Association rules for grammar inference

Alejandro Barredo Arrieta

21st June 2018

1 Introduction

In the last ten-twenty years, grammar inference has become an important topic for AI research as a path to explore natural language processing as opposed to linguistics. The aim is to extract the rules of a grammar from examples of the language, allowing to generalize the underlying rules and unveiling its use for language correction and understanding.

Picture the moment in which a machine is able to tell us if a sentence is correct from every perspective (orthographically, grammatically and within the context). Such an ability could empower machines with the task of "teaching" (helping self teaching humans) with the arduous job of learning a new language or even improving their own by allowing a continuous feedback over what the user writes, without the need of another human. Another beautiful use of this kind of knowledge could be the improvement of automatic text readers that extract information. It could upgrade such information by means of a more thorough knowledge of what and how it is written. Language may be natural or not, but this research focuses on natural languages because of the complexity they convey. However, inference techniques can be used to extract the grammar rules from synthetic communication protocols as well. There are many language (grammar) acquisition theories that focus on empirical observations, trying to answer how children learn their surrounding grammar based on the small set of sentences they have listened to. Although this seems to be a very interesting path to follow, it reduces the possibilities brought to us by the Internet and big data.

Employing huge amounts of data may work on our behalf, allowing the implementation of broadly less complex algorithms to achieve the same results. Furthermore, humans tend to understand the concepts represented by words in a communication environment, i.e. a door is a noun, and this may be encoded into the algorithm easing its job of inferring such syntactic categories. However this may not be useful in every application, since it demands a thorough understanding of the language being used before hand. Another issue that may be argued is the lack of understanding that algorithms have compared to humans, the ability of introducing such knowledge into the code could be the way to go. However, there is much discussion around the ability of computers to include semantics, since they are just symbol manipulators. Although in this situation the machine is not intended to understand, but to have some model that represents what humans call semantics.

Prior to start discussing the research that we will be looking at to steer our own, the different categories where languages could be located must be mentioned. Languages can be built from regular or context-free grammars, the former is linearly regular and has a finite alphabet along with a maximum length for its sentences, contrarily, the later is formed by an infinite set of symbols (alphabet) and it is not linearly regular. There has been a heated discussion around the context-freeness of natural language grammars since the 70s. Furthermore, the invention of context free grammars ([Cho02]) was an attempt to categorize natural language grammars but it has not lived up to its purpose. In our study, we will be treating written natural language as a regular grammar thank to the work done in [Shi85] to dissolve the discussion around it, since nothing has been said after it to place natural languages as context-free languages.

The most common feature present in the majority of the studies involves the construction of a DFA (Deterministic Finite state Automata) as the central issue. There have been many different attempts to build such DFAs and many give good results under some circumstances. Some of the approaches come from the use of Genetic algorithms as the main tool to build the DFA, others use Neural Networks to achieve the job while there are other that used recursive algorithms based on machine learning concepts. Another common procedure shown in the studies is the use of positive and negative examples in order to build up that DFA. However, there is some discussion about the type of examples needed to build a good DFA (minimum state DFA or canonical DFA). This article covers the steps and concepts found in the literature and tries to sketch a different

perspective to tackle the issues. In the second section, the concepts and terminology needed for understanding the techniques are presented. In section 3, the techniques known to be somehow successful are shown. In section 4, the sketch of the idea proposed is explained. In section 5, the reasons to consider such a proposal a promising one are explored. In section 6, the results are discussed and finally in section 7 conclusions are shown.

2 Terminology and concepts

The syntax of a language is usually ruled by the grammar laws involved in such language. These rules along with the available dictionary are the ones in charge of creating correct sentences within the language. Grammar inference is the tool used to infer such rules from the use of the language itself. It is usually defined as the process of learning a target grammar from a set of labeled examples (sentences). The task of grammar inference is usually modeled as the induction of an automata that respects the underlying rules of the language.

2.1 Regular grammar inference

Given a finite set of positive examples and a finite (possibly empty) set of negative examples of sentences that belong to the target language, find the regular grammar $G*$ that is equivalent to the target grammar G. There are different ways to represent a regular grammar, this choice comes as a design point. It can be represented as a set of production rules, a regular expression, a deterministic finite state automaton (DFA) or a non-deterministic finite state automaton (NFA). Most of the related studies have chosen DFAs as the representational structure due to its powerful characteristics: they are easy to understand, there exists a unique minimum state DFA corresponding to any regular grammar, there exist efficient polynomial algorithms for operations with DFAs (minimization, equivalence testing, inclusion testing ...), and are easily implementable in the hardware world.

2.2 Formal grammars

A formal grammar is a 4-tuple $G = (V_N, V_T, P, S)$ where V_N is the set of non-terminals, V_T is the set of terminals, P is the set of rules and $S \in V_N$ is a special symbol called start symbol. The production rules are of the form $\alpha \to \beta$ where α, β are sentences over the alphabet $(V_N \cup V_T)$ and α contains at least one non-terminal. Valid sentences of the language are sequences of terminal symbols V_T and are obtained by repeatedly applying the production rules as it is covered next. To summarize, a formal grammar is a set of rules to create strings. When writing a new sentence, it starts by adding a start symbol. Then applying the rules of the grammar more symbols are added until a non-terminal is placed. Every sentence that can be formed like this conform the language of the grammar.

2.3 Lattice of partitions

During the article, a lattice of partitions of possible DFA is mentioned. Such lattice is formed as follows. First the Maximal canonical automaton (MCA) is built from the set of positive examples available to the learner. A maximal canonical automaton is a DFA that accepts every string in the positive set and no other. This MCA allows for a path from a start state to an accepting state for every sentence in the set.
The lattice of possible grammars is built by merging the states of this MCA forming partitions. Each partition is an element of the lattice. The language represented by any of these partitions is a superset of the language of the positive set. Hence, successive merging generates more general languages.
Each element of the lattice corresponds to a FSA constructed by taking as states those cells found in the partition. Then, the start state is the same as that from the M_{S^+}. The accepting states are those that contain one or more accepting states of M_{S^+}. The alphabet is the same as in the sample set and the transition function is defined on the basis of the transitions in M_{S^+}. The lattice of possible grammars is ordered by the grammar cover relation, meaning that each grammar that follows another into a higher level contains its predecessor. The more specific or equal (MSE) test can be performed easily by just comparing the cells in the partitions. The image shown in 1 tries to illustrate the concept of a lattice of partitions from a four element set. In the image, coarser partitions are linked to finer ones by lines going down.

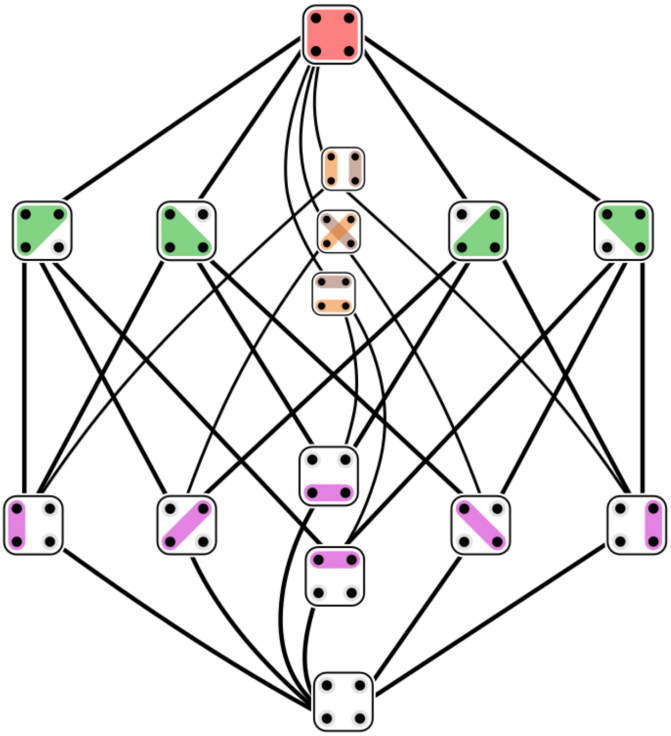

Figure 1: Lattice of partitions

2.4 Inference of Context Free Grammars

Context Free Grammars (CGF) represent the next level of abstraction after regular grammars in the Chomsky hierarchy of formal grammars. A CGF is a formal language grammar $G = (V_N, V_T, P, S)$ where the production rules are of the form: $A \rightarrow \alpha$ where $A \in V_N$ and $\alpha \in (V_T \cup V_N)^*$. Most work in inference of context free grammars has focused on learning the standardized Chomsky Normal form (CNF) of CFG in which the rules are of the form $A \rightarrow BC$ or $A \rightarrow a$ where $A, B, C \in V_N$ and $a \in V_T$. The example grammar for simple declarative English language sentences is in CNF. Given a CFG G and a string $\alpha \in V_T^*$ we are interested in determining whether or not α is generated by the rules of G and if so, how it is derived.
A parse tree graphically depicts how a particular string is derived from the rules of the grammar. The non-leaf nodes of a parse tree represent the non-terminal symbols. The root of each sub-tree together with its children (in order from left to right) represents a single production rule. Parse trees provide useful information about the structure of the string or the way it is interpreted. Often, in learning context free grammars, parse trees obtained from example sentences are provided to the learner. Parse trees simplify the induction task since they provide the learner with the necessary components of the structure of the target grammar.
The theoretical limitations that concern regular grammar inference also relate to context free grammar inference since the set of regular grammars is included in the set of context free grammars.

2.5 Finite State Automata

A deterministic finite state automata is a quintuple $A = (Q, \delta, \Sigma, q_0, F)$ where, Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and δ is the transition function: $Q \times \Sigma \rightarrow Q$. $\delta(q, a)$ denotes the state reached when the DFA in state q reads the input letter a. A state $d_0 \in Q$ such that $\forall a \in \Sigma, \delta(d_0, a) = d_0$ is called a dead state. The extension of δ to handle input strings (concatenations of symbols in Σ)is standard and is denoted by $\delta*.\delta*(q, \alpha)$denotes the state reached from q upon reading the string α. A string α is said to be accepted by a DFA if $\delta*(q_0, \alpha) \in F$. Strings accepted by the DFA are said to be positive examples of the language of the DFA. The set of all strings accepted by the DFA A is its language, $L(A)$. On the contrary, strings not accepted by the DFA are negative examples of the language of the DFA. The language of the DFA is called a regular language. DFA can be represented by state transition diagrams.
A non deterministic state automaton (NFA) is defined just like the DFA except that the transition function maps values from $Q \times \Sigma \rightarrow 2^Q$, this is, the resulting state after a certain input letter can be more than one.

Regular grammar inference is a hard problem in that regular grammars cannot be correctly identified from positive examples alone [Gol67]. Furthermore, it has been shown that there exists no efficient learning algorithm for identifying the minimum state DFA that is consistent with an arbitrary set of positive and negative examples [Gol78]. Efficient algorithms for identification of DFA assume that additional information is provided to the learner. This information is typically in the form of a set of examples S that satisfies certain properties or a knowledgeable teachers responses to queries posed by the learner. To exemplify the concept of a deterministic finite state automata the next figure shows the diagram of a simple DFA where we see that from an state, under a certain input, there is only a deterministic change of states that happens.

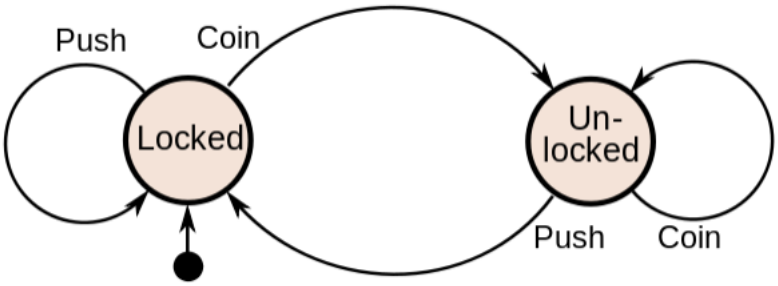

Figure 2: Deterministic finite state automata

2.6 Stochastic Finite State Automata

A stochastic finite state automaton SFA is defined as $A = (Q, \Sigma, q_0, \pi)$ where Q is the finite set of N states (numbered $q_0, q_1, q_2, ..., q_{N-1}$), Σ is the finite alphabet, q_0 is the start state, and π is the set of probability matrices. $p_{ij}(a)$ is the probability of transition from state q_i to q_j on observing the symbol a of the alphabet. A vector of N elements π_f represents the probability that each state is an accepting state. The language generated by a SFA is called a stochastic regular language and is defined as $L(A) = \omega \in \Sigma^* | p(\omega) \neq 0$, the set of sentences that have non-zero probability of being accepted by the SFA. If we compare this automata with the one before, the difference is that this time the state changing is managed by a probability distribution as shown in figure 3.

