Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 1721–1726
Marseille, 11–16 May 2020

An Automatic Tool For Language Evaluation

Fabio Fassetti, Ilaria Fassetti
DIMES - University of Calabria, Therapeia Rehabilitation Center
fabio.fassetti@unical.it, ilaria.fassetti@gmail.com

Abstract

The aim of evaluating children speech and language is to measure their communication skills. In particular, the speech language pathologist is interested in determining the child's impairments in the areas of *language*, *articulation*, *voice*, *fluency* and *swallowing*. In literature some *standardized tests* have been proposed to assess and screen developmental language impairments but they require manual laborious transcription, annotation and calculation. This work is very time demanding and, also, may introduce several kinds of errors in the evaluation phase and non-uniform evaluations. In order to help therapists, a system performing automated evaluation is proposed. Providing as input the correct sentence and the sentence produced by patients, the technique evaluates the level of the verbal production and returns a score. The main phases of the method concern an ad-hoc transformation of the produced sentence in the reference sentence and in the evaluation of the cost of this transformation. Since the cost function is related to many weights, a learning phase is defined to automatically set such weights.

Keywords: NLP, Language Evaluation, Machine Learning

1. Introduction

The purpose of a speech and language evaluation is to measure communication skills specially of children. The speech language pathologist is interested in determining the child's impairments in the following main areas: *language*, *articulation*, *voice*, *fluency*, *swallowing*. In this work several components of the human language are taken into consideration, namely *phonology*, *morphology* and *syntax*. Phonology is the speech sound system of language and the rules for how speech sounds are combined. Morphology is the study of of word structure and of rules that govern how morphemes (the smallest meaningful units of language) are used in a language. Syntax is the study of rules that govern the ways words combine to form phrases, clauses and sentences.

In this context, *standardized tests*, like the highly diffused tests (Devescovi and Caselli, 2001; Vender et al., 1981; Cianchetti and Fancello, 1997) for Italian speakers, are regularly used for assessing and screening developmental language impairments but require manual laborious transcription, annotation and calculation. This meticulous work often leads to make mistakes and to spend too much time.

Here, a system performing automated evaluation is proposed. It evaluates the different components of the human language (phonology, morphology, syntax) included the determination of phonological processes, both structural and systemic, through a sentence repetition task.

Differently from the system proposed in (Fassetti and Fassetti, 2018), where reading abilities are tested by asking the patient to rewrite read texts, here phonologic and language production abilities are tested and the patient does not directly produce written text but speaks and the therapist transcribe produced sentences to feed the system.

Sentence repetition tasks are used in the evaluation and assessment of children with language difficulties because it examines a wide range of language processing skills and are widely recognized as useful measures of individual differences in language ability and as a means for identifying children with language impairments (Vinther, 2002).

The speech pathologist seats at a table next to or opposite the children and says: "You are going to hear some sentences, you have to repeat exactly what you have just heard". So the clinician, after each sentence, asks to repeat it as close to the original as possible. No feedback is provided during the test but some words of encouragement can be provided if necessary. Children's responses are audio-recorded and later are transcribed. There are no specific sentence requests for applying the proposed technique but it is better to use a set of sentences of variable length, because if sentences are long enough the children can't simply copy them, instead they have recurse to their own grammatical system to be able to repeat the sentences by analyzing and reconstructing their meaning.

From the technical point of view, this work proposes a technique feed by the sentence as pronounced by the therapist, in the following called *reference sentence*, and the sentence as pronounced by the patient, in the following called *produced sentence*. The technique evaluates the verbal production and returns the associated score.

The tackled problem has connections with string edit problems but there are challenging aspects to address, mainly concerning:

- edit distance between sentences;
- in this scenario, swaps are more relevant than substitutions;
- semantic distances can be considered;
- in considering sentence edit distances, matching/non-matching components and erroneously reproduced component frequencies could shed lights on some critical aspects of the verbal productions.

