Evaluating Value Weighting Schemes in the Clustering of Categorical Data

Periklis Andritsos
University of Trento, Italy
periklis@dit.unitn.it

Vassilios Tzerpos
York University, Canada
bil@cs.yorku.ca

Abstract

The majority of the algorithms in the clustering literature utilize data sets with numerical values. Recently, new and scalable algorithms have been proposed to cluster data sets with categorical data, data whose inherent ordering is not obvious. However, these algorithms deem all data values present in the data sets as equally important. Thus, the resulting clusters may be influenced by values that appear almost exclusively and reflect non-natural groupings.

In this paper, we present a set of weighting schemes that allow for an objective assignment of importance on the values of a data set. We use well established weighting schemes from information retrieval, web search and data clustering to assess the importance of whole attributes and individual values. To our knowledge, this is the first work that considers weights in the clustering of categorical data.

We perform clustering in the presence of importance for the values within the LIMBO framework, a new and scalable algorithm to cluster categorical data. Our experiments were performed in a variety of domains, including data sets used before in clustering research and three data sets from large software systems. We report results as to which weighting schemes show merit in the decomposition of data sets.

1 Introduction

Many industrial data analysis applications involve dealing with large and often complex data sets containing many records and attributes. Data mining applications such as clustering, classification and numerous others are commonly applied to obtain better insight on the data sets and elicit information that can be useful to the data analysts.

In this work, we focus on the application of clustering to data sets with unlabeled tuples, *i.e.*, tuples without any class label associated with them. In brief, clustering is the process of partitioning a set of tuples into meaningful groups (called clusters), such that tuples belonging to the same group are similar to each other and dissimilar to tuples of other groups. This definition assumes that there is some well-defined notion of *similarity* – or *distance* – between tuples. When tuples are defined by a set of numerical attributes, there are natural definitions of distance based on geometric analogies. These definitions rely on the semantics of the data values themselves (for example, the values $100K and $110K are more similar than $100K and $1).

The problem of clustering becomes more challenging when the data is categorical, that is, when there is no inherent distance measure between data values. This is often the case in many domains, where data is described by a set of descriptive attributes, many of which are neither numerical nor inherently ordered in any way. Recently, considerable focus has been devoted to the development of algorithms that handle categorical data [1, 4]. These algorithms treat all attributes and individual values equally. However, a domain expert clustering the same data set would invariably assign different importance to particular attributes based on her intuition. Similarly, she might consider certain data values as more important than others for the determination of the clusters.

The premise for the work presented in this paper is that by assigning different importance to attributes and/or individual values, we can direct the clustering process towards a more meaningful result. We do this by implementing a number of weighting schemes that are based on existing techniques from information retrieval and clustering of categorical values. Experiments conducted using the newly introduced LIMBO algorithm [1] demonstrate the merit of the various weighting schemes and suggest possible improvements.

In particular, we investigate weighting schemes that apply to two different types of data sets:

1. *Relational Data Sets*. These are data sets where tuples are defined over a set of different attributes. We employ two techniques from information retrieval and one from spectral graph theory, in order to produce weights for the attributes and/or individual values in such data sets:
   - *Term Frequency-Inverse Document Frequency*.
   - *Mutual Information* a particular value conveys about the rest of the values.
   - *Linear Dynamical Systems*.
2. *Graph-based data sets*. These are data sets where the objects to be clustered are in the form of a graph that demonstrates interdependencies between them. Such structures appear in numerous domains, such as in hyperlinked documents, where the objective is to group web pages with similar content, or the reverse engineering of software systems, where the objective is to

decompose them into meaningful components in order to better understand and maintain them.

In addition to the weighting schemes applied to relational data sets, we also employ the following two weighting schemes for graph-based data sets:

- We use the well known *PageRank* algorithm to assign importance to specific values.
- We utilize usage data, such as weblogs or information obtained by profiling software systems.

Our work is different in spirit from the work presented in the literature on *Feature Selection* [14]. In feature selection for clustering [21] the main focus is on the elimination of whole attributes to improve the performance of the underlying algorithm. An initial evaluation of a weighting scheme without attribute elimination is presented by Modha and Spangler for numerical data and the k-means algorithm [16]. On the other hand *Term Weighting Schemes* have appeared in Information Retrieval to ensure better search results [3, 7, 19]. These techniques, though, assume a class label assigned to every tuple and evaluate attributes according to how well they predict these labels. Finally, Gravano et al., use the *TF.IDF* weighting schemes for approximate text joins within a database system [12].

The rest of the paper is organized as follows. In Section 2, we introduce some basic concepts from information theory. and describe LIMBO, a scalable information-theoretic clustering algorithm. Section 3 introduces the data representations we consider as well as the way we incorporate weights in the clustering process. In Section 4, we describe the set of weighting schemes we consider in our approach and Section 5 presents the experimental evaluation of all our schemes within LIMBO. Finally, Section 6 concludes our work.

2 Background

This section introduces the main concepts from Information Theory that will be used throughout the paper. We also give the formulation of the Information Bottleneck method, and present a scalable clustering algorithm based on it called LIMBO.

Let T denote a discrete random variable that takes values over the set $\mathbf{T}$[1], and let $p(t)$ denote the probability mass function of T. The *entropy* $H(T)$ of variable T is defined by $H(T) = -\sum_{t\in\mathbf{T}} p(t)\log p(t)$. Intuitively, entropy captures the "uncertainty" of variable T; the higher the entropy, the lower the certainty with which we can predict its value.

Now, let T and V be two random variables that range over sets $\mathbf{T}$ and $\mathbf{V}$ respectively. The *conditional entropy* of V given T is defined as follows

$$H(V|T) = \sum_{t\in\mathbf{T}} p(t) \sum_{v\in\mathbf{V}} p(v|t)\log p(v|t)$$

[1]For the remainder of the paper, we use italic capital letters (e.g. T) to denote random variables, and boldface capital letters (e.g. $\mathbf{T}$) to denote the set from which the random variable takes values.

Conditional entropy captures the uncertainty of predicting the values of variable V given the values of variable T.

The *mutual information*, $I(T;V)$, quantifies the amount of information that the variables convey about each other. Mutual information is symmetric, non-negative, and it is related to entropy via the equation

$$I(T;V) = H(T) - H(T|V) = H(V) - H(V|T) = I(V;T) \quad (1)$$

Relative Entropy, or the *Kullback-Leibler (KL)* divergence, is a standard information-theoretic measure of the difference between two probability distributions. Given two distributions p and q over a set $\mathbf{T}$, the relative entropy is defined as follows.

$$D_{KL}[p\|q] = \sum_{t\in\mathbf{T}} p(t)\log\frac{p(t)}{q(t)}$$

Intuitively, the relative entropy $D_{KL}[p\|q]$ is a measure of the inefficiency in an encoding that assumes the distribution q, when the true distribution is p.

2.1 Information Bottleneck

The intuitive idea of producing clusters that are informative about the objects they contain was formalized by Tishby, Pereira and Bialek [22]. They recast clustering as the compression of one random variable into a compact representation that preserves as much information as possible about another random variable. Their approach was named the *Information Bottleneck (IB)* method, and it has been applied to a variety of different areas.

More formally, given a set of objects $\mathbf{T}$ expressed over set $\mathbf{V}$, we seek a clustering $\mathbf{C}_k$ of $\mathbf{T}$ (k is the desired number of clusters), such that the mutual information $I(C_k;V)$ remains as large as possible, or otherwise the loss of information described by $I(T;V) - I(C_k;V)$ is minimum. The *IB* method is generic, imposing no semantics on specific data values.