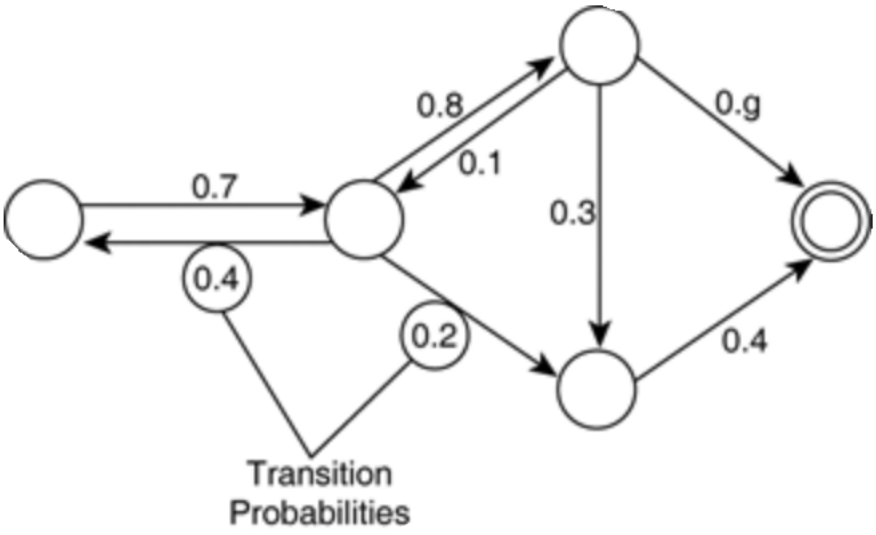

Figure 3: Stochastic Finite State Automata

2.7 Hidden Markov Models

Hidden Markov Models (HMM) are the widely used generalizations of stochastic finite state automata where both the state transitions and the output symbols are governed by probability distributions. HMMs have been successful in speech recognition and cryptography. Formally a HMM is comprised of a finite set of states

Q, a finite alphabet Σ, a state transition probability matrix A (NxN), where a_{ij} represents the probability of reaching state q_j from the state q_i, an observation symbol probability matrix B (NxN) where B_{ij} is the probability of observing the symbol σ_j in the state q_i and the initial state probability matrix π $(Nx1)$ where π_i is the probability of the model for starting in state q_i. In comparison with SFAs, the symbols in HMMs are associated with individual states and not with transitions. HMMs satisfy the Markovian property which states that the probability of the model being in a particular state at any given time depends only on the state it was in at the previous instant. Further, since each symbol is associated with possibly several different states, given a particular symbol is not possible to directly determine the state that generated it. It is for this reason that these Markov Models are referred to as hidden. The probabilistic nature of HMMs makes them suitable for their use in processing temporal sequences. For example, in the case of speech recognition, a separate HMM is created to model each word of the vocabulary. Given an observed sequence of sounds, one then determines the most likely path of the sequence through each model and selects the word associated with the most likely HMM. As before, we try to exemplify a HMM with the figure 4.

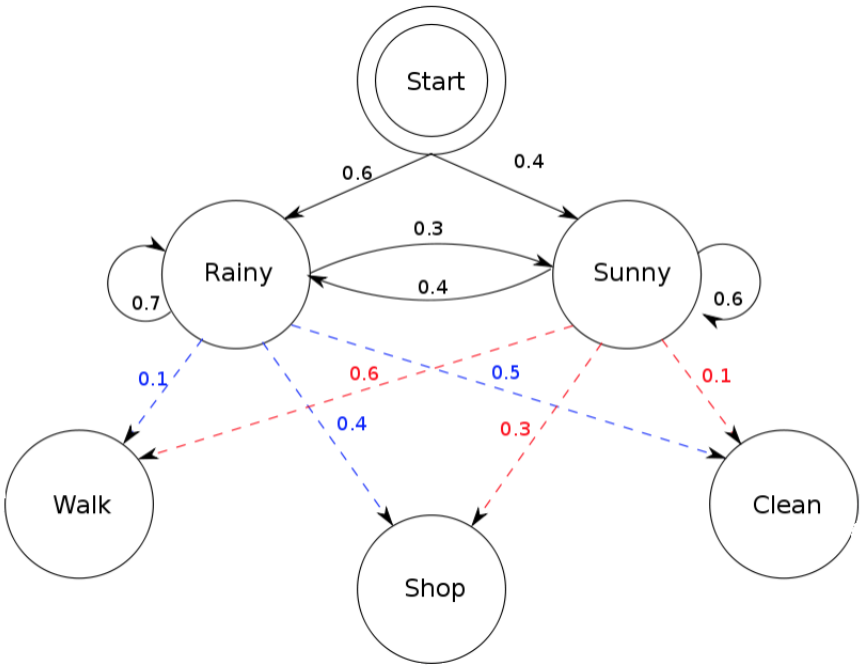

Figure 4: Hidden Markov Model

2.8 Search Space

Regular grammar inference can be formulated as a search problem in the space of all finite state automata (FSA). Clearly, the space of all FSA is infinite. One way to restrict the search space is to map a set of positive examples of he target DFA to a lattice of FSA which is constructed as follows [PC78]. Initially, the set of positive examples is used to define a prefix tree automaton (PTA). The PTA is a DFA with separated paths from the start state leading to an accepting state for each string in S^+. The PTA accepts only sentences in S^+. Hence the lattice is the set of all partitions of the set of states of the PTA together with a relation that establishes a partial order on the elements. Each element on the lattice is called a quotient automaton and is constructed by merging together the states that belong to the same block of the partition. Then the universal DFA is obtained by merging all the states of the PTA into a single state that is the most general element of the lattice. General procedures start with either the PTA or the universal DFA and use search operators such as state merging and state splitting to generate new elements within the search space.
By mapping the positive examples to a lattice of FSA, we reduce the search space into a finite space. Only if the set of positive examples is structurally complete, we can assure that the minimum state DFA equivalent to the target is included in the lattice.

2.9 Version Space

A version space can be defined to be the set of all generalizations consistent with a given set of instances. This is just a set, with no other structure and no associated algorithm. However, a version space strategy is also a particular induction technique, based on a compact way of representing the version space. The central idea of the version space strategy is that the space of generalizations defined by the representation language can be partially ordered by generality. In other words, an algorithm that follows a version space strategy is one that checks a set of previously formed hypotheses by means of positive and negative examples.

3 Known techniques for Regular grammar inference

This section covers the known techniques available in the literature to infer regular grammars from languages. Different approaches to tackle the problem are presented, such as: Search problems, Membership query based algorithms and Neural network based algorithms.
First of all, two higher order groups should be differentiated: approaches to infer regular grammars and methods to infer context-free grammars. Subsections 3.1 to 3.5 cover those algorithms related with regular grammar retrieval, leaving the rest for context-free grammars. However, between the first 5 algorithms that are mentioned we encounter another two clear subgroups. One that intends to include some type of "knowledge" (symbolic approach) into the algorithm by means of a teacher to which the learner can pose questions. The other type uses a more obscure approach in which the algorithm includes no knowledge whatsoever.
The former methods are presented in the first two subsections, the later techniques are covered in the last three subsections. The first method presented uses a well know technique for maximum descriptive subgroup discovery exported to grammar inference by building two starting states, one allowing everything in the available examples and the other none. Then it is refined until they reach a result that matches both states. The second method is based on the hypothesis testing approach. The algorithm is able to create hypotheses and the teacher is able to refute or approve them. The last three methods of the first group (regular grammars) cover the aspect of using genetic algorithms, recurrent neural networks and and ordered depth first search through the lattice of partitions available. The last subgroup referring to context-free grammars is not clearly divided into the same two groups as the previous approaches. All of the techniques shown for context-free grammars follow the second group of the regular grammar techniques that have been already introduced, those not including knowledge in the algorithm.

3.1 Bi-directional Search using Membership queries

This method [PH93][PH96] assumes that a structurally complete set of positive examples is presented to the learner along with a teacher that can answer membership queries (whether a sentence corresponds to the target grammar). The lattice is represented using two sets of finite state automata. One is a set (S) of the most specific FSA which is initialized to the PTA. The other is a set (G) of the most general FSA which is initialized to the universal DFA obtained by merging all the states of the PTA. This representation is gradually refined by eliminating those elements of the lattice that are found to be inconsistent with the observed data. At each step two automata (one from each set) are selected and compared for equivalence. If they are not equivalent, the shortest string proposed by one of the automaton but not the other is shown to the teacher. If the response of the teacher is negative, the FSAs in S that accept such sentence are removed from the set, and the FSAs in G that accept the sentence are specialized by splitting their states until the FSA does not accept the sentence. Overall, the membership queries serve their purpose to generalize the elements in S and specialize the elements in G, bringing them closer together. Although this method guarantees convergence, the speed of such convergence is not fixed and in a worst case scenario can be exponential in the size of the PTA. However, the authors mentioned in this section also propose a iterative algorithm to overcome the unavailability of structurally complete sets of positive examples in "real life" induction learning contexts. In this case, they follow the same idea as before but, each time the learner finds no member to be discarded, the teacher proposes new examples. With these new examples the learner needs to update the lattice and restart the process of candidate elimination. Whenever the example set represents a structurally complete set of examples, the iterative aspect of the algorithm is ended.
Algorithm

1. Set $S = P_0$ and $G = P_{Em-1}$.
2. While there exists an element $P_i \in S$ and $P_j \in G$ such that the corresponding FSA $M_i \neq M_j$ pick the shortest string $\in L(M_i - M_j)$ or $L(M_j - M_i)$ and pose the query $y \in L(G)$? Based on the teacher's response modify Θ as described below.

3. Return the FSA corresponding to the partition to which the search of the lattice (Ω) converges

Candidate Elimination

1. If y is a positive example:
   (a) Remove any Partition $P_k \in G$ for which the corresponding FSA M_k rejects y.
   (b) Minimally generalize any partition $P_l \in S$ if M_l does not accept y till the FSA corresponding to the generalization accepts y. Retain only those partitions in S that are MSE some partition in G.
   (c) Remove any element from S that is MGE some other element in S.
2. If y is a negative example
   (a) Remove any partition $P_k \in S$ for which the corresponding FSA M_k accepts y.
   (b) Minimally specialize any partition $P_l \in G$ if M_l accepts y till the FSA corresponding to the specializations do not accept y. Reatin only those partitions in G that are MGE some partition in S.
   (c) Remove any element from G that is MSE some other element in G.

3.2 The L* Algorithm

Similar to the method shown before, this one also uses a teacher that allows the algorithm to evaluate queries and check for grammar equivalences. The $L*$ algorithm infers a minimum state DFA corresponding to the target with the help of a minimally adequate teacher [Ang87]. The authors present in this paper the concept of minimally adequate teacher. Such a teacher conveys two abilities. One is the capacity to answer a membership query with a yes or no (MEMBER()). The other is the ability to answer a conjecture describing a a regular set. The Teacher has to be able to evaluate the equivalence between the regular set and the unknown language and give a counterexample if it is not.
Unlike the approaches described so far the $L*$ algorithm does not search a lattice of FSA. Instead it constructs a hypothesis DFA by posing membership queries to the teacher. The learner maintains a table where the rows correspond to the states of the hypothesis DFA and the columns correspond to suffix strings that distinguish between pairs of distinct states of the hypothesis. In the table, each state is labeled by the string that led to such state. The start state is always labeled with λ and the states reached from these start states are labeled in alphabetical order (a, b, c...). The column label is used to map the states that would be reached from such states after reading the label of the column. Membership queries are composed by chaining row labels and column labels. The intersection cells are numbered with a one or a zero representing the acceptance of such a string. States are presumed to be equivalent whenever all the entries for their corresponding rows are identical. When facing two states are not equivalent because of a symbol, the table is enlarged adding a column to distinguish the two states. If the learner is not able to further distinguish any two states, it poses a equivalence query to the teacher to test if the hypothesis is equivalent to the target grammar. If the answer is yes, the learner returns the current hypothesis as the result. However, when the teacher answers no, it also provides a counterexample that is accepted by the target DFA and not by the hypothesis or vice-versa. This addition modifies the columns and rows of the learner, producing a new process of membership queries and specification followed by another equivalence query. Such process will continue until the teacher responds with a yes to the equivalence test.
Algorithm

1. Initialize S and E to λ
2. Ask membership queries for λ and each $a \in A$.
3. Construct the initial observation table (S,E,T).

