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

Henning Christiansen[1] and Christina Mackeprang Dahmcke[2]

[1] Research group PLIS: Programming, Logic and Intelligent Systems
Department of Communication, Business and Information Technologies
Roskilde University, P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: henning@ruc.dk
[2] Department of Science, Systems and Models
Roskilde University, P.O.Box 260,
DK-4000 Roskilde, Denmark
E-mail: 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 lead 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 data sets 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 first initial experiments producing test data for evaluation of existing gene finders, exemplified by GENSCAN, HMMGene and genemark.hmm.

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

Proceedings of MLDM'2007,
Machine Learning and Data Mining in Pattern Recognition,
Leipzig, Germany, July 18- 20, 2007.
P. Perner (ed.)
Lecture Notes in Artificial Intelligence 4571.
Copyrighted by Springer, 2007

that increase the chance of finding errors [1]. 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, 3].

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 a 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 [5] and GenomeScan [6], 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 [7, 8], but a comparison of the predicted genes [9], 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 Burset/Guigó training set [10], with generalized features like containing few exons, all having tataboxes, having no overlapping genes and so on. Therefore most gene finders get very good at finding simple structured genes, and poor at finding the complex ones. The evaluation of gene finders does also have a problem, with sometimes large overlaps between the data sets used for training and the data sets used for estimating the accuracy of the gene finders [11].

To partly overcome these problems, test methods have been proposes 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 data sets have been tested, and a more thorough statistical foundation is required. A semiartificial data set [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, introduced by other authors [12, 13], 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 [4], HMMGene [14] and genemark.hmm [15]. 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 describe an experiment testing the three gene finders.

Section 2 gives a brief introduction to PRISM, and section 3 presents fragments a PRISM model for genomic sequences, which is still under development. Section 4 compares briefly with related work, before we show how our model is used for testing three selected gene finders. Sections 5 and 6 describe and evaluate the tests and compare the different gene finders. Section 7 discusses the quality of the test data generated by our model, and section 8 gives perspective and outline plans for future work.

2 Logic-Statistical Modelling in PRISM

By a modeling paradigm based on logic programs, we take a step upwards in formal expressibility compared with those models traditionally used in sequence analysis. Where HMMs are based on regular languages, SCFGs (Stochastic CFGs) as their name indicates on context-free language, PRISM can specify any Turing computable language. In practice, and to stay within computationally tractable cases, we need to be more modest and employ the flexibility of a general programming language to combine existing models and to introduce auxiliary data structures whenever convenient.

PRISM [12, 13] represents a logic-statistical modelling system that combines the logic programming language Prolog with probabilistic choice and machine learning, and is implemented as an extension to the B-Prolog language [16]. It includes 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 for 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 codes, 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 substituted for it, 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). Other models such as discrete Baysian networks, Stochastic Context-Free Grammars [17], and Hierarchical HMM's [18] 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 can be represented using generic msw names. For example, $P(a = x|b = y)$ is indicated by the code `msw(b,Y), msw(a(Y),X)`.

