Segment and Combine Approach for Biological Sequence Classification

Pierre Geurts, Antia Blanco Cuesta, Louis Wehenkel
Department of Electrical Engineering and Computer Science
CBIG - Center of Biomedical Integrative Genoproteomics
University of Liège, Belgium
Email: p.geurts@ulg.ac.be, antiablanco@hotmail.com, l.wehenkel@ulg.ac.be

***Abstract*— This paper presents a new algorithm based on the segment and combine paradigm, for automatic classification of biological sequences. It classifies sequences by aggregating the information about their subsequences predicted by a classifier derived by machine learning from a random sample of training subsequences. This generic approach is combined with decision tree based ensemble methods, scalable both with respect to sample size and vocabulary size. The method is applied to three families of problems: DNA sequence recognition, splice junction detection, and gene regulon prediction. With respect to standard approaches based on n-grams, it appears competitive in terms of accuracy, flexibility, and scalability. The paper also highlights the possibility to exploit the resulting models to identify interpretable patterns specific of a given class of biological sequences.**

I. Introduction

Automatic sequence classification is a rather generic problem which appears for instance in text categorization, computer-user modeling, automatic classification of alarm logs, and intrusion detection. In the context of bioinformatics, automatic sequence classification can be applied both to genomic data (DNA or RNA strings) as well as to proteomic data (strings of amino-acids), and there is a multitude of applications ranging from fast database search to the identification of patterns of some specific physical properties.

A straightforward approach for the classification of biological sequences is to use a nearest-neighbor kind of approach together with a distance measure based on sequence alignment (see [1] and the references therein). However, several other approaches based on the use of more sophisticated machine learning methods have been proposed in recent years to tackle these problems in a way less dependent on an *a priori* defined distance measure.

For example, in [2] and [3], a pattern discovery technique is first applied to extract a set of patterns that appear frequently in sequences of a certain class, and then the frequencies of occurrence of these patterns are used as inputs to machine learning techniques in order to induce a classifier. In [2] decision trees are used and in [3] linear discriminant analysis is used for the machine learning step. Both publications consider the problem of the prediction of co-regulated genes from upstream sequences. In a similar way, [4] derives linear discriminants based on a set of pre-defined features.

A step further in this direction (see e.g. [5]) consists in generating a very large number of candidate features (e.g. position dependent symbolic features, position independent features such as n-grams, ...), and then using feature selection techniques and/or methods like support vector machines and the Naive Bayes method to induce a classification model.

Further applications of support vector machines with string kernels have also been recently proposed for the classification of proteins based on their amino-acid sequences [6], [7].

Along these lines, we propose another quite generic and purely data driven approach for the automatic classification of biological sequences. The method belongs to a new paradigm for the classification of structured data, called segment and combine [8], [9], which, in the present context, classifies a sequence by aggregating the classifications of its subsequences of a given length. To this end, a subsequence classifier is derived by machine learning from a sample of subsequences randomly extracted from the sequences given in the original training set. Within this paradigm, we propose to use machine learning algorithms based on ensembles of randomized decision trees, which are highly scalable both with respect to sample size and vocabulary size. The resulting algorithm is illustrated on three families of genetic sequence classification problems: DNA sequence recognition, splice junction detection, and gene regulon prediction. It is compared to standard approaches based on n-grams, in terms of accuracy, flexibility, and scalability. The paper highlights also the possibility to exploit the method to identify interpretable patterns specific to a given class of biological sequences.

The rest of the paper is structured as follows. In Section II, we define the problem of genetic sequence classification and describe our test problems. In Section III, we introduce the proposed algorithm and discuss its intrinsic properties. In Section IV we present the results of its application on six test problems and compare them with those obtained by the n-gram approach. The paper ends with our conclusions and suggestions for future work.

II. Genetic sequence classification

A. Problem definition

Let us denote by $\mathcal{S}$ a discrete alphabet, i.e. a finite set of symbols. In the case of genetic sequences, this alphabet will be

TABLE I

SUMMARY OF THE DATASETS

Dataset	N_d	L	k	Protocol
DNA-vs-random	1000	800	2	holdout 500
Splice	3190	60	3	holdout 2000
MSN2	112	800	2	10-fold cv
ZAP1	104	800	2	10-fold cv
GLN3	62	800	2	10-fold cv
MIG1	52	800	2	10-fold cv

the set of 4 nucleotides $\{A, C, G, T\}$. A sequence (or string) S^L is the concatenation of symbols from the alphabet:

$$S^L = s_1 s_2 \dots s_L \text{ with } s_i \in \mathcal{S}, i = 1, \dots, L,$$

where L is the length of the sequence. In what follows, we will denote by $S^L(i) = s_i$ $(i = 1, \dots, L)$ the symbol at position i in the sequence S^L.

A learning sample (or dataset) is a set (ordering is considered irrelevant at this level) of N preclassified sequences denoted by

$$LS_N = \left\{ \left(S_i^{L_i}, c_i \right) \middle| \, i = 1, \dots, N \right\},$$

where $S_i^{L_i}$ is the i^{th} sequence of length L_i and c_i is its class chosen among $\mathcal{C} = \{1, \dots, k\}$. The length L_i of the different sequences of the dataset are not assumed to be identical. In our problems, however, all sequences of the same dataset will be of equal length, i.e. $L_i = L, \forall i = 1, \dots, N$.

