Introducing UWS – A Fuzzy Based Word Similarity Function with Good Discrimination Capability: Preliminary results

Joao Paulo Carvalho
Instituto Superior Técnico – Technical University of Lisbon / INESC-ID
R. Alves Redol 9, 1000-029 Lisboa
Portugal
joao.carvalho@inesc-id.pt

Luisa Coheur
Instituto Superior Técnico - Technical University of Lisbon / INESC-ID
R. Alves Redol 9, 1000-029 Lisboa
Portugal
luisa.coheur@inesc-id.pt

Abstract — **This paper introduces a novel word similarity function, the Uke Similarity Function (UWS), that fuses the most interesting characteristics of the two main philosophies in word and string matching: the edit distance and the n-gram similarity approach. It also uses fuzzy sets to integrate expert knowledge about typographical errors and to easily include phonetic and token related errors. The UWS was developed with the goal of automatic detection and correction of typographical and other word errors in unedited corpus data when creating word lists.**

Keywords: Word Similarity, Word Matching, Fuzzy Sets, Typographical Error Detection and Correction, Unedited Corpus Data.

I. Introduction[1]

Word error detection and correction in unedited text is a complex and expensive task that is often necessary for natural language processing. This task must usually be done manually, or, even if done automatically, demands a significant human intervention. The goal of the work presented in this paper is to automatically detect and correct typographical and other word errors in unedited corpus when creating word lists.

In order to approach this problem one might ask why can't a simple dictionary-based spelling correction tool be used. There are several motives that prevent such use: most automatic spelling tools often need a rather intensive human input; automatic spelling corrections are strongly limited by the number of errors (one or at most two), even if many words contain more than a single typographical error; many words present in corpora text, such as technical terms or abbreviations, are not present in dictionaries and as such, not recognized; etc. Despite the limitations associated with automatic spelling tools, dictionaries (and known word lists) can and should obviously be used to help to detect and correct word errors.

The envisioned automatic corrected word list creation from unedited corpus data proceeds as follows:

i. Count and create a list with all words in the corpus data; if necessary eliminate stop and/or functional words.
ii. Order the list based on most frequent words or on tf–idf (term frequency–inverse document frequency).
iii. Divide the word list into top-k and bottom-l lists. In principle, the top-k list should not contain words with typographical errors – the same word would have to be almost constantly miswritten (and the error would have to be exactly the same) for it to appear in top-k.
iv. Run the bottom-l list through an extensive known word list and mark words that are not recognized, creating an unknown word list.
v. Use a word similarity function to compare each word in the unknown list with each word in the top-k list.
vi. Unknown words are corrected to the most similar word in the top-k list (as long as the similarity is above a given threshold).
vii. Words that are not paired can be either left as is, or compared for similarity in the extensive known-word list (which might be a lengthy process). Often the latter should not be necessary since bottom-l words that are not related to the top-k words are not relevant in most natural language processing.

In order for the above procedure to give good results, a good word similarity function that is especially adapted to deal with common typographical errors is needed. Current research on string similarity offers a panoply of measures that can be used in this context, such as the ones based on edit distances or on the length of the longest common subsequence of the strings. However, most of the existing measures have their own drawbacks. For instance, some do not take into consideration linguistically-driven misspellings, others the phonetics of the string or the mistakes resulting from the input device (a keyboard or a mobile). Moreover, the majority of the existing

[1] This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/IVC-ESCT/4919/2012, PTDC/EMS-SIS/3220/2012, and PEst-OE/EEI/LA0021/2011.

measures do not have a strong discriminative power, i.e., both a good precision and recall, and, therefore, it is difficult to evaluate if the proposed suggestion is reasonable or not, which is a core issue in unsupervised spelling.

This paper introduces a novel word similarity function, the Uke Word Similarity (UWS), and its fuzzy variant, FUWS, that, by combining the most interesting characteristics of the two main philosophies in word and string matching, and by integrating specific fuzzy based expert knowledge concerning typographical errors, tries to achieve a good discrimination. In the paper, we test the proposed measure in two scenarios, and although preliminary, the achieved results are promising.

The paper is organized as follows: Section II. Describes typographic and other word common errors; Section III. briefly introduces the most used techniques to approach the word and string similarity problem; Section IV. presents the proposed UWS algorithm; Section V. presents some preliminary results; Finally Section VI. presents conclusions and future developments.

II. Typographical and other Word Errors

A typographical error, colloquially known as "typo", is a mistake made in the typing process. Most typographical errors consist of substitution, transposition, duplication or omission of a small number of characters. Damerau [1] considered that a simple error consists in only one of these operations.

