Automagically Encoding Adverse Drug Reactions in MedDRA

Margherita Zorzi, Carlo Combi
Department of Computer Science
University of Verona, Italy
{margherita.zorzi|carlo.combi}@univr.it

Riccardo Lora, Marco Pagliarini, Ugo Moretti
Department of Public Health and Community Medicine
University of Verona, Italy
{riccardo.lora|marco.pagliarini|ugo.moretti}@univr.it

***Abstract*—Pharmacovigilance is the field of science devoted to the collection, analysis, and prevention of Adverse Drug Reactions (ADRs). Efficient strategies for the extraction of information about ADRs from free text sources are essential to support the important task of detecting and classifying unexpected pathologies, possibly related to (therapy-related) drug use. Narrative ADR descriptions may be collected in different ways, e.g., either by monitoring social networks or through the so called "spontaneous reporting, the main method pharmacovigilance adopts in order to identify ADRs. The encoding of free-text ADR descriptions according to MedDRA standard terminology is central for report analysis. It is a complex work, which has to be manually implemented by the pharmacovigilance experts. The manual encoding is expensive (in terms of time). Moreover, a problem about the accuracy of the encoding may occur, since the number of reports is growing up day by day. In this paper, we propose MagiCoder, an efficient Natural Language Processing algorithm able to automatically derive MedDRA terminologies from free-text ADR descriptions. MagiCoder is part of VigiWork, a web application for online ADR reporting and analysis. From a practical point of view, MagiCoder reduces the encoding time of ADR reports. Pharmacologists have simply to review and validate the MedDRA terms proposed by MagiCoder, instead of choosing the right terms among the 70K terms of MedDRA. Such improvement in the efficiency of pharmacologists' work has a relevant impact also on the quality of the following data analysis.**

Our proposal is based on a general approach, not depending on the considered language. Indeed, we developed MagiCoder for the Italian pharmacovigilance language, but preliminarily analyses show that it is robust to language and dictionary changes.

***Index Terms*—pharmacovigilance; natural language processing; adverse reaction entry.**

I. Introduction

Pharmacovigilance includes all activities aimed to systematically study risks and benefits related to the correct use of marketed drugs. The development of a new drug, which begins with the production and ends with the commercialization of a drug, considers both pre-clinical studies (usually tests on animals) and clinical studies (tests on patients). After these phases, a pharmaceutical company can require the authorization for the commercialization of the new drug. Notwithstanding, whereas at this stage drug benefits are well-know, results about drug safety are not conclusive [1]. The pre-marketing tasks cited above have some limitations: they involve a small number of patients; they exclude relevant subgroups of population such as children and elders; the experimentation period is relatively short, less than two years; the experimentation does not deal with possibly concomitant pathologies, or with the concurrent use of other drugs. For all these reasons, non-common Adverse Drug Reactions (ADRs), such as slowly-developing pathologies (e.g., carcinogenesis) or pathologies related to specific groups of patients, cannot be discovered before the commercialization. It may happen that drugs are withdrawn from the market after the detection of unexpected collateral effects. Thus, it stands to reason that the control of ADRs is a necessity, considering the mass production of drugs. As a consequence, pharmacovigilance plays a crucial role in human healthcare improvement [1].

Spontaneous reporting is the main method pharmacovigilance adopts, in order to identify adverse drug reactions. Through spontaneous reporting, health care professionals, patients, and pharmaceutical companies can voluntarily send information about suspected ADRs to the national regulatory authority [1]. The spontaneous reporting is an important activity. It provides pharmacologists and regulatory authorites with early alerts, by considering every drug on the market and every patient category.

The Italian system of pharmacovigilance requires that in each local health structure there is a qualified person responsible for pharmacovigilance. Her/his assignment is to collect reports of suspected ADRs and to send them to the National Network of Pharmacovigilance (RNF) within seven days[2]. Once reports have been notified and sent to RNF, currently through a web application, they are analysed by both local pharmacovigilance centres and by the Drug Italian Agency (AIFA). Subsequently, they are sent to Eudravigilance [2] and to VigiBase [3] (the european and the worldwide pharmacovigilance network, RNF is part of, respectively). In general, spontaneous ADR reports are filled by health care professionals (medical specialists, general practitioners, nurses, and so on), but also by citizens. In the last years, Italian ADR reports have grown exponentially, going from approximately ten thousand in 2006 to around sixty thousand in 2014, as shown in Figure 1.

Since the post-marketing surveillance of drugs is of

[1] in Italy, the Drug Italian Agency AIFA –Agenzia Italiana del FArmaco, http://www.agenziafarmaco.gov.it/

[2] According to the Italian Law, Art. 132 of Legislative Decree Number 219 of 04/24/2006.

978-1-4673-9548-9/15 $31.00 © 2015 IEEE
DOI 10.1109/ICHI.2015.18

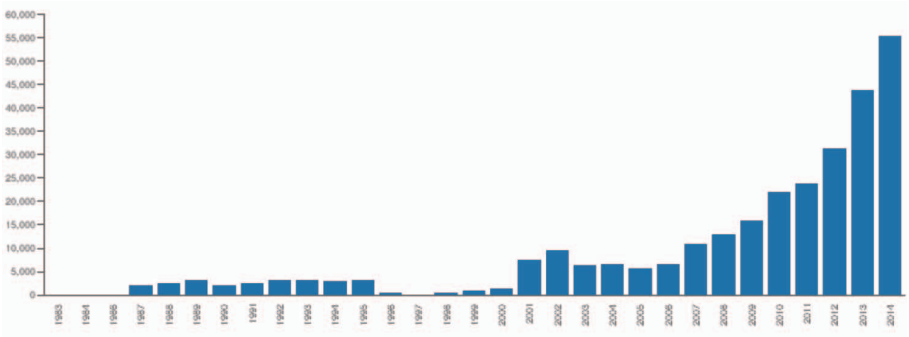

Fig. 1. The yearly increasing number of reports about suspected adverse reactions induced by drugs in Italy.

paramount importance, such an increase is certainly positive. At the same time, the manual review of reports became difficult and often unbearable both by people responsible for pharmacovigilance and by regional centres. Indeed, each report must be checked, in order to control its quality; it is consequently encoded and transferred to RNF via "copy by hand" (actually, a printed copy).

