Colour Blindness Report

Vaishnav K.V.

vaishnavk@iisc.ac.in

Important Note:

I have used the "sp. special. comb" function for finding NCR for given n and r. But sp. special is available in the latest version of scipy. And NOT AVAILABLE in the older scipy version.Pls, consider this.

Objective:

Find the configuration responsible for causing colour blindness in the given gene sequence.

Background:

- It is found that colour blindness occurs because of the overlap between the Red and Green colour receptors' domain.
- The recipe for Red and Green sensors is stored in the 23'rd chromosome.
- It is also known that there are differences at 15 points between Red and Green.
- Within a chromosome, information is only stored in axons. The address information for each axon is given in the script.

Data preprocessing(**loadLastCol,loadRefSeq,loadReads,loadMaptoRefseq**):

- Using file.read() function ,data from the files are loaded.
- By default, the above function returns a list since we need strings in the first two cases; a NumPy array is for the last function.
    - Strings: I haven't removed "\n" since we need this format later
    - Numpy array: can be directly converted.

Setting up global variables:

- We segmented the last column into sequences of Len 100,
- We followed the same template for both reference and bwt. They are lists of strings, each ending with “\n”.
- Defined some global variables. eg, Milestone(from the class) dictionary, which contains the number of “A”s before and including at a particular index i. Here indexing is over the segmented last column. This segmentation enables us to use ideas similar to absolute/relative addresses in computer architecture.
- Instead of having a single parameter to access the whole space, if we have divided the whole space(last column) into len of 100, we will have a local address as the additional parameter. This architecture will be helpful in the search operation, where we need to find the position of the first mismatch for a given read.
- The above idea is similar to “signal=bias+small_signal”.
- A similar approach can be used for “RefSeq”. Initially, I tried it directly using RefSeq without splitting it into chunks to form a big list. It is observed that the direct address(directly accessing a slice from the list) and the relative address(obtained via the partitioning approach) differed at some positions, causing the error. The above may be because of some bug in my script, but because of the limited time availability, I have used the second approach. I have commented on those lines of code. To work in that way, we need further processing also because the first 6 characters of RefSeq is a garbage value.

The flow of the program:

- We initialise the required global variables.
- We iterate over the reads. Let’s name each element of reads as read.
- **MatchReadtoLoc:**
  - If “N” is present, make it into “A.”
  - Since reverse complement is also a possibility, we need to iterate over them also.
  - Using a function, we can find the location at which the first mismatch happens, we will return the lower_adderess and upper_adderess along with the rel. Position at which the first mismatch happened.
  - Now to find the reference string, we will iterate over the address from the lower_address to the upper_address.
  - Given the address and required length, we need to get those portions from the reference since the reference is partitioned into blocks with “\n” at the end; we will append until we get the desired length.

- Since we needed a match of reading and ref sequence of the same length within an error margin of 2, if such a location/address is found, it is added to the list.
- **get_loc function:**
  - Search is a local function that is defined. Given the read, it will give the address of the occurrence of the first mismatch. It will return the address in the form of the lower_main_address,upper_main_adderess and a relative address. Here the relative address gives the position in which the first mismatch occurs. Here the iteration starts from the last char to the first, Since it is a suffix search. This loop can be seen as a generalisation of a binary search kind of algorithm. Initially, the lower_main_address is set to 0, and the upper_main_address is set to the largest value. As we iterate from the end, our effective searching area will get reduced ie (upper_main_address-lower_main_address). So effectively, we narrow our horizons. In the horizon, if the last C's relative address is lower than the first C's relative address, we will return that address. Else we will narrow the horizon and move to the next character. In the end, this will return the address of a "read" in lower_main_address.
  - **get_rel_address:**
    - This is to facilitate the movement between absolute address and relative address.

- **WhichAxon:**
  - For a read from the first function, we will get a list of potential match locations now if that location lies in the range of address of axons(R1, R2, R3, R4, R5, R6 and G1, G2, G3, G4, G5, G6 's address range is given). Then it is a match-up problem. For each element in the location, we will search if it is in any of the address ranges. If it is, then an indicator variable is set. If one location lies in both Red and Green's same axon, then 0.5 is for each (given in the qst)
  - This will return an array of indicators of size 12.
- We will accumulate all the 12*1 vectors to yield "ExonMatchCounts."
- **ComputProb:**
  - **Method 1(**this method is not implemented (commented out), the binomial-based method is implemented) Based on intuition.
  - We know that colour blindness cause axons are 2,3,4,5
  - Now using "ExonMatchCounts", we can get the fraction vector named"R_by_G."
  - Now we have a R_by_G vector of size 4, and we also have 4 configuration hypotheses, denoted by H. We need to find the probability that R_by_G is from each of the 4 hypotheses.
  - An important decision is here to choose the best distance metric. I have used **"Generalised KL"** as the divergence metric, and intuitively its inverse will give us a

score. The score will be high if R_by_G and the chosen hypothesis are “closer”. Then using sigmoid to yield a probability distribution.

- **Method 2(This method is implemented)**
  - Here I have assigned probability following the likelihood value.
  - That is, the best hypothesis is the one that maximises the likelihood. Given the axon pair (one red, one green)count separately, we could calculate the probability of getting red as per the binomial formula for a hypothesis. Now we do this computation for each of the axon pairs similarly, and we keep on accumulating the score for each hypothesis. Since we needed a distribution over the hypothesis, we normalised.
- **Observation:**
  - Method 1(based on intuition) and Method 2, give the highest probability to config index 2. (starting from 0).

- **BestMatch:**
  - This will return the hypothesis’s index, which has the largest prob.

Results:

- **ExonMatchCounts vector:**

```
[ 90.5  78.   71.  146.  261.5  222.   90.5  228.  125.  127.  327.5  222. ]
```

- **The probability distribution over the configurations:(rounded off to 4 positions)**

  [0.2441, 0.0001, 0.45078, 0.3050]
- **The highest probable hypothesis/configuration:**

  Configuration 2 is the best match

############## ##