http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 10 (1): 195-200, 2011
ISSN 1812-5638 / DOI: 10.3923/itj.2011.195.200

Semi Automatic Method for String Matching

[1]R. Manivannan and [2]S.K. Srivatsa
[1]Dr. M.G.R. University, Chennai, Tamilnadu, India
[2]St. Joseph Engineering College, Chennai, Tamilnadu, India

Abstract: String matching and string edit distance are fundamental concepts in structural pattern recognition, record linkage and schema matching problems. In this paper, a hybrid string matching approach is introduced. Given two strings, x and y, the similarity of the strings is obtained by using a combination of affine gap method, TFIDF and a custom domain specific dictionary. We'll show formal properties of the hybrid string matching approach, describe a procedure for its computation and give practical examples.

Key words: String matching, approximate string matching, semi automatic string matching, hybrid string matching, string edit distance, semantic matching

INTRODUCTION

Researchers have investigated the problem of identifying duplicate objects under several monikers, including record linkage, merge-purge, duplicate detection, database hardening, identity uncertainty, co-reference resolution and name matching. Such diversity reflects research in several areas: statistics, databases, digital libraries, natural language processing and data mining. Our research explores approaches to the name-matching problem that improves accuracy. Particularly, we employ methods that adapt to a specific domain by combining multiple string similarity methods (Rajesh and Srivatsa, 2009) that capture different notions of similarity. Because an estimate of similarity between strings can vary significantly depending on the domain and specific field under consideration, traditional similarity measures may fail to estimate string similarity correctly. At the token level, certain words can be informative when comparing two strings for equivalence, while others are ignorable. For example, ignoring the substring Street may be acceptable when comparing addresses, but not when comparing names of people (e.g., Nick Street) or newspapers (e.g., Wall Street Journal). At the character level, certain characters can be consistently replaced by others or omitted when syntactic variations are due to systematic typographical or OCR errors. Thus, accurate similarity computations (Mansi and Alnihoud, 2010) require adapting string similarity metrics for each field of the database with respect to the particular data domain.

LITERATURE SURVEY

We examine a few effective and widely used metrics for measuring similarity.

Edit distance: An important class of such metrics is edit distances. Here, the distance between strings s and t is the cost of the best sequence of edit operations that converts *s* to *t*. Edit Operations are listed as follows Eq. 1:

- Copy character from string1 over to string2 (cost 0)
- Delete a character in string1 (cost 1)
- Insert a character in string2 (cost 1)
- Substitute one character for another (cost 1)

$$D(s,t,i,j)=\min\begin{cases}D(s,t,i-1,j-1) & \text{if } s_i=t_j, \text{ and you copy } s_i \text{ to } t_j\\ D(s,t,i-1,j-1)+1 & \text{if you substitue } t_j \text{ for } s_i\\ D(s,t,i,j-1) & \text{if you insert the letter } t_j\\ D(s,t,i-1,j) & \text{if you delete the letter } s_i\end{cases} \quad (1)$$

The simple unit-cost metric just described is usually called Levenshtein distance (Hernandez and Stolfo, 1995; Joachims, 1999). There are many extensions to the Levenshtein distance function typically these alter the d(i,j) function, but further extensions can be made for instance, the Needleman-Wunch distance for which Levenshtein is equivalent if the gap distance is 1. The Levenshtein distance is calculated below (Table 1) for the term sam chapman and sam john chapman, the final distance is given by the bottom right cell, i.e., 5. This

Corresponding Author: R. Manivannan, C. Abdul Hakeem College of Engineering and Technology, Hakeem Nagar, Melvisharam, 632 509, Vellore District, Tamilnadu, India

