N-gram similarity and distance

Grzegorz Kondrak

Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2E8, Canada
kondrak@cs.ualberta.ca
http://www.cs.ualberta.ca/~kondrak

Abstract. In many applications, it is necessary to algorithmically quantify the similarity exhibited by two strings composed of symbols from a finite alphabet. Numerous string similarity measures have been proposed. Particularly well-known measures are based are edit distance and the length of the longest common subsequence. We develop a notion of n-gram similarity and distance. We show that edit distance and the length of the longest common subsequence are special cases of n-gram distance and similarity, respectively. We provide formal, recursive definitions of n-gram similarity and distance, together with efficient algorithms for computing them. We formulate a family of word similarity measures based on n-grams, and report the results of experiments that suggest that the new measures outperform their unigram equivalents.

1 Introduction

In many applications, it is necessary to algorithmically quantify the similarity exhibited by two strings composed of symbols from a finite alphabet. For example, for the task of automatic identification of confusable drug names, it is helpful to recognize that the similarity between *Toradol* and *Tegretol* is greater than the similarity between *Toradol* and *Inderal*. The problem of measuring string similarity occurs in a variety of fields, including bioinformatics, speech recognition, information retrieval, machine translation, lexicography, and dialectology [9]. A related issue of computing the similarity of texts as strings of words has also been studied.

Numerous string similarity measures have been proposed. A particularly widely-used method is *edit distance* (EDIT), also known as Levenshtein distance, which is defined as the minimum number of elementary edit operations needed to transform one string into another. Another, closely related approach relies on finding the length of the *longest common subsequence* (LCS) of the two strings. Other similarity measures are based on the number of shared n-grams, i.e., substrings of length n.

In this paper, we develop a notion of n-gram similarity and distance.[1] We show that edit distance and the length of the LCS are special cases of n-gram

[1] This is a different concept from the q-gram similarity/distance [12], which is simply the number of common/distinct q-grams (n-grams) between two strings.

distance and similarity, respectively. We provide formal, recursive definitions of n-gram similarity and distance, and efficient algorithms for computing them. We formulate a family of word similarity measures based on n-grams, which are intended to combine the advantages of the unigram and the n-gram measures. We evaluate the new measures on three different word-comparison tasks: the identification of genetic cognates, translational cognates, and confusable drug names. The results of our experiments suggest that the new n-gram measures outperform their unigram equivalents.

We begin with n-gram similarity because we consider it to be conceptually simpler than n-gram distance. The latter notion is then defined by modifying the formulation of the former.

2 Unigram similarity

In this section, we discuss the notion of the length of the LCS, which we view as *unigram similarity*, in the context of its applicability as a string similarity measure. After defining the longest common subsequence problem in a standard way, we provide an alternative but equivalent formulation of the length of the LCS. The recursive definition not only elucidates the relationship between the LCS length and edit distance, but also generalizes naturally to n-gram similarity and distance.

2.1 Standard definition

The standard formulation of the LCS problem is as follows [3]. Given a sequence $X = x_1 \ldots x_k$, another sequence $Z = z_1 \ldots z_m$ is a subsequence of X if there exist a strictly increasing sequence $i_1, \ldots, i_m$ of indices of X such that for all $j = 1, \ldots, m$, we have $x_{i_j} = z_j$. For example, *tar* is a subsequence of *contrary*. Given two sequences X and Y, we say that a sequence Z is a common subsequence of X and Y if Z is a subsequence of both X and Y. In the LCS problem, we are given two sequences and wish to find their maximum-length common subsequence. For example, the LCS of *natural* and *contrary* is *ntra*. The LCS problem can be solved efficiently using dynamic programming.

For the purpose of measuring string similarity, which is our focus here, only the *length* of the LCS is important; the actual longest common subsequence is irrelevant. The length of the LCS as a function of two strings is an interesting function in itself [2].

