Construction of Adaptive Scoring Matrix Using Similar Strings as a Training Set

Sung-Hwan Kim, Chang-Seok Ock, Jong Kyu Seo, and Hwan-Gue Cho

***Abstract*—Sequence alignment is one of the methods widely used to determine string similarity. It computes similarity by aligning the component characters a string, and summing the similarity scores of pairs of matched characters. Using an appropriate character similarity measure is important when performing the alignment-based similarity calculation, since the string similarity is highly depending on the character similarity. In this paper, we focus on the character similarity learning process for string classification, particularly for when one set of strings that belong to the same class is given. Our method uses the matching frequency to calculate the character similarity. The performance of the method is also demonstrated by experimental evaluation.**

***Index Terms*—Character similarity, string classification, sequence alignment, scoring matrix.**

I. Introduction

In information processing, it has been a challenge dealing with errors possibly included in given data [1], [2]. Particularly with strings, errors usually arise by typos and misspellings [3]. In character or speech recognition, misrecognized results occur due to the system accuracy [4]. Moreover, swear filters [5], spam filters, and plagiarism detection [6], [7] should consider even intentional errors generated to avoid such systems.

Measuring the similarity is important for dealing with such errors in data. We can calculate the similarity between a given input and strings in a pre-existing database. We can make the system robust to errors by replacing the input with the most similar example from the database.

In this sense, sequence alignment can be used for measuring the similarity of strings [8]. It was originally developed to find similar biological sequences, and it gives a score and its corresponding arrangements of two strings. Moreover, it can be directly applied to general strings, since biological sequences belong to one specific type of strings. Thus, we can use the sequence alignment to measure the string similarity, and to determine whether given strings are identical to each other by checking the score returned from the sequence alignment.

Sequence alignment is a general scheme for measuring string similarity, and is instantiated by the scoring scheme. A scoring scheme is an objective function, and is usually defined by a matrix whose rows and columns indicate each character, with elements of matched scores of corresponding characters. This character similarity of the scoring scheme indicates which characters are very similar and interchangeable, and is directly connected to the string similarity. Therefore, it is important to choose a scoring scheme that has appropriate character similarities, in order to properly determine the string similarity.

However, manual setting character similarities involves substantial effort as the size of the alphabet over which strings are defined increases. Moreover, manual setting may lead to an insufficient scoring scheme that does not sufficiently reflect the actual character similarity.

In this paper, we present a learning method for character similarity, which takes a set of similar strings and gives a matrix that has an appropriate character similarity scheme. Our method counts the relative frequency of matched characters to distinguish interchangeable character pairs and assign high scores to them. We also show the performance of our proposed method empirically.

II. Sequence Alignment

Sequence alignment is an arrangement of strings used to maximize the value of the objective function, which is usually defined by the sum of pairs of the corresponding characters derived from the arrangement. Fig. 1-(a) shows an example of sequence alignment where the objective function is defined by the character similarity in Fig. 1-(b).

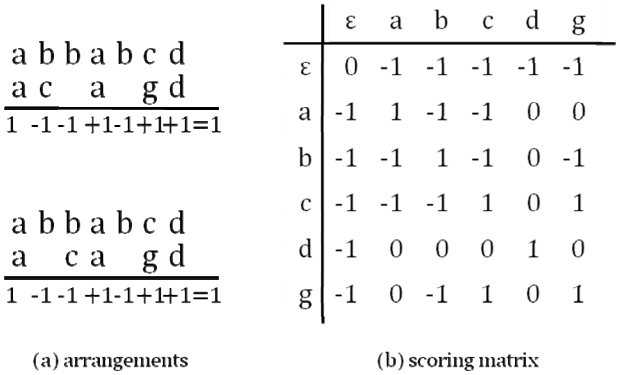


Fig. 1. An example of sequence alignment.

The sequence alignment of two strings may not be unique. As depicted in Fig. 1, there can be more than one arrangement that maximizes the value derived from the objective function. Nevertheless, the similarity score is unique, so we can use it as a similarity measure without any problems.

