A Crash Course in Bioinformatics and Image Processing For Math People

Dey, Arion

Outline

Disclaimer

My focus is on breadth, not depth:

- Bioinformatics is a massive field
- It is also much more than just genomics and image processing, but this is what I know.

Section 1

What is Bioinformatics

Despite the message of these first two slides, I love Bioinformatics

'Bioinformatics is an attempt to make molecular biology relevant to reality. All the molecular biologists, devoid of skills beyond those of a laboratory technician, cried out for the mathematicians and programmers to magically extract science from their mountain of shitty results.'

— *Fredrick J. Ross*

The ones that know coding, usually fear the math

- At the risk of proselytizing the field rather than educating, I want to make clear why I am giving this lecture

The ones that know coding, usually fear the math

- At the risk of proselytizing the field rather than educating, I want to make clear why I am giving this lecture
- Bioinformatics is a unique field, where it is a field defined by computation led by people who are not formally trained in computer science or math

The ones that know coding, usually fear the math

- At the risk of proselytizing the field rather than educating, I want to make clear why I am giving this lecture
- Bioinformatics is a unique field, where it is a field defined by computation led by people who are not formally trained in computer science or math
- The field would benefit from your perspective, as code in bionformatics is often research grade (poorly implemented, poorly documented, often not available)

Section 2

Some cool algorithms in Genomics!

Subsection 1

Smith-Waterman Algorithm

Why do we care about this

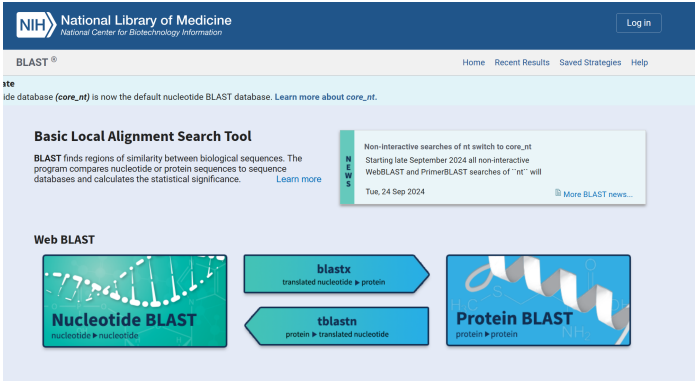
- The utility of this algorithm is to determine similar regions in nucleic acid/protein sequences

Why do we care about this

- The utility of this algorithm is to determine similar regions in nucleic acid/protein sequences
- This is useful if you want to see how related a sequence is to another

NCBI Blast

- The NCBI BLAST uses a variation of this to match sequences



Smith-Waterman Recurrence Formula

$$H_{ij} = \max \begin{pmatrix} H_{i-1,j-1} + s(a_i, b_j) \quad \text{(Match/Mismatch)} \\ \max_{k \geq 1}\{H_{i-k,j} - W_k\} \quad \text{(Gap in sequence A)} \\ \max_{l \geq 1}\{H_{i,j-l} - W_l\} \quad \text{(Gap in sequence B)} \\ 0 \quad \text{(Local Alignment, no negative scores allowed)} \end{pmatrix}$$

- This equation calculates the maximum score at position H_{ij}.
- It considers matches, mismatches, gaps, and the possibility of resetting to 0 (local alignment).

Match and Mismatch Calculation

- Match: If $a_i = b_j$, we add a positive score, e.g., $+1$.
- Mismatch: If $a_i \neq b_j$, we add a penalty, e.g., -0.3.
- The diagonal move in the matrix is used for both matches and mismatches.

$$H_{ij} = H_{i-1,j-1} + s(a_i, b_j)$$

Match and Mismatch Calculation

	Δ	C	A	G	C	C	T	C	G	C	T	T	A	G
Δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	0	0	0	0	1	0
A	0	0	1	0.7	0	0	0	0	0	0	0	0	1	0.7
T	0	0	0											
G	0	0	0											
C	0	1	0											
C	0	1	0.7											
A	0	0	2											
T	0	0	0.7											
T	0	0	0.3											
G	0	0	0											
A	0	0	1											
C	0	1	0											
G	0	0	0.7											
G	0	0	0											

Rules for Score:

Match = +1

Mismatch = -0.3

Gap = -(1 - n* 1/3)