Finding the optimal clustering is an NP-complete problem [9]. Slonim and Tishby [20] propose a greedy agglomerative approach, the *Agglomerative Information Bottleneck (AIB)* algorithm, for finding an informative clustering. If set $\mathbf{T}$ contains n objects, the algorithm starts with the clustering $\mathbf{C}_n$, in which each object $t \in \mathbf{T}$ is assigned to its own cluster. Due to the one-to-one mapping between $\mathbf{C}_n$ and $\mathbf{T}$, $I(C_n;V) = I(T;V)$. The algorithm then proceeds iteratively for $n-k$ steps, reducing the number of clusters in the current clustering by one in each iteration. At step $n-\ell+1$, two clusters c_i and c_j in clustering $\mathbf{C}_\ell$ are merged into cluster c^*, to produce a new clustering $\mathbf{C}_{\ell-1}$. As the algorithm forms clusterings of smaller size, the information that the clustering contains about $\mathbf{T}$ decreases, which means that $I(C_{\ell-1};V) \le I(C_\ell;V)$. The two clusters c_i and c_j to be merged are chosen such that the loss of information $\delta I(c_i,c_j) = I(C_\ell;V) - I(C_{\ell-1};V)$, is minimum. The new cluster $c^* = c_i \cup c_j$ has [20]:

$$p(c^*) = p(c_i) + p(c_j) \quad (2)$$

$$p(V|c^*) = \frac{p(c_i)}{p(c^*)}p(V|c_i) + \frac{p(c_j)}{p(c^*)}p(V|c_j) \quad (3)$$

Tishby et al. [22] show that

$$\delta I(c_i, c_j) = [p(c_i) + p(c_j)] \cdot D_{JS}[p(V|c_i), p(V|c_j)] \quad (4)$$

where D_{JS} is the *Jensen-Shannon (JS)* divergence, defined as follows. Let $p_i = p(V|c_i)$ and $p_j = p(V|c_j)$ and let

$$\bar{p} = \frac{p(c_i)}{p(c^*)} p_i + \frac{p(c_j)}{p(c^*)} p_j$$

Then, the D_{JS} distance is defined as:

$$D_{JS}[p_i, p_j] = \frac{p(c_i)}{p(c^*)} D_{KL}[p_i||\bar{p}] + \frac{p(c_j)}{p(c^*)} D_{KL}[p_j||\bar{p}]$$

The D_{JS} distance defines a metric and it is bounded above by one. Note that the information loss for merging clusters c_i and c_j, depends only on the clusters c_i and c_j, and not on other parts of the clustering $\mathbf{C}_\ell$.

2.2 Scalable Clustering

The *AIB* algorithm suffers from high computational complexity. It is quadratic in the number of objects to be clustered, which is prohibitive for large sets. We now describe the *scaLable InforMation BOttleneck* (LIMBO) algorithm that uses distributional summaries in order to deal with large data sets [1]. This algorithm is similar in spirit to the BIRCH [28] clustering algorithm and is based on the idea that we do not need to keep whole clusters in main memory, but instead, just sufficient statistics to describe them. The full algorithm is described in [1].

The sufficient statistics are called *Distributional Cluster Features (DCFs)*. We will use them to compute the distance between two clusters or between a cluster and a tuple. Let $\mathbf{T}$ be the set of objects to be clustered expressed on the set $\mathbf{V}$, and let T and V be the corresponding random variables. Also let $\mathbf{C}$ denote a clustering of $\mathbf{T}$ and C be the corresponding random variable.

For a cluster $c \in \mathbf{C}$, the DCF of c is defined by the pair

$$DCF(c) = \Big(p(c), p(V|c)\Big)$$

where $p(c)$ is the probability of cluster c ,($p(c) = |c|/|\mathbf{T}|$), and $p(V|c)$ is the conditional probability distribution of the values in $\mathbf{V}$ given the cluster c. If c consists of a single object $t \in \mathbf{T}$, $p(t) = 1/|T|$ and $p(V|c)$ is computed as described later in Section 3.

Let c^* denote the cluster we obtain by merging two clusters c_1 and c_2. The DCF of the cluster c^* is equal to

$$DCF(c^*) = \Big(p(c^*), p(V|c^*)\Big)$$

where $p(c^*)$ and $p(V|c^*)$ are computed using Equations 2 and 3, respectively. Finally, given two clusters c_1 and c_2, we define the distance, $d(c_1, c_2)$, between $DCF(c_1)$ and $DCF(c_2)$ as the information loss $\delta I(c_1, c_2)$ incurred after merging the corresponding clusters. $d(c_1, c_2)$ is computed using Equation 4.

The importance of DCFs lies in the fact that they can be stored and updated incrementally. The probability vectors are stored as sparse vectors, reducing the amount of space considerably. Each DCF provides a summary of the corresponding cluster, which is sufficient for computing the distance between two clusters. We use a tree data structure, termed DCF-tree. Our scalable algorithm proceeds in three phases. In the first phase, the DCF tree is constructed to summarize the data. In the second phase, the DCFs of the tree leaves are merged to produce a chosen number of clusters. In the third phase, we associate each tuple with the DCF to which it is closest.

Phase 1: Insertion into the DCF tree. The objects to be clustered are read from disk one at a time and at any point in the construction of the tree, the DCFs at the leaves define a clustering of the tuples seen so far. Each non-leaf node stores DCFs that are produced by merging the DCFs of its children. More about the construction of DCF-tree can be found in [1]. After all objects are inserted in the tree, the DCF-tree embodies a compact representation in which the data set is summarized by the information in the DCFs of the leaves. This summarization is based upon a parameter ϕ which controls the accuracy of the model represented by the tree. More precisely we use the quantity $\phi \cdot \frac{I(T;V)}{|T|}$ as a threshold and merge DCFs at the leaf level of the tree that do not exceed it. Smaller values of ϕ result in more compact summarizations. For instance, when $\phi = 0.0$, we only merge identical objects and our technique becomes equivalent to AIB.

Phase 2: Clustering. Upon the creation of the tree, we can apply AIB in a much smaller number of objects represented by the DCFs in the leaves.

Phase 3: Associating object with clusters. For a chosen value of k, Phase 2 produces k DCFs that serve as *representatives* of k clusters. In the final phase, we perform a scan over the data set and assign each tuple t to the cluster c such that $d(t, c)$ is minimized.

The next section describes in detail the representation of our data as well as the way we apply any of the weighting schemes presented later in Section 4.

3 Data Representation

The input to our problem is the set of n tuples $\mathbf{T}$ and the set of values $\mathbf{V} = \mathbf{V}_1 \cup \ldots \cup \mathbf{V}_m$, which denotes the set of all possible values in attributes $A_1, A_2, \ldots, A_m$, respectively. [2] Let d denote the size of set $\mathbf{V}$. We shall denote with T and V the random variables that range over sets $\mathbf{T}$ and $\mathbf{V}$, respectively.

3.1 Relational Data

Relational data refers to data sets where all tuples are defined over the same number of values, one from each of the attributes. We represent our data as an $n \times d$ matrix M, where $M[t, v] = 1$ if tuple $t \in \mathbf{T}$ contains value $v \in \mathbf{V}$,

[2]In the interest of brevity, we will refer to attribute values as values.

and zero otherwise. Note that the vector of a tuple t contains m 1's. For a tuple $t \in \mathbf{T}$, defined over exactly m attribute values, we then define:

$$p(t) = 1/n \tag{5}$$

$$p(v|t) = \begin{cases} 1/m & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Intuitively, we consider each tuple t to be equi-probable and normalize matrix M so that the row corresponding to tuple t holds the conditional probability distribution $p(V|t)$. Given M, we can define mutual information $I(T;V)$ and cluster the tuples in $\mathbf{T}$ into a clustering $\mathbf{C}$ such that the information loss $I(C;V) - I(T;V)$ is minimum.

