Int. J. Appl. Math. Comput. Sci., 2012, Vol. 22, No. 3, 711–721
DOI: 10.2478/v10006-012-0053-2

KIS: AN AUTOMATED ATTRIBUTE INDUCTION METHOD FOR CLASSIFICATION OF DNA SEQUENCES

RAFAŁ BIEDRZYCKI, JAROSŁAW ARABAS

Institute of Electronic Systems
Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: rbiedrzy@elka.pw.edu.pl

This paper presents an application of methods from the machine learning domain to solving the task of DNA sequence recognition. We present an algorithm that learns to recognize groups of DNA sequences sharing common features such as sequence functionality. We demonstrate application of the algorithm to find splice sites, i.e., to properly detect donor and acceptor sequences. We compare the results with those of reference methods that have been designed and tuned to detect splice sites. We also show how to use the algorithm to find a human readable model of the IRE (Iron-Responsive Element) and to find IRE sequences. The method, although universal, yields results which are of quality comparable to those obtained by reference methods. In contrast to reference methods, this approach uses models that operate on sequence patterns, which facilitates interpretation of the results by humans.

Keywords: classification, optimization, annotation, patterns.

1. Introduction

Thanks to automated high-throughput DNA sequencing technologies, an enormous amount of DNA sequences is now available in public databases. Computational methods for their analysis are, however, still under development. In this paper we discuss the DNA sequence annotation problem. It consists in predicting the functionality of a DNA fragment after analyzing its nucleotide sequence.

Currently, the most popular annotation methods use human expertise and are based on experimental evidence of different kinds (Elsik *et al.*, 2006). Libraries of such evidences exist that are used by volunteers to provide their annotation suggestions. These suggestions are later accepted or rejected by experts—*curators* of the annotation database. This approach is quite slow and it is now the bottleneck in genetics research.

To speed up the process of annotation, several machine learning techniques are used. In the supervised learning process the set of labeled examples (annotated sequences) is used to create a generalized representation of interesting DNA sites, assuming that certain types of similarity in the primary sequence structure will correlate with similar functions in the tested sequences. Therefore, to obtain good generalization results, it is necessary to correctly choose similarity patterns that represent groups of similar sequences. Below we briefly overview several approaches to the representation of similarities.

Similarity patterns can be represented by stochastic grammars. This approach is shared by algorithms such as Alergia (Carrasco and Oncina, 1994), Lapfa (Ron *et al.*, 1998) and RPNI (Oncina and Garcia, 1992). Algorithms from this group start their work by building an automaton that represents all sequences from the training set. Such an automaton is usually unable to recognize any sequences that were not included in the training set and therefore some generalization process is required. This process is performed by connecting “similar” states of the automaton. The similarity definition is arbitrary and specific for each algorithm from the group considered. After completion of the connection phase, the resulting automaton can recognize sequences belonging to the language induced from the training sequences. A similar approach is assumed in the Amnesia algorithm (Ron *et al.*, 1996), which uses a tree-based representation of the automaton.

Another approach to representing similarity patterns is based on introducing a set of random variables as a model of a sequence. In the DGSplicer algorithm (Chen *et al.*, 2005), a Bayesian network is used to represent a set of sequences. It is assumed that the interesting signal is position-dependent. For each pair of positions their dependency is tested with the use of the χ^2 statistics. Subsequently, a weighted graph is created in which every position is represented by a vertex and the dependence between positions is represented as an edge. Such a graph is then converted into a Bayesian network that allows the category for any sequence to be inferred. Another approach from that group is represented by a k-th order Markov model (Durbin *et al.*, 1998). This model is applied to define distributions of all random variables assigned for each position in the DNA strand. This means that for a position p, the probability distribution of its random variable depends solely on probability distributions assigned for positions ranging from $p-1$ up to $p-k$. The higher the order of the model, the better its ability to describe the analyzed sequences. However, the number of parameters to be estimated grows with k, which complicates computations and increases the risk of erroneous generalization.

Yet another way to represent similarity patterns is to use a set of attribute functions that translate sequences into vectors of attribute values. These vectors are used as an input for a classifier. An example application of that idea is the NNSplice algorithm (Reese *et al.*, 1997), where a neural network (a multilayer perceptron) is used as a classifier of sequences of the length k. In the NNSplice, all $k-1$ pairs of neighboring nucleotides are considered. Each such pair is assigned 16 binary attributes which represent all possible dinucleotide pairs. The attribute which represents the observed pair is set to 1, whereas all the others are set to 0. In the case of a donor, NNSplice analyzes sequences of 15 nucleotides, which gives 14 pairs of nucleotides and yields a binary input vector of 224 bits. In the NNSplice implementation that comes together with the Genie system, a neural network with 2 hidden neurons is used for donor classification, and 10 hidden neurons are used to perform acceptor classification.

An alternative way to perform classification without introducing any attributes is to use a classifier based on the Support Vector Machine (SVM). In this approach it is only needed to specify a kernel function which is a generalization of the distance function between the DNA sequences. Among other possible approaches we would like to mention those of Rätsch and Sonnenburg (2004) as well as Sonnenburg (2009) where the concept of a Weighted Degree (WD) is used as a kernel function. The main idea behind the WD is counting co-occurrences of subsequences of some length in both compared sequences. The authors claim that the WD kernel works well if the signal is highly position-specific. They also propose a WDS (Weighted Degree with Shifts) kernel that should be used to find signals that are not strongly position-related.

