OCCT: A One-Class Clustering Tree for Implementing One-to-Many Data Linkage

Ma'ayan Gafny, Asaf Shabtai, Lior Rokach, Yuval Elovici

Abstract— One-to-many data linkage is an essential task in many domains, yet only a handful of prior publications have addressed this issue. Furthermore, while traditionally data linkage is performed among entities of the same type, it is extremely necessary to develop linkage techniques that link between matching entities of different types as well. In this paper we propose a new one-to-many data linkage method that links between entities of different natures. The proposed method is based on a one-class clustering tree (OCCT) which characterizes the entities that should be linked together. The tree is built such that it is easy to understand and transform into association rules, i.e., the inner nodes consist only of features describing the first set of entities, while the leaves of the tree represent features of their matching entities from the second dataset. We propose four splitting criteria and two different pruning methods which can be used for inducing the OCCT. The method was evaluated using datasets from three different domains. The results affirm the effectiveness of the proposed method and show that the OCCT yields better performance in terms of precision and recall (in most cases it is statistically significant) when compared to a C4.5 decision tree-based linkage method.

Index Terms— Clustering, Classification, Data matching, Decision tree induction

1 INTRODUCTION

DATA linkage is the task of identifying different entries (i.e., data items) that refer to the same entity across different data sources [1]. The goal of the data linkage task is joining datasets that do not share a common identifier (i.e., a foreign key). Common data linkage scenarios include: linking data when combining two different databases [2]; data deduplication (a data compression technique for eliminating redundant data) which is commonly done as a preprocessing step for data mining tasks [3], [4]; identifying individuals across different census datasets [5]; linking similar DNA sequences [6]; and, matching astronomical objects from different catalogues [7]. It is common to divide data linkage into two types: *one-to-one* and *one-to-many* [8]. In one-to-one data linkage, the goal is to associate an entity from one dataset with a single matching entity in another dataset. In one-to-many data linkage, the goal is to associate an entity from the first dataset with a group of matching entities from the other dataset. Most of the previous works focus on one-to-one data linkage.

In this paper we propose a new data linkage method aimed at performing one-to-many (and can be extended to many-to-many) linkage. In addition, while data linkage is usually performed among entities of the same type, the proposed data linkage technique can match entities of different types. For example, in a student database we might want to link a student record with the courses she should take (according to different features which describe the student and features describing the courses). The proposed method links between the entities using a *One-Class Clustering Tree* (*OCCT*). A clustering tree is a tree in which each of the leaves contains a cluster instead of a single classification. Each cluster is generalized by a set of rules (e.g., a set of conditional probabilities) that is stored in the appropriate leaf [9], [10].

The OCCT was evaluated using datasets from three different domains; *data leakage prevention*, *recommender systems*, and *fraud detection*. In the data leakage prevention domain, the goal is to detect abnormal access to database records that might indicate a potential data leakage or data misuse. The goal is to match an action, performed by a user within a specific context, with records that can be legitimately retrieved within that context. In the recommender systems domain the proposed method is used for matching *new* users of the system with the items that they are expected to like based on their demographic attributes. In the fraud detection domain, the goal is to identify online purchase transactions that are executed by a fraudulent user and not the legitimate user (i.e., identity theft). The results show that the OCCT performs well in different linkage scenarios. In addition, it performs at least as accurate as the well known C4.5 decision tree data-linkage model, while incorporating the advantages of a one class solution. Additionally, the OCCT is preferable over the C4.5 decision tree because it can easily be translated to linkage rules.

The contribution of this work is twofold. First and foremost, we propose a method that allows performing one-to-many (and many-to-many) linkage between objects of the same or of *different* types. This is opposed to existing methods that are only able to link between objects of the same type. Secondly, we use a one-class approach. This is an important advantage because in certain domains obtaining meaningful non-matching

The authors are with the Department of Information Systems Engineering at Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel (e-mail: gafnym@bgu.ac.il; shabtaia@bgu.ac.il; rokach@bgu.ac.il; elovici@bgu.ac.il).

examples can be difficult. For example, in the fraud detection case we can easily obtain genuine matching examples; these are actually legitimate transactions performed by users. Non-matching examples (fraudulent transactions) are rare and more difficult to obtain. In such cases non-matching examples can be artificially created and added to the training set; however, we can receive examples that do not make sense. For example, a fraudulent customer purchases a product that is not being sold in the customer's country.

The rest of the paper is organized as follows. In Section 2 we review related works on data linkage and decision trees. Section 3 describes the data linkage problem, while Sections 4-7 present a description of the proposed solution. In Section 8 we describe the evaluation and the results. Section 9 presents a discussion of the results, and Section 10 concludes the paper.

2 Related Work

2.1 Data Linkage

Data linkage refers to the task of matching entities from two different data sources that do not share a common identifier (i.e., a foreign key). Data linkage is usually performed among entities of the *same type*. It is common to divide data linkage into two types, namely, one-to-one and one-to-many. In *one-to-one* data linkage, the goal is to associate one record in table T_A with a single matching record in table T_B. In the case of *one-to-many* data linkage, the goal is to associate one record in T_A with one or more matching records in T_B.

One-to-one data linkage was implemented using various algorithms including: an SVM classifier that is trained to distinguish between matching and non matching record pairs [11]; calculating Expectation Maximization [11] or Maximum Likelihood Estimation [5] in order to determine the probability of a record pair being a match; employing hierarchal clustering in order to link between pairs of entities [3], [12]; and performing behavior analysis in order to find matching entities [2]. These methods assume that the same entities appear in the two datasets to be linked, and try to match between records that refer to the same entity. Therefore, these works are less relevant for data linkage between entities of different types.

Only a few previous works have addressed one-to-many data linkage. Storkey *et al.* [7] use the Expectation Maximization algorithm for two purposes: (1) calculating the probability of a given record pair being a match, and (2) learning the characteristics of the matched records. A Gaussian mixture model is used to model the conditional magnitude distribution. No evaluation was conducted on this work.

Ivie *et al.* [6] use one-to-many data linkage for genealogical research. In their work, they performed data linkage using five attributes: an individual's name, gender, date of birth, location and the relationships between the individuals. A decision tree was induced using these five attributes. However, the main drawback of this method is that it is tailored to perform matches using specific attributes and therefore, very hard to generalize.

Christen and Goiser [13] use a C4.5 decision tree in order to determine which records should be matched to one another. In their work, they compare different decision trees which are built based on different string comparison methods. However, in their method, the attributes according to which the matching is performed are predefined and only one or two attributes are usually used.

In this paper we propose a new data linkage method aimed at performing one-to-many linkage that can match entities of *different types*. Following [13], we use the J48 Weka implementation [14] of the C4.5 decision tree [15] as a baseline for comparison with our method. The inner nodes of the tree consist of attributes referring to both of the tables being matched (T_A and T_B). The leaves of the tree will determine whether a pair of records described by the path in the tree ending with the current leaf is a *match* or a *non-match*.

Data linkage is closely related to the *Entity Resolution* problem. While in data linkage the goal is to link between related entries in one or more data sources, the goal of entity resolution is to identify non-identical records that represent the same real-world entity, and to merge them into a single representative record (also known as deduplication) [16]. Joint entity resolution attempts to improve entity resolution by using additional information that can be derived by joining the table with a related table [17].

Another related research domain is co-clustering [18]. Co-clustering refers to a two-dimensional clustering process in which the entities (i.e., instances) and the attributes are clustered at the same time. The OCCT model also results in clusters of instances, each may be described with a different set of attributes. The clusters are later modeled in a compact way. In this sense the proposed OCCT method can also be used for co-clustering; however, in this paper we focus on the linkage task.

2.2 Decision Trees

Traditionally, decision trees are used for classification and regression tasks. The training set used for inducing the tree must be labeled. However, acquiring a labeled dataset is a costly task. Therefore, we believe that using a decision model which requires examples of one class only is highly preferable.

De Comit'e *et al.* [4] introduce POSC4.5, an adaptation of the C4.5 algorithm for learning from positive examples and unlabeled examples. In addition to the given datasets, it requires knowledge of the ratio of positive examples out of the whole dataset. The novelty of this approach over the C4.5 algorithm is that it proposes a modified entropy formula which considers the weight of the positive class in the given dataset and assumes the number of negative examples in the unlabeled data according to the given distribution. In addition, only binary classification problems can be considered.

Letouzey *et al.* [19] extend the above algorithm to create a forest of trees by iterating over different possible ratios of the positive class. Then, the model which is the most accurate is chosen for use. Experimental results show very good performance when the number of positive and negative examples in the dataset is similar. However, it performs very poorly on imbalanced datasets. Therefore, the solution proposed by Letouzey *et al.* is not suitable for our problem.

Li *et al.* [20] also propose an extension to the POSC4.5 algorithm, which iterates over different possible positive ratios and chooses the tree that achieves the most accurate results. The uniqueness of their method is that it can classify streams of data very quickly. This algorithm is not suitable for our needs as well, because it cannot handle cases in which the ratio of positive examples is small.

Clustering trees are structured differently than traditional decision trees [9]. In clustering trees, each node represents a cluster (or a concept). The tree as a whole describes a hierarchy (e.g., a taxonomy). Blockeel *et al.* [10] extend this idea and describe an approach in which each of the leaves contains a cluster instead of a single classification. Each leaf of the tree is characterized by a logical expression (e.g., conjunction of literals) representing the instances belonging to it. According to [21] the main advantage of using clustering trees is that they provide a description for each of the clusters using a logical expression.

The OCCT is a decision model that is similar to a clustering tree. Additionally, it learns and represents only positive examples, and therefore it is a one-class model. In our proposed method, each leaf represents a cluster, while the characteristics of the cluster are represented by a set of rules. Our method differs from clustering trees mainly in its ability to link two different types. In particular, we create the clusters by examining the attributes representing the first table (T_A), while the data that is clustered is from the table that is linked to it (T_B). The rules that are formed refer to table T_B's attributes as well. Our method differs from existing one-class decision trees due to the fact that it represents only positive examples (examples that do not fit the description represented in the tree are classified as negative).

3 The One-to-Many Data Linkage Problem

A typical data linkage problem consists of two data tables that do not share a unique identifier. We will denote these tables as T_A and T_B. In addition, we denote A as the set of attributes of T_A, and B as the set of attributes of T_B. The goal is to match between the records of T_A with their corresponding records in T_B. Usually, it is assumed that records in T_A and T_B refer to the same type of entities. We define $|T_A|$ as the number of records in T_A and $|T_B|$ as the number of records in T_B. Since potentially, each record in T_A can be linked to all records in T_B all record pairs $(r_{(a)}, r_{(b)}) \in T_A \times T_B$ (where $r_{(a)} \in T_A$ and $r_{(b)} \in T_B$) must be considered. Therefore, the problem space is $|T_A| \times |T_B|$. However, advanced indexing techniques can be used in order to make the linkage process efficient and scalable [22]. The pairs examined are split into two groups T_{AB} and $T_{\overline{AB}}$ where $T_{AB} \subseteq T_A \times T_B$ denoting the set of matching records, and $T_{\overline{AB}} \subseteq T_A \times T_B$ denoting the set of non-matching records. A pair of records can be classified either as matching or as non-matching; therefore, $T_{AB} \cup T_{\overline{AB}} = T_A \times T_B$ and $T_{AB} \cap T_{\overline{AB}} = \emptyset$ [23]. The purpose of a data linkage algorithm is to correctly identify as many true matching pairs as possible (true positive), while minimizing the classification error (false positive). Notions used throughout the paper are summarized in Fig. 1.

