Performance Evaluation of Similarity Functions for Duplicate Record Detection

Master Thesis

By

Methaq Kadhum Alnoory

Supervised by

Prof. Musbah M. Aqel

Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Information Systems

Department of Computer Information Systems
Faculty of Information Technology
Middle East University
Amman, Jordan

August, 2011

AUTHORIZATION FORM

إقرار تفويض

أنا ميثاق كاظم ألنوري أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأفراد عند طلبها.

التوقيع:

التاريخ: 17-1-2011

Authorization Statement

I, Methaq Kadhum Alnoory, authorize Middle East University to supply copies of my thesis to libraries, establishments or individuals upon their request, according to the university regulations.

Signature:

Date: 17-1-2011

Examination Committee Decision

This is to certify that the thesis entitled "Performance Evaluation of Similarity Functions for Duplicate Record Detection" was successfully defended and approved on august 3rd, 2011.

Examination Committee Members	**Signature**
Prof. Musbah M. Aqel Professor, Department of Computer Information Systems, Faculty of Information Technology, Middle East University.	----------
Prof. Nidal F. Shilbayeh Professor, Department of Computer Science, Faculty of Information Technology, Middle East University.	----------
Dr. Ahmad H. Al-Omari Assistant Professor, Department of Computer Information Systems, Faculty of Information Technology, Applied Science Private University.	----------

DECLARATION

I do declare hereby that the present research work has been carried out by me under the supervision of Prof. Musbah M. Aqel, and this work has not been submitted elsewhere for any other degree, fellowship or any other similar title.

Signature: 

Date: 17-9-2011

Methaq Kadhum Alnoory

Department of Computer Information Systems.
Faculty of Information Technology.
Middle East University.
Amman, Jordan.

DEDICATION

This thesis is dedicated to my parents, my husband and partner for life, **muthana** and our two children, **Ali** and **Mariyam**, for their patience, understanding and support during the time of this research.

ACKNOWLEDGEMENTS

I am highly indebted to my supervisor Prof. Musbah M. Aqel, Faculty of Information Technology, Middle East University, for his eminent guidance, constant supervision, and whelming encouragement throughout my thesis work.

I am grateful to a number of friends for their moral support, encouragement, and technical discussions.

Finally, I would like to take a moment to thank my family members for their immense patient, emotional support and encouragement during my entire graduate career.

Table of Contents

List OF Tables

List of Figure

List of Acronyms

TF-IDF	Term Frequency-Inverse Document Frequency.
IR	Information Retrieval.
NYSIIS	New York State Identification and Intelligence System.
ONCA	Oxford Name Compression Algorithm.

ABSTRACT

Performance Evaluation of Similarity Functions for Duplicate Record Detection

By
Methaq Kadhum Alnoory

Duplicate record detection is an important process in data quality. Its methods usually rely on the use of similarity functions to identify pairs of records in one or more datasets that refer to the same real world entity.

There is a wide range of similarity functions and very few studies that compare the effectiveness of the various similarity functions. In our research we evaluate the quality of a number of similarity functions on synthetic datasets using a measure used in approximate querying called *discernability*. We based on the semi-automatic method to estimate optimal threshold values. Experiments were carried out to prove the technique proposed. The results show that discernability measure can determine the threshold value and measure if a similarity function is more adequate for a specific data set than another .

الملخص

تقييم أداء دالات الكشف عن التشابه للسجلات المكررة

ميثاق ألنوري

تعتبر دالات الكشف المحور الرئيسي الذي تعتمد عليه عملية اكتشاف السجلات ألمكرره في مجال تنقية البيانات لحساب نسبة التشابه بين الكيانات, من خلال إدخال كائنين وحساب نسبة التشابه بينهما وإرجاع قيمه بين الصفر والواحد وإذا ما كانت القيمة أعلى من حد ألعتبه المحدد مسبقا يعتبر الكيانان يمثلان نفس الكيان في العالم الحقيقي .

ان عملية اختيار ألداله الملائمة عمليه معقده وصعبه لوجود عدد كبير من تلك الدوال بالاضافه إلى قلة الدراسات المتعلقة بتقييمها.

في هذه الدراسة ومن خلال التجربة العملية قمنا بتقييم سلوك مجموعه من دوال التشابه باستخدام قياس

يتم من خلاله تحديد كفاءة ألداله في فصل الكيانات المتشابه عن غير المتشابه بشكل صحيح بالاضافه

إلى تحديد مدى حد ألعتبه يسمى (*Discernability*)*قابلية الدالة على الكشف*.

CHAPTER ONE

INTRODUCTION

1. 1 Introduction

Duplicate Record Detection is the problem to identify pairs of records in one or more datasets that refer to the same real world entity (e.g. patient or customer), where these individual entities might be erroneous and incomplete. In addition, there exists no unique identifying key for these entities that would allow to directly identifying them as duplicates. The problem has existed for decades and several communities have worked on it using different terminology (Hernandez, 1998). Pioneering in this area has been the statistics community, which calls the problem "record linkage" or "record matching", the database community calls it "the merge/purge, "data deduplication" , "instance identification" or ”eliminating fuzzy duplicates”, the Artificial intelligence community calls it "database hardening", "name matching", "identity uncertainty", "object identification", "object consolidation", "coreference resolution" or "entity resolution" (Elmagarmid et al., 2007) We will use the term duplicate record detection or simply duplicate detection in our study. Duplicate record detection is an important process in data integration and data cleaning process. In data integration it is necessary to collate information about an object from multiple data sources (Lim et al., 1996). In data cleaning it is critical to eliminate duplicate records (Batini and Scannapieco, 2006). In light of these demands a variety of methods have been proposed for detecting approximate duplicate records in a database: probabilistic, supervised/semisupervised learning, distance-based, and rule-based [see (Elmagarmid et al., 2007) for a recent survey].Duplicate detection methods usually depend on string similarity

functions for discriminating between match and non-match record fields, as well as on record similarity functions for combining similarity estimates from individual string fields. The similarity function measures the degree of similarity between two objects (strings, records, etc). A similarity function takes two objects values as inputs and returns a score value between in 0 and 1. If the similarity score value is greater than a given threshold value, the two object values are considered to be representations of the same real world object. A large variety of string similarity functions have been developed over the years. (Elmagarmid et al., 2007; Jaro, 1989). Due to this variety, designers often meet the task of choosing the most appropriate function for a given approximate data matching application. There are very few studies that compare the effectiveness of the various similarity functions, an issue raised by Elmagarmid et al. (2007). Hence, the goal of our research is to compare a number of similarity functions on real-world and large-scale benchmarking datasets, and evaluate their performance. We will use duplicate detection as a testbed for evaluating similarity functions.

1.2 Data Quality

In spite of the truth that presents the reality of the impossibility of attaining 100% error-free database for each field inside, data accuracy degree can be determined thoroughly, and that can determine the desired quality criteria are needed according to the determined principle interest variable for a certain database. Data Quality is a major issue nowadays for many business environments and necessity for filter zing errors inside any data storage for mining data process in order to ensure the stored data quality (Herzog, 2007). Duplicate detection removal process can introduce a direct significant enhancement in data quality through

enhancing two major dimensions: accuracy and completeness. Other five most cited properties of relevancy, timeliness, accessibility and results' clarity, comparability; coherence can determine higher data quality for the stored data but under a wider aspects determination beyond what this thesis was described (Neumann and and Her-schel, 2010) .

1.3 Similarity function in Duplicate Detection

In Duplicate detection uses similarity or distance function for comparing between data instances without knowing whether it is under a heterogeneous representation or not in real world. Any similarity function fs (s_1, s_2) can determine pair score for s_1 and s_2 values for [0,1] interval where higher the score value means more similar s_1 and s_2 values, .If two objects are considered to be similar then they are considered to represent one object from the real-world, and that would be satisfied if the similarity score surpassed the threshold value which is predefined. But to choose the value of threshold is a difficult operation and matter meanwhile if the similarity values are greater than the threshold value t, then they represent the same real world object as a very wide range to choose from, The similarity functions have wide range, that is from very simple string matching functions like Levenshtein's edit distance" (Hall and Dowling 1980; Levenshtein 1966; Navarro et al., 2001) specific to XML trees functions (Dorneles et al., 2004). It was noticed that the specifications for the matched data can affects on similarity functions' result quality.

Problems were observed in defining the most suitable threshold value for utilizing it in similarity functions calculations; another problem was also in determining specific adequate measure to see whether the similarity function is adequate for specific set of data rather than other ones. In order to calculate the amount of the similarity function, it is used

the recall/ precision curve, the curve is Information Retrieval (IR) quality measure, as an essential foundation for the current methodologies. In addition to this, it has a significance role of representing the ability of similarity function in ordering the similar outcomes with our research outcomes. Yet, the recall /precision curve doesn't suit the representing of the range of efficiency for the similarity functions to dismiss the related from unrelated similar outcomes (da silva et al., 2007).

As shown in figure 1.1, approximate data matching approaches can be classified into (a) content-based approach, (b) structural-based approach, and (c) the mixed approach between both. This thesis uses the content-based approach (Dorneles et al., 2010)

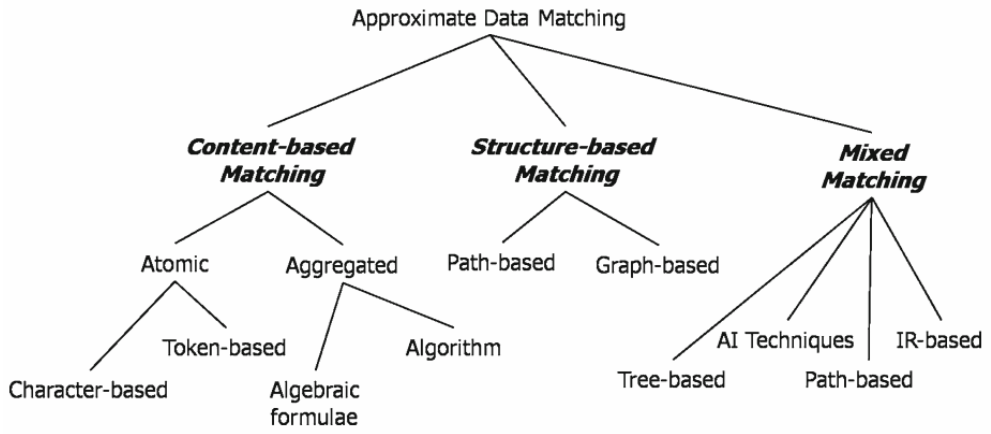


Figure 1.1: Taxonomy of approximate data matching approaches (Dorneles et al., 2010).

1.4 Motivation

With the increasing volume of data, and the improving ability of information systems for the purpose of gathering data from distributed, heterogeneous, and many resources, the problems of data quality abound. One of the most data quality intriguing problems is that of multiple, yet different representations of the same real-world object in the data. For

example, an individual might be represented multiple in the customer database, a single product might be listed several times in the online catalogue and data about a single type protein might be stored in many different scientific databases.

1.5 Problem Statement

Since variations in approximate record-matching problem representation can arise from typographical errors, misspellings, abbreviations, as well as the integration of multiple data sources, using string similarity functions that capture the source of variation is essential for accurate identification of approximate duplicates. There are several similarity available functions “Characters-based (e.g., Levenshtein distance, n-gram distances), context-based, tokens (e.g., Jaccard distance, TF-IDF(cosine similarity combined with the tf.idf weighting scheme to compute the similarity of two fields)), phonetics (e.g., soundex distance) and hybrid between them ” thus, it is often necessary to evaluate a number of them aiming at choosing the function that is more adequate to a specific data matching application that were rarely discussed and studied. This problem was presented by da Silva et al., (2007) address problems related to flexible query processing for similarity functions usage among different data sets; this problem was solved in this thesis through using the same method in da Silva (2007) in duplicate deductions.

1.6 Contribution

This thesis contributes in benchmarking between different similarity functions that are used for detecting duplicates in a certain dataset. We used an approach which was suggested for determining the adequate function for detecting duplicates, our thesis recommended applying the same proposed method of da silva et al., (2007) work that used in similarity

queries, and utilized it in blocking data for duplicate detection. The proposed approach should improve both the "accuracy" and "completeness" values for enhancing data quality through utilizes the *"tbest*" algorithm (da silva et al., 2007) in order to achieve the following objectives:

1. Conduct an experimental study to evaluate the performance of a number of similarity functions in the context of duplicate detection.
2. Find adequate measures that can decide upon the duplicity of records using several similarity measures and thresholds that will contributes in records' duplicates decision-making through distinguishing between token-based measures, edit-based measures, phonetic and hybrid measures to find the most accurate and completeness one to use.
3. Find the best threshold for each field that can best deploy string-similarity functions in order to estimates the matching likeness of the record fields and the overall records.

1.7 Thesis Outlines

The thesis will be organized as follows:

Chapter 2: Literature reviews and Similar Studies.

Chapter 3: duplicate detection framework.

Chapter 4: Discussion, Analysis and Results.

Chapter 5: Conclusion and future work.

CHAPTER TWO
Theoretical Background

In this section, we present theoretical background on the important process of duplicate record detection**.** The duplicate detection process is executed through, deleting duplicate and restriction verification or "address normalization" that in result will enhance the whole data quality (Naumann and Herschel, 2010)**..** Representative process for detecting duplicate is shown in Figure 2.1 Where (*R*) can present records' set that contains the duplication that can be expected and chosen through using certain algorithm. Similarity measures can calculate similarity threshold for separating between both the duplicate pairs and non-duplicating.

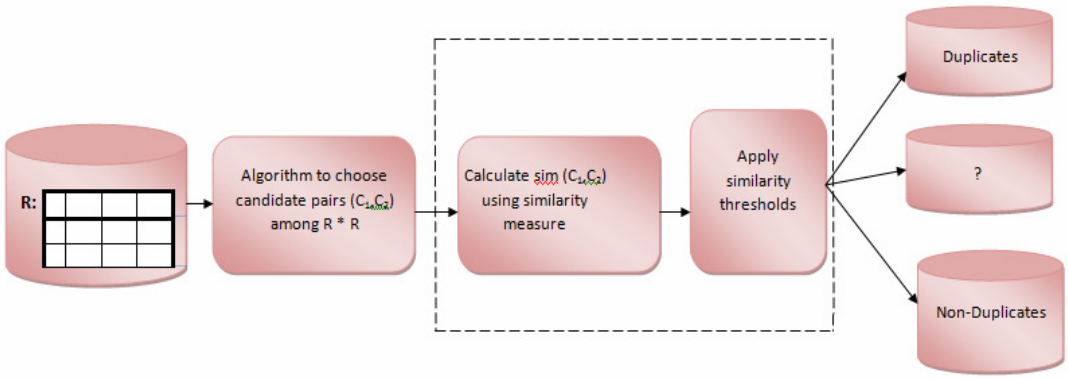


Figure 2.1: Prototypical duplicate detection process (Naumann and Herschel, 2010).

2.1 Duplicate Detection process

Duplicate detection problem needs two components to solving: similarity measure and algorithms.

2.1.1 Approximate field- Matching Similarity Functions

Duplicate detection uses similarity function for measuring similarity degree between two objects by taking the input of two objects values and the output of score value between (0 and 1) and comparing it with an agreed threshold value, considering two object values for presenting the same available objects in real world. It is typical to compare for object similarity instead of equality because it is common that mismatches happen due to typographical error .When discussing similarity function we discuss similarity distance too. Similarity distance can be calculated from similarity function measure as dist $(o_1, o_2) = 1 - fs(o_{1o_2})$. Similarity function can be categorized as character-based similarity function, token-based similarity function, hybrid similarity function and Phonetic similarity function (Elmagarmid et al., 2007 ; Numann and Herschl,2010) .In following sections we discuss this category of similarity function.

2.1.1.1 Character-based similarity function

So called edit-base similarity measure, the character-based similarity metrics are designed to handle typographical errors well. Elmagarmid (2007) showed that where mainly usual string similarity calculating techniques are classified into character-based and vector-space based techniques, it all are based on using character edit operations as a base for its work, like insertions, deletions, subsequence comparison, or even for substitutions, transforming strings into vector account will be modified and translated after then for promoting similarity calculations. Levenshtein (1966) started by introducing a character-based string similarity metric called “Levenshtein Distance” and defined it as the minimum number of insertions, deletions or substitutions necessary to transform one string into another. In his

article "Binary Codes Capable of Correcting Insertions and Reversals" as a most simple edit-distance function which based on counting the inserted and deleted character(s) and switching numbers. More complex edit-functions are affine functions, which assign a relatively lower cost to a sequence of insertions or deletions, like a missing word or missing suffix. In this section we briefly discuss the similarity function that is used in this thesis.