The objective of the sequence classification problem is to derive from LS_N a classifier $\hat{c}(S^L)$ which predicts the output class of an unseen sequence S^L as accurately as possible.

B. Datasets

The datasets we used correspond to several practical problems. Their main characteristics are summarized in Table I in terms of number of sequences N_d, sequence lengths L, number of classes k, and the protocol used in Section IV for the validation. Each problem is briefly discussed below.

1) DNA recognition: This dataset is an artificial problem. The aim of this problem is to recognize real DNA sequences from randomly generated sequences of nucleotides. The dataset contains 1000 sequences of length 800[1]. Each of the 500 positive sequences corresponds to the 800 base pairs upstream from the start codon of a gene randomly selected from the yeast *Saccharomyces cerevisiae* genome. The 500 negative sequences were obtained by drawing each nucleotide independently at random from a distribution matching the frequency of occurrence of the nucleotides in the yeast genome. Both types of sequences were obtained using the Regulatory Sequence Analysis Tools ([10]) available at (http://rsat.ulb.ac.be/rsat/). Since DNA is known to be highly non-random, it should be possible to discriminate random from true DNA sequences.

[1]This length has been chosen to match the sequence length used in the regulon datasets (see Section II-B.3)

2) Splice junction: One quite popular application of genetic sequence classification is the prediction of coding regions in DNA. In Eukaryotic organisms, usually only part of the sequence corresponding to a gene really codes for proteins: coding regions (called exons) are interrupted by non-coding regions (called introns). During the transcription, the DNA sequence corresponding to a gene is transcribed into a pre-message RNA molecule. This pre-message RNA molecule is then turned into a message RNA molecule by excising the non coding regions. This latter mechanism is called splicing. The prediction of the splice sites in DNA (junction between intron and exon) is interesting both for the analysis of unknown regions in the genomes but also for understanding the biological mechanisms responsible for the splicing.

The dataset we used contains regions of 60 nucleotides obtained from Genbank 64.1 primate data. The center of each sequence corresponds either to a junction between intron and exon (an acceptor), a junction between exon and intron (a donor), or no junction at all (with 25% I-E junction, 25% E-I junction, and 50% of non junction). A model obtained from this dataset can thus be used to locate coding regions of the DNA by sliding a window of size 60 over the DNA and detecting the limit between coding and non coding regions with this model. The dataset was obtained from the UCI machine learning repository [11] and it has been used for example in [12], [13].

3) Gene regulon: A regulon is a set of genes that are targeted by the same transcription factor. It is assumed that these transcription factors bind to some specific sites in the upstream sequence of the genes. So, from this upstream sequence, it should be possible to predict whether or not a gene will be regulated by a specific transcription factor.

These datasets were obtained from the study carried out in [3]. Each dataset corresponds to a different regulon from the yeast *Saccharomyces cerevisiae* and is denoted by the name of the corresponding transcription factor, namely MSN2, MIG1, ZAP1, and GLN3. We randomly selected four regulons from [3] among those that correspond to a reasonable number of genes (> 25). Each positive sequence in the datasets corresponds to the 800 base pairs upstream from the start codon of a gene of the regulon. Negative sequences correspond to the upstream sequences from genes randomly selected in the yeast genome. To balance the two classes, we use exactly the same number of positive and negative examples. The number of sequences in each dataset (corresponding to twice the number of genes in each regulon) is reported in Table I. All sequences were retrieved using the RSA tools (http://rsat.ulb.ac.be/rsat/). As noted in [3], regulons were not defined from an analysis of upstream sequences but from experimental studies. Hence their classification can be considered reliable. Random genes, however, may actually be member of the regulon since they are extracted at random from the yeast's genome.

TABLE II

THE TRAINING AND TEST STAGES OF THE SEGMENT AND COMBINE ALGORITHM

Training a length ℓ subsequence classifier

Subsequences sampling: For $j = 1, \ldots, N_s$ choose a sequence index $i_j \in \{1, \ldots, N\}$ randomly, then choose a subsequence offset $p_j \in \{0, \ldots, L_{i_j} - \ell\}$ randomly, and create a scalar attribute vector

$$S_{i_j}^{\ell, p_j} = \left(S_{i_j}^{L_{i_j}}(p_j + 1), \ldots, S_{i_j}^{L_{i_j}}(p_j + \ell) \right)$$

concatenating the values of the sequence between positions $p_j + 1$ and $p_j + \ell$. Collect the samples in a training set of subsequences

$$LS_{N_s}^{\ell} = \left\{ (S_{i_j}^{\ell, p_j}, c_{i_j}) \middle| \, j = 1, \ldots, N_s \right\},$$

where each subsequence is classified as the sequence from which it was taken.

Classifier training: Use a (propositional) supervised base learner to build a subsequence classifier from the subsequence samples $LS_{N_s}^{\ell}$. The "classifier" is supposed to return a class-probability vector $P_c^{\ell}(S^{\ell})$, each component of which estimates the probability of the subsequence S^{ℓ} being taken from a sequence of one of the k classes.

Classifying a sequence by voting its subsequences.