Deshpande and Karypis (2002) observed that first order Hidden Markov Models (HMMs) are classifiers which have been built upon the space of all possible dinucleotides. Thus for a standard classifier they propose an approach similar to the HMM. They acquire a transition probability matrix and change the sequence into a vector of dinucleotides. Every dinucleotide is perceived as an attribute. The attribute value is calculated as the product of the number of occurrences of the dinucleotide in the sequence and the transition probability connected with that dinucleotide. The authors used the SVM classifier with the linear kernel. A similar approach, called MC-SVM, has been proposed by Baten *et al.* (2006), the only difference being that the kernel function was a second-degree polynomial.

In this paper we present a novel approach to classification of DNA examples. We combine a tree-based classifier inspired by the ID3 algorithm (Quinlan, 1986) with problem-specific attributes to characterize sequences. In contrast to the classifier-based approaches overviewed earlier in the text, we introduce a methodology which allows attributes to be automatically defined. These automatically defined attributes are those which are best-suited to perform classification, and the attribute definition process is invoked at each step of the classifier building process. Our approach fits into the research on *attribute induction* in data mining and is independent of the classifier type, so it can be easily adopted not only for decision trees, but also for neural nets, SVMs, etc.

We consider binary attributes defined with the use of *patterns* which are in turn defined by regular expressions. Given a specific sequence, an attribute returns the value 1 if the sequence matches the pattern assigned to the attribute, otherwise the attribute returns 0.

Searching for appropriate attributes is perceived as a process of optimization in the space of all possible patterns. In this space we introduced a neighborhood relation which is based on the generality of patterns, and an objective function that attempts to predict the quality of an attribute.

The paper is composed in the following way. In Section 2 we introduce the type of the examined annotation problems mentioned earlier. Section 3 describes the KIS method and Section 4 presents results of experiments using datasets representing various annotation problems. The paper is concluded in Section 5.

2. DNA annotation problems

A popular example of DNA sequence annotation is the task of finding coding information contained in genes. This task normally consists in finding splice sites and connecting them to achieve the most probable division of a gene into coding and non-coding subsequences. Less popular but still important annotation problems exploit the information of some aspects of the spatial structure that will be formed for the sequence considered. An example of that kind of problems is the IRE finding task.

2.1. Finding splice sites in the DNA chain. In eukaryotic cells the DNA sequence is composed of regions (subsequences) that are responsible for information coding (*exons*) and those which are not (*introns*). In the protein synthesis process, a pre-mRNA sequence is generated by rewriting the DNA sequence. After that an mRNA is created by splicing, which is the process of removing introns. In the predominant splicing mechanism (U2) the process of intron excision is done by a spliceosome (Baten *et al.*, 2006). The spliceosome is able to find *splice sites* thanks to the information encoded in relatively short subsequences (*signals*) which are contained in introns close to their borders with exons. In the case of the exon/intron border the signal is called a *donor*, and in the case of the intron/exon border it is called an *acceptor*. In the final step of the protein synthesis the contents of exons (mRNA) are translated by triplets into amino acids. In certain cases some exons could be excised rather than translated. That mechanism is called alternative splicing (Berget *et al.*, 1977), and it allows production of different proteins from one DNA sequence, which makes the process of gene finding even more difficult.

Detection of donors and acceptors has been attempted by several existing algorithms; however, we are interested only in those that are able to learn. According to our knowledge all splice site detection algorithms are unable to work directly on a complete DNA chain. A common approach to overcome this limitation is to apply the *windowing* technique to create a set of donor or acceptor candidates. In this technique, the DNA sequence is scanned from the beginning position to the end position. During the scan, characteristic dinucleotides are located ('GT' for donors and 'AG' for acceptors). For each position of the chain where a characteristic dinucleotide has been found, a *window* is defined which spans n nucleotides towards the 5' end of the sequence and m nucleotides towards the 3' end. In the remainder of the text, each sequence defined by the windowing technique is called an *example*. An example is considered *positive* if it is a true donor/acceptor sequence, otherwise it is regarded as *negative*. Values of the windowing parameters (n and m) depend on the splice site type (donor or acceptor), and there is no common agreement on their settings. In various approaches, different values have been used to achieve maximum annotation quality (Sonnenburg *et al.*, 2007).

To apply learning to perform annotation, an initial set of examples is needed (a training set). This set is produced as a result of two steps. In the first step, examples are generated by application of the windowing technique to the annotated sequence. In the second step, a database of annotations is used to classify examples. Every example marked in the database as a true donor/acceptor is considered a positive example and all the others a negative one.

2.2. Finding the iron-responsive element. The Iron-Responsive Element (IRE) is the fragment of a non-coding RNA that is a binding place for proteins in the process of regulation of iron metabolism. This RNA fragment binds to itself to form a stem-loop spatial structure. The IRE model which has been introduced by Pesole *et al.* (2000) is presented in Fig. 1. In this model each circle represents a nucleotide which may be additionally specified as a single value or as a set of admissible values (listed in square brackets). Dashed lines represent complementary bonds and solid lines connect neighboring nucleotides in the sequence. Complementarity means that an Adenine (A) forms a pair with Thymine (T) and Guanine (G) forms a pair with Cytosine (C). In the RNA spatial structures it is also possible to observe weaker bonds to form G-T base pairs.

3. KIS method

In this section we define building blocks of the KIS methodology: a classifier, a pattern based attribute and an optimization problem which arises when considering the space of all possible patterns.

3.1. Classifier. Consider a number of binary attributes being functions of a form $a_i : S \to B$, where S is the set of all possible examples (DNA sequences) and $B = \{0, 1\}$. A classification function $c : S \to B$ assigns a class label to each example. A classifier is a function $\kappa : B^n \to B$ which assigns a label to each example using values of n attributes characterizing the example being considered. The classifier is used to define an approximation $\hat{c}$ to the classification function c according to the rule $\hat{c}(s) = \kappa(a_1(s), \ldots, a_n(s))$.

