Decision Support System for Comparison of Price Lists

Helena Hamplová, Jindřich Ivánek, Radim Jiroušek,
Tomáš Kroupa, Radim Lněnička, Milan Studený, Jiří Vomlel
Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 4
182 08 Prague 8
Czech Republic
{radim,kroupa,lnenicka,studeny,vomlel}@utia.cas.cz

Abstract

A problem of price lists comparison is introduced. This task can be conceived as an instance of 'database merging' or 'record linkage' techniques whose aim is to integrate possibly identical data coming from different sources and in different formats. We review some of the existing approaches and sketch a first prototype of the solution based on a simple string similarity measure.

1 Introduction

The need for merging data from various sources is evident in many branches of human activity. For example, a computer retailer, who enters a wholesale market with a demand for particular computer components, is offered a lot of price lists coming from different sources in miscellaneous formats yet all of them having one thing in common: they contain many identical computer components which are described by different data. The everyday's task of the retailer is then to identify the same items on the lists in order to select the supplier offering the lowest price. Clearly, without any kind of automated decision-support tool, this task is hardly manageable efficiently and thus many of advantageous purchases are unfortunately left unexploited. Therefore, we analyze the problem just introduced and make first steps towards the solution of automated price lists comparison. We may view this task as a 'database merging' or 'record linkage' as these notions appear frequently in the literature and are referring to many problem areas where similar problems are encountered: for instance, DNA analysis, bibliographic records integration etc. — see [3]. Our experiments and results are based on real data of a computer retailer.

2 Comparing Price Lists

Let us describe the task in more detail. The retailer is given n price lists L_i from suppliers (in practice, there are tens of them) in the spreadsheet format. Each list contains usually tens of thousands computer components. In general, we may assume that not more than only two variables are basically specifying every component:

- *description* of the component,
- *price* of the component.

In many (but not all) price lists a unique identifier of a component is present. Sometimes, this identifier is equivalent across different price lists - it is a so called part number attached to every computer component by its producer, but, not all suppliers use the same identifier[1]. All price lists are partially structured. Different price lists have different structure. In the current approach we transform every price list L_i to a table having exactly the two columns mentioned above: description of the component and price. In case that the part number is present, it must be included in the transformed list as well. In this way, we obtain n data tables L_i and our aim is to obtain an aggregated price list L which attaches to every component prices of the suppliers distributing this component. Clearly, utilization of the part number enables correct matching of the items but in all other cases items must be matched only with respect to their intuitively understood 'similarity'. We illustrate the whole situation on price lists L_1, L_2 with all prices in Czech crowns.

Figure 1: Price list of Supplier 1

L_1	
Description	*Price*
`Cisco 828 G.SHDSL Router 1E, 1G.SHDSL`	13 640
`HP LaserJet 3030 Print/Scan/Copy/Fax, Paralel,USB`	12 529
`HP PL DL360R04 X3.0/1M 1G SA6i iLO`	59 453
...	...

The first two rows of the above tables are identical differing only in description of the components. The router from L_2 is described more briefly than in L_1 and we see that all (sub)strings presented in the description of router in L_2 are presented in its description in L_1. The second row demonstrates another inconvenience connected with string-based matching of computer items: while the HP multifunction machine is characterized in English on the first price list, its description on the second price list is partially in Czech. The third item is not presented in the second price list.

[1] It would solve the problem we are addressing if all suppliers used the same identifier.

Figure 2: Price list of Supplier 2

L_2	
Description	*Price*
`Cisco 828 (G.SHDSL)`	13 740
`LaserJet 3030, tiskárna, kopírka, skener, fax`	12 199
...	...

The desired aggregation of tables L_1 and L_2 can be described by an SQL operation JOIN with a string similarity measure Sim indicating on some scale the extent to which the descriptions of two items appear to be identical:

SELECT * FROM L_1, L_2 WHERE $Sim(L_1.Description, L_2.Description) \leq \varepsilon$

Realize that the string similarity comparison must be performed in a quite complex way as it has to deal with abbreviations, synonyms in Czech and English etc. The previous query should match the first two rows of L_1 and L_2 in an ideal manner so that the resulting price list L would be like the one depicted below. The description of every component is taken from L_1.

