Evolutionary Learning of Syntax Patterns for Genic Interaction Extraction

Alberto Bartoli
DIA - University of Trieste
Italy
bartoli.alberto@units.it

Andrea De Lorenzo
DIA - University of Trieste
Italy
andrea.delorenzo@units.it

Eric Medvet
DIA - University of Trieste
Italy
emedvet@units.it

Fabiano Tarlao
DIA - University of Trieste
Italy
fabiano.tarlao@phd.units.it

Marco Virgolin
DIA - University of Trieste
Italy
marco.virgolin@gmail.com

ABSTRACT

There is an increasing interest in the development of techniques for automatic relation extraction from unstructured text. The biomedical domain, in particular, is a sector that may greatly benefit from those techniques due to the huge and ever increasing amount of scientific publications describing observed phenomena of potential clinical interest.

In this paper, we consider the problem of automatically identifying sentences that contain interactions between genes and proteins, based solely on a dictionary of genes and proteins and a small set of sample sentences in natural language. We propose an evolutionary technique for learning a classifier that is capable of detecting the desired sentences within scientific publications with high accuracy. The key feature of our proposal, that is internally based on Genetic Programming, is the construction of a model of the relevant syntax patterns in terms of standard part-of-speech annotations. The model consists of a set of regular expressions that are learned automatically despite the large alphabet size involved.

We assess our approach on two realistic datasets and obtain 74% accuracy, a value sufficiently high to be of practical interest and that is in line with significant baseline methods.

Categories and Subject Descriptors

I.5.4 [**Pattern Recognition**]: Applications—*Text processing*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; J.3 [**Computer Applications**]: Life and Medical Science—*Biology and genetics*

Keywords

Regular Expressions; Genetic Programming; Programming by Example; Machine Learning

GECCO '15, July 11–15, 2015, Madrid, Spain

http://dx.doi.org/10.1145/2739480.2754706 .

1. INTRODUCTION

A huge amount of knowledge is expressed in natural language text and the recent years have seen an explosion of interest in automated techniques capable of extracting such knowledge effectively. Biomedical information extraction, in particular, is a vast and rapidly growing field facing many important challenges [32, 22, 33, 8, 34, 28, 18]. A problem that has captured much attention is the construction of a systematic and structured description of the observed interactions between entities of biomedical interest reported in the scientific literature [19, 17, 25].

In this paper, we investigate the feasibility of using *evolutionary computing* for attacking a challenging problem in this area: how to identify automatically, in scientific papers, sentences that contain interactions between genes and proteins, based on a dictionary of genes, proteins, interactors, and a small set of example sentences [8, 34, 28]. The problem is challenging because the mere occurrence of dictionary words in a sentence does not imply that the sentence is to be extracted (see Table 1). Evolutionary computing techniques have been applied to several problems in Natural Language Processing (NLP), including text summarization [26], sentence alignment for statistical machine translation [29], part-of-speech tagging [1], grammar induction from an annotated corpus [13, 31], word sense disambiguation [14]. A very good and broad review of evolutionary approaches to NLP is [2]. The *binary classification of sentences* that we consider here, though, has received very little attention so far and we are aware of only one evolutionary proposal in this area [7]. The cited work advocates the use of a probabilistic model called the hypernetwork classifier meant to capture the correlation between sets of words and the class in which a sentence containing those words belongs to. Actual values for the correlation between triplet of words and the output class in a learning corpus are computed by means of a stochastic procedure. The experimental evaluation considered 1 200 000 such triplets and executed some preprocessing operations on the text that were not fully described, including deletion of so-called redundant components and insignificant words, as well as conversion of string values into numerical values using a customized dictionary.

In our work we propose a different evolutionary technique based on *Genetic Programming* (GP). We learn a model of the *syntax patterns* that occur in the sentences of each

class and classify sentences accordingly. Such patterns are expressed in terms of *regular expressions* over standard *part-of-speech* annotations. We solve the difficult problem of actually learning those regular expressions despite the *large alphabet size* involved, by leveraging on recent results in this area [4]. Working at the abstraction level of part-of-speech annotations has important practical advantages: one may build upon the existing state-of-the-art in NLP annotators, as well as switch to an annotator for a different language or incorporate any advances in annotation technology, without any changes in the framework.

We strove to design our framework without incorporating heuristic rules or findings from previous solutions to this problem. For example, part-of-speech patterns of interest are usually very short and include 2–4 words before and after entities of interest (i.e., genes or proteins) [19, 18]. Furthermore, one could refine a tentative pattern by carefully assigning different costs to each component of the pattern depending on its type—40 coefficients are proposed in [19] for weighing the contribution of 12 groups of part-of-speech tags depending on whether they are aligned correctly with a sample and, if not, depending on the nature of mismatch. We chose to not rely on any problem-specific heuristics because we were more interested in assessing the potential of evolutionary computation in this area than in squeezing accuracy figures. The opposite choice would have made it difficult to isolate the merit of evolutionary learning of syntax patterns from problem-specific heuristics—which of course deserve further investigation.