   Repeat

4. While(S,E,T) is not closed or not consistent:
   (a) If (S,E,T) is not consistent
   - find s_1 and s_2 in S, $a \in A$, and $e \in E$ s.t.
   - row(s_1) = row(s_2) and T($s_1 \cdot a \cdot e$) $\neq T(s_2 \cdot a \cdot e)$
   - add a $\cdot$ e to E.

- extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.

(b) If(S,E,T) is not closed.

- find $s_1 \in S$ and $a \in A$ s.t.
- row($s_1 \cdot a$) is different from row(s) for all $s \in S$,
- extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.

5. Once (S,E,T) is closed and consistent, let M = M(S,E,T)
6. Make the conjecture M
7. if the Teacher replies with a counterexample t, then
   - add t and all its prefixes to S
   - extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.

   Until the Teacher responds yes to the conjecture M
8. Halt and Output M

3.3 Randomized Search

As opposed to the previous ones, this technique falls into the second group mentioned in the description. It does not contain knowledge. However, the transformation and operations done within the method had been sculpted with knowledge about the problem. Genetic algorithms are also another approach to explore a search space [Hol92]. They usually work by evolving a randomly generated population based on the survival of the fittest principle of Darwinian evolution. A unique codification of the individuals in the search space is created (chromosomes). A fitness function is designed to asses the quality of an individual. In each generation, a selection of individuals is taken based on some model (the fittest get to mate or randomly choose from the population ...). Then, mating takes place using crossover functions designed to inherit the most important features (fitness-wise) of the parents to their children (i.e. if some part of the chromosome of an individual is highly rewarded by the fitness function, the crossover function should not try to change such part). Finally, mutation functions can also be used to alter the chromosomes (mutations are small variations that are applied to the individuals, thought for introducing some variation into an otherwise stable system with the intention of exploring a bigger chunk of the search space).
[DMV94] The learner is given a set of positive and negative examples ($S = S^+ \cup S^-$). A PTA is constructed from the positive examples. The initial population is a random selection of elements from the set of partitions of the set of states of the PTA. Each element of the initial population is thus a quotient automaton belonging to the lattice of FSA constructed from the PTA. The fitness function considers the number or states and the number of mis-classifications in the S^- as variables. In each generation a sub-population is randomly selected based on their fitness for reproduction applying structural mutation and structural crossover. Structural mutation involves randomly selecting an element from one of the existing blocks of the partition and randomly assigning it to one of the other blocks or creating a new block. Structural crossover involves choosing one block at random from each parent and both offspring inherit the block obtained by merging the selected ones. The remaining blocks for the offspring are obtained by taking the difference of the blocks of the partitions corresponding to the parents that were not chosen above with the common block inherited by both offspring. The offspring generated are valid partitions belonging to the lattice. These are added to the original population. a fitness-proportionate selection scheme that assigns high probability to individuals with higher fitness is used to randomly select the population for the next generation.

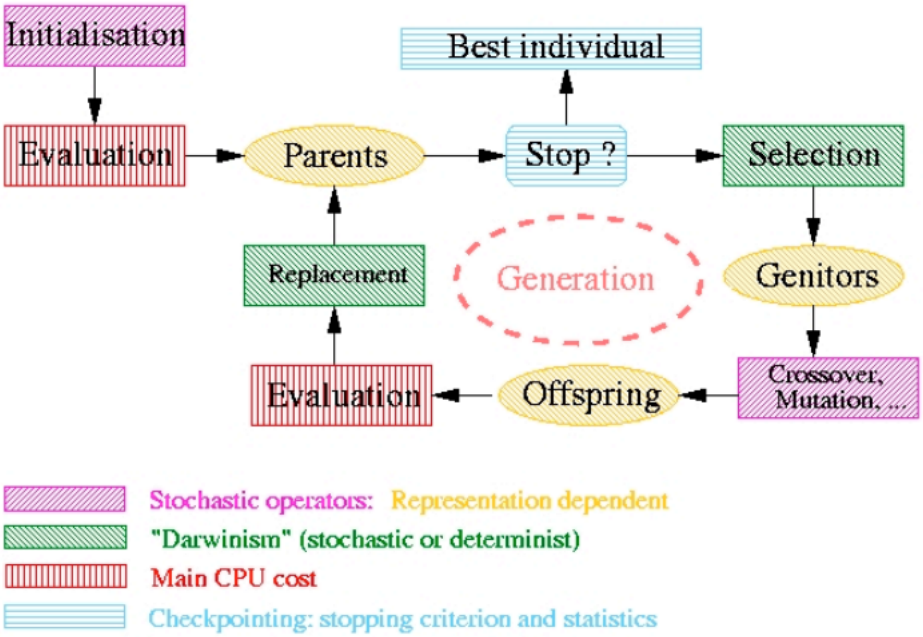

Figure 5: Genetic algorithm structure

3.4 Connectionist Methods

Following the previous approach this one also falls inside the non-knowledge group. Connectionist methods use neural networks to infer grammars. Their power resides on their ability to work with much more smaller sets and their scalability to bigger cases. Most connectionist language learning approaches specify the learning task in classical terms, for example using an automaton to parse grammatically correct sentences of the language and then implement these using a suitable neural network architecture such as recurrent neural networks RNN. Connectionist methods train a NN to fit the example set and then they extract the DFA from the inner structure generated in the neural network. As shown in [GCM+91], they exploit a recurrent neural network to achieve results. The main characteristic distinguishing RNN from normal NNs is that the formers exhibit a dynamic behavior. Their input neurons work in a sequential way instead of a simultaneous one capturing a time dependent flow, as in sentences. Throughout their paper, they present a method to extract a DFA from the inner structure of the RNN. Following the assumption that once the RNN has learnt the grammar, it has also partitioned the state space into fairly well-separated distinct regions. To achieve the resulting DFA they divide the neuron state space into q equal in size partitions, and each dimension of the partitioned state space into q equal size partitions again so resulting in a q^N equal N-dimensional volumes partitioning each neuron's state space. The paper, shows that there is a partition q such that, given a specific one of these partitions P_0, the locus of points in P_0 will map under g into another locus that is completely contained within one other single partition P_1, and by repeating this process they manage to find the DFA inside the trained RNN.

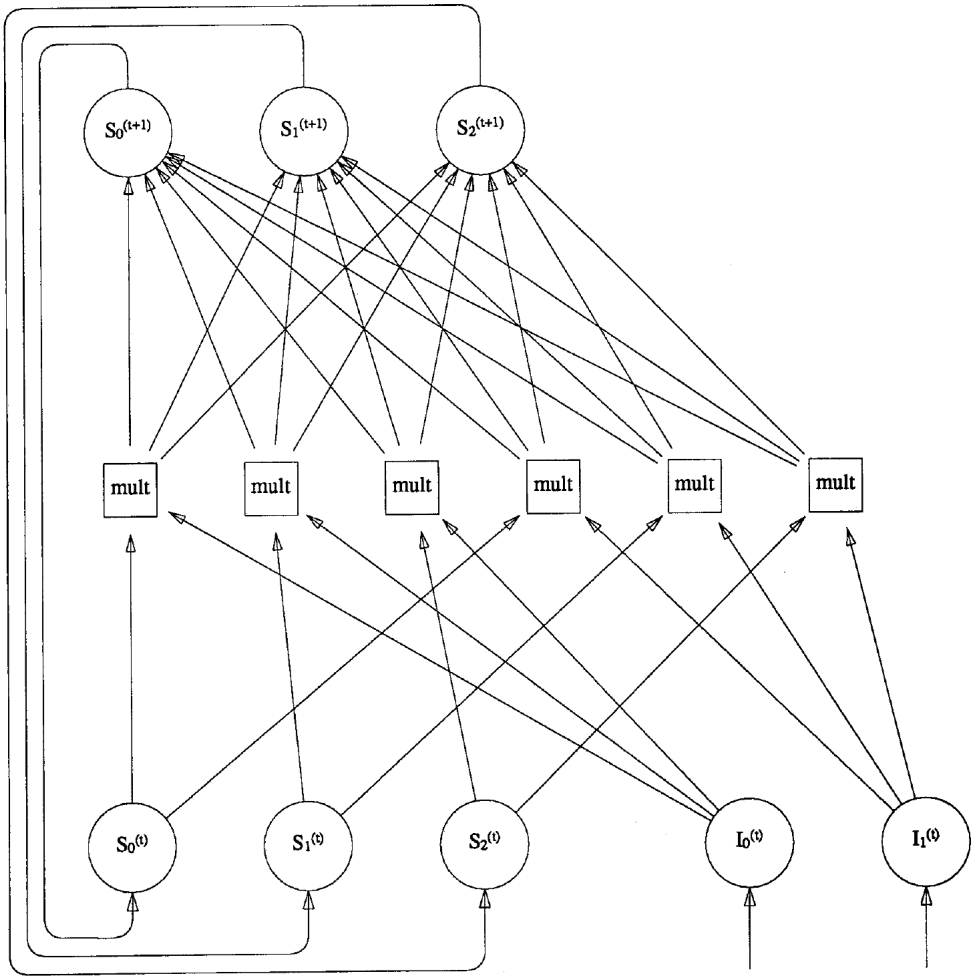

Figure 6: RNN Architecture

3.5 Ordered Depth-First Search with Backtracking

This technique is the last one related to regular grammar inference and it also goes along the lines of the last method which did not have knowledge within the algorithm. The regular positive and negative inference $RPNI$ algorithm performs an ordered depth first search of the lattice guided by the set of negative examples S^- and in polynomial time identifies a DFA consistent with a given sample $S = S^+ \cup S^-$ [OG92]. Furthermore if S happens to be a superset of a characteristic set for the target DFA, then the algorithm is guaranteed to return a canonical representation of the target DFA. A characteristic set of examples $S = S^+ \cup S^-$ is such that S^+ is structurally complete with respect to the target and S^- prevents any two states of the PTA of S^+ that are not equivalent to each other from being merged together. The algorithm first constructs a PTA for S^+. The states of the PTA are numbered in standard order as follows: The set of strings that lead from the start state to each individual state of the PTA is determined. The strings are sorted in lexicographic order. Each state is numbered based on the position of the corresponding string in the sorted list. The algorithm initializes the PTA to be the current solution and systematically merges the states of the PTA to identify a more general solution that is consistent with S.

3.6 The Alergia Algorithm for Learning SFA

For the first method treating context-free grammars we have Alergia. [CO94] Carrasco and Oncina developed this algorithm for inferring deterministic stochastic finite state automatas (DSFA). A DSFA is a SFA where for each state $q_i \in Q$ and symbol $a \in \Sigma$ exists at most one state q_j such that $p_{ij}(a) \neq 0$. Alergia is based on a state merging approach and is quite similar to the RPNI algorithm for inference of DFA. A prefix tree automaton PTA is constructed from the given set of positive examples S^+ and its states are numbered in standard order.

The initial probabilities π and π_f are computed based on the relative frequencies with which each state and transition of the PTA is visited by the examples in S^+. A quadratic loop merges the states of the PTA in order. However unlike the RPNI algorithm where the states merged were controlled by a set of negative examples S^-, Alergia merges states that are considered to be similar in terms of their transition behavior (described by the states reached from the current state) and their acceptance behavior (described by the number of positive examples that terminate in the current state). The determination of similarity is statistical in nature and is controlled by a parameter which ranges between 0 and 1. The probabilities π and π_f are re-computed after each state merge. The algorithm is guaranteed to converge to the target stochastic finite state automaton in the limit when a complete sample is provided. The worst case of Alergia is cubic in the sum of the lengths of examples in S^+.