2.2 Recursive definition

We propose the following formal, recursive definition of the function $\mathbf{s}(X, Y)$, which is equivalent to the length of the LCS. Let $X = x_1 \ldots x_k$ and $Y = y_1 \ldots y_l$ be strings of length k and l, respectively, composed of symbols of a finite alphabet. In order to simplify the formulas, we introduce the following notational shorthand, borrowed from Smyth [10]. Let $\Gamma_{i,j} = (x_1 \ldots x_i, y_1 \ldots y_j)$ be a pair of prefixes of X and Y, and $\Gamma^*_{i,j} = (x_{i+1} \dots x_k, y_{j+1} \dots y_l)$ a pair of suffixes of X and Y.

For strings of length one or less, we define s directly:

$$\mathbf{s}(x, \epsilon) = 0, \qquad \mathbf{s}(\epsilon, y) = 0, \qquad \mathbf{s}(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise} \end{cases}$$

where ϵ denotes an empty string, x and y denote single symbols.

For longer strings, we define s recursively:

$$\mathbf{s}(X, Y) = \mathbf{s}(\Gamma_{k,l}) = \max_{i,j}(\mathbf{s}(\Gamma_{i,j}) + \mathbf{s}(\Gamma^*_{i,j}))$$

The values of i and j in the above formula are constrained by the requirement that both $\Gamma_{i,j}$ and $\Gamma^*_{i,j}$ are non-empty. More specifically, the admissible values of i and j are given by the following set of pairs:

$$D(k, l) = \{0, \dots, k\} \times \{0, \dots, l\} - \{(0, 0), (k, l)\}$$

For example, $D(2, 1) = \{(0, 1), (1, 0), (1, 1), (2, 0)\}$.

It is straightforward to show by induction that $\mathbf{s}(X, Y)$ is always equal to the length of the longest common subsequence of strings X and Y.

2.3 Rationale

Our recursive definition exploits the semi-compositionality of the LCS. Clearly, LCS is not compositional in the usual sense, because the LCS of concatenated strings is not necessarily equal to the sum of their respective LCS. For example, $\|LCS(ab, a)\| = 1$ and $\|LCS(c, bc)\| = 1$, but $\|LCS(abc, abc)\| = 3$. What is certain is that the LCS of concatenated strings is always at least as long as the concatenation of their respective LCS:

$$\mathbf{s}(X_1, Y_1) + \mathbf{s}(X_2, Y_2) \leq \mathbf{s}(X_1 + X_2, Y_1 + Y_2)$$

Loosely speaking, $\mathbf{s}(X, Y)$ is superadditive, rather than compositional. It is always possible to compose the LCS of two strings by concatenating the LCS of their substrings, provided that the decomposition of the strings into substrings preserves all identity matches in the original LCS. Such a decomposition can always be found (cf. Figure 1, left pair).

2.4 A reduced set of decompositions

Any decomposition of a pair of strings can be unambiguously defined by a pair of indices. The set D contains all distinct decompositions of a pair of strings. The number of distinct decompositions of a pair of strings is $(k+1)*(l+1)-2$.

The set of decomposition can be reduced without affecting the values of the function $\mathbf{s}$. Let D' be the following set of decompositions:

$$D'(k, l) = \{k-1, k\} \times \{l-1, l\} - \{(0, 0), (k, l)\}$$

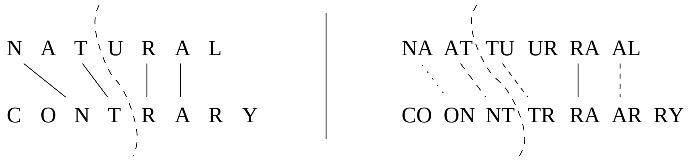

Fig. 1. A decompositions of a unigram alignment that preserves all identity matches (left), and a decomposition of a bigram alignment with various levels of bigram similarity (right).

