Available online at www.sciencedirect.com
SCIENCE DIRECT
Methods 37 (2005) 292–305
METHODS
www.elsevier.com/locate/ymeth

Computational searches for splicing signals

Xiang H.-F. Zhang [a], Christina S. Leslie [b], Lawrence A. Chasin [a,*]

[a] Department of Biological Sciences, Columbia University, New York, NY 10027, USA
[b] Department of Computer Science, Columbia University, New York, NY 10027, USA

Accepted 12 July 2005

Abstract

The removal of introns from pre-mRNA requires as an initial event the accurate molecular recognition of the proper exon–intron borders. It is now evident that RNA sequence elements in addition to the consensus splice site sequences themselves are required for this recognition. Genomic analyses have contributed to the definition of these elements as exonic and intronic splicing enhancers and silencers, comprising what has been called the ''splicing code.'' Many computational methods have been brought to bear in such studies. We describe here some of the methods we have used to discover functional splicing signals. What these methods have in common is a comparison of sequences in and around exons to sequences found elsewhere in the genome. We have especially made use of comparisons to ''pseudo exons,'' intronic sequences resembling exons by virtue of being bounded by sequences indistinguishable from splice sites. Two computational strategies are emphasized: (1) the use of a machine learning technique in which a computational algorithm, a support vector machine, is first trained on known examples and then used to predict sequences associated with splicing; and (2) straight statistical analysis of differences between regions associated with exons and other regions in the genome. In most cases, the predictions made using these methods have been validated by subsequent empirical tests. An attempt has been made to make this description understandable by researchers unfamiliar with computational practice and to include practical references to specific databases and programs.

Keywords: Pre-mRNA; Splicing; Exons; Introns; Pseudo exons; Genomics; SVM; Machine learning; Splicing enhancers; Splicing silencers; Computation; Statistics; ESE; ESS; ISE; ISS; SVM; Statistics

1. Introduction

A crucial event in constitutive as well as alternative splicing is the initial recognition of the exon–intron borders. The most obvious hallmarks of exon–intron borders are the 9, 15, and 7 nucleotide (nt) sequences comprising 5′ and 3′ splice sites and the branch point, respectively. These sequences are incompletely conserved and similar sequences (pseudo splice sites) can be found within introns at a frequency close to that expected based on their degeneracy. As their presence within introns has not been selected against there must exist additional cues responsible for distinguishing real splice sites from pseudo sites. There is good evidence that it is the two ends of the relatively short exons rather than the longer introns that are detected first and in combination, a process termed exon definition [1]. Thus, upstream pseudo 3′ splice sites (acceptors) and potential branch points can be combined with downstream 5′ splice sites (donors) to define ''pseudo exons.'' Like pseudo splice sites, pseudo exons can be easily concocted within introns, such that they outnumber the real exons by up to an order or magnitude [2,3].

The additional sequence or structural information that the cell uses to distinguish real exons from pseudo exons has been termed the ''splicing code [4].'' This code is comprised of four categories of sequences defined simply on the basis of their location and their effect: exonic splicing enhancers (ESEs) and silencers (ESSs) and intronic splicing enhancers (ISEs) and silencers (ISSs). Of these, the ESEs have been studied the most intensively and extensively. Most ESEs have been shown to bind and respond to a family of arginine–serine-rich or SR-proteins. A compendium

* Corresponding author. Fax: +1 212 531 0425.
E-mail address: lac2@columbia.edu (L.A. Chasin).

1046-2023/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ymeth.2005.07.011

of ESE sequences has been generated by iterative selections of sequences that either bind to ESEs or stimulate splicing, often in response to specific SR-proteins [5–10]. Furthermore, ESS elements have been defined through genetic selection [11].

Computational methods have also been used to decipher the splicing code. The strategies used in these searches are based on statistical differences between exons and introns and on the distinctions between exons that are predicted to differ in ESE content [12–16]. In most cases, the general validity of these results has been confirmed by empirical tests. We describe here the computational methods we have used to identify the four types of splicing elements in the human genome. Our strategy has been comprised of two approaches: straight statistical analysis and machine learning in the form of support vector machines. In most of this work, we have made use of comparisons between real exons and pseudo exons as a means to target both positive and negative acting elements. In the following description, we have attempted to explain our methodology in terms that can be understood by those not yet engaged in computational biology and to identify readily available sources of programs and databases. However, the extraction and manipulation of genomic data does often require some knowledge of programming; we have usually used perl scripts for this purpose, but C++ and Java are popular alternatives.

2. Data mining

Before one embarks on computational studies of splicing, exon–intron structures must first be collected for many genes. There are a number of databases that have annotated exon–intron structures, e.g., the Exon Intron Database (http://mcb.harvard.edu/gilbert/eid, [17]), ExInt (http://sege.ntu.edu.sg/wester/exint/index.html [18]) and Xpro (http://origin.bic.nus.edu.sg/xpro [19]). These databases are based on GenBank and contain genes of multiple species. They also provide some tools to aid data extraction and manipulation. However, these databases all have certain limitations in availability or data organization, and may not be able to meet all users' requirements. Fortunately, the development of fast sequence-aligning programs as well as the increasing availability of genomes, full-length cDNAs and ESTs have made it possible to custom build an exon–intron database with only a moderate effort. This section will describe how to do so in a fast and convenient manner. Since most of the work we have done was based on data obtained from the NCBI website, we will focus on this resource in the following paragraphs. We will also introduce some alternative resources at the end of this section.

2.1. Data resources

To generate an annotated list of exons and introns, one needs to align genomic sequences to full-length cDNAs. Therefore, these two kinds of sequences must first be obtained. An EST database is also necessary if one wants to identify alternative splicing events. Most of these files are in compressed format that can by extracted using gunzip (UNIX or LINUX), WinZip (Windows) or Stuffit (Mac).

(1) Genomic sequences of multiple species are available at the NCBI ftp server ftp://ftp.ncbi.nih.gov/genomes/. Each species occupies a separate subdirectory and most genomic sequences are organized by chromosomes. A database with all chromosomal sequences merged can be found at ftp://ftp.ncbi.nih.gov/blast/db/.
(2) Full-length cDNA sequences can be obtained from ftp://ftp.ncbi.nih.gov/refseq/. Both the genomic and cDNA sequences here are from the RefSeq Project [20], which aims to establish a high-quality non-redundant representation of genomes and transcriptomes.
(3) The NCBI ftp server also provides an EST database available at ftp://ftp.ncbi.nih.gov/repository/dbEST/. This database organizes the data according to submission time, and consists of hundreds of small files. A more integrated database can be downloaded from ftp://ftp.ncbi.nih.gov/blast/db/, where the ESTs of each species are concentrated into a single file.

Other datasets that are useful in computational studies on splicing include the following:

(1) HomoloGene (ftp://ftp.ncbi.nih.gov/pub/HomoloGene/ [21]) is a database of homologous gene pairs between sets of two different species. Each pair is recorded with identifier numbers, the quality of the alignment (distance between them) and, most importantly, whether the two are reciprocally the best match, which is indicative of an orthologous relationship.
(2) UniGene (ftp://ftp.ncbi.nih.gov/repository/UniGene/ [21]) is a database that has assembled all the ESTs and mRNAs originating from the same genes. This database can be used as an alternative to the dbEST database mentioned above. However, it is more appropriately used to examine one or a few particular genes, as it presents the data in the form of a easy-to-read graphical Web interface at http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=unigene. Users can input genes of interest and view the alignment among ESTs, mRNAs and genomic sequences.
(3) dbSNP (ftp://ftp.ncbi.nih.gov/snp/) is a database that can be used to locate the single nucleotide polymorphism (SNPs) in your sequences. SNPs in multiple species are available in several different formats. The data we used most often are in the subdirectory "rs_fasta" under the directory of each species; "rs_fasta" contains SNPs with their flanking sequences and can be formatted and used for blast searches.

2.2. Software tools

Software tools can greatly facilitate the identification of exon–intron structures as well as further studies on splicing. Most of these tools implement sequence alignment using different algorithms. The following paragraphs will focus on how to obtain these programs. Their usage will be described in a later section.

