IBERAMIA-SBIA 2000 Open Discussion Track Proceedings, p. 217-226, Atibaia - Sao Paulo (Brasil), November 19-22 2000.

Clustering of Similar Values, in Spanish, for the Improvement of Search Systems

Sergio Luján-Mora[1] and Manuel Palomar[2]

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante,
Campus de San Vicente del Raspeig,
Ap. Correos 99 – E-03080 Alicante, Spain
{slujan, mpalomar}@dlsi.ua.es

Abstract. The ability to correctly access electronically stored information is becoming increasingly important as stored information itself keeps growing continuously. One of the problems that face search systems is the inconsistency found among the stored values: i.e., the very same term may have different values, due to misspelling, a permuted word order, spelling variants and so on. The clustering of the values that refer to a given term solves this problem by replacing these clustered values with one single value. In this paper, we present a clustering method that allows us to reduce on the existing inconsistencies in databases and, thus, improve on the performance of both search and information retrieval systems. The method we propose here gives good results with a considerably low error rate.

Keywords: Natural Language Processing, Information Storage and Retrieval, Database, Data Mining, Pattern Matching

Open Discussion Track

Topics: Knowledge Discovery and Data Mining

[1] Phone: (+34) 965 90 34 00 ext. 2514   Fax: (+34) 965 90 93 26
[2] Phone: (+34) 965 90 36 53   Fax: (+34) 965 90 93 26

Clustering of Similar Values, in Spanish, for the Improvement of Search Systems

Sergio Luján-Mora and Manuel Palomar

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante,
Campus de San Vicente del Raspeig,
Ap. Correos 99 – E-03080 Alicante, Spain
{slujan, mpalomar}@dlsi.ua.es

Abstract. The ability to correctly access electronically stored information is becoming increasingly important as stored information itself keeps growing continuously. One of the problems that face search systems is the inconsistency found among the stored values: i.e., the very same term may have different values, due to misspelling, a permuted word order, spelling variants and so on. The clustering of the values that refer to a given term solves this problem by replacing these clustered values with one single value. In this paper, we present a clustering method that allows us to reduce on the existing inconsistencies in databases and, thus, improve on the performance of both search and information retrieval systems. The method we propose here gives good results with a considerably low error rate.

1 Introduction

Existing information systems provide rapid and precise access to the information stored in databases. One of the main uses of databases is find information. Traditional search systems work by matching the term that is being searched with the values stored in the corresponding database. If the information contained in databases is inconsistent (i.e., if a given term appears with different values because several denominations exist, or because it is misspelled), a search using a given value will not provide all the available information about the term.

If we consult a database that stores information about university researchers, (i.e., researcher's name, researcher's university, etc.), and we wish to obtain a list of all of the researchers who work at the University of Alicante, for example, we may easily find that there are different values for this university: "*Universidad de Alicante*" (in Spanish), "*Universitat d'Alacant*" (in Catalan) and "*University of Alicante*" or "*Alicante University*" (in English).

This problem not only affects the searches, but is also important when the same term appears with different values when it is displayed on a screen or a printer, giving a bad impression. The problem of inconsistent values in a given field, in bibliographical databases, is usually resolved by using *authority files*.

In this paper, we present an automatic method for reducing on the inconsistency found in existing databases: i.e., all the values that refer to a given term are clustered by measuring their degree of similarity.

The remainder of the paper is structured as follows: Section 2 outlines the origin of the problem and the possible causes that give rise to the different variants that appear for the same term; Section 3 introduces our method for reducing on the inconsistencies found in current databases; Section 4 explains the core of our study and details the technical aspects of our method; Section 5 provides an evaluation of the method; and finally, our conclusions are presented in Section 6.