2.1.1.1.1 Q-grame similarity function

Ullmann (1997) and Ukkonen (1992) identified the Q-grams as a short character substrings1 of length q of the database strings, Elmagarmid (2007) showed the Q-gram is a subsequence of q points from a particular sequence. The points within question might be phonemes, syllables, letters, words or base pairs as per to the application. A "unigram" refers to Q-gram of size1, "bi-gram" for size 2, "trigram" for size 3, and size 4 on more simple words called "Q-gram". The bigram and trigram for the value of the pressure is utilized to handle the job of gauging the proximities of identification. It has been exploited Letter Q-grams, including trigrams, bigrams, and/or unigrams in several ways within the text explanation and spelling correction incremental consequences for introducing definition of q-grams as short character substrings1 of length q of the database strings. The intuition behind the use of q-grams as a foundation for approximate string matching is that, when two strings s1 and s2 are similar, they share a large number of q-grams in common letter q-grams, including trigrams, bigrams, and/or unigrams, have been used in a variety of ways in text recognition and spelling correction. One natural extension of q-grams is the positional q-grams, which also record the position of the q-gram in the string. Gravano et al. (2001) showed how to use positional q-grams to efficiently locate similar strings within

a relational database." Gravno et al., (2003) extended its capabilities to handle spelling errors using Q-grams instead of words as tokens. In this setting, a spelling error minimally affects the set of common q-grams of two strings, so the two strings "Gateway Communications" and "Communication's Gateway" have high similarity under this metric, despite the block move and the spelling errors in both words. This metric handles the insertion and deletion of words nicely. The string "Gateway Communications" matches with high similarity the string "Communications Gateway International" since the Q-grams of the word "International" appear often in the relation and have low weight. Gravano et al., (2001) showed how to "use positional Q-grams for efficient location for similar strings within a relational database, Kukich (1992) explained that Q-grams Letter - including trigrams, bigrams, and/or unigrams- used different ways for spelling correction and text recognition. Sutinen and Tarhio (1995) continue the work and explained that "the one natural extension of Q-grams is the positional Q-grams". Naumann and Herschel (2010) showed that It is a common sense that on a *q*-gram measures of similarities the tokens are not specified according to the characters (white space and punctuations), they are turned into a smaller tokens based on the size *q* which are called *q*-grams or *n*-grams. In addition to that, it is known that these tokens normally overlap (one character appears in many tokens) specifically *q* tokens. In order to generate a size q a window must be sled over the string in order to be tokenized and to gather q tokens we present a special character not alphabetically and pad the string with the alphabets. As an example for generating q-grams consider the following:

For example Considering strings S_1 = Hwnrei Waternoose and S_2= Henry Waternose there must be a generation to a trigram (3-gram) for the two resulted strings in the set, the

underscore (_) was used in order to refer to the padding character and the whitespace in both the beginning and the end of the string which is noted as #

q-gram of $S_{1=}$ {##H,Hen, enr,ri_,i_W,_Wa,Wat,ate,ter,ern,rno,noo,oos,ose,se#,e##}

q-gram of S_2 = {##H,#He,Hen,enr,nry,ry_,y_W,_Wa,Wat,ate,ter,ern,rno,nos,ose,se#,e##}

The Q-grams similarity metric between two strings is constructed ranging from 0 to 1.0 using a normalized formula (Ukkonen, 1992)

$$Sim_{Q-grams(s1,s2)} = \qquad \text{.....2.1}$$

is the number $G_{s_1} \cap G_{S2}$ grams of S2 and G_{s_2} grams of first string S1 and G_{S1} Where of common Q-grams between S1 & S

=13 $G_{s_1} \cap G_{s_2}$=17, G_{s_2}=18, G_{s_1} For above example we have

2.1.1.2 Token-Based Similarity Metrics

Elmagarmid (2007) showed that Character-based similarity metrics work well for typographical errors. However, it is often the case that typographical conventions lead to reorganizing the words as *("John Smith" versus "Smith, John")* – where character-level metrics cant capture similarity entities where Token-based metrics attempt to fill the gap of the character-level while sequence-based "estimates distance between shorter strings that differ largely at character level and became too computationally expensive and less accurate for larger strings at the same time tries in computing string edit distance for larger strings such as text documents on the Web since the computational complexity is quadratic in

average string size." The vector-space model tries to overcoming the previous problems through viewing strings as "bags of tokens" while ignoring its order in which the tokens occur in the strings. By Naumann and Herschel (2010) discussed three token-based measures: i- The basic Jaccard coefficient: Jaccard similarity presented the simplest method for computing the shared token inside strings as the likeness degree of token's proportions. If strings *s* and *t* are presented as *S* and *T* token sets, Jaccard similarity is written as:

$$Sim_{jaccard} = \frac{S \cap T}{S \cup T} \quad \text{.....2.2}$$

Jaccard similarity's main issue is that it doesn't take into consideration the proportional significance of distinct tokens. Tokens that happen more than once within a presented chain must contribute highly to similarity rather than tokens which take place little time, just like such tokens which are unique between the strings' groups beyond consideration.

ii-*TF-IDF*: The Cosine Similarity using Tokens Frequency and Inverse Document Frequency (*tf-idf*), Cosine Similarity is often used in information retrieval. Let *s, t* be two multi token in a string or multiple columns in a n dimensional space and t in a filed s (e.g., all tokens in every string value of the column). The cosine similarity for strings s and t is ;

$$\textit{Cosine similarity } (s,t) = \cos(s,t) = \frac{s \cdot t}{\|s\| \cdot \left|\left|t\right|\right|} \quad \text{.....2.3}$$

Where the coefficients of vector t in filed s are defined as

$$\textit{tf-idf} = \log\left([tf]_{\downarrow}(s,t) + 1 \right) \times \log(idf_{\downarrow}(s,t)) \quad \text{.....2.4}$$

Where $tf_{s,z}$ frequency of token *t* in a filed *s*, $idf_{t,c}$ =number of record in block to number of record that contain *t,*

iii- Similarity based on tokenization string using Q-gram, string is divided into smaller tokens in size *q* .strings are fixed tokens consisting of *q* characters (where often *q* = 3) The strings s and t are broken into the sets *S* and *T* of all possible (overlapping) q-grams. These sets are in turn compared using Jaccard or Cosine similarity. Example: s = ′Henri Waternoose′ and t = ′Henry Waternose′

Trigrams for s: S = { ′##H′, ′#He′, ′Hen′, ... , ′i W′, ... , ′oos′, ′ose′, ′se#′, ′e##′ }

Trigrams for t: T = { ′##H′, ′#He′, ′Hen′, ... , ′y W′, ... , ′nos′, ′ose′, ′se#′, ′e##′ }

Jaccard similarity is 13/22 = 0.59

cos similarity using tf-idf weighting is ≈ 0.64

This thesis used the similarity based on tokenization using the Cosine Similarity Using Tokens Frequency and Inverse Document Frequency,

2.1.1.3 Hybrid Similarity Functions

Discussing those similarity hybrid functions that combine string similarity with tokenization for computing the score of final similarity were used two measures that called the “Extended Jaccard Similarity” and the “SoftTF/IDF”. All the introduced hybrid measures are combining the usage of applying both edit-based and token-based measurement equipments in order to rank the repeated edit-based measurement equipments prepared for errors in tokens since the token-based measurement equipments packaged for faults which returned to the lost tokens and the transfer of tokens. Naumann and Herschel

(2010) were more detailed in defining both hybrid measures; it is used the "SoftTF/IDF" one in this thesis.

2.1.1.3.1 SoftTF/IDF

SoftTFIDF was presented by Cohen et al. (2003) and, Naumann and Herschel (2010) as a proposed hybrid measure by which relies on normalizing the *tf-idf* weight of word tokens and can work with any arbitrary similarity function to find similarity between word tokens. In this measure, the similarity score, $sim_{SoftTFIDF}$, is defined as follows:

$$Sim_{SoftTFIDF}(s_1, s_2) = \sum_{ti=close(\theta sim,s1,s2)} \left(\frac{tf-idfti}{\|V\|} \times \frac{tf-idftj}{\|W\|} \times \max\left[Sim(ti,t\square)\right] \right) \quad \text{.....2.5}$$

Where V, W are the vector representation of s_1 and s_2 strings containing (*tf-idf*) scores:

Maxsim (t_i, t_j) = max TokenSim (t_i, t_j), where tj $\in$ Tokenizer (s_2)

2.1.1.4 Phonetic similarity function

Similarity Metrics uses the string-based representation for database records based on both the character-level and token similarity metrics. But knowing that some strings might still be phonetically similar even if are not in a token or character level similar; as can be noticed through presenting the word Kageonne as an example of being phonetically similar to Cajun in spite of the differences between both string representations to proved that such phonetic similarity metrics have the ability of matching such strings. Soundex Phonetic Similarity Function was introduced by Russell (1918 and 1922) as one of most familiar scheme of phonetic coding, Newcombe (1967) showed its unchangeable ability proved through exposing nearly "two-thirds of the spelling variations observed in linked pairs of

vital records, and that it sets aside only a small part of the total discriminating power of the full alphabetic surname." Soundex code was designed specifically for processing "Caucasian surnames" but can generally be used with various names from various different databases with some exclusions that been noticed that this code has some weaknesses in dealing with names inherent in vowel sounds that are ignoring through.

Taft (1970) introduced another different phonetic similarity metric called "New York State Identification and Intelligence System (NYSIIS)", this metric keep hold of the vowels places inside the encoded word through converting vowels into written letters but doesn't replacing letters with numbers but through replacing consonants with other similar phonetically letters so can returning a "purely alpha code –not a numeric component." For conditioned surnames of nine letters maximum length while knowing the limitation of the NYSIIS to handle only six characters; Taft (1970) compared Soundex with NYSIIS for this specific purpose using the New York City names' database and concluded the accuracy of the NYSIIS coding system by 98.72 % comparing to Soundex that presented an accuracy of 95.99%, for that reason this coding system is still used nowadays in New York State for the Division of Criminal Justice Services' usage.

After then, Gill (1997) introduced another phonetic technique called "Oxford Name Compression Algorithm (ONCA)" that works through two stages and introduced for developing the pure Soundex-ing features under a parallel fitting format based on four-character fixed length –where ONCA uses the British model of the NYSIIS in compression, then transmitting and compressing partial name as a second stage. This technique proved its effectiveness in combining similar names together.

Philips (1990) suggested using a “Metaphone and Double Metaphone algorithm” as an improved alternative of Soundex that uses 16 consonant sounds for both English and non-English phrases and providing some additional encoding choices in Double Metaphone (2000) that added the ability of providing a combined encodings specifically for dealing names with different pronunciations that shapes about 10% of total American surnames - in which in result enhanced the matching performance for multiple phonetic encodings (Elmagarmid et al., 2007).We used soundex similarity function.

2.1.1.4.1 Soundex similarity function

Elmagarmid et (2007) showed that It was previously mentioned this metric history where Soundex algorithm picks a script term, like the surname of a person, as an input which generates a character chain that determines a group of terms which phonetically resembles each other. Once the user doesn’t have a complete data, it is useful to search for a large data base. For instance, the word “Truben” resembles phonetically “Tropain” although the chain demonstrations totally differ from each other. The metrics of the phonetic resemblance are attempting to tackle some of these topics and coincide with similar chains. The approach applied through Soundex relies on six phonetic categorizations of the human speech sounds “bilabial, labiodentals, dental, alveolar, velar, and glottal” which thus are relied on the place of locating your tongue and lips in order to utter the sounds. An essential algorithm for computing Soundex is completely explained in appendix A this algorithm calculates Soundex through using the task of the same system numbers for phonetically similar sets of consonants which is mostly utilized to be identical with surnames. Soundex system regulations are as follows: (1) maintain the surname’s first letter as a prefix letter and totally disregard all the happenings of W and H in other places ;(2) allocate certain systems

to the rest of letters, (3) strengthen series of identical matches through sustaining just the first happening of the system;(4) slump the separators; and (5) maintain the prefix letter and the first three systems padding with zeros whether there are less than three systems.

2.1.2 Record Similarity function

In the previous section, we described methods that can be used to calculate the similarity of individual fields of a record. In real-life the database records consist of multiple fields, making the duplicate detection problem much more complex. Al-Khalifa et al., (2003) experimentations showed three separated strategies for Evaluating Record Similarity: (1) comparing the two records as a whole, as if they were a single field or attribute; (2) comparing each field and averaging the resulting similarity scores, which is essentially what is proposed in Broder (1997); and (3) using a feature vector to represent the fields and train a binary support vector machine classifier using this vector. Al-Khalifa continued his work by introducing a function for performing an adaptive edit-distance that performs on specific textual data, Chapman (2004) introduced a more comprehensive similarity functions from Al-Khalifa work for editing all types of textual elements, ActiveAtlas system was introduced by Chaudhuri (2006) that sets major learning rules for mapping tuples (or objects) out from the dual diverse relations (or sources) for setting uniqueness between (Dorneles et al., 2010). In this section, we discus methods that are used for duplicateing records in multiple fields. Nowadays approaches are using matching those records that includ several fields which can be classified under a general status into two groups (Elmagarmid, 2007) .

(a) Probabilistic models approach that uses both the Supervised and Semisupervised Learning Techniques and heavily introduce training data for matching records.

(b) Rule-based approaches that uses the Distance-based techniques and depends on domain knowledge or distance metrics for matching records. For this thesis we used Probabilistic method

2.1.2.1 Probabilistic Matching Decision Models

If two record A and B are to be matched Classify pairs(a,b) $\in A \times B$ as matched (M) or non matched (U).we represent each pairs of record as pair(a,b) as a random vector x=(x….xn) with n number of fields in rescored a Fellegi & Sunter(Fellegi & Sunter ,1969) considered ratios of probabilities :

$$R = \frac{\Pr[\gamma|M]}{\Pr[\gamma|U]}$$ Where γ is a random vector, e.g. {1, 0, 0, 1, 0}

The decision rule based on R is optimal: any other decision rule achieving the same error rates implies conditional probabilities (either on M or U) of not making a decision always greater than the Fellegi-Sunter rule's computed using the training set of pre-label record-based (Elmagarmid, 2007). Newcombe et al. (1959) were first introduced the duplicate detection problem and named it as a "Bayesian inference problem". Fellegi and Sunter (1969) after then officially formalized the Newcombe et al. intuition and introduced a general notation using the random vector technique for different density function of two classes. It was showed that if each class density function is well-identified, duplicate detection problem will called the Bayesian inference. Elmagarmid et al., (2007) showed many developed techniques

2.1.3 Duplicate detection algorithms

So known as blocking algorithms, the purpose of the algorithms is to reduced the number of comparisons to improve efficiency of the system particularly when operating on a large set of data. Blocking algorithm (Jaro, 1989) is achieved by sorting the database on one or combinations of a few attributes as the blocking key, then separating the records into mutually exclusive partitions based on their agreement on the blocking key. Therefore, each record will only be assigned to a block. For example, if the blocking key is the states in the United States, the dataset will be sorted and partitioned into blocks where the first block contains records with state 'AR', the second block contains records with state 'AK', and so on. Records in the same block are considered for comparison. AlDummor (2010) discussed in his master thesis "Performance Evaluation of Blocking Methods for Duplicate Record Detection", the main challenge faces the detecting duplicate records process is the complexity of the detection process itself; of the huge data base, each record in should be compared to all records in data base. Introducing blocking methods was a good solution for detecting duplicates problems that "minimized the number potential record pair comparisons by partitioning the datasets into a set of mutually exclusive blocks or clusters using a blocking key-where all records sharing the same blocking key value will be placed in the same block and only records within a block will be compared". AlDummor also made some comparative experiments in order to benchmark between most recent blocking methods: the sorted blocks and standard suffix array, with two older ones: the standard blocking and sorted neighborhood blocking within a common framework. Benchmarking was according to the quality of the candidate record pairs generated by those methods. Results showed that "sorted neighborhood blocking method outperforms the standard

blocking and that sorted blocks slightly outperforms it in terms of accuracy that the improved suffix array method was better in and also can improve the standard blocking accuracy in its turn. We utilized the traditional blocking technique.

2.1.3.1 Traditional blocking technique

This blocking achieved by creating blocking key from one or combinations of a few attributes and sorted in database to blocking data according it with using similarity function To arranged key (jaro, 1989).

2.2 Performance Evaluation Methodologies of Similarity Functions

2.2.1 Similarity Function Measures and Threshold

Bryan (2006) approach is to "generate individual rankings for each attribute present in a query statement according to a specific similarity metric and then to combine the individual rankings in order to obtain a global ranking such that distance among the individual rankings is to be minimized". Bryan built on the truth of that different score values spread have different similarity functions produce tuple matching problems where various similarity functions' results according to various attributes should jointly be gathered, its solution presents in normalizing similarity scores values into a proposed predestined range, this solution helped in a certain way but didn't solve the problem from its roots since score results from each separate function usage holds diverse unlike meanings (Dorneles et al., 2010).

2.2.2 Similarity Functions and Threshold Value

Throughout of similarity function usage, frequent score identify the two data instances similarity process if the score exceeds the certain threshold for considering both data

instances through the real-life implementation. Determining the returned score values is to be determining according to certain algorithm that is used inside similarity function. Using an approximate matching depends on the used similarity function that allocates the scores to each separate value of data pair where higher similarity introduced privileged scores. Considering two objects as "similar" means that if similarity score s surpasses a pre-defined threshold value yet the threshold value chooses considers as a hard task to perform. Threshold should be defined before the learning process estimation, and determining the result quality that measured through precision and recall calculations as scores values can vary that resulted from using different functions when an exact threshold should be taken into considerations as predefining threshold values for a certain function usage (Dorneles et al., 2010).

2.2.3 Threshold Process Definition

Threshold definition process for the majority of applications is the user responsibility in determining a proper random value to be applied inside queries' implementation as a "trial and error process" in executing possible chosen values until a satisfy result will appear in getting the necessary retrieving data, bearing in mind that score values allocation can be fluctuate extensively between one similarity function and another. Using semi-automatic methodology in calculating threshold values for a given similarity function will produce two threshold values acts as an interval [*tmin best, tmax best*] for calculating the "threshold value" and considering it as an optimal results by maximizing case's number in which $sirrel \leq tbest \leq srel$, where *srel* is the lowest score for a relevant item, *sirrel* is the highest score for an irrelevant item. The identification process of threshold includes reducing false positives and false negatives as well as through taking samples from the pre-existing

collection V of data values (v) and compares it with similarity function in order to use it for determining the distributed score values after then (da Silva, 2007).

2.2.4 Similarity function Quality Measurements

Santos (2010) proposed an estimation methodology for similarity function quality presents in terms of the recall and precision calculations at various thresholds, and according to specific application determinations, choosing threshold value will be held as the user responsibility in estimating it according to the application adequacy – based on a clustering phase executes on sample extracted from a gathered data with no human intervention for.

2.2.5 Evaluating Similarity Functions Quality

Assessing similarity functions performance and benchmark between various types can determine the quality degree of the returned relevant and irrelevant data from the IR querying as a matching degree evaluation, considering higher *fmax* resulted out from similarity function usage as a more efficient function than the one that produce smaller result, it has been noticed that the interval size of tbest presents an indicator about the function quality where larger interval are produced from more efficient similarity functions" (da silva et al., 2007).

Similarity functions have many evaluations approaches. One of the most famous and traditional approaches are the recall and precision calculations (Baeza-Yates and Ribeiro-Neto 1999; Bilenko and Mooney; Ravikumar and Fienberg; Cohen 2003) but unfortunately have only one weakness that its unsuitability for expressing "how efficient similarity functions are in telling apart relevant from irrelevant matches", for covering that gap, quality measures were customized in its design for that purpose using context-data

matching through introducing the ‚Best Threshold Value that can reduce false positives and false negatives with respect to an answer set" (da Silva et al., 2007).

2.2.6 Calculating Discernability

Discernability is a measure specifically designed for evaluating similarity functions proposed by (da Silvaet al. 2007). Besides providing a means for evaluating similarity functions, this technique also estimates the optimal threshold *t* to be used by a similarity function for a data set. This threshold aims at minimizing false negatives and false positives retrieved in response to a set of queries. Details of the discernability computation are given in da Silva et al. (2007). This section provides a brief description of the method. The calculation of discernability takes two aspects into consideration:

(i) Whether the similarity function succeeded in separating relevant and irrelevant data items. A good similarity function should assign higher scores to all relevant data items than to the irrelevant ones.

(ii) The level of separation between relevant and irrelevant data items. An ideal similarity function should not only separate relevant and irrelevant data items, but it should also place them within a reasonable distance, creating two clearly distinct sets.

The formula for calculating the discernability of a similarity function is given in the following equation:

$$\text{Discernability(L)}\ (t_{best}^{min}, t_{best}^{max}, f_{max})$$

$$= \frac{c1}{c1+c2}\ (t_{best}^{min} - t_{best}^{max}) + \frac{c2}{c1+c2} \cdot \frac{f_{max}}{2} \quad \text{.....2.6}$$

Where: L is the similarity function being analyses ; t_{best}^{min} *and* t_{best}^{max} are the limits for the optimal threshold interval; c1 and c2 are coefficients which allow the user to express the importance given to each of the two aspects considered above; f_{max}, is the number of points achieved by the threshold interval [t_{best}^{min}, t_{best}^{max}]; and n is the number of queries.