3.2 Market-Basket Data

Market-basket data describes a database of transactions for a store, where every tuple consists of the items purchased by a single customer. It is also used as a term that collectively describes a data set where the tuples are sets of values of a single attribute, and each tuple may contain a different number of values.

Let $\mathbf{V}$ be the universe of all d values that appear in the data set. Again, we represent our data as an $n \times d$ matrix M, where $M[t, v] = 1$ if attribute value $v \in \mathbf{V}$ appears in tuple $t \in \mathbf{T}$, and zero otherwise. Note that the vector of a tuple t contains $d_t \leq d$ 1's. For a tuple $t \in \mathbf{T}$, we define:

$$p(t) = 1/n \tag{7}$$

$$p(v|t) = \begin{cases} 1/d_t & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Again, we consider each tuple t to be equi-probable and normalize matrix M so that the row corresponding to tuple t holds the conditional probability distribution $p(V|t)$. Clustering, then, proceeds as in the case of relational data.

When dealing with graph-based data sets, we first transform them into market-basket data sets, and then apply the process described above. Since the objective with a graph-based data set is to cluster the nodes of the graph, the transformation into market-basket data proceeds as follows: Each node n_i of the graph corresponds to a tuple while the values in the tuple is the set of nodes that are adjacent to n_i in the graph.

3.3 Incorporating Weighting Schemes

In both relational and market-basket data, we normalized each row of matrix M in order to make it a probability distribution. This way we consider the appearance of a value in a tuple as probable as any of the other values in the same tuple. If we represent importance with numerical weights, the aforementioned conceptualizations of our data sets involve values with equal weights.

Our goal in this paper is to study how particular weighting schemes over the data we cluster influence the resulting clusters. Before introducing these schemes, we describe how to apply a weighting scheme through an example. Consider the tuples of the market-basket data set given in Table 1. According to Equation 7, we set $p(t_i) = 1/4$, $1 \leq t_i \leq 4$, and according to Equation 8, the matrix M that is used to represent this data set is given in Table 2

t_1	a	b	c	d	e
t_2	b	c	e		
t_3	d	e			
t_4	a	b	d		

Table 1: Market-Basket Data

	a	b	c	d	e
t_1	1/5	1/5	1/5	1/5	1/5
t_2	0	1/3	1/3	0	1/3
t_3	0	0	0	1/2	1/2
t_4	1/3	1/3	0	1/3	0

Table 2: Market-Basket Data Representation

By applying Equation 4 to the example data set, we can compute all pairwise values of information loss (δI). These values are given in Table 3. The value in position (i, j) indicates the information loss we would incur, if we chose to group the i-th and the j-th tuple together.

	t_1	t_2	t_3	t_4
t_1	-	0.1182	0.1979	0.1182
t_2	0.1182	-	0.2977	0.3333
t_3	0.1979	0.2977	-	0.2977
t_4	0.1182	0.3333	0.2977	-

Table 3: Pairwise δI values for vectors of Table 2

From the information losses of Table 3, we conclude that the algorithm is going to merge either pair (t_1, t_2) or (t_1, t_4), which have the lowest value of 0.1182. We also notice that pairs (t_2, t_3) and (t_3, t_4) are equidistant.

Let us now assume that a particular weighting scheme has assigned weights to the five values in the example (larger weights correspond to more important values). Denoting the vector of weights with w, we may have $w = (0.01, 0.01, 0.01, 0.96, 0.01)$. This rather extreme weight distribution considers value d to be the most important one. In order to have the importance of each value reflected in matrix M, we replace each appearance of a value in a tuple with its weight, and normalize the rows of matrix M so that they sum up to one. Using the example vector w given above the new matrix M is given in Table 4.

	a	b	c	d	e
t_1	0.01	0.01	0.01	0.96	0.01
t_2	0	0.3333	0.3333	0	0.3333
t_3	0	0	0	0.9897	0.0103
t_4	0.0102	0.0102	0	0.9796	0

Table 4: Data Representation with weights

The new pairwise distances are given in Table 5.

Our first observation is that in the presence of importance for the values, there are no ties in the information losses among the tuples. Moreover, the closest pair is now (t_1, t_4), which are almost identical since they both share the value with highest importance. The clustering algo-

	t_1	t_2	t_3	t_4
t_1	-	0.4511	0.0076	0.0050
t_2	0.4511	-	0.4833	0.4834
t_3	0.0076	0.4833	-	0.0077
t_4	0.0050	0.4834	0.0077	-

Table 5: Pairwise δI values for vectors of Table 4

rithm, as an initial step, will merge tuples t_1 and t_4 into cluster t_{14} and the new probability distribution $p(V|t_{14})$ is given in Table 6.

	a	b	c	d	e
t_{14}	0.0101	0.0101	0.0050	0.9698	0.0050
t_2	0	0.3333	0.3333	0	0.3333
t_3	0	0	0	0.9897	0.0103

Table 6: New Data Representation with weights

The new pairwise distances are given in Table 7.

	t_{14}	t_2	t_3
t_{14}	-	0.3820	0.3298
t_2	0.3820	-	0.4833
t_3	0.3298	0.4833	-

Table 7: New Pairwise δI values for vectors of Table 6

This table dictates that tuple t_3 and cluster t_{14} will be merged next. After this illustrative example we are now ready to formally define the data set representation in the presence of weights for the attribute values.

If $\mathbf{V}$ is the universe of all d values that appear in the data set and w a vector of their importance, where $|w| = d$, we represent our data as an $n \times d$ matrix M, where $M[t, v] = w(v)$ if attribute value $v \in \mathbf{V}$ appears in tuple $t \in \mathbf{T}$, and zero otherwise. For a tuple $t \in \mathbf{T}$, we define:

$$p(t) = 1/n \qquad (9)$$

$$p(v|t) = \begin{cases} w(v)/\sum_{v' \in \mathbf{V}}(w(v')) & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \qquad (10)$$

The only difference from the representations in equations 5 through 8 is in the definition of distribution $p(V|t)$, where we first substitute each entry equal to 1 with the weight of the corresponding value v and normalize each vector so that it sums up to one. Note that our definition is general enough to cover both relational and market-basket data sets. In the former case, if we only have weights for each attribute rather than for each value, we may proceed as above after giving each value the weight of its corresponding attribute.

In the following section, we present weighting schemes for both attributes and values.

4 Data Weighting Schemes

In this section, we present in detail the weighting schemes we consider for our data sets.

4.1 Mutual Information

The first weighting scheme that we propose is based on mutual information. Given a set of attributes $A_1, A_2, \ldots, A_m$, we can define a probability distribution of the values of each one of them. The dependence score for attribute A_i and A_j, is computed as the mutual information $I(A_i; A_j)$ given by Equation 1. Note that mutual information is symmetric and the lower its value the weaker the dependence between A_i and A_j . We suggest computing the weight $MI(A_i)$ for each feature A_i as the average mutual information between A_i and the rest of the attributes:

$$MI(A_i) = \frac{1}{m-1} \sum_{j=1, j \neq i}^{m} I(A_i; A_j)$$

The higher the value of $MI(A_i)$, the more important A_i is.

Given a relational data set, we compute the weight of each one of the attributes and label the values of the data sets with the weights of their corresponding attributes. More formally, if v_{ij} is the j-th value that belongs to the set of values $\mathbf{V}_i$ of attribute A_i, then $w(v_{ij}) = MI(A_i)$.