Thus, the aim of the technique can be summarized as follows: (i) transform the produced sentence in the reference sentence, (ii) evaluate the cost of such transformation, (iii) assign a score to the production.

As for step (i), notions of edit operations in sentences are introduced. In the step (ii), the cost of the sequence of needed edit operations is evaluated. Note that in evaluating the cost several weights are to be set. Step (iii) provides

as contribution an automatic approach to set the weights so the compute a score according to expert evaluation.
The sequel of the work is organized as follows. Section 2 describes preliminary notions, introduces edit operations, discusses cost computation and defines the main problem addressed here. Section 3 presents the proposed technique, the main algorithm and relevant functions and describes the phase devoted at learning cost weights. Section 4 discusses preliminary experimental results. Section 5 draws conclusions.

2. Problem Definition

2.1. Preliminaries

A *sentence* S is an ordered sequence $\langle s_1, \ldots, s_n \rangle$ of strings and a sentence $S^* = \langle s_{i_1}, \ldots, s_{i_h} \rangle$ is a *component* of S if, for any $j \in [1, h-1]$, s_{i_j} is the string that immediately precedes $s_{i_{j+1}}$ in S.

Example. Given the sentence
$S = \langle$*"This"*, *"is"*, *"an"*, *"interesting"*, *"paper."*$\rangle$,
$\langle$*"an"*, *"interesting"*$\rangle$ is a component of S, while neither $\langle$*"interesting"*, *"an"*$\rangle$ nor $\langle$*"is"*, *"interesting"*$\rangle$ are components of S since, in the former, *"interesting"* does not precede *"an"* and, in the latter, *"is"* does not *immediately* precede *"interesting"* in S.

A c-sentence $\mathcal{S}$ of a sentence S is an ordered sequence $\langle S_1, \ldots, S_k \rangle$ of components of S.
In the following, with a little abuse of notation, set operators are used with sequences. In particular, $s_i \in S$ denotes that s_i is one of the strings composing the sequence S and, $S \cup s^*$, with $S = \langle s_1, \ldots, s_n \rangle$, denotes the insertion of the string s as $(n+1)$-*th* element of the sequence S, thus $S \cup s^* = \langle s_1, \ldots, s_n, s^* \rangle$.

Proposition 1 *Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence, there are 2^{n-1} c-sentences of S.*

Proof. Since, by definition, the strings composing the components must be adjacent, any c-sentence $\mathcal{S}$ can be obtained through the following generative process:

1. create a first component $S_1 = \langle s_1 \rangle$, set $\mathcal{S}$ to $\langle S_1 \rangle$ and j to 1;
2. for any $s_i \in S$:
   - 2.1. either add s_i to S_j;
   - 2.2. or create a new component $S_{j+1} = \langle s_i \rangle$, set $\mathcal{S}$ to $\mathcal{S} \cup S_{j+1}$ and increment j.

Since, for any $i \in [2, n]$, there are two possible alternatives, the overall number of combinations of steps 2.1 and 2.2 are 2^{n-1}. □

2.2. Edit operations

As stated fundamental task to be accomplished to solve the problem addressed in this work is related to edit operations.

Definition 1 (base edit operation) *Given a sentence $S = \langle s_1, \ldots, s_n \rangle$ a base edit operation ϕ on S is the application of one of the following operators:*

Insertion: *consisting in the insertion of a string $\hat{s}$ in position h, thus obtaining $S' = [s_1, \ldots, s_{h-1}, \hat{s}, s_h, \ldots, s_n]$.*

Deletion: *consisting in the deletion of the h-th string of S, thus obtaining $S' = [s_1, \ldots, s_{h-1}, s_{h+1}, \ldots, s_n]$.*

Substitution: *consisting in the substitution of the h-th string of S, with $h \in [1, n]$, with the string $\hat{s}$ thus obtaining $S' = [s_1, \ldots, s_{h-1}, \hat{s}, s_{h+1}, \ldots, s_n]$.*