We remark that our evolutionary learning of syntax patterns is not aimed at grammar induction, a problem that has received much attention in the literature, both for natural text and for formal languages [12, 2]. Grammar induction aims at inferring a set of rules which describe the input data at differing granularity levels and, in the case of natural language, have a hierarchical structure [2, 15]. Usually, although not necessarily, the quality of a solution is assessed from a gold truth of rules [2]. Here we aim instead at *partitioning* text at the granularity level of full sentences, rules need not have any specific structure, no gold truth of rules exists and only the desired partitioning is specified. Of course, techniques for grammar induction could be useful in our scenario as well but it must be emphasized that grammar induction and binary classification of sentences are different problems.

A wealth of literature exist for building classifiers based on Genetic Programming [16], but natural text is outside of the scope of most proposals. Furthermore, classification of natural text is usually done at the level of full documents and such a granularity is excessively coarse in our context [20].

We assess our approach on a realistic dataset and compare the resulting accuracy to significant baseline methods. The results indicate that our GP-based proposal indeed learns syntax patterns from examples effectively, even in NLP scenarios of practical interest.

2. THE PROBLEM

We aim at generating a classifier $\mathcal{C}$ that takes a natural language sentence s in input and performs a binary classification based on whether s contains a genic-protein interaction.

We generate the classifier based on two *learning sets* S_L^+, S_L^-: sentences in S_L^+ contain a genic-protein interaction whereas sentences in S_L^- do not. Two dictionaries are also available of genes/proteins and interactors, as usually done in in applications of this sort [19, 17, 18].

In order to better appreciate nature and difficulty of the problem, it is useful to emphasize that sets S_L^+, S_L^-, are *not* labelled automatically based on some predefined pattern that is kept hidden to the classifier generation process. Instead, the labelling is done by domain experts which read and analyze the meaning of every single sentence as a whole. Domain experts do not choose the class for a sentence using a predefined pattern—a pattern suitable for classifying sentences as desired may not even exist.

3. OUR APPROACH

We aim at inferring a classifier capable of detecting *syntax patterns* of the sentences to be extracted, beyond the mere co-occurrence of relevant words. The classifier operates on sequences of Unicode characters which we call *ϕ-strings*. In a nutshell, we transform each sentence to a sequence of *Part-of-Speech (POS)* annotations, we group annotations of genes/proteins and interactors, and map each annotation to an arbitrarily chosen Unicode character. Full details of this procedure are provided in Section 3.1, an example is given in Figure 1.

The target classifier $\mathcal{C}$ consists of a set of *regular expressions* $\{r_1^\star, r_2^\star, \dots\}$. The output of $\mathcal{C}$ for an input ϕ-string x will be *positive* (i.e., $\mathcal{C}$ deems that the sentence corresponding to x contains a genic-protein interaction) if at least one $r_i^\star$ is matched by x or by a non-empty substring of x.

We attack the complex problem of actually generating $r_i^\star$ by means of a Genetic Programming (GP) procedure inspired by recent proposals for learning regular expressions from examples [3, 4]. The cited proposals are designed for solving text extraction problems at the level of short text snippets. We had to modify and tailor these proposals for a classification problem in a different domain—properties of text snippets and of POS sequences seem to be quite different. In particular, there are two significant differences between the two scenarios. First, regular expressions include constructs which enable generalization and compactness when applied on actual text but that are not very meaningful on sequences of POS annotations, i.e., character classes `\w` and `\d`. Second, the cited works assume that the examples consist of exactly the text snippets to be extracted, whereas in our context the examples consist of full sentences: the information to be captured by the target regular expression is much more noisy and diluted.

The number of regular expressions in the set $\mathcal{C}$ is not determined in advance and is instead discovered automatically. Specifically, we generate regular expressions one at a time by means of a *separate-and-conquer* procedure. Initially, we generate the first regular expression by using all the available data. Then, once a regular expression is found that provides adequate performance on a *subset* of the examples, we restart the evolutionary search from the scratch by using only the remaining examples that are not yet solved adequately. This procedure is also inspired by a recent proposal designed for extraction of short text snippets [5]: differently from the cited paper, here we focus on classification instead of extraction and allow the generation of regular expressions that do not exhibit perfect precision.

Sentences w/ genic-protein interaction	Sentences w/o genic-protein interaction
In this mutant, *expression* of the **spoIIG** gene, whose transcription depends on both **sigma(A)** and the phosphorylated **Spo0A** protein, **Spo0A-P**, a major transcription factor during early stages of sporulation, was greatly reduced at 43 degrees C.	From day 10, a significant increase in platelet count was observed in eight of the ten patients treated with heparin (p < 0.05), with return to the initial value after heparin cessation in six of the responders.
These results suggest that **YfhP** may act as a negative *regulator* for the transcription of **yfhQ**, **yfhR**, **sspE** and **yfhP**.	Two phosphopeptides, identified as **RS-[32P]SGASGLLTSEHHSR** and **S-[32P]SGASGLLTSEHHSR**, were obtained after stoichiometric *phosphorylation* and trypsinization of the peptide.
These results demonstrate that **sigmaK**-dependent transcription of **gerE** initiates a negative feedback loop in which GerE acts as a *repressor* to limit production of **sigmaK**.	Levels of **TSG-14** protein (also termed **PTX-3**) become elevated in the serum of mice and humans after injection with bacterial lipopolysaccharide, but in contrast to conventional acute phase proteins, the bulk of **TSG-14** *synthesis* in the intact organism occurs outside the liver.
In this study, we used footprinting and gel mobility retardation assays to reveal that bacterially sinthetized **Zta** fusion proteins *bound* directly to six **TGTGCAA**-like motifs within DSL.	No mutations were found in follicular adenomas.

