Sequences Classification by Least General Generalisations

Frédéric Tantini
Parole, CNRS/LORIA Nancy
Alain Terlutte, Fabien Torre
Mostrare (INRIA Lille Nord Europe et CNRS LIFL)
Université Lille Nord de France

March 17, 2024

Abstract

In this paper, we present a general framework for supervised classification. This framework provides methods like boosting and only needs the definition of a generalisation operator called LGG. For sequence classification tasks, LGG is a learner that only uses positive examples. We show that *grammatical inference* has already defined such learners for automata classes like *reversible automata* or *k-TSS automata*. Then we propose a generalisation algorithm for the class of balls of words. Finally, we show through experiments that our method efficiently resolves sequence classification tasks.

Keywords: sequence classification, least general automata, balls of words.

1 Introduction

We investigate in this paper the problem of sequence classification with two main ideas.

First, we want to benefit from supervised classification advances like ensemble methods (*bagging*, *boosting*, etc.). For this, we use a general framework for supervised classification based on the notion of *least general generalisation* (LGG). This framework, called VOLATA, provides various ensemble methods whenever we are able to define this LGG operator.

Second, we claim that results from the *grammatical inference* domain, like language identification with positive instances only, can be used in order to classify sequences. We consider learnability results: proofs of learnability from positive examples could offer learning algorithms close to an LGG operator. Applying this approach, we study 0-reversible automata [1], k-TSS automata [2] and balls of words [3].

The paper is organised as follow. In Section 2, we describe a LGG-based machine learning framework and give its main generic algorithms. This generic method is instantiated for some automata families in Section 3, and for balls of words in Section 4. In Section 5, we run these algorithms on well-known sequence classification problems and also on a real handwritten digit recognition problem. Finally, Section 6, we assess our results and propose some future research works.

2 Supervised learning with least general generalisations

In this section, we describe our framework which is based on the following points:

- choose example language ($\mathcal{E}$) and hypothesis language ($\mathcal{H}$) such that $\mathcal{E} \subset \mathcal{H}$;
- define a subsumption relation $\succeq$ between hypotheses of $\mathcal{H}$ that allows to check if a hypothesis subsumes an example, and also to decide if a hypothesis is more general than another one;
- given $\mathcal{H}$ and $\succeq$, prove the existence of a unique least general hypothesis for every set of examples and define an algorithm LGG to compute this hypothesis.

The last point uses the notion of *least general generalisation* defined as follow.

Definition. least general generalisation
Given a set of examples $E \subseteq \mathcal{E}$, a hypothesis $h \in \mathcal{H}$ is a *least general generalisation* of E iff:

- $\forall e \in E : h \succeq e$;
- there exists no hypothesis h' such that $\forall e \in E : h' \succeq e$ and $h \succ h'$.

Assuming a unique and computable least general generalisation, the VOLATA system is designed through three levels.

1. The first one defines the LGG operator and depends on $\mathcal{H}$ and $\succeq$. The next levels are generics, they depend neither on language representation nor on generality ordering.

2. The second one uses classes to control the generalisation. A possible algorithm here is CG (for *correct generalisation*) that aims at producing a correct hypothesis (Algorithm 1): the first example is used as a seed, then we try to generalise other positive examples using the LGG operator; each generalisation must be validated against negative examples; an example that produces an incorrect generalisation is rejected and we continue to add positive examples from the previous correct hypothesis.

Algorithm 1 CG (*correct generalisation*)

```
Require: $E = [p_1, \dots, p_n] \subseteq \mathcal{E}$ an ordered set of n examples with the same class,
    $N \subseteq \mathcal{E}$ a set of counter-examples.
Ensure: $h \in \mathcal{H}$ generalisation of some examples in $E$ and correct wrt $N$.
1: $g = p_1$
2: for $i = 2$ to $n$ do
3:     $g' = \text{LGG}(g, p_i)$ /* generalisation with $p_i$ */
4:     if $(\forall e \in N : g' \not\succeq e)$ then /* if $g'$ is correct */
5:         $g = g'$ /* $g'$ is now the current generalisation */
6:     end if
7: end for
8: return $h(x) = \text{class}(E)$ if $g \succeq x$, else 0 (abstention)
```

Let us note that the result of CG depends on the presentation order of positive examples. This characteristic will be an advantage in ensemble methods since it enhances diversity of produced hypotheses.