The relevance judgments provided by the user enable the identification of two important points in a ranking generated by a similarity function in response to a query:

Where s_{rel}: The lowest score achieved by a relevant data item, s_{irrel} - The highest score achieved by an irrelevant data item. Plotting a set of queries with their respective *srel* and *sirrel* is possible to visualize the distance between the relevant and irrelevant data items assigned by the similarity function. In our approach, such a distance is an important parameter used to evaluate the quality of a given similarity metric. As mentioned before, a good similarity function will clearly separate the relevant set from the irrelevant set.

The best thresh algorithm (da Silva et al. 2007), which finds the optimal threshold interval (highlighted gray in Figure 1), is based on a reward function. It proceeds as follows: Each threshold t in the interval [0,1] (varying according to a predefined numeric precision), is compared to s_{rel} and s_{irrel} for the rankings produced in response to a number of queries. One of three outcomes is possible from such comparisons:

(i) The threshold t is at the same time less than s_{rel} and greater than s_{irrel}. This means that it is able to separate relevant and irrelevant items, so it earns two points.

(ii) The threshold t satisfies only one of the conditions. This means that both relevant and irrelevant items are on the same side (either above or below) the line drawn by the threshold. It then scores zero points.

(iii) The threshold t fails both conditions. In that situation, the last relevant result is below the threshold line whilst the first irrelevant result is above it. As a result, t loses 2 points.

The algorithm then searches for the highest number of points (f_{max}) achieved by a threshold.

Once fmax is found, the algorithm searches for the contiguous interval of values of t ([t_{best}^{min} , t_{best}^{max}]) that achieve (f_{max}). In addition to best threshold, da Silva et al. (2007) propose a statistical method for finding the optimal threshold. This method is based on the distribution of *srel* and *sirrel* values for a sample of n queries. Experimental results show that both methods for threshold estimation are in agreement, the interval of threshold that best separates relevant and irrelevant data items - as shown in figure 2.2.

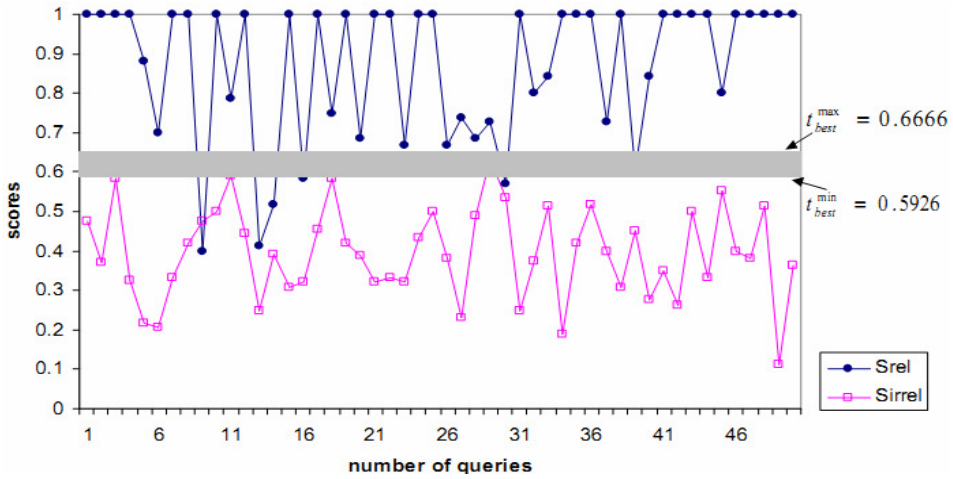


Figure 2.2: Thresholds Intervals (da Silva et al., 2007).

2.2.7 Using SimEval Tool to calculate similarity function quality

Heuser, et al. (2007) explored in their article "SimEval - A Tool for Evaluating the Quality of Similarity Functions" the Approximate data matching applications typically use similarity functions to quantify the degree of likeness between two data instances which is necessary to evaluate them in order to choose the most suitable function to a specific application. For that purpose, the paper presented "SimEval tool" that uses average precision and discernability to evaluate the quality of similarity functions. The researchers recommended making a decision of which similarity function is most suitable for a certain application in order to perform range queries. Performance enhancements can be achieved through direct implementation of the similarity functions into the database, and of the graphic generation for visualizing results.

2.3 Related Studies

In this section we are introduceing some of the most important related studies in the field of similarity function which provides us a good guidance to our work.

(**Stasiu et al., 2005**) in this paper means problem was threshold of similarity function. Semi-automatic method was proposed to fined threshold for similarity function in term of recall and precision. He was extracting samples from the dataset containing the instances to be matched by the similarity function, then (a) Take each data instance from the sample as a query instance.

(b) Compare each query instance to all instances in the sample and compute precision and recall figures at several scores

2-the human expert was informing how many distinct real world objects are represented by the instances in the samples taken from the dataset

3. For the clusters Recall and precision were computed.

4. Precision and recall at different score values over all queries taken in the previous step were computed.

The main problem in this work was that, some similarity metrics are more adequate than others for handling a specific data.

(**da silva et al. , 2007**) proposed semiautomatic methods specifically designed for efficiency of similarity functions are in ranking of data as relevant or irrelevant.

The first method is an algorithm based on a reward function, and the second is a statistical method. Both methods for threshold produced similar results. The output of such methods was used to calculate quality measure, called *discernability*.

Discernability is a quality measure for similarity function discussed two problem related on similarity function first, the threshold value and second the adequate function for specific data set. The problem with this method was human intervention.

(Elmagarmid et al., 2007): The researchers presented a thorough analysis of the literature on duplicate record detection covering similarity metrics that are commonly used to detect similar field entries. They also present an extensive set of duplicate detection techniques that can detect approximately duplicate records in a database. They also cover multiple methods for improving the efficiency and scalability of approximate duplicate detection algorithms. In addition to coverage of existing software tools with a brief discussion of the big open problems in this filed.

CHAPTER THREE

Duplicate detection framework

3.1 Introduction

Duplicate detection process consists of several stages. Our design for duplicate record detection framework takes its design from the generic frameworks suggested in solving record linkage and duplicate detection problems (Gu et al., 2003). Figure 3.1 illustrates diagram based on our search. We suggested four stages for the duplicate detection process.

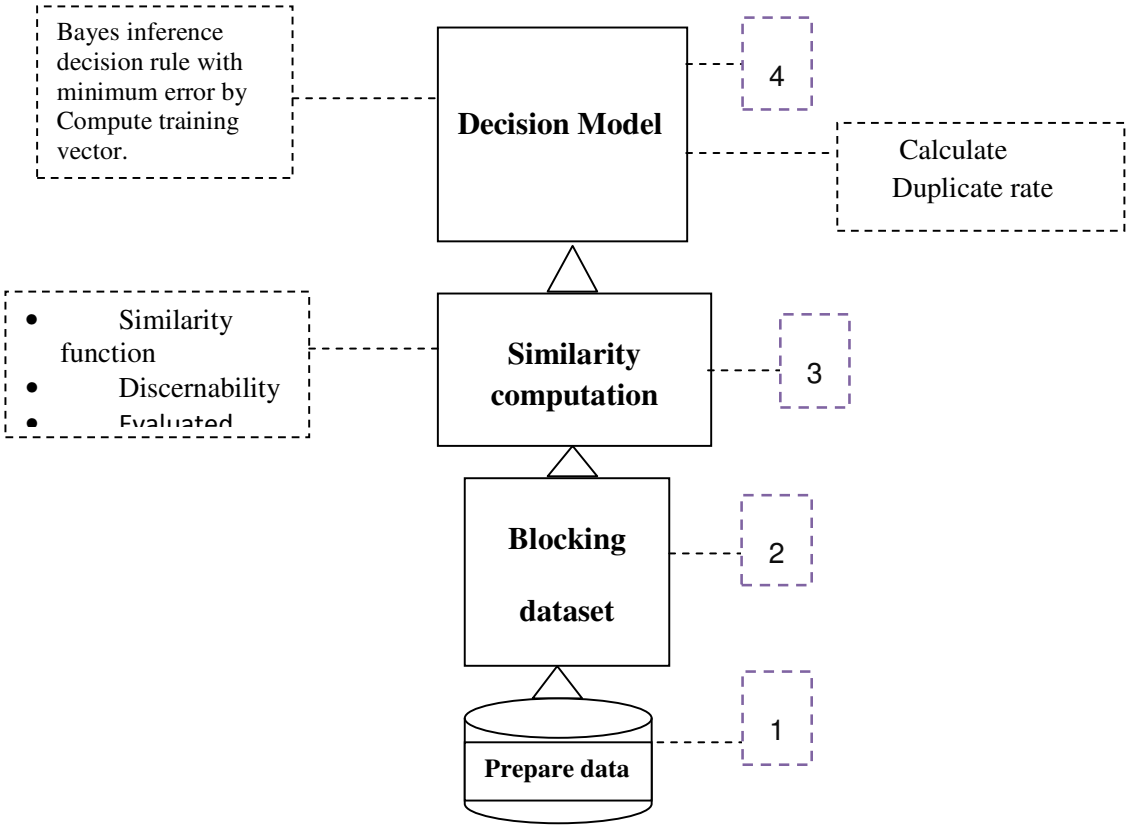


Figure 3.1: Framework for duplicate detection

Stag One: prepare data

Typical duplicate detection process is preceded by a data preparation stage in which data entries are stored in a uniform manner in the database, resolving the structural heterogeneity problem “refers to the problem of matching two databases with different domain structures, as a customer address might be stored in the attribute ’address’ in one database but represented in attributes ’street’, ’city’, and ’zip’ in another database”. In our research, we are concerned with the Lexical heterogeneity problem (refers to databases with similar structure but different representation of data, such as ’J. Smith’ and ’Smith, John’) and assumed that structural heterogeneity has been resolved a priori (i.e. the input is a set of structured and properly segmented records).

When transferring the datasets to the database, we added field to store the records blocking keys to used in next stage

Stage Two: Blocking Stage

Grete blocking key (cand key): it created by composite first three characters the select attribute/s. The following example of SQL statement shows that we only create candidate keys for records with non-empty data for the first attribute in the candidate key, the result show in fig 3.2

```
UPDATE DATA SET candKey=UPPER (CONCAT (SUBSTRING (venue, 1, 3), SUBSTRING
(given name, 1, 3))) WHERE venue! =’’
```

This is a view of the raw data

ID	Given Name	Sure Name	Street Num.	Address 1	Address 2	Key
1	peyton	bills	28	lofty close		PEY
2	peytn	bikls	28	lofty close		PEY
3	peyton	bills	28	lofty close		PEY
4	peyton	bills	12	lofty vikose		PEY
5	peyton	bills	28	loftylywise	glekede	PEY

ID	Given Name	Sure Name	Street Num.	Address 1	Address 2	Key
1	peyton	bills	28	lofty close		PEYBILLOF
2	peytn	bikls	28	lofty close		PEYBIKLOF
3	peyton	bills	28	lofty close		PEYBILLOF
4	peyton	bills	12	lofty vikose		PEYBILLOF
5	peyton	bills	28	loftylywise	glekede	PEYBILLOF
6	benjamin	tritton	81	mackenzie street		BENTRIMAC

Figure 3.2: create key for attribute/s

Then we block data by used Traditional blocking method. In this method all the blocks are non-overlapping windows. Also, they may not appear subsequently one after another, which means that there are records that do not belong to any blocks. In such cases, we ignore those records.

Blocking algorithm that is shown in figure 3.3, (Pie, 2008) display how blocks are created: recall first record (A) and below it (B) from the stored data inside memory; then computing the similarity score between candKey A and candKey B using the Needleman and Wunsch similarity metric (Amagarmid, 2007) show their class in appendix B, by used threshold was defined by user. If the similarity score equals to threshold, record (B) and record (A) put into the same block. Repeating previous process can put all subsequent records that satisfy the threshold into the same block as record A. If no subsequent records are found to satisfy the similarity test, the next record should be used for re-identifying a new block.

```
Input: All sorted records from database (data); blocking threshold
Output: Identifiers of all records in a block (ids)
1  quitSearch← false;
2  while not end of data and !quitSearch do
3       A← candidate key for the next record;
4       blockSize←1;
5       while not end of data do
6            B←candidate key for the next record;
7            score← similarity score between A and B based on Needleman and
             Wunsch similarity metric;
8            if score ≥ threshold then blockSize←blockSize+1;
9            else
10                go back to the previous record;
11                quit inner while loop;
12 if block size > 1 then
13       go back to blockSize previous records;
14       ids← identifiers of all records in block;
15       quitSearch←pairRecords(ids);
16 else skip;
```

Figure 3.3: blocking algorithm (create block)

User Interaction with stage 2

In this stage the user selects the suitable attribute to make blocking key and sets how he will be blocking data according to field or to record.

If the blocking data is according to field, the user selects one attribute and set the threshold value as (1) for accurate results, see figure 3.4 and the result in figure 3.5. when he is blocking data according to record, user select attributes that are suitable for blocking but change threshold value into (0.70) for used data in this thesis , reasons were presented in having 6000 original when take threshold in interval (1_0.75) we get block more than 6000 that means we loss delicate block, at 0.70 gets more than 5000 and less than 6000.

Figure 3.4: The user selects both the blocking keys of "Given name" and the most suitable threshold value (1).

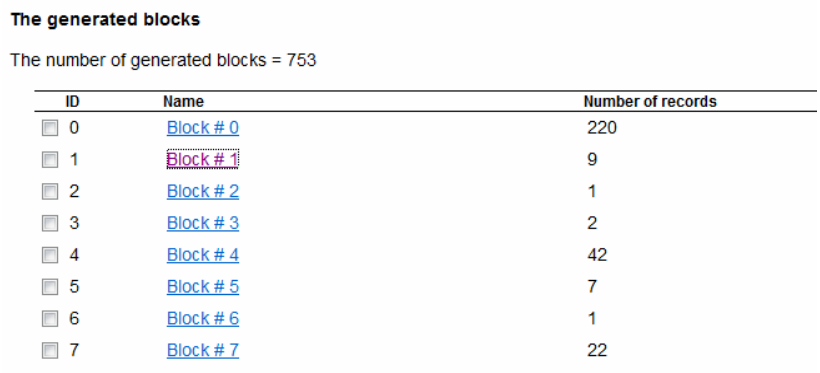


Figure 3.5: blocks created for dataset

Stage 3 Similarity Function's work

In this stage, the user sets similarity function suitable for data type. We chose the following similarity function:

1-Character-Based Similarity Metrics: we used *Q-gram* with *q*=3.

2- Token-Based Similarity Metrics: we used *TF-IDF*

3-hybrid similarity function functions: We used similarity measure that combines between *Q-grame and TF-IDF (soft tf/idf)*

4- Phonetic Similarity functions: we used *soundex* function.

In appendix B the class which is used in stages of frame work are presented.

Evaluation similarity functions:

In this thesis, we used a quality measure specifically designed for similarity functions in the context of data matching using with similarity query (Da selva , 2007) to evaluation similarity function in duplicate deduction as below**.**

Calculate *Discernability*

Discernability: a quality measure specifically designed for similarity functions in the context of data matching (da selva , 2007).

-A similarity function f (s_1, s_2) → s assigns a score s to pair of data values s_1 and s_2, values s_1 and s_2 are considered to be representing the same real world object if *s* is greater than a given threshold *t*.

- How to determine the threshold value? And how to measure if a similarity function is more adequate for a specific data set than another.

-To calculate discernability as a byproduct we provide a method for defining a threshold value that may be explanation as the "*best*" one for a given similarity function, when considering a specific data set. *Best* means a value of threshold that satisfies high duplicate rate.

Calculate *Best threshold* algorithm:

- User randomly select number of blocks had number of record more than (40) (Stasiu, 2005). See figure 3.6

The number of generated blocks = 753

ID	Name	Number of records
0	Block # 0	220
1	Block # 1	9
2	Block # 2	1
3	Block # 3	2
4	Block # 4	42
5	Block # 5	7
6	Block # 6	1
7	Block # 7	22

figure3.6: user select blocks

Next, a human expert is ranking each block in term of a relevant (*rel*), if the data value is considered to represent the same real world object or an irrelevant (*irrel*) otherwise for example shows figure 3.7 where *NS* represent relevant or irrelevant that user determine

Claculations for column GIVEN_NAME

Characteristic (Q-Gram)

Score	GIVEN_NAME	Relevant
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
0.401	zarlia	-- NS --
0.401	zarlia	-- NS --

Token (TFIDF)

Score	GIVEN_NAME	Relevant
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
0.0	zarlia	-- NS --
0.0	zarlia	-- NS --

Hyper (SoftTFIDF)

Score	GIVEN_NAME	Relevant
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
0.923	zarlia	-- NS --
0.923	zarlia	-- NS --

Soundex

Score	GIVEN_NAME	Relevant
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
1.0	zara	-- NS --
0.923	zarlia	-- NS --
0.923	zarlia	-- NS --

Figure 3.7: Calculate similarity functions to select blocks

This labeling enables us to identify two important points in the blocks: *rel(k)* which is the lowest score corresponding to a relevant item and *irrel(k)*which is the highest score attained by an irrelevant item. Notice that for some queries *irrel(k)* could be greater than *rel(k).* Such a situation indicates that the similarity function has failed to separate relevant from irrelevant items where *k* is the number of block .Then we are calculate algorithm of *tbest* threshold shows in figure 3.8 (da selva , 2007).

```
1: Input: n, tmin, tmax, sL rel(k), sL irrel(k), k = 1 . . . n, h
2: Output: tmin best, tmax best , fmax
3: fmax = −2n;
4: ndiv = (tmax − tmin)/h
5: for (a) i = 0, . . . , ndiv do
6:          t = tmin + ih;
7:         f (t) = 0;
8:        for (b) k = 1, . . . , n do
9:            d = 0;
10:          if (Srel(k) > t) then
11:                d = d + 1;
12:          else
13:               d = d − 1;
14:          end if
15:          if (S irrel(k) < t) then
16:               d = d + 1;
17:          else
18:              d = d − 1;
19:          end if
20:          f (t) = f (t) + d;
21:      end for (b)
22:      if (f (t) ≥ fmax) then
23:           fmax = f (t);
24:     end if
25: end for (a)



26: t = tmin
27: while (f (t) ≥fmax) do
28:       t = t + h
29: end while
30: tmin best= t
31: t = tmax
32: while (f (t) ≥fmax) do
33:       t = t − h
34: end while
35: tmaxbest= t
36: if fmax < 0 then
37:       aux = tmaxbest
38:       tmaxbest= tminbest
39:       tminbest= aux
40: end if
```

Figure 3.8:best threshold algorithm (da selva, 2007).

Where the *k* number of select block, *tmin* and *tmax* are, respectively, the smallest and the largest similarity scores, *h* interval division for *rel* and *irrle* , *n* The numerical precision and *f(t)* measured of similarity function for blocks.

The outputs of this algorithm are:

Interval of best threshold for each similarity and *fmax* which is the number of points achieved by best threshold.

