Rhodes College

Bioinformatics

COMP 342 Spring 2020

Lab 1: Greedy Motif Search
Due: Monday, February 3, 2020 – at the start of class

The purpose of this lab is to work with a partner to learn and implement a greedy algorithm for finding motifs and compare its run-time and correctness to the branch and bound solution discussed in class.

I have provided starter code for the implementation portion of the lab and a complete implementation of the Branch and Bound Motif Search (not Median String) discussed in class. The files are in the Box.com folder – link is on Moodle. The files you'll need are:

starterCode_greedyMotifSearch.py	**MotifSearch.py**
greedyInput1.txt	**Score.py**
motifInput1.txt	**SearchTrees.py**

In class we discussed calculating a profile for a set of possible motifs (one from each of `t` DNA sequences). To change this into a probability rather than a count of the number of times we see each nucleotide at each position, simply divide each value by `t` (the number of DNA sequences). We can then multiply these probabilities together when determining the most probable k-mer. We also discussed how to calculate the distance score for a set of motifs. In the code provided, the distance score is calculated as we saw in our notes – it looks a bit different since it is easier to work with rows of a list instead of columns. Below is an example of how the score, consensus and profile are computed, as well as how the profile can be used to compute the probability of a particular k-mer.

	T	C	G	G	G	G	g	T	T	T	t	t	3
	c	C	G	G	t	G	A	c	T	T	a	C	4
	a	C	G	G	G	G	A	T	T	T	t	C	2
	T	t	G	G	G	G	A	c	T	T	t	t	4
Motifs	a	a	G	G	G	G	A	c	T	T	C	C	3
	T	t	G	G	G	G	A	c	T	T	C	C	2
	T	C	G	G	G	G	A	T	T	c	a	t	3
	T	C	G	G	G	G	A	T	T	c	C	t	2
	T	a	G	G	G	G	A	a	c	T	a	C	4
	T	C	G	G	G	t	A	T	a	a	C	C	+ 3

SCORE(*Motifs*) $3 + 4 + 0 + 0 + 1 + 1 + 1 + 5 + 2 + 3 + 6 + 4 = \mathbf{30}$

CONSENSUS(*Motifs*) T C G G G G A T T T C C

Profile

A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
C:	.1	.6	.0	0	0	0	0	.4	.1	.2	.4	.6
G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4

Pr(ACGGGGATTACC, Profile) $= .2 \cdot .6 \cdot 1 \cdot 1 \cdot .9 \cdot .9 \cdot .9 \cdot .5 \cdot .8 \cdot .1 \cdot .4 \cdot .6$

$= 0.000839808$

`Score(motifs)`, `Profile(motifs) and Consensus(motifs, k),` where k is the length of the motif, are provided for you in the starter code. `Score(motifs)` is the total hamming distance, which means we are trying to **minimize** the score.

To complete this lab, you and your partner will need to implement the following functions, as specified in the starter code.

```
HammingDistance
profile_most_probable_kmer
greedyMotifSearch
```

I've included descriptions of parameters and return values for each of the functions, and pseudocode for the greedy algorithm. The pseudocode for the greedy algorithm is described below as well.

Before starting to write code, it's important that you understand what the code should be doing. Do the following by hand to give you a better idea of how the code should work.

Hamming Distance

Let `s1='GATTCTCA'` and `s2='GACGCTGA'`, what should `HammingDistance(s1, s2)` return? ____

Profile Most Probable K-mer

Using the profile given on Page 1 of this lab, compute `Pr(TCGTCCATTTCC | Profile)`.

Using the same profile, compute the most probable 12-mer in the following text: `GGTACGGGGATTACCT`.

To do this, first, list all possible 12-mers from the text; then compute the probability of each one. Just do this until you understand how the function should work.

Greedy Motif Search Algorithm

Our proposed greedy motif search algorithm, **GreedyMotifSearch**, tries each of the *k*-mers in *DNA*$_1$ as the first motif. For a given choice of *k*-mer *Motif*$_1$ in *DNA*$_1$, it then builds a profile matrix *Profile* for this lone *k*-mer, and sets *Motif*$_2$ equal to the *Profile*-most probable *k*-mer in *DNA*$_2$. It then iterates by updating *Profile* as the profile matrix formed from *Motif*$_1$ and *Motif*$_2$, and sets *Motif*$_3$ equal to the *Profile*-most probable *k*-mer in *DNA*$_3$. In general, after finding $i - 1$ *k*-mers *Motifs* in the first $i - 1$ strings of *DNA*, **GreedyMotifSearch** constructs *Profile*(*Motifs*) and selects the *Profile*-most probable *k*-mer from *DNA*$_i$ based on this profile matrix. After obtaining a *k*-mer from each string to obtain a collection of Motifs, **GreedyMotifSearch** tests to see whether *Motifs* outscores the current best scoring collection of motifs and then moves *Motif*$_1$ one symbol over in *DNA*$_1$, beginning the entire process of generating *Motifs* again.

Pseudocode:

```
GreedyMotifSearch(DNA, k, t)
        BestMotifs ← empty motif list
        BestScore ← t * k
        for each k-mer Motif in the first string from DNA
            Motif1 ← Motif
            for i = 2 to t
                form Profile from motifs Motif1, …, Motifi – 1
                Motifi ← Profile-most probable k-mer in the i-th string in DNA
            Motifs ← (Motif1, …, Motift)
            if Score(Motifs) < BestScore:
                BestMotifs ← Motifs
                BestScore ← Score(Motifs)
        return BestMotifs
```

Below is the input from greedyInput1.txt and the output you should receive after running your greedy algorithm successfully.

Sample Input:

```
3 5
GGCGTTCAGGCA
AAGAATCAGTCA
CAAGGAGTTCGC
CACGTCAATCAC
CAATAATATTCG
```

Sample Output:

```
CAG
CAG
CAA
CAA
CAA
Consensus = CAA
```

Analyzing Greedy Motif Search

Compare the run time of your greedy algorithm to the BranchAndBoundMotifSearch using the file motifInput1.txt. You'll just need to run MotifSearch.py and use the motifInput1.txt file. ***You may want to start the running of the branch and bound code early, since it takes a little while to complete.***

What I need from you:
Please turn in a single lab with both your and your partner's name on it that includes the following in your write-up. You may upload your code file to Moodle – no need to print it out.

1. Turn in your completed greedy algorithm code file. (upload to Moodle)
2. Write down the solution for motifInput1.txt from running your greedy algorithm code.
3. Answer the following questions.
    a. Did your greedy algorithm find the same solution as the branch and bound motif search for motifInput1.txt?
    b. What is the Big-O run time for the greedy algorithm?
    c. What is the greedy heuristic (metric) being used in the greedy algorithm?
    d. What is being minimized/maximized in the greedy algorithm?
    e. When you run the greedy algorithm code, print out the profile of the final motifs. How many zero probabilities are there? How could this affect your solution?
    f. What are some ideas you have for fixing the problem of zero probabilities in the profile?