

## Table of Contents

|   |    |
|---|----|
| 1 – Introduction .....  | 5  |
| 2 – Literature review .....   | 6  |
| 2.1 - Overview of String similarity metrics .....                       | 6  |
| 2.1.1 - Character Based similarity metrics .....                        | 6  |
| 2.1.2 - Token Based similarity metrics .....                            | 9  |
| 2.1.3 - Phonetic Similarity metrics .....                               | 9  |
| 2.2 - Overview of Frameworks for approximate matching .....             | 10 |
| 2.2.1 Character Based .....   | 10 |
| 2.2.2 Token Based .....   | 10 |
| 2.2.3 Combination of different similarity measures .....                | 12 |
| 2.2.4 Other techniques .....  | 12 |
| 2.3 - Machine Learning in Approximate String matching .....             | 13 |
| 2.3.1 Overview of Machine Learning in Approximate String matching ..... | 13 |
| 2.3.2 Challenges facing machine learning techniques.....                | 16 |
| 2.3.3 Active Learning Techniques .....                                  | 16 |
| 2.4 –Project Overview .....   | 18 |
| 2.4.1 Exorbyte’s MatchMaker and MDS .....                               | 18 |
| 2.4.2 Problem Overview .....  | 19 |
| 2.4.3 Project Aims .....  | 20 |
| 3- Design .....   | 22 |
| 3.1 – Proposed Solution .....   | 22 |
| 3.1.1 Choice of Classes .....   | 22 |
| 3.1.2 Choice of Feartures .....   | 22 |
| 3.2 – Training Data Collection .....                                    | 27 |
| 3.2.1 Criteria for selecting training data .....                        | 27 |

|   |    |
|---|----|
| 3.2.2 Training data format .....                  | 27 |
| 3.3 – Selection of learning algorithm .....       | 28 |
| 4 – Implementation .....                          | 29 |
| 4.1 – Main Steps .....                            | 29 |
| 4.2 - Language .....                              | 30 |
| 4.3 – Tools developed .....                       | 30 |
| 4.3.1 Storing data .....                          | 30 |
| 4.3.2 Connecting to MatchMaker and MDS APIs ..... | 33 |
| 4.3.3 Tool Implemented .....                      | 34 |
| 4.4 – WEKA .....                                  | 37 |
| 4.5 – Testing .....                               | 37 |
| 5 – Results and Analysis .....                    | 38 |
| 5.1 Initial Round Of Testing .....                | 38 |
| 5.2 Second Round Of Testing .....                 | 40 |
| 5.2.1 T-Tests Results .....                       | 42 |
| 5.3 Further Testing .....                         | 45 |
| 5.4 Types of Errors .....                         | 47 |
| 6 – Conclusion .....                              | 48 |
| 7 – Future work .....                             | 49 |
| 8 – References .....                              | 50 |

# 1 Introduction

Looking up an address in a database is very common, and finding a match for the query is easy if the input is exactly the same as the entry in the database, but when the input query is different (due to spelling mistakes or missing fields), then the search needs to be error-tolerant, i.e. the search would find matches that are close or similar to the input query string. In order to achieve that we need a way to compare strings that allow errors; this problem is known as approximate string matching [1] or flexible (fuzzy) string matching [2].

This can be used for searching a database for addresses or for updating a reference customer database from another database, for example assume a corporation has multiple customer databases managed independently, a customer may subscribe in more than one service and the data in both databases could differ because of typographical error or using abbreviations (e.g. name in one database could be “John Smith” and in another “J. Smith” the same for the address), so trying to merge these databases without using approximate search would fail. Or even if one is not trying to merge the two databases just update the phone numbers in one database by the numbers from the other would also need the approximate searching techniques. Another very similar problem that also needs approximate string searching is database duplicate elimination or known as database cleaning, where the same real-world address is represented by multiple rows in the database but the two entries are slightly different.

Solving the approximate string searching problem involves two steps

1. First defining a similarity metric that allows the search to be approximate, most of the similarity metrics use a cost function or a score to determine how close the strings are.
2. Define a framework that uses the similarity metric to find the matches.

In the second step we have to create some criteria to decide when an entry is close enough to be considered a match or not close enough to be considered a mismatch, this is mostly done by defining a threshold on the score such that an entry scoring above that threshold is considered a match and lower than the threshold then it is considered a mismatch.

This project is done in collaboration with Exorbyte whose product MatchMaker performs the approximate search between a query string and the rows of the database using Levenshtein (see section 2.1.1) algorithm and produces a score between 0 and 255, with 0 meaning nothing in common between the two strings and 255 means a perfect match. And to decide if there query is a match for a specific row in the database a threshold is should be defined on the resulting score; where any score higher than the threshold is considered a match.

This project deals with the problem of determining the thresholds that decide when a database entry is a match and when it is not, by applying machine learning algorithms on data samples collected from Exorbyte's MDS(Master Data Server) System.

In the next section an overview of the different metrics for string comparison are presented starting with Leveneshtein algorithm as it is the technique currently used by MatchMaker, and later other techniques used to do approximate string searching in databases are discussed.

How MatchMaker combines different techniques to do the approximate string matching efficiently, an overview of the project and MDS will also be discussed in the end of the next section (2.4).

## 2 Literature Review

In this section a review of the literature on approximate string matching and different machine learning approaches for this problem are presented.

### 2.1 String similarity metrics

To find matches to a string we need to measure how similar two strings are depending on a distance function and for that distance function to be considered a similarity metric it has to satisfy the next properties for two string  $s$  and  $t$  [1, 25]:

- 1-  $d(s,t) \geq 0$
- 2-  $d(s,t) = 0$  iff  $s = t$
- 3-  $d(s,t) = d(t,s)$
- 4-  $d(s,t) + d(t,r) \geq d(s,r)$

Most of the similarity metrics can be separated into three groups character based techniques, vector (or token) based techniques and Phonetic similarity measures [3, 4, 5].

Since MatchMaker uses a combination of those groups of similarity metrics we review these groups starting with character based similarity metrics.

#### 2.1.1 Character Based similarity metrics

Character based measures are effective when there are typographical errors between the two strings being compared, and the most famous measure is the edit distance or Levenshtein distance.

We start with Leveneshtein distance because it is the metric used by MatchMaker when performing the approximate string matching.

**Levenshtein distance[6]:** the edit distance between two string t1 and t2 is defined as the minimum number of edit operations needed to transform t1 into t2, and these operations can be insertion, deletion or replacement, assigning a cost = 1 to each operation. And it can be calculated using dynamic programming as follows:

- A table is constructed to save the values or the cost of the match.
- Starting at the top left corner cell setting the cost to zero. And the first row and column from 0 to the number of letters in each word.
- Then starting for each cell calculate the cost as

$$C(i, j) = \min \begin{cases} C(i-1, j-1) \\ C(i-1, j) \\ C(i, j-1) \end{cases} + \begin{cases} \text{Insertion} \\ \text{Deletion} \\ \text{Substitution} \end{cases} \text{ Cost of operation}$$

The cost of each operation depends on the corresponding letter in row i and column j so if the two letters are equal then the cost is 0 else the cost is set to 1.

An example of applying Levenshtein algorithm to calculate the distance between “Biology” and “Zoology” is shown in the figure below.

|   |   | Z | o | o | l | o | g | y |
|---|---|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| B | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| i | 2 | 2 | 2 | 3 | 4 | 5 | 6 | 7 |
| o | 3 | 3 | 2 | 2 | 3 | 4 | 5 | 6 |
| l | 4 | 4 | 3 | 3 | 2 | 3 | 4 | 5 |
| o | 5 | 5 | 4 | 3 | 3 | 2 | 3 | 4 |
| g | 6 | 6 | 5 | 4 | 4 | 3 | 2 | 3 |
| y | 7 | 7 | 6 | 5 | 5 | 4 | 3 | 2 |

**Figure 1 :** An example illustrating the steps to compute the Levenshtein distance, the cost of a cell with index i,j is computed as the minimum of the adjacent cells  $\min((i-1,j-1),(i-1,j),(i,j-1))$  + cost of the operation (insertion, deletion or replacement).

**Affine gap distance[7]:** is an extension of Levenshtein distance by Needleman and Wunsch that tries to address the problem of gaps between the two strings in the Levenshtein distance, gaps between strings can occur due to abbreviation e.g. when comparing the two strings “John Smith” and “J. Smith”, the cost of gaps when using Levenshtein is high because all the characters inserted to complete the gap has the same cost, so in this measure two extra

operations are introduced “open gap” and “extend gap” with a smaller cost for extending a gap causing the cost for a gap mismatch to be smaller than that of Levenshtein distance.

**Smith-Waterman Distance[22]:** is an extension of the affine gap distance assigning lower costs to gaps in the start and end of strings than gaps in the middle of the strings, for example when comparing “John Smith” and “Prof. John Smith”, since “Prof.” is at the start of the second string then the cost of this mismatch or gap is lowered and the two strings are closer using this distance metric than when using the original affine gap metric.

**Jaro distance[8]:** Jaro described an algorithm that was mainly used for name matching tasks, the idea is to compute the number of matching characters in the strings regardless of their position, so to compare two strings  $t_1$  and  $t_2$  each character from  $t_1$  is compared to all characters in  $t_2$  and two characters are considered a match if they are a match and the difference between their positions is less than a given limit. The Jaro distance is computed as

$$J(t_1, t_2) = 1/3 * \left( \frac{c}{|t_1|} + \frac{c}{|t_2|} + \frac{c - T/2}{c} \right)$$

where  $c$  is the number of common(matching) characters, and  $T$  is the number of transpositions (a transposition is an index in the common characters where the order of the characters don't match). e.g. when comparing “ABCDEZ” and “CBADEX” the number of matching characters  $c = 5$ , and in those matches there are 3 letters not in the right order so  $T = 3$  which gives  $J(\text{“ABCDEZ”, “CBADEX”}) = 0.7778$ .

**Q-gram distance[9]:** the idea behind the q-gram distance is that if the two strings are an approximate match or have small edit distance, then they should share similar parts of the string, and this is tested by sliding a window of length  $q$  over one string to get a number of substrings(grams) with length  $q$ , then the similarity to the other string is calculated as the number of matching grams in the other string.

E.g. when comparing  $t_1 = \text{“John Smith”}$  and  $t_2 = \text{“John A Smith”}$  the positional q-grams with  $q = 3$  for  $t_1 = \{(1, \#\#J), (2, \#Jo), (3, Joh), (4, ohn), (5, hn), (6, n S), (7, m), (8, Smi), (9, mit), (10, ith), (11, th%), (12, h\%\%)\}$  where “#” is the prefix of the string and “%” is the suffix. And for  $t_2 = \{(1, \#\#J), (2, \#Jo), (3, Joh), (4, ohn), (5, hn), (6, n A), (7, A), (8, A S), (9, Sm), (10, Smi), (11, mit), (12, ith), (13, th%), (14, h\%\%)\}$  and we can see that there are 11 equal grams between  $t_1$  and  $t_2$  if we ignore the position and 5 equal grams if we take the position into account.

The next section is an overview of the token based similarity metrics.

### 2.1.2 Token Based similarity metrics

The character based methods work well in case of typographical errors but when two strings have the same words but the order of these words are changed in one string (e.g.  $t_1 = \text{"John Smith"}$  and  $t_2 = \text{"Smith John"}$ ) in this case the methods discussed earlier will fail to capture that and the two strings will appear to be totally different, this problem is solved by the token based similarity methods, where a string is represented as a collection (bags) of words (tokens).

**Atomic strings[10]:** also known as the Monge Elkan distance, this method splits the string on a given punctuation character (e.g. the space character) into a number of atomic strings, and the score is computed as the number of matching atomic strings divided by the average number of atomic strings.

**Cosine similarity:** cosine similarity also known as vector space similarity is a measure usually used to find the similarity between two vectors by measuring the cosine of the angle between them, it may not be clear now how this can be used to compare strings but in section 3.2 we will show that when combined with the TF-IDF (term frequency–inverse document frequency) weighting scheme it can be used as a string similarity measure.

### 2.1.3 Phonetic Similarity metrics

The last two types of similarity measures focus on the string representations, but two strings may be phonetically similar even if their string representation is totally different. The phonetic similarity measures try to address this issue.

