String distances and uniformities

David W. Pearson and Jean-Christophe Janodet

University of Saint-Etienne, 18 r. Pr. Lauras, F-42000 St-Etienne
{david.pearson,janodet}@univ-st-etienne.fr

Abstract. The Levenstein or edit distance was developed as a metric for calculating distances between character strings. We are looking at weighting the different edit operations (insertion, deletion, substitution) to obtain different types of classifications of sets of strings. As a more general and less constrained approach we introduce topological notions and in particular uniformities.
Keywords: edit distance, classification, topology, uniformities.

1 Introduction

The Levenstein (or edit) distance was introduced in the paper [1]. It has been used in various applications concerning textual data. One particular application is linked with linguistics and natural language processing where we want to find words that are in some way "close" to a given word or sets of words. This is the initial motivation for our research. We decided to use the Levenstein metric as a starting point for our work, with the understanding that it would not satisfy all of our needs. In particular, we believe that trying to place a metric structure on a natural language may be too strong a condition. This meant that we needed to look for a structure that is less rigid and rigourous than a metric. This paper is the result of our preliminary investigations.

Metric spaces are nice to work with because the idea of distance between objects is well defined, straight forward and in most cases easy to compute. However, sometimes a metric is too strong a condition to require for certain problems. At the other end of the scale we have topological spaces, where the only thing that you can say about points in the space is that they are neighbours of each other. So, a topological space can be too general for certain problems.

A uniform space (or a uniformity) lies somewhere between the two [2–5]. In a uniformity we have a notion of closeness rather than distance and we can make statements like point a is as close to point b as point c is to point d. We believe that uniformities present potentially interesting properties for text processing and, in particular, we would like to define a uniformity for strings for classification purposes.

For our initial investigations we have used the *edit distance* to define our uniformities. Also called the *Levenshtein distance*, this distance measures the minimum number of deletion, insertion and substitution operations needed to transform one string into another [1, 6, 7]. This distance, and its variants where each operation has a weight, has been used in many fields including Computational Biology [8, 9], Language Modelling [10, 11], Pattern Recognition [12, 13] and Machine Learning [14, 15].

We think of a classification problem of strings as defining a topology (and uniformity) for the strings, *i.e.*, strings in the same class are neighbours. We must add at this

point that we refer to classification in a somewhat unrigourous fashion in that the resulting classes may overlap due to the uniform structure. A true classification would result in disjoint classes. Therefore we consider that the weights of the edit distance parameterize the uniformity. When we change the weights the uniformity may or may not change. We are interested in finding the critical parameter values where the uniformity changes.

This paper is composed of three main sections. In the following section we present the relevant theoretical background on uniformities. Then, in the next section we show how we can define uniformities for sets of strings. An example is developed in the last section before finally concluding.

2 Covering Uniformities

There are at least two ways of defining a uniformity: entourage uniformities and covering uniformities. It can be shown that they provide equivalent structures and that the choice of entourage or covering is governed by the application. The entourage approach is very popular nowadays [2, 3], but we have found the covering approach to be better adapted to our needs [4, 5].

Let X be any fixed space. A *covering* for X is a collection $\mathcal{C}$ of sets $C_i \subseteq X$ such that $\bigcup_i C_i = X$, for $C_i \in \mathcal{C}$. Given two coverings $\mathcal{U}$ and $\mathcal{V}$, $\mathcal{U}$ is said to *refine* $\mathcal{V}$, denoted $\mathcal{U} < \mathcal{V}$, if for all $U_i \in \mathcal{U}$, there exists $V_j \in \mathcal{V}$ such that $U_i \subseteq V_j$.

For a covering $\mathcal{C}$ and a subset $A \subseteq X$, the star of A is defined as follows:

$$*(A, \mathcal{C}) = \bigcup \{C_i \in \mathcal{C} : C_i \cap A \neq \emptyset\}.$$

Given two coverings $\mathcal{U}$ and $\mathcal{V}$, we say that $\mathcal{U}$ *star refines* $\mathcal{V}$, denoted $\mathcal{U} <^* \mathcal{V}$, if for all $U \in \mathcal{U}$, there exists $V \in \mathcal{V}$ such that $*(U, \mathcal{U}) \subseteq V$. In this case, the sets in $\mathcal{V}$ can be thought of as twice as big as those of $\mathcal{U}$ [5].

We now introduce the following definition for a *covering uniformity*. A family μ of coverings is called a *uniformity* if it satisfies the following conditions:

1. if $\mathcal{U}, \mathcal{V} \in \mu$, then there exists $\mathcal{W} \in \mu$ such that $\mathcal{W} < \mathcal{U}$ and $\mathcal{W} < \mathcal{V}$,
2. if $\mathcal{U} \in \mu$ and $\mathcal{U} < \mathcal{V}$ then $\mathcal{V} \in \mu$ and
3. every element of μ has a star refinement in μ.