Figure 3: Aggregated price list

L		
Description	L_1 *Price*	L_2 *Price*
`Cisco 828 G.SHDSL Router 1E, 1G.SHDSL`	13 640	13 740
`HP LaserJet 3030 Print/Scan/Copy/Fax, Paralel,USB`	12 529	12 199
`HP PL DL360R04 X3.0/1M 1G SA6i iLO`	59 453	-
...	...	...

3 Comparing Strings

Choice of a proper similarity measure plays a key role in efficiency of algorithms for comparison of data items consisting of strings. There have been proposed various string similarity measures. Needless to say, there is no 'best' similarity measure designed to deal with all kinds of applications; rather, a selection of similarity must be governed by the nature and need of the problem itself. For example, comparison of text strings by a spell-checking algorithm must inevitably count for similarities (or distances, equivalently) of individual characters while, on the other hand, an algorithm for comparing data items of medical diagnosis should implement quite the opposite, that is, a higher-level comparison based on some standardized glossary of medical synonyms and abbreviations.

Most traditional methods for calculating string similarity can be thus separated into two groups:

1. character-based techniques,

2. vector-based techniques.

3.1 Character-based techniques

As for the *character-based techniques*, the best-known similarity measure is the so-called *string edit distance (SED)* — see [2] for a detailed exposition. Roughly speaking, SED of two strings S_1 and S_2 is calculated as a minimum number of *edit operations* necessary to transform S_1 into S_2. Edit operations are character deletion, insertion and substitution; all these operations are assigned certain *costs* which can be learned from data. Notice that $\mathrm{SED}(S_1, S_2) \neq \mathrm{SED}(S_2, S_1)$ in general. Calculation of values of SED are performed in quadratic time using dynamic programming. There exists various extensions of SED devised to introduce edit operations on a higher-level of whole inseparable collections of tokens such as synonyms, abbreviations etc. In this way performance of SED can be slightly enhanced in dealing with more complex tasks — see [3].

3.2 Vector-based techniques

While character-based techniques appear to be useful for estimating distance between strings that differ due to typing errors, they become very computationally expensive and less precise for larger strings and terms. If differences between semantically equivalent strings are expressed by multiple words added, deleted, substituted or abbreviated, *vector-based techniques* [4] are more suitable. Every string is encoded as an n-dimensional vector of real numbers whose components are formed by weights of individual *tokens* (groups of characters) presented in the string.

The most popular method for computing the weights is the TF-IDF method [5], where the weight of token x in string S is defined as

$$w(x, S) \quad = \quad \frac{n(x, S)}{n(S)} \log \frac{m}{m(x)}$$

where $n(x, S)$ is the number of occurrences of token x in string S (often, it is 0 or 1), $n(S)$ is the total number of tokens in string S, m is the total number of all strings in the data, and $m(x)$ is the number of strings containing token x. The ratios $\frac{n(x,S)}{n(S)}$ and $\frac{m}{m(x)}$ are called *term frequency* (TF) and *inverse document frequency*, respectively, which explains the name of the TF-IDF method. Clearly, this method requires a vocabulary of all tokens x appearing in the entire data.

Let d denote the number of different tokens in the entire data. Similarity of

the two strings S_1 and S_2 is then computed as a normalized scalar product

$$\begin{aligned} Sim(S_1, S_2) &= \sum_{i=1}^{d} p(x_i, S_1, S_2) \\ &= \sum_{i=1}^{d} \frac{w(x_i, S_1) \cdot w(x_i, S_2)}{\sqrt{\sum_{i=1}^{d} w(x_i, S_1)^2} \cdot \sqrt{\sum_{i=1}^{d} w(x_i, S_2)^2}} \end{aligned}$$

With both vectors being rather highly sparse, this computation can be very efficiently implemented.

