

A Machine Learning Approach to Test Data Generation: A Case Study in Evaluation of Gene Finders

Henning Christiansen¹ and Christina Mackeprang Dahmcke²

^{1,2}Computer Science Dept. & ²Life Sciences Dept.
Roskilde University, P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: {henning, chmada}@ruc.dk

Abstract. Programs for gene prediction in computational biology are examples of systems for which the acquisition of authentic test data is difficult as these require years of extensive research. This has led to test methods based on semiartificially produced test data, often produced by *ad hoc* techniques complemented by statistical models such as Hidden Markov Models (HMM). The quality of such a test method depends on how well the test data reflect the regularities in known data and how well they generalize these regularities. So far only very simplified and generalized, artificial datasets have been tested, and a more thorough statistical foundation is required.

We propose to use logic-statistical modelling methods for machine-learning for analyzing existing and manually marked up data, integrated with the generation of new, artificial data. More specifically, we suggest to use the PRISM system developed by Sato and Kameya. Based on logic programming extended with random variables and parameter learning, PRISM appears as a powerful modelling environment, which subsumes HMMs and a wide range of other methods, all embedded in a declarative language. We illustrate these principles here, showing parts of a model under development for genetic sequences and indicate firstinitial experiments producing test data for evaluation of existing gene finders, exemplified by GENSCAN. The advantage of the approach is the relative ease and flexibility with which these probabilistic models can be developed.

1 Introduction

A computer program calculating a well-defined mathematical function is either correct or incorrect, and testing is a systematic process aiming to prove the software incorrect. One sort of test is black-box (or external) testing, which consists of running selected data through the program and comparing observed and expected results; established methods exist for designing test data suites that increase the chance of finding errors [14]. Systems for information retrieval and extraction, on the other hand, have no such simple correctness criteria and are evaluated by relative measurements such as precision and recall in manually marked-up test data, e.g., the text corpora provided by the TREC and MUC conferences [2, 1].

Gene finding programs, whose job is related to information extraction, aim to predict the total number and locations of genes in sequences of up to 3 billion letters. Here the situation is more subtle as the production of marked-up test sequences may require years of research. In addition to this, it may be a research project in itself, to verify that a new gene suggested by a gene finder is correct.

Following the completion and release of the human genome sequence, a wide range of gene finder programs have been published. The tools differ in which kind of knowledge they integrate in gene modelling; from the fundamental *ab initio* approach, like GENSCAN [4], where generalized knowledge of genes is used, to the more opportunistic models, like GeneWise [3] and GenomeScan [19], where already known sequences of genes, EST's and proteins are used for finding genes by sequence similarity. One major problem with the existing gene prediction tools seems to be the lack of appropriate methods to evaluate their accuracy. The two major groups of human genome sequencing, Celera and Ensemble both predicted about 30,000 genes in the human genome [8, 7], but a comparison of the predicted genes [12], revealed a very little overlap. This also seems to be the problem with other gene prediction tools. Which tools are more correct, is not easy to conclude, since most of the new predicted genes are not possible to classify as either correct genes that have not been found yet, or false predicted genes. This also applies to the underlying layer, of telling wrong exons (false positives) from those that have not been found yet. Another problem with the currently used training sets is that they usually consists of relatively simple genes, like the leading Buset/Guigó training set [5], with generalized features like containing few exons, all having tataboxes, having no overlapping genes and so on. Therefore most genefinders get very good at finding simple structured genes, and poor at finding the complex ones. The evaluation of genefinders does also have a problem, with sometimes large overlaps between the datasets used for training and the datasets used for estimating the accuracy of the gene finders [11].

To partly overcome these problems, test methods have been proposed based on semiartificially produced test data, typically by a combination of *ad hoc* techniques and statistical models such as Hidden Markov Models (HMM). The quality of such a test method depends on how well the test data reflect the regularities of known data and how well they generalize these regularities. So far only very simplified, artificial datasets have been tested, and a more thorough statistical foundation is required. A semiartificial dataset [11] was generated to overcome these problems, in which a large set of annotated genes were placed randomly in an artificial, randomly generated intergenic background. This background sequence, however, is constructed in rather simplified, using a Markov Model to generate GC content of 38%. The present work is intended as a further step in this direction, demonstrating how the logic-statistical machine-learning system PRISM [15, 16] can be used for the development of more sophisticated and reliable models. Based on logic programming extended with random variables and parameter learning, PRISM appears as a powerful modelling environment, which subsumes HMMs and a wide range of other methods, all embedded in a declarative language.