2 The Problem

The problem of the inconsistency found in the values stored in databases may have two principal causes:

1. If the number of possible values that a single field can accept is not controlled, a given person, (or different persons), may insert the same term with different values. A database that stores the names of the departments of a university, for instance, may have several different forms (i.e., the use of upper-case letters or abbreviations): "*Departamento de Lenguajes y Sistemas Informáticos*", "*Depto. de Lenguajes y Sistemas Informáticos*", "*Dpt. de lenguajes y sistemas informáticos*", etc.
2. When we try to integrate different databases, one or more of them may suffer from the above-mentioned problem. Even if the consistency of their contents has been guaranteed separately, however, the criteria used for establishing the consistency of each one might well be different and integrating them all could cause inconsistency problems. If we wish to integrate three different databases that store bibliographical information, for example, the authors might well appear in different forms in each one: i.e., full names, "*Miguel de Cervantes Saavedra*", or by last names first and then the first name, "*Cervantes Saavedra, Miguel de*", or by first name and last name only, "*Miguel de Cervantes*".

2.1 Causes

After analysing several databases with information in Spanish, we have noticed that the different values that appear for a given term are due to a combination of the following causes:

1. The omission or inclusion of the written accent: "*Asociación Astronómica*" or "*Asociacion Astronomica*".
2. The use of upper-case and lower-case letters: "*Departamento de Lenguajes y Sistemas Informáticos*" or "*Departamento de lenguajes y sistemas informáticos*".
3. The use of abbreviations and acronyms: "*Dpto. de Derecho Civil*" or "*Departamento de Derecho Civil*".
4. Word order: "*Miguel de Cervantes Saavedra*" or "*Cervantes Saavedra, Miguel de*".
5. Different denominations: "*Unidad de Registro Sismológico*" or "*Unidad de Regis-*

tro Sísmico".

6. Punctuation marks: (Hyphens, commas, semi-colons, brackets, exclamation marks, etc.): "*Laboratorio Multimedia (mmlab)*" or "*Laboratorio Multimedia – mmlab*".
7. Errors: Misspelling (apart from the written accent), typing or printing errors (absence of a character, interchange of adjacent characters, etc.): "*Gabinete de imagen*" or "*Gavinete de imagen*".
8. Use of different languages: "*Universidad de Alicante*" (Spanish) or "*Universitat d'Alacant*" (Catalan).

There has been great interest in studying the quality of the information stored in databases for a long time [8, 9], and diverse methods have been developed for the reduction of the inconsistency found in databases. In this area, James C. French [2, 3], who developed a method that allows the automatic creation of authority files for bibliographical catalogues, should be mentioned.

3 Intuitive Proposal of a Method to Reduce the Inconsistency Found in Databases

Our method was developed from French's clustering algorithm [2, 3], to which we have added a new distance and developed different evaluation measures. Our algorithm resolves all the problems detailed in Section 2, except the fifth and the eighth, which depend on how different the two strings that represent the same term are.

The method that we propose can be divided into six steps:

1. *Preparation*. It may be necessary to prepare the strings before applying the clustering algorithm.
2. *Reading*. The following process is repeated for each of the strings contained in the input file:
   - Read a string
   - Expand abbreviations and acronyms
   - Remove accents: e.g., *A* substitutes *Á* and *À*, and *a* substitutes *á* and *à*
   - Shift string to lower-case
   - Store the string: If it has been stored previously, its frequency of appearance is increased by one unit
3. *Sorting*. The strings are sorted, in descending order, by frequency of appearance.
4. *Clustering*. The most frequent string is chosen (*canonical representative*) and it is compared to the rest of the strings, using a measure of similarity. This process is repeated, successively, until all the strings have been clustered.
5. *Checking*. The resulting clusters are verified and the possible errors are located and corrected.
6. *Updating*. The original database is updated. The strings of a cluster are replaced by its canonical representative.

The four first steps are the main topic of this paper. The remaining steps will be studied and presented in a future work.

4 Technical Description of the Method

4.1 Previous Processing

The strings undergo a previous processing to obtain better results from the clustering. The objective of this processing is to avoid the three first causes of the appearance of different forms for the same term (see Section 2.1): i.e., accents, lower-case/upper-case and abbreviations. The accents are eliminated, the string is converted to lower-case and the abbreviations are expanded.

4.2 String Similarity

The similarity between any two strings must be evaluated. One solution is to use the *edit distance* or *Levenshtein distance* (LD) [6]. This distance has been traditionally used in approximate-string searching and spelling-error detection and correction (causes 6 and 7). The LD of strings x and y is defined as the minimal number of simple editing operations that are required to transform x into y. The simple editing operations considered are: the insertion of a character, the deletion of a character, and the substitution of one character with another. In our method, we have taken a unitary cost function for all the operations and for all of the characters.

