Online journal ISSN - 1676-5680
GMR Genetics and Molecular Research
EVOLUTION AND TECHNOLOGY

Splice site prediction using stochastic regular grammars

A.Y. Kashiwabara[1], D.C.G. Vieira[2], A. Machado-Lima[1] and A.M. Durham[1]

[1]Departamento de Ciência da Computação,
Instituto de Matemática e Estatística, Universidade de São Paulo,
São Paulo, SP, Brasil
[2]Bolsa de Mercadorias e Futuros (BM&F), São Paulo, SP, Brasil
Corresponding author: A.M. Durham
E-mail: alan@ime.usp.br

Genet. Mol. Res. 6 (1): 105-115 (2007)
Received August 3, 2006
Accepted November 8, 2006
Published March 20, 2007

ABSTRACT. This paper presents a novel approach to the problem of splice site prediction, by applying stochastic grammar inference. We used four grammar inference algorithms to infer 1465 grammars, and used 10-fold cross-validation to select the best grammar for each algorithm. The corresponding grammars were embedded into a classifier and used to run splice site prediction and compare the results with those of NNSPLICE, the predictor used by the Genie gene finder. We indicate possible paths to improve this performance by using Sakakibara's windowing technique to find probability thresholds that will lower false-positive predictions.

Key words: Splice sites, Gene prediction, Stochastic grammars, Machine learning

INTRODUCTION

The automatic annotation of DNA sequences is a pre-requisite today to keep pace with the rate that genomic data are being generated. One of the difficulties encountered is the correct identification of new genes, especially in eukaryotic organisms. Some approaches to this problem include HMMgene (Krogh, 1997), NetGene2 (Brunak et al., 1991), NNSPLICE (Reese et al., 1997), SpliceView (Hubbard et al., 1999), GeneID (Guigo et al., 1992), FGENEH (Salamov and Solovyev, 2000), Grail (Uberbacher and Mural, 1991), Genscan (Burge and Karlin, 1997), and MZEF (Zhang and Luo, 2003). All these programs utilize intrinsic methods, that is, they are programs that try to recognize statistical patterns in the signal sequences: promoters, start and stop codons, splice sites, etc. Among these signals, splice sites deserve special attention, since they are the ones that define the border between exons and introns.

The techniques utilized for statistical recognition of signals include hidden Markov models (HMM) (Hughey and Krogh, 1996; Baldi and Brunak, 1998), neural networks (Krogh and Vedelsby, 1995; Baldi and Brunak, 1998), discriminant analysis, weight matrix method (WMM) (Staden, 1984), weight array method (WAM) (Zhang and Marr, 1993) and decision trees (Murthy et al., 1994). Some programs use a combination of these methods (Mathé et al., 2002). Genscan, for example, uses WMM, WAM and decision tree to identify splice site candidates. FGENEH, another example, uses a combination of methods based on prediction methods and genome comparison.

However, there is another stochastic technique, stochastic regular grammars (SRGs), that is widely used in the area of computational linguistics (Abe and Warmuth, 1992) but has not been widely applied in the area of computational molecular biology. Grammars are a formalism to describe languages. Grammars can be described by a set of translation rules. We can consider the set of sequences that describe all donor (or acceptor) sites as a language, and therefore use a grammar to define it. Stochastic grammars not only describe languages, but also assign a probability value to each member of the language. SRGs are equivalent to HMMs (Eddy and Durbin, 1994). However, in particular with SRGs, we have algorithms not only for inferring the probabilities of the model (probability values assigned to the rules) but also for building the topology of the machinery (that is, building the rules). This can be an advantage when there is little biological knowledge to be embedded into the solution.

In this article, we present an approach to obtain splice site predictors based on SRG technology, indicating how such predictors can be automatically obtained from a set of grammar inference algorithms controlled by generalization parameters and a training set. We applied this approach successfully to the problem of predicting splice sites in human DNA sequences, comparing the results with the splice site predictor NNSPLICE. The best predictor generated automatically by our approach matched the best performance of NNSPLICE on their own benchmark, with the difference being that their results are based on algorithm parameters being set after the training and specifically for the validation sample.

