Cem Paya
cemp@alum.dartmouth.org
12/2003

Comparison of gene prediction algorithms

Introduction

This paper compares three different paradigms for gene prediction in DNA sequences: neural networks, hidden-markov models and context-free language parsing.

Common properties

All three approaches share a number of common properties, which we list before going on to explore their differences:

- Most algorithms locate protein-coding regions as opposed to whole genes. Nevertheless "gene prediction" remains standard nomenclature.
- There is a significant amount of application-specific knowledge going into these designs. Of the systems referenced here only VEIL comes close to being a pure application of an existing algorithm to raw nucleotide data.
- Same problems come up under different guises and are special-cased differently in each implementation. For example the frame problem: algorithm needs to determine its "phase" inside the sequence (1st, 2nd or 3rd nucleotide in the codon)
- All the algorithms involve training a model against known data-set with full annotation. Performance is measured either by cross validation or against new data disjoint from training set. Two commonly used metrics for characterizing accuracy are specificity and sensitivity.
- In spite of all the tweaking and hacks accuracy rates are still disappointing. An earlier survey of the algorithms Burset and Guigo found in [Bu95] that prediction accuracy on new data was invariably lower than what the authors claimed originally, indicating tendency for over-training. (This paper did not consider the HMM systems and included GRAIL2 instead of the original GRAIL algorithm discussed here.)
- One-sided error bias: false positives are rare, false negatives are more common. Most errors involve missing an entire coding region. As such they are well-suited for identifying candidates for subsequent experimental confirmation. With the latter process being costly, an algorithm that chases after non-existent red-herrings is less desirable than a conservative one which runs the risk of missing an exon on occasion.
- All systems described are computationally intensive. Implementation were CPU bound on stock PC hardware of the day. Some were also memory bound in terms of amount of storage required.
- Papers referenced here are dated and not reflective of the state of the art.

1.Neural networks

In an early paper describing GRAIL, Uberbacher and Mural describe the architecture for a neural network designed for gene prediction. Seven feature sensors synthesize high-level concepts from the sequence of nucleotides around a 100 nucleotide windows, ranging from standard-operating procedure (frame detection) to esoteric (fractal dimension of nucleotide pair variation):

1. Frame bias—for solving the frame alignment problem
2. Fickett algorithm
3. Dinucleotide fractal dimension; by analogy from statistical mechanics. Views the sequence as an energy fluctuation over time, with energy proportional to log likelihood of the pair of adjacent nucleotides.
4. Six-tuple word preferences
5. Six-tuple in-frame preferences

6. Word commonality
7. Six-tuple word repetitions

Authors argue that feeding these high-level inputs into the network instead of raw nucleotide symbols is an optimization, a short-hand effectively to allow a small network to cope with large amounts of data. (Compare the 6-tuple frequency as single input vs. creating a network to learn distribution of all 4**6 = 4096 possible six-tuples.) Sensors feed normalized weights in the [0,1] interval into a neural network of three layers: seven inputs, one for each sensor, distributing out to 9 perceptrons in a hidden layer, which funnels down to 5 perceptrons in a second hidden layer, finally feeding into the final output node. Output is a confidence level between 0 and 1 indicating whether current nucleotide is part of an exon encoding protein. (Because the neural network has only one output, it can only decide one property using a cut-off threshold.) This complete system is dubbed CRM or coding recognition module, and is trained using the back-propagation algorithm, which adjusts connections between perceptrons based on predicted verses actual classification of training data.

CRM did reasonably well in the authors' benchmark: >90% correct classification of sequence locations, and 8% false positives. Some of the problematic cases are expected given the design: areas contiguous with the 3' or 5' ends (because the network has no concept of these as distinct regions, it is forced to classify them according to its limited dichotomy) and exons much shorter than the 100-location window used for sensors.

The usual "philosophical" objections to neural networks regarding their black-box nature apply here. Weights assigned to the connections after training often do not easily lend themselves to interpretation. GRAIL effectively functions as an oracle that gives answers without any explanation. This is easy to overlook as long as the answers are correct but overtraining aspect poses risks. Because the "intelligence" of the network is hidden in the connection weights, there is usually no way to sanity check the system for consistency against basic empirical rules about gene structure. Also there is arbitrariness to network topologies. There is no convergence guarantee except for in simplest case of single-layer perceptrons, which are too limited in their expressiveness for complicated systems. As such there is no analytical basis for favoring say three hidden layers over two (as built in this paper.) Designing effective neural networks remains more art than engineering. The authors did experiment with simpler designs and observe "… little dependence on the precise network configuration, although performance is compromised with fewer nodes in the hidden layers."