T_A –	a given table A
T_B –	a given table B (our goal is to link records from table T_A with one or more records from T_B)
$\|T_A\|$ –	number of records in T_A
$\|T_B\|$ –	number of records in T_B
A –	the set of attributes of table T_A where a_i is the i-th attribute
$\|A\|$ –	denotes the number of attributes in T_A
B –	the set of attributes of table T_B where b_i is the i-th attribute
$\|B\|$ –	denotes the number of attributes in T_B
$r_{(a)} \in T_A$ –	a record from table T_A
$r_{(b)} \in T_B$ –	a record from table T_B
$T_A \times T_B$ –	a table that is generated by applying Cartesian product of T_A and T_B
$r=(r_{(a)},r_{(b)}) \subseteq T_A \times T_B$ –	a record of $T_A \times T_B$
$T_{AB} \subseteq T_A \times T_B$ –	denoting the set of matching records
$T_{\overline{AB}} \subseteq T_A \times T_B$ –	denoting the set of *non*-matching records
d –	a node in the OCCT model
$A_d \subseteq A$ –	the subset of attributes of T_A that were already selected as splitting attributes in the path from the root of the tree to node d.
$T_{AB}^{(d)} \subseteq T_{AB}$ –	the subset of matching instances at node d of the OCCT tree
$Split_a\left(T_{AB}^{(d)}\right) = T_{AB}^{(d)(a)}$ –	the splitting of $T_{AB}^{(d)}$ into n subsets according to attribute a such that $\forall i = 1..n\ T_{AB}^{(di)(a)} = \{r \in T_{AB}^{(d)} \| a = v_i\}$
$\sigma_p(T_{AB}^{(d)})$ –	selection operator that is used to select records in $T_{AB}^{(d)}$ that satisfy the given predicate p (in this case p is $a=v_i$)
$\pi_A(T_{AB}^{(d)})$ –	projection operator that is used to select a subset of attributes in $T_{AB}^{(d)}$ that appear in the attribute collection A

Fig. 1. Notations used in the paper.

Fig. 2 presents a general example of the data linkage task. In the example, two tables, from two different data sources, are presented. Note that in the example the entities of the two tables are *not* of the same type. The goal is to link records in T_A (i.e., users) to their matching records in T_B (i.e., movies). Each possible record pair is assigned a score that describes the probability of the records being a match. The probability of two records being a match is assumed to be derived by a pre-induced model. Such a model is induced during a training phase by applying an inducing algorithm (e.g., decision trees, SVM, or ANN) on a training set containing *labeled* examples of matching (and optionally non-matching) record pairs. Our basic requirements from the induced models are: (1) the induced model should provide a probability for each class (match/non-match); (2) the model should handle both numeric and nominal attributes; and (3) the model should handle missing values. During the evaluation (testing) phase, the induced model is applied on unlabeled record pairs. In the example presented in Fig. 2, given the values of a user's attributes (from T_A) and the values of movies' attributes (from T_B), the model derives the level of linkage (i.e., linkage score). The level of linkage is provided as a number between 0 and 1. In order to reach a final binary decision (i.e., match or non-match) a threshold has to be defined. If the score exceeds the predefined threshold, the records are considered a match and linked together; otherwise, the records are classified as non-match. For example, given that the decision threshold is 0.5, only movies 1, 4, and 5 are considered to be matches for user 1.

To solve this task we propose a one-class clustering tree that matches records originating from table T_A with records from table T_B. The entities of T_A may be of a different type of entities of T_B. The inner nodes of the tree represent attributes of T_A. The leaves of the tree provide a compact representation of records from table T_B that should be linked to records from table T_A, represented by the path from the root to the leaf.

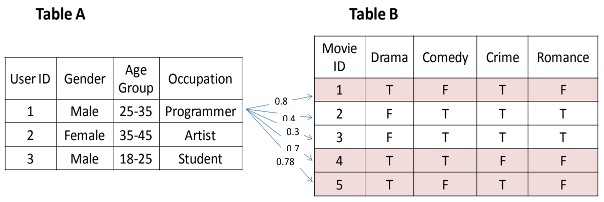

Table A

User ID	Gender	Age Group	Occupation
1	Male	25-35	Programmer
2	Female	35-45	Artist
3	Male	18-25	Student

Table B

Movie ID	Drama	Comedy	Crime	Romance
1	T	F	T	F
2	F	T	T	T
3	F	T	T	T
4	T	T	F	F
5	T	F	T	F

Fig. 2. An example of a one-to-many data linkage task.

4 The Proposed Method – An Overview

The proposed solution is divided into the following steps: (1) inducing a clustering tree linkage model; (2) building probabilistic models to represent the leaves; and (3) linking items according to the induced model.

1. Inducing a linkage model

The linkage model encapsulates the knowledge of which records are expected to match each other. The induction process includes *deriving the structure of the tree*. Building the tree requires deciding which attribute should be selected at each level of the tree. The inner nodes of the tree consist of attributes from table T_A only. Selecting the attribute is done by using one of the possible splitting criteria presented in Section 5.1. The splitting criteria ranks the attributes based on how good they are in clustering the matching examples.

In addition, a pre-pruning process is implemented. This means that the algorithm stops expanding a branch whenever the sub-branch does not improve the accuracy of the model (the proposed pruning methods are described in Section 5.2). The inducer is trained with matching examples only.

2. Representing the leaves using probabilistic models

Once the construction of the tree is completed, each leaf contains a cluster (or set) of records. A set of probabilistic models is induced for each of the leaves. Each model M_i is used for deriving the probability of a value of attribute $b_i \in B$ from table T_B, given the values of all other attributes from table T_B. There are two motivations for performing this step. First, the sets of probabilistic models result in a more compact representation of the OCCT model. Second, by representing the matching records as a set of probabilistic models, the model is better generalized and avoids overfitting.

3. Linkage

During the linkage (i.e., testing) phase, each pair of records in the testing set is cross-validated against the linkage model. The output is a score representing the probability of the record pair being a *true* match. The score is calculated using maximum likelihood estimation (MLE) [24]. The tested pair is classified as a match if the score is greater than a given, predefined, threshold or, if not, as a non-match. The threshold is defined by taking into consideration the tradeoff between the false positive rate and the true positive rate.

In the next sections we describe in detail each of the steps for generating the OCCT. In Section 5 we describe the process of inducing the linkage model. In Section 6, we describe how we represent the leaves, and in Section 7 we describe the linkage process[1].

5 Inducing a Linkage Model

The OCCT is induced using one of the proposed splitting criteria. The splitting criterion is used to determine which attribute should be used in each step of building the tree. In addition, we use a pruning process in order to decide which branches should be trimmed.

Fig. 3 describes the pseudo-code of the induction process of the OCCT model. It consists of three procedures: *buildTree*, which is the main function; *chooseBestSplit*, in which the splitting attribute is chosen; and *createModelsForLeaves*, in which a set of probabilistic models are created for the given leaf.

The input of the algorithm is a training set of matching instances T_{AB} (each instance r is a pair of records $(r_{(a)},r_{(b)})$: one from table T_A and one from table T_B; i.e., $r=(r_{(a)},r_{(b)}) \in T_{AB} \subseteq T_A \times T_B$), and two lists of attributes: A describing the attributes of table T_A, and B describing the attributes originating from table T_B.

The *buildTree* process for building the tree is an iterative process. Let $T_{AB}^{(d)} \subseteq T_{AB}$ be the subset of matching instances at node d of the tree, and let $A_d \subseteq A$ be the set of attributes of T_A that were already selected as splitting attributes in the path from the root of the tree to node d. Thus, $A \backslash A_d$ denotes the attributes of T_A that *were not* selected yet as splitting attributes. The process terminates either when the subset of matching instances T_{AB} is smaller than the given threshold t (i.e., $|T_{AB}|<t$), or when there are no more candidate attributes for split in A; i.e., $(A \backslash A_d)=\emptyset$ (line 2). Otherwise, in each iteration we find the next best splitting attribute by evaluating every attribute $a \in (A \backslash A_d)$ according to the selected splitting criterion (line 5). Let a be an attribute with n possible values $v_1,v_2,..,v_n$. We define $Split_a\left(T_{AB}^{(d)}\right) = T_{AB}^{(d)(a)} = \left\{T_{AB}^{(d1)(a)}, T_{AB}^{(d2)(a)}, \dots, T_{AB}^{(dn)(a)}\right\}$ as the splitting of $T_{AB}^{(d)}$ into n subsets according to attribute a such that $\forall i = 1..n\ T_{AB}^{(di)(a)} = \{r \in T_{AB}^{(d)} | a = v_i\}$. For simplicity reasons, in the following Section 5.1 we assume that a can have only two possible values v_1 and v_2

[1]An illustrative example is provided in: http://tlabs.bgu.ac.il/index.php?option=com_content&view=article &id=125&Itemid=108

(i.e., binary attribute). Therefore, splitting $T_{AB}^{(d)}$ according to attribute a results in two sets $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ such that $T_{AB}^{(d1)(a)}$ includes all records of $T_{AB}^{(d)}$ for which a=v_1 and $T_{AB}^{(d2)(a)}$ includes all records of $T_{AB}^{(d)}$ for which a=v_2. At the end of Section 5.1 we explain how to extend the process of building the tree to multi-valued attributes. Note that in order to handle continuous attributes, a discritization process needs to be applied on the continuous attribute prior to inducing the model.

```
buildTree(T_AB,A,B,t)
Input:    $T_{AB}$ - set of matching instances,
          $A$ - set of attributes from table $T_A$,
          $B$ - set of attributes from table $T_B$,
          $t$ - threshold; the minimum size for split
Output:   $T$ (OCCT tree)

1:   Node $T$ ← newNode()
2:   if $|T_{AB}| \le t$ OR $(A)=\emptyset$ then
3:       T.Models ← createModelsForLeaf($T_{AB}$,$B$)
4:   else
5:       $a$ ← chooseBestSplit($T_{AB}$,$A$)
6:       if not pruneTree($T_{AB}$,$a$) then
7:           set T.attribute ← $a$
8:           for each $v_i \in a$
9:               T.Child[$i$] ← buildTree($\sigma_{a=v_i}(T_{AB})$,$(A\backslash\{a\})$,$B$,$t$)
10:          end for
11:      else
12:          T.Models ← createModelsForLeaf($\pi_B(T_{AB})$,$B$)
13:      end if
14:  end if
15:  return $T$
```

```
chooseBestSplit(T_AB,A)
Input:    $T_{AB}$ - set of matching instances,
          $A$ - set of attributes from table $T_A$,
Output:   $a'$ - the attribute chosen for the split
1:   $sp_{opt}$ ← 0
2:   $a'$ ← $\emptyset$
3:   for each $a$ in $A$
4:       $sp$ ← evaluateSplit($T_{AB}$,$a$)
5:       if $sp$ is better than $sp_{opt}$ then
6:           $sp_{opt}$ ← $sp$
7:           $a'$ ← $a$
8:       end if
9:   end for
10:  return $a'$
```

```
createModelsForLeaf(T_B,B)
Input:    $T_B$ - set of matching instances from table $T_B$,
          $B$ - set of attributes from table $T_B$,
Output:   M - set of models for given dataset
1:   $M$ ← $\emptyset$
2:   for each $b \in B$
3:       Set $b$ as class attribute of $T_B$
4:       $m$ ← Build probabilistic model for $T_B$
5:       $M$ ← $M \cup m$
6:   end for
7:   return $M$
```

Fig. 3. The pseudo code of the tree induction process.