MATERIAL AND METHODS

The dataset

To train and validate the splice site algorithms, we used sequences from the human genome. These sequences were extracted from the benchmark used at the University of Cali-

fornia (http://www.fruitfly.org/sequence/human-datasets.html) to train and validate the algorithm NNSPLICE, the splice site predictor used by the Genie system.

The benchmark training and test sets

The training set consists of 1116 donors, 1110 acceptors, and 4139 false-donor and 4658 false-acceptor sites. The test set consists of 208 true-donor sites, 208 true-acceptor sites, 782 false-donor sites, and 881 false-acceptor sites (see Table 1). The donor splice site sequences are all 15 bp long, where the first 7 bases are part of the exon sequence and the last 8 bases are part of the intron. We only used donor splice sites that have the GT consensus sequence. The acceptor splice site sequences are all 90 bp long, where the first 70 bp belong to the intronic sequence and the last 20 bp to the exonic sequence. As with the donor sites, we only used acceptor sites that have the AG consensus sequence. From the original benchmark we also used two "negative" datasets, the false-donor and the false-acceptor datasets. These were obtained, respectively, by selecting 15 and 90 bp around the consensus sequences GT and AG that were not in a donor (acceptor) site. As a result we used 8 files: i) splice.test-real.D, with 208 donor samples; ii) splice.test-real.A, with 208 acceptor samples, iii) splice.test-false.D, with 782 "false-donor" samples; iv) splice.test-false.A, with 881 "false-acceptor" samples; v) splice.train-real.D, with 1116 donor samples; vi) splice.train-real.A, with 1110 acceptor samples; vii) splice.train-false.D, with 4139 "false-donor" samples, and viii) splice.train-false.A, with 4658 "false-acceptor" samples. The first four files were used in the comparison test, and the last 4 files for training the classifiers.

Table 1. Size of the training samples.

Sample type	Sample size	
	Donor	Acceptor
Training real	1116	1110
Training false	4139	4658
Testing real	208	208
Testing false	782	881

Definitions

Learning

Let Σ be a set of symbols called *alphabet*. We call *word* a sequence of symbols over Σ^* (that is, any sequence with any number of symbols). The set of all possible words is called *example space*, denoted by **X**; the elements of **X** are called *examples.* A *concept* **c** on **X** is a subset of the example space. A concept **c** can be defined as a function $\mathbf{c: X \rightarrow \{0, 1\}}$, where $\mathbf{c(x) = 1}$ indicates that **x** is *positive example* and $\mathbf{c(x) = 0}$ indicates that **x** is a *negative example*. In the set of all concepts, **C** is called *concept space*, and the set of all concepts that can

be learned is called *hypothesis space*, denoted by **H**. A *sample* of size **m** is a sequence of **m** examples, that is, a member of $\mathbf{X^m}$. A *training sample* of size **m** is an element of $\mathbf{(X \times \{0, 1\})^m}$. A *learning algorithm* relative to a *target concept* **c** is a function **L** that assigns, to each training sample, a hypothesis $\mathbf{h \in H}$.

Grammars

A *stochastic grammar* is a quadruple $\mathbf{G_s = \{V_n, V_t, P, S\}}$ where: i) $\mathbf{V_n}$ is a set of *non-terminal* symbols; ii) $\mathbf{V_t}$ is a set of *terminal symbols*; iii) $\mathbf{S \in V_n}$ is the *start symbol*; iv) $\mathbf{P \in \{(V_n \cup V_t)^* V_n (V_n \cup V_t)^* \rightarrow (V_n \cup V_t^*), p\}}$ is the set of *productions*. In a stochastic grammar we assign to each production a probability **p**, $\mathbf{0 \leq p \leq 1}$, where, to each $\mathbf{A \in V_n}$ and to each $\mathbf{\{A \rightarrow \alpha_i, p_i\} \in P, \Sigma_i p_i = 1}$.