Sequence alignment can be represented in a recursive form,

Manuscript received October 15, 2012; revised December 29, 2012. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2012-0005518). Corresponding should be addressed to Hwan-Gue Cho (hgcho@pusan.ac.kr).

Sung-Hwan Kim, Chang-Seok Ock and Hwan-Gue Cho are with Dept. of Computer Engineering, Pusan National University, Busan, South Korea (e-mail: sunghwan@pusan.ac.kr; csock@pusan.ac.kr; hgcho@pusan.ac.kr).

and it can be derived by solving the following dynamic programming equation:

$$\begin{aligned} M_{0,0} &= 0 \\ M_{i,0} &= M_{i-1,0} + i \cdot \sigma(x_i, \varepsilon) \\ M_{0,j} &= M_{0,j-1} + j \cdot \sigma(\varepsilon, y_j) \\ M_{i,j} &= \max \begin{Bmatrix} M_{i-1,j-1} + \sigma(x_i, y_i), \\ M_{i-1,j} + \sigma(x_i, \varepsilon), \\ M_{i,j-1} + + \sigma(\varepsilon, y_j) \end{Bmatrix} \end{aligned} \quad (1)$$

where ε is the gap character and σ is the character similarity function.

Fig. 2 shows the dynamic programming matrix for the sequence alignment of two strings, "aaabb" and "aab," with σ(α,β)=1 if α=β, and -1 otherwise . The similarity score is the value in the last column of the last row. The corresponding arrangements can be obtained by back-tracking the paths that earn the scores making the final value.

	ε	a	a	a	b	b
ε	0	-1	-2	-3	-4	-5
a	-1	1	0	-1	-2	-3
a	-2	0	2	1	0	-1
b	-3	-1	1	1	2	1

Fig. 2. Dynamic programming matrix for sequence alignment.

III. Related Work

As mentioned in Section II, the similarity score obtained from the sequence alignment is highly dependent on the given character similarity, which is used for the unit score of the objective function. Therefore, defining a good character similarity scheme is an important task for the similarity calculation using the sequence alignment to derive an appropriate similarity relationship between strings.

In bioinformatics, several scoring schemes have been addressed to identify similar biological objects such as DNA and protein sequences [8]. For example, PAM is one of the representative examples for the character similarity schemes. It takes a parameter k as the number of evolution units, which represents the evolutionary distance between sequences. For example, from a PAM matrix with k=1, the character similarity is calculated as follows:

$$\sigma(\alpha,\beta) = 10 \cdot \log \frac{f(\alpha,\beta)}{100\, p(\alpha) p(\beta) \sum_i \sum_{j(\neq i)} f(\alpha,\beta)} \quad (2)$$

where $p(\alpha)$ is the probability that character α appears, and $f(\alpha,\beta)$ is the frequency that characters α and β match each other.

However, this scoring scheme is specialized for biological sequences, and uses input data obtained from the ungapped local alignment, which finds similar substrings without gaps, from relatively long sequences. When dealing with words in natural languages, gaps are so essential that these methods cannot be directly applied.

IV. Matching Frequency

As mentioned above, our method computes the character similarity from the relative frequency of matched character pairs. Higher scores are assigned to characters that match very frequently in the sequence alignment. The matching frequency is obtained by aligning all pairs of strings in the given database for learning. Thus, in this section, we discuss how to count the matching frequency from an alignment of two strings.

Backtracking on the dynamic programming matrix is a straightforward solution to count the frequency of character matches. Fig. 3 shows this method. From the last column of the last row, we can trace the path to the first column of the first row and update the count table for character matches for each visited node on the path.

	ε	a	c	c	b	d
ε	0	-1	-2	-3	-4	-5
a	-1	1	←0	←1	-2	-3
b	-2	0	0	0	0	-1
g	-3	-1	-1	-1	-1	-1

char	pair	freq
ε	c	2
a	a	1
b	b	1
d	g	1

Fig. 3. Counting character matches with backtracking the matrix.