The previous definition of MI holds for relational data sets. In the case of market-basket data sets the tuples are expressed over a single attribute. Hence, we need to define the probability distribution in a different manner.

For each value $v_i \in \mathbf{V}_i$, we define the probability

$$P_{present}(v_i) = \frac{\text{number of times } v_i \text{ appears}}{n} \qquad (11)$$

Equation 11 is the probability of finding value v_i in one of the tuples in the data sets. Therefore, using $P_{present}$ and $P_{absent} = 1 - P_{present}$ we can compute the entropy $H(v_i)$ of value v_i. Similarly we can define the joint distribution of pairs of values $v_i \in \mathbf{V}_i$ and $v_j \in \mathbf{V}_j$ and compute the joint entropy $H(v_i, v_j)$. The mutual information of values $v_i \in \mathbf{V}_i$ and $v_j \in \mathbf{V}_j$ given by Equation 1 and the MI value of v_i can be computed by

$$MI(v_i) = \frac{1}{d-1} \sum_{j=1, j \neq i}^{d} I(v_i; v_j)$$

Using Equation 10, we can define the probability of the values given the tuples, $p(V|t)$, and continue with the application of the LIMBO algorithm.

4.2 Linear Dynamical Systems

Dynamical Systems have been previously used in the clustering of attribute values in a relational data set [10]. In this case, the data set is represented as a hypergraph whose nodes are the values in the data set and there is an undirected edge between two values that appear in a tuple together. An example of a relational data set together with its hypergraph is given in Figure 1.

Given a set of d values, the initial set of weights, which is called the *initial configuration*, is a d-dimensional vector w of real numbers. The dynamical system repeatedly applies a function $f : \mathbb{R}^n \to \mathbb{R}^n$ and the configuration at which the values in the d-dimensional vector do not change, or otherwise $f(w_i) = w_{i-1}$, with i representing the current weight configuration, is called a *fixed point* of the dynamical system.

	A	B	C
t_1	a	w	1
t_2	a	x	1
t_3	b	y	2

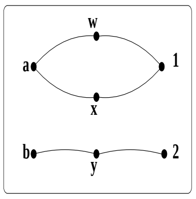

Figure 1: Relational Data Set with its hypergraph

The dynamical system that f describes is given in Figure 2 [10]. Following the steps, we update the weight w_v of each value v. In Figure 2, the symbol $\bigoplus$ denotes the combination operator. Several choices for the combination operator have appeared in the literature [10]. We shall use the summation operator, hence the term *Linear Dynamical Systems (LDS)*. Intuitively, for each value, we sum the weights of the values with which it co-occurs in the data set. To update all the values in the data set, a full pass over the data is required. Upon that, we normalize the weight vector w so that the weights sum to one and check if $f(w_i) = w_{i-1}$.If this is the case, the dynamical system has converged and the final set of weights is stored in w_i. If not, more iterations are performed until we reach a fixed point.

```
Dynamical System

To update weight $w_v$:
        For each tuple $\tau = \{v, u_1, \ldots, u_m\}$
        containing $v$ do:
                $\chi_\tau = \bigoplus(u_1, \ldots, u_m)$
        $w_v \leftarrow \sum_\tau \chi_\tau$
```

Figure 2: Updating weights in a dynamical system

Little is known with respect to the theoretical justification as to why Dynamical Systems converge [10]. Experimental results, however, have shown that a Linear Dynamical System usually converges in less than 10 iterations [10].

In our work, we use Linear Dynamical Systems to derive weights for the values in both kinds of data sets. The more a value co-occurs with other values in the data set the higher its weight. We should note that Linear Dynamical Systems elicit weights for individual values, as opposed to whole attributes in the *MI* scheme.

4.3 TF.IDF

In this subsection we introduce the use of the well-established *Term Frequency-Inverse Document Frequency, (TF.IDF)*, weighting scheme from information retrieval [3]. Given a collection of d values $\mathbf{V}$ and n tuples $\mathbf{T}$, the *TF.IDF* weight of a value $v \in \mathbf{V}$ is defined as

$$TF.IDF(v) = tf(v) \cdot \log\Big(idf(v)\Big)$$

where $tf(v)$ (term frequency) is the frequency of value v in a tuple $t \in \mathbf{T}$ (for relational data sets all values have $tf(v) = 1$ for obvious reasons) and $idf(v)$ (inverse document frequency) is the fraction n/n_v, with n_v being the number of tuples containing the value v. Drawing the analogy with information retrieval, we consider our tuples as a set of documents and our values as the set of terms over which these documents are expressed.

Intuitively, the *TF.IDF* weight of a value is high if this value appears many times within a tuple and at the same time a smaller number of times in the collection of the tuples. The latter means that this value conveys high discriminatory power. For example in a data warehouse of software artifacts, file `stdio.h`, which is used by a large number of software files will have a lower *TF.IDF* compared to file `my_vector.h`, which is connected to a smaller fraction of files.

Once vector w of the weights of all values in $\mathbf{V}$ is defined, we normalize it so that it sums up to one. Hence, the resulting weights correspond to the impact of the values in the data set. Note that the *TF.IDF* scheme can be computed both for relational and market-basket data.

4.4 PageRank

PageRank is a weighting scheme proposed and widely used in search engines [6] to compute a page's importance (or relevance). *PageRank* can be used when the relationships among different web pages are given by a graph. We shall draw an analogy with a data set whose values are related with each other and this relationship is realized through a directed graph (note that in the case of *Dynamical Systems* there is no direction associated with the edges of the hypergraph). The main idea behind *PageRank* is that a value v is deemed important if it is being pointed to by *good* values.

More precisely, let us denote with G the graph that relates the values in $\mathbf{V}$. *PageRank* performs a random walk over the nodes of G. The walk starts at a random node according to some distribution, usually uniform. Intuitively, the weight of a node n_0 is the fraction of time spent at this node, which is proportional to the number of visits to this node. The *PageRank* of a node n_0 with $C(n_0)$ outgoing links is computed as [6]:

$$PR(n_0) = (1-\alpha) + \alpha\Big(\frac{PR(n_1)}{C(n_1)} + \ldots + \frac{PR(n_s)}{C(n_s)}\Big) \quad (12)$$

where $n_1, \ldots, n_s$ are the nodes that point to n_0. The parameter α is a *damping factor*, which can be set between 0 and 1. A common value for α is 0.85, the value we used in our experiments.

The *PageRank* of each page depends on the *PageRank* of the pages that point to it. To reach a final weight vector w of *PageRank* weights, $PR(v)$ of each value can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized adjacency matrix of G. As in the case of *Dynamical Systems*, the iterations stop when there are no changes in the vector w of PR values as given by Equation 12.

4.5 Usage Data

Edges in a graph-based data set indicate only potential relationships between the objects they connect. For example, a link on a webpage indicates a potential path that a user might follow. A procedure call in the call graph of a software system may or may not be executed when the system is run. Furthermore, it is quite common that particular edges are heavily used, while others are used only rarely.

These observations indicate that the static picture of a graph-based data set might belie what actually happens when the system it represents is in use. It is intuitive to conjecture that the amount of usage of a particular object is related to its importance.

For this reason, the fifth weighting scheme we implemented for this work is based on usage data. Assuming that each edge in the graph-based data set is associated with a weight that represents its usage, each value in the corresponding market-basket data set was assigned a weight equal to the weight of the edge that connects the node represented by the value to the node represented by the tuple (the unweighted transformation of a graph-based data set to a market-basket data set was described at the end of Section 3.2). The weights of the values in the same tuple were, of course, normalized prior to the execution of LIMBO.