Recently, a new type of errors associated with smaller keyboards (either in netbooks, tablets or smartphones) became relevant due to their effect in the increase of the number of word typos. These errors are known as "fat finger" (hitting two adjacent keys), and "thumbo" – from "thumb" and "typo" (hitting a nearby key).

Two other types of typing errors have also become frequent in unedited corpora text due to the widespread use of blogs, microblogs, instant messaging, etc.: the use of multiple repeated letters, to emphasize emotions and intensity; and the replacement of letters by numbers.

Atomic typos are typographical errors that result in existing words and are therefore not detected unless sophisticated context based methods are used. E.g., "abroad" vs. "aboard" or "from" vs. "form". The proposed similarity function has no particular way to deal with this kind of typographical errors. However, the overall procedure described in the Introduction section can partially lower their impact by design: since a word in the bottom-l word list is not usually checked on a large dictionary, only atomic typos between words that are both relevant to the context (i.e., that are both in the top-k list) can occur.

Finally one must also mention linguistic errors, which are mostly due to lack of culture and or/education, and are usually the result of phonetic similarities.

III. Related Work

There is an unaccountable number of ways to define and calculate word and/or string similarity. There are basically two kinds of string similarity functions: those that measure a distance between the strings, where the larger the distance, the less similar the strings are; and those that calculate a similarity ratio, usually ranging from 0 to 1, where 1 indicates identical strings.

One of the factors that contributes to the unusually high number of existent string similarity functions is the diverse fields where such functions are applied. As expected, researchers on each field tend to look into their own field, and focus on the specific interests of their fields. This leads to parallel development and proposals of different functions, different performance metrics, different test corpora, etc. Let us see some examples:

- Approximate string matching is often referred to as "fuzzy string searching", even if the connection to the fuzzy sets and systems community is very shallow (to say the least). Fuzzy string search is the technique of finding strings that match a pattern approximately (rather than exactly), and is typically divided into two sub-problems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately, both online and offline. Performance is a key factor in fuzzy string search. In [2] one can find a very extensive work regarding fuzzy string search and description of the most used techniques.
- A very different problem that is also worried about string similarity is named entity matching. Entity name matching has been explored by a number of communities, including statistics, databases, and artificial intelligence, and where each community has formulated the problem differently, and different techniques have been proposed. In [3] one can find a comparison of the best techniques to approach named entity matching.

The best techniques to approach one problem are very different from the techniques in the other. And the same happens between those problems and the present problem of detecting and correcting typographical errors. Hence the huge variety in proposed string/word similarity functions, and the eventual need for improvement in this particular problem.

Nowadays it is possible to say that the two fields that are more active in word/string similarity functions are Natural Language Processing, and Information Retrieval.

The one common point on all textual comparison approaches from all areas, is the use of the Levenshtein edit distance [4]. An edit distance is the minimal number of elementary edit operations needed to transform one string into another. The Levenshtein distance only accounts for single character insertion, deletion or substitution. When one also accounts for transpositions one obtains the Damerau-Levenshtein distance. One can use the same or different weights for each operation. The distance between strings equals to the minimal possible weight (or cost) among the sequences that transform one string into another.

The second most frequently considered method for text comparison in the different application areas is n-gram based distances. These distances are based on counting the number of common substrings of fixed length. Such substrings are called n-grams. Kondrak [5] developed the notion of n-gram similarity and distance and proposed that the edit distance and the length of the longest common subsequence are special cases of n-gram distance and similarity. We should note that some authors refer to word n-grams, others to character n-grams. In this paper we consider word n-grams. Trigrams are especially popular.

A different approach when considering word similarity is the Metaphone approach [6], which is a phonetic algorithm for indexing words by they English pronunciation. This algorithm has the very interesting property of facilitating the detection of spelling errors based on phonetics.

A method often used for spellchecking is the noisy channel model for spelling [7][8]. It is a Bayesian based method that tries to predict the most common word that follows a previous *n* word sequence. It can be considered a contextual method. It is used by Microsoft and Google in dictionaries, and also by many search engines. It is a method that needs a very large training corpus and training time. It is a totally different approach in the sense that no similarity function is used except to prune the possible alternative choices for a given word, usually the edit distance (to a maximum of one). The main advantage of this method is that it can be tailored to employ synonyms if one ignores the edit distance restriction. Other context based approaches are possible, e.g., asymmetric word similarity (AWS) [9].