We illustrate these principles here, showing parts of a model under development in PRISM for genetic sequences and indicate first experiments producing test data for evaluation of existing gene finders, exemplified by GENSCAN. The advantage of the approach, that we can demonstrate, is the relative ease and flexibility with which these probabilistic models can be developed, while a claim that the approach may lead to biologically more well-founded models needs to be supported by more extensive testing.

PRISM embeds a both powerful and very flexible modelling language, which makes it possible to embed biological knowledge about genome sequences. A model in PRISM is parameterized by probabilities for random variables, and training the model with known data generates estimates for these probabilities. Using the same model, these probabilities can be used for generating artificial data that mimic in a faithful way sufficiently many properties of the authentic data. The generated data are marked up with information of where the model decided to put in genes, and a given gene finder can be evaluated in a precise way, by comparing its proposals for genes with those of the PRISM model (which, here, would play the role of “true” ones). We mention briefly an experiment testing the GENSCAN system.

Section 2 gives a brief introduction to PRISM, and section 3 presents fragments a PRISM model for genomic sequences, which is still under development. This model is used for conduction a first, simple test of GENSCAN described in section 4; section 5 compares with related work, and the final section 6 discusses perspectives of this work and possible improvements.

2 Logic-Statistical Modelling in PRISM

PRISM [15, 16] is a recent logic-statistical modelling system developed by T. Sato and Y. Kameya. The system is an extension to the Prolog programming language, more specifically the B-Prolog system [20], with discrete random variables called *multi-valued random switches*, abbreviated msw’s. The system is sound with respect to a probabilistic least Herbrand model semantics, provided that the different msw’s are independent (see the references above for details). As an example, we can declare a class of msw’s by selecting one out of a four different letters at random as follows.

```
values( nextLetter(_), "ACGT").
```

Recall that a string such as "ACGT" is a list of character values, and in general the `values` directive describes a finite list of possible values. The term `nextLetter(_)` includes a logical variable which means that, for any possible value which is plugged in for that variable, there exists an msw; e.g., `nextLetter(t1)`, `nextLetter(t2)`. The following fragment shows how an msw typically is used within a rule; the arrow-semicolon is Prolog’s generalized if-then-else construction; the notation `0’ char` indicates the relevant character code.

```
msw( nextLetter(t1), Letter),
```

```
(Letter = 0'A -> ...
; Letter = 0'C -> ...
; Letter = 0'G -> ...
; Letter = 0'T -> ... )
```

The dots indicate the different actions taken for the different outcomes. In this way, it is straightforward to write HMM's as PRISM programs (additional msw's may govern state transitions), but the advantage of PRISM is that the full power of Prolog can be used for the modelling process, including recursion. Other models such as discrete Bayesian networks, Stochastic Context-Free Grammars [6], and Hierarchical HMM's [10] can also be described in straightforward ways, and PRISM can be seen as a high-level tool for defining advanced and directly executable probabilistic models, as we do in the present paper. Conditional probabilities are not supported but may to a large extent be simulated by generic msw names. So for example, $P(a = x|b = y)$ can be simulated by the code `msw(b,Y), msw(a(Y),X)`.

We will show a little example, which also also illustrates the modular structure we apply in our sequence model in the following section. For the example, we imagine sequences comprising three types of substrings `t1`, `t2`, `t3`, put together in random order and each being sequences of the letters `ACGT` but with different relative frequencies for each type. An annotated sequence may be described by a goal as follows, having the annotations in a separate argument, and numbering the sequence letters starting from 1.

```
sequence("AAAACGCGCG", [t1(1,4),t2(5,10)])
```

The annotation is a sequence of descriptors for each subsequence. The following msw's govern the composition of subsequence of different types.

```
values( nextSubstringType(_), [t1,t2,t3] ).
values( continueSeq, [yes,no] ).
```