Table 1: Computing levenstein distance

	S	a	m		C	h	a	p	m	a	n
S	0	1	2	3	4	5	6	7	8	9	10
A	1	0	1	2	3	4	5	6	7	8	9
M	2	1	0	1	2	3	4	5	6	7	8
	3	2	1	0	1	2	3	4	5	6	7
J	4	3	2	1	1	2	3	4	5	6	7
O	5	4	3	2	2	2	3	4	5	6	7
H	6	5	4	3	3	2	3	4	5	6	7
N	7	6	5	4	4	3	3	4	5	6	6
	8	7	6	5	5	4	4	4	5	6	7
C	9	8	7	6	5	5	5	5	5	6	7
H	10	9	8	7	6	5	6	6	6	6	7
A	11	10	9	8	7	6	5	6	7	6	7
P	12	11	10	9	8	7	6	5	6	7	7
M	13	12	11	10	9	8	7	6	5	6	7
A	14	13	12	11	10	9	8	7	6	5	6
n	15	14	13	12	11	10	9	8	7	6	5

score indicates that only 5 edit cost operations are required to match the strings (for example, insertion of the john characters, although a number of other routes can be traversed instead).

We can evaluate this recursive definition efficiently using dynamic programming techniques. Specifically, for a fixed s and t, we can store the D(s, t, i, j) values in a matrix that is filled in a particular order. The total computational effort for D(s, t, |s|, |t|), the edit distance between s and t, is thus approximately O(|s||t|). Edit distance metrics are widely used not only for text processing but also for biological sequence alignment1and many variations are possible.

Affine gap method: Needleman and Wunsch (1970) extended the model to allow contiguous sequences of mismatched characters, or gaps, in the alignment of two strings and described a general dynamic programming method for computing edit distance. Most commonly the gap penalty is calculated using the affine model: cost(g) = s + e × l, where s is the cost of opening a gap, e is the cost of extending a gap and l is the length of a gap in the alignment of two strings, assuming that all characters have a unit cost. Usually e is set to a value lower than s, thus decreasing the penalty for contiguous mismatched substrings. Since differences between duplicate records often arise because of abbreviations or whole-word insertions and deletions, this model produces a more sensitive similarity estimate than Levenshtein distance.

String distance with affine gaps between two strings of lengths l1 and l2 can be computed using a dynamic programming algorithm that constructs three matrices in O(l1l2) computational time. The following recurrences are used to construct the matrices, where c(xi, yj) denotes the cost of the edit operation that aligns ith character of string x to jth character of string y, while c(xi, ^{2}) and c(^{2}, yj) are insertion and deletion costs for the respective characters. Matrix M represents a minimum-cost string alignment that ends with matched characters, while matrices I and D represent alignments that end with a gap in one of the two strings (Eq. 2-5):

$$M_{i,j} = \min \begin{cases} M_{i-1,j-1} + c(x_i, y_j) \\ I_{i-1,j-1} + c(x_i, y_j) \\ D_{i-1,j-1} + c(x_i, y_j) \end{cases} \quad (2)$$

$$D_{i,j} = \min \begin{cases} M_{i-1,j} + s + c(x_i, \varepsilon) \\ D_{i-1,j} + e + c(x_i, \varepsilon) \end{cases} \quad (3)$$

$$I_{i,j} = \min \begin{cases} M_{i,j-1} + s + c(\varepsilon, y_j) \\ I_{i,j-1} + e + c(\varepsilon, y_j) \end{cases} \quad (4)$$

$$S(x^T, y^V) = \min\left(I_{T,V}, D_{T,V}, M_{T,V}\right) \quad (5)$$

Needleman-Wunch distance or sellers algorithm: This approach is known by various names, Needleman-Wunch, Needleman-Wunch-Sellers, Sellers and the Improving Sellers algorithm. This is similar to the basic edit distance metric, Levenshtein distance; this adds an variable cost adjustment to the cost of a gap, i.e., insert/deletion, in the distance metric. So, the Levenshtein distance can simply be seen as the Needleman-Wunch distance with G = 1.