An interesting method for measuring string similarity due to its conceptual simplicity is the Jaro–Winkler distance [10], which usually performs very well in names and other short string comparisons, and is extensively use for duplicate detection in the linkage area. It scores the number of common character in the correct order

Finally we can also mention some commonly used similarity measures, such as Dice [11] or Jaccard [12] that can be applied to compute word similarity. However we must stress that they usually produce worse results when applied to the typographical error correction problem.

As it has been repeated before, there is a large number of different algorithms and approaches to compute text similarity. Most are variants, improvements or combinations of the above methods that are tailored to solve a given problem.

Regarding the current problem, a very interesting work by Han et. al [13], has compared and used several techniques to automatically correct text in a very large microblog corpus and create a dictionary based on the results. Since the resulting dictionary is available online, it was used to test the word similarity function proposed in the present work.

IV. Uke Word Similarity

The Uke Word Similarity function (UWS) compares the similarity between two words, a candidate word (i.e. a word that might contain one or more "typos"), and a known word, present in a list or a dictionary. It takes advantage of the most relevant characteristics of the n-gram methods (in this case, a trigram) and the edit distance methods, being formulated to address the most common typographical errors: substitution; transposition; duplication and omission. The UWS returns a normalized value between 0 and 1, where 0 indicates no similarity at all, and 1 indicates identical words. The method can be summarized as follows:

The first step consists in preprocessing the candidate word to reduce consecutive letter repetition to a maximum of two letters. This can be done without harm since in English (and most western languages) there are no words with more than two consecutive repeated letters (except for named entities like, for example, "WWW"), and partially deals with the tendency of users to repeat a letter to express intensity, as for example in "kisssssssssss".

Then all letters in the candidate word are scored up to a maximum of 2 according to the following guidelines:

i. The maximum letter score is given when the candidate letter exists in the known word within a given position vicinity, and the trigram where the letter is centered is identical in both words. E.g., when comparing the similarity of "congress" and "regression", the "r" scores 2 since the trigram "gre" is common to both words and is within a vicinity of 1 ("r" is the 5th letter in "congress" and the 4th letter in "regression").

ii. If the letter exists within the mentioned vicinity but the trigrams differ, then the score is 1 + 0.5 if the trigram contains two common letters. E.g., the "g" in congress will score 1.5, since the trigrams "ngr" and "egr" share two common letters. Note that the order of the common letter surrounding the middle letter is irrelevant; "ngr" and "rge" would also score 1.5. This allows scoring typographical transposition errors within a word. It is possible to score the order differently (e.g., 0.35 if the order is different), but preliminary tests shown better results when the distinction is not made.

iii. If the candidate letter is not found in the known word within the given vicinity, then we look for a neighboring trigram where only the candidate letter is different. E.g., if the candidate word is "congtess" and the known word is "congress", since "t" is not present in the known word, then when scoring the letter "t" we would look for trigram "g*e" within a radius 2 of position 5. If such trigram exists, then the score for the absent letter is 1, since we are probably dealing with a typing error. The last letter is never considered for this situation.

The first letter is treated as a special case: we account for the space before the first letter only if the first letter of candidate and known words are equal. This rule acknowledges the fact that usually the first letter is not typed or misspelled wrong except for some particular cases (like 'h' in Portuguese.)

The need to check for a letter within a given vicinity in the known word comes from the fact that insertions and deletions change the position within the words where letters should be compared. A vicinity of [-2,+4] characters was chosen empirically, and is extended to the right to deal with the tendency of users to repeat letters when in doubt or, as mentioned above, to express emotional intensity. Any other range can be used, and nothing prevents it from depending on word size.

After all letters in the candidate word have been scored, we obtain a partial score (*ps*) that is worth up to a maximum of twice the length of candidate word (*lencw*). Note that deletions and insertions are automatically scored by the described procedure: a deleted letter is simply not scored; an additional letter penalizes *ps* since it disrupts the trigrams and will result in a different word size.

Finally we compute UWS according to (1):

$$UWS = \frac{ps}{2*(lencw+iLD)} \quad , \qquad (1)$$

where *iLD* is a score based on the length difference between the candidate word and the known word (see section IV-A). The larger the difference in length between the candidate and the known words, the lower is UWS. The maximum value UWS=1 is obtained only when $ps = 2*lencw$ and $iLD=0$.

We should note that the UWS is not symmetrical. This implies that, in mathematical terms, it cannot be a metric or a distance. It is possible to make the UWS symmetrical by computing it twice with transposed arguments and then computing the geometrical average of the result. This would obviously imply a heavy performance penalty (more than double the computing time), but, as seen in the results section, the algorithm would still compare favorably to one of the most used metrics, the Damerau-Levenstein.