Once the best splitting attribute a is determined, each subset $T_{AB}^{(di)(a)}$ of $T_{AB}^{(d)}$ will be sent recursively to the procedure *buildTree* (line 8-10). The selection operator σ of the form $\sigma_p(T_{AB}^{(d)})$ is used to select records in $T_{AB}^{(d)}$ that satisfy the given predicate p (in this case p is a=v_i). The projection operator π of the form $\pi_A(T_{AB}^{(d)})$ is used to select a subset of attributes in $T_{AB}^{(d)}$ that appear in the attribute collection A.

5.1 The splitting criteria

The goal is to achieve a tree which contains a small amount of nodes. Smaller trees better generalize the data, avoid over fitting, and will be simpler for the human eye to understand [26]. Therefore, it is crucial to use an effective splitting criterion in order to build the tree. We would choose to perform $Split_a(T_{AB}^{(d)})$ (i.e., splitting $T_{AB}^{(d)}$ according to attribute a) if we were to gain the most information out of this split. In this section we propose four criteria that can be used for evaluating the splitting of $T_{AB}^{(d)}$ according to an attribute a. Each splitting criterion is used for measuring the similarity between two record sets T_1 and T_2, and is denoted by $sim(T_1,T_2)$. In the context of our research, the similarity function that is defined by the selected splitting criteria is used in order to determine the attribute that creates the best split of a table; i.e., splits table T into two tables, T_1 and T_2, which differ from each other as much as possible.

In the *ChooseBestSplit* procedure described in Fig. 3, each attribute in A is examined in order to determine the quality of the split it will achieve. The attribute that achieves the best score (highest/lowest- depending on the splitting criterion) will be returned and used as the next split of the tree. In this section, we present four splitting criteria which we believe will be efficient for inducing a small decision tree: coarse-grained Jaccard coefficient, fine-grained Jaccard coefficient, least probable intersection, and maximum likelihood estimation.

Coarse-grained Jaccard (CGJ) coefficient

The *Jaccard similarity coefficient*, a measure that is commonly used in clustering, measures the similarity between clusters [26]. In the context of our research we use this coefficient in order to choose the splitting attribute a, and define a subset $T_{AB}^{(di)(a)}$ of a record set $T_{AB}^{(d)}$ as a cluster. The goal is to choose the splitting attribute which leads to the *smallest* possible similarity between the subsets (i.e., an attribute that generates subsets that are different from each other as much as possible). In order to do so, we examine each of the possible (remaining) splitting attributes and measure the similarity between the subsets. The similarity between two subsets, denoted by $sim\left(T_{AB}^{(d1)(a)}, T_{AB}^{(d2)(a)}\right)$, is computed for each possible splitting attribute, $a \in (A\backslash A_d)$, using the Jaccard coefficient as the ratio between the number of records belonging to *both* $\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d1)(a)})$ and $\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d2)(a)})$, and the total number of records, as described in Equation (1). Records are considered in the intersection only if they are *completely* identical (all attributes share the same values).

$$sim\left(T_{AB}^{(d1)(a)}, T_{AB}^{(d2)(a)}\right) = \frac{\left|\left(\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d1)(a)})\right) \cap \left(\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d2)(a)})\right)\right|}{\left|\left(\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d1)(a)})\right) \cup \left(\pi_{(A\backslash A_d)\cup B}(T_{AB}^{(d2)(a)})\right)\right|} \quad (1)$$

The goal is to choose the split that creates two subsets, $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$, who are as different from each other as much as possible. Therefore, we will favor the attribute that *minimizes* the similarity measure above.

In order to minimize the computational complexity of building the model using the CGJ criterion, the values of the fields from T_B can be expressed as a single (concatenated) string. Then, a string matching algorithm can be used to find the intersection between the two subsets of records. For example, Knuth *et al.* [27] describe an algorithm for string matching whose complexity is $O\left(\left|T_{AB}^{(d)}\right| + |B|\right)$ for finding a match for a single string. Thus, if we were to search for a match for each of the strings in $T_{AB}^{(d1)(a)}$, the complexity would be bounded by $O\left(\left|T_{AB}^{(d)}\right|^2 + |B| \cdot \left|T_{AB}^{(d)}\right|\right)$. Since $T_{AB}^{(d)} \subseteq T_{AB}$, The complexity is bounded by $O(|T_{AB}|^2 + |B| \cdot |T_{AB}|)$.

There are $|A|$ possible splitting attributes examined in each level, and a total of $|A|$ levels. Therefore the total time complexity of building a model using the CGJ criterion is $O(|A|^2 \cdot (|T_{AB}|^2 + |B| \cdot |T_{AB}|))$.

Fine-grained Jaccard (FGJ) coefficient

The *fine-grained Jaccard coefficient* [26] is capable of identifying partial record matches, as opposed to the coarse-grained method, which identifies exact matches only. It not only considers records which are exactly identical, but also checks to what extent each possible pair of records is similar. Assume that r_i and r_j are two records originated from $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ respectively. According to the fine-grained Jaccard coefficient, their similarity would be calculated as the *number of attributes* containing the same values in both r_i and r_j, divided by the total number of attributes examined that *did not* contain null in either of the records. This is calculated for each possible pair of records in $T_{AB}^{(d1)(a)} \times T_{AB}^{(d2)(a)}$. The similarity of $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ is therefore calculated as the sum of similarities of all possible pairs of records, as described in Equation (2).

$$sim\left(T_{AB}^{(d1)(a)}, T_{AB}^{(d2)(a)}\right) = \sum_{r_i \in T_{AB}^{(d1)(a)}; r_j \in T_{AB}^{(d2)(a)}} \frac{|r_i \cap r_j|}{|r_i \cup r_j|} \quad (2)$$

Due to the fact that comparing each record of one leaf with all records in another node is a very expensive process in terms of run-time, we tried to optimize the calculations. In order to do so, we first take all instances of $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ and use the *k*-means algorithm [28] in order to cluster these instances into k different clusters (with k being a settable parameter that is provided as an input to the clustering algorithm). Finally, we use FGJ to evaluate the similarity of each record originally belonging to node $T_{AB}^{(d1)(a)}$ with each of the other records belonging to the *same cluster* and are originally from $T_{AB}^{(d2)(a)}$. Since the similarity between records is performed only between records that belong to the same cluster, the total processing time is decreased by a factor of k. Moreover, it lets us to easily perform the calculation in parallel and by that additionally reduce the time span by a factor of k. This, of course, introduces a tradeoff: as k (the number of clusters) is increased, the time span is reduced by a factor of k^2 in total, however, this is on the account of accuracy of the induced OCCT model. We recommend setting k as the number of available cores in the machine, such that each core will be assigned to process a different cluster.

Given that $T_{AB}^{(d)}$ is the original collection of instances (i.e., the training set), and that we are evaluating a binary split into $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$. In order to calculate an intersection (or a partial intersection), $|B|$ comparisons must be made. Then, calculating the number of intersecting instances between $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ would take $O\left(|B| \cdot |T_{AB}^{(d1)(a)}| \cdot |T_{AB}^{(d2)(a)}|\right)$. Since $T_{AB}^{(d1)(a)} \subseteq T_{AB}$, The complexity is bounded by $O(|B| \cdot |T_{AB}|^2)$.

There are $|A|$ possible attributes that are candidates for splitting, and therefore, the total complexity of identifying the first splitting attribute is $O(|A| \cdot |B| \cdot |T_{AB}|^2)$. If the tree is not pruned, there would be $|A|$ levels in the tree, therefore the process of selecting a splitting attribute is performed $|A|$ times. Thus, the overall complexity of building the model using the FGJ criterion is bounded by $O(|A|^2 \cdot |B| \cdot |T_{AB}|^2)$.

Least probable intersections (LPI)

Gershman *et al.* [29] propose a heuristic in which the optimal splitting attribute is the attribute that leads to the minimum amount of instances that are shared between two item-sets. They propose a criterion which relies on the *cumulative distribution function* (CDF) of the Poisson distribution. Assuming a random binary split of $T_{AB}^{(d)}$ into two subsets $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$, the probability P_i that a record $r_i \in T_{AB}^{(d)}$ belongs to both $\pi_{(A\setminus A_d)\cup B}(T_{AB}^{(d1)(a)})$ and $\pi_{(A\setminus A_d)\cup B}(T_{AB}^{(d2)(a)})$ is defined by Equation (3), with o_i denoting the number of appearances of item r_i in $\pi_{(A\setminus A_d)\cup B}(T_{AB}^{(d)})$.

$$P_i = 1 - \left(\frac{|T_{AB}^{(d1)(a)}|}{|T_{AB}^{(d)}|}\right)^{o_i} - \left(\frac{|T_{AB}^{(d2)(a)}|}{|T_{AB}^{(d)}|}\right)^{o_i} \quad (3)$$

In the context of our research, we refer to a distinct combination of attributes as a unique identifier of an entity. Therefore, our goal is to find a splitting attribute for which there is the least amount of identifiers that are shared, in comparison to a random split of the same size. Using the *central limit theorem* we assume that the data we are using will distribute normally. Let $j = \left|\left(\pi_{(A\setminus A_d)\cup B}(T_{AB}^{(d1)(a)})\right) \cap \left(\pi_{(A\setminus A_d)\cup B}(T_{AB}^{(d2)(a)})\right)\right|$ and $\lambda = \sum_{r_i \in T_{AB}^{(d)}} P_i$. In order to bring this distribution to the standard normal form (where $\mu=0$ and $\sigma^2 = 1$), we use the following approximation (4) to calculate Z (the test statistic), where $\mu = \lambda$, and $\sigma = \sqrt{\lambda}$. Z represents the probability of the two subsets to be created randomly.

$$Z_{(a)} = \frac{j - \lambda}{\sqrt{\lambda}} \quad (4)$$

The goal is to find the splitting attribute which is the least probable to generate the two subsets randomly. Therefore, the candidate splitting attribute with the highest score is chosen as the next attribute for split. Specifically, $Z_{(a)}$ is calculated for each candidate attribute for split $a \in (A\setminus A_d)$. The next splitting attribute of the tree will be the attribute that had achieved the highest Z score.