When we are representing the value of relevant and irrelevant in a plot diagram can determine best threshold manually show that in figure 3.9, and we can determine the distribution of samples of blocks with threshold in a plot to sets at any threshold to gets high distribution(*fmax*) manually show figure 3.10 (da Silva 2007)

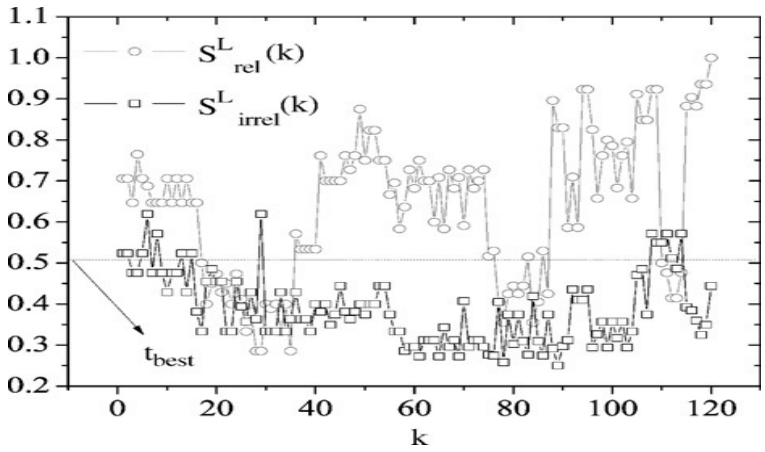


Figure 3.9: distribution relevant and irrelevant as function of the kth block for function L . (da Silva 2007)

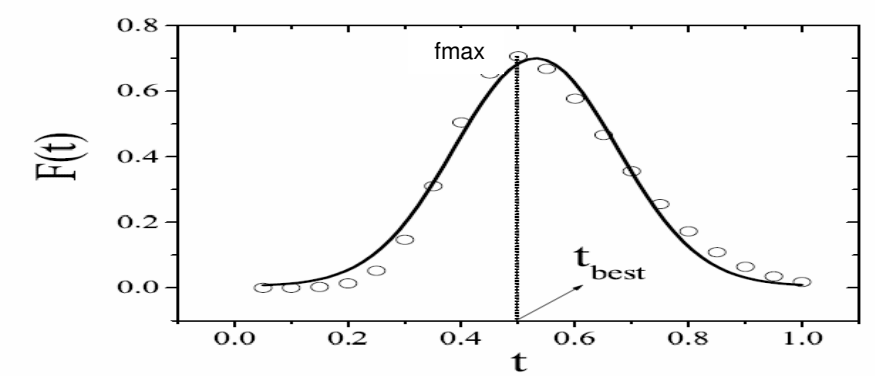


Figure 3.10: Plot of *f(t)* as function of *t* for the similarity function *t* is a threshold (da Silva 2007)

Then we calculate discernability measure to judgment which similarity is adequate to used for each attribute depending on the value of discernability. The best function has high value of discernability.

If there is more than one similarities function have high discernability, the decision depending on the value of *fmax* ,when the value of *fmax* is high that means the function is more adequate to achieved ranking of block

We set the best threshold for each attribute to be used in next stage to determine training so as vector to calculate duplicate rate.

Stage 4 decision model

We use probabilities models for duplicate detection as Bayesian inference problem techniques .This technique used when the density or threshold of each class or attribute known (Elmagarmid et al, 2007).

The comparison vector *x* is the input to a decision rule that assigns U or M. Where U and M are the un matching or matching respectively, and x the randomly vector for similarity of each attribute, to pairs of records. We create pairs of recorded by using algorithm of pairs, (pie, 2008) shows figure 3.11.

```
Input: Identifiers of all records in block
Output: Record pair (t1, t2), quitSearch
1 if evaluating a block then

2       for id1 ∈ ids do

3           t1←id1;

4           for id2 ∈ ids where position of id2 > position of t1 in ids do

5               t2←id2;
```

Figure 3.11: pairs of record algorithm (pie, 2008)

CHAPTER FOUR

Analysis and Results

4.1 Introduction

This chapter aims to use the discernability function to evaluate the quality of different similarity functions. *Discernability* takes two aspects, given by Equation (2.6): *first* aspect is how well the similarity function separates relevant from irrelevant items, given by the maximum number of points ($f_{\max}$), noticed for some blocks, irrel(k) could be greater than rel(k) which presents an indicates about the failure of this similarity function in providing an accurate separation . *Second* aspect is how far apart in the ranking the similarity function places relevant (*rel*) and irrelevant(*irrel*) can be calculated by taking the difference between $t_{maxbest}$ and $t_{minbest}$, this aspects calculated by the *BestThresh*(t_{best}) algorithm. Benchmark among the *Discernability* measure for utilized similarity function's performance is based on two aspect of *Discernability*. According to our approach, a similarity function that has a higher ($f_{\max}$), is adequate function; added the interval range size of t_{best} as another indicator for function quality. Given that a good similarity function should place both *relevant* and *irrelevant* items far away from the ranking process concluded the truth of having larger interval can produce better, also *Discernability* that presented as the difference between two coefficients c_1 and c_2 in which users expressions can shed the light on the importance of considering each of the two aspects according to the database type that can affect Our thesis experiment gave the same importance to c_1 and c_2 sing $c_1 = c_2 = 1$. The produced values through using the discernability function will

range within($t_1max = 1, t_1min = 0$)Interval under a fixed precision of h = 0.001 used in computing [$t_{maxbest}$, $t_{minbest}$] interval in *BestThresh* algorithm.

4.2 Determining the Discernability Value for Executing Experiments

The data set we have used for our experiment is called FEBRL (Freely Extensible Biomedical Record Linkage) data set. The FEBRL data set contains patients data such as names (given and surname), addresses, ages, phone numbers and social security numbers. This data set contains 9000 records among them there are 6000 original records and 3968 duplicated records. There is up to 8813 duplicates for an original record, a maximum of 3 modifications per attribute, and a maximum of 10 modifications per record in each duplicated record. Table 4.1 shows a sample duplicate records from this data set. Data set preparation by MySQL due to its free usage and fast performance, the system is built using the Java Platform. Java is chosen because of its cross-platform capabilities.

GIVEN_NAME	SURNAME	ADDRESS	AGE	PHONE_NUMBER	SOC_SEC_ID
Hannah	urquhart	florina place	-	03 50770094	1614610
hann ah	urquzart	florihaplace	-	03 50770094	1614510
Teagan	Upton	Wilkins street	18	07 45868407	1198233
Teagan	Uptiny	Wilkins street	18	07 45867407	1198233
Teatan	Upton	Wilkins street	-	07 45886407	1198233
Brooke	Uren	barrett street	31	03 38165026	5411400
Broole	Umrh	barrettaareet	31	03 38165026	5421400
Brooke	Urpn	barrettistreet	31	03 38170526	5411050

Table 4.1 Sample duplicate records from the FEBRL data set (datamining.anu.edu.au)

4.2.1 Experiments

Our experiments processes are:

1. We create blocking key.
2. Blocking dataset according to blocking key.
3. User takes number of blocks in randomly shape, and calculate similarity functions for each block.
4. Then user labels blocks as relevant rel (k) and irrelevant irrel (k).
5. After that *Discernability* is calculated.

This experiment doing through 60 blocks ,when user is taking more or less numbers showed take on constant results (even this blocking numbering changing can not affect on (t_{best}) value, and fmax distribution curve. Experiments results when changing n into 50, 40, 30, and 20 respectively are shown in appendix C).

4.2.1.1 Blocking Data According to Field

In our data set the major fields are "given name, surname and address". When we block dataset according to field the result is shown as follows.

- **Data blocking result according to" given name"**

As shown in *table* 4. 2 below; where the first column presents the similarity function name, the second column fmax values that determine the achieved point's numbers by (t_{best}) for a given function, the third column displays the results for *discernability*, and the fourth column shows the interval for (t_{best}) calculated by the *Best Thresh* algorithm. According

to results, *soundex* showed best results and was ranked as best similarity function to use while *TFIDF* was the worst. This conclusion can be proved and through the plotted curves described in figures 4.1, 4.2, 4.3&4.4, showing both the relevant and irrelevant plots. Indeed, the best separation between relevant and irrelevant data items was achieved by *soundex*, whilst with *TFIDF* these items are often shuffled and/or too close together in the ranking**:**

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	76.0	0.31667	[0.43501 , 0.43501]	0.43501
TFIDF	50.0	0.33284	[0.74101 , 0.99001]	0.99001
SoftTFIDF	72.0	0.3	[0.94601 , 0.94601]	0.94601
Soundex	106.0	0.44167	[0.94601 , 0.94601]	0.94601

Table 4.2: *Discernability* results according to Given name field.

Duplicate rate Calculations results according to the determined Discernability in table 4. 2 are shown in table 4.3.

Column (GIVEN_NAME) using Q-Gram |

Duplicate	Dissimilar
1079.0	7736.0

Column (GIVEN_NAME) using TFIDF |

Duplicate	Dissimilar
675.0	8140.0

Column (GIVEN_NAME) using SoftTFIDF |

Duplicate	Dissimilar
900.0	7915.0

Column (GIVEN_NAME) using Soundex |

Duplicate	Dissimilar
1023.0	7792.0

Table4.3: Duplicate rate Calculations according to the discernability results in table 2.

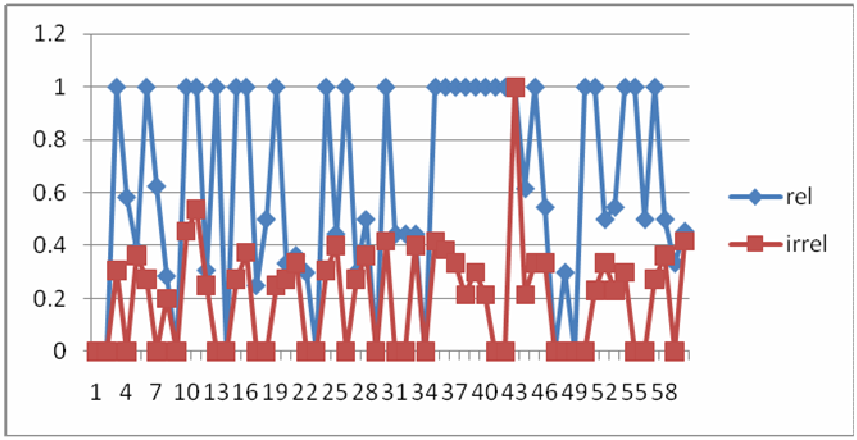


Figure 4.1: distribution relevant and irrelevant as function of the kth block for function Q-gram plot.

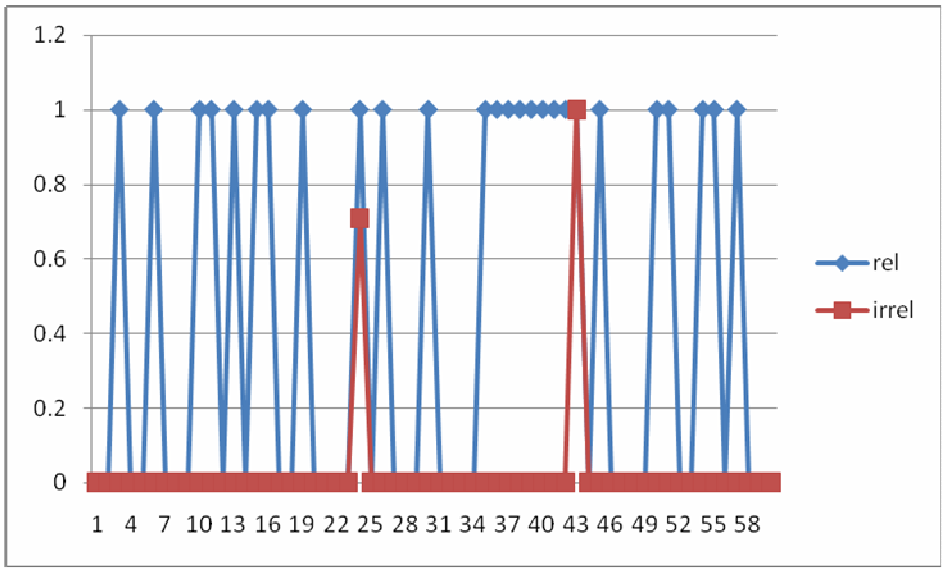


Figure 4.2: distribution relevant and irrelevant as function of the kth block for function TF-IDFplot.

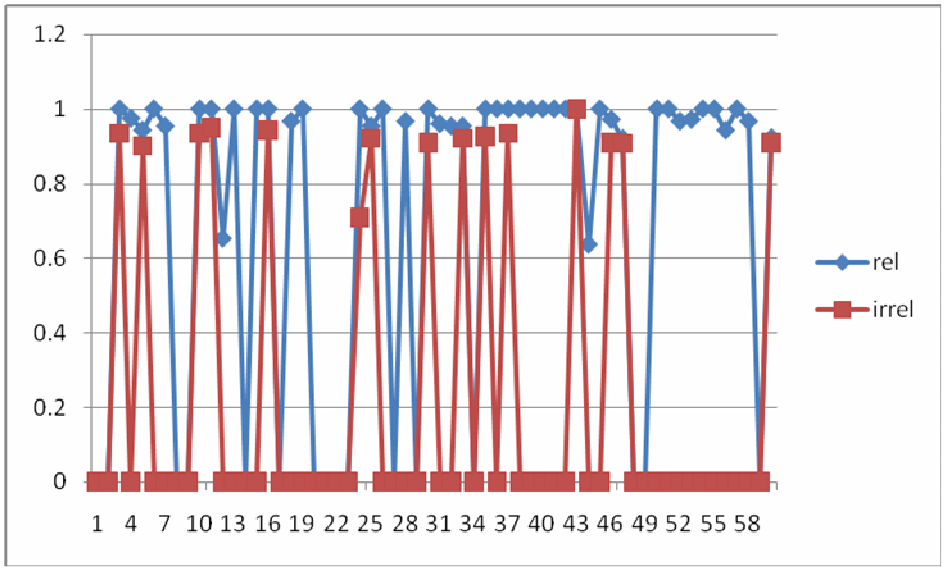


Figure 4.3: distribution relevant and irrelevant as function of the kth block for function soft tf-idf plot.

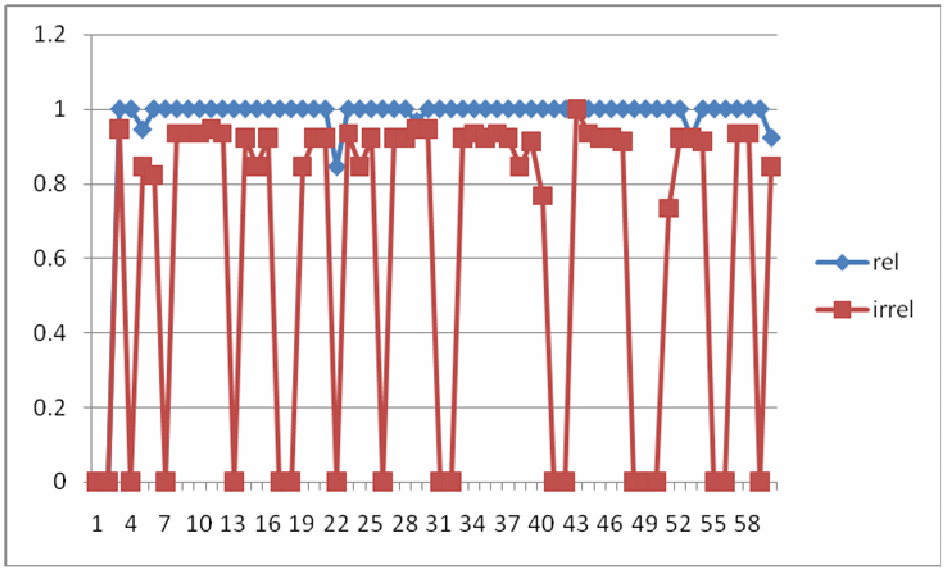


Figure 4.4: distribution relevant and irrelevant as function of the kth block for function soundex plot.

Calculating the duplicate numbers in the used dataset by using decision model for rating in this thesis was through using Q-gram and soundex function as shown in table 2 ,the resulted in more precision for Q-gram but not to consider as an adequate one in determining

similarity for chosen field even that they determine high rate of duplicate due to the irrel(k) could be greater than rel(k) .Such situation indicates that the similarity function has failed to separate relevant from irrelevant items, we have seen that in f_{max} values in table 4.2 it less than value of soundex. This can be confirmed by observing the plots in figure 4.5, 4.6, 4.7, 4.8 which show the number of points achieved by that threshold interval. Infer from that the precession is not sufficient indicator to assess the quality of similarity function.

