CMSC423: Bioinformatic Algorithms, Databases and Tools
Lecture 17

Gene finding

Sample midterm questions

- 6.22
- 8.6
- 10.12

- also see homeworks at http://www.cbcb.umd.edu/confcour/CMSC858E-syllabus.shtml

Gene finding/prediction

- Given a string of DNA, identify regions that might be genes
- Question: What does a gene look like?

- Start codon: ATG
- Stop codon: TGA, TAG, TAA
- Splicing: GT...intron...AG

- Also, DNA composition is different in genes – mutations are more likely in the third position of codons.

Simple gene finder (in bacteria)

- Find all stop-codons in the genome
- For each stop-codon, identify an in-frame start-codon upstream of it.
- Each section between a start and a stop is called an ORF – open reading frame.
- The long ORFs are likely genes – evolution prevented stop codons from occuring

- 3 stop codons, 64 possible codons => in random DNA every 22nd codon is a stop.

GGC **TAG** $\boxed{\textbf{ATG}\text{ AGG GCT CTA ACT }\textbf{ATG}\text{ GGC GCG }\textbf{TAA}}$

Gene finding as machine learning

- Main question: does the ORF look like a gene?

- Given a set of examples – genes we already know
- and a string of DNA (e.g. ORF)
- compute the likelihood that the ORF is a gene.

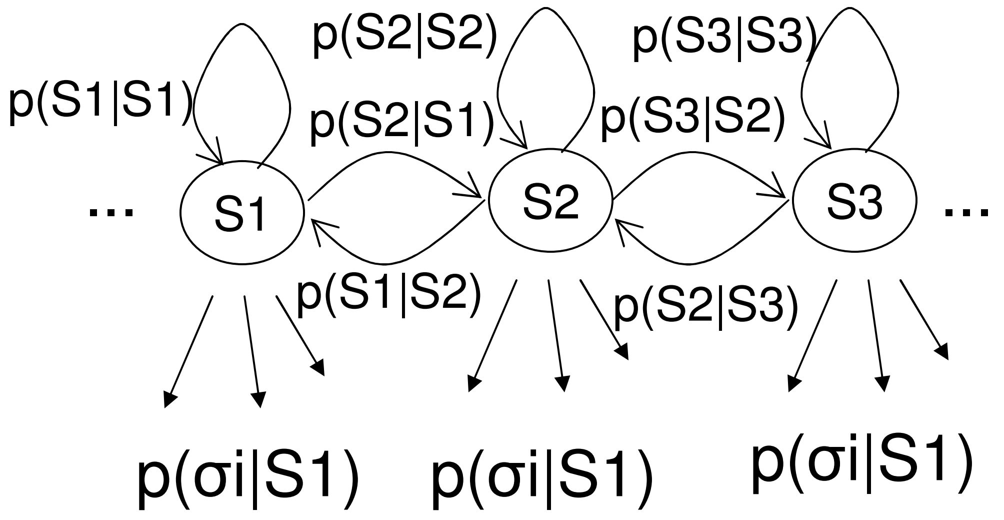
- Codon usage bias – not all codons for a same amino-acid are equally likely
- K-mer (e.g. 6-mer) frequencies

Bacillus anthracis codon usage

UUU F 0.76	UCU S 0.27	UAU Y 0.77	UGU C 0.73
UUC F 0.24	UCC S 0.08	UAC Y 0.23	UGC C 0.27
UUA L 0.49	UCA S 0.23	UAA * 0.66	UGA * 0.14
UUG L 0.13	UCG S 0.06	UAG * 0.20	UGG W 1.00
CUU L 0.16	CCU P 0.28	CAU H 0.79	CGU R 0.26
CUC L 0.04	CCC P 0.07	CAC H 0.21	CGC R 0.06
CUA L 0.14	CCA P 0.49	CAA Q 0.78	CGA R 0.16
CUG L 0.05	CCG P 0.16	CAG Q 0.22	CGG R 0.05
AUU I 0.57	ACU T 0.36	AAU N 0.76	AGU S 0.28
AUC I 0.15	ACC T 0.08	AAC N 0.24	AGC S 0.08
AUA I 0.28	ACA T 0.42	AAA K 0.74	AGA R 0.36
AUG M 1.00	ACG T 0.15	AAG K 0.26	AGG R 0.11
GUU V 0.32	GCU A 0.34	GAU D 0.81	GGU G 0.30
GUC V 0.07	GCC A 0.07	GAC D 0.19	GGC G 0.09
GUA V 0.43	GCA A 0.44	GAA E 0.75	GGA G 0.41
GUG V 0.18	GCG A 0.15	GAG E 0.25	GGG G 0.20

A more general solution

- Hidden Markov models
- States, transition probabilities, emission probabilities



- p(Si|Sj) – probability of transitioning to state i if we are in state j
- p(σi|Sj) – probability of emitting symbol σi if we are in state j

Why “Hidden”?

- Observers can see the emitted symbols of an HMM but have no ability to know which state the HMM is currently in.
- Thus, the goal is to infer the most likely hidden states of an HMM based on the given sequence of emitted symbols.