2. Hidden Markov models

HMM approach is based on viewing the nucleotide sequence as a Markov process, with the nucleotides corresponding to the symbols output. States correspond to different regions of the DNA (exon, intron, splice site etc.) and the problem involves inferring the most probably state transitions corresponding to an observed nucleotide sequence. [Hen97] describes the system VEIL based on this idea and [Ku97] extends the model such that individual states can output sequences of symbols from an arbitrary distribution of their own.

Key to describing an HMM is the transition matrix of probabilities. When the matrix is visualized as a directed graph (with zero-probability edges omitted) and each run through the model corresponds to a random walk on the graph. VEIL model is built by splicing together multiple sub-graphs that were individually trained using the E-M algorithm. For example the intron model has 12 nodes, arranged into three layers of four nodes with forward links between adjacent layers and backward links from last layer back into the first. Finally these sub-models are assembled according to high-level blue print (see BNF description in next section) and this complete HMM is trained using another round of E-M. Result is a sparse graph (241 states, 1003 edges for average out-degree <4.2) with only a small fraction of edges (12%) going between sub-models—which might explain the observation that final round of training had little effect on most transition weights. Training employs an interesting trick: each state is labeled as non-gene, exon or intron. Effectively the original alphabet of {A, C, G, T} is expanded by taking the cross-product with {N, E, I} to create 12 symbols. Data in the training set is annotate with its known characterization, and sub-models are tweaked to force alignment: for example the exon model will only consume the symbols {EA, EA, EG, ET}. After training these labels are removed by collapsing multiple symbols into one of 4 nucleotides.

Training and predicting with an HMM of this scale was computationally feasible on stock hardware circa 1997, although this meant building the system from scratch instead of using an off-the-shelf HMM generator. Each run still used multiple CPU hours on a network of workstations. The Viterbi algorithm used for prediction is memory intensive but trellis pruning cut down requirement by 20x factor. In benchmarks the algorithm had approximately 80% Sn & Sp scores for classifying nucleotides, and around 50% for exact identification of coding exons.

VEIL is by far the cleanest of three algorithms discussed here. At the same time its simplicity causes failures on exceptional situations that other algorithms handle with special cases. For example it has no concept of frames and can't keep track of frame-shifts. Authors point out other simplifying assumptions and where these can lead the algorithm into errors: the first exon of every gene beings with a start codon (there are genes violating this rule), noncoding regions and introns have exons on either side, each piece of DNA contains a single gene, coding exons have at least 7 bases. Some of these have work-arounds: for example marking every state as start and final state allows it to handle arbitrary DNA sequences containing multiple genes or fragments of genes.

One generalization of HMMs for predicting genes was proposed in [Ku97] around same time as VEIL. Generalized HMMs allow each state to emit an arbitrary sequence according to some probability distribution—in contrast to ordinary HMMs which emit exactly one symbol per state. An interesting consequence of this is that length distribution of strings generated in the model need not have a geometric tail. To see why this is necessary for an ordinary Markov model, consider any cycle in the graph reachable from a start state. Suppose the shortest path from start state into the cycle has length L and probability w, and probability of going around the cycle once is p. In that case the model can generate another string of length L + k|C| by going around the cycle k-times with probabiliy w p**k, where |C| is length of the cycle. Due to this exponential relationship between length and probability, each cycle in the graph gives rise to a geometric tail and the overall distribution is a superposition of these. This is important since empirically observed distribution of exon lengths is not geometric. On the downside training and recognition problems are significantly more complicated for GHMMs. Difficulty is associated with computing whether a given substring of nucleotides has been generated by the "supernode" representing an individual state such as intron, exon or splice-site. These correspond to feature sensors that examine chunks of DNA and come up with probability that the fragment was generated by that state. A dynamic programming algorithm combines these estimates to maximize, over all state transition of the GHMM, the probability that DNA sequence was producted by that state transition. Resulting system dubbed Genie had surprisingly about same effectiveness compared to VEIL in classifying nucleotides correctly, but an appreciably better success in identifying complete exons. (61% sensitivity & 64% specificity verses around 50% for VEIL)

3. Context-free grammars

CFGs are a natural generalization of HMM for modelling the genome. In computational terms Markov models correspond to finite automata, which recognize the set of regular languages. Context-free languages are strict super-set of regular languages. Any sequence described by Markov model can also be described by an equivalent context-free grammar. It is known that there exists context-free languages which can not be modelled by a Markov model of k-th order for any k. Intuitively these correspond to models where arbitrarily large state information must be carried forward to compute next state. Asking whether the genome is among these is the starting point for GENLANG.

