be used with all applications of incremental ER that place more emphasis on effectiveness.

For *CNP*, we cannot draw any safe conclusions, due to the unstable performance of supervised meta-blocking across the 3 datasets, caused by the incompatibility of its global training information with the local scope of this pruning algorithm. Finally, it is worth stressing that supervised meta-blocking consistently improves the run-time of the brute-force approach by at least 10 times (cf. Table 2).

9.2 Recall and Run-time Scalability

We now examine the scalability of our supervised techniques in relation to the three pruning algorithms. We apply them to seven synthetic datasets that were created by FEBRL [5] and have been widely used in the literature for this purpose [4, 13]. They pertain to Dirty ER and their sizes range from 10 thousand to 2 million entities. To derive redundancy-positive blocks, we applied Token Blocking and Block Purging to each dataset. The technical characteristics of the resulting block collections are presented in Table 5.

As baseline methods, we employ iterative blocking and unsupervised meta-blocking. The latter was combined with the *ECBS* weighting scheme across all pruning algorithms and datasets, as it consistently exhibited the best performance.

We evaluate the performance of all methods using two metrics: ΔPC assesses the impact on effectiveness, while *Relative Resolution Time (RRT)* assesses the impact on efficiency. In essence, it expresses the portion of the input blocks’ resolution time that is required by the meta-blocking method. Formally, it is defined as:

$$RRT = \frac{RT(B')}{RT(B)} \cdot 100\%$$
, where $RT(B)$ and $RT(B')$ are the resolution times of the original and the restructured block collections; the lower its value is, the more efficient is the meta-blocking method.

We applied supervised meta-blocking to *WEP*, *CEP* and *CNP*. The outcomes with respect to ΔPC are depicted in Figures 9(a)-(c), while *RRT* is presented in Figures 9(d)-(f).

WEP. We observe that iterative blocking consistently achieves the highest effectiveness, increasing *PC* by 1%, at the cost of the worst efficiency across all datasets. In fact, its *RRT* increases monotonically for higher dataset sizes, raising from 1/3 to more than 1/2. On the other extreme lies supervised meta-blocking: it reduces *PC* by at least 3%, but requires at most 1/6 of the original resolution time. It scales well to larger datasets, as its performance is relatively stable across all datasets: for each classification algorithm, the difference between their maximum and minimum ΔPC is less than 2%, while for *RRT* it is less than 5%. In the middle of these two extremes lies unsupervised meta-blocking, which reduces recall by less than 3.5%, while requiring half of the resolution time of iterative blocking. Note, though, that it does not scale well to larger datasets, as its *RRT* raises from 1/6 for D_{10K} to 1/3 for D_{2M} .

CEP. For unsupervised meta-blocking, ΔPC decreases almost linearly with the increase of the dataset size. In contrast, supervised meta-blocking scales well with respect to recall, as the variance of its ΔPC is lower than 7% (note that both classification algorithms exhibit practically identical performances). Equally stable is their efficiency, since their *RRT* is close to 1/8, on average, while its variance is less than 5%. However, the run-time of unsupervised meta-blocking scales better, as its *RRT* decreases from 1/8 for D_{10K} to 1/30 for D_{2M} .

CNP. The efficiency of unsupervised meta-blocking scales well with the increase of dataset size, dropping from 1/5 for D_{10K} to 1/30 for D_{2M} , while its effectiveness decreases. In this case, though, its recall is close to that of supervised meta-blocking, with their maximum difference amounting to 3%. In terms of efficiency, supervised meta-blocking exhibits an unstable behavior, with its *RRT* fluctuating between 1/5 and 1/10.

Conclusions. Overall, we conclude that supervised meta-blocking scales better than the unsupervised one for *WEP* with respect to both effectiveness and efficiency. For *CEP*, it scales better with respect to effectiveness, while unsupervised meta-blocking excels in efficiency in case a cheap entity matching method is employed. The same applies to *CNP*, as well. For this pruning algorithm, though, supervised meta-blocking improves effectiveness only to a minor extent. Compared to iterative blocking, supervised meta-blocking excels in efficiency, requiring a lower resolution time by at least