Min = 0

Gap = -1.33 (n=1)

Gap = -1.67 (n=2)

Gap = -2 (n=3)

Gap = -2.33 (n=4)

Gap = -2.67 (n=5)

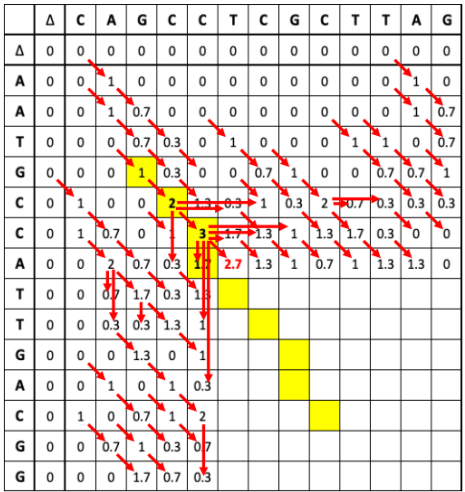
Gap Penalty Calculation

- Gaps introduce a penalty to the alignment score.
- The gap penalty depends on the number of consecutive gaps.
- For example:

$$\text{Gap Penalty} = -(1 + n \times \frac{1}{3}), \quad \text{where } n \text{ is the number of gaps.}$$

- The longer the gap, the greater the penalty.

Gap Penalty Calculation



	Δ	C	A	G	C	C	T	C	G	C	T	T	A	G
Δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	0	0	0	0	1	0
A	0	0	1	0.7	0	0	0	0	0	0	0	0	1	0.7
T	0	0	0	0.7	0.3	0	1	0	0	0	1	1	0	0.7
G	0	0	0	1	0.3	0	0	0.7	1	0	0	0.7	0.7	1
C	0	1	0	0	2	1.3	0.3	1	0.3	2	0.7	0.3	0.3	0.3
C	0	1	0.7	0	1	3	1.7	1.3	1	1.3	1.7	0.3	0	0
A	0	0	2	0.7	0.3	1.7	2.7	1.3	1	0.7	1	1.3	1.3	0
T	0	0	0.7	1.7	0.3	1.3								
T	0	0	0.3	0.3	1.3	1								
G	0	0	0	1.3	0	1								
A	0	0	1	0	1	0.3								
C	0	1	0	0.7	1	2								
G	0	0	0.7	1	0.3	0.7								
G	0	0	0	1.7	0.7	0.3								

Why Reset the Score to Zero?

- In the Smith-Waterman algorithm, we reset the score to zero whenever the alignment score becomes negative.
- This prevents continuing the alignment through poorly matching regions, which would otherwise reduce the overall score.
- By resetting to zero, the algorithm ensures that only the best local alignment is considered, avoiding penalties from distant or unrelated regions.

Local vs. Global Alignment

- Global alignments attempt to align entire sequences, even if the ends are not well-matched, possibly leading to negative scores.
- Local alignments focus only on the most similar subsections of sequences, by resetting negative scores to zero and restarting the alignment.

Example of Resetting the Score

- Suppose one part of two sequences aligns well, but another part aligns poorly.
- Without resetting the score to zero, the poorly aligned section would drag down the overall score.
- By resetting the score at negative values, we focus only on the well-aligned subsequence.

Subsection 2

Burrows-Wheeler Transform

What is the Burrows-Wheeler Transform?

- The Burrows-Wheeler Transform (BWT) is a data transformation algorithm.
- It reorders a string of characters into runs of similar characters.
- Used primarily in data compression and genome indexing algorithms.

Applications of BWT

- Data Compression: BWT is commonly used in compression algorithms like bzip2.
- Genome Sequencing: The BWT is an integral part of algorithms like Bowtie and BWA for genome alignment.
- Efficient Search Algorithms: BWT allows efficient searches through a compressed dataset without needing to decompress it.

How the Burrows-Wheeler Transform Works

- Step 1: Start with a given string and append a special character $ at the end.

How the Burrows-Wheeler Transform Works

- Step 1: Start with a given string and append a special character $ at the end.
- Step 2: Generate all cyclic rotations of the string.

How the Burrows-Wheeler Transform Works

- Step 1: Start with a given string and append a special character $ at the end.
- Step 2: Generate all cyclic rotations of the string.
- Step 3: Sort the rotations lexicographically.

