1-1-2012

A new semantic similarity join method using diffusion maps and long string table attributes

Bilal Hani Hawashin
Wayne State University,

A NEW SEMANTIC SIMILARITY JOIN METHOD USING DIFFUSION MAPS AND LONG STRING TABLE ATTRIBUTES

by

BILAL HAWASHIN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2011

MAJOR: COMPUTER SCIENCE

Approved by:

____

Advisor **Date**

____

____

____

DEDICATION

To my father, Hani Hawashin, who scarified the most for us. To my late mother, and to my brother and sisters, this work is dedicated.

ACKNOWLEDGEMENTS

First, I would like to thank God for giving me the opportunity to start my Ph.D. program in 2007 and leading me to finish my Ph.D. program in 2011. I also would like to express my sincere gratitude to my advisor and academic grandfather, Dr. Farshad Fotouhi, for all his encouragement, support, and help during my Ph.D. program. I am really proud to be a student of such a distinguished professor. With his advising and guidance, I successfully completed my work and also had a great chance to make myself more capable. In addition, I am grateful to my Dissertation Committee members: Dr. Chandan Reddy and Dr. Zaki Malik in the Department of Computer Science at Wayne State University, and Dr. William Grosky in the Department of Computer and Information Science at University of Michigan-Dearborn for being on my committee and giving me constructive suggestions and valuable comments on my work.

I would like to give my special thanks to Dr. Traian Marius Truta in the Department of Computer Science at Northern Kentucky University for working with me as a co-author in some of my research papers. I would like to thank all my friends here at Wayne State University for giving me lots of help to early adapt to school life. I also would like to thank my academic colleagues in my lab for their academic cooperation and close friendships.

I would like to thank my family and friends, especially those who supported me before I came to here. I would not be able to start the PhD program without your support. Finally, the financial support of Alzaytoonah University of Jordan is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND PROBLEM STATEMENT

1.1 Motivation

With the rapid growth of the data everyday, a new important and challenging issue is to integrate data from different and heterogeneous resources. Furthermore, there are some organizations that has different departments or parts that use different systems with lack of co-ordination. One important data type that is used commonly in such systems is the String data type. String data is everywhere, and many applications use it. Examples are product catalogs (for books, music, software, etc.), electronic white and yellow page directories, and specialized information sources such as patent databases and customer's data. The integration of string data in relational databases is done typically by the join operation. The commonly used join method is the exact join (also called equi-join or natural join), which is joining two rows from two different tables (and could be from the same table) if they have the same exact value(s) in their join attribute(s), However, exact join will not be suitable when the databases are heterogeneous. Some reasons are as follows.

- The data in heterogeneous databases could have different conventions in some fields such as name, date, and address. For example, a customer's name could be John A. Smith in some database and Smith, John in another.
- There could be inconsistent data, which means that incorrect data could occur in some database, such as having two different birthdates for the same person.

- The attributes used to refer to the same entity in different databases could be different, because the attributes used in some database depends on the domain. For example, the attribute Major could be important for a university database but is not very important in a bank database.
- The string data could contain typographical errors, as they are commonly entered by humans. For example, we could have Jonh Smith instead of John Smith. A research done by Kukich [40] used a set of large databases and showed that they contain 1% - 3.2% typing errors, 1.5% - 2.5% spelling errors, and in interesting but important note, Kukich showed that the percentage of spelling errors in Dutch surnames for example was 38%.

In this work, we assume that the tables to be joined have the same schema, which means that the fields in these two tables are identical. Other area of study concerns about solving the differences when the fields are not the same. For example, when we have the field *Name* in one table and the fields *First Name* and *Last Name* in another table, or *Address* in one table and *Street*, *City*, *Zip Code*, and *Country* in another table. In this work, the concentration is on joining the records that refer to the same entity after solving any difference in the data representation that could occur between them. This problem is commonly called ***Similarity Join*** *or* ***Approximate Join***[41][42].Another definition is to join the pairs of records whose similarity is greater than a user defined similarity threshold T.

1.2 Applications of Similarity Join

The problem of Similarity Join has been widely studied in the previous decades and referred to by many different terms such as: record linkage, entity matching, duplicate detection, merge-purge, data deduplication, instance identification, database hardening, name matching, conference resolution, and identity uncertainty. This area has many applications in different domains, such as Artificial Intelligence, Database, Statistics, Signal Processing, Information Retrieval, and Metadata Interoperability. Our concentration in this work is on its applications in the databases domain.

1.3 String Similarity Measurement Function

String Distance Measurement Function is a mapping from two strings *x* and *y* into a real number r that represents the distance between the two strings. *String Distance Measurement Function* is the opposite of the string distance measurement function (or shortly, distance function). Clearly, the more the distance between the two strings, the less the similarity between them.

Metric distance function *d* is the distance function that satisfies the following four properties.

1. $d(x, y) \geq 0$,
2. $d(x, y) = 0$ <u>if and only if</u> $x = y$,
3. $d(x, y) = d(y, x)$,
4. $d(x, z) \leq d(x, y) + d(y, z)$,

Whereas *x* and *y* are two points in the space.

Many measurements have been proposed in order to find the similarity (or distance) between two strings. Such measurements could be divided into three main categories: Character Based Measurements, Token Based Measurements, and Phonetic Based Measurements.

1.3.1 Character Based Measurements

Those measurements compare the two strings character by character in order to specify the distance (or similarity) between them. The main methods under this category are the following.

- Edit Distance [43].
- Jaro and Jaro-Winkler Distance Metrics [47][48].

1.3.2 Token Based Measurements

The previous measurements are not suitable in some cases. For example, when there is different order of the words. So, other measurements have been proposed to compare words or parts of the words (QGrams) instead of characters. Those measurements are called Token Based Measurements. Examples of these measurements are the following.

- QGrams with TF.IDF [44][45].
- Jaccard Similarity [46].

Finally, other measurements are under Phonetic Based Measurements, which consider two strings similar if they have similar sounds. One example of such measurements is Soundex[49].

1.4 Problem Statement

As stated before, many similarity measurements have been proposed in the past years. Examples are Edit Distance[43], Q Grams[44][45], Jaccard Similarity[46], Jaro[47], and Jaro-Winkler[48], and Soundex[49]. Most of such measurements are mainly used to find the similarity (distance) between short string values. We used the term *Short String Attributes* to refer to string attributes with limited number of characters, such as person name and person address. Besides, we used the term *Long String Attributes* to refer to string attributes of unlimited length, such as product description, research paper abstract, user feedback, and movie summary.

Although many works have studied Similarity Join with short attributes, a few works have included the use of long attributes to assist the similarity join process and enhance the performance. Obviously, long attributes contain much more information than short attributes. Therefore, there is a great potential that using such attributes to detect similar records could improve the overall similarity join accuracy. Furthermore, long attributes exist in most of the databases, and finding an efficient method to perform similarity join using long attributes would complement the existing work that concentrates on short attributes.

One issue here is how to differentiate long attributes from short attributes. For this purpose, we conducted a preliminary experiments on the IMDB database[17]. This database contains a set of movies, and it will be explained in details in chapter 3. We used the well known Edit Distance method [43] to detect similar records using both *Movie Name* and *Movie Summary* attributes separately. The average number of characters per Movie Name was 16.2 characters, while the average number of characters per Movie Summary was 912.7 characters. The average edit distance for similar string pairs using Movie Name was 2.7, while the average edit distance for similar string pairs using Movie Summary was 626.3, which is extremely high. This does not mean necessarily that using short attributes is better than using long attributes. In contrast, as we will show later, using long attributes would improve the similarity join performance significantly when a suitable semantic similarity method is used. Therefore, it is clear that not all similarity methods and measurements are applicable to long attributes. This introduces my contribution.

First, we proposed an efficient semantic similarity join method for joining tables according to their long attributes under supervised learning, when a training set exists. The training set has examples of similar record pairs, which would assist in detecting similar record pairs in the testing set. Such similarity join method for long attributes would assist or replace the existing short attribute similarity join methods. As part of this method, we found the best semantic similarity measurement for long string values under supervised learning.

Second, we proposed a privacy preserving similarity join protocol for joining tables using their long attributes under similarity thresholds, when no training set is available. Basically, the sources involved in the similarity join process may not want to share their data, and may seek to share the similar records only. In this case, the content of a source needs to be hidden and protected from being disclosed to other sources. A few works have been done in this area, and most of the work concentrated on methods that are applicable to short attributes only. As we explained in our first contribution, using long attributes in the similarity join can increase the similarity join accuracy. Up to our knowledge, no work proposed a privacy preserving similarity join method when the join attribute is a long attribute. Therefore, we proposed an efficient privacy preserving similarity join protocol using long attributes under similarity thresholds.

Third, we proposed a privacy preserving similarity join protocol when the join attribute is long attribute under supervised learning, when a training set is available. Using a small training set can significantly improve the similarity join performance. Again, up to our knowledge, no work has been done to propose a privacy preserving similarity join protocol for long attributes under supervised learning, even though this would improve the similarity join accuracy when there are privacy constraints. We proposed this protocol and we enhanced its performance by using selective records instead of random records. We improved the similarity join performance furthermore by using mulit-label supervised learning instead of single-label learning as the former method is closer to many real-life applications.

Fourth, we proposed an efficient semantic similarity join method to be used with long attributes under unsupervised learning, when neither training set nor similarity thresholds are used. This scenario is common in many practical applications, as it would be very expensive or even impossible to have a training set or adopt a similarity threshold. Furthermore, we proposed a solution for expandable clusters (groups). This case is common because databases are not static, and their content is updated by every transaction. Therefore, the number of clusters may increase by time. Up to our knowledge, no previous work proposed an efficient solution to similarity join method that considers database expansion.

Both Algorithm 1.1 and Fig. 1.1 provide our new semantic similarity join method.

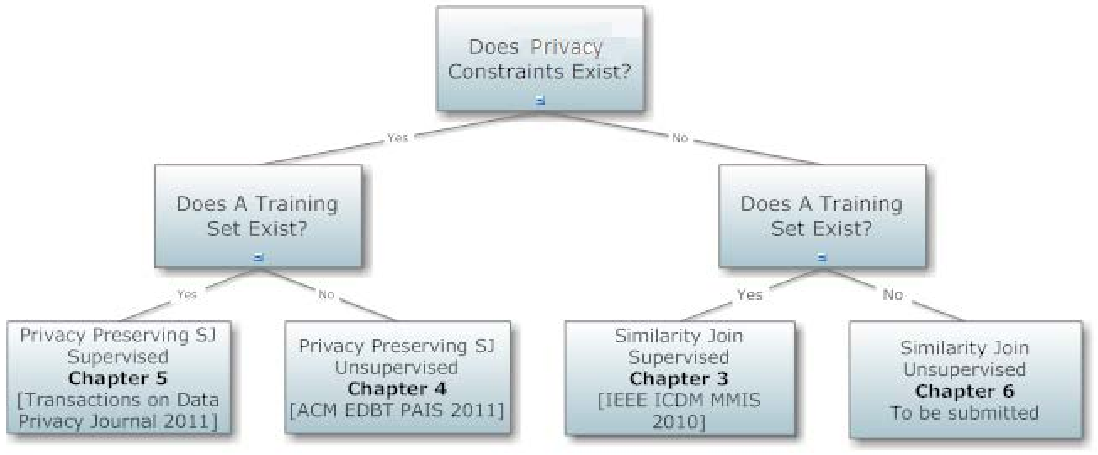

Does Privacy Constraints Exist?
Yes
No
Does A Training Set Exist?
Does A Training Set Exist?
Yes
No
Yes
No
Privacy Preserving SJ Supervised Chapter 5 [Transactions on Data Privacy Journal 2011]
Privacy Preserving SJ Unsupervised Chapter 4 [ACM EDBT PAIS 2011]
Similarity Join Supervised Chapter 3 [IEEE ICDM MMIS 2010]
Similarity Join Unsupervised Chapter 6 To be submitted

Figure 1.1: My Contribution.

Algorithm1.1: NEW SEMANTIC SIMILARITY JOIN METHOD USING DIFFUSION MAPS AND LONG STRING TABLE ATTRIBUTES

Input: Two sources A and B, each has a long attribute X.

Output: Semantically joining similar records.

Algorithm:

```
(1) If privacy constraints exist
(2)     If a training set exists
(3)         PRIVACY_PRESERVING_SEMANTIC_SJ_SUPERVISED
(4)     Else
(5)         PRIVACY_PRESERVING_SEMANTIC_SJ_UNSUPERVISED
(6)     End;
(7) Else
(8)     If a training set exists
(9)         SEMANTIC_SJ_SUPERVISED
(10)    Else
(11)        SEMANTIC_SJ_UNSUPERVISED
(12)    End;
(13) End;
```

1.5 Organization

The rest of this dissertation is organized as follows. Chapter 2 illustrates the related work. Chapter 3 describes our semantic similarity join method using long attributes under supervised learning. Chapter 4 presents our privacy preserving semantic similarity join method using long attributes under similarity thresholds. Chapter 5 presents our privacy preserving semantic similarity join method using long attributes under supervised learning. Chapter 6 describes our semantic similarity join method using long attributes under unsupervised learning and expandable databases. Finally, Chapter 7 concludes this dissertation and provides the future work directions.

CHAPTER 2

RELATED WORK

In order to find similar records, many methods have been proposed. The proposed methods are divided into the following two basic categories according to their objectives.

- Enhancing Similarity Join Accuracy.
- Minimizing Number of Record Comparisons.

Our concentrating in this dissertation is on the first research area that studies how to enhance the accuracy of the similarity join methods. For the second research area, one can refer to [81][82]. The methods that aim at enhancing similarity join accuracy could be divided into three main categories:

- Machine Learning Methods.
- Probabilistic Methods.
- Rule Based Methods.

Our concentration in this work is on Machine Learning Methods, because such methods were used extensively in our work and because of the fast growing rate of this area in the recent years.

2.1 Similarity Join Using Machine Learning Methods

These methods could be divided into the following three parts.

- Supervised Methods.
- Unsupervised Methods.
- Hybrid Methods (Semi Supervised Methods).

2.1.1 Supervised Methods

These methods use a training file to build a model that could be used later with testing cases. Such methods are divided according to the size of the used training file into the following two types.

- Supervised Methods with large training files.
- Supervised Methods with small training files.

2.1.1.1 Supervised Methods with Large Training Files

As stated earlier, these methods depend on the existence of a large training dataset that has prelabeled pairs of records. [50] proposed learning the affine gap edit distance parameters for each field alone using the Expectation Minimization algorithm (EM) using the training data. The learned parameters for each field are the parameters that produce the minimum classification error according to that field. Next, the distance vectors for all record pairs in the training file are found using the learned parameters and used by SVMlight as a training data. Finally, the trained SVMlight can decide about any record pair given its distance vector. He showed that this method outperformed other approaches such as considering the record a single large field. Later, [51] proposed a method to build an approximate operator tree depending on a training data. The training data contains pairs of records each of which is labeled as match or non-match. Some algorithms were proposed in this work to get approximate result for the similarity join operator. Other method is proposed by [24], which used a set of operations such as equal, abbreviation, concatenation, and synonym in order to transform one string field into another. A training file with matched pairs and non matched pairs is used, and the transformation graph using the previous operations is found for each pair. Every operation is given a weight according to its appearance in the transformation graphs of matched and non matched pairs. Finally, a transformation graph is constructed for the testing pair and the probability of a match given the operations in this transformation graph is calculated using the probabilities of the operations in the training transformation graphs. [53] proposed making a graph for all the records in the database, linking together those classified as matched, and consider all the records belonging to the same connected component a match. However, this method is not always correct as the transitivity assumption does not always hold.

2.1.1.2 Supervised Methods with Small Training File

The problem of the first part is that it requires a large number of records in the training set. If a training set of small number of records is available, the most confusing records could be selected and labeled manually. This would provide more information with very much smaller training data. Such methods fall under the category of active learning. Example for this is ALIAS which is proposed by Sarawagi and Bhamidipaty[54].

2.1.2 Unsupervised Methods

Unsupervised methods do not use a training dataset. These methods are more practical as it is not always easy to find a training set. However, the accuracy of these methods is not as high as that obtained by supervised methods. Unsupervised Methods could be divided into the following two major parts.

- Clustering Methods.
- Distance Methods.

2.1.2.1 Clustering Methods

Clustering is grouping similar records together according to a similarity measurement and optimization criteria. [55] considered each record pair is a point in an *n* dimensional space, where *n* is the number of fields in each of the two tables. The point that represents the rows I and *j* is represented as $P_{i,j} = [d_1, d_2, \ldots, d_n]$, such that d_1 is the dissimilarity between the rows I and *j* according to the first field, and so on. After representing all the pairs accordingly, they are clustered into three clusters: *Matched*, *Unmatched*, and *Possibly Matched*. The cluster that is closest to the origin is the Matched cluster, and the cluster that is the furthest from the origin is the Unmatched one. The Possibly Matched one is in the middle, and called sometimes *The Reject Region*, where the method failed to give a decision. Related work is done in [56] and studied the *Entity Resolution Problem.* Clear example for problem is the paper resolution problem, where each paper represents a group of references. As stated earlier, the objective is to cluster papers according to their similarity. This method used iteration to link duplicate references in different papers according to two criteria: the similarity between the two references and the similarity of the context (papers) in which the two references appear, instead of using the references similarity only.

2.1.2.2 Distance Methods

Many methods have been proposed in this category. Examples are [57][58]. Guha[59] proposed ranked list merging, which is to find the distance between two records according to one field, and to repeat this step with other fields. The result will be *n* ranked lists, assuming that we have *n* attributes. Next, those records are merged such that the resulting list has the minimum aggregate rank distance when compared to all *n* lists. This list can show the top *k* matching records. Other methods have been proposed in order to find the distance between hierarchial data, such as *Customer Address*. One of the main used distances in this context is the *Tree Edit Distance*, as the hierarchial data could be represented as labeled trees. However, this methods is expensive. Therefore, some distance approximations have been proposed such as using *pqGram Distance*[60] which is an efficient approximation for the tree edit distance, and is sensitive to the place where the two trees differ, as the leave difference is less important than the higher nodes difference.

Final issue here is how to determine the cut-off value for the match, and one solution is to use a training data to conclude this value. However, this will have the disadvantage of using a training set again, and the aim of using distance based methods is to avoid using data sets. [61] discussed this issue and argued that the very high degree of similarity means that the pair is matched. Similarly, the very low degree indicates that it is mismatched. However, the difficulty and confusion lies in the similarity values located in the middle.

2.1.3 Hybrid Methods

Some methods are composed of both supervised and unsupervised methods. Such methods have more accuracy than the unsupervised techniques and more time efficient in many cases. For example, a clustering algorithm could be applied first to classify a small portion of the record pairs as matched or unmatched, and then this portion serves as a training set to some classifier that will classify the remaining portion of the record pairs. [62] proposed an online learning algorithm to combine many basic similarity functions with weights using average perception weighting. After applying the function and finding the pairwise similarity matrix among all the rows in the two tables, they compared three clustering Hierarchical Agglomerative Clustering techniques(HAC): single link HAC, group average HAC, and complete link HAC, according to their ability to find the matched records. The results showed that the complete link HAC outperformed the other two clustering methods.

2.2 Privacy Preserving Similarity Join

A few researchers have concentrated on performing similarity join under Privacy Constraints. Examples of such works includes [29], which introduced a protocol to perform similarity join using phonetic encodings, [30], which proposed a privacy preserving record matching protocol on both data and schema levels, [31], which concentrated on the e-health applications and its intrinsic private nature, and [32], which used a Term Frequency – Inverse Document Frequency (TF.IDF) based comparison function and a secure blocking schema. Other methods concentrated on using encryption to preserve privacy such as [33][34].

2.3 Similarity Join Using Long Attributes

Regarding the use of long string attributes in the similarity join process, and up to our knowledge, no work has been done to study the effect of the long attributes in the similarity join process, even though they could improve the accuracy significantly. Furthermore, no work has found the best similarity measurement to be used with such attributes during the similarity join process in order to complement the existing work, which is concentrated on the short attributes. Such lack of literature work on a promising method was the motivation to my work in this dissertation.

2.4 Similarity Join Using Dimensionality Reduction Methods

Regarding the use of dimensionality reduction methods for similarity join, [52] used the LSI method with short string fields in order to join the values in such fields semantically. Other than this work, up to our knowledge, no work has included the dimensionality reduction methods as part of the similarity join methods. It should be noted that our concentration here is on relational databases. Dimensionality reduction methods have been used widely in unrelational databases such as document clustering and classification [20].