In fact, even if no formal theoretic comparisons have yet been done, the algorithm for implementation of the UWS is computationally quite undemanding, especially when compared to even the most efficient implementations of the Damerau-Levenshtein distance. Even if the score operations for each letter can be more complex than Damerau-Levenshtein (when the candidate letter is not found), each letter in the candidate word is searched for within the given vicinity up to a maximum of three times. In Damerau-Levenshtein one must run through all the known word for at least the number of characters in the candidate word. In terms of memory structures, one only needs an additional array with size = *lenkw* (length of the known word) apart from the strings to be compared. Such array (*checkedletters*) is used to indicate the position of each letter that has been scored in the known word. This array allows the algorithm to deal with repeated additional letters and also transpositions.

Table I shows a Python based pseudo-code for the UWS implementation:

- The code roughly follows the previous description.
- The function *RemoveReps(cw)* deals with multiple letter repetitions and is described in IV-A.
- One can distinguish three blocks containing search operations *kw.find()* that correspond to scoring phases i. to iii. Nested within each operation are the scoring conditions for each phase.
- The function *TypoMiu()* is related with the computing of the Fuzzy UWS and will be described in IV-B. In the UWS case, it returns zero by default.

A. Dealing with Double Letters and Letter Multiple Repetition

As mentioned above, multiple repetitions of a letter are quite common in unprocessed texts, especially in micro-blogs, short text messages, non-formal emails, etc. Multiple repetitions of the same letter can ruin any word similarity method. Fortunately such repetitions can be easily (and efficiently processed) on a candidate word. However, in order to guarantee that the meaning of the word is not changed, one cannot automatically reduce double letters to single letters. For example, “gooooood moooooorningggg” must be reduced to “good moorningg” and not to “good morning” without additional complex processing. If one would opt to reduce all double letters to a single letter, “good” would become “god”, and the processing of the original word would result on a false positive. Even if “moorningg” and “morning” would still provide a good match, the UWS would be lower than it should be due to the penalty in word length difference.

Double letters are also a problem by itself since words containing double letters are among the most typographical errors prone. Yet, such errors rarely are a source of confusion, and therefore, any word similarity function should basically ignore them except in the cases where the word meaning

In order to deal with these problems, double letters are ignored (counted as a single letter) when computing *iLD*, the length difference between the candidate and known words in (1). Examples of the result of using such approach are shown in section V.

B. Fuzzy Uke Word Similarity

Even if the UWS allows for an interesting way to detect most of the typographical errors, it can still be improved in order to perform better in what concerns wrong-key typing errors, spelling errors and errors that origin from phonetic similarities.

This is accomplished by the use of fuzzy logic based empirical rules in the situation where a given letter is not found in the known word, but a trigram “x*z” exists (see IV.iii) In such case, instead of scoring 0+1 (0 for the incorrect letter, 1 for the correct neighbors), we score the incorrect letter a value in the fuzzy range [0,1] according to the result of the evaluation of a set of fuzzy rules. The rules compare both mid trigram letters.

Rules that account for wrong-key typing errors take into account how far the 2 letters in question are located in the keyboard. For example, on a QWERTY keyboard, key “D” is surrounded by “W”, “E”, “R”, “S”, “F”, “X” and “C”. The most likely wrong-keys are located on the same row (“S”, “F”) and have a high membership wrong-key error ($\mu_{wrong\text{-}key}$=0.7). The keys on the above row (“W”, “E”, “R”) will have a medium/high $\mu_{wrong\text{-}key}$=0.6, and the keys located below (“X”, “C”) will have a medium $\mu_{wrong\text{-}key}$=0.5. Any other keys have a zero membership to wrong-key. For example, when computing the FUWS between candidate word “mosel” and “model”, the letter “s” would have a score of 0.7+1=1.7 instead of 1 when using UWS.