The LD of two strings m and n in length, respectively, can be calculated by a dynamic programming algorithm [5]. The algorithm requires $\Theta(mn)$ time and space.

We consider two strings to be similar if their LD is lower than a threshold (α). A fixed threshold cannot be established, as the longer strings will obviously have more errors than the shorter ones. A relative threshold is therefore established for the form:

$$\alpha_{LD} = \beta_{LD} min(|x|,|y|), \tag{1}$$

that is, the threshold is a fraction, β_{LD}, of the length of the shorter string.

4.3 Computation Reduction

As we have already mentioned, the LD has a temporal and spatial complexity of $\Theta(mn)$. In order to reduce the calculating time, the LD can be avoided when the difference in lengths between strings is greater than the threshold:

$$LD(x, y) < \alpha_{LD} \quad \text{if} \quad \| x| - |y \| < \alpha_{LD} , \tag{2}$$

as demonstrated in [2].

When the similarity of two strings is calculated, we do not consider the blanks, the punctuation marks, and so on (cause 6). We have defined a new distance, *length distance* (LEND) of two strings, which is calculated as the difference in absolute value between the characteristic lengths of the two strings. The term *characteristic length* (cl) of a string represents the number of characters (digits and letters of the Spanish alphabet: 37 altogether) that a string contains.

In using this distance, the LD will be only calculated when the LEND is lower than a threshold. As with the LD, we use a relative threshold that depends on the strings that are being compared; the threshold is a fraction of the minimum characteristic lengths of the two strings:

$$\alpha_{LEND} = \beta_{LEND} min(cl(x), cl(y)) . \quad (3)$$

Even with the use of this distance as filter, however, too many LD calculations are required. Furthermore, it is not very accurate as it is based exclusively on the length of the strings. In order to reduce on the number of calculations required, another distance that is easier to calculate than the LD is used: the *transposition-invariant distance* (TID) of two strings [1]. This distance is more difficult to calculate than the LEND, but it is also much more accurate, as it considers both the lengths of the strings and the characters that form the strings. In any case, it has been proven that $TID(x, y) \leq LD(x, y)$, and this can be done at a lower computational cost.

4.4 Distance Metric Improvement

If two strings contain the same words (variant forms of the same term) but with a permuted word order (cause 4), the LD will not permit their clustering. To solve this problem, we introduce another distance that we call the *invariant distance from word position* (IDWP). It is based on the *approximate word matching* referred to in [2]. To calculate the IDWP of two strings, they are broken up into words (we consider a word to be any succession of digits and letters of the Spanish alphabet). The idea is to pair off the words so that the sum of the LD is minimised. If the strings contain different numbers of words, the cost of each word in excess is the length of the word.

It is almost always the case that $IDWP(x, y) < LD(x, y)$, although this is not always true. This is why both criteria (IDWP and LD) are used in deciding whether two strings are similar or not. First, the LD is evaluated: two strings are similar if their LD is smaller than the corresponding threshold. Otherwise, the IDWP is then calculated, as it is more expensive to calculate (by means of a *branch and bound* scheme [7]). The IDWP needs another threshold, which is calculated from the characteristic lengths of the two strings:

$$\alpha_{IDWP} = \beta_{IDWP} min(cl(x), cl(y)) . \quad (4)$$

4.5 Algorithm

The clustering algorithm chooses the strings, from greater to smaller frequency of appearance, since it assumes that the most frequent strings have a greater probability of being correct, and thus, they are taken as being representative of the rest.

We show the resulting algorithm in Table 1. The algorithm is basically the *leader algorithm*[3] [4]. This algorithm is chosen as opposed to more elaborate algorithms (e.g. *k-means algorithm*, *Fisher algorithm*) because they are slower and the number of clusters is unknown. As seen, the algorithm depends on four parameters: β_{LEND}, β_{TID}, β_{LD}, and β_{IDWP} (used in the calculation of α_{LEND}, α_{TID}, α_{LD}, and α_{IDWP}, respectively). The two first parameters reduce the number of LD and IDWP that are calculated, diminishing the time of calculation, but making worse the produced clusters.