CHAPTER 3

SEMANTIC SIMILARITY JOIN METHOD USING LONG ATTRIBUTES UNDER SUPERVISED LEARNING

3.1 Introduction

As stated previously, many similarity join measurements have been proposed in the literature. Although much research has been done in similarity join of short attributes, a few works have included the use of long attributes to assist the similarity join process and enhance the performance. Obviously, long string values contain much more information than short string values. Therefore, it is worthwhile to study the effect of using long attributes on the similarity join performance. Besides, most databases include attributes of long string values. However, many proposed similarity join methods use measurements that are not suitable for such long values. Two main reasons are the cost of using such measurements with long string values and the deficiency in detecting the semantic correlations among terms by concentrating only on the syntax representation of the terms. For example, the complexity of the edit distance measurement depends on the lengths of the two strings to be compared. Therefore, the longer the strings, the more the similarity join running time. Besides, edit distance deals with the character representation of the strings, without considering the semantic relationships between them. It is worthwhile to find an efficient semantic method for joining long string values and study its effect on the similarity join performance. Applications of such semantic methods could be in joining values of long attributes such as paper abstracts, movies summaries, product descriptions, user comments, and so on.

In this chapter, we studied diffusion maps [1], latent semantic indexing [2], eigenvectors [3], SoftTFIDF with cosine similarity [7], TF.IDF with cosine similarity [18], and a variant of diffusion maps. Most of these methods have strong theoretical foundations and have proved their superiority in many applications. Therefore, we compared their performance as candidate semantic similarity join methods for long attributes under supervised learning. It should be noted that many short string measurements were not included in this comparison because of their high running time cost and low accuracy when applied to long string values. In order to evaluate the performance, we used two datasets, Pubmed Medical Dataset [27] and IMDB Movies dataset [17]. The SVM classifier was used with the Pubmed dataset, whereas bagging was used with the IMDB dataset because it is more commonly used with datasets having a large number of classes.

Our work in this chapter is divided into two phases. First, finding the best semantic method for joining values of long attributes. Second, comparing the performance of this method with the existing, commonly used, short string methods. For phase one, we used TF.IDF weighting [18] and Chi-square dimensionality reduction method [19] to eliminate noisy and insignificant words. Later, the diffusion maps method was compared with LSI and eigenvector–based dimensionality reduction methods to find the best method with the best number of dimensions. The best method was compared with SoftTFIDF with cosine similarity, TF.IDF with cosine similarity, and a variant of diffusion maps with the previously selected best number of dimensions. Both the Abstract attribute from the Pubmed dataset and the Movie Summary attribute from the IMDB Movies Dataset were used in this phase. Regarding phase two, after obtaining the best semantic method, we compared its performance on joining values of long attributes with the performance of SoftTFIDF method on joining values of short attributes, as SoftTFIDF is a superior short string method [7]. Both the Title and the Keywords attributes from the Pubmed dataset were used with the SoftTFIDF method, while the Abstract attribute from Pubmed was used with the best semantic method obtained from phase one. Supervised learning using an SVM or bagging was used to compare the performance of the previous methods after each phase.

The contributions of this work are as follows:

- Adopting the use of long attributes to replace or assist the short attributes to increase the similarity join accuracy under supervised learning.
- Finding an efficient semantic method that can be used for joining values of long attributes.
- Proposing a scalable solution for large datasets and large dimensionality.

The rest of this chapter is organized as follows. Section 3.2 describes the candidate semantic methods to be compared. Section 3.3 describes phase one of the work, which compares the semantic methods for joining long attributes. Section 3.4 illustrates phase two of the work, which compares the best long string method with SoftTFIDF, which is one of the top short string methods [7], and Section 3.5 is the summary.

3.2 Semantic Methods for Joining Long Attributes

In the following subsections, we will describe candidate semantic methods for joining long string values. These methods will be compared later in this chapter.

3.2.1 Diffusion Maps

Diffusion maps is a dimensionality reduction method proposed by Lafon [1]. Initially, a weighted graph is constructed whose nodes are labeled with long string values and whose edge labels correspond to the similarity between the corresponding node values. A similarity function called the kernel function, *W,* is used for this purpose. The first-order neighborhood structure of the graph is constructed using a Markov matrix *P*. In order to find similarities among non-adjacent nodes, forward running in time of a random walk is used. A Markov chain is computed for this purpose by raising the Markov matrix *P* to various integer powers. For instance, according to P_t , the t^{th} power of *P*, the similarity between two long string values *x* and *y* represents the probability of a random walk from *x* to *y* in *t* time steps. Finally, SVD( ) dimensionality reduction function is used to find the eigenvectors and the corresponding eigenvalues of $P_{t,t\geq 1.}$ The approximate pairwise long string value similarities are computed using the significant eigenvectors only. The similarity between any two long string values using such a method is called *diffusion maps similarity*. The mathematical details of diffusion maps are given below.

Consider a dataset *C* of N long string values, represented as vectors. Let *x,y* be any two vectors in *C*, $1 \leq i,j \leq N$. A weighted matrix $W_\sigma\,(x,y)$ can be constructed as

$$W_\sigma\,(x,y) = \exp(-\ \frac{D\cos(x,y)}{\sigma})\ , \tag{3.1}$$

where σ specifies the size of the neighborhoods that defines the local data geometry. As suggested in [20],

$$Dcos(x,y) = 1 - \frac{x.y}{\| x \| . \| y \|} . \quad (3.2)$$

We can create a new kernel as follows:

$$W_\sigma^\alpha(x,y) = \frac{W_\sigma(x,y)}{q_\sigma^\alpha(x) q_\sigma^\alpha(y)} , \quad (3.3)$$

Where α deals with the influence of the density in the infinitesimal

transitions of the diffusion, and

$$q_\sigma(x) = \sum_{y \in C} W_\sigma(x,y) . \quad (3.4)$$

Suppose $d_\sigma(x) = \sum_{y \varepsilon C} W_\sigma^\alpha(x,y)$, (3.5)

We can normalize the previous kernel to get an anisotropic transition kernel $p(x,y)$, as

follows:

$$\underset{\sigma}{p}(x,y) = \frac{W_\sigma^\alpha(x,y)}{d_\sigma(x)} . \quad (3.6)$$

$\underset{\sigma}{p}(x,y)$ can be considered a transitional kernel of a Markov chain on C. The diffusion

distance D_t between x and y at time t of the random walk is

$$D_t^2(x,y) = \sum_{z \varepsilon C} \frac{(p_t(x,z) - p_t(y,z))^2}{\phi_0(z)} , \quad (3.7)$$

where ϕ_0 is the stationary distribution of the Markov chain.

After using SVD( ), the Markov chain eigenvalues and eigenvectors can be obtained. Therefore, the diffusion distance D_t can be written as:

$$D_t^2(x,y) = \sum_{j\geq 1} \lambda_j^{2t} (\varphi_j(x) - \varphi_j(y))^2 . \tag{3.8}$$

We can reduce the number of dimensions by finding the summation up to a specific number of dimensions z. Thus, the mapping would be:

$$\omega : x \to (\lambda_1 \varphi_1(x), \lambda_2 \varphi_2(x), ..., \lambda_z \varphi_z(x)) . \tag{3.9}$$

We used the values of σ and α to be 10 and 1 respectively for experiments, as used in [22]. The detailed diffusion maps based algorithm for joining long string values is described later in this chapter.

3.2.2 Latent Semantic Indexing (LSI)

LSI [2] uses the Singular Value Decomposition operation to decompose the term long string value matrix M, that contains terms as rows and long string values as columns, into three matrices: T, a term by dimension matrix, S a singular value matrix, and D, a long string value by dimension matrix. The original matrix can be obtained through matrix multiplication of TSD^T. In order to reduce the dimensionality, the three matrices are truncated to z user selected reduced dimensions. Dimensionality reduction reduces noise and reveals the latent semantics present in the dataset. When the components are truncated to z dimensions, a reduced representation matrix, M_z is formed as

$$M_z = T_z S_z D_z^T \ . \tag{3.10}$$

3.2.3 EigenVectors

Here, the eigenvectors and their corresponding eigenvalues are extracted directly from the term long string value matrix [3]. Originally, each long string value is represented as a combination of all eigenvectors and their eigenvalues. A reduced number of eigenvectors, with their corresponding eigenvalues, is selected that captures most of the dataset information.

3.2.4 SoftTFIDF with Cosine Similarity

The SoftTFIDF [7] method is a modification of the well known TF.IDF weighting. In *SoftTFIDF* method, the pairs that are similar, but not identical, are included in the TF.IDF equation. According to this method, each string value is treated as a set of terms. The SoftTFIDF similarity between two string values X and Y is given as follows:

$$SoftTFIDF(X,Y)= \sum_{w \in CLOSE(\theta, X, Y)} V(w,X)V(w,Y)D(w,Y), \tag{3.11}$$

Whereas $V(w,X)$ represents the TF.IDF weighting of the term w in the string value X, $V(w,Y)$ represents the TF.IDF weighting of the term w in the string value Y, and CLOSE(θ, X, Y) represents all terms $w \in X$ such that there is some term $v \in Y$ such that $D'(w,v) > \theta$. $D(w,v)$ denotes Jaro-Winkler distance between the terms w and v. $D(w,Y) = \max_{v \in Y} (D(w,v))$. For our experiments, we used $\theta = 0.9$ as adopted in [7]. This method is a superior method for joining short string values [7], and therefore, it is worthwhile to study its performance on long string values.

It should be noted that we used this method twice in our experiments. First, in phase one, to study its performance as a candidate semantic method for joining long string values. Second, in phase two, as a superior method for short attributes to compare its performance on joining short string values with the performance of our semantic method on joining long string values.

3.2.5 TF.IDF with Cosine Similarity

TF.IDF with cosine similarity is a well known method that has been used extensively for document similarity. The TF.IDF weighting of term *w* appearing in a long string value *x* is given as follows:

$$\text{TF.IDF}(w,x)=\log(\text{tf}_{w,x}+1).\log(\text{idf}_{w}), \tag{3.12}$$

where $\text{tf}_{w,x}$ is the frequency of the term *w* in the long string value *x*, idf_w is $\frac{N}{n_w}$, where *N* is the number of long string values in the database *C*, and n_w is the number of long string values in the database that contains the term *w* in their corresponding attribute.
As any document is considered a long string value, this method is a candidate semantic method for long string values. The cosine similarity of two long string value vectors *x* and *y* is given as follows:

$$\text{Cosine\_Similarity}(x,y) = \frac{x.y}{\| x \| . \| y \|}. \tag{3.13}$$

3.3 Comparing Semantic Methods

In order to evaluate the previous methods on long string values, two datasets were used, the Pubmed Medical Dataset and the IMDB Movies Dataset. Table 3.1 below describes the use of these datasets in phase1. The following is a brief description of each dataset.

3.3.1 Pubmed Dataset

This dataset includes indexed bibliographic medical citations and abstracts. It is collected by the U.S. National Library of Medicine (NLM). It includes references from more than 4500 journals. For our experiments in phase1, we used 4000 abstracts from this dataset. We labeled every abstract to one class out of 23 classes proposed by [21]. PUBMED citations and abstracts could be accessed by PUBMED via http://www.ncbi.nlm.nih.gov/entrez/query.fcgi or by the NLM Gateway via http://gateway.nlm.nih.gov/gw/Cmd.

3.3.2 Internet Movies Database

We collected 999 movie summaries from the IMDB Movies database, which is available online via http://imdb.com. Every movie has one or more summaries, written by various users. All summaries that belong to the same movie are considered of the same class, with an average of three summaries for each class.

For our experiments, we used an Intel® Xeon® server of 3.16GHz CPU and 2GB RAM, with Microsoft Windows Server 2003 Operating System. Also, we used Microsoft Visual Studio 6.0 to read the datasets, Matlab 2008a for the implementations of the candidate semantic methods, and Weka 3.6.2 for the SVM and Bagging classifiers to get the method's performance. The semantic similarity join algorithm using long attributes under supervised learning is described hereafter and given in Algorithm 3.1 below.

Table 3.1: Datasets Description for Phase1

Dataset	**Used Number of Records**	**Number of Classes**
Pubmed	4000	23
IMDB	999	3000

For the Abstract attribute in Pubmed Dataset, we removed the stopwords and converted the text into lowercase. The term long string value frequency matrix was generated. Later, TF.IDF weighting matrix was computed, as displayed in line 02 of the algorithm. As the number of features in that matrix was around 12000 features, we used Chi-square dimensionality reduction method, presented in line 03 of the algorithm, and we selected 2% of the features (230 features) with the highest importance. The selected features were equally representing the 23 categories. The Movie Summary attribute in the IMDB dataset was manipulated similarly. The size of *M_Reduced* after using Chi-square method is R X D, such as $R < T$.

In lines 04 through 08 of the algorithm, we computed the kernel matrix using equation 3.2, given in [20], because it gives the best document clustering performance with k-means. We expected that such an equation would be efficient with supervised learning similarity join methods as well. Later, we used the Lafon implementation of diffusion maps. The function call is represented in line 09 of the algorithm. The resulting matrix Y represents the reduced diffusion maps matrix, where each long string value is represented as a row with z user selected reduced dimensions. Later, the testing long string value is processed. The TF.IDF weighting is computed for the same R terms obtained from line 03, when Chi-square method was applied to the training term long string value matrix. The resulting vector is of R cells, such that $r[i]$ represents the TF.IDF weighting of the i_{th} term in the testing long string value. The kernel of the testing value g, called *g_kernel*, is computed in line 15, and the resulting vector has D dimensions, where D is the number of training long string values in the input matrix M, such that *g_kernel[i]* represents the cosine similarity between Chi-square reduced representation of the testing value and Chi-square reduced representation of training long string value i. The diffusion maps representation for the testing value g is calculated in line 17 as adopted in [28]. The resulting vector would serve as a testing record that will be used with Y as inputs to the classifier. Finally, the classifier will predict the category for the testing long string value g, as represented in line 18. If the number of testing long string values is more than one, lines 12-17 are repeated for every testing long string value. The conversion process for N testing values is efficient, with complexity $O(c.N)$, such that c is a constant representing max(number of training long string values, Chi-square reduced number of terms). Both numbers are constants and user defined, and they are not dependent on the number of testing long string values.

Regarding LSI, we used the term long string value matrix M as an input to the SVDs( ) Matlab built-in function, along with the desired number of reduced dimensions, z. The resulting Long String Values by reduced dimensions matrix V can be used as a classifier training matrix. In order to convert testing records, we used the Matlab formulas given in [28].

For the eigenvectors method, we extracted the eigenvectors from the input term long string value matrix *M* using the Eigs( ) function. The detailed Matlab equations used are found in [28].

For SoftTfIDf, we wrote an implementation using C++. The training matrix was the pairwise SoftTFIDF similarities of the training long string values. The testing matrix was the SoftTFIDF similarity between the testing long string values and the training long string values.

After using each of the semantic methods, the SVM classifier is used with 10 fold cross validation to get the performance under that method space.

The performance measurements used for this study were *classifier F_1 rating*, *preprocessing time*, and *running tim*e. Preprocessing time includes *dataset preprocessing time*, *semantic method time*, and *classifier training time*. Running time indicates *classifier testing time*. They are defined as follows:

- Classifier F_1 rating is the harmonic mean of the classifier *recall* and the *precision*. It is given as

$$F_1 = \frac{2 * R * P}{R + P}, \tag{3.14}$$

where *R* represents the recall, which is the ratio of the relevant data among the retrieved data, and *P* represents the precision, which is the ratio of the accurate data among the retrieved data. Their formulas are given as follow:

$$R = \frac{TP}{TP + FP}, \text{ if } TP+FN > 0, \text{ otherwise undefined.} \tag{3.15}$$

$$P = \frac{TP}{TP + FN}, \text{ if } TP+FN > 0, \text{ otherwise undefined.} \tag{3.16}$$

Algorithm3.1: SIMILARITY JOIN ALGORITHM USING LONG ATTRIBUTES UNDER SUPERVISED LEARNING

Input: A vector *g* representing a test long string value.

A *T* x *D* Term Long String Value Matrix *M* with class label for each long string value.

Output: The class of the test long string value.

Algorithm:

```
(01) //process the training dataset
(02) $M\_weighted$ = find_TF.IDF _weighting($M$)
(03) $M\_Reduced$ = Chi($M\_Weighted, R$)  //$R < T$
(04) For $i$=1:$D$
(05)   For $j$=1:$D$
(06)   $Dcos(i,j)$ = 1-Cosine_Similarity($M\_Reduced_i$, $M\_Reduced_j$)
(07)   End;
(08) End;
(09) [$Y$,$S$,$V$,$A$] = Diffusion_Maps($Dcos$, 10, 1, Z)
(10) //|$Y$|= $D$ x $Z$
(11) //process the testing record g
(12) $g\_reduced$=Chi($g$) //|$g\_reduced$| = $R$
(13) $g\_weighted$ = find_TF.IDF_Weighting($g\_reduced$)
(14) For $k$=1:$D$
(15)     $g\_kernel(k)$ = Cosine_Similarity($g\_weighted$, $D_k$)
(16) End;
(17) $Diff\_Representation$=$g\_kernel$*$Y$*$S$(1:$Z$,1:$Z$)
(18) $Test\_Class$ = Classifier_Predict(Training=$Y$, Testing
(19) = $Diff\_Representation$)
```

In order to find these measurements, a two-by-two contingency table is used for each category. Table 3.2 below represents the contingency table.

To assess the global performance over all the 23 classes, the macro average F_1 measurement was used in our experiments. It is found by averaging the per-class F_1 values.

- Dataset preprocessing time represents the time needed to read the dataset and convert it to a format accepted by the candidate semantic methods.
- Semantic method time represents the time needed to perform the semantic operation on the dataset.
- Classifier training time represents the time needed by the classifier to build the model using the output of the semantic method.
- Classifier testing time represents the time to classify the testing long string values.

First, as some candidate semantic methods were dimensionality reductions methods, we compared them separately to find the best method. We used the Pubmed dataset to compare diffusion maps, latent semantic indexing, and eigenvectors on a reduced number of dimensions varying between 30 and 120 dimensions. Fig. 3.1 depicts the *F1* rating of these three methods. As stated before, 4000 abstracts were used for this purpose.

Table 3.2: The Contingency Table to Describe the Components of the Performance Measurements

Actual Class	**Predicted Class**		
		Class = Yes	***Class = No***
	Class = Yes	TP	FN
	Class = No	FP	TN

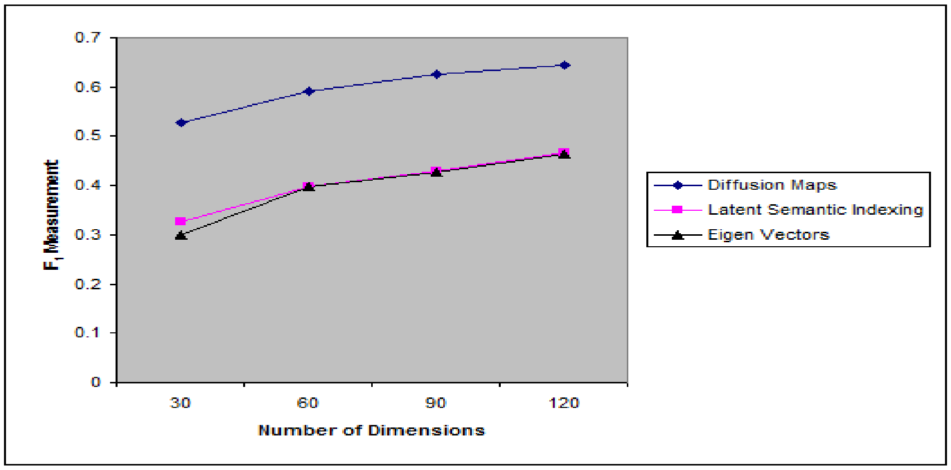


Figure 3.1: *F1* Measurement for Diff, LSI, and Eigenvectors.

Clearly, diffusion maps *F1* measurement outperformed the other two methods. Basically, diffusion maps is able to compute the relationships between attribute values, in contrast with LSI and the Eigenvectors approaches that merely map attribute values to terms. As attribute values could have overlapped terms, the ability of LSI and eigenvectors to distinguish values of different classes is less than that of diffusion maps [20].

For the preprocessing time, Table 3.3 represents the dataset preprocessing time for the three methods, whereas Fig. 3.2 and Fig. 3.3 represent the operation time and training time for these methods. Regarding the classification running time consumed by the three methods, Fig. 3.4 represents the results.

Obviously, by increasing the number of dimensions, diffusion maps algorithm tends to consume more time than that consumed by the remaining two methods. However, according to Fig. 3.1, the diffusion maps performance tends to be more stable after 60 dimensions. Even though both operation time and training time for diffusion maps on 60 dimensions is the largest among the three methods, these are one time only steps. For classification time, the time required by diffusion maps with 60 dimensions is similar to that needed by LSI and Eigenvectors.