Some texts refer to this definition as a preuniformity or a non-separating uniformity, we shall simply use the term uniformity. A separation condition can be added and some authors refer to that as a uniformity, but other authors refer to it as a Hausdorff uniformity. The separation condition is not necessary in our case.

The notion of a *normal sequence of coverings* in a uniformity is simply a sequence $\mathcal{U}_n$ such that $\cdots \mathcal{U}_{n+1} <^* \mathcal{U}_n <^* \mathcal{U}_{n-1} \cdots$.

If $\mathcal{U} \in \mu$ and $y \in X$ then a point $x \in X$ is said to be $\mathcal{U}$*-close* to y, denoted $|y - x| < \mathcal{U}$, if there exists $U \in \mathcal{U}$ such that $\{x, y\} \subseteq U$.

Finally, let d be a distance over X. For any $x \in X$, we define the ϵ-sphere around x as $S(x, \epsilon) = \{y \in X : d(x, y) \leq \epsilon\}$. Clearly, if $S(x, \epsilon) \subseteq \mathcal{U} \in \mu$, then every $y \in S(x, \epsilon)$ is $\mathcal{U}$-close to x.

3 Strings and Uniformities

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. For the sake of clarity, we shall use $\Sigma = \{a, b\}$ as a fixed alphabet throughout the rest of this paper. A *string* $w = x_1 \dots x_n$ is any finite sequence of letters. We write Σ^* for the set of all strings over Σ. Let $|w|$ denote the length of w and λ the empty string.

Following [1], we consider three sorts of edit operations:

- a pair $(x : y) \in \Sigma \times \Sigma$ is called *a substitution of letter x by letter y*,
- a pair $(x : \lambda)$ with $x \in \Sigma$ is called *a deletion of letter x*, and
- a pair $(\lambda : y)$ with $y \in \Sigma$ is called *an insertion of letter y*.

Moreover, we assume that a matrix C assignes a weight to every operation. ***E.g.***,

C	λ	a	b
λ	0	1	2
a	1	0	1.5
b	2	1.5	0

The *edit distance* between two strings w_1 and w_2, denoted $d(w_1, w_2)$, is the minimum weight of every sequence of substitutions, deletions and insertions that allows one to transform w_1 into w_2. More formally, d is recursively defined as follows:

$$d(w_1, w_2) = \min \begin{cases} 0 & \text{if } w_1 = w_2 = \lambda \\ C(x : \lambda) + d(w_1', w_2) & \text{if } w_1 = x w_1' \\ C(\lambda : y) + d(w_1, w_2') & \text{if } w_2 = y w_2' \\ C(x : y) + d(w_1', w_2') & \text{if } w_1 = x w_1', w_2 = y w_2' \end{cases}$$

It is well-known [7] that if C defines a metric over $(\Sigma \cup \{\lambda\})$, then $d(w_1, w_2)$ can be efficiently computed in time $\mathcal{O}(|w_1| \cdot |w_2|)$ by means of dynamic programming [6]. Assuming that C defines a metric means that the matrix C has a null diagonal, is positive, symmetric and for all $x, y, z \in \Sigma \cup \{\lambda\}$,

$$C(x : y) \leq C(x : z) + C(z : y). \tag{1}$$

In such a case, the edit distance is determined by only three weights: $C(a : b)$, $C(a : \lambda)$ and $C(b : \lambda)$. We group these values together into a vector $p = \begin{bmatrix} C(a : b) \\ C(a : \lambda) \\ C(b : \lambda) \end{bmatrix}$. Thus, using any fixed p that satifies Eq.(1), we can compute the edit distance between any two strings.

Our main interest is in string classification and so we assume that we have a set of strings, W of cardinality m, and some idea of which strings are together in classes. Let us consider for any $w \in W$, the ϵ-sphere around w: $S(w, \epsilon) = \{w' \in W : d(w, w') \leq \epsilon\}$. Using the edit distance and the ϵ-spheres, we can now calculate uniformities for sets of strings.

To begin with, we compute the distances between all the strings in W using the edit distance with some fixed value of the parameter vector p. If we list all the strings in W

in some order horizontally and vertically then the result is simply a symmetric $m \times m$ matrix D with zeros along the diagonal. We take all the elements above the diagonal of this matrix and list them in lexicographical order. Thus if

$$D = \begin{bmatrix} 0 & d_{12} & d_{13} & \cdots & d_{1m} \\ d_{21} & 0 & d_{23} & \cdots & d_{2m} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & \cdots & 0 \end{bmatrix}$$

where $d_{ij} = d(w_i, w_j)$, then we define the vector

$$x = \begin{bmatrix} d_{12} \\ d_{13} \\ \vdots \\ d_{ij} \\ \vdots \\ d_{m-1m} \end{bmatrix}$$