3. The last level provides generic full learners: DLG for a fast learning, GLOBO to obtain a comprehensive theory and ensemble methods like GLOBOOST, BAGGING and ADABOOST-MG that ensure better predictions. In this paper, we focus on GLOBOOST that is the simplest ensemble method: it randomly produces T correct hypotheses; at each step, a class is randomly chosen, sets of positive and negative examples are built, randomly shuffled and then the

algorithm CG is called on these sets; finally, a new example is classified by the T hypotheses in a vote. GLOBOOST is described in Algorithm 2.

Algorithm 2 GLOBOOST

```
Require: labelled examples $(x_i, y_i)$ and $T$ the number of steps.
Ensure: $H$ the final classifier.
1: for $t = 1$ to $T$ do
2:     $target$ = class randomly chosen
3:     $P = [x_i | y_i = target]$
4:     $N = [x_i | y_i \neq target]$
5:     randomly shuffle $P$
6:     $h_t$ = CG($P$,$N$) /* Call to the correct generalisation algorithm */
7: end for
8: return $H(x) = \text{sign}\left(\sum_{t=1}^{T} h_t(x)\right)$
```

The two presented algorithms, GLOBOOST and CG, are generics, they have to be instantiated by a generalisation operator LGG, itself depending on $\mathcal{H}$ and $\succeq$. In the rest of this paper, examples are words and hypotheses are either automata or balls of words. For these families, the subsumption test between an hypothesis and an example is naturally the word membership of the denoted language. What remains is the definition of the LGG operator. In *grammatical inference*, this means finding for each family an algorithm that learns with positive examples only, and provides the smallest language that contains the training set. In the two following sections, we investigate this question for k-TSS automata, 0-reversible automata and balls of words.

3 Generalisation of words to automata

We consider in this section that hypotheses are automata. The subsumption test is obviously whether or not the automaton accepts the example. We have now to define the LGG operator for each class of language we want to use, that is, an algorithm from positive examples only ensuring the smallest language that includes the given examples.

3.1 The k-TSS languages

The class of *k-testable in the strict sense* (k-TSS) languages is a well known subclass of regular languages [2]. It is characterised by the set of sequences of length k that do not appear in the word of the language. These languages are very simple and their expressivity is relatively poor.

However, there exists an algorithm learning from positive examples only. It is known to compute the smallest language including the sample, then the least general generalisation. We propose an incremental version, LGG-TSSI (Algorithm 3), for which the key point is that each state is represented by the last $(k-1)$ letters read to reach this state.

Algorithm 3 LGG-TSSI

Require: $h = (Q, \Sigma, q_0, F)$ a k-TSS automaton, e an example, a given integer k.
Ensure: h' a k-TSS automaton, least general generalisation of h, subsuming e.

```
1: $q = q_0$
2: for $i = 1$ to $|e|$ do
3:     $v = q.e_i$ /* concatenation of the word of $q$ with the $i$th letter of $e$ */
4:     if ($|v| = k$) then
5:         $v = v_{2,\ldots,|v|}$
6:     end if
7:     $nq = v$
8:     add $nq$ to $Q$ /* $nq$ may be already present in $Q$ */
9:     add $(q, e_i, nq)$ to $\delta$
10:    $q = nq$
11: end for
12: add $q$ to $F$
13: return $h' = (Q, \Sigma, \delta, q_0, F)$
```

3.2 The 0-reversible languages

The class of k-reversible languages is often said to be more interesting than the previous one, partly because it is more expressive: $(k-1)$-TSS languages are all k-reversible ones. Note that, given a fixed k, some *finite* languages are not k-reversible.

In this paper, we will focus on $k = 0$, that is 0-reversible languages. A least general generalisation computation in the class of 0-reversible languages is given in [1]. We propose an incremental version: LGG-ZR (Algorithm 4). In this algorithm, the new word is added to the current automaton in the PTA way, that induces the creation of a new branch recognising only this word. Then, merges are made in order to get a unique final state, deterministic transitions and a deterministic *mirror*-automaton.