Swap: *consisting in the movement of the h-th string of S with the k-th string, without loss of generality assume $h < k$, thus obtaining $S' = [s_1, \ldots, s_{h-1}, s_k, s_{h+1}, \ldots, s_{k-1}, s_h, s_{k+1}, \ldots, s_n]$.*

Movement: *consisting in the movement of the h-th string of S in position k, thus obtaining $S' = [s_1, \ldots, s_{h-1}, \hat{s}, s_{h+1}, \ldots, s_n]$.*

Definition 2 (edit operation sequence) *Given a sentence $\mathcal{S}$, an edit operation sequence Φ on $\mathcal{S}$ is the ordered application of a sequence of basic edit operations on $\mathcal{S}$, $\langle \phi_1, \ldots, \phi_k \rangle$, namely*

$$\Phi(\mathcal{S}) = \mathcal{S}' = \phi_k(\phi_{k-1}(\ldots \phi_1(\mathcal{S}))),$$

where k represents the cardinality *of Φ denoted as $\eta(\Phi)$.*

2.3. Word Relevance

Given a string s of sentence S, $\rho(s)$ denotes the *relevance* of the string s in S. Such value is related to the part of speech of s and to the role played by s in S.
A string s of a sentence S is one of the following part of speech *nouns*, *adjectives*, *articles*, *pronouns*, *verbs*, *adverbs*, *prepositions*, *conjunctions*, *interjections*.
Also, a string s of a sentence S plays one of the following roles *predicates*, *subjects*, *complements*, *attributes* and *appositions*.
A weight w_p is associated with each part of speech p, a weight w_r is associated with each role r and

$$\rho(s) = w_{p_s}^{1-w_{r_s}},$$

is the relevance of s in S, where p_s is the part of speech of s in S and r_s is the role played by s in S.
The weight associated with each part of speech and that associated with each role ranges from 0 to 1 and are automatically set during the learning phase (see Section 3.1).
Note that, as a consequence, also $\rho(s)$ range from 0 to 1.

2.4. Transformation Cost

A cost is associated with each operation and, in particular, let S and S' be two sentences and let $\Phi = \langle \phi_1, \ldots, \phi_k \rangle$ be a sequence of edit operations such that $S' = \Phi(S)$.
The cost $\kappa(\Phi)$ associated with Φ is computed by evaluating for each $i \in [1, k]$ the cost of the operation ϕ_i as follows. Consider five costs: w_{ins} as the weight of insertions, w_{del} as the weight of deletions, w_{sub} as the weight of substitutions, w_{swp} as the weight of swaps and w_{mov} as the weight of movements.

- If ϕ_i consists in inserting the string $\hat{s}$ of S' in S then

$$\kappa(\phi_i) = w_{ins} \cdot \rho(\hat{s}, S'),$$

namely, the cost coincides with the relevance of the omitted word.

ALGORITHM 1: Sentence Evaluation Algorithm

Input: Sentence S_{prd} to be evaluated and reference sentence S_{ref}
Output: Score of S_{prd} with respect to S_{ref}

```
1  begin
2      $Ins, Del, Sub = \textsc{SentenceMatching}(S_{prd}, S_{ref})$;
       // transform tokenized sentences according to sets Ins, Del and Sub
3      let $(S^*_{prd}, S_{ref}) = \textsc{Apply}(S_{prd}, S_{ref}, Ins, Del, Sub)$;
4      $\mathcal{S}_{ref} = \textsc{Modulize}(S_{ref})$;
5      initialize $cost = +\infty$;
6      foreach $S'_{prd} \in S^*_{prd}$ do
7          $\mathcal{S}_{prd}, \mathcal{S}_r f = \textsc{Modularize}(S'_{prd}, S_{ref})$;
8          $Swaps, Moves = \textsc{GreadyAlignment}(\mathcal{S}_{prd}, \mathcal{S}_{ref})$;
           // compute the cost of this solution
9          $cost_{curr} = \textsc{ComputeCost}(Swaps, Moves, Ins, Del, Sub)$;
           // update best result
10         if $cost_{curr} < cost$ then
11             $cost = cost_{curr}$;
12             $\mathcal{S}^{best}_{prd} = \mathcal{S}_{prd}$;
       // return normalized cost
13     return $cost$
```

- If ϕ_i consists in deleting the string $\hat{s}$ from S then

$$\kappa(\phi_i) = w_{del} \cdot w_{p_s},$$

which is a relaxed version of the relevance, where role is not taken into account, since the $\hat{s}$ is not in the reference string S' and then has no role in S'.