(1) Blast-toolbox: this software package is available at ftp://ftp.ncbi.nih.gov/blast/executables/. After downloading a "blast-" file appropriate for the available cpu and operating system and decompressing (with gunzip), one can find executable files for various sequence manipulations. Several of the programs are particularly useful for splicing studies. Megablast [22] is suitable for finding sequences that are highly similar to the query in a large database. Blast2seq performs blast searches between two sequences. It can be used to remove redundant sequences in the database. Formatdb helps to build a customized database for blast searches. Detailed documentation of these executables can be found within the package.
(2) mRNA-to-genome alignment tools: a central task in identifying exon–intron structures is to align mRNAs to genomic sequences. Several programs have been developed to fulfill this need [23–27]. A comparison of these programs has also been carried out [28]. We have used Spidey and sim4 in our work. They are freely available at http://www.ncbi.nlm.nih.gov/spidey/spideysource.html and http://globin.cse.psu.edu/globin/html/software.html, respectively. Detailed instructions for the use of these programs can be found at the download sites.
(3) RepeatMasker: RepeatMasker is a program designed to identify sequences that are highly repeated in a genome. It is widely used in genomic studies. Small-scale jobs can be submitted to the RepeatMasker server http://www.repeatmasker.org/. A local version can also be obtained at the site.
(4) Bioperl: Bioperl is a large Perl module written specifically for bioinformatic studies. It provides convenient interfaces between a Perl script and numerous popular algorithms and enables a user to automate the implementations and parse the outputs of these algorithms [29]. It can be downloaded at http://bioperl.org.

A number of other programs are also potentially useful in splicing studies, such as R (a language for statistical computing, http://r-project.org) and EMBOSS (The European Molecular Biology Open Software Suite http://emboss.sourceforge.net/ [30]), which are both comprehensive packages containing numerous functions and algorithms. We shall refer to specific functions within these packages in the following sections. A comprehensive description of the capabilities of this software is beyond the scope of this paper.

2.3. Alternative resources

(1) UCSC genome browser (http://genome.ucsc.edu/ [31]): a site containing a large collection of genomes. The data are well organized according to species. Inter-species genome-scale alignments are also available.
(2) Ensembl (http://www.ensembl.org/ [32]): a site providing genome data and annotations. It also offers supporting software to facilitate automatic extraction of the data.

2.4. Aligning mRNAs to genomes

(1) Create a local customized database: the sequences downloaded from the NCBI website are usually a simple collection of sequences in a FASTA or GenBank flatfile format. To serve as target databases in blasting, they need to be formatted. The executable "formatdb" in the blast-toolbox carries out this function. Users can find detailed instructions from the "readme.formatdb" file that comes with the toolbox. A snapshot of the Megablast output can be found at http://cubweb.biology.columbia.edu/lac2/methods/mega.txt.
(2) Localize mRNAs in the genome: accurate mRNA-to-genome alignment is much slower than the heuristic alignment carried out by blast. To finish the job in a reasonable amount of time, one needs first to approximately localize mRNA sequences in the chromosome using the much faster alignment tool Megablast. Megablast can localize tens of thousands of mRNAs in a few hours on an average modern PC. Again, one can easily learn how to use this executable by reading the instructions within the blast-toolbox. It should be noted that the same mRNA could be placed in multiple locations due to the presence of homologous genes or pseudo genes. A stringent cutoff should be used to preclude such false locations. One important parameter to which users need to pay attention is the percent identity of the alignment. If both the genomic and mRNA sequences are from the RefSeq database their quality should be high enough to ensure a close-to-perfect identity. We usually require the alignment to have at least 98% identity. If the location is real, an mRNA would ideally generate a series of separated hits that are close to one another, with each hit corresponding to an exon. Small exons would not give rise to significant hits, but in this first step the mRNA would still be correctly localized to a genomic segment as long as it contains at least one exon of average size.
(3) Accurate alignment between mRNA and the corresponding genomic region: after the mRNA is localized, a region that is likely to contain the corre-

sponding gene would then be extracted from the genomic sequence according to the coordinates of the blast hits (extending an arbitrary distance (e.g., 100 kb) on both sides in order to include the full-length gene). This mRNA would then be aligned to this region in a much more accurate manner using Spidey or Sim4. The instructions of using these programs are available at their downloading sites. An exemplar output of Sim4 can be found at http://cubweb.biology.columbia.edu/lac2/methods/sim4.txt. Bioperl contains a module that can aid users to parse sim-4 outputs within a Perl script. A typical script employing Bioperl to parse sim4 output is shown at http://cubweb.biology.columbia.edu/lac2/methods/perlsim4.txt.

2.5. Identification of alternatively spliced exons

To identify alternatively spliced exons or alternatively used splice sites in silico, one needs to align the joints of each exon and its adjacent exons to the EST database. The first step is to format the EST database again using formatdb. Next, for each exon we generate different exon–exon joint sequences assuming it is either included or skipped. We call this step "in silico splicing." The joint sequences are then aligned to the EST database. If significant alignments are found, the existence of the joint sequences are supported and the corresponding fate (skipped or included) of the exon is recorded. These processes are demonstrated in Fig. 1. It has been shown that the number of ESTs that cover certain joints is correlated with the frequency of certain splicing events. Therefore, whether an exon is mostly included or skipped can be deduced from the ratio between the numbers of ESTs that correspond to either form. Similar steps can also be applied to identify alternatively used splice sites.

We used Megablast to align joints with ESTs. Here, a critical question is how to distinguish real matches (between a joint and an EST) from false matches caused by homology. Users have to decide on some important

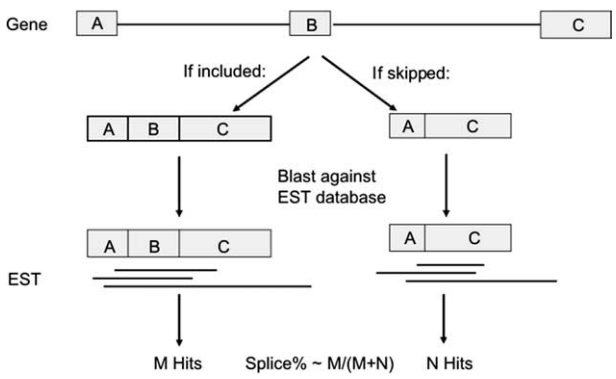

Fig. 1. Scheme for the method used to classify exons as constitutive (always included) or alternatively spliced (sometimes skipped).

parameters. One is the length of the joints. In theory, a 20-mer is long enough to be unique in the genome. In practice, we use 40-mer (20 nucleotides on each side of the joint). Another important cutoff is again the identity of the alignment. As is known, EST sequences are of lower quality compared to the RefSeqs. Therefore, an identity cutoff that is too stringent would eliminate the real matches that are tainted by sequencing errors. We usually allow one mismatch per 40 nucleotides.

3. Pseudo exons as the contrast sequences

We define pseudo exons as regions of introns that are bounded by short sequences resembling splice sites and are of lengths typical of real exons. Pseudo exons abound in introns but are efficiently ignored by the splicing machinery despite their splice site-like sequences. Studying the differences between the pseudo exons and real exons will help to identify sequence information that is used by the cell in splicing.

3.1. Scoring splice sites

To prepare a pseudo exon dataset, first one has to define pseudo splice sites and therefore must decide on a metric to evaluate how well a particular sequence matches the splice site consensus. The simplest way of doing so is to count mismatches between the sequence and the consensus. For example, the 9-mer aaggtcagc has two mismatches with the 5′ splice site consensus "MAG|gtragt," where M = A or C, r = a or g, uppercase = exonic, lowercase = intronic, and "|" = the exon–intron boundary. This approach treats each position equally and ignores the fact some positions are more conserved than others. A more sophisticated method is to use a position specific scoring matrix (PSSM). Such a matrix can be derived by aligning a sufficiently large number of real splice sites and calculating the occupancy by each nucleotide at each position. PSSMs for the 5′ and 3′ splice sites of constitutive exons are shown in Table 1. Shapiro and Senapathy [2,33] calculated a consensus value (CV) for a 9-mer 5′ splice site with a GT at fourth and fifth positions essentially as follows:

$$\mathrm{CV} = \left(\frac{\sum_i f_{i,n} - \sum_i f_{i,\min}}{\sum_i f_{i,\max} - \sum_i f_{i,\min}}\right) \times 100, \quad (1)$$

where $i = -3,-2,-1,+1,+2,+3,+4,+5$, and $+6$; $f_{i,n}$ is the frequency in the matrix of the nucleotide n (that actually occurs in the sequence) at position i; $f_{i,\min}$ is the frequency of the rarest nucleotide (among the four possible nucleotides in the matrix) at the position i; and $f_{i,\max}$ is the frequency of the most common nucleotide (among the four possible nucleotides in the matrix) at position i. As can be seen, the CV is a number ranging from 0 to 100.