We show an illustrative example of a model in PRISM, which also illustrates the modular structure we apply in our sequence model. We imagine sequences comprising three types of subsequences `t1`, `t2`, `t3`, put together in random order. Each type represent sequences of the letters `ACGT` but with different relative frequencies for each type. An annotated sequence can be described by a goal as follows, having the annotations in a separate argument, and numbering the 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 number of subsequence, the higher probability for `yes`, the longer sequence.[3] 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 the 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),
   subSeq(SubT,D1,Seq,SeqMore,N,M),
   msw(continueSeq, YesNo),
   (YesNo=no -> SeqMore=[], DMore = []
    ; Mplus1 is M+1,
      seq(SubT,SeqMore,DMore,Mplus1)).
```

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

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

Notice that we assumed an additional class of msw's to govern the length of each subtype (so this 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 generate samples of annotated sequences by a query `?-sample(sequence(S,A))` where PRISM executes the `msw`s 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, from which it calculates the distribution of 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 [12, 13]. 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 given by the program, i.e., the model for sequence structure that we designed above. PRISM runs also on 64-bit architectures so it can address quite large amounts of storage. The version used for out tests (1.9, 2006) can, with programs as the one shown, easily handle data sets of around 10 sequences of 100,000 letters. For eukaryote

[3] As is well-known, this gives a geometric distribution of lengths.

sequence analysis in general, this is a rather small number, and current research aim at increasing this number drastically.[4]

3 A PRISM model of genomic sequences

The example above illustrates the overall structure of our approach to test data generation for gene finders. A model of full genome sequences is under consideration, but for the present task of testing gene finders we have limited to a model of the intergenic regions. We argue that a detailed and biologically faithful model is essential also for these regions, as they contain patterns and fragments that may descend from obsolete functional regions, and thus may confuse a gene finder; this is, in fact, what our experiments reported below seem to indicate.

A sequence is characterized in our model by two intertwined 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 (coloured 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 a combination of these letters. A marked up sequence is represented by structure of the following form.

`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. The relationship between these two levels is somewhat tricky as a given repeat substrings can appear in a GC island, in a GC sparse region, or overlap both; however, there is a dependency in the sense that a repeat (or section thereof) with many G's and C's tend to appear more often in GC islands. Since GC islands tend to be much larger that repeats, they are considered the top level structure. The lengths of GC islands and GC sparse regions are determined by empirically determined minimum lengths combined with random variables that decide whether to continue or stop, thus giving geometric distributions upon the minimum lengths. Other random variables are given in two versions, one for GC island and one for GC sparse.

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

`msw(`*random-var*`,` *value*`,` *GC-islands*`,` *position*`)`

[4] E.g., using an array representation for sequences instead of Prolog lists as in the current version, may reduce storage for each letter from 24 bytes (on a 64 bit architecture) to 2 bits!

If a variable, say `x(a)` is referred to at position 5000 in a sequence, we call `msw(x(a), V, ···, 5000)` and the implementation choses, 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. Abstracting away a few implementation details, the overall composition of intergenic sequences 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 letter sequences between repeats depend on those as well since some repeats have a tendency to come close to each other. These letter sequences are a kind of coloured noise, where the colours indicate the different letter probabilities in GC islands vs. GC sparse regions.

```
seq(Seq,GCs,Reps):-
    seq(dummy,Seq,GCs,Reps,1).