The argument to the first one describes the type of a previous subsequence; the last one determines no. of subsequence, the higher probability for `yes`, the longer sequence.¹ The following rule describes the composition at subsequence level. An arbitrary "previous type" called `start` is assumed for the first subsequence, and an extra argument which keeps track of position in the string is added; evidently this is a HMM with 5 states, the last one being the implicit final state.

```
seq(Seq,D):-seq(start,Seq,D,1).
```

```
seq(PrevType,Seq,[D1|DMore],N):-
  msw(nextSubstringType(PrevType), SubT),
  unfoldSubSeq(SubT,D1,Seq,SeqMore,N,M),
  msw(continueSeq, YesNo),
  (YesNo=no -> SeqMore=[], DMore = [])
  ; Mplus1 is M+1, seq(SubT,SeqMore,DMore,Mplus1)).
```

¹ As is well-known from HMMs this gives a geometric distribution of lengths.

For each type T , we need to add an appropriate clause `unfoldSubSeq(T, ...)` :- ... To finish this example, we assume identical definition (but individual probabilities) for each type.

```
seqType(Type, [L|SeqMore], SeqRest, N, M) :-
    msw(nextLetter(Type), L),
    msw(continueSub(Type), YesNo),
    (YesNo=no -> SeqMore=SeqRest, N=M
     ; Nplus1 is N+1, seqType(Type, SeqMore, SeqRest, Nplus1, M)).
```

Notice that we assumed an additional class of `msw`'s to govern the length of each subtype (so hits little model resembles a 2-level Hierarchical HMM). Such a program can be used in two ways in the PRISM system. In case the probabilities are given in some way or another, the program can be used to generate samples of annotated sequences by a query `?-sample(sequence(S,A))` where PRISM executes `msw` by using a random number generator adjusted to the given probabilities. PRISM can also run in *learner mode*, which means that the program is presented with a number of observed goals, and hereafterby set to calculates the distribution probabilities that provides the highest likelihood for explaining the observed data. PRISM uses some quite advanced algorithms and data structures in order to do this in an efficient way; these topics are outside the scope of this paper and we refer to [15, 16]. Training with a single observation (somewhat artificial) being the short annotated sequence above, we get for `t1` prob. 1.0 for A and for `t2` prob. 0.5 for each of C and G. After the training phase, sampling can be used to create *similar* sequences, where the applied notion of similarity is implied by the program, i.e., the model for sequence structure that we designed above. PRISM runs in 64-bit architectures so it can address quite large amount of storage. The current version (1.9, 2006) can, with programs as the one shown, easily handle data sets of around 10 sequences of 100,000 letters. For sequence analysis, this is too small but current research indicated that this will change in near future.² The advantages of PRISM are obvious: Models can be written in a powerful declarative language suited for symbolic computation, and one and the same executable specification can be used for parameter learning as well as for data generation.

3 A PRISM model of genomic sequences

The example above illustrates the central structure of our approach to test data generation for gene finders. Ideally, we build our own model of sequences in the incarnation of a PRISM program, and have access to a large set of perfectly annotated genome sequences which we can use to train out model. With such generated probabilities, we can generate as many similar samples as needed.

A proposal for such a model is currently under development, and here we present the fragment describing the intergenetic regions of genome sequences.

² At present, sequences are represented as Prolog lists; we intend to add a specialized representation with maximum one byte per letter by using B-Prolog's C interface.

For testing gene finders, it is important with a reliable model of these regions as well, to check that no false genes are claimed here. This short paper does not leave room toFor a review all the different sources Ofon the biological literature issues that have inspired to this model, nor to argue for its feasibility from a biological point of view; see [9, 11] for an overview. We want to emphasize that this is a first proposal for a model, which is likely to be revised when more experience is gained. Our reason to show it, is to indicate the ease with which such a model can be developed in PRISM.