3.7 Bayesian Model Merging Strategy

Stolcke and Omohundro present a more general approach to the HMM learning problem. Their approach is a Bayesian model merging strategy [SO94] which facilitates learning the HMM structure as well as the model parameters from a given set of positive examples. The first step constructs an initial model comprising of a unique path from the start state q_1 to a final state q_f for each string in the set of positive examples S^+. This is similar to the PTA constructed by the Alergia algorithm for learning SFA. The initial probabilities are assigned as follows: The probability of entering the path corresponding to each string from the start state is uniformly distributed. Within each path the probability of observing the particular symbol at each state and the probability of taking the outgoing arc from the state are both set to 1. Next, a state merging procedure is used to obtain a generalized model of the HMM. At each step the set of all possible state merges of the current model M_i are considered and two states are chosen for merging, such that, the resulting model M_i maximizes the posterior probability $Pr(M_{i+1}|S^+)$ of the data. By Bayes's rule, $Pr(M_{i+1}|S^+) = Pr(M_{i+1})Pr(S^+|M_{i+1})$. $Pr(M_{i+1})$ represents the model's prior probability. Simpler models have a higher prior probability. The data likelihood $Pr(S^+|M_{i+1})$ is determined using the Viterbi path for each string in S^+. The state merging procedure is stopped when no further state merge results in an increase in the posterior probability.

3.8 Learning CFG Version Spaces

VanLehn and Ball [VB87] proposed a version space based approach for learning CFG. Given a set of labeled examples, a trivial grammar that accepts exactly the set of positive examples can be constructed. Again this is similar to the PTA constructed by several regular grammar inference algorithms. A version space is defined as the set of all possible generalizations of the trivial grammar that are consistent with the examples. Since the set of grammars consistent with a given presentation is infinite it becomes necessary to restrict the grammars included in the version space The reducedness bias restricts the version space to contain only reduced grammars (i.e. grammars that are consistent with a given sentences but no proper subset of their rules is consistent with the given sentences). A further problem in the case of CFG is that the more general than relationship which determines if two grammars G_1 and G_2 are such that $L(G_1) \supseteq L(G_2)$ is undecidable. This problem is circumvented by denning a derivational version space (i.e. a version space based on unlabeled parse trees constructed from positive sentences). The idea is to induce partitions on the unlabeled nonterminal nodes of the parse trees. The derivational version space is a union of all the grammars obtained from the unlabeled parse trees. FastCovers is an operation based on the partitions of the sets of nonterminals determines the partial order among elements of the derivational version space. Then, each new positive example causes the version space to expand by considering the new parse trees corresponding to the example. Each negative example causes the grammars that accept the negative example to be eliminated from the version space. The FastCovers relation which is a variant of the grammar covers property is used to prune those grammars in the version space that covers the grammar accepting the negative example. This approach solves small induction problems completely and provides an opportunity for incorporation of additional biases to make the learning tractable in the case of larger problems.

3.9 NPDA with Genetic Search

Lankhorst [Lan95] presented a scheme for learning nondeterministic pushdown automata (NPDA) from labeled examples using genetic search. Push down automata PDA are recognizing devices for the class of context free grammars just as FSA are recognizing devices for regular grammars. A PDA comprises of a FSA and a stack. The extra storage provided by the stack enables the PDA to recognize languages such as palindromes which are beyond the capability of FSA. Lankhorst's method encodes a fixed number of transitions of the PDA on a single chromosome. Each chromosome is represented as a bit string of a fixed length. Standard mutation

and crossover operators are used in the genetic algorithm. The fitness of each chromosome which represents a NPDA is a function of three parameters training accuracy, correct prefix identification and residual stack size. The training accuracy measures the fraction of the examples in the training set that are correctly classified. PDA that are able to parse at least a part of the input string correctly are rewarded by the correct prefix identification measure. A string is said to be accepted by a PDA if after reading the entire string either the FSA is in an accepting state or the stack is empty. The residual stack size measure assigns higher fitness to PDA that leave as few residual symbols on the stack as possible after reading the entire string.

3.10 Deterministic CFG with Connectionist Methods

Das et al. [DGS92] proposed an approach for learning deterministic context free grammars using recurrent neural network push down automata (NNPDA). This model uses a recurrent neural network similar to the one described before in conjunction with an external stack to learn a proper subset of deterministic context free languages. Moisl [Moi92] adopted deterministic push down automata (DPDA) as adequate formal models for general natural language processing (NLP). He showed how a simple recurrent neural network can be trained to implement a finite state automaton which simulates the DPDA. Using a computer simulation of a parser for a small fragment of the English language Moisl demonstrated that the recurrent neural network implementation results in a NLP device that is broadly consistent with the requirements of a typical NLP system and has desirable emergent properties. The NNPDA consists of a recurrent neural network integrated with an external stack through a hybrid error function. It can be trained to simultaneously learn the state transition function of the underlying push down automaton and the actions that are required to control the stack operation. In addition to the input and state neurons the network maintains a group of read neurons to read the top of the external stack and a single non-recurrent action neuron whose output identifies the stack action to be taken (i.e. push pop or no-op). The complexity of this PDA model is reduced by making the following simplifying assumptions. The input and stack alphabets are the same, the push operation places the current input symbol on the top of the stack and transitions are disallowed. These restrictions limit the class of languages learnable under this model to a finite subset of the class of deterministic context free languages. Third order recurrent neural networks are used in this model where the weights modify a product of the current state, the current input and the current top of stack to produce the new state and modify the stack. During training, input sequences are presented one at a time and the activations are propagated until the end of the sequence is reached. This NNPDA model is capable of learning simple context free languages such as $a^n b^n$ and *parenthesis*. However, the learning task is computationally intensive. The algorithm does not converge for more non-trivial context free languages as $a^n b^n c b^m a^m$.

3.11 Stochastic CFG

The success of HMM in a variety of speech recognition tasks forces one to ask if the more powerful stochastic context free grammars (SCFG) could be employed for speech recognition. The advantages of SCFG lie in their ability to capture the embedded structure within the speech data and their superior predictive power in comparison with regular grammars as measured by prediction entropy. The Inside-Outside [LY90] algorithm can be used to estimate the free parameters of a SCFG. Given a set of positive training sentences and a SCFG whose parameters are randomly initialized the inside-outside algorithm first computes the most probable parse tree for each training sentence. The derivations are then used to reestimate the probabilities associated with each rule and the procedure is repeated until no significant changes occur to the probability values. Prior to invoking the inside-outside algorithm one must decide on some appropriate structure for the SCFG. Stolcke and Omohundro's Bayesian Model merging approach can be used for learning both the structure and the associated parameters of the SCFG from a set of training sentences. This approach is analogous to the one for HMM induction described earlier. New sentences are incorporated by adding a top level production from the start symbol S. Model merging involves merging of nonterminals in the SCFG to produce a more general grammar with fewer nonterminals and chunking of nonterminals where a sequence of nonterminals is abbreviated using a new nonterminal. A powerful beam search which explores several relatively similar grammars in parallel is used to search for the grammar with the highest a-posteriori model probability.

4 Why is a new approach needed

There is mainly a reason that supports the approaches most people are taking to tackle grammar inference until now. Automatas are incredibly good structures to be implemented in hardware and this can bridge the gap to real time usability. However the inference of grammar rules from the actual symbols used in the language is not as useful at it seems when facing natural language, since words have two different meanings in a sentence. One is the grammatic meaning (syntactic function) and the other is the actual word meaning (semantic). Our approach tries to use the former part of the symbols to extract the rules of the grammar, as opposed to the previous techniques that used the later one to infer them. Keep in mind that we are not trying to supplant the DFA approach but to change the way the rules for such DFA are obtained. Hence the process of deciding if a sentence is part of the grammar could be the same as in the methods shown before. When in previous techniques they tried to infer the rules, they could be having the same grammatic rules formed by different symbols ("el caballo esta sentado"/"la perra esta de pie"), introducing a lot of redundancy in the final result. With our approach this redundancy is deleted but the richness of different symbols will not be lost thank to the parallel implementation of a graph containing the symbols that had appeared. Furthermore, the utilization of syntactic rules will allow us to use much less complex methods to obtain the rules. To exemplify this, assume that in a text of one thousand words, we have one hundred unique words. If we calculate all the word-wise combinations for those 100 words, we reach the million permutations quite fast. However, by reducing that set to just 12 syntactic rules (if we consider just the first level of detail) then the combinations decrease a lot allowing for much simpler methods as classical data mining techniques like association rules. The utilization of association rules for grammar inference is the innovation brought by this study to the field of grammar inference. The procedure is explained in the following sections.

5 Association Rules for regular grammar inference

As mentioned in the last section. This new approach to grammar inference comes hand in hand with the inclusion of a syntactic translation of the language examples been used to infer the grammar.

This study falls into the category of symbolic AI since it does not center its functioning around an optimization problem. The algorithm presented in this article makes use of a knowledge representation of the language explored and a set of heuristic rules and procedures to apply that knowledge to a new sentence with the purpose of correcting it.

For the knowledge representation, the algorithm includes for types of relations intended to mimic the inner relations of the language observed. These for representations are formed by a graph for the function to word relations, a cluster for the word-to-word relations, a forest to represent the structures explored and a set of association rules to represent the generation rules of the language. These representations are populated by exploring language examples (sentences) and then harvested while trying to correct a new sentence.

Following the representations, the algorithm is formed by five main parts that run sequentially. A first section in charge of checking the existence of a structure, followed by the check for punctuations in the sentence, followed by the gap retriever (gap is where the error is) and the gap corrector and finishing with the sentence filler. These parts will be explained in depth further in the article.

In the next subsections the algorithm is explained by first reviewing the knowledge representation and then reviewing the sentence corrector itself.

5.1 Knowledge representation

5.1.1 Graph

As previously mentioned, the graph is the model in charge of representing the function-to-word relations found in the explored text. This model is based on a simple directed graph. Directed graphs [BJG08] are a set of vertices connected by edges, where these edges have a direction associated with them. The graph we will be using for the algorithm is a simple directed graph which implies that there will not be more than one edge connecting two nodes and that no node will have a loop in it.

The creation of the graph itself is quite straightforward. First, the terminal nodes are created which contain the syntactic rules (12 for the first level). Then whenever a new word is presented to the algorithm, a new node is created, this time connected to its terminal node (syntactic function) by means of a directed edge pointing towards the word. Whenever an already seen word is presented to the algorithm, the node representing such word is found and its edge is updated by adding a plus one to its weight. Apart from the edges linking syntactic functions to words, there will also be edges between word nodes. Their direction will follow the flow of the sentence.

This representation will allow us to build a syntactic translator that will be used on the first steps of the algorithm when we need to retrieve the syntactic translation of the natural language sentence. Moreover, this representation also models the distribution of appearance of a certain word. This distribution can be obtained from the weights of the edges of the words. The edges can be used to show the distribution of the words of the explored text, the distribution of words within a syntactic function, or in the case of words belonging to two different syntactic functions, the distribution for that word within syntactic functions.

The figure below shows an example of a digraph created following the method explained after observing a few sentences.

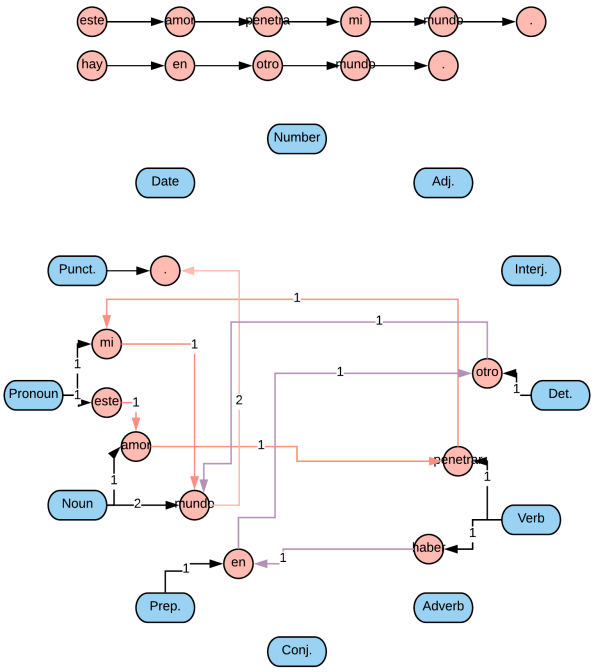

Figure 7: Graph of word-to-function relations

The graph shown in fig. 7 shows the word-to-function relations of a few sentences created by following the method explained before.