Table 3.3: Preprocessing Time of the Three Candidate Semantic Methods on Pubmed Dataset

Method	**Time (Sec.)**
Diffusion Maps	447
Latent Semantic Indexing	403
Eigenvectors	403

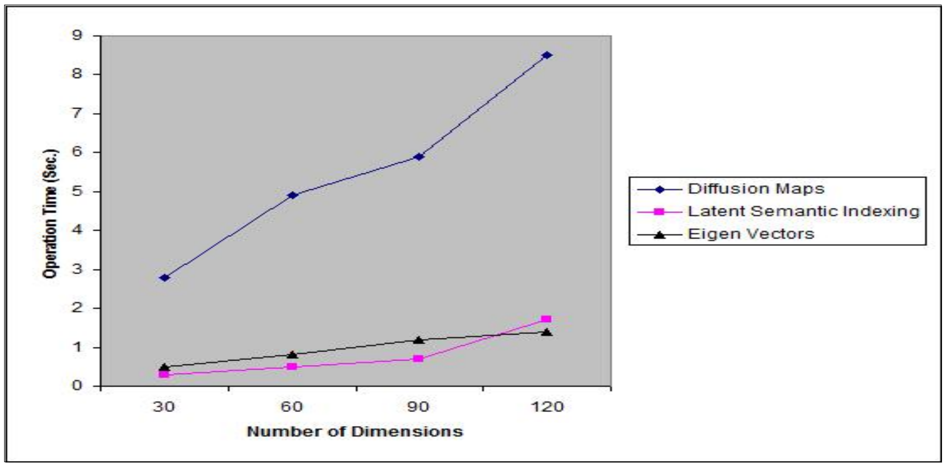


Figure 3.2: Operation Time for Diff, LSI, and Eigenvectors.

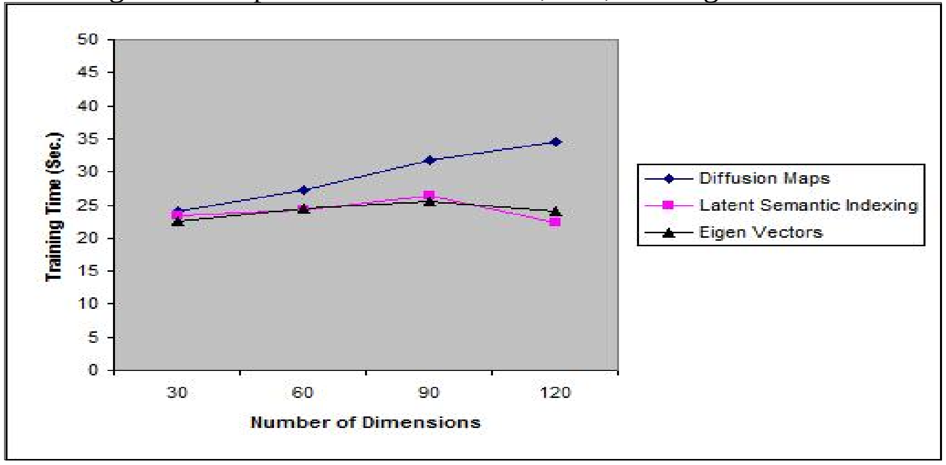


Figure 3.3: Training Time for Diff, LSI, and Eigenvectors.

Therefore, diffusion maps with 60 dimensions seems to represent the best trade-off between time and accuracy.

Later, we compared diffusion maps with 60 dimensions with three additional similarity methods. We used TF.IDF with cosine similarity, SoftTFIDF with cosine similarity, and a modification of diffusion maps, where the pairwise cosine similarity is applied to the diffusion maps reduced representations of the long string values in both the training dataset and the testing long string value. Other short string methods, such as edit distance and Jaccard similarity, were not used in this context because of their high cost and low accuracy with long string values. Fig. 3.5 and Fig. 3.6 represent the results. Fig. 3.5 depicts the F_1 measurement for the classifier. In all the experiments on Pubmed dataset, an SVM classifier was used. Fig. 3.6 represents the classification time and the preprocessing time, which includes the data preprocessing, operation, and training steps. Such steps are done once only. The term cosine similarity in both figures refers to TF.IDF with cosine similarity.

According to these two figures, diffusion maps with 60 dimensions showed the best classification time, a comparable preprocessing time, with a small loss in the F_1 measurement as compared to the remaining methods.

Experiments on the IMDB dataset showed similar trends. Diffusion maps with 120 dimensions were selected. Due to the huge number of classes used in the IMDB dataset, a bagging classifier, which is more frequently used with such cases, is used. Out of memory error occurred with the TF.IDF with cosine similarity method. Diffusion maps with 120 dimensions showed the best classification time and a comparable preprocessing time and F_1 measurement. Fig. 3.7 and Fig. 3.8 represent the IMDB comparison results.

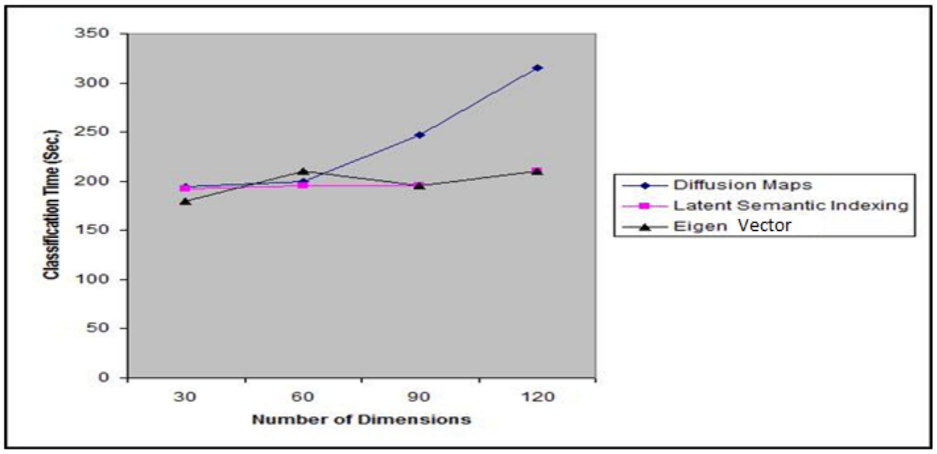


Figure 3.4: Classification Running Time for Diff, LSI, and Eigenvectors.

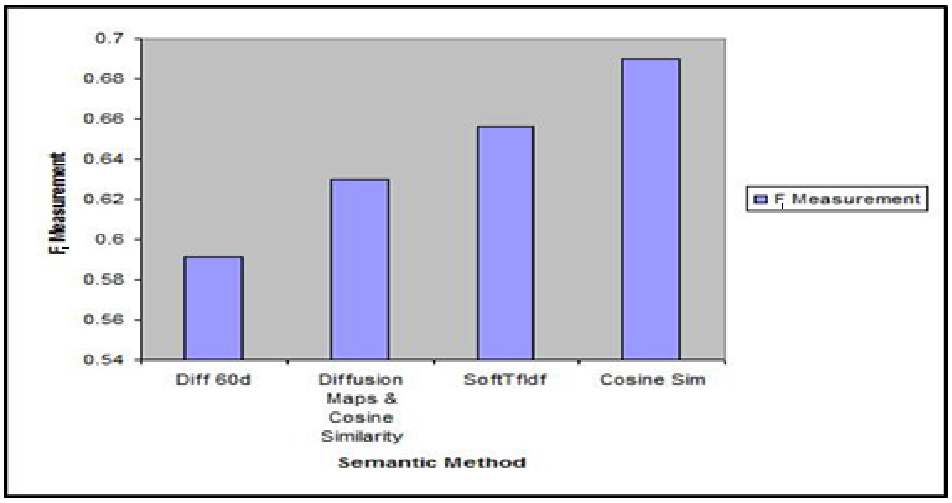


Figure 3.5: F_1 Measurement for four candidate semantic methods for Pubmed dataset.

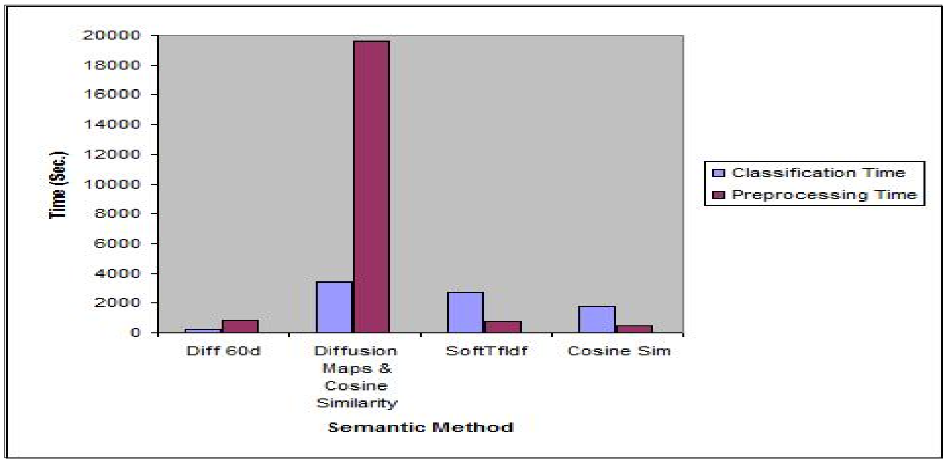


Figure 3.6: Preprocessing and Classification Time for four candidate semantic methods for Pubmed dataset.

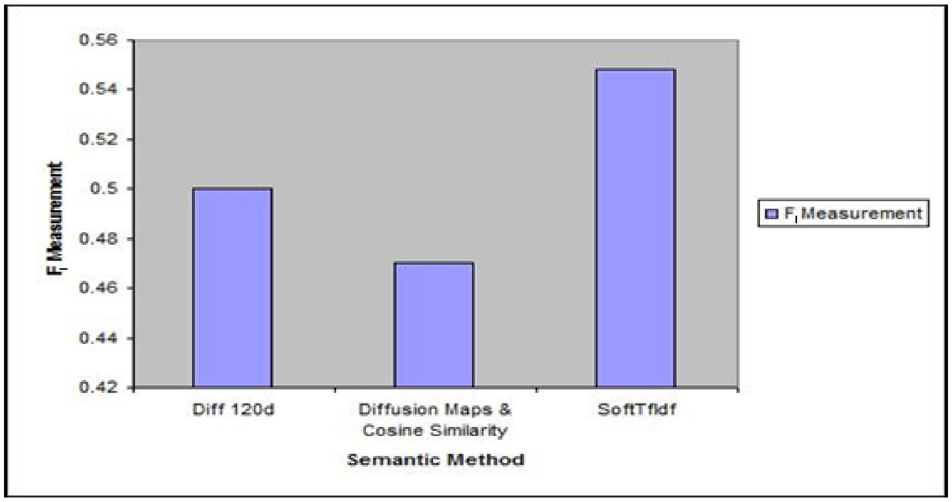


Figure 3.7: F_1 Measurement for three candidate semantic methods for IMDB Dataset. Out of Memory error occurred with the TF.IDF method.

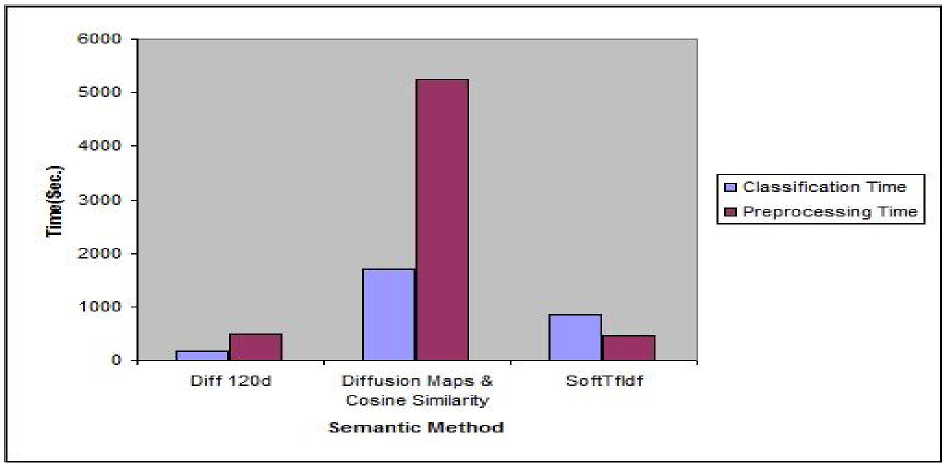


Figure 3.8: Preprocessing and Classification Time for three candidate semantic methods for IMDB Dataset. Out of Memory error occurred with the TF.IDF method.

According to the previous experiments, both Diffusion Maps and SoftTFIDF showed the best overall performance among the candidate semantic methods under study. In order to find the best method, more experiments were conducted with larger dataset of 10000 Abstracts from Pubmed. We got Out of Memory error when using SoftTFIDF with 10000

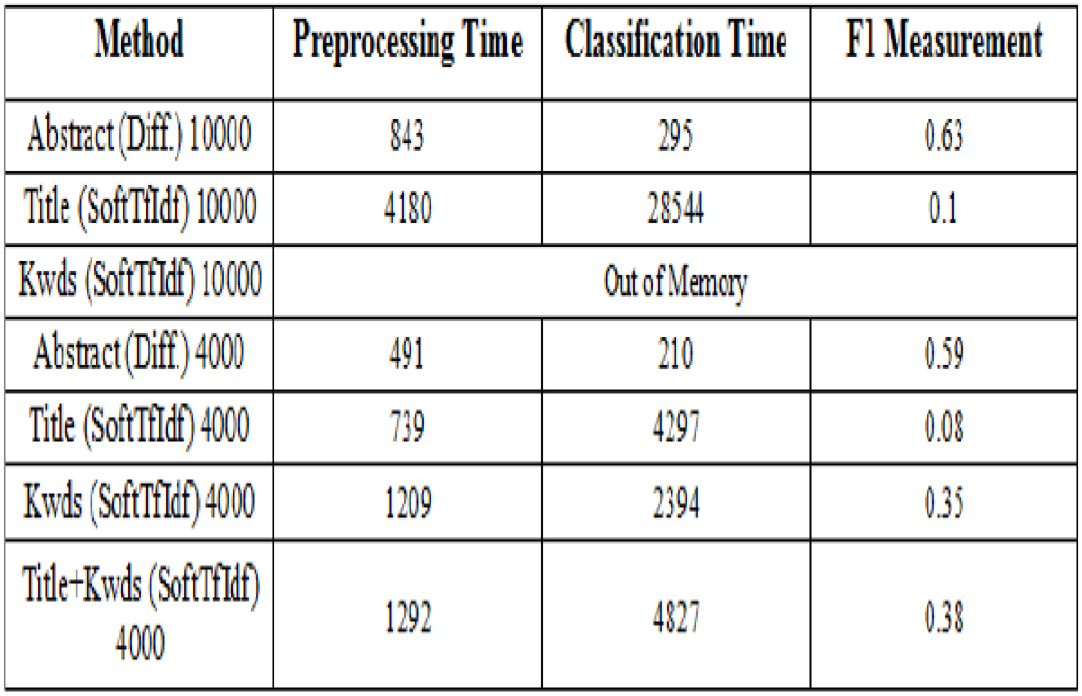
Abstracts, while we got results when applying diffusion maps with the same number of Abstracts. This showed that Diffusion Maps is the best candidate method for semantically joining attributes containing huge number of long string values.

3.4 Long string Vs Short string Evaluation

In this phase, we compared Diffusion Maps Method on the Abstract Field with the SoftTFIDF short string method with the Title and Keywords attributes. We used 10000 records in our evaluation from the Pubmed dataset. Table 3.4 represents the comparison results. Obviously, using diffusion maps method with the Abstract attribute outperformed the use of SoftTFIDF method on the Title and Keywords attributes, isolated or combined. This is reasonable because the information provided by the Abstract attribute is much more than that in both the Title and Keywords attributes. It is expected that the preprocessing time for the Abstract attribute will be longer than for the Title and the Keywords attributes, but this will be done once only. Regarding the running time, diffusion maps was the fastest due to the reduced representations for long string values. Besides, the use of a classifier provides a solution to frequently changing databases, and a sufficient number of training values is all what is needed.

Furthermore, our algorithm is able to deal with a huge number of records and large dimensionality. The number of dimensions for every testing record is the number of selected features using Chi-square when applied on the training long string values. This number will be reduced more using the diffusion maps operation. Accordingly, this algorithm will suffer less from the curse of dimensionality issue.

Table 3.4: Performance of Long and Short String Methods



Method	Preprocessing Time	Classification Time	F1 Measurement
Abstract (Diff.) 10000	843	295	0.63
Title (SoftTfIdf) 10000	4180	28544	0.1
Kwds (SoftTfIdf) 10000	Out of Memory		
Abstract (Diff.) 4000	491	210	0.59
Title (SoftTfIdf) 4000	739	4297	0.08
Kwds (SoftTfIdf) 4000	1209	2394	0.35
Title+Kwds (SoftTfIdf) 4000	1292	4827	0.38

3.5 Summary

In this chapter, we compared multiple semantic methods to find the best similarity measurement for long string values under supervised learning. The diffusion maps method showed a superior accuracy and a comparable overall preprocessing and running time. Furthermore, we also proposed a semantic similarity join method using long attributes under supervised learning, and we compared the performance of this method for joining long string values with the performance of other existing short string methods for joining short string values, and the results showed a significant difference in favor of our proposed method.

CHAPTER 4

A PRIVACY PRESERVING SEMANTIC SIMILARITY JOIN USING LONG ATTRIBUTES UNDER SIMILARITY THRESHOLDS

4.1 Introduction

In some cases, one or more sources may refuse partially or totally to share its whole data with other sources during the similarity join process, and only a few researchers have concentrated on performing similarity join under privacy constraints.

Examples of such works includes [29], which introduced a protocol to perform similarity join using phonetic encodings, [30], which proposed a privacy preserving record matching protocol on both data and schema levels, [31], which concentrated on the e-health applications and its intrinsic private nature, and [32], which used a Term Frequency – Inverse Document Frequency (TF.IDF) based comparison function and a secure blocking schema. Other methods concentrated on using encryption to preserve privacy such as [33][34].

To our knowledge, the existing protocols were proposed to perform similarity join under privacy constraints when the join attribute is a short attribute.

Again, long string values contain much more information than short string values, and we showed chapter 3 that using long string values can improve the similarity join semantic accuracy under supervised learning[35]. Adding to that, most databases include attributes of long string values. However, the previously stated protocols use measurements that are not suitable for such long values. Moreover, our previous work concentrated on using machine learning methods, and such methods are not always applicable. Here, we use similarity thresholds to decide matched records, which are much simpler and of comparable efficiency if used carefully. Finally, the previous methods concentrated on the syntax representations of the string values without considering the underlying semantics. It is worthwhile to find an efficient semantic protocol for joining long string values under privacy constraints when similarity thresholds are used.

As part of our solution, we compare diffusion maps [1], latent semantic indexing [2], and locality preserving projection [36]. These methods have strong theoretical foundations and have proved their superiority in many applications. Therefore, we compare their performance as candidate semantic similarity join methods for joining long attributes using similarity thresholds. It should be noted that the existing protocols are not included in this comparison because of their high running time cost and low accuracy when applied to long string values. For example, [29][32][33][34] used methods that do not consider the semantic similarities among the string values. While [30] introduced the use of embedded vectors for mapping, their embedding method was applicable to short string attributes. In order to evaluate the performance of our suggested methods, we use two datasets, Amazon Products dataset [37] and IMDB Movies dataset [17]. We use various similarity threshold values to define the matched records and evaluate the protocol.

The contributions of this work are as follows:

- Proposing an efficient privacy preserving protocol to perform similarity join when the join attribute is a long attribute under privacy constraints, which can improve the privacy preserving similarity join accuracy.
- Finding an existing method that can be used efficiently for joining values of long attributes under privacy constraints when similarity thresholds are used.

- Considering the semantic similarities among the string values during the privacy preserving similarity join process.
- Our protocol can assist the existing protocols, which are used mainly with short attributes, to improve the overall privacy preserving similarity join performance.

The rest of this chapter is organized as follows. Section 4.2 describes the candidate semantic methods to be compared with respect to joining long string values when similarity thresholds are used. Section 4.3 describes our privacy preserving protocol for semantic similarity join. Section 4.4 represents the experimental part where we compare the previous candidate semantic methods and study the performance of our protocol upon using the best performing methods from the previous comparison. Section 4.5 is the summary.