Recently, to increase the efficiency in collecting and managing ADR reports, a web application, called `VigiWork`, has been designed and implemented for the Italian pharmacovigilance (at https://vigiwork.vigifarmaco.it/). Through `VigiWork`, a spontaneous report can be inserted online both by healthcare professionals and by citizens (through different forms), as anonymous or registered users. `VigiWork` is user-friendly. The user is guided in compiling the report, since it has to be filled step-by-step (each phase corresponds to a different report section, i.e., "Patient", "Adverse Drug Reaction", "Drug Treatments" and "Reporter", respectively). Inserted data are then validated, since a report can be successfully sent only after completing the correct sequence of steps.

`VigiWork` is also useful for pharmacovigilance supervisors. Indeed, `VigiWork` reports are high-quality documents, since they are automatically validated (the presence, the format, and the consistency of data are validated at the filling time). As a consequence, they are easier to review (especially with respect to printed reports). Moreover, thanks to `VigiWork`, a pharmacologist can send reports to RNF by simply pressing a button, after reviewing it.

Online reports have grown up to become the 30% of the total number of Italian reports. As expected, it has been possible to observe that the average time between the dispatch of online reports and the insertion into RNF is sensibly shorter with respect to the the insertion from printed reports. Notwithstanding, there is an operation which still requires the manual work of people responsible for Pharmacovigilance also for online report revisions: the encoding in `MedDRA` terminology of the free text, through which the reporter describes one or more adverse drug reactions. The description of a suspected ADR through narrative text could seem redundant/useless. Indeed, one could reasonably imagine sound solutions based either on an autocompletion form or on a menu with `MedDRA` terms. In these solutions, the description of ADRs would be directly encoded by the reporter and no expert work for `MedDRA` terminology extraction would be required. However, such solutions are not completely suited for the pharmacovigilance domain and the narrative description of ADRs remains a desirable feature, for at least two reasons. First, the description of an ADR by means of one of the seventy thousand `MedDRA` terms is a complex task. In most cases, the reporter which points out the adverse reaction is not an expert in `MedDRA` terminology. This holds in particular for citizens, but it is still valid for several professionals. Thus, describing ADRs by means of natural language sentences is simpler. Second, the choice of the suitable term(s) from a given list or from an autocompletion field can influence the reporter and limit her/his expressiveness. As a consequence, the *quality* of the description would be also in this case undermined. Therefore, `VigiWork` offers a free-text form for specifying and ADR with all the possible details, without any restriction about the content or limits to the length of the written text. Consequently, `MedDRA` encoding has then to be manually implemented by qualified people responsible for pharmacovigilance, before the transmission to RNF. As this work is expensive in terms of time and attention required, a problem about the accuracy of the encoding may occur given the continuous growing of the number of reports.

According to the described scenario, in this paper we propose MagiCoder, a natural language processing (NLP) [4] algorithm, which automatically assigns one or more `MedDRA` term codes to each narrative ADR description in the online reports collected by `VigiWork`.

The paper is organized as follows. In Section II we provide some background notions and we discuss related work. In Section III we present the algorithm MagiCoder, by providing both a qualitative description and the pseudocode. In Section IV we spend some words about the user interface, we explain the benchmark we developed to test MagiCoder performances and we discuss first results. Finally, in Section V we discuss the main features of our work and sketch some future research lines.

II. Background and Related Work

A. Natural Language Processing and Text Mining in Medicine

Automatic detection of adverse drug reactions from text recently received an increasing interest in pharmacovigilance research. Narrative descriptions of ADRs come from heterogeneous sources: spontaneous reporting, Electronic Health Records, Clinical Reports, and social media. In [5]–[9] some NLP approaches have been proposed for the extraction of ADRs from text. In [10], the authors collect narrative discharge summaries from the Clinical Information System at New York Presbyterian Hospital. MedLEE, an NLP system, is applied to this collection, to identify medication events and entities, which could be potential adverse drug events. Co-occurrence statistics with adjusted volume tests were used to detect associations between the two types of entities, to calculate the strengths of the associations, and to determine their cutoff thresholds. In [11], the authors report on the adaptation of a machine learning-based system for the identification and extraction of ADRs in case reports. The role of NLP approaches in optimised machine learning algorithms is also explored in [12], where the authors address the problem of automatic detection of ADR assertive text segments from distinct sources, focusing on data posted by users on social media (Twitter and DailyStrenght, a health care oriented social media). Existing methodologies for NLP are discussed; an experimental comparison between NLP-based machine learning algorithms over data sets from different sources has been proposed. Moreover, the authors address the issue of data imbalance for ADR description task. In [13] the authors propose to use association mining and Proportional Reporting Ratios (PRR, a well-know pharmacovigilance statistical index) to mine the associations between drugs and adverse reactions from the user contributed content in social media. In order to extract adverse reactions from on line text (from health care communities), the authors apply the Consumer Health Vocabulary (at http://www.consumerhealthvocab.org) to generate ADR lexicon. ADR lexicon is a computerized collection of health expressions derived from actual consumer utterances (authored by consumers), linked to professional concepts and reviewed and validated by professionals and consumers. Narrative text is preprocessed following standard NLP techniques (such as stop word removal, see Section III-A). An experiment using ten drugs and five adverse drug reactions is proposed. The Food and Drug Administration alerts are used as the gold standard, to test the performance of the proposed techniques. The authors developed algorithms to identify ADRs from threads of drugs, and implemented association mining to calculate leverage and lift for each possible pair of drugs and adverse reactions in the dataset. At the same time, PRR is also calculated.

Other interesting papers about pharmacovigilance and machine learning or data mining are, e.g., [14] and [15]. In [16] a text extraction tool is implemented on the .NET platform with functionalities for preprocessing text (removal of stop words, Porter stemming and use of synonyms) and matching medical terms using permutations of words and spelling variations (Soundex, Levenshtein distance and Longest common subsequence distance [17]). Its performance has been evaluated on both manually extracted medical terms from summaries of product characteristics and unstructured adverse effect texts from Martindale (i.e. a medical reference for information about drugs and medicines) using the WHO-ART and MedDRA medical term dictionaries. A lot of linguistic features have been considered and a careful analysis of performances has been provided.