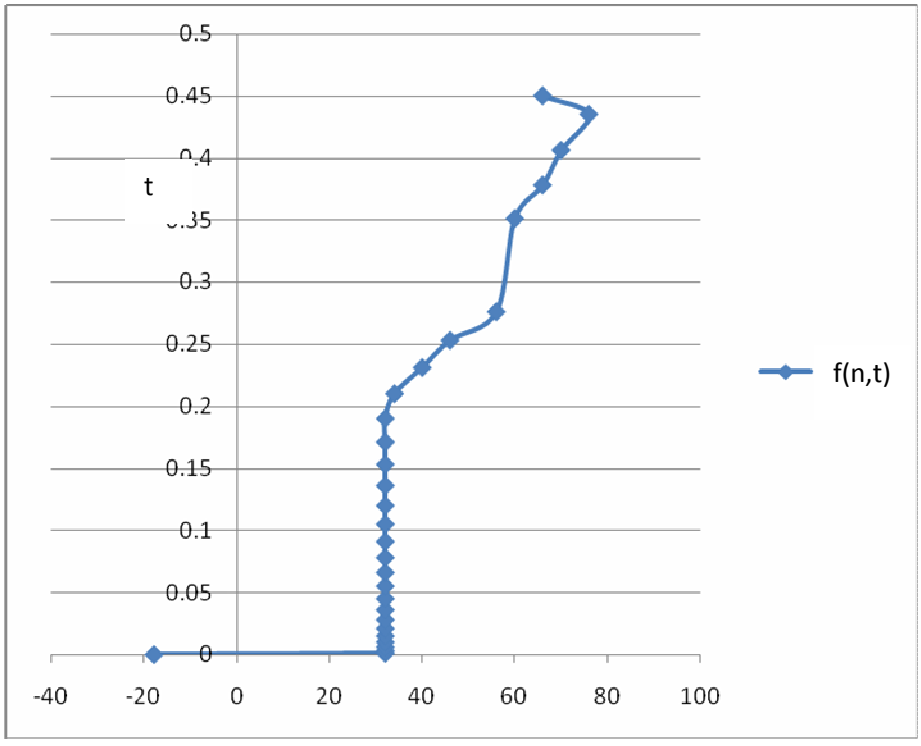


Figure 4.5: *Qgram* Distribution f(t) and t Curve

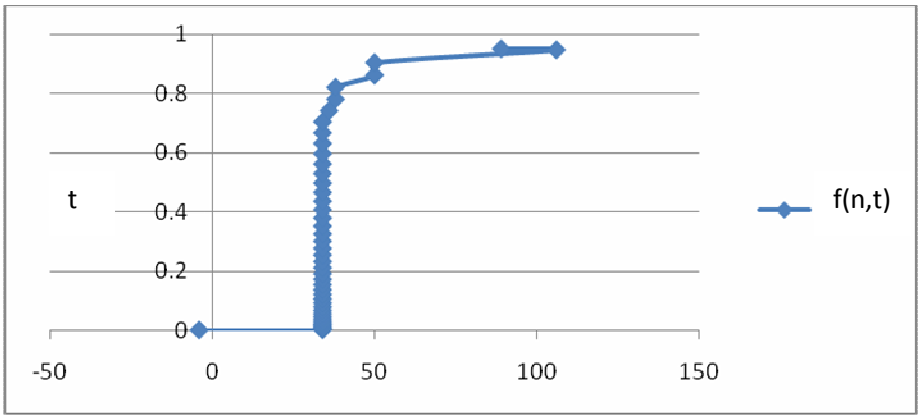


Figure 4.6: *Soundex* Distribution f(t) and t Curve

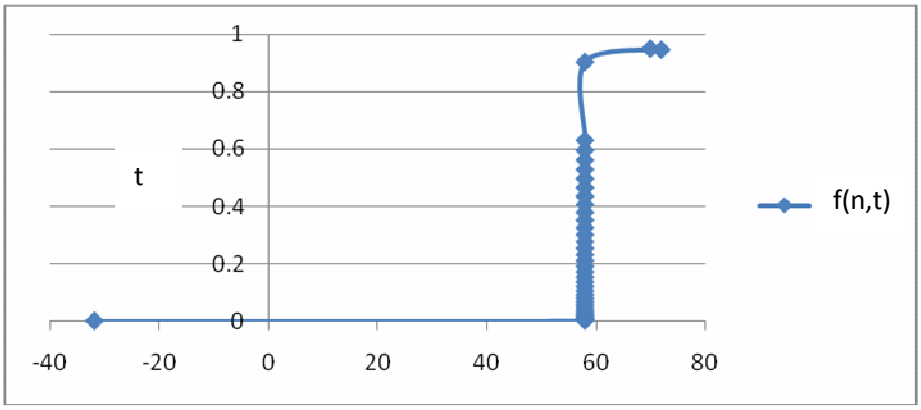


Figure 4.7: *Soft tf-idf* Distribution f(t) and t Curve

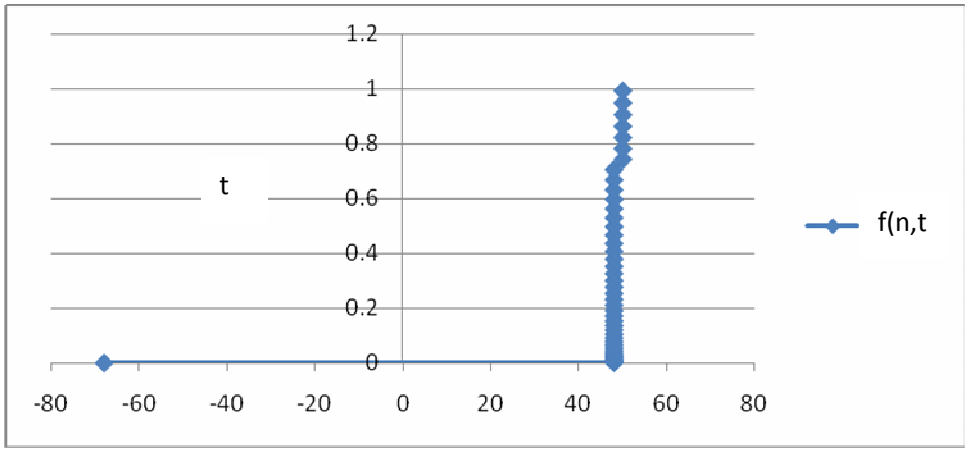


Figure 4.8: *TF-IDF* the distribution f(t) and t curves

- **Blocking Data According to "*surname*" field**

Blocking data according to "surname" the result shown in *table* 4.4.

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	84.0	0.35	[0.43501 , 0.43501]	0.43501
TFIDF	60.0	0.74451	[0.00101 , 0.99001]	0.99001
SoftTFIDF	82.0	0.34167	[0.94601 , 0.94601]	0.94601
Soundex	62.0	0.25834	[0.99001 , 0.99001]	0.99001

Table 4.4: *discernability* calculation according to "SURNAME" field.

Column (SURNAME) using Q-Gram |

Duplicate	Dissimilar
1099.0	7716.0

Column (SURNAME) using TFIDF |

Duplicate	Dissimilar
561.0	8254.0

Column (SURNAME) using SoftTFIDF |

Duplicate	Dissimilar
1100.0	7715.0

Column (SURNAME) using Soundex |

Duplicate	Dissimilar
908.0	7907.0

Table 4.5: Duplicate rate Calculations according to the discernability results in table 4.4.

The best used function in analyzing dataset was *Q-gram* and *soft-TFIDF* while the worst one was *TFIDF*. This can be confirmed by observing the plots in figures 4.9, 4.10, 4.11, 4.12 which show the distribution of srel and sirrel. Indeed, the best separation between relevant and irrelevant data items was achieved by soundex, whilst with TFIDF these items are often shuffled and/or too close together in the ranking. Table 4 shows that the duplicate

deduction for functions are the best. This can be confirmed by observing the plots in figure 4.13,4.14,4.15,4.16 which show the number of points achieved by that threshold interval.

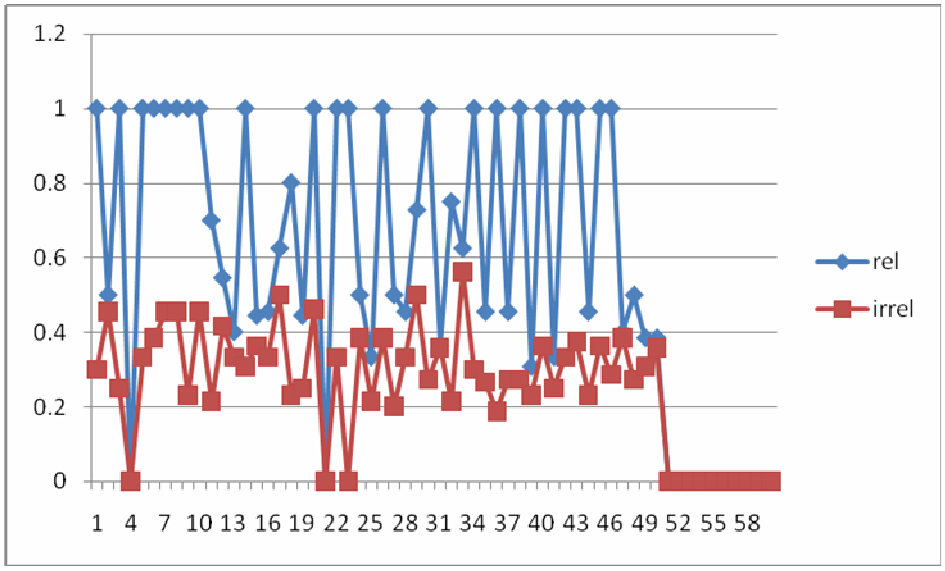


Figure 4.9: distribution relevant and irrelevant as function of the kth block for function Q-gram.

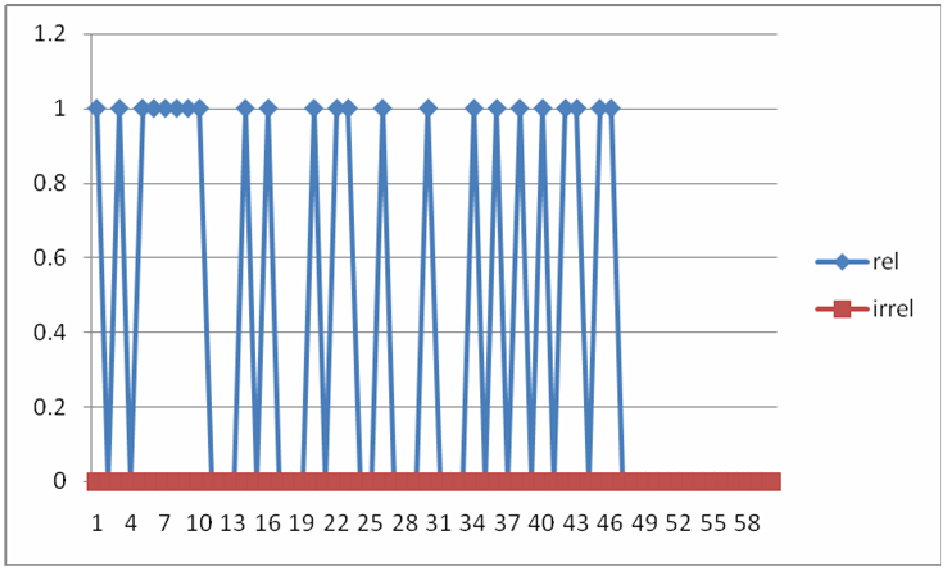


Figure 4.10: distribution relevant and irrelevant as function of the kth block for function TF-IDF plot.

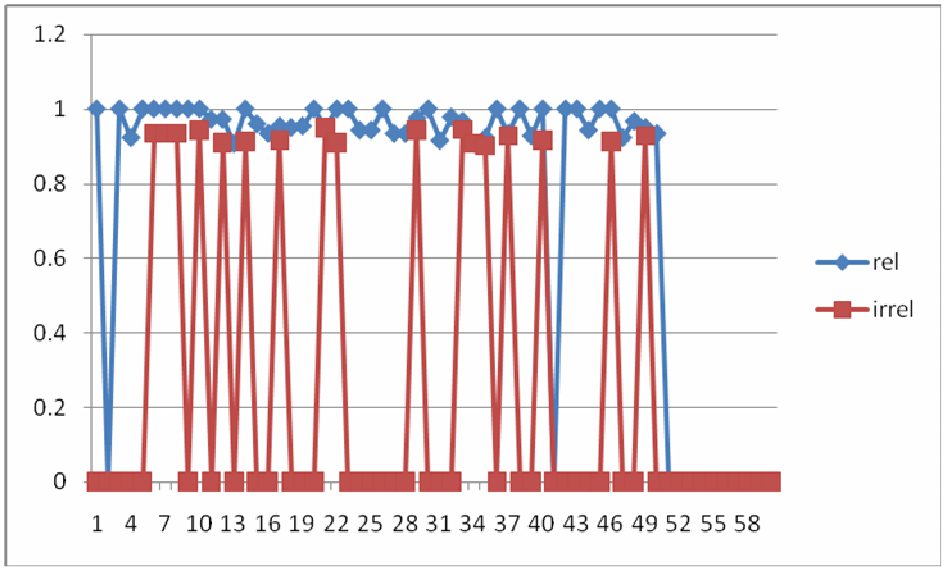


Figure 4.11: distribution relevant and irrelevant as function of the kth block for function *soft tfidf* plot.

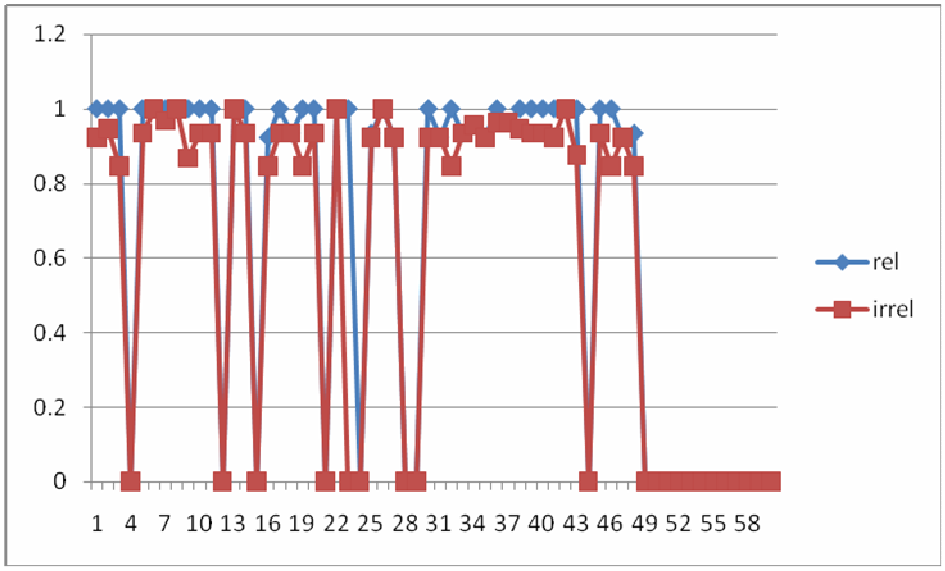


Figure 4.12: distribution relevant and irrelevant as function of the kth block for function soundex plot.

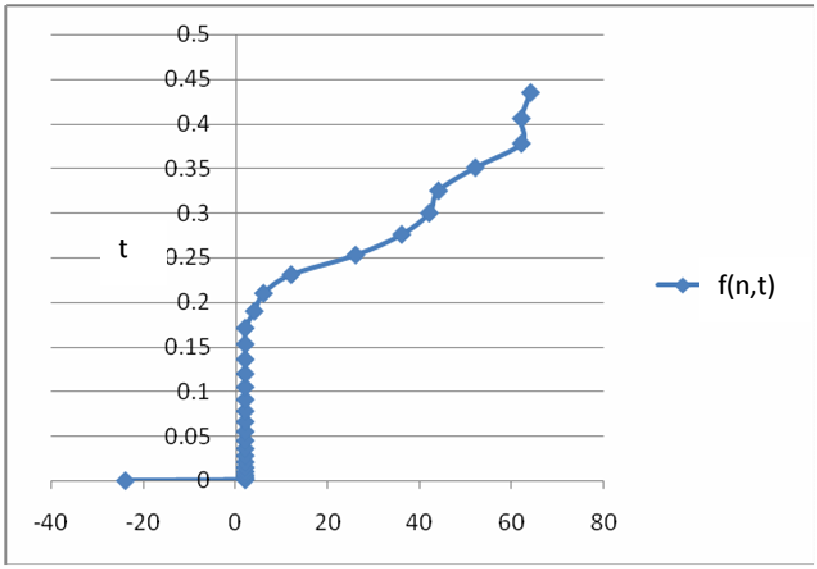


Figure 4.13: *Q-gram* Distribution f(t) and t Curve

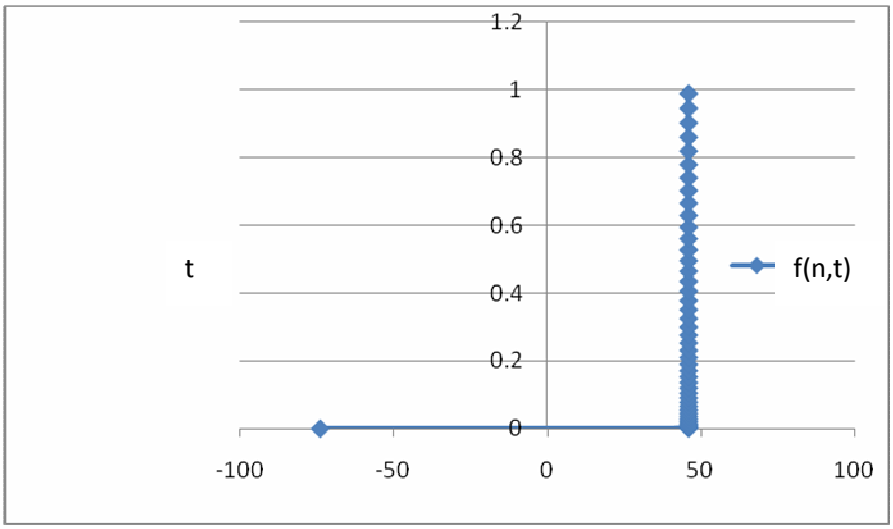


Figure 4.14: *Tfidf* Distribution f(t) and t Curve

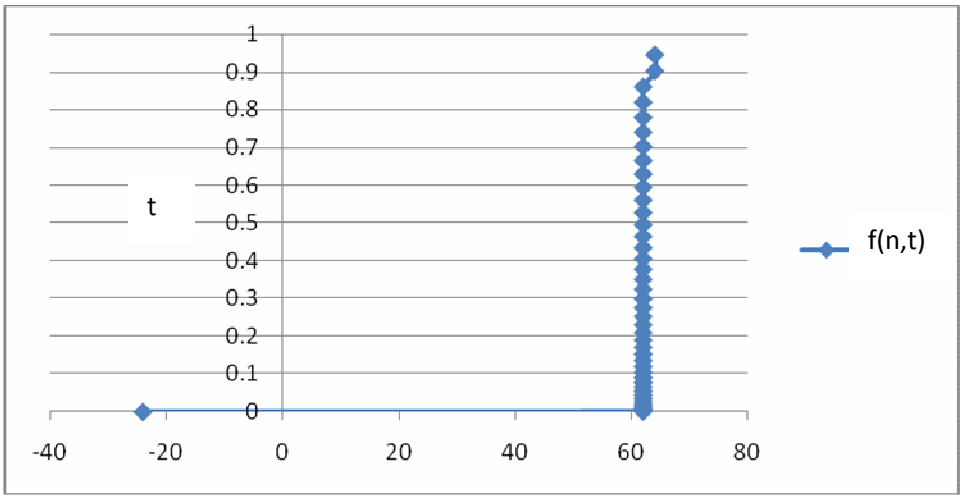


Figure 4.15: *Soft Tfidf* Distribution f(t) and t Curve

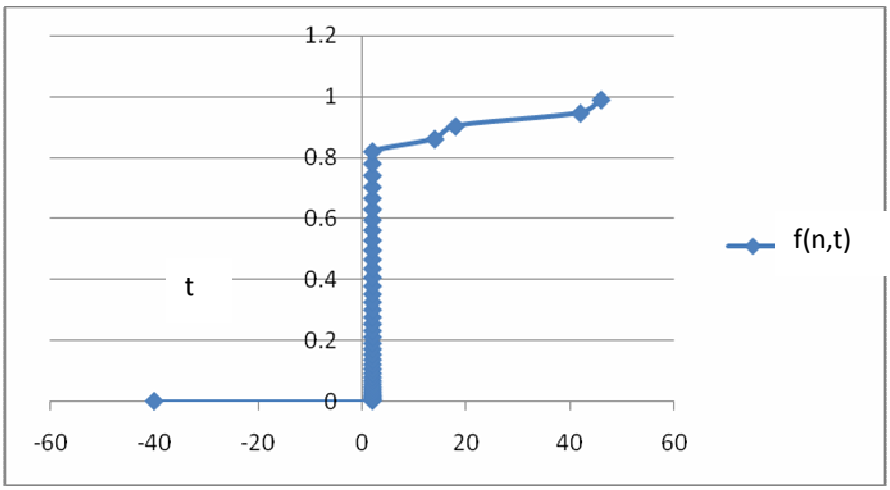


Figure 4.16: *Soundex* Distribution f(t) and t Curve

- **Blocking Data According to "ADDRESS" Field**

Results are shown in table 4.6.

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	92.0	0.38334	[0.46501 , 0.46501]	0.46501
TFIDF	60.0	0.48101	[0.52801 , 0.99001]	0.99001
SoftTFIDF	74.0	0.35934	[0.52801 , 0.63001]	0.63001
Soundex	74.0	0.30834	[0.94601 , 0.94601]	0.94601

Table 4.6: Data Blocking Results according to "ADDRESS" field.

Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1788.0	7027.0

Column (ADDRESS_1) using TFIDF |

Duplicate	Dissimilar
580.0	8235.0

Column (ADDRESS_1) using SoftTFIDF |

Duplicate	Dissimilar
1190.0	7625.0

Column (ADDRESS_1) using Soundex |

Duplicate	Dissimilar
1236.0	7579.0

Table 4.7: Duplicate Calculations according to the discernability results in table 4.6

According to table 6, the best real function for the data set analyzed was *Q-gram* and the worst was *TFIDF*. This can be confirmed by observing the plots in figures 4.17, 4.18, 4.19, 4.20 which show the distribution of srel and sirrel. Indeed, the best separation between relevant and irrelevant data items was achieved by *Q-gram*, whilst with *TFIDF* these items are often shuffled and/or too close together in the ranking. Table 6 shows that the duplicate deduction for best function is the best and the worst function calculated the lower rate of duplicate. This can be confirmed by observing the plots in figures 4.21, 4.22, 4.23, 4.24 which show the number of points achieved by that threshold interval that means the best precision and adequate function.

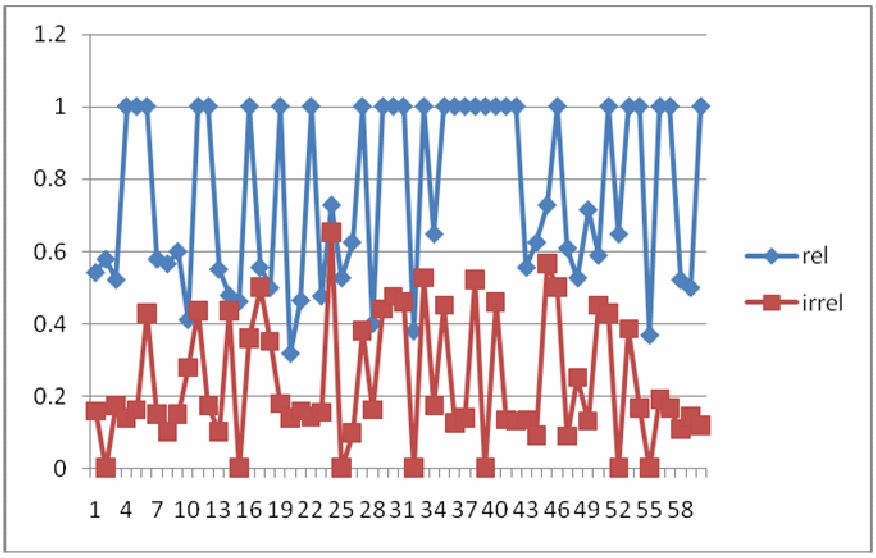


Figure 4.17: distribution relevant and irrelevant as function of the kth block for function Q-gram plot.

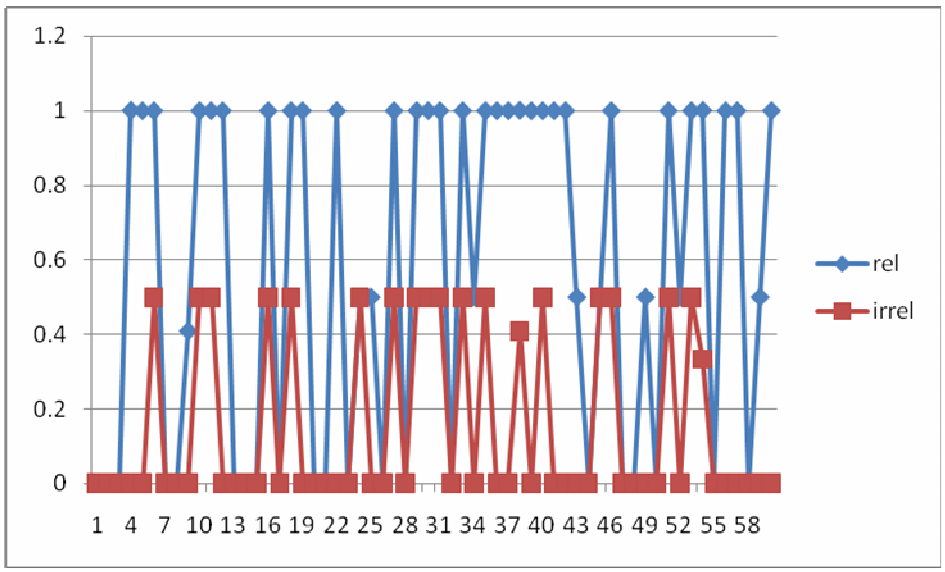


Figure 4.18: distribution relevant and irrelevant as function of the kth block for function *TFIDF* plot.

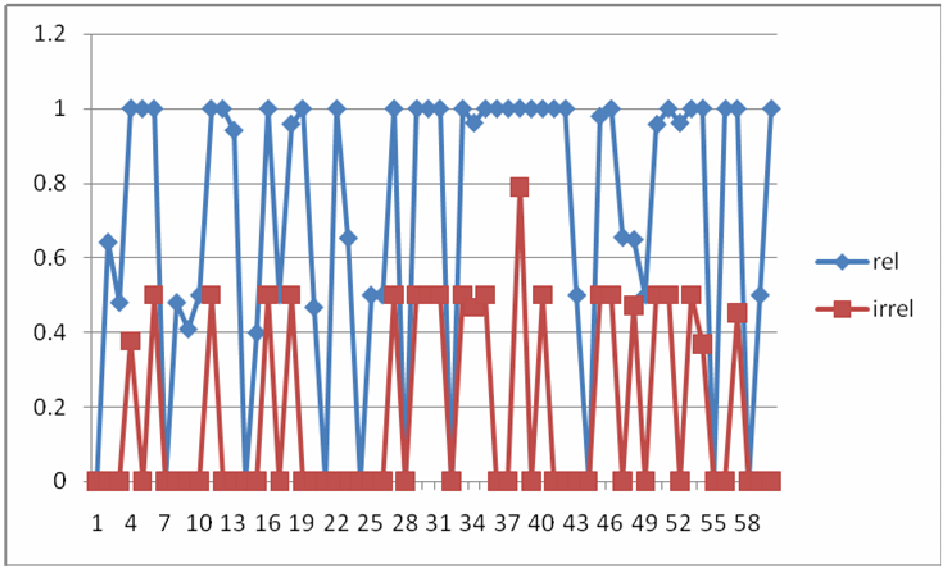


Figure 4.19: distribution relevant and irrelevant as function of the kth block for function *soft tfidf* plot.

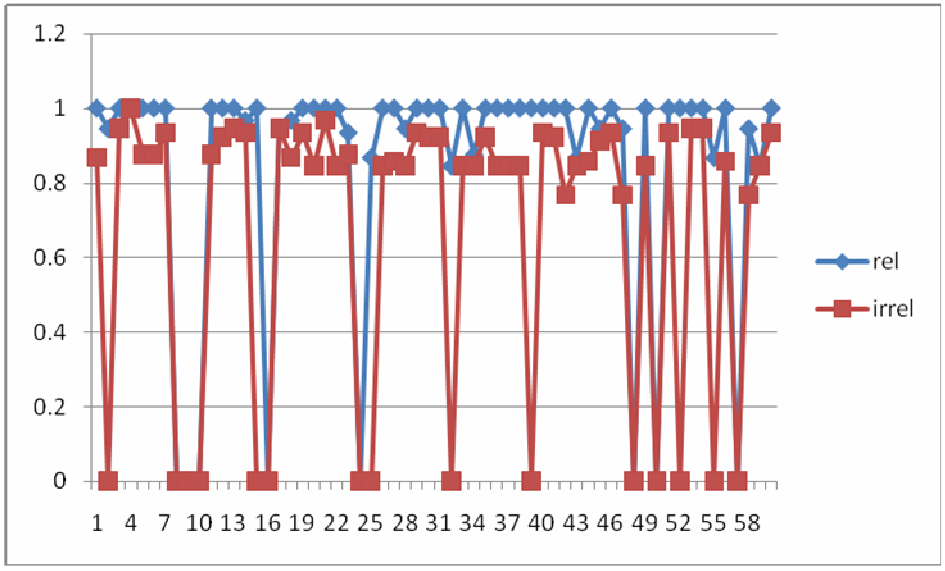


Figure 4.20: distribution relevant and irrelevant as function of the kth block for function soundex plot.

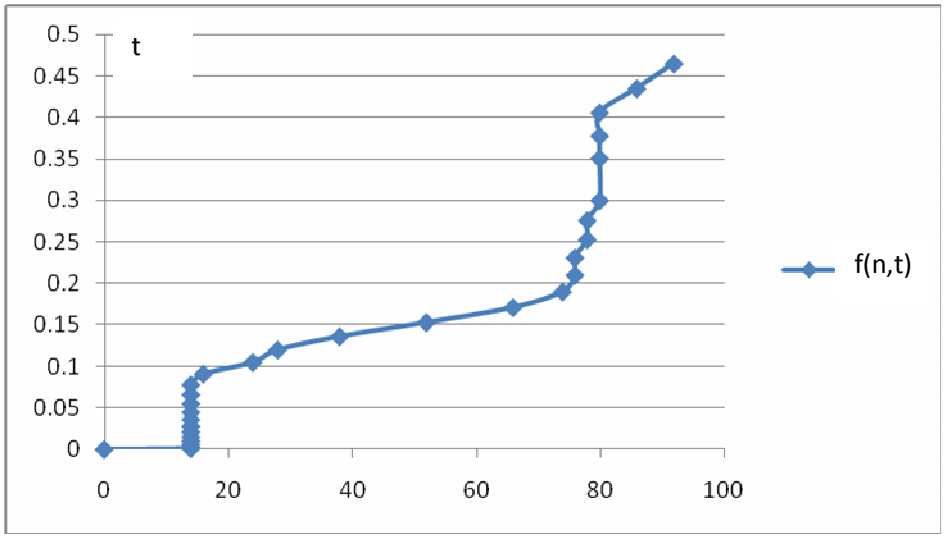


Figure 4.21: Q-gram Distribution f(t) and t Curve

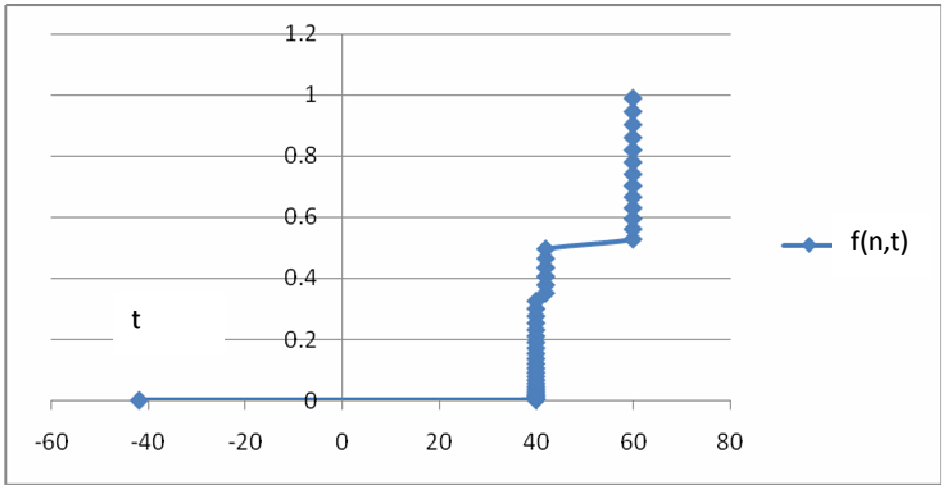


Figure 4.22: TFIDF Distribution f(t) and t Curve

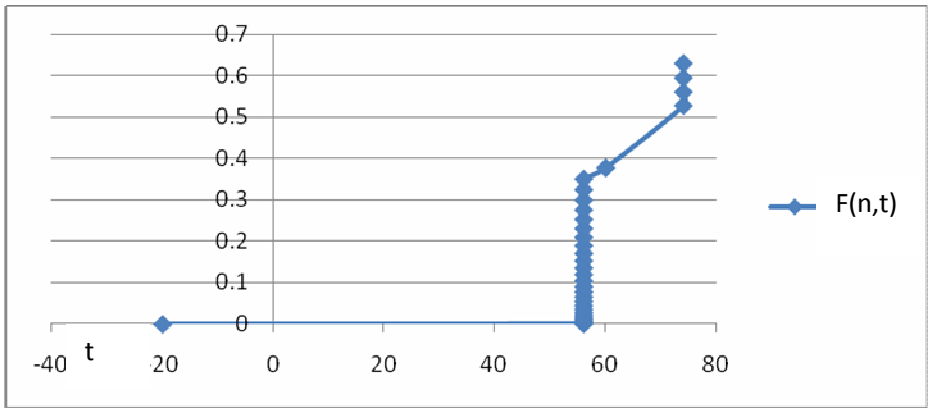


Figure 4.23: TFIDF Distribution f(t) and t Curve

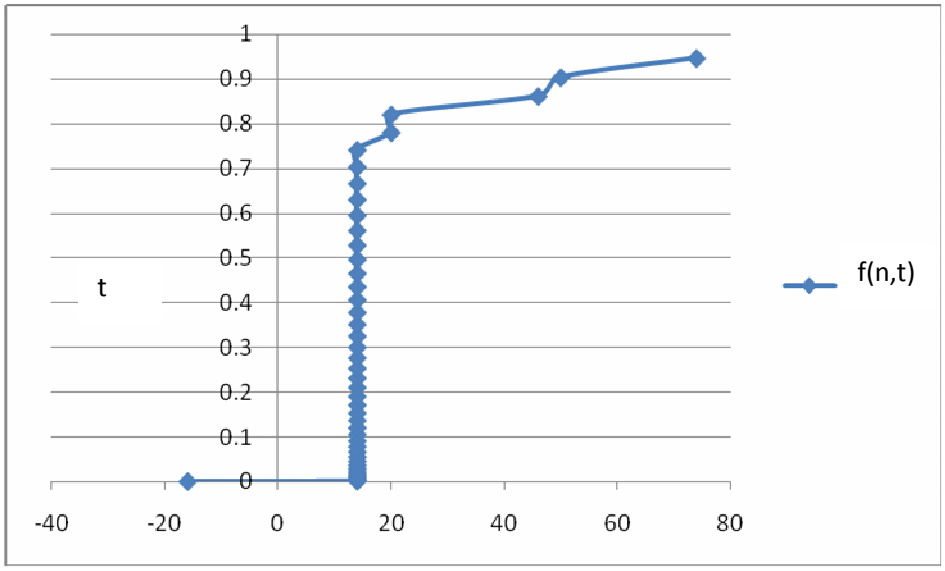


Figure 4.24: Soundex Distribution f(t) and t Curve

4.2.1.2 Blocking Data According to Record while changing blocking threshold value to (0.75)

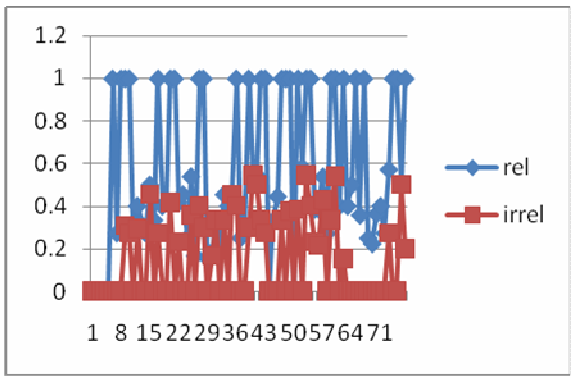
The Second stage for our experiments implementations was through taking certain "record" instead to "field" for making the necessary data blocking. Comparing results for this experiment with experment in (4.2.1.1) the results were shown in the tables 8,9,10, and confirmed in figures 25,26.27,28 29,30,31,32,33,34,35,36.

GIVEN_NAME

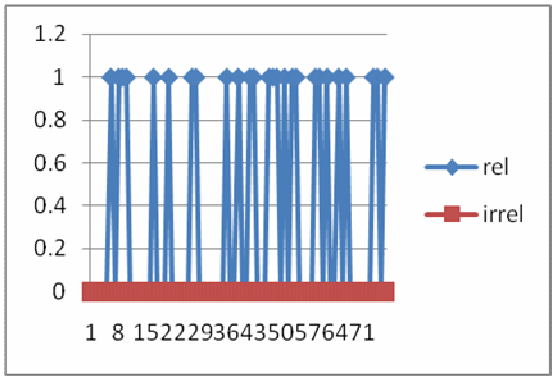
Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	74.0	0.25376	[0.35101 , 0.37801]	0.37801
TFIDF	52.0	0.66334	[0.00101 , 0.99001]	0.99001
SoftTFIDF	82.0	0.58074	[0.00101 , 0.63001]	0.63001
Soundex	84.0	0.29373	[0.86101 , 0.90301]	0.90301

Table4. 8: Comparisons among different similarity functions to "GIVEN_NAME" field, blocking according to record.

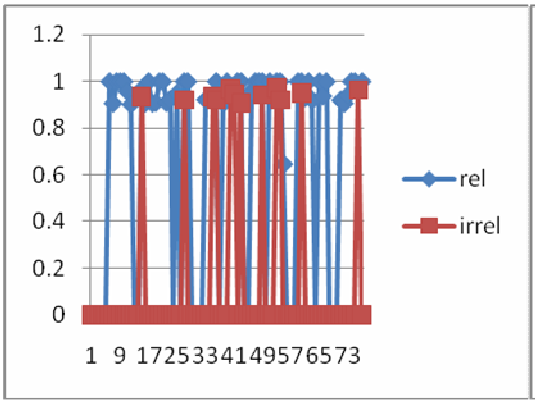
Results shown in table 8 shows that the Soundex is the best function for the Given-Name, means that it doesn't violate the results in table 1.