For a new sequence S^L, extract systematically all its subsequences of length ℓ, $S^{\ell,i}, \forall i \in \{0, \ldots, L - \ell\}$, and classify it according to

$$\hat{c}(S^L) \stackrel{\triangle}{=} \arg\max_c \left\{ \sum_{i=0}^{L-\ell} P_c^{\ell}(S^{\ell,i}) \right\}.$$

Note that if the base learner returns $0/1$ class indicators, the aggregation step merely selects the class receiving the largest number of votes. A class-probability estimator may also be derived by normalization

$$P_c(S^L) \stackrel{\triangle}{=} \left(\sum_{i=0}^{L-\ell} P_c^{\ell}(S^{\ell,i}) \right) \left(\sum_{c} \sum_{i=0}^{L-\ell} P_c^{\ell}(S^{\ell,i}) \right)^{-1}.$$

III. METHODS

A. Segment and Combine approach

Assuming all sequences in the dataset are of the same length L, the most direct way to handle such data using machine learning techniques is to transform each sequence S_i^L of the learning sample into a vector of L symbolic-valued variables:

$$(S_i^L(1), S_i^L(2), \ldots, S_i^L(L)) \tag{1}$$

and to apply a standard supervised learning algorithm using this vector as input variables. Although straightforward, this approach has two limitations. First, since standard supervised learning methods assume constant length input vectors, they can not handle directly sequences of different lengths. Second and more importantly, this representation introduces a bias in the learning algorithm towards the detection of position dependent patterns since all input variables are related to a specific position in the sequence. Although this may be appropriate for some applications, for others, it is clear that relaxing this dependence will be necessary.

The idea of the segment and combine approach proposed here is to use the same representation but for the classification of small subsequences extracted at random positions in the given sequences. More precisely, the training and test steps of the segment and combine algorithm are described in Table II. During the training stage, a new learning sample is gathered by randomly sampling N_s subsequences of length ℓ ($\ell \leq \min_i L_i$) in the original learning sample sequences and a subsequence classifier is built from this sample by using a standard propositional learning algorithm. A new sequence is then classified by voting the predictions of all its subsequences given by the subsequence classifier obtained during the training stage.

In addition to the choice of the base learner discussed below, the sole parameters of the method are the number of subsequences N_s and the subsequence length ℓ. In practice, the larger N_s, the higher the accuracy. Hence, the choice of the value of N_s is dictated by computational constraints only. In our experiments that focus on accuracy, we will use a quite large value of N_s equal to 20000. Depending on the other parameters of the problem, it may happen that N_s is greater than the total number of subsequences of length ℓ appearing in the learning sample[2]. In this case, no sampling is done in our implementation and the training set $LS_{N_s}^{\ell}$ is simply taken as the set of all possible subsequences.

On the other hand, the subsequence length ℓ should be adapted to the resolution of the problem. Small values of ℓ force the algorithm to focus on local (shift-invariant) patterns in the original sequences while larger values of ℓ amount to considering the sequences more globally.

Notice also that, in the limit of $\ell = L$ (assuming sequences of identical length), the segment and combine approach will build a classifier on the basis of the original (full length) sequences, and thus boils down to the standard approach mentioned at the beginning of this section.

[2]This happens when $N_s > \sum_{i=1}^{N} L_i - N\ell + N$

TABLE III

NUMBER OF n-GRAMS SUCH THAT $n \leq \ell_n$

ℓ_n	1	2	3	4	5	6	7	8
nb. att.	4	20	84	340	1364	5460	21844	87380

In our experiments, we will analyze the results obtained with increasing values of this parameter, and we will provide for each dataset also the results for $\ell = L$.

B. The n-gram approach

We will compare our method with the results obtained by representing the sequences by their n-gram counts prior to applying supervised learning.

An n-gram is a subsequence of n consecutive symbols extracted from a longer subsequence. These n-grams have been widely used for example in the context of text analysis and data compression. The idea here is to compute the number of occurrences of all possible n-grams of length n smaller or equal to a certain length ℓ_n in each sequence of the learning sample and then use these n-gram counts as input variables for a learning algorithm. Even if the original sequences are of variable length, all sequences are in this way transformed into a vector of numerical variables of fixed size which can be used as input representation for most existing supervised learning methods.

Within the n-gram approach, ℓ_n is a parameter of the method that regulates some tradeoffs. From an information point of view, the larger the value of ℓ_n and the higher the information about the original sequences that is kept by the n-gram representation. However, the number of input variables (attributes) grows exponentially fast with ℓ_n, which is detrimental from a computational point of view (memory requirements, preprocessing, and training times). Also, the increasing number of attributes will result in an increase of variance of the supervised learning algorithm, and hence may actually lead to a decrease of accuracy of the resulting classifiers.

To give an idea, Table III shows the number of attributes corresponding to increasing values of ℓ_n in the case of the alphabet of four nucleotides. In our experiments, we will consider only values of ℓ_n smaller or equal to 6. Notice that by construction this representation destroys most of the information about the position of patterns in the original sequences, specially for small values of ℓ_n.

C. Tree-based classifier induction

In principle, any propositional base learner (support vector machines, k-nearest neighbor, multilayer perceptrons etc.) could be used in the above approaches. We will nevertheless restrict our investigations to the use of decision tree based methods, and in particular to tree based ensemble methods. The main advantages of these methods for our application are their scalability and the fact that they can handle quite naturally symbolic variables.