As an example, we can rewrite the HMM of [Hen97] as a context-free grammar in Backus-Naur form (BNF) as follows:

```
S → U  C  D
C → Start E Stop
E → Cr Tr E
E → Cr
Cr → Exon
Tr → 5Splice Intron 3Splice
```

In this formulation S is the start symbol, all others are non-terminals corresponding to various pieces of the HMM: for example E is an exon region. For example the third rule states that an exon region can consist of a coding region, followed by transitional area, followed by another exon region. This is a recursive rule, corresponding to the cycle in the HMM architecture. In the complete grammar there would be additional production rules corresponding to each of these non-terminals, until finally every non-terminal is mapped into the terminal symbols which are the nucleotides {A, C, G, T}.

CFG parsing problem is defined as: given string of terminal symbols and identify some sequence of production rules to convert the start symbol into this string. In general CFGs can be inflexible because they either generate or fail to generate a string. By associating a probability with each production rule, we can allow for multiple different parses and associated costs. In language terms this is an "ambiguous grammar" which is generally undesirable for parsing purposes. Although not stated formally in the paper, ambiguity here is a property of the language: eg underlying language is inherently ambiguous, every CFG accepting the language will allow multiple parses by design because recognition depends on maximizing probabilities to choose between alternative parse trees. The authors propose associating costs with a particular parse tree recursively. First the grammar is written so that every production has at most two symbols on the right-hand side; basic result in complexity theory says every context-free language has a CFG in this form. (An even more stringent canonical form known as Chomsky normal form is used in the paper.) Direct costs are assigned to *leaf rules* which are complicated feature sensors instead of individual nucleotides. These costs seeks to capture the confidence level of the feature being observed—lower confidence leads to higher-cost to "discourage" use of that symbol. Every other production rule computes its cost as weighted average of the first and second symbols on the right hand-side, which correspond to left and right subtrees in the parse.

For example suppose the production rule is

$$A \rightarrow B\ \ C.$$

Cost of invoking this rule is defined as:

$$\text{Cost } (A) = (1-\mu) \text{ cost } (B) + \mu \text{ cost } (C)$$

where μ is the mixing parameter associated with this rule. There are also two thresholds, one for each subtree. The thresholds represent an absolute upper bound on the cost that can be allotted to that subtree. Exceeding the threshold on either side will completely suppresses invocation of the rule.

GENLANG uses 13 leaf rules and defines a CFG with start symbol "gene." Only one of the productions is recursive in nature, where the same symbol appears on both left-hand and right-hand side of the production. Training involves an ad hoc procedure for adjusting threshold and mixing parameters in each node. This is done iteratively. All mixing parameters are initialized to 1/2 to favor left/right costs equally and thresholds were calculated directly from the training set itself. Each step perturbs the mixing coefficient and compares prediction accuracy before/after. In benchmarks GENLANG was comparable to GRAIL in its effectiveness at locating complete exons; the former is more sensitive while the latter is more specific. GRAIL did better when viewed in terms of classifying bases correctly, a difference authors attribute to the different exon-sizes each algorithm is optimized for with GRAIL doing better on longer sequence—because of its 100 base window—and GENLANG excelling at locating shorter genes. Surprisingly GENLANG did not do appreciably better than the simpler HMM model used in VEIL, leaving open the question of whether the increased expressiveness of context-free grammars over finite automaton in modelling DNA sequences was necessary in the first place.

References

[Ub91] "Locating protein-coding regions in human DNA sequences by a multiple sensor neural-network approach" Edward C. Uberbacher, Richard J. Mural

[Hen97] "Finding genes in DNA with a Hidden Markov Model". John Henderson, Steven Salzberg, Kenneth H. Fasman

[Ku97] "A generalized Hidden Markov Model for the recognition of human genes in DNA" David Kulp, David Haussler, Martin G. Reese, Frank Eeckman

[Do94] "Gene structure prediction by linguistic methods", Shan Dong, David B. Searls.

[Bu95] "Evaluation of gene structure prediction programs", Moises Burset, Roderic Guigo.