In the presented solution we use a classifier based on a decision tree. In general, a tree is defined as a connected graph consisting of a set of nodes V and edges $E \in V \times V$. For each pair of nodes in a tree there exists exactly one sequence of edges that link these nodes. Decision trees are directed ones. If nodes $v_1 \in V$ and $v_2 \in V$ are linked and the link is directed towards v_2, then v_1 is called a *parent* of v_2, and v_2 is a *child* of v_1. Nodes with at least one child are called *internal nodes*, and nodes with no children are called *leaves* of the tree. There exists a unique node $v_0 \in V$ which has no parent and it is called the tree *root*.

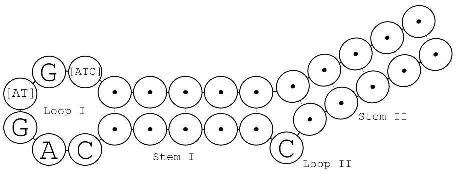

Fig. 1. IRE model.

In the decision tree each internal node v has a *test* $t_v(s)$ assigned to it. The test t_v is a function $t_v : S \to V$ that selects a child node for a tested sequence s. We consider tests based on single attributes which have the form

$$t_v(s) = \begin{cases} v_1 & \text{if } a_v(s) = 0, \\ v_2 & \text{if } a_v(s) = 1, \end{cases} \tag{1}$$

where a_v is an attribute that has been assigned to the node v. Each node v of the decision tree has a label assigned to it according to a labeling function $l : V \to B$. When a sequence s is to be classified with the decision tree, it is repeatedly examined by test functions starting from the tree root. A *decision path* from the root v_0 to a leaf v_k is defined as a sequence $v_0, v_1, \ldots, v_{k-1}, v_k$ where for each $i = 0, \ldots, k-1$ it holds that $v_{i+1} = t_{v_i}(s)$. The value of the classification function $c(s)$ is approximated by the value $l(v_k)$ of the labeling function l assigned to the terminal node v_k which has been reached for the sequence s.

When a set of examples $T \subseteq S$ is considered, we denote by $T(v) \subseteq T$ the set of examples from T that is the result of the application of a sequence of tests associated with the decision path from v_0 to v. A set of examples from $T(v)$ that has the class d will be denoted by $T(v,d), d \in B$. The labeling function l which is based on the set of examples T is defined as $l(v) = \arg\max_{d \in B} |T(v,d)|$. If the numbers of examples with both decision classes are equal, then the labeling function returns the default class value d_0. We also define the misclassification function of a labeling function l which is given by $e(l(v)) = |\{s \in T(v), l(v) \neq c(s)\}|$.

The tree building algorithm, which is based on the famous ID3 approach (Quinlan, 1986), is outlined in Fig. 2. The algorithm assumes a set of training examples $T \subset S$, a default category d_0 and an initial set of attributes A which have to be defined prior to training. In the beginning the decision tree contains only a root node and then a growing procedure is iterated. In each iteration a leaf v is selected with a nonzero misclassification function value. For that leaf an attribute is selected to maximize the quality of the test which is based on that attribute. In this paper we express the test quality as the *information gain* according to the following methodology.

```
build_tree( $T, d_0, A$ )
begin
    $v$ := create_node();
    $v.c$ := choose_category( $T, d_0$ );
    if stop_criterion( $T, A$ ) then
        return $v$;
    $v.a$ := choose_attribute( $T, A$ );
    $A := A - \{v.a\}$;
    for each $d \in B$ do
    begin
        $A_d$ := find_attributes( $T(v)$ );
        $v.v[d]$ := build_tree( $T(v), v.c, A \cup A_d$ );
    end
    return $v$;
end
```

Fig. 2. Pseudocode of the tree-building algorithm.

We define the *information content* of the set of examples $T(v)$:

$$I(T(v)) = \sum_{d \in B} -\frac{|T(v,d)|}{|T(v)|} \log \frac{|T(v,d)|}{|T(v)|}, \tag{2}$$

and the *information gain* of the attribute a:

$$g(a, T(v)) = I(T(v)) - E(a, T(v)), \tag{3}$$

where $E(a, T(v))$ is the entropy of the attribute a on the data set $T(v)$:

$$E(a, T(v)) = \sum_{r \in B} \frac{|T(v,a,r)|}{|T(v)|} E(a, r, T(v)), \tag{4}$$

and $E(a, r, T(v))$ is a conditional entropy of a value r of the attribute a which reads

$$E(a, r, T(v)) = -\sum_{d \in B} \frac{|T(v,a,r,d)|}{|T(v,a,r)|} \log_2 \frac{|T(v,a,r,d)|}{|T(v,a,r)|}, \tag{5}$$

where

$$\begin{aligned} T(v,a,r) &= \{s \in T(v) : a(s) = r\}, \\ T(v,a,r,d) &= \{s \in T(v) : a(s) = r, c(s) = d\}. \end{aligned}$$

In the node expansion process an attribute with the highest information gain is selected. The attribute is selected from the set which contains the initial set of attributes A and the attributes which have been dynamically inferred each time a node has been expanded (see the `find_attributes` method in the algorithm's outline). The attribute construction process will be described in the next section.