5.1.2 Cluster

To complete the representation of the explored text a bit more, we created a model in charge of representing the type of words that appear one after another. This model helps in the retrieval of words when we intend to fill a gap with a word.

The cluster is represented programmatically as a two dimensional matrix with syntactic functions as rows and columns. Then, in each of the cells of such matrix another two dimensional matrix is inserted. this new matrix consists of three columns and as much rows as apparitions in the explored text. The indices of the cells are the ones representing the syntactic functions of the two words in the cluster (column = first word, row = second word). The third cell of the inner matrix is the syntactic function following the previously mentioned words.

As with the graph, this representation is populated by exploring examples of the target language. Whenever a word is found, its syntactic functions is retrieved, along with the syntactic function of its previous word. These two functions point to a certain cell in the cluster. Then, inside that cell, a new row is created where the first and second word are placed followed by the syntactic function of the next word in the example.

This representation is in charge of suggesting a word whenever the algorithm is trying to fill a sentence with new words. In order to generate that suggestion, the algorithm tries to constraint as much as possible the cluster's output. This means, that we could ask for every word in the cluster maintaining that the first word is a verb , the second is a noun and the third is an adjective. Furthermore, we can look for noun suggestions that have "es" as the initial verb and followed the previous constraint. The figure below shows an example of the population of such a knowledge model.

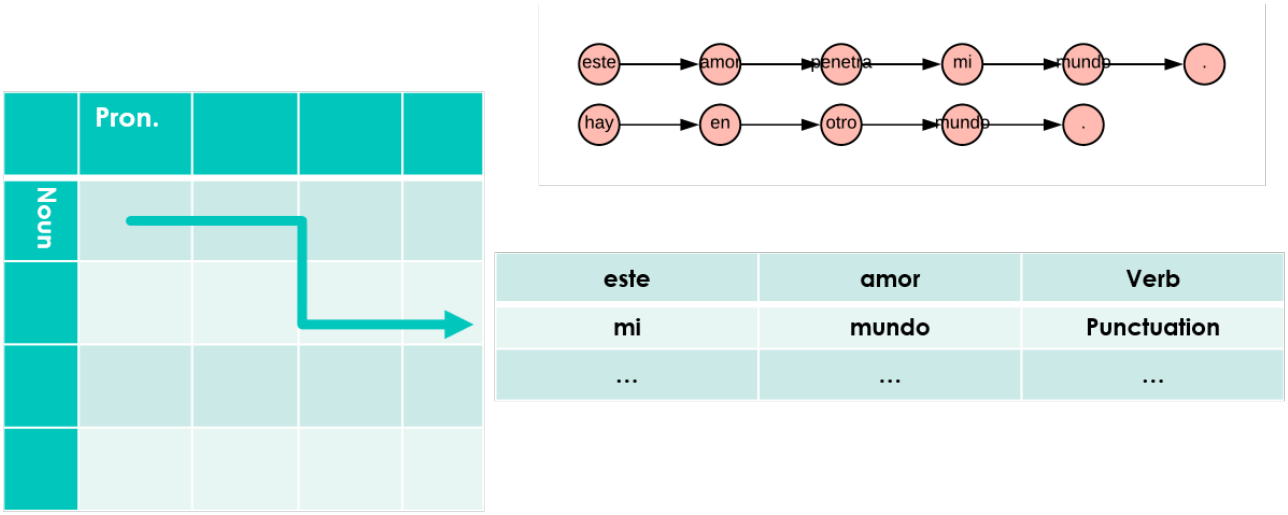

Figure 8: Matrix of word-wise relations

Fig. 8 shows a sample of the data the cluster stores after reading sentences.

5.1.3 Forest

As we introduced before, the next knowledge model included in the algorithm is based in a forest. This forest holds the syntactic structures that have been found in the explored text.

A forest is built up of trees. A tree, in graph theory, is usually an undirected graph in which any two vertices are connected by exactly one path. In this algorithm's case, every tree is a rooted tree (directed graph) in which all branches point away from the root itself. Hence, the root of each of the trees in this forest represents the first symbol of the syntactic structure of one of the sentences found in the explored text. All these trees are collected inside a forest that holds every syntactic structure found in the explored text.

This knowledge representation is the one the algorithm will check when trying to discover if a new presented sentence has an allowed syntactic structure. This action falls inside the existence function that we will mention further in the article. In order to be able to process every part of the existence function, the algorithm stores two different forests, one retrieved from sentences in a froward manner (with common prefixes) and the other retrieved from the sentences in a backward manner (with common suffixes).

As with the other representations, this one is filled in a quite straightforward manner. When exploring the text, every time a new sentence is found, its syntactic structure is retrieved. This structure is then compared with the ones in the forest. If the are no similar trees in the forest the structure is added as a tree to the forest. However, if there is another tree that shares a prefix with the sentence been explored, the new sentence's suffix will be added to that tree as a branch that extends from the common prefix. The figure shown below shows an example of the population of this representation.

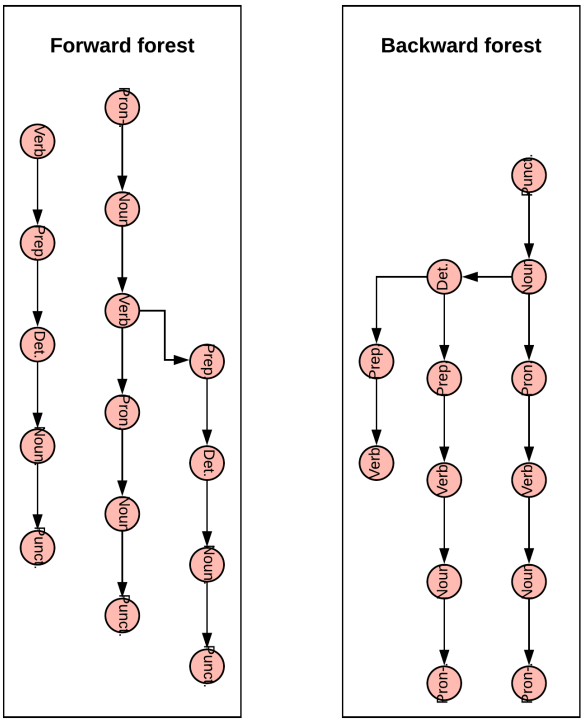

Figure 9: Forests to keep track of existing structures. (both forests represent the same three sentences)

Fig. 9 shows how three example structures are included in the forest showing both methods. One using extension of an already existing tree and the other showing the inclusion of a brand new tree in the forest.

5.2 Association Rules

This representation is the most important one for our research. Although all of the previous ones are needed, this way of representing the generation rules of the Spanish language is new in the field. As mentioned in the introduction of this section, the feasibility of a method such as Association rules is due to the inclusion of syntactic functions as the single unit of study as opposed to using just the word.
Before analyzing the method used to extract the syntactic rules, lets learn a bit about how syntactic functions work in Spanish language.

5.2.1 Syntactic functions

In our research, the target language is written Spanish. As syntactical theory explains, sentences are not just sequences of words set randomly. Sentences convey a meaning as a result of matching words to a certain order, ruled by their syntactical meaning [BGR09].

Lets see, how the usual process of syntactic analysis works:

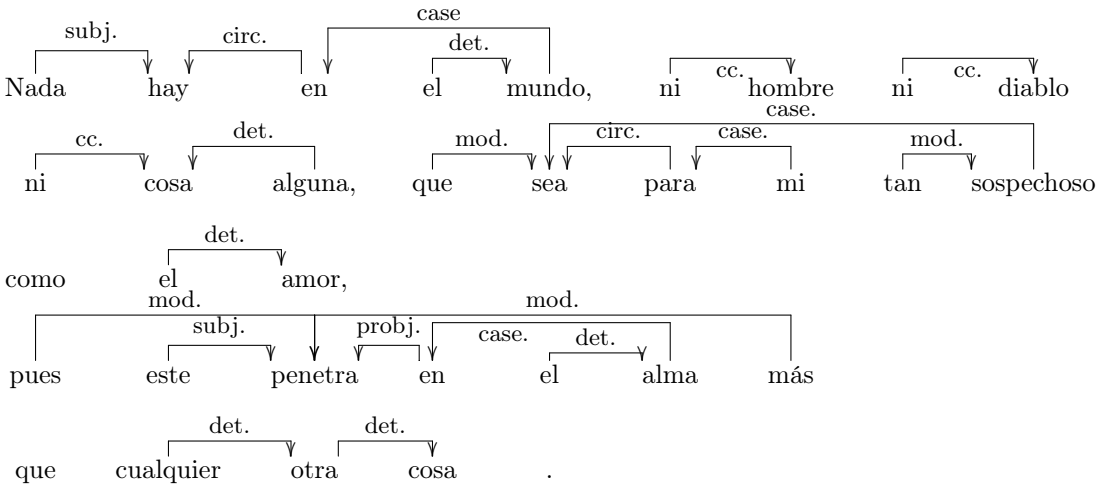

Figure 10: Syntactic Analysis of a sentence

mod → Modifier	det → Determinant	pobj → Regime supplement
circ → Circumstantial complement	subj → Subject	cc → Coordination
case → Clitic relationship		

Table 1: Meanings of the word relations in fig 10

As shown in fig. 10 the process of a syntactic analysis of a sentence, can be a bit tricky. However, the result explains the task of each word in the sentence. As stated before, inferring the underlying pattern that a sentence has is our primary goal, and to achieve so, let transform this typical syntactic analysis into a matrix. In order to make this translation, we will not be using the actual relations between the words, but the syntactical function of each of the words.

1	nada	pronoun
2	haber	verb
3	en	preposition
4	el	determinant
5	mundo	noun
6	,	
7	ni	conjunction
8	hombre	noun
9	ni	conjunction
10	diablo	noun
11	ni	conjunction
12	cosa	noun
13	alguna	adjetive indef.
14	,	
15	que	conjunction
16	ser	verb
17	para	preposition
18	mí	pronoun
19	tan	adverb
20	sospechoso	adjective
21	como	adverb
22	el	determinant
23	amor	noun
24	,	
25	pues	conjunction
26	este	pronoun
27	penetrar	verb
28	el	determinant
29	alma	noun
30	más	adverb
31	que	relative pronoun
32	cualquiera	determinant
33	otro	determinant
34	cosa	noun

Table 2: Syntactic representation of the words (The third column represents an item in the database

Keep in mind that for the sake of simplicity in the research we used just the first level of syntactic functions

in the Spanish language. This means that we have not included other eight levels of depth that would clearly improve the accuracy of our model to what humans use.

5.2.2 Grammar generation rules

As we mentioned before, the grammar generation rules that we will try to obtain for this algorithm are in essence association rules extracted from the data-set of sequences (sentences) of syntactic functions as shown above.

Once we obtain the observed data set, a sequence mining algorithm will be used to extract, first frequent item-sets and then, association rules from these frequent item-sets. First thing to note before attempting this process is that the patterns we are trying to mine from such a data-set are sequences and no simple items in a transaction. Hence, order relations are essential when mining sequences. There are many well know algorithms to mine frequent sequences in Data Mining. In general sequence mining problems can be classified as string mining which are typically based on string processing algorithms and item-set mining which are typically based on association rule learning.

String mining, is usually considered as a problem in which a relatively limited alphabet is used but quite long sequences are explored. Some well known algorithms for this purpose are GSP, FreeSpan, PrefixSpan, etc. However, this algorithms do not allow for different size gaps on their public implementations. Hence, we decided to use another approach to sequence mining introduced in [FVNN08]. This time the author introduced in the algorithm the ability to modify the size of the gap needed to consider an item-set a sequence.

By using his implementation which is a variation of prefix span with time constraints, we tried to explore different sizes of gaps with the intention of including relations in our knowledge representation that were not just direct relations in the text. However, we did not manage to find anything that was useful for our research.