**Soundex[27]:** Robert C. Russell and Margaret K. Odell created this algorithm for comparing strings by sound, it was originally created to be used on names, and it works by assigning the same soundex code to phonetically similar groups of consonants. The name's soundex code consists of a letter (the first letter of the name) then three digits encoding the rest of the consonants.

**Metaphone[11]:** this algorithm was developed to address some of the shortcomings of the soundex algorithm, unlike the soundex algorithm this one produces a variable length code, and uses a larger set of rules for the English pronunciation, and a new version called Double Metaphone was developed with more complex rule set.

## 2.2 - Overview of Frameworks for approximate matching

In this section we present some of the different frameworks developed for approximate string matching, some of the frameworks adopt the character based similarity measures or the token based similarity measures, first we start by discussing the character based similarity measures.

### 2.2.1 Character Based

The use of edit-distance based similarity metrics for approximate record matching tasks was first proposed by Monge & Elkan [23] it was used to find duplicate entries in databases; in their proposed technique they treat each database record as one long string, and then compare the strings using the Smith-Waterman distance metric. They depend on the transitive property of duplicate records (i.e. if record R1 is a duplicate of R2 and R2 is a duplicate of R3 then R1 is a duplicate of R3) to sort the database records into sets or clusters of duplicates. A new record is added to the cluster when the Smith-Waterman score is higher than a given threshold.

Another framework was proposed by Gravano et al. [14] that uses edit distance with affine gaps to find approximate matches, this approach involved two steps:

1. First they use q-grams to get an initial set of candidate matches.
2. Then they used edit distance with affine gaps to calculate the distance between each pair of the candidates to eliminate false positives.

They formulate the second step as “Given tables R1 and R2 with string attributes R1.Ai and R2.Aj, and an integer k, retrieve all pairs of records  $(t, t') \in R1 \times R2$  such that  $\text{edit distance}(R1.Ai(t), R2.Aj(t')) \leq k$ ”.

The approaches discussed so far have the same problem of needing a user defined threshold k which is the same problem we are trying to solve in this project.

### 2.2.2 Token Based

Cohen [15] presented an approach that combines the cosine similarity with the TF-IDF weighting scheme, the idea behind this is that a token(a word in the string) is more important if its frequency in the database is not high and vice versa when the token has high frequency in the database then its effect shouldn't be very important. Cosine similarity uses dot product between two vectors and to achieve that for strings a weighting based on TF-IDF is assigned for each token or word in the strings we are comparing.

This approach treats each string as a vector of words; each word (w) is assigned a weight (v) depending on the frequency of this word in the database as a whole (idf) and on the frequency in the string (tf) we are comparing.



The weight  $v$  is calculated as:

$$v(w) = \log(tf_w + 1) \cdot \log(idf_w)$$

So according to the TF-IDF the weight of the frequent words in the database is low (e.g. the word “Street” in addresses) and this is good because when comparing we need the cost of an error in such words to be low. Then after computing the weights for each word in the string cosine similarity to compare between the two vectors (strings) is calculated according to the equation:

$$sim(t_1, t_2) = \frac{\sum_{j=1} v_{t1}(j) \cdot v_{t2}(j)}{||v_{t1}|| \cdot ||v_{t2}||}$$

And using this similarity measure causes the search to be insensitive to the location of the words in the strings (e.g. “John Smith” is the same as “Smith John”) and introducing words with high frequency like “Mr.” does not affect the outcome very much (e.g. “John Smith” is similar to “Mr. John Smith”).

This approach solves the problem of word order or location in the query and the database entry, but has the problem of not approximately matching each token or word i.e. the words appearing in the query and the database record must be the same but not in the same order. This can be solved by combining this framework with a Leveneshtein-Based distance metric which we will discuss in the next section.

### 2.2.3 Combination of different similarity measures

Some frameworks were developed combining different similarity measures to form a new framework with enhanced performance over other techniques using a single similarity measure. Chaudhuri et al. [16] developed a framework to find fuzzy matches in databases that uses Fuzzy Similarity Function (fms) which is a combination of edit-distance, atomic strings (or tokens) and IDF (inverse document frequency) weighting function, they explain that the similarity between input query tuple and a database tuple is the cost of transforming the former into the latter and so they introduce three transformation operations: token replacement, token insertion and token deletion, and they compute the cost for each operation based on the edit-distance and the IDF of the token.

Fms assigns different cost to different operations, assume we have input tuple  $S$  and reference tuple  $T$  the costs are defined as:

- Token replacement: is the cost of transforming a token from tuple S to a token from tuple T and is equal to  $\text{edit-distance}(s_i, t_i) * w(s_i)$  where  $w(s_i)$  is the IDF of the token  $s_i$ ; IDF is calculated as  $w(s_i) = \log \frac{|R|}{\text{freq}(s_i)}$  where  $|R|$  is the number of tuples in the database R.
- Token insertion: The cost of inserting a token  $t$  into S is  $\text{cins} * w(t, i)$ , where the token insertion factor  $\text{cins}$  is a constant between 0 and 1.
- Token deletion: The cost of deleting a token  $t$  from S is  $w(t)$ .

And the cost of transforming tuple S into T is the summation of transformation costs for the sequence of token operations is calculated as:

$$tc(S, T) = \sum_i tc(s_i, t_i)$$

Then this cost is used to find the final  $fms$  score as:

$$fms(S, T) = 1 - \min\left(\frac{tc(S, T)}{w(S)}, 1\right)$$

Where  $w(S)$  is the sum of all token weights in the input query.

Then to find K-approximate matches the framework takes an input query and a threshold  $c$ , then finds the best  $k$  matching tuples with  $fms(S, T) \geq c$ , but this approach also has the problem of how to define that threshold.

In the next section we present some other techniques used for approximate string matching just for completeness.

## 2.2.4 Other techniques

Several techniques depending on suffix trees are used in finding approximate matches, suffix trees are a data structure useful when the pattern or the string to be searched in is available for preprocessing. Any position  $i$  in a string  $T$  automatically defines a suffix of  $T$ . A suffix tree is a trie data structure built over all the suffixes of  $T$ . At the leaf nodes the pointers to the suffixes are stored. Each leaf represents a suffix and each internal node represents a unique substring of  $S$ . Every substring of  $T$  can be found by traversing a path from the root. In [19,20] there are two frameworks introduced for finding approximate matches using suffix trees.

Bozkaya and Ozsoyoglu [25] introduced a distance based index structure called multi-vantage point (mvp) tree for similarity queries (i.e. finding approximate results for the query) on high-dimensional metric spaces, mvp tree is a data structure that partitions the data space into spherical cuts around vantage points, mvp-tree uses two vantage points in each node.

## 2.3 Machine Learning in Approximate String matching

### 2.3.1 Overview of Machine Learning in Approximate String matching

Most of the techniques discussed so far depend on generic approaches that are designed to work for any type of approximate string database searches with the parameters for the similarity measure being the same for all database e.g. unit cost for the Levenshtein edit-distance for insertion, deletion and substitution; in this section we describe how machine learning techniques can be used to improve the performance of the search by training for each different database the similarity measures or training a classifier that decides when two strings are a match.

In [12] Ristad and Yianilos use machine learning to learn the cost of different edit-distance operations (insertion, deletion, substitution); the idea behind it is that the impact of a substitution differs according to the data at hand, e.g. when comparing addresses a number substitution in the street address is more costly than a substitution in the company's name because the latter is probably a typographical error. They model the string edit distance as a memoryless stochastic transduction between two strings with each step of the transduction generating either a substitution pair or a deletion pair or an insertion pair according to a probability function, and so the edit cost learning is now converted into learning the probabilities of the transitions and this is done using an expectation maximization (EM) algorithm which consists of two steps given  $n$  matched pairs training set as an input:

1. First because a string can be matched to the other string through a number of different sequences of insertion, substitutions and deletions, this first step accumulates the expected transition probabilities of the edit operations for all these sequence for all training pairs and this is the expectation step.
2. Then the maximization step normalizes the expected probabilities by dividing each expected probability by the sum of all expected probabilities and then updates the parameters of the model.

The next step is to train a nearest neighbour classifier is using a set of labeled training examples. Four experiments were conducted, and after training the error rates of the transduction distances are from one half to one sixth the error rate of the untrained Levenshtein distance.

The features used for the nearest neighbor classifier are the individual field scores and a total row score calculated by getting the weighted average of the fields scores.

In [13] Bilenko and Mooney also introduce two learnable distance metrics, the first one is edit distance with affine gaps, they view the distance between two strings as an alignment of the strings' characters and that alignment can contain gaps; so the edit operations are substitution or a gap (insertion or deletion), an alignment is generated from the stochastic transducer shown in the figure below as a sequence of traversals along the edges.

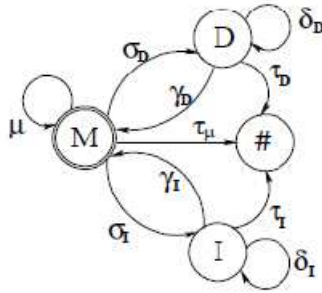


Figure 2 stochastic transducer

In the figure M is the starting state representing matching characters, I and D represent insertion and deletion of a characters (corresponding to a gap) and # is a terminal state.  $\sigma$ ,  $\mu$ ,  $\delta$  are transition probabilities between states. Given a set of approximately matched strings the model is trained by finding the expected transition probabilities using a similar EM algorithm as discussed in the previous technique.

The second distance metric presented is a learnable vector-space similarity, this metric tries to address a problem with the TF-IDF weighting scheme (see section 3.2), TF-IDF assigns weights depending on the word frequency in the database but the contribution of each word to the similarity differs from one domain to another e.g. If there are several records of addresses from the same street, the street name token (for example, '34th') will have a lower weight due to its many occurrences across the database resulting in a lower IDF value. At the same time, some addresses may contain the token 'Square', which then will be assigned approximately the same weight as '34th' since it may be as common.

While two addresses on the same street are more similar than two addresses that are both on squares [13], and this problem can be solved by the presented vector-space similarity by training the similarity function to give more weight to tokens that are actually good indicators of actual similarity, and to accomplish this the training data must contain two parts; matching string pairs and non matching string pairs (positive and negative training sets). Support vector machine is chosen for this learning task because it is able to learn well from limited training sets of high-dimensional data, and because the hypothesis learned by the classifier must be

independent of the relative sizes of the positive and negative training sets, since the proportion of duplicate pairs in the training set is likely to be much higher than in the actual database where duplicates are detected.

The whole framework developed for duplicate detection consisted of two parts the first part is dividing the database records into clusters of potential duplicates, then using a binary classifier with the features being the distance between the records in the cluster to determine which of the records are duplicates, but then they face the problem of defining a similarity threshold to separate duplicates from non-duplicates.

Experiments to detect duplicates were done on six different datasets (to show the increase in the efficiency due to change in domain) and they used precision (number of correctly identified duplicate pairs divided by the total number of identified duplicate pairs), recall (number of correctly identified duplicate pairs divided by the number of true duplicate pairs) and F-measure which is the mean of precision and recall) to compare the performance of the presented similarity measures with the original edit-distance with affine gaps and TF-IDF with cosine similarity, and the results show an increase in the F-measure when using the proposed learnable distance metrics.

The problem with the last two approaches is that they depend on training classifiers to learn the cost of different operations (insertion, deletion, substitution), the error rate may be better than using the same predefined cost because the cost will change from one domain to the other but using different costs means using a look-up table to get the learnt costs for each operation which is not practical in a search application because it will slow the comparison considerably as the size of the database grows. And so in this project we chose to keep the same cost for the different operations but add new features on the data that can produce better results and use different classifiers and compare their performance, the new features will be discussed in section (3.3).

### 2.3.2 Challenges facing machine learning techniques

The success of a method based on learning like the methods discussed in the last section depends on being able to provide a covering and challenging set of training examples that will demonstrate the different types of errors that can occur between the query string and the database records, e.g. the training set to search in addresses database should contain different types of error as misspelling in each different field (Street Name, House Number .. etc) and should also include errors resulting from abbreviations like Corporation to Corp., missing fields in the query or changes in the order of fields. Another problem when using approximate search in duplicate record detection problem is that finding such examples is hard because this requires a user to manually do a search in large records lists where the confusing record pairs might be spread far apart.