Another PSSM-based approach is called the log-odds score [34]. Rather than the raw frequencies as entries in

Table 1
Positional scoring matrices[a] for splice sites

	−3	−2	−1	1	2	3	4	5	6						
5′ splice sites															
A	0.349	0.626	0.106	0.000	0.000	0.621	0.709	0.090	0.184						
C	0.345	0.118	0.032	0.000	0.000	0.022	0.062	0.055	0.141						
G	0.181	0.111	0.778	1.000	0.000	0.328	0.119	0.788	0.178						
T	0.125	0.145	0.084	0.000	1.000	0.030	0.110	0.066	0.497						
	−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	1
3′ splice sites															
A	0.107	0.103	0.088	0.083	0.085	0.094	0.104	0.100	0.083	0.082	0.228	0.061	1.000	0.000	0.257
C	0.290	0.279	0.278	0.267	0.275	0.297	0.335	0.344	0.341	0.300	0.284	0.645	0.000	0.000	0.140
G	0.130	0.113	0.111	0.105	0.115	0.109	0.104	0.097	0.066	0.064	0.201	0.003	0.000	1.000	0.485
T	0.472	0.505	0.523	0.545	0.525	0.500	0.457	0.459	0.511	0.554	0.288	0.291	0.000	0.000	0.118

[a] Based on 118,000 constitutive exons in non-redundant genes. The Sept. 2004 release of the dbEST database was used to assess constitutivity (i.e., no evidence for skipping). Only splice sites containing GT and AG dinucleotides were considered.

the matrix, log-odds scoring uses the ratio of an observed frequency to that of a frequency based on random expectation. A log-odds score is defined as

$$l_{i,n} = \log_2(p_{i,n}/q_n), \tag{2}$$

where $p_{i,n}$ is the frequency of nucleotide n (that actually occurs in a sequence) at position i. and q_n is the background frequency of the nucleotide n. The background frequency is often taken to be 0.25 for each nucleotide, but can be based on local base compositions or some other assumptions. The choice of background model can make a significant difference in the log-odds scores, and so should be carefully considered. The final log-odds score of the sequence is the sum of the log-odds scores at each position.

$$L = \sum_i l_{i,n}, \tag{3}$$

where $i = -3,-2,-1,+3,+4,+5$, and $+6$ for the 5′ splice site. It should be noted that the unit of log-odds scores is ''bits'' according to information theory.

The approaches we described above are straightforward and widely used. A number of more sophisticated algorithms have been developed to distinguish real splice sites from pseudo splice sites. Burge and Karlin [35] used maximum dependency decomposition (MDD), a method that considers interdependency among different positions within splice sites. Pertea et al. further developed this approach and combined it with a Hidden Markov Model and information reflecting a coding–noncoding transition. Their efforts resulted in GeneSplicer http://www.tigr.org/tdb/GeneSplicer/gene_spl.html [36]. Brunak et al. used an artificial neural network to predict splice sites premRNAs http://www.cbs.dtu.dk/services/NetGene2/ [37,38]. Yeo and Burge [39] recently developed a framework of modeling splice sites based on maximum entropy principles. Zhang et al. [14] approached the question using support vector machines and considered dependencies between more than 2 positions in the consensus region (see the following sections). Despite the respectable success achieved by these methods it is apparent that no algorithm based solely on the splice sites would suffice, since the exact 9-mer sequences used elsewhere as real 5′ splice sites are also easily found within introns.

3.2. Culling pseudo exons from introns

Pseudo exons are defined as intronic sequences bounded by an upstream false 3′ splice site and a downstream false 5′ splice site. The false splice sites are short intronic sequences that exceed certain cutoffs according to a scoring scheme (as introduced above) but are not used in splicing. Choices of different scoring schemes and cutoffs will vary depending on the researchers' purposes. It is also necessary to set limits on the length of pseudo exons. In our own studies, this range has usually been between 50 and 250 nt, a range that includes most (about 75%) real exons. To make sure the pseudo exons are really ''pseudo,'' we compared pseudo exons with EST databases using ''megablast.'' Those that have significant matches are eliminated from the final dataset. We recommend a relatively high E value cutoff in Megablast, e.g., an E value $\geqslant 10^{-20}$ so as to eliminate as many suspicious pseudo exons as possible. An E value is the random chance of finding an alignment of a given quality in a database of a given size. Some remaining pseudo exons may still be problematic because the EST database is not yet complete. The scheme for the culling of pseudo exons is shown in Fig. 2.

3.3. Choosing pseudo exons

It may be useful to impose different scoring schemes and cutoffs when choosing pseudo exons. For example, if one wants to study possible exonic splicing silencers (that are likely to be overrepresented in pseudo exons), one may want to restrict the pseudo exons to those with splice sites that are as close to real exons as possible, because such pseudo exons are more likely to be actively repressed in splicing. Although the more sophisticated scoring schemes perform better in distinguishing real splice sites from the

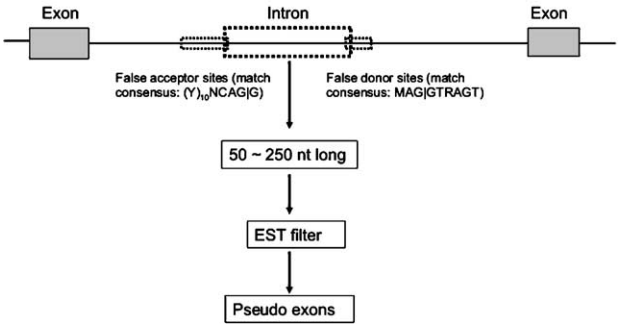

Fig. 2. Definition of pseudo exons. Only non-overlapping pseudo exons are used.

pseudo sites, they have several disadvantages compared to simpler approaches. (a) They are more complicated to implement. (b) Since trillions of subsequences may have to be examined in search of pseudo exons, computing time becomes a limiting factor for complicated algorithms. (c) The performance improvement brought about by these algorithms is only marginal (even if substantial and significant). As a result, the simpler approaches, such as the CV [2,33] and log-odds scores [34], are mostly used in bioinformatic studies that do not focus on splice sites per se. We did not find significant differences between using CV and log-odds scores [14] to distinguish real and pseudo exons. Different cutoffs greatly influence the number of pseudo exons that can be found. One way of choosing the cutoff is to ask how many real exons can actually meet the same criteria. For example, if the cutoff is set to be the lower quartile of the scores of real splice sites, 25% of real exon would fall below the cutoff for one splice site. If both splice sites combined, about one third of real exons would fail. Using this rather stringent cutoff, unique pseudo exons (i.e., non-overlapping) are approximately twice as frequent as the real exons from the same genes. The number of pseudo exons decreases exponentially as the cutoff increases. Too stringent a cutoff would result in a relatively small number of real exons as well as pseudo exons, and therefore generality may be sacrificed.

3.4. Highly repeated sequences and pseudo exons

Highly repeated sequences constitute approximately 50% of the human genome. More than 50% of pseudo exons overlap with repeats. It is not clear whether removal of the repeat-overlapping pseudo exons is the best thing one can do in computational studies, as these pseudo exons may still be seen by the cellular splicing machinery and so may represent an important class that should not be ignored. However, one needs to be cautious in interpreting any results from a repeat-containing dataset, since the repeats are highly non-random and could give rise to statistically significant results that have nothing to do with splicing. Therefore, it is always beneficial to mask repeats and compare the results before and after the repeats are masked. RepeatMasker is the program most widely used for this purpose. It should be noted that highly degenerate repeats may exist that cannot be recognized by RepeatMasker.

4. Computational studies on splicing using SVM

4.1. Overview of SVM

SVMs are a state-of-the-art technology for data classification. Readers do not need to understand all theories underlying SVMs in order to use them. However, we provide a brief overview in the following paragraphs to introduce some basic ideas behind SVMs. A more detailed description of SVMs can be found in [40]. Readers not interested in a description of SVMs can proceed to the following section.

