# Unsupervised Duplicate Detection (UDD)
# Of Query Results from Multiple Web Databases

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

Of the Requirements for the Degree

Masters of Science in Computer Science

By

Bhanupreeti Daggupati

December 2011

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

_____

Advisor: Dr. Andrzej Bieszczad Date

_____

Advisor: Dr. Peter Smith Date

_____

Advisor: Dr. Richard Wasniowski Date

APPROVED FOR THE UNIVERSITY

_____

Dr. Gary A. Berg Date

# Unsupervised Duplicate Detection (UDD) Of Query Results from Multiple Web Databases

By

Bhanupreeti Daggupati

Computer Science Program

California State University Channel Islands

# Abstract

There has been an exponential growth of data in the last decade both in public and private domain. This thesis presents an unsupervised mechanism to identify duplicates that refer to the same real-world entity. With an unsupervised algorithm, there is no need for manual labeling of training data. This thesis builds on this idea by introducing an additional classifier, known as the blocking classifier. Various experiments are conducted on a dataset to verify the effectiveness of the unsupervised algorithm in general and the additional blocking classifier in particular.

# Acknowledgements

I would like to thank Dr. Andrzej Bieszczad for his guidance and support pointing me in the right direction. I would also like to thank my husband for his endless support and hours of proofreading. Finally, I would like to thank my friends and family for putting up with my long hours and lack of availability.

# Table of Contents

# Table of Equations



# Table of Figures

# Table of Tables

# Chapter 1 – Introduction

## 1.1 Introduction

These days we have to agree to the statement "A search engine is the window to the Internet". The basic premise of a search engine is to provide search results based on query from the user. A dynamic web page is displayed to show the results as well as other relevant advertisements that seem relevant to the query. This forms the basic monetization technique used by many popular search engines. The search engine contains a database that stores these links to the web pages and a framework to decide the sequence/order these results are displayed.

With the exponential growth of the web pages and end users demand for optimal search results, there has been a huge push in using data mining techniques [1] to perfect the process of understanding the data as well as pre-processing and data preparation.

When dealing with large amount of data, it's important that there be a well defined and tested mechanism to filter out duplicate results. This keeps the end results relevant to the queries. Duplicate records exist in the query results of many Web databases, especially when the duplicates are defined based on only some of the fields in a record. Using exact matching technique as part of preprocessing, records that are exactly the same in all relevant matching fields can be merged.

The techniques that deal with duplicate detection can be broadly classified as those requiring training data (supervised learning method [2]) and those that can function without a predefined training data (un-supervised learning method [3]). As part of my thesis, I plan to explore un-supervised techniques and develop/propose a mechanism that can function with minimal supervision.

The premise of using web search engine example is to highlight the need for an algorithm that can handle large amounts of data and be able to derive a unique set that is most relevant to the user query.

## 1.2 Problem Definition

The end user has no control over the results returned by a search engine, nor can he guarantee that there will be no duplicates from the query result. The problem of duplicate records existing in a query result referring to the same real-world entity can occur when search engine uses multiple web databases. Our focus is on Web databases from the same domain, i.e., Web databases that provide the same type of records in response to user queries.

Suppose there are $s$ records in data source $A$ and there are $t$ records in data source $B$, with each record having a set of fields/attributes. Each of the $t$ records in data source $B$ can potentially be a duplicate of each of the $s$ records in data source $A$. The goal of duplicate detection is to determine the matching status, i.e., duplicate or non-duplicate, of these $s * t$ record pairs.

In order to simulate real world data, we are going to work with restaurant dataset [59] which has two databases FODORS and ZAGAT containing information on restaurants.

Below is an example of sample data from this dataset for a search of '*Los Angeles* '.

| # | **Restaurant Name** | **Address** | **City** | **Phone** | **Cuisine** | ***Database*** |
|---|---|---|---|---|---|---|
| 1 | Arnie morton's of Chicago | 435 s. la cienega blvd. | Los Angeles | 310-246-1501 | steakhouses | *FODORS* |
| 2 | Arnie morton's of Chicago | 435 s. la cienega blv. | Los Angeles | 310/246-1501 | American | *ZAGAT* |
| 3 | Locanda veneta | 8638 w. third st. | Los Angeles | 310-274-1893 | Italian | *FODORS* |
| 4 | Locanda veneta | 3rd st. | Los Angeles | 310/274-1893 | Italian | *ZAGAT* |

*Table 1.1: Sample dataset showing records from two databases FODORS and ZAGAT for a list of restaurants in Los Angeles*.

When compared, the record pairs (*1,2*) and (*3,4*) of Table 1.1 are duplicates and refer to the same real-world entity even though some of the fields have different values. This is because of how the data is classified and stored in a particular database varies from database to database. For example records *1* and *2* from Table 1.1 the only differentiating attribute is the cuisine value which is stored as "*steakhouses*" in FODORS database and "*American*" in ZAGAT database.

The problem now is that the search engine needs to identify records that refer to the same entity and display a unique set. If only exact matching algorithm is used it wrongly identifies all the four records from Table 1.1 as unique. The goal of this thesis is to develop a mechanism that can identify duplicate record pairs in the query results that refer to the same real-world entity using un-supervised algorithm.

## 1.3 Need for a Unsupervised Duplicate Detection algorithm

Duplicate detection is used interchangeably with record linkage [4] [5] [6]. The problem of duplicate detection has attracted attention from research fields including Databases [7], Data Mining [8] [9], Artificial Intelligence [10] [11], and Natural Language Processing [12] [13]. There have been many efforts in finding solution to the problem of identifying duplicates records.

As we had discussed earlier in this chapter most of the research [14] [24] for identifying duplicate records is based on predefined matching rules hand-coded by domain experts or matching rules learned offline by some learning method from a set of training examples. Such approaches work well in a traditional database environment, where all instances of the target databases can be readily accessed. In a Web database scenario, the records to match are highly query-dependent and they can only be obtained through online queries. When using a traditional database the data to be retrieved is known before hand which can be used to train and identify duplicates beforehand. This

is in contrast to Web databases where results from multiple sources have to be combined with no pre-determined training data.

The main reason to develop an alternate solution for Web database scenario are the following disadvantages of using hand-coding or offline-learning approaches.

First, offline-learning approaches rely on pre-determined training data. It uses training data to classify records as either duplicate or as unique. The training data constitutes all the possible queries on the dataset to ensure no duplicates are retrieved. Domain experts manually identify training data for all possible queries. In Web database scenario, as the records are retrieved dynamically from multiple databases, it's not possible to have pre-determined training data that can cover various queries from the end user.

Second, even though we have a training data representing the full dataset, duplicate detection may not work well for partial data retrieval. For example using pre-determined training data, if we had trained the system by giving equal weights (numerical value representing importance - lower value means the field is less important in influencing the result and vice versa) to all the fields, when we are try to retrieve data from this dataset for a search of '*Los Angeles'*, the records *1* and *2* from the above Table 1.1 are displayed as unique because the value of cuisine field is different for two records. As these two records refer to the same real-world entity, the system should have ignored or given a less weight to the *cuisine* field. To reduce the influence of such fields in determining which records pairs are duplicates, their weighting should be adjusted to be much lower than the weighting of other fields or even be zero. For some query results, for example the *cuisine* field itself can be an important field for determining which records should matched. In the case of Web databases scenario where results are retrieved after a query is entered, the weights of the fields should change based on the query. This is a shortcoming in the supervised-learning based methods, where training and weight assignment happen before hand.

## 1.4 Proposed System

The inspiration for this thesis was the IEEE paper "*Record Matching over Query Results from Multiple Web Databases* [15]" which proposes the use of Unsupervised Duplicate Detection (UDD)[16] algorithm to detect duplicate records. This thesis builds up on the idea of UDD and is an attempt to enhance the algorithm by introducing an additional classifier known as "Blocking Classifier" to improve the accuracy rate. The proponents of UDD [15] assumed that there will be no duplicates within a database and only considered the result set from multiple databases for potential duplicates. The introduction of the blocking classifier solves this issue by looking at all the data and considering the possibility that duplicates can exist within the same database.

In the proposed system the plan was to develop an unsupervised algorithm that uses three classifiers for duplicate detection which are examined in detail in Chapter 3.

The first classifier is called the Weighted Component Similarity Summing (WCSS) Classifier where the importance of the fields is determined and duplicates are identified without any training. The idea for this classifier is to calculate the similarity between pair of records by doing a field to filed comparison.

This serves as input to the second classifier which is the Support Vector Machine (SVM) Classifier which makes use of the duplicates and non duplicates identified from the WCSS classifiers as training dataset. The SVM classifier then uses the training data and processes each record to identify/classify a record as being a duplicate or otherwise.

The focus of this thesis was on developing a third classifier called the Blocking Classifier and integrating this with the current UDD process to better identify duplicates and enhance the process. This is explained in detail in the 3rd chapter. The other major enhancement would be to add functions to the WCSS classifier which deal with string comparison that can aid in identifying duplicates.

An ideal system would be one that presents the most relevant results from multiple web databases. This thesis was an attempt to explore using alternative methodologies for record duplicate detection. By developing additional classifiers, this thesis tried to prove that an un-supervised learning system is more suitable for duplicate detection with the flexibility of learning on the fly and adapting to the queries. To measure the effectiveness of the proposed system, we compare results of experiments with the original UDD system [15]. Chapter 4 contains results and analysis of such tests.

Below is the use case diagram detailing the steps in the proposed system.

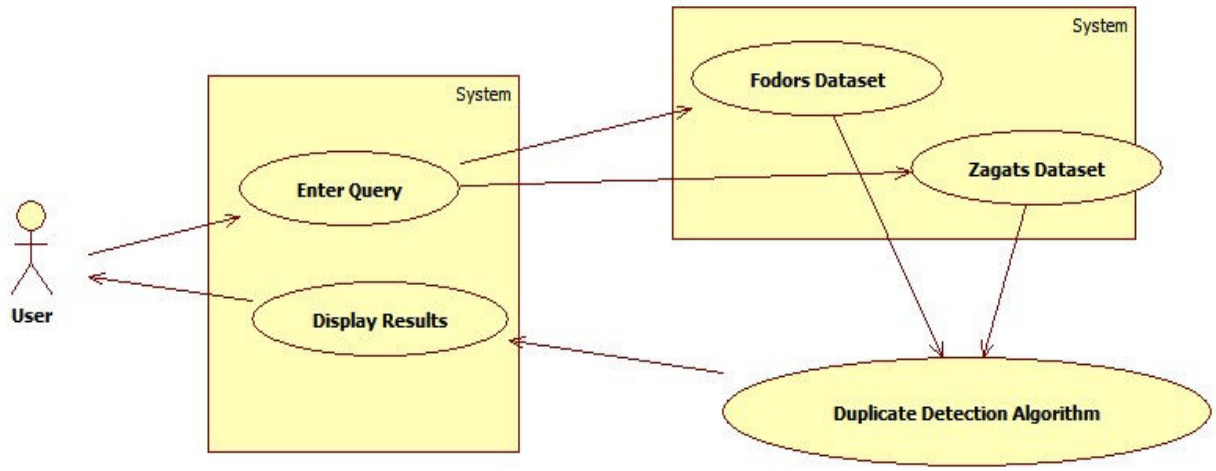



*Fig 1.1: A use case diagram of the proposed system for un-supervised duplicate detection.*

1. User enters a query
   This is normally done using a user interface either web or application that accesses the underlying database. As with any application a database or multiple databases store the information from which records are extracted for display.

2. Get all the records that match the query from multiple databases
   In a normalized database there is less possibility of duplicates existing. But in the case of Web databases this may not be the case. Also when we have multiple databases there is always a chance that the same or similar records exist that refers to the same real-world entity. This step involves in identifying all the records that match the query. There might be a challenge in streamlining the data i.e. each database might have different structure and layout. A standardized structure is used to store the records from these multiple databases.

3. Algorithm is used to identify duplicate records
   We will be using the UDD algorithm that was developed for this thesis by enhancing the original UDD algorithm.

4. Present to the user only unique records for the query
   Finally results of the algorithm are presented to the users.

## 1.5 Possible applications

One of the possible applications of duplicate detection would be data mining [66][67] whose aim is to collect data from various sources and present the same in easy to understand reports to end users. The challenge of such systems is to be able to identify duplicate data that is being processed and not including the same in the output.

The other major area and very relevant is the Master Data Management (MDM) [17] applications. These applications are to serve as a central repository of master data in an organization and the main task is to filter duplicate records that refer to the same real-world entity. MDM is used to manage non-transactional data in an organization for example customer or material master information. When there is a need to create for example a new customer, MDM system verifies if an account already exists with the given attributes and only then allow for new customer to be created. This process streamlines information and will avoid multiple accounts being created which refer to the same entity.

## 1.6 Remaining Chapters

In the second chapter, we will analyze some of the different techniques being used for duplicate detection today. We will look at various techniques for field matching that are used including character and token based similarity metrics. An overview of phonetic and numeric

similarity metrics and their relevance to duplicate detection is discussed. This chapter will also introduce the various categories of duplicate detection including the probabilistic and generic distance metrics.

In the third chapter, we will look at the custom algorithm that was developed for this thesis and the technical architecture of the system. We also look into each of the three classifiers in detail and explain their role along with technical details. This chapter has details about how the processes flows from classifier to classifier and understand the role each of them plays in the system.

In the fourth chapter, we describe the application that was built using the custom algorithm. Then we look at various experiments that were conducted. Finally we look at the results from the experiments and how they compare with expected results. We get introduced to precision, recall and *f*-measure.

In the fifth chapter, we discuss the conclusions arrived for this thesis. We will look at results from the experiments and will conclude if the new algorithm is superior to other algorithms. Finally we will look at future work and also highlight areas where further research can be done to complement this thesis.

## 1.7 Key Terms

Blocking Classifier – A classification algorithm to group data into potential duplicates and non-duplicates.

Data Mining – It is the process of discovering new patterns from large input data using computers.

Datasets - is a collection of data, usually presented in tabular form.

Duplicate Dataset – A collection of duplicate records (records that refer to the same real-world entity)

Duplicate Detection – is the process of identifying records that refer to the same real-world entity.

Duplicate Vector – An array containing the numerical equality value of two records.

Element Identification – Process of matching two datasets at field level so that matched fields refer to the same information.

Exact Matching – Matching a pair of records on the values of all the attributes

Field/Attributes – Refer to the components of a table in a database.

False positives – Records wrongly classified by the classifier as duplilcates.

Master Data Management (MDM) - comprises a set of processes and tools that consistently defines and manages the non-transactional data entities of an organization.

Non-Duplicate dataset – A collection of non-duplicate records

Potential Duplicates – A collection of records that might be referring to the same real-word entity identified by a duplicate detection process.

Record linkage – Process of linking records that refer to the same real-world entity from across multiple data sources.

Record Matching – Process of matching records that are similar

Restaurant Dataset – A collection of records/data from ZAGAT and FODORS sites about restaurant names.

Search Engine – A program used to search for information on the World Wide Web
Similarity – A numerical representation of string comparison

Similarity Vector – An array of similarity values of two records

Supervised Machine Learning - is the machine learning task of inferring a function from supervised training data.

Support Vector Machine Classifier – An algorithm used for data classification

*tf-idf* - (term frequency–inverse document frequency) is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

Training Dataset – Refers to set subset of records that are identified as a representation of the remaining records.

Unique Records – Each record uniquely represents one real-world entity.

Unsupervised duplicate Detection (UDD) –Algorithm that uses no pre-determined training data to identify duplicates that refer to the same real-world entity

Unsupervised Machine Learning - refers to the problem of trying to find hidden structure in unlabeled data with no pre-defined training data.

Web Databases – Databases that are store information accessible only through submission of online query.

WCSS - Weighted Component Similarity Summing

Weights – A numerical representation of importance

# Chapter 2 - Field Overview

In this chapter we will look at the common techniques and algorithms that are used in the field of duplicate detection. There are numerous research papers [14] being presented with the same objective of finding a better algorithm that can handle all the scenarios that come with any dataset. The most common sources of mismatches in database entries are the typographical variations of string data. Therefore, duplicate detection typically relies on string comparison techniques to deal with typographical variations.

In this section, we describe techniques that have been applied for matching fields with string data, in the duplicate record detection context. We also review briefly some common approaches for dealing with errors in numeric data. In the second part of this chapter we explore various algorithms that use these techniques.

## 2.1 Record Matching Techniques

Primarily record matching techniques can be broadly classified into the following

- Character or string based
- Token based
- Phonetic based
- Numeric similarity

### 2.1.1 Character or String based similarity metrics

These set of techniques deal with various ways in comparing strings and finding a similarity metric that can be used to group as well as identify potential duplicates. There are many papers [14] [18] published and algorithms that were developed in analyzing the correct technique for comparing strings and arriving at a differentiating factor in order to measure their similarity. The character-based similarity metrics are designed to handle common typographical errors. String based similarity metrics measure how similar (or dissimilar) two strings are, two strings are deemed identical if they have same characters in the same order.

The following are the commonly used string based similarity metrics:

- Edit distance
- Smith-Waterman distance
- Affine gap distance
- Jaro-Winkler distance metric

**2.1.1.1 Edit distance**:

This is the most commonly used technique that is based on the number of edit operations that are required to transform one string *A* to another string *B*. There are mainly three operations that constitute the edit distance.

| **Edit Operation** | **Cost** |
|---|---|
| Copy(*C*) a character from *A* to *B* | 0 |
| Delete(*D*) a character in *A* | 1 |
| Insert(*I*) a character in *B* | 1 |
| Substitute(*S*) one character for another | 1 |

*Table 2.1: List of Edit Operations along with their costs*

For example let's calculate the edit distance/cost between two strings "GUMBO" and "GAMBOL".

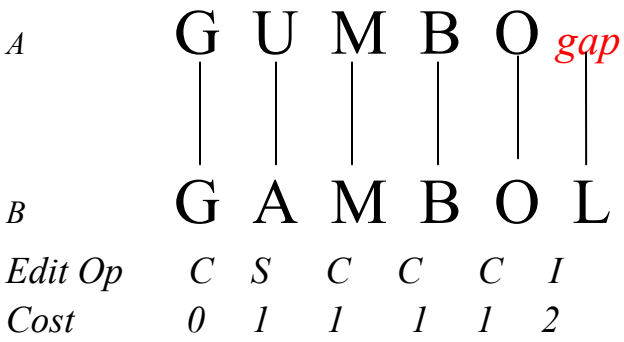



*Figure 2.1: Calculating Edit distance*

The total cost for *A* to transform into *B* is 2. Which means that by performing 2 operations on '*GUMBO*' (i.e., first operation is to substitute '*U*' with '*A*' and the second operation is to insert '*L*' at the end) it can be transformed into '*GAMBOL*'. Here the gap is nothing but an insertion operation whose cost is 1. As the strings get complex there is a need for developing an algorithm to calculate the edit distance. This calculation can be done by dynamic programming [19]. The following scoring method is used to calculate distance matrix (*D*):

$$D(i,j) = min\begin{cases} D(i-1,j-1) + Cost(A_i, B_j) & substitute/copy\ op\ is\ performed \\ D(i,j-1) + 1 & insert\ op\ is\ performed \\ D(i-1,j) + 1 & delete\ op\ is\ performed \end{cases}$$

*Equation 2.1: Formula Edit Distance Calculation*

Where *D(i,j)*=score of matrix element i, j;

*Cost*($A_i$, $B_j$) = cost of matching $A_i$ with $B_j$ //if $A_i = B_j$ then $C=0$ else $C=1$.

Using the formula let's calculate the distance matrix for the above example.

| | | **G** | **U** | **M** | **B** | **O** |
|---|---|---|---|---|---|---|
| | $D_{00}$ | $D_{01}$ | $D_{02}$ | $D_{03}$ | $D_{04}$ | $D_{05}$ |
| **G** | $D_{10}$ | $D_{11}$ | $D_{12}$ | $D_{13}$ | $D_{14}$ | $D_{15}$ |
| **A** | $D_{20}$ | $D_{21}$ *subst/copy* *D(i-1,j-1)+C($S_i$,$T_j$)* | $D_{22}$ delete *D(i-1,j)+1* | $D_{23}$ | $D_{24}$ | $D_{25}$ |
| **M** | $D_{30}$ | $D_{31}$ insert *D(i,j-1)+1* | ??? $D_{32}$ | | | |
| **B** | $D_{40}$ | | | | | |
| **O** | $D_{50}$ | | | | | |
| **L** | $D_{60}$ | | | | | |

*Table 2.2: Edit Distance Matrix*

As we can see from the above table 2.2 the minimum of all three values i.e., delete ($D_{22}$), insert ($D_{31}$) and substitute ($D_{21}$) is selected for $D_{32}$ value.

Now let's look at the actual values of this example:

Step 1: First we initialize the rows and columns and set to $D_{00} = 0$. Rows are initialized 0 to *n* and columns *0* to *m,* where *n* is the length of string *A* and *m* is the length of string *B*.