Automata

A stochastic finite state automaton (SFSA) is a quadruple $\mathbf{A = (Q, \Sigma, \pi, M)}$ where: i) **Q** is a non-empty *state set*; ii) $\mathbf{\Sigma}$ is an alphabet; iii) $\mathbf{\pi : Q \rightarrow [0, 1]}$ is a probability distribution over **Q**; iv) $\mathbf{M : Q \times Q \times \Sigma \rightarrow [0,1]}$ is a probability transition matrix, where:

$$\sum_{i \in Q} \pi = 1 \text{ and } \forall i \in Q \sum_{j \in Q, \sigma \in \Sigma} M(i, j, \sigma) = 1$$

Automaton vs grammar

To each stochastic regular grammar there is an SFSA describing the same family of sequences and vice-versa (Aho et al., 1986). Grammars are generally used as description formalism, and automata can be implemented into an efficient (linear time) recognition procedure.

Regular grammar inference

Our goal was to use stochastic regular grammars and stochastic automata to specify concepts over the alphabet {a, c, g, and t}, that is, to characterize DNA sequence families. In particular we wanted to characterize the concepts "donor site" and "splice site". We used the machine learning approach to infer candidate grammars to describe these concepts. To do so we used grammar inference algorithms.

In the present study, we developed implementations to four different grammatical inference algorithms: i) Lapfa (Ron et al., 1998); ii) Amnesia (Ron et al., 1996); iii) Alergia (Carrasco and Oncina, 1994); iv) RPNI (Oncina and Garcia, 1992). These algorithms were developed to be applied in the area of computational linguistics and, to the best of our knowledge, have not been applied to the gene prediction area. The input of these algorithms is a training sample representing a family of DNA sequences, and the output is a stochastic grammar **G** describing that family. From this grammar we generated a stochastic automaton that is a mechanism that receives sequences and assigns to them a value indicating the probability of those sequences belonging to the grammar, **P(sequence|G).**

The algorithms Lapfa, RPNI and Alergia work similarly. They initially build a *prefix tree* **T** representing all prefixes of the training set. Each path from the root of the tree to a leaf represents one example of the training set, where each edge of the tree is labeled by a letter of the word representing the example. To each internal node of the tree, we can assign a prefix of an example of the sample. To each edge of the tree, each algorithm also associates with a counter indicating the number of prefixes that used that path (Figure 1 shows an example of a prefix tree for the examples atga, atgg, atta, aaga, cgag). Each tree constitutes also an automaton, where each state corresponds to a node in the tree, the root note is the start state, and the edges are the same. Figure 1 has an example of *prefix tree*.

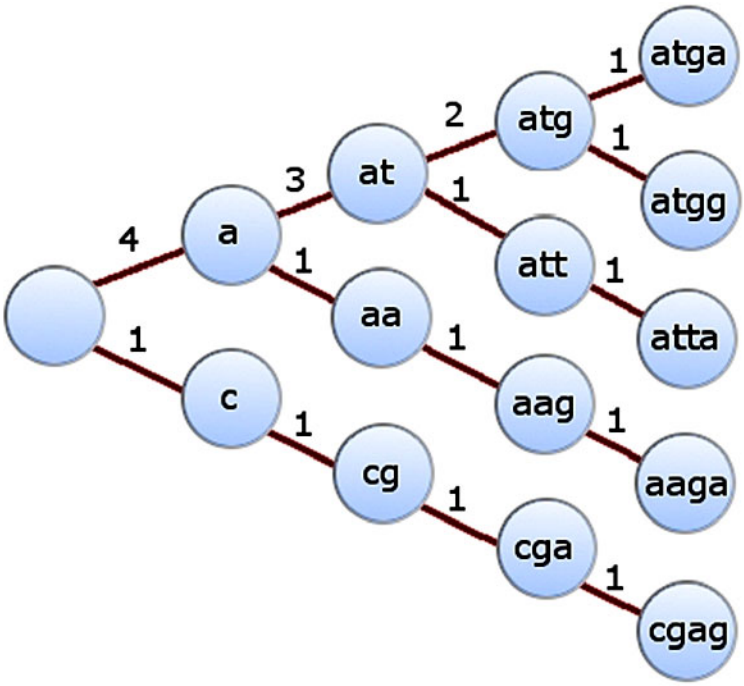

Figure 1. Prefix tree for the examples atga, atgg, atta, aaga, cgag.