It is interesting to note that, in contrast to the *PageRank* weighting scheme, the same value might be given a different non-normalized weight when it appears in different tuples (a particular procedure will not be called with the same frequency by all its callers).

4.6 Weight smoothing

An interesting observation that was confirmed by early experiments is that the weights assigned by the weighting schemes presented so far may not always be beneficial to the clustering process. For example, nodes deemed highly relevant by *PageRank* may not be as important for clustering purposes. In software clustering there is the well-established notion of "omnipresent" nodes [17], i.e. nodes with large in- or out-degree. It is often beneficial to minimize the effect such nodes have to the clustering process.

For this reason, we investigated several variations to the five weighting schemes (we call this process weight smoothing). More precisely, for each weighting scheme, we also applied the following variations:

• The values with the largest weights were identified, and their weight was modified to the minimum weight in the data set. We performed experiments where the values affected were in the top 5, 10, or 20 percentile.

• All values were sorted according to their weight. If v_1 is the value with the smallest weight, and v_d is the value with the largest weight, this variation assigns a new weight to v_i equal to the old weight of v_{d-i+1}.

In the following section, we present experiments with all weighting schemes and their variations on six different data sets.

5 Experimental Evaluation

In this section, we perform a comparative evaluation of the different weighting schemes using the LIMBO clustering algorithm on both relational and market-basket data sets.

We should note again that our intention is not to test the scalability or the performance of LIMBO under different parameter settings. We observed experimentally that the branching factor B of the DCF-tree does not significantly affect the quality of the clustering. We set $B = 4$, so that the Phase 1 insertion time is manageable (smaller values of B lead to higher insertion cost due to the increased height of the DCF tree). At the same time we have explored a large range of values for ϕ (results are given in [1]). Generally speaking, larger values for ϕ (around 1.0) delay leaf-node splits and create a smaller tree with a coarse representation of the data set. On the other hand, smaller ϕ values incur more splits but preserve a more detailed summary of the initial data set. The value $\phi = 0.0$ makes our method equivalent to the AIB, since only identical objects are merged together. The ϕ value used in the experiments described below was chosen to ensure timely results, *i.e.*, larger ϕ values were used for the larger data sets.

We experimented with the following six data sets.

5.1 Relational Data Sets

The first two data sets have been previously used for the evaluation of clustering algorithms [4, 10, 13, 1]. We also compiled a third data set of research publications as described below.

Congressional Votes. This relational data set was taken from the *UCI Machine Learning Repository* [24]. It contains 435 tuples of votes from the U.S. Congressional Voting Record of 1984. Each tuple is a congress-person's vote on 16 issues and each vote is either YES or NO (there were also 288 UNKNOWN values). The total number of values in the data set is, therefore, 48. We ran LIMBO with $\phi = 0.0$ and use this data set to test the performance of the weighting schemes on data sets with small attribute domains.

Mushroom. The Mushroom relational data set also comes from the UCI Repository. It contains 8,124 tuples, each representing a mushroom characterized by 22 attributes, such as color, shape, odour, etc. The total number of distinct attribute values is 117. There are 2,480 missing values. This data set contains attributes with variable domains. We used $\phi = 0.5$ for this data set.

DBLP Bibliography. This relational data set was created from the XML document found at `http://dblp.uni-trier.de/xml/`. This document stores information about different types of computer science publications. In order to integrate the information in a single relation, we chose to use IBM's schema mapping tool that permits the creation of queries to transform the information stored in XML format into a relational data set [18]. We specified a target schema (the schema over which the tuples in the relation are defined) containing 13 attributes (`Author`, `Publisher`, `Year`, `Editor`, `Pages`, `BookTitle`, `Month`, `Volume`, `JournalTitle`, `Number`, `School`, `Series`, `ISBN`). We specified correspondences between the source XML schema and the 13 attributes. The queries given by the mapping tool where used to create a relation that contained $50,000$ tuples and $57,187$ attribute values. Each tuple contains information about a single author. Therefore, if a particular publication involved more than one author, the mapping created additional tuples for each one of them. Moreover, the highly heterogeneous information in the source XML document (information regarding conference, journal publications, etc.) introduced a large number of NULL values in the tuples of the relation. We used this highly heterogeneous relation containing a large number of values (including missing ones) to demonstrate the strength of our approaches in suggesting a clustering. The ϕ value used was 1.2.

5.2 Market-basket data sets

The three market-basket data sets we used for our experiments came from large software systems. Such systems commonly yield large dependency graphs that can be used in order to cluster the system's resources (source files, procedures, classes) into meaningful subsystems. By transforming these dependency graphs into market-basket data sets, as explained in section 3.2, we were able to assess the merit of the applicable weighting schemes. A description of all three software systems follows:

TOBEY. This is a proprietary industrial system that is under continuous development. It serves as the optimizing back end for a number of IBM compiler products. The version we worked with was comprised of 939 source files and approximately 250,000 lines of code. An authoritative decomposition for TOBEY was obtained over a series of interviews with its developers. We used $\phi = 0.0$ for this data set.

Linux. We experimented with version 2.0.27a of this free operating system that is probably the most famous open-source system. This version had 955 source files and approximately 750,000 lines of code. An authoritative decomposition for Linux was presented in [5]. Due to the relatively small size of this data set, we used $\phi = 0.0$ again.

Mozilla. The third market-basket data set we used for our experiments was derived from Mozilla, an open-source web browser. We experimented with version 1.3 that was released in March 2003. It contains approximately 4 million lines of C and C++ source code.

We built Mozilla under Linux and extracted its static dependency graph using CPPX, and a dynamic dependency graph using *jprof*. A decomposition of the Mozilla source files for version M9 was presented in [11]. For the evaluation portion of our work, we used an updated decomposition for version 1.3 [27].

It is interesting to note that Mozilla was the only software system that was used to evaluate the usage data weighting scheme. The main reason for this was the fact that in order to extract meaningful usage data from a software system, one needs a comprehensive test suite that ensures good coverage of as many execution paths as possible. Such a test suite was not available for TOBEY or Linux. However, we were able to use the Mozilla "smoketests" for this purpose. The dynamic dependency graph we obtained contained information about 1202 of the 2432 source files that are compiled under Linux. The results presented in this section are based on the classification of these 1202 files. The ϕ value used was 0.2.

5.3 Quality Measures for Clustering

Clustering quality lies in the eye of the beholder; determining the best clustering usually depends on subjective criteria. For this reason, we will use a variety of evaluation measures in order to assess the merit of the obtained clusterings.

Category Utility (CU): Category utility [15] is defined as the difference between the expected number of attribute values that can be correctly guessed given a clustering, and the expected number of correct guesses with no such knowledge. CU depends only on the partitioning of the attribute values by the corresponding clustering algorithm and, thus, is a more objective measure. Let $\mathbf{C}$ be a clustering. If A_i is an attribute with values v_{ij}, then CU is given by the following expression:

$$CU = \sum_{c \in \mathbf{C}} \frac{|c|}{n} \sum_i \sum_j [P(A_i = v_{ij}|c)^2 - P(A_i = v_{ij})^2]$$

In order to compare clusterings with different number of clusters, Fisher's COBWEB clustering system [8] introduced the average CU value per cluster. This value is defined as $CU_{avg} = \frac{CU}{k}$, where k is the number of clusters. We use this measure in order to evaluate the clusterings obtained from the relational data sets.