We want to adjust the parameter vector p to give the required classification. As the vector x above is dependent on p, we indicate this by $x(p)$ and thus consider x to be a mapping $x : \mathbb{R}^3 \to \mathbb{R}^n$ where $n = \frac{m^2-m}{2}$ is the number of elements of D above the diagonal. Due to the condition on the values of the three parameters, p is restricted to certain areas of $\mathbb{R}^3$. The standard edit distance uses $p = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$ and so we will define the admissible values of p based on this point. Define the following vectors $p_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $p_1 = \begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix}$, $p_2 = \begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix}$ and $p_3 = \begin{bmatrix} 1 \\ 1 \\ 0.5 \end{bmatrix}$. Then, for our purposes, we can say that the admissible values of p can be defined by the simplex $P = t_0 p_0 + t_1 p_1 + t_2 p_2 + t_3 p_3$ where $t_i \geq 0$ and $\sum_{i=0}^{3} t_i = 1$.

Once a value of p has been chosen, the uniformity is defined by varying x and ϵ in $S(x, \epsilon)$. Clearly different uniformities can be defined for the same value of p, depending on the chosen values of x and ϵ. Another point to mention is that certain choices of x and ϵ will not lead to correct uniformities simply because of the star refinement property that is required. These points are best illustrated by an example, which we present in the following section.

4 Example

Let the set of strings be the following $W = \{aaab, abab, bba, baba, bbaab\}$. Applying the classical edit distance we have the following table:

W	$aaab$	$abab$	bba	$baba$	$bbaab$
$aaab$	0	1	3	3	2
$abab$	1	0	2	2	2
bba	3	2	0	1	2
$baba$	3	2	1	0	3
$bbaab$	2	2	2	3	0

We can find 3 coverings from this. First of all $S(aaab, 1)$, $S(bba, 1)$ and $S(bbaab, 1)$ supply us with

$$\mathcal{U}_2 = \{aaab, abab\}, \{bba, baba\}, \{bbaab\},$$

then $S(aaab, 2)$ and $S(bba, 2)$ give us

$$\mathcal{U}_1 = \{aaab, abab, bbaab\}, \{abab, bba, baba, bbaab\},$$

then finally $S(aaab, 3)$ gives us

$$\mathcal{U}_0 = \{aaab, abab, bba, baba, bbaab\}.$$

It can be verified that $\mathcal{U}_2 <^* \mathcal{U}_1 <^* \mathcal{U}_0$. So, with this choice of p all the strings are $\mathcal{U}_0$-close, $\{aaab, abab, bbaab\}$ and $\{abab, bba, baba, bbaab\}$ are $\mathcal{U}_1$-close and finally $\{aaab, abab\}$, $\{bba, baba\}$ and $\{bbaab\}$ are $\mathcal{U}_2$-close.

To see how the uniformity changes when p changes we carry out the same exercise but with $p = p_1$ as described above. With this value for p the distances are the following

W	$aaab$	$abab$	bba	$baba$	$bbaab$
$aaab$	0	0.5	2	1.5	1.5
$abab$	0.5	0	1.5	2	1.5
bba	2	1.5	0	1	2
$baba$	1.5	2	1	0	2
$bbaab$	1.5	1.5	2	2	0

Using these distances we can now define the following covers:

$$\begin{aligned}
\mathcal{V}_3 &= S(aaab, 0.5), S(bba, 0.5), S(baba, 0.5), S(bbaab, 0.5) \\
&= \{aaab, abab\}, \{bba\}, \{baba\}, \{bbaab\} \\
\mathcal{V}_2 &= S(aaab, 1), S(bba, 1), S(bbaab, 1) \\
&= \{aaab, abab\}, \{bba, baba\}, \{bbaab\} \\
\mathcal{V}_1 &= S(bbaab, 1.5), S(bba, 1.5) \\
&= \{aaab, abab, bbaab\}, \{abab, bba, baba\} \\
\mathcal{V}_0 &= S(aaab, 2) \\
&= \{aaab, abab, bba, baba, bbaab\}
\end{aligned}$$

and it can be verified that $\mathcal{V}_3 <^* \mathcal{V}_2 <^* \mathcal{V}_1 <^* \mathcal{V}_0$. Here we see that $\mathcal{U}_0 = \mathcal{V}_0$, but the other sets in the different levels are not the same and so the $\mathcal{U}$-uniformity and the $\mathcal{V}$-uniformity are not the same.

Changing p once again and using $p = p_2$ as defined above, leaving the details out we have the following uniformity

$$\begin{aligned}
\mathcal{W}_3 &= \{aaab\}, \{abab\}, \{bba, baba\}, \{bbaab\} \\
\mathcal{W}_2 &= \{aaab, abab\}, \{bba, baba, bbaab\} \\
\mathcal{W}_1 &= \{aaab, abab, bbaab\}, \{abab, bba, baba, bbaab\} \\
\mathcal{W}_0 &= \{aaab, abab, bba, baba, bbaab\}
\end{aligned}$$

with $\mathcal{W}_3 <^* \mathcal{W}_2 <^* \mathcal{W}_1 <^* \mathcal{W}_0$.