In terms of computational complexity, building a tree using this method is fairly cheap. In order to calculate λ, it is necessary to go over each of the distinct records in $T_{AB}^{(d)}$ and check the number of times they appear in the records set. In addition, it is necessary to calculate j (the intersection between $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$) for each possible splitting attribute. When using Knuth *et al.* algorithm for string matching [27], the cost of calculating the intersection is $O\left(|T_{AB}^{(d)}|^2 + |B| \cdot |T_{AB}^{(d)}|\right)$. Since $T_{AB}^{(d)} \subseteq T_{AB}$, the complexity is bounded by $O(|T_{AB}|^2 + |B| \cdot |T_{AB}|)$. In order to choose the best splitting attribute, the intersection is calculated $|A|$ times in each level of the tree, and in $|A|$ levels in total. Thus, the time complexity of building a tree according to the LPI criterion is $O(|A|^2 \cdot (|T_{AB}|^2 + |B| \cdot |T_{AB}|))$.

Maximum likelihood estimation (MLE)

This splitting criterion uses the *Maximum Likelihood Estimation* (MLE) [24] in order to choose the attribute that is most appropriate to serve as the next splitting attribute. Each candidate attribute from the set of attributes ($A \setminus A_d$) splits the node dataset into subsets according to its possible values. For each of the subsets, a set of probabilistic models $M_1 ... M_{|B|}$ is created, one for each attribute of table T_B. Each probabilistic model M_i is built to describe the probability of b_i given $b_1, b_2, \ldots b_{i-1}, b_{i+1}, \ldots, b_n$, $p(b_i | b_j, j=1..|B|, j \neq i)$. In order to create the probabilistic models we used Weka's J48 decision trees [14]. Each of these trees represents the probability of its class attribute values (i.e., b_i) given the values of all other attributes.

Once the set of models has been induced, the probability of each record given these models is calculated. Let a record in table T_B be represented by $r_{(b)}=(v_1, v_2, .., v_{|B|})$ where v_i is the value assigned to attributes $b_i \in B$. Then, for each record $r_{(b)}$ in the subset, we compute:

$$L(r_{(b)}) = \sum_{i=1}^{|B|} log(p(b_i = v_i | M_i;\ b_j = v_j, j = 1..n, j \neq i) \quad (5)$$

where, $p(b_i | M_i;\ b_j = v_j, j = 1..n, j \neq i)$ is the conditional distribution provided by M_i.

A subset's score is calculated as the sum of all $L(r_{(b)})$ scores of the records belonging to it (i.e., $\forall r_{(b)} \in \pi_B(T_{AB}^{(di)(a)})$). The attribute's final score is determined by the sum of the subset's individual scores.

Our goal is to choose the split that achieves the maximal likelihood, and therefore we choose the attribute with the highest likelihood score as the next splitting attribute in the tree.

The computational complexity of building a decision model using the MLE method is dependent on the complexity of building a statistical model and the time it takes to calculate the likelihood. Let us denote $M(|T_{AB}|, |B|)$ as the complexity of building the probabilistic model. This complexity varies according to the method chosen for representing the model (e.g., decision tree, naïve Bayes), to the size of the input dataset, and to the number of attributes from which the dataset is composed of. In each level, $|B|$ models are built; one for each attribute from table T_B. Thus, the time complexity of building a set of models for a single node is $O(|B| \cdot M(|T_{AB}|, |B|))$. For example, in our implementation we are using the J48 decision tree [17] as the probabilistic model. The time complexity of inducing a J48 decision tree is $O(|A| \cdot |T_{AB}| \cdot \log(|T_{AB}|) + |T_{AB}| \cdot \log^2 |T_{AB}|)$. Thus, the time complexity of building a set of models for a given records set is $O\big(|B| \cdot (|A| \cdot |T_{AB}| \cdot \log(|T_{AB}|) + |T_{AB}| \cdot \log^2 |T_{AB}|)\big)$.

The complexity of calculating the likelihood of the record set is affected by the model used and the attributes from which it was built. Thus, we denote it as *L(M,B)*. The likelihood estimation is conducted for each record in the dataset and over the set of $|B|$ models. Therefore, the time complexity for calculating the likelihood is $O(L(M,B) \cdot |T_{AB}| \cdot |B|)$. Overall, the process of building the model set is performed $|A|$ times for a single split (once for each possible split). This occurs once for each level of the tree. Thus, the total time complexity for building a model using MLE is $O(|A|^2 \cdot (|B| \cdot M(|T_{AB}|, |B|) + L(M,B) \cdot |T_{AB}| \cdot |B|))$.

Dealing with multi-valued splits

For simplicity reasons the three measures which are described above (CGJ, FGJ, and LPI) were presented for binary attributes. However, in most datasets, discrete attributes may have more than two possible values. When multi-values attributes exist in the dataset, the proposed splitting criteria are adapted as follows.

The score of a candidate splitting attribute is calculated as a weighted average of a series of possible binary splits. Each binary split, splits $T_{AB}^{(d)}$ into two sets $T_{AB}^{(d1)(a)}$ and $T_{AB}^{(d2)(a)}$ such that $T_{AB}^{(d1)(a)}$ includes all records of $T_{AB}^{(d)}$ for which $a=v_1$ and $T_{AB}^{(d2)(a)}$ includes all records of $T_{AB}^{(d)}$ for which $a \neq v_1$. A weight w_i is calculated for each binary split i such that $w_i = |T_{AB}^{(d1)(ai)}| / |T_{AB}^{(d)}|$. The weights are proportional to the size of the subsets to ensure that the resulted splitting value of an attribute will be influenced mainly by records having more dominant values and not by esoteric ones.

Note that only splitting methods that calculate the similarity between record sets require the special treatment we are proposing for multi-values splits (i.e., CGJ, FGJ, and LPI). This is because these methods are capable of measuring the similarity of only two record sets at a time, and therefore an adjustment is necessary for multi-valued splits. The fourth measure (MLE) does not measure the similarity between two given record sets, and it is computed individually for each possible subset. Then, the scores are accumulated regardless of the number of subsets. Thus, no additional action is needed in order to extend the measure from binary splitting attributes to multi-valued splitting attributes.

5.2 Pruning

Pruning is an important task in the tree induction process. A good pruning process will produce a tree which is accurate on one hand, and avoids overfitting on the other. There are two common approaches for pruning a decision tree: pre-pruning and post-pruning [9]. In *pre-pruning*, a branch is pruned during the induction process if none of the possible splits are found to be more beneficial than the current node. In *post-pruning*, the tree is grown completely, followed by a bottom-up process to determine which branches are not beneficial.

In our algorithm we follow the pre-pruning approach. This approach was chosen in order to reduce the time complexity of the algorithm. The decision whether to prune the branch or not is taken once the next attribute for split is chosen, as described in Fig. 3 (line 6 in the *buildTree* process). We propose using one of the following methods: maximum likelihood estimation (MLE) and least probable intersections (LPI).

In the *maximum likelihood* method, an MLE score is computed for each of the possible splits (as described in Section 5.1). If none of the candidate attributes achieve an MLE score which is higher than the current node's MLE score, the branch is pruned and the current node becomes a leaf.

In the least probable intersections method, Z is calculated for each possible split (as described in Section 5.1). If for all possible splitting attributes Z is smaller than a predefined threshold, then all of the possible splits are likely to be formed by a random split. Thus, we will not gain much information from any of the possible splits and the branch will be pruned.

6 Leaf Representation

Once the model is built (based on attributes from table T_A), each leaf holds a dataset containing the matching records from table T_B. In order to create a compact representation of the linkage model and for it to be more generalized each leaf is represented by a set of probabilistic models. These models represent the probability of an attribute, given the values if all other attributes in B. Formally, let there be $|B|$ attributes representing records from table T_B: $B=\{b_1,b_2,\ldots,b_{|B|}\}$. For each attribute b_i in B, a probabilistic model M_i is built to describe the probability of b_i given $b_1,b_2,\ldots b_{i-1},b_{i+1},\ldots,b_n$, $p(b_i|b_j, j=1..|B|, j\neq i)$. The pseudo code of building a set of models for a given leaf is described in the function *createModelsForLeaves* in Fig. 3.

We found that it is not necessary to save models for all possible attribute of table T_B. Instead, models are created only for the attributes that are found to have the most significant effect in the specific leaf. Therefore, prior to building the models, a feature selection process is executed on the leaf dataset in order to choose the attributes that will be represented. The goal of the feature selection process is identifying the attributes that best represent the records that appear in a leaf. Therefore, a different set of attributes might be chosen for representing each of the leaves.

In our implementation, each leaf is represented by a set of Weka's J48 decision trees [14]. We avoided the usage of full conditional probability tables by using a J48 classification tree with its generalization capabilities for encoding the conditional probabilities in a compact way. In this sense J48 is used as a probability estimation tree [30] by avoid pruning and using Laplace correction. In particular, for each attribute b_i we induce a dedicated classification tree M_i, where b_i is used as the target attribute and all other attributes are used as input attributes. A classification tree is capable to provide a conditional probability for each possible combination of the $b_1,\ldots,b_n$ because there is always a path from the root to one of the leaves that fits a given combination (assuming that for domain values of all attributes are known); the path is usually represented by a subset of the attributes of B that were selected by the J48 algorithm, which makes the tree much more compact than the full conditional probability table. Note that each leaf provides a different conditional probability and the number of leaves in the classification tree is bounded by the training set size.

For the feature selection process, we use Weka's implementation of the Correlation-based Feature Subset Selection algorithm [31] feature selection algorithm. This feature selection algorithm searches for a subset of features that have high correlation with the class attribute and which have low correlation with one another. In our case, we apply the feature selection process on a leaf dataset and we choose the class attribute to be the currently selected splitting attribute $a_i \in A$. Therefore, at the end of the feature selection process we are left with a set of features that are highly correlated with a_i, but are uncorrelated with one another.

Each path from the root of the OCCT tree to a leaf is represented by specific values of attributes of table T_A. Therefore, the records of each leaf are aggregated according to the attributes of table T_A. In this sense, in each leaf, the attributes of T_A are fixed and thus the probability of each attribute of T_B is conditioned, not only on the values of the other attributes T_B but also on the attributes of T_A.

7 Applying OCCT for Data Linkage

During the linkage (i.e., test) phase, each possible pair of test records is tested against the linkage model in order to determine if the pair is a match. This process produces a score representing the probability of the record pair being a true match. An initial score is calculated using maximum likelihood estimation [25].

Fig. 4 presents the pseudo-code of the linkage process. The input to the algorithm is an instance from T_A, and an instance of T_B. The output of the algorithm is a Boolean value determining whether the given instances should be matched or not. First, the appropriate set of models is retrieved by following the values of record a to the correct path of the tree (line 1). The likelihood for a match between the records is calculated by deriving the probability of each value in b, given all other values and the appropriate model (line 2-6).

```
linkInstances(r(a),r(b),th)
Input:    $T_{AB}$ - set of matching instances,
          $r_{(a)}$ - an instance from table $T_A$,
          $r_{(b)}$ - an instance from table $T_B$,
          th - the threshold for match
Output:   q - a boolean value determining if the records match or not
1: $M \leftarrow$ the set of models matching the values of a
2: $l \leftarrow 0$
3: for each $b_i$ in b
4:        $l_i \leftarrow \log p(b_i = v_i | M_i;\ b_j = v_j, j = 1..n, j \neq i))$
5:        $l \leftarrow l + l_i$
6: end for
7: $l \leftarrow l * cardinality(a,b)$
8: if $l >= th$ then
9:        $q \leftarrow$ true
10: else
11:       $q \leftarrow$ false
12:end if
13: return q
```

Fig. 4. The pseudo code of the linkage process.