Table 1. Clustering algorithm

```
Input:
C: Sorted strings in descending order by frequency ($c_1 \ldots c_m$)
Output:
G: Set of clusters ($g_1 \ldots g_n$)

1 Select $c_i$, the first string in C, and insert it into the new cluster $g_k$
2 Remove $c_i$ from C
3 For each string $c_j$ in C
    If $\mathbf{LEND}(c_i, c_j) < \alpha_{LEND}(c_i, c_j)$ then
      If $\mathbf{TID}(c_i, c_j) < \alpha_{TID}(c_i, c_j)$ then
        If $\mathbf{LD}(c_i, c_j) < \alpha_{LD}(c_i, c_j)$ then
          Insert $c_j$ into cluster $g_k$
          Remove $c_j$ from C
        Else
          If $\mathbf{IDWP}(c_i, c_j) < \alpha_{IDWP}(c_i, c_j)$ then
            Insert $c_j$ into cluster $g_k$
            Remove $c_j$ from C
          End if
        End if
      End if
    End if
  Next
4 Insert $g_k$ into G
5 If C is not empty then
    Go to step 1
  End if
```

5 Experimental Results and Evaluation

We have used two files for evaluating our method. They contain data from two data-

[3] The leader algorithm is very fast, requiring only one pass through the data, but it has several negative properties: the partition is not invariant under reordering of the cases, the first clusters are always larger than the later ones and the final number of clusters depends on the threshold values.

bases with inconsistency problems. Table 2 gives a description of these two files. The *optimal number of clusters* (ONC) indicates the number of hand-crafted clusters. The three last columns contain the number of single strings (not duplicated) with and without the expansion of abbreviations of both files, and the rate of reduction (on expanding the abbreviations, the number of single strings is reduced, since duplicates are removed). We have done the tests with and without expansion of abbreviations.

Table 2. File descriptions

File	Size (Bytes)	ONC	Strings in file	Strings without expansion	Strings with expansion	Reduction (%)
A	10399	92	234	234	145	38.0
B	1717706	92	37599	1212	1117	7.8

We have developed a coefficient (consistency index) that permits the evaluation of the complexity of a cluster: the greater the value of the coefficient is, the more different the strings that form the cluster are. A null value indicates that the cluster contains only one string. The *consistency index* (*CI*) of a cluster of *n* strings is defined as:

$$CI = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} LD(x_i, x_j)}{\sum_{i=1}^{n} |x_i|}. \tag{5}$$

The *file consistency index* (FCI) of a file that contains *m* clusters is defined as the average of the consistency indexes of all the existing clusters in the file:

$$FCI = \frac{\sum_{i=1}^{m} CI_i}{m}. \tag{6}$$

The FCI of the files A and B are shown in Table 3. It is obvious that the clusters of file B are more complex than those of file A. In both cases, however, the FCI is reduced when expanding the abbreviations, since the discrepancies between the strings of a given cluster tend to diminish.

Table 3. File consistency indexes

File	FCI without expansion	Standard deviation	FCI with expansion	Standard deviation
A	0.311301	0.298053	0.127598	0.269485
B	1.726352	1.267117	1.113732	1.142570

We have evaluated the clusters obtained when our method is applied by using four measures that are obtained by comparing the clusters produced by our method with the optimal clusters:

1. NC: number of clusters. Clusters that have been generated.
2. NCC: number of correct clusters. Clusters that coincide with the optimal ones: they contain the same strings. From this measure, we obtain *Precision*: NCC divided by ONC.
3. NIC: number of incorrect clusters. Clusters that contain an erroneous string. From this measure, we obtain the *Error*: NIC divided by ONC.
4. NES: number of erroneous strings. Strings incorrectly clustered.

As we have already mentioned, the algorithm depends on four parameters (see Section 4.5). Parameters β_{LEND} and β_{TID} serve to reduce the number of LD and IDWP that are calculated. We have done tests on setting the value of both to 0.3, 0.4 and 0.5. For β_{LD}, we have done tests with 0.1, 0.2 and 0.3. For β_{IDWP}, we have set the following values: 0.1, 0.2, 0.3, 0.35, 0.4, 0.45 and 0.5.