### 2.3.3 Active Learning Techniques

As discussed in the previous section there are some difficulties in collecting a training dataset that includes the most informative samples and some frameworks where built depending on active learning to try and address this problem, active learning assumes that the training examples are not all given at the start but some are collected by the learner during the learning process.

Tejada et al. [24] introduced an object identification system called Active Atlas that uses active learning to learn mapping rules for integrating information from multiple websites, the idea is that some attributes are more important in deciding whether there is a mapping between two objects or not and these attributes differ between domains.

Atlas consists of two stages the input to the first stage is different source objects (example two different web pages).

1. The first stage is called candidate generator, in this stage all shared attributes of the given objects are compared computing similarity scores for each attribute. The similarity score used is a combination of soundex, atomic strings and TF-IDF and a set of candidate mappings are proposed based on that score.
2. The second stage is learning the mapping rules. This component determines which attribute or combinations of attributes (e.g. Name, Street, Phone) are most important for mapping objects by learning the thresholds on the attribute similarity scores computed in the first stage. So from the candidate mappings collected in the first stage the learner then tries to find the most informative pairs to use as examples in learning the mapping rules, once these pairs are found the user is then

asked to classify them as mapped or unmapped. Then the learner constructs high accuracy mapping rules based on these examples, while at the same time limiting the amount of user involvement. Once the rules have been learned, they are applied to the set of candidate mappings to determine the set of mapped objects.

In the second stage the user is presented with a number of candidate mappings to classify as mapped or unmapped and this is to solve the problem of having to find the set of most informative examples before starting the learning process, in this system the learner finds the most informative candidates for the user to simply label them.

Another framework using active learning was presented by Sarawagi and Bhamidipaty [26], the framework is designed for eliminating duplicate entries in databases.

In this framework a set of similarity metrics are used to compare the two database records edit-distance, soundex on text fields and absolute integer difference for numerical fields. The main steps of the framework are:

1. First a small list (less than ten) of training pairs of records as examples is supplied to the system as an input (seed), the list should contain positive and negative examples( duplicates and non-duplicates).
2. Then using the similarity metrics the seed are labeled “1” for a duplicate and “0” for non-duplicates.
3. The classifier is then trained using this list, in their paper they used three types of classifiers decision trees, naïve Bayes and support vector machines.
4. The classifier then is used to select a subset S of the database records that would introduce the largest information gain when labeled correctly.
5. The user is presented with the subset S and a suggested labeling for each of the labels; the user then confirms the labeling or corrects it.
6. The new subset S of correctly labeled examples are added to the initial list and the classifier is trained on the new set.
7. The new classifier’s performance is evaluated on the rest of the database records, if the user is not satisfied with the performance then the system continues again from step 4.

The output of this system is a classifier that, given a database or a list of records can then find the duplicates.

In both of the last two techniques discussed, there is step where the active learner selects a subset of examples from the database or the examples that introduce the largest information gain to get the user’s help to classify, how are these examples selected ?

The initial classifier will be sure about its predictions on some unlabeled instances but unsure on most others; the number of instances where the classifier is unsure about its class will increase when the number of training examples used to train the classifier is small. These unsure instances are those that fall in the classifier's "confusion region". So if the classifier would get the correct classification for these instances, it will reduce the amount of confusion and this is done by presenting these instances to the user to correctly classify or label. This intuition forms the basis for one major criteria of active learning, namely, selecting instances about which the classifiers built on the current training set is most uncertain. And in both papers the way to quantify how uncertain the classifier is about an example is by creating a committee of classifiers that are different from one another, but a known match will get the same classification from all of them and a known non-match would also get the same classification, while the uncertain ones will get different labels from each classifier in the committee, and to quantify this uncertainty the entropy on the fraction of committee members is calculated. This is also known as query by bagging.

## 2.4 Project Overview

In this section an overview of the project and Exorbyte's MatchMaker and MDS are discussed, starting with how MatchMaker combines different similarity metrics to perform approximate string matching, then how the current MDS implementation performs the decision making process and where this project comes in as a part of the system.

### 2.4.1 Exorbyte's MatchMaker and MDS

**MatchMaker:** uses a combination of atomic strings and character-based distance metrics (discussed in the previous sections) as a measure of similarity between strings, with an optional use of a phonetic similarity measure; which when used would compromise 20% of the overall score function.

Given a string query the words or tokens of each string are compared to the tokens of the other string, each string comparison is done using Levenshtein distance as follows:

Consider comparing two strings  $t_1$  and  $t_2$  the Levenshtein edit-distance is calculated as  $C(t_1, t_2) = C_{12}$  and the smaller  $C_{12}$  the closer  $t_1$  is to  $t_2$ . Then a score  $S_{12}$  depending on the Levenshtein distance is computed and that score is an integer between 0 and 255; this is done to speed up the computations and to store the score in one byte. And the bigger the score the closer  $t_1$  is to  $t_2$  (i.e. 255 is a perfect match and 0 is all the characters in the string are different). The score function  $S$  is computed; given the Levenshtein cost  $C_{12} = C(t_1, t_2)$ , by dividing the cost of the match ( $C_{12}$ ) by the length of the longest string, which gives a real number between  $[0,1]$  and the final score  $S_{12}$  is computed as:



$$S_{12} = \left[ 1 - \left( \frac{C_{12}}{\max(|s_1|, |s_2|)} \right) \right] * 255$$

**MDS:** takes in a query normalizes it, i.e. extracts the values corresponding to each database fields from that query, then runs MatchMaker getting the scores for the database matches and based on these scores a decision is made whether there is an approximate match in the database for the query or not.

The process of making that decision is done using a “Decision Matrix” which is a rule classifier defining thresholds on the row score and other features and returns a decision; the values for these thresholds are hand written according to the experts’ opinions.

## 2.4.2 Problem Overview

Now that we discussed both MatchMaker and MDS we present an overall view of the process and the part of the process which this project affects.

This project is done on one of the applications of MDS “Address Matching” where the database is the German Telecom database, having the fields

1. Company Name.
2. Thoroughfare.
3. Premise.
4. Postcode.
5. Region.
6. Work phone.
7. fax.

And the whole process from getting the input till reaching a final decision is as follows:

1. The queries are submitted to MDS on the form “Address Line 1”, “Address Line 2”, “Phone Number” and “Fax”.
2. MDS performs normalization, extracting the values corresponding to the fields (Company Name, Thoroughfare, Premise, Postcode, Region, Work phone, fax) from the input query.

3. MatchMaker is then used to compare the normalized query to all database rows and returns the matching scores for those rows. In addition to the scores MatchMaker also returns some statistics about the matching process that are used to define new features; these features will be presented in more details in section 3.2.2.
4. Based on the scores returned MDS makes decision whether there is an approximate match in the database for the query or not using the “Decision Matrix” discussed in the previous section.

This process is shown below in figure 3.

This project focuses on the last step of the project (deciding whether there is an approximate match for the query or not) instead of using the “Decision Matrix” we are aiming at using machine learning to train a classifier to learn the thresholds that decide the class of the matches for the query. This is done by collecting query and results data and calculating some features on the matching scores a discussion on the training data is presented in section 3.3. Different classifiers are trained and a discussion of the classifiers will be presented in section 3.4.

### **2.4.3 Project Aims**

So in this project there are several aims:

- 1- Use machine learning instead of the current “Decision Matrix” for classifying the matches.
- 2- Compare the performance of using machine learning techniques against the “Decision Matrix”.
- 3- Compare the use of traditional features used in machine learning techniques applied to this problem and similar problems (e.g. eliminating fuzzy duplicates) and the performance of the same techniques after including the new features ( new features will be discussed in the next section).

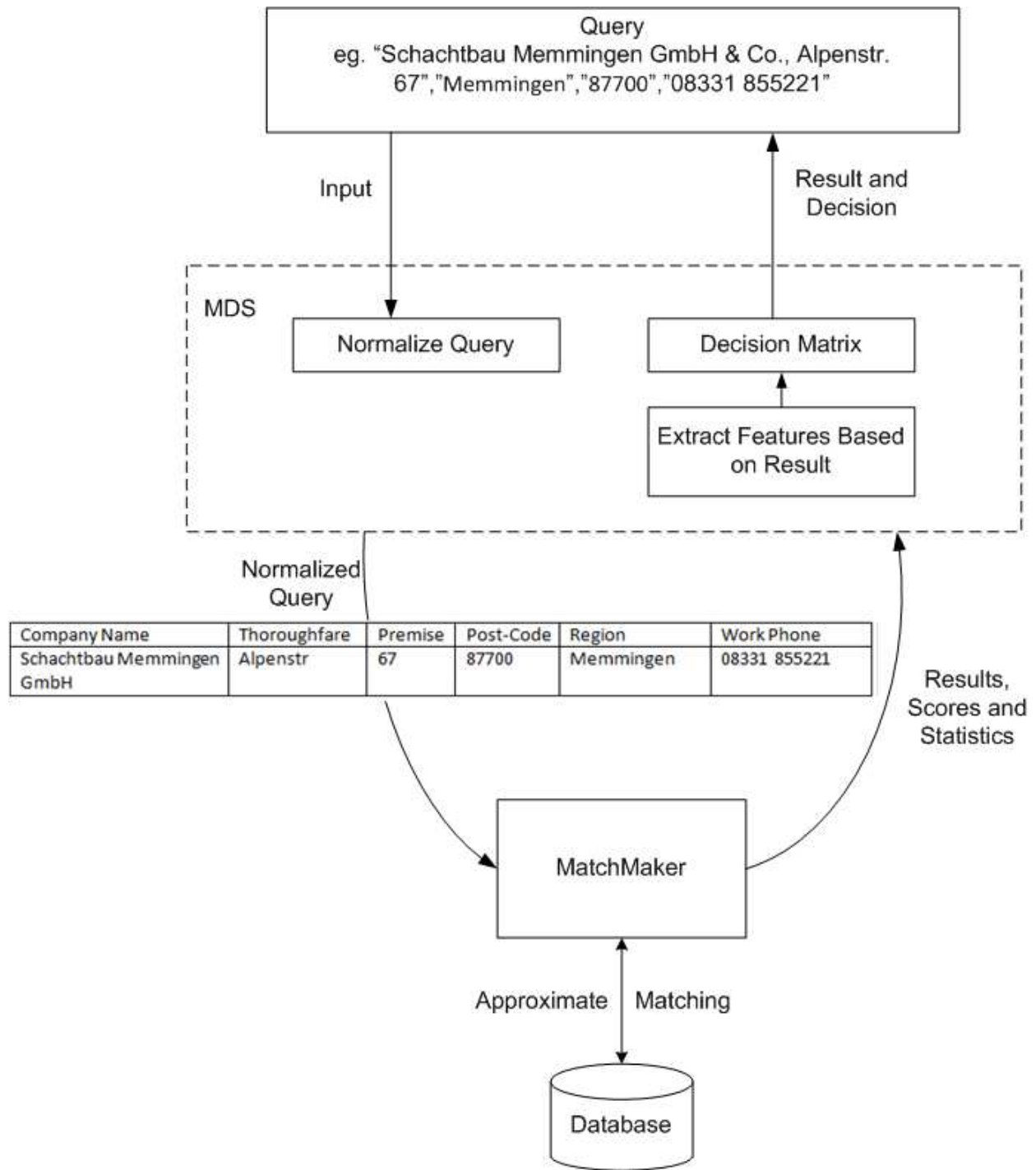


Figure 3

In this figure the flow of the query from input to the output and getting the decision. First the input query is normalized by MDS and the values for each field is extracted, then the normalized query is sent to MatchMaker where approximate matching is performed against all the rows in the database producing the Leveneshtein-based score, then returning the result and these scores to MDS which then extracts some features from the scores and the statistics, then using the "Decision matrix" the final class of the result is decided.

## 3 Design

In this section the proposed solution will be presented, we will start by discussing the problem based on the current system implementation discussed in section 2.4 and then discussing the machine learning solution proposed followed by an overview of the data collection process and the choice of learning algorithms.