For example, $D'(2,1) = \{(1,0),(1,1),(2,0)\}$. D' never contains more than three decompositions. By substituting D by D' in the recursive definition given in section 2.2, we obtain an alternative, equivalent formulation of $\mathbf{s}$:

$$\mathbf{s}(X,Y) = \mathbf{s}(\Gamma_{k,l}) = \max(\mathbf{s}(\Gamma_{k-1,l}), \mathbf{s}(\Gamma_{k,l-1}), \mathbf{s}(\Gamma_{k-1,l-1}) + \mathbf{s}(x_k, y_l))$$

The alternative formulation directly yields the well-known efficient dynamic-programming algorithm for computing the length of the LCS [14].

2.5 Beyond unigram similarity

The main weakness of the LCS length as a measure of string similarity is its insensitivity to context. The problem is illustrated in Figure 2. The two word pairs on the left demonstrate that neighbouring identity matches are a stronger indication of similarity than identity matches that are far apart. The two word pairs on the right show that parallel identity matches are a stronger indication of similarity than identity matches that are separated by unmatched symbols.

A family of similarity measures that do take context into account is based on *Dice coefficient* [1]. The measures are defined as the ratio of the number of n-grams that are shared by two strings and the total number of n-grams in both strings:

$$\frac{2 \times |n\text{-}grams(X) \cap n\text{-}grams(Y)|}{|n\text{-}grams(X)| + |n\text{-}grams(Y)|}$$

where *n-grams(X)* is a multi-set of letter n-grams in X. Dice coefficient with bigrams (DICE) is a particularly popular word similarity measure. For example, DICE(*Zantac*, *Contac*) $= (2 \cdot 3)/(5+5) = 0.6$ because three of the bigrams are shared.

Although more sensitive to context that LCS length, DICE has its own problems. First, because of its "low resolution", it often fails to detect any similarity between strings that are very much alike; for example, the pair *Verelan/Virilon* have no n-grams in common. Second, DICE can return the maximum similarity value of 1 for strings that are non-identical; for example, both *Xanex* and *Nexan* are composed of the same set of bigrams: {an,ex,ne,xa}. Finally, the measure often associates n-grams that occur in radically different word positions, as in the pair *V***ol***taren/Tramad***ol**.

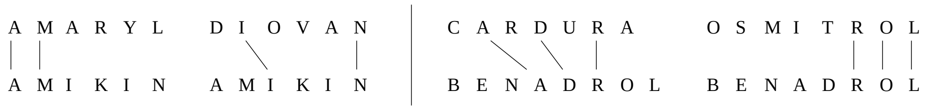

Fig. 2. Two pairs of words with different levels of similarity and the same length of the longest common subsequence.

Brew and McKelvie [1] propose an extension of DICE, called XXDICE, in which the contribution of matching n-grams to the overall score depends on their absolute positions in the strings. XXDICE performed best among several tested measures, and it has subsequently been used by other researchers (e.g., [11]). Unfortunately, the definition of XXDICE is deficient: it does not specify which matching bigrams are to be selected for the calculation of the score when bigrams are not unique. There are a number of ways to amend the definition, but it then becomes implementation-dependent, which means that the results are no longer fully replicable. The case of XXDICE serves as an illustration that it is essential to define string similarity measures rigorously.

In the next section, we formulate the notion of n-gram similarity $\mathbf{s_n}$, which is intended to combine the advantages of the LCS length and Dice coefficient while eliminating their flaws.

3 n-gram similarity

The main idea behind n-gram similarity is generalizing the concept of the longest common subsequence to encompass n-grams, rather than just unigrams. We formulate n-gram similarity as a function $\mathbf{s_n}$, where n is a fixed parameter. $\mathbf{s_1}$ is equivalent to the unigram similarity function $\mathbf{s}$ defined in Section 2.2.

3.1 Definition

For the purpose of providing a concise recursive definition of n-gram similarity, we slightly modify our convention regarding Γ. When dealing with n-grams for $n > 1$, we require $\Gamma_{i,j}$ and $\Gamma^*_{i,j}$ to contain at least one complete n-gram. This requirement is consistent with our previous convention for $n = 1$. If both strings are shorter than n, $\mathbf{s_n}$ is undefined.