TABLE I: The Uke Word Similarity algorithm in Python

```
def UkeWordSim(cw,kw):

    cw = RemoveReps(cw)

    iLD = iLengthDiff(cw,kw)

    ps = 0.0 #partial sim score
    uws = 0.0 #uke word similarity score
    lenkw=len(kw) #Length of candidate and known words
    lencw=len(cw)
    checkedletters = [0]*lenkw
    kw = string.lower(kw)+'      '
    cw = string.lower(cw)+'  '

    for i in range(lencw):
        if cw[i]==' ': break

        trigram = False
        letter = False

        pos = max(0, i-2)
        while True:
            pos = kw.find(cw[i], pos, i+4)
            if pos == -1: break
            if (checkedletters[pos]==0):
                letter = True
                if cw[i - 1] == kw[pos - 1] and cw[i + 1] == kw[pos + 1]:
                    ps = ps +2 #max score
                    checkedletters[pos]=1
                    trigram = True
                break
            else pos += 1

        if (trigram == False and letter == True):
            pos = max(0, i-2)
            while True:
                pos = kw.find(cw[i], pos, i+4)
                if pos == -1: break
                if (checkedletters[pos]==0):
                    ps = ps + 1
                    checkedletters[pos] = 1
                    if cw[i - 1] == kw[pos - 1] or cw[i + 1] == kw[pos + 1]:
                        ps = ps + 0.5
                    elif cw[i - 1] == kw[pos + 1] or cw[i + 1] == kw[pos - 1]:
                        ps = ps + 0.5
                    break
                else: pos+=1

        if ((pos == -1) and (i != 0)):
            pos = kw.find(cw[i+1], i+1, i+5)
            if ((pos != -1) and (kw[pos] != ' ')):
                if cw[i - 1] == kw[pos - 2]
                    ps=ps+1+TypoMiu(cw[i],kw[pos-1])
                    checkedletters[pos-1]=1

    return (uws= ps/2*(lencw+iLD))
```

The previous rules also account for situations where numbers or symbols are purposely used to replace single letters, like ‘0’ with ‘o’, or ‘1’ with ‘i’, or ‘$’ with ‘s’. The function applies a very high membership $\mu_{wrong\text{-}key}$=0.85 in such cases.

Fuzzy rules that account for phonetic similarity errors, like for example “s” and “z”, or “k” and “q” in certain languages, use a similar principle. However these rules can be more complex than wrong-key rules and often demand extra conditions, like checking for joint characters (bigrams). Some rules were implemented for Portuguese and English for testing purposes only. For example, for the Portuguese language, $\mu_{phonetic}$(q,k)=0.85. The Metaphone approach [6] is being used as the basis for the development of an extensive rulebase. In addition to the Metaphone approach, these rules should also account for usual phonetic similarity between numbers and sounds, such as “4” and “for”, or “8” and “ate”.

Rules for common spelling errors involving single characters for specific languages can also be implemented using similar principles.

The aggregation of the above described membership degrees is obtained by the usual fuzzy t-conorm max (2)

$$\mu_{typo} = \max\left(\mu_{wrong-key}, \mu_{phonetic}, \mu_{spelling}\right) \quad (2)$$

Examples of the efficiency of FUWS are presented in the preliminary results section.

V. Evaluation: Parametrization and Preliminary Results

The proposed similarity function is still under development, and therefore the results cannot be considered definitive or totally conclusive. Nevertheless the UWS (and the FUWS) was applied to two different corpora and compared to systems using other similarity functions. The following sections show the behavior of the UWS when applied to particular cases and the results when applied to different sized corpora.

A. UWS Similarity Threshold

The ultimate goal of a word similarity function, measure, or distance, is to indicate the most similar word present on a given list of known words (dictionary). If the list of words is large enough and it is guaranteed that it contains an approximate match for every possible candidate, then the word with the best similarity degree (or measure, or score, etc.) is chosen. However, in many problems it is not guaranteed that a similar word is present in the dictionary. Therefore, in order to be useful, any non-crisp similarity function must also indicate if any of the similar found words is similar enough. This can be referred to as the similarity threshold.

The similarity threshold for the UWS and FUWS was found empirically by testing the function with EDGAR list of 655 known words [14][15], and with a list of 50 training words that included words containing common typographical errors, and also several EDGAR not-known words.

All possible 32750 were pairs were tested. The results were analyzed and a consensus was reached that all word pairs with a UWS value equal or larger than 0.68 could be considered undoubtedly similar (True Positives), except for a single False Positive, which scored 0.68, that even if have some similarity, no user would confuse them. For the interval 0.55<=UWS<0.68 most words were considered similar to a certain degree (and considered True Positives), and some were deemed less acceptable (therefore considered False Positives). Below 0.55 only one word pair was considered possibly similar (False Negative), and the others not similar (True Negatives). Therefore one can consider 0.55 as the non-similarity threshold, above which false negatives should be highly unlikely.

Given the results, an empirical decision was made to define a fuzzy threshold where three fuzzy sets provide a linguistic indication of the similarity degree. Figure 1 shows the membership functions of the chosen fuzzy sets: *Similar*, *Maybe*, and *Not_Similar*.