### 3.1 Proposed Solution

In this section we will discuss the solution for learning the thresholds, first by discussing the choice of classes and then the features extracted from the MatchMaker scores and statistics to be used to train different classifiers. A discussion of the results and comparison of classifiers will be presented in section 5.

#### 3.1.1 Choice of Classes

In most of the literature [4] two classes are used to identify the relation of the query to the results (Match, No-Match) which results in misclassifying those instances that are unclear, or those instances that could be classified as either a match or no-match because of them being on the boundary between a match and a no-match. Because the cost of an error resulting from classifying a match a no-match is less than classifying a no-match as a match, we could bias the classifying process to classify as a no-match in case of unclear instances.

But in this project a new class “Suggest” is added to contain those instances where we are not sure of it being a match or not, so incase of being used in a search application we could output the instances contained in that class as suggestions, or if used in a de-duplication (i.e. removing database fuzzy duplicates) framework the instances in the suggest class would need a human confirmation to manually identify them being a match or not.

#### 3.1.2 Choice of Features

Based on MatchMaker results and statistics on the query matching process a number of features are extracted, which are then used to train the classifiers, some of the features are used in all the research done in this area (in other words traditional features) and there are two new features “**Difference To Second**” and “**Potential For Improvement**” introduced in this project that are extracted from the MatchMaker statistics on the matching process.

The features are:

- 1- **Individual Fields Scores:** for each field in the query and the corresponding database field a Leveneshtein-based score is computed which is then used as features, MatchMaker allows missing fields in the query and in this case the score is set to 0.
- 2- **Total Row Score:** a total row score is calculated by getting a weighted average of the individual field scores, the individual row scores are weighted because some fields are more important to the match process than other fields. For example the score of matching “Company Name” is more important than the matching score of “Work Phone”, in other words we tolerate errors in “Work Phone” field more than errors in “Company Name” so a match score of 200 out of 250 in work phone field wouldn’t affect the total row score as much as a score of 200 out of 250 in the company name field.

The weights assigned to each field are as follows:

|         | Company | Thoroughfare | Premise | PostCode | PostLocality | Region | WorkPhone | Fax |
|---------|---------|--------------|---------|----------|--------------|--------|-----------|-----|
| Weights | 250     | 100          | 30      | 80       | 100          | 50     | 100       | 100 |

- 3- **Difference To Second:** the results from MatchMaker are ordered by the total row score, this feature is extracted by getting the difference between the total row score of the best match and the next best match, the idea behind this features is that the larger the difference between the best match and the next best match the more confident we are of this row being a match, because if the difference is large then we are sure that there is no other rows with individual fields score that can be combined to form a good score.

But if the difference is small for example if the difference to second = 5 this means that the next best matches is only 5 points away and in this case the next best total row score can in fact be a better match for the query. An example showing the importance of this feature is shown in figures 4, 5.

This feature can be further scaled, as the size of the database grows the difference to second can be less useful because the probability of getting two matches for the same query with a small score difference increases, so in this case we can add difference to third , fourth ...etc. which will help cope with the increasing size of the database.

For this query:

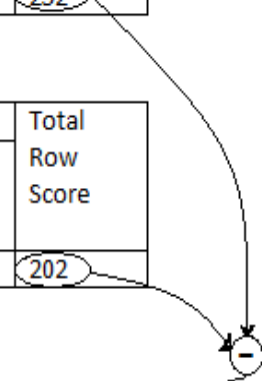
| Company Name                                | Thoroughfare     | Premise | Post-Code | Post Locality | Region | Work Phone    |
|---|------------------|---------|-----------|---------------|--------|---------------|
| Bundesministerium für Bildung und Forschung | Hannoversche Str | 28-30   | 10115     | Berlin        | Berlin | 030 285405338 |

The row with the best match is:

| Company Name                                | Thoroughfare     | Premise | Post-Code | Post Locality | Region | Work Phone | Total Row Score |
|---|------------------|---------|-----------|---------------|--------|------------|-----------------|
| Bundesministerium für Bildung und Forschung | Hannoversche Str | 30      | 10115     | Berlin        | Berlin | 030 18     |                 |
| 250   | 255              | 235     | 255       | 255           | 255    | 204        | 252             |

The second best match is:

| Company Name                           | Thoroughfare            | Premise | Post-Code | Post Locality | Region | Work Phone | Total Row Score |
|--|-------------------------|---------|-----------|---------------|--------|------------|-----------------|
| Bundesamt für Bauwesen und Raumordnung | Platz vor dem Neuen Tor | 2       | 10115     | Berlin        | Berlin | 030 283944 |                 |
| 146                                    | 105                     | 255     | 255       | 255           | 255    | 200        | 202             |



$$\text{Difference To Second} = 252 - 202 = 50$$

**Figure 4:** High Values of Difference to second, in this case the best match is clearly a very good match for the query with a total row score = 252 , and the second best match is not with a score = 202 and so the Difference To Second = 50.

For this query:

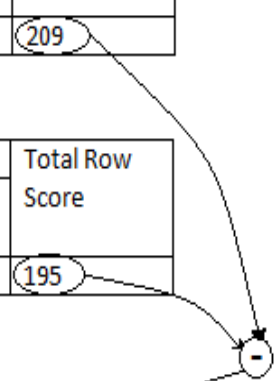
| Company Name | Thoroughfare | Premise | Post-Code | Post Locality | Region    | Work Phone |
|--------------|--------------|---------|-----------|---------------|-----------|------------|
| Übersetzer   | Merzenstr    | 46      | 70469     | Stuttgart     | Stuttgart | 0711 8554  |

The row with the best match is:

| Company Name              | Thoroughfare | Premise | Post-Code | Post Locality | Region    | Work Phone | Total Row Score |
|---------------------------|--------------|---------|-----------|---------------|-----------|------------|-----------------|
| Übersetzerin für Türkisch | Grazer Str   | 46      | 70469     | Stuttgart     | Stuttgart | 0170 67636 |                 |
| 166                       | 107          | 255     | 255       | 255           | 255       | 0          | 209             |

The second best match is:

| Company Name               | Thoroughfare | Premise | Post-Code | Post Locality | Region    | Work Phone | Total Row Score |
|----------------------------|--------------|---------|-----------|---------------|-----------|------------|-----------------|
| TransMission Übersetzungen | Hohnerstr    | 23      | 70469     | Stuttgart     | Stuttgart | 0711 93316 |                 |
| 192                        | 108          | 0       | 255       | 255           | 255       | 127        | 195             |



$$\text{Difference To Second} = 209 - 195 = 14$$

**Figure 5:** Low Values for Difference To Second, in this case the best difference between the best match and the second best match is low = 14, which causes us to think that this is not a very good match because there are other matches close to it.

From figures 4 and 5 we can see that for small values of Difference To Second the row with the best total score is not a match, and as it increases we get more certain that the row is a match.

- 4- **Potential For Improvement:** for each individual field the potential for improvement is extracted from MatchMaker statistics, and it represents the difference between the best score for this field over all the database and this field score for this row. Suppose there is a row with a better score than the first choice for a given field, e.g. Company Name, but has a score of 0 for all other fields which will cause it to have a lower total score, and will not be considered in the matching but still we know that there is another row in the database that has a better match for this specific field so now we can't be very confident that our best match is a match. So the lower the potential for improvement the more certain that the row is a match.

Since the matches are sorted based on the total row score, this feature is a way of balancing the importance of the total row score with the probability of a better match for each field even if the row with the better field score has a very low total score and can't be considered to be a match.

An example to illustrate this feature is shown in figure 6.

Suppose for this query

| Company Name       | Thoroughfare | Premise | Post-Code | Post Locality | Region      | Work Phone   |
|--------------------|--------------|---------|-----------|---------------|-------------|--------------|
| Hotel Allgäu Sonne | Am Stießberg | 1       | 87534     | Oberstaufen   | Oberstaufen | 08386 702920 |

The best match (row with the highest total score)

Total Row Score

| Company Name | Thoroughfare | Premise | Post-Code | Post Locality | Region      | Work Phone |
|--------------|--------------|---------|-----------|---------------|-------------|------------|
| Allgäu Sonne | Stießberg    | 1       | 87534     | Oberstaufen   | Oberstaufen | 08386 7029 |
| 237          | 196          | 255     | 255       | 255           | 255         | 255        |

236

The record with the highest field score for the field "Company Name":

| Company Name       | Thoroughfare | Premise | Post-Code | Post Locality | Region    | Work Phone   |
|--------------------|--------------|---------|-----------|---------------|-----------|--------------|
| Hotel Allgäu Sonne | Alpenstr     | 67      | 87700     | Memmingen     | Memmingen | 08331 855221 |
| 255                | 0            | 0       | 100       | 0             | 0         | 50           |

100

Potential For Improvement =  $255 - 237 = 18$

**Figure 6:** Potential for Improvement: Although the overall score of the second result is too low to be considered a good match but still for the field "Company Name" it has a better score than the best match, which makes us less confident in the decision that the first row is a match, so the higher the potential for improvement the lower our confidence that the row is a match.



## 3.2 Training Data Collection

In this section we will discuss the criteria on which the process of collecting training instances is based and the format of the training set.

### 3.2.1 Criteria for selecting training instances

When collecting the training data it is important to get a balanced dataset with a number of perfect matches and good matches to get the right values of the features, also to get the cases where a row with a good total score is considered a suggest instead of a match due to another row having a very close total score (Difference to Second feature) and also to get a number of no-match rows to learn the thresholds for the features for the no-match class.

It is also important to capture all types of mistakes that can occur in the query, e.g. typos.. etc. so in collecting the training set these are some types of errors considered:

1. Mistyping Mistakes or Typos: to get the full values of levenshtein-based matching the training set contains queries where typographical mistakes are made.
2. Missing Field Values: MatchMaker allows searching the database with missing values so this is one of the criteria considered to have examples with this type of error.
3. Different Field mistakes: mistakes in each field should be considered to show the relative importance of each field to the overall matching score of the row to the query.
4. Queries that cause a lot of matches to be very close suggesting that the matching row is not certain to be the match for query.

### 3.2.2 Training data format

Training data for this project is collected by getting a query and running it and getting the resulting scores for each of the previously discussed features and then manually set the ground-truth for this query by labeling each result as “Match”, “Suggest” or “No-Match” from the user’s perspective, a tool was built to perform this task; the details for this tool are discussed in section 4.

Collecting the dataset in this manner is very time consuming that’s why in the future work (see section 7) we discuss the possibility of using active learning procedures (see section 2.3) to automatically collect training set and minimizing the user’s role in the process.

The resulting data set is an array of scores corresponding to each feature and labeled as “Match”, “Suggest” or “No-Match”.

1000 training examples were collected containing:

- 300 examples belonging to the “Suggest” class.
- 300 examples belonging to the “No-Match” class.
- 400 examples belonging to the “Match” class.

### 3.3 Selection of learning algorithm

We will start with training a rule classifier as it is the closest to the current “Decision Matrix” used by MDS to find the class of the queries.

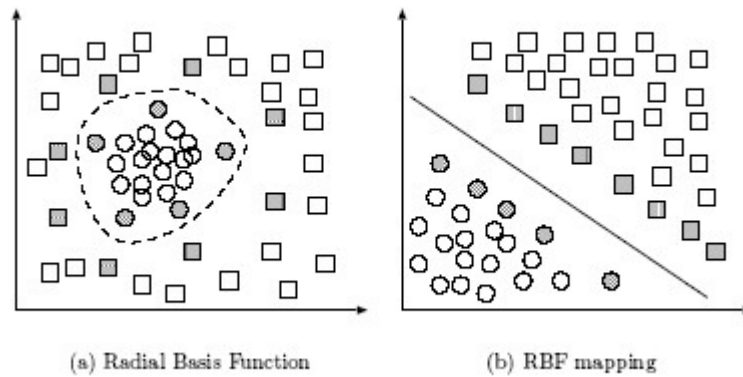
Then we will try using more complex classifiers:

1. Naïve Bayes.
2. Decision Tree.
3. Support Vector Machine.

For the SVM we will try to use different kernels and compare their performances. Each of the learning algorithms have their strengths and their weaknesses.