MoJo: Many data sets commonly used in testing clustering algorithms include a variable that is hidden from the algorithm, and specifies the class with which each tuple is associated. As mentioned above, the market-basket data sets we experimented with include such a variable, since an authoritative decomposition is available for all of them. This variable is *not* used by the clustering algorithms. While there is no guarantee that any given classification corresponds to an optimal clustering, it is nonetheless enlightening to compare clusterings with pre-specified classifications of tuples.

To do this, we used the MoJo distance measure [23, 26]. MoJo distance between two different partitions A and B of the same data set is defined as the minimum number of *Move* or *Join* operations one needs to perform in order to transform either A to B or vice versa (*Move* refers to assigning a tuple to a different cluster, while *Join* refers to merging two clusters into one). Intuitively, the smaller the MoJo distance between an automatically created clustering A and the pre-specified classification B, the more effective the algorithm that created A.

The MoJo distance measure has been used to evaluate the effectiveness of software clustering algorithms [2, 25]. Since our market-basket data sets contain software data, it seems appropriate to use MoJo distance to evaluate the merit of the weighting schemes we will apply to them.

Information Loss, (IL): We also use the information loss $I(A;T) - I(A;C_k)$ to compare clusterings. The lower the information loss, the better the clustering. For a clustering

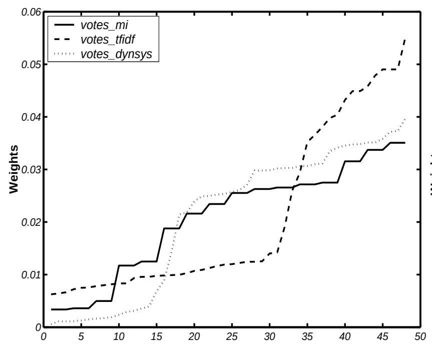

Figure 3: Votes Weights

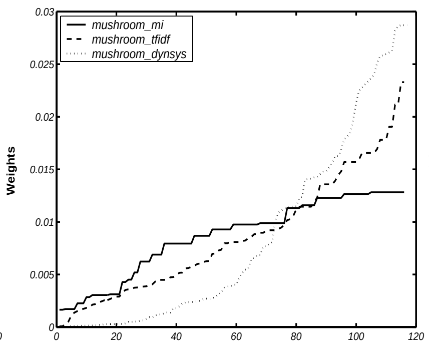

Figure 4: Mushroom Weights

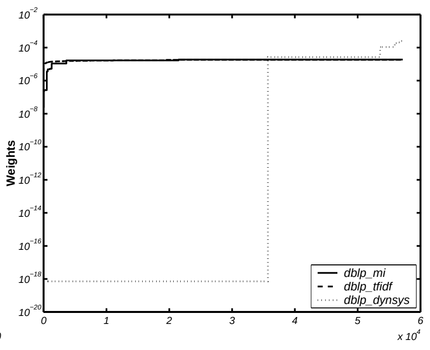

Figure 5: DBLP Weights

Votes ($\phi = 0.0$)				*Mushroom* ($\phi = 0.5$)				*DBLP* ($\phi = 1.2$)			
Scheme	k	CU_{avg}	IL**(%)**	**Scheme**	k	CU_{avg}	IL**(%)**	**Scheme**	k	CU_{avg}	IL**(%)**
None	2	1.4017	73.25	**None**	**3**	**1.6070**	**79.24**	None	3	0.4089	90.72
MI	**2**	**1.4349**	**63.38**	**MI**	**3**	**1.6070**	**72.71**	MI	2	0.6002	92.75
LDS	**2**	**1.4397**	**71.67**	LDS	4	1.0399	59.44	LDS	3	0.6172	90.00
IDF	2	1.3850	77.47	IDF	4	1.0399	58.11	**IDF**	**3**	**0.6174**	**89.04**

Table 8: **Results for relational data sets**

with low information loss, given a cluster, we can predict the attribute values of the tuples in the cluster with relatively high accuracy. We present IL as a percentage of the initial mutual information lost after producing the desired number of clusters using each algorithm. However, special attention must be paid since clusterings with smaller k values tend to incur larger values of information loss. We report the value of IL as an indication of the information content of the resulting clusters.

5.4 Relational Data: Results and Observations

For the relational data sets we ran LIMBO without any weights as well as in the presence of weights given by the *MI*, *LDS*, and *TF.IDF* weighting schemes. Figures 3, 4 and 5 depict the weight distributions of the three weighting schemes on the Votes, Mushroom and DBLP data sets, respectively (the weight values were sorted in ascending order to facilitate visualization).

For the Votes data set, Figure 3 indicates that *MI* and *LDS* assign weights in a similar fashion, although *LDS* does assign significantly smaller weight to 20% of the values. On the other hand, *TF.IDF* assigns smaller weight to about half of the values, while the weights increase for the rest of them. The latter ones correspond to YES or NO values that do not appear many times in the data set.

The Mushroom data set contains values that are almost equally distributed in the data set. Hence, as Figure 4 depicts, the *TF.IDF* scheme does not assign the highest weights as in Votes. A similar trend with respect to the weights produced by LDS is observed here as well. The weights for approximately 20% of the values are significantly lower than the rest. Finally, *MI* demonstrates behaviour similar to the one in the Votes data set, which can be characterized as more conservative than the other weighting schemes.

The main lesson learned from the distribution of the weights in the DBLP data set (the y-axis is in logarithmic scale) is that *MI* and *TF.IDF* produce similar distributions of weights. On the other hand, the large number of missing values in different attributes and the large number of values in the tuples related to the same publication forced *LDS* to elicit two significantly different categories of weights.

The results of clustering the three relational data sets (without any weight smoothing) are given in Table 8. In order to choose an appropriate number of clusters, we start by creating decompositions for all values of k between 2 and a large value. For the experiments performed for these data sets, the chosen value was 50. For these clusterings we compute the value of CU_{avg} and choose the clustering that had the maximum CU_{avg} value, *i.e.* a clustering where values can be predicted with the highest accuracy in their corresponding clusters. The weighting schemes that performed best are shown in bold.

From these results we observe that in the Votes data set there is hardly any difference among the three schemes. In all cases, the number of clusters with the lowest CU_{avg} is the same. *MI* and *LDS* produce slightly better quality results both with respect to the CU_{avg} and IL. A possible explanation for the similarity between the results could be the fact that the domain of all attributes is the same (YES, NO, UNKNOWN).

In the Mushroom data set the *MI* weighting scheme gave the best results. The value of CU_{avg} is the same as in the case where no weights were introduced. However, and for the same number of clusters for both cases, *MI* resulted in clusters with small IL value.

Finally, in the DBLP data set, all weighting schemes showed merit. This result is intuitive since weighting schemes balanced abnormalities, such as the high number of `NULL` values. For example, the *TF.IDF* scheme assigned a very small weight to the `NULL` values that appear almost exclusively in some attributes, driving the result of the clustering to more meaningful and informative clusters.