Algorithm 4 LGG-ZR

Require: $h = (Q, \Sigma, q_0, F)$ a 0-reversible automaton, e an example.

Ensure: h' a 0-reversible automaton, least general generalisation of h subsuming e.

```
 1: $i = 1$ ; $q = q_0$
    /* following existing states */
 2: while $\delta(q, e_i)$ is defined do
 3:     $q = \delta(q, e_i)$ ; $i = i + 1$
 4: end while
    /* creating new branch for remaining letters */
 5: while $i \leq |e|$ do
 6:     create state $q'$, add $q'$ to $Q$
 7:     add $(q, e_i, q')$ to $\delta$
 8:     $i = i + 1$
 9: end while
10: add $q$ to $F$
    /* merging */
11: merge all states in $F$
12: repeat
13:     if $\exists A, B \in Q$ and $\exists l \in \Sigma$ such that $\delta(A, l) = \delta(B, l)$, merge $A$ and $B$
14:     if $\exists A, B, E \in Q$ and $\exists l \in \Sigma$ such that $\delta(E, l) = \{A, B\}$, merge $A$ and $B$
15: until no fusion
16: return $h' = (Q, \Sigma, \delta, q_0, F)$
```

4 Generalisation of words to balls of strings

4.1 Definitions

We now consider hypotheses as *balls of strings.* A ball is defined by a centre-string o and a radius r, and is noted $B_r(o)$. A ball of strings $B_r(o)$ is the set of all words at distance less or equals to r from o, that is, $B_r(o) = \{w \in \Sigma^* | d(o, w) \leq r\}$. The subsumption test $\succeq$ between a hypothesis $h = B_r(o)$ and an example e is then true if the word is in the ball, that is, $h \succeq e \Leftrightarrow d(e, o) \leq r$.

The distance we use is the edit distance, or Levenshtein distance [4], for which each edit operation (among insertion, deletion, substitution) has a unit cost. It is the minimal number of symbol operations needed to rewrite one word into another one. More formally, let w and w' be two words in Σ^*, we rewrite w into w' in one step if one of the following condition is true:

1. *deletion* : $w = uav$ and $w' = uv$ with $u, v \in \Sigma^*$ and $a \in \Sigma$;

2. *insertion* : $w = uv$ and $w' = uav$ with $u, v \in \Sigma^*$ and $a \in \Sigma$;

3. *substitution* : $w = uav$ and $w' = ubv$ with $u, v \in \Sigma^*$, $a, b \in \Sigma$, $a \neq b$.

We note $w \xrightarrow{k} w'$ if w can be rewritten into w' by means of k operations.

Definition. Edit Distance
The *edit distance* between two words w and w', noted $d(w, w')$, is the smallest k such that $w \xrightarrow{k} w'$.

The edit distance $d(w, w')$ can be computed in $\mathcal{O}(|w| \cdot |w'|)$ time by dynamic programming [5]. Basically, we compute a $|w| \times |w'|$ matrix M, where $M[i][j]$ is the edit distance between the prefixes $w_{1\ldots i}$ and $w'_{1\ldots j}$. Moreover, this allows us to deduce the required edit operations to go from one word into the other.

4.2 Learning generalised balls

Non-unicity of least general generalisation.
Unlike the generalisation to automata, we can note that with balls of strings, least general generalisation are not unique anymore.

Example.
Let $E = [a, b, ab]$, $h = B_1(a)$ and $h' = B_1(b)$. Both hypotheses subsume the examples ($h \succeq E$ and $h' \succeq E$) but $h' \not\succeq h$ and $h \not\succeq h'$!

This property is obvious in $\mathbb{R}^2$; in Figure 1, three points are subsumed by several disk-hypotheses that are not comparable to each other.

Thus, we are definitely not in the ideal case of Section 2 where least general generalisation is unique.

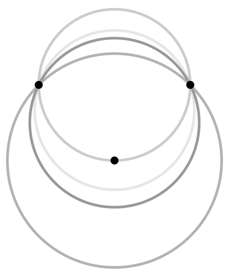

Figure 1: Endless number of disks containing 3 points and non comparable to each other.