seq(Prev,Seq1,GCs,Reps,N1):-
    msw(nextRepType(Prev),RepType,GCs,N1),
    letters(Previous,RepType,
                Seq1,Seq2,GCs,N1,N2),
    N2i is N2+1,
    (RepType=noMore -> Seq=[], Reps=[]
    ; RepType=namedRepeat ->
        namedRep(Details,Seq2,Seq3,GCs,N2i,N3),
        Reps=[named(Details)|Reps1],
        seq(RepType,Seq3,GCs,Reps1,N3)
    ; RepType=simpleRepeat ->
        simpleRep(···), Reps=···, seq(···)
    ; RepType=lowComplexRep ->
        lowCmplxRep(···), Reps=···, seq(···)).
```

Predicates `namedRep`, `simpleRep`, `lowCmplxRep` describe the detailed structure of the different sort of repeats. They apply their own random variables to govern which substring is repeated; `namedRep` has access to an 11Mbyte catalogue of known repeat string strings and employs random variables to determine which portion of the named string is used and its direction: it may appear as a plain copy, reversed, complement (i.e., with letters interchanged A↔T, C↔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 up 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 preprocess 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 adds best proposals for where and how mutations have occurred. The use of preprocessing to provide unique best proposals for lacking information in annotations appears to be essential for this form of sequence analysis, as nondeterminism in a PRISM model can introduce backtracking which may be very problematic due to the size of the data being analyzed.

The model is now ready to be trained from a set of marked up sequences; the actual training set is described below. PRISMs learning algorithm produces a set of probabilities for the random variables that gives the highest likelihood for the training data. With these probabilities, the model can thus generate artificial, marked up sequences that are faithful to those properties of the training data that are expressible in the model.

4 Previous 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 were 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 obvious since *ab initio* programs like GENSCAN, recognizes genes by small sequences like tata boxes, splicesites etc. and therefore are dependent on possible sequence variations that 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.

5 Results: Evaluation of three different gene finders

In this section we present results of testing three gene finders; GENSCAN, HMMGene and genemark.hmm, using the data generated by our model, as described in a previous section. A significant drop in accuracy was observed [11] when testing GENSCAN on the semi artificial training set described above. 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. Here we have demonstrated a test of the three gene finders using a data set founded on a statistically better ground, and showing a relative high error rate. The model was trained on a data set of 12 intergenic sequences from the human chromosome 17 (NCBI refseq NT_0101718.15), together with a list of specified positions of repeated sequences and GC islands.[5] We chose to use Repeatmasker [19] to find

[5] 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

repeated sequences in these sequences, since this tool appears well established. We used standard masking options for this. RepBase [20] was used for generating a repeater catalogue, in order to train our model on actual repeater sequences. Furthermore positions of GC island were detected by using CpGplot [21], also using standard masking options.

After training our model, the learned probabilities were used to generate 12 sequences, together with annotations of inserted repeated elements and GC islands. From these sequences test sets were assembled, with real genes inserted between the generated sequences. Three test sets were created, each consisting of four generated sequences with three real genes in between. The same three genes are used in all three test sets, making it possible to compare the artificial intergenic sequences alone. The sequences of the genes comprise the genomic region of the gene, from 200 bp before to 200 bp after the CDS.[6] The length of the generated intergenic regions ranges from 2.2 kb to 30 kb. The three test sets have varying lengths of; test set A (193,721 bp), test set B (170,545 bp), and test set C (202,368 bp). 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 term; specificity, which describes the relationship between wrong exons and the total number of predicted exons, as described by [10]. The table below shows the results of the tests performed on the three gene finders, using the three test sets.

	Genscan	**Hmmgene**	**Genemark.hmm**
Predicted genes in testset A	7	12	10
Predicted genes in testset B	10	11	11
Predicted genes in testset C	11	11	10
Average number	**9.33**	**11.33**	**10.33**
Gene Specificity (Wrong genes / predicted genes)	**0.68**	**0.74**	**0.71**
Average number of predicted exons	**54.99**	**83**	**41.66**
Wrong exons in testset A	50	79	24
Wrong exons in testset B	29	55	34
Wrong exons in testset C	33	73	28
Average number of wrong exons	**37.33**	**69**	**28.67**
Exon Specificity (Wrong exons / predicted exons)	**0.68**	**0.83**	**0.69**