- D(i-1,j-1) + d(si,tj) //subst/copy
- D(i,j) = min D(i-1,j)+G //insert
- D(i,j-1)+G //delete

where, G = gap cost and d(c,d) is again an arbitrary distance function on characters (e.g., related to typographic frequencies, amino acid substitutability, etc.). The Needleman-Wunch distance is calculated below for the term sam chapman and sam john chapman, with the gap cost G set to 2 (Table 2). The final distance is given by the bottom right cell, i.e., 10. This score indicates that only 10 edit cost operations are required to match the strings (for example, insertion of the john characters, although a number of other routes can be traversed instead).

Smith-Waterman distance: Again similar to the to Levenshtein distance, this was developed to identify optimal alignments between related DNA and protein sequences. This has two main adjustable parameters a function for an alphabet mapping to cost values for substitutions and costs, (the d function). This also allows

Table 2: Computing Needleman-Wunch distance

	s	a	m		C	H	a	p	m	a	N
s	0	2	4	6	8	10	12	14	16	18	20
a	2	0	2	4	6	8	10	12	14	16	18
m	4	2	0	2	4	6	8	10	12	14	16
	6	4	2	0	2	4	6	8	10	12	14
j	8	6	4	2	1	3	5	7	9	11	13
o	10	8	6	4	3	2	4	6	8	10	12
h	12	10	8	6	5	3	3	5	7	9	11
n	14	12	10	8	7	5	4	4	6	8	9
	16	14	12	10	9	7	6	5	5	7	9
c	18	16	14	12	10	9	8	7	6	6	8
h	20	18	16	14	12	10	10	9	8	7	7
a	22	20	18	16	14	12	10	11	10	8	8
p	24	22	20	18	16	14	12	10	12	10	9
m	26	24	22	20	18	16	14	12	10	12	11
a	28	26	24	22	20	18	16	14	12	10	12
n	30	28	26	24	22	20	18	16	14	12	10

Table 3: Computing Smith-Waterman distance

	a	A	a	a		m	N	o	p		Z	Z	z	z
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0.5	0	0	0	1	0.5	0	0	0
m	0	0	0	0	0.5	2	1.5	1	0.5	0.5	0	0	0	0
n	0	0	0	0	0	1.5	3	2.5	2	1.5	1	0.5	0	0
o	0	0	0	0	0	1	2.5	4	3.5	3	2.5	2	1.5	1
p	0	0	0	0	0	0.5	2	3.5	5	4.5	4	3.5	3	2.5
	0	0	0	0	1	0.5	1.5	3	4.5	6	5.5	5	4.5	4
y	0	0	0	0	0.5	0	1	2.5	4	5.5	5	4.5	4	3.5
y	0	0	0	0	0	0	0.5	2	3.5	5	4.5	4	3.5	3
y	0	0	0	0	0	0	0	1.5	3	4.5	4	3.5	3	2.5
y	0	0	0	0	0	0	0	1	2.5	4	3.5	3	2.5	2

costs to be attributed to a gap G (insert or delete). The final similarity distance is computed from the maximum value in the path where,

- 0 //start over
- D(i-1,j-1) -d(si,tj) //subst/copy
- D(i,j) = max D(i-1,j)-G //insert
- D(i,j-1)-G //delete

Distance is maximum over all i,j in table of D(i,j)

- G = 1 //example value for gap
- d(c,c) = -2 //context dependent substitution cost
- d(c,d) = +1 //context dependent substitution cost

The Smith-Waterman distance is calculated below for the term aaaa mnop zzzz and bbbb mnop yyyy, with the gap cost G set to 0.5 (Table 3) and where d(c,c) = -2, d(b,c) = 1. The final distance is given by the highest valued cell, i.e., 6. This score indicates that the longest approximately matching string terminates in the cell with the highest value so the sequence mnop matches in both strings.

An extension of this technique was made by Gotoh in 1982 which also allows affine gaps to be taken into consideration. This extension was incorporated into the better known Monge Elkan distance.