In spite of the fact that the tree **T** corresponds to an automaton (and therefore a grammar) that represents all the elements of the training sample, it does not constitute a good hypothesis for our problem, since the tree describes *only* sequences that belong to the training sample or that constitute a prefix of the elements of the training sample. In other words, we have to face the problem of overfitting (Dietterich, 1995). To avoid this problem all three algorithms apply a *generalization process*, where similar states of the automata are joined together into a single-new state. The difference between the algorithms is the criteria for joining states, that is, the definition of "similar states". The algorithms Lapfa and Alergia have their generalization process regulated by one or more *generalization parameters.*

The only exception to the above description is the algorithm Amnesia. In Amnesia, the algorithm builds *suffix stochastic automata*, a sub-family of SFSA's. Here, instead of building

prefix trees, the algorithm initially builds a *suffix tree*. The building of the hypothesis uses a top down approach, starting with a single node and adding new children to a tree node if the sequences that will be represented by the new nodes fulfill some statistical conditions inferred from the training sample. After the inference process, the resulting automaton is converted into an SFSA. Amnesia is also controlled by a generalization parameter.

Generating a classifier

From a stochastic grammar, we can produce an automaton to recognize the same family of sequences. To generate a classifier from grammars, we can use two approaches. We can generate a single automaton from a positive sample and determine a threshold probability value for a sequence to be accepted as part of the family, or we can generate two grammars, one from a positive sample and one from a negative sample, and create a Bayesian classifier. In a Bayesian classifier, given a sequence, we calculate the probability that the sequence belongs to each of the two families, and decide on the family with the greatest probability. Thus, in the donor case, we will generate two grammars **Gdonor** and **Gfalse_donor**. The classifier will calculate **P(sequence|Gdonor)** and **P(sequence|Gfalse_donor)**, and recognize the sequence as a donor if **P(sequence|Gdonor) > P(sequence|Gfalse_donor)**. In the work described in this article, we decided on the second approach.

To generate, validate and test the classifiers, we used a system developed on the framework by Machado-Lima (2002). This framework provides facilities to run grammar inference algorithms from given samples, automatically generating classifiers for the given samples. On the top of this framework, a series of scripts were added to automatically run the classifier generation process for different samples and input parameters, running the resulting classifiers on the test samples, and recording the results.

Choosing generalization parameters

K-fold cross-validation

As we have seen above, three of the four inference algorithms can generate innumerous classifiers for the same sample, if different generalization parameters are chosen. To choose the set of generalization parameters used, we ran various experiments using the k-fold cross-validation technique (Blum et al., 1999) to evaluate the performance of each parameter set. K-fold cross-validation is a method widely used to validate learning algorithms, especially in the presence of limited training sets. In this method, we divide the training set into K parts of equal size. We then use K-1 of these as a training set and the remaining one as the “test set”. This process is repeated K times, one for each of the possible “test sets”. In our case, we used 10-fold validation, where the sample is divided into 10 sub-samples of the same size. The method is then repeated 10 times. At each time, one of the sub-sets is chosen as the “test sample” and the other nine are joined together to form the “training set”. After the 10 runs, we compute the average error rates. The advantage of this methodology is that it lowers the effect of how the data are partitioned on the final results. Each example of the set is used exactly once in the “test sample” and 9 times in the “training sample”.

Parameters tested

A Perl script was written to carry out the task of generating a classifier with given generalization parameters and performing k-fold validation of the results. The script receives as input the training dataset, the number k to be used in the k-fold validation (we used k = 10), the name of the algorithm to be tested, and a description of the intervals in which to test each generalization parameter. We used the following parameter intervals: i) Alergia, p1 = 0.001,0.002,...,0.009, and p1 = 0.01, 0.02,...,0.1; ii) Amnesia, p1 = 0.1,0.2,...,0.9; p2 = 2,3,4,5,6; p3 = 1,2; iii) Lapfa, p1 = 5; p2 = 0.1, 0.2,0.3,..,1.0; p3 = 0.01, 0.01,...,0.1 . Table 2 shows the number of configurations tested for each algorithm.

Table 2. Number of classifiers tested in the cross-validation test.