How the Burrows-Wheeler Transform Works

- Step 1: Start with a given string and append a special character $ at the end.
- Step 2: Generate all cyclic rotations of the string.
- Step 3: Sort the rotations lexicographically.
- Step 4: Extract the last column of the sorted rotations — this is the BWT of the string.

BWT Example: Original String and Rotations

- Example: Transform the string "banana$".
  1. Original string: `banana$`
  2. Cyclic rotations:
     - `banana$`
     - `anana$b`
     - `nana$ba`
     - `ana$ban`
     - `na$bana`
     - `a$banan`
     - `$banana`

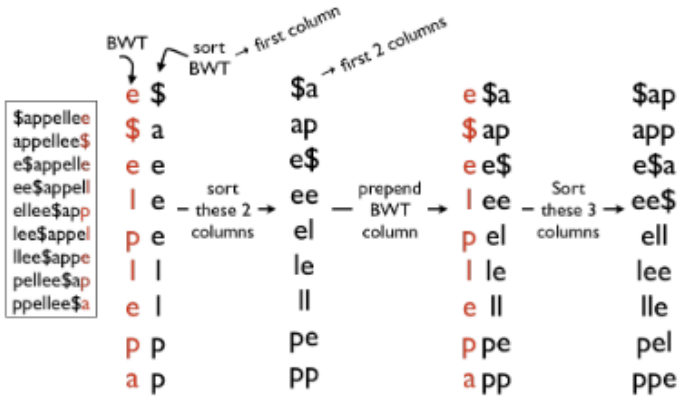
BWT Example: Sorted Rotations and Final BWT

- Sort the rotations lexicographically:
  - `$banana`
  - `a$banan`
  - `ana$ban`
  - `anana$b`
  - `banana$`
  - `na$bana`
  - `nana$ba`
- BWT string is the last column: `annb$aa`

Reversing the BWT

- Step 1: Construct the sorted matrix of all rotations using the BWT result.
- Step 2: Rebuild the original string by iterating over the first and last columns of the matrix.
- Step 3: Extract the original string from the first row.

Recovering the original String



Understanding the BWT Matrix

- The BWT matrix, M_S, contains all cyclic rotations of a string S, sorted alphabetically.
- The first column of M_S, denoted as F, is the letters of S sorted alphabetically.
- The last column L is the BWT result, representing the last characters of each sorted rotation.

$$M_S = \text{All cyclic rotations of } S \text{ sorted lexicographically}$$

Constructing the LF Matrix

- Let $L \cdot F$ denote the vertical concatenation of L and F.
- LF creates a two-column matrix where each row corresponds to the characters at the start of some cyclic rotation.
- Since L_i precedes F_i in S for each i, sorting LF reveals relationships in the original string.

$$LF = \begin{bmatrix} L_1 & F_1 \\ L_2 & F_2 \\ \vdots & \vdots \\ L_n & F_n \end{bmatrix}$$

Recursive Sorting of LF Matrix

- After forming LF, sort it to progressively recover columns of M_S.
- At each step, reconstruct a new column by sorting the concatenated $L \cdot F$.
- Repeat this process $O(n)$ times to fully reconstruct M_S.

$$\text{New } LF = \text{sort}(L \cdot F)$$

$$\begin{bmatrix} L & F & F_1 \\ L & F & F_2 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Mathematical Complexity and Completion

- The reconstruction process requires $O(n)$ iterations, where n is the length of the string.
- Each iteration involves sorting, which takes $O(n \log n)$ comparisons.
- Every comparison during sorting takes $O(n)$ time, resulting in an overall complexity of:

$$O(n) \times O(n \log n) \times O(n) = O(n^3 \log n)$$

- Although this is not the most efficient approach, it demonstrates that the BWT contains sufficient information to reconstruct the original string.
- After the final iteration, M_S is fully reconstructed, and the original string is found in the row starting with '$'.

Section 3

Brain Imaging

Subsection 1

Active Contour Models

Introduction to Active Contour Models (Snakes)

- Active Contour Models (ACMs), or Snakes, are used in image processing for object boundary detection.
- Snakes are curves that evolve to minimize an energy function and capture object boundaries.
- The energy functional is typically composed of internal and external components.