The decision tree is generated with the use of the training set T. After the tree has been built it may undergo *pruning* to reduce the risk of overfitting to the training set (Quinlan, 1993). Pruning is performed with the use of a pruning set $Tp \subset S$, i.e., a set of examples which is independent of the training set. For each internal node v, we compare the values of the misclassification error $e_1 = |\{s \in Tp(v), l(v) \neq c(s)\}|$ and $e_2 = |\{s \in Tp(v), l(t_v(s)) \neq c(s)\}|$. When $e_1 \leq e_2$, we transform the node v into a leaf.

3.2. Definition of attributes.

In the approach presented we consider binary attributes defined by *similarity patterns* which are represented by regular expressions.

An attribute a for a sequence s takes the value 1 when the sequence matches the attribute's pattern $p(a)$, i.e., there exists a subsequence $s' \in s$ which is conformant with the pattern $p(a)$.

The similarity pattern is a sequence of symbols and it is based on the traditional Unix regular expressions (regexps) syntax. We enriched the syntax by introducing special symbols, e.g., a *positioning sequence*, to classify sequences with a characteristic reference position, such as, a marked intron/exon boundary. In this way we can construct patterns that will match the classified sequence only if the positioning sequence is matched to a specific side of the reference position. Symbols that can be used to define similarity patterns are listed below.

A, C, G, T a nucleotide symbol; each one matches itself in the classified sequence.

[ ] an *ambiguous symbol*; these brackets enclose a group of nucleotide symbols. The match is observed when one of enclosed symbols matches itself in the classified sequence.

. a *gap* which matches any single nucleotide.

.{x,y} a *flexible gap* which matches any sequence composed of x up to y nucleotides, x and y being positive integers ($y \geq x$). The value of the difference $y - x$ will be called the *flexibility* of a gap.

<s> s is a *positioning sequence*.

ˆs(d) d is the distance from the beginning of the sequence s to the *positioning sequence*.

s(d)$ d is the distance from the end of sequence s to the *positioning sequence*.

R_MATCH a *positioning sequence* which is an argument of the symbol should match the substring on the right hand side of the reference position.

L_MATCH a *positioning sequence* which is an argument of the symbol should match the substring on the left hand side of the reference position.

RPM>r the number of the pattern matches on the right hand side of the reference position should be greater than the product of r and sequence length.

LPM>r the number of the pattern matches on the left hand side of the reference position should be greater than the product of r and sequence length.

L-(α)γ-L a symbol of a stem-and-loop which is composed of a stem sequence α and a loop sequence γ; thus the sequence has a structure $\alpha\gamma\beta$, where β is the reverted version of the sequence that is complementary to α.

Example patterns and sequences to which they match are presented in Table 1 (the reference position is marked by |).

There are several types of information available in DNA sequences. To exploit them, we introduce three types of attributes:

- Positioning attributes: an attribute has value 1 if its pattern is matched at a specified position of the classified sequence.
- Existence attributes: an attribute has value 1 if a pattern is matched at least once to the classified sequence.
- Frequency attributes: an attribute has value 1 if the ratio of the number of possible pattern matches and the sequence length exceeds a certain threshold.

The last two types of attributes (existence and frequency) could use the *positioning sequence* to require a match at a specified side of the reference position.

3.3. Space of patterns.

The attribute generation process is interpreted as a search task in the space Π of all possible patterns which is organized with the neighborhood relation. For each pattern $p \in \Pi$, a neighborhood $N(p) \subseteq \Pi$ is defined as a set of all patterns that can be obtained from p by applying a single *Elementary Edit Operation* (EEO). During the development of the method we have investigated several possible definitions of EEOs. For the splice site recognition task we obtained the best results using the following set of EEOs:

- operations that generalize:
    1. decreasing the lower limit of the flexible gap length,
    2. increasing the upper limit of the flexible gap length,

Table 1. Example patterns and sequences to which they match.

Pattern	Example sequences matched
AG	GAGTAG
A.G	ATG, ACG
A.{1,3}G	ATG, ACG, ATTG, ATATG
A[CT]G	ATG, ACG
ˆx(2)A<G>	TAGC, CAGC, AAGA
<A>Gx(2)$	TAGC, CAGC, AAGA
L-(..)CC-L	AGCCCT, TACCTA, CTCCAG
<A>G R_MATCH	TAC|AGC, TAC|ACCGGGCAG
<A>G L_MATCH	TAGC|TAGC, TAGCTTGG|TAC
<A>G RPM>0.49	TAGC|AGAGAGAGAGAGAGAG
<A>G LPM>0.25	AGTTAGAGTTT|TAGC

3. adding another letter to the set of letters that can match the position (which results in transforming a single letter into a 2-letter ambiguous symbol, a 2-letter into a 3-letter ambiguous symbol and a 3-letter ambiguous symbol into a gap),
4. changing an L symbol to a flexible gap;

- operations that specialize:
  1. increasing the lower limit of the flexible gap length,
  2. decreasing the upper limit of the flexible gap length,
  3. taking one letter from the set of letters that can match the position,
  4. adding one 3-letter ambiguous symbol at the beginning or at the end of the pattern,
  5. operates as in 3 but the ambiguous symbol is added after the flexible gap,
  6. adding an L symbol L-(α)γ-L, where α and γ are flexible gaps.

3.4. Scoring function. The search process in the space of patterns is driven by the pattern scoring function $\Pi \to \mathbb{R}_+$. The function definition originates in the concept of information entropy that is used to select attributes in the decision tree building process and reads

$$q(p,T) = \frac{2^{g(a(p),T)} - 1}{(1+\rho(p))^{\varrho}}, \tag{6}$$

where $a(p)$ is an attribute based on the pattern p, $g(a(p),T)$ is the information gain of the test based on the attribute $a(p)$ for the set of examples T (Eqn. (3)), $\rho(p)$ is a penalty function for the pattern p and $\varrho \in [0,1]$ is a parameter that controls the influence of the penalty on the overall evaluation of the pattern quality. The penalty ρ function is defined as

$$\rho(p) = \varsigma \cdot \eta(p) + \mu \cdot f(p), \tag{7}$$

where η is a pattern length, f is the sum of flexibility values of all used gaps and ς, μ are program parameters. The program parameters can express the actual computational costs (which rise with the pattern flexibility) or the user preferences, e.g., building the longest possible patterns with the smallest flexibility.