4.2 Semantic Methods for Joining Long Attributes Under Similarity Thresholds

In the following subsections, we describe the candidate semantic methods for joining long string values when similarity thresholds are used. Some methods were already described in section 2 of chapter 3.

4.2.1 Diffusion Maps

Diffusion maps is a dimensionality reduction method proposed by Lafon [1]. It was previously described in chapter 3. Initially, a weighted graph is constructed whose nodes are labeled with long string values and whose edge labels correspond to the similarity between the corresponding node values. A similarity function called the kernel function, *W,* is used for this purpose. The first-order neighborhood structure of the graph is constructed using a Markov matrix *P*. In order to find similarities among non-adjacent nodes, forward running in time of a random walk is used. A Markov chain is computed for this purpose by raising the

Markov matrix *P* to various integer powers. For instance, according to P_t, the t^{th} power of *P*, the similarity between two long string values *x* and *y* represents the probability of a random walk from *x* to *y* in *t* time steps. Finally, Single Value Decomposition (SVD) dimensionality reduction function is used to find the eigenvectors and the corresponding eigenvalues of $P_{t,t \geq 1.}$ The approximate pairwise long string value similarities are computed using the significant eigenvectors only. The similarity between any two long string values using such a method is called *diffusion maps similarity*. The mathematical details of diffusion maps are already given in section 3.2. For more information, refer to [1].

4.2.2 Latent Semantic Indexing (LSI)

As previously described in chapter 3, LSI [2] uses the Singular Value Decomposition operation to decompose the term by long string value matrix *M*, which contains terms (words) as rows and long string values as columns, into three matrices: *T*, a term by dimension matrix, *S*, a singular value matrix, and *D*, a long string value by dimension matrix. The original matrix can be obtained through matrix multiplication of TSD^T. In order to reduce the dimensionality, the three matrices are truncated to *z* user selected reduced dimensions. Dimensionality reduction reduces noise and reveals the latent semantics in the dataset. When the components are truncated to *z* dimensions, a reduced representation matrix, M_z is formed according to equation 3.10. Refer to [2] for a detailed explanation of this method.

4.2.3 Locality Preserving Projection

Locality preserving projection [36] is described briefly as follows. Given a set of long string values represented in the vector space $x_1, x_2, x_3, \ldots, x_n$ in R_m, where *m* represents the terms. This method finds a transformation matrix *A* that maps these long values into y_1, y_2,

y_3, …, y_n in a new reduced space R_l, $l<m$, such that $y_i = A^T x_i$. This method is particularly applicable when x_1, x_2, x_3, …, $x_n \in O$, where O is a nonlinear manifold embedded in R_m. Refer to [36] for a detailed explanation of this method.

4.3 Privacy Preserving Protocol for Semantic Similarity Join Using Long Attributes Under Similarity Thresholds

In this section, our proposed protocol is described. As stated before, this protocol is efficient in joining tables using their long string attributes. Up to our knowledge, no protocols were proposed to be used with such long attributes, and as proved in [35], using such attributes provides a better semantic join accuracy than using short attributes.

In the algorithm, we have two parties A and B, each of which has a relation, R_a and R_b respectively. First, the two parties share the similarity threshold value T that will be used later to decide similar pairs. Next, each party generates the term by long string value matrix from its long attribute, such that each row represents a term (word) and each column represents a long string value. The result is M_a and M_b for A and B respectively. For example, if A contains 1000 paper abstract values in its Paper Abstract attribute, each row in M_a represents a term, and each column represents an abstract. Later, the TF.IDF weighting is applied to both matrices. TF.IDF weighting is commonly used in information retrieval. TF.IDF weighting of a term w in a long string value x is given in equation 3.12.

Upon applying TF.IDF, both $WeightedM_a$ and $WeightedM_b$ are generated. Every row in this matrix represents a term, every column represents a long string value, and every entry represents the weight of the term in that long string value.

In the next step, both parties share the MeanTF.IDF threshold value [38] to be used and apply the MeanTF.IDF unsupervised feature selection method to both $WeightedM_a$ and

WeightedM_b. This method assigns a numerical value for each term in both *WeightedM*$_a$ and *WeightedM*$_b$. The value of a term *w* is calculated as follows:

$$\text{Val}(w) = \frac{\sum_{x=1}^{N} TF.IDF(w,x)}{N}, \tag{4.1}$$

where TF.IDF(*w*,*x*) is the TF.IDF weight of the term *w* in the long string value *x*, and *N* represents the number of long string values in the relation. The value of each term represents its importance. The terms with the highest values are the most important terms. It should be noted that terms and features are used alternatively in this context and have the same meaning.

The features from *A* with the highest values are concatenated with randomly generated features by *A* and are sent to a third party, *C*. *B* does the same. Later, *C* finds the intersection and returns those shared features, *SF*, that exist in both parties. Both parties remove their randomly generated features from *SF* and generate new matrices, *SF*$_a$ and *SF*$_b$, where each row represents an important term from *SF*, each column represents a long string value, and each entry represents the TF.IDF weighting. Later, every party adds random records to its corresponding matrix to hide its origional data. It should be noted that in this step, every record, including the randomly generated ones, is assigned a random index number. The generated matrices, *Rand_Weighted_a* and *Rand_Weighted_b* are sorted according to their index number to guarantee that the randomly generated records are randomly distributed in both matrices. Next, both matrices are sent to *C*. *C* performs the semantic operation on both matrices to produce *Red_Rand_Weighted_a* and *Red_Rand_Weighted_b*.These matrices have the concept terms as rows and the long string values as columns. In the experiments section of the paper, we will compare different candidate semantic methods when various thresholds are used, and the best method will be used here. Also, we will study the effect of adding random records on the semantic operation performance in the experimental part. The protocol continues by finding the cosine similarities for all the pairs (*x*,*y*), $x \in$ *Red_Rand_Weighted_a* and $y \in$ *Red_Rand_Weighted_b*, and if the cosine similarity is greater than a threshold *T*, the pair of the two vectors is considered a match and inserted into *Matched*. *Matched* is returned to both *A* and *B* to delete the pairs that include randomly generated records. Finally, both parties can share their Matched list after deleting the random records.

One issue with the protocol is having a randomly generated feature in the returned *SF*. This could occur when the two parties generate randomly the same feature or when one party generates a random feature that matches an important feature in the other party. In order to calculate the probability of such scenarios, we assume that the randomly generated strings have length up to *k* characters. For a specific length *s*, the number of generated strings is s^{26} for English alphabet. Therefore, the probability of generating a string that matches with an existing feature is

$$P = \frac{1}{\sum_{j=1}^{k} (26^{j})}, \tag{4.2}$$

and the probability of generating the same random feature by both parties is P^2.

For example, if we generate lengths up to 5 characters, the probability of the first scenario will be around 10^{-19} and the probability of the second one is 10^{-38}, which are very unlikely. Furthermore, these cases will not affect the running of the algorithm, but will make SF_a and SF_b different in the number of rows (features). However, adding a few features to one matrix will not affect significantly the results because we use the main eigenvectors and eigenvalues in the semantic methods.

4.4 Experiments

In order to evaluate the previous methods on long string values, two datasets were used, Amazon Products Dataset and the IMDB Movies Dataset. Table 4.1 below describes the use of these datasets in the experimental part. The following is a brief description of each dataset.

4.4.1 Amazon Products

We collected 700 records from Amazon website via http://amazon.com. In this work, we are interested in the product descriptions, which provide detailed information about the products. The product descriptions were divided into categories, such as computers, perfumes, cars, and so on. All product descriptions that belong to the same category are considered similar. The total number of categories in the collected dataset is 13 categories. The categories of the collected descriptions were of various complexities.

4.4.2 Internet Movies Database (IMDB)

We used 1000 records from the IMDB Movies database. For more details, please refer to section 3.3.2.

For our experiments, we used an Intel® Xeon® server with 3.16GHz CPU and 2GB RAM, with Microsoft Windows Server 2003 Operating System. Also, we used Microsoft Visual Studio 6.0 to read the datasets, Matlab 2008a for the implementations of the candidate semantic methods. For diffusion maps, we used Lafon implementation[1]. Regarding LSI, we used the Matlab svds( ) operation, and for locality preserving projection, we used implementation provided in [39].

Algorithm 4.1: SECURE PROTOCOL FOR SEMANTIC SIMILARITY JOIN USING LONG ATTRIBUTES UNDER SIMILARITY THRESHOLDS

Input: Two parties ***A*** and ***B***, each has a long attribute ***X***.

Output: Set of matched records sent to both ***A*** and ***B***.

Algorithm:

(1) Both ***A*** and ***B*** share the similarity threshold ***T*** to decide matched pairs.

(2) ***A*** and ***B*** generate their term by long string value matrices M_a and M_b from $R_a.X$ and $R_b.X$.

(3) TF.IDF weighting is calculated from M_a and M_b to generate $WeightedM_a$ and $WeightedM_b$.

(4) Both ***A*** and ***B*** share the MeanTF.IDF threshold value to perform MeanTF.IDF unsupervised feature selection.

(5) Both ***A*** and ***B*** return their selected features along with some randomly generated features to a third party ***C***.

(6) ***C*** finds the shared features in both parties, ***SF***, and returns them to both ***A*** and ***B***.

(7) ***A*** and ***B*** generate reduced weighted matrices SF_a and SF_b from $WeightedM_a$ and $WeightedM_b$ using ***SF*** after removing the randomly generated features.

(8) ***A*** generates random records, each of which has SF entries and add them randomly to SF_a. ***B*** does similarily.

(9) Every origional and random record in both SF_a and SF_b is assigned a random index number, and both parties keep track of the index numbers that belong to the randomly generated records.

(10) Both SF_a and SF_b are sorted according to the index number to generate ***Rand_Weighted_a*** and ***Rand_Weighted_b***, which are sent later to ***C***.

(11) **C** performs the semantic operation to generate ***Red_Rand_Weighted_a*** *a*nd ***Red_Rand_Weighted_b***.

(12) **C** finds the pairwise cosine similarities among the generated two matrices.

(13) If the cosine similarity for a pair is greater than the predefined threshold ***T***, this pair is inserted into ***Matched***.

(14) ***C*** returns ***Matched*** to both ***A*** and ***B***.

(15) Both ***A*** and ***B*** delete from ***Matched*** the randomly generated records.

Table 4.1: Datasets Description

Dataset	**Used Number of Records**	**Number of Categories**
Amazon Products	700	13
IMDB	1000	10

In order to evaluate the performance, we used F_1 measurement, *preprocessing time*, *operation time*, and *matching time*. Please refer to section 3.3 for more details regarding F_1 measurement.

- Preprocessing time is the time needed to read the dataset and generate matrices that could be used later as an input to the semantic operation.
- Operation time is the time needed to apply the semantic method.
- Matching time is the time required by the third party, C, to find the cosine similarity among the records provided by both A and B in the reduced space and compare the similarities with the predefined similarity threshold.

In phase one, we want to find the best semantic candidate method to be used with long string values when similarity thresholds are used. We compared diffusion maps, latent semantic indexing, and locality preserving projection. As every method is a dimensionality reduction method, we used the optimal number of dimensions for each method that maximizes the F_1 measurement. Fig. 4.1 shows an example of selecting the optimal number of dimensions for diffusion maps experimentally. In that Figure, we found the F_1 measurement for various numbers of dimensions ranging from 5 to 25. We used a fixed similarity threshold value in this case. Obviously, the maximum F_1 measurement was obtained using ten dimensions. The optimal number of dimensions for the remaining methods was calculated similarly. The best number of dimensions for LSI was eight, while it was five for LPP. Fig. 4.2 depicts the comparison of the three methods using various similarity thresholds on IMDB dataset. According to the Figure, both LSI and Diffusion Maps worked efficiently, with advantage given to LSI. The maximum F_1 measurement for LSI was 0.81, with threshold 0.7, while the maximum F_1 measurement for Diffusion Maps was 0.71, with threshold 0.5. Locality Preserving Projection showed the worse performance due to its linear nature.

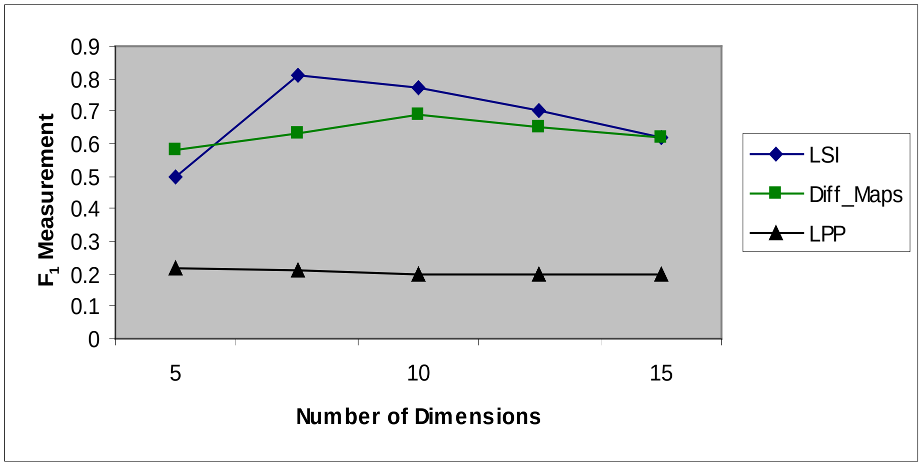


Figure 4.1: Finding best number of dimensions for diffusion maps, LSI, and LPP experimentally. IMDB dataset was used. The best numbers of dimensions were ten, eight, and five dimensions respectively, which resulted in the highest F_1 Measurements.

For Amazon Products dataset, Fig. 4.3 displays the results. Clearly, diffusion maps and LSI outperformed LPP. The performance of LSI dropped significantly in this dataset in comparison with diffusion maps. We concluded from phase one that both diffusion maps and

LSI showed efficient performance in joining long string values with advantage given to diffusion maps due to its stable performance.

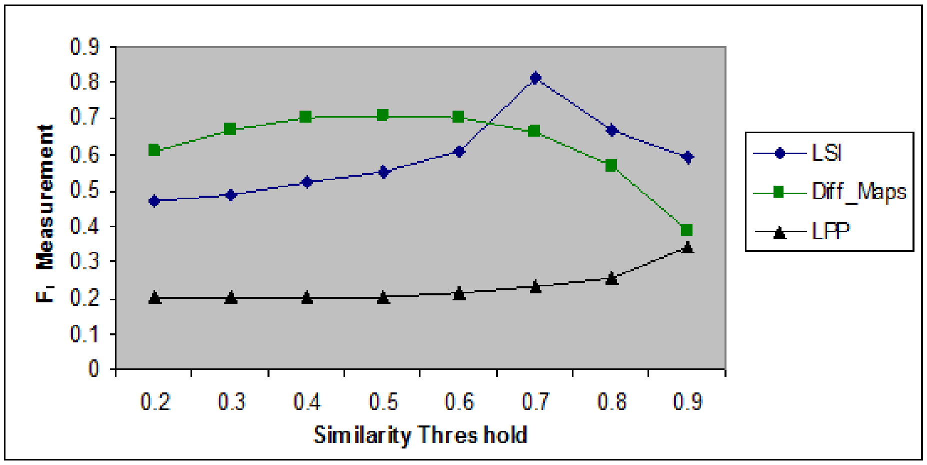


Figure 4.2: Comparing LSI, diffusion maps, and locality preserving projection to find the best semantic method for long attributes. IMDB dataset was used. Both LSI and diffusion maps showed the best performance.

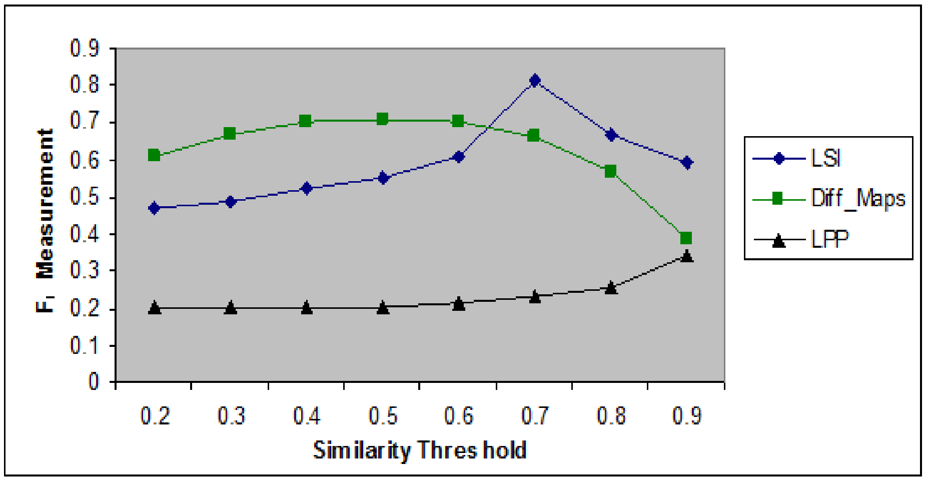


Figure 4.3: Comparing LSI, diffusion maps, and locality preserving projection to find the best semantic method for long attributes. Amazon Products dataset was used. Diffusion maps showed the best performance.

In phase two of the experimental part, we used diffusion maps and LSI, as they showed the best performance in phase one. We used them separately with our protocol and studied the protocol performance. We used both datasets in this phase. The evaluation measurements used here are preprocessing time, operation time, and matching time. It should be noted that the F_1 measurement for both methods was studied in phase one, where both methods showed efficient performance with advantage given to diffusion maps.

Regarding the preprocessing time, it took 12 seconds to read 1000 records from IMDB, while it took one second to find TF.IDF weighting, and 0.5 second to apply MeanTF.IDF. Time to find shared features by A and B was negligible (approximately zero). For Amazon Products dataset, similar trends were found.

For operation time, Fig. 4.4 represents the results for LSI and diffusion maps with various dimensions in both IMDB and Amazon Products datasets. Obviously, the time needed to perform LSI is less than that in Diffusion Maps. The difference increases with the increase in the number of dimensions. For Amazon Products dataset, similar trends were found.

It is worthwhile to mention that this operation is done once only, and therefore, does not highly affect the protocol performance. Also, it is not necessary to have large number of dimensions for diffusion maps to get the optimal performance. The optimal number of dimensions for diffusion maps in IMDB dataset was ten, while it was five for Amazon Products dataset.

Regarding the matching time, and due to the small number of dimensions used to represent each long string value, this time was negligible, even with the Cartesian product comparison of 5000 records. For Amazon Products dataset, similar trends were found.

Moreover, we studied the effect of adding random records, as stated in steps 8-10 in the algorithm, on the performance of the semantic operation, which is done in step 11. We added various portions of random records that are dataset size dependant, and we found their effect on both the F_1 measurement and the number of suggested matches. Regarding F_1 measurement, the performance increased slightly when small portion of the random records were added, then it started to decrease. This is due to the mechanism of the semantic operation itself. In diffusion maps, the important eigenvalues and eigenvectors are extracted from the dataset. The more random records are inserted, the more their effect on the real eigenvectors and eigenvalues. At some point, the algorithm will extract eigenvector(s) and eigenvalue(s) that represent the random records, which will decrease the accuracy significantly. Fig. 4.5 depicts this step. Regarding the number of suggested matches, trivially, increasing the number of records by adding random records will increase the number of candidate pairs, which in turn will increase the suggested matches. Adding random records will increase the number of candidate pairs to be compared, which will increase the number of suggested matches. Adding more random records will consume more time and place more overhead. Fig. 4.6 illustrates this step. Overall, we conclude that adding random records which compose 10% of the whole data size will hide the real data, without much effect on both the semantic operation accuracy and running overhead.

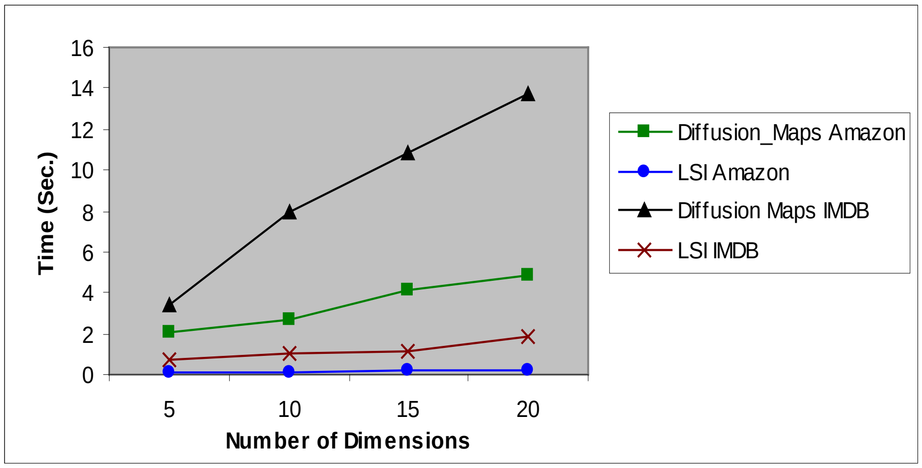


Figure 4.4: Operation Time for Diffusion Maps and LSI with various number of dimensions. Both IMDB dataset and Amazon Products dataset were used. Operation time for LSI was less than that in diffusion maps.