- If ϕ_i consists in substituting the string $\hat{s}$ of S' with the string s of S then

$$\kappa(\phi_i) = w_{sub} \cdot d(s, \hat{s})^{1-\rho(\hat{s}, S')},$$

where $d(s, \hat{s})$ denotes the distance between the two strings, in other words, the less the word is relevant the more the cost approaches the distance, the more the word is relevant the higher is the cost. In order to compute the distance $d(\cdot, \cdot)$ we resort to the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970). This algorithm measures the distance by taking into account both the score of *substituting* two characters and the gap penalties divided in *gap opening penalty* (relatively higher) and *gap extension penalty* (relatively lower).

- If ϕ_i consists in swapping the string s and the string $\hat{s}$ of S' then

$$\kappa(\phi_i) = w_{swp} \cdot \left(1 - \sqrt{(1 - \rho(s, S')) \cdot (1 - \rho(\hat{s}, S'))}\right),$$

namely the geometric mean between the relevance of the two strings, so that a relevant string provides higher contribution on the cost.

- If ϕ_i consists in moving the string s of S' then

$$\kappa(\phi_i) = w_{mov} \cdot \rho(s, S').$$

In the following, the cost associated with an edit sequence Φ, whenever the explicitation of costs is needed, is more properly defined as $\kappa(\overrightarrow{\boldsymbol{w}}, \Phi)$ with $\overrightarrow{\boldsymbol{w}} = \langle \overrightarrow{\boldsymbol{w_p}}, \overrightarrow{\boldsymbol{w_r}}, \overrightarrow{\boldsymbol{w_{op}}} \rangle$ where $\overrightarrow{\boldsymbol{w_{op}}} = \langle w_{ins}, w_{del}, w_{sub}, w_{swp}, w_{mov} \rangle$ is the vector of costs associated with operations, $\overrightarrow{\boldsymbol{w_p}}$ is the vector of weights associated with the parts of speech p and $\overrightarrow{\boldsymbol{w_r}}$ is the vector of weights associated with each role r as described in Section 2.3.

2.5. Problem

The main problem addressed in this paper can be formulated as follows.

Definition 3 (sentence matching problem) *Given two sentences $\mathcal{S}'$ and $\mathcal{S}$ the* sentence matching problem *is the problem of finding an edit operation sequence Φ with cardinality $\eta(\Phi)$ such that*

- $\Phi(\mathcal{S}) = \mathcal{S}'$*, and*
- $\nexists \Phi^* \quad s.\ t. \quad \Phi^*(\mathcal{S}) = \mathcal{S}' \quad and \quad \eta(\Phi^*) < \eta(\Phi).$

In other words, the aim is to find the *best* sequence transforming $\mathcal{S}$ in $\mathcal{S}'$, where the *best* is the one with minimum cardinality.
Once the edit operation sequence has been detected, its cost can be computed as described in Section 2.4.

Computational issues.
It is relevant to note that the edit operations defined in Section 2.2 impact on the computational cost needed to solve the problem defined in 3. Indeed, it is known that in case of *Insertions*, *Deletion* and *Substitutions* a polynomial algorithm based on dynamic programming exists (Needleman and Wunsch, 1970) for strings and can be easily adapted to c-sentence. Nevertheless, when *Swaps* and *Movements* are taken into accounts the problem can be proved to become NP-hard (Shapira and Storer, 2007;

Xia, 2008; Davuth and Kim, 2013). Thus, no polynomial algorithm it is known to exactly solve problem presented by Definition 3. In this work, the gready polynomial algorithm GreadyAlignment is proposed and is discussed in subsequent section.