Table 1: Results from the tests on gene finders; GENSCAN, HMMGene, and genemark.hmm. Specificity describes the relationship between wrong hits and total hits, where 0 is a perfect result and 1 means all predicted exons are wrong.

All the results show a large error rate, both regarding the number of genes and exons predicted. All gene finders have a gene specificity of about 0.7, which is rather high. This result is probably due to the training of the gene finders on single exon-genes, making them more likely to predict many and short genes. As to the results showing the exon specificity, the same high error rate of approximately 0.7 is seen in GENSCAN and genemark predictions. Here HMMGene has

that most genes are accounted for, and the intergenic sequence therefore is more accurate to use.

[6] The genomic sequence comprising all exons and introns of the gene.

an even higher error specificity of 0.83. These results seem to indicate that all the tested gene finders overpredict, when faced with intergenic sequences containing repeated sequences and GC islands.

6 Comparison of the tested gene finders

To determine if the quite similar high error rate is caused by the gene finders finding the same exons, we made a comparison of the predicted set exons. The figure below illustrates the combination of the three predictions, where the predicted exons are considered the same if they match in either end of the other exon.

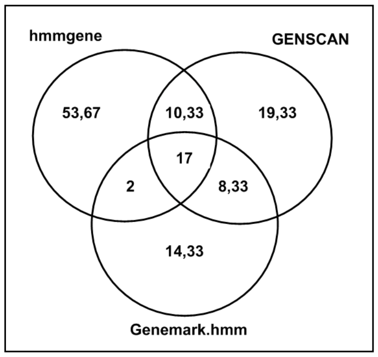
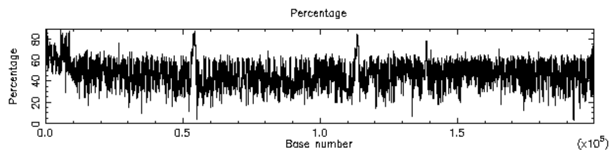


Figure 1: The predictions of the three tests were joined to find exons that are the same. The given numbers are an average value of the three tests performed.

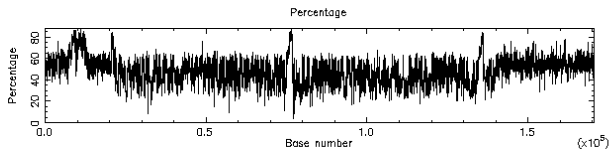
The result of this comparison shows that very few of the predicted exons are in fact the same. HMMGene has the highest number of predicted exons appearing only within its own prediction. The average number of 17 exons being present in all three predictions, and 37.66 exons being present in at least two predictions, is also not a very high number compared to a total number of 125 predicted exons. These results states that the same error rate between the three gene finders is not a consequence of the predicted exons being the same. In contrary, these differing results are probably due to differences in the underlying models of the gene finders. Therefore the results of table 1 could indicate that the precision of the tested gene finders are much lower than anticipated.

7 Appropriateness of the generated data

If the test we are performing is to be used to evaluate a gene finder, the testdata must reflect the essential properties of natural data, and appear as authentic copies. Including most possible dependencies and variables in the data is not possible, but all important features that could affect the outcome of the test must be accounted for. In our model we believe all important features are accounted for, but to examine if this is really the case, an analysis of the three test sets were performed. The length of the intergenic sequences, as well as GC islands and repeated elements did in fact resemble those of the training set. The tool used for finding CG islands in the training material, was also used on the artificial data, to observe if the overall pattern resembled the one of real sequences. The figure below illustrates the GC frequency in data set A, and a section of real sequence from the region the training material was taken from.



Real data



Test set A

Figure 2: GC frequencies in real vs. model generated data.

Figure 2 seems to indicate that the pattern of the GC percentage in the artificial data resembles the real data. However, when comparing the annotated GC islands and the ones found by CpGPlot, there is a slightly higher amount of GC islands in the CpGPlot. This is probably due to the fact that our model operates on absolute frequencies and the CpGPlot presumably uses a more complex model with relative frequencies. This difference in GC island reporting could help explaining why the three tested gene finders predict too many exons, but it does not explain all of them since the results from the three gene finders, as explained, differed greatly. These facts may also indicate the weakness of our model that it uses a first-order HMM to describe GC islands; a higher order, say 3 or 4, may seem more appropriate.

Furthermore an analysis of the dispersed repeated elements was carried out. Using Repeatmasker to find repeated sequences in the artificial data, and comparing them to the annotated elements, showed a very good match between these. Nearly all of the repeated sequences were found. The alignments of the detected repeaters returned by Repeatmasker showed a similar pattern of mutations in the artificial repeater sequences, compared to those in the training set, which thereby verifies our method for generating mutations. From this we conclude that the model generates fairly realistic artificial intergenic sequences, though this training set, as mentioned, is to be considered as an example of what the model could be trained on. Therefore the evaluation of GENSCAN, HMMGene and genemark will not be an absolute assessment. For this a larger training set is needed, making the probabilities of the model more thoroughly based. We believe however, that this new method of evaluating gene finders, is a new step in the way of finding a more accurate measure for the precision of gene finders.