However, when we use the backtracking method, we have to consider the case that more than one arrangement exists. In this case, we have to enumerate all possible paths and count every pair of matched characters that appear on each path with an equal weight. Moreover, if there are consecutive gaps and mismatches, as described in Fig. 4, a number of possible arrangements are involved. In such cases, backtracking becomes time-consuming and costly.

	ε	a	a	a	b	b
ε	0	←1	←-2	-3	-4	-5
a	-1	1	←0	←1	-2	-3
a	-2	0	2	←1	←0	-1
b	-3	-1	1	1	←2	←1

Fig. 4. An example of a case that involves a number of possible arrangements as the result of sequence alignment.

To accumulate the matching frequency on the alignment matrix without backtracking, which is possibly a very substantial task, we use a combinatorial approach. First, we construct a graph, the vertices of which correspond to each element of the alignment matrix, with each edge representing a match, mismatch, or gap. Two vertices are connected only if there is a match, mismatch, or gap. The start vertex is defined by the one associated with the first column of the first row, and the end vertex is the one associated with the last column of the last row of the matrix. Fig. 5 shows a graph representation of the sequence alignment matrix used in Fig. 4. The graph is direct, and the orientations of edges correspond to how the sequence alignment proceeds. Additionally, each edge indicates a (mis)match or gap

according to its direction.

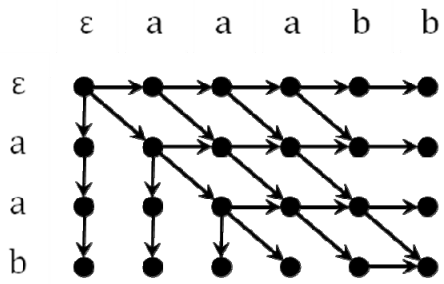


Fig. 5. Graph representation of sequence alignment matrix in Fig 4.

Then, for each vertex, we can compute the number of paths from the start vertex to the end vertex that pass the node. The ratio of this number to the total number of paths represents the portion of the characters that the corresponding vertices occupy.

To compute the number of paths that pass a specific vertex, we separately calculate the number of paths from the start vertex to the target vertex (referred to as forward paths), and those from the target vertex to the end vertex (referred to as backward paths). The product of these numbers is the same as the number of paths that we finally want to obtain.

Before presenting a more detailed explanation of the calculation process, we define the relation $a \rightarrow b$ of two vertices if there is an edge from vertex a to vertex b.

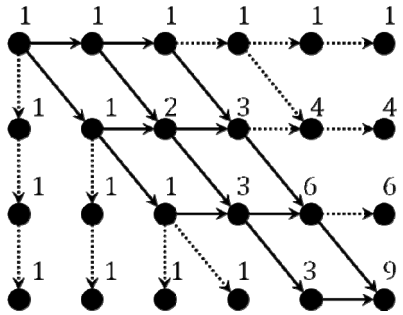


Fig. 6. Calculation of the number of forward paths.

Fig. 6 shows the process to calculate the number of forward paths from the start vertex to the target vertex. Starting with assigning 1 to the start vertex, for each vertex v, the values of all u such that $u \rightarrow v$ are accumulated. More formally, the number of forward paths for vertex v is defined as follows:

$$F(s) = 1 \tag{3}$$

$$F(v) = \sum_{u \rightarrow v} F(u) \tag{4}$$

where s is the start vertex.

Similarly, the number of backward paths from the target vertex to the end vertex is computed as described in Fig. 7. Starting by assigning 1 to the end vertex, the values of all the connected vertices are accumulated, but in this time, we sum the values of all u such that $v \rightarrow u$ to obtain the value for vertex

$$B(e) = 1 \tag{5}$$

$$B(v) = \sum_{v \rightarrow u} B(u) \tag{6}$$

where e is the end vertex.