MedDRA Level	MedDRA Term
SOC	Skin disorders
HLGT	Epidermal conditions
HLT	Dermatitis and Eczema
PT	Asteatotic Eczema
LLT	Itch

TABLE I
MedDRA Hierarchy - an Example

B. MedDRA Dictionary

The Medical Dictionary for Regulatory Activities (MedDRA) is a medical terminology used to classify adverse event information associated with the use of biopharmaceuticals and other medical products (e.g., medical devices and vaccines). Coding these data to a standard set of MedDRA terms allows health authorities and the biopharmaceutical industry to exchange and analyze data related to the safe use of medical products [18]. It has been developed by the International Conference on Harmonization (ICH); it belongs to the International Federation of Pharmaceutical Manufacturers and Associations (IFPMA); it is controlled and periodically revised by the MedDRA Mainteinance And Service Organization (MSSO). MedDRA is available for eleven European languages and for Chinese and Japanese too. It is updated twice a year (in March and in September), following a collaboration-based approach: everyone can propose new reasonable updates or changes (as effects of events as the onset of new pathologies) and a team of experts eventually decides about the publication of the updates. MedDRA terms are organised into a hierarchy: the SOC (System Organ Classes) level includes the most general terms; the LLT (Low Level Terms) level includes more specific terminologies; between SOC and LLT there are three intermediate levels (HLGT, HLT and PT).

Table I shows an example of the hierarchy: the reaction *Itch* is described starting from *Skin disorders*, *Epidermal conditions*, *Dermatitis and Eczem*, and *Asteatotic Eczema*. Preferred Terms are Low Level Terms chosen to be the representative of a group of terms. It should be stressed that the hierarchy is multiaxial: for example, a PT (Preferred Term) can be grouped in one or more HLT (High Level Term), but it belongs to only one primary SOC (System Organ Class) term.

The encoding of ADRs through MedDRA is extremely important for report analysis as for a prompt detection of problems related to drug-based treatments. Thanks to MedDRA it is possible to group similar/analogous cases described in

different ways (e.g. by synonyms) or with different details/levels of abstraction.

III. MagiCoder: an Algorithm for ADR Automatic Encoding

A natural language ADR description is a completely free text. The user has no limitations, she/he can potentially write everything: a number of online ADR descriptions actually contain information not directly related to drug effects. An NLP software has to face and solve many issues: trivial orthographical errors; the use of singular versus plural nouns; the so called "false positives" i.e. syntactically retrieved inappropriate results, which are closely resembling correct solutions; the structure of the sentence, i.e. the way an assertion is built up in a given language. Also the "intelligent" detection of linguistic connectives is a crucial issue. For example, the presence of a negation can potentially change the overall meaning of a description.

In general, a satisfactory automatization of human reasoning and work is a subtle task, and the uncontrolled extension of the dictionary with auxiliary synonymous or the naive ad-hoc management of particular cases can limit the efficiency of the algorithm. For these reasons, we carefully designed MagiCoder, even through a side-by-side collaboration between pharmacologists and computer scientists, in order to yield an efficient tool, capable to really support pharmacovigilance activities.

In literature, several NLP algorithms still exists, and several interesting approaches (such as the so called morpho-analysis of natural language) have been studied and proposed [4], [19], [20]. According to the described pharmacovigilance domain, we considered algorithms for the morpho-analysis and the part-of-speech extraction techniques [4], [19] too powerful and general purpose for the first solution to our problem.

Thus, we decided to design and develop an ad-hoc algorithm for the problem we are facing, namely that of deriving `MedDRA` terms from narrative text and mapping segments of text in effective `LLT` terms. This task has to be done in a very feasible time (we want that each interaction user/MagiCoder requires less than a second) and the solution offered to the expert has to be readable and useful. Therefore, we decided to ignore the *structure* of the narrative description and address the issue in a simpler way. Main features of MagiCoder can be summarized as follows:

- it requires *a single linear scan of the narrative description*: as a consequence, our solution is particularly efficient in terms of computational complexity;
- it has been designed and developed for the specific problem of mapping Italian text to `MedDRA` dictionary, but we claim the way MagiCoder has been developed is sound with respect to Language and Dictionary changes.
- the current version of MagiCoder is only based on the pure syntactical recognition of the text and it does not exploit any heuristic or external synonym dictionary; as it will be discussed in Section IV, experimental results are encouraging and we empirically observed that the use of an external dictionary produces a relevant improvement of performances.

A. MagiCoder: Overview

The main idea of MagiCoder is that a single linear scan of the free-text is sufficient, in order to recognize `MedDRA` terms.

From an abstract point of view, we try to recognize, in the narrative description, *single words* belonging to `LLT`terms, which do not necessarily occupy consecutive positions in the description. This way, we try to *reconstruct* `MedDRA` terms, taking into account the fact that in a description the reporter can permute or omit words. As we will show, MagiCoder has not to deal with computationally expensive tasks, such as taking into account subroutines for permutations and combinations of words (as, for example, in [16]).

We can distinguish five phases in the procedure, we will discuss in detail in the following:

1) Preprocessing of the original text;
2) Definition of ad-hoc data structures;
3) Word-by-word linear scan of the description and "voting task";
4) Weight calculation and sorting of voted terms;
5) Winning terms release.

1) Preprocessing of the original ADR description: Given a natural language ADR description, the text has to be preprocessed in order to perform an efficient computation. We adopt well-know techniques such as tokenization [21], where a phrase is reduced to *tokens*, i.e. syntactical units, which often, as in our case, correspond to words. A tokenized text can be easily manipulated as an enumerable object, e.g. an array. A *stop word* is a word which can be considered irrelevant for the text analysis (e.g. an article or an interjection). In this first release of our software we decided to not take into account *connectives*, e.g. conjunctions, disjunctions, negations. Once one has defined the set of the stop words, the original text is cleaned from such irrelevant words.

A fruitful preliminary work is the extraction of the corresponding *stemmed* version from the original tokenized (and stop-word free) text. Stemming is a linguistic technique that, given a word, reduces it to a particular kind of root form [21]. It is useful in text analysis, in order to avoid problems such as bad word recognition due to singular/plural forms (e.g., hand/hands). Stemming is also potentially harmful, since it can generate the so called "false positives" terms. A meaningful example can be found in Italian language. The plural of the word *mano* (in English, *hand*) is *mani* (in English, *hands*), and their stemmed radix is *man*, which is also the stemmed version of *mania* (in English, *mania*).