One could claim that there is nonetheless a *smallest ball* containing all the examples as we can see in Figure 1. But there is no reason that the smallest ball should be the least general generalisation. Indeed, it is not contained in the other ones, thus the two concepts are different. Furthermore, there is a computational barrier if we made this choice: finding the *centre string* of a set is NP-hard [6].

Monotonic generalisation operator.
To tackle theses problems, we propose the incremental Algorithm 5 (called G-BALLS) as the generalisation operator for balls of strings.

Algorithm 5 Generic algorithm G-BALLS of the generalised ball

```
Require: $h = B_r(o)$ a ball, $e$ an example.
Ensure: $g \in \mathcal{H}$ a least general generalisation of $h$ subsuming $e$ ($g \succeq h$ and $g \succeq e$).
1: $p = o \xrightarrow{*} e$ /* a shortest path */
2: Let $u$ be a string on the path $p$ /* $p = o \xrightarrow{x} u \xrightarrow{y} e$, $x + y = d(o,e)$ */
3: $x = d(o,u)$
4: $y = d(u,e)$
5: $k = \max(x + r, y)$
6: return $B_k(u)$
```

This algorithm requires a method to choose the new centre u on the path p, but whatever this choice is, we keep the *monotonic property*, that is the new ball subsumes the new example and the previous hypothesis:

- $d(u,e) = y \leq k \implies e \in B_k(u) \implies g \succeq e$;
- $\forall w \in B_r(o)$, $d(o,w) \leq r$ and with the triangular inequality $d(u,w) \leq d(u,o) + d(o,w)$, we deduce $d(u,w) \leq x + r \leq k$, so $w \in B_k(u)$ and then $g \succeq h$.

The algorithm mainly relies on the computation of the path between the centre and the new example, so it is, as the edit distance, polynomial in the length of the words. Unfortunately, the downside of the monotony and the complexity gain is that the new hypothesis is not always a least general generalisation anymore, but the balls of strings combinatorial complexity keeps us from a better construction. For instance: **Example.**
let $E = [a, b]$. The first hypothesis is the first example, that is $h = B_0(a)$. Then, as the path between the centre and the new example is $c : a \xrightarrow{1} b$, there are two ways of choosing u. Either G-BALLS$(h, b) = B_1(a)$, or G-BALLS$(h, b) = B_1(b)$. But the ball of radius 1 centred on the empty word $B_1(\lambda)$ contains E and is more specific than both hypothesis: $B_1(\lambda) \subseteq B_1(a)$ and $B_1(\lambda) \subseteq B_1(b)$.

However, let us note that the result of G-BALLS depends on the presentation order of examples and this is a suitable property for our ensemble methods.

CG properties.
At last, the CG is no more monotonic.

Example.
Let us suppose that the strategy for the choice of the new centre is to always take the new centre at distance 1 from the old one ($x = 1$). If the examples are λ, b, a, and the counter-example is bb, the produced hypotheses are, in order:

- $B_0(\lambda)$, the first hypothesis;
- $B_1(b)$, which is rejected since it contains bb;
- $B_1(a)$, which is accepted.

And yet, $B_1(a)$ contains b, while the addition of b was rejected in the previous step.

The only consequence of this non-monotony is AdaBoost-like implementations less efficient since covering of hypotheses should be systematically computed.

Summary for G-BALLS and comparison with LGG operators.

	LGG	G-BALLS
unique LGG	✓	×
sensibility to order	×	✓
monotony of G-BALLS	✓	✓
monotony of CG	✓	×

To summarise the properties of generalisation to balls of strings, we can then say that: G-BALLS does not produce the *least general* generalisation but is monotonic; CG depends on the presentation order of the examples and produces correct hypotheses but is not monotonic.

The only point remaining now is to choose a minimal rewriting path p, and a centre u on it.

4.3 Strategies to compute the centre of the new ball

In Algorithm 5, p is a rewriting path from o to e. Yet, there are many ways to go from a word to an other. In order to always compute the same path $p = o \overset{*}{\rightarrow} e$, we use the matrix M used for the edit distance. Starting from the last entry, we go back according to the origin of the resulting computation, in such (arbitrary) ways:

- if $M[i][j]$ can come from a deletion or another operation, we choose the deletion;
- if $M[i][j]$ can come from an insertion or a substitution, we choose the insertion;
- once all edit operations are found, we execute them from left to right.