3.5. Search method. We create patterns by searching the space Π in order to find a pattern with the highest possible scoring function value. We tested three different search techniques including the Evolutionary Algorithm (EA), Monte Carlo search and greedy search. We found that the best performance was attained by the EA (see the work of Michalewicz (1996) for a comprehensive description of EAs), therefore we provide detailed information on the EA version used in the KIS method. The pseudocode of the EA version implemented is depicted in Fig. 3.

The EA maintains populations P_c of similarity patterns which contain M elements. The algorithm starts with an initial population P_0 and then loops the following steps. A temporary population O_c is created from P_c by mutating all patterns from P_c. Mutation is defined as a transition to a neighboring pattern $p' \in N(p)$ according to the neighborhood definition implied by EEOs. Generation of the neighboring pattern is a two-step process. The first step is to decide which group of EEOs—specification or generalization—is to be used for mutation. We assumed that EEOs that specify will be selected with probability 0.33, whereas those which generalize will be selected with probability 0.67. Then a single pattern is generated by randomly selecting a pattern from an appropriate group (either more general or more specific) with uniform probability distribution.

The best pattern from the population P_c is inserted unchanged into P_{c+1}. All of the remaining $M-1$ patterns which are input into P_{c+1} are generated as a result of performing the following two steps: (i) selection with replacement of a pair of patterns from $O_c \cup P_c$, (ii) choice of a pattern with a higher score in the selected pair. After completion of the main loop, the EA returns the set of patterns with the best scoring function value.

The initial population is filled with simple starting patterns of one nucleotide length. We assumed the population size $M = 20$ and that the main loop turns 500 times, which results in evaluating up to 10,020 similarity patterns. The actual number of evaluated patterns is

```
EA($P_0, n$)
begin
    $R := \emptyset$;
    $c := 0$;
    while $c < n$
    begin
        $O_c :=$ mutation($P_c$);
        evaluation($O_c$);
        $R := R \cup$ the_best_from($O_c$);
        $P_{c+1} :=$ selection($P_c \cup O_c$);
        $c := c + 1$;
    end
    return $R$;
end
```

Fig. 3. Pseudocode of the EA algorithm.

smaller thanks to the use of an associative memory where all generated patterns are kept together with their scoring function values. Before computing the scoring function value for a newly generated pattern, we search for an identical pattern in the associative memory and use the memorized scoring function value if available. Thus we speed up computation by overcoming a well known drawback of the EA of returning to previously visited points (see Fig. 5).

4. Results and discussion

In this section we report the results of KIS obtained for popular sets of real data: DGSplicer (Chen *et al.*, 2005) and NN269 (Reese *et al.*, 1997). These datasets provide separate test sets, which allows for comparison with the reported results for several up-to-date annotation algorithms, including Alergia (Carrasco and Oncina, 1994), Amnesia (Ron *et al.*, 1996), Lapfa (Ron *et al.*, 1998), RPNI (Oncina and Garcia, 1992), NNSplice (Reese *et al.*, 1997), MC (Durbin *et al.*, 1998), MC-SVM (Baten *et al.*, 2006), WD (Rätsch and Sonnenburg, 2004), WDS (Rätsch *et al.*, 2005) and DGSplicer (Chen *et al.*, 2005). Both datasets used for comparison assume binary classification, i.e., each example is either a positive or negative. This allows a classifier to be evaluated by comparing each example class to the classifier's output. Thus the set of all examples T is divided into four disjoint sets which are true positive (TP), false positive (FP), true negative (TN) and false negative (FN). These sets are used to define the following quality measures (see the work of Davis and Goadrich (2006) for a comprehensive introduction):

- accuracy: $acc = |TP \cup TN|/|T|$,
- true positive ratio (recall): $TPR = |TP|/|TP \cup FN|$,
- false positive ratio: $FPR = |FP|/|TN \cup FP|$,
- false negative ratio: $FNR = |FN|/|TP \cup FN|$.

Accuracy and recall should be maximized while FPR and FNR should be minimized.

A decision tree is a classifier which directly classifies each example. There is, however, a group of classifiers which assign to each example a certain value of the probability that the example is positive. In such cases the user has to provide a threshold value which has to be exceeded by the aforementioned probability value to regard an example as positive. The proper choice of the threshold value can be made by using curves of Receiver Operating Characteristic (ROC), where for each threshold value we plot a pair of FPR and TPR values. This curve can be used to characterize threshold-based classifiers; in particular, the area under the curve (auROC) can be used as a quality measure which has to be maximized.

4.1. DGSplicer data.

The DGSplicer datasets have been created for the purpose of testing the DGSplicer algorithm (Chen *et al.*, 2005) and they can be downloaded from `www.fml.tuebingen.mpg.de/raetsch/suppl/splice`. There are separate datasets for donors and acceptors, and each dataset contains a pair of separate files with training and testing sets.

The data for these files were extracted from 462 annotated multiple exon human genes. A basic characteristic of these datasets is provided in Table 2. As we can see the datasets are large and examples contained in the datasets are mostly negative.