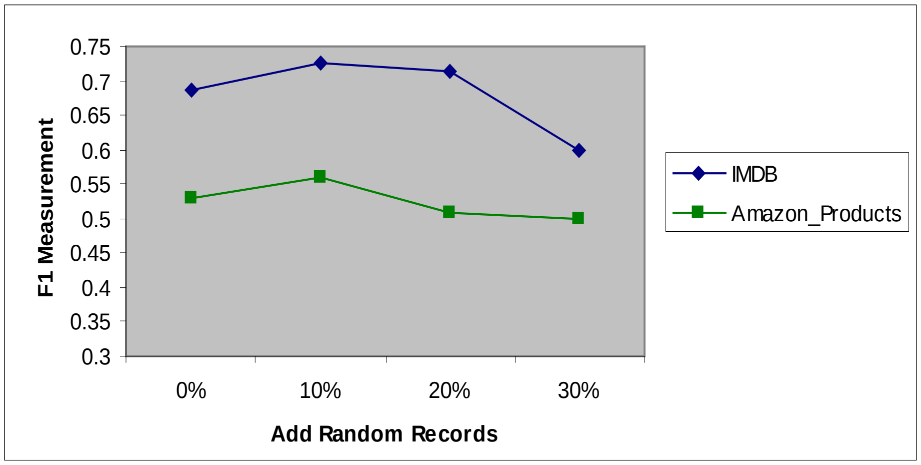


Figure 4.5: The effect of adding random records on the F_1 measurment upon using diffusion maps. F_1 measurement decreased rapily when the inserted random records size exceedes 10% of the dataset size.

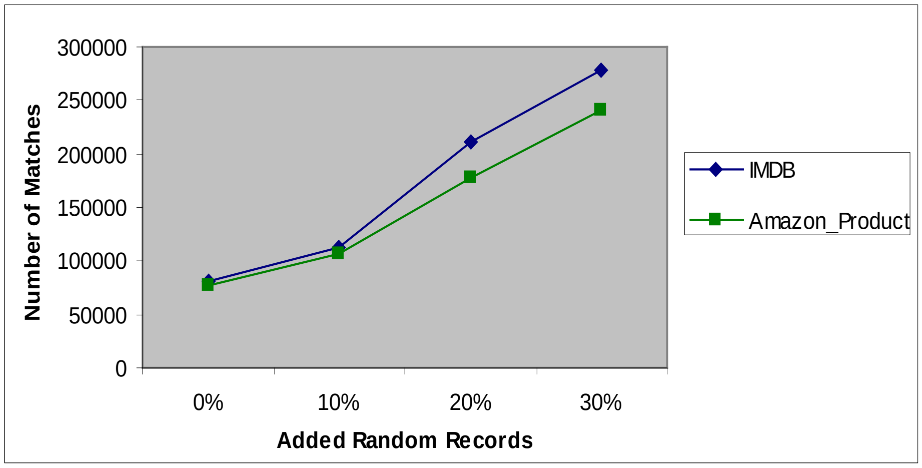


Figure 4.6: The effect of adding random records on the number of suggested matches upon using diffusion maps. Adding more reocrds inroduced more overhead by increasing the number of suggested matched records.

4.5 Summary

In this chapter, we proposed an efficient privacy preserving similarity join protocol for long string join attributes under similarity thresholds. We showed that diffusion maps method provides the best performance, when compared with other semantic similarity methods for long strings under similarity thresholds. Both mapping into the diffusion map space and adding a small portion of randomly generated records can hide the original data without affecting accuracy.

CHAPTER 5

PRIVACY PRESERVING SIMILARITY JOIN METHOD USING LONG ATTRIBUTES UNDER SUPERVISED LEARNING

5.1 Introduction

In our previous work in chapter 4[83], we proposed a privacy preserving protocol for similarity join under similarity thresholds, while in chapter 3[35], we proposed using long string attributes as join attributes to improve the semantic similarity join performance using supervised learning, when a training set exists. However, we did not consider the privacy issue under supervised learning. As shown in chapter 3, using supervised learning, if applicable, would improve the similarity join performance significantly. However, if privacy constraints exist, and up to our knowledge, no work has proposed including the supervised learning in the privacy preserving protocol. It is worthwhile to propose a privacy preserving similarity join protocol under supervised learning to benefit from the accuracy improvement when privacy constraints exist. In order to evaluate this protocol, Pubmed dataset [27] was used. The contribution of this work is as follows.

- Proposing an efficient privacy preserving similarity join protocol under supervised learning and improving the its performance using both a training set and long attributes.
- Comparing the effect of using multi-label supervised learning against single-label supervised learning on the proposed protocol.

The rest of this chapter is organized as follows. Section 5.2 briefly describes Pubmed Dataset. Section 5.3 explains our privacy preserving similarity join protocol under supervised learning. Section 5.4 studies the effect of using multi-label supervised learning on the protocol performance, and section 5.5 is the summary.

5.2 Pubmed Dataset

This dataset includes indexed bibliographic medical citations and abstracts. It is collected by the U.S. National Library of Medicine (NLM). It includes references from more than 4500 journals. The total number of categories is 23 classes proposed by [27]. Appendix A lists a description of the 23 classes. In our experiments, we used subsets of various numbers of records and numbers of categories. For more information, please refer to section 3.3.1.

It should be noted that dividing the dataset into parts to simulate the data of different sources would not make any difference from using the single undivided dataset, and this is because of the diffusion maps kernel, which requires grouping the records from all sources to find the pairwise distance among them. This would produce the same result as using a single undivided dataset. Besides, in the supervised learning context, there is no need to divide the data later because it will serve as a single training set for the testing records from all the sources.

5.3 Privacy Preserving Semantic Similarity Join Protocol Using Long Attributes Under Supervised Learning

From our previous work in chapter 3[35], it is clear that Diffusion Maps is one of the best methods to be used with long attributes using supervised learning methods. However, we need to modify the method to provide more privacy. Initially, such a method provides a level of privacy by mapping the data into the diffusion maps space. In order to increase the privacy level, random records are to be added by every source before the sharing process in order to hide the original data. However, using pure random records could be inefficient as it is easy to detect. Our faked record should be as close to an original looking record as possible, in order to make it harder to be detected. Moreover, having pure random records in the training set and assigning them to random labels would affect the classifier learning model and decrease the classification F_1 measurement when a testing set is used. Therefore, the added records need to be carefully selected to provide a privacy level and to protect the classification accuracy from being decreased. Our selection method is described next.

In order to generate each random record in some source, the source needs to pick a record from its original records randomly and change each value randomly. Epsilon is used as an upper limit to the change in each value. The equation to generate a random record vector of n values from an existing record is the following.

$$\text{Random\_Vector}(i) = \text{Existing\_Vector}(i) +/- \text{Epsilon}, \tag{5.1}$$

where i=1: n, Epsilon is a user defined value representing the maximum value change, the sign is selected to be positive or negative randomly, and Existing_Vector is the randomly selected original record. It should be noted that a different existing original record is selected as a seed for each new Random_Vector.

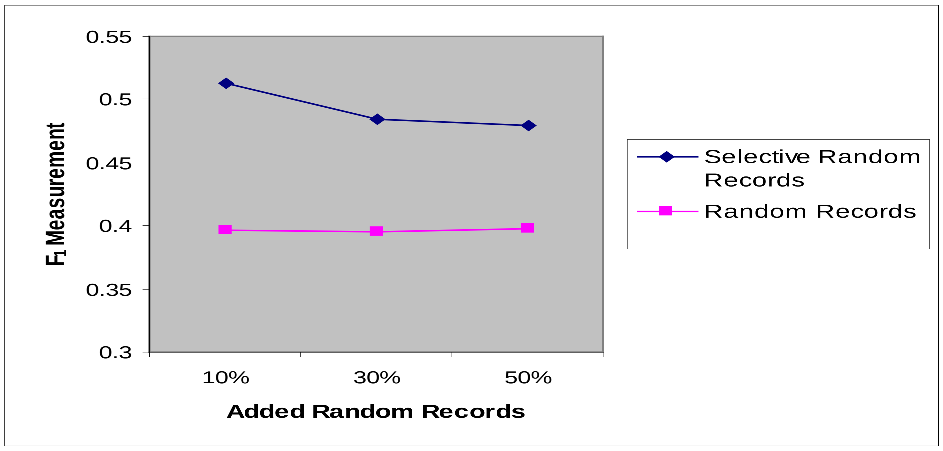


Figure 5.1: Comparing selective random records with random records. Clearly, using selective random records achieved more F_1 measurement.

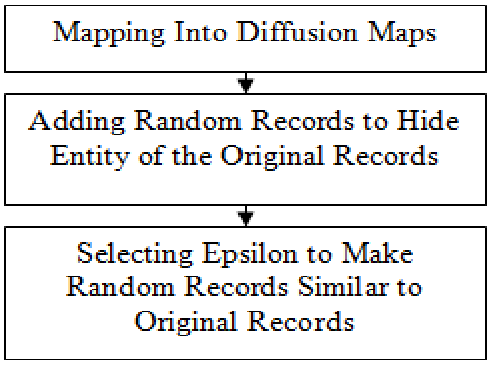

Mapping Into Diffusion Maps
Adding Random Records to Hide Entity of the Original Records
Selecting Epsilon to Make Random Records Similar to Original Records

Figure 5.2: The Privacy Layers of our Supervised Protocol.

In order to study the effect of adding selective random records instead of pure random records, we used both methods with various noise portions ranging from 10% to 50% added to the training set, and we used the SVM classifier later to classify a testing set using that training set. As displayed in Fig. 5.1, using selective random records preserved the classifier accuracy, in contrast with the pure random records, which decreased the classification accuracy significantly. Therefore, we adopted the use of selective random records in the following experiments.

Regarding the effect of adding selective random records instead of pure random records on privacy, and as explained before, using such selective random records would make it harder to be detected and distinguished from the original records, and this improves the level of privacy accordingly. Fig. 5.2 depicts the privacy layers of our privacy preserving supervised protocol. In the top layer, mapping into diffusion maps space provides the first level of privacy. Next, adding random records to the original records from each source would provide other level of privacy by hiding the entities of the original records. Finally, processing the random records to make them selective random records using the epsilon value provides the third level of privacy. It should be noted that our privacy preserving unsupervised protocol contains the top two layers only because it uses pure random records. Using selective random records in our privacy preserving unsupervised protocol would improve its privacy, and studying the effect of selective random records on that protocol is left to the future work. The Privacy preserving Similarity Join Protocol for Long Attributes Using Supervised Learning is given in Protocol 5.1 and is explained as follows.

We have two sources *A* and *B*, each of which has a relation, R_a and R_b respectively. First, each source generates the term by long string value matrix from its long attribute *X*, such that each row represents a term (word), each column represents a long string value, and each cell value, which is the intersection of row *i* and column *j*, represents the frequency of term *i* in the long string value *j*. The result is M_a and M_b for *A* and *B* respectively. For example, if *A* contains 1000 Disease Descriptions in its Disease Description attribute *X*, each row in M_a represents a term, each column represents a disease, and each cell value represents the frequency of the term in the disease description. Later, the TF.IDF weighting is applied to both matrices. TF.IDF weighting was already described in Equation 3.12.

Upon applying TF.IDF, both *WeightedM$_a$* and *WeightedM$_b$* are generated. As in protocol 4.1, every row in this matrix represents a term, every column represents a long string value, and every entry represents the weight of the term in that long string value.

In the next step, both sources share the Chi-square threshold value [19] to be used and apply this supervised feature selection method to both *WeightedM$_a$* and *WeightedM$_b$*, as used in our previous work [35]. This method assigns a numerical value for each term in both *WeightedM$_a$* and *WeightedM$_b$*. The value of a term *w* is calculated as follows:

$$\text{Val(w)} = \sqrt{\frac{(n_{pt+} + n_{nt+} + n_{pt-} + n_{nt-})(n_{pt+}n_{nt-} - n_{pt-}n_{nt+})^2}{(n_{pt+} + n_{pt\_})(n_{nt+} + n_{nt-})(n_{pt+} + n_{nt+})(n_{pt-} + n_{nt\_})}} \ , \qquad (5.2)$$

Where $\mathbf{n_{pt+}}$ and $\mathbf{n_{nt+}}$ are the number of documents in the positive category and the negative category respectively in which term **w** appears at least once. The positive and negative categories are used to find the accuracy measurements per class when multiple classes are used such that the positive category indicates a class and the negative category indicates the remaining classes. $\mathbf{n_{pt-}}$ and $\mathbf{n_{nt-}}$ are the number of documents in the positive category and the negative category respectively in which the term ***w*** doesn't occur. The value of each term represents its importance. The terms with the highest values are the most important terms.

The features from *A* with the highest values are concatenated with randomly generated features by *A* and are sent to a third source, *C*. *B* does the same. Later, *C* finds the intersection and returns those shared features, *SF*, that exist in both sources. Both sources remove their randomly generated features from *SF* and generate new matrices, *SF$_a$* and *SF$_b$*, where each row represents an important term from *SF*, each column represents a long string value, and each entry represents the TF.IDF weighting. Later, every source adds selective random records to its corresponding matrix to hide its original data. The records are generated using

Equation 5.1 as described previously. Again, every record, including the randomly generated ones, is assigned a random index number. The generated matrices, *Rand_Weighted_a* and *Rand_Weighted_b* are sorted according to their index number to guarantee that the randomly generated records are randomly distributed in both matrices. Next, both matrices are sent to *C*. *C* performs the semantic method, which is the Diffusion Maps as suggested in chapter 3 [35] for joining long string values using supervised learning. Applying the semantic method on both A and B produces *Red_Rand_Weighted_a* and *Red_Rand_Weighted_b*. These matrices have the concept terms as rows and the long string values as columns. In our experiments, we used Diffusion Maps based semantic join as described in our previous work. Later in this section, we will conduct more experiments to study the effect of adding selective random records and changing epsilon value on the semantic operation performance. The protocol continues by training a classifier using all the pairs (x,y), $x \in$ *Red_Rand_Weighted_a* and $y \in$ *Red_Rand_Weighted_b*. Again, one major difference between this protocol and Protocol 4.1 is that every long string value in the attribute *X* in Protocol 5.1 has a label that refers to its category, in contrast with Protocol 1 that manipulates unlabeled long string values. Upon training the classifier, *A* sends its testing records, along with some random records, to *C* for classification. C classifies the records and returns their predictions. *B* sends its testing records similarly. After excluding the random records, both *A* and *B* shares the labels of their original records, and the original records belonging to the intersected labels are shared between the two sources.

In this work, we used the Pubmed medical dataset to evaluate the protocol performance. Besides, we used an Intel® Xeon® server of 3.16GHz CPU and 2GB RAM, with Microsoft Windows Server 2003 Operating System. Also, we used Microsoft Visual Studio 6.0 to read the datasets, Matlab 2008a for the implementation of the Diffusion Maps, and Weka 3.6.2 for the SVM classifier to get the method's performance.

In order to evaluate the performance of Diffusion Maps as a semantic method for joining sources using their long attributes under privacy constraints, we used the same performance measurements used in chapter 3.3[35], which are F_1 measurement, operation time, *classifier training time*, and *classifier testing time*.

Initially, we labeled a subset of 816 records manually, and used them as a small labeled dataset, which includes 17 disease classes. Besides, every record was allowed to have single label only. In order to find the best diffusion maps reduced number of dimensions, we used various dimensions and we calculated the corresponding F_1 measurement. No noise was added in this phase, as this was done in the single source level, where no privacy was needed. We used Weka SVM Classifier and 10-Fold Cross Validation in order to get the F_1 measurement. The optimal number of dimensions in our experiments was eighty, as the F_1 measurement tends to be stable after this value. Fig. 5.3 depicts the results.

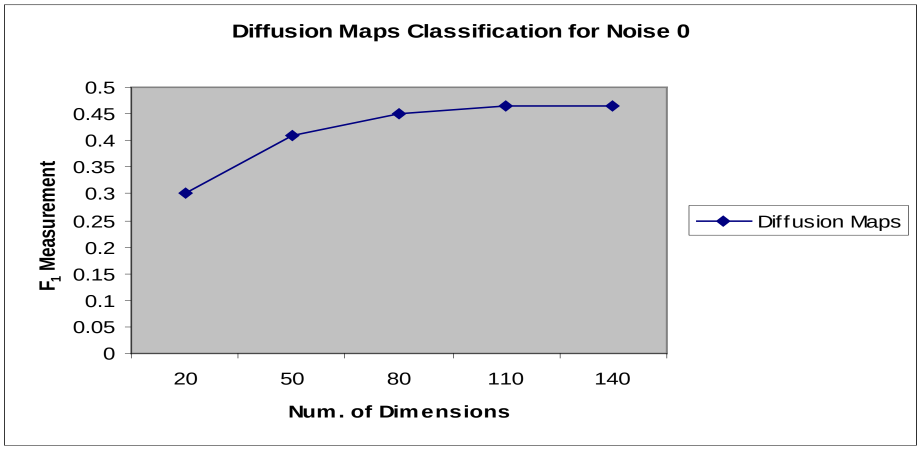


Figure 5.3: Selecting the optimal number of diffusion maps reduced dimensions. SVM with 10-Fold cross validation was used on a subset of Pubmed containing 816 records. Eighty dimensions were selected as the performance becomes stable after that number.

Protocol 5.1: SECURE SUPERVISED PROTOCOL FOR SEMANTIC SIMILARITY JOIN USING LONG ATTRIBUTES

Input: Two sources ***A*** and ***B***, each has a long attribute ***X***.

A small training set ***TR*** that has labeled long string values.

Output: A Set of matched records sent to both ***A*** and ***B***.

Protocol:

(1) ***A*** and ***B*** generate their term by long string value matrices M_a and M_b from TR_a and TR_b.

(2) TF.IDF weighting is calculated from M_a and M_b to generate $WeightedM_a$ and $WeightedM_b$.

(3) Both ***A*** and ***B*** share the Chi-square supervised feature selection threshold, and each source performs Chi-square on its own terms.

(4) Both ***A*** and ***B*** return their selected features along with some randomly generated features to a third source ***C***.

(5) ***C*** finds the shared features in both sources, ***SF***, and returns them to both ***A*** and ***B***.

(6) ***A*** and ***B*** generate reduced weighted matrices SF_a and SF_b from $WeightedM_a$ and $WeightedM_b$ using ***SF*** after removing the randomly generated features.

(7) ***A*** generates selective random records, each of which has **SF** entries using Equation 5.1 and adds them randomly to SF_a. ***B*** does similarly.

(8) Every original and random record in both SF_a and SF_b Is assigned a random index number, and both sources keep track of the index numbers that belong to the selective random generated records.

(9) Both SF_a and SF_b are sorted according to the index number to generate ***Rand_Weighted_a*** and ***Rand_Weighted_b***, which are sent later to ***C***.

(10) ***C*** performs the semantic operation to generate of ***Red_Rand_Weighted_a*** and ***Red_Rand_Weighted_b.***

(11) ***C*** trains a classifier on the training set which is composed of ***Red_Rand_Weighted_a*** and ***Red_Rand_Weighted_b.***

(12) Both ***A*** and ***B*** sends their ***X*** long values, after converting them to a suitable form as discussed in [26] , into ***C*** for classification. Random records are sent also to ***C*** to hide the original entities.

(13) ***C*** classifies the records of ***A*** and returns the labels back. ***C*** classifies the records of ***B*** similarly.

(14) ***A*** deletes the random records and extract the labels of the original records in ***X***. ***B*** does similarly.

(15) ***A*** and ***B*** send their labels to ***C,*** and ***C*** returns the shared labels to both ***A*** and ***B***.

(16) ***A*** and ***B*** share the original records that belongs to the shared labels.

Regarding the preprocessing time, it took three seconds to read the 816 training records from Pubmed, while it took negligible time to find TF.IDF weighting, and 204 second to apply Chi-square. Time to find shared features by A and B was negligible (approximately zero). For operation time, diffusion maps required three seconds, while for training time, it took 12 seconds to train the SVM classifier using 80 dimensions and 10 folds cross validation.

In the next step, we studied the effect of adding noise to the classification performance. We added various noise percentages from 10% to 30% to the training set, and we used various epsilon values from 1 to 100 to generate the random records. The SVM classifier was used to classify 4000 testing records using that modified training set. Figure 5.4 summarizes the findings.