In the TF-IDF method the importance of tokens is given by their relative frequencies and thus cannot be learned from training data. If training data are available it may be often beneficial to use a classifier that would use the values of $p(x_i) = p(x_i, S_1, S_2), i = 1, \ldots, d$ as the classifier features. The parameters of the classifier are learned from training data. Based on the parameters and feature values the classifier can decide whether two strings S_1 and S_2 are similar or not, or can order strings according to their similarity to a given string (which is sufficient in most applications).

3.3 Our technique

The technique adopted in our algorithm can be thought of as a mixture of the two basic approaches. We measure the similarity $Sim(S_1, S_2)$ of two strings S_1, S_2 by the total length of substrings of S_1 that are substrings of string S_2. We do not require the substrings of S_1 to be disjoint, which means that parts of substrings of S_1 longer than two are counted several times.

input : strings S_1 and S_2
output: similarity $Sim(S_1, S_2)$

$s = 0$;
for $i = 1$ **to** $length(S_1) - 1$ **do**
  $R = S_1[i]$;
  $j = 2$;
  $fl = TRUE$;
  while $j \leq length(S_1) - i + 1$ **and** $fl = TRUE$ **do**
    $R = R + S_1[i + j - 1]$;
    if R is a substring of S_2 **then** $s = s + j$;
    else $fl = FALSE$;
    $j = j + 1$;
  end
end
$Sim(S_1, S_2) = s$

Algorithm 1: Computation of Similarity

4 Results

We have performed tests with five price lists. The first price list was *referential*, which means that all other price lists were compared with this price list only.

We have selected 277 components from the referential price list. In every other list we found a component that had the highest similarity measured by similarity defined in Section 3.3. All results were evaluated by a human expert.

The results obtained are summarized in Table 1. The column *Items* contains the total number of components in price lists $i = 2, 3, 4, 5$. *Precision* is the ratio $100 \cdot (t/m)$, where t is the number of components classified correctly as identical and $n = 277$ is the total number of components in the referential list. Note that there are large differences between the total number of components in these price lists. Not all components from the referential list can be found in the other lists. Therefore, we further disregarded all components whose similarity has been evaluated to less than 10% of the maximal possible similarity, i.e., similarity of an identical string. The column *TLE (Too Low Evaluation) Precision* describes the precision after disregarding all components with low similarity. Finally, *PN (Part Number) Matching* states the percentage of items classified by our algorithm as identical, that have the same part number.

Table 1: Results – Comparison with Price List 1

Price List	*Items*	*Precision*	*TLE Precision*	*PN Matching*
2	9 670	19,49%	81,25%	9,75%
3	7 941	19,49%	86,84%	13,72%
4	24 076	55,96%	92,05%	33,94%
5	22 182	40,43%	94,74%	23,83%

It should be mentioned that higher values in the column *TLE Precision* are less significant, since, most components obtained TLE. For instance, only 16 components from price list 2 did not obtain TLE.

It is evident that results achieved so far are not satisfactory and there is room for further improvements. For instance, our technique underestimates similarity of two strings if they contain the same substrings that are just permuted. It is illustrated by the following example. The component described as

`HP Color LaserJet 2820mfp, A4 tisk/kop/skener`

in the referential list was incorrectly linked to the item

`HP 3y Nbd Color LaserJet 45/46 xx Hw Supp`,

while the correct matching

`LaserJet Color 2820 All-in-One A4`

has a lower similarity.

Acknowledgments

The authors gratefully acknowledge the support by the grant 1M6798555601 of the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Bilenko M. and Mooney R. J. (2003), Adaptive duplicate detection using learnable string similarity measures, In *Proc. of the Ninth ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining(KDD-2003),* Washington DC, pp. 39–48.

[2] Ristad E. S. and Yianilos P. N. (1997), Learning string edit distance. *Research report CS-TR-532-96,* Department of Computer Science, Princeton University, Princeton, NJ.

[3] Zhu Joanne J. and Ungar L. H. (2000), String edit analysis for merging databases, KDD Workshop.

[4] Salton G., Wong A., and Yang C. S. (1975), A vector space model for automatic indexing. Communications of the ACM, 18(11):613–620.

[5] Salton G. and Buckley C. (1988). Term weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513–523.