A decision tree recursively partitions the learning sample with tests based on the input variables (attributes). In the standard approach these tests are binary and based on a single attribute at a time. They merely check the appearance of the observed value of the attribute in a certain subset of all possible values of this attribute. When the tree is grown on the basis of the learning sample, the algorithm chooses at each test node an attribute and a particular subset of its possible values in order to define the split. A split for a numerical attribute is obtained by choosing a cutpoint while a split for a symbolic attribute is obtained by enumerating explicitly a subset of its possible values. The standard approach grows a tree top down, starting with the root node and splitting nodes as long as the corresponding subsets are containing objects of different classes. These trees can also be pruned afterwards, in order to reduce their complexity and adjust their bias-variance tradeoff.

A drawback of single tree classifiers is that they usually suffer from a high variance (even when pruned) which often prevents them from being competitive in terms of accuracy with other methods. Therefore, in addition to single tree classifiers, we consider various tree-based ensemble methods circumventing this problem. They grow several randomized trees and aggregate their predictions by majority vote. They only differ in the way they randomize the tree building procedure.

We consider three different ensemble methods, namely Tree bagging, Random forests and Extra-trees. Tree bagging randomizes trees by growing them from a random bootstrap replica of the original training sample [14]. Random forests is a variant of Tree bagging which further randomizes the attribute choice at each internal node [15]. Extra-trees (which stands for *extremely randomized trees* [16]) generates each tree of the ensemble from the whole learning sample and randomizes the tests by choosing both attribute and splits explicitly at random. For a numerical attribute it draws a cutpoint at random while for a symbolic attribute it draws a random subset of its possible values. The Extra-trees algorithm, detailed in the appendix, has one parameter K which allows to mitigate the strength of randomization and enhance its ability to cope with irrelevant attributes.

In the ensemble methods, the randomization allows together with the averaging over a set of trees to strongly reduce the variance with respect to single trees. On the other hand, the use of unpruned trees with these methods ensures that the bias remains as small as possible. With respect to Tree bagging and to a lesser extent Random forests, Extra-trees have a smaller variance and are significantly faster in the training stage. In particular, in the context of symbolic attributes this is potentially an important advantage because the number of different binary partitions of an alphabet $\mathcal{S}$ grows exponentially with the size of this alphabet. Thus, single trees as well as Tree bagging and Random forest that consider all such partitions for each variable when splitting a tree node have a computational complexity exponential in the size of $\mathcal{S}$, which for large alphabets implies to use some suboptimal heuristics. On the other hand, Extra-trees require only the generation of

TABLE IV
ERROR RATES ON DNA-VS-RANDOM

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	23.60%	24.00%	23.20%	23.20%	24.00%
2	10.80%	9.60%	7.40%	8.00%	9.00%
3	11.00%	10.20%	5.00%	7.20%	7.00%
4	11.00%	10.80%	5.60%	7.20%	6.60%
5	10.60%	10.60%	**4.40%**	7.40%	6.80%
6	10.60%	10.40%	4.80%	7.60%	5.80%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	**10.00%**	6.20%	5.00%	5.80%	9.20%
10	13.20%	**5.00%**	5.20%	**4.40%**	**5.40%**
20	18.80%	11.20%	**4.40%**	5.80%	8.80%
30	19.40%	8.40%	12.60%	11.60%	9.20%
40	26.40%	12.40%	9.60%	12.00%	13.00%
50	26.80%	15.20%	10.80%	11.60%	12.80%
100	28.40%	19.60%	16.20%	17.80%	18.80%
200	29.80%	21.20%	20.00%	21.20%	20.60%
800	37.40%	37.40%	40.80%	36.00%	36.80%

TABLE V
ERROR RATES ON SPLICE

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	46.30%	46.98%	46.89%	47.83%	45.62%
2	44.43%	37.53%	36.00%	37.45%	37.45%
3	41.79%	33.79%	32.34%	32.60%	33.19%
4	35.75%	28.17%	29.70%	31.15%	27.75%
5	30.47%	25.87%	29.45%	25.70%	27.92%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	41.79%	42.64%	41.96%	40.00%	40.85%
10	39.83%	32.09%	29.28%	30.30%	32.26%
20	37.02%	31.23%	29.96%	29.02%	32.17%
30	34.13%	27.32%	26.64%	26.38%	28.85%
40	31.83%	24.94%	24.77%	25.11%	25.45%
50	28.60%	20.09%	22.89%	21.45%	22.55%
60	**5.62%**	**4.68%**	**3.66%**	**5.11%**	**3.40%**

one random binary partition (at each node) for each attribute and thus their complexity is linear with respect to the alphabet size. This could be an important advantage in the context of the application of the method to amino acid sequences.

IV. EXPERIMENTS

We first describe the protocols used to estimate error rates on each problem and the parameters of the different methods. We then discuss the results in terms of the accuracy on all problems. Finally, we illustrate the possibility to extract interpretable information from subsequence classifiers.

A. Protocols and parameters

The protocol for each problem is given in the last column of Table I. On DNA-vs-Random, we use 50% of the data for learning (500 sequences) and 50% for testing (500 sequences). On Splice, we randomly split the dataset into a learning set of 2000 sequences and a test set with the remaining 1190 sequences. Because Regulon datasets are small, we use ten-fold cross-validation to estimate accuracies (using the same folds throughout all our experiments).