Whenever a given word pair is compared, if the fuzzy result includes some degree of uncertainty, i.e., $\mu_{Maybe}(FUWS)>=0$, then a fully automated decision is not advisable and the user should apply additional measures to verify if the found word is acceptabel. Such measures are either manual or must use some kind of context knowledge [7][8].

The crisp threshold of 0.68 was considered as the limit over which false positives and false negatives are highly unlikely, i.e., where both precision and recall are close to 1, and therefore a good discrimination is obtained (we consider discrimination as the harmonic mean of precision and recall). The crisp threshold was used to obtain results for most automatic experiments.

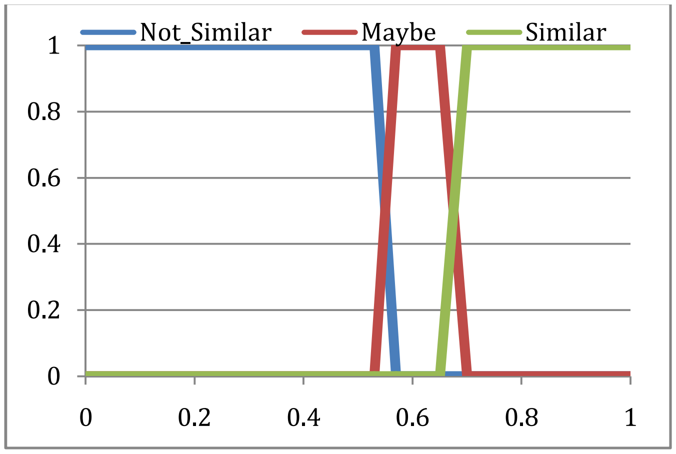

Figure 1 – Membership functions of the UWS Fuzzy Threshold.

B. *Overall Behavior*

We start by presenting some results that give an idea on the behavior of the UWS regarding word error similarity. Table II shows a comparison between Damerau-Levenstein (D-L), Jaro-Winkler (J-W), 3-gram and UWS.

Of the seven presented words, a native language user recognizes four as similar and three as not similar.

Thresholds are obviously a critical point when evaluating each method. For UWS we used 0.68. Most dictionaries usually limit D-L to 1 to limit false positives, but best results are obtained when considering the D-L threshold as 2. There is no usual consensus in what 3gram or J-W thresholds should be, but the examples show that there is no threshold that can be used to classify all the examples correctly.

The first two word pairs represent words with two typical typographic errors. They are a good example of why a pure trigram approach is not a good similarity indicator for this kind of errors (unless a very low threshold < 0.3 is considered). The other three approaches work as intended.

The third word pair has an edit distance of 3 and obviously represents two words that should not be similar. J-W considers it a similar word and fails this test. Usually this measure has some problems with longer words that are revealed here. Note that the 3gram approach gives this word pair the same value as the second word pair, even if the classification should be different.

The fourth pair shows once again the bad performance of J-W for longer words. In this case it does not recognize the similarity, while UWS is able to do it.

The fourth word pair shows how D-L by itself cannot be considered as a similarity indicator: words that are equally distant might be similar or not. This is due to the fact that D-L does not take into account the word length. The same is observed on the fifth pair. It contains a word pair with two Portuguese words that have the same root, but different termination and length. One of the words also contains a typo, but they are two forms of the same verb and are considered similar by any native language speaker.

The final pair shows that UWS can obviously also fail (even if, in this case, not by much). It should not be difficult to find a word pair that can trick any word similarity function. The problem obviously resides in finding a threshold that can optimize both the ratio of True Positives vs. False Positives and of True Positives vs. False Negatives.

TABLE II: A small example of the behavior of the four tested similarity functions

	Similar?	**D-L**	**J-W**	**3gram**	**UWS**
moorningg morning	Y	2	0.93	0.6	0.93
pertendr pretender	Y	2	0.92	0.3	0.72
Nightnight Lightweight	N	3	0.84	0.3	0.65
badketbsl basketball	Y	3	0.00	0.29	0.75
party lazy	N	2	0.63	0	0.25
comstruiram construir	Y	3	0.87	0.46	0.70
ambank albany	N	2	0.78	0.25	0.70

Despite failing in the last word, it is possible to see by the examples that the UWS exhibits a sound behavior. Further comparisons have also shown that in general it performs better than any of the other methods, except for small words, where Jaro-Winkler has an excellent performance.

Table III exemplifies the effect of using the current version of FUWS vs. UWS. The example word pairs are only considered similar (FUWS >= 0.68) by the fuzzy version. The first word pair considers the phonetic similarity of ‘s’ and ‘z’. The second example exemplifies the trend of replacing letters by similar numbers.

TABLE III: The effect of *iLD* in UWS. The edit distance (D-L) is also shown.