1. Rule classifiers divide the space for each class not considering the other classes.
2. Naïve Bayes generates relatively simple class separation boundaries (often linear), so it is not expected to perform better than the others.
3. Decision trees generate more complex separation boundaries than the Naïve Bayes classifiers so they are expected to have a better performance.
4. Support Vector Machine constructs a separating hyperplane or set of hyperplanes in the dimensional space, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class, since in general the larger the margin the lower the generalization error of the classifier. So SVMs are expected to have better performance in this problem.

Due to the nature of the problem and the results of the first round of testing we added the use of an SVM with a non-linear kernel in the second round of testing , the kernel that will be used is RBF which stands for “Radial Basis Function” with which the data is separated by radial boundaries and so allowing for more complex classification boundaries as shown by an example below in figure 7.



**Figure 7:** Radial Basis Function, showing more complex boundaries that can be formed by using the RBF, the image is from [29]

The results and comparison of the performances of those classifiers will be presented in detail in section 5.

## 4 Implementation

In this section we will discuss the implementation of tools created for collecting the training data and running the different classifier, we will start by the planning.

### 4.1 Main Steps

Because of the nature of the problem submitting a query and getting the output needs manual labeling the data this step is very time consuming so the main steps followed were:

1. Build a tool to read a number of queries (approximately 15000 test query), connect to MatchMaker APIs and collect the results.
2. Collecting training dataset: Use the tool to run the query by MatchMaker getting the results and manually labeling the query and the results returned as one of the three classes.
3. Use WEKA's java interface to train different classifiers.
4. Compare the performance using 10-fold cross validation.
5. Interpret the results and find a conclusion.

## 4.2 Language

Java was chosen because MatchMaker has java APIs and the same goes for WEKA, so this was the logical choice, and java is highly portable so it can be used on different operating systems as the primary servers for MatchMaker are Linux based systems.

## 4.3 Tools developed

Tools developed for reading queries, connecting to MatchMaker APIs running the queries and manually labeling the results and finally storing the training set and running WEKA.

First we start by discussing the design for storing data (queries and results).

### 4.3.1 Storing data

XML was chosen to save the queries and the results; as it has become the standard for storing data communicated between different applications and XML is easily readable, in the next figure 8 we give an example for a stored query.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Queries>
- <Query ID="549">
  <Auftrag>4037143</Auftrag>
  <Name2>Bundesministerium für</Name2>
  <Name3>Bildung und Forschung</Name3>
  <Name4 />
  <Ort>Berlin</Ort>
  <PLZ>10115</PLZ>
  <Straße>Hannoversche Str. 28-30</Straße>
  <Telefon>30</Telefon>
  <NbStTel>285405338</NbStTel>
  <Fax />
  <NbStFax />
</Query>
</Queries>
```

**Figure 8:** An example query stored as an xml file.

As shown in figure 8 the queries are saved with a unique ID for each query, and the different values which will then be normalized or the values corresponding to each of the database fields will be extracted from the queries.

The result after running a query and getting the result of MatchMaker and the values for all the features are stored as shown in figure 9.

The values stored are, more detailed in figure 9:

- The query value for each field with the matching value of in the database row, the match score and the potential for improvement.
- For each example the total row score.
- The difference to second value is stored.
- MDS decision.
- And finally the manually labeled decision.

Storing the values from objects to XML files and reading the XML files to objects are handled by XMLController a class created using StAX( **S**treaming **A**PI for **X**ML) which is an API for reading and writing XML files, it was chosen because it is faster than DOM and less memory and doesn't require surrendering the thread to SAX and using callback methods so it is a compromise between the two APIs.