To build decision trees, we have adopted the standard CART algorithm described in [17] with the score measure proposed in [18]. With n-grams, single trees are pruned by using cost-complexity pruning by ten-fold cross-validation [17] and the ensemble methods vote over 100 unpruned trees. In the segment and combine approach single trees are not pruned, except when the method degenerates into the use of the original sequences (i.e. for $\ell = L$). This is justified because averaging over subsequences already reduces enough the variance. For the same reason, the number of trees in the ensemble methods was reduced in this case to 50 (except for $\ell = L$).

For Random forests, the number of randomly selected attributes screened at each node was fixed to its default value, which is the square root of the total number of attributes [19] (we denote this method by RF^d). For Extra-trees, we compared two settings: K defined as in the default setting of Random forests (denoted by ET^d) and K equal to the total number of attributes (denoted by ET^N). The first method corresponds to a larger reduction of variance but a higher bias. The second one is more stringent in terms of variable selection and is usually more accurate when there are a lot of irrelevant variables. Notice that Random forests with the latter setting actually degenerate into Tree bagging.

Concerning the segment and combine method, $N_s = 20000$ subsequences are randomly extracted in each case (except when this number is larger than the total number of possible subsequences) and we report below the results obtained with different values of the other parameter ℓ. For n-gram counts, we consider all values of ℓ_n from 1 to 6 except for the largest problem, Splice, where we stopped at $\ell_n = 5$.

B. Accuracy

Tables IV, V, and VI gather the results on the six datasets. We highlight in bold-face the best result in each column and underline the overall best result in each table. The last line in each table corresponds to the degenerated case of the segment and combine approach, when $\ell = L$.

1) DNA vs Random: Results for this problem are reported in Table IV. We observe that both n-grams and segment and combine yield a very good accuracy. In the latter method, the optimal value of ℓ is usually quite small (10-20), suggesting that there exist small position invariant patterns typical of the DNA that do not appear as frequently in random sequences. The important improvement with n-grams when going from $\ell_n = 1$ to $\ell_n = 2$ further shows the already known fact that nucleotides in DNA sequences are not independent.

Among tree-based methods, single trees are clearly inferior while ET^d obtains the lowest error rate (4.40%). We also note that the results depend less strongly on the ensemble method in the segment and combine approach than with n-grams.

2) Splice: On this problem the n-gram approach behaves very badly (see Table V) which is due to its inability to capture

TABLE VI

ERROR RATES ON THE REGULON DATASETS

MSN2

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	53.57%	39.29%	44.64%	42.86%	30.36%
2	**22.32%**	41.96%	35.71%	31.25%	32.14%
3	28.57%	37.50%	40.18%	30.36%	31.25%
4	33.93%	31.25%	27.68%	27.68%	29.46%
5	75.00%	32.15%	44.64%	49.11%	37.50%
6	73.21%	33.93%	36.61%	50.89%	33.93%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	24.11%	**20.54%**	**16.07%**	16.96%	**19.65%**
10	67.86%	56.25%	49.11%	**16.07%**	51.79%
20	41.96%	59.82%	37.50%	53.57%	47.32%
30	49.11%	41.96%	58.04%	27.68%	43.75%
40	48.21%	40.18%	63.39%	17.86%	50.00%
50	35.71%	42.86%	47.32%	20.54%	27.68%
100	33.93%	31.25%	22.32%	22.32%	22.32%
200	43.75%	36.61%	20.54%	31.25%	26.79%
800	56.25%	57.14%	68.75%	75.00%	48.21%

ZAP1

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	58.66%	53.85%	52.89%	50.00%	46.15%
2	25.00%	50.00%	46.15%	52.89%	46.15%
3	39.42%	48.08%	56.73%	51.92%	66.35%
4	55.77%	**28.85%**	40.39%	35.58%	52.89%
5	57.69%	48.08%	30.77%	30.77%	32.69%
6	48.08%	43.27%	29.81%	**21.15%**	50.96%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	**24.04%**	**28.85%**	31.73%	47.12%	70.19%
10	26.92%	35.58%	**26.92%**	28.85%	52.89%
20	25.96%	70.19%	39.42%	44.23%	42.31%
30	29.81%	72.12%	38.46%	53.85%	44.23%
40	41.19%	39.42%	40.39%	48.08%	43.27%
50	50.96%	34.62%	60.58%	46.15%	32.69%
100	66.35%	47.12%	55.77%	73.08%	47.12%
200	50.96%	68.27%	63.46%	43.27%	**28.85%**
800	33.60%	35.60%	47.12%	43.27%	36.54%

GLN3

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	45.16%	41.94%	40.32%	43.55%	41.94%
2	45.16%	**33.87%**	**33.87%**	41.94%	**33.87%**
3	43.55%	43.55%	41.94%	45.16%	38.71%
4	50.00%	51.61%	45.16%	46.77%	48.39%
5	**33.87%**	43.55%	40.32%	40.32%	41.94%
6	38.71%	50.00%	46.77%	**30.65%**	41.94%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	46.77%	**33.87%**	35.48%	32.26%	35.48%
10	43.55%	43.55%	35.48%	37.10%	40.32%
20	46.77%	50.00%	50.00%	41.94%	50.00%
30	41.94%	46.77%	48.39%	48.39%	46.77%
40	37.10%	45.16%	43.55%	40.32%	45.16%
50	38.71%	50.00%	41.94%	43.55%	46.77%
100	40.32%	46.77%	46.77%	43.55%	43.55%
200	43.55%	41.94%	46.77%	46.77%	48.39%
800	67.74%	56.45%	53.23%	59.68%	53.23%