Obviously, increasing the noise percentage can provide more privacy but it would decrease the F_1 measurement. This is reasonable because it is affecting the SVM training model. Likewise, increasing the epsilon value (up to a maximum limit) would improve the privacy and decrease the F_1 measurement. However, a large increase in epsilon value would have negative effect on the privacy, because the records will be easy to detect and excluded as faked. Selecting the noise percentage and epsilon value is domain dependant and depends on the application privacy requirements versus the join accuracy.

Protocol5.2: SECURE PROTOCOL FOR SEMANTIC SIMILARITY JOIN USING LONG ATTRIBUTES FOR SUPERVISED LEARNING.

Input: A new test case arriving a source.

Output: Classifying this test case to the up to date knowledge and joining it according to semantic similarity.

Protocol:

(1) The training model is sent to both A and B.

(2) For every new test case arriving any source, the training model is used to classify it.

(3) This test case is joined to the shared records of its same label.

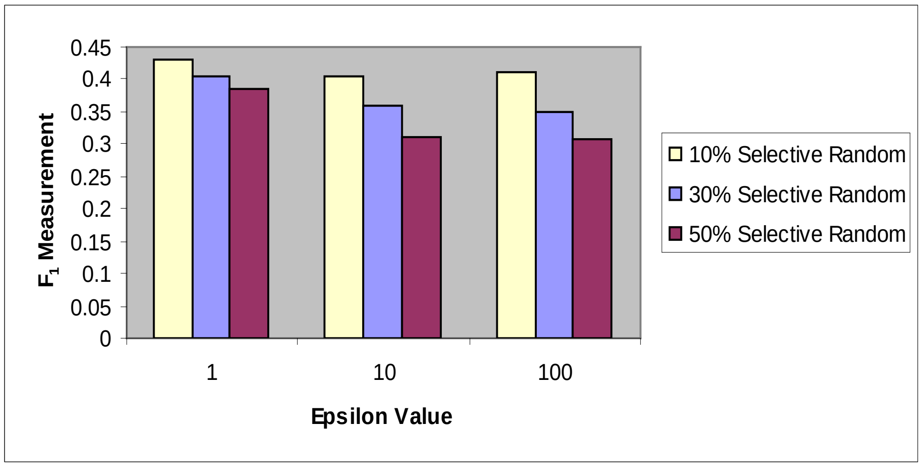


Figure 5.4: The effect of adding selective random records and changing Epsilon value on the F_1 measurment upon using diffusion maps. F_1 measurement decreased as the selective random records portion and epsilon value increase.

Besides, it is worthwhile mentioning that it took two seconds to read the 4000 testing records and negligible time to find the TF.IDF weighting. For the classifier testing time, it took two seconds to classify those testing records using the previous training set of 816 records.

Protocol 5.2 is used after Protocol 5.1 in order to join every new testing record arriving at any source. It is worth mentioning that Protocol 5.1 is used once, while Protocol 5.2 is used with every testing record.

5.4 Privacy Preserving Semantic Similarity Join Protocol Using Long Attributes Under Multi-Label Supervised Learning

One limitation of the previous supervised solution is its constraint on the number of labels per record. So far, we forced every record in the training and testing sets to have one label. However, this is not always correct. In many real life applications, a record can belong to various entities and refer to multiple labels simultaneously. For example, a disease could be a virus disease (Label 1) and affect infants only (Label 2). Using multi-label classification would provide a model which is closer to real-life applications.

Again, to our knowledge, no work has been done to benefit from multi-label classification techniques in the privacy preserving supervised protocol for similarity join. Therefore, we studied the performance of various multi label classifiers for privacy preserving semantic similarity join. We compared RBF Networks, SVM, and kNN multi label classifiers. We used a subset of the Pubmed dataset consisting of an 800 records training set, with 10% selective random records to be added later, and a 3000 records testing set. Each record is allowed to have up to four labels. We used k = 3, and we used the polynomial kernel SMO for SVM. Finally, we used the SVM single label classification results as a baseline. Fig. 5.5 depicts the results. Clearly, using multi label classification outperformed single label classification in terms of F_1 measurement. This is reasonable because the ideal performance of any single label classifier will not exceed $\frac{1}{N}$ of its corresponding multi label classifier, where N is the maximum allowed number of labels for each record.

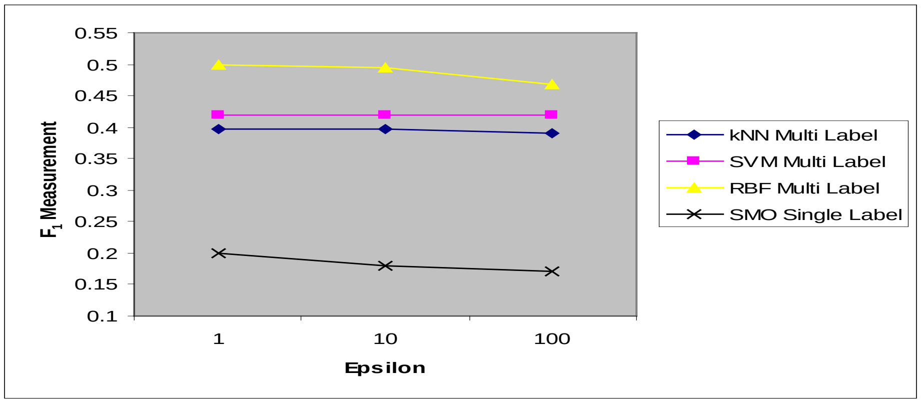


Figure 5.5: Comparing various multi-label classifiers with a single label classifier using various Epsilon values. Multi-label classification significantly outperformed single-label classification, and RBF Network classifier has the best F_1 measurement. 10% Noise was used.

Besides, the RBF Network classifier outperformed both SVM and kNN classifiers in this dataset. This is due to its non linear nature, in contrast with the polynomial SVM and lazy kNN. The Pubmed dataset, due to its overlapped topics and some noisy records, needs a non linear classifier to produce the best classification accuracy. One more advantage of an RBF Network classifier is that it is not highly affected by the parameter optimization step. An SMO classifier has an RBF non-linear kernel option, which could be comparable to that of an RBF Networks classifier; however, it performs poorly without the parameter optimization step.

In order to study the effect of adding selective random records and changing epsilon value on the multi-label classification accuracy, we used the RBF Network classifier with various epsilon values ranging from 1 to 100, and various portions of the selective random records ranging from 10% to 50%. Fig. 5.6 illustrates the findings. We noted that both increasing the portion of the added selective random records and increasing the epsilon value decreased the classifier F_1 measurement. However, as we discussed previously, we do not need to add a large portion of the records nor largely increase the epsilon value. Adding a small portion with a small epsilon value would provide an adequate level of privacy without affecting the F_1 measurement. Furthermore, the added portion of random records and the epsilon value are domain dependant, and depend on the domain error tolerance and the required level of privacy.

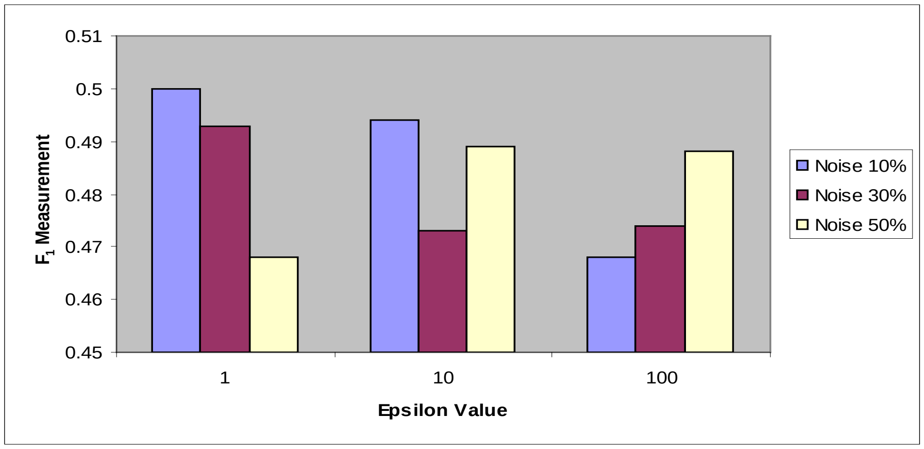


Figure 5.6: The effect of changing epsilon and and adding selective random records proportional to the dataset size on the multi-label classification. RBF classifier was used. Obviously, increasing the added selective random records and increasing epsilon decreases F_1 measurement.

5.5 Summary

In this chapter, we proposed a similarity join privacy preserving protocol using long attributes under supervised learning. We proposed an efficient privacy preserving protocol for long string join attributes that uses Diffusion Maps and selective random records, which are hard to detect and does not affect the classification accuracy. Moreover, we enhanced the performance by using the multi-label supervised learning, when every record can refer to multiple entities simultaneously.

CHAPTER 6

A SIMILARITY JOIN METHOD USING LONG ATTRIBUTES UNDER UNSUPERVISED LEARNING

6.1 Introduction

In many real-life cases, it is very expensive or even impossible to create a training set to assist the similarity join method. In this case, similarity join method could be done under unsupervised learning. Many methods have been proposed to solve unsupervised similarity join [55][56][57][58]. Up to our knowledge, all these solutions are used mainly with short attributes.

We showed in our work in chapter 3 [35] that using long attributes would improve the similarity join performance under supervised learning. Therefore, it is worthwhile to study the use of long attributes in unsupervised similarity join. Unfortunately, most of the proposed preprocessing methods are not suitable for long attributes. Our first objective is to compare the effect of using long attributes and short attributes on the unsupervised similarity join performance.

On the other hand, databases are intrinsically dynamic. Records are inserted, updated, and deleted frequently. This could change the number of clusters accordingly. Most of the previous work assumed the database static. Therefore, our second objective is to provide a similarity join method that is efficient with expandable databases.

Our work is divided into four phases. First, finding the best semantic method for joining long attributes under unsupervised learning. Second, comparing the effect of using long attributes against using short attributes in the similarity join performance under unsupervised learning. Third, providing and evaluating our similarity join unsupervised method. Fourth, providing a solution that is efficient with expandable databases. It should be noted that many short string preprocessing methods were not included in this comparison because of their high running time cost and low accuracy when applied to long string values. In phase one, we are comparing diffusion maps[1], latent semantic indexing[2], eigenvectors[3], and independent component analysis[76]. In phase two, we compared the best method from phase one with TF.IDF and SoftTF.IDF[13]. KMeans[77] was used to cluster the output of each method. In order to evaluate the performance, we used three datasets, Amazon Product Descriptions[37], IMDB Movies dataset[17] , and Pubmed[27]. The contributions of this work are as follows:

- Adopting the use of long attributes to replace or assist the short attributes to increase the similarity join preprocessing methods under unsupervised learning.
- Finding an efficient semantic preprocessing method that can be used for joining values of long attributes when no training set exists.
- Providing an efficient solution for expandable databases.

The rest of this chapter is organized as follows. Section 6.2 compares various semantic methods for joining long attributes under unsupervised learning. Section 6.3 compares the effect of using long attributes against short attributes on the similarity join performance. Section 6.4 explains our proposed similarity join method using long attributes under unsupervised learning. Section 6.5 introduces the expandable databases scenario and provides a solution for such an issue. Finally, section 6.6 is the summary.

6.2 Comparing Semantic Similarity Join Methods Using Long Attributes Under Unsupervised Learning

In this section, we are going to compare various semantic preprocessing methods using long string attributes. The best method will be used as part of our solution. We are comparing diffusion maps, latent semantic indexing, eigenvectors, and independent component analysis. For more details about these methods, refer to [1][2][3][76]. We use only dimensionality reduction methods as candidate semantic preprocessing methods because the clustering process is very sensitive to the number of dimensions. Using non-dimensionality reduction methods such as TF.IDF with cosine similarity as input to the clustering algorithm will increase significantly the clustering time. In order to evaluate the previous methods in joining long string values, two datasets are used, which are Amazon products and IMDB. For detailed descriptions of these two datasets, please refer to sections 4.4 and 3.3 respectively. It should be noted that the number of records in the datasets is irrelevant to the performance of the algorithms as records are processed sequentially.

For our experiments, we used an Intel® Xeon® server of 3.16GHz CPU and 2GB RAM, with Microsoft Windows Server 2003 Operating System. Also, we used Microsoft Visual Studio 6.0 to read the datasets, Matlab 2008a for the implementations of the candidate semantic methods and KMeans.

In this phase, for the movie summary attribute in IMDB Dataset, we removed the stopwords and converted the text into lowercase. The term long string value frequency matrix was generated. Later, TF.IDF[18] weighting matrix was computed. Later, we used mean TF.IDF unsupervised dimensionality reduction method[38] to eliminate insignificant words, and we selected the 2% of the features with the highest importance. The values in the Product Description attribute from Amazon Products datasets were processed similarily.

For Diffusion Maps, we used Lafon Matlab implementation[1]. We used the values of σ and α to be 10 and 1 respectively as used in [35]. Regarding LSI, we used the SVDs( ) Matlab built-in function. For the eigenvectors method, we used the Eigs( ) Matlab function. For ICA, we used FastICA package [78].

The performance measurements used for this phase were *Silhouette value, Purity, Clustering time, and Operation time.* They are defined as follows:

Silhouette Value for a point *x*, which is assigned to cluster *c* of *n* points, is a measurement of the assignment suitability for this point during the clustering process. It is calculated using the following formulas:

$$\text{Silh}(x) = 1 - \frac{a(i)}{b(i)}, \qquad \text{If } a(i) < b(i) \tag{6.1}$$

$$\text{Silh}(x) = \frac{b(i)}{a(i)} - 1, \qquad \text{Otherwise} \tag{6.2}$$

$$\text{Where } a(i) = \frac{\sum_{y \in c} dist(x, y)}{n}, \tag{6.3}$$

$$\text{and } b(i) = \min\left(\frac{\sum_{y \notin c} dist(x, y)}{n} \right). \tag{6.4}$$

Purity measures the overall clustering accuracy in correspondence with the actual cluster labels. Let $C = \{C_1, C_2, C_3, \ldots, C_k\}$ represents the set of clusters, and let $L = \{L_1, L_2, L_3, \ldots, L_m\}$ represents the set of labels (classes). Purity is calculated using the following formula:

$$\text{Purity}(C,L) = \frac{\sum_k \max_m (C_k \cap L_m)}{n}, \qquad (6.5)$$

Where n is the total number of points in the dataset.

Clustering Time is the time required to perform the clustering algorithm.

Operation Time is the time required to perform the dimensionality reduction operation on the dataset.

After using each of the semantic methods, the KMeans clustering algorithm was used to get the performance for each method. We used KMeans, which is an example of a partitional clustering method, because it outperformed both hierarchical and suffix tree clustering methods [79]. During the clustering process, we experimentally selected the optimal number of reduced dimensions and the optimal number of clusters for KMeans. In detail, we used a fixed initial value for the number of clusters and used KMeans with that value to cluster the output of the diffusion maps algorithm using various numbers of dimensions. After finding the optimal number of diffusion maps dimensions, we used it with KMeans clustering with various number of clusters. We used the highest silhouette value after clustering with KMeans to indicate the optimal number of diffusion maps dimensions and optimal number of clusters. Figure 6.1 displays this step. The other semantic preprocessing methods were manipulated similarly. Later, we used both clustering time and cluster purity to evaluate the accuracy of the resulting clusters. The comparison of the semantic preprocessing methods according to the clustering time for Amazon and IMDB showed no significant differences among the compared methods. This is because of the similarity in the output of these methods according to the number of reduced dimensions. Figure 6.2 and Figure 6.3 show the comparison of the four methods according to the purity in Amazon and IMDB respectively. Clearly, diffusion maps showed the best performance. The performance of the other methods could vary depending on the complexity of the dataset. This is clear in Figure 6.3 when the methods were applied to the IMDB dataset, which is relatively easy and contains disjoint clusters. According to that figure, the four methods showed high performance, and the performance decreased in LSI, ICA, and eigenvectors when using the Amazon dataset, which is more complex and contains overlapped clusters. diffusion maps proved to have the most stable performance. Table 6.1 shows the operation time The methods were ordered as follows:

LSI < EIG < Diff < ICA. This is due to the larger amount of information contained in the input matrix of the ICA and diffusion maps, which are document-by-document matrices, in contrast with the simple, relatively sparse input matrices to LSI and eigenvectors. diffusion maps operation time is not very slow, in contrast with ICA, and could be compensated with the gain in accuracy upon using this method. As diffusion maps showed the best performance, it was adopted in our solution.

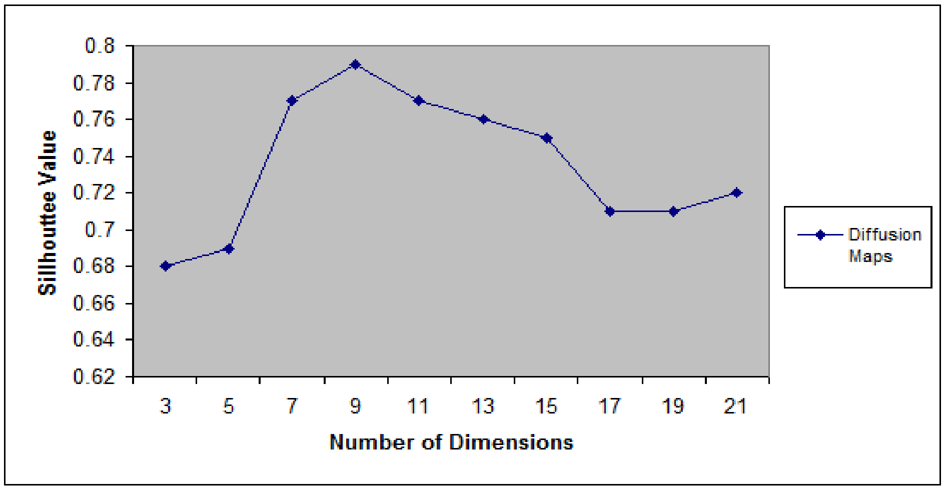


Figure 6.1: Determining the best number of clusters for KMeans under diffusion maps space. The best number of dimensions was nine dimensions. We used 700 product descriptions from Amazon Products dataset.

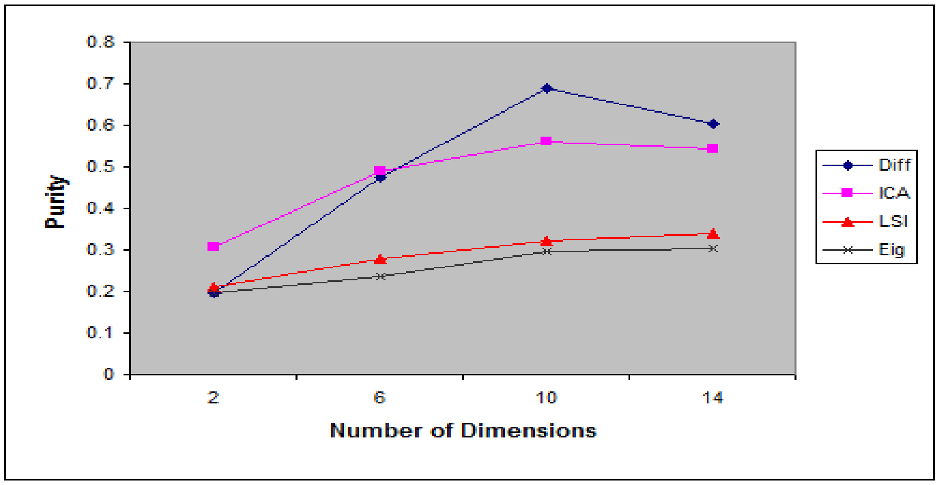


Figure 6.2. Comparing the purity of the KMeans clustering under diffusion Maps, ICA, LSI, and eigenvectors. Diffusion Maps showed the best performance. We used 700 product descriptions from Amazon Products dataset.

