CS262 (Winter 2015)
Lecture 5 (January 20)
Scribe: Kat Gregory

BLAST / HIDDEN MARKOV MODELS

BLAST CONTINUED

HEURISTIC LOCAL ALIGNMENT

- **Use**
  - Commonly used to search vast biological databases (on the order of terabases/tetrabases)
  - Quickly find all "good" local alignments of query to databases
    - "Good" = Above threshold of length (ex 100+) and sequence similarity (ex 60+%)
  - Heuristic
    - Relies on assumption that a good match is likely to contain exact matching "k-mers"
      - "K-mer" = Word of length k
      - Why is this helpful? B/c exact words can be easily indexed and sorted
    - Faster than full alignment (Smith-Waterman)
    - But cannot guarantee optimal alignment
- Original BLAST (BLAST = **B**asic **L**ocal **A**lignment **S**earch **T**ool)
  - *Altschul et al.* (1990) is probably most cited paper in bioinformatics
    - One of most cited papers in science in general
  - Followed previous tool FASA (1985)
  - 100s to 1000s papers written on this technique
- **Procedure**
  1. Create **table of all word occurrences** in databases
     - Table length: 4^k
       - Can be in memory
     - Each location in table:
       - Denotes a specific word
       - Points to an array of all occurrences of this specific word in database
  2. Use sliding window of length k to **consider every k-mer in query**
     - Find all matches of each query k-mer to database above alignment score threshold
  3. For every match, **extend** to left and right to create potential local alignment
     - Original BLAST:
       - Ungapped extensions until score below statistical threshold
       - In DP matrix, equates to matching along diagonal
     - Modification allows for gapped extensions
       - Compute a dynamic programming matrix where you cannot cross a cell if it would make score cross threshold a given amount below the best score seen so far
- Take home messages:
  - BLAST works and is used in practice
  - But we pay a cost in memory and time to find local alignments
  - And there is a lot of science about how to minimize these costs and make BLAST work best

SENSITIVITY-SPEED TRADEOFF

- Quantify:
  - **Sensitivity**:
    - "Of all existing local alignments between query and database, how many do we find?"
    - Defined for a given length (say 100) sequence similarity (say 60, 80%) threshold
      - Assume this sequence identity is distributed uniformly across local alignment
        - EX) For 80% similar match, assume each letter independently has 80% chance of matching

- **Speed**:
  - "How many random matches of words between query and database are generated?"
  - Every match not associated with a local alignment is costs computation time because must extend alignment to left and right to check
- Findings from Kent (2002):
  - Aside on Jim Kent:
    - Credited with saving the public human genome project as PhD student in 2000
      - Complex especially because of heterogeneity of different data sets
      - Competing with Celera Genomics, who might have made data about human genome proprietary
      - Called "Most famous graduate student in the world"
    - Also developed a faster version of BLAST called BLAT
      - **B**last **L**ike **A**lignment **T**ool

Table 3. Sensitivity and Specificity of Single Perfect Nucleotide K-mer Matches as a Search Criterion

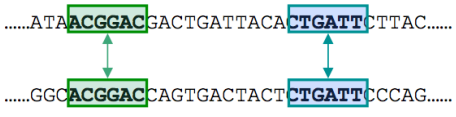
		7	8	9	10	11	12	13	14
A.	81%	0.974	0.915	0.833	0.726	0.607	0.486	0.373	0.314
	83%	0.988	0.953	0.897	0.815	0.711	0.595	0.478	0.415
	85%	0.996	0.978	0.945	0.888	0.808	0.707	0.594	0.532
	87%	0.999	0.992	0.975	0.942	0.888	0.811	0.714	0.659
	89%	1.000	0.998	0.991	0.976	0.946	0.897	0.824	0.782
	91%	1.000	1.000	0.998	0.993	0.981	0.956	0.912	0.886
	93%	1.000	1.000	1.000	0.999	0.995	0.987	0.968	0.957
	95%	1.000	1.000	1.000	1.000	0.999	0.998	0.994	0.991
	97%	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999
B.	K	7	8	9	10	11	12	13	14
	F	1.3e+07	2.9e+06	635783	143051	32512	7451	1719	399

(*A*) Columns are for K sizes of 7–14. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated from equation 3 assuming a homologous region of 100 bases. The larger the value of K, the fewer homologies are detected.
(*B*) K represents the size of the perfect match. F shows how many perfect matches of this size expected to occur by chance according to equation 4 in a genome of 3 billion bases using a query of 500 bases.

