BIOL591: Introduction to Bioinformatics
Construction of programs to detect anomalous genes

Outline:

I. Problem: Getting from Shakespeare to pathogenicity islands
II. Modifying Hamlet.pl to act on sets of DNA sequences
III. Using a Markov model to assess the similarity of unknown genes to the training set (later)

I. Problem: Getting from Shakespeare to pathogenicity islands

You are now familiar with the program Hamlet.pl and how it uses a Markov model to analyze textual material. If our goal were to produce a body of faux Shakespeare (or help those chimpanzees complete their life's task), we'd be through, but our goal instead is to find a way to predict which genes are foreign to a genome. I hope that you see that this program may well be pertinent to our goal. If a simple Markov model can capture something recognizable about Shakespeare or Christmas carols, why can't it also capture something important about a set of sequences we decide are native? We thus have two steps to consider in going from Hamlet.pl to a way of deciding on the alien status of a gene:

1. How do we modify Hamlet.pl so that it will use a set of sequences known to be native as the raw material to build the Markov model?
2. How do we use the Markov model on unknown sequences to assess the likelihood that those sequences are alien to a genome?

II. Modifying Hamlet.pl to act on sets of DNA sequences

Changing the input file of Hamlet.pl from HamletSpeech.txt to Carols.txt was sufficient to induce the program to switch its output from perverted Hamlet to perverted Christmas carols. Perhaps offering the program an input file of DNA sequences (e.g. the file 6803_training_set.nt) is all that we need to do to make a Markov model based on native genes.

Unfortunately, life isn't that simple. Examine both HamletSpeech.txt and 6803_training_set.nt and you'll see a couple of critical differences. Each textual unit in HamletSpeech.txt is enclosed in brackets ({...}), while the genes in 6803_training_set.nt are given in the usual FastA format.

SQ1. Try running Hamlet.pl using 6803_training_set.nt as the input file. What happens?

You ***could*** solve this problem by altering 6803_training_set.nt, giving each sequence brackets, but that would require writing a program to modify the file, and if you're going to expend programming effort, it might as well be towards making Hamlet.pl run on DNA sequences, not quick fixes.

We can divide the task of modifying Hamlet.pl into two parts:

A. Read in the training set
B. Go through the genes in the training set, adding each in turn to the Markov model

You might think of other ways to go about modifying the program. I chose this way, because I recalled from the last unit that we already have a module, FastA_module.pm, that knows how to read in a file of FastA-formatted genes.

SQ2. Modify Hamlet.pl to use FastA_module.pm to read in all of the genes within 6803_training_set.nt (don't try to access individual genes at this time). Give the program a new name, like MakeMarkov.pl, because it will be a significantly different program.

If you're using ThreadProtein.pl as a model (and I trust you are), reading in ***all*** the genes should have posed few problems, but the next step, considering each gene individually is a different story. ThreadProtein.pl reads just two sequences and extracts information from them by hand (I mean by passing Get_sequence_info specific sequence numbers). This approach is impractical when you have many tens of genes within the training set. Clearly we need a loop.

The loop should look something like:

```
foreach $gene (1 ..
```
...to what? How many genes were read in?

What we've done in the past is to find out the size of the array that contains all of the quantities we're interested in. That's not possible now, because FastA_module.pm is taking care of the array. We have no access to it! Well, this situation would be intolerable, unless the module has anticipated this situation. Time to look at the documentation.

SQ3. Take a look at the source code for FastA_module.pm. Do you see any way of finding out how many sequences are read by the subroutine Read_FastA_sequences?

Here's a peculiarity of Perl (though C and Java programmers may not find it peculiar): Every subroutine call returns a value, whether or not you chose to catch that value. So calling Read_FastA_sequences returns the number of sequences read, which disappears unless you provide a variable to capture the information, something like:

```
$number_of_genes = Read_FastA_sequences(name of file)
```

With that variable set, now you can write your loop. Within the loop, you'll get one sequence (using Get_sequence_info), storing the header and sequence within variables (as ThreadProtein.pl does). Then you'll Break_up the sequence into its component letters, and... the rest of the loop should be as in Hamlet.

SQ4. Make these changes and run the program.

You no doubt encountered problems, ending with the error message:

```
Line ATG...TAG doesn't start with {
```

Well, that's true. The genes don't start with {. Nor do they end with }. We'd like to get rid of that requirement. It looks like we can get rid of the error message just by getting rid of the line that produces it.

SQ5. Delete the two lines:

```
$letters[0] eq... or die...
```

and

```
$letters[@letters-1] eq... or die...
```

You ought to fix a more subtle problem, deleting the line:

```
shift @letters;
```

This prevents the first nucleotide from being lost. Now run the program.

(The program may appear to hang for a long time. That's because it's attempting to produce a very long pseudosequence. Put it out of its misery by breaking out of the program with Ctrl-C).

If all went well, you now have a working program... or do you? The purpose of the program was to make a Markov model based on the training set. Maybe it did this, but who knows? The program is (a) wasting its time making pseudo sequences and (b) failing to save the Markov for use after the program is halted. We need to solve these two problems.

The first problem is readily solved: simply tear out the subroutines (and calls to the subroutines) involved in producing new text. The second problem was solved last week, using the subroutine Output_model ( ) conveniently placed at the end of the program.

SQ6. Remove subroutines related to producing new text, and insert a call to Output_model, so that the model produced from the training set will be preserved after the program terminates. Run the program.

How can you check if the model was properly constructed and properly saved? We answered this question last week, with a service program called PrintModel.pl.

SQ7. Run PrintModel.pl to display the model constructed by your program. Capture the output by redirecting it to another file (e.g. `Perl PrintModel.pl > temp.txt`). Examine the model.

a. What start codons are used by the training set genes? At what ratio?

b. What tetranucleotide is the least common in the training set? Why? (Difficult to answer, but take a look at Fig. 2 of the notes from last Monday).

III. Using the Markov model to assess the similarity of unknown genes to the training set

(Subject of later notes)