We denote such a path as an *edit path*. By setting these choices, the algorithm CG is then deterministic and depends on the order of the examples.

For instance, let w = ABACD and w' = EAFCGD. In Table 1 we show the computation of the edit distances d(ABACD, EAFCGD) and d(EAFCGD, ABACD). Gray entries indicate the trace of the edit operation selection for the edit path computation, as previously defined. If we apply the operation from left

		E	A	F	C	G	D
	0	**1**	2	3	4	5	6
A	1	1	**1**	2	3	4	5
B	2	2	2	**2**	3	4	5
A	3	3	2	**3**	3	4	5
C	4	4	3	3	**3**	**4**	5
D	5	5	4	4	4	4	**4**

		A	B	A	C	D
	0	1	2	3	4	5
E	1	**1**	**2**	3	4	5
A	2	1	2	**2**	3	4
F	3	2	2	**3**	3	4
C	4	3	3	3	**3**	4
G	5	4	4	4	**4**	4
D	6	5	5	5	5	**4**

Table 1: Edit path computation matrices of ABACD $\overset{*}{\rightarrow}$ EAFCGD (left) and EAFCGD $\overset{*}{\rightarrow}$ ABACD (right)

to right, we obtain the following edit path:

- $\underline{\cdot}ABACD \rightarrow EA\underline{B}ACD \rightarrow EAF\underline{A}CD \rightarrow EAFC\underline{\cdot}D \rightarrow EAFCGD$
- $\underline{E}AFCGD \rightarrow A\underline{\cdot}AFCGD \rightarrow ABA\underline{F}CGD \rightarrow ABAC\underline{G}D \rightarrow ABACD$

Finally, several strategies are conceivable to choose the new centre u along the edit path. This choice relies on the problem we are dealing with, or heuristics if we want to give importance to the seed, or favour long examples, *etc.*

- simple: $x+r=y$ (if k is even, otherwise, $x+r=y+1$). *Naive* version, that we would use in Euclidian space. It "minimises" the new ball radius.
- seed: weighted by the number of examples in the ball. The seed is privileged.
- longCentre: weigthed by the old centre length $\left(x = d(o,e) \times \left\lceil \frac{d(o,e)}{d(o,e)+|o|} \right\rceil\right)$. The longer the centre, the closer we come to it.
- shortCentre: weigthed by the old centre length $\left(x = d(o,e) \times \left\lceil \frac{|o|}{d(o,e)+|o|} \right\rceil\right)$. The longer the centre, the farther we move away from it.
- random: the new centre is on the path, randomly chosen ($x \in [0; d(o,e)]$).

We have obviously considered balls centred on the first example. In this case, neither LGG nor CG are needed to learn the radius: taking the distance to the nearest counterexample is enough. We are not driven by examples anymore and we lose in diversity: each example leads to one unique ball. These suspicions are confirmed with poor results in experiments with this strategy.

5 Experimentations

Among the methods that are suitable for including our generalisation algorithm, we will keep GLOBOOST, previously described in Algorithm 2 and implemented in the VOLATA[1] system.

We have set the following protocol: 10-fold cross validation, with 10 runs of GLOBOOST on each fold. A given result is then the average of 100 runs.

[1]`http://www.grappa.univ-lille3.fr/~torre/Recherche/Softwares/volata/`

5.1 UCI Repository datasets

In this section, we will use the sequential datasets from the *UCI Repository* [7], namely: `tic-tac-toe`, `badges`, `promoters`, `us-first-name` and `splice`. With these few problems, we tackle alphabets of size 3 to about 30. In the described protocol, the data is split in 90% for learning an 10% for testing. Our goal is to compare GLOBOOST instantiated with least general generalisation computations to classical grammatical inference methods (such as RPNI [8]). Results are given Table 2 (with 1 000, 10 000 and 100 000 balls, 1 000 automata, all produced by GLOBOOST, for each considered problem).

Missing values are due to a lack of time with our available implementations.