SVMs are used to learn a binary classification rule from labeled (positive or negative) training data. Each training example is represented by a vector of real-valued features $\underline{x} = (x^1, x^2, \ldots, x^n)$, together with a class label y, which is either $+1$ (positive example) or -1 (negative example). The features x^j are real-valued descriptors of the data; in our analysis below, each feature represents the number of times a particular sequence pattern occurs in the example sequence. The n-dimensional vector space in which the training vectors reside is called the *feature space*. Given a training set of N labeled vectors, the SVM solves an optimization problem to learn a linear decision function in feature space, $f(\underline{x}) = \langle \underline{w}, \underline{x} \rangle + b$, where $\underline{w}$ is the normal vector to the linear decision boundary, b is the bias term, and $\underline{x}$ represents the feature vector of the example to be classified. More precisely, SVM training determines a set of real-valued *weights* $\alpha_i \geqslant 0$ such the normal vector can be expressed as a linear combination of training vectors, $\underline{w} = \sum_{i=1\ldots N} y_i \alpha_i \underline{x}_i$. Training vectors $\underline{x}_i$ having non-zero weight are called *support vectors*; they are the "difficult to classify" training examples and precisely the ones that determine the SVM solution. Once the SVM is trained, predictions can be made on a test sequence $\underline{x}$ by predicting positive if $f(\underline{x}) > T$ for a given threshold T, and negative otherwise.

SVMs learn "large margin" classification functions. If a linear decision boundary separates the positive and negative training examples, its margin is defined as the distance of the nearest training example to the decision boundary. SVMs try to maximize the margin, that is, learn a decision boundary such that the nearest training example is as far away from it as possible. In practice, the data may not always be separable, and one wants to allow some training examples to be misclassified or at least to violate the margin of separation achieved by the rest of the training set. One therefore uses a "soft margin" SVM, which realizes a trade-off between maximizing the margin and minimizing margin violations. This trade-off is determined by a positive real-valued parameter; once the parameter is set, the solution to the SVM optimization problem is unique.

Theoretical analysis attributes the excellent generalization performance of SVMs—that is, the ability of SVMs to make accurate predictions on test examples—to this large margin property. In particular, SVMs are ideal for very high-dimensional feature spaces, since it is the size of the margin rather than the number of features that is important for generalization. SVMs have been successfully used for many high-dimensional classification tasks, including document categorization, classification of tumor samples using gene expression data, and image classification.

Another key property of SVMs is the ability to use kernels rather than explicit feature vectors. To train or make predictions with an SVM, one only needs to use the inner products between pairs of training vectors, not the vectors themselves. The inner product between two input vectors $\underline{x} = (x_1, x_2, \ldots, x_n)$ and $\underline{u} = (u_1, u_2, \ldots, u_n)$ is defined as $\langle \underline{x}, \underline{u} \rangle = \sum_{i=1\ldots n} x_i \cdot u_i$. Instead of using the standard inner product for SVM training, one can use a kernel function, which implicitly represents the inner product of two examples after they have been mapped (usually non-linearly) into a higher dimensional feature space. For example, we can replace the inner product by the degree two polynomial kernel given by $K(\underline{x}, \underline{u}) = (\langle \underline{x}, \underline{u} \rangle + C)^2$, where C is any real number. This means that whenever the SVM learning algorithm needs to compute the similarity of two training vectors $\underline{x}$ and $\underline{u}$, it calculates $K(\underline{x}, \underline{u})$ instead of $\langle \underline{x}, \underline{u} \rangle$. For the special case where the input vectors are two-dimensional and the parameter C is 0, one can readily check that this particular kernel is equivalent to first mapping the input vectors $\underline{x} = (x_1, x_2)$ to higher dimensional feature vectors $\left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$ and then taking the standard inner product of the feature vectors for $\underline{x}$ and $\underline{u}$; one can similarly work out an equivalent non-linear feature map for the general case of a polynomial kernel. The SVM learns a linear decision boundary in this implicit feature space, which corresponds to a non-linear boundary in the original input vector space. In our previous example, we can think of the SVM as implicitly defining a linear boundary in the three-dimensional feature space, but when we visualize the boundary in the two-dimensional input space, it is curved. The use of kernels with SVMs therefore provides a way of learning non-linear classification rules while retaining the theoretical justification of the large margin linear classifier.

4.2. Practical use of SVM

In this section, we use real exons and pseudo exons as positive and negative datasets. However, SVM can easily be extended to other comparisons, such as constitutive exons vs. alternative exons [41]. An SVM software package called GIST written by William Noble can be downloaded at http://microarray.cpmc.columbia.edu/gist/. It comprises the core SVM algorithms as well as a rich source of accessories. A web server is also available at http://svm.sdsc.edu/cgi-bin/nph-SVMsubmit.cgi. The preprocessing steps that we describe below are specific to the GIST package; other SVM software packages, such as the widely used SVM-light package http://svmlight.joachims.org/ by Thorsten Joachims [42], require similar inputs but assumes different file formats. A flow-chart of SVM analysis is shown in Fig. 3; the following paragraphs are organized according to the flow chart.

(1) Datasets: both the real exons and pseudo exons are randomly divided into a training set and a test set (shown as thick curred gray arrows in Fig. 3). Typically, one assigns between 10 and 40% of the data to the test set and uses the remainder as training data. The training set is used for SVM to learn the rules, or classifier, for distinguishing the two kinds of exons. The learned classifier is then applied to the untouched test set; the SVM performance on the test set reflects its predictive power.

(2) Feature selection: to apply an SVM, each sequence must be represented by a vector of feature values. What features are used is a critical decision. For example, one could use base composition as a source of features to represent a sequence. Each sequence is transferred to a vector: $\{f_A, f_C, f_G, f_T\}$ For instance, a 10 nt sequence "ACTTAAGATG" would correspond to a vector {0.4, 0.1, 0.2, 0.3} reflecting the abundance of each base. Feature values can be continuous, integer and even binary. For instance, we could ask what dimers are present and use this information for features. Since there are 16 possible dimers from "AA" to "TT," the feature vector would have 16 components and the general form can be written as: $\{b_{AA}, b_{AC}, \ldots, b_{TG}, b_{TT}\}$, where b is a Boolean function. When the corresponding dimer is present, $b = 1$; otherwise $b = 0$. The particular vector for "ACTTAAGATG" is: {1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1}. The selection of features reflects the hypotheses users are making. By using a base composition vector, users hypothesize that the difference between the positive and negative datasets can be best described by the relative abundance of each nucleotide; whereas by using dimer presence as features,

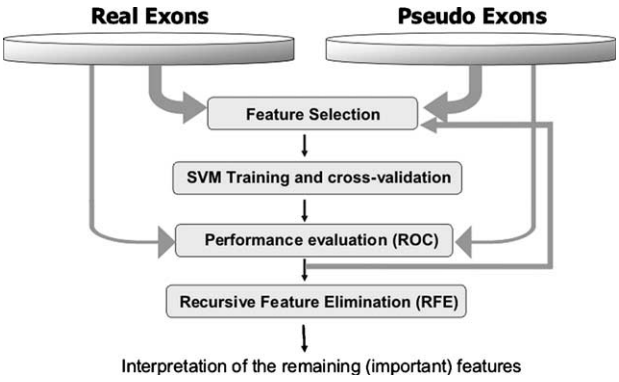

Fig. 3. Flow chart for SVM training and testing.

one assumes dimers are more important than monomers and presence is more important than abundance. Of course, these hypotheses would be rejected if SVM performance is not satisfactory, as will be discussed.The selection of features also involve deciding what regions one wants to examine. For instance, one could use 5-mer frequencies in exon bodies as features to distinguish real exons from pseudo exons. The same thing could also be done in the 400 nt neighboring intronic regions of exons (which we call "flanks"), with the hypothesis being that this region is distinctive. Features from different regions (e.g., all 5-mer frequencies in exon bodies and in 400-nt flanks) could be combined, which is equivalent to concatenating the feature vectors that represent different regions. The concatenation would result in a longer vector. Each specific manipulation of features implies an hypothesis.