Table 1: Eight sentences of the corpus used in our experimentation, 4 which include (left) and 4 which do not include (right) genic-protein interactions. For the sake or comprehension, we highlighted in bold those words belonging to the D_{genes} dictionary and in italic those belonging to the $D_{\text{interactors}}$ dictionary (see Section 3.1).

3.1 Sentence representation

Let D_{genes} and $D_{\text{interactors}}$ be the statically available dictionaries of words representing genes and interactors, respectively: Table 2 shows portions of the two dictionaries which we actually used in our experimentation. We transform each sentence s into a ϕ-string x, as follows. For ease of presentation, in the following we will always use the term "gene" to mean either a gene or a protein.

1. We split s into a sequence $\{t_1, \ldots, t_n\}$ of tokens according to the Penn-Treebank procedure[1].
2. We execute a *Part-of-Speech* (POS) annotator over $\{t_1, \ldots, t_n\}$ and obtain a sequence $\{a_1, \ldots, a_n\}$ of annotations, with $a_i \in A$. The set A of possible annotations depends on the specific POS annotator being used; we assume that three disjoint subsets $A_{\text{verb}}, A_{\text{adj}}, A_{\text{noun}}$ of A exist which correspond to verbs, adjectives, and nouns, respectively—A may include other elements not contained in $A_{\text{verb}} \cup A_{\text{adj}} \cup A_{\text{noun}}$. In our experiments we used the annotator developed by the Stanford Natural Language Processing Group[2]: Table 3 shows a partial list of the elements of the set A corresponding to this POS annotator.
3. We modify the sequence $\{a_1, \ldots, a_n\}$ of annotations, as follows. We say that $t \in^* D$ if t is equal to or starts with an element of D—e.g., if $D = \{\texttt{sigmaB}, \texttt{katX}\}$, then $\texttt{sigmaB-dependent} \in^* D$ and $\texttt{katX} \in^* D$. For each i,
   - if $t_i \in^* D_{\text{genes}}$, we set $a_i := \texttt{GENEPTN}$;
   - if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{verb}}$, we set $a_i := \texttt{IVERB}$;
   - if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{adj}}$, we set $a_i := \texttt{IADJ}$;
   - Finally, if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{noun}}$, we set $a_i := \texttt{INOUN}$.

   We denote with $A' = A \cup \{\texttt{GENEPTN}, \texttt{IVERB}, \texttt{IADJ}, \texttt{INOUN}\}$ the set of possible annotations after this step.

[1] http://www.cis.upenn.edu/~treebank

[2] http://nlp.stanford.edu/software/tagger.shtml

D_{genes}	$D_{\text{interactors}}$
amyE	abrogation
bmrUR	activation
ComK	destabilization
DksA	expression
Esig29	repression
GerE	affect
katX	bind
KinC	destabilize
sigmaH	exhibit
SpoIIAB	regulate
others	*others*

Table 2: Portions of the two dictionaries of genes/proteins and interactors used in our experimentation.

4. Let U be the set of characters including digits, lowercase letters and uppercase letters and let $\phi : A' \to U$ be an injective function which maps annotations to characters (see rightmost column of Table 3). We obtain the ϕ-string x from $\{a_1, \ldots, a_n\}$ by concatenating the characters resulting from the application of ϕ to each element of the sequence of annotations, i.e., we set $x = \phi(a_1) \ldots \phi(a_n)$.

Figure 1 shows the intermediate and final outcomes of the procedure here described when applied to an example sentence.

3.2 Regular expression generation

We here describe our procedure based on GP for obtaining a regular expression $r^\star$ from two *training sets* X^+, X^- of ϕ-strings. The aim of this procedure is to generate a regular expression $r^\star$ such that: (i) for each ϕ-string x in X^+, x, or a non-empty substring of x, match $r^\star$; and (ii) for each ϕ-string x in X^-, x and all the substrings of x do not match $r^\star$. The relation between the training sets and the learning sets available for synthesizing the classifier (Section 2) will be clarified later.

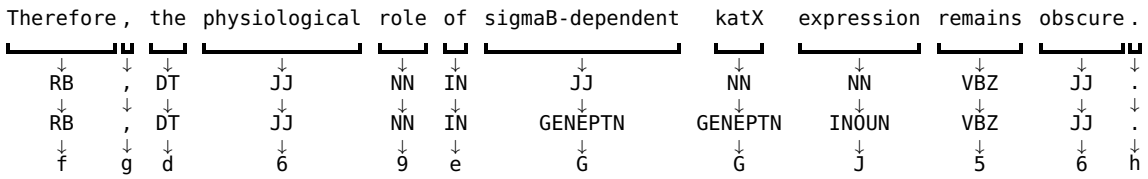

Figure 1: Example of sentence representation. The original textual sentence s (top) is transformed into a ϕ-string x (bottom) through two cascaded transformations: one producing a sequence of POS annotations and another where certain annotations are modified using the dictionaries (Section 3.1).