We also applied the same weighting schemes but with smoothed weights, as explained in section 4.6. The ob-

Votes ($\phi = 0.0$)			
Scheme	k	CU_{avg}	IL**(%)**
MI-5%	**2**	**1.4031**	**69.91**
MI-10%	2	1.1889	73.34
MI-20%	3	0.9266	65.52
LDS-5%	2	1.4321	68.97
LDS-10%	**2**	**1.4393**	**68.89**
LDS-20%	2	1.4206	71.26
IDF-5%	2	1.4386	76.73
IDF-10%	2	1.4426	75.82
IDF-20%	**2**	**1.4433**	**73.29**

Mushroom ($\phi = 0.5$)			
Scheme	k	CU_{avg}	IL**(%)**
MI-5%	4	1.0399	60.68%
MI-10%	3	1.0666	69.09%
MI-20%	**3**	**1.0670**	**69.24%**
LDS-5%	3	0.9719	68.71%
LDS-10%	3	0.9718	65.68%
LDS-20%	3	0.9717	65.50%
IDF-5%	4	1.0399	59.14%
IDF-10%	4	1.0401	58.56%
IDF-20%	**3**	**1.0666**	**69.17%**

DBLP ($\phi = 1.2$)			
Scheme	k	CU_{avg}	IL**(%)**
MI-5%	3	0.6001	90.68%
MI-10%	3	0.6001	92.09%
MI-20%	**3**	**0.6201**	**89.24%**
LDS-5%	3	0.6174	89.97%
LDS-10%	3	0.6171	90.20%
LDS-20%	3	0.6089	91.04%
IDF-5%	4	0.6171	89.45%
IDF-10%	3	0.6072	90.56%
IDF-20%	**3**	**0.6233**	**89.18%**

Table 9: **Results for relational data sets with smoothed weights**

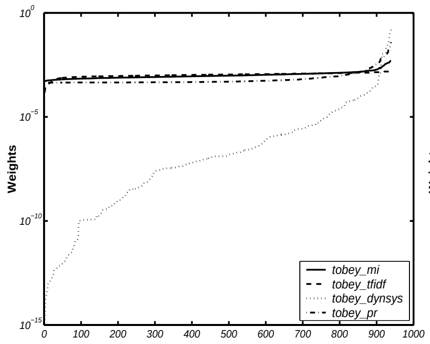

Figure 6: TOBEY Weights

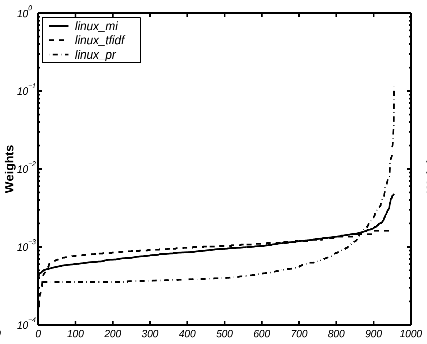

Figure 7: LINUX Weights

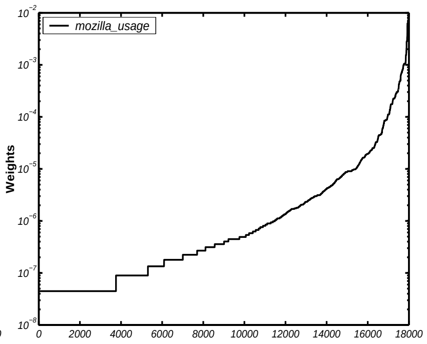

Figure 8: MOZILLA Weights

tained results are shown in Table 9.

Our first observation in the Votes data set is that clustering results are deteriorating as *MI* and *LDS* weight values are smoothed. On the contrary, the results of *TF.IDF* have improved. *TF.IDF* is a scheme that gives higher weights to values that appear less often in the tuples. The smoothing results prove that in the case of this data set, such values are less important than the scheme considers and by decreasing their value the results are better.

The same observation holds for the other two data sets. In the case of DBLP, where single authors or conference names appear with high weights in the case of *TF.IDF*, we see that smoothing out the weights of these values to the weights that correspond to `NULL` values, the clusters are more informative and the publications better separated.

5.5 Market-Basket Data: Results and Observations

LIMBO was also applied to the three market-basket data sets using all weighting schemes and their variations. In the same way as in the relational data sets, in order to choose an appropriate number of clusters, we start by creating decompositions for all values of k between 2 and a large value. For the experiments performed in market-basket data sets, the chosen value was 150, a value that turned out to be sufficient for our purposes.

Let C_k be a clustering of k clusters and C_{k+1} a clustering of $k+1$ clusters. If the cluster representatives created in Phase 2 reflect inherent groupings in the data, then these neighbouring clusterings must differ in only one cluster. More precisely, if two of the clusters in C_{k+1} get merged, this should result in C_k. Using MoJo [26], we can detect these clusterings by computing the distance between C_{k+1} and C_k, *i.e.* the value of $MoJo(C_{k+1}, C_k)$. If this value is equal to one, the difference between the two clusterings is a single join of two clusters of C_{k+1}, to produce the k clusters of C_k. As a result, k is chosen as the smallest value, for which $MoJo(C_{k+1}, C_k) = 1$.

Figures 6, 7, and 8 present the weight distribution for the three market-basket data sets and the applicable weighting schemes. In all figures, the y-axis is in logarithmic scale.

In Figure 6 we present the weight distribution of all four schemes. We observe that *MI*, *TF.IDF* and *PageRank* produce weights in the same range. In the case of *LDS*, the weights produced are rather smaller and with a larger range. Larger weights correspond to nodes with large in- and out- degrees.

The weight distribution in the LINUX data set for all schemes follows the same pattern as in the TOBEY data set. To illustrate the differences among the weights of *MI*, *TF.IDF* and *PageRank* schemes we chose not to draw the distribution of *LDS* in Figure 7. This figure shows that *MI* and *TF.IDF* elicit similar and more conservative weights compared to *PageRank*, which gives a high weight to a number of values. These values correspond to nodes that are pointed to by other important nodes in the graph of the LINUX system.

Finally, Figure 8 depicts the weight distribution produced in the presence of usage data. The distributions of the other four schemes are similar to the previous two data sets described above. The main observation from the distribution of usage weights is that there is a wide range of weights (the smallest weights are 5 orders of magnitude smaller than the largest ones).

The clustering results we obtained are shown in Table 10 (weighting schemes performing best are shown in bold).

An immediate observation is that the *TF.IDF* weighting

TOBEY ($\phi = 0.0$)			
Scheme	k	$MoJo$	IL**(%)**
None	80	311	30.59
MI	33	341	42.94
LDS	59	476	20.61
IDF	**102**	**292**	**27.17**
PageRank	24	571	41.33

LINUX ($\phi = 0.0$)			
Scheme	k	$MoJo$	IL**(%)**
None	56	237	36.03
MI	70	237	30.95
LDS	41	286	31.31
IDF	**81**	**225**	**20.09**
PageRank	24	340	39.66

MOZILLA ($\phi = 0.2$)			
Scheme	k	$MoJo$	IL**(%)**
None	10	600	63.25
MI	125	428	23.64
LDS	32	528	36.31
IDF	**68**	**406**	**33.19**
PageRank	48	478	33.14
Usage	61	440	47.21

Table 10: **Results for market-basket data sets**

TOBEY ($\phi = 0.0$)			
Scheme	k	$MoJo$	IL**(%)**
MI-5%	97	323	28.51
MI-10%	16	383	53.75
MI-20%	38	328	41.69
LDS-5%	42	486	20.61
LDS-10%	28	540	30.15
LDS-20%	37	447	34.27
IDF-5%	67	369	33.87
IDF-10%	27	333	46.41
IDF-20%	27	333	46.41
PageRank-5%	82	310	29.08
PageRank-10%	61	312	34.33
PageRank-20%	53	321	37.02
InvPageRank	**86**	**297**	**29.90**

LINUX ($\phi = 0.0$)			
Scheme	k	$MoJo$	IL**(%)**
MI-5%	94	245	27.15
MI-10%	68	240	32.12
MI-20%	90	256	27.79
LDS-5%	52	281	28.29
LDS-10%	45	315	31.11
LDS-20%	31	305	37.02
IDF-5%	97	248	26.42
IDF-10%	47	257	37.60
IDF-20%	46	257	37.95
PageRank-5%	56	244	29.08
PageRank-10%	62	235	32.96
PageRank-20%	36	230	28.00
InvPageRank	**79**	**226**	**29.26**