3. Algorithm

This section presents the proposed technique to solve the problem described in Definition 3. The main algorithm is reported as Algorithm 1.

Given the reference sentence S_{ref} and the sentence S_{prd} to be evaluated, the first step consists in mining string to be deleted, inserted or substituted so that the strings composing S_{ref} and the strings composing S_{prd} match.

At this step, the algorithm is able to identify matchings but, in case of duplicated non-matching strings, it is not able to discriminate between them. As for example, let S_{ref} be "*This paper is interesting*" and let S_{prd} be "*This interesting pape is interesting*". Function SentenceMatching identifies the string "*pape*" to be substituted with "*paper*" and the string "*interesting*" to be deleted but does not identify which occurrence of "*interesting*" should be removed.

Thus, subsequent function Apply transforms the sentence S_{prd} according to the results returned by function SentenceMatching and returns a set of possible transformations S^*_{prd}.

Subsequently, for each sentence S'_{prd} in the set S^*_{prd}, the sentences S_{ref} and S'_{prd} are modularized and are aligned by function GreadyAlignment so that the cost can be computed. For computing the cost, equations presented in Section 2.4 are employed, where weights are those returned by the learning phase described in Section 3.1.

The transformation achieving the best alignment is returned.

As far as Function SentenceMatching is concerned, it builds a bipartite graph between strings in S_{ref} and strings in S_{prd}. Thus, the bipartite matching problem is solved to achieve the best matching. Note that the cost of the edge between a string $s_{ref} \in S_{ref}$ and a string $s_{prd} \in S_{prd}$ is related to the edit distance between the two strings and the proximity, namely the matching of neighbors. Moreover, even if this is not currently implemented, the matching problem could take into account also semantic similarity between strings, namely could consider synonyms.

As for Function GreadyAlignment, the algorithm search for the first non-matching module. Assume that the i module s^{ref}_i of the reference sentence $\mathcal{S}_{ref}$ does not match with the i-th module s^{prd}_i of the sentence $\mathcal{S}_{prd}$ to be checked. Since, when this function is called the input c-sentence are composed by the same modules, let s^{prd}_j be the module of $\mathcal{S}_{prd}$ that matches with s^{ref}_i and let s^{ref}_k be the module of $\mathcal{S}_{ref}$ that matches with s^{prd}_i. Thus, the algorithm tries three alternative paths: (i) swap the module s^{prd}_i with the module s^{prd}_j, (ii) move the module s^{prd}_j in position i, (iii) move the module s^{prd}_i in position k. According to the gready strategy the algorithm select the locally best choice.

Example. In order to clarify the three different paths, consider the following.

$$\mathcal{S}_{ref} = \langle \text{“This”}, \text{“paper”}, \text{“is”}, \text{“very”}, \text{“interesting”} \rangle.$$
$$\mathcal{S}_{prd} = \langle \text{“This”}, \text{“interesting”}, \text{“is”}, \text{“paper”}, \text{“very”} \rangle.$$
$$i = 1, \quad j = 3, \quad k = 4$$
$$s^{ref}_i = s^{prd}_j = \text{“paper”}$$
$$s^{prd}_i = s^{ref}_k = \text{“interesting”}$$
$$\textsc{Swap} := \langle \text{“This”}, \text{“paper”}, \text{“is”}, \text{“interesting”}, \text{“very”} \rangle.$$
$$\textsc{Move}_1 := \langle \text{“This”}, \text{“paper”}, \text{“interesting”}, \text{“is”}, \text{“very”} \rangle.$$
$$\textsc{Move}_2 := \langle \text{“This”}, \text{“is”}, \text{“paper”}, \text{“very”}, \text{“interesting”} \rangle.$$

Once the three possible transformations have been built, the associated cost is computed and the locally best transformation is chosen.