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 | | | | | |
| G | | | | | | |
| A | | | | | | |
| M | | | | | | |
| B | | | | | | |
| O | | | | | | |
| L | | | | | | |

*Table 2.3: First step in Distance matrix- initialize to $D_{00} = 0$*

Step 2: Calculate for $D_{01}$ cell using the equation 2.1.

$$D(0,1) = min \begin{cases} D(0-1,1-1) + Cost(A_1, B_1) & substitute/copy\ op\ is\ performed \\ D(0,0) + 1 & insert\ op\ is\ performed \\ D(0-1,1) + 1 & delete\ op\ is\ performed \end{cases}$$

As this is the first row there are no delete or substitute operations and the only available cell is the $D_{00}$. Minimum value is derived from it.

$$D(1,1) = min \begin{cases} \infty & substitute/copy\ op\ is\ performed \\ 0 + 1 & insert\ op\ is\ performed \\ \infty & delete\ op\ is\ performed \end{cases}$$

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 insert → 1<br>*D(i,j-1)+1* | | | | | |
| G | | | | | | |
| A | | | | | | |
| M | | | | | | |
| B | | | | | | |
| O | | | | | | |
| L | | | | | | |

*Table 2.4: Distance matrix calculation – derive value for $D_{01}$*

Step 3: Now let's look at calculation for cell $D_{10}$.

$$D(1,0) = min \begin{cases} D(1-1,0-1) + Cost(A_1, B_1) & substitute/copy\ op\ is\ performed \\ D(1-1,0) + 1 & insert\ op\ is\ performed \\ D(1,0-1) + 1 & delete\ op\ is\ performed \end{cases}$$

As this is the first column there are no insert or substitute operations and the only available cell is the $D_{00}$. Minimum value is derived from it.

$$D(1,1) = min \begin{cases} \infty & copy\ op\ is\ performed \\ \infty & insert\ op\ is\ performed \\ 0 + 1 & delete\ op\ is\ performed \end{cases}$$

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 delete<br>*D(i-1,j)+1* | 1 | | | | |
| G | 1 | | | | | |
| A | | | | | | |
| M | | | | | | |
| B | | | | | | |
| O | | | | | | |
| L | | | | | | |

*Table 2.5: Distance matrix calculation – derive value for $D_{10}$*

Step 4: Let's look at calculation of cell $D_{11}$

$$D(1,1) = min \begin{cases} D(0,0) + Cost(A_1, B_1) & substitute/copy\ op\ is\ performed \\ D(1,0) + 1 & insert\ op\ is\ performed \\ D(0,1) + 1 & delete\ op\ is\ performed \end{cases}$$

$$D(1,1) = min \begin{cases} 0 + 0 & copy\ op\ is\ performed \\ 1 + 1 & insert\ op\ is\ performed \\ 1 + 1 & delete\ op\ is\ performed \end{cases}$$

*Cost ($A_1$,$B_1$ ) = 0 because $A_1 = B_1$*

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 | 1 | | | | |
| G | 1 | 0 | | | | |
| A | | | | | | |
| M | | | | | | |
| B | | | | | | |
| O | | | | | | |
| L | | | | | | |

*Table 2.6: Distance matrix calculation – derive value for $D_{11}$*

As we can see from the above calculation we derive a value of 0 for cell $D_{11}$.

Step 5: Then the values for the rest of the cells is calculated by using the above formula/ equation 2.1.

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 1 | 0 | 1 | 2 | 3 | 4 |
| A | 2 | 1 | 1 | 2 | 3 | 4 |
| M | 3 | 2 | 2 | 1 | 2 | 3 |
| B | 4 | 3 | 3 | 2 | 1 | 2 |
| O | 5 | 4 | 4 | 3 | 2 | 1 |
| L | 6 | 5 | 5 | 4 | 3 | 2 |

*Table 2.7: Distance matrix for 'GUMBO' and 'GAMBOL'*

The edit distance/cost of the two strings is in the lower right hand corner of the matrix, i.e. 2 (circled above). This distance is used for calculating the similarity between the two strings. If the calculated edit distance is greater than the threshold value then similarity between them is low and vice versa. The threshold value is set by the user and is based on the algorithm being used for duplicate detection. Edit distance is popularly known as *Levenshtein distance* [20]. The biggest advantage of using this method is that it is very effective in catching typical typographical errors. Common applications that use this method are the file revision, spelling correction and speech recognition.

**2.1.1.2 Smith-Waterman distance**:

This is further extension of edit distance metric which was proposed by Smith and Waterman [21][22]. In this method two strings are compared with a view to identifying highly similar sections within them. This approach assumes that mismatches at the beginning and the end of strings have lower costs than mismatches in the middle. For example this method correctly identifies "*Stanford U*" as being similar to "*Stanford University*". This is because when the two strings are compared, a common substring "*tanford U*" is extracted and the algorithm assumes charters at the start and end have a low cost. If the cost is less than threshold value it classifies these as being similar or vice versa. The threshold value is user configurable and is based on the algorithm being used for duplicate detection.

These similar sections are identified by calculating alignment matrix (*D*) using the following scoring method:

$$D(i,j) = max \begin{cases} 0 & Start\ over \\ D(i-1,j-1) + Cost(A_i, B_j) & substitute/copy\ op\ is\ performed \\ D(i-1,j) + Gap & insert\ op\ is\ performed \\ D(i,j-1) + Gap & delete\ op\ is\ performed \end{cases}$$

*Equation 2.2: Smith-Waterman Distance Calculation*

Where *D(i,j)*=score of matrix element *i, j*;

*Cost*($A_i$, $B_j$) = cost of matching $A_i$ with $B_j$ //if $A_i = B_j$ then *Cost=1* else *Cost=-1*.

Gap = -2

The calculation of alignment matrix in Smith-Waterman algorithm is slightly different from Edit distance.

1. The first step of Smith-Waterman algorithm is to set first row and column to zero.
2. The cost of each of the operation is also different. For example the cost for insertion or deletion is -2 which is represented by "*Gap"* in the above formula.

The rest of calculation in the matrix is the same as in edit distance.

| | | G | U | M | B | O |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 1 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 0($D_{22}$) | 0 | 0 | 0 |
| M | 0 | 0 | 0 | 1($D_{33}$) | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 2($D_{44}$) | 0 |
| O | 0 | 0 | 0 | 0 | 0 | 3($D_{55}$) |
| L | 0 | 0 | 0 | 0 | 0 | 1 |

*Table 2.8: Smith-Waterman Distance matrix*

The alignment matrix in Smith-Waterman is used to find the similarity section between the two strings. So once the alignment matrix is obtained we need to trace back from the element in the matrix with the maximum score until a cell with a score of zero is reached.

We had seen earlier how the cell values are calculated, let's consider the example from table 2.8, here the highest score is *3* ($D_{55}$) in the entire matrix. Starting from this cell in the matrix by doing a trace back we obtain $D_{44}$ (this is from where the $D_{55}$ value was initially calculated i.e., $D_{55} = D(i-1,j-1) + Cost(A_i, B_j) = 2+1 = 3$) and from $D_{44}$ we obtain $D_{33}$ and so on until we reach a zero value. This process gives us the similar section between the two strings:

Alignment: GUMBO
GAMBOL

Once we get the alignment matrix, cost are given to the strings. Lower costs are assigned to mismatches at the beginning and ending of the strings and higher cost to mismatches in middle of the strings. If cost is less than threshold value it classifies these as being similar or vice versa. The cost and threshold values are user configurable and are based on the particular algorithm being used for duplicate detection.

**2.1.1.3 Affine gap distance:**

The Affine gap distance [23] methodology differs slightly from the Smith-Waterman metric. Smith-Waterman fails on some pairs that seem quite similar: For example

William W. Cohen
William W. 'Some String' Cohen

This is because it gives higher weights to mismatches in the middle of the strings. Affine gap algorithm works well for string comparisons that have continuous insertions or deletions of characters/words. Intuitively, single long insertions are "cheaper" than a lot of short insertions. A regular gap extension method would assign a fixed cost per gap. An affine gap penalty encourages the extension of gaps rather than the introduction of new gaps.

For example let's consider two strings "*W I L LL I A A M*", "*W A L I M*" and compare the score for them using regular gap penalty and affine gap penalty.

The scores/cost for regular gap penalty (edit distance algorithm) are:
Match = 0
Substitution (mismatch) = 1
Insertion/deletion of a Gap =1

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | W | I | L | L | L | I | A | A | M |
| T | W | A | - | L | - | I | - | - | M |
| Regular Cost | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 |

*Figure 2.2: Example to calculate the distance/score under regular gap penalty system*

Now let's calculate the score by making a slight change to the alignment. For this new alignment, the *'L'* has been shifted one position to the right.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | W | I | L | L | L | I | A | A | M |
| T | W | A | - | - | L | I | - | - | M |
| Regular Cost | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 5 |

*Figure 2.3: Modified alignment. Equivalent under regular gap penalty system*

This modification does not change the score, thus the two alignments are equivalent under this scoring system.

Now let's look at these two alignments using affine gap penalties.
Match = 0
Substitution (mismatch) = 1
Gap open score = 1. //It can be for insertion or deletion
Gap extension score = 0. //It can be for insertion or deletion.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | W | I | L | L | L | I | A | A | M |
| T | W | A | - | L | - | I | - | - | M |
| Affine Cost | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 |

*Figure 2.4: Alignment from Figure 2.2 re-scored using affine gap penalties.*

For the first alignment this results in a new score of 4 (Figure 2.4) as now when a gap is extended (there is more than one _ in a row) the score is 0, whereas previously each gap received a score of 1 whether it was a new gap or an extension.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S | W | I | L | L | L | I | A | A | M |
| T | W | A | - | - | L | I | - | - | M |
| Affine Cost | 0 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 |

*Figure 2.5: The alignment from Figure 2.3 re-scored using affine gap penalties.*

When re-scoring the second alignment using affine gap penalties, the new alignment score is 3 (Figure 2.5). This score is lower than the score achieved by the first alignment, and as such is the preferred alignment. Normally this is implemented using dynamic programming [69].

This example above demonstrates the difference using affine gap penalties rather than regular gap penalties. Instead of inserting small gaps, affine gap penalties provide incentive for the alignment algorithm to keep sequence together where possible. Such a behavior is more desirable, and so most alignment algorithms [68] make use of affine gap penalties.

#### 2.1.1.4 Jaro-Winkler distance metric:

The Jaro-Winkler distance measure was developed for comparing names gathered in the U.S. Census and was proposed by Jaro[25]. It is specifically designed to look and compare surnames to surnames and given names to given names. This method couldn't be used to compare the whole name.

Given two strings $S_1$ and $S_2$, calculating the Jaro-Winkler distance includes the following steps:

i. Compute the string lengths $(|S_1|, |S_2|)$.

ii. Find the number of common characters in the two strings, common are all the characters in $S_1[i]$ and $S_2[j]$ for which $S_1[i] = S_2[j]$ and |i-j| <= ½ *min* ($|S_1|, |S_2|$).
iii. Find the number of transpositions. Here transposition refers to those characters from one string that is out of order with respect to their corresponding common characters from the other string.

The formula for calculating the Jaro-Winkler distance is:

$$d_j = \frac{1}{3}\left(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{m}\right)$$

*Equation 2.3: Jaro-Winkler Distance*

Where $m$ is the number of matching characters.

$t$ is half the number of transpositions.

To better understand the intuition behind this metric, consider the matrix $M$ in (Figure 2.6), which compares the strings $S_1$ = *WILLLAIM* and $S_2$ = *WILLIAM*. The entries have $M(i, j) = 1$ if and only if the $i^{th}$ character of $S_1$ equals the $j^{th}$ character of $S_2$. The Jaro metric is based on the number of characters in $S_1$ that are in common with $S_2$.

| | W | I | L | L | I | A | M |
|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| L | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| L | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| L | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| M | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

*Table 2.9: Jaro-Winkler Distance*

In comparing the two string there are 6 matching characters, i.e., *m=6*. Even though *'M'* and *'I'* are not in sequence they are considered as matching characters. This is because they appear in both strings which satisfies the first condition $S_1[i] = S_2[j]$ *(i.e.,* $S_1[7] = S_2[5]$ *&*$S_1[8] = S_2[7]$*)* and

For letter *'I'* $|i-j| <= ½ \min(|S_1|,|S_2|)$

$|7-5| <= ½ \min(7, 8)$

$2 <= 3.5$

For letter *'M'* $|i-j| <= ½ \min(|S_1|,|S_2|)$

$|8-7| <= ½ \min(7, 8)$

$1 <= 3.5$

$|S_1| = 8$

$|S_2| = 7$

The number of matching (but different sequence order) characters divided by the numeric value '2' defines the number of transpositions *(t)*. There are 2 mismatched characters *I* and *M* leading to t =2/2 = 1.

We calculate a Jaro score of:

$$d_j = \frac{1}{3}\left(\frac{6}{8} + \frac{6}{7} + \frac{6-1}{6}\right) = 0.33$$

The main advantage of using this method is that it can adjust weights when two strings do not agree on a character and is an efficient method in matching person names when there is a possibility of typographical errors in a large dataset.

### 2.1.2 Token-based similarity metrics

As we have seen character based comparison work effectively in catching typographical errors, but they sometime fall short when comparing a rearranged string that has the same meaning. For example when comparing "*Jane Doe*" to "*Doe, Jane*", characters based metrics fail and wrongly classify the two strings being different even though they refer to the same person name. In order to avoid such error token based similarity measures are used, where comparison of two strings is done first by dividing them into a set of tokens (a token is a single word). A common practice is to split the string at white space and form the tokens. Thus in our example the string "*Jane Doe*" becomes a token array [*"Jane", "doe"*]. As we will see further in this section, one of the main advantages of using such an approach is that we are less worried about the placement of the words in a string.

Below are some the popular token based similarity metrics:

- Jaccard coefficient
- Cosine Similarity
- q-GRAMS

### 2.1.2.1 Jaccard coefficient:

Jaccard coefficient or index is used for comparing the similarity and diversity of a given sample set. The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets [26].

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

*Equation 2.4: Jaccard coefficient similarity*

For example take two strings *s1 = 'Jane Miss Doe'* and *s2 = 'Doe Jane',* these can be split into two token arrays *s1* = [“*Jane*“, “*Miss*“, “*Doe*“] and *s2* = [“*Doe”,” Jane”*].

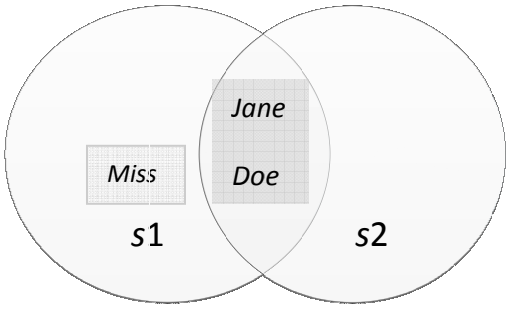



*Figure 2.6: Example of Jaccard Coefficient*

From the above Fig 2.6, we can calculate, ***J_similarity*(*s1,s2*) = 2 / 5** , where 2 is the similar words in both the strings and 5 is the total number of words in two strings.

There are certain disadvantages of using such an approach, for example if comparing two similar strings a missing value or a token can result in a skewed outcome. The other drawback is it's very sensitive to typographical errors.

As discussed the main advantage is that the result if not dependent on the location of the word in a string because all it matters is a token to be present in a string and not the location of it.

### 2.1.2.2 Cosine Similarity:

When comparing tokens of two strings represented as *n*-dimensional vectors, cosine similarity is the measure of the cosine angle that separates these vectors. The cosine value falls between 0 and 1, where 0 means that the two strings are perfectly similar and 1 means they are not. For example if two strings *A* and *B* were to be compared the first step would be

to tokenize the string (split the string at whitespace to form an array of words) and calculate the cosine similarity between the resulting two arrays of tokens. Cosine similarity is one of the most popular similarity measure applied to text documents, such as in numerous information retrieval applications [27]

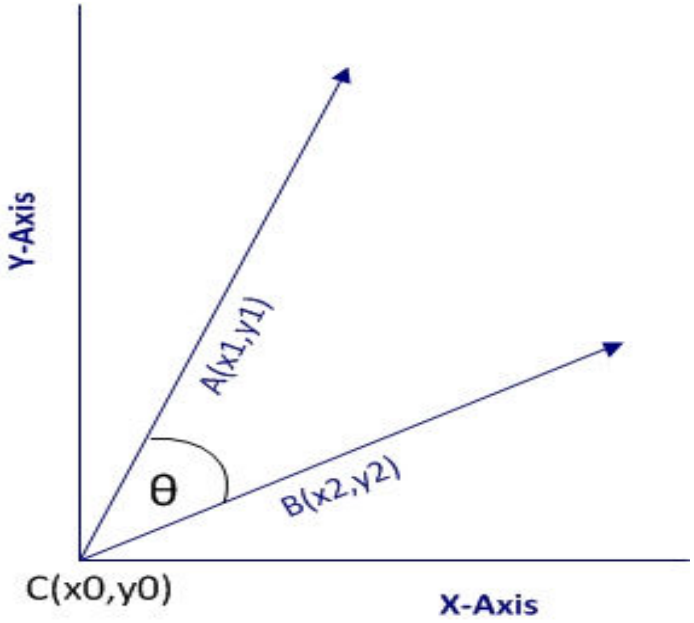



*Figure 2.7: Representing A and B strings tokenized.*

Let's consider an example of two string *A* and *B* with tokens *[A1,A2]* and *[B1 and B2]*.

$$CosineSim(A,B) = cosine\ \theta = \frac{A.B}{|A||B|} = \frac{x1 * x2 + y1 * y2}{\sqrt[2]{x1^2 + y1^2} * \sqrt[2]{x2^2 + y2^2}}$$

*Equation 2.5: Calculating the cosine angle between A and B.*

Where $x1 = log(1+tf_{A1}) * log(idf_{A1})$
$y1 = log(1+tf_{A2}) * log(idf_{A2})$
$x2 = log(1+tf_{B1}) * log(idf_{B1})$
$y2 = log(1+tf_{B2}) * log(idf_{B2})$

$tf_{A1}$ = *term frequency of token A1 in string A*
$idf_{A1}$ = *inverse document frequency of token A1 in string A*

Term Frequency- Inverse Document Frequency (*tf-idf*) is a statistical measure to measure importance of a word to a document in a collection [50].
As with Jaccard coefficient, cosine similarity too is not dependent on the placement of the words in the string

### 2.1.2.3 q-GRAMS:

A q-gram is a substring of a text, where the length of the substring is q. The idea is to break the string into tokens of the q-grams and compare with another to find similarities and count the number of matches between the strings. Additional padding is done to account for characters at the start and end so as to ensure they are not ignored in the comparison. q-gram algorithms aren't strictly phonetic matching in that they do not operate based on comparison of the phonetic characteristics of words. Instead, q-grams can be thought to compute the "distance," or amount of difference between two words. Utilizing the q-gram algorithm [28] method is highly favorable, as it can match misspelled or mutated words, even if they are determined to be "phonetically disparate."

Consider an example, when comparing two strings "*Elvis*" and "*Jarvis*" and use q=2 (length of substring), we get the following substrings, for *"Elvis"* we get *"#E" "El" "lv"* ***"vi" "is" "s#"***and for *"Jarvis"* we get *"#J" "Ja" "ar""rv"* ***"vi" "is" "s#"***. We would get a match of three in this case (highlighted above). Higher the numbers, similar are the words. This approach includes the first and last letters in the word by adding padding so that they too are considered for comparison which is in contrast to some methods that give less importance to them (e.g. Smith-Waterman distance).

| *ELVIS* | *JARVIS* |
|---|---|
| *#E* | *#J* |
| *EL* | *JA* |
| *LV* | *AR* |
| ***VI*** | *RV* |
| ***IS*** | ***VI*** |
| ***S#*** | ***IS*** |
| | ***S#*** |

*Table 2.10: Example of q-Grams when q=2*

### 2.1.3 Phonetic similarity metrics

Strings may be phonetically similar even if they are not similar at character or token level. For example the word "*Color*" is phonetically similar to "*Colour*" despite the fact that the string representations are very different. The phonetic similarity metrics try to address such issues.

#### 2.1.3.1 Soundex:

Soundex can be defined as a hashing mechanism for English words. It constitutes in converting words into a 4 character string that is made up of the first letter in the word and three numbers that are calculated by the hash function. This code would describe how a

word sounds and thus can be used to compare and find similar sounding words. An example of the same is given below.

Below are steps for deriving the American Soundex code [29]:

1. Retain the first letter of the name and drop all other occurrences of a, e, h, i, o, u, w, y.
2. Replace consonants with digits as follows (after the first letter):
   - b, f, p, v => 1
   - c, g, j, k, q, s, x, z => 2
   - d, t => 3
   - l => 4
   - m, n => 5
   - r => 6
3. Two adjacent letters with the same number are coded as a single number.
4. Continue until you have one letter and three numbers. If you run out of letters, fill in 0s until there are three numbers.

Using the above steps we can derive Soundex code for e.g. for "*Eastman*" as "*E235*", "*Easterman*" as "*E236*" and "*Westminster*" as "*W235*". By comparing the generated Soundex codes we can group words that sound similar. As in the above example, "*Eastman*" and "*Easterman*" can be grouped together as the Soundex code can be calculated as being near to each other when compared to the Soundex code of "*Westminster*".

Other approaches include New York State Identification and Intelligence System (NYSIIS) [30] and Oxford Name Compression Algorithm (ONCA) [31].

### 2.1.4 Numeric Similarity Metrics

As we have seen there are numerous approaches that have been developed for comparing strings. When comparing numerical values the methods are primitive. In most cases when it makes sense to compare numbers, it's a basic comparison and queries can be developed to extract numerical data with ease. There has been continuing research in using cosine similarity and other algorithms in analyzing numerical data [32]. For example data in numbers can be compared with primitive operators like equal, greater than and can used to calculate the difference between two numeric strings.

## 2.2 Detecting Duplicate Records

Until now we have seen various methods that can be used to compare strings or individual fields and use a metric to understand their similarity or lack of it. When applied to real-

world situations where data is multivariate and the number of fields is as dynamic as the data itself, this makes the field of duplicate detection more complicated. There have been numerous papers [14] and approaches addressing this issue. Broadly these approaches can be classified as below.

- Probabilistic approaches
- Supervised Learning
- Active-learning based techniques
- Distance based techniques
- Rule based techniques
- Un-supervised learning

We further explore each of these approaches in detail below.

### 2.2.1 Probabilistic Matching Models

As the name suggests probabilistic matching [33] uses likelihood ratio (probability) theory to classify data as duplicates. The idea of using probability for duplicate detection was proposed by Fellegi and Sunter [6][34].

Below are the steps used by a typical probabilistic algorithm [35].

i. Extract/identify a sample set of data from the complete data.
ii. Using this extracted data manually label records as being duplicate or differing pairs.
iii. Then statistics are calculated from the agreement of fields on matching and differing records to determine weights on each field.
iv. During the process, the agreement or disagreement weight for each field is added to get a combined score that represents the probability that the records refer to the same real-world entity.
v. Often there is one threshold above which a pair is considered a match, and another threshold below which it is considered not to be a match. Between the two thresholds a pair is considered to be potentially duplicate. These threshold values are defined beforehand; usually by a domain expert.

One of the popular models is the Bayesian Decision Model [36] which is based on the Bayes theorem to calculate suitable probabilities used to decide whether or not two records refer to the same entity according to user-set thresholds.