A sequence is conceived by two overlying structures. The first level concerns the distribution of *GC islands* which are long stretches with significantly higher frequencies for the two letters; GC islands are important as their presence often indicates subsequences containing genes; remaining regions of the sequences are called *GC sparse*. At a second level, the sequence is considered to consist of mostly arbitrary letters (white noise) interspersed with so-called repeat strings, which are either subsequences of a known catalogue of named strings, simple repeats (e.g., (ACCT)*n* meaning a repetition of the indicated pattern), and low-complexity strings such as CT-rich which is an arbitrary combination of these letters (however, with a recognizable central pattern). A marked up sequence is represented by structure as follows.

`sequence(sequence-of-ACGT, GC-islands, repeats)`

The first component is the bare sequence of letters, the second indicates positions of GC islands and GC sparse, and the third one describes the repeats in the sequence, including their location. The relationship between these two levels is somewhat tricky as a given repeater substrings can appear in a GC island, in GC sparse regions, as well as overlap both; however, there is of course a dependency in the sense that, say, e.g. a repeat with many G's and C's will appear more often in GC islands. Since GC islands tend to be much larger than repeats, they are considered the top level structure. In our model, the lengths of GC islands and GC sparse regions are governed by random variables that decides whether or not a GC island (or GC sparse) should continue or stop, thus giving a geometric distribution as explained in section 2. Most other random variables (and their probabilities) are given in two versions, one for GC island and one for GC sparse.

This two-level structure can be implemented by hiding the management of GC islands in an abstract data type, implemented as an alternative version of PRISM's `msw` predicate with the following parameters.

`msw(Random-var-name, value, GC-Islands, position)`

If a random variable, say `x(a)` is referred to at position 5000 in a sequence, we call `msw(x(a), V, ..., 5000)` and the implementation chooses, depending on how position 5000 is classified, the relevant of `msw(x(gcIsland,a),V)` and `msw(x(gcSparse,a),V)`; in addition, the extended version of `msw` includes the machinery that describes GC islands in terms of other random variables as explained. With this GC sensitive version of `msw`, we can describe the model of intergenetic sequences as a composition of arbitrary letters and repeats according to the pattern indicated in section 2. Abstracting away a few implementation details, it can be presented as follows; notice that the choice of repeat depends

on the previous one, as biologists indicate that certain repeats tend to come in groups; the arbitrary letter sequences between repeats depend on those as well since some repeats has a tendency to close to each other.

```
seq(Seq,GCs,Reps):- seq(dummy,Seq,GCs,Reps,1).
seq(Previous,Seq1,GCs,Reps,N1):-
  msw(nextRepType(Previous),RepType,N1),
  letters(Previous,RepType,Seq1,Seq2,GCs,N1,N2), N3 is N2+1,
  (RepType=noMore -> Seq=[], Reps=[])
  ; RepType=namedRepeat ->
    namedRep(Details,Seq,Seq1,GCs,N3,N4), Reps=[named(Details)|Reps1],
    seq(RepType,Seq1,GCs,Reps1,N4)
  ; RepType=simpleRepeat ->
    simpleRepeat(...), Reps=..., seq(...)
  ; RepType=lowComplexRep ->
    lowComplexRepeat(...), Reps=..., seq(...).
```

Definitions of predicates `namedRep`, `simpleRepeat`, `lowComplexRepeat` describes the detailed structure of the different sort of repeater; for efficiency, the sequence is given as a difference list, i.e., as two arguments. The two latter have their own random variables to define which substring is repeated and how many times `namedRep` has access to a catalogue of known repeat string strings and has random variables concerning which portion of the given known string is used and its direction: it may appear as a plain copy, reversed, complement (i.e., with letters interchanged $A \leftrightarrow T$, $C \leftrightarrow G$), or reverse complement. These predicates are defined in relatively straightforward ways, although a few technicalities are needed in order to handle mutations, which here means that a few noise letters have been inserted or deleted in the sequences. Again, we used random variables to govern this and to control which noise letters are chosen.

Named repeats were described in a simplified version in the marked sequences we had available, as the exact portion used as well as its direction were not given, only its name. To cope with this, we used a preprocessing to determine one unique best match of portion and direction and added it to the annotations, where “best” is defined by a few heuristics and a lowest penalty scheme. In addition, our best match algorithm also adds best proposals for where and how mutations are added. The use of preprocessing to provide unique best proposals for lacking information in annotations is essential for this form of sequence analysis, because nondeterminism in a PRISM model can introduce backtracking which may be very problematic due to the size of the data being analyzed.