Function SentenceMatching($\mathcal{T}_{prd}$,$\mathcal{T}_{ref}$)

```
Input: Sentence $S_{prd}$ to be evaluated and reference
       sentence $S_{ref}$
Output: set of insertions $I$, set of deletions $D$ and set of
        substitutions $S$
begin
    $\mathcal{G} = \texttt{ProxBipartiteMatching}(S_{prd}, S_{ref})$;
    foreach isolated node $n \in \mathcal{G}$ do
        let $s$ be the string associated with $n$;
        if $s \in \mathcal{S}_{ref}$ then
            $I = I \cup \{s\}$;
        else
            $D = D \cup \{s\}$;
    foreach edge $(t_{prd}, t_{ref}) \in \mathcal{G}$ do
        if there is in $\mathcal{G}$ an edge $(t_{prd}, t_{ref})$ then
            $S = S \cup \{(t_{prd}, t_{ref})\}$;
```

3.1. Learning Phase

During this phase, weights for cost computation are learned to provide a score. Firstly, by letting $\overrightarrow{\boldsymbol{w}} = \langle \overrightarrow{\boldsymbol{w_p}}, \overrightarrow{\boldsymbol{w_r}}, \overrightarrow{\boldsymbol{w_{op}}} \rangle$, and *Swaps*, *Moves*, *Ins*, *Del* and *Sub* be the sets to be evaluated, the cost function can be rewritten as the dot product between $\overrightarrow{\boldsymbol{w_{op}}}$ and the vector consisting in the costs associated with sets *Swaps*, *Moves*, *Ins*, *Del* and *Sub* as follows.

$$\begin{aligned}\kappa(\overrightarrow{\boldsymbol{w}}, \Phi) = &\langle w_{ins}, w_{del}, w_{sub}, w_{swp}, w_{mov} \rangle \circ \\ &\Big\langle \sum_{s \in Ins} w_{p_s}^{1-w_{r_s}}, \\ &\sum_{s \in Del} w_{p_s}, \\ &\sum_{s \in Sub} d^{1-w_{p_s}^{1-w_{r_s}}}, \\ &\sum_{s \in Swaps} \left(1 - \sqrt{\left(1 - w_{p_s}^{1-w_{r_s}}\right) \cdot \left(1 - w_{p_{\hat{s}}}^{1-w_{r_{\hat{s}}}}\right)}\right), \\ &\sum_{s \in Moves} w_{p_s}^{1-w_{r_s}} \Big\rangle.\end{aligned} \tag{1}$$

The learning phase can be then performed as described next. Assume that a training set $\mathcal{T}$ of triple $\langle S_{ref}, S_{prd}, \sigma \rangle$

```
Function GreadyAlignment($\mathcal{S}_{prd}$,$\mathcal{S}_{ref}$)
Input:
Output:
begin
    $\Phi = \emptyset$;
    aligned = false;
    $n_{ref}$ = LENGHT($S_{ref}$);
    while not aligned do
        $i = 0$;
        while $i < n_{ref}$ and $S_{ref}[i] == S_{prd}[i]$ do
            $i = i + 1$;
        if $i < n_{ref}$ then
            $S$ = SWAP($S_{prd}, i, S_{ref}$);
            $c[0] = distance(S_{ref}, S) + 1$;
            $S, from_1, to_1$ = MOVE$_1$($S_{prd}, i, S_{ref}$);
            $c[1] = distance(S_{ref}, S) + 1$;
            $S, from_2, to_2$ = MOVE$_2$($S_{prd}, i, S_{ref}$);
            $c[2] = distance(S_{ref}, S) + 1$;
            $op = argmin(c)$;
            switch op do
                case 0 do
                    $\Phi = \Phi \cup \langle x(S_{ref}[i], S_{prd}[i])\rangle$
                case 1 do
                    $Moves = Moves \cup \{\langle order, S_{ref}[i], from_1, to_1\rangle\}$
                case 2 do
                    $Moves = Moves \cup \{\langle order, S_{ref}[i], from_2, to_2\rangle\}$
        else
            aligned = true;
    return Swaps, Moves
```