	W	I	1	L	I	A	M
W	1	0	0	0	0	0	0
I	0	1	0	0	1	0	0
L	0	0	1	1	0	0	0
L	0	0	1	1	0	0	0
L	0	0	1	1	0	0	0
A	0	0	0	0	0	1	0
I	0	1	0	0	1	0	0
M	0	0	0	0	0	0	1

Fig. 1: Computation of *Jaro Metric*

The Jaro metric and variants: Another effective similarity metric is the Jaro metric (Cook and Holder, 1994; Durbin *et al*., 1998; Baeza-Yates and Ribeiro-Neto, 1999) which is based on the number and order of common characters between two strings. 4-6 Given strings s = a1 aK and t = b1, bL, define a character ai in s to be in common with t iff there is a bj = ai in t such that i -≤j≤ i + H, where H = min(|s|,|t|)/2. Let be the characters in s that are common with t (in the same order they appear in s) and let be analogous. Then define a transposition for s', t' to be a position i such that ai = bi. Let Ts', t' be one-half the number of transpositions for s' and t'. The Jaro metric for s and t is Eq. 6:

$$\text{Jaro}(s,t) = \frac{1}{3}\cdot\left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|}\right) \quad (6)$$

To better understand the intuition behind this metric, consider the matrix M in (Fig. 1), which compares the strings s = WILLLAIM and t = WILLIAM. The boxed entries are the main diagonal and M(i, j) = 1 if and only if the ith character of s equals the jth character of t. The Jaro metric is based on the number of characters in s that are in common with t. In terms of the matrix M of the figure, the ith character of s is in common with t if Mi,j = 1 for some entry (i, j) that is "sufficiently close" to M's main diagonal, where sufficiently close means that |i ¯ j| < min(|s|, |t|)/2 (shown in the matrix in bold). William Winkler proposed a variant of the Jaro metric that also uses the length P of the longest common prefix of s and t. Letting P' = max(P, 4), we define Jaro-Winkler(s, t) = Jaro(s, t) + (P'/10) . (1 ¯ Jaro(s, t)). This emphasizes matches in the first few characters. The Jaro and Jaro-Winkler metrics

seem to be intended primarily for short strings (for example, personal first or last names).

Jaccard index: The Jaccard index, also known as the Jaccard similarity coefficient (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets Eq. 7. The Jaccard coefficient measures similarity between sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (7)$$

The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union Eq. 8:

$$J_\delta(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (8)$$

Tanimoto coefficient (extended Jaccard coefficient): Cosine similarity is a measure of similarity between two vectors of n dimensions by finding the angle between them, often used to compare documents in text mining. Given two vectors of attributes, A and B, the cosine similarity, θ, is represented using a dot product and magnitude as in Eq. 9:

$$\theta = \arccos\frac{A.B}{\|A\|\|B\|} \quad (9)$$

For text matching, the attribute vectors A and B are usually the tf-idf vectors of the documents. Since the angle, θ, is in the range of $[0,\pi]$, the resulting similarity will yield the value of ð as meaning exactly opposite, $\pi/2$ meaning independent, 0 meaning exactly the same, with in-between values indicating intermediate similarities or dissimilarities.

This cosine similarity metric may be extended such that it yields the Jaccard coefficient in the case of binary attributes. This is the Tanimoto coefficient, T(A,B), represented as in Eq. 10:

$$T(A,B) = \frac{A.B}{\|A\|^2 + \|B\|^2 - A.B} \quad (10)$$

TF/IDF: In many situations, word order is unimportant. For instance, the strings Ray Mooney and Mooney, Ray are likely to be duplicates, even if they aren't close in edit distance. In such cases, we might convert the strings s and t to token multi-sets (where each token is a word) and consider similarity metrics on these multi-sets. One simple and often effective token-based metric is Jaccard similarity. Between the word sets S and T, Jaccard similarity is simply $(|S \cap T|)/(|S \cup T|)$. We can define term frequency-inverse document frequency (TF-IDF) (Robertson, 2004) or cosine similarity-which the information retrieval community widely uses (Baeza-Yates and Ribeiro-Neto, 1999) - as in Eq. 11:

$$TF-IDF(S,T) = \sum_{W \in S \cap T} V(w,S).V(w,T) \quad (11)$$

where, Tfw, S is the frequency of word w in S, IDFw is the inverse of the fraction of names in the corpus that contain w, V'(w, S) = log(TFw,S + 1) . log(IDFw) and for name matching, you might collect the statistics used to compute IDFw from the complete corpus of names to be matched. TF-IDF distance is attractive in that it weights agreement on rare terms more heavily than agreement on more common terms. This means that Ray Mooney and Wray Mooney will be considered more similar than, say, Ray Mooney and Ray Charles.

