# Debtor Identification

## 1- Problem Definition:

Finding a person based on their identifying features (Name, Address …) is easily achieved when the information entered is identical to one of the instances of a dictionary. However personal information can be distorted in a number of ways based on the method of entry: misreading, mishearing or mistyping by a human or an automated machine, each method has a pattern of errors that is handled in a specific way. Names are widely used as an identifier in record linkage but names can vary greatly due to differences in transcription and character usage depending on the language, typographical errors, phonetic errors or a number of other issues. Several studies have been conducted on the subject of approximate string matching, we will arrive at the optimal way to approach this problem based on their findings. The success of a project of this nature is measured by how well it deals with discrepancies in the provided string to be matched. Recovering as many relevant matches as possible while keeping the number of irrelevant matches low. In many cases it is difficult to determine if a query is a different spelling of a particular name or is a different name altogether

The goal of this paper is to propose an approximate string matching solution for fast and efficient search and identification of prospective debtors.

## 2- Names Variations:

- Typographical errors: Due to motor coordination slips, the writer does not know the correct spelling or has a misconception about how the word is spelled.
- Phonetic errors: occur when the writer substitutes a phonetically sound spelling of the word but an incorrect one nonetheless. Phonetic errors tend to distort the string more than other variations.
- Double names: surnames and first names can consist of two elements which are not always shown.
- Alternate names: Where an individual changes their name or is called another name at a later stage in their life.
- Qualifiers (Mr., Jr…)
- Abbreviated names.

## 3- Spelling error patterns:

More than 80 percent of all spelling errors are one of four classes of single error-misspelling:

- Insertion of a letter: Misreading, pressing a key twice…
- Omission of a letter
- Substitution of a letter
- Transposition of two adjacent letters

In a study by Pollock on 50000 misspelled words, over 90 percent were single error-misspellings. The next most common multi-error misspellings are different combinations of two of the preceding four classes. Furthermore 20% of errors occurred in the third letter and 8% in the first. As we will see phonetic algorithms usually use the first letter as an identifier for the word under the assumption that errors are rare in the first letter.

This analysis applies to misspelled words in general. Even so they are informative and might give at least an indication to error patterns in spelling names.

## 4- The Dictionary:

The dictionary will contain all the information relating to each name in the form of a dataset composed primarily of six features:

- Collected first names, assigned unique numbers(This also applies to middle and surnames)
- Collected middle names
- Collected surnames
- The occurrence frequency of each entry.
- All possible Q-grams of each entry are recorded and then gathered in a heap. For every possible q-gram a list is constructed comprised of all the numbers of entries containing that q-gram.
- The Soundex index.

The entries of the dictionary are collected from the following sources:

- Debtor lists.
- Collectia employee list.

- A list of the most common first and surnames in relevant countries (the relevance of a country is measured by the number of debtors from that country)
- A list of variations in the spelling of the most frequently occurring names.

An entry from the dictionary will be judged relevant if it looks or sounds like the query entry.

The dictionary will be divided into several subsets with sizes ranging from a small number to the full dictionary. The purpose of this practice is to test the speed of each algorithm against the size of the dictionary.

It is imperative to try not to repeat string processing or calculations that's why q-grams and the Soundex index will be stored in the dataset rather than re-computing every time we need to perform a matching procedure.

## 5- Edit Distance Metrics:

To determine how similar two words are it makes sense to have a measure of distance between them.

Approximate string matching, colloquially referred to as "Fuzzy string searching" is a subset of pattern matching in the wider text processing domain, where algorithms are applied to find the pattern which approximately matches an entry in a given dictionary. It consists in finding one or more or all the occurrences of a string or patterns.

There are many different metrics each provides a different approach to measuring the distance between two stings. These metrics apply some or all of the classes of error mentioned in (3- Spelling error patterns) and they obey some or all of the following properties:

- Non-negativity. The distance between two strings is greater than or equal two 0. $d(s,t) >= 0$
- Identity. Distance equals 0 only when the two strings are identical. $d(s,t) = 0$ only when $s = t$
- Symmetry. $d(s,t) = d(t,s)$
- Triangle inequality $d(s,u) <= d(s,t) + d(t,u)$

Where s,t,u are strings. d() is a distance function.

The precision and recall of each method will be measured and compared to find the most effective methods. In a hybrid model through weighting the methods' findings based on their effectiveness, the one with the highest precision will have a proportional effect on the output. The query entries can be pre-processed before matching so this is an offline approximate string search.

## 5.1. Longest common Substring distance:

Calculated by counting the number of insertions and omissions necessary to convert a string into another. It measures the longest sequence formed by pairing characters from the two strings while keeping their order intact.

## 5.2. The generalized Levenshtein distance:

Counts the weighted sum of insertions, omissions and substitutions. The weights are positive penalties on the previous operations that specify what class of error is costlier than the others. The weights are assigned based on the nature of the errors which are as mentioned dependent on the method of entry.