a	Meaning	Subset	$\phi(a)$
VB	verb, base form	A_{verb}	0
VBD	verb, past tense		1
VBG	verb, pres. part. or gerund		2
VBN	verb, past participle		3
VBP	verb, pres. tense, ¬3rd p. s.		4
VBZ	verb, pres. tense, 3rd p. s.		5
JJ	adjective or numeral, ordinal	A_{adj}	6
JJR	adjective, comparative		7
JJS	adjective, superlative		8
NN	noun, common, sing. or mass	A_{noun}	9
NNP	noun, proper, singular		a
NNPS	noun, proper, plural		b
NNS	noun, common, plural		c
CC	conjunction, coordinating		d
CD	numeral, cardinal		e
DT	determiner		f
	other annotations		
GENEPTN	gene or protein	$A' \setminus A$	G
IVERB	verb interactor		R
IADJ	adjective interactor		P
INOUN	noun interactor		J

Table 3: A partial list of the elements of A': for space constraints, only a subset of $A \setminus (A_{\text{verb}} \cup A_{\text{adj}} \cup A_{\text{noun}})$ is shown.

3.2.1 Solution representation and fitness definition

We represent a candidate solution, i.e., an *individual*, with a tree. Each tree represents a regular expression, as follows. The set of terminal nodes is composed of the wildcard character `.` and of each character in range(ϕ), i.e., each character in U which corresponds to an annotation in A'. The set of function nodes is composed of: the concatenator `¤¤`; the character class `[¤]` and negated character class `[^¤]`; the possessive quantifiers `¤*+`, `¤++`, `¤?+` and `¤{¤,¤}+`; and the non-capturing group `(?:¤)`. A tree represents a regular expression by means of a depth-first visit in which each `¤` symbol in a non-terminal node is replaced by the representation the corresponding child node.

The *fitness* of an individual r quantifies the behavior of the individual over the training sets. We define the fitness as a tuple $f(r) := (\text{FPR}(r, X^-) + \text{FNR}(r, X^+), \text{FPR}(r, X^-), \ell(r))$, where $\ell(r)$ is the length of the regular expression represented by r and $\text{FPR}(r, X^-)$ and $\text{FNR}(r, X^+)$ are the False Positive Rate and False Negative Rate, respectively, of r on the training sets. In more detail, $\text{FPR}(r, X^-)$ is the percentage of ϕ-strings $x \in X^-$ for which x, or a non-empty substring of x, match r; $\text{FPR}(r, X^+)$ is the percentage of ϕ-strings $x \in X^+$ for which x and all the substrings of x do not match r. For all the elements of the fitness tuple, the lower, the better. The first objective (i.e., $\text{FPR}(r, X^-) + \text{FNR}(r, X^+)$) is a proxy for the accuracy of classification of r on the examples in X^-, X^+. We chose not to use accuracy in order to accommodate possibly unbalanced learning sets (our experimental evaluation used balanced sets, though).

We *rank* individuals based on their fitness tuples according to Pareto-dominance and lexicographic order, as follows. First, individuals are sorted by their Pareto frontier: an individual belongs to the i-th frontier if and only if it is Pareto-dominated only by individuals, if any, belonging to j-th frontier, with $j < i$—an individual Pareto-dominates another individual if it is better on at least one fitness element and not worse on the other two elements. Second, a total ordering is established among individuals in the same Pareto frontier: individuals with lower FPR + FNR come first; in case of equal FPR + FNR individuals with lower FPR come first; in case of equal FPR + FNR and equal FPR individuals with lower ℓ come first.

As pointed out in the introduction, we purposefully avoided to include any problem-specific knowledge in the fitness definition—e.g., part-of-speech tags associated with either genes or interactors do not have any special status. We use a multiobjective approach aimed at maximizing accuracy while promoting compact solutions for preventing bloat [11]. We add a third objective beyond accuracy and length because, as detailed in Section 3.3, the classifier does not consist of a single regular expression and is instead composed of a set of regular expressions generated from progressively smaller training sets. A form of evolutionary pressure on the FPR of each member of the set turns out to be beneficial to the aggregate FPR of the full set.

3.2.2 Search procedure

We execute the GP search with a population of n_{pop} individuals. The initial population is composed of a portion of randomly generated individuals and a portion of individuals designed to match at least one ϕ-string in X^+. In detail, we build 3 individuals generated from each $x \in X^+$:

1. an individual r representing a regular expression which is equal to the ϕ-string x;

2. an individual r' obtained by replacing in r each leaf node not included in $\{\phi(\texttt{GENEPTN}), \phi(\texttt{IVERB}), \phi(\texttt{IADJ}), \phi(\texttt{INOUN})\}$ with a subtree corresponding to the regular expression `[^`ϕ`(GENEPTN)`ϕ`(IVERB)`ϕ`(IADJ)`ϕ`(INOUN)]`, i.e., with the character class which excludes characters corresponding to genes and interactions;

3. an individual r'' obtained by replacing in r' consecutive repetitions of the character class which excludes genes and interactions with `[^`ϕ`(GENEPTN)`ϕ`(IVERB)`ϕ`(IADJ)`,

ϕ(`INOUN`)`]++`, i.e., with the subtree corresponding to one or more repetitions of the character class.