is available, where an expert has manually evaluated the produced sentence S_{prd} and assign to it the score σ.
By letting $\overrightarrow{\sigma}$ be the vector of the actual scores and by computing the edit sequence Φ for the produced sentence in each triple $t \in \mathcal{T}$, thus obtaining the set $\overrightarrow{\Phi}$,

$$\kappa\left(\overrightarrow{w}, \overrightarrow{\Phi}\right) = \overrightarrow{\sigma}. \qquad (2)$$

Thus, the problem is now to find the weights that minimize the function

$$f(\overrightarrow{w}) = \left\|\kappa\left(\overrightarrow{w}, \overrightarrow{\Phi}\right) - \overrightarrow{\sigma}\right\|_2$$

namely the 2-norm between the difference of the two terms of Equation (2). This can be performed through the stochastic gradient descend, SGD, algorithm thus iteratively updating weights by means of the gradient of the function $f(\overrightarrow{w})$, thus

$$\overrightarrow{w}^{k+1} = \overrightarrow{w}^{k} - \nabla f\left(\overrightarrow{w}^{k}\right).$$

4. Experiments

The whole system has been implemented in Python and a preliminary set of experiments have been conducted. A set of historical evaluations made by experts have been collected. Note that, in this phase, age, sex and other extra information is not taken into account.

To clarify the output of the technique, some notable examples among analyzed real cases are presented. Note that, for the sake of presentation, the sentences have been translate in English even if they are originally in Italian.

Case 1. The reference sentence S_{ref} is

"*At school I learn many things*",

the produced sentence S_{prd} is

"*tool ear hings*".

First of all, the technique individuates the following transformations:

insertions: ⟨"*at*", "*I*", "*many*"⟩,

deletions: none,

substitutions: ⟨⟨"*school*", "*tool*"⟩, ⟨"*learn*", "*ear*"⟩, ⟨"*things*", "*hings*"⟩⟩.

then technique extracts modules obtaining the following c-sentences

$\mathcal{S}_{ref}$ = ⟨⟨"*At*", "*school*", "*I*", "*learn*", "*many*", "*things*"⟩⟩

$\mathcal{S}_{prd}$ = ⟨⟨"***At***", "***school***", "***I***", "***learn***", "***many***", "***things***"⟩⟩

and performs next operations to transform $\mathcal{S}_{prd}$ in $\mathcal{S}_{ref}$:

swaps: none,

movements: none.

Case 2. The reference sentence S_{ref} is

"*If I fall and I hurt, I cry a lot*",

the produced sentence S_{prd} is

"*If I cry and it hurts*".

First of all, the technique individuates the following transformations:

insertions: ⟨"*I*", "*fall*", "*a*", "*lot*"⟩,

deletions: none,

substitutions: ⟨⟨"*I*", "*it*"⟩, ⟨"*hurt*", "*hurts*"⟩⟩.

then extracts modules obtaining the following c-sentences

$$\mathcal{S}_{ref} = \langle\langle\text{"If", "I"}\rangle, \langle\text{"fall"}\rangle, \langle\text{"and", "I", "hurt"}\rangle, \langle\text{"I", "cry", "a", "lot"}\rangle\rangle$$

$$\mathcal{S}_{prd} = \langle\langle\text{"If", "I"}\rangle, \langle\textbf{"fall"}\rangle, \langle\textbf{"I"}, \text{"cry"}, \textbf{"a"}, \textbf{"lot"}\rangle, \langle\text{"and"}, \textbf{"I"}, \textbf{"hurt"}\rangle\rangle$$

and performs next operations to transform $\mathcal{S}_{prd}$ in $\mathcal{S}_{ref}$:

swaps: ⟨⟨"*I*", "*cry*", "*a*", "*lot*"⟩, ⟨"*and*", "*I*", "*hurt*"⟩⟩

movements: none.