4 Experimental Results: Evaluation of GENSCAN

A significant drop in accuracy was observed [11] when testing GENSCAN on a semi artificial training set. This could be the first step in giving an accurate measure of gene finders, but due to the simplicity of the data set, some degree of uncertainty is evident. We demonstrate a test of GENSCAN, using a more statistical founded data set, showing a relative high error rate. A minor number

of sequences were generated, by sampling artificial intergenic sequence from the learned probabilities, and hereafter inserting real genes in between. The intergenic regions range in sizes from 6,000 to 100,000, and three genes was placed in between these, resulting in a 200,000 long sequence. We report the outcome of predicted exons in the intergenetic regions, as this is the main area of interest in this paper. Therefore only false positives are reported. For this we use the evaluation terms; specificity and sensitivity, as described by [5]. GENSCAN predicted 17 genes in the 200,000 bp long sequence, where 10 of the predicted genes were placed in the intergenetic regions. Further more 3 predicted genes overlapped the intergenetic region. These results seem to indicate that GENSCAN overpredicts, when faced with intergenic sequences containing repeated sequences and GC islands. Since this test is only preliminary and an example of the possibilities of this model, these results are not adequate as background for a thorough assessment of GENSCANs efficiency. For this a much larger training set is needed, for the probabilities of the model to be realistic. A more elaborative test could also compare GENSCAN results, when performed on data with simple intergenic sequence, like [Guigo] carried out, and composite sequence data produced by this novel model.

5 Comparison with Related Work

The first attempt to overcome the problem of not having large, precisely marked data sets, for evaluation of gene finders, was made by embedding random genes in a random sequence approximating intergenic sequence [11]. 42 sequences with accurate gene annotation were used, having an average length of 177160, containing 4.1 genes with an average of 21 exons and 40% GC content. Knowledge about repeated sequences, the existence of GC islands and pseudogenes are not taken into account, and only a simple average GC frequency of 38% was generated by using a fifth order HMM. Not surprisingly, the test of GENSCAN on these sequences did show a lower rate of correct predictions. This is logical since ab initio programs like GENSCAN, recognizes genes by small sequences like tata boxes, splicesites etc. and therefore are dependent on possible sequence variationsthat will inform it about possibly being close to something relevant; e.g. GC islands. Our approach is to concentrate on including more knowledge about intergenic regions in the model, incorporating GC content variations and intergenic repeats. Our training set is generated from the chromosome 17 sequence (NT_0101718.15) and gene annotations are adopted from NCBI. Six intergenic regions between gene locations are used for training the model, together with a list of specified positions of repeated sequences and gc islands. Repeatmasker [18] was used to find repeated sequences in these sequences, using standard masking options, and RepBase [13] was used for generating a repeater catalogue. Predefined GC island positions were adopted from the NCBI website [<http://www.ncbi.nlm.nih.gov/>].

One of the reasons for choosing chromosome 17 is due to the fact that it is a rather well studied chromosome, and the relative gene density is high. This could mean that most genes are accounted for, and the intergenic sequence therefore

is more accurate to use. This training set is to be considered as an example of what the model can be trained on. Therefore the evaluation of GENSCAN will not be an absolute assessment, but we believe, this new method for creating a testing set, is a new step in the way of finding a more precise evaluation of gene finders.

6 Perspectives and Future Work

A further development of the model presented in this paper, will be a more comprehensive model, which takes more specialized features into account, together with the inclusion of genes. The major issue for a successful training of the model is a larger and better marked up set of sequences; At this point perfectly marked up genomic sequences, which include both genetic and intergenetic regions remain sparse and difficult to access.