## 5.3. Damerau-Levenshtein distance:

An extension of the generalized Levenshtein distance. It counts the weighted sum of insertions, omissions, substitutions and transpositions. A minimization over possible transpositions between the current character and all untreated characters where the cost of a transposition increases with the distance between transposed characters. The weights can be set depending on the likelihood of letters substituting for each other.

## 5.4. Q-grams:

A sequence of q consecutive characters. A dictionary will store all possible q-grams for names in a character vectors which can be obtained using a sliding window. The more q-grams two strings have in common the more likely they are a match. Q-gram distance equals infinity when q is smaller than the strings being matched.

The Jaccard coefficient equals to 1 - (shared q-grams / q-grams of both strings discarding duplicates).

$$d_{\text{jaccard}}(s,t;q) = 1 - \frac{|\mathcal{Q}(s;q) \cap \mathcal{Q}(t;q)|}{|\mathcal{Q}(s;q) \cup \mathcal{Q}(t;q)|}$$

The value of q is dependent on the application and for our purposes 1-grams hold little information for they are just a group of every letter in the string. We are dealing with short strings so bigrams (q=2) and trigrams (q=3) are the main focus.

## 5.6. Jaro-Winkler distance:

This is a heuristic method based on the reasoning that substitutions and transpositions of characters are caused by typing error and characters that are too many positions apart are less likely to occur taking into account keyboard layouts and other common error patterns. It counts the number of matching characters in two strings that are positioned closely. A penalty is also added for character transpositions.

Winkler extended the metric by adding a penalty for mismatches in the first four letters. Mistakes in the first letter are rare and mistakes in the first two letters is even more infrequent, so it would extremely unlikely that a writer will get the first four letters wrong. From this we deduce that the more mismatches in the first four letters the less likely the two strings are a match.

The Jaro-Winkler distance is measured on a normalized scale (0-1). 1 indicates a perfect match and 0, a complete mismatch. This method was designed with short character vectors in mind so it is expected to outperform other distance measurements in precision.

## 5.7. Soundex:

A similarity key class of algorithms based on the phonetic structure of a word and the general behaviour of the sound of consonants and vowels. The idea is to map each word into a key that approximately matching keys will be identical or at least similar. These algorithms are therefore fast. A similarity key is computed for the misspelled word, the keys are used to

construct pointers to correction candidates and a fast lookup in the dictionary. Two words are equivalent if they have a similar phonetic structure.

Phonetic algorithms are language specific which poses an obvious problem when applied to a different language which have differing pronunciations for letters.

The similarity key (phonetic hash) in the case of the Soundex algorithm is constructed by converting a word to a four-character code, which is used to identify prospective matches. An inverted index is built from these reduced forms called the Soundex index. Similar sounding words have equivalent or similar keys. It is computed as follows:

- Retain the first letter.
- change all occurrences of the following letters to '0'
  A, E, I, O, U, H, W, Y.
- Assign the following numbers to the appropriate letters
  - B, F, P, V to '1'
  - C, G, J, K, Q, S, X, Z to '2'
  - D,T to '3'
  - L to '4'
  - M, N to '5
  - R to '6'
- Recursively remove one out of each pair of consecutive equivalent numbers.
- Remove all zeros from the resulting string.
- Pad the result if necessary by adding trailing zeros or removing the rightmost numbers, to complete the four character sequence (letter, number, number, number)

Vowels are viewed as interchangeable when transcribing names and consonants with similar sounds are grouped in equivalent classes. Taking these rules into account, similar sounding names have approximately the same Soundex index.

Soundex was designed for the English language, applying it to Danish names or an international name setting would require either transliterating the words into the English vocabulary or expanding the vocabulary of the algorithm and modifying it to deal with the differences. It has been modified to many languages and does a fairly good job of

matching. Even better than modifications and extensions introduced by other methods such as Metaphone. The Phonex algorithm is a hybrid of Soundex and Metaphone and can substantially reduce the number of false positive matches in successive pruning operations.

# 6- Knowledge based Algorithms:

## 6.1. Rule based algorithm:

An attempt to represent common knowledge of spelling error patterns as a set of rules on how to transform a misspelled word into a valid one. Candidate matches are produced by applying the set of rules and isolating entries that satisfy these rules. Each candidate is ranked by a score based on the probability of an error being corrected by a corresponding rule.

## 6.2. Neural networks:

Neural networks are a set of massively parallel simple processing units called neurons, the interconnections between those neurons play a vital rule in the learning capability of the network.

A neural network can be adapted to string matching since they are empirically universal function approximators. They have an inherent ability to deal with incomplete or noisy input, in this case spelling errors. It can also be trained to recognize the writer's error patterns and gradually improve over repeated usage getting closer to maximum accuracy possible.