In the simplest case, when there is only one complete n-gram in either of the strings, n-gram similarity is defined to be zero:

$$\mathbf{s_n}(\Gamma_{k,l}) = 0 \text{ if } (k = n \wedge l < n) \vee (k < n \wedge l = n)$$

Let $\Gamma^n_{i,j} = (x_{i+1} \ldots x_{i+n}, y_{j+1} \ldots y_{j+n})$ be a pair of n-grams in X and Y. If both strings contain exactly one n-gram, our initial definition is strictly binary: 1 if the n-grams are identical, and 0 otherwise. (Later, we will consider modifying this part of the definition.)

$$\mathbf{s_n}(\mathit{\Gamma}_{n,n}) = \mathbf{s_n}(\mathit{\Gamma}^n_{0,0}) = \begin{cases} 1 & \text{if } \forall_{1 \le u \le n}\ x_u = y_u \\ 0 & \text{otherwise} \end{cases}$$

For longer strings, we define n-gram similarity recursively:

$$\mathbf{s}(X, Y) = \mathbf{s_n}(\mathit{\Gamma}_{k,l}) = \max_{i,j}(\mathbf{s_n}(\mathit{\Gamma}_{i+n-1,j+n-1}) + \mathbf{s_n}(\mathit{\Gamma}^*_{i,j}))$$

The values of i and j in the above formula are constrained by the requirement that both $\mathit{\Gamma}_{i,j}$ and $\mathit{\Gamma}^*_{i,j}$ contain at least one complete n-gram. More specifically, the admissible values of i and j are given by $D(k-n+1, l-n+1)$, where D is the set defined in Section 2.2.

3.2 Computing n-gram similarity

As in the case of $\mathbf{s}$, a set of three decompositions is sufficient for computing $\mathbf{s_n}$.

$$\mathbf{s_n}(\mathit{\Gamma}_{k,l}) = \max(\mathbf{s_n}(\mathit{\Gamma}_{k-1,l}), \mathbf{s_n}(\mathit{\Gamma}_{k,l-1}), \mathbf{s_n}(\mathit{\Gamma}_{k-1,l-1}) + \mathbf{s_n}(\mathit{\Gamma}^n_{k-n,l-n}))$$

An efficient dynamic-programming algorithm for computing n-gram similarity can be derived directly from the alternative formulation. For $n = 1$, it reduces to the well-known algorithm for computing the length of the LCS. The algorithm is discussed in detail in Section 5.

3.3 Refined n-gram similarity

The binary n-gram similarity defined above is quite crude in the sense that it does not differentiate between slightly different n-grams and totally different n-grams. We consider here two possible refinements to the similarity scale. The first alternative, henceforth referred to as *comprehensive* n-gram similarity, is to compute the standard unigram similarity between n-grams:

$$\mathbf{s_n}(\mathit{\Gamma}^n_{i,j}) = \frac{1}{n}\mathbf{s_1}(\mathit{\Gamma}^n_{i,j})$$

The second alternative, henceforth referred to as *positional* n-gram similarity, is to simply to count identical unigrams in corresponding positions within the n-grams:

$$\mathbf{s_n}(\mathit{\Gamma}^n_{i,j}) = \frac{1}{n}\sum_{u=1}^{n}\mathbf{s_1}(x_{i+u}, y_{j+u})$$

The advantage of the positional n-gram similarity is that it can be computed faster than the comprehensive n-gram similarity.

Figure 1 (right) shows a bigram decomposition of a pair of words with various levels of bigram similarity. The solid link denotes a complete match. The dashed links are partial matches according to both positional and comprehensive n-gram similarity. The dotted link indicates a partial match that is detected by the comprehensive n-gram similarity, but not by the positional n-gram similarity.