These three uniformities are clearly different. To see how the changes occur we traced out the vector x at various points between p_1 and p_2 by setting $p = (1-t)p_1 + tp_2$ with t ranging from 0 to 1 in increments of 0.1. The individual components of the vector x can be seen in figure 1. The uniformity actually changes between the values $t = 0.5$ and $t = 0.6$, *i.e.*, for values of p between $p = \begin{bmatrix} 0.75 \\ 0.75 \\ 1 \end{bmatrix}$ and $p = \begin{bmatrix} 0.8 \\ 0.7 \\ 1 \end{bmatrix}$.

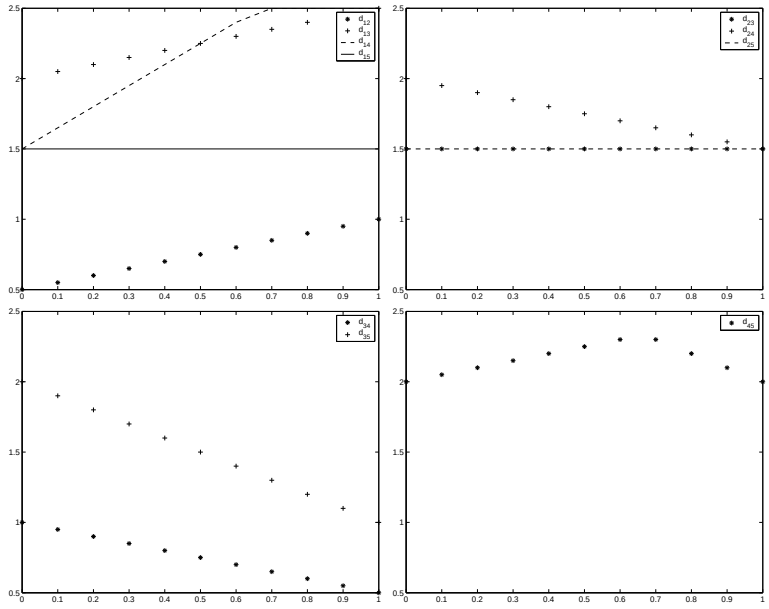

Fig. 1. The distances for the strings $aaab$ (top left), $abab$ (top right), bba (bottom left) and $baba$ (bottom right).

We notice that with the $\mathcal{V}$-uniformity the two strings $aaab$ and $abab$ are always together but bba and $baba$ are separated in the $\mathcal{V}_3$-uniformity. Whilst for the $\mathcal{W}$-uniformity bba and $baba$ remain together but $aaab$ and $abab$ get separated in $\mathcal{W}_3$.

5 Conclusion

We have introduced an approach to string classification based on uniformities. We believe that this approach has potential because it falls between one which is too general based on a topology and one which is too rigorous based on a metric.

We are fully aware of the fact that we need a metric to actually calculate the ϵ-spheres and thus the uniformity, but we wanted to test our ideas in the first instance and so we used the Levenstein distance to advance more quickly. We believe that the results obtained so far are promising and so we are continuing along these lines. Our work is now concentrating on how to define a uniformity and carry out calculations without the need of a metric.

References

1. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4) (1965) 845–848
2. Kelley, J.L.: General Topology. D. Van Nostrand (1955)
3. James, I.M.: Topologies and Uniformities. Springer-Verlag (1999)
4. Howes, N.R.: Modern Analysis and Topology. Springer-Verlag (1995)
5. Willard, S.: General Topology. Adison-Wesley (1970)
6. Wagner, R., Fisher, M.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–178
7. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press (2007)
8. Gusfield, D.: Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press (1997)
9. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge University Press (1998)
10. Amengual, J.C., Sanchis, A., Vidal, E., Benedí, J.M.: Language simplification through error-correcting and grammatical inference techniques. Machine Learning Journal **44**(1-2) (2001) 143–159
11. Amengual, J.C., Dupont, P.: Smoothing probabilistic automata: An error-correcting approach. In: Proc. of the 5th International Colloquium in Grammatical Inference (ICGI'00), LNAI 1891 (2000) 51–64
12. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys **33**(1) (2001) 31–88
13. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys **33**(3) (2001) 273–321
14. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning balls of strings from edit corrections. Journal of Machine Learning Research **9** (2008) 1823–1852
15. Delhay, A., Miclet, L.: Analogical equations in sequences: Definition and resolution. In: Proc. 7th International Colloquium in Grammatical Inference (ICGI'04), LNAI 3264 (2004) 127–138