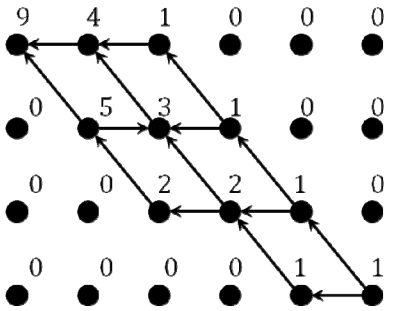


Fig. 7. Calculation of the number of backward paths.

Note that the number of forward paths of the end vertex is the same as that of the backward paths of the start vertex. This is also exactly the same as the number of possible optimal arrangements for the sequence alignment. We denote this total number of paths from the start vertex to the end vertex by T:

$$T = F(e) = B(s) \tag{7}$$

To accumulate the matching count of characters α and β, we first have to compute the number of paths that pass a specific edge (u,v) that indicates (mis)match of the characters α and β. We denote this by $N(u,v)$, which is directly calculated from the product of the number of forward paths for u and that of backward paths for v:

$$N(u,v) = F(u) \cdot B(v) \tag{8}$$

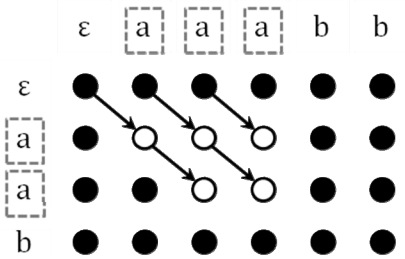


Fig. 8. Edges that indicate character match A(a,a).

Let $A(\alpha, \beta)$ be the set of all edges that represent the (mis)match of characters α and β (see Fig. 8). To compute the matching frequency $m(\alpha, \beta)$ for characters α and β, we take the sum of $N(v)$ for all $(u,v) \in A(\alpha, \beta)$ as $m(\alpha, \beta)$:

$$m(\alpha,\beta) = \sum_{(u,v) \in A(\alpha,\beta)} N(u,v) \tag{9}$$

Now we have the probability $p(\alpha, \beta)$ that characters α and β match as the ratio of the matching frequency $m(\alpha, \beta)$ to the total number of paths T:

$$p(\alpha,\beta) = \frac{m(\alpha,\beta)}{T} \tag{10}$$

In other words, $p(\alpha, \beta)$ is the frequency of matches between characters α and β for a given pair of strings. In the following section, we use this to compute the final character similarity of α and β.

V. Similarity Calculation

In this section, we derive the character similarity using the results of the previous section. As mentioned above, the sequence alignment finds the arrangements that maximize the score of the objective function. This maximum score is the similarity score, and it is usually defined by the sum of the character similarity $\sigma(\alpha, \beta)$. If we rewrite $\sigma(\alpha, \beta)$ as the logarithm of another function $\tau(\alpha, \beta)$, finding the optimal alignment can be interpreted as the maximization of the product of $\tau(\alpha, \beta)$.

Under the assumption that each edit operation is performed independently, including insertion, deletion, and substitution of a character, the product of $\tau(\alpha, \beta)$ can give the odds for the whole sequences if $\tau(\alpha, \beta)$ gives the odds that α and β match in strings in the same class with that in any randomly selected strings. Therefore, it is reasonable to use the character similarity $\sigma(\alpha, \beta)$ as the logarithm of the odds of matching events.

Let $p_S(\alpha, \beta)$ be the probability that characters α and β match when we align two strings in set S. This can be obtained by parameterization of $p(\alpha, \beta)$ such that it takes two strings x and y. We denote the parameterized $p(\alpha, \beta)$ by $p(\alpha, \beta; x, y)$, where x and y are given strings. It represents the probability that characters α and β match when strings x and y are aligned. We can then define $p_S(\alpha, \beta)$ more formally as follows:

$$p_S(\alpha, \beta) = \frac{1}{|S|^2} \sum_{x,y \in S} p(\alpha, \beta; x, y) \tag{11}$$

Finally, we have the similarity of characters α and β as the logarithm of the ratio of the probability $p_S(\alpha, \beta)$ that the characters match between strings in the set S of similar strings to the random chance $\varphi(\alpha, \beta)$:

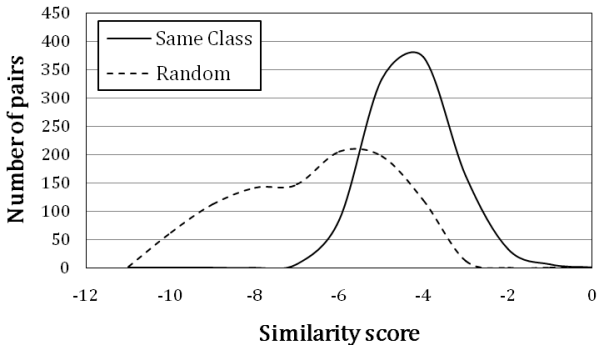
$$\sigma(\alpha, \beta) = \log \frac{p_S(\alpha, \beta) + C}{\varphi(\alpha, \beta) + C} \tag{12}$$

The constant term C is a small positive number, in order to avoid both dividing by zero and taking the logarithm of zero. For the efficiency of computation in the sequence alignment, the character similarity can be scaled and subjected to the floor function to make it an integer.

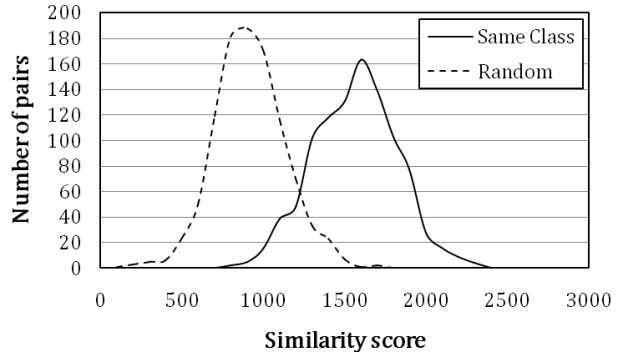
The random chance $\varphi(\alpha, \beta)$ that characters α and β match can be computed by a combinatorial method or Monte Carlo simulation. In the experiments of the following section, we use the latter for an intuitive and simple implementation, which is equivalent to $p_R(\alpha, \beta)$ where R is the random strings.

VI. Experimental Evaluation

The dataset used in the experiments is synthetic string sets generated by the following steps: (i) a string is chosen as the seed in a random manner and inserted into the dataset; (ii) a random symmetric matrix is generated, each of whose elements indicate the substitution probability between its corresponding character pair; and (iii) choosing a string from the dataset is repeated in a random manner and a random edit operation is performed according to the probability matrix.



(a) Levenshtein distance matrix



(b) Scoring matrix derived from the proposed method

Fig. 9. Distribution of the maximum similarity scores with respect to two different scoring matrices and different sets.

We generate two sets of 1K strings from the same seed. One set is used for training, the other for evaluation. We also use the randomly generated sets without any skewed probability setting to obtain $\varphi(\alpha, \beta)$ and evaluate the performance.

We determine whether a given string is included in the pre-determined class by checking the maximum similarity score against the set of known strings. Fig. 9 depicts the distribution of the maximum similarity scores with respect to the scoring matrix and string sets. Dashed lines indicate the score distribution for the random set, and solid lines represent that for the set of strings in the class.

The Levenshtein distance can be represented as a matrix in which all elements are -1, but those on the diagonal are 0. Sequence alignment using this matrix returns the Levenshtein distance with a negative sign. Fig. 9-(a) shows the distribution of the Levenshtein distance. We can easily observe that the range is too narrow and the overlapped area is substantial. In contrast, the distribution derived from our proposed scheme has a wider variance and relatively small overlapped portion. This means that our method would perform well for string classification.