MIG1

n-gram counts					
ℓ_n	ST	BAG	ET^d	ET^N	RF^d
1	42.31%	36.54%	38.46%	46.15%	48.08%
2	42.31%	30.77%	38.46%	34.62%	53.85%
3	51.92%	32.69%	42.31%	48.08%	26.92%
4	40.39%	32.69%	44.23%	42.31%	46.15%
5	48.08%	40.39%	50.00%	40.39%	44.23%
6	48.08%	51.92%	53.85%	50.00%	50.00%
Segment and Combine					
ℓ	ST	BAG	ET^d	ET^N	RF^d
5	**28.85%**	**26.92%**	**25.00%**	**26.92%**	36.54%
10	32.69%	28.85%	30.77%	30.77%	**25.00%**
20	38.46%	32.69%	34.62%	30.77%	34.62%
30	38.46%	30.77%	44.23%	36.54%	32.69%
40	30.77%	40.39%	36.54%	40.39%	32.69%
50	53.85%	36.54%	38.46%	36.54%	34.62%
100	36.54%	40.39%	38.46%	40.39%	44.23%
200	46.15%	36.54%	50.00%	44.23%	50.00%
800	61.54%	53.85%	48.08%	50.00%	53.85%

the perfectly centered information about intron-exon junctions. The segment and combine method gives significantly better results and it is by far optimal in the degenerated case where $\ell = L = 60$. The very strong improvement when going from $\ell = 50$ to $\ell = 60$ is certainly due the very strict centering of the sequences.

We note also that the different ensemble methods yield very close results on this dataset.

3) Regulons: These problems are more difficult from a machine learning point of view, because the datasets are quite small with respect to the length of the sequences and because there is potentially some noise on the labeling of negative sequences. In addition, gene regulation may be a quite complex process involving several factors binding to multiple sites in the upstream sequence of the gene. Hence, the prediction of co-regulated genes might require the detection and the combination of multiple patterns in sequences, which may explain why error rates in Tables VI are so high.

On MSN2 and on MIG1, segment and combine gives better results than n-grams whatever the tree-based method. On ZAP1, n-gram counts are only better than segment and combine with ET^N, which is the combination yielding globally the lowest error rate on this problem. On GLN3, n-grams are systematically better than segment and combine although the difference is not very large.

With segment and combine, the lowest error rate is usually obtained with the smallest values of ℓ. This confirms that regulons are usually defined by the occurrence of small shift-invariant patterns in the upstream sequence of the genes.

We observe also that on all these problems all tree-based methods, even single trees, provide rather close results to each other, while the globally best result is always obtained by one of the two Extra-trees variants.

4) Discussion: Among the two approaches compared, the segment and combine algorithm appears clearly to be the most flexible one. By the adaptation of the subsequence length, it could indeed handle as well problems with highly centered data, like the Splice problem, as problems characterized by position independent patterns, like DNA-vs-random and regulons. The optimization of ℓ for a new problem also gives information about which of these two families the problem belongs to.

On the other hand, n-grams are not appropriate to handle problems characterized by position specific patterns, even when increasing l_n[3]. Furthermore, this approach requires the generation and the counting of all n-grams of a given length and thus does not scale well with respect to its parameter l_n.

Among the tree-based methods, the best results are usually obtained by the Extra-trees method (except on Splice where Random forest were found slightly better). Since this method is also by far the most efficient one from a computational point of view, we consider it to be the most appropriate one in the context of the segment and combine approach.

C. Interpretability

An interesting property of the segment and combine approach is that it is possible to extract useful interpretable information from the subsequence classifiers. Indeed, these classifiers provide for each subsequence of length ℓ a probability vector that estimates for each class the probability that this subsequence has been extracted from a sequence of this class. Patterns specific of one class are thus patterns that are classified by the subsequence classifier with high confidence into this class, i.e. belonging to subsequences that receive a high probability for this class. Among these subsequences, the most interesting ones would be those that also appear frequently in sequences of the considered class, i.e. which have a high rate of covering. This suggests to combine in a score measure both the estimated probability $P(c|S^\ell)$ and $P(S^\ell|c)$, for example in the form of a product.

Note that $P(c|S^\ell)$ can be estimated directly by the subsequence classifier model, whereas $P(S^\ell|c)$ can be estimated by the proportion of training sequences of class c which have the corresponding sequence as subsequence. However, in biological problems there are generally many subsequences that will appear in all sequences irrespectively of their class. Using a product type score measure $P(c|S^\ell)P(S^\ell|c)$, these latter would obtain a rank close to 50%, and hence mask all specific patterns covering less than 50% of the sequences of class c. In order to filter out these latter we propose the following procedure, to identify subsequences representative of a given class c:

1) For each sequence S_i^L of class c in the learning sample:
   a) Extract all different subsequences $S_i^{\ell,j}$ ($j = 0, \ldots, L-\ell$) and compute the probability $P(c|S_i^{\ell,j})$ with the subsequence classifier;
   b) Among these, retain the m subsequences that receive the highest probability for this class.
2) For each subsequence S^ℓ so obtained, compute a score equal to $P(c|S^\ell).P(S^\ell|c)$, where $P(S^\ell|c)$ is estimated by the proportion of sequences of class c from the learning sample where this subsequence appears.
3) Return the subsequences ordered by decreasing value of the score.