### 2.2.2 Supervised Learning

Like humans learning from past experiences, a computer system does not have "experiences". A computer system learns from data, which represents some "past experiences" of an application domain.
Below is an example of such training which is used for classification of credit card applications based on certain criteria.

| Age_Customer | Has_Job | Own_House | Credit_Score | Decision |
|---|---|---|---|---|
| Old | Yes | No | Excellent | Yes |
| Middle | No | No | Good | No |
| Young | Yes | No | Fair | Yes |

*Table 2.11: Sample training data used for supervised learning*

Using the above training data (table 2.8) the system/application is "taught/trained" how to respond given a set of criteria. Once training is completed, system uses this data to classify new records. For example using the above training data, the system can respond to a new credit card application based on the criteria defined. The training data is normally selected by a manual process. Users identify criteria that would suit all possible scenarios in that domain. Supervised learning [37] gets its name due to the use of such pre-determined training data. Such an approach is also known as classification or inductive learning.

SVMs (Support Vector Machines) [38] are a useful technique for data classification. We will examine in detail the technical details of SVM in Chapter 3 as we are using this technique in this thesis. SVM learns by examples (the first step is training of SVM with sample values) to classify data and assign labels to (classify) objects. In this thesis we will be using linear SVM classifier (making a classification decision based on the value of a linear combination of the input data) that find a hyperplane to separate two classes of data, positive and negative. This is illustrated in figure 2.8 below.

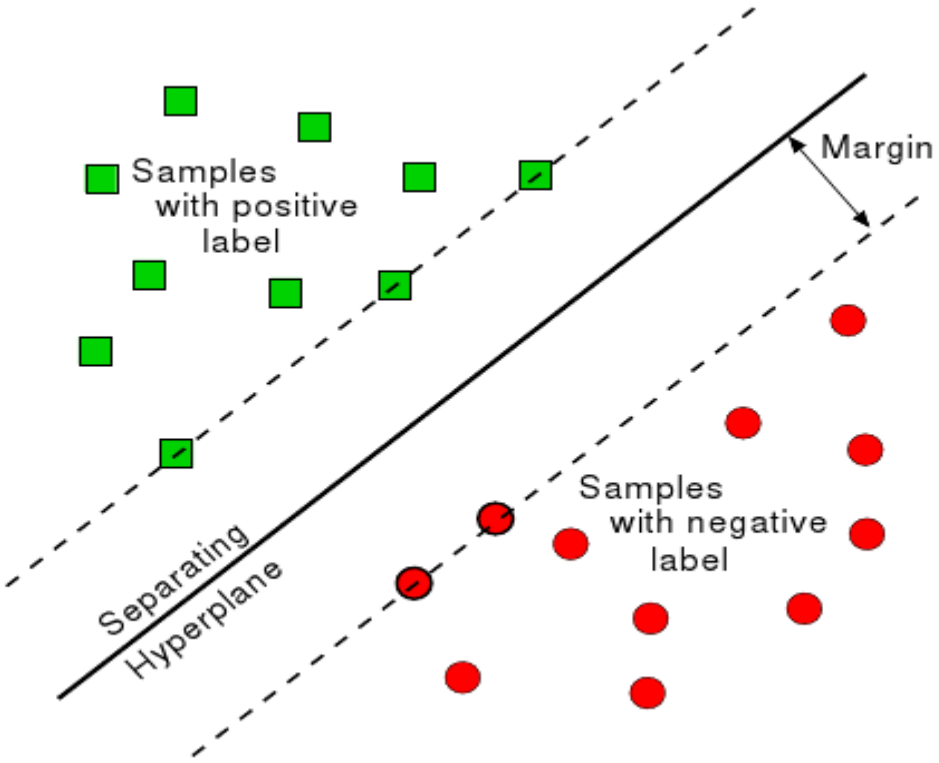



*Figure 2.8: SVM example*

SVM uses the training data as the basis of classifying new data. It is important to note that results of classification depend on the initial training data. If the input training data is true representative of the whole dataset being classified, then SVM is expected to correctly classify the data. Having such a training data is sometimes a problem with techniques that don't use active learning [39] (explained in the next section below) as they requires a large number of training examples. The labeling of training data is done manually, hence the name supervised learning.

### 2.2.3 Active-Learning-Based Techniques

When training examples are selected for a learning task at random, they may be suboptimal in the sense that they do not lead to a maximally attainable improvement in performance. As we have seen earlier SVM and other supervised learning techniques depend on pre-determined training data. It may not always be possible to have a training data that is true representative of the whole dataset. Active learning methods [40] attempt to identify those examples that lead to maximal accuracy improvements when added to the training set.

During each round of active learning, examples are added to the training set. Only examples that improve the overall performance are identified and labeled. The system is then re-trained on the training set including the newly added labeled example. Three broad classes

of active learning methods exist: (1) uncertainty sampling techniques [41] attempt to identify examples for which the learning algorithm is least certain in its prediction; (2) query-by-committee methods [42] utilize a committee of learners and use disagreement between committee members as a measure of training example's informativeness; (3) estimation of error reduction techniques [43] select examples which, when labeled, lead to greatest reduction in error by minimizing prediction variance.

### 2.2.4 Distance-Based Techniques

Active learning techniques require some training data or some human effort to create the matching models. In the absence of such training data or the ability to get human input, supervised and active learning techniques are not appropriate. One way of avoiding the need for training data is to define a distance metric for records which does not need tuning through training data.

Distance-based approaches [44] that combine each record into one big field may ignore important information that can be used for duplicate detection. A simple approach is to measure the distance between individual fields, using the appropriate distance metric (discussed in section 2.1) for each field, and then compute the weighted distance between the records. In this case, the problem is the computation of the weights and the overall setting becomes very similar to the probabilistic setting. One of the problems of the distance-based techniques is the need to define the appropriate value for the matching threshold. In the presence of training data, it is possible to find the appropriate threshold value (a numerical value that can determine if a record pair is similar). However, this would nullify the major advantage of distance-based techniques, which is the ability to operate without training data.

### 2.2.5 Rule-based Approaches

The Rule-based approach [45] is the use of rules to define whether two records are similar or not. It differs from the Probabilistic-based approach in the sense that each attribute is given either a weight of one or zero whilst each attribute in the Probabilistic-based approach is assigned a weight.
Below is an example of using rule-based approach to identify duplicate records.

*Given two records, r1 and r2*
*IF the name of r1 equals the name of r2,*
*AND the cuisine value differs slightly,*
*AND the address of r1 equals the address of r2*
*THEN*
*r1 is equivalent to r2*

The above rule can be implemented programmatically for e.g. *the cuisine value differs slightly,* by using a distance based technique. A string-matching algorithm is applied on the cuisine field of two records to determine the typographical gap. The next step would be to compare this value with a threshold value to decide if pair of records are duplicate. Importance should be given to determining this threshold value as it can result in false positives or false negatives. Having a lenient threshold would result in classifying non-duplicate pairs as duplicates, while on the other hand a strict threshold would prevent "real" duplicates pairs as being identified as duplicates. Hence it is very important for rule-based approaches to select a relevant string distance function and a proper threshold value. The threshold value is normally obtained through experimental evaluation

The main advantage of rule-based approach is it's highly customized to be domain specific as the rules are defined at field level (as seen in the above example). This can leverage domain specific information and correctly identify duplicates that refer to the same real-world entity. Rule-based approach might be helpful when there is a lot of labeled data to train. Such an approach may not be scalable and as these rules are domain specific they can't be reused on other domains.

### 2.2.5 Unsupervised Learning

As compared to supervised learning unsupervised learning requires no formal training data.
Unsupervised learning[46] is a general title for several types of learning scenarios in which the input data set contains the data points themselves without any additional information(training data), e.g., their classification. It is one the tasks of unsupervised learning system to develop classifications automatically. Unsupervised algorithms seek out similarity between pieces of data in order to determine whether they can be characterized as forming a group.
Unsupervised learning may be divided into two types of problems – data clustering and feature extraction. Data clustering, or exploratory data analysis, aims to unravel the structure of the provided data set. Feature extraction, on the other hand, often seeks to reduce the dimensionality of the data so as to provide a more 'compact' representation of the data set.

We explore in detail the concepts of unsupervised learning and how they were used as a part of this thesis.

# Chapter 3 Technical details

## 3.1 Architecture of the system

This Unsupervised Duplicate Detection (UDD) system consists of four modules: data retrieval, pre-processing and blocking, the main algorithm and data presentation. The same can be seen in fig 3.1 below.

Below is a simple architecture diagram illustrating the system.

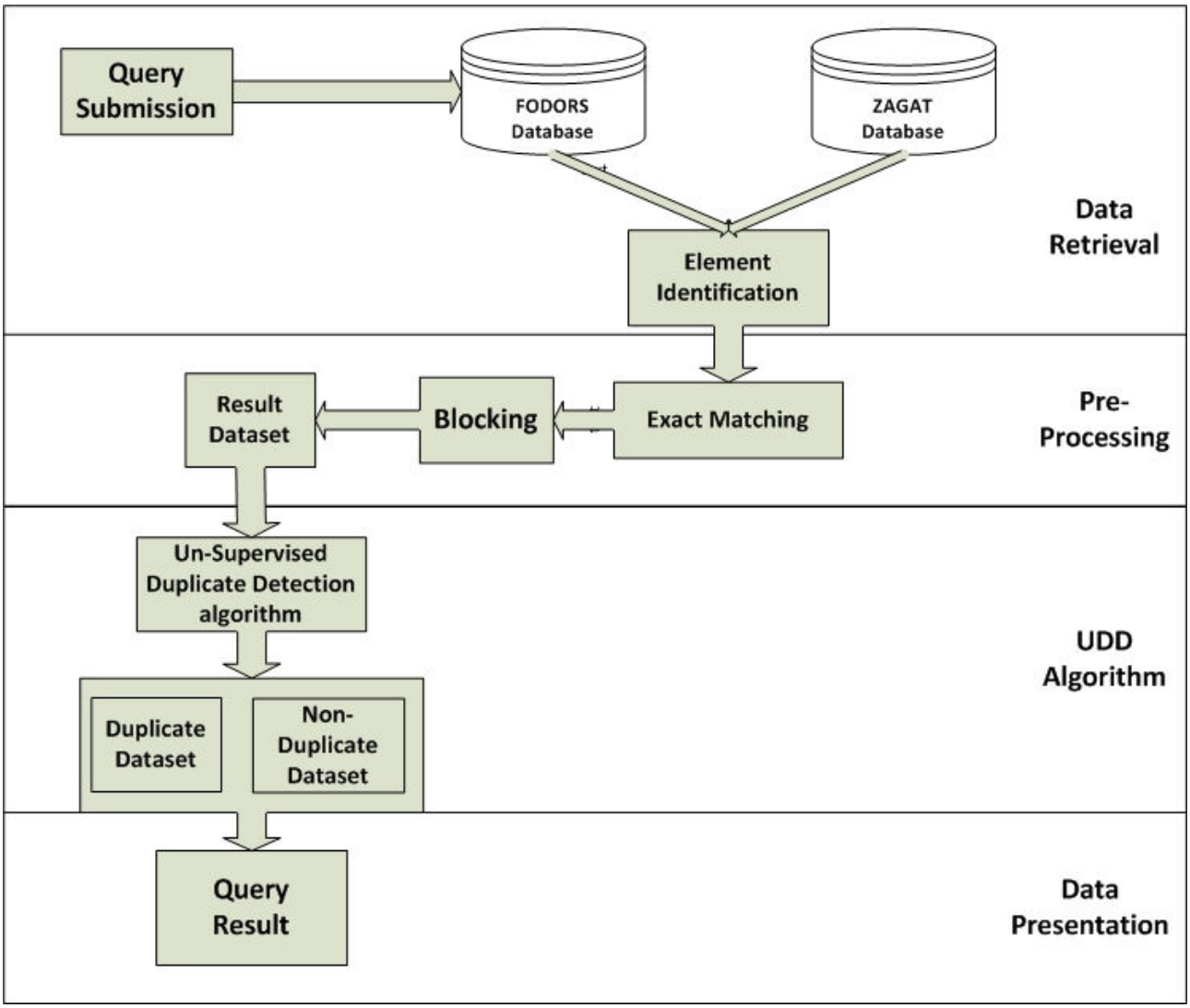



*Figure 3.1: System architecture*

### 3.1.1 Data Retrieval:

The data retrieval module consists of an interface to read the user query along with the actual data retrieval from the database. To make the application interactive and simulate a real world application a search box is given where the user can enter a query which can be for a particular word or run wide open to query all the records in the database for analysis.

The data exists in two tables and has the same structure (field names). Element identification is the process of mapping the fields of two tables and reading the information. We are dealing with a simple dataset (explained in section 3.2 below) with fixed field names. This may not be the case in a real world application where data can be stored in databases with different names/elements and the process of mapping fields between the data sources is a very crucial step in ensuring the integrity of the application.

### 3.1.2 Pre-Processing and Blocking:

The second module in the application consists of pre-processing and blocking. In this step generally data is cleansed and parsed into one structure. This step also involves removing special characters from the raw data and converting them to a lower case for accurate comparison.

As part of pre-processing, data is sorted and exact matching records are deleted. This is done comparing all the fields (taking each row as one string) and comparing it to others. This ensures the same data doesn't exist and is a basic check that can be done.

The next major and crucial step in this module is blocking. Blocking "typically refers to the procedure of subdividing data into a set of mutually exclusive subsets (blocks) under the assumption that no matches occur across different blocks [47]". Normally in order to classify records in a table, a unique hash function is generated for each record and compared with all other records in the table. Records having the same or similar hash value are categorized into groups. We will look at the details and techniques that are used for blocking and also look at the efficacy of each of them.

### 3.1.3 UDD Algorithm

The main component of the system is the module that has the UDD algorithm. In this we look at developing an algorithm that can train itself and aid in identifying duplicates. This algorithm consists of a component that calculates the similarity vectors of the selected dataset, assigning weights to the selected vectors and finally using the support vector

machine (SVM) to classify the data. The technical details and further information on each of these components is discussed at length later in this section.

### 3.1.4 Data Presentation

The final module consists of presenting the data to the user. The unique data along with statistics is presented to the user.

## 3.2 Loading of Data/Data Collection:

The dataset for this thesis is the restaurant dataset that is commonly used for record linkage analysis and is available at http://www.cs.utexas.edu/users/ml/riddle/data/restaurant.tar.gz. There are 864 restaurant names and addresses with 112 duplicates with 432 in FODORS and 432 in ZAGAT tables. These individual datasets are unique. The attributes of the dataset are name, address, city, phone and cuisine. Below is the sample data from the dataset

| **Name** | **Address** | **City** | **Phone** | **Cuisine** | ***Dataset*** |
|---|---|---|---|---|---|
| bel-air hotel | 701 stone canyon rd | bel air | 310-472-1211 | Californian | *FODORS* |
| hotel bel-air | 701 stone canyon rd. | bel air | 310/472-1211 | Californian | *ZAGAT* |
| le chardonnay (los Angeles) | 8284 melrose ave. | los Angeles | 213-655-8880 | French bistro | *FODORS* |
| le chardonnay | 8284 melrose ave. | los Angeles | 213/655-8880 | French | *ZAGAT* |

*Table 3.1: Sample data from Restaurant Dataset*

The data was transformed to excel sheet and uploaded to an Access database. As we are using a simple application to read the data, no indices were created and the SQL query was designed to look for the query string in all the fields. When there was no query string entered, the application performs a scan of all entries and analysis was done using all the table entries.

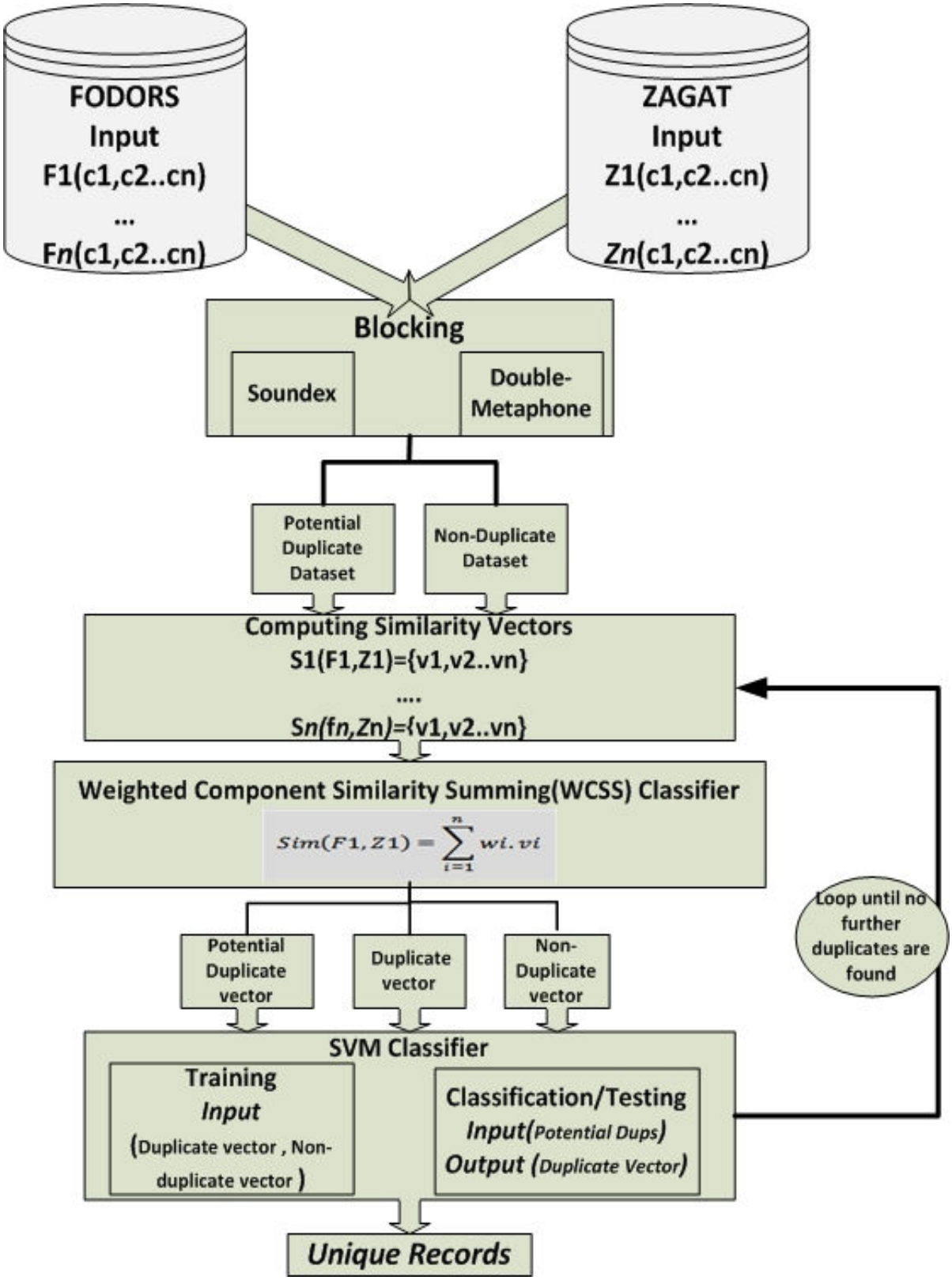



*Figure 3.2: Technical architecture*

## 3.3 Blocking

One of the main concerns that we need to address for duplicate record detection is the process of segregating the total records into potential duplicate and non-duplicate sets. This is a crucial step that can play an important role in the final outcome of the results. There has been a lot research [16][7] on addressing the issue but a perfect solution is hard to find and can vary based on the data being considered.

The focus of this thesis was to develop a working blocking technique that can be used for duplicate detection in general and as a tool for restaurant dataset in particular. Some research papers [15] assume that records from the same dataset (for e.g. FODORS / ZAGAT ) don't have duplicates within them. While this is a fair assumption, it will not hold true when we consider data from a non-relational database. This is very much possible for web databases where the data is retrieved on the fly (after the query is entered by the user, system fetches the relevant data) and such databases many not be properly structured or normalized. This thesis focuses on assuming the possibility that duplicates can exist in any database. As can be seen from the above Figure 3.2 blocking plays an important role on how the raw data is fed to the main algorithm.

The underlying assumption of this thesis is to use a blocking algorithm to classify records irrespective of the dataset they originated from. Blocking in simple terms is a technique to group data. Normally in order to classify records in a table, a unique hash function is generated for each record and compared with all other records in the table. Records having the same or similar hash value are categorized into groups. These groups are categorized as potential duplicates and non-duplicates. Blocking "refers to the procedure of subdividing data into a set of mutually exclusive subsets (blocks) under the assumption that no matches occur across different blocks [47]". There are many techniques that are used in this approach such as Soundex [29], NYSIIS[30] and Metaphone[48].

As we have seen Soundex function in Chapter 2, let's look at the double Metaphone function in detail. As with Soundex, Metaphone is a phonetic algorithm which was developed by Lawrence Philips [48]. Instead of using a fixed length as output (Soundex generates a fixed length output of 4 characters), Metaphone uses variable length keys as output. Words with similar phonetics have similar or same key. In this thesis we are applying the principles of Soundex and double Metaphone algorithms for blocking. These algorithms are applied to the *Name* field of the restaurant dataset. The blocking algorithm helps us to divide the records in to two groups - potential duplicates and non-duplicates.