4 n-gram distance

Since the standard edit distance is almost a dual notion to the length of the LCS, the definition of n-gram distance differs from the definition of n-gram similarity only in details:

1. Recursive definition of edit distance:
$$\mathbf{d}(x,\epsilon)=1, \qquad \mathbf{d}(\epsilon,y)=1, \qquad \mathbf{d}(x,y)=\begin{cases}0 \text{ if } x=y\\ 1 \text{ otherwise}\end{cases}$$
$$\mathbf{d}(X,Y)=\mathbf{d}(\Gamma_{k,l})=\min_{i,j}(\mathbf{d}(\Gamma_{i,j})+\mathbf{d}(\Gamma^*_{i,j}))$$
2. An alternative formulation of edit distance with a reduced set of decompositions:
$$\mathbf{d}(X,Y)=\mathbf{d}(\Gamma_{k,l})=\min(\mathbf{d}(\Gamma_{k-1,l})+1,\mathbf{d}(\Gamma_{k,l-1})+1,\mathbf{d}(\Gamma_{k-1,l-1})+\mathbf{d}(x_k,y_l))$$
3. Definition of n-gram edit distance:
$$\mathbf{d_n}(\Gamma_{k,l})=1 \text{ if } (k=n \wedge l<n)\vee(k<n \wedge l=n)$$
$$\mathbf{d_n}(\Gamma_{n,n})=\mathbf{d_n}(\Gamma^n_{0,0})=\begin{cases}0 & \text{if } \forall_{1\le u\le n} x_u=y_u\\ 1 & \text{otherwise}\end{cases}$$
$$\mathbf{d_n}(\Gamma_{k,l})=\min_{i,j}(\mathbf{d_n}(\Gamma_{i+n-1,j+n-1})+\mathbf{d_n}(\Gamma^*_{i,j}))$$
4. An alternative formulation of n-gram distance:
$$\mathbf{d_n}(\Gamma_{k,l})=\min(\mathbf{d_n}(\Gamma_{k-1,l})+1,\mathbf{d_n}(\Gamma_{k,l-1})+1,\mathbf{d_n}(\Gamma_{k-1,l-1})+\mathbf{d_n}(\Gamma^n_{k-n,l-n}))$$
5. Three variants of n-gram distance $\mathbf{d_n}(\Gamma^n_{i,j})$:
   (a) The binary n-grams distance, as defined in 3.
   (b) The comprehensive n-grams distance: $\mathbf{d_n}(\Gamma^n_{i,j})=\frac{1}{n}d_1(\Gamma^n_{i,j})$.
   (c) The positional n-gram distance: $\mathbf{d_n}(\Gamma^n_{i,j})=\frac{1}{n}\sum_{u=1}^{n}d_1(x_{i+u},y_{j+u})$.

The positional n-gram distance is equivalent to the the comprehensive n-gram distance for $n=2$. All three variants are equivalent for $n=1$.

5 n-gram word similarity measures

In this section, we define a family of word similarity measures (Table 1), which include two widely-used measures, the longest common subsequence ratio (LCSR) and the normalized edit distance (NED), and a series of new measures based on n-grams, $n>1$. First, however, we need to consider two measure-related issues: normalization and affixing.

Normalization is a method of discounting the length of words that are being compared. The length of the LCS of two randomly-generated strings grows with

	$n=1$	$n=2$	$n=3$	...	n
Similarity	LCSR	BI-SIM	TRI-SIM	...	n-SIM
Distance	NED_1	BI-DIST	TRI-DIST	...	n-DIST

Table 1. A classification of measures based on n-grams.

the length of the strings [2]. In order to avoid the length bias, a normalized variant of the LCS is usually preferred. The longest common subsequence ratio (LCSR) is computed by dividing the length of the longest common subsequence by the length of the longer string [8]. Edit distance is often normalized in a similar way, i.e. by the length of the longer string (e.g., [5]). However, Marzal and Vidal [6] propose instead to normalize by the length of the editing path between strings, which requires a somewhat more complex algorithm. We refer to these two variants of Normalized Edit Distance as NED_1 and NED_2, respectively.