	Donor	Acceptor
Lapfa	1000	1000
Alergia	190	190
Amnesia	900	900
Total	2090	2090

The best performing classifiers were used

After generating classifiers to all combinations of generalization parameters specified above, we chose, for each algorithm, the parameters that had the lowest error rate on average for the 10-fold validation process. These parameters were then used to generate one classifier per algorithm, this time utilizing all the sequences of the training set of the benchmark, not only 9/10 of them, as we did in the 10-fold cross-validation process.

Evaluating results

Once we obtained one classifier for each inference algorithm, we used these classifiers on the testing samples (datasets splice.test-real.A, splice.test-false.A, splice.test-real.D, splice.test-false.D), and compared the results with the ones obtained by NNSPLICE using the same benchmark. For the donor and acceptor test benchmarks we computed the false-positive and false-negative error rates. The rates were computed using the formulas:

%False-positive = (number of false-positives)/(size of negative sample)

%False-negative = (number of false-negatives)/(size of positive sample)

The rates for NNSPLICE were obtained from the internet page http://www.fruitfly.org/seq_tools/spliceHelp.html.

RESULTS

Cross-validation results

Tables 3 and 4 show the two best generalization parameter sets for each inference

algorithm, obtained in the donor and acceptor training datasets using the 10-fold validation process. The algorithm that generated the classifier with the best results was Lapfa, with only 5.92% total average error rate for donor recognition and 6.38% average error rate for acceptor recognition. The performance of classifiers generated by the other algorithms was clearly inferior. The performance of Amnesia's best classifier had an average error rate of 24.09% for donor recognition and 40.33% for acceptor recognition. Alergia's best classifier had better performance with 10.56% average error rate for donor and 30.95% for acceptor error rate. Cross-validation was not used in RPNI, since there were no generalization parameters.

Table 3. Performance of the algorithms in the cross-validation test for donor classifiers.

Algorithm	Parameters			Error (standard deviation inside parentheses)		
	p1	p2	p3	% total	% false-positives	% false-negatives
Lapfa	5.000	0.600	0.006	5.519 (1.001)	3.890 (1.072)	11.557 (3.643)
	5.000	0.600	0.002	5.538 (0.932)	3.672 (1.096)	12.453 (3.773)
Alergia	0.020	-	-	9.839 (1.514)	4.615 (1.282)	29.202 (7.584)
	0.008	-	-	9.857 (1.608)	4.639 (0.467)	29.202 (6.780)
Amnesia	0.400	5.000	2.000	24.017 (1.996)	24.137 (2.095)	23.572 (5.207)
	0.700	6.000	2.000	24.017 (1.996)	24.137 (2.095)	23.572 (5.207)

Table 4. Performance of the algorithms in the cross-validation test for acceptor classifiers.

Algorithm	Parameters			Error (standard deviation inside parentheses)		
	p1	p2	p3	% total	% false-positives	% false-negatives
Lapfa	5.000	0.900	0.010	6.398 (0.643)	5.196 (0.636)	11.441 (3.467)
	5.000	1.000	0.010	6.398 (0.643)	5.196 (0.636)	11.441 (3.467)
Alergia	0.020	-	-	28.051 (2.103)	18.270 (3.085)	69.099 (4.630)
	0.010	-	-	25.594 (2.452)	19.365 (2.858)	72.523 (5.884)
Amnesia	0.400	5.000	2.000	40.794 (4.3451)	40.898 (6.935)	40.360 (8.478)
	0.700	6.000	2.000	40.794 (4.3451)	40.898 (6.935)	40.360 (8.478)

Results of the best classifiers

Tables 5 and 6 show the performance of the grammar-based classifiers selected above on the validation dataset used by NNSPLICE, comparing the results with the ones given at NNSPLICE's internet site. As it would be expected from the 10-fold validation step, the best performing algorithm was Lapfa, with only 5.86% error rate for donor recognition and 6.70% for acceptor recognition. The second best performance was by the algorithm generated by Alergia, with 12.63% error rate for donor recognition and 27.64% error rate for acceptor recognition. Amnesia's error rates were 22.32% for donor and 32.32% for acceptor. Finally, RPNI

had the worst performance with 23.47% (donor) and 32.32% (acceptor) error rates. The results obtained by the classifier generated using Lapfa were comparable to the best results obtained by NNSPLICE, that is 5.20% error rate for donor and 7.87% error rate for acceptor. The difference was that NNSPLICE's parameters were adjusted during the validation step.