```

▼<Examples>
  ▼<Example>
    ▼<Field>
      <Name>company</Name>
      <queryValue>Schmidt Phler</queryValue>
      <refValue>Schmidt</refValue>
      <score>205</score>
      <pot4Improv>37</pot4Improv>
    </Field>
    ▼<Field>
      <Name>thoroughfare</Name>
      <queryValue>Siegfriedstr.</queryValue>
      <refValue>Siegfriedstr</refValue>
      <score>255</score>
      <pot4Improv>0</pot4Improv>
    </Field>
    ▼<Field>
      <Name>premise</Name>
      <queryValue>33</queryValue>
      <refValue>33</refValue>
      <score>255</score>
      <pot4Improv>-255</pot4Improv>
    </Field>
    ▼<Field>
      <Name>postcode</Name>
      <queryValue>33615</queryValue>
      <refValue>33615</refValue>
      <score>255</score>
      <pot4Improv>0</pot4Improv>
    </Field>
    ►<Field>...</Field>
    ▼<Field>
      <Name>region</Name>
      <queryValue>Bielefeld</queryValue>
      <refValue>Bielefeld</refValue>
      <score>255</score>
      <pot4Improv>0</pot4Improv>
    </Field>
    ▼<Field>
      <Name>work_phone</Name>
      <queryValue>05211 368020</queryValue>
      <refValue/>
      <score>0</score>
      <pot4Improv>255</pot4Improv>
    </Field>
    ►<Field>...</Field>
    ►<Field>...</Field>
    ►<Field>...</Field>
    ►<Field>...</Field>
    <rowScore>250</rowScore>
    <diff2Sec>39</diff2Sec>
    <mdsDecision>5</mdsDecision>
    <newDecision>0</newDecision>
  </Example>
</Examples>

```

**Figure 9:** The result after running the query.

### 4.3.2 Connecting to MatchMaker and MDS APIs

#### MatchMaker

MatchMaker has java APIs that are used to run a query against the database. To run a query:

- Open a session with MatchMaker by creating a SessionWrapper object.
- Get the session object from the wrapper by calling SessionWrapper.getSession() which returns a MatchMakerSession object.
- Run the match process for the input query using MatchMakerSession.script().
- The results is returned in MatchMakerResult object, which contains the results and statistics on the data including the best match value for each individual field from which the feature potential for improvement is extracted.

#### MDS

To get MDS's decision for the query a connection to MDS's API is opened:

- Pass the query and the result to DecisionMatirx object.
- The decision is returned as a Rule object.

MDS's current configuration classifies each row as being in one of six classes displayed below in the decreasing order of how confident the system that it is a match:

1. "Exact" which is simply as it says an exact match with all the fields having perfect 255 match score.
2. "Match" a good match not perfect but is confident that the query and the database row are a match.
3. "Confirm" the row belonging to this class is probably a match but not very confident about the decision so it may need user intervention to confirm the row being a match.
4. "Select" rows belonging to this class are displayed to a user which then selects one of them to be the match.
5. "Suggest" rows belonging to this class probably are not a very good match but are displayed as a suggestion in case a user wants to revise them and see if they are a match.
6. "Sample" and "None" rows in these classes are considered a no-match.

In this project only three classes are used so a mapping from those classes to the three classes used is displayed in the table below.

| MDS's Class | Project Class | Comments   |
|-------------|---------------|--|
| Exact       | Match         | Those two classes represent the instances where we are sure they are a match to the query                      |
| Match       |               |  |
| Confirm     | Suggest       | Instances belonging to these three classes are considered unsure instances so they are mapped to Suggest class |
| Select      |               |  |
| Suggest     |               |  |
| Sample      | No-Match      | Instances in these two classes are considered a no-match   |
| None        |               |  |

**Table 1:** Mappings between MDS's classes and the classes used in this project

This MDS decision is stored in the output XML as shown in the previous figure 8 to be used for comparison and further analysis when comparing performances.

#### 4.3.3 Tool Implemented

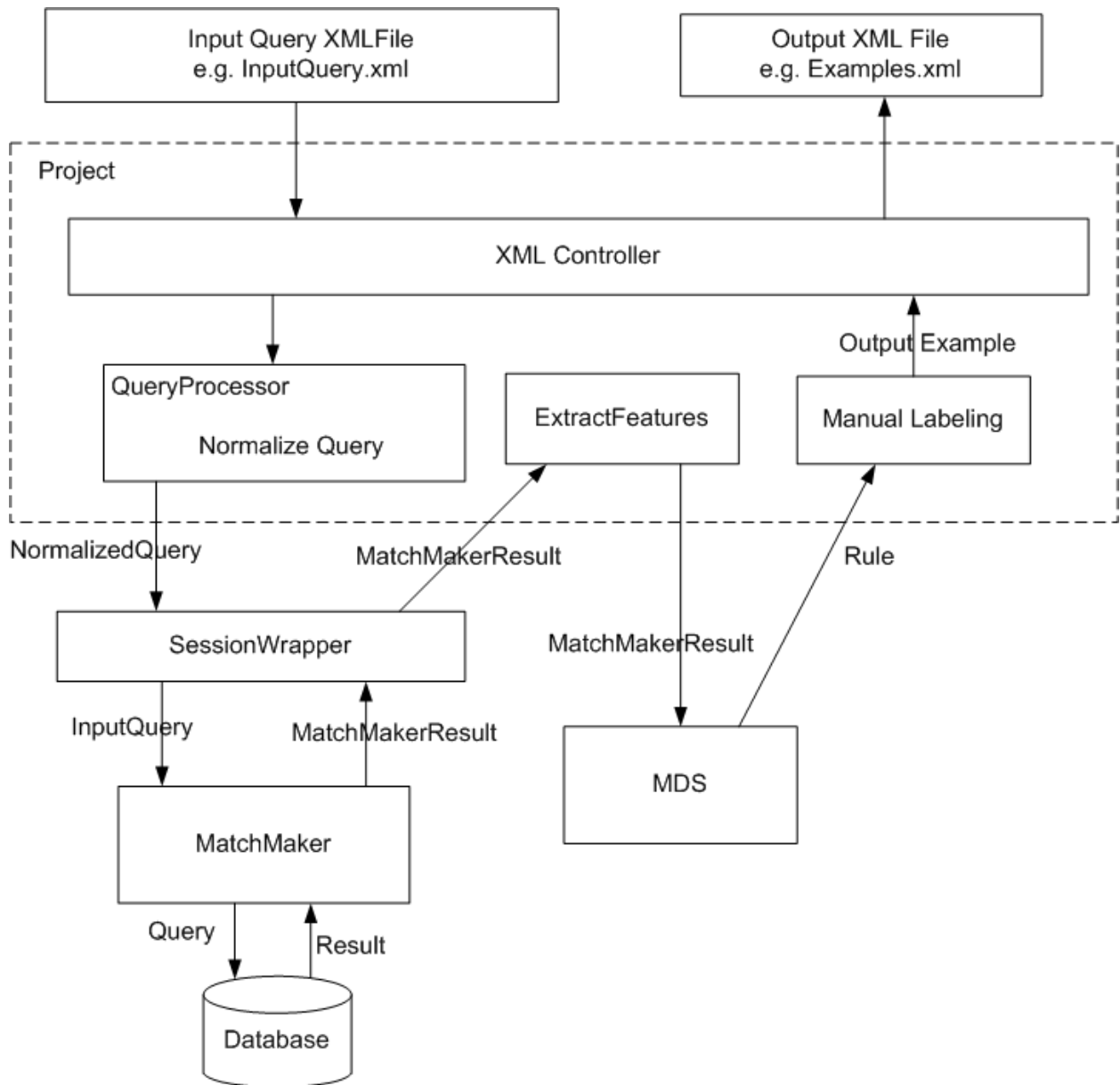
Now we talk about the whole flow of the project and the tools implemented

The flow of the implemented tool for collecting and labeling the examples is as follows:

- Get the input queries as XML file.
- Normalize the query.
- Connect to MatchMaker API.
- Submit the query.
- Get the result and extract the features.
- Get MDS's decision for the query.
- Manually label the output: this step is done because we can't depend on the decision made by MDS's decision matrix so the result should be manually labeled but MDS's decision is saved in the output for comparison and further comparisons between the new learning classifiers.

This process is displayed below in Figure 10.





**Figure 10:** The process starting from getting the input from the xml file till getting the final training set. Each box represents a process running independently within the project, and the arrows represent the flow of input from and output to each of the processes.

The process of labeling the data is done using a user interface which displays the query, result with each field match score, and the value for all the features, then the class is selected and the example can be added to the final training set or just skip it for the next query.

A snapshot of the manual labeling tool is shown below in figure 11.

Select Examples File:

Examples File:

D:\Exorbyte\data42\test\_data\FinalExamples\_1.xml Browse

Queries Loaded Successfully

Queries and Results:

Query: 9 of 500

| company                | thoroughfare      | premise | postcode | post_locality | region  | work_phone    | fax |
|------------------------|-------------------|---------|----------|---------------|---------|---------------|-----|
| SYSDAT GmbH Logisti... | Lise-Meitner-Str. | 5       | 50259    | Pulheim       | Pulheim | 02234 9855305 | 0   |

Result:

| company                 | thoroughfare               | premise | postcode | post_locality | region             | work_phone | fax |
|-------------------------|----------------------------|---------|----------|---------------|--------------------|------------|-----|
| Sysdat Computer-Syst... | Lise-Meitner-Str Meitne... | 5       | 50259    | Pulheim       | Pulheim Brauweiler | 02234 9855 |     |
| 163                     | 255                        | 255     | 255      | 255           | 255                | 255        | 0   |
| 45                      | 0                          | -255    | 0        | 0             | 0                  | 0          | 0   |

Number of Examples Collected

203

Total Row Score: 214

Difference to second: 4

Current Decision: No Match

Match Suggest No Match

77 63 63

Set Class

No Match ▼ Add To Training Set

<-- Jump 100 Jump 100 -->

Previous Query Next Query

Done ! Save Training Set

**Figure 11:** A snapshot of the manual labeling tool implemented.

The Query table contains the values of each field in the query row.

The Results table contains in the first row the value of each field of the match in the database, in the second row it contains the score match for each field, in the third row it contains the potential for improvement for each field.

The total row score and the value for Difference To Second feature is displayed under the result table.

The class of the query and the match is selected from the drop down list, and then if this is a good candidate to use in the training set, the example is added by clicking the “Add To Training Set” button.

Some labels are put to keep track of the number of examples collected so far, and some buttons to navigate queries.

## 4.4 WEKA

Weka [28] is an open source collection of machine learning algorithms, and since it was implemented in Java it is used in this project to train the classifiers and getting results.

As input Weka uses arff format which is a comma separated line of the features for each instance and the label for each instance, so the examples as shown in figure 8 were converted to the arff format. To input the instances the class “weka.core.Instances” is used.

And for classification and evaluation the classes “weka.classifiers.Classifier” and “weka.classifiers.Evaluation” where used, the evaluation class has the option of performing 10-fold cross validation which we will discuss in the next section.

## 4.5 Testing

First we will talk about the testing mechanisms that will be used in this project, and then the different types of testing applied on the data.

### 10-Fold cross validation:

This technique is used to estimate the performance of the classifiers. In the case of not having a two different training set and testing set, 10-fold cross validation is used to calculate the accuracy of the classifiers on unseen instances by dividing the set of examples into 10 equal sets and training the classifier on 9 parts or subsets and using the tenth subset as a testing set, and repeating the last steps 10 times choosing different 9 subsets each time for training, and getting the average performance on every testing set. So overall the classifier is tested on a number of examples equal to the set of examples we have.

### T-test:

T-test is a test that shows the significance of the results, by getting the probability of getting similar result by accident; this is called the p-value. And if the p-value is smaller than 0.05 then that means the results are statistically significant and unlikely to be due to chance. This test will be used for comparing the results of the classifiers and the current MDS implementation.

In this project different types of test will be performed:

- Testing the use of machine learning techniques against the current MDS implementation. And this is done by using 10-fold cross validation and then T-test. To decide if using machine learning techniques is better for classifying matches.
- Testing different learning algorithms to see which performs better in this training set.

- Testing the use of new features extracted from the data ( see section 3.1.2) against the use of traditional features in training the classifiers.

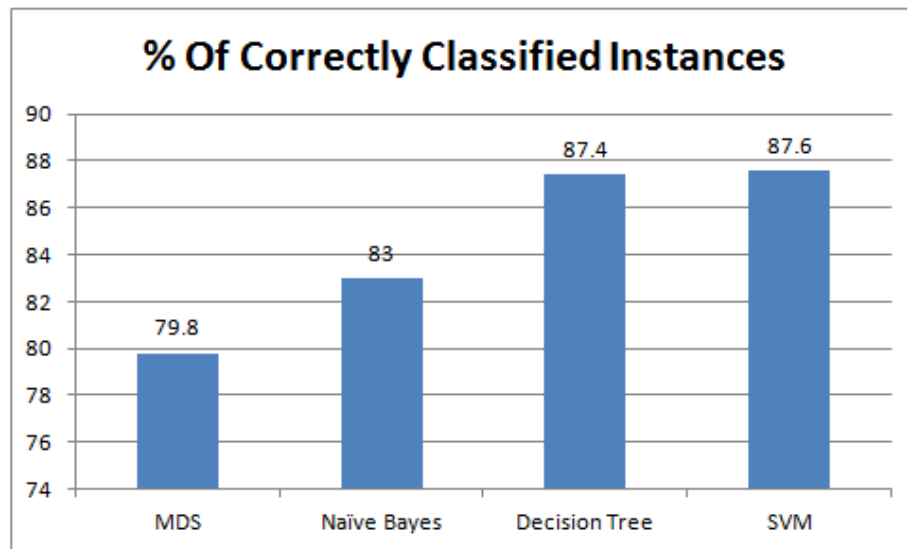
## 5 Results and Analysis

### 5.1 Initial Round Of Testing

For the first round of testing a training set of 500 examples was collected containing, 200 examples of the class “Match”, 150 examples of the class “Suggest” and 150 examples of the class “No-Match”.

Naïve Bayes, Decision Trees and SVM classifiers were trained on that set using the traditional features (excluding the features “Difference To Second” and “Potential For Improvement”), and the result of the testing is shown in figure 12.