	D-L	UWS	FUWS
prsize precise	3	0.64	0.70
f00die foodie	2	0.58	0.73

Table IV. shows the effect of the iLD in UWS output. One can see that the results are exactly as intended, and typing errors involving double letters are basically ignored by the system.

TABLE IV: The effect of *iLD* in UWS

	UWS (without iLD)	UWS (with iLD)
moorningg morning	0.72	0.93
Massachusetts Massachussetts	0.91	0.98
Minnessotta Minnesota	0.77	0.94

C. *Tests using the EDGAR corpus*

Here we test the current version of the FUWS function capability to find the similarity of a 100 candidate word test list against the EDGAR corpus. Of the 100 words, 50 are not present in Edgar and do not have any semantic or apparent similarity connection to the EDGAR list of known words. The other 50 candidate words contain EDGAR known words to which several typographic errors were applied.

Of the unknown words, 38 were classified by FUWS as Not Similar (True Negatives), 10 were classified as Maybe, and 2 classified as Similar (False Negatives). The words that belong to several classes were assigned to the class where the membership degree was higher. We should note that both False Negatives had a non-null membership degree to Maybe.

Of the 50 known words containing typographic errors, 41 were classified as similar (True Positives) and 9 classified as Maybe or Maybe/Similar. No False Negatives were found. All of the 9 non-True Positives contained several typographical errors and scored 3 or higher using the Damerau-Levenstein measure.

Considering a crisp threshold of 0.68, i.e., assuming a totally automatic system that accounts Maybe as Not-Similar, the 10 Maybe unknown words become True Negatives, and the 9 Maybe known words become False Negatives, we obtain:

$$Precision = \frac{TP}{TP + FP} = \frac{41}{43} = 95.3\%$$

$$Recall = \frac{TP}{TP + FN} = \frac{41}{50} = 82\%$$

$$Discrimination = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{41}{43} = 88.3\%$$

The results show a very good discrimination, as originally intended.

D. *Tests using Han et. Al Microblog Dictionary*

In this section we test the FUWS performance against a microblog normalization dictionary [13] that was automatically constructed using several word similarity techniques: standard edit distance [4], edit distance over double metaphone codes (phonetic edit distance)[6], longest common subsequence ratio over the consonant edit distance of the paired words [16], and a string subsequence kernel [17]. The resulting dictionary contains about 40K word pairs mined from 80 million (mostly) English tweets from September 2010 to January 2011. Each pair is formed by an "incorrect" tweeted word that can be an abbreviation, slang and/or contain all types of errors, and an automatically found corrected word.

We started by applying the FUWS to all dictionary word pairs. Results show that 39601 out of the 41181 words, i.e., 96.1% of the word pairs, are also considered similar by the FUWS. Typing errors like, for example, "backgrnd", "backgorund" or "backgrou" were easily classified as "background". Overall such result is excellent if one considers the kind of words included in the dictionary.

An analysis of the remaining 1580 word pairs shown that most mismatches considered dissimilar by the FUWS consist in abbreviations and/or slang that usually can only be find using a contextual approach due to low string similarity. E.g., "bf" is paired in the dictionary to "boyfriend", and "congrats" is paired to "congratulations" instead of a more text-similar match like "contracts". Naturally, it was also easy to find several False Positives in the proposed dictionary, like for example, "1shot" paired to shoot instead of "shot".

Based on these observations we decided to apply FUWS to the 1580 words against a large list consisting of 109584 known English words that include commonly used abbreviations and also named entities [18]. The FUWS found a similar pair for 1175 of the 1580 words. These numbers might seem impressive, but a relevant number of the results is highly questionable. For example, the tweeted word "wowowo" was paired to known word (?) "bowwows", "xover" was paired to "over", "wassuupp" paired to "wasps", "tweetnation" to "testation" and so on. It was not possible to validate the results for all 1580 words, but a small sample reveals that probably around 20% of the results are False Positives. 20% might seem a very high number for False Positives, but it indeed corresponds to roughly 0.6% of the total number of checked words (0.2*1175/41181), which is a very good result. And one must not forget that these are highly unorthodox words being checked against an unusually large (and containing questionable words) wordlist.

E. *UWS performance*

Finally a small test was conducted to compare the performance of the UWS algorithm with an efficient implementation of Damerau-Levenshtein [19]. Candidate words with increasing length were compared to the SIL list of English known words (approximately 109Kwords). The results (TABLE V.) show that the UWS algorithm is always faster, and that increase in word length increases the difference in favor of UWS.