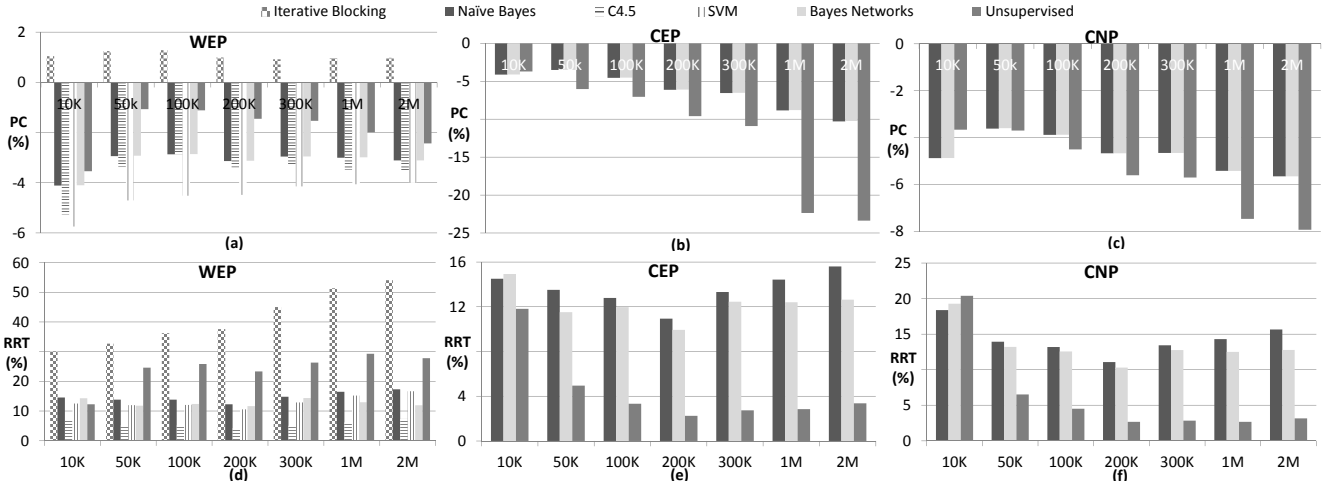


Figure 9: Scalability analysis over the synthetic datasets with respect to (a)-(d) WEP, (b)-(e) CEP and (c)-(f) CNP.

2/3, but achieves significantly lower recall. Compared to the brute-force approach, supervised meta-blocking improves the run-time by at least 5 times, as its *RRT* lies consistently lower than 20%.

10. CONCLUSIONS

In this work, we demonstrated how supervised meta-blocking can be used to enhance the performance of existing, unsupervised meta-blocking methods. For this task, we proposed a small set of generic features that combine a low extraction cost with high discriminatory power. We showed that supervised meta-blocking can achieve high performance with small training sets that can be manually created, and we verified that most configurations of established classification algorithms have little impact on the overall performance. We analytically compared our supervised approaches with baseline and competitor methods.

In the future, we will apply transfer learning techniques to supervised meta-blocking, so that a classification model trained over a labeled set maintains its high performance over another, unlabeled one. In addition, we will explore the use of active learning and crowdsourcing techniques in the creation of training sets.

Acknowledgements. This research has been co-financed by the EU (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

- [1] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.
- [2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.
- [3] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *SIGMOD*, pages 207–218, 2009.
- [4] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012.
- [5] P. Christen and A. Pudjijono. Accurate synthetic generation of realistic personal information. In *PAKDD*, pages 507–514, 2009.
- [6] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*, pages 475–480, 2002.
- [7] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 1565–1568, 2009.

- [8] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. Tailor: A record linkage tool box. In *ICDE*, pages 17–28, 2002.
- [9] L. Getoor and A. Machanavajjhala. Entity resolution for big data. In *KDD*, 2013.
- [10] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.
- [11] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.
- [12] M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.
- [13] B. Kenig and A. Gal. Mfiblocks: An effective blocking algorithm for entity resolution. *Inf. Syst.*, 38(6):908–926, 2013.
- [14] H. Kim and D. Lee. HARRA: fast iterative hashed record linkage for large-scale data collections. In *EDBT*, pages 525–536, 2010.
- [15] R. Longadge, S. Dongre, and L. Malik. Class imbalance problem in data mining: Review. *Int’l Journal of Computer Science and Network (IJCSN)*, 2(1), 2013.
- [16] Y. Ma and T. Tran. Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *WSDM*, pages 325–334, 2013.
- [17] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.
- [18] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.
- [19] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.
- [20] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: Large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.
- [21] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.
- [22] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolution to the next level. *IEEE Trans. Knowl. Data Eng.*, 26(8):1946–1960, 2014.
- [23] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.
- [24] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.
- [25] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.
- [26] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.