  - Simulation paper quantifies sensitivity and speed for different ways of indexing
    - Setup: A 100 bp region of given similarity (81 – 97%) within a 500 bp query
      - Human genome as database
      - Tested different indexing scheme parameters (% similarity and k)
    - Evaluated:
      - "How often is this 100 long alignment found with this indexing scheme?" (Sensitivity)
      - "And how many random hits do we have?" (Speed)
    - Example Interpretation from above table:
      - "With k=8, will find an 81% similar match 91.5% of the time, but will find a 97% similar match 100% of the time. The cost of using k=8 is that a 500bp query will generate 2.9 million spurious 8-mer matches between query and database. With k=15, will find 81% matches only 30% of the time but will spend 10,000x less time."
    - Results:
      - **Short** words favor **sensitivity**
      - **Long** words favor **speed**

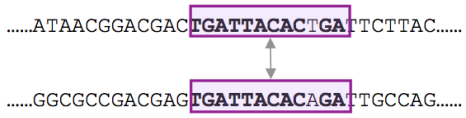
<u>METHODS TO IMPROVE SENSITIVITY / SPEED</u>

1. Using **pairs of words**
   - Instead of insisting on an exact word match of at least length k, pick a k' < k and insist on matching 2 exact words of length k' spaced within certain distance on same diagonal.



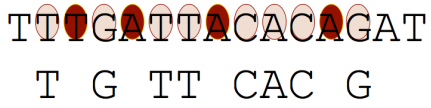
   - Improvement
     - Pair of 8-mers separated by at most 40 nucleotides at 81% alignment give 68% sensitivity.
       - Compared to an inexact 16-mer, twice as fast and slightly more sensitive
       - And easier to implement, so used more frequently than inexact words

2. Using **inexact words**
   - Increase k but allow for 1 or 2 mismatches in the k-mers
   - Index scheme modification for building dictionary of database k-mers:
     - For every k-mer in database, create all k-mers with at least one mismatch and record each occurrence in the table of all word occurrences

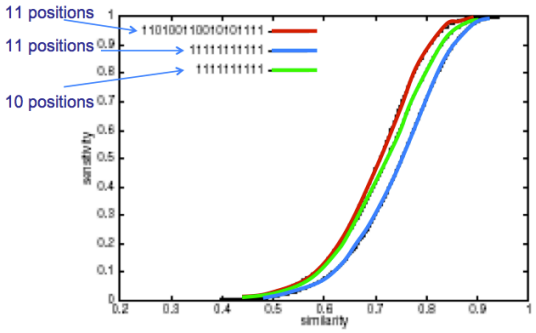


   - Improvement:
     - Sensitivity of 13-mer with one mismatch is better than that of 9-mer with no mismatch. Generates only 68k spurious alignments (vs 635k)
       - More sensitive and ~10x faster
     - In all cases, can do much better moving to inexact k-mers
       - Drawback: Harder to implement

3. Using **word patterns**
   - Elegant contribution from ~2000
     - Nothing necessitates that in indexing we have all positions be consecutive
     - Looks for same pattern with same shift between letters throughout



   - Improvement: (Example: 100-long, 70% conserved region)
     - Decreases expected number of hits (1.07 to 0.97)
     - Increases probability of ≥ 1 match within a long conserved region (0.3 to 0.47)
     - When plot sensitivity as a function of sequence similarity on 100 positions (graph to right):
       - Nonconsecutive 11-mers (red) more sensitive than consecutive 10-mers while being 4x faster because generates 4x fewer random hits

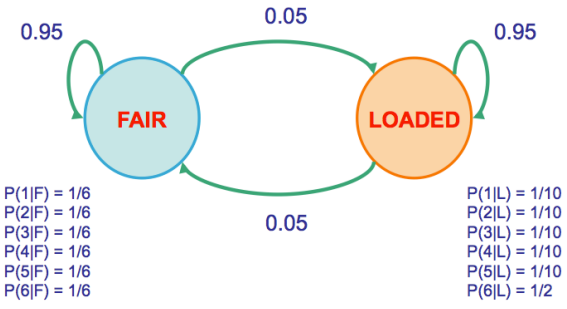


   - Multiple Patterns
     - Multiple patterns can improve sensitivity even further.
     - But how many patterns is best? Becomes NP hard problem given model (probability distributions) for homology between 2 positions to find the best K to optimize both speed and sensitivity
       - K patterns takes k times longer to scan
       - But patterns compliment each other

