Data Analytics
Assignment - 4

Arman Gupta (19220) - Mtech CSA

Description and Implementation of Functions-

For implementing the assignment, I followed the instruction provided in lectures. Below is a detailed explanation of each function I have implemented-

1. loadLastCol - This function takes the path of the file corresponding to the last column (BWT) as input and simply reads the file and stores it as a string.
2. loadRefSeq - This function takes the path of the file corresponding to the reference sequence as input and simply reads the file and stores it as a string.
3. loadReads - This function takes the file path of the file containing all the reads as input and simply reads the file and stores the reads as a list of strings.
4. loadMapToRefSeq - This function takes the path of the file containing the mapping of the first column to the reference sequences and loads it into the NumPy array.
5. rankDataStructure - This function precomputes the rank of each base in the last column and stores them at a fixed 'delta' interval. This delta is the hyperparameter that is fixed based on the memory available in the RAM.
6. findrank - This function finds the rank of the desired base from some fixed reference point. For calculating the first rank of the base, it takes the starting point of a band as the referenced index and then moves upward to the nearest index where the rank has been stored in the rank data structure and counts the number of times desired base is encountered before the referenced index(inclusive). After getting all the counts of the desired base, it calculates rank either as count or count+1 based on if we find the base at referenced index itself or after the referenced index. For calculating the last rank of the base, it takes the ending point of a band as the referenced index and then moves upward to the nearest index where the rank has been stored in the rank data structure and counts the number of times desired is encountered before the referenced index(inclusive). This gives the last rank of the base. This function computes rank in O(delta) time.
7. getFirstCol - Compute the first column by sorting LastCol and computing the first and last occurrence of each base in the first column
8. searchForSubstring2 - This function computes the band (range of indices) where the read of (say) length n exactly matches the first n characters in the string in the band
9. MatchReadToLoc - This function uses the searchForSubstring2 method to get the band. In case we get the empty band, which indicates the read has no exact matching in the reference sequence, we are breaking the string into 3 equal parts and calling the searchForSubstring2 method for each part to get the potential band, and then using **Map** that we have found in loadMapToRefSeq, we get the locations in the reference corresponding to indices in the bands and perform the string matching operation to check if we are getting mismatches less than equal to 2. This method uses **Map** that maps indices within the band to potential locations in the reference sequence.
10. WhichExon - It takes the positions from the MatchReadToLoc method and increments to the count of each exon. It counts reads mapping to exons of the red and green genes, counting 1 for each read that unambiguously maps to one of the two genes, and 1/2 for each gene for a read that maps ambiguously.
11. ComputeProb - It determines the probability of generating these counts given that configuration. In this function, I have considered that only 4 configurations are possible which are
    - c_0 = [0.5,0.5,0.5,0.5]
    - c_1 = [1,1,0,0]
    - c_2 = [0.33,0.33,1,1]
    - c_3 = [0.33,0.33,0.33,1].

    Now from the exact count matches **C**, I have calculated the vector **v = [C[1] / C[7], C[2] / C[8], C[3] / C[9], C[4] / C[10]].**

    With the loss of generality, we can say that the probability of generating the counts given the configurations is equivalent to $P(v|c_i)$ and we need to find the configuration c_i^* such that $c_i^* = \text{argmax}_{c\_i} P(v|c_i)$. Using Bayes Theorem, we can write

$$argmax_{c_i} P(v|c_i) = argmax_{c_i} \frac{P(c_i|v) * P(v)}{P(c_i)}$$

Now, argmax does not depend on vector v, so we can remove it and assume the prior probability of getting each configuration is equal which is a fair assumption to make here as we can get any of the four configurations in our case. So instead of finding the probability of vector v given configuration, we will find the probability of configurations given vector v considering our sample space consists of only 4 configurations. Thus, we will find c_i^* = argmax$_{c\_i}$ P(c_i|v).

To find P(c_i| v), I calculated the cosine similarity of the vector v with each configuration to get how similar v is with each configuration. Then I applied the softmax function to get the probability distribution. The equations are illustrated below-

$$cosine - similarity(v, c) = \frac{v.c}{||v||\ ||c||}$$

Now , Let cs$_{\{i\}}$ = cosine-similarity(v , c_i) and now we can compute softmax probability for each configuration as,

$$softmax(cs_i) = \frac{e^{cs_i}}{\sum_{i=0}^{3} e^{cs_i}}$$

Note that, the sum of softmax probabilities will be 1 here thus we get a probability distribution here.

12. BestMatch - This function returns the best configuration based on the probability distribution we have computed in the ComputeProb function.

Observations and Results-

The Exon match counts for Red and Green Exons are as follows-

1. Red Exons - [176.5 , 121. , 106.5, 188.5 , 328. , 422.5]
2. Green Exons- [217.5, 278., 171.5, 163.5 , 400. , 422.5]

Observation:
We are noticing the number of counts for exon 1 in the case of red and green are mismatching. Although this should not be the case, my counter-argument is that exon 1 matches exactly in red and green but the read may intersect with exons at the ending position of read and not at the beginning of the read. Now the string of bases before exon1 of red and green doesn't need to be the same. Thus the read may not align with the string of bases before exon1 of red but align with the string of bases before and in exon1 of green which justifies the mismatches. This mismatch of the count is not observed in exon6 as the count of exon6 matches is the same for green and red.

Result:
We get v = [0.435, 0.621 , 1.152 , 0.82]. Notice that this v is quite comparable to the c_2 configuration. We have computed the probability distribution [0.28189024, 0.17473333, 0.28977165, 0.25360478] which indicates that the probability for the c_2 configuration is the highest among all the configurations. Thus Configuration number 2 is the best match here.