2) Definition of ad hoc data structures: The algorithm proceeds with a word-by-word comparison. We iterate on the preprocessed text and we test if a single word w (a token) occurs into one or many `LLT` terms.

In order to efficiently test if a token belongs to one or more `LLT` terms, we need to create a further level of the `MedDRA` dictionary. The `LLT` level of `MedDRA` is actually a set of *phrases*, i.e. *sequences of words*. By scanning these

sequences, we built a *meta-dictionary* of all the words which compose LLT terms. As we will describe in Section III-B, in $O(mk)$ time units (where m and k are the cardinality of the set of LLT terms and the length of the longest LLT term in MedDRA, respectively) we can build a hash table having all the words occurring in MedDRA as keys, where the value associated to key w_i contains information about the set of LLTs containing w_i. This way, we can verify the presence in MedDRA of a word w encountered in the ADR description in constant time. We call this meta-dictionary DictByWord. We build a meta dictionary also from a stemmed version of MedDRA, to verify the presence of stemmed descriptions. We call it DictByWordStem.

Also the MedDRA dictionary is loaded for the computation into hash tables and, in general, all our main data structures are dictionaries. We aim to stress that, to retain efficiency, we preferred exact string matching with respect to approximate string matching, when looking for a word into the meta dictionary. Approximate string matching would allow us to retrieve terms that would be lost in exact string matching (e.g., we could recognize misspelled words in the ADR description), but it would worsen the performances of the text recognition tool, since direct access to the dictionary would not be possible. We discuss the problem of addressing orthographical errors in Section V.

3) Word-by-word linear scan of the description and voting task: Algorithm MagiCoder scans the text word-by-word (remember that each word corresponds to a token) one time and performs a "voting task": at the i-th step, it marks (i.e. "votes"), with index i each LLT term t containing the current (i-th) word of the ADR description. Moreover, it keeps track of the position where the i-th word occurs in LLT terms. MagiCoder tries to find a word match both for the exact and the stemmed version of the meta dictionary and keeps track of the kind of match it has eventually found. It updates a flag, initially set to 0, if at least a stemmed matching is found. If a word w has been exactly recognized in a term t, the match between the stemmed versions of w and t is not considered. At the end of the scan, the procedure has built a sub-dictionary containing only terms "voted" at least by a word. We will call $\mathsf{Voted}_{\texttt{LLT}}$ the sub dictionary of voted terms.

Each selected term t is equipped with two auxiliary data structures, containing, respectively:

1) the positions of the voting words in the ADR description; we will call voters_t this sequence of indexes;
2) the positions of the voted words in the MedDRA term t; we will call voted_t this sequence of indexes.

Moreover, we endow each selected term with a third structure that will contain the sorting criteria we define below; we will call it $\mathsf{weights}_t$.

Let us now introduce some notations we will use in the following. We denote as $t.size$ the function that, given a LLT term t, returns the number of words contained in t. We denote as $\mathsf{voters}_t.length$ (resp. $\mathsf{voted}_t.length$) the function that returns the number of indexes belonging to voters_t (resp. voted_t). We denote as $\mathsf{voters}_t.min$ and $\mathsf{voters}_t.max$ the functions that return the maximum and the minimum indexes in voters_t, respectively.

4) Weight calculation and sorting: After the voting task, selected terms have to be ordered. Notice that a purely syntactical recognition of words in LLT terms potentially generates a large number of voted terms. So we have to: i) filter a subset of highly feasible solutions; ii) choose a good final selection criteria (this will be discuss in Section III-A5).

To this end, we define five criteria as "weights" to assign to voted terms. In the following, $\frac{1}{t.size}$ is a normalization factor (w.r.t. the length, in terms of words, of the LLT term t). For the first four criteria the optimum value is 0.

Criterion one: Coverage

First, we consider how many words of each voted LLT term have been recognized.

$$\mathsf{C}_1(t) = \frac{t.size - \mathsf{voters}_t.length}{t.size}$$

Criterion two: Type of Coverage

The algorithm considers whether a perfect matching has been performed using or not stemmed words. $\mathsf{C}_2(\cdot)$ is simply a flag. $\mathsf{C}_2(t)$ holds if stemming has been used at least once in the voting procedure.

Criterion three: Coverage Distance

The use of stemming allows one to find a number of (otherwise lost) matches. As side effects, we often obtain a quite large set of joint winner candidate terms. In this phase, we introduce a string distance comparison between recognized words in the original text and retrieved LLT terms. Among the possible string metrics, we use the so called pair distance [22], which is robust with respect to word permutation. So,

$$\mathsf{C}_3(t) = pair(t, \bar{t})$$

where $pair(s, r)$ is the pair distance function (between strings s and r) and $\bar{t}$ is the term "rebuilt" from the words in ADR description corresponding to indexes in voters_t.

Criterion four: Coverage Density

We want to estimate how an LLT term has been covered.

$$\mathsf{C}_4(t) = \frac{(\mathsf{voters}_t.max - \mathsf{voters}_t.min) + 1}{t.size}$$

The intuitive meaning of the criterion is to quantify the "quality" of the coverage. If an LLT term has been covered by nearby words, it will be considered a good candidate for the solution. This Criterion has to be carefully implemented, taking into account possible duplicate words.

Criterion Five: Coverage Distribution

After the evaluation and the sorting by the criteria described above, good solutions are sorted in the first positions. We add a further criterion, the only one based on the assumptions we made about the

structure of (Italian) sentences. The following formula simply sums the index of the covered words for $t \in \mathsf{Voted}_{\texttt{LLT}}$:

$$\mathsf{C}_5(t) = \sum_{i=0}^{\mathsf{voted}_t.length-1} \mathsf{voted}_t[i]$$

If $\mathsf{C}_5(t)$ is small, it means that words in the first positions of term t have been covered. We introduce this criterion to discriminate between possibly joint winning terms. Indeed, an Italian medical description of a pathology has frequently the following shape: *name of the pathology*+*"location" or adjective*. Intuitively, we privilege terms, for which the recognized word(s) are probably the one(s) describing the pathology.