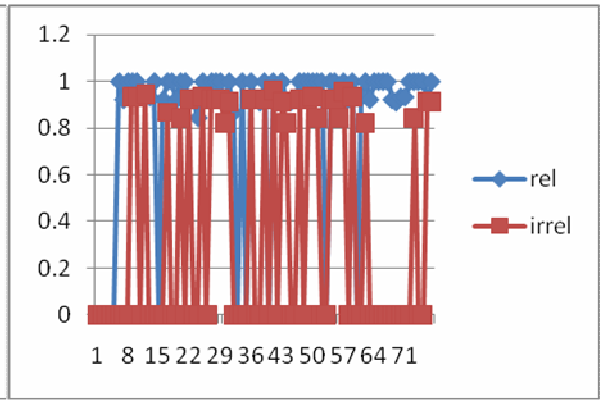
Figures 4.25: Q-gram



figures 4.26: TFIDF



Figures 4.27: Soft TFIDF



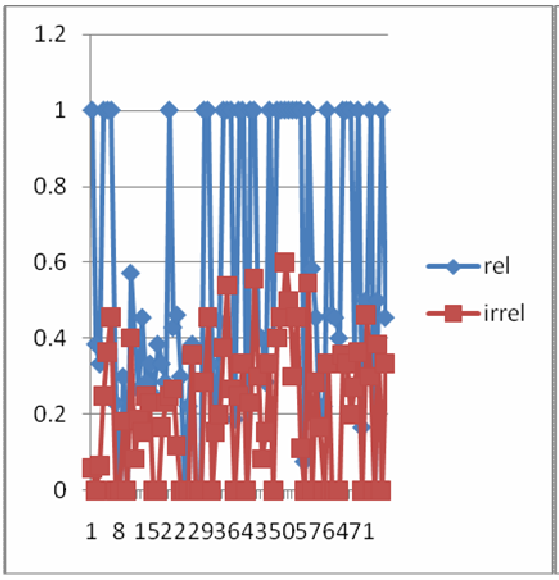
figures 4.28: Soundex

SURNAME

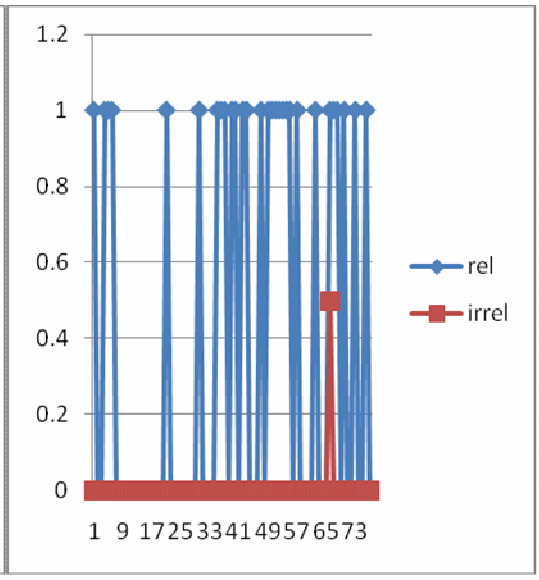
Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	74.0	0.24026	[0.37801 , 0.37801]	0.37801
TFIDF	58.0	0.41932	[0.52801 , 0.99001]	0.99001
SoftTFIDF	72.0	0.28477	[0.52801 , 0.63001]	0.63001
Soundex	50.0	0.16234	[0.99001 , 0.99001]	0.99001

Table 4. 9: Comparisons among different similarity functions to "SURNAME" field, blocking according to record.

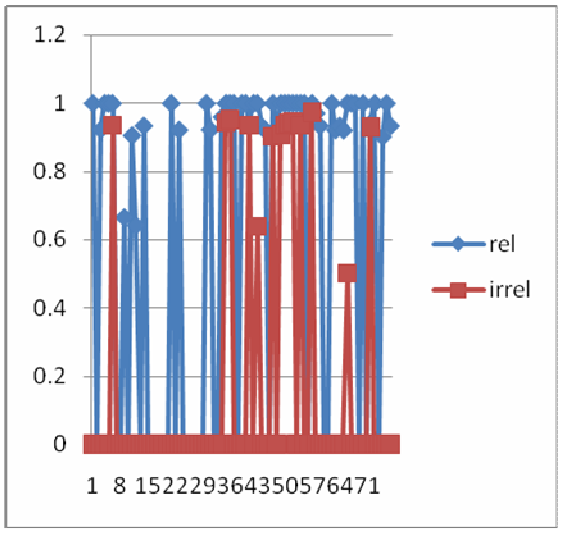
Results were shown in table 4.9 which shows that Q-gram and Soft TFIDF are the best function for this field "Surname", this result matches the results of table 4.3.



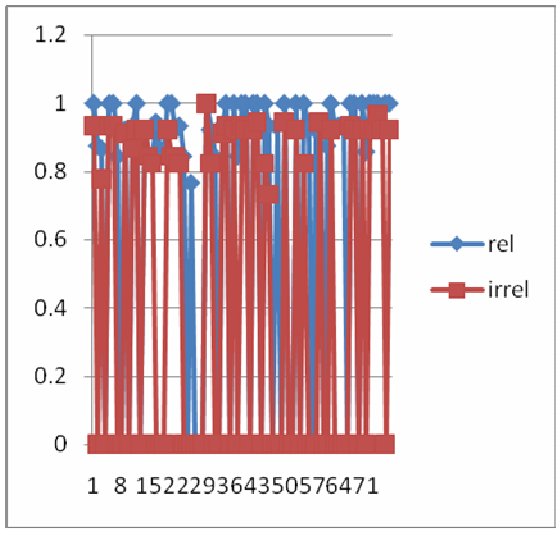
Figures 4.29: Q-gram



figures 4.30: TFIDF



Figures 4.31: Soft TFIDF



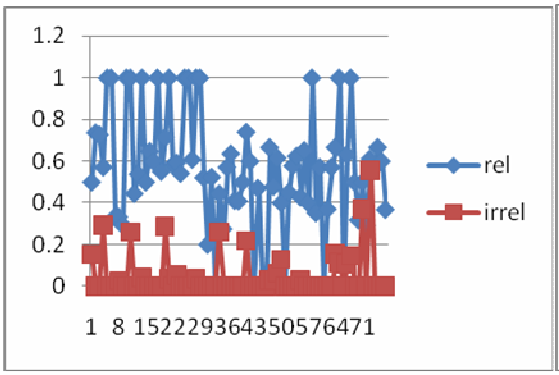
figures 4.32: Soundex

ADDRESS_1

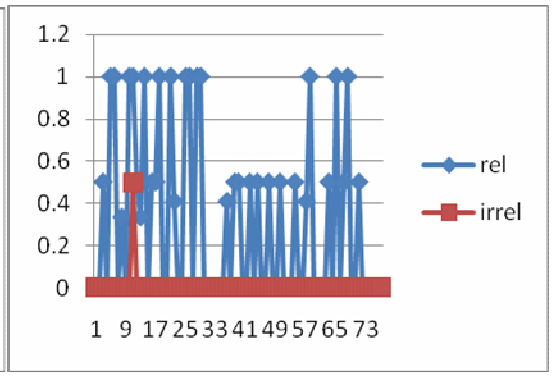
Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	128.0	0.42759	[0.27601 , 0.30001]	0.30001
TFIDF	64.0	0.3698	[0.00101 , 0.32501]	0.32501
SoftTFIDF	108.0	0.55315	[0.00101 , 0.40601]	0.40601
Soundex	120.0	0.38962	[0.74101 , 0.74101]	0.74101

Table 4.10: Comparisons among different similarity functions to "ADDRESS" field, blocking according to record.

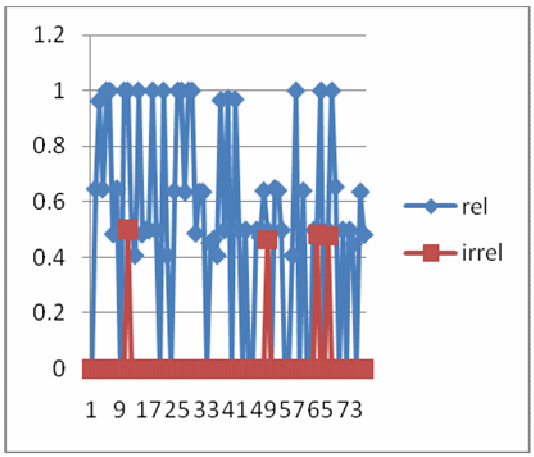
Results shown in table 4.10 show that Q-gram is the best function for this field "Address" that matches the results of table 4.6.



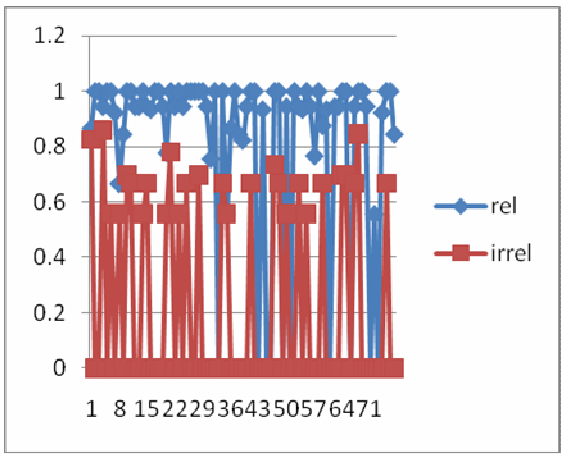
figures 4.33: Q-gram



figures 4.34: TFIDF



Figures 4.35: Soft TFIDF



figures 4.36: Soundex

4.3 Finding

Based on the above results, duplicate numbers in our decisions model are shown below in table 4.11. Ranking the used similarity functions were according to the used "field" and duplicate rate calculation was according to different similarity function to each determined filed. Those results approved the first stage results, where *Soundex* function was the best one for the "*Given-Name*", "*Hybrid*" and "*Q-gram*" were better for the "Surename", and *Q-gram* was the best function to use for "Address". *TFIDF* similarity function was the worst one to use for all fields and we show that the *Q-gram* and *soundax* functions calculate the same duplicate rate but *Q-gram* is not adequate because it fails to correctly separate relevant item from irrelevant item in blocks in a consistent manner. Consequently, this similarity function calculates items as duplicate where they are not duplicated and vice versa.

****Column (GIVEN_NAME) using Soundex | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Q-Gram | best**

Duplicate	Dissimilar
1574.0	7241.0

***Column (GIVEN_NAME) using Q-Gram | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Q-Gram |**

Duplicate	Dissimilar
1390.0	7425.0

Column (GIVEN_NAME) using TFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1158.0	7657.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1381.0	7434.0

Column (GIVEN_NAME) using Soundex | Column (SURNAME) using TFIDF | Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1443.0	7372.0

****Column (GIVEN_NAME) using Soundex | Column (SURNAME) using SoftTFIDF | Column (ADDRESS_1) using Q-Gram | best**

Duplicate	Dissimilar
1584.0	7231.0

Column (GIVEN_NAME) using Soundex | Column (SURNAME) using Soundex | Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1516.0	7299.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Q-Gram |

Duplicate	Dissimilar
1381.0	7434.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using TFIDF |

Duplicate	Dissimilar
874.0	7941.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using SoftTFIDF |

Duplicate	Dissimilar
1182.0	7633.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using Q-Gram | Column (ADDRESS_1) using Soundex |

Duplicate	Dissimilar

1330.0	7485.0

Column (GIVEN_NAME) using TFIDF | Column (SURNAME) using TFIDF | Column (ADDRESS_1) using TFIDF | bad

Duplicate	Dissimilar
498.0	8317.0

Column (GIVEN_NAME) using SoftTFIDF | Column (SURNAME) using SoftTFIDF | Column (ADDRESS_1) using SoftTFIDF |

Duplicate	Dissimilar
1185.0	7630.0

Column (GIVEN_NAME) using Soundex | Column (SURNAME) using Soundex | Column (ADDRESS_1) using Soundex |

Duplicate	Dissimilar
1463.0	7352.0

Table 4.11: Duplicate Rate Results for the Second Stage of Experiments

4.4 Discussion

Da silva (2007) proposes measure called *discernability*, which was used to compare a number of similarity functions applied to an experimental data set to measure quality of similarity function, Results of his study can be summarized as follows:

One similarity function can be considered better than another if it provides better separation of relevant and irrelevant data items returned in response to a query.

According to his approach, a similarity function that has a higher *fmax* is considered better than another function that has a smaller *fmax*. Also, the size of the range of the interval for *tbest* is another indicator of the quality of the function. In our approach we used the same measure on blocks of data set instead of query .we found that, a similarity function that has

a higher fmax is considered better than another function that has a smaller fmax so the similarity function which has high *fmax*,it calculates high rate of duplicate but the size of the range of the interval for tbest is not presented another indicator because the value of $t_{max\_est} = t_{min\_est}$, mostly .

In our approach and da silva (2007) the problems are:

1- Process relies on human intervention.

2-What should be the size of the sample used for the evaluation?

3-Quality of a similarity function varies with different data sets.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5.1Conclusion

Similarity functions is a mean processes in several data management application such as duplicate detaction and similarity query to defining whether two data represent the same real world .similarity function return score if similarity score greater than define threshold the objects are same ,there are wide range of similarity function . It is difficult choose suitable function in this thesis we used a measure called disernabilety(ability of the similarity function)which was used to compare a number of similarity functions applied to an experimental data set to chose adequate function in duplicate detection application.

The results of this thesis emphasize the fact that the Discernability method in addition the Query similarity can be applied in duplicate detection

Based on the results of this study, determining the adequate function to matching duplicate can be done in an early stage using the Duplicate Detection process instead of reaching the last stage to determine the adequate function using the Precision/ Recall method.

Therefore, the accurate result means that similarity function has succeeded to separate relevant from irrelevant items so that the function which calculates high duplicate rate is not necessarily the most suitable function for this field or record. This is due to the fact that the similarity function may provide inaccurate information by determining some objects as relevant while they are irrelevant leading to a risk in the accuracy of the database which

may have a negative effect on the integrity and precision of data specially in sensitive industries such as medical data.

When using the Discernability measure, it enables the user to utilize the used data based on his/ her needs. Therefore, the user will have the option to decide whether he/ she needs the data to be accurate or with a number of recall precision or a balance of both.

5.2 Future Work

Our work on quality of similarity function is how to choose the adequate function. The main problem in the study lies in the need for the human intervention which is needed to identify relevant from irrelevant data items since the process is semi- automatic. Based on results of Santos et al, (2010), an automatic method is proposed and might be possibly employed. Other suggestions are:

- Using other similarity function or a hybrid of sundex and characteristic functions can be focused at in future work to provide more accurate results in the different fields.

-The traditional blocking method was used in this thesis, however; other blocking methods can be used as well.

- The study used content-based approach. Thus, it is recommended to conduct research using the structural – based approach or mixing both approaches.

- In current study, the researcher used English dataset. It is recommended to conduct future studies loc using on Arabic dataset.

REFERENCES

.

ALdummor, R.M. 2010. Performance Evaluation of Blocking Methods for Duplicate Record Detection, *Department of Computer Information Systems, Faculty of Information Technology, Middle East University*, Amman, Jordan.

AL-khalifa, S. YU, C. & JAGADISH, HV. *Querying Structured Text In An Xml Database*. In: Proceedings of the 29th ACM SIGMOD international conference on management of data (SIGMOD). 2003. ACM.

BATINI, C. and SCANNAPIECO, M. 2006. Data Quality: Concepts, *Methodologies and Techniques, Springer*. USA.

Baeza-Yates, R.A., Baeza-Yates, R., Ribeiro-Neto, B. 1999. Modern Information Retrieval. *Addison-Wesley Longman Publishing Co., Inc.* Boston, MA, USA.

Bilenko M. 2003. *Learnable Similarity Functions and Their Applications to Record Linkage and Cluttering*. PHD proposal. Department of Computer Science. University of Texas. Austin.

Broder, A. 1997. On The Resemblance and Containment of Documents. In: Proceedings of the compression and complexity of sequences (SEQUENCES). *IEEE Computer Society*, Washington.

Chapman, S. 2004. SimMetrics: a Java and C# .NET Library of Similarity Metrics. Available at: http://sourceforge.net/projects/simmetrics/ ,
Accessed in: 13th March 2011.

CHAUDHURI, S. **GANTI**, V. and **KAUSHIK**, **R.** ***A*** *primitive operator for similarity joins in data cleaning*. In: Proceedings of the 22nd international conference on data engineering (ICDE). 2006. Washington, IEEE Computer Society.

Christen, P. and Goiser, K. Quality and Complexity Measures for Data Linkage and Deduplication. *Quality Measures in Data Mining, Studies in Computational Intelligence*. **43**. pp. 127-151.

Cohen, W. Ravikumar, P. and Fienberg, S. 2003. *A Comparison of String Metrics for Matching Names and Records*. In Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation. 2003a. Washington, DC.

Dorneles C.F. Goncalves, R. and Mello, R.D.S. 2010. Approximate data instance matching: a survey. *Springer*. **27** (1). London.

Da Silva, R. K. Stasiu, V. M. Orengo, and C. A. Heuser.2007 Measuring quality of similarity functions approximate data matching. *Journal of Informetrics*,1(1):35{46, January 2007.

Elfeky, M. Vevykios, V. and Elmagarmid, A. *TAILOR: A Record Linkage Toolbox*. In: Proceedings of the 18th Int. Conf. on Data Engineering. 2002. IEEE.

Elmagarmid, A.K.[1] Ipeirotis, P.G.[2] and Verykios, V.S.[3] 2007. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*. **19**(1). [1]Senior Member, IEEE. [2]Member, IEEE Computer Society. [3]Member, IEEE Computer Society.

Fellegi, I.P. and Sunter, A.B. 1969. A Theory for Record Linkage. *Journal of the American Statistical Association*. **64**(328). pp. 1183–1210.

GILL, L.E. *OX-LINK: The Oxford Medical Record Linkage System*. Proc. Int'l Record Linkage Workshop and Exposition. 1997.

Gravano, L. Iperrotis, P.G. Jagadissh, H.V. Koudas, N. Muthukrishnan, S. Pietainen, L. and Srivastava, D. 2001. Using Q-Grams in a DBMS for Approximate String Processing. *IEEE*. **24**(4). pp. 28-34.

Gravano, L. Ireirotis, P.G. Koudas, N. and Srivastava, D. 2003. Text Joins In an Rdbms For Web Data Integration. In *WWW 2003*, pages 90{101, New York, NY, USA, 2003. ACM Press.
Available at: http://portal.acm.org/citation.cfm?id=775166

HERNÁNDEZ M. and STOLFO S. 1998. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Journal of Data Mining and Knowledge Discovery*. **2** (1). pp. 9 – 37.

Herzog, TH.N. Scheuren, F.J. and Winkleir W.E. 2007. *Data Quality and Record Linkage Techniques*. Springer.

Heuser, C.A. Krieser, F.N.A. and Orengo, V.M. 2007. SimEval - A Tool for Evaluating the Quality of Similarity Functions. *UFRGS - Instituto de Informtica*. Brazil, Copyright ©2007, Australian Computer Society, Inc.

Joachims, T. 1999 .*Transductive Inference for Text Classification Using Support Vector Machines*. In Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99). 1999. Bled, Slovenia.

Koudas, N. Marathe, A. and Srivastava. D. 2004. *Flexible String Matching Against Large Databases in Practice*. In VLDB. 2004. Toronto, Canada.

Kukich, K. 1992. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*. **24**(4). pp. 377-439.

Levenshtein. 1966. Binary Codes Capable of Correcting Insertions and Reversals. *Soviet Physics Doklady*. **10** .pp. 707–710.

Lim, E. Srivastava, J. Prabhakar, S. and Richrdson, J. 1996. *Entity Identification in Database Integration*. *IEEE*. **89**(1-2). pp. 1–38

Naumann, F. and Herrschel, M. 2010. *An Introduction to Duplicate Detection*. USA. Morgan & Claypool Publishers.

Newcombe, H.B Kennedy, J.M. Axford, S.J. and James, A.P. 1959. *Automatic linkage of vital records. Science*. **130** (3381). pp. 954–959.

Newcombe, H.B. 1967. Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories. *Am. J. Human Genetics.* **19** (3). pp. 335-359.