Line Functional

- The line functional represents the intensity of the image, denoted as:

$$E_{\text{line}} = I(x, y)$$

- The sign of E_{line} determines whether the snake is attracted to bright or dark areas.
- Some smoothing or noise reduction can be applied, modifying the line functional to:

$$E_{\text{line}} = \text{filter}(I(x, y))$$

- The line functional guides the snake toward intensity features in the image.

Edge Functional

- The edge functional is based on the gradient of the image and attracts the snake toward edges:

$$E_{\text{edge}} = -\left|\nabla I(x,y)\right|^2$$

- The snake moves towards areas of high intensity gradient, i.e., object boundaries.
- Scale-space continuation is applied using Gaussian smoothing, refining the edge detection:

$$E_{\text{edge}} = -\left|G_\sigma \cdot \nabla^2 I(x,y)\right|^2$$

Termination Functional

- Let $C(x, y)$ be the smoothed image:

$$C(x, y) = G_\sigma \cdot I(x, y)$$

- The gradient angle θ is defined as:

$$\theta = \arctan\left(\frac{C_y}{C_x}\right)$$

- The termination functional minimizes energy using the second derivatives along the gradient direction, defined as:

$$E_{\text{term}} = \frac{\partial \theta}{\partial n_\perp} \frac{\partial^2 C}{\partial n_\perp^2} - \frac{\partial \theta}{\partial C}$$

- This captures features such as corners and terminations in the image.

Mathematical Overview of the Snake's Energy Function

- The total energy of the snake is the sum of all functional energies:

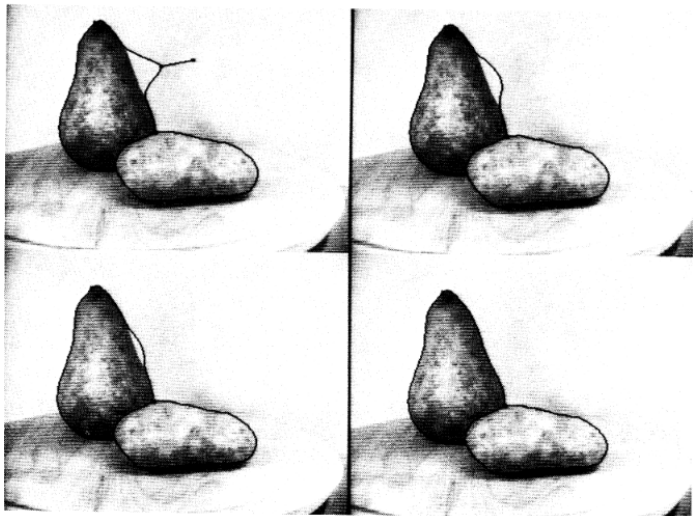
$$E_{\text{snake}} = E_{\text{line}} + E_{\text{edge}} + E_{\text{term}}$$

- The snake evolves to minimize this total energy, balancing smoothness (internal energy) and attraction to image features (external energy).
- Internal energy ensures smoothness and continuity of the snake, while external energy pulls it toward edges and key features.

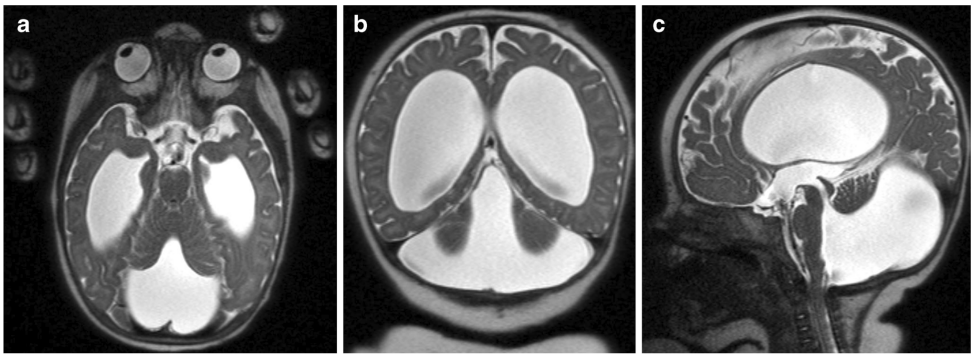
Summary of Active Contour Energy Functionals

- *Line Functional*: Guides the snake toward intensity regions in the image.
- *Edge Functional*: Pulls the snake toward edges, detected using gradients.
- *Termination Functional*: Captures corners and key feature terminations.
- The snake minimizes these energies to conform to object boundaries while maintaining smoothness.