HIDDEN MARKOV MODELS

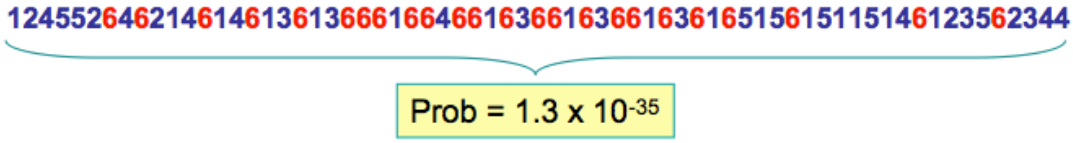
DISHONEST CASINO MODEL

- **Dishonest Casino Model**:
  a. You bet $1
  b. You roll a fair die
  c. Dishonest casino player rolls his die. It can be either fair or loaded, and you can't tell which he is using or when he switches
  d. Highest number wins 2$
- Analysis:
  - Fair/Loaded are hidden states of system
  - Sequence of rolls is observable variables
  - Memory-less because not necessary to consider previous rounds

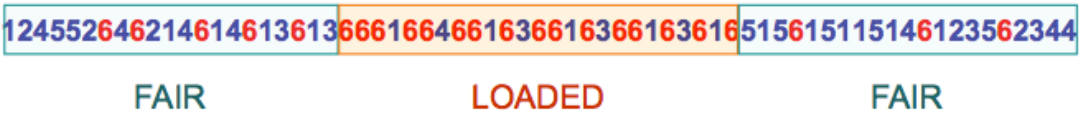


3 MAIN QUESTIONS WE CAN ANSWER

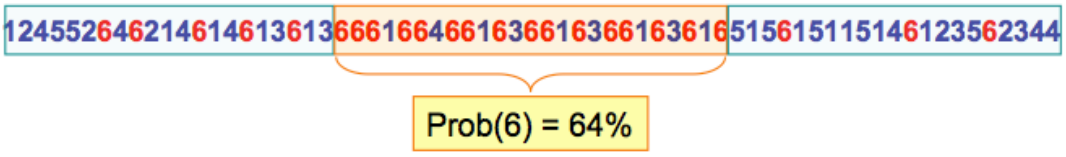
- Question 1: **EVALUATION**
  - Given a sequence of rolls by the casino player, ask "How likely is this sequence given how our model of the casino works?"
  - For example, if we suspect that the dishonest casino player chose "Fair – Loaded – Fair" and then observe the sequence 12455264621461461361366616646616366163661636616515615115146123562344 we might suspect it fits out model but would need a way to formulate this precisely.



- Question 2: **DECODING**
  - Given a sequence of rolls by the casino player, ask "What portion of the sequence was generated with fair die and what with loaded die?"
    - Not a priori an easy question
  - Must know how fair loaded die is and how often the player can switch between dice
    - If he can change die each roll, would be most likely to consider each 6 in the sequence to be from the loaded die
    - But if he cannot change die at all, we should consider either the whole sequence fair or loaded

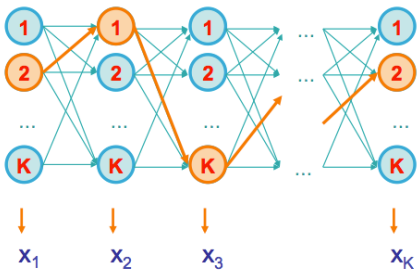


- Question 3: **LEARNING**:
  - Given a sequence of rolls by the casino player, ask "How loaded is the loaded die? How fair is the fair die? How often does the casino player change from fair to loaded and back?"
  - Lower threshhol = count # sizes across sequence
  - Maybe do something fancy and look around middle
  - Precisely mathematically is not so blear
  - Will find optimal parameters (or try – NP hard problem)



HIDDEN MARKOV MODELS (HMMS)

- **Hidden Markov Model**: Statistical Markov model where the state is not directly visible but the output is visible
  - Simple to implement and "works like magic"
- Consist of:
  - Alphabet: $\Sigma=\{b_1,b_2, \ldots,b_M\}$
  - Set of States: $Q = \{1, \ldots, K\}$
  - Transition Probabilities between any 2 states
    - a_{ij} = Transition probability from state I to state j
    - $a_{i1} + \ldots + a_{iK} = 1$ for all states i = 1 … K
  - Start probabilities a_{oi}
    - $a_{01} + \ldots + a_{0K} = 1$
  - Emission probabilities for each state
    - $e_i(b) = P(x_i \mid \pi_i = k)$
    - $e_i(b_1) + e_i(b_M) = 1$ for all states i = 1 …
  - [We will not use End Probabilities a_{i0} as defined in Durbin]