After computing (and storing) the weights related to the above criteria, for each voted term t we have the structure $\mathsf{weights}_t = [\mathsf{C}_1(t), \mathsf{C}_2(t), \mathsf{C}_3(t), \mathsf{C}_4(t), \mathsf{C}_5(t)]$, containing the weights corresponding to the five criteria.

We finally proceed by ordering voted terms by multiple value sorting (on elements in $\mathsf{weights}_t$, $t \in \mathsf{Voted}_{\texttt{LLT}}$) and call $\mathsf{SortedVoted}_{\mathsf{LLT}}$ the sorted dictionary.

5) Release of winning terms: In order to provide an effective support to pharmacovigilance experts' work, it is important to offer, among the "good" solutions of the algorithm (well positioned LLT terms in sorted output), a small subset of candidate solutions, typically from one to six terms recognized as the best match of the ADR description. We will call $\mathsf{Selected}_{\texttt{LLT}}$ such a set. This is a subtle task. As previously said, the pure syntactical recognition of MedDRA terms into a free-text generates a possibly large set of syntactically good results. Therefore, the releasing strategy has to be carefully designed. The main idea is to select and return a subset of voted terms which "covers" the ADR description. We iterate on the ordered dictionary and for each $t \in \mathsf{SortedVoted}_{\mathsf{LLT}}$ we iterate on voters_t and we select t if the following conditions hold: 1) t does not belong to $\mathsf{Selected}_{\texttt{LLT}}$; 2) t is not a prefix of another selected term $t' \in \mathsf{Voted}_{\texttt{LLT}}$; 3) for any $w_i \in \mathsf{voters}_t$, w_i has not been covered or w_i has not been exactly covered (only the stemmed version has been eventually recognized) or t has been "voted" without stemming. [3]. We keep track of the words of the ADR description covered by the selection. We consider all the sorted dictionary $\mathsf{SortedVoted}_{\mathsf{LLT}}$, but the selection actually ends when all the words of the description have been covered. The user interface (UI) of VigiWork (described in Section IV) further filters winning terms, by releasing from zero up to six solutions.

In MagiCoder we do not need to consider ad hoc subroutines to address permutations and combinations of words (as it is done, for example, in [16]). In Natural Language Processing, permutations and combinations of words are important, since in spoken language the order of words can change w.r.t. the formal structure of the sentences. Moreover, some words can be omitted, while the sentence still retains the same meaning. These aspects come for free from our voting procedure: after the scan, we retrieve the information that *a set of words covers a term* $t \in \mathsf{Voted}_{\texttt{LLT}}$, but *the order between words does not matter*.

[3] In the implementation we add also the following thresholds: we choose only terms t such that $\mathsf{C}_3(t) < 0.5$ and $\mathsf{C}_5(t) < 3$. We extracted these threshold by means of some empirical tests. We plain to eventually adjust them after some further performance tests

B. MagiCoder: the Algorithm

Figure 2 depicts the pseudocode of MagiCoder. Here we provide a high-level description of the procedure. We represent dictionaries either as sets of words or as sets of functions. As usually, the formula $w \in \mathsf{LLTDict}$ means "word w belongs to dictionary DictByWord (similarly for $\mathsf{DictByWordStem}$, $\mathsf{Voted}_{\texttt{LLT}}$, $\mathsf{SortedVoted}_{\mathsf{LLT}}$, $\mathsf{Selected}_{\texttt{LLT}}$). Procedure $Preprocessing$ takes the narrative description, puts it into an array of words and performs tokenization and stop-word removal. Procedures $CreateMetaDict$ and $CreateMetaDictStem$ get the dictionary of LLT terms and create a dictionary of *words* and of their stemmed versions, respectively, which belong to LLT terms, retaining the information about the set of terms containing each word. By the functional notation $\mathsf{DictByWord}(j)$ (similarly, $\mathsf{DictByWordStem}(j)$), we refer to the set of LLT terms containing the word j (or its stemmed version). Function $stem(i)$ returns the stemmed version of word i. Function $indx_t(j)$ returns the position of word j in term t. $stem\_usage_t$ is a flag, initially set to 0, which holds 1 if at least a stemmed matching with the MedDRA term t is found. adr_clear, voters_t, voted_t are arrays and $\mathsf{add}[A, l]$ denotes the insertion of l in array A, where l is an element or a sequence of elements.

C_i $(i = 1, \ldots, 5)$ are criteria defined in Section III-A4 and procedure $sortby(v_1, \ldots, v_k)$ performs the multi-value sorting of values $v_1, \ldots, v_k$. Procedure $prefix(S, t)$, where S is a set of terms and t is a term, tests whether t (considered as a string) is *prefix* of a term in S. Dually, procedure $remove\_prefix(S, t)$ tests if in S there are one or more prefixes of t, and eventually remove them from S. Function $mark(j)$ specifies whether a word j has been already covered in the (partial) solution during the term release: $mark(j)$ holds 1 if j has been covered (with or without stemming) and it holds 0 otherwise. We assume that before starting the final phase of building the solution (i.e., the returned set of LLT terms), $mark(j) = 0$ for any word j belonging to the description.