One major drawback of using neural nets in spelling corrections is the training time, running the training cycles requires a long time for a small dictionary of less than a thousand names. So running a neural network model on the entire dictionary might be infeasible.

## 6.3. Support vector machine:

An SVM can be imagined as a surface that creates a boundary between points of data plotted in multi-dimensions that represents attributes and instances of these attributes. The goal is to create a flat boundary

(Hyperplane) to make the classes of the data points as homogeneous as possible.

The combined outputs of the edit distance algorithms can be pooled and used as training data for a learning scheme using an SVM.

The problem with applying such methods to string matching is that to optimize the process and produce statistically significant results we need expert analysis of the genealogy of names and a comprehensive study of error patterns specific to each supplier of information. This would require a massive amount of research and computation and consulting linguistic experts to form a learning scheme capable of recognizing patterns of error and the origin and possible variations of a name. Studies applying these techniques to string matching show only a slight improvement of accuracy over individual edit distance metrics.

Furthermore since the vast majority of errors are single error-misspellings, the edit distance measurements are favourable in this situation, taking into account the complexity of the resulting system, the benefits reaped from increasing the complexity, the time required to build the system in terms of computation and the speed of information retrieval.

## 7- Implementation:

The main environment we'll be using in this project is R. An objection to use R in a string processing task might be that R is not a scripting language like Perl, TCL or Python.

The reason behind using R is the flexibility and intuitive programming environment that it provides. Data manipulation is a specialty of R, even though string are not. There are many packages that can be used to manipulate strings (including matching using regular expressions) which are more than enough for our purposes. There is an added benefit of a large array of machine learning algorithms implemented in R, in many cases by the researchers who invented them, which can be used to extend the project while remaining in the same environment.

Some of the necessary steps to construct the system are:

1- Cleaning the strings to remove any possible distortions that may impede the matching process. That means special characters, extra blank spaces, duplicates and qualifiers.

2- Parsing the text into its initial components and storing these components in separate fields in a dataset.
3- Counting the occurrence frequency of each entry to isolate the most common ones and expanding them with possible spelling variations. A possible improvement could be writing the entries as regular expressions to accommodate most spelling variations in a compact form.
4- Omitting duplicates to get a set of unique entries.
5- Generating Q-grams (Q=1, 2, 3, 4) and storing them in lists linked to each entry.
6- Generating Soundex index.

After the dataset is complete we start with applying the algorithms to the dictionary by selecting a group of test subjects. Experimentation will start on a small set to figure out the optimal way to handle the processing of the complete set. Care is taken to ensure that the code for fetching the results is equivalent in all algorithms to accurately compare their performance.

Each algorithm will generate a list of possible matches with some exceptions that return one match. The accuracy is measured by counting the number of relevant matches against the number of all matches retrieved.

The test set will be processed by all the algorithms and, in an effort to combine the power of the algorithms, exact string matching techniques are used to compare the results and see which matches have the highest vote. That means that if a match is suggested by two algorithms then it will have two votes.

Of course the performance of the algorithms is not equal consequently the votes should not be equal so, based on the measurements of accuracy obtained from the test set, the output of each algorithm will be weighted insuring that the method with the higher accuracy will have more effect on the final result.

A user interface will be implemented using C# which will essentially deal with the end-users and export data into R for processing and import the results.

The user enters a debtor or a list of debtors to be matched and each part of each name will be assessed individually and a group of possible

alternatives to the corrupt data and combinations of first, middle and surnames will be presented.

To further automate the process we have multiple website (Experian, TDC, IDQ, Google…) that can be used to match the debtors. Some are based on other identifying features such as phone numbers or addresses….

Each entry in the list provided is matched against the databases of the websites by querying them using the C# interface. If multiple matches are returned the user will be consulted as to which is the correct one.

The dictionary needs to be updated regularly to accommodate new debtor information and have a more comprehensive coverage of possible entries. In the case that a name is not found the user adds the corrected entry to the dictionary.

## 8- Conclusions:

The choice of approximate string matching algorithm is largely dependent on the nature of the data, the required level of accuracy and the error patterns.

Increasing the complexity only makes sense when it provides a big improvement of performance. As mentioned the vast majority of errors are single error-misspellings so it doesn't make sense to make the system more complex than it needs to be.

There is a good chance that our problem will be largely solved by the simple edit distance measurements. After implementing and optimizing the system, based upon statistical results the need for a more complex structure will be evaluated. The use of sophisticated machine learning algorithms on this task might be ill advised because of the additional time and effort and the relatively small rewards reaped from applying such a system.

A proper application of machine learning algorithms which utilizes the full power of these methods could be using a semi-supervised learning scheme to create a knowledge based system for clustering the debtors into homogeneous groups, predicting the outcome of attempting to collect the debt (Essentially is the debtor good or bad), and scoring debtors based on a number of factors also determined by machine learning algorithms.