| Technique                  | Percentage of Correctly Classified Instances |
|----------------------------|--|
| Current MDS implementation | 79.8 %                                       |
| Naïve Bayes                | 83 %   |
| Decision Tree              | 87.4 %                                       |
| SVM                        | 87.6 %                                       |



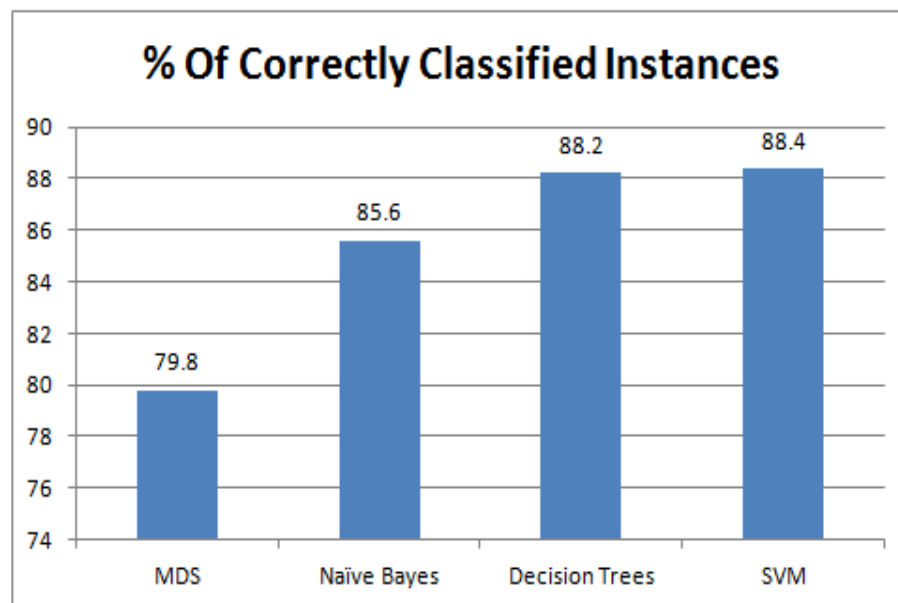
**Figure 12:** Results of 10-Fold cross validation, comparing the performance of MDS and other classifiers using the traditional features (excluding Difference To Second and Potential For Improvement).

As shown in the figure, the accuracy of all the machine learning techniques used on the data set is better than MDS’s “Decision Matrix” which is a very important result, as one of the aims of

this project is to show that using machine learning techniques give better results for this project than the hand-written rule classifier.

To address the other aims of this project by studying the effect of adding the new features (“Difference To Second” and “Potential For Improvement” for each of the individual fields), 10-fold cross validation was performed on the same set of training examples but this time including the new features, and the results are shown in figure 13.

| Technique                  | Percentage of Correctly Classified Instances |
|----------------------------|--|
| Current MDS implementation | 79.8 %                                       |
| Naïve Bayes                | 85.6 %                                       |
| Decision Tree              | 88.2 %                                       |
| SVM                        | 88.4 %                                       |



**Figure 13:** Results of 10-Fold cross validation, comparing the performance of MDS and other classifiers after adding the “Difference To Second” and “Potential For Improvement” for all individual fields.

The inclusion of the new features had the greatest impact on the naïve bayes classifier, and the other classifiers improved but not much improvement over the results when using the traditional features.

In figure 12, as expected the performance of Decision trees is better than Naïve Bayes. But SVM with an accuracy of 87.6 is approximately the same as the performance of Decision Trees 87.4,

which is a surprising result as SVM usually tend to perform better than Decision Trees. The same result was found in figure 13 after including the new features.

This is probably because of the use of the SVM with a linear kernel, which causes it to divide the classes linearly and so its performance is not better than Decision Trees which also divides the search space linearly, and the problem has more of a non-linear nature.

Another possibility is the set of training examples, the examples collected didn't concentrate on the difficult cases on the boundaries between the classes.

To try and improve the results of the training algorithms two steps were taken:

- First a bigger training set is collected containing 1000 examples containing:
  - 400 examples belonging to the class "Match".
  - 300 examples belonging to the class "Suggest".
  - 300 examples belonging to the class "No-Match".

The examples in each of the classes were carefully selected to be balanced so that half of the examples belonging to each class are easily or certain to be contained in the class and the other half to be the difficult cases lying on the boundaries between the classes and so to be able to use the full potential of all features.

- Secondly using a non-linear kernel with the SVM to try to improve the performance of the SVM, the kernel that is used is RBF discussed in section 3.4.

The results of the 10-fold cross validation should be confirmed by doing T-test comparing the classifiers but this will be delayed to be run on the next set of training examples.

The results of the new testing sets are discussed in the next section.

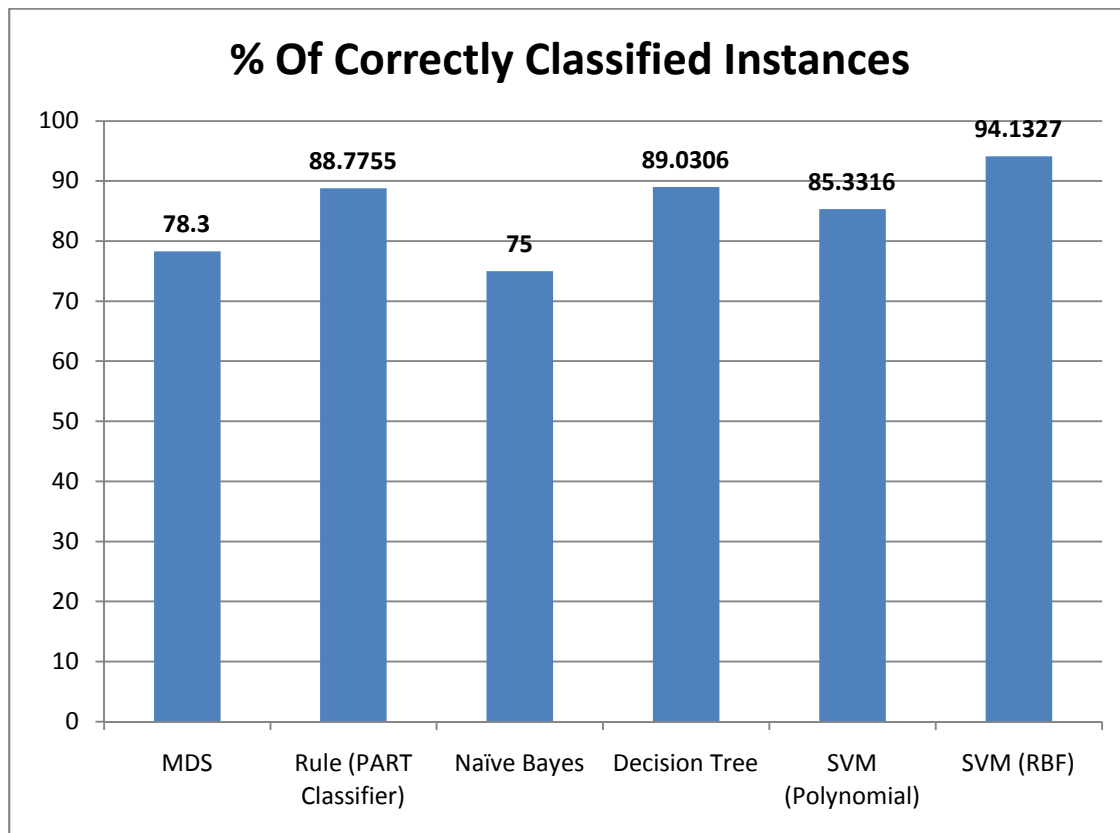
## 5.2 Second Round Of Testing

In this second round of testing, the number of examples collected was increased to 1000 examples, and SVM with RBF kernel was used to test if a non-linear classifier would perform better in this problem.

The classifiers trained for this training set are, "Rule Classifier", "Naïve Bayes", "Decision Tree", "SVM" with polynomial kernel, "SVM" with radial kernel.

We will start by the results without the new features (excluding “Difference To Second” and “Potential For Improvement”).

The results for 10-fold cross validation are shown in figure 14.

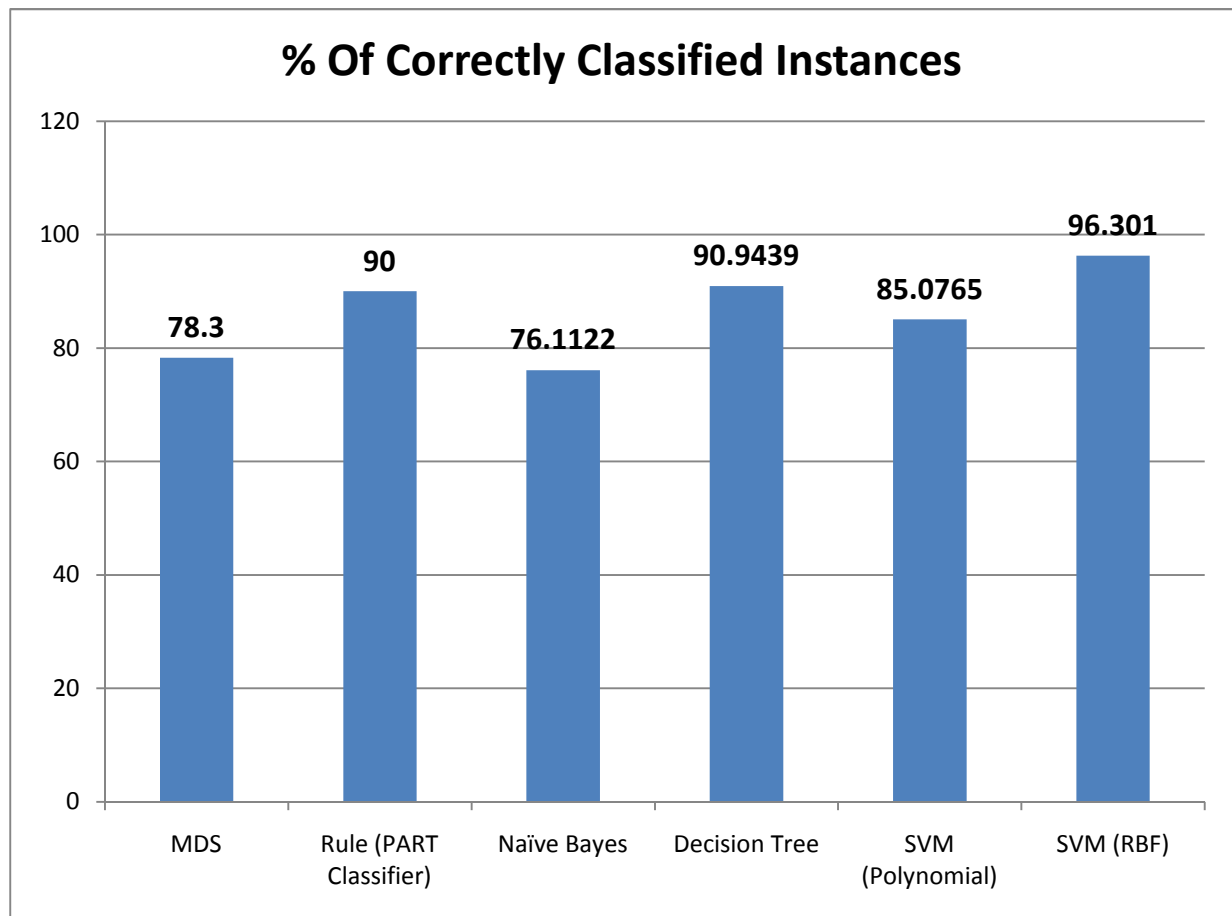


**Figure 14:** 10-Fold Cross Validation results, excluding the features “Difference To Second” and “Potential for Improvement”

As shown in figure 14, in the new training set the Naïve Bayes does worse than MDS and Decision Tree does better than SVM with a linear kernel, rule classifier has the same accuracy as Decision Tree.

SVM with a radial kernel has the best performance in the algorithms used with about 16% improvement over the current MDS decision mechanism.

In figure 15, we show the results of 10-Fold cross validation after adding the new features to the training set.



**Figure 15:** Results of 10-Fold cross validation, comparing the performance of MDS and other classifiers after adding the “Difference To Second” and “Potential For Improvement” for all individual fields.

Adding the features improves the accuracy of the classifiers but a small improvement.

Now we will use T-test to test:

- Machine Learning algorithms against MDS.
- Using SVM with radial kernel against SVM with linear kernel.
- Adding the new features against the traditional features.

### 5.2.1 T-Tests Results

We now check if the improvement in accuracy between different classifiers is statistically significant, this was done by running 10\*10-Fold Cross Validation runs, and reporting the average percentage of correctly classified instances (average accuracy) for each run then using these values to run a paired T test as all the algorithms are run on the same data set.



First we start by comparing the machine learning algorithms with best results against MDS's Decision matrix.

For all the results following the null hypothesis is:

- $H_0$ : Both classifiers have the same accuracy when tested on the training set.

In the next table we show the p-values resulting from testing MDS's Decision Matrix against the machine learning algorithms.

|                       | Rule Classifier | Decision Tree | SVM         | SVM(RBF)    |
|-----------------------|-----------------|---------------|-------------|-------------|
| MDS's Decision Matrix | 9.69627E-12     | 2.16345E-12   | 1.34805E-13 | 3.84561E-08 |

**Table 2:** p-values resulting from comparing machine learning algorithms against MDS's Decision Matrix.

From table 2 we can see that for machine learning algorithms p-value  $< 0.05$  which means we can say that the increase in accuracy when using machine learning algorithms is statistically significant.

Now we test the significance of the increase in accuracy when using a radial kernel function in the SVM classifier

|                             |                    |
|-----------------------------|--------------------|
|                             | SVM(Linear Kernel) |
| SVM (Radial Basis Function) | 6.33794E-06        |

**Table 3:** p-value resulting from comparing the accuracy of SVM with a linear kernel and SVM with radial kernel function.

From table 3, since p-value is  $< 0.05$  this means that we can reject the null hypothesis and say that increase of accuracy of the SVM classifier when using radial or non-linear kernel function is statistically significant.

From tables 3, 4 we conclude that using SVM with RBF kernel gives the best accuracy for this problem, and the improvement of its accuracy over the current MDS Decision Matrix is statistically significant.

Now we test the effect of adding the new features.

| Training the Algorithms using the traditional features | Training the algorithms after introducing the new features |               |             |             |
|--|--|---------------|-------------|-------------|
|  | Rule Classifier  | Decision Tree | SVM         | SVM(RBF)    |
|  | 3.92495E-07  | 3.20636E-05   | 0.005422338 | 0.484120065 |

**Table 4:** p-values resulting from comparing machine learning algorithms using traditional features and after including the new features, e.g. the value in the first cell is the result of testing Rule Classifier trained with traditional features against Rule Classifier after introducing the new features.

From table 4, for the SVM classifiers trained the p-value is  $> 0.05$  which means that we can't reject the null hypothesis and the increase in accuracy is statistically insignificant. But for other classifiers (Rule Classifier and Decision Trees) we get a p-value  $< 0.05$  which means that the increase in accuracy when using other classifiers due to the introduction of the new features is statistically significant.

Now we try to determine the best classification algorithm for this problem, we can see from the results so far (figure 15 and table 2) that SVM with radial basis function kernel produces the best accuracy for this problem even though the improvement in accuracy due to adding the new features is statistically insignificant. And to test this theory we perform T-tests to compare it against all other classifiers. The results for the tests are shown in table 5.

|                             | Rule Classifier with new features | Decision Trees with new features | SVM(linear kernel) with new features |
|-----------------------------|-----------------------------------|----------------------------------|--------------------------------------|
| SVM (RBF) with new features | 0.032015138                       | 0.028071947                      | 3.08089E-07                          |

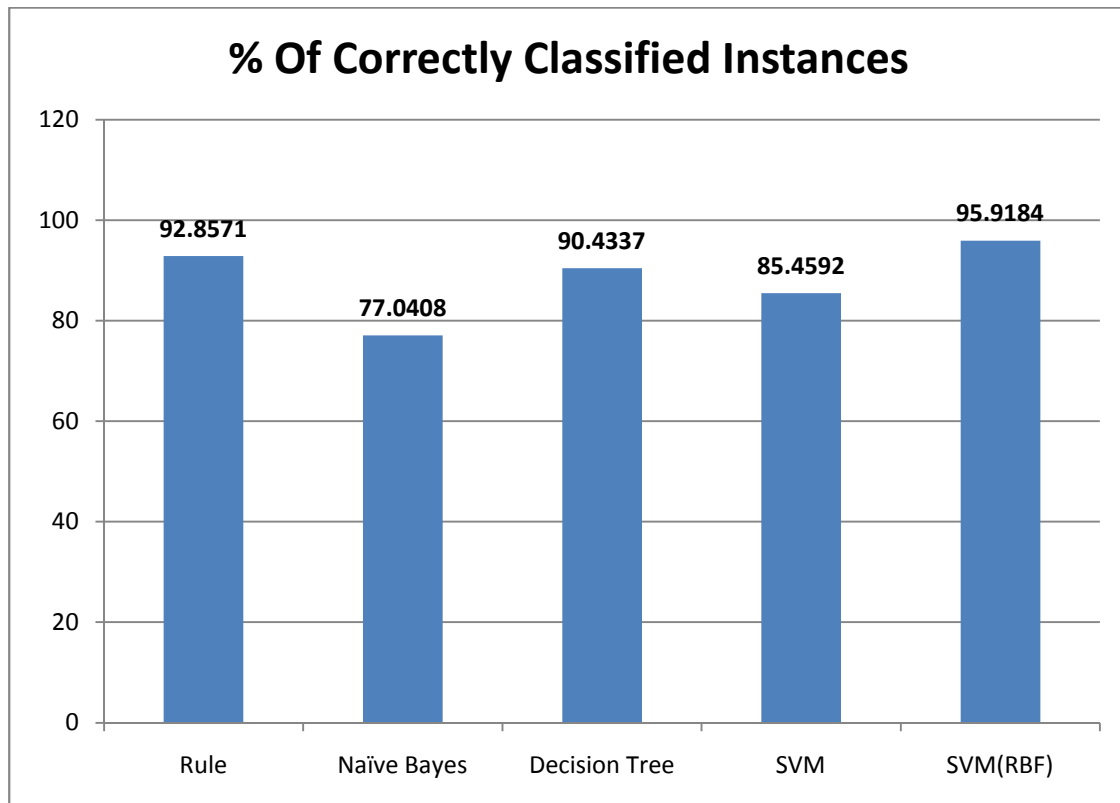
**Table 5:** p-values for comparing SVM(RBF) with new features against other classifiers with new features.

From table 5, p-values is  $< 0.05$  for all other classifiers which confirms SVM with Radial Basis Function kernel as the best classifier for this problem.

## 5.3 Further Testing

In the previous tests we added the new two new features at the same time to the training set, in this section we will try and separate the new features and add them one at a time and observe their individual effect on the accuracy of the classifiers used.

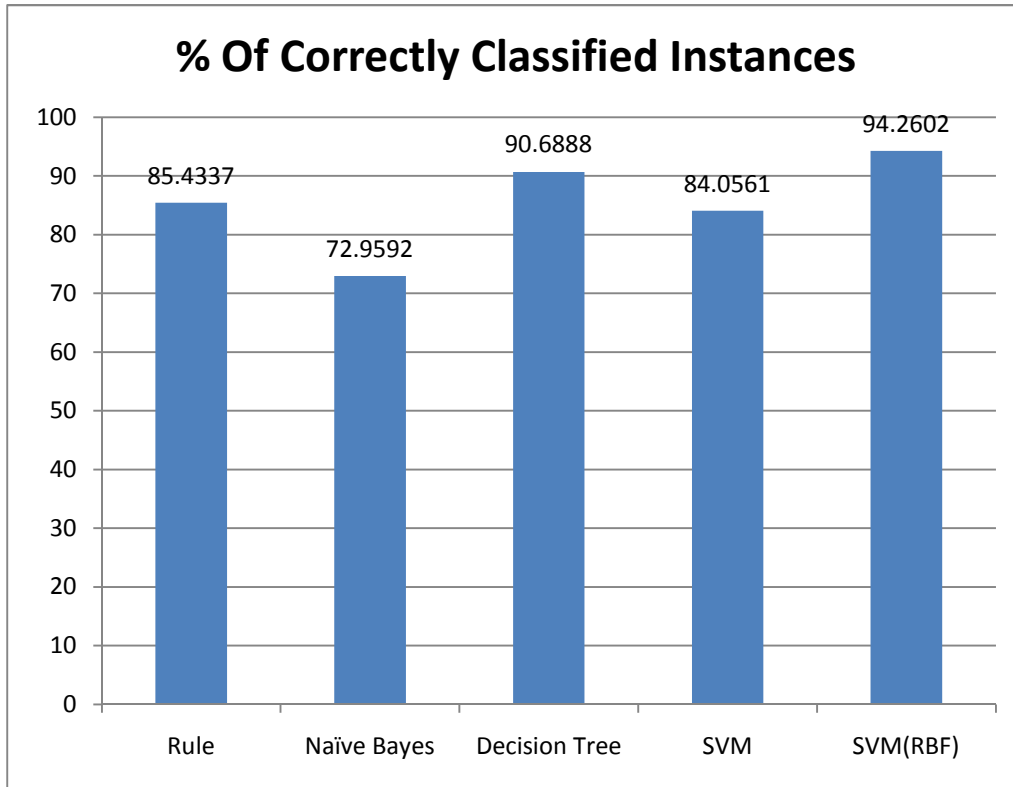
First the 10-Fold Cross validation results after adding only “Difference To Second” feature are shown in figure 16.