Let us now conclude this section by sketching the analysis of computational complexity of MagiCoder. Let n be the input size (the length, in terms of words, of the ADR description). Let m be the cardinality of the medical dictionary (i.e., the number of terms). Moreover, let m' be the number of words occurring in the dictionary and let k be the length of the longest $t \in$ LLT. For MedDRA, we have around 70K terms and 20K words. Notice that k is a very small constant. We assume that all update operations on auxiliary data structures require constant time. Building meta-dictionaries DictByWord

```
Procedure MagiCoder(D description, LLTDict dictionary)
Input: D: the narrative description; LLTDict: a data structure containing the LLT terms of MedDRA dictionary
Output: a set of LLT ordered terms
DictByWord = CreateMetaDict(LLTDict);
DictByWordStem = CreateStemMetaDict(LLTDict);
adr_clear = Preprocessing(D);
adr_length = adr_clear.length;
foreach (i ∈ [0, adr_length − 1]) do
    /* test whether the current word belongs to MedDRA                                   */
    if adr_clear[i] ∈ DictByWord then
        /* for each term containing the word                                             */
        foreach (t ∈ DictByWord(adr_clear[i])) do
            /* keep track of the index of the voting word                                */
            add[voters_t,i];
            /* keep track of the index of the recognized word in t                       */
            add[voted_t, indx_t(adr_clear[i])];
            Voted_LLT = Voted_LLT ∪t;
    /* test if the current (stemmed) word belongs the stemmed MedDRA                     */
    if stem(adr_clear[i]) ∈ DictByWordStem then
        foreach t ∈ DictByWordStem(stem(adr_clear[i])) do
            /* test if the current term has not been exactly voted by the same word      */
            if i ∉ voters_t then
                add[voters_t, i];
                add[voted_t, indx_t(adr_clear[i])];
                /* keep track that t has been covered by a stemmed word                  */
                stem_usage_t = true;
            Voted_LLT = Voted_LLT ∪ t
/* for each voted term, calculate the five weights of the corresponding criteria         */
foreach t ∈ Voted_LLT do
    add[weights_t, C1(t), C2(t), C3(t), C4(t), C5(t)]
/* multiple value sorting of the voted terms                                             */
SortedVoted_LLT = Voted_LLT.sortby(C1, C2, C3, C4, C5);
foreach t ∈ SortedVoted_LLT do
    foreach index ∈ voters_t do
        /* select a term t if its i-th voting word has not been covered or if its i-th voting word has been perfectly recognized in t
           and if t is not prefix of another already selected terms                       */
        if ((stem_usage_t = false OR (mark(adr_clear(index))=0)) AND t ∉ Selected_LLT AND prefix(Selected_LLT,t)=false) then
            mark(adr_clear(index))=1;
            /* remove from the selected term set all terms which are prefix of t         */
            Selected_LLT = remove_prefix(Selected_LLT,t);
            Selected_LLT = Selected_LLT∪ t
return Selected_LLT
```

Fig. 2. Pseudocode of MagiCoder

and DictByWordstems requires $O(mk)$ time units. In fact, the simplest procedure to build the hash table is to scan the LLT dictionary and, for each term t, to verify for each word w belonging to t whether w is a key in the hash table (this can be done in constant time). If w is a key, then we have to update the values associated to w, i.e., we add t to the set of terms containing w. Otherwise, we add the new key w and the associated term t to the hash table. Therefore, it can be easily verified that the voting procedure requires in the worst case $O(nm)$ steps, when a word belongs to all the LLT terms. The computation of criteria-related weights requires $O(n)$ time units; the complexity of multi-value sorting can be approximated to $O(nlogn)$ time units (since the number of the criteria-related weights involved in the multi-sorting is fixed to be 5). Finally, deriving the best solutions actually requires $O(nl)$ steps.

The computational complexity of MagiCoder is likely to be lower than that of the tool proposed in [16]. Indeed, in [16] the author describes a sophisticated procedure which considers also approximate string matching. This feature does not allow constant time search for text-dictionary matches (i.e., it is not always possible to exploit direct data access through optimal data structures, such as hash tables). Moreover, explicitly considering word permutation and combination is a computationally expensive task. We claim that the efficiency of MagiCoder can be preserved also extending it with more advanced features, such as recognition of words in presence of orthographical errors. As a future work, we plan to provide

formal and experimental comparisons of performances of MagiCoder with respect to the software proposed in [16].

IV. Software Implementation and Testing

A. The User Interface

MagiCoder has been implemented as a VigiWork plug-in: people responsible for pharmacovigilance can extract the auto-encoding of the narrative description and then revise and validate it. Figure 3 shows a screenshot of VigiWork for the part supporting back-end tasks (done by responsibles for pharmacovigilance revision activities). In the high part of the screen it is possible to observe the five sections composing a report. The screenshot actually shows the result of a human-MagiCoder interaction: by pressing the button "Autocodifica in MedDRA" (in English, "MedDRA autoencoding"), the responsible for pharmacovigilance obtains a MedDRA encoding for the natural language ADR in the field "Descrizione" (in English, "Description"). Six solutions are proposed as the best MedDRA term candidates: the responsible can refuse a term (through the trash icon), change one or more terms (by an option menu), or simply validate the automatic encoding and switch to the next section "Farmaci" (in English, "Drugs"). We are testing MagiCoder performance in the daily pharmacovigilance activities. Preliminary qualitative results show that MagiCoder drastically reduces the amount of work required for the revision of a report, allowing the pharmacovigilance stakeholders to provide high quality data about suspected ADRs.

B. Testing

As a preliminary step in evaluating MagiCoder performances, we developed a benchmark, which automatically compares MagiCoder behavior with human encoding on already manually revised and validated ADR reports.

To this end, we exploited VigiSegn, a data warehouse and OLAP system for the italian pharmacovigilance activities developed for the Italian Pharmacovigilance National Center [23]. This system is based on the open source business intelligence suite Pentaho. VigiSegn offers a large number of *encoded* ADRs. The encoding has been manually performed and validated by experts working at pharmacovigilance centres. Encoding results have then been sent to the national regulatory authority AIFA.

We performed a test, composed by the following steps.

1) We launch an ETL procedure through Penthao Data Integrator. The procedure transfers reports from VigiSegn to an ad-hoc database TestDB. The dataset covers all the 6780 reports received, revised, and validated during the year 2014 for the Italian region Veneto.
2) We launch an ETL procedure which extracts from reports stored in TestDB the narrative descriptions. For each description, the procedure calls MagiCoder from VigiWork; the output, i.e., a list of MedDRA terms, is stored in a table of TestDB.
3) Manual and automatic solutions, i.e., LLT term sets, are finally compared through an SQL query. We compute how much manual solutions are "covered" by MagiCoder. In other words, we perform a similarity test between the two output sets. In order to have two uniform data sets, we map each LLT term, both from the manual and the automatic solutions, to its corresponding preferred term.

Table II shows the results of this first performance test.