The utilization of the obtained association rules is not as straightforward as the usage of the previous models. First we need to introduce the situation in which such association rules are used. Association rules are used in this algorithm when the generation of a syntactic rule is needed to fill a sentence. For this purpose, we pull from our representation of the grammar generation rules and retrieve a syntactic function that better fits (ordered by confidence) our situation. Confidence in association rules is the rule set by the user when retrieving them that represents the amount of times a rule has appeared against the amount of times its precedent has appeared in the database. This means, the confidence of a rule of type X -> Y is the amount of times such rule appears from the set of every sequence in which X appears with items following it. This been said, the association rules retrieved in for this model will always follow that X -> Y structure, however, X and Y can be sequences themselves and not only single items.

Whenever we are trying to guess a new syntactic structure for a slot in a sentence, we will be following these steps: First, the size of the sequences of both sides of the error are measured and the biggest one is chosen. Then, we look in the database for an association rule matching the biggest possible sequence of the selected side of the error (ordered by confidence). When an association rule is found, the syntactic rule to fit the error is generated and the new sentence is checked for existence. If the new sentence is correct, the process is ended. If it is not correct, the next matching association rule is chosen to generate the syntactic rule. To choose from a set of different size association rules, we will look first for the ones with bigger precedent and consequent. Then, between rules of the same size, is the confidence that dictates the order of application. If none of the association rules generates a function that leads to a correct sentence that sentence is taken to be impossible to correct.

5.3 The Algorithm

The algorithm is composed by six inner sections that process different parts of the correction process. Each of these six parts pulls information from one or many of the knowledge representations that we have already mentioned to fulfill its task. The overall idea of the algorithm is shown in the following figure.

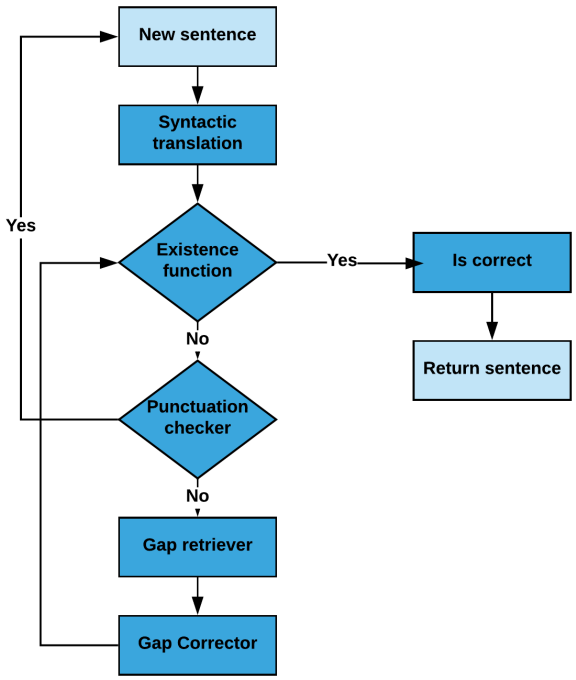

Figure 11: Algorithm's work-flow

As shown in the fig. 11, the first part of the algorithm is that of retrieving the syntactic translation of the new sentence. This gives pass to the existence function to check if the sentence needs to be corrected. Then, when the sentence is wrong, it is passed to the punctuation checker function to check for punctuations and call the algorithm recursively for the cases in which we find more than one punctuation. Once we have assured that the sentence left is a sentence with just one punctuation, we send the sentence to the gap retriever. The gap retriever finds the error in the sentence by means of a process explained later and returns the obtained error to the next function to correct it. Once it is corrected, the sentence then goes to the final step in which it is refilled with new words to match the new functions introduced.

5.3.1 First part: Syntactic translation

As we stated over the article, this algorithm uses mainly syntactic functions to work with Spanish sentences. Usually, written sentences are not in their syntactic form, thus a translation must be done to achieve what the algorithm needs.

For this purpose, the algorithm has two methods to proceed. The first one is actually part of the algorithm, the second one is a patch that is used sometimes until the explored text is big enough to have every word in the vocabulary in it.

The first way of retrieving the syntactic translation of a sentence is by pulling information from the directed graph we explained before. In order to achieve this, each time a word is found, we pull the parent of such word in the algorithm (syntactic function). If the word is in two different syntactic functions, we first look if in the previous and latter words we encounter similar syntactic functions in the graph, if we do, then that function is the one chosen. If we do not find differences between the surrounding functions in any of the two words, then we select a weighted random pick from both functions with the weights of the edges.
It is clear that it may happen that we do not have the word that must be translated due to the amount of text we have explored in the knowledge creation phase of the algorithm. To solve that problem we use the second method. In this second method we made use of the GCP (Google Cloud Platform) which has a natural language API to retrieve many things from an inputted word or sentence as its syntactic tags. In the case we do not have a certain word in the graph, that word's syntactic function was retrieved by means of GCP and then inputted to the graph as a new node with its connection to the syntactic function.

This two methods can work together to avoid fails in the work-flow of the algorithm. Take note that when it is said that the new word (translated by GCP) is included in the graph, it is only included as a single node with its edge coming from the syntactic function. It will not be linked to any other word in the graph since without the validation of that sentence been correct we cannot include the sentence in the graph.

5.3.2 Second part: Existence function

This second method of the algorithm receives a sentence in both forms as an input. What this method does internally is quite simple. It only checks if the sentence exists in the knowledge retrieved from the explored text. For that, it looks in the forest for a tree that is equal to that of the new sentence. This function can also tell if the structure appears as a subtree of one of the trees in the forest.

To tell if a new structure is a subtree of an existing one, we only considered full subtrees from the start or the end of the existing structure. This means that the algorithm only checks subtrees that are equal to a part of the starting or the ending of an existing tree, and it does not look into the trees itself. The reason to do this is that we look for prefix or suffixes that already exist in the forest and not random chunks in the trees. This comes from the idea that if we have an error in a sentence, the left-most part of that sentence before the error, or the right-most part of that sentence after the error can be seen as prefixes or suffixes in other sentences. To achieve this we have the earlier mentioned two forests. The forward retrieved forest to check for prefixes and the backward retrieved forest to check fro suffixes.

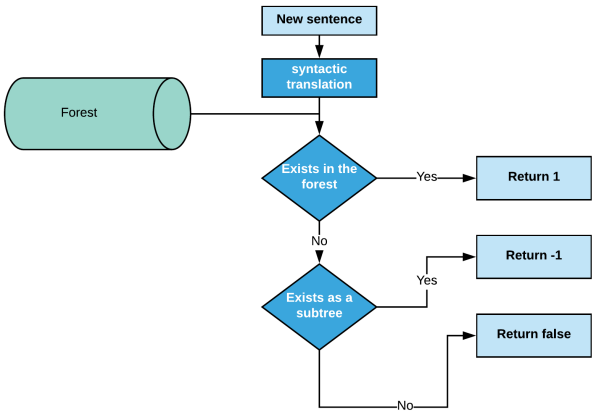

Figure 12: Existence function's work-flow

As shown in the fig. 12, this function returns three outputs. It will return a one if the structure exists as a whole tree in the forest, a minus one if it exists as a subtree in the forward or the backward forest. And a zero if it does not exists as none of the two before.

5.3.3 Third part: Punctuation checker

To follow with the correction, we first check for the punctuations the sentence has. Any sentence must have a mandatory punctuation that goes in the end of the sentence, but many times, we use punctuation to connect different chunks of a sentence and that is way we include this method in the algorithm.

The combination of punctuations that we can include in a sentence and still be correct makes impossible for the algorithm to have all those combinations as individual structures from the explored text. To avoid that, we cut the sentences after those punctuations to check each chuck as a single sentence and group them together in the end. To achieve that, this method includes recursivity, it calls the whole algorithm again, this time inputting just the chunk it has retrieved from the original sentence as the new sentence.

5.3.4 Fourth part: Gap retriever

This fourth method of the algorithm is the one in charge of finding the error within the sentence. To approach this problem we used the existence function again.

We first create a matrix of two row vectors filled with zeros and the same length as the sentence to be checked. The upper vector will contain the forward check and the vector in the bottom the backward check. The forward and backward checks are the process of one by one inputting the subsentence (from the start/back to a certain position) to the existence function and filling that position in the vector with the output of the existence function.

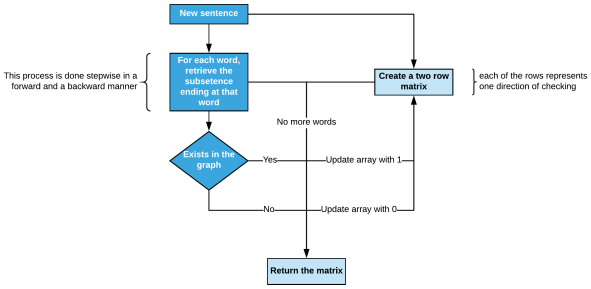

Figure 13: Gap retriever's work-flow

The fig. 13 shows the resulting matrix of a process like the one explained above. What the resulting matrix tells us is which parts of the sentence we are trying to check actually exit as a prefix or suffix in the algorithm. The minus ones represent that the subsentence from the closest side to that position is included as a prefix/suffix in the algorithm. This matrix allows us to find the place in which the error resides. The part of the matrix in which we find the zeros is where the error is. For now on, the error will be called the gap since is the gap of non minus ones that we need to fill in order to correct the sentence.

5.3.5 Fifth part: Gap corrector

This method follows the gap retriever, and it tries to correct the gap by different means. Heuristically we came to realize that errors were different based on the properties of the gap. A gap of a single space means that a wrong word was put there, instead a bigger size error may imply that a combination of words were not correct (order-wise) or that the error is actually a three word size error.

Based on the size of the error we treat it with three different methods. For errors of size one, we try to fill the gap by simply pulling from association rules and getting the best matching association rule to fill it. If the new sentence does not exist, we will use the next best matching association error and so on. Recall the method we used to find the best matching association error form the section in which we explained them (association rules). If the error is bigger that one but smaller that four, we first try to permute the words in the error to check if there exists a permutation that does exist. If not, we follow by filling the error again with association rules. Finally if the error is bigger than four, we directly fill it with association rules without attempting permutations. The following diagram shows the scheme of the algorithm and its work-flow.

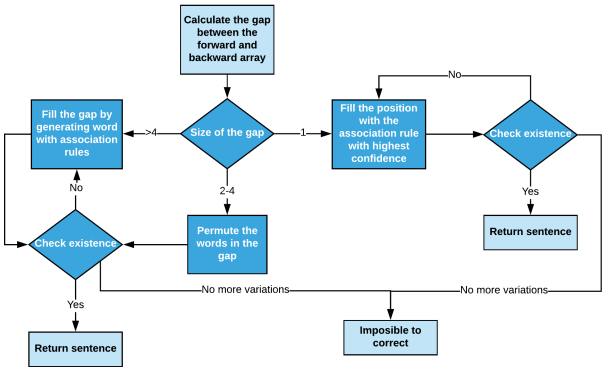

Figure 14: Gap corrector's work-flow

5.3.6 Sixth part: Is correct

The last part of the algorithm attempts to fill the corrected sentence with words where the new syntactic functions have been put. For that, the algorithm retrieves the set of words from the cluster explained in the previous section. From that cluster, the algorithm is able to generate a list of words that may go in a certain position. These words are linked to the weights its edges have in the graph. In order to select a word for a position, a weighted random guess is performed using as weights the edges of the words.

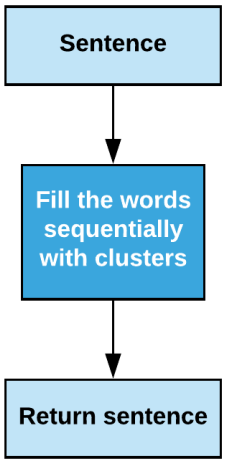

Figure 15: Is correct's work-flow

6 Results

There is an inherent complication when assessing the performance of this type of algorithms (language related) since language can be seen as correct in many aspects influenced by subjectivity. To tackle that problem, it has been decided to create two test procedures, one in which subjectivity has been completely removed and another in which just comprehensiveness is contemplated.