Affixing is a way of increasing sensitivity to the symbols at string boundaries. Without affixing, the boundary symbols participate in fewer n-grams than the internal symbols. For example, the word *abc* contains two bigrams: *ab* and *bc*; the initial symbol a occurs in only one bigram, while the internal symbol b occurs in two bigrams. In the context of measuring word similarity, this is a highly undesirable effect because the initial symbols play crucial role in human perception of words. In order to avoid the negative bias, extra symbols are sometimes added to the beginnings and/or endings of words.

Algorithm N-SIM (X, Y)

$K \leftarrow \text{length}(X)$
$L \leftarrow \text{length}(Y)$
for $u \leftarrow 1$ **to** $N-1$ **do**
    $X \leftarrow x'_1 + X$
    $Y \leftarrow y'_1 + Y$
for $i \leftarrow 0$ **to** K **do**
    $S[i, 0] \leftarrow 0$
for $j \leftarrow 1$ **to** L **do**
    $S[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** K **do**
    for $j \leftarrow 1$ **to** L **do**
        $S[i, j] \leftarrow max($
            $S[i-1, j],$
            $S[i, j-1],$
            $S[i-1, j-1] + \mathbf{s_N}(\Gamma^N_{i-1,j-1}))$
return $S[K, L]/\max(K, L)$

Algorithm N-DIST (X, Y)

$K \leftarrow \text{length}(X)$
$L \leftarrow \text{length}(Y)$
for $u \leftarrow 1$ **to** $N-1$ **do**
    $X \leftarrow x'_1 + X$
    $Y \leftarrow y'_1 + Y$
for $i \leftarrow 0$ **to** K **do**
    $D[i, 0] \leftarrow i$
for $j \leftarrow 1$ **to** L **do**
    $D[0, j] \leftarrow j$
for $i \leftarrow 1$ **to** K **do**
    for $j \leftarrow 1$ **to** L **do**
        $D[i, j] \leftarrow min($
            $D[i-1, j] + 1,$
            $D[i, j-1] + 1,$
            $D[i-1, j-1] + \mathbf{d_N}(\Gamma^N_{i-1,j-1}))$
return $D[K, L]/\max(K, L)$

Fig. 3. The algorithms for computing N-SIM and N-DIST of strings X and Y.

The proposed n-gram similarity and distance measures N-SIM and N-DIST incorporate both normalization and affixing (Figure 3). Our affixing method is aimed at emphasizing the initial segments, which tend to be much more important than final segments in determining word similarity. A unique special symbol is defined for each letter of the original alphabet. Each word is augmented with a prefix composed of $n - 1$ copies of the special symbol that corresponds to the initial letter of the word. For example, if $n = 3$, *amikin* is transformed into *ââamikin*. Assuming that the original words have lengths K and L respectively, the number of n-grams is thus increased from $K + L - 2(n - 1)$ to $K + L$. The normalization is achieved by simply dividing the total similarity score by $\max(K, L)$, the original length of the longer word. This procedure guarantees that the new measures return 1 if and only if the words are identical, and 0 if and only if the words have no letters in common.

6 Experiments

In this section we describe three experiments aimed and comparing the effectiveness of the standard unigram similarity measures with the proposed n-gram measures. The three experiments correspond to applications in which the standard unigram measures have been used in the past.

6.1 Evaluation methodology

Our evaluation methodology is the same in all three experiments. The underlying assumption is that pairs of words that are known to be related in some way (e. g., by sharing a common origin) exhibit on average much greater similarity than unrelated pairs. We evaluate the effectiveness of several similarity measures by calculating how well they are able to distinguish related word pairs from unrelated word pairs. In order for a measure to achieve 100% accuracy, any related pair would have to be assigned a higher similarity value than any unrelated pair. In practice, most of the related pairs should occur near the top of a list of pairs sorted by their similarity value.