It is worth noting that this test simply estimates how much MagiCoder behavior is similar to the manual work on the *whole set of solutions*, without considering the quality of the manual encoding. We may observe that for short descriptions MagiCoder results are very close to those from manual encoding. The percentage of similarity decreases with the growing of the number of characters, but it is stable beyond a certain threshold. It could suggest that MagiCoder will behave very well on very long (intractable) descriptions: as a human reviewer, the procedure does not encode redundant text. Since we did not evaluate the quality of the human solutions we take into account, we are working on a further quantitative analysis of MagiCoder performances. We are developing an experimental test, involving three experts in report revision. Two experts (and a third one, in case a reconciliation of diverging encoding is needed) are manually encoding a representative sample of ADR descriptions (about 200), in order to build a ground truth data set. These "certified" manual solutions will be compared, report by report, with MagiCoder's outputs. The test has been designed to effectively measure soundness and completeness of MagiCoder. Informally, soundness can be estimated with respect to false positive terms provided by MagiCoder; completeness can be estimated according to LLT terms omitted by MagiCoder. We will precisely quantify the difference between the human and the automatic encoding (taking into account also syntactically different but semantically equivalent solutions) and, thus, we will be able to compute the standard deviation of the behavior of the procedure w.r.t. the expected performance.

C. Examples

Table III provides some examples of the behavior of MagiCoder. We propose some free-text ADR descriptions from TestDB and we provide both the manual and the automatic encodings into LLT terms. We also provide the English translation of the natural language text (we actually provide a quite straightforward *literal* translation).

D1– anaphylactic shock (hypotension + cutaneous rash) 1 hour after taking the drug.

D2– swelling in vaccination location left from 11/5; temperature less than 39,5 from 11/21; vesicles, blisters around the cheek from 11/10.

D3– extended local reaction, local pain, headache, fever for two days.

In Table III we use the following notations: $\underline{t_1}^n$ and $\underline{t_2}^n$ are two *identical* LLT terms retrieved both by the human and the automatic encoding; $\overline{t_1}^n$ and $\overline{t_2}^n$ are two *semantically equivalent* or *similar* LLT terms retrieved by the human and the automatic encoding, respectively; we use bold type to denote terms that are recognized by MagiCoder and that have not

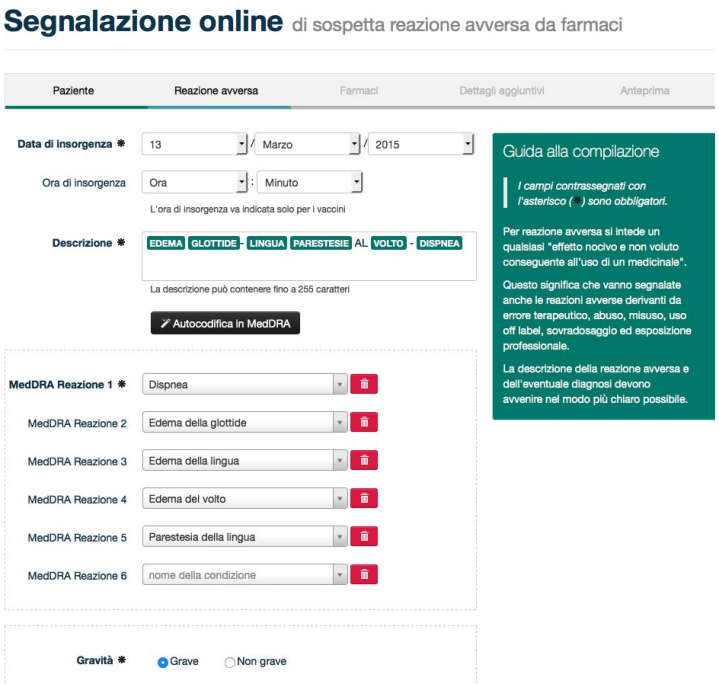

Fig. 3. A partial screenshot of `VigiWork` User Interface

Length of the Description (# chars)	Percentage of global identical solutions at the PT level
Short descriptions (up to 20 chars)	81%
Short/medium descriptions (from 20 up to 40 chars)	62%
Medium descriptions (from 40 up to 100 chars)	62%
Long descriptions (from 100 up to 250 chars)	61%

TABLE II
FIRST RESULTS OF MAGICODER PERFORMANCES

#	Narrative Description	LLT Human Encoding	LLT MagiCoder Encoding
D1	Shock anafilattico (*ipotensione* + rash cutaneo) 1 h dopo assunzione x os del farmaco	Shock anafilattico[1]	**Ipotensione**, Shock anafilattico[1]
D2	gonfiore in sede di vaccinazione sx dal 5/11, febbre meno di 39,5 dal 21/11, vescicole, *bolle* presso la guancia dal 10/11	Gonfiore in sede di vaccinazione[1], Piressia[2], Vescicole[3]	**Bolle**, Febbre[2], Gonfiore in sede di vaccinazione[1], Vescicole in sede di vaccinazione[3]
D3	Reazione locale estesa, *dolore* locale; cefalea e febbre per due giorni	Cefalea[1], Febbre[2], Reazione in sede di vaccinazione[3]	Cefalea[1], **Dolore**, Febbre[2], Reazione locale[3]

TABLE III
EXAMPLES OF MAGICODER BEHAVIOR

been encoded by the reviewer; we use italic type in D1, D2, D3, to denote text recognized only by MagiCoder. For example, in description D3, "cefalea" (in English, "headache") is retrieved and codified both by the human reviewer and MagiCoder; in description D2, ADR "febbre" (in English, "fever') has been codified with the term itself by the algorithm, whereas the reviewer codified it with its synonym "piressia"; in D1, ADR "ipotensione" (in English, "hypotension"), has been retrieved only by MagiCoder.

V. CONCLUSIONS AND FUTURE WORK

In this paper we propose MagiCoder, a simple and efficient NLP software, able to provide a concrete support to pharmacovigilance task, in the revision of ADR spontaneous reports. MagiCoder takes in input a narrative description of a suspected ADR and produces as outcome a list of `MedDRA` terms that "cover" the medical meaning of the free-text description. We presented and implemented the first version of the algorithm, and preliminary results about its performances are encouraging.

Finally, let us sketch here some ongoing and future work.