TABLE V: UWS vs. Damerau-Levenshtein performance - time to compute similarity of candidate word with 109583 know-words

Candidate Word length:	**3**	**5**	**8**	**11**
UWS	1.8s	2s	2.8s	4.1s
D-L	4.1s	5.7s	9.5s	13.2s

VI. Conclusions and Future Work

This work presents a proposal for a new word similarity function with a good discrimination capability. Preliminary results are quite interesting but cannot be considered definitive. The Fuzzy UWS implementation is still under development, and the presented results are mostly based on the UWS. The UWS presented promising results when applied to the test corpora and when compared to other word similarity approaches, but more conclusive and extensive testing is necessary and is currently underway.

One of the test corpora where the FUWS and the automatic procedure described in Section I. is being applied, is the medical text annotations corpus of the Intensive Care Unit (ICU) database (MIMIC II)[20], which contains data from different ICUs at Beth Israel Deaconess Medical Center, Boston. It is an extensive database where the text data is unedited, where typographical errors are frequent, and where most relevant information consists of short phrases containing complex medical technical terms that are often referred to using abbreviations. These facts make automatic word correction in this corpus a difficult technical challenge.

Further future work includes, among others: complete the implementation of the FUWS; threshold optimization using more extensive annotated corpora and computational intelligence methods; make a formal study on the UWS complexity in order to test its viability for online application using very large dictionaries.

References

[1] Damerau, F.J. "A Technique for Computer Detection and Correction of Spelling Errors". *Communications of the ACM*, Março 1964, pp. 171-176.

[2] Boytsov, L. 2011. Indexing methods for approximate dictionary searching: Comparative analysis. ACM J. Exp. Algor. 16, 1, Article 1 (May 2011), 91 pages.

[3] Cohen, W., Ravikumar, P., Fienberg, S. “A Comparison of String Distance Metrics for Name-Matching Tasks”, In Proceedings of IJCAI-03 Workshop on Information Integration (August 2003), pp. 73-78

[4] Levenshtein, V. “Binary codes capable of correcting deletions, insertions, and reversals” Soviet Physics Doklady, 1966. 10:707–710.

[5] Kondrak, G., “N-gram similarity and distance”, Proc. Twelfth Int’l Conf. on String Processing and Information Retrieval, 2005

[6] Philips, L., “The double metaphone search algorithm”. C/C++ Users Journal, 2000. 18:38–43.

[7] Jurafsky, D., Martin, J., “Speech and Language Processing - An introduction to natural language processing, computational linguistics, and speech recognition”, 2nd Edition, Prentice-Hall, 2009

[8] Kernighan, M. D., et al, “A spelling correction program based on a noisy channel model”, Proc. of the COLING, Helsinki, 1990, Vol. II,, pp. 205-211

[9] Azmi-Murad, M., Martin, T.P., “Using Fuzzy Sets in Contextual Word Similarity”, Intelligent Data Engineering and Automated Learning – IDEAL 2004,LNCS, Vol.3177, 2004, pp 517-522

[10] Winkler, W.E.. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage". Proceedings of the Section on Survey Research Methods (American Statistical Association): 1990, 354–359.

[11] Dice, Lee R., (1945) "Measures of the Amount of Ecologic Association Between Species". Ecology, 26 (3): 297–302

[12] Jaccard, Paul (1901), "Étude comparative de la distribution florale dans une portion des Alpes et des Jura", Bulletin de la Société Vaudoise des Sciences Naturelles 37: 547–579.

[13] Han, B., Cook, P., Baldwin, T., “Automatically constructing a normalisation dictionary for microblogs”, EMNLP-CoNLL '12, Proc. of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 2012, 421-432

[14] Moreira C., Mendes, A.C., Coheur, L., Martins B., “Towards the Rapid Development of a Natural Language Understanding Module”, *IVA 2011, the 11th International Conference on Intelligent Virtual Agents*, Sep. 2011

[15] https://edgar.l2f.inesc-id.pt/m2/edgar.php

[16] Contractor, D., Faruquie, T.A., Subramaniam, L.V.,”Unsupervised cleansing of noisy text”, In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 189–196, 2010, Beijing, China.

[17] Lodhi H. et al, “Text classification using string kernels”. J. Mach. Learn. Res., 2002, 2:419– 444.

[18] http://www.sil.org/sil/

[19] http://mwh.geek.nz/2009/04/26/python-damerau-levenshtein-distance/

[20] Saeed M, Lieu C, Mark R., “MIMIC II: A massive temporal ICU database to support research in intelligence patient monitoring”, Computers in Cardiology 2002; 29: 641–644.