Table 5. Performance of the algorithms in donor identification.

Algorithms	Parameters			Error		
	p1	p2	p3	% total	% false-positives	% false-negatives
Lapfa	5	0.6	0.006	5.657	3.836	12.500
Alergia	0.02	-	-	10.910	4.730	34.130
Amnesia	0.40	5	2	22.320	21.995	23.558
RPNI	-	-	-	23.470	23.920	22.440
NNSPLICE	threshold = 0.40			5.20	6.80	5.54

Table 6. Performance of the algorithms in acceptor identification.

Algorithms	Parameters			Error		
	p1	p2	p3	% total	% false-positives	% false-negatives
Lapfa	5	0.9	0.01	5.326	4.654	8.173
Alergia	0.02	-	-	29.844	1.8422	76.442
Amnesia	0.4	5	2	37.370	34.050	51.440
RPNI	-	-	-	31.460	28.110	45.500
NNSPLICE	threshold = 0.40			7.87	3.00	26.20

DISCUSSION

In the present study, we have introduced the use of SRGs and also of grammar inference, to the problem of splice site prediction. Four different grammatical inference algorithms were used to produce 4 different splice site predictors. The performance of three of the predictors was not satisfactory, with average error rates ranging from 9.839% up to 40.764%. However, the fourth predictor, the one generated by the Lapfa algorithm, produced good results with an error rate of 5.657% for donor recognition and 5.326% for acceptor recognition. The better performance of the Lapfa algorithm can be explained by the fact that it is the only algorithm that does not produce "cycles" in the grammar. That means that the resulting grammar recognizes sequences with limited size (in our case fixed size). All the other three algorithms assume in the generalization process that the "language" being inferred can have some repetitive "stretches". The higher rate of failure can then be explained by overgeneralization over the larger sample that is the negative sample.

The results of this splice site predictor were very close to those of NNSPLICE, being more accurate for acceptor sites (5.326 *vs* 7.87%) and slightly less accurate for donor sites

(5.657 *vs* 5.20%). One advantage of our approach was that the parameters for these results were automatically set from the training sample using the benchmark, while the results for NNSPLICE were hand-tuned based on the results of the testing set.

The use of the k-fold validation technique, the use of a framework for inference of grammar-based classifiers, and the development of scripts for automating the process of generating and validating classifiers with various values for up to three generalization parameters, enabled us to generate 1465 different classifiers using 4 different inference algorithms, and to select the best one associated with each algorithm. The whole process consumed about 5 days of a Pentium-based Linux workstation and could now be repeated for a larger coverage of parameter possibilities.

For three of the algorithms, we generated Bayesian classifiers, generating characterizations of both "true-splice sites" and "false-splice sites" (DNA sequences with the consensus di-nucleotides, but that did not correspond to true-splice sites). However, the probability value generated by the classifiers can also be used to control the number of false-positives. That is, if we set a threshold value, we may be able to eliminate most of the wrongly predicted sites. The main candidate for this approach is the algorithm RPNI, where, since the algorithm uses a positive- and a false-training set, only one grammar is generated. The threshold value can also be used with any of the other 3 algorithms, if we change the approach from using a Bayesian classifier to one using a single grammar and the threshold value. Sakakibara et al. (1994) developed a technique to calculate such thresholds in the presence of large negative samples, and we are currently implementing the programs to test this alternative approach.

CONCLUSIONS

A new promising approach was applied to generate splice site predictors. This *ab initio* approach can also be applied to other signals like promoters. The current results are encouraging, matching those of NNSPLICE, the splice site predictor of the gene finder Genie. We have also extended a classifier generation framework to enable massive generation of candidate classifiers and their selection using k-fold validation. Using this environment, we intend to extend our work by also introducing the calculation of minimal thresholds using the windowing technique, and to extend the comparison to other splice site predictors. In the future, we intend to integrate the best performing splice site predictors into a gene finder.

ACKNOWLEDGMENTS

We would like to acknowledge CNPq for partially financing the research. We would also like to acknowledge Ariane Machado-Lima for the use of the framework developed as part of her Master's thesis.