We also evaluate the performance of our proposed method for string classification in terms of the sensitivity and specificity, which are defined as follows:

$$Sensitivity = \frac{TP}{FN + TP} \tag{14}$$

$$Specificity = \frac{TN}{FP + TN} \tag{15}$$

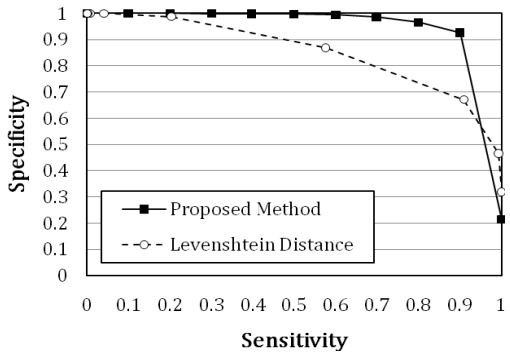


Fig. 10. The performance comparison between the proposed method and the levenshtein distance for classification.

Fig. 10 shows the performance of our method comparing against the Levenshtein distance. It is observed that our method outperforms the Levenshtein distance for string classification. When the sensitivity is 90%, the specificity of our method is above 0.92, while that of the Levenshtein distance is only 0.671.

VII. Conclusions

We have presented a new construction method of the scoring scheme for sequence alignment. Our method counts the relative frequency of matched characters relative to random chance, and computes the similarity score as its logarithm. We have demonstrated the performance of our method by comparing against the Levenstein distance in terms of sensitivity and specificity. The notable contributions of our study are summarized as follows:

- We have presented several combinatorial methods to calculate the number of paths on the sequence alignment matrix. It is useful to compute the matching frequencies of characters.
- Based on this, we have also presented a probabilistic model to construct the scoring matrix from a given set of strings that are included in the same class.
- Our method outperforms the Levenshtein distance by about 0.3 in terms of the specificity when the sensitivity is 0.9.

We have also observed several aspects to be further improved upon as follows:

- More extensive experiments on real datasets and other synthetic datasets with various configurations are necessary.
- We used a set of random strings to obtain $\varphi(\alpha, \beta)$, the random chance that characters match. We will discover various cases where additional characteristics are imbued on the string set.
- We expect that our method can be extended to much more general cases with complex character sets. If possible, it can be applied to much broader fields, such as information retrieval and data compression.
- We used only the global sequence alignment that arranges the whole strings. Local sequence alignment that finds similar substrings can be considered as an extension of our method.

References

[1] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no.1, pp. 31-88, 2001.
[2] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin, "Searching in metric spaces," *ACM Computing Surveys*, vol.33, no.3, 2001.
[3] H. Duan and B.-J. Hsu, "Online spelling correction for query completion," in *Proc. of WWW*, 2011, pp.117-126.
[4] J. Droppo and A. Acero, "Context dependent phonetic string edit distance for automatic speech recognition," in *Proc. of IEEE ICASSP*, 2010, pp. 4358-4361.
[5] T.-J. Yoon, S.-Y. Park, and H.-G. Cho, "A smart filtering system for newly coined profanities by using approximate string alignment," in *Proc. of IEEE CIT*, 2010, pp. 643-650.
[6] J. P. Kumar and P. Govindarajulu, "Duplicate and near duplicate documents detection," *Journal of Discrete Algorithms*, vol. 13, pp. 32-46, 2012.
[7] S.-Y. Park, S.-Y. Kim, S.-H. Kim, and H.-G. Cho, "A global dictionary based approach to fast similar text search in document repository," in *Proc. IEEE CIT*, 2011, pp. 526-532.
[8] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology,* PWS Publishing Company, 1997.

Sung-Hwan Kim is a M.S student in Pusan National University. He received the B.S. degree from Pusan National University. His research interests are information retrieval and string processing.

Chang-Seok Ock is a M.S. student in Pusan National University. He received the B.S. degree from Pusan National University. His research interests are information retrieval, human-computer interaction, and computer graphics.

Jong Kyu Seo is a M.S student in Pusan National University. He received the B.S degree from Pusan National University. His research interests are information retrieval and string processing.

Hwan-Gue Cho is a Professor in Pusan National University. He received the B.S. degree from Seoul National University, Korea, and the M.S and Ph.D. degrees from Korea Advanced Institute of Science and Technology, Korea. His research interests are computer algorithms and bioinformatics.