The probability of a match is also determined by the cardinality of the examined instances (i.e., the number of times that the record from table T_B is linked to table T_A). For example, assume we are examining two possible pairs who achieved the same MLE score. However, the record $r_{i(b)}$ entity from the first pair had only one matching record from table T_A in the training set, while record $r_{j(b)}$ from the second pair was a match for hundreds of records in the training set. It is reasonable to assume that it is much more likely that the second pair would be a match rather than the first pair. Therefore, we multiply the MLE score that was calculated by the cardinality of the examined records (line 7).

Finally, it is determined whether the given records are a match or not by comparing the likelihood score that was calculated to the given threshold *th* (line 7-11). If the pair's score is greater than *th*, it is classified as a match (lines 8-9); otherwise, it is classified as a non-match (lines 10-11). The threshold is defined by taking into consideration the tradeoff between the false positive rate and the true positive rate.

8 Evaluation

We set several goals for the evaluation process. Our first goal was to examine the different settings which were suggested; i.e., the four splitting criteria and the pruning process, and to identify the most suitable settings for the environment which was tested. Second, we wanted to compare between OCCT and a binary class decision tree, commonly used for one-to-many data linkage. For this purpose we use Weka's J48 decision tree. Third, we wanted to verify that the proposed method is generic and can be used for data linkage under different scenarios, and when executed on different domains.

To answer the first two research questions, we measured the true positive rate (TPR) - the ratio between the number of pairs correctly classified as a match and the total number of matching pairs, and false positive rate (FPR) - the ratio between the number of pairs incorrectly classified as a match, and the total number of pairs which were actually non-matches.

In order to evaluate the tradeoff between the TPR and the FPR, we used the receiver operating characteristic (ROC) graph [32]. This graph plots the TPR vs. the FPR as the threshold changes. The quality of the classification rates is measured using the area under the curve (AUC). The goal is to reach the largest area possible (1.0), implicating that 100% of the records were classified correctly. In general, a model which achieves a larger area under the curve is considered to be a better model. The curves of different settings of the OCCT model and the J48 model were compared using the ROCKIT platform [33] which statistically analyzes the AUC results using the univariate z-score test (bivariate binormal model). In our case, the null hypothesis (H0) is that the two datasets are from binormal ROC curves with equal area (i.e., equal AUC). The alternative hypothesis (H1) is that one ROC curve is with a significantly higher AUC than the other one. For each significant level (we set the significant level to be α=0.05), the z-test has a single critical value which can be compared with the test-statistic to determine whether to accept or reject the null hypothesis. Therefore, when using α=0.05 the null hypothesis (two ROC with equal area under the curve) will be rejected whenever the test-statistics is smaller than the critical value of 1.65 (which matches α=0.05). Similarly, if the p-value, which is the probability that matches the test-statistic, is smaller than α=0.05, the null hypothesis is rejected (i.e., the difference between the ROCs is statistically significant).

In addition, we also calculated the recall and precision measurements which are often used for evaluating data linkage methods. *Recall* is the number correctly identified matching records (i.e., TP) divided by the total number of matching records in the test set (TP+FN). Precision is the number of correctly identified matching records (i.e., TP) divided by the number of pair of records that were identified as matching (TP+FP). The recall and precision measurements are computed individually for each entity in table T_A and then averaged over all entities.

In order to answer the third research question, we performed our evaluation by applying the method on three different tasks, originating from three different domains: the data leakage detection task, originating from the information security domain; the task of recommending items to new users, originating from the recommender systems domain; and the task of identifying masquerade attacks, originating in the fraud detection domain. The domains vary in the goal of the linkage, in the cardinality between the tables, and in the size of the datasets. In each experiment the dataset is split into two mutually exclsive subsets: training set and test set. The training set is used for inducing the OCCT model and the test set is used for evaluating the induced model. Using an independent test set helps to objectively measure the strength of the predictive model on one hand, and its level of generalization on the other (making sure that the model does not suffer from overfitting to the training data).

8.1 The Database Misuse Domain

8.1.1 Scope and purpose

Most of the research efforts in the database misuse domain focus on deriving user profiles that define normal access patterns to the data stored in the database and issue an alert whenever a user's behavior deviates from the normal profile. The most common approach is by extracting various features from the SQL query string submitted by the application server to the database (as a result of a user's requests) [34]. Another approach for representing a user's behavioral profile, known as the *data centric* approach, focuses on analyzing the data exposed to the user following her request, i.e., the result-sets [35]. Mathew *et al.* [35] claim that whenever analyzing a user's request for data, features that define the context in which the request has been issued (such as the time of request and location of the user) should be considered. However, these features have not been used in related works which focused only on analyzing access to database records.

It is expected that the records retrieved following a user's request contain data which is legitimate for the user to view in his line of duty. We define a request which retrieves only legitimate records as a benign request. However, some users, intentionally or unintentionally, execute requests that retrieve records which should not be exposed to them. Nevertheless, if the result-set contains a significant amount of illegitimate records, the query is considered malicious.

The goal of using the OCCT in this domain is to link a set of records, representing the context of the request (i.e., the actual access to certain data), with a set of records representing the data which can be legitimately retrieved within the specific context. Thus, the inner nodes of the OCCT represent contextual attributes in which the request occurs (table T_A), and the probabilistic models in the leaves represent the data which can be legitimately retrieved in the specific context (table T_B). Pairs which are detected as non-matching are assumed to be illegitimate (i.e., malicious), and will trigger an alert to the organization's security officer. By analyzing both the context of the request as well as the data that the user is exposed to, the OCCT method can improve the detection accuracy and better distinguish between a normal and abnormal request.

8.1.2 Evaluation environment

Since no real dataset was available for evaluation, we opted to generate a simulated dataset which was used in [36]. In the simulated scenario, customer data of an organization is shared with a business partner. Therefore, the simulated data includes requests for customer records of an organization, submitted by a business partner of the organization.

Contextual information (i.e., table T_A attributes) on the request includes the time of execution, day of execution, geographical location of the request, the user’s role, and the type of request. Sensitive information of customers (i.e., table T_B attributes) includes the customer’s name, address, zip code, place of work, and the customer type (e.g., business, private).
The simulated requests were generated according to one of the following three behavior types:

Normal: An employee searching for customer records *within the same geographical location*, *during store opening hours*.

Malicious1: An employee searching, *during opening hours*, for a customer record that is *not in the same geographical location* as the store.

Malicious2: An employee searching for *any customer record after closing hours*.

In addition, we defined two types of users; a *benign* user who submits legitimate requests most of the time; and a *malicious* user who queries the database for a purpose other than his work (e.g., data harvesting). We believe that a malicious user might try to hide his malicious intentions by mixing malicious queries with legitimate ones. The settings for this evaluation were chosen according to what we presumed to be typical behavior. Therefore, most users were benign (95%) and only a small amount of users (5%) were malicious. 70% of the queries performed by a malicious user were considered normal (type 'Benign') and only 30% were malicious (type 'Malicious1' and 'Malicious2'). A benign user was configured to perform normal actions (type 'Benign') 98% of the time with the rest being malicious.

Overall, the dataset consisted of almost 150,000 transactions and their matching result sets, ranging over a period of two weeks. The transactions of the first week (about 75,000 transactions) were used for training, while the transactions of the second week were used for testing the induced model.

8.1.3 Results

In Table 1 we compared the results achieved when using different settings. Four splitting criteria (CGJ, FGJ, LPI, and MLE) and three pruning options (no pruning, LPI, and MLE) resulted in 12 different settings. Table 1 summarizes the AUC achieved in each of the 12 possible configurations.

TABLE 1
AUCs FOR THE 12 POSSIBLE SETTINGS (DATABASE MISUSE)

Splitting / Pruning	CGJ	FGJ	LPI	MLE
No pruning	0.9207	0.9201	0.9200	0.9201
LPI	0.9197	0.9268	**0.9268**	0.9194
MLE	0.9208	0.9230	0.9230	0.9206

We used the univariate z-test in order to compare the AUCs of the 12 settings. When no pruning was applied, we found no significant difference between the four splitting criteria (see Table 2). When using the LPI pruning method the FGJ and the LPI splitting criteria are both significantly better than the CGJ and the MLE methods. When applying MLE as the pruning method, we found the FGJ splitting criterion to be significantly better than the CGJ and the MLE splitting criteria.

TABLE 2
COMPARING THE SPLITTING CRITERIA (DATABASE MISUSE)

		CGJ	LPI	MLE
No pruning	FGJ	p-value= 0.461 (◊) statistic= 0.0975	p-value= 0.396 (◊) statistic= 0.2624	p-value= 0.440 (◊) statistic= 0.1494
	CGJ		p-value= 0.33 (◊) statistic= 0.4400	p-value= 0.445 (◊) statistic= 0.1381
	LPI			p-value= 0.312 (◊) statistic= 0.4899
LPI	FGJ	p-value= 0.001(◄) statistic= 3.0221	p-value= 0.468 (◊) statistic= 0.0787	p-value= 0.000(◄) statistic= 3.1777
	CGJ		p-value= 0.001(◄) statistic= 3.0355	p-value= 0.027(◄) statistic= 1.9142
	LPI			p-value= 0.000(◄) statistic= 3.1904
MLE	FGJ	p-value= 0.000(◄) statistic= 9.3044	p-value= 0.463 (◊) statistic= 0.0906	p-value= 0.00 (◄) statistic= 7.6250
	CGJ		p-value= 0.041(▲) statistic= 1.7308	p-value= 0.118 (◊) statistic= 1.1805
	LPI			p-value= 0.402 (◊) statistic= 0.2473

The '◄' symbol indicates that the AUC of the Row's splitting criterion is significantly higher than the Column's splitting criterion. The '▲' symbol indicates that the AUC of the Row's criterion is significantly lower, and the '◊' symbol indicates no significant difference.

Table 3 compares the three pruning methods. When applying the CGJ and MLE splitting criteria, we found that MLE pruning was better than LPI pruning. For the FGJ and the LPI splitting criteria, we found that LPI pruning was significantly better than no pruning at all.

Overall, we found that applying LPI pruning with either the FGJ or the LPI splitting criteria yields the best results in the inspected domain.

Fig. 5 depicts the ROC curves of OCCT and J48 algorithms. By comparing the curves using the ROCKIT tool [33] which statistically analyses the AUC results using the univariate z-score test (bivariate binormal model), we found that OCCT is significantly better than J48 (with p-value: 0.00, test statistic: 18.20), where J48's AUC is 0.7637.