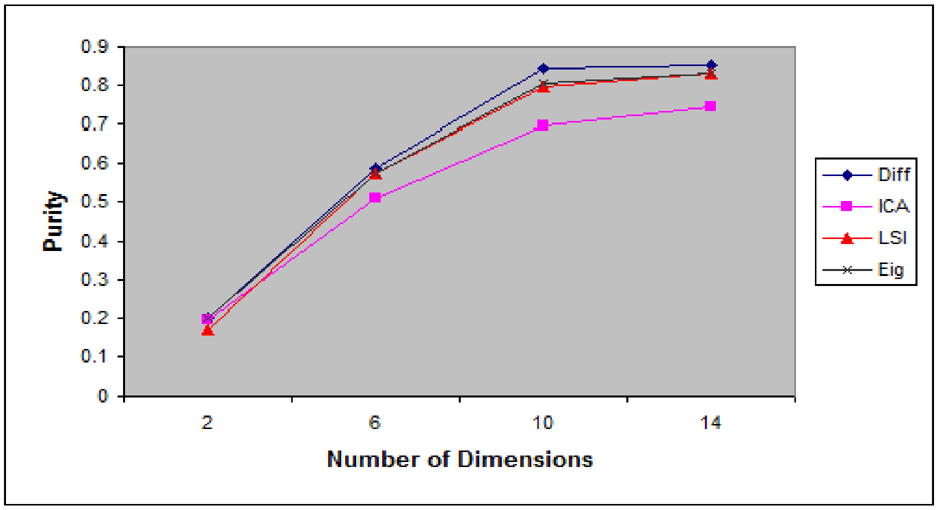


Figure 6.3. Comparing the purity of the KMeans clustering under diffusion Maps, ICA, LSI, and eigenvectors. Diffusion Maps showed the best performance. We used 1000 movie summaries from IMDB dataset.

Table 6.1. Operation Time (in seconds) for the Candidate Methods in the Two Datasets

Method	**IMDB**	**Amazon**
Diffusion Maps	2.5	1.35
LSI	0.24	0.1
ICA	10	3.6
Eigenvectors	0.45	0.23

6.3 Long String VS Short String Evaluation

For phase two, we compared the best semantic preprocessing method for long attributes with top existing preprocessing method for short attributes. According to phase one, diffusion maps proved to be the best semantic method, among the compared ones, for long attributes, when no training set exists. In this phase, clustering using long attributes represented in diffusion maps space was compared with clustering using short attributes represented using existing short methods. The performance measurements used in this phase were purity and clustering time. We used Product Title and Product Description attributes from Amazon products dataset to represent short attribute and long attribute respectively. We used 700 records for this purpose. For long attributes, we used KMeans to cluster the Product Description values that are represented in diffusion maps space. For short attributes, we used Product Title values that are represented using pairwise SoftTF.IDF [7] similarities, pairwise SoftTF.IDF similarities reduced using diffusion maps, pairwise TF.IDF similarities reduced using diffusion maps. KMeans was used to cluster the output of the three methods.

It should be noted that we did not use many existing unsupervised similarity join methods such as [56][57][58] because of their poor performance with long string values. We used two performance measurements, purity and clustering time. Table 2 depicts the results.

Table 6.2: KMeans Clustering Using Long and Short Attributes

Method	**Purity**	**ClusTime**
Prod Desc (Diff)	0.69	0.05
Prod Title (SoftTF.IDF)	0.405	1.2
Prod Title (SoftTF.IDF+ Diff)	0.41	0.08
Prod Title (TF.IDF + Diff)	0.51	0.1

Clearly, KMeans clustering of long string values represented by diffusion maps proved to have the best purity, which is reasonable because long attributes tend to have much more information than short attributes, which will increase the clustering accuracy. According to the clustering time, all the previous methods were comparable except the SoftTF.IDF alone. The reason is that this method is not a dimensionality reduction method, and the number of dimensions affects significantly the clustering time performance. Overall, we conclude that using diffusion maps semantic method with long attributes showed a better performance than using the existing unsupervised similarity join methods that use short attributes.

6.4 Similarity Join Method Using Long Attributes Under Unsupervised Learning

After showing that using long string attributes with diffusion maps and clustering the output using kMeans can provide an efficient performance in comparison with other unsupervised similarity join methods, we adopt this in our algorithms. In this section, we provide and discuss our unsupervised similarity join method, and evaluate its performance on new testing records. Basically, our method is composed of two algorithms, Algorithm 6.1 and Algorithm 6.2. Algorithm 6.1 takes as an input an initial set of unlabelled records and apply the similarity join operation on them using long attributes and diffusion maps. The output of this algorithm is a set of clusters, where every cluster represents a set of records that are joined according to their semantic similarity. Algorithm 6.2 takes as an input the set of clusters from Algorithm 6.1, optimizes it, and for every newly arriving testing record, it will apply the similarity join on it. In other words, it will assign it to one of the existing clusters. We explain the details of each algorithm next.

In Algorithm 6.1, the input is a dataset represented as a term document matrix, where each record represents a term (word) and every column represents a long string value. The output is a set of clusters, where every cluster represents a set of semantically similar items.

We assume here that record labels are not known. In the algorithm, after preprocessing the dataset by applying the TF.IDF weighting and reducing the dimensionality using the Mean TF.IDF unsupervised dimensionality reduction method, the diffusion maps method is applied to obtain the reduced representations of the long string values, *Y*, as stated in line 11. Every row in *Y* represents a long string value, and every column in *Y* represents a reduced dimension. Later, the KMeans algorithm is applied to cluster the long string values in the reduced space, and the silhouette value is calculated. We need to select the optimal values of both *Z* in line 11 and *Num_Clusters* in line 14 experimentally in order to maximize the silhouette value. After obtaining the optimal Z and *Num_Clusters*, Kmeans is applied using both values to output the optimal set of clusters. It should be noted that this algorithm is applied once only, and it is applied to any initial set of unlabelled records.

After obtaining the set of clusters using Algorithm 6.1, Algorithm 6.2 is used to assign every newly arriving record to its suitable cluster among the existing clusters. Algorithm 6.2 converts the arriving testing record into its reduced diffusion maps representation. Next, it finds the cosine similarity between the reduced testing record representation and all the cluster centroids. The testing record is assigned to the cluster whose centroid is the closest.

In the evaluation part, it should be noted that Algorithm 6.1 was already evaluated in the previous section and it outperformed the compared unsupervised similarity join methods. In order to evaluate the Algorithm 6.2, we inserted various numbers of records belonging to existing clusters, and we computed the similarity join accuracy, which represents the record-cluster assignment accuracy. Three datasets were used in this experiment, which are IMDB[17], Amazon Products[37], and Pubmed[27], and the results are illustrated in Table 6.3.

Clearly, the algorithm can assign the newly arriving records to the existing clusters with a high accuracy. It is obvious also that its accuracy is data-dependant. The algorithm works better with datasets of disjoined clusters, such as IMDB, than those of overlapped ones, such as Amazon Products.

Algorithm 6.1: DIFFUSION MAPS BASED SEMANTIC PREPROCESSING USING LONG STRING VALUES

Input: The term by long string value matrix *M* for the set of unlabeled *D* records

Output: Candidate similar records *Y_Clustered_Opt* represented as clusters.

Algorithm:

```
(01)  //process the dataset
(02)  M_weighted = find_TF.IDF _weighting(M)
(03)  M_Red = MeanTF.IDF(M_Weighted, R)  //R < T
(04)  For i=1:D
(05)    For j=1:D
(06)      Dcos(i,j) = 1-Cosine_Similarity($M\_Red_i$,$M\_Red_j$)
(07)    End;
(08)  End;
(09)  Fix Num_Clusters
(10)  For Z = Initial_Z : Final_Z
(11)      [Y,S,V,A] = Diffusion_Maps(Dcos, 10, 1, Z)
(12)      //|Y|= D x Z
(13)      //Cluster the reduced records
(14)      Y_Clustered=KMeans(Y, Num_Clusters)
(15)      New_Silh[ ] = Find_Silh(Y_Clustered)
(16)  End;
(17)  //Use Z_Opt that resulted in largest New_Silh[ ] value
(18)  [Y_Opt,S,V,A] = Diffusion_Maps(Dcos, 10, 1, Z_Opt)
(19)  For Num_Clusters = Initial_Clusters : Final_Clusters
(20)      Y_Clustered_Opt = KMeans(Y_Opt, Num_Clusters)
(21)      New_Silh[ ] = Find_Sillh(Y_Clustered_Opt)
(22)   End;
(23)   //Use Num_Clusters_Opt corresponding to largest
(24)  New_Silh[ ] value
(25)  Y_Clustered_Opt = KMeans(Y_Opt, Num_Clusters_Opt)
(26)  Return Y_Clustered_Opt
```

```
Protocol6.2:  SIMILARITY JOIN METHOD UNDER
              UNSUPERVISED LEARNING

Input:        A new testing record t arriving a source.

Output:       Join the testing record to one cluster

Protocol:

(1)  Convert the new test record t into the Diffusion Maps
     reduced representation t_red.

(2)  For c = 1: Num_Clusters

(3)  Cos_Sim[i] = Find_Cos_Sim(t_red, centroid[c])

(4)  End;

(5)  Add the testing record t to the cluster with max
     Cos_Sim[i].
```

Table 6.3: Algorithm 6.2 Accuracy on Three Datasets.

Method	**Avg. Accuracy**
IMDB	0.89
Pubmed	0.76
Amazon Products	0.73

6.5 Dynamically Expandable Semantic Similarity Join Protocol Using Long Attributes

The classification categories are not always static. Commonly, new categories could be created over time. Our protocol should have the ability to expand to include such new categories. There are many real life applications that need such expansion. Hereafter, we list two examples.

Example 6.1: New Diseases Detection

Recently, new diseases have been brought to the world’s attention. The ability to detect new diseases is crucial. The existing protocols should be able to detect when test cases that belong to new non existing labels are being introduced. Such ability can speed the detection process and minimizes its consequences. Moreover, the retraining process is also important to consider the newly added categories when classifying new test cases.

Example 6.2: Dividing Movie Classifications

In many cases, one starts with an initial number of categories, and later, one category is divided into two categories or more. For example, in the past, movie categories were limited. However, over time, each category started to contain many subcategories, and the differences among these subcategories have been increased. This process is a continuous process, and the existing protocols are supposed to detect when the category needs to be divided, and to retrain itself on the new subcategories. Protocol 6.3 represents the basic model for such an expandable supervised protocol.

In the following two subsections, we compare various methods to detect records of non-existing categories and study the effect of reclustering.

6.5.1 Detecting Records of Non-Existing Clusters

Here, our goal is to detect when records of new non-existing clusters are being introduced. In order to do this, we compare two detection measurement: Cosine Distance and Sillhouette value. These two measurements are computed in equations 3.2 and 6.1 respectively. These two measurements are computed for every arriving record. In the Cosine Distance, the maximum value of the detection measurement is returned, as there will be a value for each cluster . If the measurement is less than a predefined threshold, we consider the record belonging to a non-existing cluster.

Protocol6.3: Expandable Secure Protocol for Semantic Similarity Join using Long Attributes for Supervised Learning.

Input: A new test case arriving a source.

Output: Determine if there is a need to divide a category or introduce a new one.

Protocol:

(1) The source classifies the test case using the training model, as proposed in Protocol5.2, and join it to the shared records of its same label.

(2) The source updates a shared flag, which is used to detect the confidence of the assignment and the state of the category after the assignment.

(3) If the flag exceeds a defined threshold, divide that corresponding cluster into two clusters using a clustering protocol, change the labels assigned to the records in that divided cluster to reflect the new clusters.

(4) Retrain the Classifier using the new labels, and share the updated training model among the sources.

In order to compare the two measurements, we used both records of existing clusters and records of new clusters and computed their detection measurement values using both methods. Clearly, the efficient detection measurement is supposed to distinguish records of existing clusters and records of new clusters by showing a significant difference between their average measurement values. We used IMDB[17], PUBMED[27], and Amazon Products[37] datasets. It should be noted that in this scenario, the arriving records are processed sequentially, which makes the dataset size irrelevant to the performance. The results of using Cosine Distance and Silhoutte measurements are displayed in Table 6.4 and Table 6.5 respectively.

Table 6.4: Comparing Existing-Cluster records and New-Cluster records using Cosine Distance

	Avg. Cos Dist Existing-Cls	**Avg. Cos Dist New-Cls**	**Percentage Drop**
IMDB	0.95	0.77	19%
Pubmed	0.91	0.82	10%
Amazon	0.91	0.88	3%

Table 6.5: Comparing Existing-Cluster records and New-Cluster records using Silhouette measurement.

	Avg. Silh Existing-Cls	**Avg. Silh New-Cls**	**Percentage Drop**
IMDB	0.86	0.57	34%
Pubmed	0.81	0.7	14%
Amazon	0.77	0.67	13%

Apparently, using sillhoutte measurement resulted in a better isolation between both record types. Another observation is that the drop percentage when a new-cluster record is introduced is dataset dependent, as not all datasets have the same properties.

6.5.2 Reclustering Analysis

Reclustering is needed when the number of records belonging to a non-existing cluster becomes large. Reclustering would create a new cluster(s) to minimize the clustering error. When a criteria reaches a user-defined threshold, reclustering is applied. The criteria could be the number of records with detection measurement less than a specific value. For example, if the number of inserted records with silhoutte value less than 0.5 exceeds 50, reclustering is needed. Various domains could use various thresholds depending on their error tolerance. In order to find a suitable threshold value, we inserted a sample of records that belong to existing clusters, computed the silhouette measurement after each insertion, and found the minimum sillhoutte value. This value was used as the threshold value. In order to illustrate the motivation behind using a reclustering criteria, we conducted an experiment that calculates the percentage of records with a sillhoutte value less than the threshold. We used both types of records(existing-cluster and new-cluster) separately in two different groups. Two dataset were used here, IMDB and Pubmed. We denote the records that satisfied the reclustering criteria as Satisfying Records. Table 6.6 represents the results.

Table 6.6: Comparing Existing-Cluster records and New-Cluster records according to the percentage of satisfying records among them.

	% Records Satisfying Criteria in Existing Clusters	**% Records Satisfying Criteria in New Clusters**
IMDB	12%	69%
Pubmed	27%	48%

From Table 6.6, it is clear that records of non-existing clusters have lower sillhoutte values than those of existing clusters, and that using the minimum sillhouette value of the sample as a threshold value is promising.

Next, we studied the cost and effect of the reclustering process. Two methods were proposed here: labeling the new records manually, or using a clustering method to label them. Only the records that satisfied the reclustering criteria are labeled. Ideally, all the new-cluster records are supposed to satisfy the criteria and none of the existing-cluster records are supposed to satisfy it. From Table 6.6, we can see that around 55% of the new-cluster records satisfied the criteria, and 20% of the existing-cluster records did. Regarding the percentage of the new-cluster records, it needs to be representative to have accurate results. If none of the new-cluster records that belong to a cluster c satisfied the criteria, the cluster will not be represented. Commonly, the new-cluster records that satisfy the criteria are representative set of the new clusters. Regarding the existing-cluster records percentage, they would not affect the results as they would be eliminated during the labeling phase.

Obviously, using the manual labeling method would result in better accuracy and more execution time that using a clustering method for labeling. For comparison reasons, we estimated the manual labeling accuracy to be 0.9, and the time to be 3 minutes per a long string value(Assuming its average length is 75 words). After labeling the records, the feature selection method needs to be repeated to include the new cluster(s). Initially, the long string values are represented as a vector of the important terms in the existing clusters. In order to ensure a fair comparison, the important terms from the new cluster needs to be extracted and included in the representation of the long string values. In IMDB dataset, after inserting 300 new-cluster records of three new clusters into the origional dataset, which is composed of

1000 records of 10 classes, the feature selection method took 1013 seconds for 13 classes (In comparison to 809 seconds for the 10 basic classes), while In Pubmed dataset, after inserting 300 new-cluster records of three new clusters into the original dataset, which is composed of 200 records of 5 classes, the feature selection method took 34 seconds for 8 classes (In comparison to 17 seconds for the 5 basic classes). It should be noted that the feature selection method is affected mainly with the number of records, and this explains the difference in the running time in the two datasets. Finally, to study the effect of the reclustering process in the record-cluster accuracy for those records that belong to the newly created cluster, we inserted 490 records and the accuracy was 0.75, which is sufficient in many domains.

Regarding the second method, which uses clustering to assign the new records, we used the sillhoutte measurement with various numbers of new clusters. For each number of clusters, it took approximately 20 seconds to compute the silhoutte value. If we used 5 numbers, the process would take 100 seconds to label all the records, which is far less than the 180 seconds taken by human to label a single record. However, the decrease in time would cause a decrease in the labeling accuracy. The labeling accuracy for the clustering method when we used the silhouette value is dataset dependant, and is 0.65 correct on average. Therefore, we prefer to use the first method because of the error propagation problem that could occur in the second method due to incorrect clustering assignment.

Finally, as an estimation to the frequency of reclustering, we inserted random records from IMDB and Pubmed, and we used various similarity thresholds and various numbers of satisfied records. We recorded the order of that number of satisfying records among the random records. Table 6.7 and Table 6.8 represent the results for IMDB and Pubmed respectively.

Table 6.7: Reclustering Frequency using Various Thresholds and Numbers of Satisfying Records on IMDB

Threshold	**0.8**			**0.7**			**0.6**		
Number Satisfying Records	25	50	75	25	50	75	25	50	75
Order	50	112	161	105	194	328	169	400	-

Table 6.8: Reclustering Frequency using Various Thresholds and Numbers of Satisfying Records on Pubmed

Threshold	**0.8**			**0.7**			**0.6**		
Number Satisfying Records	25	50	75	25	50	75	25	50	75
Order	57	110	160	90	155	218	80	176	292

6.6 Summary

In this work, we proposed an efficient similarity join method using long attributes under unsupervised learning. This method can create initial set of semantically joined records, and can join newly arriving records to the suitable cluster according to its similarity. Furthermore, we proposed a model for similarity join under expandable databases. In this part, we compared some detection methods and studied both the reclustering process time and the effect of reclustering in the join performance of future testing records.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary

Similarity Join is grouping pairs of records whose similarity is greater than a threshold *T*. It has many applications in various fields. Although many works has studied Similarity Join with short string attributes, a few works have included the use of long string attributes to assist the similarity join process and enhance the performance. Obviously, long string attributes contain much more information than short string attributes. Therefore, using such attributes to detect similar records could improve the overall similarity join accuracy. Furthermore, long attributes exist in most of the databases, and finding an efficient method to perform similarity join using long attributes would complement the literature work that concentrates on short attributes.

7.2 Contributions

Our contributions are explained as follows.

- First, we proposed an efficient semantic similarity join method for joining tables according to their long attributes under supervised learning, when a training set exists. The training set has examples of similar record pairs, which would assist in detecting similar record pairs in the testing set. Such similarity join method for long attributes would assist or replace the existing short attribute similarity join methods. As part of this method, we found the best semantic similarity measurement for long string values.

- Second, we proposed a privacy preserving similarity join protocol for joining tables using their long attributes under similarity thresholds, when no training set is available. Basically, the sources involved in the similarity join process may not want to share their data, and may want to share the similar records only. In this case, the content of a source is supposed to be hidden and protected from being disclosed to other sources. A few works have been done in this area, and most of the work concentrated on methods that are applicable on short attributes only. As we explained in our first contribution, using long attributes in the similarity join can increase the similarity join accuracy. Up to our knowledge, no work proposed a privacy preserving similarity join method when the join attribute is a long attribute. Our proposed protocol showed its efficient performance for long attributes, which improved the overall similarity join accuracy under privacy constraints.
- Third, we proposed a privacy preserving similarity join protocol when the join attribute is long attribute using supervised learning, when a training set is available. Using a small training set can significantly improve the similarity join performance. Again, up to our knowledge, no work has been done to propose a privacy preserving similarity join protocol for long attributes under supervised learning, even though this would improve the similarity join accuracy when there are privacy constraints. Furthermore, we enhanced the performance by using selective records instead of random records. Moreover, we improved the similarity join performance by using mulit-label supervised learning, as the latter is closer to many real-life applications.
- Fourth, we proposed an efficient semantic similarity join method to be used with long attributes under unsupervised learning, when no training set exists. This scenario is

common in many practical applications, as it would be very expensive or even impossible to have a training set. Furthermore, we proposed a solution for scenarios that allow the number of groups (clusters) to expand by time. This case is also common because databases are not static, and their content is updated with every transaction. Up to our knowledge, no previous work proposed an efficient solution to similarity join method that considers database expansion.

7.3 Future Work Directions

Some future work directions are suggested as follows.

- We proposed a baseline model for similarity join with expandable databases and studied some reclustering detection methods and the effect of reclustering on performance. However, this area needs much more work to enhance both the detection method and the reclustering method.
- Using Diffusion Maps to reduce the dimensionality and extract the semantic relationships among long string values proved its efficiency. However, this method could pose overhead when the dataset is large. Even though this case is rare, as the training set needs not to be very large for the best join performance, but in some cases, especially when the number of record labels (or clusters) is large, the training set will be large. A future work could be done to find a scalable Diffusion Maps algorithm.
- In my work, I compared some dimensionality reduction methods according to their ability in joining long string values. I compared Diffusion Maps, Latent Semantic Indexing, Locality Preserving Projection, Independent Component Analysis, and Eigen Vectors. Diffusion Maps showed the best performance. In order to optimize the results, more dimensionality reduction methods need to be compared with Diffusion Maps.