8 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 intergenic regions remain sparse and difficult to access.

The use of artificial data sets is a subject under debate. As discussed by [22] the use of artificial data sets 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 [22] 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 data set might not be ideal for another set. Furthermore, 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 the the best out of few training date data and and too little out of large sets. Only testing and analyze indicate the right level.

As noticed above, we believe that the specific model described in this paper can be improved concerning the modeling of GC island by using a higher order HMM, and we may also refer to [23] who discusses consequences of different orders, and proposes an extension to HMM which may be interesting to apply in PRISM-based models.

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 plans for extensions of PRISM includes specialized representations for sequences (as opposed to list structures!) as well as adaptation to computer clusters. We hope that the continued refinement of this sort of models of genome sequences, based on a very flexible, powerful and theoretically well-founded paradigm, may lead to models that can be used in future gene finders that provide a higher accuracy than what we can observe at present.

The overall approach of using artificially test data produced by sophisticated, statistical models and machine learning was here motivated by the conditions for sequence analysis, that authentic test data are difficult to require. Similar methods may also have their relevance for embedded systems, with the statistical model simulating a world, and context-aware systems.

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

References

1. Myers, G.J., Sandler, C. (Revised by), Badgett, T. (Revised by), Thomas, T.M. (Revised by): The Art of Software Testing, 2nd Edition. Wiley (2004)
2. TREC: Text REtrieval Conference ( ) http://trec.nist.gov/.
3. MUC: Message Understanding Conferences ( ) http://www-nlpir.nist.gov/related_projects/muc/.
4. Burge, C., Karlin, S.: Prediction of complete gene structures in human genomic DNA. Journal of Molecular Biology **268** (1997) 78–94
5. Birney, E., Clamp, M., Durbin, R.: GeneWise and Genomewise. Genome Res. **14**(5) (2004) 988–995
6. Yeh, R.F., Lim, L.P., Burge, C.B.: Computational Inference of Homologous Gene Structures in the Human Genome. Genome Res. **11**(5) (2001) 803–816
7. Venter, J.C. *et al* (> 300 authors): The Sequence of the Human Genome. Science **291**(5507) (2001) 1304–1351
8. Lander, E.S. *et al* (> 300 authors): Initial sequencing and analysis of the human genome. Nature **409**(6822) (2001) 860–92
9. Hogenesch, J.B., Ching, K.A., Batalov, S., Su, A.I., Walker, J.R., Zhou, Y., Kay, S.A., Schultz, P.G., Cooke, M.P.: A comparison of the Celera and Ensembl predicted gene sets reveals little overlap in novel genes. Cell **106**(4) (2001) 413–415
10. Burset, M., Guigó, R.: Evaluation of Gene Structure Prediction Programs. Genomics **34**(3) (1996) 353–367
11. Guigó, R., Agarwal, P., Abril, J.F., Burset, M., Fickett, J.W.: An Assessment of Gene Prediction Accuracy in Large DNA Sequences. Genome Res. **10**(10) (2000) 1631–1642
12. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. (JAIR) **15** (2001) 391–454
13. Sato, T., Kameya, Y.: Statistical abduction with tabulation. In Kakas, A.C., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond. Volume 2408 of Lecture Notes in Computer Science., Springer (2002) 567–587
14. Krogh, A.: Using database matches with for HMMGene for automated gene detection in Drosophila. Genome Research **10**(4) (2000) 523–528
15. Lukashin, A., Borodovsky, M.: Genemark.hmm: new solutions for gene finding. Nucleic Acids Research **26**(4) (1998) 1107–1115
16. Zhou, N.F.: B-Prolog web site (1994–2006) http://www.probp.com/.
17. Charniak, E.: Statistical Language Learning. The MIT Press (1993)
18. Fine, S., Singer, Y., Tishby, N.: The hierarchical hidden markov model: Analysis and applications. Machine Learning **32**(1) (1998) 41–62
19. Smit, A., Hubley, R., Green, P.: Repeatmasker web site (2003) http://repeatmasker.org.

20. Jurka, J., Kapitonov, V., Pavlicek, A., Klonowski, P., Kohany, O., Walichiewicz, J.: Repbase Update, a database of eukaryotic repetitive elements. Cytogenetic and Genome Research **110**(1-4) (2005) 462–467
21. EMBL-EBI: CpGplot ( ) http://www.ebi.ac.uk/emboss/cpgplot/#.
22. Scott, P.D., Wilkins, E.: Evaluating data mining procedures: techniques for generating artificial data sets. Information & Software Technology **41**(9) (1999) 579–587
23. Wang, J., Hannenhalli, S.: Generalizations of Markov model to characterize biological sequences. BMC Bioinformatics **6** (2005)