(3) SVM training and cross-validation: after the feature is decided, each sequence is transformed into a vector. Two input files need to be prepared for the SVM. One is a tab-delimited file with labels of sequences in the first columns and features in the subsequent columns. Each row represents a sequence and each column represents the same feature. The second is a two-column tab-delimited file with the labels in the first column and a binary value in the second column. The binary is the label of the class (positive or negative). Detailed instructions can be found at the SVM software downloading site.

We usually carried out cross-validation during the process of training for a temporary idea of how well and stably the SVM performs on unseen data. The training set is split into K subsets of equal size. The training algorithm is performed on $K-1$ subsets and the learned classifier is applied to the remaining subset to test the performance. This process is repeated K times, each time using a different subset as the test set. The mean and variance of the K performances (as ROC scores, see below) are then examined. Small variances indicate that the SVM has a stable predictive power that is not influenced by a few particular samples. This procedure is called a K-fold cross-validation. Ideally, one would like to do the extreme, letting K equal the sample size N so that only one sample is left out at each time. This is called leave-one-out cross-validation (LOOCV). In this way, one could maximally utilize the data (waste only one sample in training at each time) and spot individual samples that greatly influence the performance (especially when N is small). However, LOOCV is computationally expensive since the training has to be run N times.

(4) Sample size, number of features, and the test set: if the number of features greatly exceeds the number of samples, one should be cautious about *overfitting*. Overfitting is usually defined as trying to learn too complex a decision rule from limited training data, leading the classification algorithm to learn random patterns in the training dataset and hence fail to generalize to new examples. In general, overfitting can occur if there are too few training examples, too many irrelevant features, or too many parameters to fit, or if the user tries many different parameters for the classifier and chooses the one with the best performance on a held-out set. Note that if our training examples belong to a feature space of sufficiently high dimension, it is always possible to linearly separate the positive and negative examples (in fact, there are infinitely many such separating linear decision boundaries), and many learning algorithms are vulnerable to overfitting in this setting. In theory, the SVM classifier offers protection against overfitting by selecting a "large margin" linear decision rule, and theoretical statements about SVM generalization are independent of the dimension of the feature space. In practice, SVMs are often successfully used in very high-dimensional feature spaces. However, in situations where there are a large number of irrelevant or noisy features or possibly when too complex a kernel is used, the SVM can still overfit the training data and fail to generalize. The best methodology to avoid overfitting and reporting overly optimistic results is to use cross-validation on the training set to determine parameters and to perform feature selection and then to apply the learned classifier to a new dataset that is totally independent of the training set and examine its performance there. We call this independent dataset the test set. The test set should be sufficiently large so that the performance is not expected to have a large variance. Thus, one can incorporate as many features as an hypothesis demands as long as one has sufficient computing power and an independent test set is used for evaluation.

The opposite problem, *failure to learn*, occurs when the representation of the classifier is not complex enough to describe a good decision rule for the training data. This problem can occur with SVMs, for example, if one uses a linear feature representation when in fact a more complex (non-linear) kernel is needed. Failure to learn is easier to detect, however, because the proportion of misclassified examples on the training set will be high.

(5) ROC score: sensitivity (Se) and specificity (Sp) are the most popular indices of the predictive power of a classifier. Sensitivity indicates its ability to detect positives (true positives, TP) and can be expressed as the true positive rate, $TP/(TP+FN)$ where $FN =$ false negatives. Specificity is related to its ability to detect negatives (true negatives, TN) and can be expressed by the true negative rate, $TN/(TN+FP)$, where $FP =$ false positives. Specificity can also be considered as the ability to avoid negatives; in this case it

is convenient to use the false positive rate, which is 1− the true negative rate, or $1 - TN/(TN + FP) = FP/(TN + FP)$. The better the classifier, the higher the true positive rate and the lower the false positives rate. As the cutoff value of the classifier is varied, Se and Sp vary in opposing directions; i.e., as one wants to capture more real positives by lowering the cutoff more false positives are inevitably included. Therefore, the values of both indices depend on the choice of cutoffs, and a particular Sp and Se pair cannot provide a complete view of the performance of a classifier. To depict the trade-off between Sp and Se, one can draw a curve by plotting a series of points with $1 - Sp$ as the x-coordinate and Se as the y-coordinate. This plot of the true positive rate as a function of the false positive rate is called the receiver operating characteristic curve (ROC curve). An example of ROC curve is shown in Fig. 4. The closer the curve is to the left-hand and top borders, the better the performance. The area under the curve is called a ROC score, which ranges from 0 to 1. A random classifier has an expected ROC score of 0.5 on a test set with an equal number of positive and negative values, whereas a perfect classifier would yield a score of 1.0. Unlike Se and Sp, which change as a cutoff changes, a ROC score is independent of cutoff choice and summarizes the ranking performance of a classifier.

In our studies, we asked whether certain features are more distinguishable than others by comparing ROC score results using different features or combination of features [14]. In this way, we learned the best features that can distinguish real exons from pseudo exons. For example, we found that examining the 50 nt immediate flanks of exons is much more distinctive than using 400 nt flanks and therefore concluded that most intronic splicing information is within 50 nt but not 400 nt flanks. We also learned that 5-mer frequencies are better features than 3-mer and 2-mer frequencies, which indicates that the relevant sequence motifs are likely to be 5-nt or longer. The comparison of ROC scores of SVM tests permits a relatively rapid empirical testing of different features reflecting different hypotheses.

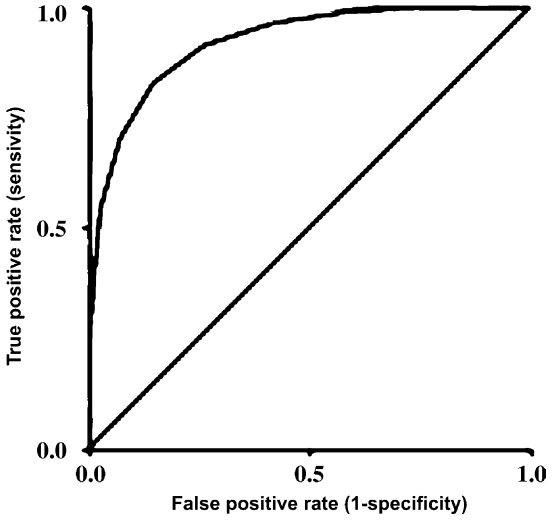

Fig. 4. Receiver operating characteristic (ROC) curve example. The area under the curve is used to assess the performance of a classifying algorithm (maximum = 1.0, minimum ~0.5). The ordinate is a measure of sensitivity, $TP/(TP + FN)$; the abscissa is a measure of specificity, the false positive rate ($FP/(FP + TN)$), being 1− the true negative rate ($1 - TN/(FP + TN)$), where TP = true positive; FN = false negatives; TN = true negatives; and FP = false positives.

(6) Feature extraction: in the SVM solution, the normal vector to the hyperplane decision boundary is defined by:

$$\underline{w} = \sum_{i=1\ldots m} y_i \alpha_i \underline{x}_i, \tag{4}$$

where $\underline{x}_i$ are the training feature vectors, $y_i = \pm 1$ are the labels, and α_i are weights learned during training, which are accessible in the output of the SVM program. The coordinates of the vector $\underline{w}$ can be used to rank the importance of features: if a coordinate $|w_j|$ is large in absolute value, then the jth feature is important for SVM; the sign of w_j shows whether it is indicative of positive or negative examples. In standard recursive feature elimination (RFE), one trains an SVM, uses the ranking induced by the $|w_j|$ to eliminate the bottom half of the features and recursively retrains on the smaller feature set.

In our previous work, we concatenated k-mer feature vectors for the upstream and downstream flanks and then used a degree 2 polynomial kernel to combine both sources of information. We could no longer easily compute $\underline{w}$, since we implicitly used a non-linear feature mapping. However, we could use the learned SVM weights α_i to compute vectors $\underline{w}_{up}$ and $\underline{w}_{down}$ using Eq. (4) with the k-mer feature vectors for the upstream and downstream flanks, respectively. We then eliminated the bottom half of the features for each flank separately and retrained on the smaller feature set. This procedure approximates RFE in our setting. The recursive process was ended when the ROC score for the SVM on an untouched test set fell below 90% of the original ROC score (obtained using the full feature set). The result was a subset of features that was responsible for most of the discriminative power of the classifier.