On each problem, one of our method is the best one. Ensemble method gives better performances in prediction. GB-B is generally the best choice, then comes GB-TSSI. GB-ZR is not very good: it can be explained by the fact that the 0-reversible class is rich and leads to nearly learn by heart on some data. A unique 0-reversible can then subsume all positive data without accepting counterexamples. We will now concentrate on balls of strings as hypotheses.

5.2 Handwritten digit classification

We consider now the *Nist special database 3*. This database consists in 128 × 128 bitmap images of handwritten letters and digits. We will focus on a subset of digits, written by 100 different writers. Each class (from 0 to 9) has about 1 000 instances, giving a 10 568 digits corpus.

As we are working on words, each image is transformed in an octal string, with the algorithm described in [9]: from the upper left pixel, we follow the border of the digit until going back to the first one. Each direction gives a different letter of the string (see Figure 2).

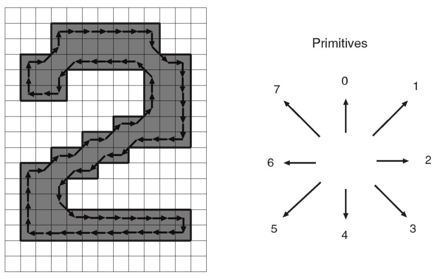

Figure 2: Handwritten digit example. The corresponding string is "2"=22222 24324444466656565654322222222466666666660000212121212100766665466600210

We aim at comparing our approach to the one of [10], thanks to the use of SEDiL [11] and a weighted edit matrix. The matrix is learnt on the same data as

Table 2: Precision of the methods on UCI Repository databases. GB-M denotes GLOBOOST instancied by M, with M a k-TSS automata computation(TSSI), a 0-reversible computation (ZR), or a ball of strings computation (B). In this case, the choice of the new centre strategy is given in subscript.

(references)	tic-tac-toe	badges	promoters	first-name	splice
Majority	65.34 %	**71.43 %**	50.00 %	81.62 %	50.26 %
RPNI	91.13 %	62.24 %	-	81.42 %	-
TRAXBAR	90.81 %	57.48 %	56.60 %	81.37 %	**58.33 %**
RED-BLUE	**93.89 %**	61.09 %	**63.02 %**	**82.83 %**	54.65 %

(GLOBOOST $\times$1 000)	tic-tac-toe	badges	promoters	first-name	splice
GB-TSSI	91.47 %	**72.69 %**	**61.13 %**	**89.50 %**	**78.07 %**
GB-ZR	**98.36 %**	71.43 %	50.00 %	83.07 %	-

(GLOBOOST $\times$1 000)	tic-tac-toe	badges	promoters	first-name	splice
GB-B$_{\text{SIMPLE}}$	92.64 %	81.10 %	**88.58 %**	87.24 %	**93.78 %**
GB-B$_{\text{SEED}}$	**92.77 %**	**81.72 %**	86.13 %	**87.45 %**	93.63 %
GB-B$_{\text{LONGCENTRE}}$	91.04 %	81.12 %	86.53 %	86.84 %	93.48 %
GB-B$_{\text{SHORTCENTRE}}$	74.89 %	80.43 %	87.55 %	86.95 %	92.70 %
GB-B$_{\text{RANDOM}}$	92.62 %	80.41 %	87.63 %	87.10 %	93.76 %

(GLOBOOST $\times$10 000)	tic-tac-toe	badges	promoters	first-name	splice
GB-B$_{\text{SIMPLE}}$	94.54 %	81.38 %	**88.90 %**	88.59 %	95.16 %
GB-B$_{\text{SEED}}$	94.14 %	82.04 %	86.30 %	88.93 %	95.29 %
GB-B$_{\text{LONGCENTRE}}$	94.69 %	**82.21 %**	86.75 %	**89.47 %**	95.54 %
GB-B$_{\text{SHORTCENTRE}}$	74.46 %	81.12 %	88.63 %	89.37 %	**95.83 %**
GB-B$_{\text{RANDOM}}$	**94.69 %**	81.39 %	88.43 %	88.80 %	95.63 %