In Table 3 we report the comparison of results for the DGSplicer datasets which have been obtained by KIS and by other methods reported by Sonnenburg *et al.* (2007). They provided results for the best model that was selected on a subset of training examples. For KIS, since the method is stochastic, we report not only the best result but also the average and the standard deviation of results obtained in 10 independent runs of the algorithm. Unfortunately, Sonnenburg *et al.* (2007) provided only areas under curves (auROC and auPRC) which cannot be directly obtained for the binary decision tree. Therefore, for the purpose of comparison, we adopted our classifier to provide probabilities for each classification. We assumed that for each node v the probability of assigning the positive class to an example is estimated from the training set T and equals $P_v(s) = |\{s \in T(v), c(s) = 1\}|/|T(v)|$. Then s is classified as a positive example when the value of $P_v(s)$ of the leaf appropriate for s exceeds the user specified threshold value. This method of measuring classifier quality made it necessary to switch off the pruning step in the classifier induction.

Table 2. Basic characteristic of DGSplicer datasets.

	Donor	Acceptor
Sequence length	18	36
Consensus position	10	26
Training examples	228 268	322 156
Testing examples	57 067	80 539
Positive examples (%)	0.8	0.6

Table 3. Values of the auROC measure for the DGSplicer dataset.

Algorithm	Donor [%]	Acceptor [%]
WD	97.84	97.50
MC	98.34	97.23
WDS	97.47	97.28
DGSplicer	96.88	95.91
KIS-best	97.29	94.35
MC-SVM	95.08	95.35
KIS-av. KIS-sd.	96.53 (0.41)	93.66 (0.50)

From the results presented in Table 3 it can be seen that all methods under comparison perform better for donors than for acceptors. Values of auROC obtained by KIS are outperformed by most of methods under comparison, but the difference between KIS and the winner is not high. The best KIS result is only 1% worse (donors) and 3% worse (acceptors) than the best result of all compared methods, and it is 2% better (donors) than the worst of all methods under comparison.

We believe that the relative KIS quality could be higher if algorithms were compared by accuracy and FPR/FNR rather than by the auROC measure.

4.2. NN269 data. The NN269 datasets (Reese *et al*., 1997) consist of data which have been organized similarly to the DGSplicer datasets, i.e., separate pairs of training and testing sets are provided for donors and acceptors. These datasets have been created to test the NN269 algorithm and they are available at `http://www.fruitfly.org/sequence/human-datasets.html`. The data for these files were extracted from 269 annotated human genes. The basic characteristic of these datasets is provided in Table 4.

The results for all algorithms except for KIS have been reported by Kashiwabara *et al*. (2007). He provided the results of the best models that have been selected on subsets of training examples. For KIS, in addition to the best result, we report the average and the standard deviation of results obtained in 25 independent runs of the algorithm. Providing the KIS-best value is reasonable because we were able to detect the best model by assessing the classifier quality on the pruning set.

The results for the NN269 dataset are reported in Tables 5 (for donors) and 6 (for acceptors). We provide the values of accuracy, FPR and FNR. Results yielded by KIS are reported after pruning of the classifier. This pruning was performed using 33% examples taken randomly from the training set.

Table 4. Basic characteristic of NN269 datasets.

	Donor	Acceptor
Sequence length	15	90
Consensus position	8	69
Training examples	5 256	5 788
Testing examples	990	1 087
Positive examples (%)	21.0	19.4

Table 5. Classification results for the NN269 dataset (donor).

Algorithm	Acc. [%]	FPR [%]	FNR [%]
NNSplice	94.80	6.80	5.54
KIS-best	94.65	2.43	11.06
Lapfa	94.34	3.84	12.50
KIS-av. KIS-sd.	93.65 (0.48)	3.70 (0.63)	16.29 (2.52)
Alergia	89.09	4.74	34.13
Amnesia	77.68	22.00	23.56
RPNI	76.53	23.92	22.44

It can be observed that the accuracy of the best solution yielded by KIS is the second best one in both tables (5 and 6). The best algorithm for the donor dataset was different than the best one for the acceptor dataset. Moreover, even the average quality of results yielded by KIS is significantly better than best solutions yielded by Alergia, Amnesia and RPNI. The results produced by KIS are also characterized by a very low FPR error, which is important for this task. For the donor dataset, even an average solution yielded by KIS has a better FPR value than the best of the alternative algorithms. For the acceptor dataset, the FPR value of the best KIS result is the second best of the alternatives. The average FPR value of result by KIS is outperformed only by the NNSplice and Alergia. When considering the FNR error, the best result by KIS is the second best for both the donor and acceptor datasets, and the average of KIS results on the fourth and the third position, respectively.

Models produced by KIS are not only highly accurate, but are also likely to be readable by humans, which would therefore facilitate better understanding of donor and acceptor structures. If many algorithms are capable to give comparable results, then it seems to be reasonable that the one that can explain its decisions should be preferred. Figure 4 provides an example decision tree

Table 6. Classification results for the NN269 dataset (acceptor).

Algorithm	Acc. [%]	FPR [%]	FNR [%]
Lapfa	94.67	4.65	8.17
KIS-best	93.39	2.61	12.02
NNSplice	92.13	3.00	26.20
KIS-av. KIS-sd.	91.83 (0.99)	4.59 (1.29)	23.33 (5.67)
Alergia	70.16	1.84	76.44
RPNI	68.54	28.11	45.50
Amnesia	62.63	34.05	51.44

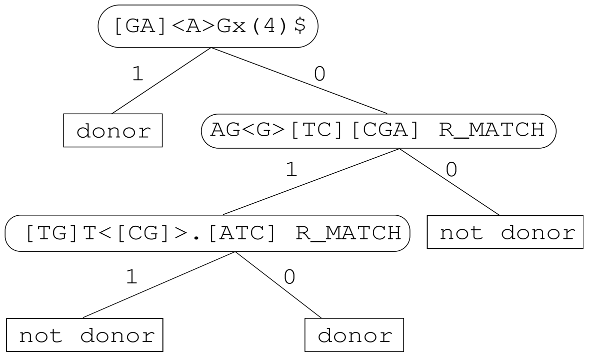

Fig. 4. Decision tree generated for the NN269-donor dataset.