[3]The numbers of occurrences of n-grams of maximum length are non zero only each for one sequence of the learning sample and thus do not allow generalization of the resulting rules.

TABLE VII
PATTERNS FOUND ON MSN2 AND ZAP1

MSN2				ZAP1			
Pattern	$P(c\|S^\ell)$	$P(S^\ell\|c)$	Score	Pattern	$P(c\|S^\ell)$	$P(S^\ell\|c)$	Score
ACTCA	1.00	0.61	0.61	CCAAA	0.85	0.75	0.59
CCCTT	0.84	0.71	0.60	TTGCA	0.81	0.67	0.51
CCCCT	0.95	0.59	0.56	ATAAC	0.78	0.69	0.50
ACCTT	0.93	0.59	0.55	TCCGT	1.00	0.52	0.48
CAAGG	0.77	0.71	0.55	TAGAT	0.79	0.62	0.45
AAACG	0.79	0.70	0.55	GTACC	1.00	0.48	0.45
AGGGG	0.92	0.59	0.54	ACACC	0.83	0.56	0.43
ATAAC	0.75	0.71	0.54	GCACT	0.89	0.52	0.43
GGGGT	1.00	0.54	0.54	GGCTT	0.85	0.54	0.42
TCCCC	0.89	0.55	0.49	AACGT	0.87	0.52	0.42

Step 1.b avoids to retrieve subsequences that are frequent but not typical of class c.

We have applied this procedure on the positive instances of the MSN2 and ZAP1 problem, with $m = 10$, $\ell = 5$, and bagging as a subsequence classification method. The ten best patterns together with their $P(c|S^\ell)$, $P(S^\ell|c)$, and scores are given in Table VII. On MSN2 for example, the first pattern appears in 61% (34 among 56) of the positive sequences and never in negative ones (since $P(c|S^\ell) = 1$). The second one is less specific but it appears in 71% of the positive sequences.

Note that here we have used the subsequence classifier to retrieve fully defined patterns. However, tree-based subsequence classifiers do not necessarily use all ℓ nucleotides for the classification of a subsequence. Hence, it would be possible to use the subsequence classifier to find partially defined patterns, i.e., patterns with wildcards. This possibility will be the subject of future investigations. It would be also very interesting to compare the patterns retrieved by our method with those obtained with other more standard pattern extraction techniques used in this field (e.g. [20]).

V. CONCLUSION AND FUTURE WORK

In this paper we have introduced a new segment and combine approach for the automatic classification of biological sequences. This approach is in principle able to cope both with problems characterized by position dependent and position independent patterns, as well as sequences of variable length. From a computational point of view, its combination with extremely randomized trees is most attractive, since this method scales very well when increasing the number of attributes and samples. In particular, in the context of larger alphabet (e.g. proteomic sequence data) this is a strong advantage with respect to existing approaches. The empirical results provided in the paper suggest that the method could be competitive with other state-of-the-art approaches on several genetic sequence classification problems. The paper has also highlighted how this method could be used to explicitly identify patterns specific of a certain class of biological sequences.

In the future, we plan to further develop this method in the context of biological sequence classification problems. First, since the value of ℓ is critical to the performance of the method, it is highly desirable to determine automatically

its value for a given problem. One straightforward way to do this is to use an internal cross-validation procedure in the learning sample (e.g., 10-fold cross-validation) to obtain unbiased estimates of the error corresponding to different values of ℓ and then use the optimal value to grow the final model on the whole learning sample. The evaluation of this procedure will be the subject of future works. In addition, we aim at evaluating the performances of the method on a larger set of test-benches (including proteomic data) along with experimental protocols proposed by other researchers, and to compare its results with a broader set of methods including in particular tree-boosting, Naive Bayes and support vector machines. We intend also to work further on the extraction of informative patterns from an ensemble of subsequence classifiers.

Of course, the application of the method on other types of sequential data, such as for instance text classification would also be interesting.

ACKNOWLEDGMENT

Pierre Geurts is a postdoctoral researcher at the FNRS, Belgium. The authors would like to thank Jacques van Helden for providing the regulon datasets.

REFERENCES

[1] J. T.-L. Wang, S. Rozen, B. A. Shapiro, D. Shasha, Z. Wang, and M. Yin, "New techniques for DNA sequence classification," *Journal of Computational Biology*, vol. 6, no. 2, pp. 209–218, 1999.

[2] Y. Hu, S. Sandmeyer, C. McLaughlin, and D. Kibler, "Combinatorial motif analysis and hypothesis generation on a genomic scale," *Bioinformatics*, vol. 16, pp. 222–232, 2000.

[3] N. Simonis, S. Wodak, G. Cohen, and J. van Helden, "Combining pattern discovery and discriminant analysis to predict gene co-regulation," *Bioinformatics*, vol. 20, pp. 2370–2379, 2004.