REFERENCES

Abe N and Warmuth MK (1992). On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning* 9: 205-260.

Aho A, Sethi J and Ullman J (1986). Compilers: principles, techniques, and tools. Addison-Wesley Longman Publishing Co., Inc., Boston.

Baldi P and Brunak S (1998). Bioinformatics: the machine learning approach. MIT Press, Cambridge.

Blum A, Kalai A and Langford J (1999). Beating the hold-out: bounds for k-fold and progressive cross-validation. Proceedings of the Twelfth Annual Conference on Computational Learning Theory, July 7-9. ACM Press, Santa Cruz, 203-208.

Brunak S, Engelbrecht J and Knudsen S (1991). Prediction of human mRNA donor and acceptor sites from the DNA sequence. *J. Mol. Biol.* 220: 49-65.

Burge C and Karlin S (1997). Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.* 268: 78-94.

Carrasco RC and Oncina J (1994). Learning stochastic regular grammars by means of a state merging method. Proceedings of the Second International Colloquium. September 21-23. ICG, Alicante, 139-152.

Dietterich T (1995). Overfitting and undercomputing in machine learning. *ACM Comput. Surv.* 27: 326-327.

Eddy SR and Durbin R (1994). RNA sequence analysis using covariance models. *Nucleic Acids Res.* 22: 2079-2088.

Guigo R, Knudsen S, Drake N, Smith T (1992). Prediction of gene structure. *J. Mol. Biol.* 226: 141-157.

Hubbard T, Birney E, Bruskiewich R, Clamp M, et al. (1999). Abstracts of papers presented at the 1999 meeting on genome sequencing and biology. Cold Spring Harbor Laboratory Press, Cold Spring Harbor.

Hughey R and Krogh A (1996). Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Comput. Appl. Biosci.* 12: 95-107.

Krogh A (1997). Two methods for improving performance of a HMM and their application for gene finding. Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology. June 21-26. AAAI Press, Halkidiki, 179-186.

Krogh A and Vedelsby J (1995). Neural network ensembles, cross validation, and active learning. In: Advances in neural information processing systems (Tesauro G, Touretzky DS and Leen TK, eds.). The MIT Press, Cambridge, 231-238.

Machado-Lima A (2002). Laboratório de geração de classificadores de seqüências. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, USP, São Paulo.

Mathe C, Sagot MF, Schiex T and Rouze P (2002). Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Res.* 30: 4103-4117.

Murthy SK, Kasif S and Salzberg S (1994). A system for induction of oblique decision trees. *J. Artificial Intelligence Res.* 2: 1-32.

Oncina J and Garcia P (1992). Inferring regular languages in polynomial update time. In: Pattern recognition and image analysis (de la Blanca NP, Sanfeliu A and Vidal E, eds.). World Scientific Publishing, Singapore, 49-61.

Reese MG, Eeckman FH, Kulp D and Haussler D (1997). Improved splice site detection in Genie. *J. Comput. Biol.* 4: 311-323.

Ron D, Singer Y and Tishby N (1996). The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning* 25: 117-150.

Ron D, Singer Y and Tishby N (1998). On the learnability and usage of acyclic probabilistic finite automata. *J. Comput. Syst. Sci.* 56: 133-152.

Sakakibara Y, Brown M, Underwood RC, Mian IS, et al. (1994). Stochastic context-free grammars for modeling RNA. Proceedings of the 27th Annual Hawaii International Conference on System Sciences (Hunter L), January 4-7. IEEE Computer Society Press, Honolulu, 284-293.

Salamov AA and Solovyev VV (2000). *Ab initio* gene finding in *Drosophila* genomic DNA. *Genome Res.* 10: 516-522.

Staden R (1984). Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res.* 12: 505-519.

Uberbacher EC and Mural RJ (1991). Locating protein-coding regions in human DNA sequences by a multiple sensor-neural network approach. *Proc. Natl. Acad. Sci. USA* 88: 11261-11265.

Zhang MQ and Marr TG (1993). A weight array method for splicing signal analysis. *Comput. Appl. Biosci.* 9: 499-509.

Zhang L and Luo L (2003). Splice site prediction with quadratic discriminant analysis using diversity measure. *Nucleic Acids Res.* 31: 6214-6220.