Below is an example that explains the steps that are used to classify the records for search query of "*Los Angeles*".

i. Blocking module receives data from both the databases after deleting records that are exactly same by comparing all the fields in the record.

| Name | Address | City | Phone | Cuisine | Source/ Row # |
|---|---|---|---|---|---|
| *Arnie morton's of Chicago* | *435 s. la cienega blvd* | *Los Angeles* | *310-246-1501* | *steakhouses* | FODORS 1 |
| *Campanile spice restaurant* | *624 s. la brea ave.* | *los angeles* | *213-938-1447* | *californian* | FODORS 2 |
| *dynasty room* | *930 hilgard ave.* | *Los Angeles* | *310-208-8765* | *continental* | FODORS 3 |
| *pinot hollywood* | *1448 n. gower st.* | *los angeles* | *213-461-8800* | *californian* | FODORS 4 |
| *le dome* | *8720 sunset blvd.* | *Los Angeles* | *310-659-6919* | *french* | FODORS 5 |
| *Arnie morton's of Chicago* | *435 s. la cienega boulevard* | *Los Angeles* | *310/246-1501* | *American* | ZAGAT 1 |
| *Campanile spices* | *624 s. la brea ave.* | *los angeles* | *213/938-1447* | *american* | ZAGAT 2 |
| *the mandarin* | *430 n. camden dr.* | *los angeles* | *310/859-0926* | *asian* | ZAGAT 3 |

*Table 3.2: Sample data for a query of "Los Angeles" from FODORS and ZAGAT databases*

ii. For each record a Soundex or Metaphone key is generated using the *Name* field.

| Name | **Soundex Code for Name** | Address | City | Phone | Cuisine | Source/ Row # |
|---|---|---|---|---|---|---|
| *Arnie morton's of Chicago* | ***A655*** | *435 s. la cienega blvd* | *Los Angeles* | *310-246-1501* | *steakhouses* | FODORS 1 |
| *Campanile spice restaurant* | ***C515*** | *624 s. la brea ave.* | *los angeles* | *213-938-1447* | *californian* | FODORS 2 |
| *dynasty room* | ***D523*** | *930 hilgard ave.* | *Los Angeles* | *310-208-8765* | *continental* | FODORS 3 |
| *pinot hollywood* | ***P534*** | *1448 n. gower st.* | *los angeles* | *213-461-8800* | *californian* | FODORS 4 |
| *le dome* | ***L350*** | *8720 sunset blvd.* | *Los Angeles* | *310-659-6919* | *french* | FODORS 5 |
| *Arnie morton's of Chicago* | ***A655*** | *435 s. la cienega boulevard* | *Los Angeles* | *310/246-1501* | *American* | ZAGAT 1 |
| *Campanile spices* | ***C515*** | *624 s. la brea ave.* | *los angeles* | *213/938-1447* | *american* | ZAGAT 2 |
| *the mandarin* | ***T553*** | *430 n. camden dr.* | *los angeles* | *310/859-0926* | *asian* | ZAGAT 3 |

*Table 3.3: Sample data for a query of "Los Angeles" with Soundex value calculated for Name field*

iii. Sort all the records in the dataset using the generated key.

| Name | **Soundex Code for Name** | Address | City | Phone | Cuisine | Source/ Row # |
|---|---|---|---|---|---|---|
| *Arnie morton's of Chicago* | ***A655*** | *435 s. la cienega blvd* | *Los Angeles* | *310-246-1501* | *steakhouses* | FODORS 1 |
| *Arnie morton's of Chicago* | ***A655*** | *435 s. la cienega boulevard* | *Los Angeles* | *310/246-1501* | *American* | ZAGAT 1 |
| *Campanile spice restaurant* | ***C515*** | *624 s. la brea ave.* | *los angeles* | *213-938-1447* | *californian* | FODORS 2 |
| *Campanile spices* | ***C515*** | *624 s. la brea ave.* | *los angeles* | *213/938-1447* | *american* | ZAGAT 2 |
| *dynasty room* | ***D523*** | *930 hilgard ave.* | *Los Angeles* | *310-208-8765* | *continental* | FODORS 3 |
| *le dome* | ***L350*** | *8720 sunset blvd.* | *Los Angeles* | *310-659-6919* | *french* | FODORS 5 |
| *pinot hollywood* | ***P534*** | *1448 n. gower st.* | *los angeles* | *213-461-8800* | *californian* | FODORS 4 |
| *the mandarin* | ***T553*** | *430 n. camden dr.* | *los angeles* | *310/859-0926* | *asian* | ZAGAT 3 |

*Table 3.4: Sample data for a query of "Los Angeles" sorted on Soundex value*

iv. If there are more than two records with the same Soundex key, then those records are sent to potential duplicate set otherwise the record is moved to the non-duplicate set.

| Name | Address | City | Phone | Cuisine | Source/ Row # |
|---|---|---|---|---|---|
| *Arnie morton's of Chicago* | *435 s. la cienega blvd* | *Los Angeles* | *310-246-1501* | *steakhouses* | FODORS 1 |
| *Arnie morton's of Chicago* | *435 s. la cienega boulevard* | *Los Angeles* | *310/246-1501* | *American* | ZAGAT 1 |
| *Campanile spice restaurant* | *624 s. la brea ave.* | *los angeles* | *213-938-1447* | *californian* | FODORS 2 |
| *Campanile spices* | *624 s. la brea ave.* | *los angeles* | *213/938-1447* | *american* | ZAGAT 2 |

*Table 3.5: Sample data for a query of "Los Angeles" grouped into potential duplicates*

| Name | Address | City | Phone | Cuisine | Source/ Row # |
|---|---|---|---|---|---|
| *dynasty room* | *930 hilgard ave.* | *Los Angeles* | *310-208-8765* | *continental* | FODORS 3 |
| *le dome* | *8720 sunset blvd.* | *Los Angeles* | *310-659-6919* | *french* | FODORS 5 |
| *pinot hollywood* | *1448 n. gower st.* | *los angeles* | *213-461-8800* | *californian* | FODORS 4 |
| *the mandarin* | *430 n. camden dr.* | *los angeles* | *310/859-0926* | *asian* | ZAGAT 3 |

*Table 3.6: Sample data grouped into non-duplicates based on the Soundex value*

v. Subsequent processing of calculating similarity vectors is done, which is explained in detail in the following section

## 3.4 Similarity Vector Calculation

The next step is the similarity vector calculation which holds comparison of two records. Inputs to this process are the potential duplicate dataset and non-duplicate dataset (outputs of the blocking classifier). The output of a similarity vector function is a set of attribute similarity scores for each pair of records in the dataset. In this step, the UDD algorithm calculates the similarity of record pairs in both datasets grouped by the blocking classifier. The output of this process serves as input to Weighted Component Similarity Summing (WCSS) Classifier and SVM classifier which are examined in detail in the following sections.

Similarity vector calculation involves comparing each field in a record to the corresponding field in another record. We represent a pair of records $S_{12}$= {$r_1$, $r_2$}, where $r_1$and $r_2$ can come from the same or different data sources, as a similarity vector $V_{12}$ = <$v_1$, $v_2$, .....$v_n$>, in which $v_i$ represents the $i^{th}$ field similarity between $r_1$ and $r_2$:0 <= $v_i$<= 1. $V_i$ =1 means that the $i^{th}$ fields of $r_1$ and $r_2$ are equal and $v_i$ = 0 means that the $i^{th}$ fields of $r_1$ and $r_2$ are totally different.

UDD can employ any similarity function (one or multiple) to calculate the field similarity (some of these are explained later in this section).The similarity calculation quantifies the similarity between a pair of record fields. As the query results from the database are stored in string format, this thesis is limited to only string similarity.

Given a pair of strings ($S_a$, $S_b$) , a similarity function calculates the similarity score between $S_a$ and $S_b$, which must be between 0 and 1. In my experiments, a transformation based similarity calculation [49] method is adapted in which, given two strings $S_a$ = {$t_{a1}$, $t_{a2}$, ..... $t_{am}$}and $S_b$= {$t_{b1}$, $t_{b2}$, .... $t_{bn}$ } containing a set of tokens, a string transformation from $S_a$ to $S_b$ is a sequence of operations that transforms the tokens of $S_a$ to tokens of $S_b$.

For example let's consider two records, one from *ZAGAT* and another from *FODORS.*

| ***Restaurant Name*** | ***Address*** | ***City*** | ***Phone*** | ***Cuisine*** | ***Database*** |
|---|---|---|---|---|---|
| *Campanile spice restaurant* | *624 s. la brea ave.* | *los angeles* | *213-938-1447* | *californian* | *FODORS* |
| *Campanile spices* | *624 s. la brea ave.* | *los angeles* | *213/938-1447* | *american* | *ZAGAT* |
| *0.5* | *1* | *1* | *1* | *0* | |

*Figure 3.3: Example of Similarity vector calculation*

As can be seen from the above example, for a pair of records each of the fields/attributes are compared individually (e.g. *Restaurant Name with Restaurant Name, City with City* etc.). By doing this we can better establish two record's similarity or lack of it. Using the same above example (figure 3.3) we know the records refer to the same real world entity and the only differentiating attribute is the cuisine value. The similarity vector value for this pair of records would be *<0.5,1,1, 1,0>*. Note that even though the value for phone attribute is stored differently, this was resolved by ignoring text differences (we used a text formatting routine to remove special characters).

Now let's examine how field to filed comparison was done. To calculate the similarity scores between the two field/attribute values first we need to determine the number of transformations that can be performed between two strings.

### 3.4.1 Text Transformations

Below are some of the text transformation types that can be used to compare two words (tokens). Some of these methods are domain-independent and can be applied to all of the attribute values in every application domain and for every attribute. Here is a brief summary of transformation types [49].

- **Equality** tests if two tokens contain the same characters in the same order.
- **Stemming** converts a token into its stem or root. Computes if one token is equal to a continuous subset of the other starting at the first character or last character.
- **Soundex** converts a token into a Soundex code. Tokens that sound similar have the same code.
- **Levenshtein distance** is minimum number of edits needed to transform one string into the other.
- **Metaphone** generates a code when compared between two tokens reveals the similarity
- **Substring** computes if one token is equal to a continuous subset of the other, but does not include the first or last character.
- **Acronym** computes if all characters of one token are initial letters of all tokens from the other object, (e.g. CPK, California Pizza Kitchen).
- **Drop** determines if a token does not match any other token

In order to understand the field to field comparison, let's consider the values of *Restaurant Name* attribute from Fig 3.3:'*Campanile spice restaurant'* and*'Campanile spices'.*

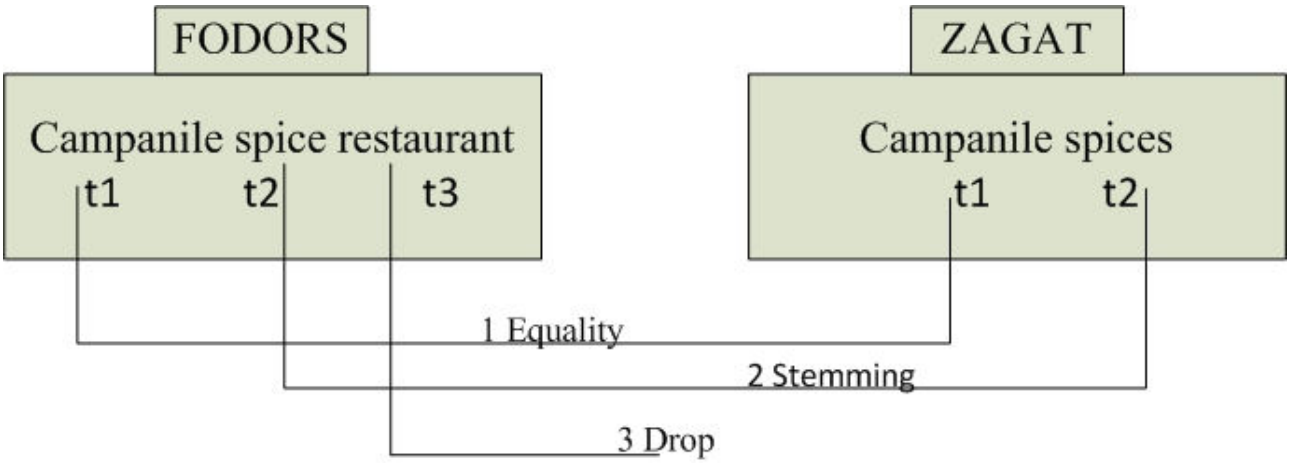



*Figure 3.4: Example of a transformation*

The process of transformation can be explained by using the above example (fig 3.4).

i. Both the strings are converted to lower case and any special characters are removed.
ii. Split each of the strings into tokens using the white space as delimiter. For e.g. *FODORS's* record will have the following token list (*"Campanile", "spice", "restaurant"*).
iii. Each of the token from *FODORS* record is compared with all the tokens in *ZAGAT's* to determine if text transformations ( as listed above in 3.4.1) exist between tokens.
iv. Once the transformation is determined between the tokens, the transformation count is increased by 1 and the tokens are deleted.
v. In our example equality transformation exists for the first token of *FODORS's* record and first token of *ZAGAT's* record as they are same. The stemming is performed on the second tokens of both the records. As token 3("*restaurant"*) in *FODORS* record doesn't exist in *ZAGAT's*, drop function is performed (add 1 to transformation count).
vi. The total number of transformations is calculated. In our example that would be a total of 3 transformations.

### 3.4.2 Computing Attribute Similarity Scores

Once the number of transformations is obtained, we can apply this count for calculating the attribute similarity scores. Here we use the cosine measure [27] (as discussed in Chapter 2) which is commonly used in information retrieval engines with the *tf-idf* (Term Frequency- Inverse Document Frequency is a statistical measure to measure importance of a word to a document in a collection) [50] weighting scheme to calculate the similarity of each of the attribute value. Because the attribute values of the object are very short, term frequency weighting is binary. The document/term frequency is 1 if the term exists in the attribute value and 0 otherwise.

The similarity score for a pair of attribute values is computed using this attribute similarity formula:

$$Similarity(A,B) = \frac{\sum_{i=1}^{t}(w_{ia} * w_{ib})}{\sqrt[2]{(\sum_{i=1}^{t} {w_{ia}}^2) * (\sum_{i=1}^{t} {w_{ib}}^2)}}$$

*Equation 3.1: Similarity of two attribute values A, B*

Where *A* and *B* are two attribute values for which the similarity is calculated

*t* is the number/count of transformations

*weight of token i in attribute value a* ($w_{ia}$) $= (0.5 + 0.5freq_{ia}) * IDF_i$

*weight of token i in attribute value b* ($w_{ib}$) $= freq_{ib} * IDF_i$

*frequency of token i in attribute value a* $= freq_{ia}$

*frequency of token i in attribute value b* $= freq_{ib}$

*IDF (Inverse Document Frequency) of token i in the entire collection*= $IDF_i$ *(i.e., if token i exists in any of the two attribute values its value is -1).*

The terms $w_{ia}$ and $w_{ib}$ correspond to the weights computed by the *tf-idf* weighting function. This function outputs set of attribute similarity scores for the pair of records that are being compared (as in table 3.7). These values are sent to the next function (Classifier 1: Weighted Component Similarity Summing (WCSS) Classifier- discussed in the next section) to calculate the total object similarity score as a weighted sum of the attribute similarity scores.

We call a similarity vector (example table 3.7) formed by potential duplicate record pair as potential duplicate vector and a similarity vector formed by a non-duplicate record pair(example table 3.8) as non-duplicate vector. Given the non-duplicate vector set *N*, our goal is to try to identify the set of actual duplicate vectors *D* from the potential duplicate vector set *P*.

Using this approach similarity vectors are generated by comparing the record pairs in the potential duplicate and non-duplicate dataset.

Below is an example of similarity vectors for the sample records.

| *Name* | *Address* | *City* | *Phone* | *Cuisine* | *Source/ Row #* | *Source/ Row #* |
|---|---|---|---|---|---|---|
| *1.000* | *0.8000* | *1.000* | *1.000* | *0.000* | *FODORS/1* | *ZAGAT/1* |
| *0.5* | *1.000* | *1.000* | *1.000* | *0.000* | *FODORS/2* | *ZAGAT/2* |

*Table 3.7: Similarity vector of potential duplicates for records in table 3.5*

| *Name* | *Address* | *City* | *Phone* | *Cuisine* | *Source/ Row #* | *Source/ Row #* |
|---|---|---|---|---|---|---|
| *0.000* | *0.000* | *1.000* | *0.000* | *0.000* | *FODORS/3* | *FODORS/4* |
| *0.000* | *0.000* | *1.000* | *0.000* | *0.000* | *FODORS/3* | *FODORS/5* |
| *0.000* | *0.000* | *1.000* | *0.000* | *0.000* | *FODORS/3* | *ZAGAT/3* |
| *0.000* | *0.000* | *1.000* | *0.000* | *0.000* | *FODORS/4* | *FODORS/5* |
| *0.000* | *0.145* | *1.000* | *0.000* | *0.000* | *FODORS/4* | *ZAGAT/3* |
| *0.000* | *0.000* | *1.000* | *0.000* | *0.000* | *FODORS/5* | *ZAGAT/3* |

*Table 3.8: Similarity vector of non-duplicates for records in table 3.6*

## 3.5 Classifier 1: Weighted Component Similarity Summing (WCSS) Classifier

Weighted Component Similarity Summing (WCSS) Classifier is the main algorithm of UDD system in identifying duplicates. Inputs to this classifier are the similarity vectors of record pairs from potential duplicates and non-duplicate sets. As we want to develop an unsupervised method there is no training required for this classifier. This classifier tries to find out the duplicates from non-duplicate and potential duplicate datasets (explained in detail through this section). The output from this classifier is a duplicate dataset identified from the potential duplicates and non-duplicate sets. As we have discussed in chapter 1 the weights of the field are also important in trying to decide whether a record is duplicate or not. Before trying to identify the similarity between the records, this classifier determines the weights of the fields which are required for calculation (discussed in section 3.5.1 below).

Once the similarity vectors are found, WCSS and SVM classifiers (explained in section 3.6 below) iterate until no further duplicates are to be found. During this process weights of the fields change with each iteration (discussed in section 3.5.1). The figure 3.5 below is a representation of this process. Here *N* represents the non-duplicate set and *P* represents the potential duplicate set. In the first iteration WCSS classifier (represented by (i) and (iii) in figure 3.5) finds the duplicate pairs in *N* and *P* named as f and *d1* respectively. In the next step dataset *N-f* and *d1+f = d* are sent to SVM classifier (represented by (ii) and (iv) in figure 3.5) to train the system which helps to identify the duplicates from *P-d1*. These duplicates are represented as *d2*. This process repeats until no further duplicates are found in *P* and *N*.

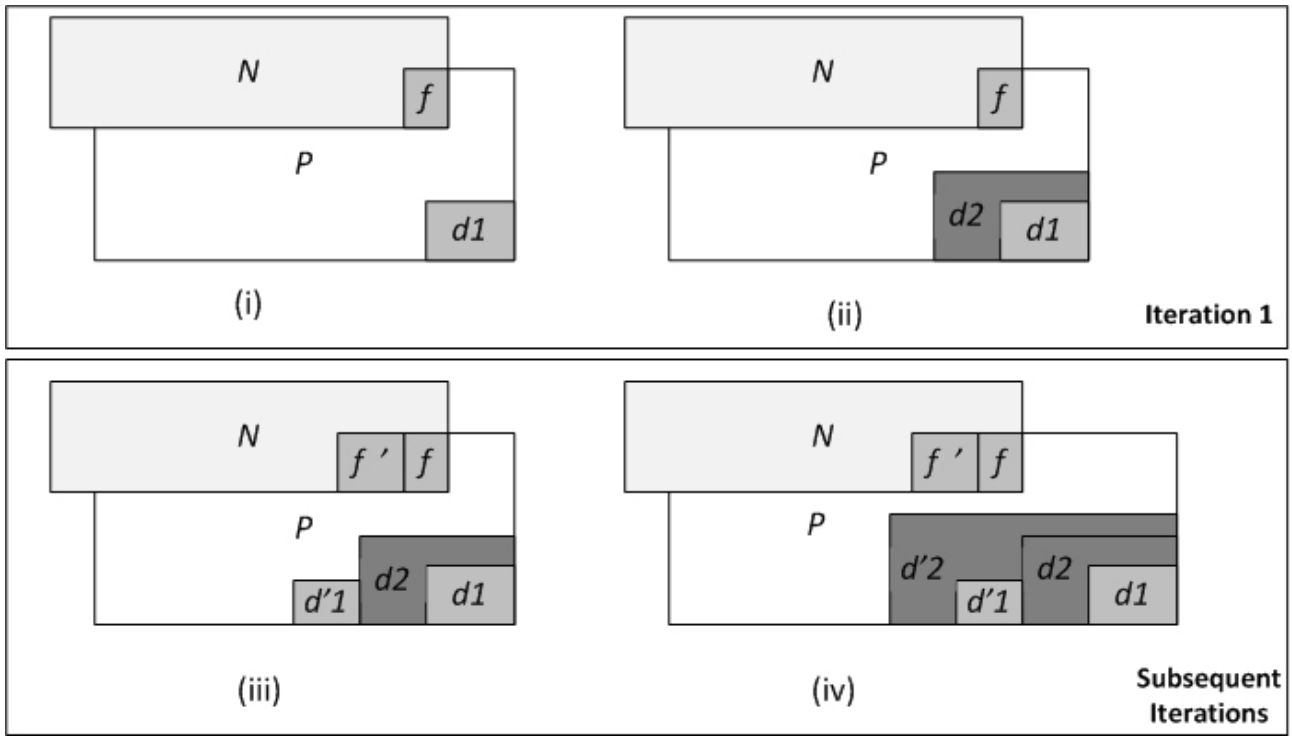



*Figure 3.5: Weighted Component Similarity Summing (WCSS) Classifier*

### 3.5.1 Attribute Weight Assignment

In the WCSS classifier, we assign weight to an attribute to indicate its importance. The weights of a field/attribute are given in such a way that the sum of all fields/attributes weights is equal to 1. In non-duplicate vector most of the fields will have small similarity score (Table 3.8) for all record pairs where as in duplicate vector most of the fields will have large similarity score (values will be similar to Table 3.7) for all record pairs. In general, WCSS classifier employs duplicate and non-duplicate intuitions for assigning weights. Inputs to this function are the similarity vector of non-duplicate records and duplicate records.

**Duplicate Intuition:** For duplicate records the similarity between them should be close to 1. For a duplicate vector $V_{12}$ that is formed by a pair of duplicate records *r1*and *r2*, we need to assign large weights to the fields with large similarity values and small weights to the fields with small similarity values. This will ensure that a record pair with most similarity gets classified as duplicates.

$w_{di} = \frac{p_i}{\sum_{j=1}^{m} p_j}$ // m is the number of fields in dataset

$p_i = \sum_{k=1}^{n} v_{ki}$ // n is the number of records in duplicate vector set

*Equation 3.2: Weight calculation for all the fields using duplicate vectors.*

*Where* $w_{di}$ = Normalized weight for $i^{th}$ attribute

$p_i$ = Accumulated $i^{th}$ attribute similarity value for all duplicate vectors

For each attribute, $p_i$ value will be large when it has a large similarity value in the duplicate vector, which will result in a large weight value being assigned to $i^{th}$ field. On the other hand, the field will be assigned a small weight if it usually has a small similarity value in the duplicate vectors.