(7) Using SVM to score splice sites: as discussed in the previous section, there exist a number of different algorithms for predicting splice sites. We describe our own scoring scheme in the following paragraph for a double purpose: (1) to introduce it as a separate and independent approach, which is potentially useful to identify and characterize other sequence motifs; and (2) to further exemplify the usage of SVM.

Scoring schemes based on PSSMs assume that different positions within splice sites are independent of each other. More complicated algorithms take into account the dependency among different positions [35,37,39]. However, these algorithms only consider the dependency of two different positions and ignore higher-order dependencies. We used SVM to explore higher order base dependencies. For example, in a 5′ splice site a G at position +3 along with an A at position +6 is a two-way base combination (denoted as +3G_+6A). If it occurs within a given splice site, we assign a value "1" to the corresponding feature; otherwise we assign a "0." In this way, each splice site is transformed into a vector composed of binary values, each representing the occurrence of a particular two-way base combination. Pseudo splice sites can be transformed in the same manner. SVM was applied to distinguish real and pseudo splice sites and the performance was recorded. We then repeated this procedure using three-way, four-way and up to seven-way base combinations to distinguish real and pseudo 5′ splice sites (positions +1 and +2 were not allowed to vary from GT). A comparison of SVM performances showed that three-way and four-way combinations were significantly better than all the other choices. An example of a three-way base combination is −2A_+3A_+6T. This result not only provides a novel way of measuring the strength of a splice site but also indicates that there is information in higher-order base combinations within splice sites.

5. Computational studies on splicing using straight statistics

SVM is powerful but has some disadvantages: (a) it can be slow, the efficiency depending on the algorithm used to solve the SVM optimization problem, the number of features and the sample size. Large values require sizeable computer memory. These consideration dictate limits on the application of SVM to problems involving large amount of data. For instance, processing a sample of 12,000 sequences and 10 features requires more than 3 h using the GIST package on a PC with 1G memory and a 1.8 GHz CPU. For very large datasets (>50,000 training examples), it is more appropriate to use efficient SVM implementations such as SVM-light (see above). (b) Although important features can be extracted from SVM, no P values can be associated with these features. As a result, their statistical significance cannot be readily evaluated. Straightforward statistical approaches are complementary to SVM in this regard. In this section, we will present an overview of some statistical approaches in the study of splicing.

5.1. Genomic analysis: identification of candidate sequence motifs by over (under) representation

This strategy rests on the assumption that a function for a sequence motif is implied if it is found to occur more/less frequently than expected within a certain context. In the case of splicing, both positive signals (splicing enhancers) and negative signals (splicing silencers) have been experimentally identified in exons and introns. It seems reasonable that enhancers should be overrepresented and silencers underrepresented in and around most exons. Therefore, by systematically searching for over/underrepresented sequence motifs in the corresponding regions, one should be able to collect a list of candidate splicing signals. Below we discuss several aspects of such statistical searches.

(1) The location of intronic signals: intronic splicing enhancers/silencers (ISE/Ss) are much less well studied than their exonic counterparts. According to the studies where individual ISE/Ss are documented, they can occur more than 100 nt from intron–exon borders. On the other hand, SVM can distinguish real exons from pseudo exons to a much better degree when examining sequence motifs limited to upstream and downstream 50 nt flanks [14]. Therefore, we hypothesized that the vast majority of intronic splicing signals are concentrated in these regions. The statistical analysis described below showed that the non-randomness of mammalian introns is likewise limited to the same regions.
(2) Protein coding and splicing: unlike introns, the vast majority of exons code for proteins and so comprise codons, which present a confounding difficulty in defining exonic splicing signals. Protein coding is highly non-random due to (a) non-random distributions of amino acids in the proteins; (b) codon usage bias [43]; and (c) non-random dicodon frequencies [44]. This information is presumably unrelated to splicing signals. As a result, one must find exonic splicing signals apart from this "noise." One way to achieve this purpose is to compare two different groups of coding sequences. Thus, Fairbrother et al. [16] compared exons with weak splice sites against those with strong splice sites, assuming that the former need more exonic splicing enhancers. Since both sets are coding for proteins, the noise is canceled out in the comparison. A second way is to use exons that do not code for proteins. Zhang and Chasin compared non-coding internal exons in 5′-UTRs with nearby pseudo exons and 5′-UTRs of intronless genes. All three sets do not code proteins [13]. In the above two examples, Fairbrother et al. set the background to be protein coding, whereas Zhang and Chasin avoid protein-coding in the foreground.
(3) Statistical measurement of over (under) representation: to collect a candidate list of motifs, one has to examine the frequency of each possible oligomer in the foreground dataset and then compare it to the expected frequency. The expected frequency can be derived in several different ways, and this important choice can greatly influence (and possibly bias) the results. Suppose for a particular oligomer O with length L, one gets a count, O_f, in the foreground

dataset. The expected count, can be calculated from a background model: $O_b = N \times B(O)$, where B is a background model, $B(O)$ is the frequency of O according to the background model and N is the total number of oligomers with length L in the foreground set. For example, one can use base composition as a background model and calculate $B(O)$ by multiplying the nucleotide frequency at each position: $B(O) = \prod_L f_n$, where f_n is the frequency of the nucleotide occurring at each position of O. This model gives the mathematical expectation when there are no interdependencies among bases and the motif examined occurs randomly in the dataset. More complicated and presumably more accurate background models can be made based on sophisticated assumptions [13].

O_b could also be a count of O in another dataset, which serves as a background set. For example, when looking for splicing signals in exon flanks, we have used deep intron sequences as a background set, i.e., O_b is the count of O in the deep intron sequences [14]. In some cases, it is not clear which dataset is foreground and which is background. For instance, when we compared non-coding internal exons with pseudo exons, each dataset served as background for the other since enhancers are likely to be overrepresented in the former whereas silencers are likely to be more frequent in the latter.

Once O_f and O_b are decided, one has to judge whether their difference is significant or not. When the O_f and O_b counts are sufficiently large (e.g., >20), one can use Z-scores to measure the difference

$$Z = \frac{\frac{O_f}{N_f} - \frac{O_b}{N_b}}{\sqrt{\left(\frac{1}{N_f} + \frac{1}{N_b}\right)\left(\frac{O_f+O_b}{N_f+N_b}\right)\left(1 - \frac{O_f+O_b}{N_f+N_b}\right)}}. \quad (5)$$

In theory, Z follows a standard normal distribution (normal distributions with mean = 0 and variance = 1) so the corresponding P value can be easily calculated from a table or in Excel (using 1-NORMSDIST(Z-score)). In reality, the variance of the Z should be larger than 1 due to dependencies among different oligomers (e.g., the pentamer next to AAAAT in a sequence can only be AAATA, AAATC, AAATG or AAATT, but not the remaining 1020 pentamers). Therefore, a P values calculated according the standard normal distribution is an underestimate. One empirical solution is to do control experiments comparing one set of background sequences with a different set of background sequences. The distribution of resulting Z-scores indicates chance variation and can be then used to determine a cutoff defining significance. Note that the absolute values of Z-scores increase with sample size. When the sample size is large small percent differences may result in sizeable Z-scores. Although statistically reasonable, these small differences must be considered when interpreting biological significance.When the sample size is extremely small (e.g., <10), one could use Fisher's exact test, which computes P values directly. Let $Q_f = N_f - O_f$ and $Q_b = N_b - O_b$, then:

$$P = \frac{N_f! N_b! (O_f + O_b)! (Q_f + Q_b)!}{(N_f + N_b)! O_f! O_b! Q_f! Q_b!}. \quad (6)$$

If O_f is large enough, and if O_b is from a theoretical background model or a background set that is much larger than the foreground set, a simplified version of a Z-score can be calculated

$$Z = \frac{O_f - (O_f \times \mu)}{\sqrt{O_f}}, \quad \text{where } \mu = B(O) = O_b/N_b. \quad (7)$$

If the foreground dataset is small, and if O_b is from a theoretical background model or a background set that is much larger than the foreground set, a P value can be calculated directly from the binomial distribution

$$P = \sum_{i=0}^{O_f} C_{N_f}^{O_f} p^i (1-p)^{N_f - i}, \quad \text{where } p = B(O) = O_b/N_b. \quad (8)$$

(4) The issue of multiple tests and false discovery rate: a P value indicates the probability that you would get the current result if the null hypothesis is true. In search for sequence motifs with length L, one usually needs to examine all possible 4^L L-mers. For each L-mer one statistical test is carried out so that 4^L tests would be conducted in total. Among such many tests, it would be not surprising to find some L-mers with significant P values just by chance. There are two kinds of solutions for this issue. The first one is to adjust the raw P values so that it can more accurately reflect statistical significance. A simple and conservative way of doing so is to multiply the raw P values by 4^L. This is known as a Bonferroni correction. The second is to ask how many L-mers would pass a certain threshold under null hypothesis and compare the number with the number of L-mers that actually pass the threshold. For example, suppose $L = 8$ and $4^L = 65{,}536$ and the cutoff of P to be 10^{-4}. Then there should be about 6.5 ($65{,}536 \times 10^{-4}$) 8-mers beyond the cutoff under the null hypothesis. If in reality, we got 1000 such 8-mers, we can simply claim that among these 1000, about 6.5 should be false positive and therefore the "false discovery rate" is about 0.65% (6.5/1000). Unlike the first method, we do not attempt to exclude the false positives but rather we get a notion of what portion of our selections could be false positives. A detailed discussion of the false discovery rate can be found in reference [45].