TABLE 3
COMPARING THE PRUNING METHODS (DATABASE MISUSE)

		LPI	MLE
CGJ	No pruning	-------	p-value= 0.4137 (◊) statistic= 0.2179
	LPI		p-value= 0.0132 (▲) statistic= 2.2197
FGJ	No pruning	p-value= 0.0047 (▲) statistic= 2.6007	p-value= 0.0752 (◊) statistic= 1.4383
	LPI		p-value= 0.1096 (◊) statistic= 1.2286
LPI	No pruning	p-value= 0.0047 (▲) statistic=2.6007	p-value= 0.0752 (◊) statistic= 1.4383
	LPI		p-value= 0.1096 (◊) statistic= 1.2286
MLE	No pruning	-------	p-value= 0.4137 (◊) statistic= 0.2179
	LPI		p-value= 0.0132 (▲) statistic= 2.2197

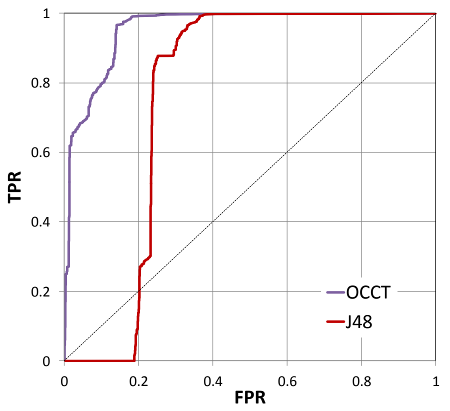

Fig. 5. ROC graphs of the OCCT (when using LPI as the splitting criteria and LPI for pruning) and the J48 algorithms.

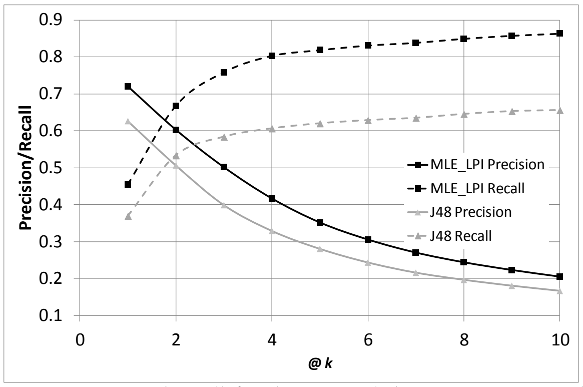

Fig. 6. Precision and recall for the OCCT (when using MLE as the splitting criteria and LPI for pruning) and for the J48 algorithms.

Finally, in Fig. 6 we present the precision and recall for the OCCT (when using MLE as the splitting criteria and LPI for pruning, which yield the best precision/recall performance) and J48 algorithms. Using the Pearson correlation test we found that both the precision and recall of the OCCT are significantly better than the J48 (with p-value: 5.9463E-06, test statistic: -4.5918 for the precision, and p-value: 2.3419E-05, test statistic: -4.5918 for the recall).

8.2 The Movie Recommender Domain

8.2.1 Scope and purpose

There is a rapidly growing need for a personalized recommender system that presents users with items or contents which are likely to interest them [37]. Burke [38] describes the recommendation problem as a classification problem in which a classifier determines whether or not the user is likely to like a certain item.

Decision trees have been extensively used in recommendation systems. Golbandi *et al.* [39] suggest using a decision tree in order to adjust the initial set of questions that are presented to a new user in the system. Kim *et al.* [40] propose using decision trees for automatically extracting demographic marketing rules. Gershman *et al.* [29] proposed a tree-based recommendation system which produces lists of recommended items at the leaves of the tree. Li and Yamada [41] propose modeling user preferences using a C4.5 decision tree. Lee [42] proposes using decision trees, in order to find links between items. Bouza *et al.* [43] induce a decision tree model that can explain user ratings using semantic information (features) available for the items.

The task of recommending items to new users with whom the system is not familiar with is extremely difficult. Some existing researches propose profiling users not as individuals, but rather according to their demographic characteristics. This is done by clustering users (or finding similarities between users) according to their demographic characteristics [44].

In order to solve the problem of recommending items to new users, we propose using our matching algorithm and encapsulating both a demographic and content-based recommendation approach. This is done by matching demographic features that represent users (attributes of table T_A), with the content of the items they typically like (attributes of table T_B). The OCCT model would cluster the users according to their demographic attributes and describe the likelihood of each group of users to like different features describing the items (i.e. content features).

8.2.2 Evaluation environment

Movielens (www.movielens.org) is an online movie recommender system whose main goal is to collect ratings of different users regarding different movies for research purposes. Users log on to the web site and rate different movies, and in return, the system generates a personalized list of recommended movies according to predicted user preferences. The dataset we used consisted of 873,899 rating of 2,871 movies generated by 6,040 users. Three demographical features were available for describing users: gender (male or female), age group (split into six age intervals), and occupation description (21 different possible values).

Different features describing the movies were collected from IMDB, an internet movie database that features information regarding movies and television series (www.imdb.com). We extracted seven types of features describing a movie: the movie's directors, its star actors, the countries in which it was filmed, the languages spoken in the film, the genres to which it belongs to, the year it was released, and a set of keywords describing it. The collected data was pivoted to fit a tabular format. Then, a feature selection process was executed, resulting in a table containing 140 different features for each movie. The Movielens dataset provides a rating for each movie on a scale of 1 (do not like) to 5 (like very much). For the purposes of our research, a binary scale was necessary, namely, like and dislike. In order to convert the given ratings into a binary scale we calculated the average rating for each user. If a movie's score was higher than the average, it was assumed that the user liked the movie. Otherwise, the movie was considered disliked by the user. Evaluation was by splitting the dataset randomly into training and testing sets; 80% of the data was used for training and the remaining records were used for testing.

8.2.3 Results

Table 4 summarizes the AUC achieved in each of the 12 possible configurations. Overall, we found that applying the MLE pruning along with any of the four splitting criteria yields the best results. When comparing the AUCs of the 12 settings using the univariate *z*-test, we found that when pruning is not applied, or is applied using the MLE technique, all splitting methods perform equally as good (see Table 3 in Appendix B). However, when LPI pruning was applied, we found that the FGJ method is significantly worse than all other methods. In addition, pruning (either LPI or MLE) is always significantly better than the no pruning option, while in most cases MLE pruning is better than the LPI pruning (see Table 4 in Appendix B).

TABLE 4
AUCS FOR THE 12 POSSIBLE SETTINGS (MOVIE RECOMMENDER)

Splitting / Pruning	CGJ	FGJ	LPI	MLE
No pruning	0.6315	0.6326	0.6304	0.6315
LPI	0.6426	0.6376	0.6435	0.6429
MLE	0.6457	0.6449	**0.6493**	0.6476

In Fig. 7 we compared the results of OCCT with the results of the J48 algorithm. The graph shows minor advantage of the OCCT model over the J48 algorithm, especially for low FPRs, which are more relevant for most practical cases. Moreover, when statistically comparing the curves (using the univariate z-score test), the OCCT is significantly better than J48 (with p-value: 0.0001, test statistic: -3.0336), where J48's AUC is 0.6430.

Finally, in Fig. 8 we present the precision and recall for the OCCT (when using MLE as the splitting criteria and LPI for pruning, which yield the best precision/recall performance) and J48 algorithms. Using the Pearson correlation test we found that the precision of the OCCT is significantly better than the J48 (p-value: 0.0011, test statistic: -3.2634) and no difference in the recall (p-value: 0.6274, test statistic: 0.4855).

We compared the performance of the proposed method with nearest neighbors collaborative filtering (NNCF) using Jaccard coefficient. The results indicate that the proposed method has better performance. More specifically while NNCF obtains a precision of 0.4 at *k*=5, OCCT obtains a precision of 0.85 at *k*=5 (note that both methods have similar recall performance). We choose to implement NNCF with Jaccard coefficient because we also employ the same metric in our own algorithm. It should be noted that this comparison does not intend to be exhaustive. There are other CF methods like SVD, which may provide better results. However the aim of this paper is not to introduce a new CF method. In fact our method utilizes the users' demographic and items' properties. In this sense our method builds a hybrid of demographic and content-based recommender system and not a collaborative filtering-based system. Hybrid content-demographic recommender systems [45] are frequently solved by converting the problem into a supervised learning task. For example, classification tree models, which are hierarchical like the OCCT model, were used successfully in the past (e.g., [46]). Due to the last reason and due to limited space we present a detailed comparison to J48 and not to collaborative filtering.

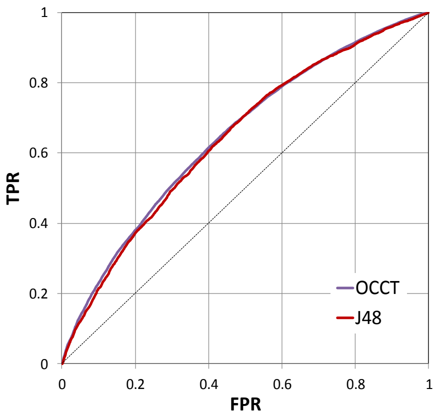

Fig. 7. ROC graphs of the OCCT (when using MLE as the splitting criteria and MLE for pruning) and the J48 algorithms.

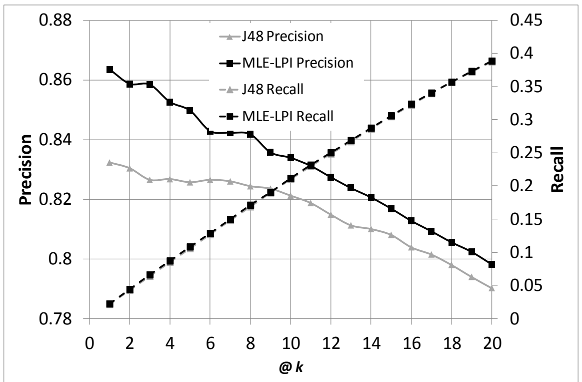

Fig. 8. Precision and recall for the OCCT (when using MLE as the splitting criteria and LPI for pruning) and for the J48 algorithms.

8.3 The Fraud Detection Domain

8.3.1 Scope and purpose

In this section we address the scenario of detecting online requests performed by entities that unauthorizingly logon to the system using the identity of another legitimate user. This scenario is often addressed in the literature as *masquerade detection* [47]. Most current works address this problem by profiling normal user behavior and alerting deviate behavior from the profiles learned [48], [49].

In order to solve the problem of detecting online masquerading attacks, we propose using our matching algorithm. The model will match between demographic attributes describing the user (table T_A), and the attributes describing the transaction (table T_B). The OCCT will represent the characteristics of transactions which are executed by legitimate users. If a transaction deviates from the user's normal behavioral profile, it might indicate that it was executed by a masquerader, and in such case, a notification will be sent to the system administrator.

8.3.2 Evaluation environment

For the purpose of evaluation, we used a real dataset of an online shopping website. The dataset consists of purchase transactions; data describing both the user who is purchasing the item, and the transaction itself.

The dataset consisted of a total of 921 customers and their purchases (a total of 14,192 transactions). The dataset was split randomly into training and testing sets; 80% of the data was used for training and the remaining records were used for testing. Since the data consisted only of transactions which were legitimately executed by the users, 1000 *non-matching* transactions were added to the test set. These transactions were generated by matching between a random user and a random transaction.

The goal of using the OCCT was to determine whether a given user-transaction pair is indeed legitimate or if user is actually a masquerader. Ten attributes described the user (e.g., the user's gender, residential city, residential country, age). The transaction was described by 21 different attributes such as the purchase amount, the means of payment (credit card/ debit card etc.), and the day of week and time of day in which the acquisition was made.