If the number of individuals generated by this procedure is greater than n_{pop}, then we remove exceeding individuals chosen at random (this event does not occur in our experimental setting); otherwise, we generate missing individuals at random with a ramped half-and-half method.

We evolve the initial population by means of the following procedure. At each iteration (or *generation*), we generate n_{pop} new individuals: 80% of them by crossover between individuals in the current population; 10% of them by mutation of individuals in the current population; 10% of them at random with a ramped half-and-half method. We build the new current population by retaining only the n_{pop} individuals with best fitness from the resulting $2n_{\text{pop}}$ individuals (current population and new generated individuals). We select individuals for either crossover or mutation by means of a tournament selection of size 7. Whenever we generate an individual which represents a not valid regular expression, we discard that individual and generate a new one.

We enforce *genotypic diversity* among candidate solutions, i.e., whenever an individual r_1 is generated which represents the same regular expression represented by another individual r_2 in the current population, then r_1 is discarded and another individual is generated—a similar mechanism is used in [24] for preventing the creation of duplicated solutions. We chose to include this simple mechanism in the search—not present in [4]—because our earlier experiments clearly demonstrated its effectiveness in our scenario, the quality of generated solutions being substantially improved without any significant increase in processing time. Such an improvement is perhaps not surprising, because we do not start the search with a fully random population and because a multiobjective GP search may greatly benefit from an explicit mechanism for promoting forms of diversity among candidate solutions, either at the genotypic level or in the objective space or in their behavior [11, 9]. However, we remark that the mechanism that we have chosen is extremely easy to implement and, in particular, does not involve the problem of choosing how to *quantify* the amount of diversity between individuals. Furthermore, it does not involve the need of defining diversity as a *further objective* to be taken into account during the search, which might be difficult to achieve in our scenario since the resulting fitness would be composed of four indexes and thus more radical changes to the evolutionary strategy could be required [21]. A detailed analysis of the effectiveness of this mechanism, as well as of other possible diversity enforcement criteria (e.g., [11, 9]), is beyond the scope of this paper, though.

The search terminates when one of the following occurs: (i) a predefined number of n_{gen} iterations has been executed; or, (ii) the fitness tuple of the best individual has remained unchanged for n_{stop} consecutive iterations. The regular expression represented by the best individual of the final population is the outcome of the search procedure.

3.3 Classifier generation

We apply the procedure described in Section 3.1 and transform the *learning* sets S_L^+, S_L^- to sets of ϕ-strings X_L^+, X_L^-, respectively. Then, we randomly sample X_L^+, X_L^- to build the *training* sets X^+, X^-, respectively, ensuring that $\frac{|X^+|}{|X^-|} = \frac{|X_L^+|}{|X_L^-|}$.

We start from an initially empty set of regular expressions ($\mathcal{C} = \emptyset$) and repeat the following iterative procedure (τ_{FPR} is a predefined threshold): 1) execute a search on X^+, X^- and obtain $r^\star$; 2) if either $\text{FPR}(r^\star, X^-) \leq \tau_{\text{FPR}}$ or the search terminated after executing n_{gen} generations, then assign $\mathcal{C} := \mathcal{C} \cup \{r^\star\}$, otherwise terminate; 3) remove from X^+ the ϕ-strings x for which x, or a substring of x, match $r^\star$; 4) if X^+ is empty, then terminate, otherwise go to step 1. The outcome of this procedure is a classifier $\mathcal{C}$.

We perform n_{job} independent executions of the procedure, all starting with the same training sets X^+, X^- but with different random seeds. Thus, we obtain n_{job} (possibly) different classifiers and choose the one with lowest error rate on the learning sets X_L^+, X_L^-. In other words, we validate the n_{job} classifiers on data which was not available during the training in order to prevent overfitting the data.

In our experimentation, we set $\tau_{\text{FPR}} = 0.3$, $n_{\text{gen}} = 1000$, $n_{\text{stop}} = 200$, $n_{\text{pop}} = 1000$, and $n_{\text{job}} = 8$; we chose these values after preliminary experimentation and basing (in particular for n_{gen}, n_{pop}, and n_{job} parameters) on the abundant literature about GP.

4. EXPERIMENTAL EVALUATION

4.1 Datasets and baselines

We used a corpus of 456 sentences built by joining two corpora, both derived from genic-protein interactions extraction challenges of biomedical interest[3]. Each sentence was labelled by a domain expert. Table 1 shows some samples. It can be seen that the mere presence of words in the dictionaries D_{genes} and $D_{\text{interactors}}$ does not suffice to qualify a sentence as containing a genic-protein interaction.

In order to assess our results, we implemented several alternative classification techniques: a state-of-the-art evolutionary approach for inferring patterns from examples, two approaches that embed a large amount of problem-specific knowledge and two established methods for text classification.

4.1.1 DFA-based pattern evolutionary inference

We implemented a classifier based on the *Smart State Labelling Evolutionary Algorithm (SSLEA)* proposed in [27]. The cited work is a state-of-the-art algorithm for learning *deterministic finite automata (DFA)* from examples of the desired classification behavior. SSLEA was developed a few years after a competition that was highly influential in the grammar learning community and outperformed (optimized versions of) the winners of the competition, on the same class of problems [23, 10] and even in the presence of noisy data.