In Table 4 we show the results for file A, without (WO) and with (W) expansion of abbreviations, for β_{LEND} and β_{TID} equal to 0.4 and β_{LD} to 0.1. Those values provide the best results. For file B, the best results are obtained by setting β_{LD} equal to 0.3, as we see in Table 5. In the two tables, the best results are in bold letters.

Table 4. File A results, with $\beta_{LEND} = 0.4$, $\beta_{TID} = 0.4$, and $\beta_{LD} = 0.1$

β_{IDWP}	**NC**		**NCC**		**NIC**		**NES**		**Precision (%)**		**Error (%)**	
	WO	W	WO	W	WO	W	WO	W	WO	W	WO	W
0.1	179	130	21	62	0	0	0	0	22.8	67.4	0.0	0.0
0.2	148	107	40	**78**	0	0	0	0	43.5	**84.8**	0.0	0.0
0.3	119	97	63	73	4	3	9	8	68.5	79.3	4.3	3.3
0.35	114	94	**65**	71	7	5	15	11	**70.7**	77.2	7.6	5.4
0.4	106	86	63	64	11	8	22	18	68.5	69.6	12.0	8.7
0.45	101	83	**65**	63	11	11	28	22	**70.7**	68.5	12.0	12.0
0.5	95	74	57	48	16	17	42	32	62.0	52.2	17.4	18.5

Table 5. File B results, with $\beta_{LEND} = 0.4$, $\beta_{TID} = 0.4$, and $\beta_{LD} = 0.3$

β_{IDWP}	**NC**		**NCC**		**NIC**		**NES**		**Precision (%)**		**Error (%)**	
	WO	W	WO	W	WO	W	WO	W	WO	W	WO	W
0.1	123	108	57	66	8	5	51	52	62.0	71.7	8.7	5.4
0.2	123	108	57	66	8	5	51	52	62.0	71.7	8.7	5.4
0.3	122	107	58	66	8	6	51	52	63.0	71.7	8.7	6.5
0.35	117	104	**62**	**67**	8	6	59	75	**67.4**	**72.8**	8.7	6.5
0.4	109	99	**62**	65	11	8	87	94	**67.4**	70.7	12.0	8.7
0.45	107	99	56	60	15	10	110	115	60.9	65.2	16.3	10.9
0.5	97	92	40	41	26	21	186	203	43.5	44.6	28.3	22.8

As observed in the results obtained for the file A, the best precision is obtained when β_{LD} is equal to 0.1, and β_{IDWP} is equal to 0.35 without expansion of abbreviations and to 0.2 with the expansion. For file B, the best results are obtained when β_{LD} is equal to 0.3, and β_{IDWP} is equal to 0.35 in both cases. We should emphasise that for the file A, the greatest precision it is obtained by using different values for β_{IDWP} whether the expansion of abbreviations is used or not, whereas for file B this value is the same in both cases.

In both files, the expansion of abbreviations produces improvements: it increases the precision and reduces the error rate. For files A and B, a maximum precision of 70.7 % and 67.4%, respectively, is obtained without expansion, and 84.8% and 72.8% with the expansion.

The number of distances that are calculated varies enormously for the different values of the parameters and when the expansion of abbreviations takes place. Table 6 presents the number of calculated distances of each type for the tests of Table 4 (file A). In Table 7 we present the values for the tests of Table 5 (file B).

Table 6. Number of calculated distances for file A, with $\beta_{LEND} = 0.4$, $\beta_{TID} = 0.4$, and $\beta_{LD} = 0.1$

β_{IDWP}	**LEND**		**TID**		**LD**		**IDWP**	
	WO	W	WO	W	WO	W	WO	W
0.1	19903	9253	9398	4733	3472	1936	3422	1927
0.2	15372	7721	7134	3883	2547	1587	2505	1580
0.3	11956	6695	5429	3495	1860	1462	1824	1457
0.35	11071	6458	5106	3348	1789	1404	1756	1399
0.4	9978	5802	4592	2989	1616	1264	1587	1260
0.45	9196	5616	4276	2919	1504	1241	1475	1238
0.5	8257	4601	3852	2348	1367	989	1342	986