The evaluation procedure is as follows:

1. Establish a gold standard set G of word pairs that are known to be related.
2. Generate a much larger set C of candidate word pairs, $C \supset G$.
3. Compute the similarity of all pairs in C using a similarity measure.
4. Sort the pairs in C according to the similarity value, breaking ties randomly.
5. Compute the 11-point interpolated average precision on the sorted list.

The 11-point interpolated average precision is an information-retrieval evaluation technique. Precision is computed for the recall levels of 0%, 10%, 20%, . . . , 100%, and then averaged to yield a single number. We uniformly set the precision value at 0% recall to 1, and the precision value at 100% recall to 0.

6.2 Data

Genetic cognates Cognates are words of the same origin that belong to distinct languages. For example, English *father*, German *vater*, and Norwegian *far* constitute a set of cognates, since they all derive from a single Proto-Germanic word (reconstructed as **faδēr*). The identification of cognates is one of the principal tasks of historical linguistics. Cognates are usually similar in their phonetic form, which makes string similarity an important clue for their identification.

In the first experiment, we extracted all nouns from two machine-readable word lists that had been used to produce an Algonquian etymological dictionary [4]. The two sets contain 1628 Cree nouns and 1023 Ojibwa nouns, respectively. The set C of candidate pairs was created by generating all possible Cree-Ojibwa pairs (a Cartesian product). An electronic version of the dictionary, which contains over four thousand Algonquian cognate sets, served as the gold standard G. The task was to identify 409 cognate pairs among 1,650,780 candidate word pairs (approx. 0.025%).

Translational cognates Cognates are usually similar in form *and* meaning, which makes string similarity a useful clue for word alignment in statistical machine translation. Both LCSR and edit distance have been employed for cognate identification in bitext-related tasks (e.g., [8]).

In the second experiment, we used *Blinker*, a word-aligned French-English bitext containing translations of 250 sentences taken from the Bible [7]. For the evaluation, we manually identified all cognate pairs in the bitext, using word alignment links as clues. The candidate set of pairs was generated by taking a Cartesian product of words in corresponding sentences. This time, the task was to identify those 959 pairs among 36,879 candidate pairs (approx. 2.6%).

Confusable drug names Many drug names either look or sound similar, which causes potentially dangerous errors. An example of a confusable drug name pair is *Zantac* and *Zyrtec*. Orthographic similarity measures have been applied in the past for detecting confusable drug names. For example, Lambert et al. [5] tested edit distance, normalized edit distance, and LCS, among other measures.

In the final experiment, we extracted 582 unique drug names form an online list of confusable drug names [13]. The candidate set of pairs was the Cartesian product of the names. The list itself served as the gold standard. The task was to identify 798 confusable pairs among 338,142 candidate pairs (approx. 0.23%).

6.3 Results and discussion

Table 2 compares the average precision achieved by various measures in all three experiments. The similarity-based measures are given first, followed by the distance-based measures. PREFIX is a baseline-type measure that returns the length of the common prefix divided by the length of the longer string.

	DICE	XXDICE	LCS	LCSR	BI-SIM			TRI-SIM		
					bin	pos	com	bin	pos	com
Drug names	.262	.308	.152	.330	.377	.403	.400	.356	.393	.396
Genetic cognates	.394	.519	.141	.564	.526	.597	.595	.466	.593	.589
Transl. cognates	.775	.815	.671	.798	.841	.841	.846	.829	.838	.832

	PREFIX	EDIT	NED_1	NED_2	BI-DIST			TRI-DIST		
					bin	pos	com	bin	pos	com
Drug names	.256	.275	.364	.369	.389	.399	.399	.352	.391	.391
Genetic cognates	.276	.513	.592	.592	.545	.602	.602	.468	.589	.589
Transl. cognates	.721	.681	.821	.823	.840	.838	.838	.828	.829	.830

Table 2. The average interpolated precision for various measures on three word-similarity tasks.