**Non-Duplicate Intuition:** In non-duplicate records the similarity between them should be close to 0. Hence, for a non-duplicate vector $V_{12}$ that is formed by a pair of non-duplicate records *r1* and *r2*, we need to assign small weights to the fields with large similarity values and large weights to the fields with small similarity values. This will ensure that a record pair with less similarity gets classified as non-duplicates.

$$w_{ni} = \frac{q_i}{\sum_{j=1}^{m} q_j} \quad \text{// m is the number of fields in dataset}$$

$$q_i = \sum_{k=1}^{n}(1 - v_{ki}) \quad \text{// n is the number of records in non-duplicate vector set}$$

*Equation 3.3: Weight calculation for all the fields using duplicate vectors.*

*Where* $w_{di}$ = Normalized weight for $i^{th}$ attribute

$q_i$ = Accumulated $i^{th}$ attribute dissimilarity value for all non-duplicate vectors

In non-duplicate vectors the dissimilarity value of i$^{th}$ field is *1-*$v_i$ (where vi is the similarity of $i^{th}$ field). For each field, if it usually has a large similarity value in the non-duplicate vectors, it will have a small accumulated dissimilarity ($q_i$) and will, in turn, be assigned a small weight. On the other hand, it will be assigned a large weight if it usually has a small similarity value in the non-duplicate vectors.

For example let's calculate weight for fields '*Restaurant Name'* and '*City*' in non-duplicate vector set from Table 3.8:

$q_{\text{name}} = \sum_{k=1}^{n}(1 - v_{ki})$
= (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0) = 6
$q_{\text{address}}$ = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.145)+(1-0.0) = 5.85
$q_{\text{city}}$ = (1-1)+(1-1)+(1-1)+(1-1)+(1-1)+(1-1) = 0
$q_{\text{phone}}$ = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0) = 6

$q_{\text{cuisine}}$ = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0) = 6

$\sum_{j=1}^{m} q_j$ = 6 + 5.85 + 0 + 6 + 6 = 23.85

$\text{Weight}_{\text{name}}$ = 6/23.85 = .25

$\text{Weight}_{\text{city}}$ = 0/23 = 0

As we can see from above example weight of *city* is 0 because it has high similarity vectors for all the records pairs and as *restaurant name* has low similarity vectors its weight is high. In the same way weights for duplicate records can be calculated (using the Equation 3.2). For the first iteration as weights of the fields are calculated before any duplicates are found by WCSS classifier, the input for this (Attribute weight calculation) function would be only non-duplicate vectors.

The final weight of attribute is the combination of two intuitions weighting schemes. As part of experiments, each scheme was given a weight to show its importance:

$$w_i = a * w_{di} + (1 - a) w_{ni}$$

*Equation 3.4: Weight of all the fields combining duplicate weight and non duplicate weight.*

Where $a \in [0,1]$ denotes the importance of duplicate vectors versus non-duplicate vectors.
The first time UDD algorithm is run there are no identified duplicates. Hence, *a* is assigned to be 0. As more duplicate vectors are discovered, we increase the value of *a*. We initially set *a* to be 0.5 at the 2nd iteration to indicate that duplicates and non-duplicates are equally important and incrementally add 0.1 for each of the subsequent iterations.

## 3.5.2 Duplicate Identification

Once we get the weights of each field and the similarity vectors of non-duplicate and potential duplicate datasets, the duplicate detection can be done by calculating the similarity between the records. Hence, we define the similarity between records as:

$$Similarity(r_1, r_2) = \sum_{i=1}^{n} w_i * v_i$$

*Equation 3.5: WCSS similarity for records $r_1$ and $r_2$.*

Where $r_1$, $r_2$ are the two records for which the similarity is being calculated.

$w_i$ is the weight of field($i$).

$v_i$ is the similarity vector of two records $r_1$, $r_2$ of field($i$) .

Two records $r_1$ and $r_2$ are duplicates if Similarity ($r_1$, $r_2$) >= $T_{sim}$, i.e., if their similarity value is equal to or greater than a similarity threshold (user defined value to indicate duplicates). In general, the similarity threshold $T_{sim}$ should be close to 1 to ensure that the identified duplicates are correct. Increasing the value of $T_{sim}$ will reduce the number of duplicate vectors identified by classifier one while, at the same time, the identified duplicates will be more precise. The influence of $T_{sim}$ on the performance of our algorithm will be illustrated in chapter 4.

## 3.6 Classifier 2: Support Vector Machine (SVM) classifier

As we had seen earlier in chapter 2, SVM [51] classifier is a tool used to classify data. SVM uses a two-step process, training and classification. In the training step, labeled data is supplied to the classifier, labeling each record as either positive or negative. SVM internally plots the information (fig 2.3) by separating the data into groups. During the classification step, when the system is supplied with data to be classified, it classifies the record as either being positive or negative based on the training data.

The WCSS classifier outputs three sets of similarity vectors namely potential duplicate vectors, non-duplicate vectors and identified duplicate vectors. From these vectors, the identified duplicate vectors *D* are sent as positive examples and non-duplicate vectors *N* as negative examples for training purpose. These two vectors serve as input (training data) to the SVM classifier. We can train SVM classifier and use this trained classifier to identify new duplicate vectors from the potential duplicate vector *P*.

### 3.6.1 Why choose SVM classifier?

There are many statistical tools [52] that aid in data classification. The main reason for using the SVM classifier over other classifiers is that SVM classifier is not sensitive to the number of positive or negative examples that it uses as part of training data. This is the case in our algorithm when initially there may not be any identified duplicate records. SVM is suitable to UDD algorithm as the algorithm classifies data in iterative way, any classification due lack of training data might result in wrong classification. As the algorithm progresses through iterations the number of positive examples increase.

As part of this thesis, an implementation of SVM by Thorsten Joachims [53] [54] was used. This is one of the popular implementations of SVM. The SVM$^{light}$ [55] is implemented in C++ and is available as DLL that can be used with any application.

Below is an example of the two step process of how SVM classifier works.

### 3.6.2 Training / Learning:

Training data for SVM classifier is the similarity vectors from duplicates (identified by the WCSS classifier) and non-duplicates. Duplicates are labeled as positive and non-duplicates as negative.

| | | | | |
|---|---|---|---|---|
| +1 1:1.000 | 2:0.8000 | 3:1.000 | 4:1.000 | 5:0.000 |
| +1 1:0.5 | 2:1.000 | 3:1.000 | 4:1.000 | 5:0.000 |
| -1 1:0.000 | 2:0.000 | 3:1.000 | 4:0.000 | 5:0.000 |
| -1 1:0.000 | 2:0.000 | 3:1.000 | 4:0.000 | 5:0.000 |
| -1 1:0.000 | 2:0.000 | 3:1.000 | 4:0.000 | 5:0.000 |
| -1 1:0.000 | 2:0.000 | 3:1.000 | 4:0.000 | 5:0.000 |
| -1 1:0.000 | 2:0.145 | 3:1.000 | 4:0.000 | 5:0.000 |
| -1 1:0.000 | 2:0.000 | 3:1.000 | 4:0.000 | 5:0.000 |

*Table 3.9: Training data for SVM classifier combing entries from table 3.6 and table 3.7*

### 3.6.3 Classification:

Once training is complete, similarity vectors from potential duplicate dataset are sent to the SVM classifiers classify function to determine if they are duplicates. The output from the classify function is either a positive value (duplicate) or negative value (non-duplicate). Next chapter contains details about various experiments that were conducted and how SVM classifier was able to classify data.

# Chapter 4

## 4.1 Tool introduction

As we had seen in the previous chapter there are many components that are needed to make the application work. The technical architecture in figure 3.2 is a representation for the use case in figure 1.1. In this chapter we will look at the technologies that were used to build the UDD application as well as look various experiments that were conducted and finally compare the results to similar applications that used the same dataset.

The UDD application was built using the following technologies.

### 4.1.1 Microsoft Visual Studio

The core of the system was built using the Microsoft Visual Studio (Version 2008). The code was developed in C# .Net, because it was a robust programming language and for the ability to use LINQ[1](.NET Language-Integrated Query). The Visual Studio IDE (Integrated Development Environment) is popular choice among researchers to develop applications primarily for its ability to access data (using drivers) and also provide all tools to deploy applications. The software and the license were obtained from MSDNAA Access[2]

### 4.1.2 Microsoft Access Database

The database for this thesis was the Access database. As the dataset that was used (*restaurant*) was small (total of 864 entries) and native connectivity with Vistula Studio was the reason for choosing the access database.

### 4.1.3 Microsoft Jet 4.0 OLE DB Provider

Visual Studio has built in database connectivity tool known as the Microsoft Jet 4.0 OLE DB provider for connecting to Access database**.**

### 4.1.4 Other components

Other components that were used for this thesis include the following.

- SVM$^{Light}$[3] is a windows implementation of SVM algorithm developed by Thorsten Joachims [56]. There are others variants [57] of SVM implementations, but SVM$^{Light}$ was chosen as it provides easy integration to Visual Studio.

[1] http://msdn.microsoft.com/en-us/library/bb308959.aspx
[2] http://msdn07.e-academy.com/csuci_cs
[3] http://mihagrcar.org/svmlightlib/

- A variation of Metaphone which was proposed by Lawrence Philips [48] and implemented by Adam Nelson [63] was used for calculating the Metaphone value in this thesis.

## 4.2 Introduction to system

The UDD application developed for this thesis was designed with the objective of providing simple interface and has flexibility to make changes to variables that have an impact on the results.
Below is the screen shot of the user interface:

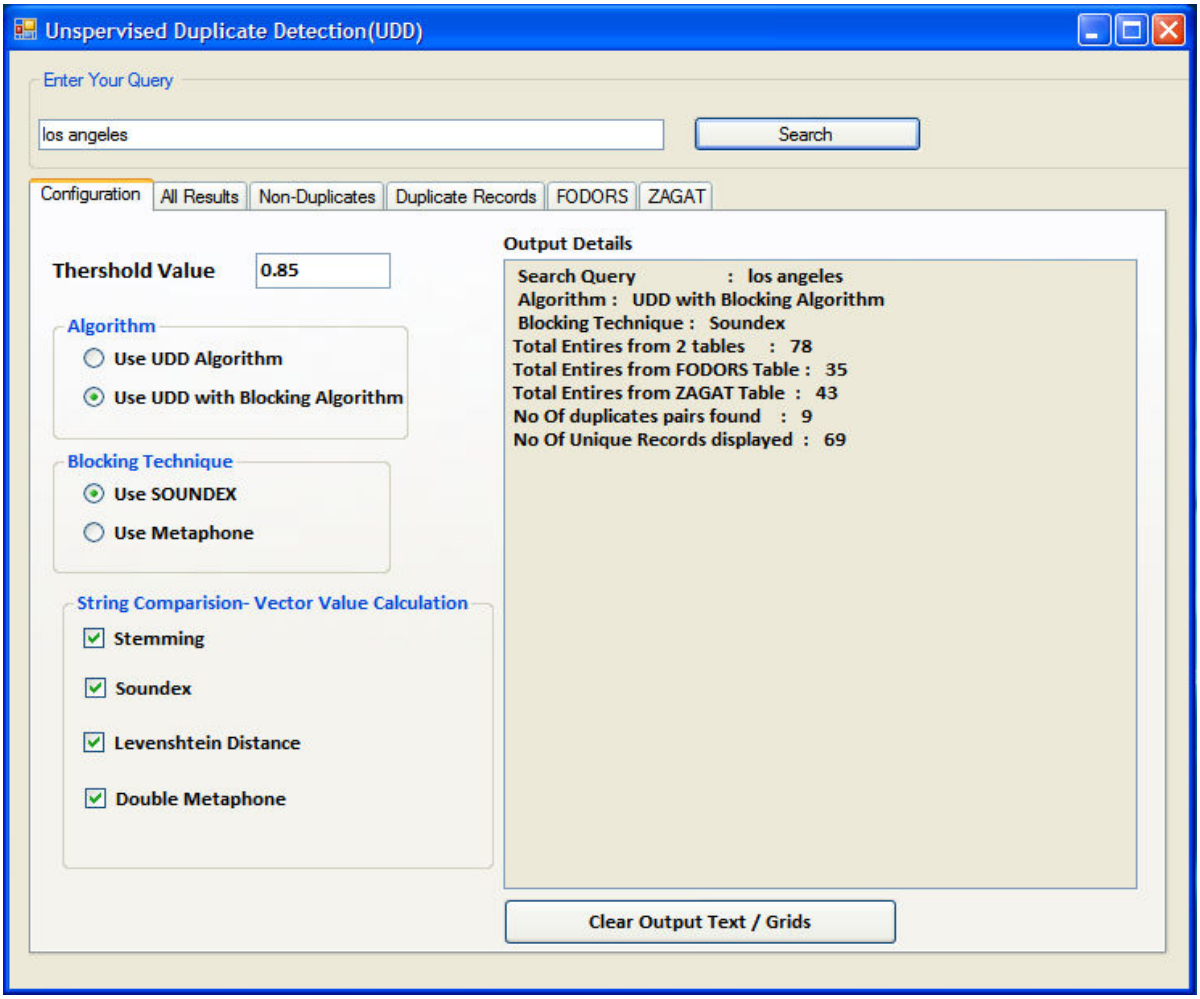



*Figure 4.1: Application User Interface with output details for search query 'los angeles'*

The user interface has several tabs which are explained in detail below.
The *configuration tab* (figure 4.1), as the name implies is the place where one can select/set the parameters for the application. The following options can be selected from the configuration tab.

- Select the algorithm to use – The user has the option to select the standard UDD algorithm or the enhanced UDD algorithm with blocking
- When blocking is selected, we can specify either to use Soundex or Metaphone as the blocking mechanism/technique.

- For both the algorithms one can choose to use the string similarity metrics that were discussed in section 3.4. By default all the metrics are selected.
- Finally, we can set the threshold value that is used by the both the algorithm in the application. We will see later in this chapter, the impact of changing threshold value on the final results.

The application has a textbox that outputs the summary of the results with the option to clear the logs and the information on the other tabs.

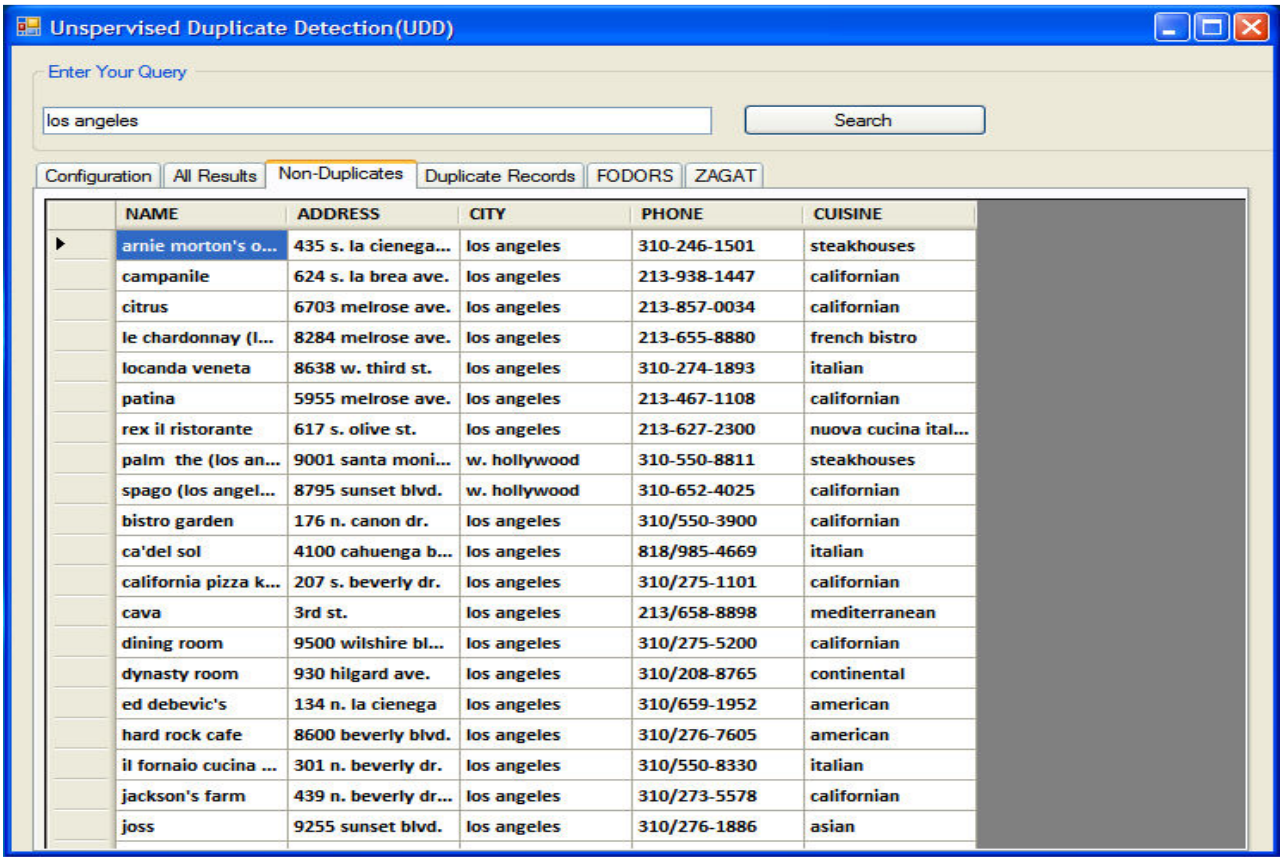
We will briefly look at what information is displayed on the other tabs.

*All Results* contains all the records that were fetched from both the FODORS and ZAGAT tables.

*FODORS* tab has all the records fetched for the query from FODORS table

*ZAGAT* tab has all the records fetched for the query from ZAGAT table

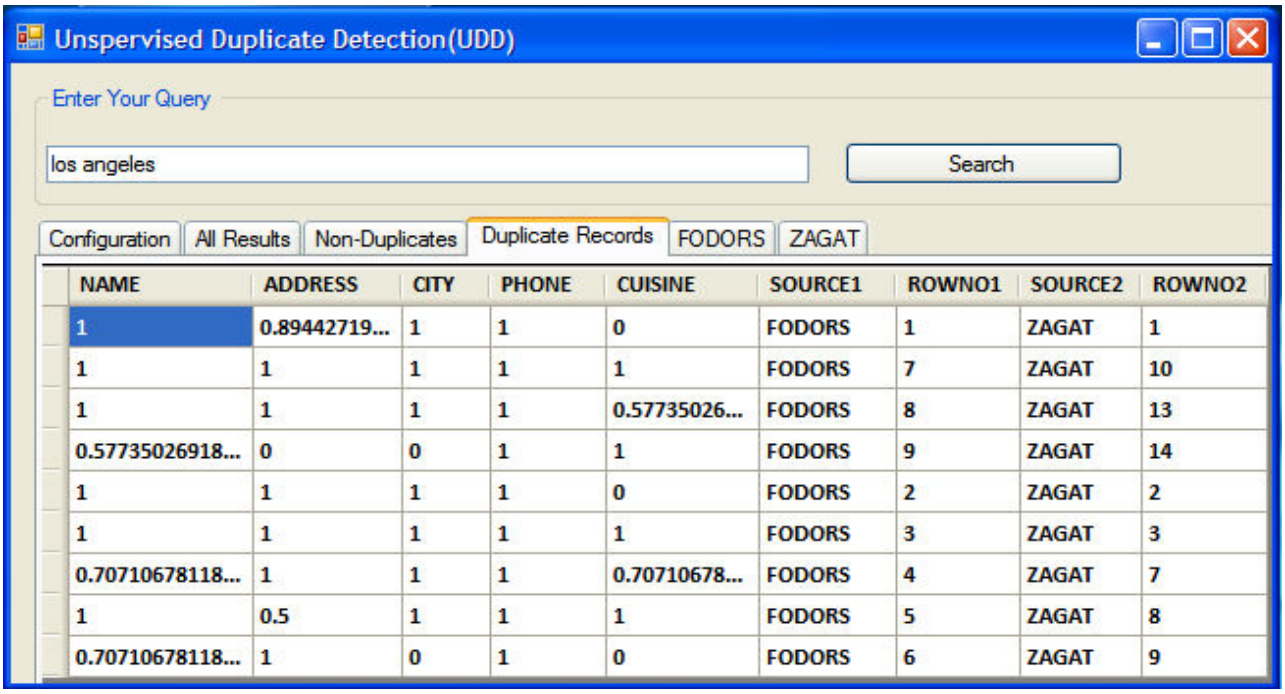
The two tabs that are of importance to us are the *Non-Duplicates* and *Duplicate Records*.




| NAME | ADDRESS | CITY | PHONE | CUISINE |
|---|---|---|---|---|
| arnie morton's o... | 435 s. la cienega... | los angeles | 310-246-1501 | steakhouses |
| campanile | 624 s. la brea ave. | los angeles | 213-938-1447 | californian |
| citrus | 6703 melrose ave. | los angeles | 213-857-0034 | californian |
| le chardonnay (l... | 8284 melrose ave. | los angeles | 213-655-8880 | french bistro |
| locanda veneta | 8638 w. third st. | los angeles | 310-274-1893 | italian |
| patina | 5955 melrose ave. | los angeles | 213-467-1108 | californian |
| rex il ristorante | 617 s. olive st. | los angeles | 213-627-2300 | nuova cucina ital... |
| palm the (los an... | 9001 santa moni... | w. hollywood | 310-550-8811 | steakhouses |
| spago (los angel... | 8795 sunset blvd. | w. hollywood | 310-652-4025 | californian |
| bistro garden | 176 n. canon dr. | los angeles | 310/550-3900 | californian |
| ca'del sol | 4100 cahuenga b... | los angeles | 818/985-4669 | italian |
| california pizza k... | 207 s. beverly dr. | los angeles | 310/275-1101 | californian |
| cava | 3rd st. | los angeles | 213/658-8898 | mediterranean |
| dining room | 9500 wilshire bl... | los angeles | 310/275-5200 | californian |
| dynasty room | 930 hilgard ave. | los angeles | 310/208-8765 | continental |
| ed debevic's | 134 n. la cienega | los angeles | 310/659-1952 | american |
| hard rock cafe | 8600 beverly blvd. | los angeles | 310/276-7605 | american |
| il fornaio cucina ... | 301 n. beverly dr. | los angeles | 310/550-8330 | italian |
| jackson's farm | 439 n. beverly dr... | los angeles | 310/273-5578 | californian |
| joss | 9255 sunset blvd. | los angeles | 310/276-1886 | asian |

*Figure 4.2: Non-duplicate(Unique) records for search query 'los angeles'.*

The *Non-Duplicates* tab (figure 4.2) contains the unique records for the search query.




| NAME | ADDRESS | CITY | PHONE | CUISINE | SOURCE1 | ROWNO1 | SOURCE2 | ROWNO2 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.89442719... | 1 | 1 | 0 | FODORS | 1 | ZAGAT | 1 |
| 1 | 1 | 1 | 1 | 1 | FODORS | 7 | ZAGAT | 10 |
| 1 | 1 | 1 | 1 | 0.57735026... | FODORS | 8 | ZAGAT | 13 |
| 0.57735026918... | 0 | 0 | 1 | 1 | FODORS | 9 | ZAGAT | 14 |
| 1 | 1 | 1 | 1 | 0 | FODORS | 2 | ZAGAT | 2 |
| 1 | 1 | 1 | 1 | 1 | FODORS | 3 | ZAGAT | 3 |
| 0.70710678118... | 1 | 1 | 1 | 0.70710678... | FODORS | 4 | ZAGAT | 7 |
| 1 | 0.5 | 1 | 1 | 1 | FODORS | 5 | ZAGAT | 8 |
| 0.70710678118... | 1 | 0 | 1 | 0 | FODORS | 6 | ZAGAT | 9 |

*Figure 4.3: Duplicate Records tab showing the similarity vector values of duplicate records.*

The *Duplicate Records* tab contains the similarity vectors of duplicate records that were identified by the UDD algorithm. As can be seen it has the similarity vector values of two records from the two tables. *Source* columns related to the source table and *row number* column refers to the record number of the dataset that was fetched from the source table.