Table 7. Number of calculated distances for file B, with $\beta_{LEND} = 0.4$, $\beta_{TID} = 0.4$ and $\beta_{LD} = 0.3$

β_{IDWP}	**LEND**		**TID**		**LD**		**IDWP**	
	WO	W	WO	W	WO	W	WO	W
0.1	53308	43285	24700	21563	9620	9122	8532	8114
0.2	53262	43274	24676	21555	9618	9123	8531	8116
0.3	53102	43122	24641	21514	9603	9094	8519	8107
0.35	51416	42751	23877	21288	9397	9024	8340	8054
0.4	48267	39993	22723	20372	9056	8659	8034	7722
0.45	46830	38524	22071	19731	8839	8455	7845	7541
0.5	42915	32622	19941	16429	7987	7069	7061	6224

As we can observe in Tables 6 and 7, the expansion of the abbreviations reduces the number of distances that must be calculated. In the file A, the reduction is greater than in file B; this is because the reduction of the number of strings in file A is greater than in file B (38.0% as opposed to 7.8%), as can be seen in Table 2.

In Tables 6 and 7 we also see that the number of distances that are calculated diminishes when the value of the parameter β_{IDWP} increases. This is because the strings are previously clustered since the threshold is greater (equation 4), which is why the number of strings that must be compared is smaller in the following steps of the algorithm.

6 Conclusions

The inconsistency of values in databases that have not been controlled is a very common problem. In this paper, we have proposed a method that reduces such inconsistencies. This method achieves successful results in all the experiments done, but it does not eliminate the need to review the clusters obtained. In any case, it certainly improves on data quality and data accessing. The method is effective and efficient enough to be used in production environments.

The expansion of abbreviations improves on the results, in most cases, but we have detected some cases in which it actually makes the results worse; we are currently evaluating the expansion of abbreviations. In addition, we have seen that the IDWP permits the clustering of similar strings with permuted word order, at the expense of increased computational cost.

The value that is assigned to the four parameters of the method greatly influences the results. The parameters β_{LEND} and β_{TID} reduce on the number of calculations required. The values of the parameters β_{LD} and β_{IDWP} that achieve the best results are not fixed: they depend on the strings we decide to cluster.

We are currently applying other similarity measures to the algorithm (*Dice*, *Cosine* and *Jaccard Coefficient*) and we use a different clustering algorithm. The preliminary results are promising.

References

1. M. Díaz, J. Pérez, O. Santana. Distancia Dependiente de la Subsecuencia Común más Larga entre Cadenas de Caracteres. In *Anales de las II Jornadas de Ingeniería de Sistemas Informáticos y de Computación*, pages 117-123, Quito (Ecuador), 1993.
2. James C. French, Allison L. Powell, Eric Schulman. Applications of Approximate Word Matching in Information Retrieval. In Forouzan Golshani, Kia Makki, editors, *Proceedings of the Sixth International Conference on Information and Knowledge Management (CIKM 1997)*, pages 9-15, Las Vegas (USA), November 1997.
3. James C. French, Allison L. Powell, Eric Schulman, John L. Pfaltz. Automating the Construction of Authority Files in Digital Libraries: A Case Study. In Carol Peters, Costantino Thanos, editors, *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries (ECDL 1997)*, pages 55-71, Pisa (Italy), September 1997.
4. John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York (USA), 1975.
5. D.S. Hirschberg. Serial Computations of Levenshtein Distances. In A. Apostolico, Z. Galil, editors, *Pattern Matching Algorithms*. Oxford University Press, 1997.
6. V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, **10**:707-710, 1966.
7. Sergio Luján-Mora. An Algorithm for Computing the Invariant Distance from Word Position. Available at `http://www.dlsi.ua.es/~slujan/files/idwp.ps`, June 2000.
8. Edward T. O'Neill, Diane Vizine-Goetz. Quality Control in Online Databases. *Annual Review of Information Science and Technology*, **23**:125-156, 1988.
9. Edward T. O'Neill, Diane Vizine-Goetz. The Impact of Spelling Errors on Databases and Indexes. In *National Online Meeting Proceedings*, pages 313-320, New York (USA), May 1989.