In this section we will start by discussing the data used for the experiments. Although the ideal situation would be having uncountable published works as the input data from which the algorithm would acquired its knowledge, that procedure on its own can be very difficult. In order to test our algorithm we decided to use [TMR08] given to us by the University of Catalonia with sentences retrieved from published newspaper articles in Spanish. Then we will follow with the in depth explanation of the examples and the results brought by them to finish with the conclusions extracted from those results. The actual sentences used in the examples are shown in the appendix with the intention of leaving the article easy to read and not having many pages with just Spanish sentences and their corrections.

6.1 Data observation

Before starting to code the algorithm it is our diligent responsibility to check how well the database represents the language at stake. Spanish language is a complex language in which rules reign over exceptions. At least, maybe more than for other languages. That was one of our first premises to aboard this study. This sections tries to summarize some simple statistics to describe the characteristics of the database [TMR08] we have.

Keep in mind that this database will never represent Spanish language as a whole. As mentioned by the authors of the database, they collected sentences from articles in journals and newspapers. This constraints our database towards a more serious vocabulary and construction. However, our purpose with this exploration is just to check that there is nothing that catches our eye as being wrong inside the Spanish language.

For the statistics, we covered the distributions (histograms) of some of the characteristics that we deemed interesting: sentence length and grammatical type apparition by sentence.

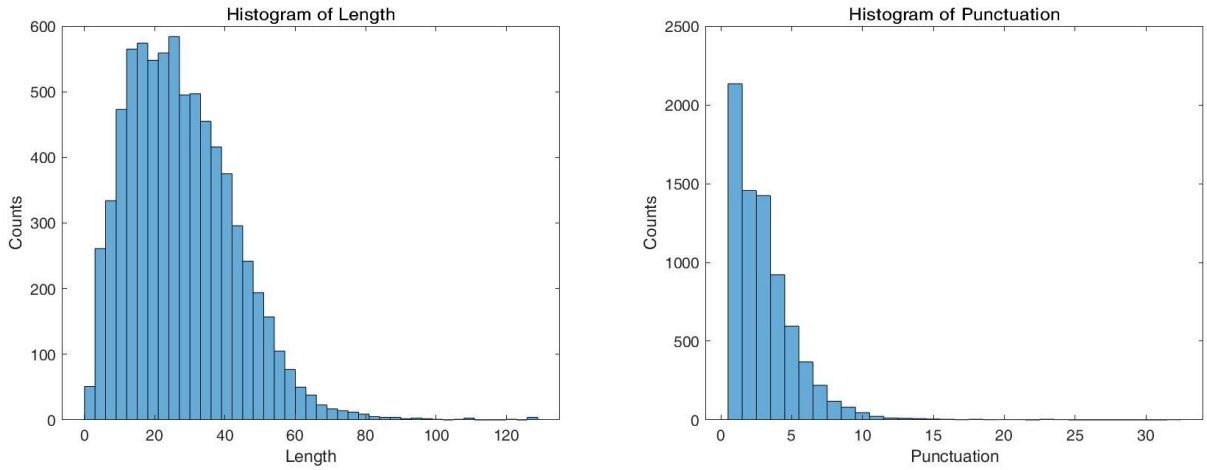

Figure 16: Length and Punctuation apparition histograms in the database.

The image above 16 shows that the average length of sentences in Spanish is around the thirties. And that its distribution resembles the normal distribution. For the Punctuation appearance, the distribution is closer to Pareto, reflecting that the most typical thing is for a sentence to have one two or three punctuation marks.

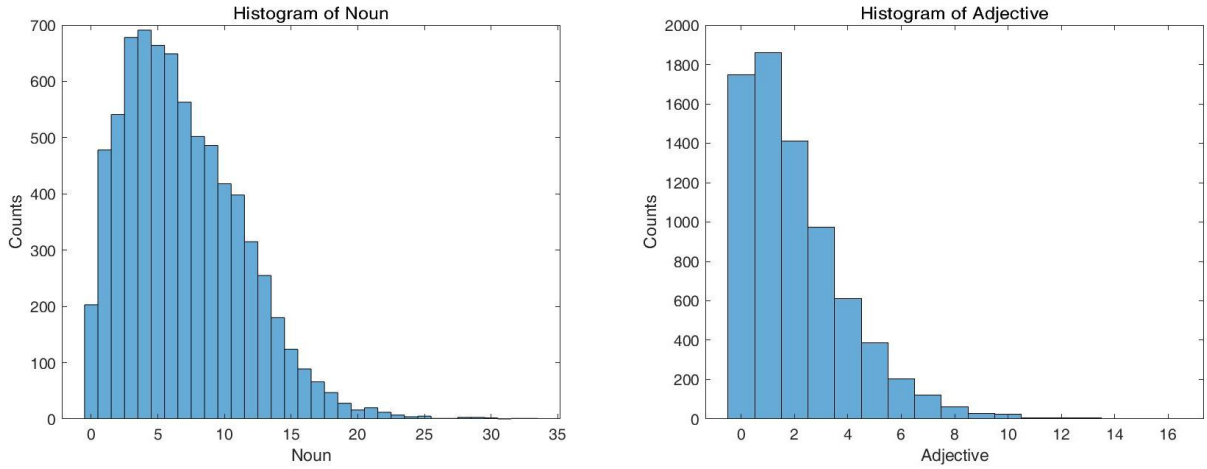

Figure 17: Noun and Adjective apparition histograms in the database.

The image above 17 shows the average apparition of Nouns doubles the average of Adjectives. And that its distribution resembles a normal distribution again. For the Adjective apparition again, the distribution is closer to Pareto, reflecting that the most typical thing is for a sentence to have none or a couple adjectives per sentence.

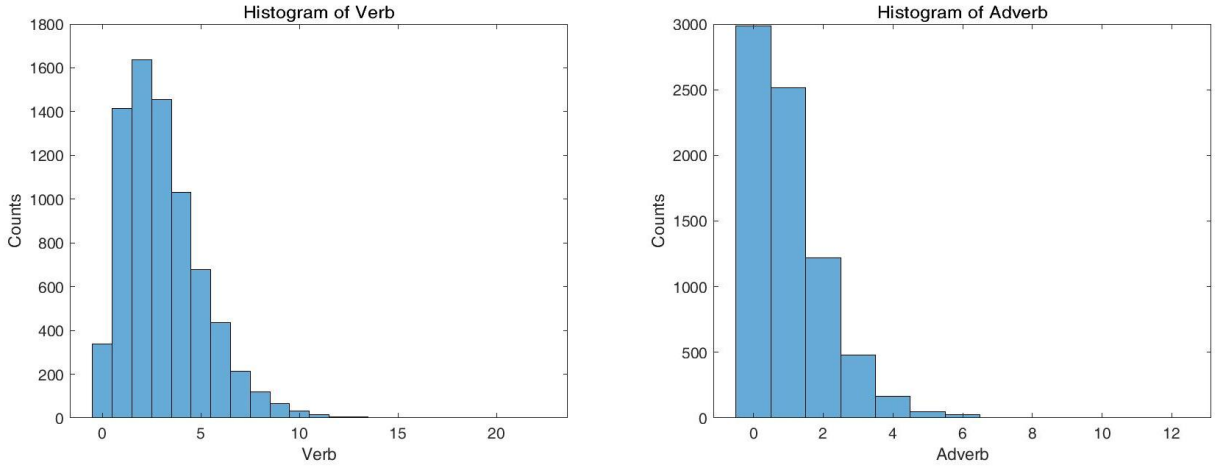

Figure 18: Verb and Adverb apparition histograms in the database.

The image above 18 shows the average apparition of Verbs doubles the average of Adverbs, while they are around half of the previous grammatical types. Verbs distribution resembles a normal distribution again with the detail of an almost non appearing zero. For the Adverb apparition again, the distribution is closer to Pareto, reflecting that the most typical thing is for a sentence to have none or a couple adverbs per sentence.

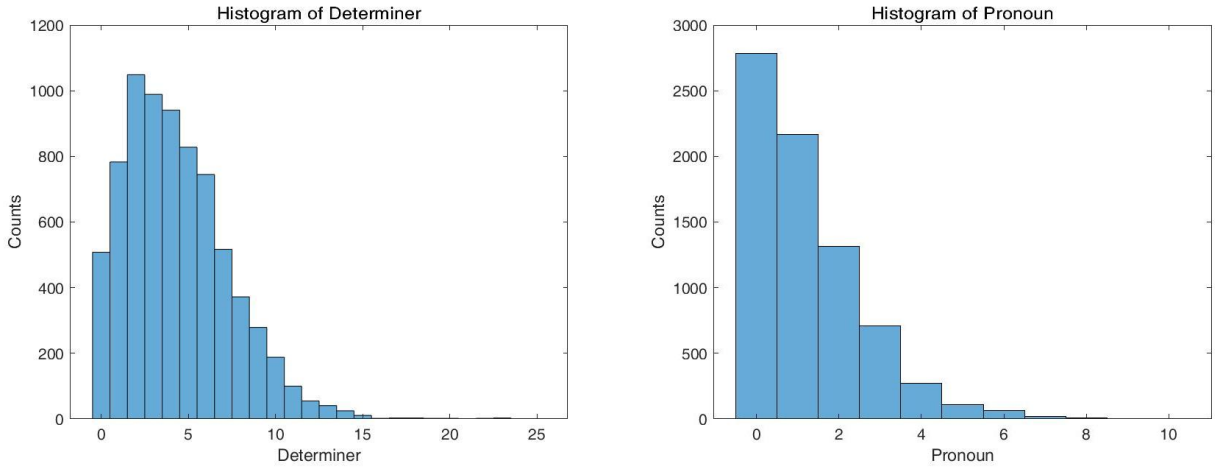

Figure 19: Determiner and Pronoun apparition histograms in the database.

The image above 19 shows that the apparition behavior of Determiners follows that of Noun. However, this seems logical since determiners are what introduce nouns in a sentence. Determiner's distribution resembles a normal distribution again. For the Pronouns, the distribution is closer to Pareto again, reflecting that the most typical thing is for a sentence to have none or a couple adverbs per sentence.

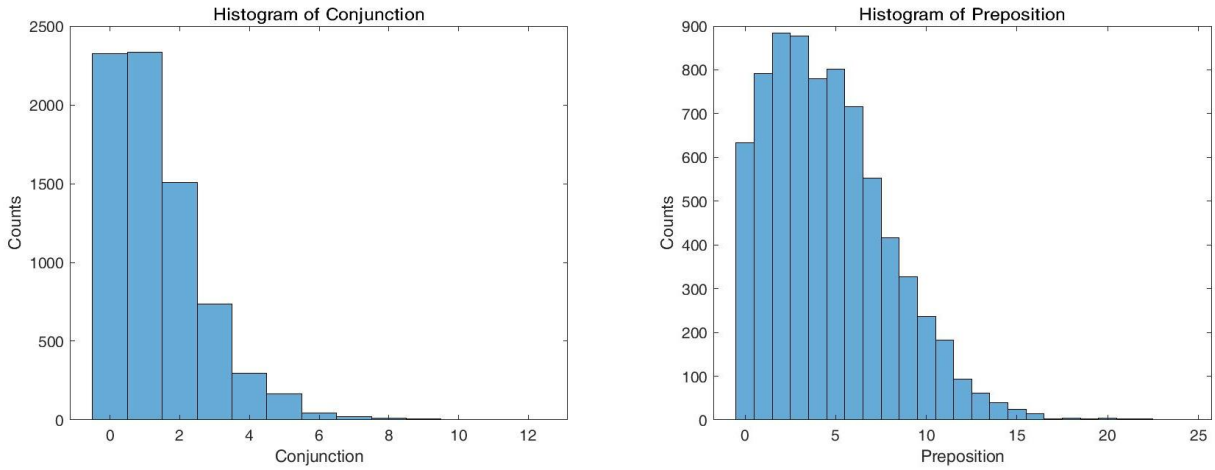

Figure 20: Conjunction and Preposition apparition histograms in the database.

The image above 20 shows that the apparition behavior of Prepositions follows that of Noun. However, this seems logical since prepositions are used to create dependencies between words, usually with Nouns. Conjunction's distribution resembles a Pareto distribution again. Being zero and one apparition the most common. Also logical since they are used to concatenate different sections in the same sentence.