## 4.3 Evaluation Metric

Most of the duplicate detection approaches use precision, recall and f-measure [58] to measure the performance of the algorithm, which are defined as follows:

1. Recall, which is the fraction of duplicates correctly classified over the total number of duplicates in the dataset. The formula is:

$$Recall = \frac{correctly\ identified\ duplicate\ pairs}{true\ duplicate\ pairs}$$

2. Precision, which is the fraction of correct duplicates over the total number of record pairs classified as duplicates by the system. The formula is:

$$Precision = \frac{correctly\ identified\ duplicate\ pairs}{Number\ of\ duplicate\ pairs\ found}$$

3. F-measure or F-Score, which is the harmonic mean of the precision and recall values and is given by:

$$f - meausre = \frac{(2 * Precision * Recall)}{(Precision + Recall)}$$

## 4.4 Experiments

To analyze the effectiveness of the UDD algorithm in identifying duplicates various experiments were conducted using the *restaurant* dataset. From the original dataset [59] we know that there are 112 identified duplicates pairs. The application that was developed for this thesis includes various configuration options (as described in section 4.2) that need to be tested and results validated.

Experiments were divided into two categories, one involved querying using random words (experiments 1-6) and other is the analysis of the complete *restaurant* dataset (experiment 7).

### 4.4.1 Random query experiments

**Experiment 1:** Let's consider a search query = "*new york"* using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value be at *0.85*.

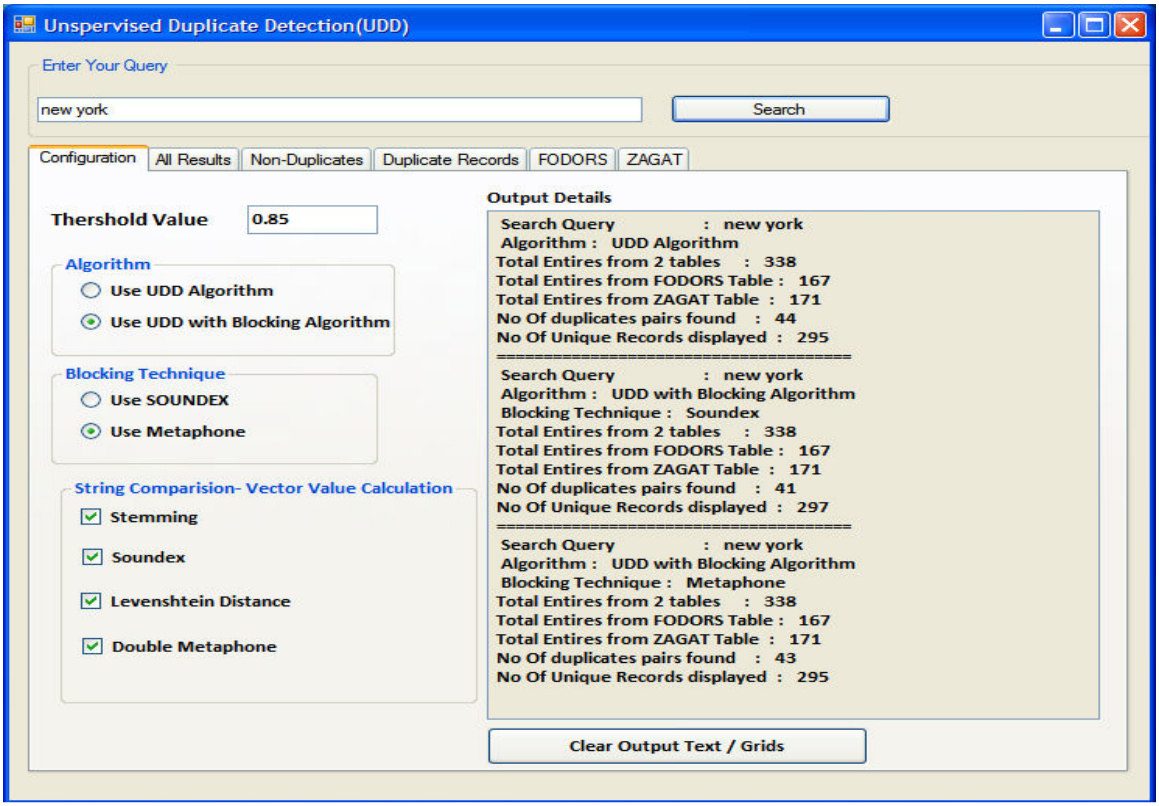



*Figure 4.4: Experiment 1 for search query 'new york' output details of all algorithms*

As can be seen from the above figure 4.4, we can derive the following evaluation metrics.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 44 | 40 | 43 | 0.909 | 0.930 | 0.919 |
| UDD with Soundex | 41 | 40 | 43 | 0.975 | 0.930 | 0.952 |
| UDD with Metaphone | 43 | 40 | 43 | 0.930 | 0.930 | 0.930 |

*Table 4.1: Precision, Recall and f-measure for search query 'new york'*

As we can see from the above table or the below graph precision, recall and *f*-measure are better in blocking algorithm using Soundex than the standard UDD algorithm. It is also observed that blocking with Metaphone works better than the standard UDD algorithm.

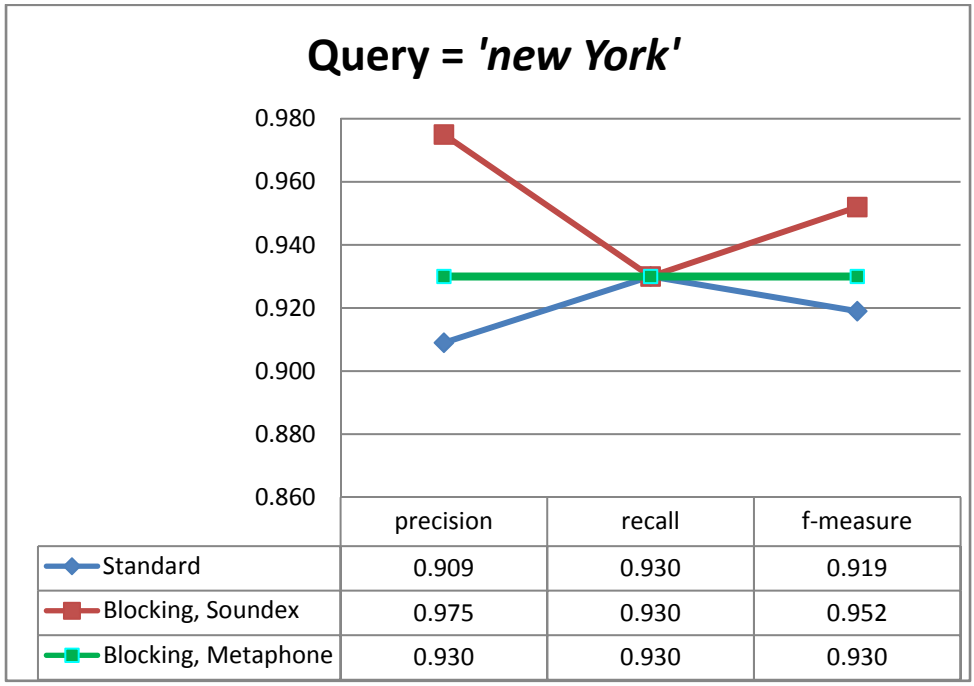



*Figure 4.5: Graph showing the evaluation metric details for all three algorithms*

**Experiment 2:** Now let's consider search query = "*america*" using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value remain at *0.85*.

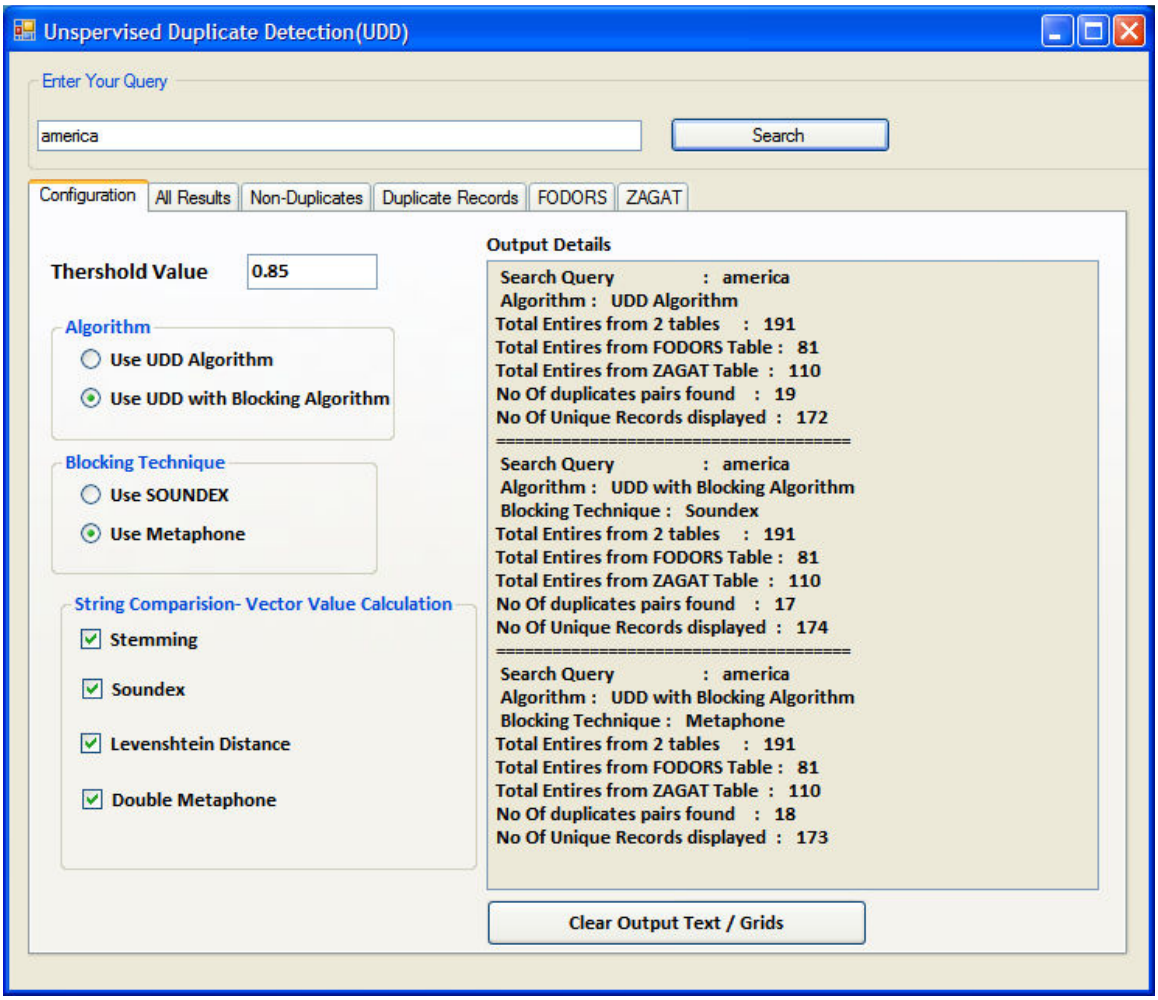


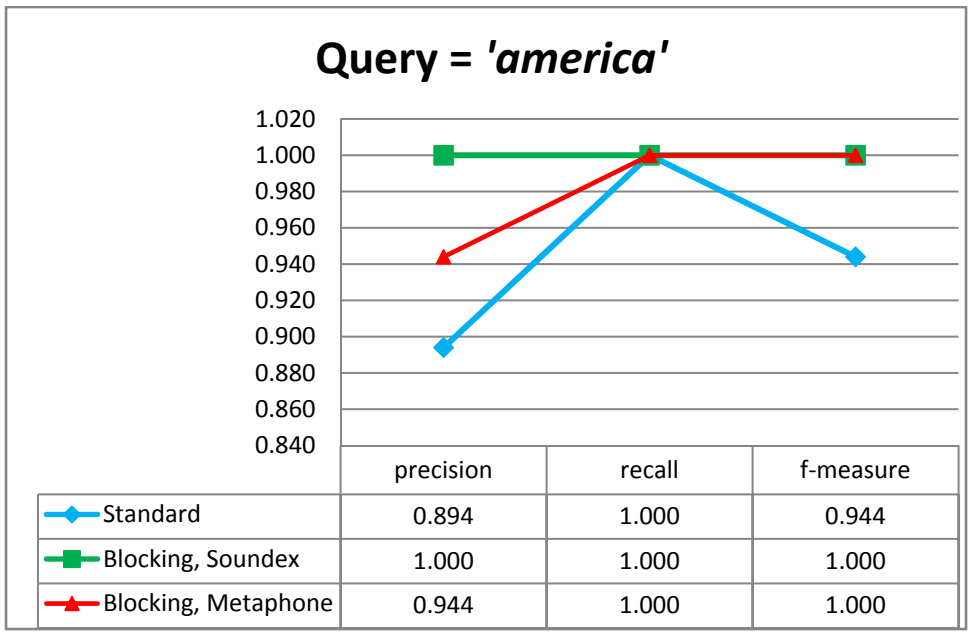

*Figure 4.6: Experiment 2 for search query 'america' output details of all algorithms*

As can be seen from the above figure 4.5, we can derive the following evaluation metrics.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 19 | 17 | 17 | 0.894 | 1.000 | 0.944 |
| UDD with Soundex | 17 | 17 | 17 | 1.000 | 1.000 | 1.000 |
| UDD with Metaphone | 18 | 17 | 17 | 0.944 | 1.000 | 1.000 |

*Table 4.2: Precision, Recall and f-measure for search query 'america'*

For this experiment, all the metrics -precision, recall and *f*-measure are better in the UDD with blocking algorithm as compared to the standard UDD algorithm.




| | precision | recall | f-measure |
|---|---|---|---|
| Standard | 0.894 | 1.000 | 0.944 |
| Blocking, Soundex | 1.000 | 1.000 | 1.000 |
| Blocking, Metaphone | 0.944 | 1.000 | 1.000 |

*Figure 4.7: Graph showing the evaluation metric details for all three algorithms*

**Experiment 3:** Now let's consider search query = "*los angeles"* using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value remain at *0.85*.

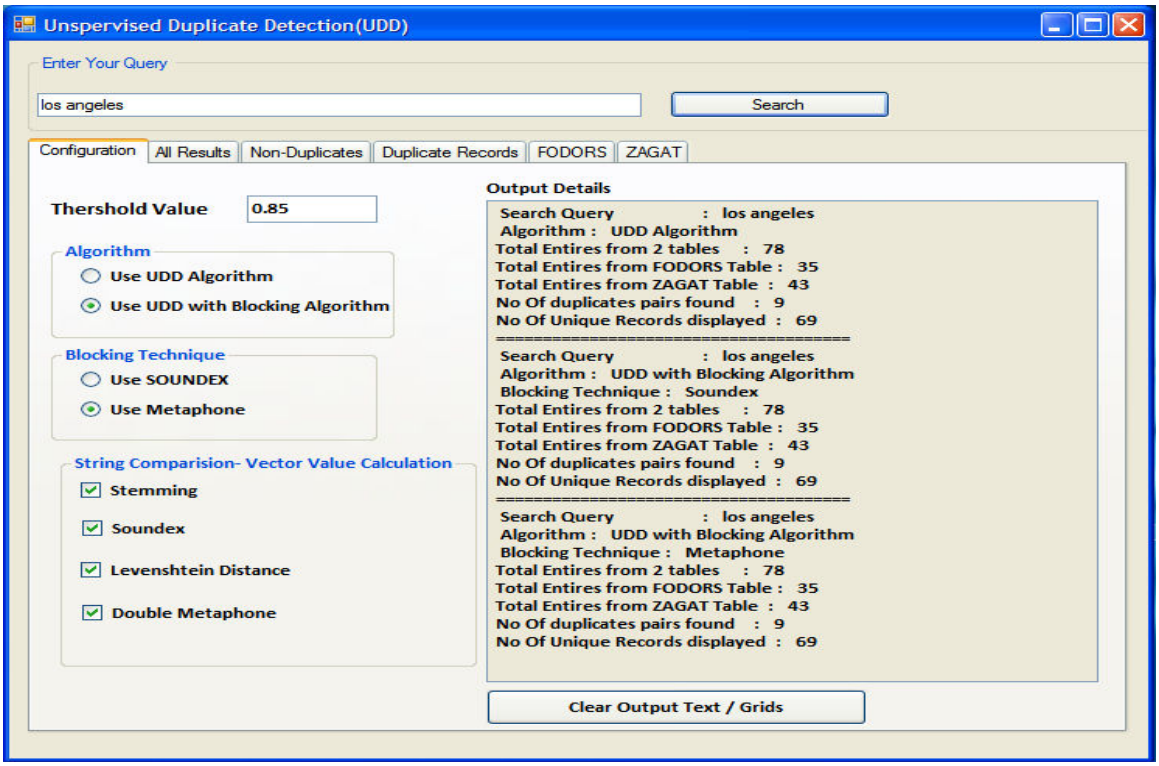



*Figure 4.8: Experiment 3 for search query 'los angeles' output details of all algorithms*

As can be seen from the above figure 4.8, we can derive the following evaluation metrics.

| | Number of Duplicate pairs | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 9 | 9 | 9 | 1.000 | 1.000 | 1.000 |
| UDD with Soundex | 9 | 9 | 9 | 1.000 | 1.000 | 1.000 |
| UDD with Metaphone | 9 | 9 | 9 | 1.000 | 1.000 | 1.000 |

*Table 4.3: Precision, Recall and f-measure for search query 'los angeles'*

For this experiment, both the algorithms, UDD and UDD with blocking have the same metrics - precision, recall and *f*-measure.