The use of artificial data sets is a subject under debate. As discussed by [17] the use of artificial datasets have not been used much, since it is very difficult to reflect the whole complexity of real data sets. But how many details must be included in a model for artificial data generation? As [17] state, the goal must be to include those characteristics of real data which may affect the performance of real data sets. Multivariable dependencies are also very important in terms of reflecting real data, and can often be very difficult to reproduce. PRISM however, seems to be a rather powerful modelling tool, which also are capable of reflecting multivariable dependencies, by the inclusion of one msw in another (see section 2). Even if these characteristics are accounted for, the ideal data mining method for one dataset might not be ideal for another set. Further more, undiscovered correlations could influence the outcome of a given model. Our model only accounts for those characteristics verified by a biological expert, and therefore undiscovered multivariable dependencies are naturally not included. Another important issue, as mentioned, is the balance between including enough information in the model to make it realistic, and including too much, which results in overfitting. This will also be very important to have in mind, when expanding the model. Another important issue is the balance between overfitting and generalization, and the granularity of the model; while a fine-grained model is well suited when large training sets are available but likely subject of over-fitting in case a few training data, it is the other way round for a coarse model which may be the best out of few training data and too little out of large sets. Only testing and analyze indicate the right level. Finally our experiences indicates that the elegance of logic programming is not incompatible with the processing of large data sets. With the present technology we can handle around one million letters, which is clearly to little, although with reasonable execution times are reasonable, and Pplans for extensions of PRISM includes specialized representations for sequences (as opposed to list structures!) as well as adaptation to computer clusters.

Acknowledgement: This work is supported by the CONTROL project, funded by Danish Natural Science Research Council.

References

1. Message Understanding Conferences (MUC). http://www-nlpir.nist.gov/related_projects/muc/.
2. Text REtrieval Conference (TREC). <http://trec.nist.gov/>.
3. Ewan Birney, Michele Clamp, and Richard Durbin. GeneWise and Genomewise. *Genome Res.*, 14(5):988–995, 2004.
4. Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.
5. Moises Burset and Roderic Guigó. Evaluation of Gene Structure Prediction Programs. *Genomics*, 34(3):353–367, 1996.
6. Eugene Charniak. *Statistical Language Learning*. The MIT Press, 1993.
7. E.S. Lander *et al* (> 300 authors). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–92, 2001.
8. J. Craig Venter *et al* (> 300 authors). The Sequence of the Human Genome. *Science*, 291(5507):1304–1351, 2001.
9. James W. Fickett. Finding genes by computer: the state of the art. *Trends in Genetics*, 12(8):316–320, 1996.
10. Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.
11. Roderic Guigó, Pankaj Agarwal, Josep F. Abril, Moises Burset, and James W. Fickett. An Assessment of Gene Prediction Accuracy in Large DNA Sequences. *Genome Res.*, 10(10):1631–1642, 2000.
12. John B. Hogenesch, Keith A. Ching, Serge Batalov, Andrew I. Su, John R. Walker, Yingyao Zhou, Steve A. Kay, Peter G. Schultz, and Michael P. Cooke. A comparison of the Celera and Ensembl predicted gene sets reveals little overlap in novel genes. *Cell*, 106(4):413–415, 2001.
13. J. Jurka, V.V. Kapitonov, A. Pavlicek, P. Klonowski, O. Kohany, and J. Walichiewicz. Repbase Update, a database of eukaryotic repetitive elements. *Cytogenetic and Genome Research*, 110(1-4):462–467, 2005.
14. Glenford J. Myers, Corey Sandler (Revised by), Tom Badgett (Revised by), and Todd M. Thomas (Revised by). *The Art of Software Testing, 2nd Edition*. Wiley, 2004.
15. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res. (JAIR)*, 15:391–454, 2001.
16. Taisuke Sato and Yoshitaka Kameya. Statistical abduction with tabulation. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2408 of *Lecture Notes in Computer Science*, pages 567–587. Springer, 2002.
17. Paul D. Scott and Elwood Wilkins. Evaluating data mining procedures: techniques for generating artificial data sets. *Information & Software Technology*, 41(9):579–587, 1999.
18. A.F.A. Smit, R. Hubley, and P. Green. Repeatmasker web site, 2003. <http://repeatmasker.org>.
19. Ru-Fang Yeh, Lee P. Lim, and Christopher B. Burge. Computational Inference of Homologous Gene Structures in the Human Genome. *Genome Res.*, 11(5):803–816, 2001.
20. Neng-Fa Zhou. B-Prolog web site, 1994–2006. <http://www.probp.com/>.