Philips, L. 1990. Hanging on the Metaphone. *Computer Language Magazine*. **7**(12). pp. 39-44. Dec. 1990.
Available at: http://www.cuj.com/documents/s=8038/cuj0006philips/.

Russell Index, R.C. U.S. Patent 1,261,167,
Available at: http://www.patft.uspto. gov/netahtml/srchnum.htm

Sutinen, E. and Tarhio, J. On Using Q-Gram Locations in Approximate String Matching. *Proc. Third Anual European Symposium Algorithms (ESA '95)*. 1995. UK. Springer-Verlag.

Stasiu,R.K., Heuser C.A.,Da Silva,2005, Estimating recall and precision for vague queries in databases, in: *Advanced Information Systems Engineering, 17th International Conference,* , Proceedings, Lecture Notes in Computer Science, vol. 3520, Springer, pp. 187–200

Taft, R.L. 1970. Name Search Techniques. Technical Report Special Report No. 1. *New York State Identification and Intelligence System*. Albany, N.Y.

Ukkonen, E. 1992. Approximate String Matching with q-Grams and Maximal Matches. *Theoretical Computer Science*. 92(1). pp. 191-211.

Ulimann, J.R. 1997. A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion, and Reversal Errors in Words. *The Computer Journal*. 20(2). pp. 141-147.

APPENDICES

Appendix A: Soundex String-Coding Algorithm

Soundex String-Coding Algorithm includes the following steps:

1. Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2. Retain the first letter of the word.
3. Change all occurrence of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
4. Change letters from the following sets into the digit given:
   - 1 = 'B', 'F', 'P', 'V'
   - 2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z'
   - 3 = 'D','T'
   - 4 = 'L'
   - 5 = 'M','N'
   - 6 = 'R'
5. Remove all pairs of digits which occur beside each other from the string that resulted after step (4).
6. Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7. Pad the string that resulted from step (6) with trailing zeros and return only the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

Appendix B
Traditional Blocking Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Blocking;

import java.sql.*;
import ApplicationGlobals.*;

/**
 *
 * @author Administrator
 */
public class Standard {
   public void Standard()
   {}

   public String StartBlocking(double Acceptance) throws SQLException
   {
      String Results = "";
      Statement stmt = null;
      Statement innerStmt = null;
      Statement checkStmt = null;

      String query = "";
      try {
         drdDataBase.DB db = new drdDataBase.DB();
         Connection conn = db.dbConnect();

         query = "update main_data set BLOCKING = null;";
         stmt = conn.createStatement();
         innerStmt = conn.createStatement();
         checkStmt = conn.createStatement();
         innerStmt.execute(query);

         query = "select RECID, KEY1 from main_data order by KEY1 asc;";
         ResultSet rs = stmt.executeQuery(query);

         int nCurrentBlock = 0;

         while (rs.next()) {
            String CurrentKey = rs.getString("KEY1");
            String strCurrentRecoredID = rs.getString("RECID");

            query = "select BLOCKING from main_data where RECID = ".concat(strCurrentRecoredID).concat(";");
            ResultSet rsCheck = checkStmt.executeQuery(query);
            boolean ProcessInnert = false;
            while (rsCheck.next())
            {
               if(rsCheck.getString("BLOCKING") == null)
               {
                  ProcessInnert = true;
               }
            }

            if (ProcessInnert)
```

```
            {
                ProcessInnerSelect(conn, CurrentKey, nCurrentBlock, Acceptance);
                nCurrentBlock = nCurrentBlock + 1;
            }
          }
        }catch (SQLException e ) {
          Results = e.getMessage().concat(",,,").concat(query);
        } finally {
          if (stmt != null)
          {
             stmt.close();
          }
          if (innerStmt != null)
          {
             innerStmt.close();
          }
          return Results;
        }
    }

    private String ProcessInnerSelect(Connection conn, String CurrentKey, int nCurrentBlock, double Acceptance) throws
SQLException
    {
        String Results = "";
        Statement updStmt = null;
        Statement innerStmt = null;
        String query = "";

        try {
           query = "select RECID, KEY1 from main_data where BLOCKING is null order by KEY1 asc";
           innerStmt = conn.createStatement();
           updStmt = conn.createStatement();

           ResultSet rsCompareResultSet = innerStmt.executeQuery(query);
           while (rsCompareResultSet.next())
           {
              String ID = rsCompareResultSet.getString("RECID");
              String KEY = rsCompareResultSet.getString("KEY1");

              Blocking.NW nwBlocking = new Blocking.NW(CurrentKey,KEY);
              nwBlocking.fillScoreArray();

              if( nwBlocking.GetSimResults() >= Acceptance)
              {
                 query = "update main_data set BLOCKING = ".concat(Integer.toString(nCurrentBlock))
                       .concat(" where RECID = ").concat(ID).concat(" and BLOCKING is null");
                 updStmt.execute(query);
              }
           }
           Results = "DONE";
        }catch (SQLException e ) {
          Results = e.getMessage().concat(",,,").concat(query);
        } finally {
          if (innerStmt != null)
          {
             innerStmt.close();
          }
          return Results;
        }
    }
}
```

Needleman –Wunsch Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Blocking;

/**
 *
 * @author Administrator
 */
public class NW {
    // Using simple linear gap score (-2 per indel)
    // and 4 for a match, -1 for a mismatch
    // Feel free to change this
    public static final int gapscore = -2;
    public static final int matchscore = 4;
    public static final int mismatchscore = -1;

    private String x;  // First string
    private String y;  // Second string
    private int xlen, ylen; // their lengths
    private int[][] scoreArray;

    public NW(String a, String b) {
        x = a;
        y = b;
        xlen = x.length();
        ylen = y.length();
        scoreArray = new int[ylen+1][xlen+1];
    }

    public void fillScoreArray() {
        int row, col;    // for indexing through array
        int northwest, north, west;  // (row, col) entry will be max of these
        int best;   // will be the max
        // Fill the top row and left column:
        for (col=0; col <= xlen; col++) scoreArray[0][col] = gapscore*col;
        for (row=0; row <= ylen; row++) scoreArray[row][0] = gapscore*row;
        // Now fill in the rest of the array:
        for (row=1; row <= ylen; row++) {
            for (col=1; col <= xlen; col++) {
                if (x.charAt(col-1)==y.charAt(row-1))
                    northwest = scoreArray[row-1][col-1] + matchscore;
                else northwest = scoreArray[row-1][col-1] + mismatchscore;
                west = scoreArray[row][col-1] + gapscore;
                north = scoreArray[row-1][col] + gapscore;
                best = northwest;
                if (north>best) best = north;
                if (west>best) best = west;
                scoreArray[row][col] = best;
            }
        }
    }

    public int GetOptimumValue()
    {
        int Opt = 0;
```

```
        for (int i = 1; i < scoreArray.length; i++)
        {
           Opt = Opt + scoreArray[i][i];
        }
        return Opt;
    }

    public double GetSimResults() {
        double sim = 0;
        for (int i = 1; i < scoreArray.length; i++)
        {
           sim = sim + scoreArray[i][i];
        }

        NW n = new NW(x,x);
        n.fillScoreArray();
        int Opemam = n.GetOptimumValue();

        return (sim/Opemam);
    }
}
```

Qgram Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package MatchingAlgorithms;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Administrator
 */
public class Qgram
{
   private class result
   {
      private String theWord;
      private int theCount;

      public result(String w, int c)
      {
         theWord = w;
         theCount = c;
      }

      public void setTheCount(int c)
      {
         theCount = c;
      }

      public String getTheWord()
      {
         return theWord;
      }

      public int getTheCount()
      {
         return theCount;
      }
   }

   private List<result> results;

   public Qgram()
   {
      results = new ArrayList<result>();
   }

   public double getSimilarity(String wordOne, String wordTwo)
   {
      if(wordOne.equals(""))
      {
         return 0;
      }
      List<result> res1 = processString(wordOne, 3);
      //displayResult(res1);
```

```
    List<result> res2 = processString(wordTwo, 3);
    //displayResult(res2);
    int c = common(res1,res2);
    int u = union(res1,res2);
    double sim = (double)c/(double)u;

    return sim;
}

private int common(List<result> One, List<result> Two)
{
    int res = 0;

    for (int i = 0; i < One.size(); i++)
    {
        for (int j = 0; j < Two.size(); j++)
        {
            if (One.get(i).theWord.equalsIgnoreCase(Two.get(j).theWord)) res++;
        }
    }

    return res;
}

private int union(List<result> One, List<result> Two)
{
    List<result> t = One;

    for (int i = 0; i < Two.size(); i++)
    {
        int pos = -1;
        boolean found = false;
        for (int j = 0; j < t.size() && !found; j++)
        {
            if (Two.get(i).theWord.equalsIgnoreCase(t.get(j).theWord))
            {
                found = true;
            }
            pos = j;
        }

        if (!found)
        {
            result r = Two.get(i);
            t.add(r);
        }
    }

    return t.size();
}

private List<result> processString(String c, int n)
{
    List<result> t = new ArrayList<result>();

    String spacer = "";
    for (int i = 0; i < n-1; i++)
    {
        spacer = spacer + "%";
    }
    c = spacer + c + spacer;
```

```
        for (int i = 0; i < c.length(); i++)
        {
            if (i <= (c.length() - n))
            {
                if (contains(c.substring(i, n+i)) > 0)
                {
                    t.get(i).setTheCount(results.get(i).getTheCount()+1);
                }
                else
                {
                    t.add(new result(c.substring(i,n+i),1));
                }
            }
        }
        return t;
    }

    private int contains(String c)
    {
        for (int i = 0; i < results.size(); i++)
        {
            if (results.get(i).theWord.equalsIgnoreCase(c))
                return i;
        }
        return 0;
    }

    private void displayResult(List<result> d)
    {
        for (int i = 0; i < d.size(); i++)
        {
            System.out.println(d.get(i).theWord+" occurred "+d.get(i).theCount+" times");
        }
    }
}
```

TFIDF Class

```
package com.wcohen.secondstring;

import java.util.*;
import com.wcohen.secondstring.tokens.*;
```

```
/**
 * TFIDF-based distance metric.
 */

public class TFIDF extends AbstractStatisticalTokenDistance
{
	public TFIDF(Tokenizer tokenizer) { super(tokenizer);	}
	public TFIDF() { super(); }

	public double score(StringWrapper s,StringWrapper t) {
		BagOfTokens sBag = (BagOfTokens)s;
		BagOfTokens tBag = (BagOfTokens)t;
		double sim = 0.0;
		for (Iterator i = sBag.tokenIterator(); i.hasNext(); ) {
	  Token tok = (Token)i.next();
	  if (tBag.contains(tok)) {
				sim += sBag.getWeight(tok) * tBag.getWeight(tok);
			}
		}
		//System.out.println("common="+numCommon+" |s| = "+sBag.size()+" |t| = "+tBag.size());
		return sim;
	}

	/** Preprocess a string by finding tokens and giving them TFIDF weights */
	public StringWrapper prepare(String s) {
		BagOfTokens bag = new BagOfTokens(s, tokenizer.tokenize(s));
		// reweight by tdfidf
		double normalizer = 0.0;
		for (Iterator i=bag.tokenIterator(); i.hasNext(); ) {
			Token tok = (Token)i.next();
			if (collectionSize>0) {
				Integer dfInteger = (Integer)documentFrequency.get(tok);
				// set previously unknown words to df==1, which gives them a high value
				double df = dfInteger==null ? 1.0 : dfInteger.intValue();
				double w = Math.log( bag.getWeight(tok) + 1) * Math.log( collectionSize/df );
				bag.setWeight( tok, w );
				normalizer += w*w;
			} else {
				bag.setWeight( tok, 1.0 );
				normalizer += 1.0;
			}
		}
		normalizer = Math.sqrt(normalizer);
		for (Iterator i=bag.tokenIterator(); i.hasNext(); ) {
			Token tok = (Token)i.next();
			bag.setWeight( tok, bag.getWeight(tok)/normalizer );
		}
		return bag;
	}

	/** Explain how the distance was computed.
	 * In the output, the tokens in S and T are listed, and the
	 * common tokens are marked with an asterisk.
	 */
	public String explainScore(StringWrapper s, StringWrapper t)
	{
//		BagOfTokens sBag = (BagOfTokens)s;
//		BagOfTokens tBag = (BagOfTokens)t;
//		StringBuffer buf = new StringBuffer("");
//		PrintfFormat fmt = new PrintfFormat("%.3f");
```

```
//                      buf.append("Common tokens: ");
//                      for (Iterator i = sBag.tokenIterator(); i.hasNext(); ) {
//              Token tok = (Token)i.next();
//                              if (tBag.contains(tok)) {
//                                      buf.append(" "+tok.getValue()+": ");
//                                      buf.append(fmt.sprintf(sBag.getWeight(tok)));
//                                      buf.append("*");
//                                      buf.append(fmt.sprintf(tBag.getWeight(tok)));
//                              }
//                      }
//                      buf.append("\nscore = "+score(s,t));
//                      return buf.toString();
        return "";
           }

           public String toString() { return "[TFIDF]"; }

           static public void main(String[] argv) {
                   doMain(new TFIDF(), argv);
           }
}
```

Soft FIDF Class

```
package com.wcohen.secondstring;

import java.util.*;
import com.wcohen.secondstring.tokens.*;

/**
 * TFIDF-based distance metric, extended to use "soft" token-matching.
 * Specifically, tokens are considered a partial match if they get
 * a good score using an inner string comparator.
 *
 * <p>On the WHIRL datasets, thresholding JaroWinkler at 0.9 or 0.95
 * seems to be about right.
 */

public class SoftTFIDF extends TFIDF
{
	// distance to use to compare tokens
	private StringDistance tokenDistance;
	// threshold beyond which tokens are considered a match
	private double tokenMatchThreshold;
	// default token distance
	private static final StringDistance DEFAULT_TOKEN_DISTANCE = new JaroWinkler();

	public SoftTFIDF(Tokenizer tokenizer,StringDistance tokenDistance,double tokenMatchThreshold) {
		super(tokenizer);
		this.tokenDistance = tokenDistance;
		this.tokenMatchThreshold = tokenMatchThreshold;
	}
	public SoftTFIDF(StringDistance tokenDistance,double tokenMatchThreshold) {
		super();
		this.tokenDistance = tokenDistance;
		this.tokenMatchThreshold = tokenMatchThreshold;
	}
	public SoftTFIDF(StringDistance tokenDistance) {
		this(tokenDistance, 0.9);
	}

	public void setTokenMatchThreshold(double d) { tokenMatchThreshold=d; }
	public void setTokenMatchThreshold(Double d) { tokenMatchThreshold=d.doubleValue(); }
	public double getTokenMatchThreshold() { return tokenMatchThreshold; }

	public double score(StringWrapper s,StringWrapper t) {
		BagOfTokens sBag = (BagOfTokens)s;
		BagOfTokens tBag = (BagOfTokens)t;
		double sim = 0.0;
		for (Iterator i = sBag.tokenIterator(); i.hasNext(); ) {
	    Token tok = (Token)i.next();
	    if (tBag.contains(tok)) {
				sim += sBag.getWeight(tok) * tBag.getWeight(tok);
			} else {
				// find best matching token
				double matchScore = tokenMatchThreshold;
				Token matchTok = null;
				for (Iterator j=tBag.tokenIterator(); j.hasNext(); ) {
					Token tokJ = (Token)j.next();
					double distItoJ = tokenDistance.score( tok.getValue(), tokJ.getValue() );
					if (distItoJ>=matchScore) {
						matchTok = tokJ;
```

```
                                                matchScore = distItoJ;
                                                }
                                        }
                                        if (matchTok!=null) {
                                                sim += sBag.getWeight(tok) * tBag.getWeight(matchTok)
* matchScore;
                                        }
                                }
                        }
                        //System.out.println("common="+numCommon+" |s| = "+sBag.size()+" |t| = "+tBag.size());
                        return sim;
                }

                /** Explain how the distance was computed.
                 * In the output, the tokens in S and T are listed, and the
                 * common tokens are marked with an asterisk.
                 */
                public String toString() { return "[SoftTFIDF thresh="+tokenMatchThreshold+";"+tokenDistance+"]"; }
}
```

Soundex Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package MatchingAlgorithms;

/**
 *
 * @author Administrator
 */
public class Soundex {
    public Soundex()
    {}

    public double Calculate(String A, String B)
    {
        uk.ac.shef.wit.simmetrics.similaritymetrics.Soundex objSoundex = new
uk.ac.shef.wit.simmetrics.similaritymetrics.Soundex();
        return (double)objSoundex.getSimilarity(A, B);
    }
}
```

Appendix C:

Similarity Functions' Results for the "GIVEN-NAME" field while changing Blocking Numbers

When n=60

Discernability GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	76.0	0.31667	[0.43501 , 0.43501]	0.43501
TFIDF	50.0	0.33284	[0.74101 , 0.99001]	0.99001
SoftTFIDF	72.0	0.3	[0.94601 , 0.94601]	0.94601
Soundex	106.0	0.44167	[0.94601 , 0.94601]	0.94601

Column (GIVEN_NAME) using Soundex |

Duplicate	Dissimilar
1023.0	7792.0

When n=50

Discernability GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	58.0	0.24167	[0.43501 , 0.43501]	0.43501
TFIDF	42.0	0.29951	[0.74101 , 0.99001]	0.99001
SoftTFIDF	58.0	0.24167	[0.94601 , 0.94601]	0.94601
Soundex	90.0	0.375	[0.94601 , 0.94601]	0.94601

When n=40

Discernability GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	46.0	0.19167	[0.43501 , 0.43501]	0.43501
TFIDF	34.0	0.26617	[0.74101 , 0.99001]	0.99001
SoftTFIDF	48.0	0.20001	[0.94601 , 0.94601]	0.94601
Soundex	72.0	0.3	[0.94601 , 0.94601]	0.94601

When n=30

Discernability
GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	32.0	0.13334	[0.43501 , 0.43501]	0.43501
TFIDF	24.0	0.22451	[0.74101 , 0.99001]	0.99001
SoftTFIDF	32.0	0.13334	[0.94601 , 0.94601]	0.94601
Soundex	56.0	0.23334	[0.94601 ,0.94601]	0.94601

When n=25

Discernability
GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	26.0	0.18784	[0.27601 , 0.43501]	0.43501
TFIDF	20.0	0.20784	[0.74101 , 0.99001]	0.99001
SoftTFIDF	26.0	0.10834	[0.94601 , 0.94601]	0.94601
Soundex	46.0	0.21367	[0.94601 , 0.99001]	0.99001

Column (GIVEN_NAME) using Soundex |

Duplicate	Dissimilar
1010.0	7805.0

When n=20

GIVEN_NAME

Function	F-max	Discernability	[T-Min , T-Max]	T-Best
Q-Gram	24.0	0.10001	[0.37801 , 0.37801]	0.37801
TFIDF	18.0	0.56951	[0.00101 , 0.99001]	0.99001
SoftTFIDF	22.0	0.09167	[0.94601 , 0.94601]	0.94601
Soundex	36.0	0.17201	[0.94601 , 0.99001]	0.99001

Column (GIVEN_NAME) using Q-Gram |

Duplicate	Dissimilar
1225.0	7590.0