Table 7. KIS results on *IRE100x* (in %).

Model/Algorithm	Acc. [%]	FNR [%]	FPR [%]
Model by Pesole *et al.*	82,61	86,96	0,00
KIS-av	88,17	11,74	11,85
KIS-best	99,13	0,00	1,09

which has been generated by KIS for the donor dataset. Its interpretation seems quite intuitive. First, a matching for a sequence ‘`[GA]AG...$`’ is checked. If a match can be found, then the example sequence considered is classified as a donor. Otherwise, matching of ‘`AG<(G)>[TC][CGA]`’ is checked on the right from the reference position (from the donor consensus ’`GT`’). If this pattern is matched, then the example is classified as not being a donor. Otherwise, the classification process follows according to the decision tree structure.

4.3. IRE finding. The next example task for KIS was to find the IRE sequences in noise and to provide a model of the IRE common to various species. The IRE dataset was prepared by the authors of this paper using data extracted from the 22nd release of UTR.db. It consists of IRE sequences observed for various species (from *fungi* and viruses through invertebrates to human). The task was to verify the hypothesis that structures of the IRE sequence for various species reveal some common features. The secondary goal of the analysis was to find the IRE signals in the presence of noise.

We generated a dataset with 232 positive examples. To make the problem more difficult, the IRE signal was enclosed within randomly generated sequences. They have been generated by concatenation of nucleotides drawn uniformly from the DNA alphabet. Since the IRE model has 27 nucleotides, the authors decided that the examples would be 100 times longer, i.e., 2,700 nucleotides (27 enclosed with random 2 673). The negative examples were generated as random sequences. The number of generated negative examples was 928. This preliminary dataset was randomly divided into training (10%) and testing (90%) datasets. These files are available at `http://staff.elka.pw.edu.pl/~rbiedrzy/KIS/index.html`.

It was not possible to find other tools that solve the IRE detection problem, so KIS was compared with a model introduced by Pesole *et al.* (2000), which has been briefly discussed in Section 2.2 and Fig. 1. The comparison is provided in Table 7. In this case pruning was disabled. KIS was run 10 times, and an average and the best result are provided. Several decision trees of good quality were generated and detected. These good models were investigated more deeply. Most of them used the classifier that was testing one attribute only. This attribute (‘`CL-(.{5,5})CAG[AT]G[ATC]-L MATCH`’) was responsible for detection of *Stem I* and both loops (see Fig. 1). Another good classifier used two attributes: one responsible for the detection of *Loop I*, and the other for the detection of *Stem I*, *Loop II* and a fragment of *Loop I*. For the examined dataset the model introduced by Pesole *et al.* (2000) achieved lower accuracy and a higher false negative ratio than the best result yielded by KIS! Therefore it can be supposed that *Stem I* and both loops are well conserved fragments across the various species, which is not the case for *Stem II*. Thus, KIS was able to build a human readable model with high accuracy by selecting important signals from data that came from various species.

4.4. KIS scalability. Even the best theoretical approach may turn out to be useless for large datasets due to an unacceptable computation time. Therefore KIS scalability was verified. The computing time (in seconds) of the most time-consuming part of an algorithm (which is the attribute construction step) was measured. The measurement was performed using the data generated artificially from the DGSplicer-donor dataset. The dataset was used to derive random samples with the sample size ranging from 200 to 200,000 examples. It was assumed that in every sample of the dataset at least 100 positive examples should be contained. Except for the smallest samples, the proportion of examples from both decision classes was identical for the original dataset and for its random sample. In Fig. 5 the running time of the KIS method is plotted as a function of the sample size. The logarithmic scale was used on both dimensions and the measured points were connected to indicate the slope inclination. In addition, two lines were plotted which correspond to $o(n)$ and $o(n\log(n))$ complexity. It was concluded that the KIS computational complexity is approximately linear when the cache is enabled and it scales like $n\log(n)$ without a cache.

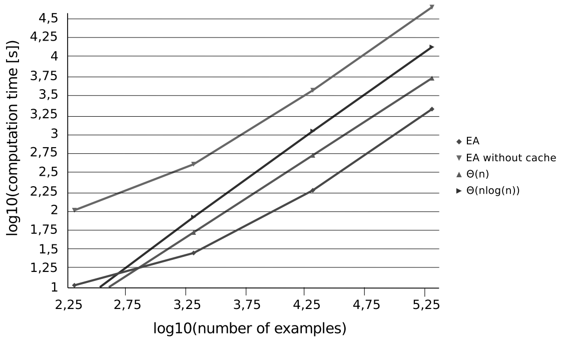

Fig. 5. KIS scalability.

5. Conclusions

We have developed a general method which can be successfully used in various tasks that involve classification of DNA strands. We have demonstrated a case when the method is used to recognize splice sites. In this particular application, KIS generated results of a quality comparable to other tools dedicated to solve this task. This was despite the fact that the length of sequences in both datasets was carefully tuned to maximize the performance of the NNSplice and DGSplicer algorithms. We have also shown that KIS is able to find IRE, i.e., to find characteristic elements that are forming 2D structures. An important advantage of the method is the readability of the generated model, which is usually not the case for other methods. In some applications (e.g., medical diagnosis), model readability is required to assess classifier reliability (Tickle *et al*., 1998; Diederich, 2008). In other cases, it can be learned from the acquired model, which is common in the given family of sequences (like in the IRE case). The program scales well with a number of examples and is able to handle long sequences.