Three values are given for the N-SIM and N-DIST measures corresponding to the binary, positional, and comprehensive variants, respectively.

Although the average precision values vary depending on the difficulty of a particular task, the relative performance of the measures is quite consistent across the three experiments. The positional and comprehensive variants of the n-gram measures outperform the standard unigram measures (the only exception is that NED slightly outperforms TRI-DIST on genetic cognates). The difference is especially pronounced in the drug names experiment. The bigram methods are overall somewhat more effective than the trigram methods. The differences between the positional and the comprehensive n-gram variants, where they exist, are insignificant, but the binary variant is sometimes much worse. The normalized versions substantially outperform the un-normalized versions in all cases. NED consistently outperforms LCSR, but the differences between the similarity-based methods and the distance-based methods for $n > 1$ are minimal.

6.4 Similarity vs. distance

Interestingly, there is a considerable variation in performance among the unigram measures, but not among the multigram measures. The reason may lie in LCSR's lack of context-sensitivity, which we mentioned in Section 2.5. Consider again the two pairs on the right side of Figure 2. The LCS lengths are identical in both cases (3), but edit distances differ (7 and 5, respectively). Notice the highly parallel arrangement of the identity links between the second pair, a phenomenon which usually positively correlates with perceptual similarity. Since by definition LCSR is concerned only with the number of identity matches, it cannot detect such a clue. The multigram measures, on the other hand, are able to recognize the difference, because n-grams provide an alternative mechanism for taking context into account.

7 Conclusion

We have formulated a new concept of n-gram similarity and distance, which generalizes the standard unigram string similarity and distance. On that basis, we have formally defined a family of new measures of word similarity, We have evaluated the new measures on three different word-comparison tasks. The experiments suggest that the new n-gram measures outperform the unigram measures. In general, normalization by word length is a must. With respect to the unigram measures, we have argued that the normalized edit distance may be more appropriate than LCSR. For $n \geq 2$, BI-SIM with positional n-gram similarity is recommended as it combines relative speed with high overall accuracy.

Acknowledgments

Thanks to Bonnie Dorr for collaboration on the confusable drug names project. This research was supported by Natural Sciences and Engineering Research Council of Canada.

References

1. Chris Brew and David McKelvie. 1996. Word-pair extraction for lexicography. In *Proc. of the 2nd Intl Conf. on New Methods in Language Processing*, pages 45–55.
2. Vaclàv Chvátal and David Sankoff. 1975. Longest common subsequences of two random sequences. *Journal of Applied Probability*, 12:306–315.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. The MIT Press, second edition.
4. John Hewson. 1993. *A computer-generated dictionary of proto-Algonquian*. Hull, Quebec: Canadian Museum of Civilization.
5. Bruce L. Lambert, Swu-Jane Lin, Ken-Yu Chang, and Sanjay K. Gandhi. 1999. Similarity As a Risk Factor in Drug-Name Confusion Errors: The Look-Alike (Orthographic) and Sound-Alike (Phonetic) Model. *Medical Care*, 37(12):1214–1225.
6. A. Marzal and E. Vidal. 1993. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(9):926–932.
7. I. Dan Melamed. 1998. Manual annotation of translational equivalence: The Blinker project. Technical Report IRCS #98-07, University of Pennsylvania.
8. I. Dan Melamed. 1999. Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1):107–130.
9. D. Sankoff and J. B. Kruskal, editors. 1983. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley.
10. Bill Smyth. 2003. *Computing Patterns in Strings*. Pearson.
11. Dan Tufis. 2002. A cheap and fast way to build useful translation lexicons. In *Proc. of the 19th Intl Conf. on Computational Linguistics*, pages 1030–1036.
12. Esko Ukkonen. 1992. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211.
13. Use caution — avoid confusion. *United States Pharmacopeial Convention Quality Review*, No. 76, March 2001. Available from *http://www.bhhs.org/pdf/qr76.pdf*.
14. Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.