Visual Example of a Snake



Why would we use this?



Subsection 2

K-Means Clustering

Applications of K-Means in MRI

- Brain Tumor Segmentation: K-means can help identify and separate tumor tissue from healthy brain tissue by clustering pixels based on intensity.
- Tissue Classification: Different tissue types (e.g., gray matter, white matter, cerebrospinal fluid) can be distinguished by clustering based on pixel intensity and texture.
- Organ Segmentation: K-means can be used to segment organs, such as the liver or kidneys, to facilitate volumetric analysis.

How K-Means Works in MRI Segmentation

- Input: The algorithm takes the pixel intensities from MRI images as input data points.
- Clustering: K-means groups pixels with similar intensities into clusters, with each cluster ideally representing a specific type of tissue or region.
- Output: The resulting clusters highlight areas of interest, such as lesions, tumors, or other abnormalities.

$$D(V, X) = \frac{1}{N} \sum_{i=1}^{N} \min_{k} d^2(v_i, x_k)$$

Introduction to the K-Means Greedy Algorithm

- The K-Means Greedy Algorithm is a variant of the traditional K-means clustering algorithm.
- It uses a greedy approach to iteratively improve cluster assignments.
- The goal is to reduce the overall cost (or error) by moving points to the cluster that offers the largest improvement.

Pseudocode for the K-Means Greedy Algorithm

ProgressiveGreedyK-Means(k)

```
Select an arbitrary partition P into k clusters
while forever
bestChange ← 0
for every cluster C
for every element i not in C
if cost(P) - cost(P_{i→C}) > bestChange
bestChange ← cost(P) - cost(P_{i→C})
i* ← I
C* ← C
if bestChange > 0
Change partition P by moving i* to C*
else
return P
```

Explanation of the Algorithm - Initialization

- *Step 1*: Start with an arbitrary partition P of the data into k clusters.
- *Step 2*: Initialize a variable, `bestChange`, to track the maximum improvement in cost by reassigning points.

Explanation of the Algorithm - Cluster Evaluation

- For each cluster C, evaluate the cost impact of moving each element i (not already in C) to C.
- Calculate the change in cost if i were moved to C:

$$\text{bestChange} \leftarrow \text{cost(P) - cost}(\text{P}_{\text{i}\rightarrow\text{C}})$$

- Update `i*` and `C*` to the element and cluster that maximize this cost reduction.

Explanation of the Algorithm - Reassignment

- If the best change in cost is positive (`bestChange` > 0), reassign element $i*$ to cluster $C*$.
- If no assignment results in a positive cost reduction, terminate the algorithm and return the current partition.
- This process is repeated until no further cost improvements are possible.

Questions?

Brainteaser

	Δ	C	A	G	C	C	T	C	G	C	T	T	A	G
Δ	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	0	0	0	0	1	0
A	0	0	1	0.7	0	0	0	0	0	0	0	0	1	0.7
T	0	0	0	0.7	0.3	0	1	0	0	0	1	1	0	0.7
G	0	0	0	1	0.3	0	0	0.7	1	0	0	0.7	0.7	1
C	0	1	0	0		1.3	0.3	1	0.3	2	0.7	0.3	0.3	0.3
C	0	1	0.7	0	1		1.7	1.3	1	1.3	1.7	0.3	0	0
A	0	0	2	0.7	0.3		2.7	1.3	1	0.7	1	1.3	1.3	0
T	0	0	0.7	1.7	0.3	1.3		2.3	1	0.7	1.7	2	1	1
T	0	0	0.3	0.3	1.3	1	2.3		2	0.7	1.7	2.7	1.7	1
G	0	0	0	1.3	0	1	1	2		2	1.7	1.3	2.3	2.7
A	0	0	1	0	1	0.3	0.7	0.7		3	1.7	1.3	2.3	2
C	0	1	0	0.7	1	2	0.7	1.7	1.7		2.7	1.3	1	2
G	0	0	0.7	1	0.3	0.7	1.7	0.3	2.7	1.7	2.7	2.3	1	2
G	0	0	0	1.7	0.7	0.3	0.3	1.3	1.3	2.3	1.3	2.3	2	2