The KIS approach could be used when model readability is important or in tasks where no other tools are available.

Further tests are envisaged to assess the performance of KIS with classification algorithms and other optimization methods.

References

Baten, A.K.M.A., Chang, B.C.H., Halgamuge, S.K. and Li, J. (2006). Splice site identification using probabilistic parameters and SVM classification, *BMC Bioinformatics* **7**(Suppl 5): S15, DOI:10.1186/1471-2105-7-S5-S15.

Berget, S.M., Moore, C. and Sharp, P.A. (1977). Spliced segments at the 5’ terminus of adenovirus 2 late mRNA, *Proceedings of the National Academy of Sciences* **74**(8): 3171–3175.

Carrasco, R.C. and Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method, *ICGI’94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications, Alicante, Spain*, pp. 139–152.

Chen, T.-M., Lu, C.-C. and Li, W.-H. (2005). Prediction of splice sites with dependency graphs and their expanded Bayesian networks, *Bioinformatics* **21**(4): 471–482.

Davis, J. and Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves, *ICML’06: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA*, pp. 233–240.

Deshpande, M. and Karypis, G. (2002). Evaluation of techniques for classifying biological sequences, *Pacific-Asia Conference on Knowledge Discovery and Data Mining, Taipei, Taiwan*, pp. 417–431.

Diederich, J. (2008). *Rule Extraction from Support Vector Machines*, Studies in Computational Intelligence, Vol. 80, Springer, Berlin/Heidelberg.

Durbin, R., Eddy, S.R., Krogh, A. and Mitchison, G. (1998). *Biological Sequence Analysis—Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, Cambridge.

Elsik, C.G., Worley, K.C., Zhang, L., Milshina, N.V., Jiang, H., Reese, J.T., Childs, K.L., Venkatraman, A., Dickens, C.M., Weinstock, G.M. and Gibbs, R.A. (2006). Community annotation: Procedures, protocols, and supporting tools, *Genome Research* **16**(11): 1329–1333.

Kashiwabara, A.Y., Vieira, D.C.G., Machado-Lima, A. and Durham, A.M. (2007). Splice site prediction using stochastic regular grammars, *GMR* **6**(1): 105–115.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs,* 3rd Edn., Springer-Verlag, London.

Oncina, J. and Garcia, P. (1992). Inferring regular languages in polynomial update time, *in* A. Sanfeliu, N. Pérez de la Blanca and E. Vidal (Eds.), *Pattern Recognition and Image Analysis*, World Scientific Publishing, Singapore, pp. 49–61.

Pesole, G., Grillo, G., Larizza, A. and Liuni, S. (2000). The untranslated regions of eukaryotic mRNAs: Structure, function, evolution and bioinformatic tools for their analysis, *Briefings in Bioinformatics* **1**(3): 236–249.

Quinlan, J.R. (1986). Induction of decision trees, *Machine Learning* **1**(1): 81–106.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Francisco, CA.

Rätsch, G. and Sonnenburg, S. (2004). *Accurate Splice Site Detection for* Caenorhabditis Elegans, MIT Press, Cambridge, MA.

Rätsch, G., Sonnenburg, S. and Schölkopf, B. (2005). RASE: Recognition of alternatively spliced exons in *C. elegans*, *Bioinformatics* **21**(Suppl 1): i369–i377.

Reese, M.G., Eeckman, F.H., Kulp, D. and Haussler, D. (1997). Improved splice site detection in Genie, *Journal of Computational Biology* **4**(3): 311–324.

Ron, D., Singer, Y. and Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length, *Machine Learning* **25**(2): 117–149.

Ron, D., Singer, Y. and Tishby, N. (1998). On the learnability and usage of acyclic probabilistic finite automata, *Journal of Computer and System Sciences* **56**(2): 133–152.

Sonnenburg, S. (2009). *Machine Learning for Genomic Sequence Analysis*, Ph.D. thesis, Technischen Universität Berlin, Berlin.

Sonnenburg, S., Schweikert, G., Philips, P., Behr, J. and Rätsch, G. (2007). Accurate splice site prediction using support vector machines, *BMC Bioinformatics* **8**(Suppl 10): S7.

Tickle, A., Andrews, R., Golea, M. and Diederich, J. (1998). The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks, *IEEE Transactions on Neural Networks* **9**(6): 1057–1068.

Rafał Biedrzycki received his M.Sc. (2003) and Ph.D. (2009) degrees in computer science from the Warsaw University of Technology. Currently he is an assistant professor in the Institute of Electronic Systems, Faculty of Electronics and Information Technologies, Warsaw University of Technology. His research interests include machine learning, data fusion, evolutionary algorithms and bioinformatics.

Jarosław Arabas was born in Poland in 1970. He received the M.Sc., Ph.D. and D.Sc. degrees from the Warsaw University of Technology (WUT), Poland, in 1993, 1996 and 2005, respectively. Since 1993 he has been with the Faculty of Electronics and Computer Engineering, WUT, where he is currently a professor. His main areas of research interest include evolutionary computation, optimization methods and learning systems. Since 1995 he has been consulting applications of these methods for power plant control, decision making and risk management in electricity markets.

Received: 19 May 2011
Revised: 4 November 2011
Re-revised: 27 February 2012