- Semantic Similarity Join Under Privacy Constraints is a promising future work direction. Most of the existing works have concentrated on hiding the data itself. A clear example of this is encryption. However, in many cases, the shared information needs to be represented and joined together semantically. Semantic Encryption of Concepts would achieve such objective.

APPENDIX A

The 23 subcategories of MeSH category C ‘Diseases’

C01 Bacterial Infections and Mycoses

C02 Virus Diseases

C03 Parasitic Diseases

C04 Neoplasms

C05 Musculoskeletal Diseases

C06 Digestive System Diseases

C07 Stomatognathic Diseases

C08 Respiratory Tract Diseases

C09 Otorhinolaryngologic Diseases

C10 Nervous System Diseases

C11 Eye Diseases

C12 Urologic and Male Genital Diseases

C13 Female Genital Diseases and Pregnancy Complications

C14 Cardiovascular Diseases

C15 Hemic and Lymphatic Diseases

C16 Neonatal Diseases and Abnormalities

C17 Skin and Connective Tissue Diseases

C18 Nutritional and Metabolic Diseases

C19 Endocrine Diseases

C20 Immunologic Diseases

C21 Disorders of Environmental Origin

C22 Animal Diseases

C23 Pathological Conditions, Signs and Symptoms

BIBLIOGRAPHY

[1] R.R. Coifman and S. Lafon, “Diffusion Maps,” Applied and Computational Harmonic Analysis, vol. 12(1), pp. 5 - 30, 2006.

[2] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, “Indexing by Latent Semantic Analysis,” Journal of the American Society for Information Science, vol. 41(6), pp. 391-407, 1990.

[3] C.A. Kumar and S. Srinivas, “Latent Semantic Indexing Using Eigenvalue Analysis for Efficent Information Retreival,” Intl. Journal of Applied Mathmatics and Computer Science., vol. 16(4), pp. 551- 558, 2006.

[4] S. Sarawagi,, “Special Issue on Data Cleaning,” IEEE Data Eng. Bull., vol. 23(4), 2000.

[5] Y. M. Cheong, J. C. Tay, “Approximate String Matching for Multiple-Attribute, Large-Scale Customer Address Databases,” In Proc. of the Int. Conf. on Asian Digital Libraries (ICADL), pp. 168-172, 2003.

[6] McCallum , K. Nigam ,and L. Ungar ,”Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching,” In Proc. of the Intl. Conf on Knowledge Discovery and Data Mining (ACM SIGKDD), pp. 169-178, 2000.

[7] W. Cohen, P. Ravikumar, and S. Fienberg, “A Comparison of String Distance Metrics for Name-Matching Tasks,” In Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI), pp. 73- 78, 2003.

[8] W.W. Cohen, “Data Integration Using Similarity Joins and a Word- Based Information Representation Language,” In Proc. of the ACM Trans. Information Systems, vol. 18(3), pp. 288-321, Jul. 2000.

[9] V. Levenstine, “Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones,” Problems of Information Transmission, vol.1 pp. 8-17, 1965.

[10] J.R. Ullmann, “A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion, and Reversal Errors in Words,” The Computer Jornal, vol. 20(2), pp. 141-147, 1977.

[11] E. Ukkonen, “Approximate String Matching with q-Grams and Maximal Matches,” Theoretical Computer Science, vol. 92(1), pp. 191 - 211, 1992.

[12] R.C. Russell Index, U.S. Patent 1,261,167, http://patft.uspto. gov/netahtml/srchnum.htm, 1918.

[13] R.C. Russell Index, U.S. Patent 1,435,663, http://patft.uspto. gov/netahtml/srchnum.htm, 1922.

[14] P. Jaccard, “Étude Comparative de la Distribution Florale Dans Une Portion des Alpes et des Jura,” Bulletin del la Société Vaudoise des Sciences Naturelles, vol. 37, pp.547 - 579, 1901.

[15] M.A. Jaro, “Unimatch: A Record Linkage System: User’s Manual,” technical report, US Bureau of the Census, Washington, D.C., 1976.

[16] W. E. Winkler, “The State of Record Linkage and Current Research Problems,” Statistics of Income Division, Internal Revenue Service Publication R99/04, 1999.

[17] Internet Movies Database. http://www.imdb.com. Accessed April, 2010.

[18] Y. Yang, “Expert Network: Effective and Efficient Learning From Human Decisions in Text Categorization and Retrieval,” In Proc. of the Intl Conf on Research and Development in Information Retrieval (ACM SIGIR), pp. 13–22, 1994.

[19] Y., Yang and J. O. Pedersen, “A Comparative Study on Feature Selection in Text Categorization,” In Proc. of the Intl. Conf. on Machine Learning (ICML), pp. 412-420, 1997.

[20] F. Ataa-Allah, W. I. Grosky, and D. Aboutajdine, “Document Clustering Based on Diffusion Maps and a Comparison of the k-Means Performances in Various Spaces,” In Proc. of the IEEE Symposium on Computers and Communications (IEEE ISCC), pp.579-584, Jul. 2008.

[21] T. Joachims, “Text Categorization with Support Vector Machines:Learning with Many Relevant Features,” Proc. of the European Conference on Machine Learning (ECML), pp. 137–142, 1998.

[22] R. Agrawal, C.H. Wu, W. I. Grosky, and F. Fotouhi, “Diffusion Maps-Based Image Clustering,” In Proc. of the Intl. Workshop On Research Issues in Digital Libraries (IWRIDL), pp.1393-1403, 2006.

[23] M., Bilenko and R.J., Mooney, "Adaptive Duplicate Detection Using Learnable String Similarity Measures," In Proc of the Intl. Conf. on Knowledge Discovery and Data Mining (ACM SIGKDD), pp. 39-48, 2003.

[24] S. N., Minton, C., Nanjo, C. A., Knoblock, M., Michalowski, and M., Michelson, "A Heterogeneous Field Matching Method for Record Linkage," In Proc. of the Intl. Conf. on Data Mining (IEEE ICDM), pp. 314-321, 2005.

[25] I., Battacharya and L., Getoor, "Iterative Record Linkage for Cleaning and Integration," In Proc. of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery, pp. 11-18, 2004.

[26] M. Bilenko, S. Basu, and M. Sahami, "Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping," In Proc. of the Intl. Conf. on Data Mining (IEEE ICDM), pp. 58- 65, 2005.

[27] Pubmed Dataset. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi or http://gateway.nlm.nih.gov/gw/Cmd.

[28] P. Praks, L. Machala, and V. Snasel, "On SVD-free Latent Semantic Indexing for Iris Recognition of Large Databases," Springer, In: V. A. Petrushin and L. Khan (Eds.) Multimedia Data mining and Knowledge Discovery (Part V, Chapter 24), 2007.

[29] A., Karakasidis, and V.S. Verykios, "Privacy Preserving Record Linkage Using Phonetic Codes," In Proc. of BCI, pp. 101-106, 2009.

[30] M., Scannapieco, I., Figotin, E., Bertino, and A. K., Elmagarmid, "Privacy Preserving Schema and Data Matching," In Proc. of ACM SIGMOD, pp. 653-664, 2007.

[31] T., Churces and P. Christen, "Some Methods for Blindfolded Record Linkage," BMC Medical Informatics and Decision Making, vol. 4(9), pp. 1-17, 2004.

[32] A., Al-Lawati, D., Lee, and P., McDaniel, "Blocking-aware Private Record Linkage," In Proc. of ACM SIGMOD workshop on Information Quality in Information Systems, pp. 59-69, 2005.

[33] R., Agrawal, A., Evfimievski, and R., Srikant, "Information Sharing Across Private Databases," In Proc. of ACM SIGMOD Intl. Conf. on Management, pp. 86-97, 2003.

[34] M.J., Freedman, K., Nissim, and B., Pinkas, "Efficient Private Matching and Set Intersection," In Proc. of EUROCRYPT, pp. 1-19, 2004.

[35] B., Hawashin, F., Fotouhi, and W., Grosky, "Diffusion Maps: A Superior Semantic Method to Improve Similarity Join Performance," In Proc. of ICDM MMIS Workshop, pp. 9-16, 2010.

[36] X., He and P., Niyogi, "Locality Preserving Projections," Advances in Neural Information Processing Systems, Cambridge, MA: MIT. Press, 2003.

[37] Amazon Website: http://amazon.com. Accessed Nov. 2010.

[38] B., Tang, M., Shepherd, E., Milios, and M., Heywood, "Comparing and Combing Dimension Reduction Techniques For Efficient Test Clustering," In Proc. of SIAM Workshops, 2005.

[39] http://www.zjucadcg.cn/dengcai/Data/data.html.

[40] K., Kukich, "Techniques For Automatically Correcting Words in Text," ACM Computing Surveys. vol. 24(4), pp. 377– 439, 1992.

[41] W.W. Cohen, "Data Integration Using Similarity Joins and a Word-Based Information Representation Language," ACM Trans. Information Systems, vol. 18(3), pp. 288-321, 2000.

[42] N., Koudas and D., Srivastava, "Approximate Joins: Concepts and Techniques," In Proc. of the Intl. Conf. on Very Large Databases (VLDB) , pp.13-63, 2005.

[43] E., Monge and C. P., Elkan, "The Field Matching Problem: Algorithms and Applications," In Proc. of the Intl. Conf. on Knowledge Discovery and Data Mining (KDD), pp. 267 – 270, 1996.

[44] W.W. Cohen, "Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity," In Proc. of ACM SIGMOD, pp. 201-212, 1998.

[45] L. Gravano, P.G. Ipeirotis, N. Koudas, and D. Srivastava, "Text Joins in an RDBMS for Web Data Integration," In Proc. of Intl. World Wide Web Conf. (WWW), pp. 90-101, 2003.

[46] I.P. Fellegi and A.B. Sunter, "A Theory for Record Linkage," J. Am. Statistical Assoc., vol. 64(328), pp. 1183-1210, 1969.

[47] M.A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida," J. Am. Statistical Assoc., vol. 84(406), pp. 414-420, 1989.

[48] W.E. Winkler, “Improved Decision Rules in the Felligi-Sunter Model of Record Linkage,” Technical Report, Statistical Research Report Series RR93/12, US Bureau of the Census, Washington, D.C., 1993.

[49] R.C. Russell Index, U.S. Patent 1,261,167, http://patft.uspto. gov/netahtml/srchnum.htm, Apr. 1918.

[50] M., Bilenko and R.J., Mooney, “Adaptive Duplicate Detection Using Learnable String Similarity Measures,” In Proc. of Intl. Conf. on Knowledge Discovery and Data Mining(ACM SIGKDD), pp. 39-48, 2003.

[51] S., Chaudhuri , B. C., Chen , V. Ganti , and R. Kaushik, “Example-Driven Design of Efficient Record Matching Queries,” In Proc. of the Intl. Conf. on Very Large Databases (VLDB), pp. 327-338, 2007.

[52] Michael Szymon Spiz, “Using Latent Semantic Indexing for Data Deduplication,” Industrial Conference on Data Mining, pp. 37- 48, 2006.

[53] A.E. Monge and C.P. Elkan, “An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records,” In Proc. of ACM SIGMOD DMKD Workshop, pp. 23-29, 1997.

[54] S. Sarawagi and A. Bhamidipaty, “Interactive Deduplication Using Active Learning,” In Proc. of ACM Intl. Conf. on Knowledge Discovery and Data Mining (ACM SIGKDD), pp. 269-278, 2002.

[55] P.S. Bradley and U.M. Fayyad. “Refining Initial Points for K-Means Clustering,” In Proc. of Intl. Conf. on Machine Learning(ICML), pp. 91-99, 1998.

[56] I., Battacharya and L., Getoor, “Iterative Record Linkage for Cleaning and Integration,” In Proc. of ACM SIGMOD Workshop on Data Mining and Knowledge Discovery, pp. 11-18, 2004.

[57] W.W. Cohen, “Data Integration Using Similarity Joins and a Word-Based Information Representation Language,” ACM Trans. Information Systems, vol. 18(3), pp. 288-321, 2000.

[58] N. Koudas, A. Marathe, and D. Srivastava, “Flexible String Matching against Large Databases in Practice,” In Proc. of Intl. Conf. Very Large Databases (VLDB), pp. 1078-1086, 2004.

[59] S., Guha, N., Koudas, A., Marathe, and D., Srivastava, "Merging The Results of Approximate Match Operations," In Proc. of Intl. Conf. Very Large Databases (VLDB), pp. 636-647, 2004.

[60] N. Augsten, M. Bhlen, and J. Gamper, "Approximate Matching of Hierarchical Data Using pq-grams," In Proc. of Intl. Conf. on Very Large Databases (VLDB), pages 301–312, 2005.

[61] A. Chatterjee and A. Segev, "Approximate Matchings in Scientific Databases," In Proc. of HICSS, pp. 448-458, 1994.

[62] M. Bilenko, S. Basu, and M. Sahami, "Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping," In Proc. of the Intl. Conf. on Data Mining (IEEE ICDM), pp. 58-65, 2005

[63] M., Bilenko, B., Kamath, and R. J., Mooney, "Adaptive Blocking: Learning to Scale Up Record Linkage," In Proc. of Intl. Conf. on Data Mining(IEEE ICDM), pp. 87 – 96, 2006.

[64] R., Baxter, P., Christen, and T., Churches. "A Comparison of Fast Blocking Methods for Record Linkage," In Proc. of ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, pp. 25-27, 2003.

[65] M.A. Herna´ndez and S.J. Stolfo, "Real-World Data Is Dirty: Data Cleansing and the Merge/Purge Problem," Data Mining and Knowledge Discovery, vol. 2(1), pp. 9-37, 1998.

[66] P., Christen and T., Churches, "Joint Computer Science Technical Report," Technical Report, TR-CS-02-05: Febrl – Freely extensible biomedical record linkage. Canberra: Australian National University. 2002. http://cs.anu.edu.au/techreports/2002/TR-CS-02-05.html

[67] W.W. Cohen and J. Richman, "Learning to Match and Cluster Large High Dimensional Data Sets for Data Integration," In Proc. of Int'l Conf. Knowledge Discovery and Data Mining (ACM SIGKDD), pp. 475-480, 2002.

[68] Y. M. Cheong and J. C. Tay, "Approximate String Matching for Multiple-Attribute, Large-Scale Customer Address Databases," In Proc. of Intl. Conf. on Asian Digital Libraries (ICADL), pp. 168-172, 2003.

[69] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, “Robust and Efficient Fuzzy Match for Online Data Cleaning,” In Proc. of ACM SIGMOD, pp. 313-324, 2003.

[70] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, “Approximate String Joins in a Database (Almost) for Free,” In Proc. of Intl. Conf. Very Large Databases (VLDB), pp. 491-500, 2001.

[71] C. Xiao, W. Wang, X. LIN, and J.X. Yu, “Efficient Similarity Joins for Near Duplicate Detection,” In Proc. of WWW, pp. 131-140, 2008.

[72] S. Chaudhuri, V. Ganti, and R. Kaushik. “A Primitive Operator for Similarity Joins in Data Cleaning,” In Proc. of Intl. Conf. on Data Engineering (IEEE ICDE), pp. 5-5, 2006

[73] C., Xiao, W., Wang, and X., Lin, “Ed-Join: An Efficient Algorithm for Similarity Joins with Edit Distance Constraints,” In Proc. of Intl. Conf. Very Large Databases (VLDB), pp. 933-944, 2008.

[74] A., McCallum , K., Nigam ,and L., Ungar, “Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching,” In: Ramakrishnan R, Stolfo S (eds) Proc. of the ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, vol. 6. ACM Press, New-York, NY, USA, 2000.

[75] R., Baxter, P., Christen, and T., Churches, “A Comparison of Fast Blocking Methods for Record Linkage,” In Proc. of ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, pp. 25-27, 2003.

[76] Comon, P. , “Independent Component Analysis: a New Concept?,” Signal Processing, Elsevier vol. 36(3), pp. 287-314, 1994.

[77] Lloyd., S.P. , “Least squares quantization in PCM,” IEEE Trans. on Information Theory, vol 28(2), pp. 129-137, 1957.

[78] H., Gävert, J., Hurri, J., Särelä, and A., Hyvärinen, “FastICA Software Package for Matlab,” 1996 - 2005.

[79] I., Yoo and X, Hu, "A Comprehensive Comparison Study of Document Clustering for a Biomedical Digital Library MEDLINE," In Proc. of ACM/IEEE-CS joint conference on Digital Libraries, pp. 220 - 229, 2006.

[80] B., Hawashin, F., Fotouhi, T.M., Truta, and W., Grosky, "Efficient Privacy Preserving Protocols for Similarity Join", Transactions on Data Privacy, vol 4(3), 2011. (To appear).

[81] P., Christen, "A Survey of Indexing Techniques for Scalable Record Linkage," IEEE Trans. on Knowledge and Data Discovery, vol. PP (99), pp. 1 - 20, 2011.

[82] A., Elmagarmid, P., Ipeirotis, and V., Verykios, "Duplicate Record Detection: A Survey," IEEE Trans. Knowl. Data Eng., vol 19(1), pp. 1-16 , 2007.

[83] B., Hawashin, F., Fotouhi, and T. M., Truta, " A Privacy Preserving Efficient Protocol for Semantic Similarity Join Using Long String Attributes," In Proc. of ACM EDBT/ICDT PAIS Workshop, ACM Digital Library, doi 10.1145/1971690.1971696, 2011.

ABSTRACT

A NEW SEMANTIC SIMILARITY JOIN METHOD USING DIFFUSION MAPS AND LONG STRING TABLE ATTRIBUTES

by

BILAL HAWASHIN

December 2011

Advisor: Dr. Farshad Fotouhi

Major: Computer Science

Degree: Doctor of Philosophy

With the rapid increase of the distributed data sources, and in order to make information integration, there is a need to combine the information that refers to the same entity from different sources. However, there are no global conventions that control the format of the data, and it is impractical to impose such global conventions. Also, there could be some spelling errors in the data as it is entered manually in most of the cases. For such reasons, the need to find and join similar records instead of exact records is important in order to integrate the data. Most of the previous work has concentrated on similarity join when the join attribute is a short string attribute, such as person name and address. However, most databases contain long string attributes as well, such as product description and paper abstract, and up to our knowledge, no work has been done in this direction. The use of long string attributes is promising as these attributes contain much more information than short string attributes, which could improve the similarity join performance. On the other hand, most of the literature work did not consider the

semantic similarities during the similarity join process.

To address these issues, 1) we showed that the use of long attributes outperformed the use of short attributes in the similarity join process in terms of similarity join accuracy with a comparable running time under both supervised and unsupervised learning scenarios; 2) we found the best semantic similarity method to join long attributes in both supervised and unsupervised learning scenarios; 3) we proposed efficient semantic similarity join methods using long attributes under both supervised and unsupervised learning scenarios; 4) we proposed privacy preserving similarity join protocols that supports the use of long attributes to increase the similarity join accuracy under both supervised and unsupervised learning scenarios; 5) we studied the effect of using multi-label supervised learning on the similarity join performance; 6) we found an efficient similarity join method for expandable databases.

AUTOBIOGRAPHICAL STATEMENT

BILAL HAWASHIN

EDUCATION

- Doctor of Philosophy (Computer Science), December 2011
  Wayne State University, Detroit, Michigan, United States

- Master of Science (Computer Science), September 2003
  New York Institute of Technology, Irbid, Jordan

- Bachelor of Science (Computer Science), February 2002
  University of Jordan, Amman, Jordan

PUBLICATIONS

Bilal Hawashin, Farshad Fotouhi, and William Grosky, "Diffusion Maps: A Superior Semantic Method to Improve Similarity Join Performance," In Proc. of IEEE ICDM MMIS Workshop, pp. 9-16, 2010.

Bilal Hawashin, Farshad Fotouhi, and Traian Marius Truta, " A Privacy Preserving Efficient Protocol for Semantic Similarity Join Using Long String Attributes", In Proc. of ACM EDBT/ICDT PAIS Workshop, 2011.

Bilal Hawashin, Farshad Fotouhi, Traian Marius Truta, and William Grosky, "Efficient Privacy Preserving Protocols for Similarity Join", Transactions on Data Privacy, 2011. (To appear).

Bilal Hawashin, Farshad Fotouhi, and William Grosky, "An Efficient Unsupervised Similarity Join Method using Diffusion Maps". (To be submitted)

TEACHING EXPERIENCE

Full Time Instructor at Department of Computer Information Systems, Jordan University of Science and Technology, Irbid, Jordan. 2003 – 2007.

AWARDS AND HONORS

- Golden Key Honor Society. Wayne State University Chapter of 2011.