SSLEA represents a candidate solution (i.e., a DFA) by a pair composed of an output vector of size n and a transition matrix of size $n \times |\alpha|$, where n is the number of states in the target DFA and $|\alpha|$ is the number of symbols in the input alphabet: the former has one element for each DFA state and each element contains the label (accept or reject) for the corresponding state; the latter contains, for each state and transition, the corresponding destination state index. SSLEA implements a form of hill-climbing in which the fitness of a solution is the rate of examples classified correctly.

[3] `http://genome.jouy.inra.fr/texte/LLLchallenge/#task1` and `https://www2.informatik.hu-berlin.de/~hakenber/corpora/#bc`

The search terminates when either a DFA with perfect fitness is found or a predefined number n_{it} of iterations have been executed. We refer the reader to the cited work for full details.

We implemented SSLEA and applied it to ϕ-strings. After some exploratory experimentation, we found that it delivers best results with $\alpha = \text{range}(\phi)$, $n = 7$ and $n_{\text{it}} = 5000$ and we used these values in our assessment. In particular, we verified experimentally that increasing the number n_{it} of available iterations, even by a large amount, did not lead to better accuracy on the testing data.

4.1.2 Problem-specific baselines

We implemented two classifiers that embed a substantial amount of problem-specific knowledge.

The classifier that we call *Annotations-Co-occurrence* classifies a sentence s positively if and only if s contains at least two genes/proteins and at least one interactor—i.e., it is tightly tailored to this specific problem.

The classifier that we call *Annotations-LLL05-patterns* is built on results from [19], which proposes a method for identifying syntax alignment-patterns that describe interactions between genes and proteins in scientific text. An alignment-pattern is described in terms of sequences of salient POS annotations which must occur in a sentence, but that does not specify any constraint on type and quantity of further annotations that might occur between those salient annotations. The method exploits a fair amount of domain-specific knowledge for tuning tens of coefficients used for weighting alignment-pattern errors related with specific POS annotations, as well as with genes and interactors. The cited work provides a list of the 10 most frequent alignment-patterns learned from the *whole* corpus used in that paper, which is much larger than ours ($\approx$ 1000 sentences). We built a classifier which classifies a sentence s positively if and only if s matches at least one of the 10 alignment-patterns in the aforementioned list. Our corpus is composed, for $\approx 90\%$ of the sentences, of a strict subset of that corpus.

4.1.3 Established methods for text classification

Finally, we implemented two well-established schemes for text classification [30], one based on Naive Bayes and the other based on Support Vector Machines (SVM).

We pre-process each sentence s as follows: (i) we replace each occurrence of a string contained in $D_{\text{interactors}}$ with `_interactor`; (ii) we replace each occurrence of a string contained in D_{genes} with `_geneptn`; (iii) we convert s to lowercase; (iv) we replace each non alphabetic (i.e., `[^a-zA-Z_]`) character with a space; (v) we perform stemming to each word (except of `_geneptn` and `_interactor`). We execute steps i and ii in order to exploit the knowledge embedded in the dictionaries, which would otherwise not available to the classifier.

In order to build the classifier:

1. We build a sorted set W of words composed of all words occurring at least once in the learning sets S_L^+, S_L^-.

2. We transform each sentence s into a vector $\mathbf{f}'$ in which the ith element corresponds to the number of occurrences in s of the word $w_i \in W$.

3. We execute a *feature selection* procedure (which is detailed below) for identifying the k elements of $\mathbf{f}'$ which best discriminate between sentences in S_L^+ or S_L^-. Let $\mathbf{f}$ be the vector obtained by keeping only those k elements from $\mathbf{f}'$—i.e., $\mathbf{f}$ contains only the occurrence counts of the words selected in the feature selection procedure.

4. Finally, we train a binary classifier using the vectors $\mathbf{f}$ corresponding to the sentences in the learning sets S_L^+, S_L^-.

Having built the classifier, to classify a previously unseen sentence s we: (i) pre-process s as described above; (ii) obtain the corresponding feature vector $\mathbf{f}$; and, finally, (iii) input $\mathbf{f}$ to the trained classifier.

The feature selection procedure is based on the vectors $\mathbf{f}'$. This procedure takes two numerical parameters k, k', with $k' \gg k$, and works in two steps, as follows. In the first step we compute, for each ith feature the relative difference δ_i between its mean value across sentences of the two sets:

$$\delta_i = \frac{\left| \frac{1}{|S_L^+|} \sum_{S_L^+} f_i' - \frac{1}{|S_L^-|} \sum_{S_L^-} f_i' \right|}{\max_{S_L^+ \cup S_L^-} f_i'}$$

We then select the k' features with the largest δ_i—among those for which $\max_{S_L^+ \cup S_L^-} f_i' > 0$. In the second step we compute, for each ith feature among the k' selected at the previous step, the mutual information I_i with the label—the label being a binary value which is positive for elements in S_L^+ and negative for elements in S_L^-. We then select the k features with the greatest I_i.