Based on the discussions above we can come to a conclusion that no single method independently can provide optimum results. In this paper we propose a hybrid method which combines the affine gap method, a domain specific dictionary and TFIDF in a novel way to get the benefits of all the methods.

HYBRID STRING MATCHING PROCESS

We first tokenize the strings to be matched based on common delimiters like space, comma, semicolon, underscore etc and if there is a change in the case i.e., from lower case to upper case between successive letters of the string. If the strings are made up of just a single word then we compute the similarity between the strings using dictionary first. If they are synonyms then a match value of 1 is assigned to the pair. If they are not synonyms then the similarity is computed using the affine gap method and the match value between the pairs is then obtained. As in the literature, the match value between any pair of strings fall between 0 tol. Any string pair whose match value is greater than a pre specified threshold is taken as equivalent or matching.

If the strings are made up of multiple words then each word from the two strings are matched with one another using the above described method. And the matching word pairs between the two strings are

obtained. Now the TFIDF value between the two strings is computed using this set of matched word pairs identified between the two strings. The strings are considered to be matching or equivalent if the TFIDF value of the two strings exceeds a predefined threshold. Thus instead of taking the set of words present in both the strings to compute the TFIDF we are taking the set of similar words as identified by the affine gap method or dictionary to compute the TFIDF.

The affine gap method ensures that words with spelling mistakes, acronyms, prefix and suffixes match properly. The dictionary ensures that synonyms are matched properly. And TFIDF ensures that words found rarely are given higher weightage in the matching process than words that are very common in the corpus. For example let us take the matching of the following set of words:

{shipTo, billTo} and {deliverTo, invoiceTo}

The above words when tokenized results in the following set of tokens,

{ship,To, bill, To} and {deliver, To, invoice, To}

Here the token To is found in all the words. Hence when matching is done between the words, To is given very low weightage compared to the other words. Hence just having the token To in common doesn't make the words similar. By combining the previously stated approaches, we were able to increase the matching efficiency considerably.

EXPERIMENTAL EVALUATION

This experiments were conducted on six datasets. Restaurant is a database of 864 restaurant names and addresses containing 112 duplicates obtained by integrating records from Fodor's and Zagat's guidebooks. Cora is a collection of 1295 distinct citations to 122 Computer Science research papers from the Cora Computer Science research paper search engine. The citations were segmented into multiple fields such as author, title, venue, etc. by an information extraction system, resulting in some crossover noise between the fields. Reasoning, Face, Reinforcement and Constraint are single-field datasets containing unsegmented citations to computer science papers in corresponding areas from the Citeseer scientific literature digital library. 2 Reasoning contains 514 citation records that represent 196 unique papers, Face contains 349 citations to 242 papers, Reinforcement contains 406 citations to 148 papers and Constraint contains 295 citations to 199 papers. Tables 4-6 contain sample duplicate records from the Restaurant, Cora and Reasoning datasets.

During each trial, duplicate detection was performed, at each iteration, the pair of records with the highest similarity was labeled a duplicate and the transitive closure of groups of duplicates was updated. Precision, recall and F-measure defined over pairs of duplicates were computed after each iteration, where precision is the fraction of identified duplicate pairs that are correct, recall is the fraction of actual duplicate pairs that were identified and F-measure is the harmonic mean of precision and recall Eq. 12-14:

Table 4: Sample duplicate records from the Cora database

Authors	Title	Venue	Address	Year	Pages
Yoav Fr`eund, H. Sebastian Seung, Eli Shamir and Naftali Tishby	Information, prediction and query by committee	Advances in neural Information Processing system	San Mateo, CA	1993	Pages 483-490
Freund, Y., Seung, H. S., Shamir, E. and Tishby, N.	Information, prediction and query by committee	Advances in neural information Processing systems	San Mateo, CA.	–	(pp. 483-490)

Table 5: Sample duplicate records from the Restaurant database

Name	Address	City	Phone	Cuisine
Fenix	8358 sunset blvd. west	Hollywood	213/848-6677	American
Fenix at the argyle	8358 sunset blvd.	w. Hollywood	213-848-6677	French (new)

Table 6: Sample duplicate records from the Reasoning database

L.P. Kaelbling. An architecture for intelligent reactive systems. In Reasoning About Actions and Plans: Proceedings of the 1986 Workshop. Morgan Kaufmann, 1986
Kaelbling, L.P., 1987. An architecture for intelligent reactive systems. In M.P. Georgeff and A. L. Lansky, eds., Reasoning about Actions and Plans, Morgan Kaufmann, Los Altos, CA, 395 410

Table 7: F-measures from our experiments

Distance Metric	Restaurant name	Restaurant address	Reasoning	Face	Reinforcement
Edit distance	0.290	0.686	0.927	0.952	0.893
Vector - space	0.365	0.380	0.897	0.922	0.903
Learned - Vector space	0.433	0.532	0.924	0.875	0.808
Hybrid matcher	0.354	0.712	0.938	0.966	0.907

$$\text{Precision} = \frac{\#\text{ of correctly identified duplicate pairs}}{\#\text{of identified duplicate pairs}} \quad (12)$$

$$\text{Recall} = \frac{\#\text{ of correctly identified duplicate pairs}}{\#\text{of true duplicate pairs}} \quad (13)$$

$$\text{F}-\text{measure} = \frac{2\times \text{Precision}\times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (14)$$

As more pairs with lower similarity are labeled as duplicates, recall increases, while precision begins to decrease because the number of non-duplicate pairs erroneously labeled as duplicates increases. Precision was interpolated at 20 standard recall levels following the traditional procedure in information retrieval. The result of our experimentation is shown in Table 7.

CONCLUSION

Duplicate detection is an important problem in data cleaning, record linkage and schema matching problems. We have proposed a way of combining two matching strategies along with a domain specific dictionary to increase their matching efficiency. In future we can work a way for identifying suitable learning strategies which can be used to learn matching strings in a particular domain. And also we could try other combination of matchers to see their matching efficiency in various domains.

REFERENCES

Baeza-Yates, R. and B. Ribeiro-Neto, 1999. Modern Information Retrieval. ACM Press, New York.

Cook, D.J. and L.B. Holder, 1994. Substructure discovery using minimum description length and background knowledge. J. Artifi. Intell. Res., 1: 231-255.

Durbin, R., S. Eddy, A. Krogh and G. Mitchison, 1998. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press. Cambridge.

Hernandez, M.A. and S.J. Stolfo, 1995. The merge/purge problem for large databases. Proceedings of the ACM SIGMOD/PODS International Conference on Management of Data, (ASICMD'95), San Jose, CA., pp: 127-138.

Joachims, T., 1999. Making Large-Scale SVM Learning Practical. In: Advances in Kernel Methods-Support Vector Learning, Scholkopf, B., C.J.C. Burges and A.J. Smola (Eds.). MIT Press, Cambridge, MA., ISBN-10: 0-262-19416-3, pp: 169-184.

Mansi, R.H. and J.Q. Alnihoud, 2010. An efficient ASCII-based algorithm for single pattern matching. Inform. Technol. J., 9: 453-459.

Needleman, S.B. and C.D. Wunsch, 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. J. Mol. Biol., 48: 443-453.

Rajesh, A. and S.K. Srivatsa, 2009. A hybrid path matching algorithm for XML schemas. Inform. Technol. J., 8: 378-382.

Robertson, S., 2004. Understanding inverse document frequency: On theoretical arguments for IDF. J. Documentation, 60: 503-520.