- **Memory-Less**: At each time step t, the only thing that affects future is current state π_t
  - For states (π_t)
    - $P(\pi_{t+1} = k \mid$ "whatever happened so far") =
    - $P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$
    - $P(\pi_{t+1} = k \mid \pi_t)$
  - For emission (x_t)
    - $P(\pi_{t+1} = k \mid$ "whatever happened so far") =
    - $P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) =$
    - $P(\pi_{t+1} = k \mid \pi_t)$
  - On making the "memory-less" assumption:
    - To determine if this "memory-less" assumption holds, we could observe and prove rules to show that assumption fails
      - Ex) Show that every time the casino player rolls 6 times, he definitely switches dice
    - The assumption is a powerful tool because great in modeling
    - Convenient to train, document, and run but in general, it is true that assumptions do fail
    - In a way, HMMs are very simplistic ways to model sequences (especially proteins)
      - We will cover powerful, more recent technique called Conditional Random Fields

PARSES

- "**Parse**" of a sequence:
  - Given a sequence $x = x_1$ to x_n, a parse of sequence x is the underlying sequence of states $\pi = \pi_1, \ldots, \pi_n$
- Calculate precise **likelihood of parse** as the product of transmission and emission probabilities
  - $P(x=x_1, \ldots, x_n, \pi = \pi_1, \ldots, \pi_n) = P(x_N \mid \pi_n)\, P(\pi_N \mid \pi_{N-1}) \ldots P(x_2 \mid \pi_2)\, P(\pi_2 \mid \pi_1) * P(x_1 \mid \pi_1)\, P(\pi_1)$
    $= a_{0\pi1}\, a_{\pi1\pi2} \ldots a_{\pi N-1\pi N} * e_{\pi1}(x_1) \ldots e_{\pi N}(x_N)$
  - Useful alternative approach:
    - Observe that in our casino example HMM, we have a finite, often small number of parameters. In this model, we have only 4 transition parameters (2 independent). We also have 12 emission probabilities (1-6 from each of 2 states) and 2 start probabilities.
      - $4 + 2 + 12 = 18$ parameters called $\theta_1 - \theta_{18}$
    - Imagine that you play 1 million rolls. That underlying sequence has 2 million arrows.
      - $P_{i\,(az - a1000)} * P_{i\,(e1 \ldots e1000000)}$
    - Another way to approach is having these exponentiated by the number times they occur. In the parse, create an indicator (called the feature counts):
      - $F(j, x, \pi)$ = # parameter θ_j occurs in (x, π)
    - Then:
      - $P(x, \pi) = \Pi_{j=1\ldots n}\, \theta_j^{F(j, x, \pi)}$
        $= \exp[\Sigma_{j=1\ldots n} \log(\theta j) * F(j, x, \pi)]$
    - Could think of this as a score
      - We will come back to this and talk about optimizing with respect to parameters
- If casino player couldn't switch dice, is it more likely all the rolls were done with the fair die or the loaded one?
  - Given: Sequence of rolls is $x = \{1, 2, 1, 5, 6, 2, 1, 5, 2, 4\}$
    - $P(\text{Fair}) = ½ * P(1 \mid \text{Fair})\, P(\text{Fair} \mid \text{Fair}) * P(2 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) * \ldots * P(4 \mid \text{Fair})$
      $= ½ * (1/6)^{10} * (0.95)^9$
      $= 0.521 * 10^{-9}$
    - $P(\text{Loaded}) = ½ * P(1 \mid \text{Loaded})\, P(\text{Loaded} \mid \text{Loaded}) \ldots P(4 \mid \text{Loaded})$
      $= ½ * (1/10)^9 * (1/2)^1 * (0.95)^9$
      $= 0.16 * 10^{-9}$
    - $P(\text{Fair}) > P(\text{Loaded})$
  - Given: Sequence of rolls is $x = \{1, 6, 6, 5, 6, 2, 6, 6, 3, 6\}$
    - $P(\text{Fair}) = ½ * (1/6)^{10} * (0.95)^9$
      $= 0.521 * 10^{-9}$
    - $P(\text{Loaded}) = ½ * (1/10)^4 * (1/2)^6 * (0.95)^9$
      $= 0.5 * 10^{-7}$
    - $P(\text{Fair}) < P(\text{Loaded})$

DECODING APPLICATION IN BIOLOGY

- Given a sequence of outputs, find the most likely sequence of states that produced it
- Biological application: If we have a model of gene structure, may ask what is the most likely potential gene structure in a new gene sequence.
    - Work on gene structure first focused on finding genes computationally
    - However, at conference of gene finding, Chris Burge presented **GENSCAN**
        - Used a HMM
        - Worked 10% better than other methods
        - Though later improved upon, this marks the introduction of HMMs into computational biology. They have since become very popular, trending along with other graphical/probabilistic models of modeling data