We built two binary classifiers with this scheme, which we call *Words-NaiveBayes* and *Words-SVM*, respectively. Concerning the feature selection parameters k' and k, we set $k' = 1000$ for both the classifiers and we chose the value of k for which each classifier obtained the best accuracy on the learning sets, i.e., $k = 25$ for *Words-NaiveBayes* and $k = 50$ for *Words-SVM*. Concerning SVM parameters, we used a Gaussian radial kernel with the cost parameter set to 1.

4.2 Results

We executed a 5-fold cross-validation, i.e., we generated 5 different problem instances from the corpus at random. For each instance we used $\approx 80\%$ of the data for learning and $\approx 20\%$ for testing, i.e., we built the learning sets S_L^+, S_L^- (with $|S_L^+| = |S_L^-| = 188$) and left two testing sets S_T^+, S_T^- aside for assessing the accuracy of the generated classifiers (with $|S_T^+| = |S_T^-| = 40$). We used the very same learning sets and testing sets for all the considered methods—our method and the baseline methods described in the previous section. For ϕ-SSLEA and our method, which are stochastic, we repeated the execution 10 times for each fold.

Table 4 shows the results obtained by the 6 methods. The results are expressed in accuracy, FPR and FNR, averaged across the 5 folds: for the two stochastic methods, we also report the accuracy standard deviation of the 10 executions averaged across the 5 folds.

It can be seen that our method and Words-SVM obtain the highest accuracies ($\approx 74\%$) among those 4 methods which learn a classifier only from examples and dictionaries—i.e., without any problem-specific knowledge. GP is indeed able to learn syntax patterns from examples effectively, even in NLP scenarios of practical interest. We believe this result is particularly relevant.

Classifier	Accuracy avg	sd	FPR avg	FNR avg
Ann.-Co-occurrence	77.8		40.0	4.5
Ann.-LLL05-patterns	82.3		25.0	10.5
Words-NaiveBayes	51.3		25.0	95.0
Words-SVM	73.8		29.0	23.5
ϕ-SSLEA	59.8	3.8	44.0	33.5
Our method	73.7	1.7	23.5	22.5

Table 4: Results of our method and the 5 baselines.

$r_1 =$ `GENEPTN [ ^ RB ] [ ^ NNS VBN GENEPTN ] ++`
`GENEPTN [ ^ LRB DT NNS RRB ] [ ^ LRB NNS ]`
$r_2 =$ `. INOUN IN GENEPTN . [ ^ DT NN ]`

Figure 2: An example of a generated classifier composed of two regular expressions, shown using annotations, instead of symbols of U, for readability.

The accuracy of the two problem-specific classifiers is 82.3% and 77.8% for Annotations-LLL05-patterns and Annotations-Co-occurrence, respectively. We believe that the better accuracy exhibited by these methods is not surprising having considered that these methods build on a substantial amount of problem-specific knowledge, as clarified above. Indeed, the accuracy of Annotations-Co-occurrence is only slightly better than our method.

It is interesting to note that the accuracy obtained by ϕ-SSLEA is much worse than ours although the two approaches are based, broadly speaking, on similar tools—evolutionary learning of a DFA vs. evolutionary learning of sets of regular expressions. While this result might appear somewhat surprising—as pointed out above, ϕ-SSLEA exhibits state-of-the-art performance in DFA learning—we believe the reason is because benchmark problems in DFA learning consider short sequences of binary symbols, with training data drawn uniformly from the input space. Settings of this sort do not fit the needs of practical NLP applications, which have to cope with much longer sequences of symbols, from a much larger alphabet, not drawn uniformly from the space of all possible sequences. Our interpretation is corroborated by earlier claims from different authors: benchmark problems for DFA learning from examples are not inspired by any real world application [10] and the applicability of the corresponding learning algorithms to other application domains is still largely unexplored [6].

Figure 2 shows one of the classifiers obtained during our experimentation, composed of two regular expressions. For the sake of comprehension, expressions are shown using annotations (A') instead of symbols of U. It can be seen that the generated expressions include salient annotations (`GENEPTN` and `INOUN`) although those annotations were not given any special status. The two patterns are not easily readable, which is not surprising since we did not include any mechanism for favoring readable expressions. Indeed, readability could be a valuable objective to be pursued and we plan to investigate this respect in future work.

5. CONCLUDING REMARKS

We presented an evolutionary method for learning syntax patterns in natural text from examples. We applied this method to a sentence classification problem from the biomedical domain that is practically relevant and very challenging.

Our method is based on GP and builds on recent results for the automatic generation of regular expressions from examples of the desired behavior. We propose a technique for representing sentences as strings of symbols which can be manipulated by common regular expressions. Symbols correspond to POS annotations augmented with problem-specific dictionaries. Working at the abstraction level of POS annotations has many practical advantages, including modularity and the possibility of leveraging the steadily improving state-of-the-art in this area. The number of patterns to be learned from a set of example sentences is not known in advance, but is instead automatically determined by means of a separate-and-conquer procedure.

We assessed experimentally our method on a challenging corpus of 456 hand-labeled sentences and compared it against 5 significant baseline methods. We obtained good results which indicate that GP may indeed learn syntax patterns from examples effectively, even in NLP scenarios of practical interest.

6. REFERENCES

[1] L. Araujo. Symbiosis of evolutionary techniques and statistical natural language processing. *Trans. Evol. Comp*, 8(1):14–27, February 2004.