MOZILLA ($\phi = 0.2$)			
Scheme	k	$MoJo$	IL**(%)**
MI-5%	97	425	27.33
MI-10%	70	423	32.42
MI-20%	100	411	26.96
LDS-5%	100	423	26.90
LDS-10%	89	435	34.02
LDS-20%	69	452	32.84
IDF-5%	28	482	47.01
IDF-10%	132	419	23.19
IDF-20%	132	419	23.23
PageRank-5%	80	436	27.18
PageRank-10%	55	435	34.23
PageRank-20%	**98**	**407**	**27.12**
InvPageRank	91	416	28.34
Usage-5%	68	665	46.79
Usage-10%	73	673	46.80
Usage-20%	80	673	46.74
InvUsage	89	678	49.01

Table 11: **Results for market-basket data sets with smooth weights**

scheme outperforms all others, including the scheme that uses no weights. This can be attributed to the fact that the way *TF.IDF* assigns weights corresponds well to the way software architects would assign importance to artifacts of their system. Artifacts used by the majority of the system are probably library functions that are not very important (low *idf*), while artifacts rarely used are unlikely to be central to the system's structure (low *tf*).

A further observation is that the *LDS* weighting scheme performs quite poorly. Its weight distribution forecasted a deviant behaviour, but the most likely explanation is the fact that it assigns large importance to nodes of large in- and out-degree. This property is shared by the *PageRank* weighting scheme. Our results corroborate that this is not a desirable property for software data.

Another interesting observation is that the usage data weighting scheme performs rather well with the Mozilla data set. Even though it is outperformed by *TF.IDF*, it still improves significantly on using the static dependency graph (represented by the None weighting scheme). Further experiments with more software systems are, of course, required to determine whether this is generally true.

Finally, it is interesting to note that the *MI* weighting scheme performs well consistently. With the exception of TOBEY, it is only slightly edged by *TF.IDF*. This might indicate that it is a weighting scheme that is not influenced from the type of data set used, a quite desirable property.

We also performed experiments with the smoothed variations of the weighting schemes. The results are shown in Table 11.

It is interesting to note that, in agreement with our observations above, the performance of the *LDS* weighting scheme improves in certain cases. This phenomenon is even more dramatic with the *PageRank* weighting scheme. In many cases, the obtained clustering is only slightly worse than the one produced by *TF.IDF*.

It was quite intriguing to observe that the inverse *PageRank* weighting scheme produced results that were among the best. This indicates that importance for web search engines does not imply importance for clustering algorithms. In fact, quite the opposite seems to be the case.

As expected, the performance of *TF.IDF* dropped when its weight structure was modified. A similar behaviour was observed for the usage data weighting scheme. This indicates that these weighting schemes in their pure form encapsulate well the properties of software decompositions.

Finally, the performance of *MI* does not fluctuate significantly, confirming our belief that it is a conservative, stable, and effective weighting scheme.

6 Conclusions

This paper presented an evaluation of certain weighting schemes within a clustering algorithm for categorical data. We implemented and experimentally assessed the usefulness of such schemes on a variety of relational and market-basket data sets, the latter ones from the field of software reverse engineering.

Our approach has the added benefit that no changes need to be performed on the clustering algorithm per se. Rather, the only necessary step to be taken is the analysis of the data set and elicitation of value weights. From our experiments, we can reach the following general conclusions:

- When the number of clusters in a data set is small the weighting schemes do not offer considerable merit. This is shown through our initial experiments on relational data sets.
- The *MI* weighting scheme performs consistently well in a variety of domains.
- When graph-based data sets are clustered, the *TF.IDF* weighting scheme seems to perform best. Especially in software systems, this scheme decreases the effect of “omnipresent” nodes appropriately in order for the clusters to reflect natural groupings of the data.
- The *PageRank* weighting scheme seems to be inappropriate for clustering purposes. Interestingly, the inverse *PageRank* weighting scheme performs well in the software clustering domain.
- The performance of the *LDS* weighting scheme is overall lower than the rest of the schemes. However, it was interesting to discover that it assigns weights in a more dramatic fashion than other weighting schemes. Such a property might be desirable in domains other than the ones examined in this paper.

Certain avenues for further research present themselves. For example, we should experiment with different combination operators for Dynamical Systems before assessing their overall performance. We would also like to collect usage data from more software systems in order to assess the usefulness of the usage data weighting scheme better. We are definitely excited to investigate other types of weighting schemes that are potentially useful to the clustering process. Finally, applying LIMBO on both numerical and categorical data with relative weights to the various types of data used as input, is also a possibility for future work.

References

[1] P. Andritsos, P. Tsaparas, and R. J. M. K. C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In *EDBT*, Mar. 2004.

[2] P. Andritsos and V. Tzerpos. Software Clustering based on Information Loss Minimization. In *WCRE*, 2003.

[3] R. B.-Yates and B. R.-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, 1999.

[4] D. Barbará, J. Couto, and Y. Li. COOLCAT: An entropy-based algorithm for categorical clustering. In *CIKM*, McLean, VA, 2002.

[5] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a case study: Its extracted software architecture. In *ICSE*, 1999.

[6] S. Brin and L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks*, 30(1–7):107–117, 1998.

[7] F. Debole and F. Sebastiani. Supervised Term Weighting for Automated Text Categorization. In *SAC-03*, Melbourne, FL, USA, 2003.

[8] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[9] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[10] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. In *VLDB*, New York, NY, 1998.

[11] M. W. Godfrey and E. H. S. Lee. Secrets from the monster: Extracting mozilla’s software architecture. In *CoSET*, 2000.

[12] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text Joins in an RDBMS for Web Data Integration. In *WWW*, Budapest, Hungary, 2003.

[13] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Atributes. In *ICDE*, Sydney, Australia, 1999.

[14] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Pubs, 1998.

[15] M. Gluck and J. Corter. Information, Uncertainty, and the Utility of Categories. In *COGSCI*, Irvine, CA, USA, 1985.

[16] D. S. Modha and W. S. Spangler. Feature Weighting in k-Means Clustering. *Machine Learning*, 52(3), 2003.

[17] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5:181–204, Dec. 1993.

[18] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, and R. Fagin. Translating web data. In *VLDB*, Hong Kong, China, Aug. 2002.

[19] G. Salton and C. Buckley. Term-weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[20] N. Slonim and N. Tishby. Agglomerative Information Bottleneck. In *Neural Information Processing Systems, (NIPS-12)*, pages 617–623, 1999.

[21] L. Talavera. Feature Selection as a Preprocessing Step for Hierarchical Clustering. In *ICML*, 1999.

[22] N. Tishby, F. C. Pereira, and W. Bialek. The Information Bottleneck Method. In *37th Annual Allerton Conference on Communication, Control and Computing*, Urban-Champaign, IL, 1999.

[23] V. Tzerpos and R. C. Holt. MoJo: A distance metric for software clusterings. In *WCRE*, 1999.

[24] UCI. ML Repository. *http://www.ics.uci.edu/∼mlearn/MLRepository.html*.

[25] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *In Submission*.

[26] Z. Wen and V. Tzerpos. An optimal algorithm for MoJo distance. In *IWPC*, May.

[27] C. Xiao. Software clustering using static and dynamic data. *Master’s thesis, Department of Computer Science, York University, in preparation*.

[28] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient Data Clustering Method for Very Large Databases. In *SIGMOD*, 1996.