(GLOBOOST $\times$100 000)	tic-tac-toe	badges	promoters	first-name	splice
GB-B$_{\text{SIMPLE}}$	94.43 %	81.21 %	**89.79 %**	88.72 %	**96.05 %**
GB-B$_{\text{SEED}}$	94.26 %	81.39 %	86.58 %	89.11 %	95.64 %
GB-B$_{\text{LONGCENTRE}}$	**95.03 %**	**81.98 %**	87.58 %	**89.93 %**	95.68 %
GB-B$_{\text{SHORTCENTRE}}$	74.45 %	81.13 %	89.20 %	89.88 %	**96.02 %**
GB-B$_{\text{RANDOM}}$	94.90 %	81.36 %	89.08 %	89.06 %	95.42 %

Marc Sebban with a stochastic transducer on 8 000 (input, output) pairs of strings (the input is the string from the learning set, the output the 1-nearest-neighbour). The final class is given according to the 1-nearest-neighbour computed with the weighted distance matrix learnt. We have kept the same protocol (10-fold cross validation), with 10% of the data for the learning set, 90% for the test set (the matrix of SEDiL has been learnt on the same test examples, thus inducing a bias in its favour). Results are given Table 3.

Table 3: Precision on the Nist special database 3, for 1 000, 10 000 and 100 000 produced balls by globoost (SEDiL performance: 95.86 %)

	×1 000	×10 000	×100 000
GB-B$_{\text{SIMPLE}}$	92.59 %	95.14 %	95.57 %
GB-B$_{\text{SEED}}$	93.74 %	95.77 %	96.16 %
GB-B$_{\text{LONGCENTRE}}$	93.64 %	95.89 %	96.22 %
GB-B$_{\text{SHORTCENTRE}}$	92.92 %	95.73 %	96.17%
GB-B$_{\text{RANDOM}}$	**93.81 %**	**95.93 %**	**96.27 %**

5.3 Experimental observations and discussion

Apart from the shortCentre counterperformance on the tic-tac-toe problem, we can consider that our strategies to compute the new centre are very close. Note also that, with few exceptions, predicting qualities increase with the number of produced balls. Finally, combining balls is more competitive in prediction terms than the other tested methods, especially on genomic data (`promoters` and `splice` problems) and on handwritten recognition, where we overcome SEDiL in spite of our protocol.

Being able to produce 100 000 hypotheses is characteristic of balls of strings. It is inconceivable for 0-reversible or k-TSS automata. On the one hand, runs of these algorithms are too long to give such a large amount of hypotheses this quickly; with SEDiL, it is the classification that requires a quadratic number of distance computations. On the other hand, the produced automata are quickly the same. In other words, balls are diverse and fast to compute. Note that wide diversity is usually considered as an important point for ensemble method [12].

Another observation is that for each experiment, examples are on the border of the learnt ball and its centre is never an example of the concept. Even if we can explain this by our hypotheses construction and the intrinsic properties of balls, this is nevertheless noteworthy. Indeed, when used to learn from noisy data (as in [13]), the centre is usually a non-noisy data, and the radius is seen as a noise tolerance level. Here, the centre of the final hypothesis is rather considered as a median string of the positive examples.

Example.
On the `tic-tac-toe` data set, which encodes possible board configurations at the end of the game, positives examples being "win for `x`": we learn the ball with radius 5 and centre `bbbb`. It covers no negative examples but 120 positive ones, all of them at distance 5 from the centre: `xxxoobbbb`, `xobxbbxbo`, `xbboxbobx`, `obxbbxobx`, `boxoxbxbb`, `bbxobxobx`, etc.

Example.
On the `us-first-name` data set, which contains american first name, classes being female versus male first name: we learn the ball with radius 7 and centre `LRLRTSVKCA`. It covers 346 female first names but no male first names. Here again, covered examples are on the border of the ball.

These *hollow* balls are also part of the proof of the balls VC-dimension [14].

Theorem. [14]
The VC-dimension of balls, with a 2-letter alphabet, is infinite.