8.3.3 Results

Table 5 summarizes the AUC results obtained in each of the 12 settings. When comparing the AUCs of the 12 settings using the univariate *z*-test, we found that when pruning is not applied, CGJ is significantly better than the other splitting criteria. When LPI or MLE pruning is applied, the LPI criterion is significantly better than all other criteria (see Table 5 in Appendix B). In addition, we found that LPI pruning is preferable in all of the splitting criteria except for the CGJ in which the no pruning option yielded the best results (see Table 6 in Appendix B). Overall, we found that using the LPI splitting criterion with LPI pruning yielded the best results. When we compared the results of this setting with the results of the J48 algorithm (using the univariate z-score test), we found no significant difference between the results (with p-value: 0.1317, test statistic: 0.6487), where J48's AUC is 0.6298.

Finally, in Fig. 9 we present the precision and recall for the OCCT (when using MLE as the splitting criteria and PLI for pruning, which yield the best precision/recall performance) and J48 algorithms. Using the Pearson correlation test we found that both the precision and recall of the OCCT are significantly better than the J48 (with p-value: 1.4935E-30, test statistic: -12.0695 for the precision and p-value: 1.0722E-06, test statistic: 7.2574 for the recall).

TABLE 5
AUCS FOR THE 12 POSSIBLE SETTINGS (FRAUD DETECTION)

Splitting / Pruning	CGJ	FGJ	LPI	MLE
No pruning	0.6504	0.5668	0.5884	0.5868
LPI	0.6012	0.5948	**0.6627**	0.6437
MLE	0.5359	0.5682	0.6437	0.5474

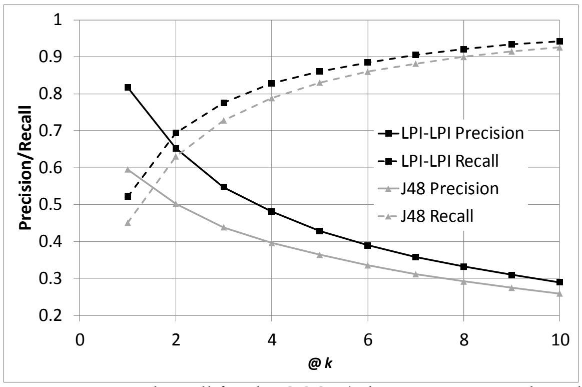

Fig. 9. Precision and recall for the OCCT (when using LPI as the splitting criteria and LPI for pruning) and for the J48 algorithms.

9 DISCUSSION

The OCCT was evaluated using three datasets from three different domains: the data leakage/misuse prevention domain, the recommender systems domain, and the fraud detection domain. The first goal of the evaluation process was to identify the most suitable settings for each of the domains. In Section 3, we proposed four different splitting criteria which can be used to induce the tree. The criteria differ from one another in different aspects and are each suitable for different types of domains. Additionally, three options for pruning were proposed.

In both the data misuse and the movie recommender domains, we found that when no pruning is applied, all four criteria yield similar results in terms of the linkage accuracy (measured by AUC). We explain this by the fact that without pruning, the clusters that are created in the leaves are actually identical, and thus the four models are actually identical in their meaning (the order of the inner nodes does not make a difference when no pruning is applied).

However, we found that when pruning is applied, some splitting criteria were better than others. When applying LPI pruning in the recommender systems domain, we found that FJG yielded significantly poorer results than other settings. In the database misuse domain, we found that FJG and LPI, both with LPI pruning, are equally effective and are significantly better than the other criteria.

When applying MLE pruning, we found that in the data misuse domain, the FGJ criterion is significantly better than most others, while in the movie recommendation domain, all four criteria are equally effective.

In the fraud detection domain we found that the LPI criterion is significantly better than all other criteria when either of the pruning methods is applied.

Overall, we had observed that in most cases, pruning avoids overfitting and enhances the results of the linkage process. Additionally, in the database misuse and the fraud detection domains, the settings which produced the best results involved LPI pruning. However, they were not significantly better than the results achieved by MLE pruning using the same criterion. In addition, MLE pruning is preferable to LPI pruning as it does not require fine-tuning of any parameters. Thus, MLE pruning should be preferred in most domains.

Table 6 summarizes the pros and cons that were identified for each of the criterions. Overall, there is no specific criterion that is always preferred over others. Instead, the four criteria should be tested on the domain in order to identify which one produces the best results. However, if two or more criteria are equally effective, we would recommend using the criterion with the lowest computational complexity.

TABLE 6
SUMMARY OF THE SPLITTING CRITERIA

Method	Advantages	Disadvantages
Coarse-grained Jaccard coefficient	- Low computational complexity	- Can only handle binary splits - Produces bias results when the attributes are not distributed uniformly
Fine-grained Jaccard coefficient	- Takes partial intersections into consideration - In some domains produces results which are significantly better than other criteria	- Can only handle binary splits - High computational complexity
Least probable intersections	- Can be used as a criterion for pruning - Can handle non-uniformly distributed attributes - Most durable to noise in the data	- Can only handle binary splits
Maximum Likelihood estimation	- Can handle multiple way splits - Can be used as a criterion for pruning	- Assumes that the attributes are independent with one another - High computational complexity (dependent on the type of probability model used)

Table 7 compares the execution time for inducing the OCCT model in the fraud detection scenario. The results indicate that the most time consuming settings is the FGJ splitting measure. The clustering-based optimization reduces the execution time of FGJ calculation by 10, but still it is significantly higher than the other three splitting measures. The CGJ and LPI splitting measures are the fastest approaches having a similar execution time as expected from the computational analysis in Section 5.1. The fastest execution time is accepted when using the LPI pruning method. We also analyzed the execution time of each step: evaluating a potential splitting attribute, feature selection, and creating models at leafs, for each possible setting. From the results presented in Table 8, it can be observed that the most time consuming step is evaluating a potential splitting attribute, especially when FGJ is used as the splitting criterion and when using MLE for pruning.

Another aspect of our evaluation process was the comparison between the results of OCCT and the results of the J48 decision tree. The results show that in the data misuse domain, OCCT works significantly better than the J48 algorithm. In the recommender systems domain, there is a minor but significant difference in the AUC scores, and the ROC curve of OCCT is better than the J48 especially for low FPR values.

TABLE 7
OVERALL EXECUTION TIME FOR THE FRAUD DETECTION SCENARIO (MILLISECOND)

Pruning \ Splitting	CGJ	FGJ	LPI	MLE
No pruning	137,908	682,303 7,537,885 (no clustering)	145,989	297,829
LPI	91,294	633,396	107,222	157,760
MLE	120,482	565,659	123,587	107,004

Executed on 64-bit Windows Server 2008 Enterprise ed., Intel Xeon CPU 1.6Ghz, 2Gb Memory (RAM).

TABLE 8
EXECUTION TIME FOR EACH STEP EXTRACTED FROM THE FRAUD DETECTION SCENARIO (MILLISECOND)

Step	Pruning \ Splitting	CGJ	FGJ	LPI	MLE
Evaluate splitting attribute	No Pruning	671	3,797	738	1,621
	LPI	1,745	12,978	1,332	4,637
	MLE	5,964	35,014	7,376	5,253
Feature selection	No Pruning	114	73	145	103
	LPI	402	330	298	886
	MLE	1,560	889	1,919	1,045
Create models at leafs	No Pruning	62	55	61	57
	LPI	83	78	79	78
	MLE	94	83	87	80

Executed on 64-bit Windows Server 2008 Enterprise ed., Intel Xeon CPU 1.6Ghz, 2Gb Memory (RAM).

In the fraud detection domain, no significant difference was found between the OCCT and J48. However, the AUC of OCCT was higher than the AUC achieved by J48. Overall, we conclude that the OCCT is as effective as the alternative binary-class linkage algorithm.

Finally, when comparing precision and recall (which are often used for evaluating data linkage methods) the OCCT yield better performance and in most cases it is statistically significant.

Evaluation results show that the OCCT model behaves differently on different datasets and for different evaluation measure (i.e., precision/recall or ROC). The experimental results indicate that the preferred configuration for the AUC criterion is LPI-LPI (i.e., using LPI for the splitting criteria and LPI for pruning). If the precision and recall measures are used the preferred configuration is MLE-LPI. Therefore, for a specific problem we recommend to focus and check only the four possible combinations defined by MLE and LPI.

The OCCT model also uses three types of thresholds: threshold t is used to decide if to further split the current node (based on the number of records left in the node before splitting); threshold th is used to decide whether a pair of records are matched or not matched; and the last threshold is used when the LPI method is used for pruning. The threshold th should be set by the operator of the method/system based on the required tradeoff of detection and error rates. In order to set the two other thresholds we recommend applying a cross-validation parameter selection method as proposed in [50]. In the experimental study we used this procedure to tune the splitting threshold and the LPI pruning threshold. We found that for the splitting threshold t=10 provided the best performance in all scenarios (i.e., a node will not be split if the number of records in the node is less than 10).

10 Conclusions and Future Work

We present OCCT, a one-class decision tree approach for performing one-to-many and many-to-many data linkage. The proposed method is based on a one class decision tree model that encapsulates the knowledge of which records should be linked to each other. In addition, we proposed four possible splitting criteria and two possible pruning methods that can be used for inducing the data models. Our evaluation results show that the proposed algorithm is effective when applied in different domains. Our goal is to link a record from a table T_A with records from another table T_B. The generated model is in the form of a tree in which the inner nodes represent attributes from T_A and the leafs hold a *compact representation* of a subset of records from T_B which are more likely to be linked with a record from T_A, whose values are according to the path from the root of the tree to the leaf.

Our decision to use only the attributes of T_A as the splitting attributes of the tree was taken in order to keep the generated model simple and easy to understand. Thus, the proposed OCCT tree can be easily used to obtain all records in T_B that match a given record $r_{(a)}$ in T_A. For this purpose we need only to traverse the OCCT tree using $r_{(a)}$. The leaf that is ultimately being reached holds all predicted matching records in T_B. On the other hand, a regular classification tree which mixes T_A and T_B attributes is less convenient for this task. In particular, in order to get all records in T_B that match a given $r_{(a)}$, we need to classify all possible links (i.e., $r_{(a)}$ with any record in T_B) one-by-one as either match or not-match using the classification tree. This results in a tiresome process.

The J48 decision tree that we tested as our baseline does consider attributes from both tables T_A and T_B; however, it is less understandable because it mixes attributes from T_A and T_B. Moreover, J48 requires both matching and non-matching examples in the training set. The proposed algorithm, on the other hand, needs only matching instances. Note that we are interested in reducing the computation of both the model induction and the actual linkage. However, it is more critical to reduce the linkage computation since inducing the OCCT model can be done offline while the linkage phase is more important for real world problems. Therefore, splitting the tree by using attributes from both T_A and T_B would increase the linkage time.

The contribution of this work is threefold. First and foremost, our method allows performance of one-to-many and many-to-many linkage between objects of the same or of different types. Secondly, we used a one-class approach, and thus the training set requires only examples of matching pairs. Since the algorithm assumes that all examples in the training set are positive, non-matching (negative) pairs would confuse the algorithm and lead to a less accurate model. However, preliminary experiments that we conducted showed that the generalization capability of the proposed model can overcome training sets that contain a relatively small number of non-matching (negative) pairs and still generate an accurate model. The evaluation of the proposed model on training sets that contain non-matching examples as well is left for future work. However, we believe that when enough non-matching examples are available, the J48 model is preferable and would probably work better. Third, an important advantage of the OCCT model over a decision tree-based data linkage solution is the simplicity of the model which can easily be transformed to rules of the type $A \rightarrow B$. This is not the case in other decision tree based linkage models where the inner nodes of the tree consist of attributes from both tables T_A and T_B, thus making them difficult to read and almost impossible to translate into rules.