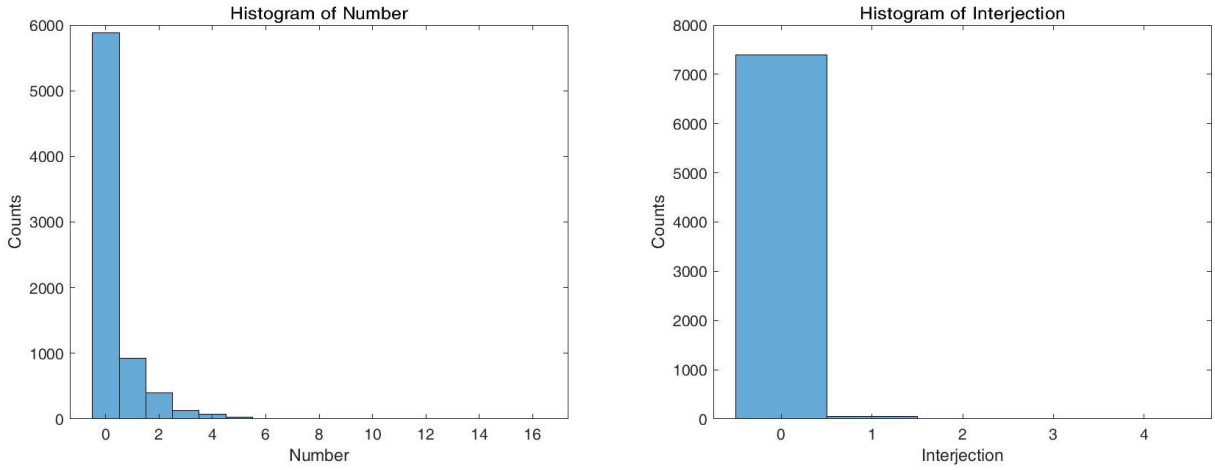

Figure 21: Number and Interjection apparition histograms in the database.

Finally, 21 shows that nor numbers nor Interjections are very used in this database. This follows from what we see, since it is advised to avoid numbers when writing in Spanish. Interjection on the other hand express emotions, and usually are not part of serious writings as the ones we have at hand.

6.2 Experiments

This subsection presents the experiments done in the data previously explored. As mentioned before, these experiments try to cover every aspect of the algorithm. we will first start by introducing a subjective test in which the intention is just to check the comprehensiveness of the corrected sentences to be able to make an assessment of the strengths and weaknesses of the algorithm. Following that, we will present an objective test in which we ask the algorithm to correct synthetic errors. Then, as a third experiment we tried to use the algorithm as a text classifier and check if it could be possible to contemplate the idea of using it as a classifier. Finally we show the usability of the algorithm as a software that may help non Spanish users in writing Spanish sentences.

6.2.1 Experiment 1: Subjective

As stated before, from this first experiment is hard to tell if the algorithm is doing right or at least how much is doing right, since there is not objective metrics to look at. However, since we are developing an algorithm that works with Natural Language and such languages are very complex and their correctness is very subjective, we decided to try this first experiment as a test to get a feeling of the strengths and weaknesses of the algorithm.

For this experiment we asked nine non Spanish speakers to write sentences in Spanish. Among the users, we have people from very different nationalities and many different levels of knowledge of the Spanish language. This means that the errors in the sentences will be very different from one another and that as are errors made from people in very different levels, these errors will be very complex in their structure.

We managed to achieve around thirty sentences. The mean length of the sentences is around fifteen words since the users with lowest level of Spanish were not able to construct long sentences.

Original sentence :
" Me -> llamo -> Jordan -> , -> soy -> frances -> , -> y -> vivo -> en -> bilbao -> desde -> hace -> 2 -> meses -> . "
Original structure :
" Pronoun -> Verb -> Noun -> Punctuation -> Verb -> Adjective -> Punctuation -> Conjunction -> Verb -> Preposition -> Noun -> Preposition -> Verb -> Number -> Noun -> Punctuation " .

Algorithms answer:
Corrected sentence :
" Me -> llamo -> Jordan -> , -> soy -> frances -> , -> y -> vivo -> en -> bilbao -> . "
Corrected structure :
" Pronoun -> Verb -> Noun -> Punctuation -> Verb -> Adjective -> Punctuation -> Conjunction -> Verb -> Preposition -> Noun -> Punctuation " .

Humans answer:
Corrected sentence :
" Me -> llamo -> Jordan -> , -> soy -> frances -> y -> vivo -> en -> bilbao -> desde -> hace -> 2 -> meses -> . "
Corrected structure :
" Pronoun -> Verb -> Noun -> Punctuation -> Verb -> Adjective -> Punctuation -> Conjunction -> Verb -> Preposition -> Noun -> Preposition -> Verb -> Number -> Noun -> Punctuation " .

Our feeling from this experiment is that the algorithm performs quite well for errors of size less than four. When the size of the error is bigger than four, it is quite difficult for the algorithm to maintain the meaning of the sentence since the filling of the corrected sentence is done word-wise.

6.2.2 Experiment 2: Objective

This experiment was derived from the first experiment with the intention to assess the objective performance of the algorithm in the parts we thought it was strong. This means that we created four tests in which we tried to asses the performance for each of the error sizes we deemed interesting. These error sizes are one, two, three and four. Within these size errors we chose to take the first two sizes as actual errors (the word in that position is wrong) and the two left as ordering errors. This decision was steered by the fact that if errors of size three-four are wrong, then the filling will quite easily end up with a non matching sentence.

For the tests, we obtained 200 sentences from a newspaper (El País). In the case of the errors of size one, we randomly choose a random position in the sentence and changed the word in that position to another picked randomly. For the case of errors of size two we did the same thing but we choose two words together in a random position of the sentence and we filled with two random words. For the case of three-four size errors we choose a random position and we permuted the words in that position choosing a permutation that is not correct. The following table shows the results of this experiment.

Number of sentences	Error size	Accuracy M1	Accuracy M2	Error type
200	1	78%	98%	random
200	2	76%	95%	random
200	3	65%	93%	swap
200	4	63%	91%	swap

Figure 22: Results of correction of synthetic errors (M1: corrected sentence equal to original, M2: corrected sentence is accepted in Spanish language

As shown in the table, the results indicate that the algorithm performs well in errors of size one and two. However its performance decreases greatly when the error size is increased. Something to note from this experiment is that, our permutation attempt seems to have problems when selecting a suitable correcting. That could be due to not having any way of ordering the possible permutations, allowing the algorithm to find other permutations that are also allowed first.

6.2.3 Experiment 3: Text classification

For the third experiment, we decided we wanted to test if the algorithm had enough information in it to differentiate between text types. By text type we mean texts from different sources in which the writing style clearly differs.

This time, we collected ten articles from three different sources. First source is that of El Pais, Spanish newspaper which in our perspective should resemble the text used to feed the algorithm quite a bit. The second source is La Gaceta, another Spanish newspaper, this time a much more extremist newspaper with a different tint in its articles and a much smaller reader population. This second source should still be somewhere inside the knowledge of the algorithm since it is a newspaper but we expect it to be less representative of the knowledge the algorithm has. Finally the third source comes from Quora. Quora is a web-based Newspaper-type platform in which people is allowed to post their articles without a publisher reviewing them.

For the experiment, we inputted each of the sentences of each of the articles to the algorithm and we noted if the algorithm thought if the sentence was corrected or not. The table below, shows the results of this experiment.

Source	N° articles	Avg. sentence/article	Percentage of sentences approved
El Pais	10	150	92%
La Gaceta	10	122	70%
Quora	10	60	30%

Figure 23: Classification results for third experiment

As shown in the table above, the results show that there is a great difference between the texts that have been published from those that have not. In the fig 23 we see that there is a gap between the two newspapers and Quora. That gap amounts to a 50% difference in the worst situation. This results could imply that algorithms like this could be used to classify texts. Another approach to this problem would be to train this model with articles of a single newspaper. This could lead to have a classifier that differentiates articles that are from that newspaper.

6.2.4 Experiment 4: Real time sentence writing help

This last experiment shows a possible utilization of an algorithm like this. Although there is not much to show since it is a practical attempt, the output of that example clearly showed that an algorithm like this could easily help people in their task of learning and writing in a foreign language, Spanish specifically.

First, the user was asked to write a sentence in Spanish, when he did not know how to follow or if what he wrote was right, that was inputted to the algorithm. Then, the algorithm would correct it. If an error was found, the user is able choose from the set of corrections the algorithm has found to be correct in order to keep the meaning of the sentence. After that, with a small modification of the algorithm, it would also propose a set of functions to continue the sentence (pulling from the association rules as we explained earlier) for the user to choose from. Once a function is selected, the algorithm looks for the words in the cluster matching the actual situation and it will propose a set of words for the user to choose from.

The main problem this usage has is that, if the user does not know Spanish, the set of possibilities proposed to him must be translated or accompanied by a brief explanation in English for example.

Although there is nothing to show in this experiment, since it is really hard to show in paper something that has no clear output. This last experiment shows that the usage of the knowledge representations created in this paper can actually be used for different separated tasks to help humans in writing Spanish. How well it does at it is something that should be studied in depth.

7 Conclusion

In this study we have revisited grammar inference with a difference to every other study published. Here we claimed the power of using syntactic rules as the key component for grammar inference in Spanish. We used Association rules as the Grammar-Generation-Rules from the language extracted from a sequence database (newspaper sentences in Spanish). We created three other knowledge representations to aid these association rules for checking, translating and suggesting new words for a sentence. Two of them are based in graph theory. The forest, keeps track of existing structures and the graph, keeps track of the relations between words and functions in the explored text and a matrix keeps track of pair-wise word relations in the explored text.

All in all, we managed to obtain an algorithm that achieved a 78.250% accuracy when correcting size one random errors in a natural language experiment.

Clearly there are things that must be improved. Increase the amount of text explored by automating the process of retrieval of valid text. Deepen the levels of syntactic rules that we use for the analysis of the text, allowing for a much more complex understanding of it.

References

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[BGR09] Ignacio Bosque and Javier Gutiérrez-Rexach. *Fundamentos de sintaxis formal*. Ediciones Akal, 2009.

[BJG08] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[Cho02] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002.

[CO94] Rafael C Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

[DGS92] Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, page 14, 1992.

[DMV94] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? *Grammatical Inference and Applications*, pages 25–37, 1994.

[FVNN08] Philippe Fournier-Viger, Roger Nkambou, and Engelbert Mephu Nguifo. A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems. In *Mexican International Conference on Artificial Intelligence*, pages 765–778. Springer, 2008.

[GCM+91] C Lee Giles, D Chen, CB Miller, HH Chen, GZ Sun, and YC Lee. Second-order recurrent neural networks for grammatical inference. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 273–281. IEEE, 1991.

[Gol67] E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[Gol78] E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[Hol92] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[Lan95] Marc Martijn Lankhorst. *A genetic algorithm for the induction of nondeterministic pushdown automata*. University of Groningen, Department of Mathematics and Computing Science, 1995.

[LY90] Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

[Moi92] Hermann Moisl. Connectionist finite state natural language processing. *Connection Science*, 4(2):67–91, 1992.

[OG92] José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

[PC78] Tsyh-Wen Pao and John W Carr. A solution of the syntactical induction-inference problem for regular languages. *Computer languages*, 3(1):53–64, 1978.

[PH93] Rajesh G Parekh and Vasant Honavar. Efficient learning of regular languages using teacher-supplied positive samples and learner-generated queries. 1993.

[PH96] Rajesh Parekh and Vasant Honavar. An incremental interactive algorithm for regular grammar inference. *Grammatical Interference: Learning Syntax from Sentences*, pages 238–249, 1996.

[Shi85] Stuart M Shieber. Evidence against the context-freeness of natural language. *The Formal complexity of natural language*, 33:320–332, 1985.

[SO94] Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. *Grammatical inference and applications*, pages 106–118, 1994.

[TMR08] Mariona Taulé, Maria Antònia Martí, and Marta Recasens. Ancora: Multilevel annotated corpora for catalan and spanish. In *Lrec*, 2008.

[VB87] Kurt Vanlehn and William Ball. A version space approach to learning context-free grammars. *Machine learning*, 2(1):39–74, 1987.