5.2. Comparative genomics and splicing

Functional elements in the genome are usually subject to negative selection and are thus more conserved among species than the non-functional sequences. This rule has been applied to studies on splicing. For example, Sorek and Ast discovered that the intronic flanking sequences of exons are more conserved between human and mouse for alternative exons than for constitutive exons. This result indicates that intronic splicing signals play a more important role in alternative splicing than in constitutive splicing [46]. Yeo et al. [47] employed human–mouse conservation to aid the discovery of alternatively spliced exons. These studies focus on the conservation of both exon and intron sequences. Conservation at the level of splicing events is also informative. Modrek and Lee [48] found that if an alternative exon is mostly skipped it tends not to be conserved among human, mouse, and rat. Subsequent works from Lee and colleagues provided further evidence that alternative splicing is likely to be associated with exon creation/loss in evolution [49,50].

An assessment of the possible conservation of sequence motifs can be carried out using two different strategies:

(1) The alignment based method: to conduct comparative genomic studies, one has to first identify orthologous genes among different species. This work can be done in two different ways. (i) As introduced in the data mining section, one can use HomoloGenes to identify orthologous gene pairs. The dataset can be found at ftp://ftp.ncbi.nih.gov/pub/HomoloGene/. Note that all homology information among all genes is provided here, and only the mutually best matches should be used for orthology [48]. (ii) The UCSC Genome Center http://genome.ucsc.edu/ provides alignment among several genome assemblies. The orthologous gene pairs can therefore be detected by aligning cDNAs of different species to the aligned genomes [47].

The second step is to align the orthologous gene pairs. If the data are downloaded from the UCSC Genome Center, the alignment is already done. If it is from HomoloGene, one has to perform the alignment. A number of programs are available for full-length gene alignment. Please refer to [51] for an overview and benchmark tests.

Finally, sequence characteristics and alignment statistics are analyzed. Several questions can be asked in the analyses. (i) What is the background level of conservation? This question can be addressed by examining apparently non-functional regions such as deep introns. (ii) What are the proportions of alignment in different regions? Obviously exons should be more conserved and better aligned than introns. But are there any intronic regions that tend to be more conserved than background [68]? (iii) Is a particular L-mer overrepresented in the aligned sequences compared to the non-aligned sequences? If an L-mer is functional, it should be more conserved and therefore more frequently aligned.

(2) The word-based methods: an alignment based method depend heavily on the accuracy of the alignment algorithm. It also assumes that the location as well as the sequence of the functional elements is conserved. While this might be generally true for many regulatory elements, it puts unnecessary constraints on the search for splicing signals. It is well known that one of the most important splicing motifs, the branch point, does not occur in a fixed position. It floats between -18 and -40 upstream of most exons. Alignment-based comparative approaches may miss such splicing signals. To avoid this problem, we developed a word-based approach that obviates the need for an alignment. Suppose we have N pairs of orthologous sequences between species A and B. For each L-mer we ask for the following counts: C_{11}, the number of pairs in which the L-mer occurs in both species; C_{10}, the number of pairs in which the L-mer occurs in species A but not B; C_{01}, the number of pairs in which the L-mer occurs in species B but not A; and C_{00}, the number of pairs in which the L-mer occurs in neither species. Obviously the four possibilities are mutually exclusive and they add up to N. The four numbers constitute a 2×2 table and a chi-square can be computed from the table as a measurement of association. The larger the chi-square is the stronger the association is between the occurrences of the L-mer in the two species and therefore more likely it is that the L-mer is functional. Detailed instructions for computing the chi-square can be obtained at http://home.ubalt.edu/ntsbarsh/Business-stat/otherapplets/Catego.htm. The Fisher exact test (formula 5) is a more accurate alternative to chi-square. One can substitute O_b, O_f, Q_b, and Q_f with the four counts and derive a P value directly from formula 5. This approach is suitable when the two species are distant and the non-functional conservation is low. We used this approach using orthologous human–mouse flanks (edges of introns) to help identify candidate splicing signals [52]. To define significant conservation, we used distributions of chi-squares derived from deep introns as the background distribution. Elemento et al. [53] independently developed a very similar approach in the studies of genome-wide conserved regulatory elements, where they used a hypergeometric test (a specialized Fisher exact test) to identify significant conservation.

5.3. Additional considerations in genomic studies on splicing

(1) Base composition bias in the genome: it has been known for 30 years that the vast majority of mammalian and avian genomes are not homogeneous in base

composition. These genomes can be divided into long segments (0.3–10 M bp) that vary greatly from one another but are internally homogeneous in C + G% [54]. A number of genomic characteristics correlate with C + G% such as surrounding gene density, gene length, intron length, expression level, codon usage bias, mutation rate, and recombination rate [55–62]. The evolution and function of this large-scale C + G% variation is still a controversial topic [63] and is beyond the scope of this paper. Here, we would like to point out the adverse influence such a variation could bring to bioinformatic studies on splicing.Splicing signals are known to be affected by the C + G% variation. For example, the 5′ splice site (5′ SS) consensus MAG|gtragt could have four different versions with A or C at position −3 and A or G at position +3. In high C + G% regions of the genome, these two positions tend to be C and G, respectively, whereas in low C + G% regions, the two positions both tend to be A [64]. Although this difference may very well be insignificant in function, it could bias the scoring of splice sites. Suppose one uses the base composition of the transcriptome as background model and the splice sites are scored using log-odds methods (formula 2,3). Since the transcriptome is rich in AT (~58%), the background frequencies of A or T (q_A or q_T in formula 2) would be higher. As a consequence, AT-rich 5′ SSs would generally receive lower scores compared to GC-rich 5′ SSs. This is unlikely to be a true reflection of splice site strength, since there is no reason to think AT-rich 5′ SSs are generally weaker than GC-rich 5′ SSs. However, when exons are divided into different groups according to different splice site strengths, GC-rich and AT-rich exons would tend to fall into different groups. The comparison between these groups would result in sequences that have bias in C + G% [16].

(2) Sequence composition bias in genes: 50% of genes have upstream CpG islands, which are thought to be important for their transcription [65]. CpG islands very often extend into the first exons, first introns or even more deeply into genes [66]. In searching for splicing signals, we compared internal non-coding exons in 5′-UTRs with pseudo exons and 5′-UTRs of intronless genes. Among the three datasets, the 5′-UTRs of intronless genes are closest to the CpG islands and have the highest CpG content. As a result, the putative ESEs that we defined to be overrepresented in internal exons versus the other two datasets have a relatively low CpG content, biasing against CpG-containing ESEs [13]. In another study, Fedorov et al. compared full-length intron-containing and intron-lacking genes to search for splicing signals. Again, since intronless genes are generally much shorter, a larger proportion of their sequence would overlap with CpG islands and therefore are expected to have a higher CpG content. In fact, the overrepresented oligomers in intronless genes showed an extremely high CpG content [67], a bias that could be unrelated to splicing.