Although, our focus is on the one-to-many case, the OCCT model can be used for many-to-many linkage, for example, by simply changing the roles of the two tables T_A and T_B and using table T_B as the source table instead. Note that in the Movielens dataset (movie recommender domain) we are actually solving a many-to-many problem in which groups of users are matched with common matching movies.

For future work we plan to compare the OCCT with other data linkage methods. In addition, we plan to extend the OCCT model to the many-to-many case and to handle continuous attributes. We also propose evaluating the results on additional domains, and characterizing which splitting criterion and pruning methods should be applied for each type of domain.

References

[1] I.P. Fellegi, and A.B. Sunter, "A Theory For Record Linkage," *Journal of American Statistical Society*, vol. 64, no. 328, pp. 1183-1210, Dec. 1969.

[2] M. Yakout, A.K. Elmagarmid, H. Elmeleegy, M. Quzzani, and A. Qi, "Behavior Based Record Linkage," in *Proc. of the VLDB Endowment*, vol. 3, no. 1-2, pp. 439-448, 2010.

[3] J. Domingo-Ferrer, and V. Torra, "Disclosure Risk Assessment in Statistical Microdata Protection via Advanced Record Linkage," *Statistics and Computing*, vol. 13, no. 4, pp. 343-354, 2003.

[4] F. De Comit'e, F. Denis, R. Gilleron, and F. Letouzey, "Positive and Unlabeled Examples Help Learning," *Algorithmic Learning Theory*, Springer, pp. 219-230, 1999.

[5] M. D. Larsen, and D.B. Rubin, "Iterative Automated Record Linkage Using Mixture Models," *Journal of the American Statistical Association*, vol. 96, no. 453. pp. 32-41, March 2001.

[6] S. Ivie, G. Henry, H. Gatrell, and C. Giraud-Carrier, "A Metric-Based Machine Learning Approach to Genealogical Record Linkage," in *Proc. of the 7th Annual Workshop on Technology for Family History and Genealogical Research*, 2007.

[7] A. J. Storkey, C. K. I. Williams, E. Taylor, and R. G. Mann, "An Expectation Maximisation Algorithm for One-to-Many Record Linkage," University of Edinburgh Informatics Research Report, 2005.

[8] P. Christen, and K. Goiser, "Quality and Complexity Measures for Data Linkage and Deduplication," *Quality Measures in Data Mining*, Springer, pp. 127-151, 2007.

[9] P. Langley, *Elements of Machine Learning*. San Francisco: Morgan Kaufmann, 1996.

[10] H. Blockeel, L. D. Raedt, J. Ramon, "Top-Down Induction of Clustering Trees," *ArXiv Computer Science e-prints*. pp. 55-63, 1998.

[11] D. J. Rohde, M. R. Gallagher, M. J. Drinkwater, and K. A. Pimbblet, "Matching of Catalogues by Probabilistic Pattern

Classification," *Monthly Notices of the Royal Astronomical Society*, vol. 369, no. 1, pp. 2-14, May 2006.

[12] L. Gu, and R. Baxter, "Decision Models for Record Linkage," *Data Mining*, vol. 3755, Springer, pp. 146-160, 2006.

[13] P. Christen, and K. Goiser. "Towards Automated Data Linkage and Deduplication," Australian National University, Technical Report, 2005.

[14] E. Frank, M.A. Hall, G. Holmes, R. Kirkby, and B. Pfahringer, "WEKA - A Machine Learning Workbench for Data Mining," *The Data Mining and Knowledge Discovery Handbook*, pp. 1305-1314, 2005.

[15] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[16] O. Benjelloun, H. Garcia, D. Menestrina, Q. Su, S. Whang, and J. Widom, "Swoosh: a generic approach to entity resolution," *The VLDB Journal*, vol. 18, no. 1, pp. 255–276, 2009.

[17] S.E. Whang, and H. Gercia-Molina, "Joint Entity Resolution," Stanford University, Technical Report, 2009.

[18] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," In *Proc. the 9th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pp. 89-98, Washington, USA, 2003.

[19] F. Letouzey, F. Denis, and R. Gilleron, "Learning From Positive and Unlabeled Examples," *Algorithmic Learning Theory*, Springer, pp. 71-85, 2009.

[20] C. Li, Y. Zhang, and X. Li, "OcVFDT: One-Class Very Fast Decision Tree for One-Class Classification of Data Streams," in *Proc. the 3rd Int. Workshop on Knowledge Discovery from Sensor Data*, pp. 79-86, Paris, France, 2009.

[21] J. Struyf, S. Dzeroski, "Clustering Trees with Instance Level Constraints," in *Proc. the 18th European Conf. on Machine Learning*, pp. 359-370, Warsaw, Poland, 2007.

[22] P. Christen, "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," *IEEE trans. on knowledge and data engineering*, DOI:10.1109/TKDE.2011.127, 2011.

[23] V. Torra, and J. Domingo-Ferrer, "Record Linkage Methods for Multidatabase Data Mining," *Studies in Fuzziness and Soft Computing*, Springer, pp. 101-132, 2003.

[24] D. D. Dorfman, and E. Alf, "Maximum-Likelihood Estimation of Parameters of Signal-Detection Theory and Determination of Confidence Intervals — Rating-Method Data," *Journal of Math Psycho*logy, vol. 6, no. 3, pp. 487–496, 1969.

[25] J. R. Quinlan, "Induction of Decision Trees," *Machine learning*, vol. 1, no. 1, pp. 81-106, March 1986.

[26] S. Guha, R. Rastogi, and K. Shim, "Rock: A Robust Clustering Algorithm for Categorical Attributes," *Information Systems*, vol. 25, no. 5, pp. 345-366, July 2000.

[27] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323-350, 1977.

[28] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proc. the 5th Symposium on Mathematical Statistics and Probability*, pp. 281-297, 1967.

[29] A. Gershman *et al*., "A Decision Tree Based Recommender System," in *Proc. the 10th Int. Conf. on Innovative Internet Community Services*, pp. 170-179, Trondheim, Norway, 2010.

[30] F. Provost, and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, no. 3, pp 199-215, 2003.

[31] M. A. Hall, "Correlation-based Feature Subset Selection for Machine Learning," University of Waikato, New Zealand, Technical Report, 1998.

[32] C. Ferri, P. Flach, and J. Hernández-Orallo, "Learning Decision Trees Using the Area Under the ROC Curve," in *Proc. the 9th Int. Conf. on Machine Learning*, pp. 139-146, 2002.

[33] C. A. Metz, "ROCKIT software," http://metz-roc.uchicago.edu/MetzROC, 2003.

[34] A. Kamra, E. Terzi, and E. Bertino, "Detecting Anomalous Access Patterns in Relational Databases," *Journal on Very Large Databases*, vol. 17, no. 5, pp. 1063-1177, 2008.

[35] S. Mathew, M. Petropoulos, H. Ngo, S. and Upadhyaya, "A Data-Centric Approach to Insider Attack Detection in Database Systems," *Recent Advances in Intrusion Detection*, Springer, vol. 6307, pp. 382-401, 2009.

[36] M. Gafny, A. Shabtai, L. Rokach, and Y. Elovici, "Detecting Data Misuse By Applying Context-Based Data Linkage," in *Proc. ACM CCS Workshop on Insider Threats*, Chicago, USA, 2010.

[37] G. Adomavicius, and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-Of-The-Art and Possible Extensions," *IEEE trans. on knowledge and data engineering*, vol. 17, no. 6, pp. 739-749, 2005.

[38] R. Burke, "Knowledge-Based Recommender Systems," *Encyclopedia of Library and Information Systems*, vol. 69, no. 32, pp. 175-186, 2000.

[39] N. Golbandi, Y. Koren, and R. Lempel, "Adaptive Bootstrapping of Recommender Systems Using Decision Trees," in *Proc. the 4th ACM Int. Conf. on Web search and data mining*, pp.595-604, Honk Kong, 2011.

[40] J. W. Kim, *et al*.,"Application of Decision-Tree Induction Techniques to Personalized Advertisements on Internet Storefronts," *Int. Journal of Electronic Commerce*, vol. 5, no. 3, pp. 45-62, 2001.

[41] P. Li, and S. Yamada, "A Movie Recommender System Based On Inductive Learning," in *Proc. IEEE Conf. on Cybernetics and Intelligent Systems*, pp. 318-323, Singapore, 2004.

[42] S. L. Lee, "Commodity Recommendations Of Retail Business Based On Decision Tree Induction," *Expert Systems with Applications*, vol. 37, no. 5, pp. 3685-3694, May 2010.

[43] A. Bouza, G. Reif, A. Bernstein, and H. Gall, "Semtree: Ontology-based Decision Tree Algorithm for Recommender Systems," *Int. Semantic Web Conf.*, 2008.

[44] B. Krulwich, "Lifestyle Finder: Intelligent User Profiling Using Large-Scale Demographic Data," *Artificial Intelligence Magazine*, vol. 18, no. 3, pp. 37-46, 1997.

[45] M. Pazzani, and J. Michael "A framework for collaborative, content-based and demographic filtering" *Artificial Intelligence Review* vol. 13, no. 5, pp. 393-408, 1999.

[46] M. Pazzani, and D. Billsus, "Content-based recommendation systems," *The Adaptive Web*, Springer, pp. 325—341, 2007.

[47] G. A. Wang, H. Chen, J. J. Xu, and H. Atabakhsh, "Automatically detecting criminal identity deception: an adaptive detection algorithm," *IEEE Trans. on SMC, Part A: Systems and Humans*, vol. 36, no. 5, pp. 988-999, Sep. 2006.

[48] M. B. Salem, and S. J. Stolfo, "Modeling User Search Behavior for Masquerade Detection," in *Proc.* the 14th Symposium on Recent Advances in Intrusion Detection, Ca, USA, 2011.

[49] K. Wang, and S. J. Stolfo, "One-class training for masquerade detection," in *Proc. Workshop on Data Mining for Computer Security*, pp. 19-22, Florida, USA, 2003.

[50] R. Kohavi, Wrappers for Performance Enhancement and Oblivious Decision Graphs. Ph.D. Thesis, Stanford University, STAN-CS-TR-95-1560, 1995.

Prof. Lior Rokach is a senior lecturer at the Dept. of Information Systems Eng. at Ben-Gurion University. His main areas of interest are data mining and information retrieval.

Prof. Yuval Elovici is an associate professor at the Dept. of Information Systems Eng. at Ben-Gurion University. His main research interests are Computer and Network Security, Cyber Security, Web Intelligence, Information Warfare and Machine Learning.

Dr. Asaf Shabtai is a lecturer at the Dept. of Information Systems Eng. at Ben-Gurion University. His main areas of interests are computer and network security, machine learning and data mining.

Ma'ayan Dror holds a M.Sc. in Information Systems Eng. from Ben-Gurion University.