Case 3. The reference sentence S_{ref} is

"*The child eats an apple and a banana*",

the produced sentence S_{prd} is

"*Child eass a banana and the fruit*".

First of all, the technique individuates the following transformations:

insertions: ⟨"*an*"⟩,

deletions: none,

substitutions: ⟨⟨"*eats*", "*eass*"⟩, ⟨"*fruit*", "*apple*"⟩⟩.

then extracts modules obtaining the following c-sentences

$$\mathcal{S}_{ref} = \langle\langle\text{"The"}\rangle, \langle\text{"child", "eats"}\rangle, \langle\text{"an", "apple"}\rangle, \langle\text{"and"}\rangle, \langle\text{"a", "banana"}\rangle\rangle$$

$$\mathcal{S}_{prd} = \langle\langle\text{"Child", "eats"}\rangle, \langle\text{"a", "banana"}\rangle, \langle\text{"and"}\rangle, \langle\text{"the"}\rangle, \langle\textbf{"an"}, \textbf{"apple"}\rangle\rangle$$

and performs next operations to transform $\mathcal{S}_{prd}$ in $\mathcal{S}_{ref}$:

swaps: ⟨⟨"*a*", "*banana*"⟩, ⟨"*an*", "*apple*"⟩⟩

movements: ⟨⟨"*the*"⟩⟩.

As for the learning phase, the input set has been split in training set and test set. Results are reported in Figure 1. The y axis reports accuracy while the x axis reports the iterations for the SGD algorithm. The number of iterations have been set to 4000 and results are reported each 200 iterations.

Note that the learning phase is accomplished just at building time, since once the weights have been learned the score of sentences can be directly computed by Equation (1).

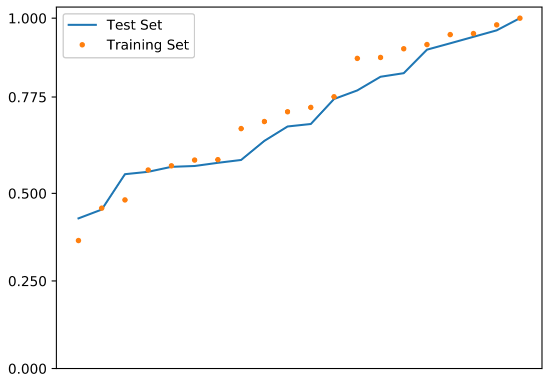

Figure 1: Accuracy of the technique

5. Conclusions

The work presents a preliminary study for automatic evaluation of language. The aim is to help specialist in obtaining an evaluation suggestion together with the critical components. Experimental results show that the approach is promising.

6. Bibliographical References

Cianchetti, C. and Fancello, G. S. (1997). *Test TVL. Test di valutazione del linguaggio*. Erickson.

Davuth, N. and Kim, S.-R. (2013). A better edit distance measure allowing for block swaps. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, RACS '13, pages 7–11. ACM.

Devescovi, A. and Caselli, M. (2001). Una prova di ripetizione di frasi per la valutazione del primo sviluppo grammaticale. 3:341–364.

Fassetti, F. and Fassetti, I. (2018). Mining string patterns for individuating reading pathologies. In *Proceedings of the ACM Symposium on Applied Computing, Pau, France, April 9-13, 2018*, pages 1–5.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453.

Shapira, D. and Storer, J. A. (2007). Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380 – 392.

Vender, C., Borgia, R., Bruno, S. C., Freo, P., and Zardini, G. (1981). Un test di ripetizione di frasi. analisi delle performances in bambini normali. *Neuropsichiatria Infantile*, 243:819–831.

Vinther, T. (2002). Elicited imitation: a brief overview. *International Journal of Applied Linguistics*, 12(1):54–73.

Xia, T. (2008). An edit distance algorithm with block swap. In *International Conference for Young Computer Scientists*, pages 54–59.