HMM Parameters

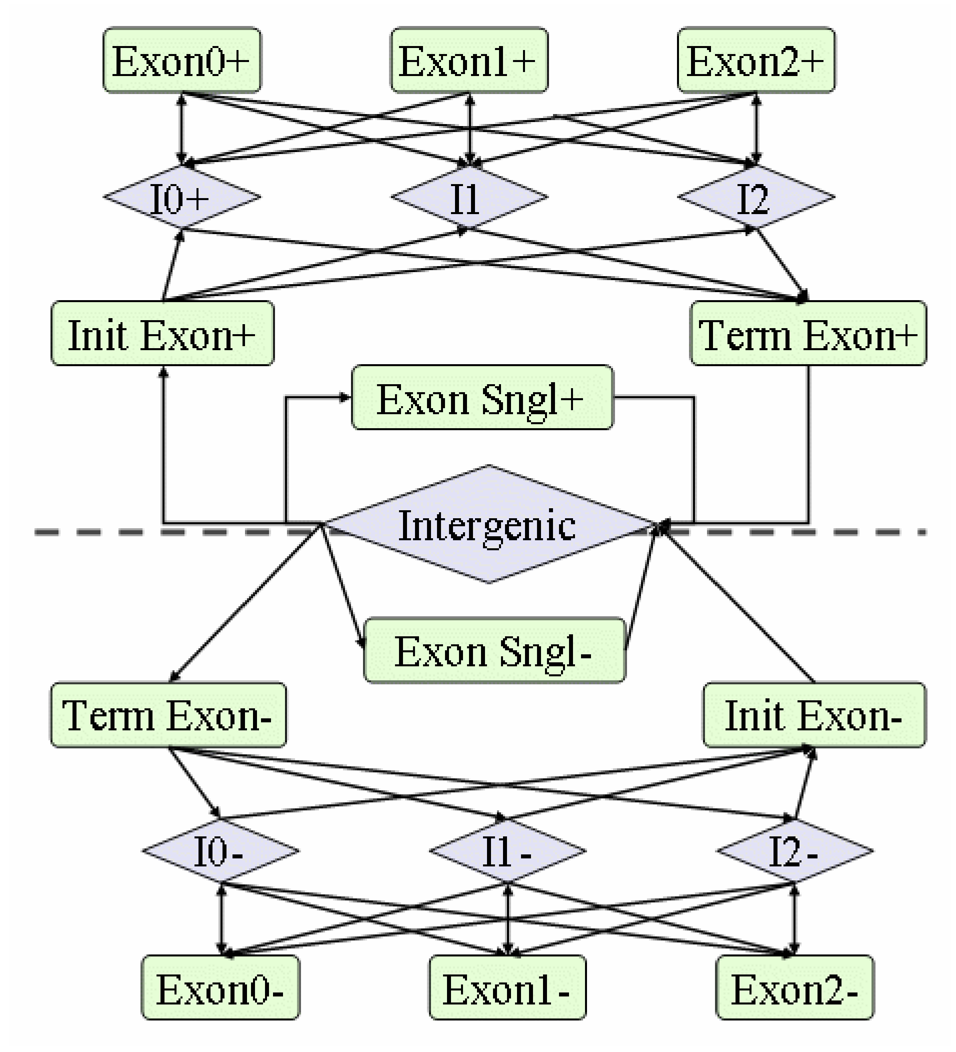
- Σ: set of emission characters.
  - Ex.: Σ = {H, T} for coin tossing
  - Σ = {1, 2, 3, 4, 5, 6} for dice tossing
  - Σ = {A, C, T, G} for DNA

- Q: set of hidden states, each emitting symbols from Σ.
  - Q={Fair,Biased} for coin tossing
  - Q={gene, not gene} for bacteria
  - Q={exon, intron, intergenic) for eukaryotes

HMM Parameters (cont'd)

- A = (akl): a |Q| x |Q| matrix of probability of changing from state k to state l.
  - aFF = 0.9   aFB = 0.1
  - aBF = 0.1   aBB = 0.9
- E = (ek(b)): a |Q| x |Σ| matrix of probability of emitting symbol b while being in state k.
  - eF(0) = ½   eF(1) = ½
  - eB(0) = ¼   eB(1) = ¾

GlimmerHMM model



Questions we can ask with HMMs

- Given an observed sequence of emitted characters (a string of DNA), what is the most likely sequence of states that generated the observed sequence?
  - given a string of DNA and the model, break it up into genes
  - solved by Viterbi algorithm
- Given an observed sequence of emitted characters, what is the most likely state the model was in at time t?
  - given a string of DNA, how likely is it that a certain location is inside a gene?
  - solved by forward-backward algorithm

Training – the key to HMMs

- So far we've assumed that all probabilities are known.
- The training problem:
  - given an HMM (just the states and connections)
  - given several examples (e.g. known genes and intergenic regions)
  - compute the transition and emission probabilities
- Training is difficult!!
- Baum-Welch algorithm – iterative optimization
  - start with estimates of the probabilities
  - run model with training data
  - re-estimate probabilities based on performance on training data