[2] Lourdes Araujo. How evolutionary algorithms are applied to statistical natural language processing. *Artif. Intell. Rev.*, 28(4):275–303, December 2007.

[3] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. Automatic generation of regular expressions from examples with genetic programming. In *International Conference on Genetic and evolutionary computation (GECCO)*, pages 1477–1478. ACM, 2012.

[4] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47:72–80, 2014.

[5] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Learning text patterns using separate-and-conquer genetic programming. In *18-th European Conference on Genetic Programming (EuroGP)*, 2015.

[6] Josh Bongard and Hod Lipson. Active coevolutionary learning of deterministic finite automata. *The Journal of Machine Learning Research*, 6:1651–1678, 2005.

[7] Jakramate Bootkrajang, Sun Kim, and Byoung-Tak Zhang. Evolutionary hypernetwork classifiers for protein-proteininteraction sentence filtering. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 185–192, New York, NY, USA, 2009. ACM.

[8] Markus Bundschus, Mathaeus Dejori, Martin Stetter, Volker Tresp, and Hans-Peter Kriegel. Extraction of semantic biomedical relations from text using conditional random fields. *BMC Bioinformatics*, 9(1):207, 2008.

[9] Edmund K Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, 2004.

[10] Orlando Cicchello and Stefan C Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research*, 4:603–632, 2003.

[11] Edwin D De Jong and Jordan B Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.

[12] Colin De La Higuera. A bibliographical study of grammatical inference. *Pattern recognition*, 38(9):1332–1348, 2005.

[13] Guy De Pauw. Evolutionary computing as a tool for grammar development. In *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartI*, GECCO'03, pages 549–560, Berlin, Heidelberg, 2003. Springer-Verlag.

[14] Bart Decadt, , Bart Decadt, Véronique Hoste, Walter Daelemans, and Antal Van Den Bosch. Gambl, genetic algorithm optimization of memory-based wsd. In *In Proceedings of ACL/SIGLEX Senseval-3*, pages 108–112, 2004.

[15] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36(1):1–27, 2011.

[16] Pedro G Espejo, Sebastián Ventura, and Francisco Herrera. A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121–144, 2010.

[17] Katrin Fundel, Robert Küffner, and Ralf Zimmer. Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007.

[18] Sonal Gupta, Diana L MacLean, Jeffrey Heer, and Christopher D Manning. Induced lexico-syntactic patterns improve information extraction from online medical forums. *Journal of the American Medical Informatics Association*, 21(5):902–909, 2014.

[19] Jörg Hakenberg, Conrad Plake, Ulf Leser, Harald Kirsch, and Dietrich Rebholz-Schuhmann. LLL'05 challenge: Genic interaction extraction-identification of language patterns based on alignment and finite state automata. In *Proceedings of the 4th Learning Language in Logic workshop (LLL05)*, pages 38–45, 2005.

[20] Laurence Hirsch, Robin Hirsch, and Masoud Saeedi. Evolving lucene search queries for text classification. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1604–1611. ACM, 2007.

[21] Evan J Hughes. Evolutionary many-objective optimisation: many once or one many? In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 222–227. IEEE, 2005.

[22] Seth Kulick, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, Lyle Ungar, Scott Winters, and Pete White. Integrated annotation for biomedical information extraction. In *Proc. of HLT/NAACL*, pages 61–68, 2004.

[23] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, page 1–12. Springer, 1998.

[24] W.B. Langdon and M. Harman. Optimizing existing software with genetic programming. *Evolutionary Computation, IEEE Transactions on*, 19(1):118–135, Feb 2015.

[25] Jiexun Li, Zhu Zhang, Xin Li, and Hsinchun Chen. Kernel-based learning for biomedical relation extraction. *Journal of the American Society for Information Science and Technology*, 59(5):756–769, 2008.

[26] Marina Litvak, Mark Last, and Menahem Friedman. A new approach to improving multilingual summarization using a genetic algorithm. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 927–936, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[27] Simon M Lucas and T Jeff Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

[28] Hoifung Poon, Chris Quirk, Charlie DeZiel, and David Heckerman. Literome: Pubmed-scale genomic knowledge base in the cloud. *Bioinformatics*, 2014.

[29] Luis Rodríguez, Ismael García-Varea, and José A. Gámez. On the application of different evolutionary algorithms to the alignment problem in statistical machine translation. *Neurocomput.*, 71(4-6):755–765, January 2008.

[30] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[31] J. Ignacio Serrano. Evolutionary algorithm for noun phrase detection in natural language processing. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computing (IEEE Computer Society*, 2005.

[32] Martin Volk, Bärbel Ripplinger, Špela Vintar, Paul Buitelaar, Diana Raileanu, and Bogdan Sacaleanu. Semantic annotation for concept-based cross-language medical information retrieval. *International Journal of Medical Informatics*, 67(1):97–112, 2002.

[33] Akane Yakushiji, Yusuke Miyao, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 284–292. Association for Computational Linguistics, 2006.

[34] Lin Yao, Cheng-Jie Sun, Xiao-Long Wang, and Xuan Wang. Relationship extraction from biomedical literature using maximum entropy based on rich features. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, volume 6, pages 3358–3361, July 2010.