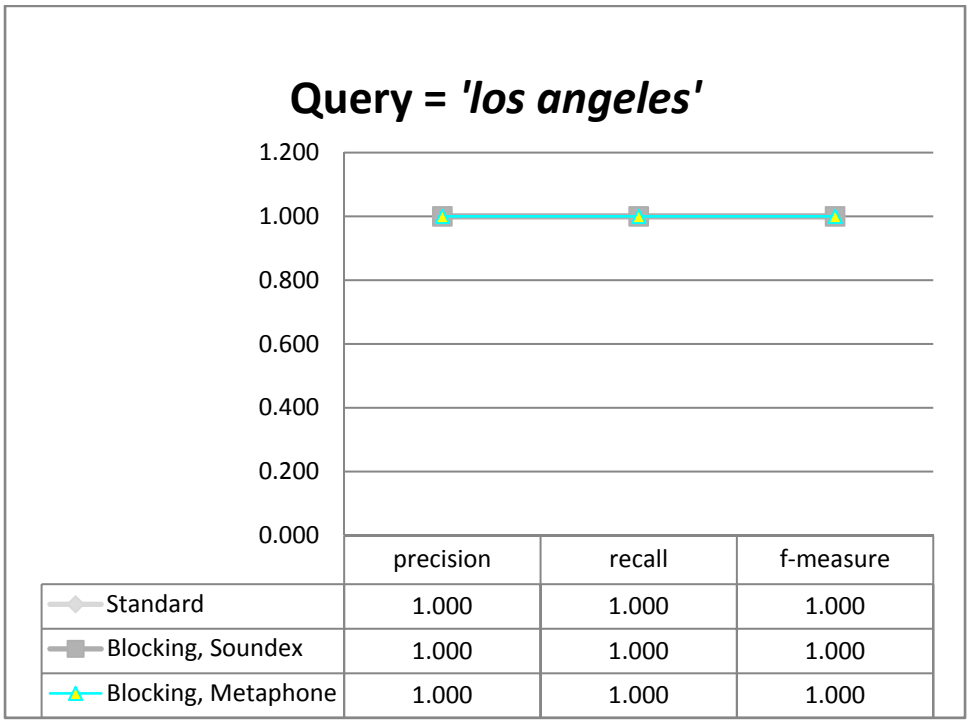



*Figure 4.9: Graph showing the evaluation metric details for all three algorithms*

**Experiment 4:** Let's consider another search query = "*cafe"* using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value be at *0.85*

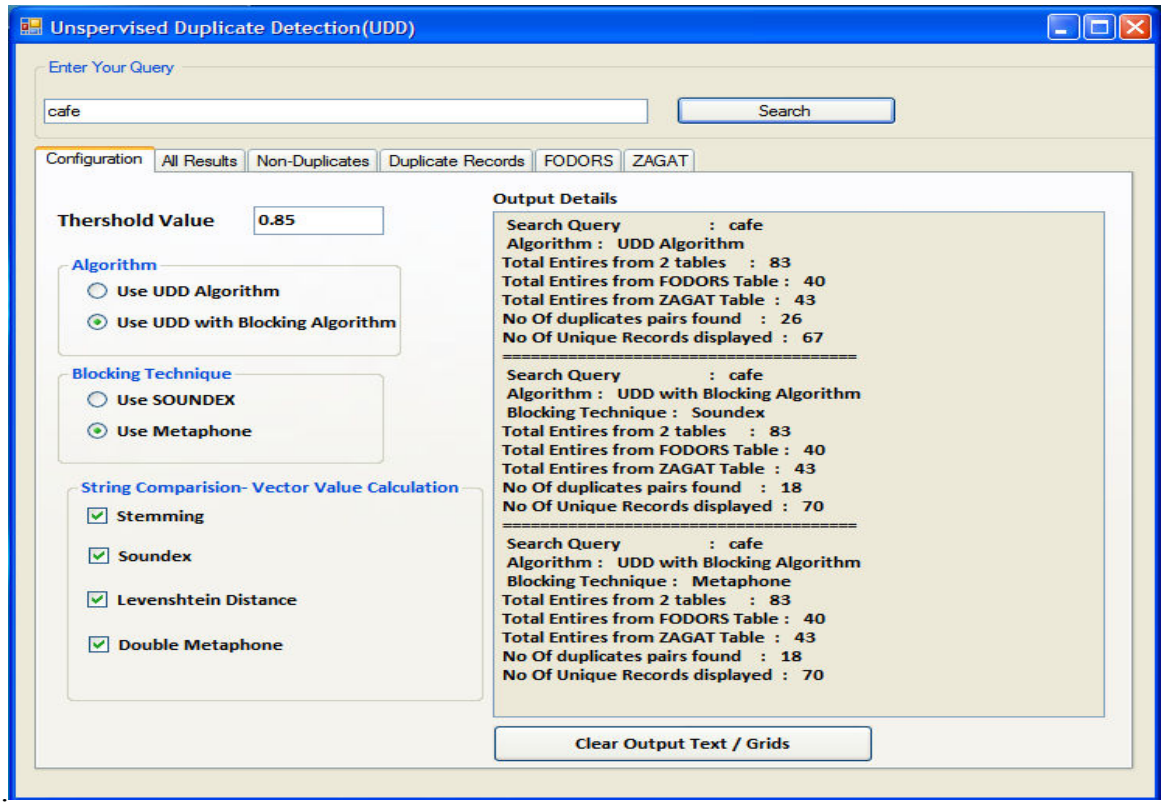


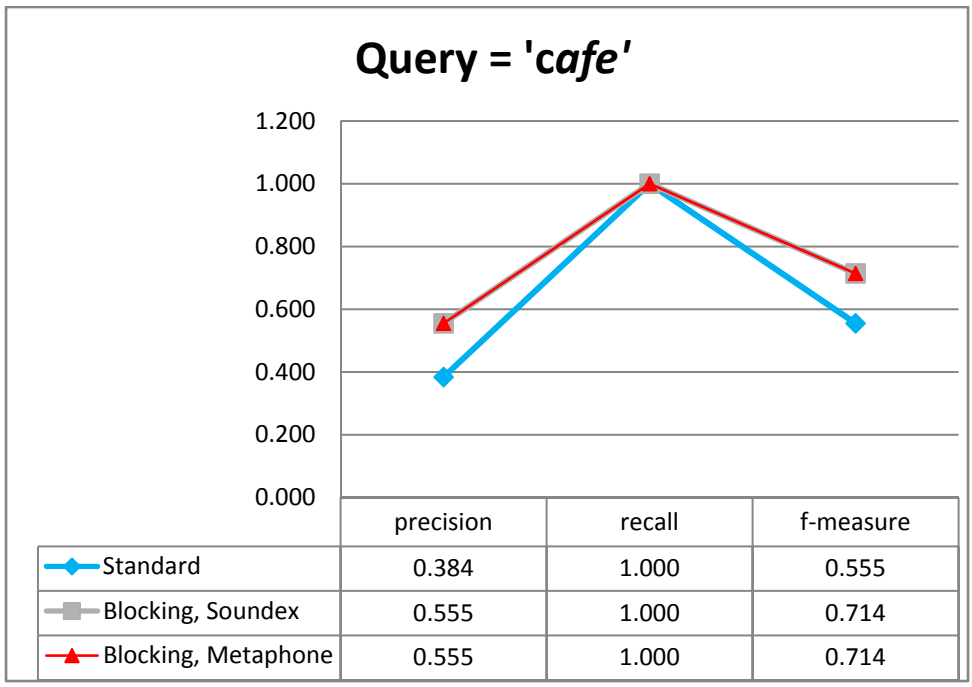

*Figure 4.10: Experiment 4 for search query 'cafe' output details of all algorithms*

As can be seen from the above figure 4.10, we can derive the following evaluation metrics.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 26 | 10 | 10 | 0.384 | 1.000 | 0.555 |
| UDD with Soundex | 18 | 10 | 10 | 0.555 | 1.000 | 0.714 |
| UDD with Metaphone | 18 | 10 | 10 | 0.555 | 1.000 | 0.714 |

*Table 4.4: Precision, Recall and f-measure for search query 'cafe'*

As we can see from the above table or the below graph the precision, recall and *f*-measure are better in blocking algorithm using Soundex or Metaphone than the standard UDD algorithm.




| | precision | recall | f-measure |
|---|---|---|---|
| Standard | 0.384 | 1.000 | 0.555 |
| Blocking, Soundex | 0.555 | 1.000 | 0.714 |
| Blocking, Metaphone | 0.555 | 1.000 | 0.714 |

*Figure 4.11: Graph showing the evaluation metric details for all three algorithms*

**Experiment 5:** Let's consider another search query = "*san francisco"* using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value be at *0.85*.

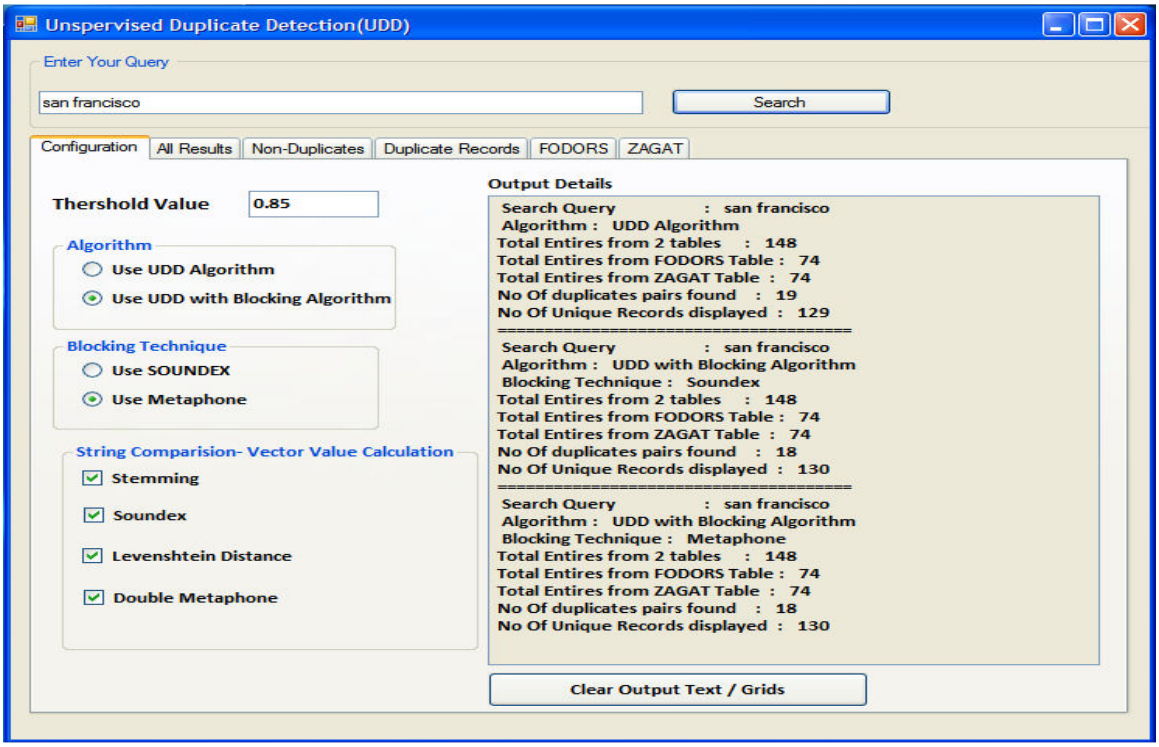



*Figure 4.12: Experiment 5 for search query 'san francisco' output details of all algorithms*

As can be seen from the above figure 4.12, we can derive the following evaluation metrics.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 19 | 18 | 18 | 0.947 | 1.000 | 0.972 |
| UDD with Soundex | 18 | 17 | 18 | 0.944 | 0.944 | 0.944 |
| UDD with Metaphone | 18 | 17 | 18 | 0.944 | 0.944 | 0.944 |

*Table 4.5: Precision, Recall and f-measure for search query 'san francisco'*

As we can see from the above table or below graph the precision, recall and *f*-measure are better in the standard UDD algorithm when compared to UDD with blocking algorithm.

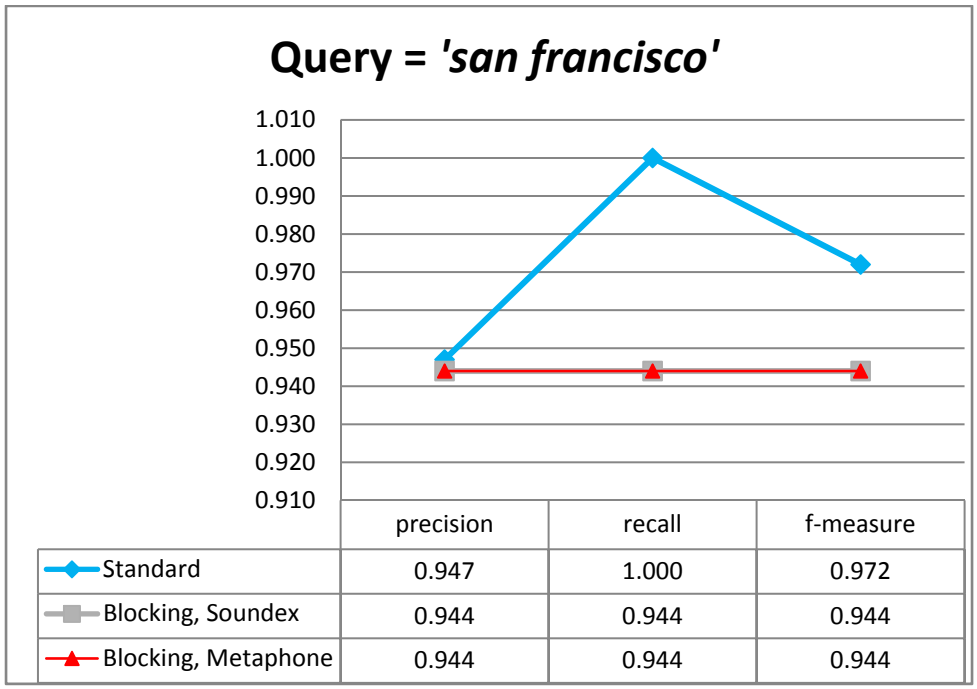



*Figure 4.13: Graph showing the evaluation metric details for all three algorithms*

**Experiment 6:** Finally let's consider a search query = "*french*" using the standard UDD algorithm and UDD algorithm with blocking techniques (Soundex and Metaphone). By default we also chose all the string comparison functions (Stemming, Soundex etc) and let the threshold value be at *0.85*.

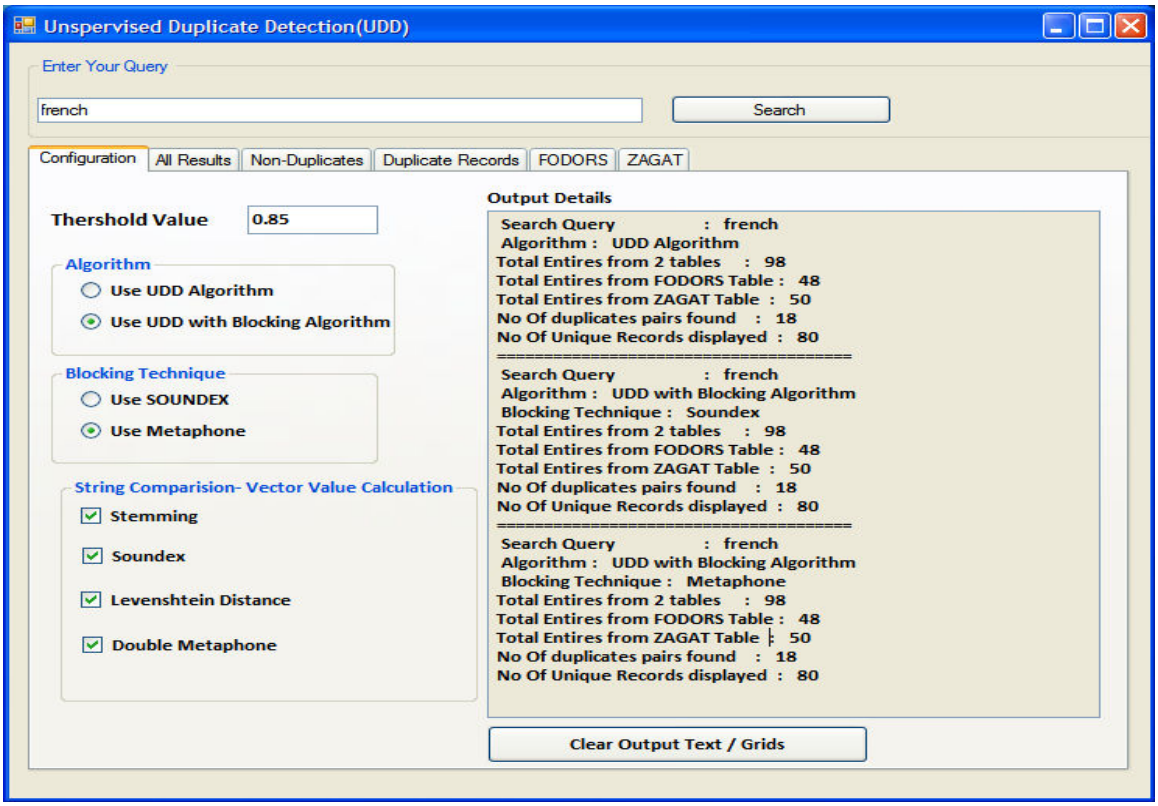


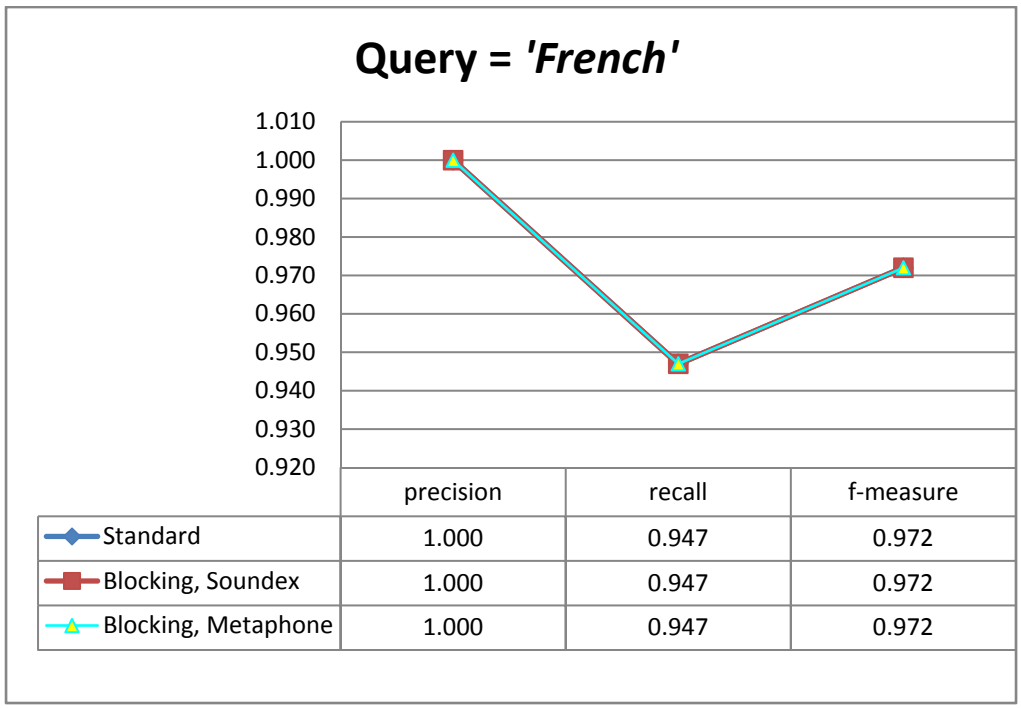

*Figure 4.14: Experiment 6 for search query 'french' output details of all algorithms*

As can be seen from the above figure 4.4, we can derive the following evaluation metrics.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 18 | 18 | 19 | 1.000 | 0.947 | 0.972 |
| UDD with Soundex | 18 | 18 | 19 | 1.000 | 0.947 | 0.972 |
| UDD with Metaphone | 18 | 18 | 19 | 1.000 | 0.947 | 0.972 |

*Table 4.6: Precision, Recall and f-measure for search query 'french'*

As we can see from the above table or below graph the precision, recall and *f*-measure are the same for the both the UDD algorithms.



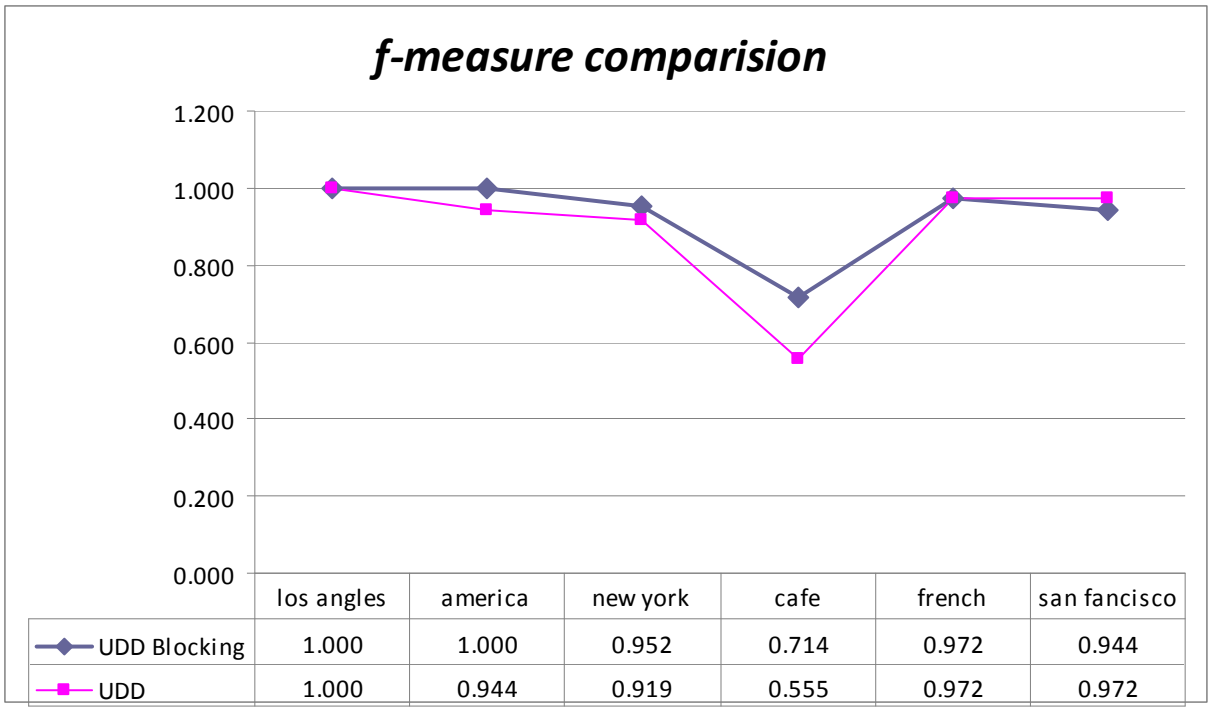

| | precision | recall | f-measure |
|---|---|---|---|
| Standard | 1.000 | 0.947 | 0.972 |
| Blocking, Soundex | 1.000 | 0.947 | 0.972 |
| Blocking, Metaphone | 1.000 | 0.947 | 0.972 |

*Figure 4.15: Graph showing the evaluation metric details for all three algorithms*

**Analysis from the above experiments with random queries:**

As we can see from the below graph (figure 4.16) among the 6 experiments, 3 of the experiments blocking algorithm worked better than the standard UDD algorithm, where as in 2 experiments both of algorithms have the same values and in only 1 experiment standard UDD algorithm works better than blocking technique. Another observation is that in majority of the cases we were getting the same results when using blocking with Soundex or Metaphone, hence only one set of data is presented.

By further looking at the above experiments, we can see that recall remains the same for both the algorithms (UDD algorithm and UDD with blocking), it's the precision which is much better in UDD with blocking technique. This leads us to conclude that during the duplicate detection process UDD with blocking avoided adding false positive duplicate records when compared to the standard UDD algorithm. It was also observed that the threshold value also did not make any difference in final results, so it was kept constant for all the experiments.




| | los angles | america | new york | cafe | french | san fancisco |
|---|---|---|---|---|---|---|
| UDD Blocking | 1.000 | 1.000 | 0.952 | 0.714 | 0.972 | 0.944 |
| UDD | 1.000 | 0.944 | 0.919 | 0.555 | 0.972 | 0.972 |

*Figure 4.16: Graph showing the f-measure comparison for all the experiments with random queries*

### 4.4.2 Complete Dataset

As a true test of the effectiveness of the application, experiments were conducted for the complete dataset. When there is no value entered in the query box, the application assumes it as query of all the records in the database.

Below figure 4.17 is the screenshot of the application with logs after it was run for the complete dataset.