[4] M. Zhang, "Discriminant analysis and its application in DNA sequence motif recognition," *Briefings in Bioinformatics*, vol. 1, pp. 331–342, 2000.

[5] Y. Saeys, S. Degroeve, D. Aeyels, P. Rouzé, and Y. Van de Peer, "Feature selection for splice site prediction: a new method using eda-based feature ranking," *BMC Bioinformatics*, vol. 5, p. 64, 2004.

[6] C. Leslie, E. Eskin, J. Weston, and W. S. Noble, "Mismatch string kernels for SVM protein classification," *Advances in Neural Information Processing Systems*, vol. 15, pp. 1417–1424, 2003.

[7] J.-P. Vert, H. Saigo, and Akutsu, *Kernel methods in computational biology*. MIT Press, 2004, ch. Local alignment kernels for biological sequences.

[8] P. Geurts and L. Wehenkel, "Segment and combine approach for non-parametric time-series classification," in *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, October 2005.

[9] R. Marée, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, 2005.

[10] J. van Helden, "Regulatory sequence analysis tools," *Nucleic Acids Research*, pp. 3593–6, 2003.

[11] C. Blake and C. Merz, "UCI repository of machine learning databases," 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[12] M. O. Noordewier, G. G. Towell, and J. W. Shavlik, "Training knowledge-based neural networks to recognize genes in dna sequences," in *Advances in Neural Information Processing Systems*, vol. 3. Morgan Kaufmann, 1991.

[13] H. Hirsh and M. Noordewier, "Using background knowledge to improve inductive learning of dna sequences," in *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*. IEEE Computer Society Press, 1994, pp. 351–357.

[14] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[15] ——, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[16] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," 2005, submitted.

[17] L. Breiman, J. Friedman, R. Olsen, and C. Stone, *Classification and Regression Trees*. Wadsworth International (California), 1984.

[18] L. Wehenkel, *Automatic learning techniques in power systems*. Boston: Kluwer Academic, 1998.

[19] L. Breiman, *Setting up, using, and understanding random forests V4.0*, University of California, Department of Statistics, 2003.

[20] J. van Helden, B. Andre, and J. Collado-Vides, "Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies," *Journal of Molecular Biology*, 1998.

APPENDIX

The Extra-Trees algorithm is described in Table VIII. As score measure, it uses a normalized version of information quantity [18] and it has two meta-parameters: $n_{\min}$, the number of objects required to split a node, and K, the number of random candidate splits. In our experiments, we used the minimal value of $n_{\min} = 2$ and two settings of K: $K = \sqrt{n}$ and $K = n$, where n is the number of input variables.

TABLE VIII

PROCEDURE USED TO BUILD AN ENSEMBLE OF EXTRA-TREES

Build_an_extra_tree_ensemble(LS).
Input: a training set LS. Output: a tree ensemble $\{\mathcal{T}_1,\ldots,\mathcal{T}_T\}$.

- For i=1 to T
  - Generate a tree $\mathcal{T}_i$=Build_an_extra_tree(LS);
- Return $\{\mathcal{T}_1,\ldots,\mathcal{T}_T\}$.

Build_an_extra_tree(LS).
Input: a training set LS. Output: a tree $\mathcal{T}$.

- Return a leaf labeled by class frequencies in LS if
  (i) $\#LS < n_{min}$, or
  (ii) all input variables are constant in LS, or
  (iii) the output variable is constant over the LS,
- Otherwise:
  1) Select K attributes, $\{a_1, ..., a_K\}$, at random, without replacement, among all (non constant) input variables;
  2) Generate K tests $\{T_1, \ldots, T_K\}$, where T_k = Generate_a_random_test$(LS, a_k), \forall k = 1, \ldots, K$;
  3) Select a test T_j such that Score(T_j, LS) = $\max_{k=1,\ldots,K}$ Score(T_k, LS);
  4) Split LS into LS_l and LS_r according to the test T_j;
  5) Build $\mathcal{T}_l$ = Build_an_extra_tree(LS_l) and $\mathcal{T}_r$ = Build_an_extra_tree(LS_r) from these subsets;
  6) Create a node with the test T_j, attach $\mathcal{T}_l$ and $\mathcal{T}_r$ as left and right subtrees of this node and return the resulting tree.

Generate_a_random_test(LS,a)
Input: a training set LS and an attribute a. Output: a test T.

- If the attribute a is numerical:
  - Compute the maximal and minimal value of a in LS, denoted respectively by $a^{LS}_{\min}$ and $a^{LS}_{\max}$;
  - Draw a cutpoint a_{th} uniformly in $]a^{LS}_{\min}, a^{LS}_{\max}]$;
  - Return the test $[a < a_{th}]$.
- If the attribute a is symbolic:
  - Compute $\mathcal{A}_{LS}$ the subset of values from the domain of a, denoted $\mathcal{A}$, that appear in LS;
  - Randomly draw a non empty and proper subset $\mathcal{A}_1$ from $\mathcal{A}_{LS}$ and a subset $\mathcal{A}_2$ from $\mathcal{A}\backslash\mathcal{A}_S$;
  - Return the test $[a \in \mathcal{A}_1 \cup \mathcal{A}_2]$.