References

[1] S.M. Berget, J. Biol. Chem. 270 (1995) 2411–2414.
[2] P. Senapathy, M.B. Shapiro, N.L. Harris, Methods Enzymol. 183 (1990) 252–278.
[3] H. Sun, L.A. Chasin, Mol. Cell. Biol. 20 (2000) 6414–6425.
[4] X.D. Fu, Cell 119 (2004) 736–738.
[5] L.R. Coulter, M.A. Landree, T.A. Cooper, Mol. Cell. Biol. 17 (1997) 2143–2150.
[6] H.X. Liu, S.L. Chew, L. Cartegni, M.Q. Zhang, A.R. Krainer, Mol. Cell. Biol. 20 (2000) 1063–1071.
[7] H.X. Liu, M. Zhang, A.R. Krainer, Genes Dev. 12 (1998) 1998–2012.
[8] Y. Cavaloc, C.F. Bourgeois, L. Kister, J. Stevenin, RNA 5 (1999) 468–483.
[9] R. Tacke, J.L. Manley, Curr. Opin. Cell Biol. 11 (1999) 358–362.
[10] T.D. Schaal, T. Maniatis, Mol. Cell. Biol. 19 (1999) 1705–1719.
[11] Z. Wang, M.E. Rolish, G. Yeo, V. Tung, M. Mawson, C.B. Burge, Cell 119 (2004) 831–845.
[12] L.P. Lim, C.B. Burge, Proc. Natl. Acad. Sci. USA 98 (2001) 11193–11198.
[13] X.H. Zhang, L.A. Chasin, Genes Dev. 18 (2004) 1241–1250.
[14] X.H. Zhang, K.A. Heller, I. Hefter, C.S. Leslie, L.A. Chasin, Genome Res. 13 (2003) 2637–2650.
[15] M. Sironi, G. Menozzi, L. Riva, R. Cagliani, G.P. Comi, N. Bresolin, R. Giorda, U. Pozzoli, Nucleic Acids Res. 32 (2004) 1783–1791.
[16] W.G. Fairbrother, R.F. Yeh, P.A. Sharp, C.B. Burge, Science 297 (2002) 1007–1013.
[17] S. Saxonov, I. Daizadeh, A. Fedorov, W. Gilbert, Nucleic Acids Res. 28 (2000) 185–190.
[18] M. Sakharkar, F. Passetti, J.E. de Souza, M. Long, S.J. de Souza, Nucleic Acids Res. 30 (2002) 191–194.
[19] V. Gopalan, T.W. Tan, B.T. Lee, S. Ranganathan, Nucleic Acids Res. 32 (2004) D59–D63.
[20] K.D. Pruitt, T. Tatusova, D.R. Maglott, Nucleic Acids Res. 33 (Database Issue) (2005) D501–D504.
[21] D.L. Wheeler et al., Nucleic Acids Res. 33 (Database Issue) (2005) D39–D45.
[22] Z. Zhang, S. Schwartz, L. Wagner, W. Miller, J. Comput. Biol. 7 (2000) 203–214.
[23] J. Usuka, W. Zhu, V. Brendel, Bioinformatics 16 (2000) 203–211.
[24] X. Huang, M.D. Adams, H. Zhou, A.R. Kerlavage, Genomics 46 (1997) 37–45.
[25] S.J. Wheelan, D.M. Church, J.M. Ostell, Genome Res. 11 (2001) 1952–1957.
[26] W.J. Kent, Genome Res. 12 (2002) 656–664.
[27] L. Florea, G. Hartzell, Z. Zhang, G.M. Rubin, W. Miller, Genome Res. 8 (1998) 967–974.
[28] N. Volfovsky, B.J. Haas, S.L. Salzberg, Genome Res. 13 (2003) 1216–1221.
[29] J.E. Stajich et al., Genome Res. 12 (2002) 1611–1618.
[30] P. Rice, I. Longden, A. Bleasby, Trends Genet. 16 (2000) 276–277.
[31] F. Hsu, T.H. Pringle, R.M. Kuhn, D. Karolchik, M. Diekhans, D. Haussler, W.J. Kent, Nucleic Acids Res. 33 (2005) D454–D458.
[32] T. Hubbard et al., Nucleic Acids Res. 33 (2005) D447–D453.
[33] M.B. Shapiro, P. Senapathy, Nucleic Acids Res. 15 (1987) 7155–7174.
[34] T.D. Schneider, J. Theor. Biol. 189 (1997) 427–441.
[35] C. Burge, S. Karlin, J. Mol. Biol. 268 (1997) 78–94.
[36] M. Pertea, X. Lin, S.L. Salzberg, Nucleic Acids Res. 29 (2001) 1185–1190.

[37] S. Brunak, J. Engelbrecht, S. Knudsen, J. Mol. Biol. 220 (1991) 49–65.
[38] E. Eden, S. Brunak, Nucleic Acids Res. 32 (2004) 1131–1142.
[39] G. Yeo, C.B. Burge, J. Comput. Biol. 11 (2004) 377–394.
[40] V.N. Vapnik, Statistical Learning Theory, Springer, 1998.
[41] G. Dror, R. Sorek, R. Shamir, Bioinformatics 21 (2005) 897–901.
[42] T. Joachims, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, 1999.
[43] P.M. Sharp, W.H. Li, Nucleic Acids Res. 15 (1987) 1281–1295.
[44] J.W. Fickett, C.S. Tung, Nucleic Acids Res. 20 (1992) 6441–6450.
[45] J.D. Storey, R. Tibshirani, Proc. Natl. Acad. Sci. USA 100 (2003) 9440–9445.
[46] R. Sorek, G. Ast, Genome Res. 13 (2003) 1631–1637.
[47] G.W. Yeo, E. Van Nostrand, D. Holste, T. Poggio, C.B. Burge, Proc. Natl. Acad. Sci. USA 102 (2005) 2850–2855.
[48] B. Modrek, C.J. Lee, Nat. Genet. 34 (2003) 177–180.
[49] A. Resch, Y. Xing, A. Alekseyenko, B. Modrek, C. Lee, Nucleic Acids Res. 32 (2004) 1261–1269.
[50] Y. Xing, C.J. Lee, Trends Genet. 20 (2004) 472–475.
[51] D. Pollard, C. Bergman, J. Stoye, S. Celniker, M. Eisen, BMC Bioinformatics 5 (2004) 6.
[52] X.H-F. Zhang, C.S. Leslie, L.A. Chasin, Genome Res. 15 (2005) 768–779.
[53] O. Elemento, S. Tavazoie, Genome Biol. 6 (2005) R18.
[54] G. Bernardi, Gene 259 (2000) 31–43.
[55] E.S. Lander et al., Nature 409 (2001) 860–921.
[56] C.I. Castillo-Davis, S.L. Mekhedov, D.L. Hartl, E.V. Koonin, F.A. Kondrashov, Nat. Genet. 31 (2002) 415–418.
[57] G. D'Onofrio, Gene 300 (2002) 155–160.
[58] L. Duret, D. Mouchiroud, C. Gautier, J. Mol. Evol. 40 (1995) 308–317.
[59] K. Gardiner, Trends Genet. 12 (1996) 519–524.
[60] S. Karlin, J. Mrazek, J. Mol. Biol. 262 (1996) 459–472.
[61] R. Versteeg, B.D.C. van Schaik, M.F. van Batenburg, M. Roos, R. Monajemi, H. Caron, H.J. Bussemaker, A.H.C. van Kampen, Genome Res. 13 (2003) 1998–2004.
[62] S. Zoubak, O. Clay, G. Bernardi, Gene 174 (1996) 95–102.
[63] A. Eyre-Walker, L.D. Hurst, Nat. Rev. Genet. 2 (2001) 549–555.
[64] F. Clark, T.A. Thanaraj, Hum. Mol. Genet. 11 (2002) 451–464.
[65] M. Gardiner-Garden, M. Frommer, J. Mol. Endocrinol. 12 (1994) 365–382.
[66] R.V. Davuluri, I. Grosse, M.Q. Zhang, Nat. Genet. 29 (2001) 412–417.
[67] A. Fedorov, S. Saxonov, L. Fedorova, I. Daizadeh, Nucleic Acids Res. 29 (2001) 1464–1469.
[68] X. Xie, J. Lu, E.J. Kulbokas, T.R. Golub, V. Lindblad-Toh, E.S. Lander, M. Kellis, Nature 434 (2005) 338–345.