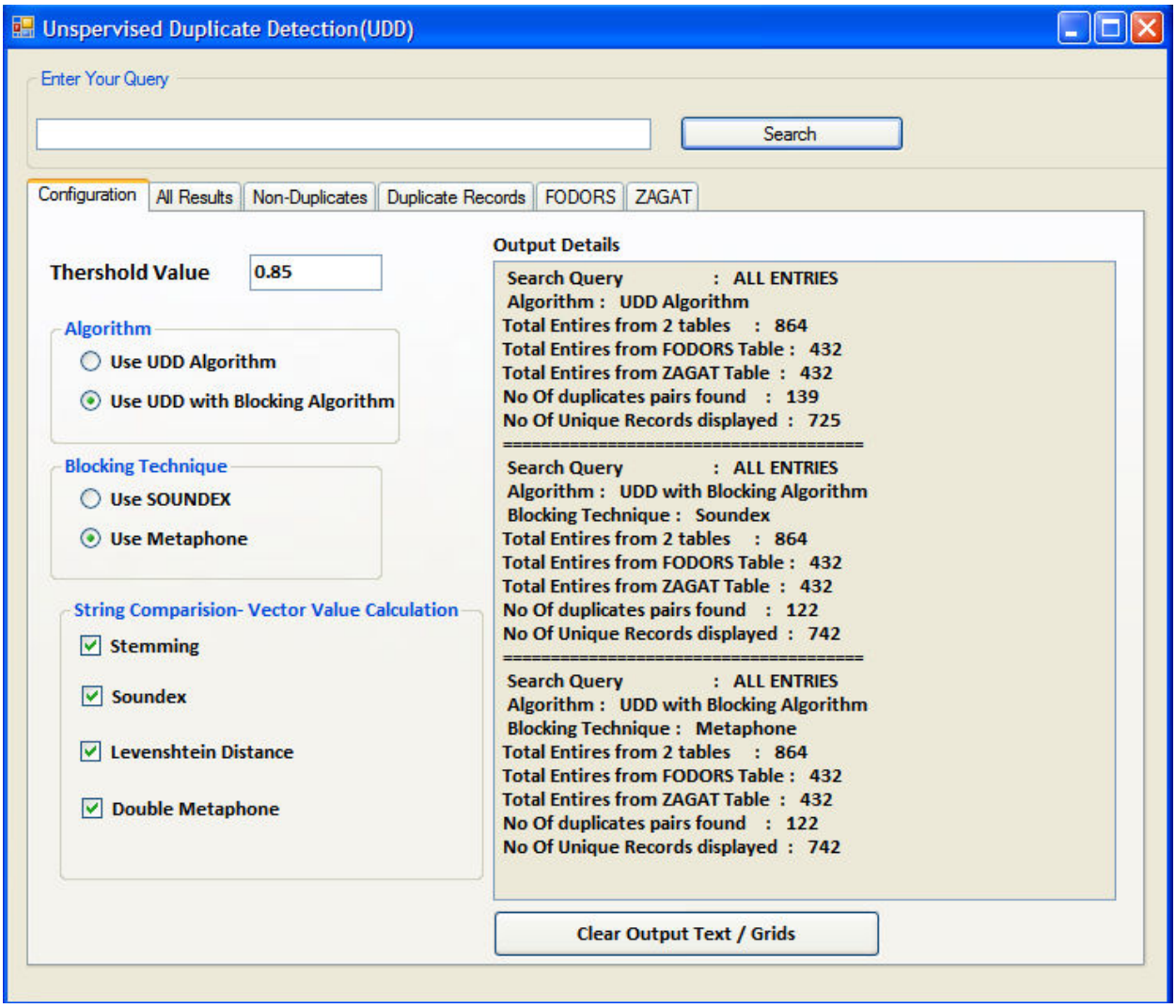



*Figure 4.17: Experiment showing details for the complete dataset*

The evaluation metrics below are calculated for the complete dataset.

| | Number of duplicate pairs found | Correctly identified duplicate pairs | True duplicate pairs | Precision | Recall | *f*-measure |
|---|---|---|---|---|---|---|
| UDD algorithm | 139 | 98 | 112 | 0.705 | 0.875 | 0.780 |
| UDD with Soundex | 122 | 101 | 112 | 0.827 | 0.901 | 0.862 |
| UDD with Metaphone | 122 | 101 | 112 | 0.827 | 0.901 | 0.862 |

*Table 4.7: Precision, Recall and f-measure for the complete restaurant dataset*

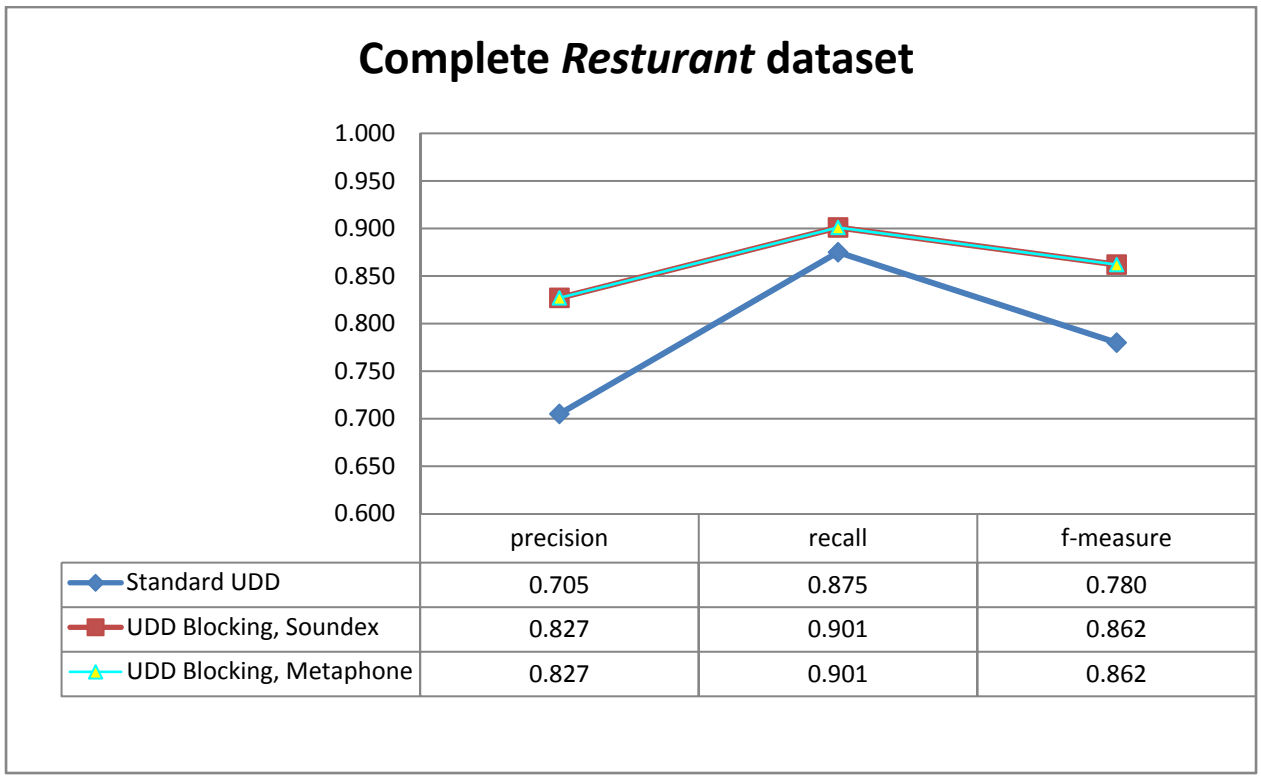



*Figure 4.18: Evaluation metrics for the complete restaurant dataset*

This experiment was run for the complete dataset using UDD and also employing blocking with Soundex and Metaphone. There were 112 identified duplicates [65] from the *restaurant* dataset.

The standard UDD algorithm found a total 139 duplicate pairs with 41 false positives (wrongly classified by the classifier as duplicates). After accounting for this we have an accuracy of 87.5% and *f*-measure of 0.780.

UDD with blocking classifier produced similar results when using Soundex and Metaphone techniques. When compared to the standard UDD algorithm, the blocking algorithm identified 122 duplicate pairs with 21 false positives. The accuracy rate is 90.2% which is better than the standard UDD algorithm. The main difference is the reduction of false positives. Blocking algorithm classified 21 false positives when compared to 41 by the standard UDD. This is a big improvement and helps in reducing wrong classification of records as duplicates.

## 4.5 Comparisons with Other Works

In this section, we make a comparison between the results we obtained in this thesis with other works that used the same dataset.

The dataset in this thesis, the *Restaurant* dataset, was also used by Ravi Kumar and Cohen [60]. Their approach is also based on unsupervised method, similar to this thesis. Their proposed approach is based on a hierarchical graphical model to learn to match record pairs. However, their results were not as favorable, where their best result is only at 82% for precision and 84.4% for F-score. Our precision for the UDD algorithm with blocking approach is 82.7% whilst F-score is 86.2%.

Another work that used the *Restaurant* dataset is the work by Tejada et.al. [61]. They developed a system called Active Atlas [64], which was based on supervised learning. The training data for the application was obtained via decision tree learning based on the computed weights and user input. The results obtained from this approach are 109 true positives and 2 false positives out of 112 duplicates. Meanwhile, our results using the UDD algorithm with blocking approach are 98 true positives and 24 false positives out of 112 duplicates. Our result is slightly worse but at the advantages of not requiring training data and user input.

Cohen and Richman [62] also used the *Restaurant* and *Cora* datasets in their research. Theirs is a supervised approach where the system learns from training data how to cluster records that refer to the same real-world entity in an adaptive way. The results they obtained for the Restaurant dataset is the best insofar, at 100% for both precision and recall.

# Chapter 5- Conclusion and future work

## 5.1 Conclusion

This thesis concentrated on the development of an Unsupervised Duplicate Detection algorithm that can serve as foundation for developing applications that use Web databases. As we had seen from the results, using an additional classifier (like blocking) can result in higher accuracy.

With exponential growth of data, duplicate detection is an important problem that needs more attention, using an UDD algorithm that learns to identify duplicate records has some advantages over offline/supervised learning methods. Although the focus of the UDD application in the thesis was limited to *restaurant* dataset, the same principles can be used broadly to other domains.

When compared to traditional databases, Web-based retrieval system in which records to match are greatly query-dependent, a pre-trained approach is not appropriate as the set of records in response to a query is a biased subset of the full data set. UDD algorithm which is an unsupervised, online approach for detecting duplicates is a suitable solution, when query results are fetched from multiple Web databases. The core of UDD algorithm relies on using WCSS and SVM classifiers to assign weights and classify data. This thesis is a step forward in enhancing the UDD algorithm by adding an additional classifier.

Experimental results demonstrate that using blocking classifier; we were able to limit the number of record comparisons that take place thereby improving accuracy. This was done by effectively grouping source data using similarity metrics like Soundex and Metaphone. One observation was that using Soundex or Metaphone produced almost similar results, which leads us to conclude that any hash function can be used in a blocking algorithm.

A comparison of experimental results with the standard UDD algorithm showed that by using blocking classifier, we were able limit the number of false positives. Although the experiments were limited, this holds a promise that UDD combined with blocking is comparable to other approaches for duplicate detection.

Finally to summarize, we can conclude the following after reviewing the experiments:

1. Unsupervised duplicate detection algorithm is comparable to other supervised algorithms in identifying duplicates (section 4.5). The advantage of UDD is there is no training required.
2. Using blocking classifier in UDD improves overall accuracy rate.

## 5.2 Future Work

This thesis was an effort in developing a solution for the problem of duplicate detection. As we have seen there are numerous approaches and algorithms that aim to address this issue. Although there may not be a perfect solution that would address the problem of duplicate detection, there is scope to develop new algorithms that are generic to meet various requirements. This thesis built upon the idea of UDD and proposed a general framework for using blocking classifier in duplicate detection. There are several directions in which this approach can be extended.

As we had observed, similarity metrics forms the basis for determining the similarity of two strings/objects. There is a need to develop new algorithms that can present accurate results while taking into account various forms of data.

Most of the current duplicate detection systems focus on two aspects, one using machine learning based algorithms to speed up the process of identifying duplicates and second, developing knowledge based approach for matching pairs of records. An interesting direction for future research is to develop techniques that combine these two approaches.

These days there are many applications that extract and present information from the Web. This information is either unstructured or imprecise. Duplicate record detection techniques are crucial for improving the quality of the extracted data. This calls for development of robust and scalable solutions. More research is needed in the area of data cleaning and information quality in general and in the area of duplicate record detection in particular.

# References:


1 Fayyad, Usama; Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996) "*From Data Mining to Knowledge Discovery in Databases*".

2 SB Kotsiantis ,"*Supervised learning: A review of classification techniques*" Informatica, vol. 31, pp. 249–268, 2007.

3 Peter Dayan *" Unsupervised Learning"* http://www.gatsby.ucl.ac.uk/~dayan/papers/dun99b.pdf

4 R. Baxter, P. Christen, and T. Churches, *"A Comparison of Fast Blocking Methods for Record Linkage,"* Proc. KDD Workshop Data Cleaning, Record Linkage, and Object Consolidation, pp. 25-27, 2003.

5 W. E. Winkler. *The state of record linkage and current research problems*. Technical Report RR99/04, US Census Bureau, 1999.

6 I.P Fellegi and A. B. Sunter. *A theory for record linkage*. Journal of the American Statistical Association, 40, 1969.

7 M. A. Hernandez ´ and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery, 2(1):9–37, 1998

8 V.S. Verykios, G.V. Moustakides, and M.G. Elfeky*, "A Bayesian Decision Model for Cost Optimal Record Matching,"* The VLDB J., vol. 12, no. 1, pp. 28-40, 2003.

9 A. McCallum, K. Nigam, and L. H. Ungar. "*Efficient clustering of high-dimensional data sets with application to reference matching"*. In KDD, pages 169–178, 2000.

10 R. Ananthakrishna, S. Chaudhuri, and V. Ganti. "*Eliminating fuzzy duplicates in data warehouses"*. In VLDB, pages 586– 597, 2002.

11 S. Chaudhuri, V. Ganti, and R. Motwani, *"Robust Identification of Fuzzy Duplicates"* Proc. 21st IEEE Int'l Conf. Data Eng., pp. 865876,2005.

12 M. Bilenko and R.J. Mooney, "*Adaptive Duplicate Detection Using Learnable String Similarity Measures"* Proc. ACM SIGKDD, pp. 39-48, 2003

13 A. Culotta and A. McCallum, "*A Conditional Model of Deduplication for Multi-Type Relational Data*" Technical Report IR-443, Dept. of Computer Science, Univ. of Massachusetts Amherst, 2005

14 A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios, "*Duplicate Record Detection: A Survey*" IEEE Trans. Knowledge and Data Eng., vol. 19, no. 1, pp. 1-16, Jan. 2007.

15 Weifeng Su, Jiying Wang, and Federick H.Lochovsky, *" Record Matching over Query Results from Multiple Web Databases*" IEEE transactions on Knowledge and Data Engineering, vol. 22, N0.4,2010.

16 Partrick Lehti(2006), "*Unsupervised Duplicate Detection Using Sample Non-duplicates"*, Lecture Notes in Computer Science, NUMB 4244, pages 136-164.

17 http://msdn.microsoft.com/en-us/library/bb190163.aspx#mdm04_topic4

18 http://www.cs.utexas.edu/~ml/papers/marlin-kdd-03.pdf
19 Bellman R. E. (1957) "*Dynamic Programming*". Princeton University Press, Princeton, NJ
20 "Levsiten Distance: Ristad, E.S.; Yianilos, P.N.; , *"Learning string-edit distance"* Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.20, no.5, pp.522-532, May 1998
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=682181&isnumber=14993
21 Zhan Su; Byung-RyulAhn; Ki-YolEom; Min-Koo Kang; Jin-Pyung Kim; Moon-Kyun Kim; , *"Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman algorithm,"* Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on , vol., no., pp.569, 18-20 June 008doi:0.1109/ICICIC.2008.422
22 T.F. Smith and M.S. Waterman, *"Identification of common molecular sub-sequences"*, Journal of Molecular Biology, 147:195-197, 1981.
23 Waterman, Michael S.; Temple F. Smith and William A. Beyer (1976). "Some biological sequence metrics". Advances in Mathematics 20 (4): 367—387
24 http://www.avatar.se/molbioinfo2001/seqali-dyn.html
25 Jaro, Matthew A. (1976), *UNIMATCH: A Record Linkage System*: User's Manual, U.S. Bureau of the Census, Washington, D.C.
26 http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap2_data.pdf
27 R. B. Yates and B. R. Neto. *"Modern Information Retrieval"*. ADDISON-WESLEY, New York, 1999.
28 Ukkonen E. "*Approximate string-matching with q-grams and maximal matches (1992)"* Theoretical Computer Science, 92 (1), pp. 191-211.
29 http://www.archives.gov/research/census/soundex.html
30 Taft, Robert L. (1970), *"Name Search Techniques",* Special Report No. 1 (New York State Identification and Intelligence System, Albany, NY)
31 Gill, L.E. (1997) OX-LINK: The Oxford Medical Record Linkage System. Record Linkage Techniques - Proceedings of an International Workshop and Exposition, March 21-21, 1997, Arlington, VA, USA, 15-33.
32 Jie Song, Yubin Bao, and Ge Yu. 2010. A multilevel and domain-independent duplicate detection model for scientific database. In Proceedings of the 11th international conference on Web-age information management (WAIM'10), Lei Chen, Changjie Tang, Jun Yang, and Yunjun Gao (Eds.). Springer-Verlag, Berlin, Heidelberg, 729-741.
33 Fellegi, Ivan Peter; Alan B. Sunter (dec 1969). "A theory for record linkage". Journal of the American Statistical Association 64 (328): 1183--1210.
34 W.E. Winkler, *"Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage,"* Proc. Section Survey Research Methods, pp. 667-671, 1988
35 Stefania Cardinaleschi, Fabiana Rocci,Vincenzo Spinelli. *Integration of Administrative Registers and Statistical Archives. The Case of the Eurostat Structure of Earning Survey in Italy.*
36 http://www.autonlab.org/tutorials/bayesstruct.html

37 Elmagarmid, A., Ipeirotis, P., Verykios, V. (2007). Duplicate Record Detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1):1–1
38 http://www.autonlab.org/tutorials/svm.html
39 *Support Vector Machine Concept-Dependent Active Learning For Image Retrieval. Edward Chang, Simon Tong*, KingsbyGoh, Chang-Wei Chang IEEE Transactions on Multimedia 2005
40 S. Sarawagi and A. Bhamidipaty. *Interactive deduplication using active learning*. In KDD, pages 269–278, 2002.
41 P. Juszczak and R. P. W. Duin. *Uncertainty sampling methods for one-class classifiers.* In Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets, 2003.
42 R. Gilad-Bachrach, A. Navot, and N. Tishby. *Query by committee made real*. In Advances in Neural Information Processing Systems (NIPS), volume 18, pages 443–450. MIT Press, 2006.
43 N. Roy and A. McCallum. *Toward optimal active learning through sampling estimationof error reduction.* In Proceedings of the International Conference on Machine Learning (ICML), pages 441–448. Morgan Kaufmann, 2001
44 M. Bilenko and R.J. Mooney, *"Adaptive Duplicate Detection Using Learnable String Similarity Measures,"* Proc. ACM SIGKDD, pp. 39-48, 2003
45 Y. R. Wang and S. E. Madnick. *The inter-database instance identification problem in integrating autonomous systems.* In Proc. fifth IEEE Int'l Conf. data Eng. (ICDE '89), 1989.
46 Becker, S &Plumbley, M (1996).*Unsupervised neural network learning procedures for feature extraction and classification.* International Journal of Applied Intelligence, 6, 185-203.
47 U. Draisbach, F. Naumann, A comparison and generalization of blocking and windowing algorithms for duplicate detection, in: Proceedings of QDB 2009 Workshop at VLDB, 2009.
48 Lawrence Philips, *The Double Metaphone Search Algorithm*, C/C++ Users Journal, June 2000.
49 http://www.isi.edu/integration/papers/tejada02-kdd.pdf
50 Robertson S (2004).*Understanding inverse document frequency: On theoretical arguments for IDF*. Journal of Documentation, 60(5), 503-52
51 http://research.microsoft.com/pubs/67119/svmtutorial.pdf
52 http://www.autonlab.org/tutorials/
53 http://nlp.stanford.edu/IR-book/html/htmledition/choosing-what-kind-of-classifier-to-use-1.html
54 http://www.cs.cornell.edu/People/tj/
55 http://svmlight.joachims.org/
56 http://svmlight.joachims.org/
57 http://www.csie.ntu.edu.tw/~cjlin/libsvm/

58 http://www.cs.utexas.edu/~ml/papers/marlin-kdd-wkshp-03.pdf
Mikhail Bilenko and Raymond J. Mooney. In *Proceedings of the KDD-03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 7-12, Washington, DC, August 2003.
59 http://www.cs.utexas.edu/users/ml/riddle/data/restaurant.tar.gz.
60 P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *20th Conf. Uncertainty in Artificial Intelligence (UAI '04)*, 2004.
61 http://www.isi.edu/info-agents/papers/tejada02-thesis.pdf
62 W.W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02)*, 2002.
63 Implement Phonetic ("Sounds-like") Name Searches with Double Metaphone Part V: .NET Implementation, *CodeProject*. *http://www.codeproject.com/KB/recipes/dmetaphone5.aspx*
64 http://www.isi.edu/integration/Apollo/
65 http://www.cs.utexas.edu/users/ml/riddle/data.html
66 Manku GS, et al. Proceedings of the 16th international conference on World Wide Web. *ACM, New York, NY, USA; 2007*. Detecting near-duplicates for web crawling; p. 141-150.
67 Broder, Andrei Z.; Steven C. Glassman and Mark S. Manasse and Geoffrey Zweig (1997). "Syntactic Clustering of the Web". *Proceedings of the Sixth International World Wide Web Conference (WWW6)*. pp. 1157--1166.
68 Bilenko, M., and Mooney, R. 2002. Learning to combine trained distance metrics for duplicate detection in databases. Technical Report Technical Report AI 02-296, Artificial Intelligence Lab, University of Texas at Austin. Available from http://www.cs.utexas.edu/users/ml/papers/marlin-tr-02.pdf.
69 R. Manivannan and S.K. Srivatsa, 2011. Semi Automatic Method for String Matching. *Information Technology Journal, 10: 195-200.*