First, we aim to prove that MagiCoder is robust with respect to language (and dictionary) changes. The way the algorithm has been developed suggests that MagiCoder can be a valid tool also for narrative descriptions written in English. Indeed, the algorithm retrieves a set of words, which covers an LLT term t, from a free-text description, without considering the order between words or the structure of the sentence. This way, we avoid the problem of "specializing" MagiCoder for any given language. Furthermore, MagiCoder performances can be strengthened, still maintaining the simple "skeleton" we proposed, eventually embedding new features *inspired* to advanced NLP techniques. Even though negative sentences seem to be uncommon in ADR descriptions (at least in the data set we analyzed), the detection of negative forms is a short-term issue we aim to address. As a first step, we plan to recognize words that may represent negations and to signal them to the reviewer through the graphical UI. In this way, the software sends to the report reviewer an alert about the (possible) failure of the syntactical word-by-word recognition. Moreover, we plan to address the management of orthographical errors possibly contained in narrative ADR descriptions. We did not take into account this issue in the current version of MagiCoder. A solution could be including an ad-hoc (medical term-oriented) spell checker in VigiWork, to point out to the user that she/he is doing some error in writing the current word in the free description field. This should drastically reduce users' orthographical errors without heavy side effects in MagiCoder development and performances. As a further extension of MagiCoder, we will enrich the algorithm with heuristics and synonyms dictionaries. Moving towards the use of ad-hoc thesaurus dictionaries, our idea is to progressively (through everyday learning and feedback coming from experience) *extend* MedDRA with synonyms of LLT terms. Finally, we aim to apply MagiCoder (and its refinements) to several different sources for ADR detection, such as, for example, drug information leaflets.

REFERENCES

[1] N. Arthur, A. Bentsi-Enchill, and R. Couper et al., *The Importance of Pharmacovigilance - Safety Monitoring of Medicinal Products.* World Health Organization, 2002.

[2] J. Borg, G. Aislaitner, M. Pirozynski, and S. Mifsud, "Strengthening and rationalizing pharmacovigilance in the EU: where is europe heading to? a review of the new EU legislation on pharmacovigilance," *Data Safety*, vol. 34, no. 3, pp. 187–197, 2011.

[3] L. Aagaard, J. Strandell, L. Melskens, P. Petersen, and E. Holme Hansen, "Global patterns of adverse drug reactions over a decade: analyses of spontaneous reports to vigibase," *Drug Safety*, vol. 35, pp. 1171–1182, 2012.

[4] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[5] A. Bate and S. Evans, "Quantitative signal detection using spontaneous ADR reporting," *Pharmacoepidemiology and Drug Safety*, vol. 18, no. 6, pp. 427–436, 2009.

[6] X. Wang, G. Hripcsak, M. Markatou, and C.Friedman, "Active computerized pharmacovigilance using natural language processing, statistics, and electronic health records: A feasibility study," *JAMIA*, vol. 16, no. 3, pp. 328–337, 2009.

[7] C. Friedman, "Discovering novel adverse drug events using natural language processing and mining of the electronic health record," in *Artificial Intelligence in Medicine*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5651, pp. 1–5.

[8] E. Aramaki, Y. Miura, M. Tonoike, T. Ohkuma, H. Masuichi, and K. Waki, "Extraction of adverse drug effects from clinical records," *Stud Health Technol Inform*, vol. 160, no. Pt1, pp. 739–43, 2012.

[9] M. G. R. Reichley, P. Kilbridge, L. Noirot, R. N. R, W. Dunagan, and T. Bailey, "Natural language processing to identify adverse drug events," in *AMIA Annu Symp Proc.*, 2008.

[10] P. M. Kilbridge, L. A. Noirot, R. M. Reichley, and T. C. Bailey, "Computerized surveillance for adverse drug events in a pediatric hospital," *J Am Med Inform Assoc.*, vol. 16, no. 5, pp. 607–612, 09.

[11] H. Gurulingappa, A. Mateen-Rajput, and L. Toldo, "Extraction of potential adverse drug events from medical case reports," *Journal of Biomedical Semantics*, vol. 3, no. 15, pp. 1–10, 2012.

[12] A. Sarker and G. Gonzalez, "Portable automatic text classification for adverse drug reaction detection via multi-corpus training," *Journal of Biomedical Informatics*, vol. 53, pp. 196–207, 2015.

[13] C. C. Yang, H. Yang, L. Jiang, and M. Zhang, "Social media mining for drug safety signal detection," in *Proc. of the 2012 Int. Workshop on Smart Health and Wellbeing, SHB 2012*, 2012, pp. 33–40.

[14] R. Harpaz, H. S. Chase, and C. Friedman, "Mining multi-item drug adverse effect associations in spontaneous reporting systems," *BMC Bioinformatics*, vol. 11, no. S-9, p. S7, 2010.

[15] N. Nissim, M. Boland, R. Moskovitch, N. Tatonetti, Y. Elovici, Y. Shahar, and G. Hripcsak, "An active learning framework for efficient condition severity classification," in *Artificial Intelligence in Medicine (AIME'15)*, ser. Lecture Notes in Computer Science. Springer, 2015, vol. 9105, pp. 13–24.

[16] G. Dalhberg, "Implementation and evaluation of a text extraction tool for adverse drug reaction information," 2010, master Thesis, Uppsala University School of Engineering.

[17] M. Collins, "Tutorial: Machine learning methods in natural language processing," in *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory*, 2003, p. 655.

[18] P. Radhakrishna, "Upversioning MedDRA dictionary - insights from a seasoned coder," *Data Basics*, vol. 20, no. 3, pp. 1171–1182, 2014.

[19] L. Bauer, "Introducing linguistic morphology," 2003.

[20] K. Kishida, "Technical issues of cross-language information retrieval: A review," *Inf. Process. Manage.*, vol. 41, no. 3, pp. 433–455, May 2005.

[21] A. Clark, C. Fox, and S. Lappin, Eds., *The Handbook of Computational Linguistics and Natural Language Processing*, ser. Blackwell Handbooks in Linguistics. John Wiley & Sons, 2010.

[22] J. Piskorski and M. M. Sydow, "String distance metrics for reference matching and search query correction," in *Business Information Systems*, ser. Lecture Notes in Computer Science, W. Abramowicz, Ed. Springer Berlin Heidelberg, 2007, vol. 4439, pp. 353–365.

[23] A. Sabaini, "Temporal data analysis and mining: A multidimensional approach and its application in a medical domain," Ph.D. dissertation, Department of Computer Science, University of Verona - Italy, 2015.