**Figure 16:** 10-fold cross validation results after adding only “Difference To Second” feature.

From figure 16, we can see that using only “Difference To Second” feature Naïve bayes and Rule Classifier have higher accuracy on the training set than when adding both features, but this is not the case for decision tree and SVM classifiers.

Now we see the effect of adding only the potential for improvement features in figure 17.



**Figure 17:** 10-Fold Cross validation results when adding only “Potential For Improvement” features to the training dataset.

In figure 17, Naïve Bayes and Rule classifiers both have lower accuracy when adding only potential for improvement features than using only the tradition features, and the same case as the results shown in figure 16 SVM and Decision Tree classifiers have a slight improvement in accuracy when adding potential for improvement features over the traditional features.

To sum up “Difference To Second” Feature alone improves the accuracy of all the classification algorithms, but adding “Potential For Improvement” features causes a slight improvement in SVM and Decision Trees but causes the Rule Classifier and Naïve Bayes to perform worse.

Which means that “Difference To Second” is the important feature to add and using “Potential For Improvement” features might not be very helpful.

## 5.4 Types of Errors

In this classification problem; different types of errors have different costs, since there are three classes that are arranged in descending order of how close the query to being a match or how close the query to the database entry:

1. “Match”.
2. “Suggestion”.
3. “No-Match”.

classifying a “Match” class instance as “Suggestion” is not as costly as classifying a “No-Match” instance as “Match”, so we present the types of errors in the decreasing order of their cost to the overall performance of the system:

1. Classifying a “No-Match” example as “Match”.
2. Classifying a “Match” example as a “No-Match”.
3. Classifying a “Suggest” example as a “Match”.
4. Classifying a “Match” example as a “Suggest”.
5. Classifying a “Suggest” example as a “No-Match”.
6. Classifying a “No-Match” example as a “Suggest”.

Since the classes are also arranged in the decreasing order of being a match, the first two types of errors in the list are the most costly; because they don’t appear on the boundary between classes but they skip a class. Unlike the rest of error types in the list where the error appears between the boundaries of two neighboring classes.

We now show the types of errors encountered by the learning algorithms while performing the 10-fold cross validation on the training set.

For SVM(RBF) with new features we show the percentage of each type of error of the overall training set size in table 6.

|          | Match  | Suggest  | No-Match |
|----------|--------|----------|----------|
| Match    | -      | 0.539%   | 0%       |
| Suggest  | 1.617% | -        | 1.617%   |
| No-Match | 0%     | 1.88679% | -        |

**Table 6:** The percentage of each type of error in the size of the whole training set for SVM(RBF).

For Decision Tree with new features, the percentage of each type of error is shown in table 7.

|          | Match | Suggest | No-Match |
|----------|-------|---------|----------|
| Match    | -     | 2.8302% | 2.2911%  |
| Suggest  | 0%    | -       | 2.2911%  |
| No-Match | 0%    | 2.1563% | -        |

**Table 7:** The percentage of each type of error in the size of the whole training set for Decision tree classifier.

In tables 6 and 7, the cells with red background represent the most costly types of errors.

In table 6, when using SVM with RBF there is no errors of the first two types made, and the highest percentage of the errors made are of the least costly type (error 6 from the list above). In table 7 when using Decision Tree a percentage of the error made lies in the top right cell which represents the second highest costly error, and the highest percentage of errors are of type 4.

These results support our theory that SVM with RBF kernel function is the most accurate classifier for this problem.

## 6 Conclusions

In conclusion, this project shows:

- A statistically significant improvement in the accuracy of the decision mechanism when using machine learning algorithms over the hand written "Decision Matrix".
- The introduction of the new features causes improvement, but this improvement is not statistically significant in SVMs.
- SVM with Radial Basis Function kernel is the best classifier for this problem, even though the improvement due to the introduction of the new features is statistically insignificant, but it still has statistically better accuracy than all the other classifiers.

## 7 Future Work

As mentioned the process of getting the best training set is very time consuming and getting the perfect training set that provides the highest information gain is a very hard task, and so to solve this problem implementing active learning (see section 2.3.3) will help in getting a good training set requiring minimum user intervention during the process.

In active learning a committee of different classifiers are trained using an initial seed (a small number e.g. 10) of labeled examples, then given a bigger list of unlabeled examples going over each example, each classifier in the committee classifies the new example, if the classifiers in the committee reach a consensus over the class of the new example, then this new example is added to the training set with the new label acquired automatically.

But if the classifiers of the committee disagree on the classification of the new example then the user is asked to confirm the class of the example and the new example with its new label is added to the training set. And this is repeated in cycles until adding new examples to the training set until we are satisfied with the accuracy of the classifier or we reach a pre-determined number of examples.

The result of this process is ending with a training set that offers the highest information gain and only requiring user's insight in the difficult instances on which the committee disagrees.

Another aspect that can be improved is scaling the "Difference To Second" feature to include difference to more matches (e.g. Third, Fourth .... etc.), since as the size of the database increases the probability of getting two close matches for a query increase and only using difference to second may not be very useful but if we scale this feature to include more differences we think that it will cause an improvement in the accuracy of the classifier, and maybe will cause the use of the new features to cause a statistically significant improvement in the accuracy of the classifiers.

## 8 References

1. Patrick A. V. Hall and Geo R. Dowling, "Approximate string matching". *Computing Surveys*, 12(4):381-402, December 1980.
2. N. Koudas, A. Marathe, and D. Srivastava, "Flexible String Matching against Large Databases in Practice," *Proc. 30th Int'l Conf. Very Large Databases (VLDB'04)*, pp. 1078-1086, 2004.
3. W. Cohen, P. Ravikumar, and S. Fienberg. "A comparison of string distance metrics for name-matching tasks". In *Proc. IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*. 2003.
4. G. Navarro, "A Guided Tour to Approximate String Matching", *ACM Computing Surveys*, vol. 33, no. 1, pp. 31-88, 2001
5. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. "Duplicate record detection: A survey." *TKDE*, 19(1):1-16, 2007.
6. Levenshtein VI, "Binary codes capable of correcting deletions, insertions, and reversals", *Soviet Physics Doklady* 10: 707-10, 1966
7. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443-453, 1970.
8. M.A. Jaro, "Unimatch: A Record Linkage System: User's Manual," technical report, US Bureau of the Census, Washington, D.C., 1976.
9. E. Ukkonen, "Approximate String Matching with q-Grams and Maximal Matches," *Theoretical Computer Science*, vol. 92, no. 1, pp. 191-211, 1992.
10. A.E. Monge and C.P. Elkan, "The Field Matching Problem: Algorithms and Applications," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD '96)*, pp. 267-270, 1996.
11. L. Philips, "Hanging on the Metaphone," *Computer Language Magazine*, vol. 7, no. 12, pp. 39-44, Dec. 1990.
12. Ristad, E. S., and Yianilos, P. N. "Learning string edit distance". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(5):522-532. 1998.
13. M. Bilenko and R. J. Mooney. "Adaptive duplicate detection using learnable string similarity measures". In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pages 39-48, 2003.
14. L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate String Joins in a Database (Almost) for Free," *Proc. 27th Int'l Conf. Very Large Databases (VLDB '01)*, pp. 491-500, 2001.



15. W. Cohen. "Data integration using similarity joins and a word-based information representation language". *ACM Transactions on Information Systems*, 18(3):288--321, July 2000.
16. S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and Efficient Fuzzy Match for Online Data Cleaning," *Proc. 2003 ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03)*, pp. 313-324, 2003.
17. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19--27, 2001.
18. W. Cohen and J. Richman. Learning to match and cluster entity names. In *proceedings of SIGKDD*, Edmonton, July 2002.
19. E. Ukkonen, Approximate string-matching over suffix trees, in *Combinatorial Pattern Matching, CPM'93* (A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, eds.), *Lect. Notes in Computer Science*, vol.684, Springer-Verlag, 1993, pp. 228-242.
20. A. Cobbs. Fast approximate matching using suffix trees. In *Combinatorial Pattern Matching, 6th Annual Symposium (CPM'95)*, pages 41-54, 1995.
21. A.E. Monge and C.P. Elkan, "The Field Matching Problem: Algorithms and Applications," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD '96)*, pp. 267-270, 1996.
22. T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology*, vol. 147, pp. 195-197, 1981.
23. Monge, A., and Elkan, C. 1997. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery*.
24. Tejada S., Knoblock C. A., and Minton S. 2001. Learning object identification rules for information integration. *Information Systems* 26(8):607-633.
25. T. Bozkaya and Z. M. Ozsoyoglu. Distance based indexing for high dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD Conference on Management of Data*, pages 357-368, 1997.
26. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta, 2002.
27. US patent 1261167, R. C. Russell, "(untitled)", issued 1918-04-02.
28. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); *The WEKA Data Mining Software: An Update*; *SIGKDD Explorations*, Volume 11, Issue 1.
29. <http://www.dtreg.com/svm.htm>.