Proof. We take n words, all of length n, defined as follows: the ith word is made with only `a`s, except for the ith letter which is a `b`. Let us suppose now that these words are labelled: k positives, $(n-k)$ negatives. We can build a ball covering only positive examples as follows: the centre is the word of length n that has `b`s at the same places than the positive examples, `a`s everywhere else, and the radius is $(k-1)$. By construction, positive examples can be reached from the centre by $(k-1)$ substitutions, and negative examples are at distance strictly greater. □

In this proof, we can note that the ball contains more words than the sample set (thus there is a generalisation) and that words are on the border of the ball (indicating that the learnt ball remains relatively specific to the sample).

6 Conclusion

Our goal in this paper is the classification of sequences, by deciding whether a word belongs in some language or not. In other words, we try to guess a target language, by minimising the generalisation error. We have chosen to integrate classical grammatical inference techniques in a general framework resulting from supervised classification: our hypotheses are automata or balls of strings, that we combine using GLOBOOST algorithm.

We have shown through experiments that our approach is generally better than classical methods and require usually less examples: we learn a combination of automata that are individually simpler than a unique corresponding automaton. Leveraging grammatical inference learners induces good sequence classifiers.

Although least general generalisation is supposed to be unique, our method can cope with multiple ones such as balls of strings. We have then considered to follow one of the generalisations. The ball with the smallest radius is certainly attractive, but its computation is exponential. Finally, we have chosen a more general ball, still close to the examples.

Experimental results tend to show that our approach is valid: balls of strings combinations are better than automata combinations on classical problems of sequences classification, and than the reference method on the handwritten recognition problem. Moreover, learning balls of strings is fast since operations are on words; on the contrary, operations on automata are more complex (merging, determinisation, *etc.*).

Finally, we have been able to deal with multiple least general generalisation in the VOLATA framework, dedicated to unique least general generalisation. This allows us to explore the integration of many more hypotheses classes. Among other perspectives, we think that our methods can be improved by using weighted edit distance, learning one distance for each specific problem. At last, we are encouraged by the good results in genomic data to experiment in this field.

Acknowledgments. We would like to thank Jean-Christophe JANODET for providing us the result about the VC dimension of balls and Marc SEBBAN for the discussions on handwritten recognition and SEDiL.

This work was partially supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the *Contrat de Projets Etat Region* (CPER) 2007-2013.

References

[1] Angluin, D.: Inference of reversible languages. Journal of the ACM **29**(3) (1982) 741–765

[2] García, P., Vidal, E.: Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. **12**(9) (1990) 920–925

[3] de la Higuera, C., Janodet, J.C., Tantini, F.: Learning languages from bounded resources: The case of the dfa and the balls of strings. In Clark, A., Coste, F., Miclet, L., eds.: Proceedings of the 9th International Conference in Grammatical Inference. Volume 5278 of Lecture Notes in Artificial Intelligence., Springer (2008) 43–56

[4] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4) (1965) 845–848

[5] Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–178

[6] de la Higuera, C., Casacuberta, F.: Topology of strings: median string is NP-complete. Theoretical Computer Science **230** (2000) 39–48

[7] Asuncion, A., Newman, D.: UCI machine learning repository (2007)

[8] Oncina, J., García, P.: Identifying regular languages in polynomial time. In: Advances in Structural and Syntactic Pattern Recognition, World Scientific Publishing (1992) 99–108

[9] Micó, L., Oncina, J.: Comparison of fast nearest neighbour classifiers for handwritten character recognition. Pattern Recognition Letter **19**(3-4) (1998) 351–356

[10] Oncina, J., Sebban, M.: Learning stochastic edit distance: Application in handwritten character recognition. Pattern Recognition **39**(9) (2006) 1575–1587

[11] Boyer, L., Esposito, Y., Habrard, A., Oncina, J., Sebban, J.: Sedil: Software for Edit Distance Learning. In Daelemans, W., Goethals, B., Morik, K., eds.: Proceedings of the 19th European Conference on Machine Learning, Springer (2008) 672–677

[12] Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning **51**(2) (2003) 181–207

[13] Tantini, F., de la Higuera, C., Janodet, J.C.: Identification in the limit of systematic-noisy languages. In: Proceedings of the 9th International Conference in Grammatical Inference. (2006) 19–31

[14] Janodet, J.C.: The vapnik-chervonenkis dimension of balls of strings is infinite (2010) Personal Communication.